Review of selected Semantic Web technologies

Weronika T. Furmańska
AGH University of Science and Technology
Department of Automatics
Kraków, POLAND
wtf@agh.edu.pl

Grzegorz J. Nalepa
AGH University of Science and Technology
Department of Automatics
Kraków, POLAND
gjn@agh.edu.pl

Published online: 10.12.2009

Computer Science Laboratory Technical Reports (CSL TR) series serves as an accessible platform for publishing research results of the CS Lab staff members at Department of Automatics of AGH University of Science and Technology.

See the http://cslab.ia.agh.edu.pl/csltr:start for the series homepage.

The editorial board:

- Main Editor: Marcin Szpyrka, Ph. D., D. Sc.
- Prof. Antoni Ligeza, Ph. D.
- Prof. Tomasz Szmuc, Ph. D.
- Grzegorz J. Nalepa, Ph. D.

Cover design: Marcin Szpyrka, Ph. D., D. Sc.

LATEXclass design: Marcin Szpyrka, Ph. D., D. Sc.

Contact us at: mszpyrka@agh.edu.pl
Abstract. The report introduces the concepts and technologies of the Semantic Web project, its main ideas and aims. The layered architecture is explained and supported by multiple references to the standards developed by the W3C. The logical foundations of the Semantic Web formalisms are introduced. Among them, Description Logics concepts are presented and explained in more details. Knowledge representation based on ontologies is discussed, and ontology languages, including OWL are presented.

Keywords: semantic web, description logics, rdf, rdfs, owl, ontology

1 Introduction to the Semantic Web

1.1 Motivation

World Wide Web is a huge and heterogeneous source of data with new content being added all the time. In order to find useful information people use various search engines. Most of the tools operate on the syntactic layer of the WWW by searching for keywords in the text. The meaning of the words is not understood by machines, so the results may sometimes be misleading. Consequently, most of the time people are forced to search for useful information themselves. If the task is more complicated (for example a person wants to plan a trip based on information from various websites and services), then it might be necessary to spend time on browsing tens of webpages.

The main idea of the Semantic Web [13] is to represent the meaning of data in a machine-understandable format. This would lead to improve searching for information, because the explicit semantics of data would be used. In a vision article from 2001, Tim Berners-Lee, James Hendler and Ora Lassila described the Semantic Web as an environment, where programs, so-called software agents perform complex searching, reasoning and planning tasks. Exposing meaningful information in a machine-readable formats would allow for sharing and exchanging data among applications. Formal definitions and categorization of relationships between web resources would help to integrate information from various sources. With use of appropriate knowledge representation methods and reasoning mechanisms, it would be possible to turn random information into knowledge, that could be further process.

*The paper is supported by the AGH UST Project No. 10.10.120.105.

1W3C – World Wide Web Consortium, see http://www.w3.org/.
1.2 Semantic Web and Knowledge Representation

Looking from a computer systems levels perspective (see [50]), the Semantic Web approach is related to the symbolic level. It does not aim at making machines more intelligent, but at making the data more meaningful. It allows machines to operate more efficiently by providing a common framework to share data on the Web across application boundaries. According to Ivan Herman [36], an Artificial Intelligence approach is to “teach computers to infer the meaning of Web data” by means of e.g. natural language processing, image recognition, etc., whereas the Semantic Web approach is to make data easier to find and process in an intelligent way. Despite the differences, it is definitely possible and worthwhile to use some of the AI achievements and experiences in the context of the Semantic Web.

When comparing to environments typically considered in AI, the Internet poses several challenges. The first trait which makes the Internet powerful yet hard to manage is its scale. Secondly, the Web is a distributed system. Therefore, it is impossible to organize and manage it in a structured and centralized way. Moreover, there exist thousands of authors of the Web content and this content is being changed all the time. Information provided by different sources may very well contain different or even contradictory ideas. The questions arise, how to cope with such situations or which sources to trust. The authors of [13] concluded that:

"Knowledge representation is currently in a state comparable to that of hypertext before the advent of the Web: it is clearly a good idea, and some very nice demonstrations exist, but it has not yet changed the world. It contains the seeds of important applications, but to unleash its full power it must be linked into a single global system.”

1.3 Architecture of the Semantic Web

The standards of the Semantic Web are developed in a layered architecture (see Fig. 1). At the bottom of the so-called layer cake there are standards responsible for encoding and identifying data. Information about the resources on the Internet is presented in a structured form. Above come the layers of semantics. The main paradigm of knowledge representation in the Semantic Web is to model the relations between different resources. This is done on several levels, with increasingly expressive standards. Theoretically, each level should exploit the features and extend the capabilities of the layers below it. Since 2001 the original layer cake presented in [11] has evolved by incorporating new standards and techniques. Basic standards and levels of the semantic stack are described below.

Encoding and Identifying Resources – Unicode and URI At the bottom of the so-called layer cake there are standards such as Unicode [22] char set for encoding the Web content, and URI – Uniform Resource Identifier [12] for identifying resources.

Unicode is a set of characters used all around the world. It is then possible to encode data in an arbitrary language. The Semantic Web sees the Internet as a "web of data" and identifies resources as the most basic components. A resource is an abstraction which may describe a web page or a part of it, an image, an object, a person, or a place. Each resource is uniquely identified with a Uniform Resource Identifier [12] (URI for short).

Serialization of Data – XML In order to facilitate integration and exchange of data, the XML [52] language is used. On the one hand, it enables authors to arbitrarily specify the structure of the resource descriptions, on the other it provides an universal way of exchanging data between applications. The grammar is provided by XML Schema Language [60, 41]. XML and XML Schema are described in more details in Sect. 2 and Sect. 2.3.

Metadata Level – RDF A basic standard for metadata notation is Resource Description Framework ([44], [20]). Relations between resources are modelled in a form of directed graphs. These
graphs contains resources as nodes and relations on the edges. The interpretation is that resources have some properties which have values. Therefore, the resources can be described by making statements about these properties’ values. Section 3 discusses RDF in more details.

**Ontology Layer - Categorizing Knowledge on the Web** In order to improve the expressiveness of RDF, the RDF Schema language [18] has been designed. It provides additional features enabling for basic categorization, enhancing descriptions of the resources and relations between them. Specifically, it is possible to introduce some sort of classification and hierarchy among concepts and relations denoted by the resources. RDFS is discussed in more details in Sect. 5.

The main formalism behind the Semantic Web is Description Logic [5], on which the Web Ontology Language (OWL) ([61], [25], [43]) is based. It lays the logical foundation for ontologies, which are a method of knowledge representation and provide basic inference capability. Description Logics are introduced and explained in Section 6. OWL is described in more detail in Section 7.

### 1.3.1 Unified Logic and Rules

In order to achieve Semantic Web goals, it is necessary to have some rule formalism which could operate on the knowledge expressed in ontologies. There are several approaches to the integration of ontologies and rules. Several languages have been designed, based on union or intersection of the ontology language and a rule one. For instance, an intersection of OWL and Datalog rules can be observed in so called Description Logic Programs (DLP) [29], while an union of them has been used to design Semantic Web Rule Language (SWRL) [38]. World Wide Web Consortium (W3C) has been also working on a Rule Interchange Format, RIF [39].

### 1.3.2 Security, Trust and Application Interface

Some people argue that the security and privacy issues have been left out in the Semantic Web research, while they will be the most important in the future network (see: [62]). In fact, there does not exist any official W3C working group concerned with those issues so far.
1.4 Evolution of the Semantic Web

Since 2001, the idea of the Semantic Web has gained a great interest, both in academic and business environments. Intensive research has been conducted worldwide, the solutions are presented and discussed at numerous conferences and workshops. Standards are gradually being adopted and applied in various applications. According to [28] in 2006 there existed between five to ten million Semantic Web documents on the Web. The number of available semantic tools has risen steadily for the last decade. There has been a noticeable increase of interest in semantic technologies from big companies over the past few years [35]. While Semantic Web basically deals with standards for describing data relationships, semantic technologies denote more, including software agents or systems using metadata or complex logical procedures. Around 270 companies involved in the development of semantics technologies and over 100 consumer and enterprise application categories have been described in Project 10X’s Report titled: "Semantic Wave 2008: Industry Roadmap to Web 3.0 and Multibillion Dollar Market Opportunities” [24]. The report comprises the evolution of the Internet and World Wide Web, explains various semantic techniques and solutions. Moreover, it presents 150 case studies in 14 horizontal and vertical market sectors.

The development and state-of-the-art of the Semantic Web may be observed from various perspectives. The following summary is based on [36].

The basic language of the Semantic Web, RDF, has had a stable specification since 2004. Its RDF/XML syntax and formal semantics is clearly defined. There exist numerous tools, such as RDF programming environments for over 14 programming languages, more than 13 Triple Stores, i.e, database systems to store RDF datasets, as well as converters to and from RDF. A great number of tutorials, overviews, and books have been published, active developers’ communities exist and large RDF datasets are accumulating constantly.

