

# OWLIM Primer



Semantic Repository for RDF(S) and OWL

## **OWLIM** *Primer*

### **Table of Contents**

- [Primer Overview](#)
- [Primer Background Knowledge](#)
- [Primer Introduction to OWLIM](#)
- [Primer Glossary](#)
- [Primer References](#)

## Primer Overview

This guide forms the introduction to the set of OWLIM's user documentation. OWLIM is Ontotext's database system for storing, processing and querying structured data formatted according to the [Resource Description Framework \(RDF\)](#). It is packaged as a Storage and Inference Layer (SAIL) for the [Sesame](#) RDF framework. If you need a quick overview of OWLIM or a download link to its latest releases, please visit OWLIM's [product page](#).

Among structured data repositories, OWLIM is the world leader for its unsurpassed processing capacity, both in terms of volume of data and loading/inference speed on commodity hardware.

This guide provides the reader with a conceptual overview of OWLIM – with some general information about its structure and functioning – as well as an overview of how to install the system.

In addition, a large part of the document gives a general introduction to semantic concepts and technologies, which provides the inexperienced reader with sufficient background to understand the role of OWLIM and the problems it solves.

- [Purpose, Intended Readership and Overview of This Document](#)
- [How to Use This Document](#)
- [Conventions](#)
- [Credits and Licensing](#)

## Purpose, Intended Readership and Overview of This Document

This document is intended for project managers, engineers and systems designers who wish to integrate OWLIM in to their applications, but who do not necessarily have a thorough grounding in semantic technologies and concepts. It is also useful for system administrators who need to support and maintain an OWLIM-based system.

This document presumes that the reader is familiar with databases and is expected to be able to draw from his knowledge of relational databases. The required minimum of Semantic Web concepts and related information is provided in the course of this document.

An understanding of XML would also be beneficial, as it is the most widely used meta-data description language on the Web and for this reason is used as a standard method for illustrating Semantic Web concepts in the reference literature.

The purpose of this document is:

1. To deliver a general overview of the OWLIM system and to acquaint the reader with its architecture and operational logic
2. To introduce the Semantic Web and its concepts and to provide enough information to begin using OWLIM in practical applications

The [background knowledge](#) section gives a brief conceptual description of the Semantic Web. There is a great deal of information about the Semantic Web available both over the Internet and in print, therefore the topics in this section serve to highlight the required knowledge rather than to cover them in great depth. However, ample references are provided for every topic.

The [introduction to OWLIM](#) section deals with OWLIM itself and provides introductory information that forms a basis for the more detailed information given in the User Guide for the particular OWLIM edition.

## How to Use This Document

The [background knowledge](#) section is optional depending on the reader's familiarity with Semantic Web concepts. For readers unfamiliar with this subject, it is recommended to read this entire section first and preferably check as many of the references as time permits. Readers who have some understanding, should read this section selectively, skipping over content that is already familiar. Experts in this field can skip the entire section.

The [introduction to OWLIM](#) is essential reading and should be read sequentially from beginning to end without skipping any topics. This knowledge is required before using OWLIM.

## Conventions

The following formatting conventions are used in this book:

- Code examples are listed in `typewriter-like font`.
- The first occurrence of a term that is explained in the Glossary at the end of this book is hyper-linked to its Glossary entry and is formatted [like this](#).
- Other important terms, when first introduced, may be written in *italics*.
- Formulas are always in *italics*.
- References are given in square brackets, e.g. [3] means "Refer to item #3 in the References section".

## Credits and Licensing

OWLIM uses Sesame as a library, taking advantage of its APIs for storage and querying, as well as the support for a wide variety of query languages (e.g. SPARQL and SeRQL) and RDF syntaxes (e.g. RDF/XML, N3, Turtle).

The development of OWLIM is partly supported by [SEKT](#), [TAO](#), [TripCom](#), [LarKC](#), and other [FP6](#) and [FP7](#) European research projects.

This document and the products discussed in it are copyrighted and subject to licensing as follows:

- **OWLIM-Lite**, © Copyright Ontotext AD. 135 Tsarigradsko Shosse, Sofia 1784, Bulgaria, <http://www.ontotext.com>.

- **OWLIM-SE**, © Copyright Ontotext AD. 135 Tsarigradsko Shosse, Sofia 1784, Bulgaria, <http://www.ontotext.com>.
- **OWLIM-Enterprise**, © Copyright Ontotext AD. 135 Tsarigradsko Shosse, Sofia 1784, Bulgaria, <http://www.ontotext.com>.
- **Sesame**, © Copyright Aduna b.v. Stadsring 181, 3817 BA Amersfoort, The Netherlands Sesame is an open-source library, available under the GNU Lesser GPL (<http://www.gnu.org/copyleft/lesser.html>)
- All other trademarks mentioned in this document, belong to their respective owners.

Full licensing information is available at <http://www.ontotext.com/owlim/>, as well as in the licence files located in the main folder of the distribution package.

## Primer Background Knowledge

The background knowledge needed to make best use of OWLIM comprises some basic [Semantic Web](#) concepts and general understanding of the Sesame framework. This chapter provides introductions to both.

- Introduction to Semantic Web Knowledge Management Concepts
  - Resource Description Framework (RDF)
    - Uniform Resource Identifiers (URIs)
    - Statements: Subject-Predicate-Object Triples
    - Properties
    - Named Graphs
  - RDF Schema (RDFS)
    - Describing Classes
    - Describing Properties
    - Sharing Vocabularies
    - Dublin Core Metadata Initiative
  - Ontologies and Knowledge Bases
    - Classification of Ontologies
    - Knowledge Bases
      - Proton
  - Logics and Inference
    - Logic Programming
    - Predicate Logic
    - Description Logics
  - The Web Ontology Language (OWL) and its Dialects
    - OWL DLP
    - OWL Horst
    - OWL2 RL
    - OWL Lite
    - OWL DL
  - Query Languages
    - RQL, RDQL
    - SPARQL
    - SeRQL
  - Reasoning Strategies
  - Semantic Repositories
- Introduction to Sesame
  - Sesame Architecture
  - The SAIL API

## Introduction to Semantic Web Knowledge Management Concepts

The Semantic Web represents a broad range of ideas and technologies that attempt to bring meaning to the vast amount of information available via the Web. The intention is provide information in a structured form so that it can be processed automatically by machines. The combination of structured data and inferencing can yield much information not explicitly stated.

The aim of the Semantic Web is to solve the most problematic issues that come with the growth of the non-semantic (HTML-based or similar) Web that results in a high level of human effort for finding, retrieving and exploiting information. For example, contemporary search engines are extremely fast, but tend to be very poor at producing relevant results. Of the thousands of matches typically returned, only a few point to truly relevant content and some of this content may be buried deep within the identified pages. Such issues dramatically reduce the value of the information discovered as well as the ability to automate the consumption of such data. Other problems related to classification and generalisation of identifiers further confuse the landscape.

The Semantic Web solves such issues by adopting unique identifiers for concepts and the relationships between them. These identifiers, called "[Universal Resource Identifiers](#)" (URIs) (a "resource" is any 'thing' or 'concept') are similar to Web page URLs, but do not necessarily identify documents from the Web. Their sole purpose is to uniquely identify objects or concepts and the relationships between them.

The use of URIs removes much of the ambiguity from information, but the Semantic Web goes further by allowing concepts to be associated with hierarchies of classifications, thus making it possible to infer new information based on an individual's classification and relationship to other concepts. This is achieved by making use of [ontologies](#) – hierarchical structures of concepts-- to classify individual concepts.

## Resource Description Framework (RDF)

The World-Wide Web has grown rapidly and contains huge amounts of information that cannot be interpreted by machines. Machines cannot understand meaning, therefore they cannot understand Web content. For this reason, most attempts to retrieve some useful pieces of information from the Web require a high degree of user involvement – manually retrieving information from multiple sources (different Web pages), 'digging' through multiple search engine results (where useful pieces of data are often buried many pages deep), comparing differently structured result sets (most of them incomplete), and so on.

For the machine interpretation of semantic content to become possible, there are two prerequisites:

1. Every concept should be uniquely identified. (For example, if one and the same person owns a web site, authors articles on other sites, gives an interview on another site and has profiles in a couple of social media sites like Facebook and LinkedIn, then the

- occurrences of his name/identifier in all these places should be related to one and the same unique identifier.)
2. There must be a unified system for conveying and interpreting meaning that all automated search agents and data storage applications should use.

One approach for attaching semantic information to Web content is to embed the necessary machine-processable information through the use of special meta-descriptors (meta-tagging) in addition to the existing meta-tags that mainly concern the layout.

Within these meta tags, the [resources](#) (the pieces of useful information) can be uniquely identified in the same manner in which Web pages are uniquely identified, i.e. by extending the existing URL system into something more universal – a URI ([Uniform Resource Identifier](#)). In addition, conventions can be devised, so that resources can be described in terms of properties and values (resources can have properties and properties have values). The concrete implementations of these conventions (or vocabularies) can be embedded into Web pages (through meta-descriptors again) thus effectively 'telling' the processing machines things like:

[resource] **John Doe** has a [property] **web site** which is [value] **www.johndoesite.com**

The **Resource Description Framework (RDF)** developed by the World Wide Web Consortium (W3C) makes possible the automated semantic processing of information, by structuring information using individual **statements** that consist of: Subject, Predicate, Object. Although frequently referred to as a 'language', RDF is mainly a data model. It is based on the idea that the things being described have properties which have values, and that resources can be described by making statements. RDF prescribes how to make statements about resources, in particular, Web resources, in the form of subject-predicate-object expressions. The 'John Doe' example above is precisely this kind of statement. The statements are also referred to as "[triples](#)", because they always have the subject-predicate-object structure [32].

The basic RDF components include statements, Uniform Resource Identifiers, properties, blank nodes and literals. They are discussed in the topics that follow.

## Uniform Resource Identifiers (URIs)

A unique Uniform Resource Identifier (URI) is assigned to any resource or thing that needs to be described. Resources can be authors, books, publishers, places, people, hotels, goods, articles, search queries, and so on. In the Semantic Web, every resource has a URI. A URI can be a URL or some other kind of unique identifier. Unlike URLs, URIs do not necessarily enable access to the resource they describe, that is, in most cases they do not represent actual web pages. For example, the string <http://www.johndoesite.com/aboutme.htm> if used as a URL (Web link) is expected to take us to a Web page of the site providing information about the site owner, the person John Doe; the same string can however be used simply to identify that person on the Web (URI) irrespective of whether such a page exists or not.

Thus URI schemes can be used not only for Web locations, but also for such diverse objects as telephone numbers, ISBN numbers, and geographic locations. For more information about URIs see [40]. In general, we assume that a URI is the identifier of a resource and can be used as either the subject or the object of a statement (see below). Once the subject is assigned a URI, it can be treated as a resource and further statements can be made about it.

