W3C Recommendation

OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)

W3C Recommendation 11 December 2012

This version:
http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/
Latest version (series 2):
http://www.w3.org/TR/owl2-mapping-to-rdf/
Latest Recommendation:
http://www.w3.org/TR/owl-mapping-to-rdf
Previous version:
http://www.w3.org/TR/2012/PER-owl2-mapping-to-rdf-20121018/
Editors:
Peter F. Patel-Schneider, Nuance Communications
Boris Motik, University of Oxford
Contributors: (in alphabetical order)
Bernardo Cuenca Grau, University of Oxford
Ian Horrocks, University of Oxford
Bijan Parsia, University of Manchester
Alan Ruttenberg, Science Commons (Creative Commons)
Michael Schneider, FZI Research Center for Information Technology

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

A color-coded version of this document showing changes made since the previous version is also available.

This document is also available in these non-normative formats: PDF version.

See also translations.

Abstract

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents. The OWL 2 Document Overview describes the overall state of OWL 2, and should be read before other OWL 2 documents.

This document defines the mapping of OWL 2 ontologies into RDF graphs, and vice versa.

W3C Recommendation

Status of this Document

May Be Superseded

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/.

Summary of Changes

There have been no substantive changes since the previous version. For details on the minor changes see the change log and color-coded diff.

Please Send Comments

Please send any comments to public-owl-comments@w3.org (public archive). Although work on this document by the OWL Working Group is complete, comments may be addressed in the errata or in future revisions. Open discussion among developers is welcome at public-owl-dev@w3.org (public archive).

Endorsed By W3C

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. 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.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent.

Table of Contents

W3C Recommendation

1 Introduction and Preliminaries

This document defines two mappings between the structural specification of OWL 2 [*OWL 2 Specification*] and RDF graphs [*RDF Concepts*]. The mapping presented in Section 2 can be used to transform any OWL 2 ontology *O* into an RDF graph *T(O)*. The mapping presented in Section 3 can be used to transform an RDF graph *G* satisfying certain restrictions into an OWL 2 DL ontology O_G. These transformations do not incur any change in the formal meaning of the ontology. More precisely, for any OWL 2 DL ontology *O*, let *G = T(O)* be the RDF graph obtained by transforming *O* as specified in Section 2, and let O_G be the OWL 2 DL ontology obtained by applying the reverse transformation from Section 3 to *G*; then, *O* and O_G are logically equivalent — that is, they have exactly the same set of models.

The mappings presented in this document are backwards-compatible with that of OWL 1 DL: every OWL 1 DL ontology encoded as an RDF graph can be mapped into a valid OWL 2 DL ontology using the mapping from Section 3 such that the resulting OWL 2 DL ontology has exactly the same set of models as the original OWL 1 DL ontology.

The syntax for triples used in this document is the one used in the RDF Semantics [*RDF Semantics*]. Full IRIs are abbreviated using the prefixes from the OWL 2 Specification [*OWL 2 Specification*]. OWL 2 ontologies mentioned in this document should be understood as instances of the structural specification of OWL 2 [*OWL 2 Specification*]; when required, these are written in this document using the functional-style syntax.

The following notation is used throughout this document for referring to parts of RDF graphs:

- `*:x` denotes an IRI;
- `_:x` denotes a blank node;
- `x` denotes a blank node or an IRI;
- `lt` denotes a literal; and
- `xlt` denotes a blank node, an IRI, or a literal.

The italicized keywords *must*, *must not*, *should*, *should not*, and *may* are used to specify normative features of OWL 2 documents and tools, and are interpreted as specified in RFC 2119 [*RFC 2119*].

W3C Recommendation

2 Mapping from the Structural Specification to RDF Graphs

This section defines a mapping of an OWL 2 ontology *O* into an RDF graph *T(O)*. The mapping is presented in three parts. Section 2.1 shows how to translate axioms that do not contain annotations, Section 2.2 shows how to translate annotations, and Section 2.3 shows how to translate axioms containing annotations.

2.1 Translation of Axioms without Annotations

Table 1 presents the operator *T* that maps an OWL 2 ontology *O* into an RDF graph *T(O)*, provided that no axiom in *O* is annotated. The mapping is defined recursively; that is, the mapping of a construct often depends on the mappings of its subconstructs, but in a slightly unusual way: if the mapping of a construct refers to the mapping of a subconstruct, then the triples generated by the recursive invocation of the mapping on the subconstruct are added to the graph under construction, and the *main node* of the mapping of the subconstruct is used in place of the recursive invocation itself.

