OWL Web Ontology Language Reference

W3C Recommendation 10 February 2004

This version:
>   http://www.w3.org/TR/2004/REC-owl-ref-20040210/

Latest version:
>   http://www.w3.org/TR/owl-ref/

Previous version:
>   http://www.w3.org/TR/2003/PR-owl-ref-20031215/

Editors:
>   Mike Dean, BBN Technologies
>   Guus Schreiber, Free University Amsterdam

Authors:
>   Sean Bechhofer, University of Manchester
>   Frank van Harmelen, Free University Amsterdam
>   Jim Hendler, University of Maryland
>   Ian Horrocks, University of Manchester
>   Deborah L. McGuinness, Stanford University
>   Peter F. Patel-Schneider, Bell Labs Research, Lucent Technologies
>   Lynn Andrea Stein, Franklin W. Olin College of Engineering

Please refer to the **errata** for this document, which may include some normative corrections.

See also translations.

Abstract

The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language. This document contains a structured informal description of the full set of OWL language constructs and is meant to serve as a reference for OWL users who want to construct OWL ontologies.

Status of this document

This document has been reviewed by W3C Members and other interested parties, and it has been endorsed by the Director as a W3C Recommendation. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This is one of six parts of the W3C Recommendation for OWL, the Web Ontology Language. It has been developed by the Web Ontology Working Group as part of the W3C Semantic Web Activity (Activity Statement, Group Charter) for publication on 10 February 2004.

The design of OWL expressed in earlier versions of these documents has been widely reviewed and satisfies the Working Group's technical requirements. The Working Group has addressed all comments received, making changes as necessary. Changes to this document since the Proposed Recommendation version are detailed in the change log.

Comments are welcome at public-webont-comments@w3.org (archive) and general discussion of related technology is welcome at www-rdf-logic@w3.org (archive).

A list of implementations is available.

The W3C maintains a list of any patent disclosures related to this work.

This section describes the status of this document at the time of its publication.

Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at http://www.w3.org/TR/.

Acknowledgments

Parts of this document are derived from the DAML+OIL (March 2001) Reference Description [*DAML+OIL*] which was submitted as part of the [DAML+OIL W3C Note](#) . The sponsors of this document and its predecessor documents are gratefully acknowledged.

Jeremy Carroll, Jim Hendler, Brian McBride and Peter Patel-Schneider provided substantive reviews and contributed text to this document. Jeff Heflin contributed the section on deprecation. Jerome Euzenat contributed the example for an enumerated datatype.

This document is the result of extensive discussions within the [Web Ontology Working Group](#) as a whole. The participants in this Working Group included: Yasser alSafadi, Jean-François Baget, James Barnette, Sean Bechhofer, Jonathan Borden, Frederik Brysse, Stephen Buswell, Jeremy Carroll, Dan Connolly, Peter Crowther, Jonathan Dale, Jos De Roo, David De Roure, Mike Dean, Larry Eshelman, Jérôme Euzenat, Tim Finin, Nicholas Gibbins, Sandro Hawke, Patrick Hayes, Jeff Heflin, Ziv Hellman, James Hendler, Bernard Horan, Masahiro Hori, Ian Horrocks, Jane Hunter, Francesco Iannuzzelli, Rüdiger Klein, Natasha Kravtsova, Ora Lassila, Massimo Marchiori, Deborah McGuinness, Enrico Motta, Leo Obrst, Mehrdad Omidvari, Martin Pike, Marwan Sabbouh, Guus Schreiber, Noboru Shimizu, Michael Sintek, Michael K. Smith, John Stanton, Lynn Andrea Stein, Herman ter Horst, David Trastour, Frank van Harmelen, Bernard Vatant, Raphael Volz, Evan Wallace, Christopher Welty, Charles White, and John Yanosy.

Contents

1. Introduction

1.1 Purpose of this document

This document gives a systematic, compact and informative description of all the modelling primitives of OWL, using the RDF/XML exchange syntax for OWL. We expect this document to serve as a reference guide for users of the OWL language.

This document is one component of the description of OWL, the Web Ontology Language, being produced by the W3C Web Ontology Working Group. The Document Roadmap section of the OWL Overview document describes each of the different parts and how they fit together. Readers unfamiliar with OWL may wish to first consult the OWL Overview document [*OWL Overview*], and subsequently the OWL Guide [*OWL Guide*] for a more narrative description and examples of the use of the language.

This document assumes the reader is familiar with the basic concepts of RDF [*RDF Concepts*] and has a working knowledge of the RDF/XML syntax [*RDF/XML Syntax*] and of RDF Schema [*RDF Vocabulary*].

The normative reference on the precise syntax of the OWL language constructs can be found in the OWL Semantics and Abstract Syntax document [*OWL S&AS*]. That document also contains a precise definition of the meaning of the language constructs in the form of a model-theoretic semantics. Notions such as consistency of OWL ontologies are discussed in that document.

Use cases and requirements for the OWL language are described in the OWL requirements document [*OWL Requirements*]. Test cases for OWL tools (*e.g.*, entailment tests, consistency tests) are specified in the Test document [*OWL Test Cases*] .

1.2 OWL Full/DL/Lite

As also discussed in the OWL Overview document [*OWL Overview*], and subsequently the OWL Guide [*OWL Guide*], the OWL language provides two specific subsets that we believe will be of use to implementors and language users. OWL Lite was designed for easy implementation and to provide users with a functional subset that will get them started in the use of OWL. OWL DL

(where DL stands for "Description Logic") was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems. The complete OWL language (called OWL Full to distinguish it from the subsets) relaxes some of the constraints on OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violate the constraints of Description Logic reasoners.

> NOTE: RDF documents will generally be in OWL Full, unless they are specifically constructed to be in OWL DL or Lite.

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 the 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 the formal definitions of the differences between OWL Full and OWL DL the reader is referred to the Semantics and Abstract Syntax document [*OWL S&AS*]. Sec. 8.2 "OWL DL" summarizes the differences between OWL Full and OWL DL.

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 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 Sec. 8.3. 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 [*OWL S&AS*].

> NOTE: RDF users upgrading to OWL should be aware that OWL Lite is *not* simply an extension of RDF Schema. OWL Lite is a light version of OWL DL and puts constraints on the use of the RDF vocabulary (*e.g.*, disjointness of classes, properties, etc.). OWL Full is designed for maximal RDF compatibility and is therefore the natural place to start for RDF users. When opting for either OWL DL or OWL Lite one should consider whether the advantages of OWL DL/Lite (*e.g.*, reasoning support) outweigh the DL/Lite restrictions on the

use of OWL and RDF constructs.

NOTE: OWL Lite is defined in this document as a number of additional restrictions on OWL DL. This mean that, OWL DL constructs are also part of OWL Lite, unless explicitly stated otherwise. Sec. 8.3.provides a summary of these additional OWL Lite restrictions.

1.3 OWL syntax

An OWL ontology is an RDF graph [*RDF Concepts*], which is in turn a set of RDF triples. As with any RDF graph, an OWL ontology graph can be written in many different syntactic forms (as described in the RDF/XML Syntax Specification (Revised) [*RDF/XML Syntax*]). The current document uses some specific syntactic forms of RDF/XML for representing triples (as does the Guide document) . However, the meaning of an OWL ontology is solely determined by the RDF graph. Thus, it is allowable to use other syntactic RDF/XML forms, as long as these result in the same underlying set of RDF triples. Such other syntactic forms would then carry exactly the same meaning as the syntactic form used in this document.

As a simple example of an alternative syntactic form resulting in the same RDF triples, consider the following RDF/XML syntax:

```
<owl:Class rdf:ID="Continent"/>
```

The following RDF/XML syntax:

```
<rdf:Description rdf:about="#Continent">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/
owl#Class"/>
</rdf:Description>
```

encodes the same set of RDF triples, and therefore would convey the same meaning.

1.4 OWL and RDF semantics

OWL is a vocabulary extension of RDF [*RDF Semantics*]. Thus any RDF graph forms an OWL Full ontology. Further, the meaning given to an RDF graph by OWL includes the meaning given to the graph by RDF. OWL Full ontologies can

thus include arbitrary RDF content, which is treated in a manner consistent with its treatment by RDF. OWL assigns an additional meaning to certain RDF triples. The OWL Semantics and Abstract Syntax document [*OWL S&AS*] specifies exactly which triples are assigned a specific meaning, and what this meaning is.

> NOTE: As remarked before, OWL DL and OWL Lite extend the RDF vocabulary, but also put restrictions on the use of this vocabulary. Therefore, RDF documents will generally be in OWL Full, unless they are specifically constructed to be in OWL DL or Lite.

1.5 A note about the examples

For readability purposes the examples in this document assume the XML entities `&rdf;, &rdfs;, &owl;` and `&xsd;` (for XML Schema datatypes) are defined in the same way as in Appendix B. The same assumption holds for the corresponding namespaces `rdf:, rdfs:, owl:` and `xsd:`.

The examples in this document are meant to serve as illustrations of the use of OWL language constructs. They do not form one consistent ontology. For an extended example the reader is referred to the Guide document [*OWL Guide*].

1.6 Data Aggregation and Privacy

OWL's ability to express ontological information about individuals appearing in multiple documents supports linking of data from diverse sources in a principled way. The underlying semantics provides support for inferences over this data that may yield unexpected results. In particular, the ability to express equivalences using owl:sameAs can be used to state that seemingly different individuals are actually the same. Similarly, owl:InverseFunctionalProperty can be used to link individuals together. For example, if a property such as `SocialSecurityNumber` is an `owl:InverseFunctionalProperty`, then two separate individuals could be inferred to be identical based on having the same value of that property. When individuals are determined to be the same by such means, information about them from different sources can be merged. This *aggregation* can be used to determine facts that are not *directly* represented in any one source.

The ability of the Semantic Web to link information from multiple sources is a desirable and powerful feature that can be used in many applications. However, the capability to merge data from multiple sources, combined with the inferential power of OWL, does have potential for abuse. It may even be illegal to create or

to process such linked information in countries with data protection laws, especially in the EU, without having a valid legal reason for such processing. Therefore, great care should be taken when using OWL with any kind of personal data that might be linked with other data sources or ontologies. Detailed security solutions were considered out of scope for the Working Group. Work currently underway elsewhere is expected to address these issues with a variety of security and preference solutions -- see for example SAML and P3P

1.7 Appendices to this document

This document has a set of appendices containing additional information.

Links in this document that are attached to definitions of language constructs provide access to the OWL Semantics and Abstract Syntax [*OWL S&AS*]. Appendix A contains a systematic set of links for each language construct to the corresponding sections in the Guide and the S&AS documents.

Appendix B contains a RDF schema for the OWL language constructs. This schema provides information about the OWL vocabulary that could be a useful reference point for ontology builders and tool developers. The restrictions provided by the schema on the OWL classes and properties are informative and not complete. Also, this schema does not make distinctions between OWL Full, OWL DL and OWL Lite. Conventionally, classes have a leading uppercase character; properties a leading lowercase character. Thus, `owl:Ontology` is a class, and `owl:imports` is a property.

> NOTE: The RDF Schema file for OWL is not expected to be imported explicitly (*i.e.,* with `owl:imports`) into an ontology. The schema has an informative status and is meant to provide the classes and properties to be used in the RDF/XML syntax. People that do import this schema should expect the resulting ontology to be an OWL Full ontology.

Appendix C gives a tabular overview of the OWL vocabulary in terms of the built-in OWL classes and properties (the latter with their domain and range).

For readers familiar with DAML+OIL, Appendix D lists many of the changes between DAML+OIL and OWL.

Finally, Appendix E provides a set of practical guidelines for specifying OWL DL ontologies in RDF.

2. OWL document

Information in OWL is gathered into ontologies, which can then be stored as documents in the World Wide Web. One aspect of OWL, the importing of ontologies, depends on this ability to store OWL ontologies in the Web.

2.1 Content

An OWL document consists of optional ontology headers (generally at most one) plus any number of class axioms, property axioms, and facts about individuals. Please note that "axiom" is the formal term used in the S&AS document. These axioms are somewhat more informally called "definitions" in the Guide and Overview documents.

> NOTE: OWL does not impose any order on OWL components. Humans writing ontologies are likely to use some sort of ordering, for example putting the ontology header in the beginning, but this has no impact on the meaning. Tools should not assume any order.

As with most RDF documents, the OWL code should be subelements of a `rdf:RDF` element. This enclosing element generally holds XML namespace and base declarations. Also, an OWL ontology document often starts with several entity declarations. For a typical example of this sort of information, see the example wine and food ontologies discussed in the Guide document [*OWL Guide*].

2.2 OWL built-in vocabulary

The built-in vocabulary for OWL all comes from the OWL namespace

```
http://www.w3.org/2002/07/owl#
```

conventionally associated with the namespace name `owl`. It is recommended that ontologies not use names from this namespace except for the built-in vocabulary. OWL tool builders should feel free to signal a warning if other names from this namespace are used, but should otherwise continue as normal.

2.3 MIME type

The Web Ontology Working Group has not requested a separate MIME type for

OWL documents. Instead, we recommend to use the MIME type requested by the RDF Core Working Group, namely `application/rdf+xml` [*RDF Concepts*], or alternatively the XML MIME type `application/xml`.

As file extension, we recommend to use either `.rdf` or `.owl`.

3. Classes

Classes provide an abstraction mechanism for grouping resources with similar characteristics. Like RDF classes, every OWL class is associated with a set of individuals, called the *class extension*. The individuals in the class extension are called the *instances* of the class. A class has an intensional meaning (the underlying concept) which is related but not equal to its class extension. Thus, two classes may have the same class extension, but still be different classes.

When in this document we use wording such as "a class of individuals ..", this should be read as "a class with a class extension containing individuals ...".