This idea of using URIs to identify "things" and the relations between them is important. This approach goes some way towards a global, unique naming scheme. The use of such a scheme greatly reduces the homonym problem that has plagued distributed data representation in the past.

## Statements: Subject-Predicate-Object Triples

To make the information in following sentence

"The web site [www.johndoesite.com](http://www.johndoesite.com) is created by John Doe."

machine-accessible, it must be expressed in the form of an RDF statement, i.e. a subject predicate object triple:

"[subject] the web site [www.johndoesite.com](http://www.johndoesite.com) [predicate] has a creator [object] called John Doe."

This statement emphasizes the fact that in order to describe something, there needs to be a way to name or identify a number of things:

- the thing the statement describes (Web site "[www.johndoesite.com](http://www.johndoesite.com)")
- a specific property ("creator") of the thing the statement describes
- the thing the statement says is the value of this property (who the owner is)

The respective RDF terms for the various parts of the statement are:

- the subject is the URL "[www.johndoesite.com](http://www.johndoesite.com)"
- the predicate is the expression "has creator"
- the object is name of the creator, which has the value "John Doe"

Next, each member of the subject-predicate-object triple should be identified using its URI, for example:

- the subject is "<http://www.johndoesite.com>"
- the predicate is "<http://purl.org/dc/elements/1.1/creator>" (this is according to a particular RDF Schema (see section 3.1.2), namely, the Dublin Core Metadata Element Set (see below))
- the object is "<http://www.johndoesite.com/aboutme>" (which may not be an actual web page)

Note that in this version of the statement, instead of identifying the creator of the web site by the character string "John Doe", we used a URI, namely "<http://www.johndoesite.com/aboutme>". An advantage of using a URI is that the identification of the statement's subject can be more precise, i.e. the creator of the page is neither the character string "John Doe", nor any of the thousands of other people with that name, but the particular John Doe associated with that URI (whoever created the URI defines the association). Moreover, since there is a URI to refer to John Doe, he is now a full-fledged resource and additional information can be recorded about him simply by adding additional RDF statements with John's URI as the subject.

What we basically have now is a the logical formula  $P(x, y)$ , where the binary predicate  $P$  relates the object  $x$  to the object  $y$  – we may also think of this formula as written in the form  $(x, P, y)$ . In fact, RDF offers only binary predicates (properties). If more complex relationships are to be defined, this is done through sets of multiple RDF triples. Therefore, we can describe the statement as:

```
<http://www.johndoesite.com> <http://purl.org/dc/elements/1.1/creator>
<http://www.johndoesite.com/aboutme>
```

There are several conventions for writing abbreviated RDF statements, as used in the RDF specifications themselves. This shorthand employs an XML *qualified name* (or *QName*) without angle brackets as an abbreviation for a full URI reference. A QName contains a prefix that has been assigned to a namespace URI, followed by a colon, and then a local name. The full URI reference is formed from the QName by appending the local name to the namespace URI assigned to the prefix. So, for example, if the QName prefix "foo" is assigned to the namespace URI "http://example.com/somewhere/", then the QName "foo:bar" is shorthand for the URI "http://example.com/somewhere/bar".

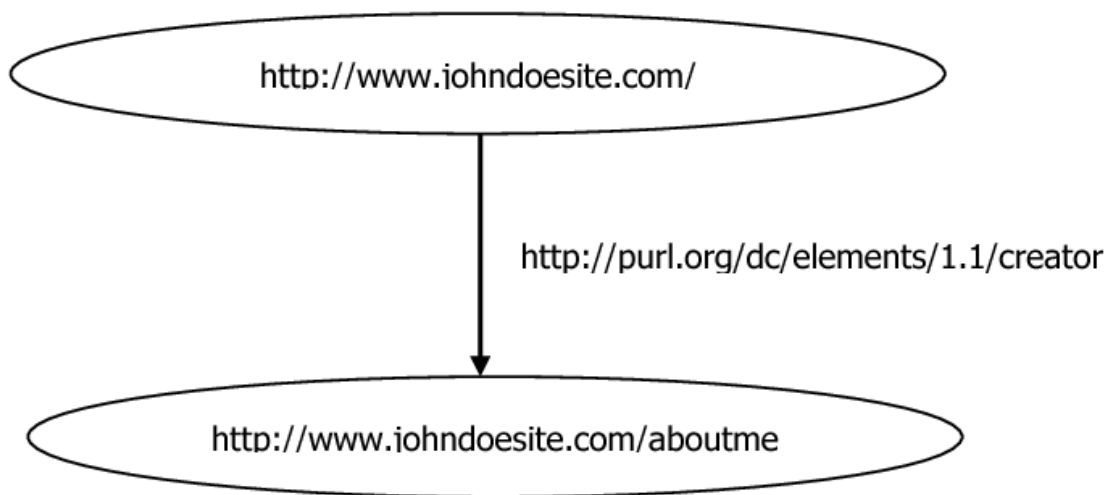
In our example, we can define the namespace "jds" for "http://www.johndoesite.com" and use the Dublin Core Metadata namespace "dc" for "http://purl.org/dc/elements/1.1/." So the shorthand form for the example statement is simply:

jds: dc:creator jds:aboutme

Objects of RDF statements can (and very often do) form the subjects of other statements leading to a graph like representation of knowledge (the [RDF graph model](#) is defined in [34]). Using this notation, a statement is represented by:

- a node for the subject
- a node for the object
- an arc for the predicate, directed from the subject node to the object node.

So the RDF statement above could be represented by the following graph:



This kind of graph is known in the artificial intelligence community as a "semantic net" [32]

In order to represent RDF statements in a machine-processable way, RDF uses mark-up languages, namely (and almost exclusively) the Extensible Mark-up Language (XML). Because an abstract data model needs a concrete syntax in order to be represented and transmitted, RDF has been given a syntax in XML. As a result, it inherits the benefits associated with XML. However, it is important to understand that other syntactic representations of RDF, not based on XML, are also possible; XML-based syntax is not a necessary component of the RDF model. XML was designed to allow anyone to design their own document format and then write a document in that format. RDF defines a specific XML mark-up language, referred to as RDF/XML, for use in representing RDF information and for exchanging it between machines. Written in RDF/XML, our example will look as follows:

```
<?xml version="1.0" encoding="UTF-16"?>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:jds="http://www.johndoesite.com/">

  <rdf:Description rdf:about="http://www.johndoesite.com/">
    <dc:creator rdf:resource="jds:aboutme">
  </rdf:Description>
</rdf:RDF>
```



RDF/XML uses the namespace mechanism of XML, but in an expanded way. In XML, namespaces are only used for disambiguation purposes. In RDF/XML, external namespaces are expected to be RDF documents defining resources, which are then used in the importing RDF document. This mechanism allows the reuse of resources by other people who may decide to insert additional features into these resources. The result is the emergence of large, distributed collections of knowledge.

Also observe that the `rdf:about` attribute of the element `rdf:Description` is, strictly speaking, equivalent in meaning to that of an ID attribute, but it is often used to suggest that the object about which a statement is made has already been "defined" elsewhere. Strictly speaking, a set of RDF statements together simply forms a large graph, relating things to other things through properties, and there is no such concept as "defining" an object in one place and referring to it elsewhere. Nevertheless, in the serialized XML syntax, it is sometimes useful (if only for human readability) to suggest that one location in the XML serialization is the "defining" location, while other locations state "additional" properties about an object that has been "defined" elsewhere. There is much more to RDF/XML logic and syntax than can be covered here. For a discussion of the principles behind the modelling of RDF statements in XML (known as 'striping'), together with a presentation of the available RDF/XML abbreviations and other details and examples about writing RDF in XML, see the (normative) RDF/XML Syntax Specification from the W3C [33].

## Properties

Properties are a special kind of resource: they describe relationships between resources, for example **written by**, **age**, **title**, and so on. Properties in RDF are also identified by URIs (in most cases, these are actual URLs). Therefore, properties themselves can be used as the subject in other statements which allows for an expressive ways to describe properties, e.g. by defining property hierarchies.

## Named Graphs

A named graph (NG) is a set of triples named by a URI. This URI can then be used outside or within the graph to refer to it [26]. The ability to name a graph allows separate graphs to be identified out of a large collection of statements and further allows statements to be made about graphs.

Named graphs represent an extension of the RDF data model, where quadruples `<s, p, o, ng>` are used to define statements in an RDF multi-graph. This mechanism allows, for example, the handling of provenance when multiple RDF graphs are integrated into a single repository. For information on the semantics and the abstract syntax of named graphs refer to [6].

From the perspective of OWLIM, named graphs are important, because comprehensive support for SPARQL (the most popular RDF query language and current W3C recommendation – see section 3.1.6.2) requires NG support.

## RDF Schema (RDFS)

While being a universal model that lets users describe resources using their own vocabularies, RDF does not make assumptions about any particular application domain, nor does it define the semantics of any domain. It is up to the user to do so using an RDF Schema (RDFS) vocabulary.

**RDF Schema** is a vocabulary description language for describing properties and classes of RDF resources, with a semantics for generalisation hierarchies of such properties and classes. Be aware of the fact that the RDF Schema is conceptually different from the XML Schema even though the common term *schema* suggests similarity. XML Schema constrains the structure of XML documents, whereas RDF Schema defines the vocabulary used in RDF data models. Thus RDFS makes semantic information machine-accessible, in accordance with the Semantic Web vision. RDF Schema is a primitive ontology language. It offers certain modelling primitives with fixed meaning.

RDF Schema does not provide a vocabulary of application-specific classes. Instead, it provides the facilities needed to describe such classes and properties, and to indicate which classes and properties are expected to be used together (for example, to say that the property 'JobTitle' will be used in describing a class 'Person'). In other words, RDF Schema provides a type system for RDF.

The RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. For example, RDFS allows resources to be defined as instances of one or more classes. In addition, it allows classes to be organised in a hierarchical fashion; for example a class 'Dog' might be defined as a subclass of 'Mammal' which itself is a subclass of 'Animal', meaning that any resource that is in class 'Dog' is also implicitly in class 'Animal' as well.

RDF classes and properties, however, are in some respects very different from programming language types. RDF class and property descriptions do not create a straight-jacket into which information must be forced, but instead provide additional information about the RDF resources they describe.

The RDFS facilities are themselves provided in the form of an RDF vocabulary; that is, as a specialised set of predefined RDF resources with their own special meanings. The resources in the RDFS vocabulary have URIs with the prefix <http://www.w3.org/2000/01/rdf-schema#> (conventionally associated with the namespace prefix `rdfs`). Vocabulary descriptions (schemas) written in the RDFS language are legal RDF graphs. Hence, systems that process RDF information that do *not* understand the additional RDFS vocabulary can still interpret a schema as a legal RDF graph consisting of various resources and properties. However, such a system will be oblivious to the additional built-in meaning of the RDFS terms. To understand these additional meanings, software that processes RDF information must be extended to include these language features and to interpret their meanings in the defined way.