The definition of the operator *T* uses the operator *TANN* in order to translate annotations. The operator *TANN* is defined in Section 2.2. It takes an annotation and an IRI or a blank node and produces the triples that attach the annotation to the supplied object.

In the mapping, each generated blank node (i.e., each blank node that does not correspond to an anonymous individual) is fresh in each application of a mapping rule. Furthermore, possible conditions on the mapping rules are enclosed in curly braces '`{ }`'. Finally, the following conventions are used in this section to denote different parts of OWL 2 ontologies:

- `OP` denotes an object property;
- `OPE` denotes an object property expression;
- `DP` denotes a data property;
- `DPE` denotes a data property expression;
- `AP` denotes an annotation property;
- `C` denotes a class;
- `CE` denotes a class expression;
- `DT` denotes a datatype;
- `DR` denotes a data range;
- `U` denotes an IRI;
- `F` denotes a constraining facet;
- `a` denotes an individual (named or anonymous);
- `*:a` denotes a named individual;
- `lt` denotes a literal;
- `as` denotes an annotation source; and
- `av` denotes an annotation value.

In this section, T(SEQ y_1 ... y_n) denotes the translation of a sequence of objects from the structural specification into an RDF list, as shown in Table 1.

Table 1. Transformation to Triples

Element *E* of the Structural Specification	**Triples Generated in an Invocation of *T(E)***	**Main Node of *T(E)***
SEQ		*rdf:nil*

SEQ y_1 ... y_n	_:x *rdf:first* T(y_1) . _:x *rdf:rest* T(SEQ y_2 ... y_n) .	_:x
Ontology(ontologyIRI [versionIRI] Import(importedOntologyIRI$_1$) ... Import(importedOntologyIRI$_k$) annotation$_1$ ... annotation$_m$ axiom$_1$ ... axiom$_n$)	ontologyIRI *rdf:type* *owl:Ontology* . [ontologyIRI *owl:versionIRI* versionIRI] . ontologyIRI *owl:imports* importedOntologyIRI$_1$ . ... ontologyIRI *owl:imports* importedOntologyIRI$_k$. TANN(annotation$_1$, ontologyIRI) . ... TANN(annotation$_m$, ontologyIRI) . T(axiom$_1$) . ... T(axiom$_n$) .	ontologyIRI
Ontology(Import(importedOntologyIRI$_1$) ... Import(importedOntologyIRI$_k$) annotation$_1$ ... annotation$_m$ axiom$_1$ ... axiom$_n$)	_:x *rdf:type* *owl:Ontology* . _:x *owl:imports* importedOntologyIRI$_1$ . ... _:x *owl:imports* importedOntologyIRI$_k$. TANN(annotation$_1$, _:x) . ... TANN(annotation$_m$, _:x) . T(axiom$_1$) . ... T(axiom$_n$) .	_:x
C		C
DT		DT
OP		OP
DP		DP
AP		AP
U		U
a		a
"abc@"^^*rdf:PlainLiteral*		"abc"
"abc@langTag"^^*rdf:PlainLiteral*		"abc"@langTag
lt { where lt is a literal of		lt

W3C Recommendation

datatype other than *rdf:PlainLiteral* }		
Declaration(Datatype(DT))	T(DT) *rdf:type rdfs:Datatype* .	
Declaration(Class(C))	T(C) *rdf:type owl:Class* .	
Declaration(ObjectProperty(OP))	T(OP) *rdf:type owl:ObjectProperty* .	
Declaration(DataProperty(DP))	T(DP) *rdf:type owl:DatatypeProperty* .	
Declaration(AnnotationProperty(AP))	T(AP) *rdf:type owl:AnnotationProperty* .	
Declaration(NamedIndividual(*:a))	T(*:a) *rdf:type owl:NamedIndividual* .	
ObjectInverseOf(OP)	_:x *owl:inverseOf* T(OP) .	_:x
DataIntersectionOf(DR_1 ... DR_n)	_:x *rdf:type rdfs:Datatype* . _:x *owl:intersectionOf* T(SEQ DR_1 ... DR_n) .	_:x
DataUnionOf(DR_1 ... DR_n)	_:x *rdf:type rdfs:Datatype* . _:x *owl:unionOf* T(SEQ DR_1 ... DR_n) .	_:x
DataComplementOf(DR)	_:x *rdf:type rdfs:Datatype* . _:x *owl:datatypeComplementOf* T(DR) .	_:x
DataOneOf(lt_1 ... lt_n)	_:x *rdf:type rdfs:Datatype* . _:x *owl:oneOf* T(SEQ lt_1 ... lt_n) .	_:x

