14 UML Profile for RDF and OWL

This profile is based on the UML Kernel package defined in “Unified Modeling Language: Superstructure”, version 2 [UML2] as well as on the Profiles section of the same document. It is designed to support modelers developing vocabularies in RDF and richer ontologies in the Web Ontology Language (OWL) through reuse of UML notation using tools that support UML2 extension mechanisms. The profile

- Reuses UML constructs when they have the same semantics as OWL, or, when this is not possible, stereotypes UML constructs that are consistent and as close as possible to OWL semantics.

- Uses standard UML 2 notation, or, in the few cases where this is not possible, follows the clarifications and elaborations of stereotype notation defined in UML 2.1 (See [UML2.1], Profiles chapter).

- Leverages the model library provided in Appendix A, Foundation Library (M1) for RDF and OWL.

The profile has been partitioned to support users who wish to restrict their vocabularies to RDF/S, as well as to reflect the structure of the RDF and OWL metamodels (and the languages themselves). It leverages stereotypes extensively and also uses stereotype properties in traditional fashion. It complements the metamodels defined in Chapter 10, The RDF Metamodel, and in Chapter 11, The OWL Metamodel, respectively, for overall structure, semantics and language mapping. It also depends on model elements included in Appendix A, Foundation Library (M1) for RDF and OWL, for certain basic definitions, such as the M1 level elements discussed in Chapter 8, Design Rationale.

14.1 UML Profile for RDF

14.1.1 RDF Profile Package

Description

The following sections specify the set of stereotypes that comprise the UML2 profile for using UML to represent RDF/S vocabularies. It is designed to support all of RDF, as reflected in the RDF metamodel (Base, RDFS, and Web, together) provided in Chapter 10, The RDF Metamodel, and in Chapter 11, The OWL Metamodel, respectively, for overall structure, semantics and language mapping. It also depends on model elements included in Appendix A, Foundation Library (M1) for RDF and OWL, for certain basic definitions, such as the M1 level elements discussed in Chapter 8, Design Rationale.

Constraints

[1] All classes in a package stereotyped by «rdfDocument» must be stereotyped by «rdfsClass» (or an appropriate subclass, such as «owlClass»).

14.1.2 RDF Documents

Stereotypes and other profile elements corresponding to the RDF Web and document specific definitions given in 10.7 (“RDF Documents and Namespaces (RDFWeb Package)”), are defined in this section.
14.1.2.1 NamespaceDefinition

Description
A namespace is declared using a family of reserved attributes. These attributes, like any other XML attributes, may be provided directly or by default. Some names in XML documents (constructs corresponding to the nonterminal Name) may be given as qualified names. The prefix provides the namespace prefix part of the qualified name, and must be associated with a namespace URI in a namespace declaration.

Stereotype and Base Class
No stereotype is defined for this class.

Parent
None.

Properties
- namespacePrefix: String [1] - the string representing the namespace prefix
- namespaceURI: String [1] - the string representing the namespace URI

Constraints
- [1] The string value of namespacePrefix must conform to the specification given in [XMLNS].
- [2] The string value of the namespace URI must conform to the character encoding (including escape sequences and so forth) defined in [RDF Syntax] and [XMLNS].

14.1.2.2 RDFDocument

Description
An RDF document represents the primary namespace mechanism / container for an RDF/S vocabulary. The ordered set of definitions (statements) that comprise an RDF vocabulary are contained in a document. This set of statements may also correspond to one or more graphs contained in the document.

Note that this approach supports RDF graphs that span multiple documents, and enables multiple graphs to occur within a particular document (although it is more natural from a UML modelling perspective to assume that there is a 1-1 mapping between a graph and a document).

Stereotype and Base Class
«rdfDocument» with base class of UML::Package

Parent
None.

Properties
RDF namespace declarations are associated with the RDF document that acts as the container for the set of RDF graphs that make up the vocabulary or ontology component, rather than with the optional ontology header definition (in the case of an OWL ontology) or other statements.
- defaultNamespace: String [0..1] – provides the default namespace, or base for the document, if available.
- xmlBase: String [0..*] – provides zero or more base namespaces used in the document.
• namespaceDefinition: NamespaceDefinition [0..*] – defines zero or more namespace definitions used in the document.

• statementForDocument: InstanceSpecification [1..*] {ordered} – the set of statements in the document

Constraints

[1] Either defaultNamespace must be present or at least one xmlBase must be specified. (An «rdfDocument» must have an IRI/URI.)

[2] The string value for any defaultNamespace property must conform to the character encoding (including escape sequences and so forth) defined in defined in [RDF Syntax] and [XMLNS].

[3] The string value for any xmlBase property must conform to the character encoding (including escape sequences and so forth) defined in defined in [RDF Syntax] and [XMLNS].

[4] The classifier of the InstanceSpecification(s) for the statements in a document must be RDFStatement.

14.1.2.3 UniformResourceIdentifier

Description

The RDF abstract syntax is concerned primarily with URI references. This definition of a URI and related stereotype, distinct from URI reference, is included primarily for mapping purposes (i.e., mapping across RDF vocabularies and OWL ontologies, as well as among the paradigms covered in this specification). The distinction between a URI and URI reference is covered more thoroughly in Chapter 10, The RDF Metamodel. Also, see [RDF Syntax] for further definition detail.

Note that URIs and IRI/URI references specified using this RDF profile are globally defined, in contrast to naming and namespace conventions in UML2, which can be limited to the package level or to a set of nested namespaces.

Stereotype and Base Class

«uniformResourceIdentifier» with base class of UML::LiteralString

Parent

None.

Properties

None.

Constraints

[1] The string value for the «uniformResourceIdentifier» must conform to the character encoding (including escape sequences and so forth) defined in defined in [RDF Syntax] and [XMLNS].

[2] The string value for the «uniformResourceIdentifier» must be present (i.e., the IRI/URI).

14.1.2.4 URIReference

Description

RDF uses URI references to identify resources and properties. A URI reference within an RDF graph (an RDF URI reference) is a Unicode string conforming to the characteristics defined in [RDF Concepts] and [RDF Syntax].

RDF URI references can be:

• given as XML attribute values interpreted as relative URI references that are resolved against the in-scope base
URI to give absolute RDF URI references

- transformed from XML namespace-qualified element and attribute names (QNames)
- transformed from rdf:ID attribute values.

**Stereotype and Base Class**

«uriReference» with base class of UML::LiteralString

**Parent**

None.

**Properties**

- uri: LiteralString [0..1] – links a URIReference to the URI/IRI it contains/represents.

**Constraints**

1. The string value of the «uriReference» must conform to the constraints defined in [RDF Syntax] and [XMLNS].
2. The value of the uri property must be a UML::LiteralString that is stereotyped by «uniformResourceIdentifier».
3. (Semantic) The string value of the «uriReference» corresponds to the optional fragment identifier of the IRI/URI.

### 14.1.3 RDF Statements

The stereotypes and other profile elements corresponding to the RDF base definitions given in 10.2 (“RDFBase Package, RDF Statements”) are defined in this section.