Web Ontology Language OWL (version 1) has a stable specification since 2004 [61] with separate layers defined, balancing expressibility and implementability (three sub-languages: OWL-Lite, OWL-DL, OWL-Full). The layers have been redefined in the new version of the language (see below). It has turned out that for a number of applications RDFS is not enough, but even OWL Lite is too much. There have been a number of proposals, papers, prototypes of so called "light ontologies": RDFS++, OWL Feather, or pD* [35]. This has led to the proposal of OWL 1.1 [53] formulated by the W3C OWL Working Group, which was an intermediate language before OWL 2.

OWL 2 [37] is a new version of the Web Ontology Language. It introduces three Profiles [46], tailored towards specific applications, such as modelling simple but large ontologies, querying ontologies or expressing some sorts of rules. OWL 2 has formally defined syntax [49], semantics [48], XML serialization [47] and mapping to RDF graphs [56].

Ontologies proved to be an useful and well accepted knowledge representation method. Various tools for OWL have been developed, including programming environments, stand-alone reasoners and ontology editors. Some core vocabularies have evolved and are used by applications. These vocabularies include FOAF 3 for describing people and their connections, Dublin Core 4 for information resources, SKOS 5 for knowledge systems or SIOC 6 for online communities. Large ontologies are being developed, converted from other formats or defined in OWL and used successfully in various domains.

Various Working Groups at W3C have been working on Best Practices for the Semantic Web 7 or development of standards. The standards include SPARQL [57] for querying RDF data and GRDDL [21, 23] for extracting RDF from existing data. The approaches of mapping relational data

---

3See http://www.foaf-project.org/.
4See http://dublincore.org/.
5See http://www.w3.org/2004/02/skos/.
6See http://sioc-project.org/.
7See http://www.w3.org/2001/sw/BestPractices/.
to RDF are examined and classified by W3C RDB2RDF Incubator Group which will potentially be responsible for proposing a standard. Works on Rule Interchange Format are led by RIF Working Group.

There are some alternative paths for the Semantic Web to explore, such as fuzzy logic or probabilistic statements. Fuzzy logic may be thought of as an alternative for Description Logic or used to extend RDF(S) with fuzzy notions. Probabilistic statements could be introduced in OWL by adding a specific probability to the class membership or combining reasoners with Bayesian networks. Security, trust, provenance should be conceived, either by combining cryptographic techniques with the RDF model or signing a portion of the graph or in other way.

2 eXtensible Markup Language – XML

XML is a markup language, designed to describe structured documents. Unlike HTML it has no semantics indicating how to present or display the documents. With use of XML any structure may be encoded, and users can create their own tags.

2.1 Syntax

An XML document consists of text and markup, in the form of tags, which is interpreted by application programs. A simple example of an XML document is as follows:

1. <?xml version="1.0" encoding="UTF-8"
   xmlns="http://agh.edu.pl/xml/agh"?>
2. <msc_thesis>
3.  <author>
4.   <firstname>Weronika</firstname>
5.   <secondname>Teresa</secondname>
6.   <lastname>Furmanska</lastname>
7.   <shortname>WTF</shortname>
8.  </author>
9.  <supervisor>
10.  <firstname>Grzegorz</firstname>
11.  <secondname>Jacek</secondname>
12.  <lastname>Nalepa</lastname>
13.  <shortname>GJN</shortname>
14. </supervisor>
15. <title lang="en">Visual Rule Design Methods for SW Apps</title>
16. <title lang="pl">Metody wizualnego projektowania regul
decyzyjnych dla aplikacji Sieci Semantycznej</title>
17. <year>2009</year>
18. <passed month="June"/>
19. </msc_thesis>

XML document may start with a processing instruction tag, which denotes the XML version and used character encoding. XML document includes a number of elements. An element typically consists of a start tag, the element content and an end tag matching the start one. An element may have an empty content as in the example above in line 18. The start tag or the empty tag may have zero or more attributes (as in line 15). An attribute consists of a name, an equal sign, and a value.

An XML document must satisfy certain syntactic requirements. For instance, every element must have either a start and and end tag, or be denoted as an empty tag. The elements must be nested properly, i.e. the inner element must be fully included in the outer one. Nesting elements defines the structure of the XML document. Element names are case-sensitive. The attribute values must be quoted. There must be exactly one root element. A document conforming to these syntactic rules is called a well-formed XML document.

See http://www.w3.org/2005/Incubator/rdb2rdf/.
Table 1: Examples of commonly used namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>URI Reference</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
<td>xsd:</td>
</tr>
<tr>
<td>RDF</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
<td>rdf:</td>
</tr>
<tr>
<td>RDF Schema</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td>rdfs:</td>
</tr>
<tr>
<td>OWL</td>
<td><a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a></td>
<td>owl:</td>
</tr>
<tr>
<td>Dublin Core</td>
<td><a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a></td>
<td>dc:</td>
</tr>
</tbody>
</table>

2.2 Namespaces

A namespace is a collection of names, identified by an URI. The namespaces define common vocabularies. Some popular namespaces are presented in Table 1.

Because anyone using XML can create arbitrary tags, there is a possibility that in two applications the same word (tag name) will be used for different purposes. To avoid ambiguity the tag names may be identified by unique URIs, consisting of a namespace URI, the character "#" and the tag name. Such identifiers are long and uncomfortable to use in documents. Therefore, in XML document one can define the used namespaces and ascribe a prefix to it. Then in the document tag names from different namespaces may be used, each preceded with an appropriate namespace prefix. For one document multiple namespaces may be defined. Each one is introduced with xmlns, for example: xmlns:xsl="http://www.w3.org/1999/XSL/Transform". One can also introduce a default namespace (see line 1 in the example above), for which no prefix must be specified.

2.3 XML Schema Language – XSD

XML Schema Language or XML Schema Definition (XSD) [27] is designed to formally specify the structure of XML documents. XML Schema defines elements and attributes that can appear in a document. It also says which elements are child elements and defines the number and order of child elements. In XSD one can define whether an element must be empty or can include text. Finally, data types for elements and attributes as well as default and fixed values for elements and attributes are introduced in XSD. The restrictions on elements, attributes and their values may be very specific.

Syntax  XML Schema uses XML syntax. XSD documents are XML documents. Therefore, their grammar is also specified by the XML Schema. An example schema for the document from Section 2 is presented below:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:group name="names">
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="secondname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="shortname" type="xs:string"/>
    </xs:sequence>
  </xs:group>

  <xs:element name="msc_thesis">
    <xs:complexType>
      <!-- ComplexType definition goes here -->
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:sequence>
  <xs:element name="author">
    <xs:complexType>
      <xs:group ref="names"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="supervisor">
    <xs:complexType>
      <xs:group ref="names"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="title">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="lang" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="year" type="xs:positiveInteger"/>
  <xs:element name="passed">
    <xs:complexType>
      <xs:attribute name="month" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

Datatypes In XML Schema several built in datatypes are defined. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

XML Schema datatypes are used in other markup languages, for example RDF, RDFS and OWL.

3 Resource Description Framework – RDF

RDF ([44, 20]) is the most recognized Semantic Web standard. It is an assertional language that is designed to represent relations between resources in the form of triples, i.e., statements of the form: subject, predicate, object. The relations may be represented in a form of a directed graph (see Fig. 2). "RDF predicates may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs" [34]. Each part of the triple is in fact an URI which points to a
resource. In case of subject and object it can also be a so-called blank node and the object may be a plain literal. The use of URI makes it possible for different RDF documents to refer to the same resources (subjects or objects). The URI of the predicate points to a definition of that property stored on the Web. It ensures existence of a strict definition of each property. Moreover, everyone can define their own predicate if they provide its definition and identify it by an unique URI.

3.1 RDF/XML Notation

There exist three popular serialization formats for RDF: RDF/XML [9], Notation 3 (N3) and Turtle. Most popular serialization format is RDF/XML. Because RDF triples may be serialized in XML, the RDF files can be exchanged and interpreted by different applications.

RDF/XML files are built according to XML documents rules (see Sect. 2). To avoid using long URIs, namespaces are used extensively. The main elements of RDF documents are the root element, `<RDF>`, and the `<Description>` element, which identifies a resource. The root element `<rdf:RDF>` defines the XML document to be an RDF document. It also contains a reference to the RDF namespace.

In RDF/XML attributes (properties of subjects) can be defined in three ways: as XML elements, attributes, or resources. The following examples (see Figs. 3, 4 and 5) come from the W3C Schools Tutorial available at: http://www.w3schools.com/rdf/default.asp.

1. Properties as XML elements

```xml
<?xml version=\"1.0"?>

<rdf:RDF
 xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
 xmlns:si=\"http://www.recshop.fake/siteinfo\#"
  
  <rdf:Description rdf:about=\"http://www.w3schools.com/RDF\">
    <si:author>Jan Egil Refsnes</si:author>
    <si:title>http://www.w3schools.com</si:title>
  </rdf:Description>

</rdf:RDF>
```

Figure 3: Example 1 as a directed graph.
2. Properties as attributes

```xml
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

  <rdf:Description
    rdf:about="http://www.recshop.fake/cd/Empire Burlesque"
    cd:artist="Bob Dylan"
    cd:country="USA"
    cd:company="Columbia"
    cd:price="10.90"
    cd:year="1985" />

</rdf:RDF>
```

3. Properties as Resources

```xml
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

</rdf:RDF>
```
3.2 Notation3, N-Triples and Turtle

Notation3 or N3 is another syntax for writing RDF documents. It is more human readable than RDF/XML and mostly intended for education purpose. A Notation3 (N3) document encodes a set of statements, and its meaning is the conjunction of the meaning of the statements. The statement of the form \( x \ p \ y \). asserts that the relation \( p \) holds between \( x \) and \( y \). The semantics of statements, where they are valid RDF statements, are those described in the RDF abstract syntax document. In property lists, the semicolon \( ; \) is shorthand for repeating the subject. In object lists, the comma \( , \) is shorthand for repeating the verb. There are several ways in N3 of representing a blank, or unnamed node: the underscore namespace, the square bracket syntax, and the path syntax.