DatatypeRestriction(DT
    F_1 lt_1
    ...
    F_n lt_n
)
	_:x *rdf:type rdfs:Datatype* .
	_:x *owl:onDatatype* T(DT) .
	_:x *owl:withRestrictions* T(SEQ _:y_1 ... _:y_n) .
	_:y_1 F_1 lt_1 .
	...
	_:y_n F_n lt_n .
		_:x

ObjectIntersectionOf(CE_1 ... CE_n)	_:x *rdf:type owl:Class* . _:x *owl:intersectionOf* T(SEQ CE_1 ... CE_n) .	_:x
ObjectUnionOf(CE_1 ... CE_n)	_:x *rdf:type owl:Class* . _:x *owl:unionOf* T(SEQ CE_1 ... CE_n) .	_:x
ObjectComplementOf(CE)	_:x *rdf:type owl:Class* . _:x *owl:complementOf* T(CE) .	_:x

ObjectOneOf(a_1 ... a_n)	_:x rdf:type owl:Class . _:x owl:oneOf T(SEQ a_1 ... a_n) .	_:x
ObjectSomeValuesFrom(OPE CE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:someValuesFrom T(CE) .	_:x
ObjectAllValuesFrom(OPE CE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:allValuesFrom T(CE) .	_:x
ObjectHasValue(OPE a)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:hasValue T(a) .	_:x
ObjectHasSelf(OPE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:hasSelf "true"^^xsd:boolean .	_:x
ObjectMinCardinality(n OPE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:minCardinality "n"^^xsd:nonNegativeInteger .	_:x
ObjectMinCardinality(n OPE CE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:minQualifiedCardinality "n"^^xsd:nonNegativeInteger . _:x owl:onClass T(CE) .	_:x
ObjectMaxCardinality(n OPE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:maxCardinality "n"^^xsd:nonNegativeInteger .	_:x
ObjectMaxCardinality(n OPE CE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) . _:x owl:maxQualifiedCardinality "n"^^xsd:nonNegativeInteger . _:x owl:onClass T(CE) .	_:x
ObjectExactCardinality(n OPE)	_:x rdf:type owl:Restriction . _:x owl:onProperty T(OPE) .	_:x

	_:x *owl:cardinality* "n"^^*xsd:nonNegativeInteger* .	
ObjectExactCardinality(n OPE CE)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(OPE) . _:x *owl:qualifiedCardinality* "n"^^*xsd:nonNegativeInteger* . _:x *owl:onClass* T(CE) .	_:x
DataSomeValuesFrom(DPE DR)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:someValuesFrom* T(DR) .	_:x
DataSomeValuesFrom(DPE_1 ... DPE_n DR), n ≥ 2	_:x *rdf:type owl:Restriction* . _:x *owl:onProperties* T(SEQ DPE_1 ... DPE_n) . _:x *owl:someValuesFrom* T(DR) .	_:x
DataAllValuesFrom(DPE DR)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:allValuesFrom* T(DR) .	_:x
DataAllValuesFrom(DPE_1 ... DPE_n DR), n ≥ 2	_:x *rdf:type owl:Restriction* . _:x *owl:onProperties* T(SEQ DPE_1 ... DPE_n) . _:x *owl:allValuesFrom* T(DR) .	_:x
DataHasValue(DPE lt)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:hasValue* T(lt) .	_:x
DataMinCardinality(n DPE)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:minCardinality* "n"^^*xsd:nonNegativeInteger* .	_:x
DataMinCardinality(n DPE DR)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:minQualifiedCardinality* "n"^^*xsd:nonNegativeInteger* . _:x *owl:onDataRange* T(DR) .	_:x
DataMaxCardinality(n DPE)	_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* T(DPE) .	_:x