#### 14.1.3.1 BlankNode

**Description**

A blank node is a node that is not a URI reference or a literal. In the RDF abstract syntax, a blank node is simply a unique node that can be used in one or more RDF statements, but has no intrinsic name. Blank nodes are an integral part of the RDF language, and are used extensively in OWL class descriptions. In practice in a UML tool environment, it is likely that they will be needed when reverse engineering RDF vocabularies and OWL ontologies, and most importantly for coreference resolution when mapping across ontologies.

**Stereotype and Base Class**

«blankNode» with base class of UML::InstanceSpecification

**Parent**

«rdfsResource»

**Properties**

- nodeID: String [0..1] – provides an optional blank node identifier.

**Constraints**

1. The uriRef property inherited from «rdfsResource» must not have a value (i.e., must be empty).
An InstanceSpecification cannot be stereotyped by «blankNode» and «uriReferenceNode» at the same time.

14.1.3.2 RDFGraph

Description
An RDF graph is the container for the set of statements (subject, predicate, object subgraphs) in an RDF/S vocabulary. In UML this container is a package. This definition of a graph is included in the profile to support identification / componentization of RDF vocabularies through the use of named graphs, and for vocabulary mapping purposes.

Stereotype and Base Class
«rdfGraph» with base class of UML::Package

Parent
None.

Properties
- graphName: LiteralString [0..1] – the optional URI reference naming the graph.
- statementForGraph: InstanceSpecification [1..*] – the set of statements in the graph

Constraints
[1] The string value of the graphName property must be a UML::LiteralString that is stereotyped by «uriReference».
[2] The value of any statementForGraph property must be a UML::InstanceSpecification classified by RDF-Statement, and may have the «rdfsResource» stereotype applied.

14.1.3.3 RDFSLiteral

Description
Literals are used to identify values such as numbers and dates by means of a lexical representation. Anything represented by a literal could also be represented by a URI, but it is often more convenient or intuitive to use literals.

A literal may be the object of an RDF statement, but not the subject or the predicate.

Literals may be plain or typed. The string value associated with an «rdfsLiteral» corresponds to its lexical form.

Stereotype and Base Class
«rdfsLiteral» with base class of UML::LiteralString

Parent
None.

Properties
None.

Constraints
[1] The string value associated with an «rdfsLiteral» must be a Unicode string in Normal Form C [Unicode].
14.1.3.4 RDFSResource

Description
All things described by RDF are called resources.

The uriRef property is used to uniquely identify an RDF resource globally. Note that this property has a multiplicity of [0..*] which provides for the possibility of the absence of an identifier, as in the case of blank nodes, and the possibility of multiple identifiers.

A particular resource may be identified by more than one URI reference, and may be reified by another resource (see «reifies», below).

Stereotype and Base Class
«rdfsResource» with base class of UML::InstanceSpecification

Parent
None.

Properties
• uriRef: LiteralString [0..*] – the URI reference(s) associated with a resource.
• memberOf: LiteralString [0..*] – a URI reference(s) relating a particular resource to another in terms of membership (i.e., in a class or container).

Constraints
[1] The string value(s) of the uriRef property must be a UML::LiteralString that is stereotyped by «uriReference».
[2] The string value(s) of the memberOf property must be a UML::LiteralString that is stereotyped by «uriReference».

14.1.3.5 RDFStatement

Description
An RDF triple contains three components:
• the subject, which is an RDF URI reference or a blank node.
• the predicate, which is an RDF URI reference, and represents a relationship.
• the object, which is an RDF URI reference, a literal or a blank node.

An RDF triple is conventionally written in the order subject, predicate, object. The relationship represented by the predicate is also known as the property of the triple. The direction of the arc is significant: it always points toward the object.

Stereotype and Base Class
No stereotype is defined for this class.

Parent
None.
Properties

- reification: ReificationKind [1] – indicates whether or not a particular statement (triple) is reified but not asserted, reified, or neither; default value is “none”.
- subject: InstanceSpecification [0..1] – the resource that is the subject of the statement
- predicate: AssociationClass [0..1] – the predicate for the statement
- object: InstanceSpecification [0..1] – the resource that is the object of the statement

Constraints

[1] The value of the subject property must be a UML::InstanceSpecification that is classified by «uriReferenceNode» or «blankNode»
[2] The value of the predicate property must be a UML::AssociationClass that is classified by «rdfProperty»
[3] The value of the object property must be a UML::InstanceSpecification that is classified by «rdfsResource»
[4] If the value of the reification property is “reified”, then the subject, predicate, and object must be filled.

14.1.3.6 Reification

Description

A particular resource may be identified by more than one URI/IRI reference, and may be reified by another resource, represented by a dependency between instance specifications (resources) stereotyped by «reifies».

Stereotype and Base Class

«reifies» with base class of UML::Dependency

Parent

None.

Properties

None.

Constraints

[1] The «reifies» stereotype can only be applied to a UML::Dependency that links two UML::InstanceSpecifications that are stereotyped by «rdfsResource». The dependency must be navigable from the reifying resource to the resource it reifies.

14.1.4 ReificationKind

Description

ReificationKind is an enumerated type used by the reification property on RDFStatement. It has three possible values: none, which is the default value, reified (meaning that the statement is both asserted and reified), and reifiedOnly (meaning that a statement is reified but not asserted - providing a placeholder in a UML model representing an RDF vocabulary for such a statement, which is necessary when we want to say things about statements that occur in some external vocabulary that are not available to this model).

Stereotype and Base Class

No stereotype is defined for this enumeration.
14.1.4.1 URIReferenceNode

Description
A URI reference or literal used as a node identifies what that node represents. URIReferenceNode is included in order to more precisely model the intended semantics (i.e., not all URI references are nodes). A URI reference used as a predicate identifies a relationship between the things represented by the nodes it connects. A predicate URI reference may also be a node in the graph.

Stereotype and Base Class
«uriReferenceNode» with base class of UML::InstanceSpecification

Parent
«rdfsResource»

Properties
None.

Constraints

14.1.5 Literals

The stereotypes associated with the definitions given in 10.3 ("RDFBase Package, RDF Literals"), in addition to those given above, are defined in this section.

14.1.5.1 PlainLiteral

Description
A plain literal is a string combined with an optional language tag. This may be used for plain text in a natural language. Plain literals are self-denoting.

Stereotype and Base Class
«plainLiteral» with base class of UML::LiteralString
Parent
«rdfsLiteral»

Properties
• language: String [0..1] - the optional language tag.

Constraints
[1] A LiteralString cannot be stereotyped by «plainLiteral» and «typedLiteral» at the same time.
[2] The string value of the language property, if present, must conform to the syntax and encoding specified in [RFC3066].

14.1.5.2 RDFSCComment

Description
A comment is a plain literal, contained by the resource it describes, and can be applied to any resource. Because it seems natural from a UML perspective to use a UML Comment for this feature, rather than inheriting or having a relationship with «plainLiteral», we've added a property to Comment to optionally support language tags.

Stereotype and Base Class
«rdfsComment» with base class of UML::Comment

Parent
None.

Properties
• language: String [0..1] - an optional language tag.

Constraints
[1] The string value of the language property, if present, must conform to the syntax and encoding specified in [RFC3066].

14.1.5.3 RDFSLLabel

Description
A label is a plain literal, contained by the resource it describes, and provides a human-readable label, or “pretty name” that can be applied to any resource.