N-Triples \([10]\) is a line-based, plain text format for encoding an RDF graph. It was designed to be a fixed subset of N3. N-Triples is an extremely constrained language; it is optimized for reading by scripts, and comparing using text tools. It allows one triple on each line. It was designed for the RDF test suite parser reference output. An example of N-Triple notation:

```
<http://www.w3.org/rdf-test/> <dc:creator> "Jan Grant" .
_:a <dc:source> <http://www.w3.org/> .
```

Another syntax for RDF is Turtle. It is an extension of N-Triples "carefully taking the most useful and appropriate things added from Notation 3 while keeping it in the RDF model".\([10]\)

3.3 Extracting, Storing and Querying RDF Triples

RDF technology proved to be useful and there already exist applications using it. Such applications use so-called RDF Triple Stores as their knowledge base and query them with appropriate languages such as SPARQL \([57]\), SerQL \([2, 19]\) or the "general-purpose data processor for the Semantic Web" called cwm.\([11]\)

The first Recommendation of SPARQL \([57]\) has been released in January 2008. The work on this standard is still in progress. However, there are a number of implementations already, and there also exist SPARQL "endpoints" on the Web. Those endpoints given a query and a reference to data over HTTP GET, send back the result in XML or JSON (JavaScript Object Notation).\([12]\) Therefore, applications may not need any direct RDF programming, having just a SPARQL endpoint.

Another topic is extracting RDF data. There exist some tools for extracting data from images or websites (e.g. from the flickr\([13]\), photo management and sharing online application). Getting structured data to RDF is possible by means of GRDDL \([21, 23]\). A "GRDDL Processor" runs the script and produces RDF on-the-fly allowing for accessing existing structured data and "bringing"
it to RDF. Another way to acquire RDF triples is semantic annotation of XHTML pages with use of RDFa [1]. This language extends XHTML with a set of attributes to include structured data into XHTML. As for exporting data from relational databases, so-called SQL-RDF bridges are being developed. The approaches of mapping relational data to RDF are examined and classified by W3C RDB2RDF Incubator Group 14 which will potentially be responsible for proposing a standard.

Table 2 summarizes some of the available Triple Stores. For more information see http://esw.w3.org/topic/LargeTripleStores.

4 Ontologies and Ontology Languages

4.1 Ontology and Ontologies

Ontology (gr. ὠντολογία) as a term originates from philosophy, where it denotes a field of study concerned with the nature of existence. In computer science the term has been adapted and given various specific technical meanings. Used in the context of knowledge representation it is no longer Ontology, but an ontology or ontologies. There exist several definition for an ontology in computer science.

According to [17] the definition of ontology most frequently quoted in the Semantic Web literature is: "An ontology is a formal explicit specification of a shared conceptualization”, formulated by T. Gruber [30] and refined by R. Studer. In this definition "conceptualization stands for an abstract model; explicit means that the elements must be clearly defined; and formal indicates that the specification should be machine processable” [17]. In other words, an ontology consist of a vocabulary used to describe (a particular view of) some domain and an explicit specification of the intended meaning of this vocabulary. It often includes classification-based information and constraints capturing additional knowledge about the domain. The incorporation of classification-based information is emphasized in the definition by J. F. Sowa: "The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D” [59].

A reason for the variety of definitions for ontology/ontologies is the large spectrum of their possible uses. This includes representation, organization and reuse of knowledge, communication between humans and implemented computer systems, as well as automated inference [17]. Ontologies may support browsing and searching, consistency checking, configuration, validation and verification testing [42]. They have been used in various areas of computer science, including knowledge representation, natural language processing, software engineering and the Semantic Web.

4.2 Taxonomies and Thesauri

In order to represent some domain of interest, several sorts of systems have been proposed. These include taxonomies and thesauri. In this section the differences between those and the ontologies are described.

A taxonomy "classifies terms hierarchically, using the father-son (generalization, is-a, or type-of) relationship” [17]. This means that the relations between the concepts are restricted to those mentioned above and that others, such as part-of, cause-effect or association are excluded. Taxonomies are used in systems which categorize objects, for example by libraries to catalogue their collections. An example of a taxonomy is also the Porphyry Tree presented in Fig. 4.2.

Thesauri extend the taxonomies’ expressiveness with ”a set of semantic relationships, such as equivalence, inverse, and association, that hold among the concepts” [17]. The relations allowed in a thesaurus are finite and well defined. Thesauri may be used ”to guarantee consistency across multiple databases, which facilitates indexing and retrieval” [17].

14See http://www.w3.org/2005/Incubator/rdb2rdf/.
<table>
<thead>
<tr>
<th>Name</th>
<th>Implementation</th>
<th>RDFS / OWL Support</th>
<th>Storing</th>
<th>Interface</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>3store</td>
<td>C</td>
<td>-</td>
<td>MySQL</td>
<td>RDQL, C API, BERKELEY DB</td>
<td><a href="http://sourceforge.net/projects/threestore/">http://sourceforge.net/projects/threestore/</a></td>
</tr>
<tr>
<td>Berkeley DB</td>
<td>SPARQL</td>
<td>Memory</td>
<td>RDFS</td>
<td>Java</td>
<td><a href="http://www.aktors.org/technologies/3store/">http://www.aktors.org/technologies/3store/</a></td>
</tr>
<tr>
<td>Redland</td>
<td>C</td>
<td>Memory</td>
<td>RDFS</td>
<td>Java</td>
<td><a href="http://librdf.org/">http://librdf.org/</a></td>
</tr>
<tr>
<td>ARC</td>
<td>Java</td>
<td>Memory</td>
<td>RDFS</td>
<td>Java</td>
<td><a href="http://arc.semsol.org/">http://arc.semsol.org/</a></td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>PHP</td>
<td>Memory</td>
<td>RDFS</td>
<td>Java</td>
<td><a href="http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/">http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/</a></td>
</tr>
<tr>
<td>AllegroGlass</td>
<td>SPARQL</td>
<td>Memory</td>
<td>RDFS</td>
<td>PHP</td>
<td><a href="http://e-culture.multimedian.nl/software/ClioPatria.shtml">http://e-culture.multimedian.nl/software/ClioPatria.shtml</a></td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>PHP</td>
<td>Memory</td>
<td>RDFS</td>
<td>PHP</td>
<td><a href="http://www.arcangolo.ai/">http://www.arcangolo.ai/</a></td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>PHP</td>
<td>Memory</td>
<td>RDFS</td>
<td>PHP</td>
<td><a href="http://www.arcangolo.ai/">http://www.arcangolo.ai/</a></td>
</tr>
<tr>
<td>AllegroGlass</td>
<td>PHP</td>
<td>Memory</td>
<td>RDFS</td>
<td>PHP</td>
<td><a href="http://www.arcangolo.ai/">http://www.arcangolo.ai/</a></td>
</tr>
<tr>
<td>Oracle 11g</td>
<td>Java</td>
<td>RDFS, OWL Full</td>
<td>MySQL</td>
<td>PHP</td>
<td><a href="http://www.oracle.com/technology/tech/oracledatabase/owlsupport/owlfull">http://www.oracle.com/technology/tech/oracledatabase/owlsupport/owlfull</a> support/</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>Java</td>
<td>RDFS, OWL Full</td>
<td>MySQL</td>
<td>PHP</td>
<td><a href="http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/">http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/</a></td>
</tr>
</tbody>
</table>
Figure 6: The tree of Porphyry

Probably the most popular thesaurus nowadays is WordNet – a lexical database for the English language, where each word is associated with one or more *senses* (definitions) and other words. WordNet uses the typical thesaurus relations, including equivalence (for synonyms), associative (for related term) and hierarchical (for broader or narrower terms).

### 4.3 Ontologies Classification

Ontologies may be classified under various perspectives. In [42] McGuinness proposed a Spectrum Classification, in which complexity and sophistication of the elements of the ontologies are taken into account. An illustration of "spectrum of definitions" is shown in Fig. 7

Figure 7: McGuinness ontology spectrum classification [42].

The picture illustrates different interpretations of ontologies. They range from controlled vocabularies (a list of terms used e.g. in catalogs), glossaries (a list of terms and meanings), thesauri (with the additional semantic relations), informal term hierarchies, to strict subclass hierarchies and systems including formal instance relationships, frame systems, and finally value restrictions and most
expressive ontologies allowing for first order logic constraints.

In order to consider something an ontology, McGuinness requires the following properties to hold:

- Finite controlled (extensible) vocabulary,
- Unambiguous interpretation of classes and term relationships, and
- Strict hierarchical subclass relationships between classes.

Systems conforming to the above requirements are shown to the right from the separating line on the picture.