W3C Recommendation

	_:x *owl:maxCardinality* "n"^^*xsd:nonNegativeInteger* .	
DataMaxCardinality(n DPE DR)	_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:maxQualifiedCardinality* "n"^^*xsd:nonNegativeInteger* . _:x *owl:onDataRange* T(DR) .	_:x
DataExactCardinality(n DPE)	_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:cardinality* "n"^^*xsd:nonNegativeInteger* .	_:x
DataExactCardinality(n DPE DR)	_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperty* T(DPE) . _:x *owl:qualifiedCardinality* "n"^^*xsd:nonNegativeInteger* . _:x *owl:onDataRange* T(DR) .	_:x
SubClassOf(CE_1 CE_2)	T(CE_1) *rdfs:subClassOf* T(CE_2) .	
EquivalentClasses(CE_1 ... CE_n)	T(CE_1) *owl:equivalentClass* T(CE_2) . ... T(CE_{n-1}) *owl:equivalentClass* T(CE_n) .	
DisjointClasses(CE_1 CE_2)	T(CE_1) *owl:disjointWith* T(CE_2) .	
DisjointClasses(CE_1 ... CE_n), n > 2	_:x *rdf:type* *owl:AllDisjointClasses* . _:x *owl:members* T(SEQ CE_1 ... CE_n) .	
DisjointUnion(C CE_1 ... CE_n)	T(C) *owl:disjointUnionOf* T(SEQ CE_1 ... CE_n) .	
SubObjectPropertyOf(OPE_1 OPE_2)	T(OPE_1) *rdfs:subPropertyOf* T(OPE_2) .	
SubObjectPropertyOf(ObjectPropertyChain(OPE_1 ... OPE_n) OPE)	T(OPE) *owl:propertyChainAxiom* T(SEQ OPE_1 ... OPE_n) .	
EquivalentObjectProperties(OPE_1 ... OPE_n)	T(OPE_1) *owl:equivalentProperty* T(OPE_2) . ... T(OPE_{n-1})	

W3C Recommendation

	owl:equivalentProperty T(OPE_n) .	
DisjointObjectProperties(OPE_1 OPE_2)	T(OPE_1) *owl:propertyDisjointWith* T(OPE_2) .	
DisjointObjectProperties(OPE_1 ... OPE_n), n > 2	_:x *rdf:type* *owl:AllDisjointProperties* . _:x *owl:members* T(SEQ OPE_1 ... OPE_n) .	
ObjectPropertyDomain(OPE CE)	T(OPE) *rdfs:domain* T(CE) .	
ObjectPropertyRange(OPE CE)	T(OPE) *rdfs:range* T(CE) .	
InverseObjectProperties(OPE_1 OPE_2)	T(OPE_1) *owl:inverseOf* T(OPE_2) .	
FunctionalObjectProperty(OPE)	T(OPE) *rdf:type* *owl:FunctionalProperty* .	
InverseFunctionalObjectProperty(OPE)	T(OPE) *rdf:type* *owl:InverseFunctionalProperty* .	
ReflexiveObjectProperty(OPE)	T(OPE) *rdf:type* *owl:ReflexiveProperty* .	
IrreflexiveObjectProperty(OPE)	T(OPE) *rdf:type* *owl:IrreflexiveProperty* .	
SymmetricObjectProperty(OPE)	T(OPE) *rdf:type* *owl:SymmetricProperty* .	
AsymmetricObjectProperty(OPE)	T(OPE) *rdf:type* *owl:AsymmetricProperty* .	
TransitiveObjectProperty(OPE)	T(OPE) *rdf:type* *owl:TransitiveProperty* .	
SubDataPropertyOf(DPE_1 DPE_2)	T(DPE_1) *rdfs:subPropertyOf* T(DPE_2) .	
EquivalentDataProperties(DPE_1 ... DPE_n)	T(DPE_1) *owl:equivalentProperty* T(DPE_2) . ... T(DPE_{n-1}) *owl:equivalentProperty* T(DPE_n) .	
DisjointDataProperties(DPE_1 DPE_2)	T(DPE_1) *owl:propertyDisjointWith* T(DPE_2) .	