Stereotype and Base Class
«rdfsLabel» with base class of UML::LiteralString

Parent
«plainLiteral»

Properties
None.
Constraints
No additional constraints.

14.1.5.4 RDFXMLLiteral

Description
rdf:XMLLiteral is a predefined RDF datatype used specifically for encoding XML in an RDF document. The value of the datatypeURI property must correspond to the URI for rdf:XMLLiteral.

Stereotype and Base Class
«rdfXMLLiteral» with base class of UML::LiteralString

Parent
«typedLiteral»

Properties
None.

Constraints
[1] The value of the datatypeURI property must be the IRI/URI for rdf:XMLLiteral.

14.1.5.5 TypedLiteral

Description
Typed literals have a lexical form, which is a Unicode string, and a datatype URI being an RDF URI reference.

The datatype URI refers to a datatype. For XML Schema built-in datatypes, URIs such as http://www.w3.org/2001/XMLSchema#integer are used. The URI of the datatype rdf:XMLLiteral may be used. There may be other, implementation dependent, mechanisms by which URIs refer to datatypes.

The value associated with a typed literal is found by applying the lexical-to-value mapping associated with the datatype URI to the lexical form. If the lexical form is not in the lexical space of the datatype associated with the datatype URI, then no literal value can be associated with the typed literal. Such a case, while semantically in error, is syntactically well-formed.

Stereotype and Base Class
«typedLiteral» with base class of UML::LiteralString

Parent
«rdfsLiteral»

Properties
- datatypeURI: LiteralString [1] – specifies the URI for the datatype specification that defines its type (of which it is an instance).

Constraints
[1] A typed literal must have a value for the datatypeURI property.
The string value of the datatypeURI property must be a UML::LiteralString that is stereotyped by «uriReference».

For built-in datatypes (i.e., those that are not user-defined), the string value of the datatypeURI must be that of an XML Schema Datatype as defined in [XML Schema Datatypes], and as given in Appendix A.

A LiteralString cannot be stereotyped by «plainLiteral» and «typedLiteral» at the same time.

(Semantic) The value of the datatypeURI property must match a URI corresponding to a datatype definition of the appropriate type.

14.1.6 Classes and Utilities

The stereotypes associated with the definitions given in 10.4 (“RDFS Package, Classes and Utilities”), in addition to those given above, are defined in this section.

14.1.6.1 RDFSClass

Description
The collection of resources that represents RDF Schema classes is itself a class, called rdfs:Class. Classes provide an abstraction mechanism for grouping resources with similar characteristics. The members of a class are known as instances of the class. Classes are resources. They are often identified by RDF URI References and may be described using RDF properties.

An RDFS class maps closely to the UML definition of a class; one notable exception is that an RDFS class may have a URI reference. The definition of the «rdfsClass» stereotype corresponds to 10.4.1 (“RDFSClass”).

Stereotype and Base Class
«rdfsClass» with base class of UML::Class

Parent
None.

Properties
• uriRef: LiteralString [0..*] – the URI reference(s) associated with an «rdfsClass».

Constraints
[1] The value of the uriRef property must be a UML::LiteralString that is stereotyped by «uriReference».

14.1.6.2 RDFSDatatype

Description
rdfs:Datatype represents the class of datatypes in RDF. Instances of rdfs:Datatype correspond to the RDF model of a datatype described in the RDF Concepts specification [RDF Concepts]. Note that built-in instances of rdfs:Datatype correspond to the subset of datatypes (defined in [XML Schema Datatypes]) allowable for use in RDF, as specified in [RDF Concepts]. These are provided for use with the metamodel(s) and profile(s) in the model library given in Appendix A (“Foundation Library (M1) for RDF and OWL”). Use of user-defined datatypes should be carefully considered against any desire for reasoning over an RDF vocabulary, OWL ontology, or knowledge base.

Stereotype and Base Class
«rdfsDatatype» with base class of UML::Datatype
Parent
None.

Properties
• uriRef: LiteralString [0..*] – the URI reference(s) associated with the datatype.

Constraints
[1] A class stereotyped by «rdfsDatatype» must have a value for the uriRef property.
[2] The value of the uriRef property must be a UML::LiteralString that is stereotyped by «uriReference».
[3] For built-in datatypes (i.e., those that are not user-defined), the string value of the uriRef must be that of an XML Schema Datatype as defined in [XML Schema Datatypes], and as given in Appendix A.
[4] (Semantic) The value of the uriRef property must link the «rdfsDatatype» to either an XML Schema Datatype or user-defined type corresponding to a datatype definition of the appropriate type.

14.1.6.3 RDFSisDefinedBy

Description
rdfs:isDefinedBy provides a means to indicate that a particular resource (the source, or owning classifier) is defined by another resource (the target resource). Note that RDF does not constrain the usage of rdfs:isDefinedBy, though in practice, vocabularies that use this construct, such as the Dublin Core, will do so.

Stereotype and Base Class
«rdfsIsDefinedBy» with base class of UML::Dependency

Parent
«rdfsSeeAlso»

Properties
None.

Constraints
[1] (Semantic) The «rdfsIsDefinedBy» stereotype is used to state that a particular resource (the subject of the RDF statement, i.e., the classifier owning the dependency) is defined by another resource (the object of the RDF statement, i.e., the type of the dependency). In theory, this stereotype can be applied to a dependency between any two “generic” resources, but in practice, we recommend that it is applied to a UML::Dependency that links two UML::InstanceSpecifications that are stereotyped by «rdfsResource», or, at a minimum, that the type of the dependency should be a UML::InstanceSpecifications stereotyped by «rdfsResource».

14.1.6.4 RDFSseeAlso

Description
rdfs:seeAlso indicates that more information about a particular resource (the source, or owning classifier) can be found at the target resource.

Stereotype and Base Class
«rdfsSeeAlso» with base class of UML::Dependency
Ontology Definition Metamodel

Parent
None.

Properties
None.

Constraints
[1] (Semantic) The «rdfsSeeAlso» stereotype is used to state that additional information about a particular resource (the subject of the RDF statement, i.e., the classifier owning the dependency) is given by another resource (the object of the RDF statement, i.e., the type of the dependency). As with «rdfsIsDefinedBy», this stereotype can be applied to a dependency between any two “generic” resources, but in practice, we recommend that it is applied to a UML::Dependency that links two UML::InstanceSpecifications that are stereotyped by «rdfsResource», or, at a minimum, that the type of the dependency should be a UML::InstanceSpecifications stereotyped by «rdfsResource».

14.1.6.5 RDFSsubClassOf

Description
rdfs:subClassOf indicates that the resource is a subclass of the general class; It has the semantics of UML Generalization. However, classes on both ends of the generalization must be stereotyped «rdfsClass», or «owlClass», if used with the profile for OWL.

Note in OWL DL that mixing inheritance among RDFS and OWL classes is permitted, as long as proper subclassing semantics is maintained. In order for a model to be well formed an OWL class can be a subclass of an RDFS class but not vice versa. In other words for OWL DL, once you're in OWL you need to stay there.

Stereotype and Base Class
«rdfsSubClassOf» with base class of UML::Generalization

Parent
None.

Properties
None.

Constraints
[1] Classes on both ends of the generalization must be stereotyped «rdfsClass», or «owlClass», if used with the profile for OWL.