Ontologies may also be classified based on their generality. **Top-level ontologies** describe general concepts like space, time, matter, object, event, action, etc. These concepts are independent of a particular problem or domain. Top-level ontologies should be unified for large communities of users and enable for constructing new ontologies based on them. **Domain ontologies** and **task ontologies** describe the vocabulary related to a generic domain (like medicine, or automobiles) or a generic task or activity (like diagnosing or selling), respectively. They specialize the terms introduced in the top-level ontology. "**Application ontologies** describe concepts depending both on a particular domain and task, which are often specializations of both the related ontologies" [31].

### 4.4 Ontology Languages for the Semantic Web

In general, there exist a variety of languages which allow for explicit specification of the universe of discourse. These include graphical notations, such as semantic networks, topic maps, UML diagrams or RDF graphs. A number of them are based on logics of different expressiveness level. These include languages based on Description Logics [5] (e.g., OIL, DAML+OIL, OWL), rules (e.g., RuleML, Logic Programs/Prolog), First Order Logic (e.g., KIF) and higher order logics (e.g., LBase). Non-classical logics (e.g., F-logic [40], modalities) are also being analyzed, as well as probabilistic and fuzzy approach. Degree of formality varies widely; increased formality makes languages more amenable to machine processing (e.g., automated reasoning) [8], but may make them more difficult to learn.

The classification in ontologies may vary, because there is no universal definition *what a class is*. Ontologies developed for different reasons and in various disciplines may demonstrate different approaches. Classes (terms) may be defined by sets of attributes the objects that belongs to the class have, or by similarity of individuals being instances of the same class. In the context of the Semantic Web the classification "should take into consideration the automation possibilities, and not the way humans organize their own knowledge" [17]. The main goal of the Semantic Web ontologies is to allow software agents and applications to share data in a significant way by providing a common model. Therefore, the "classical vision" approach has been chosen and ontologies categorize concepts based on their common characteristic. An object is an instance of a class iff it possesses all the properties or attributes specified by the concept description.

In order to develop a successful language for the Web, a set of requirements has been proposed in [33] and later refined in [32]. The major points have been summarized in [17]. These include:

- **XML-compatibility**, which means that a language should have serialization syntax, and that it should use XML Schema datatypes,
- provision of Description Logics constructs, such as concepts (classes), roles (properties) and individuals,
- identification of ontology and its vocabulary definitions with use of URI references,
- **ability of a language to develop ontologies in a distributed manner, versioning and reuse of ontologies.**
Moreover, the design of the language should extend existing Web standards such as XML or RDF, be easy to understand and use, yet formally specified and based on familiar Knowledge Representation idioms.

Several ontology languages have been designed for the Semantic Web. They differ in terms of expressiveness and computational tractability. A language built on top of the Resource Description Framework is RDF Schema. It is described in more details in Section 5. After identifying the above mentioned requirements next two languages were developed. Ontology Interface Layer (OIL) was designed by a group of (largely) European researchers (several from EU OntoKnowledge project). DARPA Agent Markup Language (DAML) was developed by a group of (largely) US researchers (in DARPA DAML programme). The efforts were later merged to produce DAML+OIL. Development was carried out by ”Joint EU/US Committee on Agent Markup Languages”. DAML+OIL was submitted to W3C as basis for standardization. A Web-Ontology (WebOnt) Working Group was formed. This group developed OWL language based on DAML+OIL. OWL language is now a W3C Recommendation. It is based on Description Logics. This formalism is described in Section 6. More on the OWL language can be found in Section 7.

5 RDF Schema

5.1 Basic Ideas

RDFS provides a frame-like formalism (see [45]). It extends RDF with schema vocabulary such as `subClass`, `subProperty`, `type` or `domain` and `range` and allows for specifying relations between properties of the resources. “RDF offers enormous flexibility but, apart from the `rdf:type` property, which has a predefined semantics, it provides no means for defining application-specific classes and properties. RDFS allows for defining classes and properties and hierarchies” [17]. The RDF Schema language has been designed to facilitate modelling the domain. Unlike the case of XML and XML Schema, RDF Schema does not define RDF grammar, but introduces new vocabulary and semantics. In RDF one can express arbitrary relations between certain individual resources, but is unable to say anything about general relations among classes of individuals. RDF Schema allows for describing classes and properties and restrict their use (for instance by defining domain and range of a property). It enables for introducing class hierarchy and therefore, use of inheritance. Hierarchical relationships can also be defined for properties. The RDFS layer enriches the semantic description of a set of resources, despite its not being a fully formalized language.

5.2 Syntax and Semantics

The RDFS language is based on RDF and follows its syntactic rules (see Sect. 1.3 in Chapter 1). RDFS document is a RDF one extended with RDFS primitives.

In particular, in RDF Schema

A class is any resource having an `rdf:type` property whose value is the qualified name `rdfs:Class` of the RDF Schema vocabulary. A class C is defined as a subclass of a class D by using the predefined `rdfs:subClassOf` property. This property is transitive in RDF Schema. Examples:

- `<Person,type,Class>`
- `<Professor,subClassOf,Person>`

An instance of a class C is a resource having an `rdf:type` property whose value is C. A resource may be an instance of more than one class. Examples:

- `<Carole,type,Professor>`
Table 3: Core RDFS classes and properties

<table>
<thead>
<tr>
<th>Core Classes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:Resource</td>
<td>class of all resources</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>class of all classes</td>
</tr>
<tr>
<td>rdfs:Literal</td>
<td>class of all literals</td>
</tr>
<tr>
<td>rdf:Property</td>
<td>class of all properties</td>
</tr>
<tr>
<td>rdfs:Statement</td>
<td>class of all reified statements (see [44])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
<td>relates a resource to its class</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>relates a class to one of its superclass</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>relates a property to one of its superproperties</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>specifies the domain of a property</td>
</tr>
<tr>
<td>rdf:range</td>
<td>specifies the range of a property</td>
</tr>
</tbody>
</table>

A property is any instance of the class rdfs:Property. The rdfs:domain property is used to indicate that a particular property applies to a designated class, and the rdfs:range property restrict the values of a particular property to instances of a designated class or, alternatively, to XML Schema datatypes. Examples:

- <hasColleague,type,Property>
- <hasColleague,range,Person>
- <hasColleague,domain,Person>

Core RDFS classes and properties (as presented in [3]) are collected in Table 3:

Semantics of RDF Schema is given by RDF Model Theory (MT) as presented in [8]. In RDF Model Theory, an interpretation $I$ of a vocabulary $V$ consists of:

- $IR$, a non-empty set of resources
- $IS$, a mapping from $V$ into $IR$
- $IP$, a distinguished subset of $IR$ (the properties)
- $IEXT$, a mapping from $IP$ into the powerset of $IR \times IR$
- $IL$, a mapping from typed literals into $IR$

Class interpretation $ICEXIT$ is simply induced by $IEXIT(IS(type))$:

$$ICEXIT(C) = \{x | (x, C) \in IEXIT(IS(type))\}$$  \hspace{1cm} (1)

An example interpretation is shown in Figure 8.

RDFS semantics is non-standard and it is thus difficult to provide a native reasoning support. One can observe it in Fig. 8. However, it may be possible to reason via First Order axiomatization as shown in [3]. Instead of using full Predicate Calculus reasoning, it is possible to reason by an application of set of reasoning rules. In this case, the inference system consists of rules of the form:

IF \hspace{1cm} E contains certain triples
THEN \hspace{1cm} add to E certain additional triples
where \( E \) is an arbitrary set of RDF triples (see [3] for details).

RDFS is very "liberal" [8]. For instance, there is no distinction between classes and instances (individuals). Properties can themselves have properties. Moreover, no distinction between language constructors and ontology vocabulary exists, so constructors can be applied to themselves and to each other. An example consequence of this liberality is presented in [3]: "\texttt{rdfs:Class} is a subclass of \texttt{rdfs:resource} (every class is a resource), and \texttt{rdfs:Resource} is an instance of \texttt{rdfs:Class} (\texttt{rdfs:Resource} is the class of all resources, so it is a class!). For the same reason, every class is an instance of \texttt{rdfs:Class}" [3].

At the same time, the expressiveness of RDFS is limited. Both RDF and RDFS lack any notion of negation or disjunction and have limited notion of existential quantification. The inconsistency cannot be expressed, neither in RDF nor RDFS. The limitations may sometimes be an advantage. For instance, it has turned out to be practical to perform an exhaustive forward-chaining inference in applications based on large RDF Triple Stores (databases designed to store and retrieve RDF triples). This is possible due to the weakness of the languages. Languages with richer expressiveness suffer from an exponential computational explosion when computing the entire deductive closure of asserted knowledge.

RDF and RDFS are not sufficient for the Semantic Web as general. They are too weak to describe resources precisely enough. No localized range and domain constraints mean that a certain property cannot be used with different domains and ranges when applied to different classes (e.g. one cannot say that the range of \texttt{hasChild} is a person when applied to persons and an elephant when applied to elephants [8]). The lack of existence or cardinality constraints or defining properties as transitive, inverse or symmetrical makes the knowledge representation insufficient in some applications and prevents from conducting an advanced inference.
### 6 Description Logics

#### 6.1 Introduction to Description Logics

Description Logics are a family of knowledge representation languages. Historically related to semantic networks and frame languages, they describe the world of interest by means of concepts, individuals and roles. Concepts are interpreted as sets of objects, individuals as objects in the universe of discourse, and roles as binary relations between objects of the world. Contrary to their predecessors, DL provide a formal semantics and automated reasoning capabilities.