W3C Recommendation

DisjointDataProperties(DPE_1 ... DPE_n), n > 2	_:x *rdf:type* *owl:AllDisjointProperties* . _:x *owl:members* T(SEQ DPE_1 ... DPE_n) .	
DataPropertyDomain(DPE CE)	T(DPE) *rdfs:domain* T(CE) .	
DataPropertyRange(DPE DR)	T(DPE) *rdfs:range* T(DR) .	
FunctionalDataProperty(DPE)	T(DPE) *rdf:type* *owl:FunctionalProperty* .	
DatatypeDefinition(DT DR)	T(DT) *owl:equivalentClass* T(DR) .	
HasKey(CE (OPE_1 ... OPE_m) (DPE_1 ... DPE_n))	T(CE) *owl:hasKey* T(SEQ OPE_1 ... OPE_m DPE_1 ... DPE_n) .	
SameIndividual(a_1 ... a_n)	T(a_1) *owl:sameAs* T(a_2) . ... T(a_{n-1}) *owl:sameAs* T(a_n) .	
DifferentIndividuals(a_1 a_2)	T(a_1) *owl:differentFrom* T(a_2) .	
DifferentIndividuals(a_1 ... a_n), n > 2	_:x *rdf:type* *owl:AllDifferent* . _:x *owl:members* T(SEQ a_1 ... a_n) .	
ClassAssertion(CE a)	T(a) *rdf:type* T(CE) .	
ObjectPropertyAssertion(OP a_1 a_2)	T(a_1) T(OP) T(a_2) .	
ObjectPropertyAssertion(ObjectInverseOf(OP) a_1 a_2)	T(a_2) T(OP) T(a_1) .	
NegativeObjectPropertyAssertion(OPE a_1 a_2)	_:x *rdf:type* *owl:NegativePropertyAssertion* . _:x *owl:sourceIndividual* T(a_1) . _:x *owl:assertionProperty* T(OPE) . _:x *owl:targetIndividual* T(a_2) .	
DataPropertyAssertion(DPE a lt)	T(a) T(DPE) T(lt) .	
NegativeDataPropertyAssertion(DPE a lt)	_:x *rdf:type* *owl:NegativePropertyAssertion* . _:x *owl:sourceIndividual* T(a) . _:x *owl:assertionProperty*	

W3C Recommendation

	T(DPE) . _:x *owl:targetValue* T(lt) .	
AnnotationAssertion(AP as av)	T(as) T(AP) T(av) .	
SubAnnotationPropertyOf(AP_1 AP_2)	T(AP_1) *rdfs:subPropertyOf* T(AP_2) .	
AnnotationPropertyDomain(AP U)	T(AP) *rdfs:domain* T(U) .	
AnnotationPropertyRange(AP U)	T(AP) *rdfs:range* T(U) .	

2.2 Translation of Annotations

The operator *TANN*, which translates annotations and attaches them to an IRI or a blank node, is defined in Table 2.

Table 2. Translation of Annotations

Annotation ***ann***	Triples Generated in an Invocation of ***TANN(ann, y)***
Annotation(AP av)	T(y) T(AP) T(av) .
Annotation(annotation$_1$ ... annotation$_n$ AP av)	T(y) T(AP) T(av) . _:x *rdf:type* *owl:Annotation* . _:x *owl:annotatedSource* T(y) . _:x *owl:annotatedProperty* T(AP) . _:x *owl:annotatedTarget* T(av) . TANN(annotation$_1$, _:x) ... TANN(annotation$_n$, _:x)

Example:

Let *ann* be the following annotation.

```
Annotation( rdfs:label "Peter Griffin" )
```

An invocation of *TANN(ann, a:Peter)* then produces the following triples.

```
a:Peter rdfs:label "Peter Griffin" .
```

Example:

Let *ann* be the following annotation, which is itself annotated.

```
Annotation( Annotation( a:author a:Seth_MacFarlane )
    rdfs:label "Peter Griffin" )
```

An invocation of *TANN(ann, a:Peter)* then produces the following triples:

```
a:Peter rdfs:label "Peter Griffin" .
_:x rdf:type owl:Annotation .
```

W3C Recommendation

```
_:x owl:annotatedSource a:Peter .
_:x owl:annotatedProperty rdfs:label .
_:x owl:annotatedTarget "Peter Griffin" .
_:x a:author a:Seth_MacFarlane .
```

2.3 Translation of Axioms with Annotations

If an axiom *ax* contains embedded annotations annotation$_1$ ... annotation$_m$, its serialization into RDF depends on the type of the axiom. Let *ax*' be the axiom that is obtained from *ax* by removing all axiom annotations.