## Describing Classes

A class can be thought of as a set of elements. Individual objects that belong to a class are referred to as instances of that class. A class in RDFS corresponds to the generic concept of a type or category, somewhat like the notion of a class in object-oriented programming languages such as Java. RDF classes can be used to represent any category of objects, such as web pages, people, document types, databases or abstract concepts. Classes are described using the RDF Schema resources `rdfs:Class` and `rdfs:Resource`, and the



properties `rdf:type` and `rdfs:subClassOf`. The relationship between instances and classes in RDF is defined using `rdf:type`. An important use of classes is to impose restrictions on what can be stated in an RDF document using the schema. In programming languages, typing is used to prevent incorrect use of objects (resources) and the same is true in RDF: imposing a restriction on the objects to which the property can be applied. In logical terms, this is a restriction on the domain of the property.

## Describing Properties

In addition to describing the specific classes of things they want to describe, user communities also need to be able to describe specific properties that characterise those classes of things (such as 'numberOfBedrooms' to describe an apartment). In RDFS, properties are described using the RDF class `rdf:Property`, and the RDFS properties `rdfs:domain`, `rdfs:range` and `rdfs:subPropertyOf`. All properties in RDF are described as instances of class `rdf:Property`. So a new property, such as `ex:weightInKg`, is defined with the following RDF statement:

```
ex:weightInKg    rdf:type    rdf:Property .
```

RDFS also provides vocabulary for describing how properties and classes are intended to be used together. The most important information of this kind is supplied by using the RDFS properties `rdfs:range` and `rdfs:domain` to further describe application-specific properties.

The `rdfs:range` property is used to indicate that the values of a particular property are members of a designated class. For example, to indicate that the property `ex:author` has values that are instances of class `ex:Person`, the following RDF statements are used:

```
ex:Person    rdf:type    rdfs:Class .
ex:author    rdf:type    rdf:Property .
ex:author    rdfs:range  ex:Person .
```

These statements indicate that `ex:Person` is a class, `ex:author` is a property, and that RDF statements using the `ex:author` property have instances of `ex:Person` as objects.

The `rdfs:domain` property is used to indicate that a particular property is used to describe a specific class of objects. For example, to indicate that the property `ex:author` applies to instances of class `ex:Book`, the following RDF statements are used:

```
ex:Book      rdf:type    rdfs:Class .
ex:author    rdf:type    rdf:Property .
ex:author    rdfs:domain ex:Book .
```

These statements indicate that `{ex:Book}` is a class, `ex:author` is a property, and that RDF statements using the `ex:author` property have instances of `ex:Book` as subjects, see [32].

## Sharing Vocabularies

RDFS provides the means to create custom vocabularies. However, it is generally easier and better practice to use an existing vocabulary created by someone else who has already been describing a similar conceptual domain. Such publicly available vocabularies, called "shared vocabularies" are not only cost-efficient to use, but they also promote the shared understanding of the described domains. Considering the earlier example, in the statement:

```
jds:    dc:creator    jds:aboutme .
```

the predicate `dc:creator`, when fully expanded as a URI, is an unambiguous reference to the *creator* attribute in the Dublin Core metadata attribute set, a widely-used set of attributes (properties) for describing information of this kind. So this triple is effectively saying that the relationship between the web site (identified by <http://www.johndoesite.com/>) and the creator of the site (a distinct person, identified by <http://www.johndoesite.com/aboutme>) is exactly the property identified by <http://purl.org/dc/elements/1.1/creator>. This way, anyone familiar with the Dublin Core vocabulary or those who find out what `dc:creator` means (say, by looking up its definition on the Web) will know what is meant by this relationship. In addition, this shared understanding based upon using unique URIs for identifying concepts is exactly the requirement for creating computer systems that can automatically process structured information.

However, the use of URIs does not solve all identification problems, because, different URIs can be created for referring to the same thing. For this reason, it is a good idea to have a preference towards using terms from existing vocabularies (such as the Dublin Core) where possible, rather than making up new terms that might overlap with those of some other vocabulary. Appropriate vocabularies for use in specific application areas are being developed all the time, however even so, the sharing of these vocabularies in a common 'Web space' provides the opportunity to identify and deal with any equivalent terminology.

## Dublin Core Metadata Initiative

An example of a shared vocabulary that is readily available for reuse is [The Dublin Core](#), which is a set of elements (properties) for describing documents (and hence, for recording metadata). The element set was originally developed at the March 1995 Metadata Workshop in Dublin, Ohio, USA. Dublin Core has subsequently been modified on the basis of later Dublin Core Metadata workshops and is currently maintained by the [Dublin Core Metadata Initiative](#).

The goal of Dublin Core is to provide a minimal set of descriptive elements that facilitate the description and the automated indexing of document-like networked objects, in a manner similar to a library card catalogue. The Dublin Core metadata set is suitable for use by



resource discovery tools on the Internet, such as Web crawlers employed by search engines. In addition, Dublin Core is meant to be sufficiently simple to be understood and used by the wide range of authors and casual publishers of information to the Internet. Dublin Core elements have become widely used in documenting Internet resources (the Dublin Core 'creator' element was used in the earlier examples). The current elements of the Dublin Core are defined in [10] and contain definitions for properties such as 'title' (a name given to a resource), 'creator' (an entity primarily responsible for making the content of the resource), 'date' (a date associated with an event in the life-cycle of the resource) and 'type' (the nature or genre of the content of the resource). Information using the Dublin Core elements may be represented in any suitable language (e.g. in HTML meta elements). However, RDF is an ideal representation for Dublin Core information. The following example adapted from [19] uses Dublin Core by itself to describe an audio recording of a guide to growing rose bushes:

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

  <rdf:Description rdf:about="http://media.example.com/audio/guide.ra">
    <dc:creator>Mr. Dan D. Lion</dc:creator>
    <dc:title>A Guide to Growing Roses</dc:title>
    <dc:description>Describes planting and nurturing rose bushes.
    </dc:description>
    <dc:date>2001-01-20</dc:date>
  </rdf:Description>
</rdf:RDF>
```

The same RDF statements in Notation-3:

```
@prefix dc: <[http://purl.org/dc/elements/1.1/]> .
@prefix rdf: <[http://www.w3.org/1999/02/22-rdf-syntax-ns#]> .

<http://media.example.com/audio/guide.ra> dc:creator "Mr. Dan D. Lion" ;
  dc:title "A Guide to Growing Roses" ;
  dc:description "Describes planting and nurturing rose bushes." ;
  dc:date "2001-01-20" .
```

## Ontologies and Knowledge Bases

In general, an ontology formally describes a (usually finite) domain of related concepts (classes of objects) and their relationships. For example, in a company setting, staff members, managers, company products, offices, and departments might be some important concepts. The relationships typically include hierarchies of classes. A hierarchy specifies a class *C* to be a subclass of another class *C'* if every object in *C* is also included in *C'*. For example, all managers are staff members. Apart from subclass relationships, ontologies may include information such as:

- properties (*X* is subordinated *Y*);
- value restrictions (only managers may head departments);
- disjointness statements (managers and general employees are disjoint);
- specifications of logical relationships between objects (every department must have at least three staff members).

Ontologies are important because [semantic repositories](#) use ontologies as semantic schemata. This makes automated reasoning about the data possible (and easy to implement) since the most essential relationships between the concepts are built into the ontology.

Formal [knowledge representation](#) (KR) is about building models. The typical modelling paradigm is mathematical logic, but there are also other approaches, rooted in information and library science. KR is a very broad term; here we only refer to its mainstream meaning, of the world (of a particular state of affairs, situation, domain or problem), which allow for automated reasoning and interpretation. Such models consist of ontologies defined in a formal language. Ontologies can be used to provide formal semantics (i.e. machine-interpretable meaning) to any sort of information: databases, catalogues, documents, Web pages, etc. Ontologies can be used as semantic frameworks: the association of information with ontologies makes such information much more amenable to machine processing and interpretation. This is because ontologies are described using logical formalisms, such as [OWL](#) [8], which allow automatic inferencing over these ontologies and datasets that use them, i.e. as a vocabulary. An important role of ontologies is to serve as schemata or "intelligent" views over information resources. Comments in the same spirit are provided in [14]. This is also the role of ontologies in the Semantic Web.. Thus they can be used for indexing, querying, and reference purposes over non-ontological datasets and systems, such as databases, document and catalogue management systems. Because ontological languages have a formal semantics, ontologies allow a wider interpretation of data, i.e. inference of facts which are not explicitly stated. In this way, they can improve the interoperability and the efficiency of the usage of arbitrary datasets.

An ontology *O* can be defined as comprising the 4-tuple. A more formal and extensive mathematical definition of an ontology is given in [11]. The characterisation offered here is suitable for the purposes of our discussion, however:

$$O = \langle C, R, I, A \rangle$$

where

- *C* is a set of classes representing *concepts* from the domain we wish to describe (e.g. invoices, payments, products, prices, etc);
- *R* is a set of relations (also referred to as properties or predicates) holding between (instances of) these classes (e.g. Product

*hasPrice Price*);

- $\mathcal{I}$  is a set of instances, where each instance can be a member of one or more classes and can be linked to other instances or to literal values (strings, numbers and other data-types) by relations (e.g. *product23* compatibleWith *product348* or *product23* hasPrice €170);
- $\mathcal{A}$  is a set of axioms (e.g. if a product has a price greater than €200, then shipping is free).

## Classification of Ontologies

Ontologies can be classified as light-weight or heavy-weight according to the complexity of the KR language and the extent to which it is used. Light-weight ontologies allow for more efficient and scalable reasoning, but do not possess the highly predictive (or restrictive) power of more powerful KR languages. Ontologies can be further differentiated according to the sort of conceptualization that they formalize: upper-level ontologies model general knowledge, while domain and application ontologies represent knowledge about a specific domain (e.g. medicine or sport) or a type of application, e.g. knowledge management systems. Basic definitions regarding ontologies can be found in [13], [14], [15], and [16].

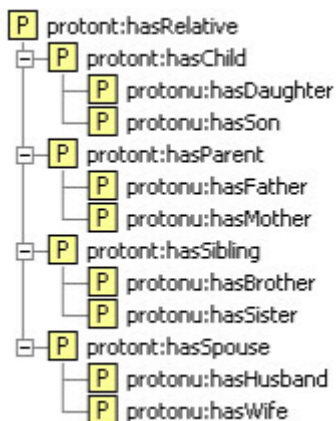
Finally, ontologies can be distinguished according to the sort of semantics being modelled and their intended usage. The major categories from this perspective are:

- Schema-ontologies: ontologies which are close in purpose and nature to database and object-oriented schemata. They define classes of objects, their properties and relationships to objects of other classes. A typical use of such an ontology involves using it as a vocabulary for defining large sets of instances. In basic terms, a class in a schema ontology corresponds to a table in a relational database; a relation – to a column; an instance – to a row in the table for the corresponding class;
- Topic-ontologies: taxonomies which define hierarchies of topics, subjects, categories, or designators. These have a wide range of applications related to classification of different things (entities, information resources, files, Web-pages, etc). The most popular examples are library classification systems and taxonomies, which are widely used in the knowledge management field. [Yahoo](#) and [DMoz](#) are popular large scale incarnations of this approach. A number of the most popular taxonomies are listed as encoding schemata in [Dublin Core](#)
- Lexical ontologies: lexicons with formal semantics, which define lexical concepts. We use 'lexical concept' here as some kind of a formal representation of the meaning of a word or a phrase. In Wordnet, for example, lexical concepts are modelled as synsets (synonym sets), while word-sense is the relation between a word and a synset, word-senses and terms. These can be considered as semantic thesauri or dictionaries. The concepts defined in such ontologies are not instantiated, rather they are directly used for reference, e.g. for annotation of the corresponding terms in text. [WordNet](#) is the most popular general purpose (i.e. upper-level) lexical ontology.

## Knowledge Bases

[Knowledge base](#) (KB) is a broader term than ontology. Similar to an ontology, a KB is represented in a KR formalism, which allows automatic inference. It could include multiple axioms, definitions, rules, facts, statements, and any other primitives. In contrast to ontologies, however, KBs are not intended to represent a shared or consensual conceptualization. Thus, ontologies are a specific sort of a KB. Many KBs can be split into ontology and instance data parts, in a way analogous to the splitting of schemata and concrete data in databases. A broader discussion on the different terms related to ontology and semantics can be found in [23].

### Proton



PROTON [38] is a light-weight upper-level schema-ontology developed in the scope of the SEKT project [35]. It is used in the KIM system [30] for semantic annotation, indexing and retrieval. We will also use it for ontology-related examples within this section. PROTON [31] is encoded in [OWL Lite](#) [8] and defines about 300 classes and 100 properties, providing good coverage of named entity types and concrete domains, i.e. modelling of concepts such as people, organizations, locations, numbers, dates, addresses, etc. A snapshot of the PROTON class hierarchy is shown above.

## Logics and Inference

The topics that follow take a closer look at the logic that underlies the retrieval and manipulation of semantic data and the kind of programming that supports it.

### Logic Programming

Logic programming involves the use of logic for computer programming, where the programmer uses a declarative language to assert statements and a reasoner or theorem-prover is used to solve problems. A reasoner can interpret sentences, such as IF A THEN B, as a means to prove B from A. In other words, given a collection of logical sentences, a reasoner will explore the solution space in order to find a path to justify the requested theory. For example, to determine the truth value of 'C' given the following logical sentences:

```
IF A AND B THEN C
B
IF D THEN A
D
```

a reasoner will interpret the IF . THEN statements as rules and determine that C is indeed inferred from the KB. This use of rules in logic programming has led to 'rule-based reasoning' and 'logic programming' becoming synonymous, although this is not strictly the case.

In LP, there are rules of logical inference that allow new (implicit) statements to be inferred from other (explicit) statements, with the guarantee that if the explicit statements are true, so are the implicit statements. Because these rules of inference can be expressed in purely symbolic terms, applying them is the kind of symbol manipulation that can be carried out by a computer. This is what happens when a computer executes a logical program: it uses the rules of inference to derive new statements from the ones given in the program, until it finds one that expresses the solution to the problem that has been formulated. If the statements in the program are true, then so are the statements that the machine derives from them, and the answers it gives will be correct.

The program can give correct answers only if the following two conditions are met:

1. The program must contain only true statements;
2. The program should contain enough statements to allow solutions to be derived for all the problems that are of interest.

There must also be a reasonable time frame for the entire inference process. To this end, much research has been carried out to determine the complexity classes of various logical formalisms and reasoning strategies. Generally speaking, to reason with Web-scale quantities of data requires a low-complexity approach. A tractable solution is one whose algorithm requires finite time and space to complete.

## Predicate Logic

From a more abstract viewpoint, the subject of the previous topic is related to the foundation upon which logical programming resides, which is logic, particularly in the form of [predicate logic](#) (also known as "first order logic"). Some of the specific features of predicate logic render it very suitable for making inferences over the Semantic Web, namely:

- It provides a high-level language in which knowledge can be expressed in a transparent way and with high expressive power;
- It has a well-understood formal semantics, which assigns an unambiguous meaning to logical statements;
- There exist proof systems that can automatically derive statements syntactically from a set of premises. Its proof systems are both sound (meaning that all derived statements follow semantically from the premises) and complete (all logical consequences of the premises can be derived in the proof system);
- It is possible to trace the proof that leads to a logical consequence. (This is because the proof system is sound and complete.) In this sense, the logic can provide explanations for answers.

The languages of RDF and OWL (Lite and DL) can be viewed as specializations of predicate logic. One reason for such specialized languages to exist is that they provide a syntax that fits well with the intended use (in our case, Web languages based on tags). The other major reason is that they define reasonable subsets of logic. This is important because there is a trade-off between the expressive power and the computational complexity of certain logics: the more expressive the language, the less efficient (in the worst case) the corresponding proof systems. As previously stated, OWL Lite and [OWL DL](#) correspond roughly to a [description logic](#), a subset of predicate logic for which efficient proof systems exist.

Another subset of predicate logic with efficient proof systems comprises the so-called rule systems (also known as [Horn logic](#) or *definite logic programs*). A rule has the form:

$$A_1, \dots, A_n \rightarrow B$$

where  $A_i$  and B are atomic formulas. In fact, there are two intuitive ways of reading such a rule:

- If  $A_1, \dots, A_n$  are known to be true, then B is also true. Rules with this interpretation are referred to as 'deductive rules'.
- If the conditions  $A_1, \dots, A_n$  are true, then carry out the action B. Rules with this interpretation are referred to as 'reactive rules'.

Both approaches have important applications. The deductive approach, however, is more relevant for the purpose of retrieving and managing structured data. This is because it relates better to the possible queries that one can ask, as well as to the appropriate answers and their proofs.

## Description Logics

Description Logics (DL) have historically evolved from a combination of frame-based systems and predicate logic. Its main purpose is to overcome some of the problems with frame-based systems and to provide a clean and efficient formalism to represent knowledge. The main idea of DL is to describe the world in terms of 'properties' or 'constraints' that specific 'individuals' must satisfy. DL is based on the following basic entities [1]:

- **Objects** – Correspond to single 'objects' of the real world such as a specific person, a table or a telephone. The main properties of an object are that it can be distinguished from other objects and that it can be referred to by a name. DL objects correspond to the individual constants in predicate logic;

- **Concepts** – Can be seen as "classes of objects". Concepts have two functions: on one hand, they describe a *set of objects*, on the other hand they determine *properties* of objects. For example the class "table" is supposed to describe the set of all table objects in the universe. On the other hand it also determines some properties of a table such as having legs and a flat horizontal surface that one can lay something on. DL concepts correspond to unary predicates in first order logic and to classes in frame-based systems;
- **Roles** – Represent relationships between objects. For example the role 'lays on' might define the relationship between a book and a table, where the book lays upon the table. Roles can also be applied to concepts. However they do not describe the relationship between the classes (concepts), rather they describe the properties of those objects that are members of that classes;
- **Rules** – In DL, rules take the form of 'if condition  $x$  (left side), then property  $y$  (right side)' and form statements that read as 'if an object satisfies the condition on the left side, then it has the properties of the right side'. So, for instance, a rule can state something like 'all objects that are male and have at least one child are fathers'.

The family of description logic system consists of many members that differ mainly with respect to the constructs they provide. Not all of the constructs can be found in a single DL system. For a listing of some concrete constructs with a brief explanation of their semantics, refer to [1].

## The Web Ontology Language (OWL) and its Dialects

In order achieve the goal of a broad range of shared ontologies using vocabularies with expressiveness appropriate for each domain, the Semantic Web requires a scalable high-performance storage and reasoning infrastructure. The major challenge towards building such an infrastructure is the expressivity of the underlying standards: RDF [24], RDFS [3], OWL [8] and OWL 2 [43]. Even though RDFS can be considered a simple KR language, it is already a challenging task to implement a repository for it, which provides performance and scalability comparable to those of relational database management systems (RDBMS). Even the simplest dialect of OWL (OWL-Lite) is a description logic (DL) that does not scale due to reasoning complexity. Furthermore, the semantics of OWL-Lite are incompatible with that of RDF(S), see [12].

### Naïve OWL Fragments Map

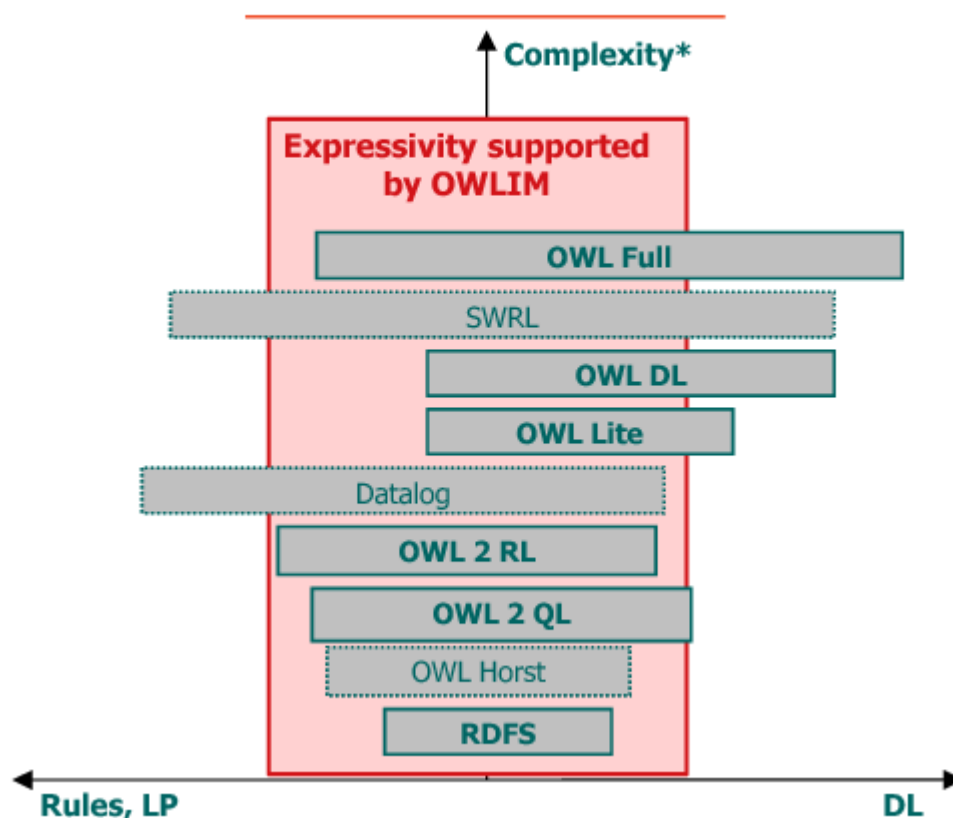


Figure 3 - OWL Layering Map

### OWL DLP

[OWL DLP](#) is a non-standard dialect, offering a promising compromise between expressive power, efficient reasoning, and compatibility. It is defined in [12] as the intersection of the expressivity of OWL-DL and logic programming. In fact, OWL DLP is defined as the most expressive sub-language of OWL DL, which can be mapped to [Datalog](#). OWL DLP is simpler than OWL-Lite. The alignment of its semantics to RDFS is easier, as compared to OWL-Lite and OWL-DL dialects. Still, this can only be achieved through the enforcement of some additional modelling constraints and transformations. A broad collection of information related to OWL DLP can be found on [27].

Horn logic and description logics are orthogonal (in the sense that neither of them is a subset of the other). OWL DLP is the "intersection" of Horn logic and OWL; it is the Horn-definable part of OWL, or stated another way, the OWL-definable part of Horn logic. DLP has certain advantages:

- From a modeller's perspective, there is freedom to use either OWL or rules (and associated tools and methodologies) for modelling purposes, depending on the modeller's experience and preferences.
- From an implementation perspective, either description logic reasoners or deductive rule systems can be used. This feature provides extra flexibility and ensures interoperability with a variety of tools.

Experience with using OWL has shown that existing ontologies frequently use very few constructs outside the DLP language [42].

## OWL Horst

In [37] ter Horst defines RDFS extensions towards rule support and describes a fragment of OWL, more expressive than DLP. He introduces the notion of [R-entailment](#) of one (target) RDF graph from another (source) RDF graph on the basis of a set of entailment rules *R*. R-entailment is more general than the [D-entailment](#) used by Hayes [17] in defining the standard RDFS semantics. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are "extended" RDF statements, where variables can take any of the three positions.

The head of the rule comprises one or more consequences, each of which is, again, an extended RDF statement. The consequences may not contain free variables, i.e. which are not used in the body of the rule. The consequences may contain blank nodes.

The extension of R-entailment (as compared to D-entailment) is that it "operates" on top of so-called generalized RDF graphs, where blank nodes can appear as predicates. R-entailment rules without premises are used to declare axiomatic statements. Rules without consequences are used to detect inconsistencies.