> NOTE: In OWL Lite and OWL DL an individual can never be at the same time a class: classes and individuals form disjoint domains (as do properties and data values). OWL Full allows the freedom of RDF Schema: a class may act as an instance of another (meta)class.

OWL classes are described through "class descriptions", which can be combined into "class axioms". We first describe class descriptions and subsequently turn to class axioms.

3.1 Class descriptions

A class description is the term used in this document (and in the OWL Semantics and Abstract Syntax) for the basic building blocks of class axioms (informally called class definitions in the Overview and Guide documents). A class description describes an OWL class, either by a class name or by specifying the class extension of an unnamed anonymous class.

OWL distinguishes six types of class descriptions:

1. a class identifier (a URI reference)
2. an exhaustive enumeration of individuals that together form the instances of a class
3. a property restriction

4. the intersection of two or more class descriptions
5. the union of two or more class descriptions
6. the complement of a class description

The first type is special in the sense that it describes a class through a *class name* (syntactically represented as a URI reference). The other five types of class descriptions describe an anonymous class by *placing constraints on the class extension*.

Class descriptions of type 2-6 describe, respectively, a class that contains exactly the enumerated individuals (2nd type), a class of all individuals which satisfy a particular property restriction (3rd type), or a class that satisfies boolean combinations of class descriptions (4th, 5th and 6th type). Intersection, union and complement can be respectively seen as the logical AND, OR and NOT operators. The four latter types of class descriptions lead to nested class descriptions and can thus in theory lead to arbitrarily complex class descriptions. In practice, the level of nesting is usually limited.

A type 1 class description is syntactically represented as an named instance of `owl:Class`, a subclass of `rdfs:Class`:

```
<owl:Class rdf:ID="Human"/>
```

This will assert the triple `"ex:Human rdf:type owl:Class ."`, where `ex:` is the namespace of the relevant ontology .

> NOTE: In OWL Lite and OWL DL, `owl:Class` (or `owl:Restriction`, see further) must be used for all class descriptions.

> NOTE: `owl:Class` is defined as a subclass of `rdfs:Class`. The rationale for having a separate OWL class construct lies in the restrictions on OWL DL (and thus also on OWL Lite), which imply that not all RDFS classes are legal OWL DL classes. In OWL Full these restrictions do not exist and therefore `owl:Class` and `rdfs:Class` are equivalent in OWL Full.

The other five forms of class descriptions consist of a set of RDF triples in which a blank node represents the class being described. That blank node has an `rdf:type` property whose value is `owl:Class`.

> NOTE: If one provides an RDF identifier for class descriptions of the

enumeration, intersection, union or complement type, this is not considered to be a class description, but a special kind of class axiom for complete classes. See Sec. 3.2.3 for details.

NOTE: In this document we sometimes use for readability purposes the shorthand "class description" to refer to "the class being described by the class description". Strictly speaking, these are different in the case of class descriptions of type 2-6: the class is represented by the corresponding blank node; the class description is represented by the triples that have this blank node as their subject.

Two OWL class identifiers are predefined, namely the classes owl:Thing and owl:Nothing. The class extension of `owl:Thing` is the set of all individuals. The class extension of `owl:Nothing` is the empty set. Consequently, every OWL class is a subclass of `owl:Thing` and `owl:Nothing` is a subclass of every class (for the meaning of the subclass relation, see the section on `rdfs:subClassOf`).

3.1.1 Enumeration

A class description of the "enumeration" kind is defined with the owl:oneOf property. The value of this built-in OWL property must be a list of individuals that are the instances of the class. This enables a class to be described by exhaustively enumerating its instances. The class extension of a class described with `owl:oneOf` contains exactly the enumerated individuals, no more, no less. The list of individuals is typically represented with the help of the RDF construct `rdf:parseType="Collection"`, which provides a convenient shorthand for writing down a set of list elements. For example, the following RDF/XML syntax defines a class of all continents:

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Eurasia"/>
    <owl:Thing rdf:about="#Africa"/>
    <owl:Thing rdf:about="#NorthAmerica"/>
    <owl:Thing rdf:about="#SouthAmerica"/>
    <owl:Thing rdf:about="#Australia"/>
    <owl:Thing rdf:about="#Antarctica"/>
  </owl:oneOf>
</owl:Class>
```

The RDF/XML syntax `<owl:Thing rdf:about="..."/>` refers to some individual (remember: all individuals are by definition instances of `owl:Thing`).

In the section on datatypes we will see another use of the `owl:oneOf` construct, namely to define an <u>enumeration of data values</u>.

NOTE: Enumeration is not part of OWL Lite

3.1.2 Property restrictions

A property restriction is a special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfy the restriction. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints.

A <u>value constraint</u> puts constraints on the range of the property *when applied to this particular class description*. For example, we might want to refer to those individuals whose value of the property `adjacentTo` should be some `Region`, and then use this within a class axiom, perhaps even a class axiom for `Region` itself. Note that this is different from `rdfs:range`, which is applied to all situations in which the property is used.

A <u>cardinality constraint</u> puts constraints on the number of values a property can take, *in the context of this particular class description*. For example, we might want to say that for a soccer team the `hasPlayer` property has 11 values. For a basketball team the same property would have only 5 values.

OWL also supports a limited set of constructs for defining global property cardinality, namely <u>owl:FunctionalProperty</u> and <u>owl:InverseFunctionalProperty</u> (see the section on properties).

Property restrictions have the following general form:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="(some property)" />
  (precisely one value or cardinality constraint, see
below)
</owl:Restriction>
```

The class <u>owl:Restriction</u> is defined as a subclass of <u>`owl:Class`</u>. A restriction class should have exactly one triple linking the restriction to a particular property,

using the owl:onProperty property. The restriction class should also have exactly one triple that represents the value constraint *c.q.* cardinality constraint on the property under consideration, *e.g.*, that the cardinality of the property is exactly 1.

Property restrictions can be applied both to datatype properties (properties for which the value is a data literal) and object properties (properties for which the value is an individual). For more information about this distinction, see the section on properties.

3.1.2.1 Value constraints

3.1.2.1.1 owl:allValuesFrom

The value constraint owl:allValuesFrom is a built-in OWL property that links a restriction class to either a class description or a data range. A restriction containing an `owl:allValuesFrom` constraint is used to describe a class of all individuals for which all values of the property under consideration are either members of the class extension of the class description or are data values within the specified data range. In other words, it defines a class of individuals x for which holds that if the pair (x,y) is an instance of P (the property concerned), then y should be an instance of the class description or a value in the data range, respectively.

A simple example:

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

This example describes an anonymous OWL class of all individuals for which the `hasParent` property only has values of class `Human`. Note that this class description does not state that the property always has values of this class; just that this is true for individuals that belong to the class extension of the anonymous restriction class.

> NOTE: In OWL Lite the only type of class description allowed as object of `owl:allValuesFrom` is a class name.

An `owl:allValuesFrom` constraint is analogous to the universal (for-all)

quantifier of Predicate logic - for each instance of the class that is being described, every value for P must fulfill the constraint. Also notice that the correspondence of `owl:allValuesFrom` with the universal quantifier means that an `owl:allValuesFrom` constraint for a property P is trivially satisfied for an individual that has no value for property P at all. To see why this is so, observe that the `owl:allValuesFrom` constraint demands that all values of P should be of type T, and if no such values exist, the constraint is trivially true.

3.1.2.1.2 owl:someValuesFrom

The value constraint owl:someValuesFrom is a built-in OWL property that links a restriction class to a class description or a data range. A restriction containing an `owl:someValuesFrom` constraint describes a class of all individuals for which at least one value of the property concerned is an instance of the class description or a data value in the data range. In other words, it defines a class of individuals x for which there is at least one y (either an instance of the class description or value of the data range) such that the pair (x,y) is an instance of P. This does not exclude that there are other instances (x,y') of P for which y' does not belong to the class description or data range.

The following example defines a class of individuals which have at least one parent who is a physician:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:someValuesFrom rdf:resource="#Physician" />
</owl:Restriction>
```

The `owl:someValuesFrom` constraint is analogous to the existential quantifier of Predicate logic - for each instance of the class that is being defined, there exists at least one value for P that fulfills the constraint.

> NOTE: In OWL Lite the only type of class description allowed as object of `owl:someValuesFrom` is a class name.

3.1.2.1.3 owl:hasValue

The value constraint owl:hasValue is a built-in OWL property that links a restriction class to a value V, which can be either an individual or a data value. A restriction containing a `owl:hasValue` constraint describes a class of all individuals for which the property concerned has at least one value *semantically*

equal to V (it may have other values as well).

> NOTE: for datatypes "semantically equal" means that the lexical representation of the literals maps to the same value. For individuals it means that they either have the same URI reference or are defined as being the same individual (see owl:sameAs).

> NOTE: the value constraint `owl:hasValue` is not included in OWL Lite.

The following example describes the class of individuals who have the individual referred to as `Clinton` as their parent:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:hasValue rdf:resource="#Clinton" />
</owl:Restriction>
```

3.1.2.2 Cardinality constraints

In OWL, like in RDF, it is assumed that any instance of a class may have an arbitrary number (zero or more) of values for a particular property. To make a property required (at least one), to allow only a specific number of values for that property, or to insist that a property must not occur, cardinality constraints can be used. OWL provides three constructs for restricting the cardinality of properties locally within a class context.

> NOTE: OWL Lite includes the use of all three types of cardinality constraints, but only when used with the values "0" or "1".

3.1.2.2.1 owl:maxCardinality

The cardinality constraint owl:maxCardinality is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype `nonNegativeInteger`. A restriction containing an `owl:maxCardinality` constraint describes a class of all individuals that have *at most* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding `rdf:datatype` attribute.

The following example describes a class of individuals that have at most two

parents:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality rdf:datatype="&xsd;
nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>
```

RDF datatyping is discussed in more detail in Sec. 6.

3.1.2.2.2 owl:minCardinality

The cardinality constraint owl:minCardinality is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype `nonNegativeInteger`. A restriction containing an `owl:minCardinality` constraint describes a class of all individuals that have *at least* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding `rdf:datatype` attribute.

The following example describes a class of individuals that have at least two parents:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:minCardinality rdf:datatype="&xsd;
nonNegativeInteger">2</owl:minCardinality>
</owl:Restriction>
```

Note that an `owl:minCardinality` of one or more means that all instances of the class must have a value for the property.

3.1.2.2.3 owl:cardinality

The cardinality constraint owl:cardinality is a built-in OWL property that links a restriction class to a data value belonging to the range of the XML Schema datatype `nonNegativeInteger`. A restriction containing an `owl:cardinality` constraint describes a class of all individuals that have *exactly* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. Syntactically, the

cardinality constraint is represented as an RDF property element with the corresponding `rdf:datatype` attribute.

This construct is in fact redundant as it can always be replaced by a pair of matching <u>owl:minCardinality</u> and <u>owl:maxCardinality</u> constraints with the same value. It is included as a convenient shorthand for the user.

The following example describes a class of individuals that have exactly two parents:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:cardinality rdf:datatype="&xsd;
nonNegativeInteger">2</owl:cardinality>
</owl:Restriction>
```

3.1.3 Intersection, union and complement

The three types of class descriptions in this section represent the more advanced class constructors that are used in Description Logic. They can be viewed as representing the AND, OR and NOT operators on classes. The three operators get the standard set-operator names: intersection, union and complement. These language constructs also share the characteristic that they contain nested class descriptions, either one (complement) or more (union, intersection).

3.1.3.1 owl:intersectionOf

The <u>owl:intersectionOf</u> property links a class to a list of <u>class descriptions.</u> An `owl:intersectionOf` statement describes a class for which the class extension contains precisely those individuals that are members of the class extension of all class descriptions in the list.

An example:

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
```

```
      </owl:Class>
      <owl:Class>
        <owl:oneOf rdf:parseType="Collection">
          <owl:Thing rdf:about="#Turandot" />
          <owl:Thing rdf:about="#Tosca" />
        </owl:oneOf>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>
```

In this example the value of `owl:intersectionOf` is a list of two class descriptions, namely two enumerations, both describing a class with two individuals. The resulting intersection is a class with one individual, namely `Tosca`. as this is the only individual that is common to both enumerations.

> NOTE: This assumes that the three individuals are all different. In fact, this is not by definition true in OWL. Different URI references may refer to the same individuals, because OWL does not have a "unique names" assumption. In Sec. 5 one can find OWL language constructs for making constraints about equality and difference of individuals.

> NOTE: In this example we use enumerations to make clear what the meaning is of this language construct. See the OWL Guide [*OWL Guide*] for more typical examples.

> NOTE: OWL Lite is restricted in its use of `owl:intersectionOf`. This is discussed later in this document, see Sec. 3.2.3

`owl:intersectionOf` can be viewed as being analogous to logical conjunction.

3.1.3.2 owl:unionOf

The owl:unionOf property links a class to a list of class descriptions. An `owl:unionOf` statement describes an anonymous class for which the class extension contains those individuals that occur in at least one of the class extensions of the class descriptions in the list.