In Description Logics one can make statements about individual objects or concepts, interpreted as sets of individuals. Statements about individuals may be of two kinds:

- **concept assertions**, e.g.
  - `Person(fred)` - Fred is a person
  - `Cat(tibbs)` - Tibbs is a cat
- **role assertions**, e.g.
  - `isAPetOf(tibbs, fred)` - Tibbs is a Fred’s pet
  - `hasPet(fred, tibbs)` - Fred has a pet which is Tibbs

Statements about concepts are:

- **concept definitions**, which state necessary and sufficient conditions, e.g.
  - `Man ≡ Person ⊓ Adult ⊓ Male` - A man is an adult male person
  - `Cat_liker ≡ Person ⊓ ∃likes.Cat` - A cat liker is a person and there exists a cat that he/she likes
- **relations between concepts**, e.g.
  - `Cat_liker ⊑ ∃likes.Cat` - (every) cat liker likes a cat
  - `Sheep ⊑ ∀eats.Grass` - (every) sheep eats only grass
- **General Concept Inclusion (GCI) axioms** e.g.
  - `Cat ⊑ Animal` - a cat is an animal (hierarchy of concepts)
  - `Sheep ⊑ Animal ⊓ ∀eats.grass` - a sheep is an animal which eats only grass. It is worth mentioning that in this case only necessary, but not sufficient conditions are expressed. This expression does not define a concept, but constraints the way the concept or the role is interpreted.

There exist a number of DL languages. They are defined and distinguished by the concept descriptions they allow, which influent the languages’ expressiveness. This will be discussed in more details in Sect. 6.2.3 and Sect. 6.3.

#### 6.1.1 Relation of Description Logics to Other Formalisms

**Semantic Networks and Frames** The ancestor of DL systems, KL-ONE [15], was the first implementation of the so-called structured inheritance networks and it used the concepts of semantic networks [45]. In KL-ONE knowledge was represented in a form of a terminological network (see Fig. 9). Concepts were represented as ovals, and the arcs between the ovals represented various relations, so-called roles.

Simple hierarchical relations in frame languages evolved into complex descriptions in DL and became more formal. "In the case of frames, the KB designer could create the hierarchy in any arbitrary
way desired, simply by adding whatever IS-A and INSTANCE-OF slot-fillers seemed appropriate. However, with DLs the logic of concepts dictates what each concept means, as well as what must be above or below it in the resulting taxonomy” [14].

**First Order Logic**  Most of the Description Logic languages are subsets of the Predicate Calculus. Concept names correspond to unary predicates and role descriptions to binary ones. Concept descriptions can be generally seen as formulae with one free variable and role expressions as two-variables-formulae (see Table 4).

Description Logics take advantage of their relation to Predicate Calculus. On the one hand they adopt its semantics, which makes them more expressive than a Propositional Logic. On the other, by restricting the syntax to formulae with maximum two variables, they remain decidable and more human-readable. Description Logics are “carefully tailored such that they combine interesting means of expressiveness with decidability of the important reasoning problems” [6].

Mapping of some extensions to basic DL such as transitive closure of roles or fixpoint semantics in cyclic terminologies require second-order logic (see [5] for details).

**Modal Logics**  Close relation of Description Logics to modal logics ”was discovered relatively late, but has since the been exploited quite successfully to transfer complexity and decidability results as well as reasoning techniques” [6]. It has been shown that DL concepts may be transformed to modal
Table 5: Description Logics and set algebra

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>Set algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 \sqcap ... \sqcap C_n$</td>
<td>$C_1 \cap ... \cap C_n$</td>
</tr>
<tr>
<td>$C_1 \sqcup ... \sqcup C_n$</td>
<td>$C_1 \cup ... \cup C_n$</td>
</tr>
<tr>
<td>$\neg C$</td>
<td>$C^c$ (complement of set)</td>
</tr>
<tr>
<td>${x_1} \sqcup ... \sqcup {x_n}$</td>
<td>${x_1} \cup ... \cup {x_n}$</td>
</tr>
<tr>
<td>$\forall P.C$</td>
<td>$\pi_Y(P) \subseteq C$</td>
</tr>
<tr>
<td>$\exists P.C$</td>
<td>$\pi_Y(P) \cap C \neq \emptyset$ (projection)</td>
</tr>
<tr>
<td>$\leq n P$</td>
<td>$</td>
</tr>
<tr>
<td>$\geq n P$</td>
<td>$\text{card}(P) \geq n$</td>
</tr>
<tr>
<td>$C \equiv D$</td>
<td>$C \equiv D$</td>
</tr>
<tr>
<td>$C \subseteq D$</td>
<td>$C \subseteq D$</td>
</tr>
</tbody>
</table>

K formulas.

Set Algebra  In Description Logics concepts are interpreted as sets of individuals. Therefore, one can express the meaning of DL descriptions in set algebra notation (see Table 5).

6.1.2 Evolution of DL and DL-Based Systems

Development of Description Logics techniques and applications has gone through several phases [6]. The first DL system was KL-ONE mentioned in Section 6.1. It “signalled the transition from semantic networks to more well-founded terminological (description) logics” [5].

In the subsequent phase of DL-based systems implementation, the so-called structural subsumption algorithms (see Section 6.5.2) were used. The first industrial-strength DL system was CLASSIC [55]. At the time it was developed a formal investigation of the complexity of reasoning in DLs had been undertaken. Among other results it was shown that seemingly small addition to the expressive power of a language may make subsumption intractable. For example a general negation is a problem for structural subsumption algorithms. The CLASSIC system’s implementors reacted to this by restricting the expressive power of the DL underlying the system.

Tableau-based algorithms and discovery of close relation to modal logic caused a growth in interest in application of Description Logics. Inference procedures for very expressive DLs based on tableau approach were developed and translations to modal logics were used to optimize the DL systems.

Today there exist some industrial-strength DL systems, the logic is extensively used in the Semantic Web, medical informatics and bio-informatics. Also, there has been a shift from DL-centered systems to ones that have a DL component. A visible trend is research on less expressive DLs [7], with applications in tools to operate on large knowledge bases and/or assertional knowledge bases (see [6] for further references).

6.1.3 Applications of DL and DL-Based Systems

DLs are used as general purpose languages for knowledge representation and reasoning. As a modelling formalisms DL overlap with other languages from the fields of programming and database management, but seem superior in their providing reasoning capabilities.

Description Logics have been mainly used to model “terminological knowledge of an application domain in a structured way” [6] in various application domains. One of the first was Software Engineering with a prime example of the CLASSIC – Software Information System. The other is
configuration, where DL are used in the design process of complex systems created by multiple components. Medicine is another domain with the main application of DL in decision support and large ontologies. Natural Language processing has taken advantage of DL by employing them to construct ontologies. Applications in database management have been investigated for a long time and DL are used in several ways. They proved useful in reasoning about ERD (Entity Relationship Diagrams) as well as query processing and optimization and reasoning with and about views. Description Logics constitute a formal foundation for ontology languages such as OIL, DAML+OIL, OWL. They have even been used for describing dynamic systems.

"There is currently a great deal of interest in the idea of combining DLs with other KR formalisms. Important contributions in this area include work on rule support in the CLASSIC system, the integration of Datalog with DLs (...), the integration of answer set programming with DLs, and the extension of DLs with so-called DL-safe rules” [6].

6.2 The Main Formalism - $\mathcal{L}$ Languages Family

The building blocks of vocabulary in Description Logic languages are concepts, which denote sets of individuals, and roles, which denote the binary relations between individuals.

Elementary descriptions in Description Logics are atomic concepts and atomic roles. More complex descriptions can be built inductively from them using concept constructors. Respective DL languages are distinguished by the constructors they provide. The minimal language of practical interest is the Attributive Language ($\mathcal{AL}$).

6.2.1 $\mathcal{AL}$ Syntax

Let $A$ denote an atomic concept and $R$ an atomic role. In basic $\mathcal{AL}$ concept descriptions $C$ and $D$ can be in one of the following forms:

- $A$ atomic concept (2)
- $\top$ universal concept (3)
- $\bot$ bottom concept (4)
- $\neg A$ atomic negation (5)
- $C \sqcap D$ intersection (6)
- $\forall R.C$ value restriction (7)
- $\exists R.\top$ limited existential quantification (8)

It should be noted that basic $\mathcal{AL}$ allows only atomic negation and limited existential quantification. It means that negation can only be applied to atomic concepts, and only the top concept is allowed in the scope of an existential quantification.

6.2.2 $\mathcal{AL}$ Semantics

In order to define a formal semantics, an interpretation $I = (\Delta^I, \cdot^I)$ is considered. The interpretation consists of a non-empty set $\Delta^I$ the domain of interpretation and an interpretation function, which assigns to every atomic concept $A$ a set $A^I \subseteq \Delta^I$ and to every atomic role $R$ a binary relation $R^I = R^I \subseteq \Delta^I \times \Delta^I$.