2.3.1 Axioms that Generate a Main Triple

If the row of Table 1 corresponding to the type of *ax*' contains a single *main* triple s p xlt ., then the axiom *ax* is translated into the following triples:

```
s p xlt .
_:x rdf:type owl:Axiom .
_:x owl:annotatedSource s .
_:x owl:annotatedProperty p .
_:x owl:annotatedTarget xlt .
TANN(annotation1, _:x)
...
TANN(annotationm, _:x)
```

This is the case if *ax*' is of type **SubClassOf**, **DisjointClasses** with two classes, **SubObjectPropertyOf** without a property chain as the subproperty expression, **SubDataPropertyOf**, **ObjectPropertyDomain**, **DataPropertyDomain**, **ObjectPropertyRange**, **DataPropertyRange**, **InverseObjectProperties**, **FunctionalObjectProperty**, **FunctionalDataProperty**, **InverseFunctionalObjectProperty**, **ReflexiveObjectProperty**, **IrreflexiveObjectProperty**, **SymmetricObjectProperty**, **AsymmetricObjectProperty**, **TransitiveObjectProperty**, **DisjointObjectProperties** with two properties, **DisjointDataProperties** with two properties, **ClassAssertion**, **ObjectPropertyAssertion**, **DataPropertyAssertion**, **Declaration**, **DifferentIndividuals** with two individuals, or **AnnotationAssertion**.

Example:

Consider the following subclass axiom:

```
SubClassOf( Annotation( rdfs:comment "Children are people." ) a:Child
a:Person )
```

Without the annotation, the axiom would be translated into the following triple:

```
a:Child rdfs:subClassOf a:Person .
```

Thus, the annotated axiom is transformed into the following triples:

```
a:Child rdfs:subClassOf a:Person .
_:x rdf:type owl:Axiom .
```

W3C Recommendation

```
_:x owl:annotatedSource a:Child .
_:x owl:annotatedProperty rdfs:subClassOf .
_:x owl:annotatedTarget a:Person .
_:x rdfs:comment "Children are people." .
```

For *ax*' of type **DisjointUnion**, **SubObjectPropertyOf** with a subproperty chain, or **HasKey**, the first triple from the corresponding row of Table 1 is the *main* triple and it is subjected to the transformation described above; the other triples from the corresponding row of Table 1 — called *side* triples — are output without any change.

Example:

Consider the following subproperty axiom:

```
SubObjectPropertyOf( Annotation( rdfs:comment "An aunt is a mother's
sister." ) ObjectPropertyChain( a:hasMother a:hasSister ) a:hasAunt ) )
```

Without the annotation, the axiom would be translated into the following triples:

```
a:hasAunt owl:propertyChainAxiom _:y1.
_:y1 rdf:first a:hasMother .
_:y1 rdf:rest _:y2 .
_:y2 rdf:first a:hasSister .
_:y2 rdf:rest rdf:nil .
```

In order to capture the annotation on the axiom, the first triple plays the role of the main triple for the axiom, so it is represented using a fresh blank node _:x in order to be able to attach the annotation to it. The original triple is output alongside all other triples as well.

```
_:x rdf:type owl:Axiom .
_:x owl:annotatedSource a:hasAunt .
_:x owl:annotatedProperty owl:propertyChainAxiom .
_:x owl:annotatedTarget _:y1 .
_:x rdfs:comment "An aunt is a mother's sister." .

a:hasAunt owl:propertyChainAxiom _:y1.
_:y1 rdf:first a:hasMother .
_:y1 rdf:rest _:y2 .
_:y2 rdf:first a:hasSister .
_:y2 rdf:rest rdf:nil .
```

Example:

Consider the following key axiom:

```
HasKey( Annotation( rdfs:comment "SSN uniquely determines a person." )
a:Person () ( a:hasSSN ) )
```

Without the annotation, the axiom would be translated into the following triples:

W3C Recommendation

```
a:Person owl:hasKey _:y .
_:y rdf:first a:hasSSN .
_:y rdf:rest rdf:nil .
```

In order to capture the annotation on the axiom, the first triple plays the role of the main triple for the axiom, so it is represented using a fresh blank node _:x in order to be able to attach the annotation to it.

```
_:x rdf:type owl:Axiom .
_:x owl:annotatedSource a:Person .
_:x owl:annotatedProperty owl:hasKey .
_:x owl:annotatedTarget _:y .
_:x rdfs:comment "SSN uniquely determines a person." .

a:Person owl:hasKey _:y .
_:y rdf:first a:hasSSN .
_:y rdf:rest rdf:nil .
```