An example:

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Turandot" />
        <owl:Thing rdf:about="#Tosca" />
      </owl:oneOf>
    </owl:Class>
  </owl:unionOf>
</owl:Class>
```

This class description describes a class for which the class extension contains three individuals, namely `Tosca`, `Salome`, and `Turandot` (assuming they are all different).

> NOTE: `owl:unionOf` is not part of OWL Lite.

`owl:unionOf` is analogous to logical disjunction.

3.1.3.3 owl:complementOf

An owl:complementOf property links a class to precisely one class description. An `owl:complementOf` statement describes a class for which the class extension contains exactly those individuals that do *not* belong to the class extension of the class description that is the object of the statement. `owl:complementOf` is analogous to logical negation: the class extension consists of those individuals that are NOT members of the class extension of the complement class.

As an example of the use of complement, the expression "not meat" could be written as:

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

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

The extension of this class description contains all individuals that do not belong to the class `Meat`.

> NOTE: `owl:complementOf` is not part of OWL Lite.

3.2 Class axioms

Class descriptions form the building blocks for defining classes through class axioms. The simplest form of a class axiom is a class description of type 1, It just states the existence of a class, using `owl:Class` with a class identifier.

For example, the following class axiom declares the URI reference `#Human` to be the name of an OWL class:

```
<owl:Class rdf:ID="Human"/>
```

This is correct OWL, but does not tell us very much about the class `Human`. Class axioms typically contain additional components that state necessary and/ or sufficient characteristics of a class. OWL contains three language constructs for combining class descriptions into class axioms:

> [rdfs:subClassOf](#) allows one to say that the class extension of a class description is a subset of the class extension of another class description. [owl:equivalentClass](#) allows one to say that a class description has exactly the same class extension as another class description. [owl:disjointWith](#) allows one to say that the class extension of a class description has no members in common with the class extension of another class description.

Syntactically, these three language constructs are properties that have a class description as both domain and range. We discuss these properties in more detail in the following subsections.

In addition, OWL allows class axioms in which a class description of the enumeration or the set-operator type is given a name. These class axioms are semantically equivalent to class axioms with a `owl:equivalentClass` statement, so these will be discussed right after that subsection (see Sec. 3.2.3 "Axioms for complete classes without using owl:equivalentClass").

3.2.1 rdfs:subClassOf

> AXIOM SCHEMA: *class description* `rdfs:subClassOf` *class description*

The `rdfs:subClassOf` construct is defined as part of RDF Schema [*RDF Vocabulary*]. Its meaning in OWL is exactly the same: if the class description C1 is defined as a subclass of class description C2, then the set of individuals in the class extension of C1 should be a subset of the set of individuals in the class extension of C2. A class is by definition a subclass of itself (as the subset may be the complete set).

An example:

```
<owl:Class rdf:ID="Opera">
  <rdfs:subClassOf rdf:resource="#MusicalWork" />
</owl:Class>
```

This class axiom declares a subclass relation between two OWL classes that are described through their names (`Opera` and `MusicalWork`). Subclass relations provide necessary conditions for belonging to a class. In this case, to be an opera the individual also needs to be a musical work.

> NOTE: In OWL Lite the subject of an `rdfs:subClassOf` statement must be a class identifier. The object must be either a class identifier or a property restriction.

For any class there may be any number of subClassOf axioms. For example, we could add the following axiom about the class `Opera`:

```
<owl:Class rdf:about="#Opera">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasLibrettist" />
      <owl:minCardinality rdf:datatype="&xsd;
nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

This class axiom contains a property restriction. The example states that `Opera` is a subclass of an anonymous OWL class (remember: `owl:Restriction` is a

subclass of `owl:Class`) that has as its class extension the set of all individuals for which the property `hasLibrettist` has at least one value. Thus, operas should have at least one librettist.

Class axioms can get more complex when class descriptions are nested. For example, property restrictions with an `owl:allValuesFrom` or `owl:someValuesFrom` statement may point to any class description. Consider the following example:

```
<owl:Class rdf:ID="TraditionalItalianOpera">
  <rdfs:subClassOf rdf:resource="#Opera"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasOperaType"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:oneOf rdf:parseType="Collection">
            <owl:Thing rdf:about="#OperaSeria"/>
            <owl:Thing rdf:about="#OperaBuffa"/>
          </owl:oneOf>
        </owl:Class>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

This example shows the use of the `owl:oneOf` construct. The class axiom defines traditional Italian opera as a subclass of a class of operas that have as opera type either *opera seria* or *opera buffa* (without an additional cardinality constraint, it could actually have both values).

More examples can be found in the Guide document [*OWL Guide*]. Subclass axioms provide us with partial definitions: they represent necessary but not sufficient conditions for establishing class membership of an individual. In the next subsection we will see that for defining necessary *and* sufficient conditions OWL provides the `owl:equivalentClass` construct. As a stepping stone to such axioms, consider the following example:

```
<owl:Class rdf:ID="Operetta">
  <rdfs:subClassOf rdf:resource="#MusicalWork"/>
  <rdfs:subClassOf>
    <owl:Restriction>
```

```
            <owl:onProperty rdf:resource="#hasLibrettist" />
            <owl:minCardinality rdf:datatype="&xsd;
nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Class>
            <owl:complementOf rdf:resource="#Opera"/>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Class>
```

This class axiom states that an operetta is a musical work, that has a at least one librettist and is not an opera. The use of the subclass relation leaves open the possibility that there are other musical works that have a librettist and are not operas. If we had wanted to say that operetta's are *exactly* those musical works that have a librettist but are not operas, we would need to use the <u>owl:equivalentClass</u> construct.

3.2.2 owl:equivalentClass

> AXIOM SCHEMA: *class description* `owl:equivalentClass` *class description*

A class axiom may contain (multiple) <u>owl:equivalentClass</u> statements. `owl:equivalentClass` is a built-in property that links a class description to another class description. The meaning of such a class axiom is that the two class descriptions involved have the same class extension (i.e., both class extensions contain exactly the same set of individuals).

In its simplest form, an equivalentClass axiom states the equivalence (in terms of their class extension) of two named classes. An example:

```
<owl:Class rdf:about="#US_President">
  <equivalentClass rdf:
resource="#PrincipalResidentOfWhiteHouse"/>
</owl:Class>
```

> NOTE: The use of `owl:equivalentClass` does not imply class equality. Class equality means that the classes have the same intensional meaning (denote the same concept). In the example above, the concept of "President of the US" is related to, but not

equal to the concept of the principal resident of a certain estate. Real class equality can only be expressed with the <ins>owl:sameAs</ins> construct. As this requires treating classes as individuals, class equality can only be expressed in OWL Full.

Axioms with `owl:equivalentClass` can also be used to define an enumerated class by linking a type 1 class description (a class identifier) to a type 2 class description (an enumeration). An example:

```
<owl:Class rdf:ID="DaPonteOperaOfMozart">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <Opera rdf:about="#Nozze_di_Figaro"/>
        <Opera rdf:about="#Don_Giovanni"/>
        <Opera rdf:about="#Cosi_fan_tutte"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

This class axiom defines the class of operas that together represent the "Da Ponte operas of Mozart" (a popular subject in musicology). By using the equivalentClass construct we can state necessary and sufficient conditions for class membership, in this case consisting of an enumeration of three individuals, no less, no more.

> NOTE: OWL DL does not put any constraints on the types of class descriptions that can be used as subject and object of an `owl:equivalentClass` statement. In OWL Lite the subject must be a class name and the object must be either a class name or a property restriction.

> NOTE: Although in principle different types of class descriptions are allowed as the subject of an equivalentClass statement, in practice it usually is some class identifier. This is also true for the examples in this section.

It is possible to have multiple equivalentClass axioms about the same class. However, this requires care. Both axioms must lead to the same outcome, i.e. exactly the same class extension. For example, an alternate equivalentClass axiom for Mozart's "Da Ponte operas" could be the following one:

```
<owl:Class rdf:about="#DaPonteOperaOfMozart">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasComposer"/>
          <owl:hasValue rdf:
resource="#Wolfgang_Amadeus_Mozart"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:
resource="#hasLibrettist"/>
          <owl:hasValue rdf:
resource="#Lorenzo_Da_Ponte"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

This states that the class extension of the Da Ponte operas of Mozart corresponds exactly to those operas which are composed by Mozart and for which the libretto is written by Da Ponte (note: intersection = "and"). This axiom indeed defines a class with exactly the same instances as the previous axiom.

> NOTE: If we wanted to "upgrade" an axiom of the form "A subClassOf B" to "A equivalentClass B" (meaning that the class extension of A is not just any subset, but in fact the same set as the class extension of B), we could add a second subClassOf axiom of the form (B subClassOf A), which by definition makes the two class extensions equivalent (and thus has the same meaning as "A equivalentClass B"). Such subClassOf "cycles" are explicitly allowed. As OWL is usable in a distributed environment, this can be a useful feature.

3.2.3 Axioms for complete classes without using owl:equivalentClass

> AXIOM SCHEMA: named *class description* of type 2 (with `owl:oneOf`) or type 4-6 (with `owl:intersectionOf, owl:unionOf` or `owl:complementOf`

OWL allows users to define class axioms by giving a name to class descriptions

of the enumeration or set-operator type. Such a class axiom defines necessary and sufficient conditions for establishing class membership. An example:

```
<owl:Class rdf:ID="DaPonteOperaOfMozart">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Nozze_di_Figaro"/>
    <owl:Thing rdf:about="#Don_Giovanni"/>
    <owl:Thing rdf:about="#Cosi_fan_tutte"/>
  </owl:oneOf>
</owl:Class>
```

This class axiom should be interpreted as follows: the class extension of the class `DaPonteOperaOfMozart` is exactly defined by the enumeration.

This class axiom is semantically equivalent to the first opera example in the previous section, which included an additional `owl:equivalentClass` statement. Axioms of this type can also be constructed with `owl:intersectionOf, owl:unionOf` and `owl:complementOf`. An example with a union could be:

```
<owl:Class rdf:ID="LivingBeing">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Plant"/>
    <owl:Class rdf:about="#Animal"/>
  </owl:unionOf>
</owl:Class>
```

This class axiom states that the class extension of `LivingBeing` exactly corresponds to the union of the class extensions of `Plant` and `Animal`.

> NOTE: OWL Lite only includes class axioms of this type which are constructed with the `owl:intersectionOf` property. The values of the intersectionOf list must be class identifiers and/or property restrictions. Thus, "complete class" axioms using enumeration, complement and union are not allowed in OWL Lite.

3.2.4 owl:disjointWith

> AXIOM SCHEMA: *class description* `owl:disjointWith` *class description*

A class axiom may also contain (multiple) owl:disjointWith statements. `owl:`

disjointWith is a built-in OWL property with a class description as domain and range. Each `owl:disjointWith` statement asserts that the class extensions of the two class descriptions involved have no individuals in common. Like axioms with `rdfs:subClassOf`, declaring two classes to be disjoint is a partial definition: it imposes a necessary but not sufficient condition on the class.

This is a popular example of class disjointness:

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

Whether this is actually true is a matter for biologists to decide. The following example shows a common use of class disjointness in subclass hierarchies:

```
<owl:Class rdf:about="#MusicDrama">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Opera"/>
        <owl:Class rdf:about="#Operetta"/>
        <owl:Class rdf:about="#Musical"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="#Opera">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
</owl:Class>

<owl:Class rdf:about="#Operetta">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
  <owl:disjointWith rdf:resource="#Opera"/>
</owl:Class>

<owl:Class rdf:about="#Musical">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
  <owl:disjointWith rdf:resource="#Opera"/>
  <owl:disjointWith rdf:resource="#Operetta"/>
</owl:Class>
```

Here, owl:disjointWith statements are used together with owl:unionOf in order to define a set of mutually disjoint and complete subclasses of a superclass. In natural language: every MusicDrama is either an opera, an Operetta, or a Musical (the subclass partitioning is complete) and individuals belonging to one subclass, *e.g.*, Opera, cannot belong to another subclass, *e.g.*, Musical (disjoint or non-overlapping subclasses). This is a common modelling notion used in many data-modelling notations.

> NOTE: OWL Lite does not allow the use of owl:disjointWith.

4. Properties

OWL distinguishes between two main categories of properties that an ontology builder may want to define:

> *Object properties* link individuals to individuals.
> *Datatype properties* link individuals to data values.

> NOTE: OWL also has the notion of annotation properties (owl:AnnotationProperty) and ontology properties (owl:OntologyProperty). These are needed in OWL DL for semantic reasons. See Sec. 7 and the OWL Semantics and Abstract Syntax document [*OWL S&AS*].

An object property is defined as an instance of the built-in OWL class owl:ObjectProperty. A datatype property is defined as an instance of the built-in OWL class owl:DatatypeProperty. Both owl:ObjectProperty and owl:DatatypeProperty are subclasses of the RDF class rdf:Property (see Appendix B).

> NOTE: In OWL Full, object properties and datatype properties are not disjoint. Because data values can be treated as individuals, datatype properties are effectively subclasses of object properties. In OWL Full owl:ObjectProperty is equivalent to rdf:Property In practice, this mainly has consequences for the use of owl:InverseFunctionalProperty. See also the OWL Full characterization in Sec. 8.1.

A property axiom defines characteristics of a property. In its simplest form, a property axiom just defines the existence of a property. For example:

```
<owl:ObjectProperty rdf:ID="hasParent"/>
```

This defines a property with the restriction that its values should be individuals.

Often, property axioms define additional characteristics of properties. OWL supports the following constructs for property axioms:

> RDF Schema constructs: `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`
> relations to other properties: `owl:equivalentProperty` and `owl:inverseOf`
> global cardinality constraints: `owl:FunctionalProperty` and `owl:InverseFunctionalProperty`
> logical property characteristics: `owl:SymmetricProperty` and `owl:TransitiveProperty`

In the next subsections, the various types of property axioms are discussed in more detail.

> NOTE: In this section we use the term "property extension" in a similar fashion to "class extension". The property extension is the set of instances that is associated with the property. Instances of properties are not single elements, but subject-object pairs of property statements. In relational database terms, property instances would be called "tuples" of a binary relation (the property).

4.1 RDF Schema constructs

The constructs in this section are discussed in detail in the RDF Schema document [*RDF Vocabulary*]. The description in this section provides a synopsis of these constructs and provides some OWL-specific aspects and examples.

4.1.1 rdfs:subPropertyOf

A `rdfs:subPropertyOf` axiom defines that the property is a subproperty of some other property. Formally this means that if P1 is a subproperty of P2, then the property extension of P1 (a set of pairs) should be a subset of the property extension of P2 (also a set of pairs).

An example:

```
<owl:ObjectProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

This states that all instances (pairs) contained in the property extension of the property "hasMother" are also members of the property extension of the property "hasParent".