The interpretation function is further extended over concept descriptions by the following defini-
### Table 6: \(\mathcal{AL}\) languages family syntax and semantics

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Symbol</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>(A)</td>
<td>(A^I = A^I \subseteq \Delta^I)</td>
<td></td>
</tr>
<tr>
<td>atomic role</td>
<td>(R)</td>
<td>(R^I = R^I \subseteq \Delta^I \times \Delta^I)</td>
<td></td>
</tr>
<tr>
<td>universal concept</td>
<td>(\top)</td>
<td>(\top^I = \Delta^I)</td>
<td></td>
</tr>
<tr>
<td>bottom concept</td>
<td>(\bot)</td>
<td>(\bot^I = \emptyset)</td>
<td></td>
</tr>
<tr>
<td>atomic negation</td>
<td>(\neg A)</td>
<td>((\neg A)^I = \Delta^I \setminus A^I)</td>
<td></td>
</tr>
<tr>
<td>intersection</td>
<td>(C \cap D)</td>
<td>((C \cap D)^I = C^I \cap D^I)</td>
<td></td>
</tr>
<tr>
<td>value restriction</td>
<td>(\forall R.C)</td>
<td>((\forall R.C)^I = {a \in \Delta^I</td>
<td>\forall b, (a, b) \in R^I \rightarrow b \in C^I})</td>
</tr>
<tr>
<td>limited existential quantification</td>
<td>(\exists R.\top)</td>
<td>((\exists R.\top)^I = {a \in \Delta^I</td>
<td>\exists b, (a, b) \in R^I})</td>
</tr>
<tr>
<td>union</td>
<td>(U)</td>
<td>(C \sqcup D)</td>
<td>((C \sqcup D)^I = C^I \cup D^I)</td>
</tr>
<tr>
<td>full negation</td>
<td>(\neg C)</td>
<td>((-C)^I = \Delta^I \setminus C^I)</td>
<td></td>
</tr>
<tr>
<td>full existential quantification</td>
<td>(\exists R.C)</td>
<td>((\exists R.C)^I = {a \in \Delta^I</td>
<td>\exists b, (a, b) \in R^I \land b \in C^I})</td>
</tr>
<tr>
<td>number restrictions (\geq n)</td>
<td>(\geq n)</td>
<td>((\geq n)^I = {a \in \Delta^I</td>
<td>{b</td>
</tr>
<tr>
<td>number restrictions (\leq n)</td>
<td>(\leq n)</td>
<td>((\leq n)^I = {a \in \Delta^I</td>
<td>{b</td>
</tr>
</tbody>
</table>

### 6.2.3 Other \(\mathcal{AL}\) Languages

The basic language can be extended by allowing other concept constructors, such as union \((U)\), full negation \((C)\), full existential quantification \((E)\) or number restriction \((N)\). Resulting formalisms are called using the letters indicating the allowed constructors, for example \(\mathcal{ALC}\), \(\mathcal{ALCN}\), \(\mathcal{ALUE}\) etc. A smallest propositionally closed language is \(\mathcal{ALC}\) (equivalent to \(\mathcal{ALUE}\)). Concepts constructed in \(\mathcal{ALC}\) may use boolean symbols: \(\sqcup, \sqcap, \neg\), restricted quantifiers: \(\exists, \forall\). Role descriptions are restricted to atomic roles only. Syntax and semantics of \(\mathcal{AL}\) languages is summarized in Table 6.

### 6.3 Language Extensions

Basic Description Logics allow only atomic roles (i.e. role names) in role descriptions. Different extensions to basic Description Logics are introduced by allowing role constructors. They enable for introducing various constraints and properties of roles, such as transitive closure, intersection, composition and union or complement and inverse roles.

Another kinds of extension are obtained by allowing nominals in concept definitions or introducing primitive datatypes. These modifications proved extremely valuable and important in the context of the Semantic Web and ontologies built for it. However, they are sources of high computational complexity of reasoning in the resulting ontologies. Some extensions to basic DL languages are presented in Table 7.
Table 7: Extensions to the basic DL

<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Extension</th>
<th>Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept constructors</td>
<td>functional number restrictions</td>
<td>$F$</td>
<td>$\leq 1$ hasMother</td>
</tr>
<tr>
<td></td>
<td>number restrictions</td>
<td>$N$</td>
<td>$\geq 2$ hasChild, $\leq 3$ hasChild</td>
</tr>
<tr>
<td></td>
<td>qualified number restrictions</td>
<td>$Q$</td>
<td>$\geq 2$ hasChild.Doctor</td>
</tr>
<tr>
<td></td>
<td>nominals/singleton classes</td>
<td>$O$</td>
<td>${Italy}$</td>
</tr>
<tr>
<td>role constructors</td>
<td>role intersection</td>
<td>$\cap$</td>
<td>hasMother $\cap$ hasFather</td>
</tr>
<tr>
<td></td>
<td>role union</td>
<td>$\cup$</td>
<td>hasSon $\cup$ hasDaughter</td>
</tr>
<tr>
<td></td>
<td>complement roles</td>
<td>$\neg$</td>
<td>$\neg$ hasCat</td>
</tr>
<tr>
<td></td>
<td>chain (composition) of roles</td>
<td>$\circ$</td>
<td>hasCat $\circ$ isMadAbout</td>
</tr>
<tr>
<td></td>
<td>inverse roles</td>
<td>$I$</td>
<td>isChildOf $\equiv$ hasChild$^-$</td>
</tr>
<tr>
<td>additional</td>
<td>role transitivity</td>
<td>$S$</td>
<td>isAncestor</td>
</tr>
<tr>
<td>role axioms</td>
<td>role hierarchy</td>
<td>$H$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td></td>
<td>complex role inclusions</td>
<td>$R$</td>
<td>know $\circ$ hasFriends $\subseteq$ know</td>
</tr>
<tr>
<td>other</td>
<td>use of datatype properties, data</td>
<td>(D)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>values or data types</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For expressive DLs the naming convention introduced in Section 6.2.3 would be too long. Therefore, for the basic $\text{ALC}$ language extended with transitive roles, the letter $S$ is often used. The letter $H$ is used to represent role hierarchy, $O$ to indicate nominals in concept descriptions, $T$ denotes inverse roles, $N$ represents number restrictions and $Q$ – qualified number restrictions. The letter $D$ in parenthesis indicates the integration of some concrete domain/datatypes. The DL underlying OWL DL language includes all of those constructs and is therefore called $\text{SHOIN(D)}$. OWL 2 DL is a very expressive language based on $\text{SROIQ(D)}$ Description Logic.

Variant languages of Description Logics have been thoroughly investigated in terms of their reasoning complexity. A number of DL languages, from less expressive but tractable (in polynomial time), to very expressive ones have been closely analyzed and the results have been used in practical applications of DL systems. Naturally, the more expressive a language is, the more complex is the reasoning over it. A useful tool comprising studies about different DL is called Complexity Navigator maintained by Evgeny Zolin (see Fig. 10).\textsuperscript{15}

6.4 Knowledge Representation with Description Logics – Terminology and World Description

Description Logics provide tools to build a Knowledge Base and to reason over it. In systems based on DL, the Knowledge Base consist of two parts, namely TBox and ABox (see Figure 11).

TBox provides a terminology and contains a taxonomy expressed in a form of set of axioms. They define concepts (interpreted as set of individuals) and specify relations between them. Therefore, a TBox stores an implicit knowledge about individuals in the world of interest. Formally, a terminology $T$ is a finite set of terminological axioms. If $C$ and $D$ denote concept names, and $R$ and $S$ role names, then the terminological axioms may be in two forms: $C \sqsubseteq D (R \sqsubseteq S)$ or $C \equiv D (R \equiv S)$. Equalities that have an atomic concept on the left-hand side are called definitions. Axioms of the form $C \sqsubseteq D$ are called specialization statements. Equalities express necessary and sufficient conditions, whereas specialization statements specify constraints (necessary conditions) only.

An interpretation (function) $I$ maps each concept name to a subset of the domain. The interpretation satisfies an axiom $C \sqsubseteq D$ if: $C^I \subseteq D^I$. It satisfies a concept definition $C \equiv D$ if: $C^I = D^I$. If the interpretation satisfies all the definitions and all axioms in $T$, it satisfies the terminology $T$ and

\textsuperscript{15}See http://www.cs.manchester.ac.uk/~ezolin/dl/.
is called a model of $T$.

The other part of Knowledge Base is an ABox. This part contains explicit assertions about individuals in the conceived world. They represent extensional knowledge about the domain of interest. Statements in ABox may be: concept assertions, e.g. $C(a)$ or role assertions, $R(b,c)$. An interpretation $I$ maps each individual name to an element in the domain. With regards to terminology $T$, the interpretation satisfies a concept assertion $C(a)$ if $a \in I$ and a role assertion $R(b,c)$ if $\langle b, c \rangle \in I$.

Although terminology and world description share the same model-theoretic semantics, it is convenient to distinguish these two parts while designing a Knowledge Base or stating particular inference tasks.

### 6.5 Reasoning in Description Logics

#### 6.5.1 Inference Tasks for Description Logics

Inference in Description Logics can be separated into reasoning tasks for TBox and ABox. With regards to terminology $T$, one can pose a question whether a concept is satisfiable, one concept subsumes another, or if two concepts are equivalent or disjoint.