In this document we refer to this extension of RDFS as "[OWL Horst](#)". As outlined in [37], this language has a number of important characteristics:

- It is a proper (backward-compatible) extension of RDFS. In contrast to OWL DLP, it puts no constraints on the RDFS semantics. The widely discussed meta-classes (classes as instances of other classes) are not disallowed in OWL Horst. It also does not enforce the unique name assumption;
- Unlike DL-based rule languages, like [SWRL](#) [20] and [25], R-entailment provides a formalism for rule extensions without DL-related constraints;
- Its complexity is lower than SWRL and other approaches combining DL ontologies with rules; see section 5 of [37].

In Figure 3, the pink box represents the range of expressivity of OWLIM, i.e. including OWL DLP, OWL Horst, OWL 2 RL, most of OWL Lite. However, none of the rule sets include support for the entailment of typed literals (D-entailment); more details on the semantics supported by OWLIM can be found in section 4.6.

OWL Horst is close to what SWAD-Europe has intuitively described as [OWL Tiny](#) [36]. The major difference is that OWL Tiny (like the fragment supported by OWLIM) does not support entailment over data types.

## OWL2 RL

OWL 2 [43] is a re-work of the OWL language family by the OWL working group. This work includes identifying fragments of the OWL 2 language that have desirable behavior for specific applications/environments.

The OWL 2 RL profile is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate both OWL 2 applications that can trade the full expressivity of the language for efficiency, and RDF(S) applications that need some added expressivity from OWL 2. This is achieved by defining a syntactic subset of OWL 2 which is amenable to implementation using rule-based technologies, and presenting a partial axiomatisation of the OWL 2 RDF-Based Semantics [45] in the form of first-order implications that can be used as the basis for such an implementation. The design of OWL 2 RL was inspired by Description Logic Programs [12] and pD\* [46].

## OWL Lite

The original OWL specification [8], now known as OWL 1, provides two specific subsets of OWL Full designed to be of use to implementers and language users. The OWL Lite subset was designed for easy implementation and to provide users with a functional subset that provides an easy way to start using OWL.

OWL Lite is a sublanguage of OWL DL that supports only a subset of the OWL language constructs. OWL Lite is particularly targeted at tool builders, who want to support OWL, but who want to start with a relatively simple basic set of language features. OWL Lite abides by the same semantic restrictions as OWL DL, allowing reasoning engines to guarantee certain desirable properties. A summary of the language constructs allowed in OWL Lite is given in section 8.3 of [8]. For a more formal description of the subset of OWL language constructs supported by OWL Lite the reader is referred to the Semantics and Abstract Syntax document [47].

## OWL DL

The OWL DL (where DL stands for "Description Logic") subset was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems.

OWL Full and OWL DL support the same set of OWL language constructs. Their difference lies in restrictions on the use of some of those features and on the use of RDF features. OWL Full allows free mixing of OWL with RDF Schema and, like RDF Schema, does not enforce a strict separation of classes, properties, individuals and data values. OWL DL puts constraints on mixing with RDF and requires disjointness of classes, properties, individuals and data values. The main reason for having the OWL DL sublanguage is that tool builders have developed powerful reasoning systems which support ontologies constrained by the restrictions required for OWL DL. For formal definitions of the differences between OWL Full and OWL DL see [47].

## Query Languages



In this section, we introduce some query languages for RDF. This may beg the question as to why we need RDF-specific query languages at all instead of using an XML query language. The answer is that XML is located at a lower level of abstraction than RDF. This fact would lead to complications if we were querying RDF documents with an XML-based language. The RDF query languages explicitly capture the RDF semantics in the language itself.

All the query languages discussed below have an SQL-like syntax, but there are also a few non-SQL-like languages like Versa and Adenine.

The query languages supported by Sesame (which is the Java framework within which OWLIM operates) and therefore by OWLIM, are SPARQL and SeRQL.

## RQL, RDQL

RQL (RDF Query Language) was initially developed by the Institute of Computer Science at Heraklion, Greece, in the context of the European IST project MESMUSES.3. RQL adopts the syntax of OQL (a query language standard for object-oriented databases), and, like OQL, is defined by means of a set of core queries, a set of basic filters, and a way to build new queries through functional composition and iterators.

The core queries are the basic building blocks of RQL, which give access to the RDFS-specific contents of an RDF triple store. RQL allows queries such as Class (retrieving all classes), Property (retrieving all properties) or Employee (returning all instances of the class with name Employee). This last query, of course, also returns all instances of subclasses of Employee, since these are also instances of the class Employee by virtue of the semantics of RDFS.

RDQL (RDF Data Query Language) is a query language for RDF first developed for Jena models. RDQL is an implementation of the SquishQL RDF query language, which itself is derived from rdfDB. This class of query languages regards RDF as triple data, without schema or ontology information unless explicitly included in the RDF source.

Apart from Sesame, the following systems currently provide RDQL. All these implementations are known to derive from the original grammar: Jena, RDFStore, PHP XML Classes, 3Store and RAP (RDF API for PHP).

## SPARQL

SPARQL (pronounced "sparkle") is currently the most popular RDF query language; its name is a recursive acronym that stands for "SPARQL Protocol and RDF Query Language". It was standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is now considered a key Semantic Web technology. On 15 January 2008, SPARQL became an official W3C Recommendation.

SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. Several SPARQL implementations for multiple programming languages exist at present.

## SeRQL

SeRQL (Sesame RDF Query Language, pronounced "circle") is an RDF/RDFS query language developed by Sesame's developer – Aduna – as part of Sesame. It selectively combines the best features (considered by its creators) of other query languages (RQL, RDQL, N-Triples, N3) and adds some features of its own. As of this writing, SeRQL provides advanced features not yet available in SPARQL. Some of SeRQL's most important features are:

- Graph transformation;
- RDF Schema support;
- XML Schema data-type support;
- Expressive path expression syntax;
- Optional path matching.

## Reasoning Strategies

The two principle strategies for rule-based inference are forward-chaining and backward-chaining.

- **Forward-chaining:** to start from the known facts (the explicit statements) and to perform inference in a deductive fashion. The goals of such reasoning can vary: to compute the [inferred closure](#); to answer a particular query; to infer a particular sort of knowledge (e.g. the class taxonomy);
- **Backward-chaining:** to start from a particular fact or query and to verify it or get all possible results, using deductive reasoning. In a nutshell, the reasoner decomposes (or transforms) the query (or the fact) into simpler (or alternative) facts, which are available in the KB or can be proven through further recursive transformations.

Both of these strategies has its advantages and disadvantages, which have been well studied in the history of KR and expert systems. Attempts to overcome the weak points have led to the development of various hybrid strategies (involving partial forward- and backward-chaining) which have proven efficient in many contexts.

Imagine a repository which performs total forward-chaining, i.e. it tries to make sure that after each update to the KB, the inferred closure is computed and made available for query evaluation or retrieval. This strategy is generally known as [materialisation](#). In order to avoid ambiguity with various partial materialisation approaches, let us call such an inference strategy, taken together with the monotonic entailment. When new explicit facts (statements) are added to a KB (repository) new implicit facts will likely be inferred. Under a monotonic logic, adding new explicit statements will never cause previously inferred statements to be retracted. In other words, the addition of new facts can only monotonically extend the inferred closure. assumption, [total materialisation](#).

The principle advantages and disadvantages of the total materialisation are discussed at length in [3]; here we provide just a short summary of them:

- Upload/store/addition of new facts is relatively slow, because the repository is extending the inferred closure after each transaction. In fact, all the reasoning is performed during the upload;

- Deletion of facts is also slow, because the repository should remove from the inferred closure all the facts that can no longer be proved;
- The maintenance of the inferred closure usually requires considerable additional space (RAM, disk, or both, depending on the implementation);
- Query and retrieval are fast, because no deduction, satisfiability checking, or other sorts of reasoning are required. The evaluation of queries becomes computationally comparable to the same task for relation database management systems (RDBMS).

Probably the most important advantage of the inductive systems, based on total materialisation, is that they can easily benefit from RDBMS-like query optimization techniques, as long as all the data is available at query time. The latter makes it possible for the query evaluation engine to use statistics and other means in order to make 'educated' guesses about the 'cost' and the 'selectivity' of a particular constraint. These optimizations are much more complex in the case of deductive query evaluation.