[2] In OWL DL, a class stereotyped by «owlClass» may specialize a class stereotyped by «rdfsClass», but not vice versa.

14.1.6.6 RDFType

Description
rdf:type maps to the relation between instance and classifier in UML. This is equivalent in UML to the relation from an element in a model to an element in the UML metamodel, or between an instance specification and its classifiers. Note that resources in RDF can be multiply classified. No stereotype is needed.
14.1.7 Properties in RDF

14.1.7.1 RDFProperty

Description
A property in UML can be defined as part of an association or not. When it is not part of an association, the property is owned by the class that defines its domain, and the type of the property is the class that defines its range. When a property is part of an association, the association is binary, with the class that defines the domain of the property owning that property.

Properties in RDF and OWL are defined globally, that is, they are available to all classes in all ontologies – not only to classes in the ontology they are defined in, but to classes in ontologies that are imported. For RDF properties that are defined without specifying a domain or range, the profile uses an anonymous class (analogous to owl:Thing in OWL ontologies) for the “missing” end class.

Stereotype and Base Class
«rdfProperty» with base class of UML::AssociationClass and UML::Property

Parent
None.

Properties
- uriRef: LiteralString [1] – the URI reference(s) associated with an «rdfProperty».

Constraints
[1] The value of the uriRef property must be a UML::LiteralString that is stereotyped by «uriReference».
[2] Association classes with «rdfProperty» applied are binary, and have unidirectional navigation (i.e., explicitly from the class that defines its domain to the class that defines its range, in other words from the class that owns it to its type).
[3] Properties cannot have the same value twice (i.e., in UML, isUnique=true).
[4] Property values are not ordered (i.e., in UML, isOrdered=false).

Graphical Representation
There are several alternatives for representing various aspects of RDF properties in UML, as follows.

A. Properties without a specified domain are considered to be defined on an anonymous class, (or possibly on owl:Thing in the case of an OWL ontology), for example, as shown in Figure 30.

![Figure 30 Property hasColor Without Specified Domain](image)

From a UML perspective, properties are semantically equivalent to binary associations with unidirectional navigation (“one-way” associations).
Figure 31 shows that there is efficient navigation from an instance of an anonymous class to an instance of Color through the hasColor end, just like a UML property. The only difference from a property on the anonymous class is that the underlying repository will have an association with the hasColor property as one of its ends. The other end will be owned by the association itself, and be marked as nonnavigable.

Associations can be classes, as shown in Figure 32:

An association class can have properties, associations, and participate in generalization as any other class. Notice that the association has a (slightly) different name than the property, by capitalizing the first letter, to distinguish the association class (of links, tuples) from the mapping (across those links, tuples). A stereotype «rdfProperty» is introduced to highlight such binary, unidirectional association classes, as shown in Figure 32. In the examples given in the remainder of the profile, the notation showing properties in class rectangles is sometimes used, but unidirectional associations and association classes could be used instead.

B. Properties with a domain are defined on a UML class for the domain, where the property is not inherited from a supertype.

Normally UML models introduce properties and restrict them with multiplicities in the same class. This translates to RDF/OWL as global properties of an anonymous class (or possibly to owl:Thing in OWL, and to restrictions on subclasses of owl:Thing). An optional stereotype «rdfGlobal» is introduced to highlight properties on the class where they are introduced, which will translate to global properties in OWL. Properties that are inherited are distinguished in UML by subsetting or redefinition, as discussed below.

C. Properties with a defined range have the range class as their type in UML. Properties with no range have an anonymous class as their type in UML, as shown in Figure 33.
D. Properties with a range have the range class as their type in UML. Property types are shown to the right of the colon after the property name, as shown in Figure 30.

E. Two ways of representing RDF/S and OWL property subtyping (i.e., `rdfs:subPropertyOf`) use UML property/unidirectional association subsetting or association class subtyping. The UML semantics for both is that all links (instances, tuples) of the subtype properties or associations are links (instances, tuples) of all the supertypes properties or associations.

One option for property subsetting in UML is to use “{subsets <super-property-name>}” at the end of the property entry in a class, as shown in Figure 34.

![Figure 34 Property Subsetting, Notation on Property Entry for Class](image)

Alternatively, the representation given in Figure 35 may be used for unidirectional association subsetting.

![Figure 35 Property Subsetting, Unidirectional Association Representation](image)

For use with association classes, the representation shown in Figure 36, which uses a UML Generalization with the stereotype «rdfsSubPropertyOf», is preferred. Note that «rdfsSubPropertyOf» may not be required – it does not change the semantics, only adds constraints on its own usage.

![Figure 36 Property Subsetting, Association Class Representation](image)
14.1.7.2 RDFGlobalProperty

Description
An optional stereotype on a unidirectional association class or property with «rdfProperty» applied, indicating the association/property is defined globally, *i.e.* that class having the property, or on the nonnavigable end of the association, is the class on which the property/association is introduced, *i.e.*, the class does not inherit the property or association from a superclass.

Stereotype and Base Class
«rdfGlobal» with base class of UML::AssociationClass and UML::Property

Parent
«rdfProperty»

Properties
None.

Constraints
[1] The property being stereotyped must be on an anonymous class (or possibly on an instance of owl:Thing in OWL), or on an association class for a unidirectional association that has an anonymous class on the nonnavigable end.

14.1.7.3 RDFSsubPropertyOf

Description
rdfs:subPropertyOf is used to specialize RDF properties, similar to class generalization/specialization, and indicates that all the instances of the extension of the subproperty are instances of the extension of the super property. See above for further discussion and representation options, if used.

Stereotype and Base Class
«rdfsSubPropertyOf» with base class of UML::Generalization

Parent
None.

Properties
None.

Constraints
[1] Association classes on both ends of the generalization must be stereotyped «rdfProperty» (or «objectProperty» or «datatypeProperty»), if used with the profile for OWL.

[2] In OWL DL, an association class stereotyped by «objectProperty» or «datatypeProperty» may specialize a class stereotyped by «rdfProperty», but not vice versa.
14.1.8 Containers and Collections

The stereotypes associated with the definitions given in 10.6 (“RDFS Package, Containers and Collections”) are defined Appendix A (“Foundation Library (M1) for RDF and OWL”) including definition of container membership properties and lists.
14.2 UML Profile for OWL

This section specifies the UML profile for OWL. It is organized loosely on the structure of the OWL metamodel, with sections reordered to facilitate understanding and utility.

14.2.1 OWL Profile Package

The following sections specify the set of stereotypes, stereotype properties, and other elements that comprise the UML2 profile for OWL. As shown in Figure 37, the OWL profile package provides the container for the profile and imports the «rdf» profile.

![Figure 37 Web Ontology Language (OWL) Profile Package](image)

14.2.2 OWL Ontology

Description

An OWL ontology consists of zero or more optional ontology headers (typically at most one), which may include a set of ontology properties, such as `owl:Imports` statements, plus any number of class axioms, property axioms, and facts about individuals.

In a UML representation, we capture some of the header constructs by specializing «rdfDocument». Others are specified as ontology and annotation properties, defined below.

**Stereotype and Base Class**

«owlOntology» with base class of UML::Package

**Parent**

«rdfDocument»

**Properties**

None.

**Constraints**

[1] All classes (except association classes) in a package stereotyped by «owlOntology» must be stereotyped by «rdfsClass» or by «owlClass» (or an appropriate subclass).