Subproperty axioms can be applied to both datatype properties and object properties.

> NOTE: In OWL DL the subject and object of a subproperty statement must be either both datatype properties or both object properties.

4.1.2 rdfs:domain

For a property one can define (multiple) `rdfs:domain` axioms. Syntactically, `rdfs:domain` is a built-in property that links a property (some instance of the class `rdf:Property`) to a [class description](). An `rdfs:domain` axiom asserts that the subjects of such property statements must belong to the class extension of the indicated class description.

Multiple `rdfs:domain` axioms are allowed and should be interpreted as a conjunction: these restrict the domain of the property to those individuals that belong to the *intersection* of the class descriptions. If one would want to say that multiple classes can act as domain, one should use a class description of the [owl:unionOf]() form. For example, if we want to say that the domain of the property `hasBankAccount` can be either a `Person` or a `Corporation`, we would need to say something like this:

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Corporation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

> NOTE: In OWL Lite the value of `rdfs:domain` must be a class

identifier.

4.1.3 rdfs:range

For a property one can define (multiple) `rdfs:range` axioms. Syntactically, `rdfs:range` is a built-in property that links a property (some instance of the class `rdf:Property`) to to either a [class description](#) or a [data range](#). An `rdfs:range` axiom asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range.

Multiple range restrictions are interpreted as stating that the range of the property is the *intersection* of all ranges (i.e., the intersection of the class extension of the class descriptions c.q. the intersection of the data ranges). Similar to [rdfs:domain](#), multiple alternative ranges can be specified by using a class description of the [owl:unionOf](#) form (see the previous subsection).

Note that, unlike any of the [value constraints](#) described in the section on class descriptions, `rdfs:range` restrictions are global. Value constraints such as [owl:allValuesFrom](#) are used in a class description and are only enforced on the property when applied to that class. In contrast, `rdfs:range` restrictions apply to the property irrespective of the class to which it is applied. Thus, `rdfs:range` should be used with care.

> NOTE: In OWL Lite the only type of class descriptions allowed as objects of `rdfs:range` are class names.

4.2 Relations to other properties

4.2.1 owl:equivalentProperty

The [owl:equivalentProperty](#) construct can be used to state that two properties have the same property extension. Syntactically, `owl:equivalentProperty` is a built-in OWL property with `rdf:Property` as both its domain and range.

> NOTE: Property equivalence is not the same as property equality. Equivalent properties have the same "values" (i.e., the same property extension), but may have different intensional meaning (i.e., denote different concepts). Property equality should be expressed with the [owl:sameAs](#) construct. As this requires that properties are treated as individuals, such axioms are only allowed in OWL Full.

4.2.2 owl:inverseOf

Properties have a direction, from domain to range. In practice, people often find it useful to define relations in both directions: persons own cars, cars are owned by persons. The owl:inverseOf construct can be used to define such an inverse relation between properties.

Syntactically, `owl:inverseOf` is a built-in OWL property with `owl:ObjectProperty` as its domain and range. An axiom of the form `P1 owl:inverseOf P2` asserts that for every pair (x,y) in the property extension of P1, there is a pair (y,x) in the property extension of P2, and vice versa. Thus, `owl:inverseOf` is a symmetric property.

An example:

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

4.3 Global cardinality constraints on properties

4.3.1 owl:FunctionalProperty

A functional property is a property that can have only one (unique) value y for each instance x, i.e. there cannot be two distinct values y1 and y2 such that the pairs (x,y1) and (x,y2) are both instances of this property. Both object properties and datatype properties can be declared as "functional". For this purpose, OWL defines the built-in class owl:FunctionalProperty as a special subclass of the RDF class `rdf:Property`.

The following axiom states that the `husband` property is functional, *i.e.,* a woman can have at most one husband (a good example of culture dependence of ontologies):

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

As always, there are syntactic variations. The example above is semantically equivalent to the one below:

```
<owl:ObjectProperty rdf:ID="husband">
  <rdfs:domain rdf:resource="#Woman" />
  <rdfs:range  rdf:resource="#Man" />
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:about="#husband" />
```

4.3.2 owl:InverseFunctionalProperty

If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual). More formally, if we state that P is an `owl:InverseFunctionalProperty`, then this asserts that a value y can only be the value of P for a single instance x, i.e. there cannot be two distinct instances x1 and x2 such that both pairs (x1,y) and (x2,y) are instances of P.

Syntactically, an inverse-functional property axiom is specified by declaring the property to be an instance of the built-in OWL class owl: InverseFunctionalProperty, which is a subclass of the OWL class `owl: ObjectProperty`.

> NOTE: Because in OWL Full datatype properties are a subclass of object properties, an inverse-functional property can be defined for datatype properties. In OWL DL object properties and datatype properties are disjoint, so an inverse-functional property cannot be defined for datatype properties. See also Sec. 8.1 and Sec. 8.2.

A typical example of an inverse-functional property:

```
<owl:InverseFunctionalProperty rdf:
ID="biologicalMotherOf">
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:InverseFunctionalProperty>
```

This example states that for each object of `biologicalMotherOf` statements (some human) one should be able to uniquely identify a subject (some woman). Inverse-functional properties resemble the notion of a key in databases.

One difference with functional properties is that for inverse-functional properties no additional object-property or datatype-property axiom is required: inverse-functional properties are by definition object properties.

Notice that <u>owl:FunctionalProperty</u> and <u>owl:InverseFunctionalProperty</u> specify global cardinality constraints. That is, no matter which class the property is applied to, the cardinality constraints must hold. This is different from the cardinality constraints contained in <u>property restrictions</u>. The latter are class descriptions and are only enforced on the property when applied to that class.

4.4 Logical characteristics of properties

4.4.1 owl:TransitiveProperty

When one defines a property P to be a transitive property, this means that if a pair (x,y) is an instance of P, and the pair (y,z) is also instance of P, then we can infer the the pair (x,z) is also an instance of P.

Syntactically, a property is defined as being transitive by making it an instance of the the built-in OWL class <u>owl:TransitiveProperty</u>, which is defined as a subclass of <u>owl:ObjectProperty</u>.

Typical examples of transitive properties are properties representing certain part-whole relations. For example, we might want to say that the `subRegionOf` property between regions is transitive:

```
<owl:TransitiveProperty rdf:ID="subRegionOf">
  <rdfs:domain rdf:resource="#Region"/>
  <rdfs:range  rdf:resource="#Region"/>
</owl:TransitiveProperty>
```

From this an OWL reasoner should be able to derive that if `ChiantiClassico`, `Tuscany` and `Italy` are regions, and `ChiantiClassico` is a subregion of `Tuscany`, and `Tuscany` is a subregion of `Italy`, then `ChiantiClassico` is also a subregion of `Italy`.

Note that because `owl:TransitiveProperty` is a subclass of `owl:ObjectProperty`, the following syntactic variant is equivalent to the example above:

```
<owl:ObjectProperty rdf:ID="subRegionOf">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Region"/>
  <rdfs:range  rdf:resource="#Region"/>
</owl:ObjectProperty>
```

> NOTE: OWL DL requires that for a transitive property no local or global cardinality constraints should be declared on the property itself or its superproperties, nor on the inverse of the property or its superproperties.

4.4.2 owl:SymmetricProperty

A symmetric property is a property for which holds that if the pair (x,y) is an instance of P, then the pair (y,x) is also an instance of P. Syntactically, a property is defined as symmetric by making it an instance of the built-in OWL class owl:SymmetricProperty, a subclass of owl:ObjectProperty.

A popular example of a symmetric property is the friendOf relation:

```
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range  rdf:resource="#Human"/>
</owl:SymmetricProperty>
```

The domain and range of a symmetric property are the same.

5. Individuals

Individuals are defined with individual axioms (also called "facts"). We discuss two types of facts:

1. Facts about class membership and property values of individuals
2. Facts about individual identity

5.1 Class membership and property values

Many facts typically are statements indicating class membership of individuals and property values of individuals. As an example, consider the following set of statements about an instance of the class Opera:

```
<Opera rdf:ID="Tosca">
  <hasComposer rdf:resource="#Giacomo_Puccini"/>
  <hasLibrettist rdf:resource="#Victorien_Sardou"/>
  <hasLibrettist rdf:resource="#Giuseppe_Giacosa"/>
  <hasLibrettist rdf:resource="#Luigi_Illica"/>
  <premiereDate rdf:datatype="&xsd;date">1900-01-14</premiereDate>
  <premierePlace rdf:resource="#Roma"/>
  <numberOfActs rdf:datatype="&xsd;positiveInteger">3</numberOfActs>
</Opera>
```

This example includes a number of facts about the individual `Tosca`, an instance of the class `Opera`. `Tosca` is composed by Giacomo Puccini. The opera has three libretto writers. The property `premiereDate` links an opera to a typed literal with the XML Schema datatype `date`. The XML schema document on datatypes [*XML Schema Datatypes*] contains the relevant information about syntax and semantics of this datatype.

Individual axioms need not necessarily be about named individuals: they can also refer to anonymous individuals. As an example, consider the piece of RDF/XML below. The example defines some facts about an anonymous instance of the class `Measurement`, a quantitative observation for which facts such as the subject under observation, the observed phenomenon, the observed value, and the observation time are listed:

```
<Measurement>
  <observedSubject rdf:resource="#JaneDoe"/>
  <observedPhenomenon rdf:resource="#Weight"/>
  <observedValue>
    <Quantity>
      <quantityValue rdf:datatype="&xsd;float">59.5</quantityValue>
      <quantityUnit rdf:resource="#Kilogram"/>
    </Quantity>
  </observedValue>
  <timeStamp rdf:datatype="&xsd;dateTime">2003-01-24T09:00:08+01:00</timeStamp>
</Measurement>
```

This individual axiom contains two anonymous individuals, namely some `Measurement` and some `Quantity`. In natural language, for the subject Jane

Doe the measured value of the phenomenon `Weight` is some quantity, which has a value of 59.5 using the unit of kilogram. The time of measurement is January 24, 2003, eight seconds past nine in the morning, in the time zone UTC +1 (winter time in Amsterdam, Berlin, Paris). As before, `float` and `dateTime` are XML Schema datatypes, the syntactic and semantic details of which can be found in the relevant XML Schema documentation [*XML Schema Datatypes*].

5.2 Individual identity

Many languages have a so-called "unique names" assumption: different names refer to different things in the world. On the web, such an assumption is not possible. For example, the same person could be referred to in many different ways (i.e. with different URI references). For this reason OWL does not make this assumption. Unless an explicit statement is being made that two URI references refer to the same or to different individuals, OWL tools should in principle assume either situation is possible.

OWL provides three constructs for stating facts about the identity of individuals:

> `owl:sameAs` is used to state that two URI references refer to the same individual.
> `owl:differentFrom` is used to state that two URI references refer to different individuals
> `owl:AllDifferent` provides an idiom for stating that a list of individuals are all different.

5.2.1 owl:sameAs

The built-in OWL property owl:sameAs links an individual to an individual. Such an `owl:sameAs` statement indicates that two URI references actually refer to the same thing: the individuals have the same "identity".

For individuals such as "people" this notion is relatively easy to understand. For example, we could state that the following two URI references actually refer to the same person:

```
<rdf:Description rdf:
about="#William_Jefferson_Clinton">
  <owl:sameAs rdf:resource="#BillClinton"/>
</rdf:Description>
```

The `owl:sameAs` statements are often used in defining mappings between ontologies. It is unrealistic to assume everybody will use the same name to refer to individuals. That would require some grand design, which is contrary to the spirit of the web.

In OWL Full, where a class can be treated as instances of (meta)classes, we can use the `owl:sameAs` construct to define class equality, thus indicating that two concepts have the same intensional meaning. An example:

```
<owl:Class rdf:ID="FootballTeam">
  <owl:sameAs rdf:resource="http://sports.org/
US#SoccerTeam"/>
</owl:Class>
```

One could imagine this axiom to be part of a European sports ontology. The two classes are treated here as individuals, in this case as instances of the class `owl:Class`. This allows us to state that the class `FootballTeam` in some European sports ontology denotes the same concept as the class `SoccerTeam` in some American sports ontology. Note the difference with the statement:

```
<footballTeam owl:equivalentClass us:soccerTeam />
```

which states that the two classes have the same class extension, but are not (necessarily) the same concepts.

> NOTE: For details of comparison of URI references, see the section on RDF URI references in the RDF Concepts document [*RDF Concepts*].

5.2.2 owl:differentFrom

The built-in OWL owl:differentFrom property links an individual to an individual. An `owl:differentFrom` statement indicates that two URI references refer to different individuals.

An example:

```
<Opera rdf:ID="Don_Giovanni"/>

<Opera rdf:ID="Nozze_di_Figaro">
  <owl:differentFrom rdf:resource="#Don_Giovanni"/>
```

```
    </Opera>

    <Opera rdf:ID="Cosi_fan_tutte">
      <owl:differentFrom rdf:resource="#Don_Giovanni"/>
      <owl:differentFrom rdf:resource="#Nozze_di_Figaro"/>
    </Opera>
```

This states that there are three different operas.

5.2.3 owl:AllDifferent

For ontologies in which the unique-names assumption holds, the use of `owl:differentFrom` is likely to lead to a large number of statements, as all individuals have to be declared pairwise disjoint. For such situations OWL provides a special idiom in the form of the construct [owl:AllDifferent](). `owl:AllDifferent` is a special built-in OWL class, for which the property [owl:distinctMembers]() is defined, which links an instance of `owl:AllDifferent` to a list of individuals. The intended meaning of such a statement is that all individuals in the list are all different from each other.