Total materialisation is adopted as the reasoning strategy in a number of popular Semantic Web repositories, including some of the standard configurations of Sesame and Jena. Based on publicly available evaluation data, it is also the only strategy which allows scalable reasoning in the range of a billion of triples; such results are published by BBN (for DAML DB) and ORACLE (for RDF support in ORACLE 11g).

## Semantic Repositories

Over the last decade the Semantic Web has emerged as an area where semantic repositories become as important as HTTP servers are today. This perspective boosted the development, under W3C driven community processes, of a number of robust metadata and ontology standards. Those standards play the role, which SQL had for the development and spread of the relational DBMS. Although designed for Semantic Web, these standards face increasing acceptance in areas like Enterprise Application Integration and life sciences.

In this document the term 'semantic repository' is used to refer to a system for storage, querying, and management of structured data with respect to ontologies. At present, there is no single well-established term for such engines. Weak synonyms are: reasoner, ontology server, metastore, semantic/triple/RDF store, database, repository, knowledge base. The different wording usually reflects a somewhat different approach to implementation, performance, intended application, etc. Introducing the term 'semantic repository' is an attempt to convey the core functionality offered by most of these tools.

Semantic repositories can be used as a replacement for database management systems (DBMS), offering easier integration of diverse data and more analytical power. In a nutshell, a semantic repository can dynamically interpret metadata schemata and ontologies, which define the structure and the semantics related to the data and the queries. Compared to the approach taken in a relational DBMS, this allows for easier changing and combining of data schemata and automated interpretation of the data.

## Introduction to Sesame

Sesame is a framework for storing, querying and reasoning with RDF data. It is implemented in Java by Aduna as an open source project and includes various storage back-ends (memory, file, database), query languages, reasoners and client-server protocols.

There are essentially two ways to use Sesame:

- as a standalone server;
- embedded in an application as a Java library.

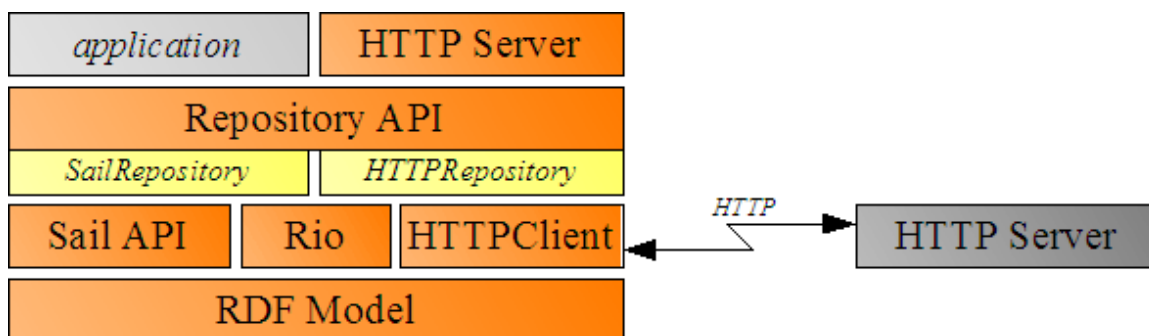
Sesame supports the W3C's SPARQL query language and Aduna's own query language SeRQL. It also supports most popular RDF file formats and query result formats.

Sesame offers a JDBC-like user API, streamlined system APIs and a RESTful HTTP interface. Various extensions are available or are being developed by third parties.

From version 2.0 onwards, Sesame requires a Java 1.5 virtual machine. All APIs use Java 5 features such as typed collections and iterators. Sesame version 2.1 added support for storing RDF data in relational databases. The supported relational databases are MySQL, PostgreSQL, MS SQL Server, and Oracle. As of version 2.2, Sesame also includes support for Mulgara (a native RDF database).

## Sesame Architecture

A schematic representation of Sesame's architecture is shown in Figure 4. Following is a brief overview of the main components.



**Figure 4 -- The Sesame architecture** Reproduced from the Sesame 2 online documentation at <http://www.openrdf.org>

The Sesame framework is as a loosely coupled set of components, where alternative implementations can be easily exchanged. Sesame comes with a variety of Storage And Inference Layer (SAIL) implementations that a user can select for the desired behaviour (in memory storage, file-system, relational database, etc). OWLIM is a plug-in SAIL component for the Sesame framework.



Applications will normally communicate with Sesame through the Repository API. This provides a high enough level of abstraction so that the details of particular underlying components remain hidden, i.e. different components can be swapped in without requiring modification of the application.

The Repository API has several implementations, one of which uses HTTP to communicate with a remote repository that exposes the Repository API via HTTP.

## The SAIL API

The SAIL API is a set of Java interfaces that support the storage and retrieval of RDF statements. The main characteristics of the SAIL API are:

- It is the basic interface for storing/retrieving/deleting RDF data;
- It is used to abstract from the actual storage mechanism, e.g. an implementation can use relational databases, file systems, in-memory storage, etc;
- It is flexible enough to support small and simple implementations, but also offers enough freedom for optimizations that huge amounts of data can be handled efficiently on enterprise-level machines;
- It is extendable to other RDF-based languages;
- It supports stacking of SAILs, where the SAIL at the top can perform some action when the modules make calls to it, and then forward these calls to the SAIL beneath it. This process continues until one of the SAILs finally handles the actual retrieval request, propagating the result back up again;
- It handles concurrency and takes care of read and write locks on repositories. This setup allows for supporting concurrency control for any type of repository.

Other proposals for RDF APIs are currently under development. The most prominent of these are the Jena toolkit and the Redland Application Framework. The SAIL shares many characteristics with both approaches, however an important difference between these two proposals and SAIL, is that the SAIL API specifically deals with RDFS on the retrieval side: it offers methods for querying class and property subsumption, and domain and range restrictions. In contrast, both Jena and Redland focus exclusively on the RDF triple set, leaving interpretation of these triples to the user application. In SAIL, these RDFS inferencing tasks are handled internally. The main reason for this is that there is a strong relationship between the efficiency of inference and the actual storage model being used. Since any particular SAIL implementation has a complete understanding of the storage model (e.g. the database schema in the case of an RDBMS), this knowledge can be exploited to infer, for example, class subsumption more efficiently.

Another difference between SAIL and other RDF APIs is that SAIL is considerably more lightweight: only four basic interfaces are provided, offering basic storage and retrieval functionality and transaction support. This minimal set of interfaces promotes flexibility and looser coupling between components.

The current Sesame framework offers several implementations of the SAIL API. The most important of these is the SQL92SAIL, which is a generic implementation for SQL92 [21]. This allows for connecting to any RDBMS without having to re-implement a lot of code. In the SQL92SAIL, only the definitions of the datatypes (which are not part of the SQL92 standard) have to be changed when switching to a different database platform. The SQL92SAIL features an inferencing module for RDFS, based on the RDFS entailment rules as specified in the RDF Model Theory [17]. This inferencing module computes the closure of the data's schema and asserts these implications as derived statements. For example, whenever a statement of the form **(foo, rdfs:domain, bar)** is encountered, the inferencing module asserts that **(foo, rdf:type, property)** is an implied statement. The SQL92SAIL has been tested in use with several DBMSs, including PostgreSQL8 and MySQL9. [5]

## Primer Introduction to OWLIM

OWLIM is a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. This section describes the various editions of OWLIM.

- [Advantages of OWLIM](#)
- [Limitations of OWLIM](#)
- [OWLIM Interoperability and Architecture](#)
- [The TRREE Engine](#)
- [Comparison of OWLIM-Lite and OWLIM-SE](#)
- [Supported Semantics](#)
  - [Pre-defined Rule Sets](#)
  - [Custom Rule-Sets](#)
  - [OWL Compliance](#)

OWLIM is based on Ontotext's Triple Reasoning and Rule Entailment Engine (TRREE) – a native RDF rule-entailment engine. The supported semantics can be configured through the definition of rule-sets. The most expressive pre-defined rule-set combines unconstrained RDFS and OWL-Lite. Custom rule-sets allow tuning for optimal performance and expressivity. OWLIM supports RDFS (section 3.1.2), OWL DLP (section 3.1.5.1), OWL Horst (section 3.1.5.2), most of OWL Lite (section 3.1.5.4) and OWL2 RL (section 3.1.5.3).

The three editions of OWLIM are OWLIM-Lite, OWLIM-SE (standard edition) and OWLIM-Enterprise (cluster configuration). With OWLIM-Lite, reasoning and query evaluation are performed in-memory, while, at the same time, a reliable persistence strategy assures data preservation, consistency, and integrity. OWLIM-SE is the high-performance 'enterprise' edition that scales to massive quantities of data. Typically, OWLIM-Lite can manage millions of explicit statements on desktop hardware, whereas OWLIM-SE can manage billions of statements and multiple simultaneous user sessions. OWLIM-Enterprise is an enterprise grade cluster management component that uses a collection of OWLIM instances to provide a resilient, high-performance semantic database.

The key differences between the editions of OWLIM are discussed in section 4.5 and in the OWLIM presentation [28]. The results form a number of benchmarks, as well as plenty of other performance evaluation and analysis information, are available on the Web site <http://www.ontotext.com/owlim/>.

## Advantages of OWLIM

One of the main advantages of OWLIM-Lite is the in-memory reasoning implementation: the full content of the repository is loaded and maintained in main memory, which allows for efficient retrieval and query answering. Although the reasoning is handled in-memory, the OWLIM-Lite SAIL offers a relatively comprehensive persistence and backup strategy.

The persistence of OWLIM-Lite is implemented via writing to file in N-Triple format. The repository can be split into several files, where all of these except one are read-only; the writable file is considered as both the source from which the triples are loaded and the target where the new statements are stored. This backup strategy ensures that no loss of newly asserted triples can occur in cases of power failure or abnormal termination. Although relatively simple, this strategy had proven to be very efficient and reliable over the years [22].