[2] For applications intending to support OWL DL, all classes (except association classes) in a package stereotyped by «owlOntology» must be stereotyped by «owlClass» (or one of its subclasses).
14.2.3 OWL Annotation Properties

OWL annotation properties correspond, for the most part, to other stereotype properties defined in RDF or in this profile, although users may define their own.

14.2.3.1 OWLAnnotationProperty

Description

«owlAnnotation» represents the class of user-defined annotation properties in OWL (corresponding roughly to 11.4.3 (“OWLAnnotationProperty”). OWL annotations can be applied to any ontology element (e.g., ontologies, classes, properties, individuals).

Stereotype and Base Class

«owlAnnotation» with base class of UML::Property

Parent

«rdfProperty»

Properties

None.

Constraints

None.

14.2.3.2 owl:versionInfo

Description

An owl:versionInfo property generally has a string that provides information about the version of the element to which it is applied, for example RCS/CVS keywords. It does not contribute to the logical meaning of the ontology other than that given by the RDF(S) model theory.

Although typically used to make statements about ontologies, it may be applied to instance of any OWL construct. For example, one could apply an owl:versionInfo property to a class stereotyped by «owlClass», or to an instance of the RDFStatement class.

Stereotype and Base Class

No stereotype; implemented as a UML Property of the stereotype or model library class it describes.

- versionInfo: String [0..*] – the string containing the version information.

Parent

None.

Properties

None.

Constraints

None.
Graphical Representation
In the case of an ontology, with a package stereotyped by «owlOntology» or «rdfDocument», the normal stereotype notation can be used, with property values specified in braces under the stereotype label\(^3\), as shown in Figure 38 (“Representation for versionInfo Applied to an Ontology or RDF Document”).

Figure 38 Representation for versionInfo Applied to an Ontology or RDF Document

14.2.4 OWL Ontology Properties

OWL ontology properties are similar to annotation properties, in that they support annotations on OWL ontologies. The «owlOntologyProperty» stereotype can be applied to a property on a package stereotyped by «owlOntology» or «rdfDocument», and should be typed by another package that is similarly stereotyped.

OWL provides several built-in ontology properties, and also allows users to define such properties. Users can use some discretion in defining ontology properties, using either UML::Property or UML::Constraint as a base class, as appropriate.

14.2.4.1 owl:OntologyProperty

Description
owl:OntologyProperty represents the class of ontology properties in OWL, both built-in and user defined, corresponding to 11.4.6 (“OWL OntologyProperty”).

User-defined ontology properties are properties defined on the «owlOntology» or «rdfDocument» stereotypes, that can apply only between packages having these stereotypes.

Stereotype and Base Class
«owlOntologyProperty» with base class of UML::Property and UML::Constraint

Parent
None.

Properties
None.

Constraints

1. Applies only to properties of «owlOntology» or «rdfDocument».

2. Types of properties stereotyped by «owlOntologyProperty» must be stereotyped by «owlOntology» or «rdfDocument».

### 14.2.4.2 owl:backwardCompatibleWith

**Description**

`owl:backwardCompatibleWith` refers to another ontology, and identifies the specified ontology as a prior version, and further indicates that it is backward compatible with it.

**Stereotype and Base Class**

«backwardCompatibleWith» with base class of UML::Constraint

**Parent**

None.

**Properties**

None.

**Constraints**

1. Applies only to constraints between packages stereotyped by «owlOntology» or «rdfDocument».
2. Classes and properties in the new version that have the same name as classes and properties in the earlier version must either be equivalent to or extend those in the earlier versions.
3. The later version must be logically consistent with the earlier version.
4. (Semantic) Identifiers in the later version have the same interpretation in the earlier version.

**Graphical Representation**

Dashed line between two instances with stereotype label, arrowhead towards the earlier version, as shown in Figure 39 (“Stereotype Representation for owl:backwardCompatibleWith”).

![Figure 39 Stereotype Representation for owl:backwardCompatibleWith](image)

### 14.2.4.3 owl:imports

**Description**

`owl:imports` 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.

**Stereotype and Base Class**

«owlImports» with base class of UML::PackageImports
**Parent**
None.

**Properties**
None.

**Constraints**
[1] Applies only to imports between packages stereotyped by «owlOntology» or «rdfDocument».

**Graphical Representation**
Dashed line between two instances with stereotype label, arrowhead towards the imported ontology, as shown in Figure 40 (“Stereotype Representation for owl:imports”).

![Figure 40](image-url)

**14.2.4.4 owl:incompatibleWith**

**Description**
owl:incompatibleWith indicates that the containing ontology is a later version of the referenced ontology, but is not necessarily backward compatible with it. Essentially, this allows ontology authors to specify that a document cannot be upgraded without verifying consistency with the specified ontology.

**Stereotype and Base Class**
«incompatibleWith» with base class of UML::Constraint

**Parent**
None.

**Properties**
None.

**Constraints**
[1] Applies only to constraints between packages stereotyped by «owlOntology» or «rdfDocument».

**Graphical Representation**
Dashed line between two instances with stereotype label, arrowhead towards the earlier version, as shown in Figure 41 (“Stereotype Representation for owl:incompatibleWith”).

![Figure 41](image-url)
Note: While it might seem reasonable to eliminate the arrowhead in this case, and make the relationship bi-directional, all RDF graphs and thus such relationships are unidirectional in RDF, RDF Schema and OWL. Applications that leverage this representation may optionally allow the user to indicate that they want a particular instance of «incompatibleWith» to be bidirectional, eliminate the arrowhead and use a single dashed line; the interpretation of such notation should be two instances of «incompatibleWith», however.

14.2.4.5 owl:priorVersion

Description

owl:priorVersion 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.

Because of the lack of semantics, there is no obvious UML element to reuse or stereotype for this particular OWL property. However, assuming that the spirit of this property is similar to though not quite as strong as that of «backwardCompatibleWith», the same base class is used.

Stereotype and Base Class

«priorVersion» with base class of UML::Constraint

Parent

None.

Properties

None.

Constraints

[1] Applies only to constraints between packages stereotyped by «owlOntology» or «rdfDocument».

Graphical Representation

Dashed line between two instances with stereotype label, arrowhead towards the earlier version, as shown in Figure 42 (“Stereotype Representation for owl:priorVersion”).
 Classes provide an abstraction mechanism for identifying the common characteristics among a group of resources. 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, (e.g., classes representing “the morning star” and “the evening star”).

A class description is the term used in [OWL S&AS] for the basic building blocks of class axioms. 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 typically describe an anonymous class by placing constraints on the class extension. They consist of a set of RDF triples in which a blank node represents the class being described. This blank node has a type property whose value is «owlClass». Note that multiple class descriptions can be applied to the same class, however, such that these anonymous classes can ultimately also be named.

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. Stereotypes for OWL class descriptions are given below.

### 14.2.5.1 OWLClass

**Description**

owl:Class describes a class through a class name, and corresponds to 11.3.5 (“OWLClass”).
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.

**Stereotype and Base Class**

«owlClass» with base class of UML::Class

**Properties**

- `isDeprecated`: Boolean [0..1] – provides an additional annotation that indicates a particular class definition is deprecated.

**Constraints**

None.