An example:

```
    <owl:AllDifferent>
      <owl:distinctMembers rdf:parseType="Collection">
        <Opera rdf:about="#Don_Giovanni"/>
        <Opera rdf:about="#Nozze_di_Figaro"/>
        <Opera rdf:about="#Cosi_fan_tutte"/>
        <Opera rdf:about="#Tosca"/>
        <Opera rdf:about="#Turandot"/>
        <Opera rdf:about="#Salome"/>
      </owl:distinctMembers>
    </owl:AllDifferent>
```

This states that these six URI references all point to different operas.

> NOTE: `owl:distinctMembers` is a special syntactical construct added for convenience and should always be used with an `owl:AllDifferent` individual as its subject.

6. Datatypes

In a number of places in this document we have seen the notion of a data range for specifying a range of data values. OWL allows three types of data range specifications:

> A RDF datatype specification.
> The RDFS class rdfs:Literal.
> An enumerated datatype, using the `owl:oneOf` construct.

The minimal level of tool support for datatypes is discussed in Sec. 6.3.

6.1 RDF Datatypes

OWL makes use of the RDF datatyping scheme, which provides a mechanism for referring to XML Schema datatypes [*XML Schema Datatypes*]. For a detailed description the reader is referred to the RDF documents, *e.g.*, [*RDF Concepts*]. For the convenience of the reader, we provide here a synopsis of the use of RDF datatypes.

Data values are instances of the RDF Schema class `rdfs:Literal`. Literals can be either plain (no datatype) or typed. Datatypes are instances of the class `rdfs:Datatype`. In RDF/XML, the type of a literal is specified by an `rdf:datatype` attribute of which the value is recommended to be one of the following:

> A canonical URI reference to an XML Schema datatype of the form:

> `http://www.w3.org/2001/XMLSchema#NAME`

> where "NAME" should be the name of a *simple* XML Schema *built-in* datatype, as defined in Section 3 of [*XML Schema Datatypes*], with the provisos specified below.
> The URI reference of the datatype `rdf:XMLLiteral`. This datatype is used to include XML content into an RDF/OWL document. The URI reference of this datatype is:

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

> For details about this datatype, see the RDF Concepts document [*RDF*

Concepts].

The RDF Semantics document [*RDF Semantics*, Section 5]. recommends use of the following simple built-in XML Schema datatypes.

The primitive datatype xsd:string, plus the following datatypes derived from xsd:string: xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name,and xsd:NCName.

The primitive datatype xsd:boolean.

The primitive numerical datatypes xsd:decimal, xsd:float, and xsd:double, plus all derived types of xsd:decimal (xsd:integer, xsd:positiveInteger. xsd:nonPositiveInteger, xsd:negativeInteger, xsd:nonNegativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte)

The primitive time-related datatypes: xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, and xsd:gMonth.

The primitive datatypes xsd:hexBinary, xsd:base64Binary, and xsd:anyURI.

NOTE: It is not illegal, although not recommended, for applications to define their own datatypes by defining an instance of rdfs:Datatype. Such datatypes are "unrecognized", but are treated in a similar fashion as "unsupported datatypes" (see Sec. 6.3 for details about how these should be treated by OWL tools).

When using datatypes, please note that even if a property is defined to have a range of a certain datatype, RDF/XML still requires that the datatype be specified each time the property is used. An example could be the declaration of a property that we used earlier in the Measurement example:

```
<owl:DatatypeProperty rdf:about="#timeStamp">
  <rdfs:domain rdf:resource="#Measurement"/>
  <rdf:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<Measurement>
  <timeStamp rdf:datatype="&xsd;dateTime">2003-01-
24T09:00:08+01:00</timeStamp>
```

```
</Measurement>
```

6.2 Enumerated datatype

In addition to the RDF datatypes, OWL provides one additional construct for defining a range of data values, namely an enumerated datatype. This datatype format makes use of the `owl:oneOf` construct, that is also used for describing an [enumerated class](#). In the case of an enumerated datatype, the subject of `owl:oneOf` is a blank node of class `owl:DataRange` and the object is a list of literals. Unfortunately, we cannot use the `rdf:parseType="Collection"` idiom for specifying the literal list, because RDF requires the collection to be a list of RDF node elements. Therefore we have to specify the list of data values with the basic list constructs `rdf:first`, `rdf:rest` and `rdf:nil`.

> NOTE: Enumerated datatypes are not part of OWL Lite.

The example below specifies the range of the property `tennisGameScore` to be the list of integer values {0, 15, 30, 40}:.

```
<owl:DatatypeProperty rdf:ID="tennisGameScore">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
            <rdf:first rdf:datatype="&xsd;integer">0</rdf:first>
            <rdf:rest>
              <rdf:List>
                <rdf:first rdf:datatype="&xsd;integer">15</rdf:first>
                <rdf:rest>
                  <rdf:List>
                    <rdf:first rdf:datatype="&xsd;integer">30</rdf:first>
                    <rdf:rest>
                      <rdf:List>
                        <rdf:first rdf:datatype="&xsd;integer">40</rdf:first>
                        <rdf:rest rdf:resource="&rdf;nil" />
                      </rdf:List>
                    </rdf:rest>
```

```
                         </rdf:List>
                      </rdf:rest>
                   </rdf:List>
                </rdf:rest>
             </rdf:List>
          </owl:oneOf>
       </owl:DataRange>
    </rdfs:range>
</owl:DatatypeProperty>
```

6.3 Support for datatype reasoning

Tools may vary in terms of support for datatype reasoning. As a minimum, tools must support datatype reasoning for the XML Schema datatypes `xsd:string` and `xsd:integer`. OWL Full tools must also support `rdf:XMLLiteral`. For unsupported datatypes, lexically identical literals should be considered equal, whereas lexically different literals would not be known to be either equal or unequal. Unrecognized datatypes should be treated in the same way as unsupported datatypes.

7. Annotations, ontology header, imports and version information

7.1 Annotations

OWL Full does not put any constraints on annotations in an ontology. OWL DL allows annotations on classes, properties, individuals and ontology headers, but only under the following conditions:

The sets of object properties, datatype properties, annotation properties and ontology properties must be mutually disjoint. Thus, in OWL DL `dc:creator` cannot be at the same time a datatype property and an annotation property.
Annotation properties must have an explicit typing triple of the form:

```
        AnnotationPropertyID rdf:type owl:
    AnnotationProperty .
```

Annotation properties must not be used in property axioms. Thus, in OWL DL one cannot define subproperties or domain/range constraints for

annotation properties.
The object of an annotation property must be either a data literal, a URI reference, or an individual.

Five annotation properties are predefined by OWL, namely:

```
owl:versionInfo
rdfs:label
rdfs:comment
rdfs:seeAlso
rdfs:isDefinedBy
```

Here is an example of legal use of an annotation property in OWL DL:

```
<owl:AnnotationProperty rdf:about="&dc;creator"/>

<owl:Class rdf:about="#MusicalWork">
  <rdfs:label>Musical work</rdfs:label>
  <dc:creator>N.N.</dc:creator>
</owl:Class>
```

The example assumes `&dc;` and `dc:` point respectively to the Dublin Core URI and namespace. Thus, using Dublin Core properties as annotation properties in OWL DL requires an explicit typing triple. This ensures annotations are handled in a semantically correct fashion by OWL DL reasoners (see the OWL Semantics and Abstract Syntax document [*OWL S&AS*] for details).

Once we define `dc:creator` as an annotation property, OWL DL does NOT allow property axioms such as the range constraint below:

```
<-- This is illegal in OWL DL -->

<owl:AnnotationProperty rdf:about="&dc;creator">
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:AnnotationProperty>
```

Note that one can still specify the value type of a literal in an annotation-property statement:

```
<Opera rdf:about="#Tosca">
  <dc:creator rdf:datatype="&xsd;string">Giacomo
Puccini</dc:creator>
```

```
</Opera>
```

7.2 Ontology header

A document describing an ontology typically contains information about the ontology itself. An ontology is a resource, so it may be described using properties from the OWL and other namespaces, e.g.:

```
<owl:Ontology rdf:about="">
  <owl:versionInfo> ... </owl:versionInfo>
  <rdfs:comment>...</rdfs:comment>
  <owl:imports rdf:resource="..."/>
</owl:Ontology>
```

This is commonly called the ontology header and is typically found near the beginning of the RDF/XML document. The line

```
<owl:Ontology rdf:about="">
```

states that this block describes the current ontology. More precisely, it states the current base URI identifies an instance of the class `owl:Ontology`. It is recommended that the base URI be defined using an `xml:base` attribute in the `<rdf:RDF>` element at the beginning of the document.

A sample ontology header could look like this:

```
<owl:Ontology rdf:about="">
  <owl:versionInfo>v 1.17 2003/02/26 12:56:51 mdean</
owl:versionInfo>
  <rdfs:comment>An example ontology</rdfs:comment>
  <owl:imports rdf:resource="http://www.example.org/
foo"/>
</owl:Ontology>
```

The following sections describe the various types of statements that are typically used within the header.

> NOTE: The ontology-import construct owl:imports and the ontology-versioning constructs owl:priorVersion, owl:backwardCompatibleWith and owl:incompatibleWith are defined in the OWL vocabulary as instances of the OWL built-in class `owl:OntologyProperty`.

Instances of `owl:OntologyProperty` must have the class `owl:Ontology` as their domain and range. It is permitted to define other instances of `owl:OntologyProperty`. In OWL DL for ontology properties the same constraints hold as those specified for annotation properties in [Sec. 7.1](#).

7.3 Importing an ontology

An [owl:imports](#) statement references another OWL ontology containing definitions, whose meaning is considered to be part of the meaning of the importing ontology. Each reference consists of a URI specifying from where the ontology is to be imported. Syntactically, `owl:imports` is a property with the class `owl:Ontology` as its domain and range.

The `owl:imports` statements are transitive, that is, if ontology A imports B, and B imports C, then A imports both B and C.

Importing an ontology into itself is considered a null action, so if ontology A imports B and B imports A, then they are considered to be equivalent.

Note that whether or not an OWL tool must load an imported ontology depends on the purpose of the tool. If the tool is a complete reasoner (including complete consistency checkers) then it must load all of the imported ontologies. Other tools, such as simple editors and incomplete reasoners, may choose to load only some or even none of the imported ontologies.

Although owl:imports and namespace declarations may appear redundant, they actually serve different purposes. Namespace declarations simply set up a shorthand for referring to identifiers. They do not implicitly include the meaning of documents located at the URI. On the other hand, owl:imports does not provide any shorthand notation for referring to the identifiers from the imported document. Therefore, it is common to have a corresponding namespace declaration for any ontology that is imported.

NOTE: `owl:imports` is an instance of `owl:OntologyProperty`.

7.4 Version information

7.4.1 owl:versionInfo

An [owl:versionInfo](#) statement generally has as its object a string giving

information about this version, for example RCS/CVS keywords. This statement does not contribute to the logical meaning of the ontology other than that given by the RDF(S) model theory.

Although this property is typically used to make statements about ontologies, it may be applied to any OWL construct. For example, one could attach a `owl:versionInfo` statement to an OWL class.

> NOTE: `owl:versionInfo` is an instance of `owl:AnnotationProperty`.

7.4.2 owl:priorVersion

An <u>owl:priorVersion</u> statement contains a reference to another ontology. This identifies the specified ontology as a prior version of the containing ontology. This has no meaning in the model-theoretic semantics other than that given by the RDF(S) model theory. However, it may be used by software to organize ontologies by versions.

`owl:priorVersion` is a built-in OWL property with the class `owl:Ontology` as its domain and range.

> NOTE: `owl:priorVersion` is an instance of `owl:OntologyProperty`.

7.4.3 owl:backwardCompatibleWith

An <u>owl:backwardCompatibleWith</u> statement contains a reference to another ontology. This identifies the specified ontology as a prior version of the containing ontology, and further indicates that it is backward compatible with it. In particular, this indicates that all identifiers from the previous version have the same intended interpretations in the new version. Thus, it is a hint to document authors that they can safely change their documents to commit to the new version (by simply updating namespace declarations and <u>owl:imports</u> statements to refer to the URL of the new version). If `owl:backwardCompatibleWith` is not declared for two versions, then compatibility should not be assumed.

`owl:backwardCompatibleWith` has no meaning in the model theoretic semantics other than that given by the RDF(S) model theory.

`owl:backwardCompatibleWith` is a built-in OWL property with the class `owl:Ontology` as its domain and range.

> NOTE: `owl:backwardCompatibleWith` is an instance of `owl:OntologyProperty`.

7.4.4 owl:incompatibleWith

An <u>owl:incompatibleWith</u> statement contains a reference to another ontology. This indicates that the containing ontology is a later version of the referenced ontology, but is not backward compatible with it. Essentially, this is for use by ontology authors who want to be explicit that documents cannot upgrade to use the new version without checking whether changes are required.

`owl:incompatibleWith` has no meaning in the model theoretic semantics other than that given by the RDF(S) model theory.

`owl:incompatibleWith` is a built-in OWL property with the class `owl:Ontology` as its domain and range.

> NOTE: `owl:backwardCompatibleWith` is an instance of `owl:OntologyProperty`.

7.4.5 owl:DeprecatedClass and owl:DeprecatedProperty

Deprecation is a feature commonly used in versioning software (for example, see the Java programming language) to indicate that a particular feature is preserved for backward-compatibility purposes, but may be phased out in the future. Here, a specific identifier is said to be of type <u>owl:DeprecatedClass</u> or <u>owl:DeprecatedProperty</u>, where `owl:DeprecatedClass` is a subclass of `rdfs:Class` and `owl:DeprecatedProperty` is a subclass of `rdf:Property`. By deprecating a term, it means that the term should not be used in new documents that commit to the ontology. This allows an ontology to maintain backward-compatibility while phasing out an old vocabulary (thus, it only makes sense to use deprecation in combination with backward compatibility). As a result, it it easier for old data and applications to migrate to a new version, and thus can increase the level of adoption of the new version. This has no meaning in the model theoretic semantics other than that given by the RDF(S) model theory. However, authoring tools may use it to warn users when checking OWL markup.