A concept $C$ is satisfiable with respect to $T$ if there exists a model $I$ of $T$ such that $C^I$ is not empty.

A concept $C$ is subsumed by a concept $D$ w.r.t. $T$ if $C^I \subseteq D^I$ for every model $I$ of $T$.

Two concepts $C$ and $D$ are equivalent w.r.t. $T$ if $C^I = D^I$ for every model $I$ of $T$. 

---

Figure 10: Complexity navigator.
Finally, two concepts $C$ and $D$ are disjoint w.r.t. $T$ if $C^I \cap D^I = \emptyset$ for every model $I$ of $T$.

Satisfiability and subsumption checking are the main reasoning tasks for TBox. All the tasks can be reduced to them, and also either can be reduced to the other. For instance, $C$ and $D$ are disjoint iff $C \sqcap D$ is subsumed by $\bot$. $C$ is subsumed by $D$ iff $C \sqcap \neg D$ is unsatisfiable.

For ABox there exist four main inference tasks: consistency checking, instance checking, realization and retrieval.

An ABox $A$ is consistent w.r.t. a TBox $T$, if there exists an interpretation that is a model of both $A$ and $T$. Furthermore, we say that an ABox is consistent, if it is consistent w.r.t. the empty TBox (see [5]).

Instance checking tests whether an assertion is entailed by the ABox.

Realization task consists in finding the most specific concept for a given individual, and

Retrieval returns individuals which are instances of a given concept.

All the task for both ABox and TBox can be reduced to consistency checking of the ABox with respect to the TBox.

6.5.2 Reasoning Algorithms for Description Logics

Structural subsumption algorithms are ones that normalize the concept descriptions and recursively compare the syntactic structure of the normalized descriptions. Their main advantage is the high efficiency – they finish in polynomial time. However, these algorithms are only complete for inexpressive DL. In particular, they cannot handle DL with full negation and union constructors [5].

An alternative is constituted by algorithms based on tableau approach. Tableau algorithms take advantage of possibility of reducing reasoning tasks for DL, in particular the fact that $C$ is subsumed by $D$ iff $C \sqcap \neg D$ is unsatisfiable. They start from ground facts (ABox axioms). Then apply tableaux expansion rules, which correspond to constructors in logic ($\sqcap$, $\sqcup$, etc.). Some rules are nondeterministic, e.g. $\sqcup$ or $\leq$. In practice, this requires undertaking search.

Other reasoning techniques include automata based approach described in [6].

7 OWL – Web Ontology Language

OWL Web Ontology Language is currently a recommended language for development of ontologies for the Web. It has been designed based on previous experiences with languages like DAML, OIL,
Table 8: Computational complexity of chosen DL languages and their OWL counterparts

<table>
<thead>
<tr>
<th>OWL sublanguage</th>
<th>DL</th>
<th>Concept satisfiability</th>
<th>ABox consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWL Lite</td>
<td>SHIF</td>
<td>ExpTime-complete</td>
<td>ExpTime-complete</td>
</tr>
<tr>
<td>OWL DL</td>
<td>SHOIN</td>
<td>NExpTime-complete</td>
<td>NExpTime-complete</td>
</tr>
</tbody>
</table>

DAML+OIL, as well as on research on Description Logics. It seems impossible to fulfill all the requirements formulated for a web ontology language. "Simple extending RDF Schema would work against obtaining expressive power and efficient reasoning” [3]. RDFS has some very powerful primitives (e.g. rdfs:Class, rdfs:Property) which, when not controlled by some strict logical rules, would lead to serious computational problems.

7.1 OWL(1) Sublanguages

The problem has been partially solved by introducing three, increasingly expressive sublanguages of OWL(1): OWL Lite, OWL DL and OWL Full. Each of them gears towards different aspects of the full set of requirements. The least expressive OWL Lite is sufficient for taxonomies and thesauri design and migration, and is computationally simple. OWL DL enables creating more complex construct, though it still imposes some restrictions. These two sublanguages are based on Description Logic formalism and adopt its semantic and reasoning capabilities. OWL Full does not have its counterpart in DL. It can be viewed as an extension of RDF and RDFS, whereas OWL Lite and OWL DL are extensions of restricted forms of RDF [17].

Every OWL document (regardless of the OWL sublanguage) is an RDF/XML document. Every legal RDF/XML document is a legal OWL Full document, and any RDF(S) conclusion is also a valid OWL Full conclusion. However, not all RDF/XML documents are OWL Lite or OWL DL ones. RDF(S) is more expressive than OWL DL and OWL Lite, because its semantics imposes no restrictions on how resource can be related to each other. For instance, it is allowed in RDF for an object to be a class and an instance of a class, whereas in OWL Lite and OWL DL the sets resource identifiers which denote classes and individuals have to be disjoint.

In OWL DL and OWL Lite the use of constructs is restricted, so that the statements have their counterparts in Description Logics (see Table 9). It lets the languages take advantage of thorough investigation of computational properties of respective DL languages (see Table 8).

7.2 Syntax and Semantics

Several syntax conventions for OWL have been defined. The primary one is RDF’s XML-based syntax, machine-readable and used for exchange. However, other syntactic forms have been defined as well [3]:

- an XML-based syntax, more human-readable and not following RDF conventions [26],
- an abstract syntax which is more compact and readable [54],
- a graphic syntax based on Unified Modelling Language (UML). 16

Ontology in OWL usually starts with a section describing the ontology itself. The constructs in this section include versioning and comments, as well as import directives. The following part is modelled with use of classes (concepts in DL) and properties (roles in DL).

Classes are defined using an owl:Class elements (which is a subclass of rdfs:Class). Two predefined classes, owl:Thing and owl:Nothing are used to denote whole domain and an empty

set respectively. In class definitions one can use boolean combinations, like union or intersection. Depending on a language’s expressiveness explicit class definitions (using owl:oneOf construct) may be allowed or not.

Sample (and incomplete) definition of a class:

```xml
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
  ...
</owl:Class>

<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing" /> 
  ...
</owl:Class>
```

Properties are divided into Object Properties which relate objects to other objects, and Datatype Properties, which relate objects with data type values. Although OWL does not have its own datatypes, it allow for using XML Schema ones. Relations in which the object remains may be restricted with appropriate language constructs (see Table 9). Defining property restrictions, one can specify the domain and range:

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

A sequence of elements without an explicit operator represents an implicit conjunction. The property madeFromGrape has a domain of Wine and a range of WineGrape. That is, it relates instances of the class Wine to instances of the class WineGrape. Multiple domains mean that the domain of the property is the intersection of the identified classes (and similarly for range).