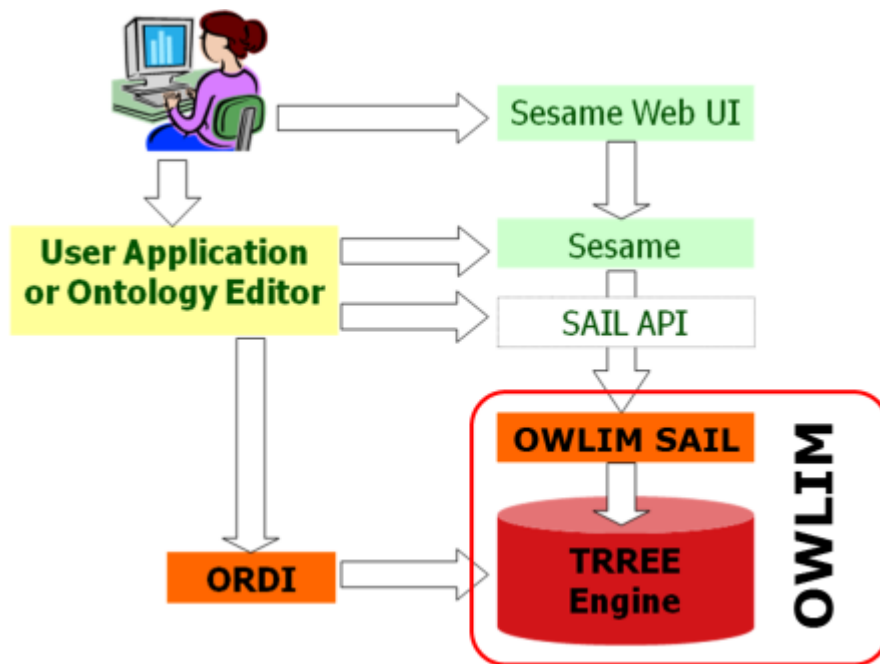
## Limitations of OWLIM

The limitations of OWLIM are related to its reasoning strategy. In general, the expressivity of the language supported cannot be extended in the Description Logic direction, because the semantics must be able to be captured in (Horn) rules. The total materialisation strategy has drawbacks when changes to the explicitly asserted statements occur frequently. For expressive semantics and certain ontologies, the number of implicit statements can grow quickly with the expected degradation in performance. OWLIM-SE has a number of optimisations to reduce this problem, e.g. special handling of **owl:sameAs**. Removing explicit statements can adversely affect performance if the full closure needs to be recomputed. Again, OWLIM-SE uses special techniques to avoid this situation. Another limitation of OWLIM-Lite is that the volume of data it can process is limited by the size of the computer's main memory. Considering currently available commodity hardware, OWLIM-Lite can handle millions of statements on desktop machines and above ten million on entry-level servers..

## OWLIM Interoperability and Architecture

OWLIM version 3.X is packaged as a Storage and Inference Layer (SAIL) for Sesame version 2.x and makes extensive use of the features and infrastructure of Sesame, especially the RDF model, RDF parsers and query engines.

Inference is performed by the TRREE engine [39], where the explicit and inferred statements are stored in highly-optimized data structures that are kept in-memory for query evaluation and further inference. The inferred closure is updated through inference at the end of each transaction that modifies the repository.



**Figure 5 - OWLIM Usage and Relationship to Sesame and ORDI**

OWLIM implements the Sesame SAIL interface so that it can be integrated with the rest of the Sesame framework, e.g. the query engines and the web UI. A user application can be designed to use OWLIM directly through the Sesame SAIL API or via the higher-level functional interfaces. When an OWLIM repository is exposed using the Sesame HTTP Server, users can manage the repository through the Sesame Workbench Web application, or with other tools integrated with Sesame, e.g. ontology editors like Protégé and TopBraid Composer.

The easiest way for developers to integrate their applications with OWLIM is to use it with the Sesame framework as a set of libraries. The installation and configuration of OWLIM are discussed in the quick start and user guides. More information on the various aspects of the Sesame specifications, its architecture and implementations can be found in section 3.2.

## The TRREE Engine

OWLIM is implemented on top of the TRREE engine. TRREE [39] stands for 'Triple Reasoning and Rule Entailment Engine'. The TRREE performs reasoning based on forward-chaining of entailment rules over RDF triple patterns with variables. TRREE's reasoning strategy is total materialisation, see section 3.1.7, although various optimisations are used as described in the following sections.

The semantics used is based on R-entailment [37] with the following differences:

- Free variables in the head of a rule (without a binding in the body) are treated as blank nodes. This feature can be considered 'syntactic sugar';
- Variable inequality constraints can be specified in the body of the rules, in addition to the triple patterns. This leads to lower complexity as compared to R-entailment;
- the **[cut]** operator can be associated with rule premises, the TRREE compiler interprets it like the **!** operator in Prolog;
- Two types of inconsistency checks are supported. Checks without any consequences indicate a consistency violation if the body can be satisfied. Consistency checks with consequences indicate a consistency violation if the inferred statements do not exist in the repository;
- Axioms can be provided as a set of statements, although those are not modelled as rules with empty bodies.

Further details of the rule language can be found in the corresponding user guides.

The TRREE can be configured via the **rule-sets** parameter, that identifies a file containing the entailment rules, consistency checks and axiomatic triples. The implementation of TRREE relies on a compile stage, during which custom rule-sets are compiled into Java code that is further compiled and merged in to the inference engine.

The edition of TRREE used in OWLIM-Lite is referred to as 'SwiftTRREE' and performs reasoning and query evaluation in-memory. The edition of TRREE used in OWLIM-SE is referred to as 'BigTRREE' and utilises data structures backed by the file-system. These data structures are organized to allow query optimizations that dramatically improve performance with large datasets, e.g. with one of the standard tests OWLIM-SE evaluates queries against 7 million statements three times faster than OWLIM-Lite, although it takes between two and three times more time to initially load the data.

## Comparison of OWLIM-Lite and OWLIM-SE

The two OWLIM editions – OWLIM-Lite and OWLIM-SE – are identical in terms of usage and integration except for a few minor differences in some configuration parameters. The editions differ in which version of the TRREE engine they are based upon, but share the same inference and semantics (rule-compiler, etc).

OWLIM-Lite is designed for medium data volumes and prototyping. Its key features are:

- reasoning and query evaluation performed in main memory;
- persistence strategy that assures data preservation and consistency;
- extremely fast loading of data (including inference and storage).

OWLIM-SE is suitable for massive volumes of data and heavy query loads. It is designed as an enterprise-grade semantic repository system. It features:

- file-based indices (enables it to scale to billions of statements even on desktop machines.)
- inference and query optimizations (ensures fast query evaluations.)

Parameter	OWLIM-Lite	OWLIM-SE
<b>Scale</b>	10 MSt, using 1.6 GB RAM <b>100 MSt</b> , using 16 GB RAM	130 MSt, using 1.6GB <b>1068 MSt</b> , using 12GB
<b>Processing speed</b> (load+infer+store)	30 KSt/s on notebook <b>200 KSt/s</b> on server	5 KSt/s on notebook <b>60 KSt/s</b> on server
<b>Query optimization</b>	No	Yes
<b>Persistence</b>	Back-up in N-Triples	Binary data files and indices
<b>Efficient owl:sameAs</b>	No	Yes
Advanced features	none	RDF Rank Full-text search Geo-spatial extension
<b>Licence and Availability</b>	Free-for-use	Commercial. Research and evaluation copies provided for free

*Table 2 - Comparison between OWLIM-Lite and OWLIM-SE*

## Supported Semantics

OWLIM offers several predefined semantics by way of standard rule sets (files), but can also be configured to use custom rule sets with semantics better tuned to the particular domain. The required semantics can be specified through the **ruleset** for each specific repository instance. Applications, which do not need the complexity of the most expressive supported semantics, can choose one of the less complex, which will result in faster inference.

## Pre-defined Rule Sets

The pre-defined rule-sets are layered such that each one extends the preceding one. The following list is ordered by increasing expressivity:

- **empty**: no reasoning, i.e. OWLIM operates as a plain RDF store;
- **rdfs**: supports standard RDFS semantics;
- **owl-horst**: OWL dialect close to OWL Horst; the differences are discussed below;
- **owl-max**: a combination of most of OWL-Lite with RDFS.
- **owl2-rl**: Fully conformant OWL2 RL profile [44] except for D-Entailment, i.e. reasoning about data types;

## Custom Rule-Sets

OWLIM has an internal rule compiler that can be used to configure the TRREE with a custom set of inference rules and axioms. The user may define a custom rule-set in a \*.pie file (e.g. MySemantics.pie). The easiest way to do this is to start modifying one of the .pie files that were used to build the precompiled rule-sets – all pre-defined .pie files are included in the distribution. The syntax of the .pie files is easy to follow.

## OWL Compliance

Regarding OWL compliance, OWLIM supports several OWL like dialects: OWL Horst [37] (**owl-horst**), OWL Max (**owl-max**) that covers most of OWL-Lite and RDFS, OWL2 QL (**owl2-ql**) and OWL2 RL (**owl2-rl**).

With the **owl-max** rule-set OWLIM supports the following semantics:

- full RDFS semantics without constraints or limitations, apart from the entailments related to typed literals (known as D-entailment). For instance, meta-classes (and any arbitrary mixture of class, property, and individual) can be combined with the supported OWL semantics
- most of OWL-Lite
- all of OWL DLP

The differences between OWL Horst [37], and the OWL dialects supported by OWLIM (**owl-horst** and **owl-max**) can be summarized as follows:

- OWLIM does not provide the extended support for typed literals, introduced with the D\*-entailment extension of the RDFS semantics. Although such support is conceptually clear and easy to implement, it is our understanding that the performance penalty is too high for most applications. One can easily implement the rules defined for this purpose by ter Horst and add them to a custom rule-set;
- There are no inconsistency rules by default;
- A few more OWL primitives are supported by OWLIM (rule-set **owl-max**). These are listed in the OWLIM User Guides;
- There is extended support for schema-level (T-Box) reasoning in OWLIM.

Even though the concrete rules pre-defined in OWLIM differ from those defined in OWL Horst, the complexity and decidability results reported for R-entailment are relevant for TRREE and OWLIM. To put it more precisely, the rules in the **owl-horst** rule-set, do not introduce new B-Nodes, which means that R-entailment with respect to them takes polynomial time. In KR terms, this means that the **owl-horst** inference within OWLIM is tractable.

Inference using **owl-horst** is of a lesser complexity compared to other formalisms that combine DL formalisms with rules. In addition, it puts no constraints with respect to meta-modelling.

The correctness of the support for OWL semantics (for those primitives which are supported) is checked against the normative Positive- and Negative-entailment OWL test cases [7]. These tests are provided in the OWLIM distribution and documented in the OWLIM user guides.