### 14.2.5.2 EnumeratedClass

**Description**

«enumeratedClass» describes a class by enumerating the set of individuals, or UML instance specifications, that are members of the class, and corresponds to 11.3.2 (“EnumeratedClass”).

**Stereotype and Base Class**

«enumeratedClass» with base class of UML::Class

**Parent**

«owlClass»

**Properties**

- `isComplete`: Boolean [0..1] – indicates whether the set of enumerated individuals is complete, meaning, that this provides a complete specification for the class.

**Constraints**

None.

### 14.2.5.3 RestrictionClass

**Description**

`owl:Restriction` reifies a special kind of class. The restriction class is a subtype of the domain of the restricted property, and identifies a class specifying exactly the necessary and sufficient conditions that make a particular individual a member of that class. It describes an anonymous class, namely a class of all individuals that satisfy the restriction. It can be used with other classes in a number of ways, and when paired with exactly one other class, will be either a supertype, a subtype, or equivalent (necessary, sufficient, or both). When used in another class, the restriction class is effectively a supertype of the containing class, applying the restriction to all individuals of the containing class.

OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints. Property restrictions
can be applied both to datatype properties (properties for which the value is a data literal) and object properties (props-
erties for which the value is an individual).

Note: although restriction classes are typically anonymous, they are not required to be and can be named (via a class ID
URI reference/name).

**Stereotype and Base Class**

«owlRestriction» with base class of UML::Class

**Parent**

«owlClass»

**Properties**

- onProperty: Property [1] – identifies the property to which the restriction applies.

**Constraints**

| Editorial Comment: Issue 10841 |

[1] (Semantic) Instances of the class are all and only those instances satisfying the constraint restriction.

### 14.2.5.4 Cardinality Constraints

**Description**

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: owl:maxCardinality,
owl:minCardinality, and owl:Cardinality. These constructs are analogous to multiplicity in UML, thus the approach
taken is

- for properties whose initial definition includes the cardinality constraint, simply apply multiplicities as appropri-
ate.
- for inherited properties, redefine the property with new multiplicity.

Value specifications for multiplicities in OWL must be non-negative integer literals. Additionally, isOrdered = false on
OWL properties, and isUnique = true on OWL properties, meaning that the values are a set, not a a bug.

**Stereotype and Base Class**

None. UML multiplicities are presented using the standard presentation options defined in section 7.3.32, “Unified
Modelling Language: Superstructure”, version 2 [UML2].

**Parent**

None.

**Properties**

None.
Constraints

None.

Graphical Representation
For inherited properties, show the property with restricted multiplicity in subtype, and using “{redefines <restricted-property-name> }” at the end of the property entry in a class (can be elided), as shown in Figure 43.

Figure 43 owl:Cardinality - Restricted Multiplicity in Subtype

Alternatively, when unidirectional associations are desirable, cardinality constraints can be represented as shown in Figure 44.

Figure 44 owl:Cardinality - Restricted Multiplicity in Subtype

14.2.5.5 owl:allValuesFrom Constraint

Description
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 owl:allValuesFrom specifies a class or data range for which all values of the property under consideration are either members of the described class, or are data values within the specified data range.

Essentially, owl:allValuesFrom is used to redefine the type of a particular property. In effect, this constraint defines a subproperty similar to UML property redefinition.

Stereotype and Base Class
None. Uses UML Generalization and property redefinition, as shown under Graphical Representation, below.

Note that the domain and/or target (for owl:allValuesFrom) for the subproperty will not always be a direct descendent of the superclass that the property is defined on, as it happens to be in the examples.
If the attribute form of representation is used, then “{redefines <parent-class>::<property-name>}” should be given at the end (i.e., to the right) of the property entry. The parent class is optional if the property inherits from only one parent.

**Parent**
None.

**Properties**
None.

**Constraints**

1. Property name is not changed in redefinition.
2. The redefined child class (or data range) must be stereotyped «owlClass» (or «owlDataRange»).

**Graphical Representation**

Several representation approaches are provided here, in keeping with the representation used for properties in the profile for RDF/S. First, we can show the property with restricted type in subtype, by adding “{redefines <restricted-property-name>}” at the end of the property entry (can be elided), as in Figure 45.

![Figure 45 Simple Property Redefinition Example For owl:allValuesFrom](image1)

Secondly, we can show the same thing using unidirectional association style properties, as shown in Figure 46.

![Figure 46 Property Redefinition For owl:allValuesFrom With Unidirectional Associations](image2)

An alternative using association classes is shown below.
14.2.5.6 owl:someValuesFrom and owl:hasValue Constraints

Description

Similar to owl:allValuesFrom, owl:someValuesFrom 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:someValuesFrom constraint is used to describe a class or data range for which at least one value of the property concerned is either a member of the class extension of the class description or a data value within the specified 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 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 an 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).

Again, like owl:allValuesFrom, owl:someValuesFrom and owl:hasValue are used to redefine the type of a particular property, similar to UML property redefinition.

Stereotype and Base Class

A stereotype «owlValue» with base class of UML::Property is applied to properties that are redefined. The stereotype has these properties.

Parent

None.

Properties

- hasValue: InstanceSpecification [0..*] – identifies the individual value(s) or data value(s)
- someValuesFrom: Class [0..1] – identifies a class stereotyped by «owlClass» or «owlDataRange»
Constraints

[1] Can be applied to properties stereotyped by «rdfProperty», «objectProperty», «datatypeProperty» or any of their children, but only to properties that redefine other properties.

[2] The value of the someValuesFrom property must be stereotyped «owlClass» or «owlDataRange».

Graphical Representation

Put before the property name: “«owlValue» {hasValue = <instance-name>, <instance-name>; someValuesFrom = <class-name>, <class-name>}”, for example, as shown in Figure 48, where volume is an individual of type physical dimension4.

![Figure 48 Example Using owl:hasValue Constraint](image)

14.2.5.7 owl:intersectionOf Class Description

Description

owl:intersectionOf links a class to a list of class descriptions, describing an anonymous class for which the class extension contains precisely those individuals that are members of the class extension of all class descriptions in the list. «intersectionOf» is analogous to logical conjunction.

Stereotype and Base Class

«intersectionOf» with base class of UML::Constraint

Parent

None.

Properties

None.

Constraints

[1] Applies to generalizations with a common subtype.

---

[2] All instances that are instances of every super type along generalizations that are stereotyped by «intersectionOf» are instances of the subtype.

[3] (Semantic) All instances of the subtype are instances of all of the super types.

**Graphical Representation**

Dashed line between generalization lines with stereotype label, as shown in Figure 49.

![Figure 49 Example Using owl:intersectionOf](image)

Figure 49 Example Using owl:intersectionOf

The stereotype is based on UML::Generalization rather than UML::Class, so there can be other supertypes not required by the intersection. Use of UML::GeneralizationSet was prohibited in this case, because it requires one supertype – its semantics refers to the instances of the subtypes, not the supertypes.

### 14.2.5.8 owl:unionOf Class Description

**Description**

owl:unionOf links a class to a list of class descriptions, describing 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. owl:unionOf is analogous to logical disjunction.