An example of deprecation is:

```
<owl:Ontology rdf:about="">
  <rdfs:comment>Vehicle Ontology, v. 1.1</rdfs:comment>
  <owl:backwardCompatibleWith
          rdf:resource="http://www.example.org/vehicle-
1.0"/>
  <owl:priorVersion rdf:resource="http://www.example.
org/vehicle-1.0"/>
</owl:Ontology>

<owl:DeprecatedClass rdf:ID="Car">
  <rdfs:comment>Automobile is now preferred</rdfs:
comment>
  <owl:equivalentClass rdf:resource="#Automobile"/>
  <!-- note that equivalentClass only means that the
classes have the same
        extension, so this DOES NOT lead to the
entailment that
        Automobile is of type DeprecatedClass too --
>
</owl:DeprecatedClass>

<owl:Class rdf:ID="Automobile" />

<owl:DeprecatedProperty rdf:ID="hasDriver">
  <rdfs:comment>inverse property drives is now
preferred</rdfs:comment>
  <owl:inverseOf rdf:resource="#drives" />
</owl:DeprecatedProperty>

<owl:ObjectProperty rdf:ID="drives" />
```

8. OWL Full, OWL DL and OWL Lite

In the introduction we briefly discussed the three sublanguages of OWL. In this section an informative specification is given of the differences between the three "species" of OWL. A formal account of the differences is given in the Semantics and Abstract Syntax document [*OWL S&AS*].

8.1 OWL Full

OWL Full is not actually a sublanguage. OWL Full contains all the OWL language constructs and provides free, unconstrained use of RDF constructs. In

OWL Full the resource `owl:Class` is equivalent to `rdfs:Class`. This is different from OWL DL and OWL Lite, where `owl:Class` is a proper subclass of `rdfs:Class` (meaning that not all RDF classes are OWL classes in OWL DL and OWL Lite). OWL Full also allows classes to be treated as individuals. For example, it is perfectly legal in OWL Full to have a "Fokker-100" identifier which acts both as a class name (denoting the set of Fokker-100 airplanes flying around the world) and as an individual name (*e.g.*, an instance of the class `AirplaneType`).

In OWL Full all data values are considered also to be part of the individual domain. In fact, in OWL Full the universe of individuals consists of all resources (`owl:Thing` is equivalent to `rdfs:Resource`). This means that object properties and datatype properties are not disjoint. In OWL Full `owl:ObjectProperty` is equivalent to `rdf:Property`. The consequence is that datatype properties are effectively a subclass of object properties. (Note: the fact that `owl:ObjectProperty` and `owl:DatatypeProperty` are both subclasses of `rdf:Property` is not inconsistent with this).

OWL Full will typically be useful for people who want to combine the expressivity of OWL with the flexibility and metamodelling features of RDF. However, use of the OWL Full features means that one loses some guarantees (see below) that OWL DL and OWL Lite can provide for reasoning systems.

> NOTE: RDF documents will generally be in OWL Full, unless they are specifically constructed to be in OWL DL or Lite.
>
> NOTE: Thus, in OWL Full `owl:Thing` is equivalent to `rdfs:Resource`, `owl:Class` is equivalent to `rdfs:Class`, and `owl:ObjectProperty` is equivalent to `rdf:Property`,

8.2 OWL DL

OWL DL is a sublanguage of OWL which places a number of constraints on the use of the OWL language constructs. Briefly, these constraints are the following:

> OWL DL requires a pairwise separation between classes, datatypes, datatype properties, object properties, annotation properties, ontology properties (*i.e.,* the import and versioning stuff), individuals, data values and the built-in vocabulary. This means that, for example, a class cannot be at the same time an individual.
> In OWL DL the set of object properties and datatype properties are disjoint. This implies that the following four property characteristics:

- ❍ [inverse of](#),
- ❍ [inverse functional](#),
- ❍ [symmetric](#), and
- ❍ [transitive](#)

can never be specified for [datatype properties](#)

OWL DL requires that no cardinality constraints (local nor global) can be placed on transitive properties or their inverses or any of their superproperties.

Annotations are allowed only under certain conditions. See [Sec. 7.1](#) for details.

Most RDF(S) vocabulary cannot be used within OWL DL. See the OWL Semantics and Abstract Syntax document [*OWL S&AS*] for details.

All axioms must be well-formed, with no missing or extra components, and must form a tree-like structure.

The last constraint implies that all classes and properties that one refers to are explicitly typed as OWL classes or properties, respectively. For example, if the ontology contains the following component:

```
<owl:Class rdf:ID="C1">
  <rdfs:subClassOf rdf:resource="#C2" />
</owl:Class>
```

then the ontology (or an ontology imported into this ontology) should contain a `owl:Class` triple for `C2`.

Axioms (facts) about individual equality and difference must be about named individuals.

These constraints of OWL DL may seem like an arbitrary set, but in fact they are not. The constraints are based on work in the area of reasoners for Description Logic, which require these restrictions to provide the ontology builder or user with reasoning support. In particular, the OWL DL restrictions allow the maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure can exist for an OWL reasoner.

> NOTE: [Appendix E](#) provides a set of practical guidelines for specifying OWL DL ontologies in RDF.

8.3 OWL Lite

OWL Lite abides by all the restrictions OWL DL puts on the use of the OWL language constructs. In addition, OWL Lite forbids the use of:

owl:oneOf
owl:unionOf
owl:complementOf
owl:hasValue
owl:disjointWith
owl:DataRange

OWL Lite also requires that:

the subject of `owl:equivalentClass` triples be class names and the object of `owl:equivalentClass` triples be class names or restrictions. the subject of `rdfs:subClassOf` triples be class names and the object of `rdfs:subClassOf` triples be class names or restrictions; `owl:intersectionOf` be used only on lists of length greater than one that contain only class names and restrictions;

NOTE: This is a prototypical example of legal use of `owl:intersectionOf` in OWL Lite:

```
<owl:Class rdf:ID="Woman">
  <owl:intersectionOf rdf:
parseType="Collection">
      <owl:Class rdf:about="#Female"/>
      <owl:Class rdf:about="#Human"/>
   </owl:intersectionOf>
</owl:Class/>
```

the object of `owl:allValuesFrom` and `owl:someValuesFrom` triples be class names or datatype names; the object of `rdf:type` triples be class names or restrictions; the object of `rdfs:domain` triples be class names; and the object of `rdfs:range` triples be class names or datatype names.

The idea behind the OWL Lite expressivity limitations is that they provide a minimal useful subset of language features, that are relatively straightforward for tool developers to support. The language constructs of OWL Lite provide the basics for subclass hierarchy construction: subclasses and property restrictions.

In addition, OWL Lite allows properties to be made optional or required. The limitations on OWL Lite place it in a lower complexity class than OWL DL. This can have a positive impact on the efficiency of complete reasoners for OWL Lite.

Implementations that support only the OWL Lite vocabulary, but otherwise relax the restrictions of OWL DL, cannot make certain computational claims with respect to consistency and complexity. However, such implementations may be useful in providing interoperability of OWL systems with RDFS models, databases, markup tools, or other non-reasoning tools. The Web Ontology Working Group has not provided a name for this potentially useful subset.

Appendix A. Index of all language elements

NOTE: This appendix only contains the OWL-specific constructs. For the RDF/RDFS constructs see the relevant RDF documentation, in particular the RDF Schema document [*RDF Vocabulary*].

[OWL Reference] (this document)	[*OWL Semantics*] (normative)	[*OWL Guide*] (examples)
owl:AllDifferent	owl:AllDifferent	owl:AllDifferent
owl:allValuesFrom	owl:allValuesFrom	owl:allValuesFrom
owl:AnnotationProperty	owl:AnnotationProperty	
owl: backwardCompatibleWith	owl: backwardCompatibleWith	owl: backwardCompatibleWith
owl:cardinality	owl:cardinality	owl:cardinality
owl:Class	owl:Class	owl:Class
owl:complementOf	owl:complementOf	owl:complementOf
owl:DataRange	owl:DataRange	
owl:DatatypeProperty	owl:DatatypeProperty	owl:DatatypeProperty
owl:DeprecatedClass	owl:DeprecatedClass	owl:DeprecatedClass
owl:DeprecatedProperty	owl:DeprecatedProperty	owl:DeprecatedProperty
owl:differentFrom	owl:differentFrom	owl:differentFrom
owl:disjointWith	owl:disjointWith	owl:disjointWith
owl:distinctMembers	owl:distinctMembers	owl:distinctMembers

owl:equivalentClass	owl:equivalentClass	owl:equivalentClass
owl:equivalentProperty	owl:equivalentProperty	owl:equivalentProperty
owl:FunctionalProperty	owl:FunctionalProperty	owl:FunctionalProperty
owl:hasValue	owl:hasValue	owl:hasValue
owl:imports	owl:imports	owl:imports
owl:incompatibleWith	owl:incompatibleWith	owl:incompatibleWith
owl:intersectionOf	owl:intersectionOf	owl:intersectionOf
owl:InverseFunctionalProperty	owl:InverseFunctionalProperty	owl:InverseFunctionalProperty
owl:inverseOf	owl:inverseOf	owl:inverseOf
owl:maxCardinality	owl:maxCardinality	owl:maxCardinality
owl:minCardinality	owl:minCardinality	owl:minCardinality
owl:Nothing	owl:Nothing	owl:Nothing
owl:ObjectProperty	owl:ObjectProperty	owl:ObjectProperty
owl:oneOf	owl:oneOf	owl:oneOf
owl:onProperty	owl:onProperty	owl:onProperty
owl:Ontology	owl:Ontology	owl:Ontology
owl:OntologyProperty	owl:OntologyProperty	
owl:priorVersion	owl:priorVersion	owl:priorVersion
owl:Restriction	owl:Restriction	owl:Restriction
owl:sameAs	owl:sameAs	owl:sameAs
owl:someValuesFrom	owl:someValuesFrom	owl:someValuesFrom
owl:SymmetricProperty	owl:SymmetricProperty	owl:SymmetricProperty
owl:Thing	owl:Thing	owl:Thing
owl:TransitiveProperty	owl:TransitiveProperty	owl:TransitiveProperty
owl:unionOf	owl:unionOf	owl:unionOf
owl:versionInfo	owl:versionInfo	owl:versionInfo

Appendix B. RDF Schema of OWL

See [Sec. 1.7](#) for a description of the purpose of this appendix. The RDF/XML version of this appendix can be found at http://www.w3.org/2002/07/owl

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-
syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-
schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#"
>
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   ]>

<rdf:RDF
  xmlns      ="&owl;"
  xmlns:owl ="&owl;"
  xml:base  ="http://www.w3.org/2002/07/owl"
  xmlns:rdf ="&rdf;"
  xmlns:rdfs="&rdfs;"
>

<Ontology rdf:about="">
  <imports rdf:resource="http://www.w3.org/2000/01/rdf-
schema"/>
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/
TR/2004/REC-owl-semantics-20040210/" />
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/
TR/2004/REC-owl-test-20040210/" />
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/
TR/2004/REC-owl-features-20040210/" />
  <rdfs:comment>This file specifies in RDF Schema
format the
     built-in classes and properties that together form
the basis of
     the RDF/XML syntax of OWL Full, OWL DL and OWL
Lite.
     We do not expect people to import this file
     explicitly into their ontology. People that do
import this file
     should expect their ontology to be an OWL Full
ontology.
  </rdfs:comment>
```

```
      <versionInfo>10 February 2004</versionInfo>
</Ontology>

<rdfs:Class rdf:ID="Class">
  <rdfs:label>Class</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<Class rdf:ID="Thing">
  <rdfs:label>Thing</rdfs:label>
  <unionOf rdf:parseType="Collection">
    <Class rdf:about="#Nothing"/>
    <Class>
      <complementOf rdf:resource="#Nothing"/>
    </Class>
  </unionOf>
</Class>

<Class rdf:ID="Nothing">
  <rdfs:label>Nothing</rdfs:label>
  <complementOf rdf:resource="#Thing"/>
</Class>

<rdf:Property rdf:ID="equivalentClass">
  <rdfs:label>equivalentClass</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="&rdfs;subClassOf"/>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="disjointWith">
  <rdfs:label>disjointWith</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="equivalentProperty">
  <rdfs:label>equivalentProperty</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="&rdfs;
subPropertyOf"/>
</rdf:Property>

<rdf:Property rdf:ID="sameAs">
```

```
    <rdfs:label>sameAs</rdfs:label>
    <rdfs:domain rdf:resource="#Thing"/>
    <rdfs:range rdf:resource="#Thing"/>
</rdf:Property>

<rdf:Property rdf:ID="differentFrom">
    <rdfs:label>differentFrom</rdfs:label>
    <rdfs:domain rdf:resource="#Thing"/>
    <rdfs:range rdf:resource="#Thing"/>
</rdf:Property>

<rdfs:Class rdf:ID="AllDifferent">
    <rdfs:label>AllDifferent</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="distinctMembers">
    <rdfs:label>distinctMembers</rdfs:label>
    <rdfs:domain rdf:resource="#AllDifferent"/>
    <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdf:Property rdf:ID="unionOf">
    <rdfs:label>unionOf</rdfs:label>
    <rdfs:domain rdf:resource="#Class"/>
    <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdf:Property rdf:ID="intersectionOf">
    <rdfs:label>intersectionOf</rdfs:label>
    <rdfs:domain rdf:resource="#Class"/>
    <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdf:Property rdf:ID="complementOf">
    <rdfs:label>complementOf</rdfs:label>
    <rdfs:domain rdf:resource="#Class"/>
    <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="oneOf">
    <rdfs:label>oneOf</rdfs:label>
    <rdfs:domain rdf:resource="&rdfs;Class"/>
    <rdfs:range rdf:resource="&rdf;List"/>
```