## Primer Glossary

Datalog	A query and rule language for deductive databases that syntactically is a subset of Prolog.
D-entailment	A vocabulary entailment of an RDF graph that respects the 'meaning' of data types.
Description Logics	A family of formal knowledge representation languages that are subsets of first order logic, but have more efficient decision problems.
Horn Logic	Broadly means a system of logic whose semantics can be captured by Horn clauses. A Horn clause has at most one positive literal and allows for an IF...THEN interpretation, hence the common term 'Horn Rule'.
Knowledge Base	(In the Semantic Web sense) is a database of both assertions (ground statements) and an inference system for deducing further knowledge based on the structure of the data and a formal vocabulary.
Knowledge Representation	An area in artificial intelligence that concerns representing knowledge in a formal way such that it permits automated processing (reasoning).
Materialisation	The process of inferring and storing (for later retrieval or use in query answering) every piece of information that can be deduced from a knowledge base's asserted facts and vocabulary.
Named Graph	A group of statements identified by a URI. It allows a subset of statements in a repository to be manipulated or processed separately.
Ontology	A shared conceptualisation of a domain, described using a formal (knowledge) representation language.
OWL	See 'Web Ontology Language'
OWL Horst	An entailment system built upon RDF Schema, see R-entailment.
Predicate Logic	Generic term for symbolic formal systems like first-order logic, second-order logic, etc. Its formulas may contain variables which can be quantified.
RDF Graph Model	The interpretation of a collection of RDF triples as a graph, where resources are nodes in the graph and predicates form the arcs between nodes. Therefore one statement leads to one arc between two nodes (subject and object).
RDF Schema	A vocabulary description language for RDF with formal semantics.
Resource	An element of the RDF model that represents a thing that can be described, i.e. a unique name to identify an object or concept.
R-entailment	A more general semantics layered on RDFS where any set of rules (i.e. rules that extend or even modify RDFS) are permitted. Rules are of the form IF...THEN... and use RDF statement patterns in their premises and consequences, with variables allowed in any position.
Resource Description Framework (RDF)	A family of W3C specifications for modelling knowledge with a variety of syntaxes.
Semantic Repository	A knowledge base engine that is specifically designed to support RDF and the semantics of the various formal languages based upon it.
Semantic Web	The concept of attaching machine understandable metadata to all information published on the internet, so that intelligent agents can consume, combine and process information in an automated fashion.
SPARQL	The most popular RDF query language.
Statement	See 'triple'.
Triple	A basic unit of information expression in RDF. A triple consists of a subject-predicate-object.
Universal Resource Identifier (URI)	Is a string of characters used to (uniquely) identify a resource.
Web Ontology Language (OWL)	A family of W3C knowledge representation languages that can be used to create ontologies.

## Primer References

1. Bergmann, F. Introduction to Description Logics. Web page, <http://www.fraber.de/sitec/dl.html>
2. Box, D; Ehnebuske, D; Kakiyaya, G; Layman, A; Mendelsohn, N; Nielsen, H F; Thatte, S; Winer, D. Simple Object Access Protocol (SOAP) 1.1, W3C note, World Wide Web Consortium, May 2000 <http://www.w3.org/TR/SOAP/>
3. Brickley, D., Guha, R.V; (eds.). Resource Description Framework(RDF) Schemas, W3C <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>
4. Broekstra, J. Storage, Querying and Inferencing for Semantic Web Languages. Ph.D. thesis, SIKS Dissertation Series No. 2005-09, ISBN 90 9019 2360, Vrije Universiteit Amsterdam. 2005. <http://www.cs.vu.nl/~jbroeks/#pub>
5. Broekstra, J; Kampman, A; van Harmelen, F. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. *International Semantic Web Conference*, Sardinia, Italy, 2002.
6. Carroll, J J; Bizer, C; Hayes, P; Stickler, P. Named Graphs, Provenance and Trust. *International Semantic Web Conference*, Hiroshima, Japan, 2004.
7. Carroll, J. J; De Roo, J. OWL Web Ontology Language: Test Cases. *W3C Recommendation* 10 Feb. 2004. <http://www.w3.org/TR/owl-test/>
8. Dean, M; Schreiber, G; (eds.); Bechhofer, S; van Harmelen, F; Hendler, J; Horrocks, I; McGuinness, D L; Patel-Schneider, P F; Stein, L A. OWL Web Ontology Language Reference. *W3C Recommendation*. 10 Feb. 2004. <http://www.w3.org/TR/owl-ref/>
9. Dublin Core Metadata Element Set, Version 1.1. <http://dublincore.org/documents/dces/>
10. Dublin Core Metadata Element Set, Version 1.1: Reference Description. <http://dublincore.org/documents/2003/06/02/dces/>
11. Ehrig, M; Haase, P; Hefke, M; Stojanovic, N. Similarity for ontologies – a Comprehensive Framework. *Proc. 13th European Conference on Information Systems*, May 2005.
12. Grosz, B; Horrocks, I; Volz, R; Decker, S. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of WWW2003*, Budapest, May 2003.
13. Gruber, T R. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993. [http://ksl-web.stanford.edu/KSL\\_Abstracts/KSL-92-71.html](http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html)
14. Gruber, T R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Presented at the Padua workshop on Formal Ontology. March 1993, later published in *International Journal of Human-Computer Studies*, Vol. 43, Issues 4-5, Nov. 1995, pp. 907-928. <http://tomgruber.org/writing/onto-design.htm>
15. Guarino, N; Giaretta, P. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam: pp. 25-32. 1995
16. Guarino, N. Formal Ontology in Information Systems. *Proceedings of FOIS'98*, Trento, Italy, June 6-8, 1998. Amsterdam, IOS Press. <http://www.loa-cnr.it/Papers/FOIS98.pdf>
17. Hayes, P. RDF Model Theory. *Working draft*, World Wide Web Consortium. September 2001. <http://www.w3.org/TR/rdf-mt/>
18. Hayes, P. RDF Semantics. *W3C Recommendation*. Feb. 10, 2004. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
19. Hillmann, D.; Using Dublin Core; *DCMI Recommended Resource*. Nov. 7, 2005
20. Horrocks, I. Patel-Schneider, P F, Bechhofer, S, Tsarkov, D. OWL Rules: A Proposal and Prototype Implementation. *Journal of Web Semantics*, 3 (2005), pp. 23-40.
21. ISO. Information Technology-Database Language SQL. *Standard No. ISO/IEC 9075:1999*, International Organization for Standardization (ISO), 1999. (Available from American National Standards Institute, New York, NY 10036, (212) 642-4900.)
22. KIM. Home page, <http://www.ontotext.com/kim>
23. Kiryakov, A. Ontologies for Knowledge Management, Chapter 7 in: Davies, J; Studer, R; Warren, P. (eds.). *Semantic Web Technologies: Trends and Research in Ontology-based Systems*. Wiley, UK, 2006.
24. Klyne, G; Carroll, J. J; (eds). (2004). Resource Description Framework(RDF): Concepts and Abstract Syntax. *W3C Recommendation* 10 Feb. 2004. <http://www.w3.org/TR/rdf-concepts/>
25. Motik, B; Sattler, U; Studer R. Query Answering for OWL-DL with Rules. *Journal of Web Semantics*, issue 3 (2005), pp. 41-60.
26. Named Graphs. W3C Overview. <http://www.w3.org/2004/03/trix/>
27. Ontology Logic and Reasoning at Semantic Karlsruhe. Home page, <http://logic.aifb.uni-karlsruhe.de/>
28. OWLIM – Pragmatic OWL Semantic Repository. Presentation slides, Ontotext AD, 2008 <http://www.ontotext.com/owlim/OWLIMPRes.pdf>
29. OWLIM Tests and Benchmarks. Ontotext Lab. 2007. <http://www.ontotext.com/owlim/v2.9.0/doc/OWLIMTest.pdf>
30. Popov, B; Kiryakov, A; Kirilov, A; Manov, D; Ognyanoff, D; Goranov, M. KIM – Semantic Annotation Platform. In *The Semantic Web - ISWC 2003*, Sanibel Island, USA, 2003.
31. PROTON Ontology (PROTo Ontology). Home page. <http://proton.semanticweb.org/>
32. RDF Primer. In *W3C Recommendation* 10 February 2004. <http://www.w3.org/TR/rdf-primer/>
33. RDF/XML Syntax Specification (Revised). In *W3C Recommendation*, 10 February 2004. <http://www.w3.org/TR/rdf-syntax-grammar/>
34. Resource Description Framework(RDF): Concepts and Abstract Syntax, section Graph Data Model. In *W3C Recommendation* 10 February 2004. <http://www.w3.org/TR/rdf-concepts/#section-data-model>
35. Semantic Knowledge Technologies (SEKT). Home page. <http://www.sekt-project.com/>
36. SWAD-Europe Workshop on Semantic Web *Storage and Retrieval*. Amsterdam, Holland, November 2003. [http://www.w3.org/2001/sw/Europe/reports/dev\\_workshop\\_report\\_4/#owl-tiny](http://www.w3.org/2001/sw/Europe/reports/dev_workshop_report_4/#owl-tiny).
37. ter Horst, H J. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In *Proc. of ISWC 2005*, Galway, Ireland, Nov. 6-10, 2005. LNCS 3729, pp. 668-684.
38. Terziev, I; Kiryakov, A. PROTo Ontology: A Base Upper-Level Ontology for the Semantic Web, *SEKT Q4 Meeting*. Innsbruck, Austria., Jan. 17-19, 2005. <http://proton.semanticweb.org/PROTON.ppt>
39. TRREE – Triple Reasoning and Rule Entailment Engine. Home page. <http://ontotext.com/tree/>
40. Uniform Resource Identifier, Wikipedia. <http://en.wikipedia.org/wiki/URI>
41. User Guide of Sesame. Aduna b. v. <http://www.openrdf.org/doc/sesame/users/index.html>
42. T.D. Wang, B. Parsia, and J. Hendler. A survey of the web ontology landscape. *Lecture Notes in Computer Science*, 4273:682, 2006.
43. Hitzler, Pascal; Krötzsch, Markus; Parsia, Bijan; Patel-Schneider, Peter F.; Rudolph, Sebastian (27 October 2009). "OWL 2 Web Ontology Language Primer". OWL 2 Web Ontology Language. World Wide Web Consortium. <http://www.w3.org/TR/2009/REC-owl2-primer-20091027/>
44. Motik, Boris; Cuenca Grau, Bernardo; Horrocks, Ian; Wu, Zhe; Fokoue, Achille; Lutz, Carsten (27 October 2009) "OWL 2 Web



Ontology Language Profiles". OWL 2 Web Ontology Language. World Wide Web Consortium.  
<http://www.w3.org/TR/owl2-profiles/>

45. Carroll, Jeremy; Herman, Ivan; Patel-Schneider, Peter F. (27 October 2009) "OWL 2 Web Ontology Language RDF-Based Semantics". OWL 2 Web Ontology Language. World Wide Web Consortium. <http://www.w3.org/TR/owl-rdf-based-semantics/>
46. ter Horst, Herman J. (2005) "Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary". Journal of Web Semantics 3(2-3):79-115, 2005
47. Patel-Schneider, Peter F.; Hayes, Patrick; Horrocks, Ian (10 February 2004) "OWL Web Ontology Language Semantics and Abstract Syntax". W3C Recommendation <http://www.w3.org/TR/owl-semantics/>