A property can be defined to be a specialization (subproperty) of an existing property using rdfs:subPropertyOf:

```xml
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

It is possible to specify property characteristics, which provides a powerful mechanism for enhanced reasoning about a property. Properties can be: transitive, symmetric, functional, inverse, or inverse functional (see [43] for details). One can add further restrictions using owl:allValuesFrom, owl:someValuesFrom, owl:cardinality, owl:hasValue.

Individuals and property instances are declared as in RDF Schema using rdf:type. Two different URIs may refer to the same individual and one can define it using owl:sameAs construct.

Sample description of an individual:
Table 9: OWL and DL syntax comparison

<table>
<thead>
<tr>
<th>OWL Constructor</th>
<th>DL syntax</th>
<th>OWL Axiom</th>
<th>DL syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thing</td>
<td>⊤</td>
<td>equivalentClass</td>
<td>C ≡ D</td>
</tr>
<tr>
<td>Nothing</td>
<td>⊥</td>
<td>subClassOf</td>
<td>C ⊆ D</td>
</tr>
<tr>
<td>complementOf</td>
<td>(¬)</td>
<td>equivalentProperty</td>
<td>S ≡ R</td>
</tr>
<tr>
<td>intersectionOf</td>
<td>(C ∩ D)</td>
<td>subPropertyOf</td>
<td>S ⊑ R</td>
</tr>
<tr>
<td>unionOf</td>
<td>(C ∪ D)</td>
<td>inverseOf</td>
<td>S ⊑ R</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>(∀R.C)</td>
<td>transitiveProperty</td>
<td>R⁺ ⊑ R⁻</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>(∃R.C)</td>
<td>functionalProperty</td>
<td>⊤ ⊑ ≤ 1R</td>
</tr>
<tr>
<td>minCardinality</td>
<td>(≥ nR)</td>
<td>inverseFunctionalProperty</td>
<td>⊤ ⊑ ≤ 1R⁻</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>(≤ nR)</td>
<td>sameIndividualAs</td>
<td>a = b</td>
</tr>
<tr>
<td>oneOf</td>
<td>{a₁,...,aₙ}</td>
<td>differentFrom</td>
<td>a ≠ b</td>
</tr>
</tbody>
</table>

More details about OWL can be found in its specifications ([61, 25, 43]) and the Semantic Web books [3, 17].

All the constructs may be used in OWL Full in any combinations, including redefinitions of concepts by applying constructors to themselves. In OWL DL the following constraints must be obeyed [3]:

1. Sets of classes, data types, data type properties, object properties, individuals, data types and built-in vocabulary must be pairwise disjoint.

2. The resource partitioning must be explicitly stated (e.g. if a class A is defined as a subclass of B, then B must also be defined as a class).

3. Datatype properties are disjoint with object properties, and it is impossible to characterize a datatype property as inverse, functional or symmetric.

4. No cardinality restrictions may be placed on transitive properties.

5. Anonymous classes are only allowed as the range of rdfs:range and as a domain or range of owl:equivalentClass or owl:disjointWith.

OWL Lite ontology must be a OWL DL one with further restrictions satisfied. The constructors: owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf and owl:hasValue are not allowed. Cardinality statements may only include values 0 and 1 (functional properties). The construct owl:equivalentClass must not be used with anonymous classes.

The semantics of OWL Lite and OWL DL follows the Description Logics (see Section 6.2.2). A comprehensive comparison presented in [51] is shown in Figures 12 and 13.

OWL Full is upward compatible with RDF and RDF Schema both with respect to syntax and semantics.

7.3 Web Ontology Language 2

OWL 2 is a new version of the language, published as a W3C Recommendation in October 2009. Underpinning Description Logic is called $SROIQ(D)$. The language is compatible with RDF and
<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>DL Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$ (URI Reference)</td>
<td>$A$</td>
<td>$A^2 \subseteq \Delta^2$</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>$\top$</td>
<td>owl:Thing$^\bot = \Delta^2$</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>$\bot$</td>
<td>owl : Nothing$^\bot = \emptyset$</td>
</tr>
<tr>
<td>intersectionOf($C_1, C_2$)</td>
<td>$C_1 \cap C_2$</td>
<td>$C_1 \cap C_2$</td>
</tr>
<tr>
<td>unionOf($C_1, C_2$)</td>
<td>$C_1 \cup C_2$</td>
<td>$C_1 \cup C_2$</td>
</tr>
<tr>
<td>complementOf($C$)</td>
<td>$\neg C$</td>
<td>$\Delta \setminus C$</td>
</tr>
<tr>
<td>oneOf(${o_1, \ldots}$)</td>
<td>${o_1, \ldots}$</td>
<td></td>
</tr>
<tr>
<td>restriction($R$ someValuesFrom($C$))</td>
<td>$\exists R.C$</td>
<td>${x \exists y \ (x, y) \in R^2 \land y \in C^2}$</td>
</tr>
<tr>
<td>restriction($R$ allValuesFrom($C$))</td>
<td>$\forall R.C$</td>
<td>${x \forall y \ (x, y) \in R^2 \land y \in C^2}$</td>
</tr>
<tr>
<td>restriction($R$ hasValue($o$))</td>
<td>$R : o$</td>
<td>${x (x, o^2) \in R^2}$</td>
</tr>
<tr>
<td>restriction($R$ minCardinality($n$))</td>
<td>$\geq nR$</td>
<td>${a \in \Delta^2 \mid {b \mid (a, b) \in R^2} \geq n}$</td>
</tr>
<tr>
<td>restriction($R$ maxCardinality($n$))</td>
<td>$\leq nR$</td>
<td>${a \in \Delta^2 \mid {b \mid (a, b) \in R^2} \leq n}$</td>
</tr>
<tr>
<td>restriction($U$ someValuesFrom($D$))</td>
<td>$\exists U.D$</td>
<td>${x \exists y \ (x, y) \in U^2 \land y \in D^2}$</td>
</tr>
<tr>
<td>restriction($U$ allValuesFrom($D$))</td>
<td>$\forall U.D$</td>
<td>${x \forall y \ (x, y) \in U^2 \land y \in D^2}$</td>
</tr>
<tr>
<td>restriction($U$ hasValue($v$))</td>
<td>$U : v$</td>
<td>${x (x, v^2) \in U^2}$</td>
</tr>
<tr>
<td>restriction($U$ minCardinality($n$))</td>
<td>$\geq nU$</td>
<td>${a \in \Delta^2 \mid {b \mid (a, b) \in U^2} \geq n}$</td>
</tr>
<tr>
<td>restriction($U$ maxCardinality($n$))</td>
<td>$\leq nU$</td>
<td>${a \in \Delta^2 \mid {b \mid (a, b) \in U^2} \leq n}$</td>
</tr>
</tbody>
</table>

Data Ranges ($D$)

<table>
<thead>
<tr>
<th>$D$ (URI reference)</th>
<th>$D$</th>
<th>$D^2 \subseteq \Delta_D^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>oneOf(${d_1, \ldots}$)</td>
<td>${d_1, \ldots}$</td>
<td>${d_1, \ldots}$</td>
</tr>
</tbody>
</table>

Object Properties ($R$)

<table>
<thead>
<tr>
<th>$R$ (URI reference)</th>
<th>$R$</th>
<th>$\Delta^2 \times \Delta^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg R$</td>
<td>$(\neg R)$-</td>
<td></td>
</tr>
</tbody>
</table>

Datatype Properties ($U$)

<table>
<thead>
<tr>
<th>$U$ (URI reference)</th>
<th>$U$</th>
<th>$U^2 \subseteq \Delta^2 \times \Delta_D^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals ($o$)</td>
<td>$o$</td>
<td>$o^2 \in \Delta^2$</td>
</tr>
</tbody>
</table>

Data Values ($v$)

| $v$ (RDF literal) | $v^2$ | $v^2$ |

Figure 12: OWL and DL descriptions [51].

RDFS. It has various syntaxes defined [49]. Its semantics [48] is fully declarative, and adheres to Open World Assumption. The language is expressive yet decidable and efficient for real knowledge bases.

Several new constructs or features have been defined, such as property chain constructor, Self property and keys as in databases. Some useful materials and OWL 2 tutorials may be found at: http://semantic-web-book.org/page/Slides.

OWL 2 defines OWL 2 DL and OWL 2 Full. Moreover, there exist three new profiles. They replace OWL(1) Lite, which proved to be inexpressive, but almost as computationally complex as OWL DL. The design principle for OWL 2 profiles was to identify maximal OWL 2 sublanguages that can be reason about in polynomial time.

**OWL 2 EL**

OWL 2 EL is based on $\mathcal{EL}++$ Description Logic [4]. It focuses on terminological expressiveness used for light-weight ontologies. It allows defining property domains, class and property hierarchies, class intersections, disjoint classes/properties, property chains, Self property, nominals and keys. It disallows inverse and symmetric properties, as well as negation and disjunction.

OWL 2 EL is used in large ontologies, with a prime example of SNOMED CT.\(^{17}\) Reasoning for

\(^{17}\)See http://www.ihtsdo.org/snomed-ct/.
OWL 2 EL is PTime-complete and there exist fast implementations for it.

**OWL 2 QL**

OWL 2 QL has been designed to facilitate ontology querying. It defines different restrictions on subclasses and superclasses, as well as restrictions of property ranges and domains. The profile disallows disjunction, universal quantification, Self property, keys, nominals, equality, property chains, transitive properties, cardinalities or functional properties.

Standard reasoning in OWL 2 QL is PTime, instance retrieval is LogSpace. There exist fast
implementations on top of legacy database systems (relational or RDF), highly scalable to very large datasets.

**OWL 2 RL.** OWL 2 RL is a profile which resembles an OWL-based rule language. The design principle was to enable expressing rules using ontology axioms (especially subclass axioms). The rules are syntactically restricted to Horn clauses. Production rules cannot be expressed in OWL 2 RL. Some examples are shown below:

- \( \text{hasParent} \circ \text{hasBrother} \sqsubseteq \text{hasUncle} \)
- \( \text{Floats} \sqcap \exists \text{sameWeightAs}.T \sqsubseteq \text{Floats} \)
- \( \text{hasParent} \circ \text{hasParent} \sqsubseteq \text{hasGrandparent} \)
- \( \text{Monogamous} \sqsubseteq \leq \text{1married.Alive} \)

Reasoning in OWL 2 RL is PTime. Rule-based modelling may be more intuitive and thus even naive implementations are useful. There exist some of them, and they are fast and scalable.

**OWL 2 DL and OWL 2 Full.** While the OWL 2 profiles have been mainly designed to keep tractability of resulting ontologies, there is still a possibility to use more complex languages, namely OWL 2 DL and OWL 2 Full. Reasoning in OWL 2 DL is decidable but complex (N2ExpTime) and is based on \(SROIQ\) DL. OWL 2 Full provides RDF-based semantics and is undecidable in general case.

### 8 Conclusions

The report summarizes the main objectives of the Semantic Web and introduces its most important concepts and technologies. In Section 1 an overview of the architecture is given and a brief summary of the evolution and state-of-the-art is presented. The following sections introduce Semantic Web technologies and standards from the simplest to more expressive and formal. Foundations of the markup languages used in the Semantic Web, such as XML and RDF are described in Sect. 2 and Sect. 3. The XML language provides the structure for the Semantic Web documents, whereas the RDF is the main language for representing metadata and relations among resources. Categorization on the Semantic Web is done with use of ontologies. They are described in Section 4. A simple ontological language based on RDF is RDF Schema introduced in Section 5. The main logical foundation of the Semantic Web are Description Logics. Their concepts are introduced in Section 6. They underpin the well accepted Web Ontology Language OWL (version 1 and 2), which is described in Section 7.

### References