```
        </rdf:Property>

        <rdfs:Class rdf:ID="Restriction">
          <rdfs:label>Restriction</rdfs:label>
          <rdfs:subClassOf rdf:resource="#Class"/>
        </rdfs:Class>

        <rdf:Property rdf:ID="onProperty">
          <rdfs:label>onProperty</rdfs:label>
          <rdfs:domain rdf:resource="#Restriction"/>
          <rdfs:range rdf:resource="&rdf;Property"/>
        </rdf:Property>

        <rdf:Property rdf:ID="allValuesFrom">
          <rdfs:label>allValuesFrom</rdfs:label>
          <rdfs:domain rdf:resource="#Restriction"/>
          <rdfs:range rdf:resource="&rdfs;Class"/>
        </rdf:Property>

        <rdf:Property rdf:ID="hasValue">
          <rdfs:label>hasValue</rdfs:label>
          <rdfs:domain rdf:resource="#Restriction"/>
        </rdf:Property>

        <rdf:Property rdf:ID="someValuesFrom">
          <rdfs:label>someValuesFrom</rdfs:label>
          <rdfs:domain rdf:resource="#Restriction"/>
          <rdfs:range rdf:resource="&rdfs;Class"/>
        </rdf:Property>

        <rdf:Property rdf:ID="minCardinality">
          <rdfs:label>minCardinality</rdfs:label>
          <rdfs:domain rdf:resource="#Restriction"/>
          <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
        </rdf:Property>

        <rdf:Property rdf:ID="maxCardinality">
          <rdfs:label>maxCardinality</rdfs:label>
          <rdfs:domain rdf:resource="#Restriction"/>
          <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
        </rdf:Property>

        <rdf:Property rdf:ID="cardinality">
```

```
      <rdfs:label>cardinality</rdfs:label>
      <rdfs:domain rdf:resource="#Restriction"/>
      <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
    </rdf:Property>

    <rdfs:Class rdf:ID="ObjectProperty">
      <rdfs:label>ObjectProperty</rdfs:label>
      <rdfs:subClassOf rdf:resource="&rdf;Property"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="DatatypeProperty">
      <rdfs:label>DatatypeProperty</rdfs:label>
      <rdfs:subClassOf rdf:resource="&rdf;Property"/>
    </rdfs:Class>

    <rdf:Property rdf:ID="inverseOf">
      <rdfs:label>inverseOf</rdfs:label>
      <rdfs:domain rdf:resource="#ObjectProperty"/>
      <rdfs:range rdf:resource="#ObjectProperty"/>
    </rdf:Property>

    <rdfs:Class rdf:ID="TransitiveProperty">
      <rdfs:label>TransitiveProperty</rdfs:label>
      <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="SymmetricProperty">
      <rdfs:label>SymmetricProperty</rdfs:label>
      <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="FunctionalProperty">
      <rdfs:label>FunctionalProperty</rdfs:label>
      <rdfs:subClassOf rdf:resource="&rdf;Property"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="InverseFunctionalProperty">
      <rdfs:label>InverseFunctionalProperty</rdfs:label>
      <rdfs:subClassOf rdf:resource="&owl;ObjectProperty"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="AnnotationProperty">
      <rdfs:subClassOf rdf:resource="&rdf;Property"/>
```

```
          </rdfs:Class>

          <AnnotationProperty rdf:about="&rdfs;label"/>
          <AnnotationProperty rdf:about="&rdfs;comment"/>
          <AnnotationProperty rdf:about="&rdfs;seeAlso"/>
          <AnnotationProperty rdf:about="&rdfs;isDefinedBy"/>

          <rdfs:Class rdf:ID="Ontology">
            <rdfs:label>Ontology</rdfs:label>
          </rdfs:Class>

          <rdfs:Class rdf:ID="OntologyProperty">
            <rdfs:subClassOf rdf:resource="&rdf;Property"/>
          </rdfs:Class>

          <rdf:Property rdf:ID="imports">
            <rdfs:label>imports</rdfs:label>
            <rdf:type rdf:resource="#OntologyProperty"/>
            <rdfs:domain rdf:resource="#Ontology"/>
            <rdfs:range rdf:resource="#Ontology"/>
          </rdf:Property>

          <rdf:Property rdf:ID="versionInfo">
            <rdfs:label>versionInfo</rdfs:label>
            <rdf:type rdf:resource="#AnnotationProperty"/>
          </rdf:Property>

          <rdf:Property rdf:ID="priorVersion">
            <rdfs:label>priorVersion</rdfs:label>
            <rdf:type rdf:resource="#OntologyProperty"/>
            <rdfs:domain rdf:resource="#Ontology"/>
            <rdfs:range rdf:resource="#Ontology"/>
          </rdf:Property>

          <rdf:Property rdf:ID="backwardCompatibleWith">
            <rdfs:label>backwardCompatibleWitesh</rdfs:label>
            <rdf:type rdf:resource="#OntologyProperty"/>
            <rdfs:domain rdf:resource="#Ontology"/>
            <rdfs:range rdf:resource="#Ontology"/>
          </rdf:Property>

          <rdf:Property rdf:ID="incompatibleWith">
            <rdfs:label>incompatibleWith</rdfs:label>
```

```
      <rdf:type rdf:resource="#OntologyProperty"/>
      <rdfs:domain rdf:resource="#Ontology"/>
      <rdfs:range rdf:resource="#Ontology"/>
   </rdf:Property>

   <rdfs:Class rdf:ID="DeprecatedClass">
     <rdfs:label>DeprecatedClass</rdfs:label>
     <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
   </rdfs:Class>

   <rdfs:Class rdf:ID="DeprecatedProperty">
     <rdfs:label>DeprecatedProperty</rdfs:label>
     <rdfs:subClassOf rdf:resource="&rdf;Property"/>
   </rdfs:Class>

   <rdfs:Class rdf:ID="DataRange">
     <rdfs:label>DataRange</rdfs:label>
   </rdfs:Class>

   </rdf:RDF>
```

Appendix C. OWL Quick Reference

Classes in the OWL vocabulary:

rdfs:Class
owl:AllDifferent
owl:AnnotationProperty
owl:Class
owl:DataRange
owl:DatatypeProperty
owl:DeprecatedClass
owl:DeprecatedProperty
owl:FunctionalProperty
owl:InverseFunctionalProperty

owl:Nothing	
owl:ObjectProperty	
owl:Ontology	
owl:OntologyProperty	
owl:Restriction	
owl:SymmetricProperty	
owl:Thing	
owl:TransitiveProperty	

Properties in the OWL vocabulary:

rdf:Property	rdfs:domain	rdfs:range
owl:allValuesFrom	owl:Restriction	rdfs:Class
owl:backwardCompatibleWith	owl:Ontology	owl:Ontology
owl:cardinality	owl:Restriction	xsd:nonNegativeInteger
owl:complementOf	owl:Class	owl:Class
owl:differentFrom	owl:Thing	owl:Thing
owl:disjointWith	owl:Class	owl:Class
owl:distinctMembers	owl:AllDifferent	rdf:List
owl:equivalentClass	owl:Class	owl:Class
owl:equivalentProperty	rdf:Property	rdf:Property
owl:hasValue	owl:Restriction	
owl:imports	owl:Ontology	owl:Ontology
owl:incompatibleWith	owl:Ontology	owl:Ontology
owl:intersectionOf	owl:Class	rdf:List
owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty
owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger
owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger
owl:oneOf	owl:Class	rdf:List
owl:onProperty	owl:Restriction	rdf:Property

owl:priorVersion	owl:Ontology	owl:Ontology
owl:sameAs	owl:Thing	owl:Thing
owl:someValuesFrom	owl:Restriction	rdfs:Class
owl:unionOf	owl:Class	rdf:List
owl:versionInfo		

Appendix D. Changes from DAML+OIL

This section summarizes the changes from DAML+OIL [*DAML+OIL*] to OWL.

1. The semantics have changed significantly. With respect to the three sublanguages, the DAML+OIL semantics are closest to the OWL DL semantics.
2. The namespace was changed to http://www.w3.org/2002/07/owl.
3. Various updates to RDF and RDF Schema from the RDF Core Working Group were incorporated, including
   - cyclic subclasses are now allowed
   - multiple `rdfs:domain` and `rdfs:range` properties are handled as intersection
   - RDF Semantics [*RDF Semantics*]
   - datatypes

       RDF and OWL use the XML Schema namespace http://www.w3.org/2001/XMLSchema rather than http://www.w3.org/2000/10/XMLSchema.
       OWL does not support using datatypes as types, e.g.

       ```
       <size>
         <xsd:integer rdf:value="10"/>
       </size>
       ```

       Instead use

       ```
       <size
       rdf:datatype="http://www.w3.org/2001/
       XMLSchema#integer">10</size>
       ```

   - the `daml:List` construct used to represent closed collections was largely incorporated into RDF

[rdf:parseType="Collection"](#) replaces `rdf:`
`parseType="daml:collection"`
`rdf:List`, `rdf:first`, `rdf:rest` and `rdf:nil` replace
`daml:List`, `daml:first`, `daml:rest`, and `daml:nil`
`daml:item` is not supported. As this feature was typically
used to create typed lists, we include here an example of
creating such a list without using `daml:item`:

```
<rdfs:Class rdf:ID="OperaList">
  <rdfs:subClassOf rdf:
resource="&rdf;List"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:
resource="&rdf;first"/>
      <owl:allValuesFrom rdf:
resource="#Opera"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:
resource="&rdf;rest"/>
      <owl:allValuesFrom rdf:
resource="#OperaList"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</rdfs:Class>
```

This example defines a list of which the elements most be
operas. This is achieved by two restrictions, one on the `rdf:`
`first` value (representing the type of the list element) and a
second restriction on the `rdf:rest` value (which should the
name of the list being defined).

4. Qualified restrictions were removed from the language, resulting in the
   removal of the following properties:
   - `daml:cardinalityQ`
   - `daml:hasClassQ`
   - `daml:maxCardinalityQ`
   - `daml:minCardinalityQ`
5. Various properties and classes were renamed, as shown in the following
   table:

DAML+OIL	OWL
daml:differentIndividualFrom	owl:differentFrom
daml:equivalentTo	owl:sameAs
daml:sameClassAs	owl:equivalentClass
daml:samePropertyAs	owl:equivalentProperty
daml:hasClass	owl:someValuesFrom
daml:toClass	owl:allValuesFrom
daml:UnambiguousProperty	owl:InverseFunctionalProperty
daml:UniqueProperty	owl:FunctionalProperty

6. owl:SymmetricProperty was added.
7. Also added were owl:AnnotationProperty, owl:OntologyProperty and owl:DataRange.
8. An owl:DatatypeProperty may be an owl:InverseFunctionalProperty in OWL Full.
9. Synonyms for RDF and RDF Schema classes and properties were removed from the language, resulting in the removal of:
    - daml:comment
    - daml:domain
    - daml:label
    - daml:isDefinedBy
    - daml:Literal
    - daml:Property
    - daml:range
    - daml:seeAlso
    - daml:subClassOf
    - daml:subPropertyOf
    - daml:type
    - daml:value
10. daml:disjointUnionOf was removed from the language, since it can be effected using owl:unionOf or rdfs:subClassOf and owl:disjointWith.
11. daml:equivalentTo was renamed to owl:sameAs, and is no longer a superproperty of owl:equivalentClass and owl:equivalentProperty,
12. The following properties and classes were added to support versioning:

- ❍ [owl:backwardCompatibleWith](#)
- ❍ [owl:DeprecatedClass](#)
- ❍ [owl:DeprecatedProperty](#)
- ❍ [owl:incompatibleWith](#)
- ❍ [owl:priorVersion](#)

13. [owl:AllDifferent](#) and [owl:distinctMembers](#) were added to address the Unique Names Assumption.

Appendix E. Rules of Thumb for OWL DL ontologies

The OWL Abstract Syntax and Semantics document [*OWL S&AS*] provides a characterization of OWL ontologies in terms of an abstract syntax, along with a mapping to RDF triples.

The following rules give an informal characterization of the conditions for an RDF graph to be a DL ontology. This is not intended to replace the characterization given in S&AS, but instead gives some general pointers — the idea is that if you stick to these guidelines, you're more likely to produce OWL DL ontologies. Nor is this intended to tell you how to turn the triple representation into something closer to the abstract syntax.

Don't mess with the vocabulary

The built-in properties and classes should **not** be redefined. In general this means that things in the OWL, RDF or RDFS namespaces should not appear as subjects of triples.

Provide Explicit Typing

Everything should have a type[1]. If any URI reference x is used where a class is expected, the graph should contain a triple stating that

```
x rdf:type owl:Class
```

Similarly, if a property p is used where an object property is expected then there should be a triple[2]

```
p rdf:type owl:ObjectProperty
```

If a property q is used where a data property is expected then there should be a

triple

```
q rdf:type owl:DatatypeProperty
```

If a property `o` is used where an ontology property is expected then it should either be one of the built in ontology properties (`owl:imports`, `owl: priorVersion`, `owl:backwardCompatibleWith`, and `owl: incompatibleWith`) or there should be a triple:

```
o rdf:type owl:OntologyProperty
```

If a property `a` is used where an annotation property is expected then it should either be one of the built in annotation properties (`owl:versionInfo`, `rdfs: label`, `rdfs:comment`, `rdfs:seeAlso`, and `rdfs:isDefinedBy`) or there should be a triple:

```
a rdf:type owl:AnnotationProperty
```

Any individuals that occur in the ontology should have at least one type specified, i.e. for an individual `i`, there must be a triple:

```
i rdf:type c
```

where `c` is an `owl:Class` or `owl:Restriction`.

Keep names separate

URI references for classes, properties (object, datatype, ontology and annotation) and individuals should be disjoint. Thus we cannot have things like:

```
x rdf:type owl:Class
x rdf:type owl:ObjectProperty
```

In particular, this means that we cannot use *classes as instances*, i.e.