**Stereotype and Base Class**

No stereotype needed. Use UML::GeneralizationSet with isCovering = true, as shown in Figure 50. For consistency with the other class descriptions, vendors can also optionally define a «unionOf» stereotype of UML::Constraint, applied to UML::Generalization (similar to intersection, above).

**Parent**

None.

**Properties**

None.

**Constraints**

1. (Semantic) All instances of the supertype are instances of at least one of the subtypes.

**Graphical Representation**

Dashed line between generalization lines labeled with “{complete}”.

Ontology Definition Metamodel
14.2.5.9 owl:complementOf Class Description

Description

owl:complementOf links a class to precisely one other class, and describes a class for which the extension contains exactly those individuals that do not belong to the extension of the other class. owl:complementOf is analogous to logical negation: the class extension consists of those individuals that are NOT members of the extension of the complement class.

Stereotype and Base Class

«complementOf» with base class of UML::Constraint.

Parent

None.

Properties

None.

Constraints

[1] Applies to constraints between exactly two classes.

[2] (Semantic). All instances (of owl:Thing) are instances of exactly one of the two classes.

Graphical Representation

Dashed line between two classes with stereotype label. An arrowhead should be used opposite from the class that will have owl:complementOf in XML syntax (since all RDF, RDF Schema, and OWL graphs are unidirectional, by definition). Shorthand representation that eliminates the arrowhead may be used within an ontology, but XML production in this case should result in two instances of owl:complementOf — one for each “side” of the bidirectional constraint.
14.2.5.10 owl:disjointWith Class Axiom

Description

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

Stereotype and Base Class

«disjointWith» with base class of UML::Constraint, or use disjoint UML generalizations with no stereotype.

Parent

None.

Properties

None.

Constraints

[1] Applies only to constraints between classes.

[2] (Semantic). An individual can only be a member of one class participating in a particular disjoint set of
classes.

Graphical Representation

Dashed line between two classes with stereotype label. An arrowhead should be used opposite from the class that will
have «disjointWith» in XML syntax (since all RDF, RDF Schema, and OWL graphs are unidirectional, by definition).
Shorthand representation that eliminates the arrowhead may be used within an ontology, but XML production in this
case should result in two instances of «disjointWith» – one for each “side” of the bidirectional constraint.

Figure 52 Example Using owl:disjointWith

In cases where there are multiple participants in the same «disjointWith» class axiom, a constraint note with
stereotype label and dashed lines to more than one class should be used, as shown in Figure 53.
Alternatively, if the classes have a common supertype, use UML::GeneralizationSet with isDisjoint = true. Representation is dashed line between generalization lines labeled with “{disjoint}”.

14.2.5.11owl:equivalentClass Class Axiom

Description
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). A class axiom may contain (multiple) owl:equivalentClass statements.

Stereotype and Base Class
«equivalentClass» with base class of UML::Constraint.

Parent
None.

Properties
None.

Constraints
[1] Applies only to constraints between classes.
Graphical Representation

Dashed line between two classes with stereotype label. An arrowhead should be used opposite from the class that will have owl:equivalentClass in XML syntax. Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production in this case should result in two instances of «equivalentClass» – one for each “side” of the bidirectional constraint.

Alternatively two UML::Generalizations may be used, again within a given ontology, if such circular definitions are supported by the tool (i.e., class a generalizes class b and vice versa).

In cases where there are multiple participants in the same «equivalentClass» class axiom, a constraint note with stereotype label and dashed lines to more than one class should be used, similarly to the example used for «disjointWith».

14.2.6 Properties

OWL distinguishes between two main categories of properties:

- 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).

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

The relevant RDF Schema concepts are defined in 14.1.7 (“Properties in RDF”); global cardinality constraints and logical property characteristics are represented as UML properties on either «owlProperty» or «objectProperty», as given below.

14.2.6.1 owl:DatatypeProperty

Description

A datatype property is defined as an instance of the built-in OWL class owl:DatatypeProperty. The built-in class, owl:DatatypeProperty, is a subclass of the built-in class rdf:Property.
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, \texttt{owl:ObjectProperty} is equivalent to \texttt{rdf:Property}. In practice, this mainly has consequences for the use of \texttt{owl:InverseFunctionalProperty}.

**Stereotype and Base Class**

«datatypeProperty» with base class of UML::AssociationClass and UML::Property.

**Parent**

«owlProperty»

**Properties**

None.

**Constraints**

1. (Semantics) The values of a property stereotyped by «datatypeProperty» must be either strings that can be represented as UML::LiteralString stereotyped by «rdfsLiteral», values corresponding to the enumerated literals represented as a UML::Enumeration stereotyped by «dataRange», or instances of a UML::Class stereotyped by «rdfsDatatype» (i.e., one of the XML Schema Datatypes or a built-in datatype).

### 14.2.6.2 owl:ObjectProperty

**Description**

An object property is defined as an instance of the built-in OWL class \texttt{owl:ObjectProperty}. The built-in class, \texttt{owl:ObjectProperty}, is a subclass of the built-in class \texttt{rdf:Property}.

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 \texttt{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 \( x_1 \) and \( x_2 \) such that both pairs \((x_1, y)\) and \((x_2, y)\) are instances of \( P \). See section 4.3.2 of [OWL Reference] for additional detail, including an explanation of the notion of global cardinality constraints and use of \texttt{owl:InverseFunctionalProperty} to represent keys in the context of a relational database.

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 \( P \).

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 pair \((x, z)\) is also an instance of \( P \).

**Stereotype and Base Class**

«objectProperty» with base class of UML::AssociationClass or UML::Property.

**Parent**

«owlProperty»

**Properties**

- isInverseFunctional: Boolean \([0..1]\) – when true, indicates that the property in question is inverse functional.

- isSymmetric: Boolean \([0..1]\) – when true, indicates that the property in question is symmetric.
• isTransitive: Boolean [0..1] – when true, indicates that the property in question is transitive.

Constraints

[1] The type of a property stereotyped by «objectProperty» must be a UML::Class stereotyped by «owlClass».

[2] In OWL Full, the isInverseFunctional, isSymmetric, and isTransitive properties apply only to properties stereotyped by «owlProperty», «objectProperty», or «datatypeProperty».

[3] In OWL DL, the isInverseFunctional, isSymmetric, and isTransitive properties apply only to properties stereotyped by «objectProperty».

[4] The type of a property with isSymmetric set to true must be the same as the class on which it is defined.

[2] In OWL DL, no local or global cardinality constraints can be declared on a property with isTransitive set to true, or on any of its super properties, nor on its inverse or on any super properties of its inverse.

14.2.6.3 owl:Property

Description

The notion of an owl:Property, as defined in the metamodel and redefined here in the profile is an abstract class.

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”, thus, we introduce it at owl:Property.

Stereotype and Base Class

«owlProperty» with base class of UML::AssociationClass or UML::Property.

Parent

«rdfProperty»

Properties

• isDeprecated: Boolean [0..1] – indicates a particular property definition is deprecated.

• isFunctional: Boolean [0..1] – when true, indicates that the property in question is functional.

Constraints

[1] The isFunctional property applies only to properties stereotyped by «owlProperty», «objectProperty», or «datatypeProperty».

14.2.6.4 owl:equivalentProperty Relation

Description

owl:equivalentProperty 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 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.
Stereotype and Base Class
«equivalentProperty» stereotype of UML::Constraint between classes stereotyped as «rdfProperty», «owlProperty», «objectProperty», or «datatypeProperty».

Parent
None.

Properties
None.

Constraints
[1] Applies only to constraints between properties with «rdfGlobal» applied, or properties on the class at which they are introduced.

[2] (Semantic) Instances of equivalent properties (property extensions, or sets of tuples) are the same.

Graphical Representation
Dashed line between two association classes with stereotype label. An arrowhead should be used opposite from the association class that will have «equivalentProperty» in XML syntax. Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production should result in two instances of «equivalentProperty» – one for each “side” of the bidirectional constraint.

In cases where there are multiple participants in the same «equivalentProperty» relation, a constraint note with stereotype label and dashed lines to more than one association class representing the property should be used, similarly to the example for «disjointWith».

14.2.6.5 owl:inverseOf Relation

Description
OWL 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. owl:inverseOf 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 OWL 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.

Stereotype and Base Class
«inverseOf» with base class of UML::Association, or use bidirectional associations with no stereotype

Parent
None.

Properties
None.

Constraints
[1] Applies only to associations with «rdfGlobal» applied, or to properties on the class at which they are introduced.
[2] Applies only to binary, unidirectional associations.
[3] (UML) A property cannot be an inverse of itself (use the isSymmetric property).

**Graphical Representation**

We propose several options for modeling/representing inverses in UML.

A. The first is to use a simple association with properties as ends, *i.e.*, a line between classes with properties on the ends closest to their ranges, for example, as shown in Figure 56.

![Figure 56 Using owl:inverseOf With Bidirectional Representation](image)

Additional constraint if this approach is taken:

[4] (UML) A property can have at most one inverse.

B. Alternatively, one could use an «inverseOf» stereotype of UML::Constraint between association classes for binary, unidirectional associations, as shown in Figure 57. An arrowhead should be used opposite from the association class that will have `owl:inverseOf` in XML syntax. Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production should result in two instances of «inverseOf» – one for each “side” of the bidirectional constraint.

![Figure 57 Using owl:inverseOf Between Association Classes](image)

C. A third notational option would be to use a stereotype «inverse» of a UML::Property with a property:

- OF of type UML::Property

Using a similar representation to the approach taken in 14.2.5.6 (“owl:someValuesFrom and owl:hasValue Constraints”), put before the property name: “«inverse» {of = <property-name>, <property-name>}”.

Additional constraint if this approach is taken:

[5] Value of OF property must refer to a property with «rdfGlobal» applied, or to properties on the class at which they were introduced.
14.2.7 Individuals

Individuals are defined with individual axioms (also called “facts”). These include:

- Facts about class membership and property values of individuals
- Facts about 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.

14.2.7.1 Class Membership and Property Values of Individuals

Description

Many facts typically are statements indicating class membership of individuals and property values of individuals. Individual axioms need not necessarily be about named individuals: they can also refer to anonymous individuals.

Stereotype and Base Class

No stereotype, use UML::InstanceSpecification typed by a class having the properties desired for the individual. The class may be stereotyped by «singleton» to indicate it is for a specific individual. Classes stereotyped by «singleton» are not translated to OWL, and their properties appear in OWL as properties of the individual.

Parent

None.

Properties

None.

Constraints

[1] Classes stereotyped by «singleton» have exactly one instance each.

Graphical Representation

Instance specifications use the same symbol as classes, but their names are underlined, and have a colon separating the instance name from the class name. Singleton classes can be anonymous, omitted from the representation, and generated by tools. Instances of anonymous classes show nothing after the colon.

5. UML supports individuals without classes and properties on such individuals, for tools that choose to support it.
14.2.7.2 owl:sameAs Relation

Description

owl:sameAs links an individual to an individual, indicating that two URI references actually refer to the same thing: the individuals have the same “identity”. owl:sameAs statements are often used in defining mappings between ontologies.

Additionally, in OWL Full, where a class can be treated as instances of (meta)classes, owl:sameAs can be used to define class equality, thus indicating that two concepts have the same intensional meaning.

Stereotype and Base Class

«sameAs» with base class of UML::Constraint.

Parent

None.

Properties

None.

Constraints

[1] Applies only to constraints between instance specifications, or for modeling OWL Full, between instances or between classes.

Graphical Representation

Dashed line between two instances (or classes) with stereotype label. An arrowhead can be used opposite from the instance (or class) that will have «sameAs» in XML syntax.

Constraint note with stereotype label and dashed lines to more than one instance (or class - translates to multiple «sameAs» statements).

Figure 58 Using owl:sameAs Between Instances

Figure 59 Using owl:sameAs Between Instances
14.2.7.3 owl:differentFrom Relation

Description
The built-in owl:differentFrom property links an individual to an individual., indicating that two URI references refer to different individuals.

Stereotype and Base Class
«differentFrom» with base class of UML::Constraint.

Parent
None.

Properties
None.

Constraints
[1] Applies only to constraints between instance specifications.

Graphical Representation
Dashed line between two instances with stereotype label. An arrowhead can be used opposite from the instance that will have «differentFrom» in XML syntax.

![Figure 60 Using owl:differentFrom Between Instances](image)

Constraint note with stereotype label and dashed lines to more than one instance (translates to multiple «differentFrom» statements).

![Figure 61 Using owl:differentFrom Between Instances](image)

14.2.7.4 owl:AllDifferent Construct

Description
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 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.

**Stereotype and Base Class**

«allDifferent» with base class of UML::Constraint.

**Parent**

None.

**Properties**

None.

**Constraints**

1. Applies only to constraints between instance specifications.

**Graphical Representation**

Constraint note with stereotype label and dashed lines to more than one instance.

![Diagram of allDifferent constraint](image)

**14.2.7.5 Individual Property Values**

**Description**

In RDF, RDF Schema, and OWL, properties of individuals are accessed essentially through the triples (or statements), where the individual is the subject of the triple. In this profile, while we have optionally provided explicit access to the elements of the triple in a way that identifies the subject for this purpose, we also provide a more intuitive representation from a UML perspective.

**Stereotype and Base Class**

No stereotype, use UML::Slot to represent properties on individuals.

**Parent**

None.

**Properties**

None.
Constraints
[1] Values must conform to constraints on the property, such as type and multiplicity.

Graphical Representation
Put values after equal sign at end of property entry in instance.

14.2.8 Datatypes

OWL allows three types of data range specifications:

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

OWL makes use of the RDF datatyping scheme, which provides a mechanism for referring to XML Schema datatypes. Data values are instances of the RDF Schema class rdfs:Literal. Datatypes are instances of the class rdfs:Datatype.

The RDF Semantics document recommends use of a subset of the simple built-in XML Schema datatypes. The set of XML Schema datatypes that are allowable for use in OWL DL are given in the model library provided in Appendix A, Foundation Library (M1) for RDF and OWL.

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”.

14.2.8.1 Enumerated Data Values

Description
In addition to the RDF datatypes, OWL provides one additional construct for defining a range of data values, namely an enumerated datatype, where the enumerated values are the enumeration literals (a kind of instance specification) of the enumeration.

In OWL, 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.

Stereotype and Base Class
«dataRange» with base class of UML::Enumeration

Parent
None.

Properties
None.

Constraints
None.
Graphical Representation

Use UML enumeration notation.