```
x rdf:type owl:Class
y rdf:type owl:Class
x rdf:type y
```

is not valid OWL DL. A general rule here is that if there is a node `x` in the graph with a triple:

```
x rdf:type owl:Class
```

then `x` should not appear as the subject of any other triple with predicate `rdf:type`.[3]

Restrictions

If a node `x` has `rdf:type owl:Restriction` then the following should be the case:

- It is a blank node (i.e. unnamed).
- It is not the subject of any other triple with predicate `rdf:type`[3].
- It is the subject of exactly **one** triple with predicate `owl:onProperty`, with the object of that triple being an `owl:ObjectProperty` or `owl:DatatypeProperty`.
- It is the subject of exactly **one** of:
  - A triple with predicate `owl:someValuesFrom`. In this case, the type of the property which is the object of the `owl:onProperty` triple should be appropriate. By this we mean that if the object of this triple is a datatype, the property should be an `owl:DatatypeProperty`. If the object is a class description, the property should be an `owl:ObjectProperty`. This typing information should be present (due to the restrictions outlined [above](#)).
  - A triple with predicate `owl:allValuesFrom`. Similar restrictions hold as for `owl:someValuesFrom`
  - A triple with predicate `owl:hasValue`. If the type of the property involved in the `owl:onProperty` triple is `owl:ObjectProperty` then the object of this triple should be an individual. If the type of the property involved in the `owl:onProperty` triple is `owl:DatatypeProperty` the the object of this triple should be a data literal.
  - A triple with predicate `owl:minCardinality`. The object of this triple should be a data literal representing a non negative integer.
  - A triple with predicate `owl:maxCardinality`. Restriction as for `owl:minCardinality`.
  - A triple with predicate `owl:cardinality`. Restriction as for `owl:minCardinality`.
- Any other triples in which `x` is the subject should have predicate `owl:equivalentClass` or `owl:disjointWith`.

Class axioms

For any triples with predicate `rdfs:subClassOf` or `owl:equivalentClass` or `owl:disjointWith`, both the subject and object of the triples should be an `owl:Class` or `owl:Restriction`, i.e. if we have:

```
x rdfs:subClassOf y
```

then the graph must contain one of:

```
x rdf:type owl:Class
```

or

```
x rdf:type owl:Restriction.
```

and one of

```
y rdf:type owl:Class
```

or

```
y rdf:type owl:Restriction.
```

Property axioms

For any triples with predicate `rdfs:subPropertyOf` or `owl:equivalentProperty`, both the subject and object of the triples should have the same type which should be one of `owl:ObjectProperty` or `owl:DatatypeProperty`, i.e. if we have:

```
p owl:equivalentProperty q
```

then the graph must contain either:

```
p rdf:type owl:ObjectProperty
q rdf:type owl:ObjectProperty.
```

or

```
       p rdf:type owl:DatatypeProperty
       q rdf:type owl:DatatypeProperty.
```

Triples with predicate `rdfs:domain` should have as their subject an `owl:ObjectProperty` or `owl:DatatypeProperty` and as their object an `owl:Class` or `owl:Restriction`.

Triples with predicate `rdfs:range` should have as their subject either an `owl:ObjectProperty` or an `owl:DatatypeProperty`. In the case of the former, the object of the triple should then be an `owl:Class` or `owl:Restriction`, in the case of the latter, the object should be either an XML Schema datatype, `rdfs:Literal`, or an `owl:oneOf` specifying a data range with type `owl:DataRange`.

Both the subject and object of an `owl:inverseOf` triple must have type `owl:ObjectProperty`.

Individual axioms

For any triples with predicate `owl:sameAs` or `owl:differentFrom`, the subject and object must be individuals.

Note that relating two classes via `owl:sameAs` is a very different thing to relating them via `owl:equivalentClass`. The former says that the two objects are in fact the same, is actually an example of *class as instance*, and thus pushes the ontology out of OWL DL. The latter is an assertion that the extension (e.g. the collection of members) of the classes is equivalent.

Similarly, relating classes via `owl:differentFrom` is not the same as relating them via `owl:disjointWith` (and is again an example of an OWL Full construct). Two classes may be different objects but still share the same extension.

If a node `x` has `rdf:type owl:AllDifferent`, then the following should be the case:

> It is a blank node (i.e. unnamed).
> It is the subject of a triple with predicate `owl:distinctMembers`, the object of which should be a (well-formed) `rdf:List`, all of whose elements are individuals.
> It is not the subject (or object) of any other triples.

Boolean class expressions

Boolean Operators (and, or, not) are represented in OWL using `owl:intersectionOf`, `owl:unionOf` and `owl:complementOf`.

The subject of an `owl:complementOf` triple must be an `owl:Class`, the object must be either an `owl:Class` or `owl:Restriction`.

The subject of an `owl:unionOf` or `owl:intersectionOf` triple must be an `owl:Class`, the object must be a (well-formed) `rdf:List`, all of whose elements are either `owl:Class` or `owl:Restriction`. These could either be represented explicitly using expanded `rdf:Lists`, or if RDF-XML is being used, an `rdf:parseType="Collection"` attribute.

```
<owl:Class>
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="x"/>
  <owl:Class rdf:about="y"/>
 </owl:intersectionOf>
</owl:Class>
```

If the `owl:Class` is a blank node (i.e. the class is unnamed), then it can only be the subject of at most one triple with predicate `owl:intersectionOf`, `owl:unionOf` or `owl:complementOf`. If the class is named, any number of such triples are allowed.

Enumerations

The subject of any triple with predicate `owl:oneOf` must be either an `owl:Class` or an `owl:DataRange`. In the case of the former, the object must be a (well-formed) `rdf:List`, all of whose elements are individuals. In the case of the latter, the object must be a (well-formed) `rdf:List`, all of whose elements are data literals. Again, as with the boolean operators, `rdf:parseType="Collection"` can be used.

Ontology and Annotation Properties

The subject and object of any triple with an ontology predicate should be an ontology, e.g. a node `x` such that there is a triple:

```
x rdf:type owl:Ontology
```

The subject of any triple with an annotation predicate should be a named (i.e. non-bnode) class, a property, an individual or an ontology. The object of a triple with an annotation predicate should be an individual, a data literal or an arbitrary URI reference.

With the exception of predicates within the OWL, RDF and RDFS vocabularies, annotation properties are the **only** predicates that should appear in triples with a class or property as the subject.

Annotation and ontology properties themselves should be [typed](#), and should not appear as the subject or object of triples other than as the subject of a triple with predicate `rdf:type` or an annotation property.

Avoid structure sharing

In general, the S&AS description of OWL does not permit *structure sharing* in the RDF representation. This effectively means that an anonymous node in the RDF graph representing a particular description should only occur once (as the object of a triple). Thus things like:

```
x1 rdf:type owl:Class
x1 rdfs:subClassOf _:y
x2 rdf:type owl:Class
x2 rdfs:subClassOf _:y
_:y rdf:type owl:Class
_:y owl:complementOf z
```

should be avoided. There are some tricky corner cases where this *is* permitted. In general, however, graphs should use distinct blank nodes whenever a class description is used in more than one place.

Avoid orphan blank nodes

In general, blank nodes occurring in the graph either represent unnamed individuals, or should be exactly **one** of the following:

The object of an `rdfs:subClassOf, owl:equivalentClass, owl:disjointWith, owl:someValuesFrom, owl:allValuesFrom` or `rdf:type` triple.

The subject of an `rdf:type` triple with object `owl:AllDifferent`.

An element in an `rdf:List`.

Orphan blank nodes, i.e. those which are not the object of a triple are, in general, not allowed (other than the `owl:AllDifferent` case described above).

Ground facts

Ontologies may contain assertions of ground facts (e.g. triples that assert the properties of individuals). The properties used in these assertions must be an `owl:ObjectProperty` or `owl:DatatypeProperty`. The subject of any such triple must be an individual (which should be [typed](#)). The object can either be a reference to an individual (if the property is an `owl:ObjectProperty`) or a data literal (if the property is an `owl:DatatypeProperty`).

OWL Lite

OWL Lite documents should follow the same rules as OWL DL documents, with a number of extra restrictions, primarily concerning the vocabulary allowed. An OWL Lite document should not use any of the following vocabulary:

```
owl:unionOf
owl:complementOf
owl:oneOf
owl:hasValue
owl:disjointWith
```

Any objects which are the object or subject of a triple with predicate `owl:equivalentClass` should not be b-nodes.

The object of any triples with predicate `owl:minCardinality, owl:maxCardinality` or `owl:cardinality` should be a data literal representing the integer 0 or 1.

The situation regarding the use of `owl:intersectionOf` in OWL Lite is a little more complex. The predicate should **not** be used to form arbitrary expressions, but **is** needed in order to represent complete class definitions. The restrictions above tell us that the subject of any triple with predicate `owl:intersectionOf` should be a `owl:Class`. In OWL Lite, we have the further restriction that this class should be named, i.e. the subject should not be a bnode.

Miscellaneous

Be careful of the use of `owl:Thing`. For example, the following OWL-RDF fragment:

```
<owl:Class rdf:about="#A">
  <rdfs:subClassOf>
    <owl:Thing/>
  </rdfs:subClassOf>
</owl:Class>
```

does **not** describe a class `A` that is a subclass of `owl:Thing`, but in fact describes a class `A` that is a subclass of some anonymous instance of `owl:Thing`. This is thus a use of class as instance and is outside OWL DL. The desired effect of a subclass of `owl:Thing` is obtained through:

```
<owl:Class rdf:about="#A">
  <rdfs:subClassOf>
    <owl:Class rdf:about="http://www.w3.org/2002/07/
owl#Thing"/>
  </rdfs:subClassOf>
</owl:Class>
```

Be careful not to confuse `owl:Class` and `rdfs:Class`. The following is **not** in DL due to the fact that `c` is not given an appropriate type.

```
c rdf:type rdfs:Class
```

Notes

[1] Of course the necessity to type everything does not apply to things from the OWL, RDF or RDFS namespaces.

[2] Strictly speaking, if the property is defined as being an `owl:TransitiveProperty`, `owl:SymmetricProperty` or `owl:InverseFunctionalProperty` then this is not necessary.

[3] An exception here is that we can have:

```
x rdf:type rdfs:Class
x rdf:type owl:Class
```

```
p rdf:type rdf:Property
p rdf:type owl:ObjectProperty
```

or

```
q rdf:type rdf:Property
q rdf:type owl:DatatypeProperty
```

In addition, for restrictions, we can have:

```
x rdf:type owl:Restriction
x rdf:type rdfs:Class
x rdf:type owl:Class
```

Appendix F. Change Log since PR

1. Removed spurious endnote [4] in Appendix E.
2. Added rdfs:label element to AnnotationProperty and OntologyProperty in Appendix B (RDF Schema of OWL) after comment from Jacco van Ossenbruggen.
3. Fixed broken section reference plus corrected section reference style.
4. Standardized reference section.
5. Editorial revision of requirement description for rdf:RDF element after comment from Minsu Jang.
6. Several editorial changes in response to a review by Lacy.
7. Added explanatory text to Sec. 7.1 about OWL DL constraints on the use of annotation properties, after some public comments (for example, see the comments by Benjamin Nowack. Added also a sentence to Sec. 7.2 to indicate that the same constraints hold for ontology properties.
8. Several small editorial corrections after final read-though by editor.

References

[OWL Overview]

OWL Web Ontology Language Overview, Deborah L. McGuinness and Frank van Harmelen, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/ . Latest version available at http://www.w3.org/TR/owl-features/ .

[OWL Guide]

OWL Web Ontology Language Guide, Michael K. Smith, Chris Welty, and Deborah L. McGuinness, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/ . Latest version available at http://www.w3.org/TR/owl-guide/ .

[OWL Semantics and Abstract Syntax]

OWL Web Ontology Language Semantics and Abstract Syntax, Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/ . Latest version available at http://www.w3.org/TR/owl-semantics/ .

[OWL Test]

OWL Web Ontology Language Test Cases, Jeremy J. Carroll and Jos De Roo, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-test-20040210/ . Latest version available at http://www.w3.org/TR/owl-test/ .

[OWL Requirements]

OWL Web Ontology Language Use Cases and Requirements, Jeff Heflin, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-webont-req-20040210/ . Latest version available at http://www.w3.org/TR/webont-req/ .

[RDF Concepts]

Resource Description Framework (RDF): Concepts and Abstract Syntax, Graham Klyne and Jeremy J. Carroll, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ . Latest version available at http://www.w3.org/TR/rdf-concepts/ .

[RDF Syntax]

RDF/XML Syntax Specification (Revised), Dave Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ . Latest version available at http://www.w3.org/TR/rdf-syntax-grammar/ .

[RDF Semantics]

RDF Semantics, Pat Hayes, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ . Latest version available at http://www.w3.org/TR/rdf-mt/ .

[RDF Vocabulary]

RDF Vocabulary Description Language 1.0: RDF Schema, Dan Brickley and R. V. Guha, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ . Latest version available at http://www.w3.org/TR/rdf-schema/ .

[DAML+OIL]

DAML+OIL (March 2001) Reference Description. Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. W3C Note 18 December 2001. Latest version is available at `http://www.w3.org/TR/daml+oil-reference`.

[XML-SCHEMA2]

XML Schema Part 2: Datatypes - W3C Recommendation, World Wide Web Consortium, 2 May 2001.