2.3.2 Axioms that are Translated to Multiple Triples

If the axiom *ax*' is of type **EquivalentClasses**, **EquivalentObjectProperties**, **EquivalentDataProperties**, or **SameIndividual**, its translation into RDF can be broken up into several RDF triples (because RDF can only represent binary relations). In this case, each of the RDF triples obtained by the translation of *ax*' is transformed as described in previous section, and the annotations are repeated for each of the triples obtained in the translation.

Example:

Consider the following individual equality axiom:

```
SameIndividual( Annotation( a:source a:Fox ) a:Meg a:Megan
a:Megan_Griffin )
```

This axiom is first split into the following equalities between pairs of individuals, and the annotation is repeated on each axiom obtained in this process:

```
SameIndividual( Annotation( a:source a:Fox ) a:Meg a:Megan )
SameIndividual( Annotation( a:source a:Fox ) a:Megan a:Megan_Griffin )
```

Each of these axioms is now transformed into triples as explained in the previous section:

```
a:Meg owl:sameAs a:Megan .
_:x1 rdf:type owl:Axiom .
_:x1 owl:annotatedSource a:Meg .
_:x1 owl:annotatedProperty owl:sameAs .
_:x1 owl:annotatedTarget a:Megan .
_:x1 a:source a:Fox .

a:Megan owl:sameAs a:Megan_Griffin .
_:x2 rdf:type owl:Axiom .
_:x2 owl:annotatedSource a:Megan .
```

```
_:x2 owl:annotatedProperty owl:sameAs .
_:x2 owl:annotatedTarget a:Megan_Griffin .
_:x2 a:source a:Fox .
```

2.3.3 Axioms Represented by Blank Nodes

If the axiom *ax*' is of type **NegativeObjectPropertyAssertion**, **NegativeDataPropertyAssertion**, **DisjointClasses** with more than two classes, **DisjointObjectProperties** with more than two properties, **DisjointDataProperties** with more than two properties, or **DifferentIndividuals** with more than two individuals, then its translation already requires introducing a blank node `_:x`. In such cases, *ax* is translated by first translating *ax*' into `_:x` as shown in Table 1, and then attaching the annotations of *ax* to `_:x`.

Example:

Consider the following negative object property assertion:

```
NegativeObjectPropertyAssertion( Annotation( a:author a:Seth_MacFarlane )
a:brotherOf a:Chris a:Stewie )
```

Even without the annotation, this axiom would be represented using a blank node. The annotation can readily be attached to this node, so the axiom is transformed into the following triples:

```
_:x rdf:type owl:NegativePropertyAssertion .
_:x owl:sourceIndividual a:Chris .
_:x owl:assertionProperty a:brotherOf .
_:x owl:targetIndividual a:Stewie .
_:x a:author a:Seth_MacFarlane .
```

3 Mapping from RDF Graphs to the Structural Specification

This section specifies the results of steps CP 2.2 and CP 3.3 of the canonical parsing process from Section 3.6 of the OWL 2 Specification [*OWL 2 Specification*] on an ontology document *D* that can be parsed into an RDF graph *G*. An OWL 2 tool *may* implement these steps in any way it chooses; however, the results *must* be structurally equivalent to the ones defined in the following sections. These steps do not depend on the RDF syntax used to encode the RDF graph in *D*; therefore, the ontology document *D* is identified in this section with the corresponding RDF graph *G*.

An *RDF syntax ontology document* is any document accessible from some given IRI that can be parsed into an RDF graph, and that then be transformed into an OWL 2 ontology by the canonical parsing process instantiated as specified in this section.

The following sections contain rules in which triple patterns are matched to *G*. Note that if a triple pattern contains a variable number of triples, the maximal possible subset of *G* *must* be matched.

The following notation is used in the patterns:

- The notation `NN_INT(n)` can be matched to any literal whose value `n` is a nonnegative integer.
- Possible conditions on the pattern are enclosed in curly braces '{ }'.
- Some patterns use optional parts, which are enclosed in square brackets '[ ]'.
- The abbreviation T(SEQ y_1 ... y_n) denotes the pattern corresponding to RDF lists, as shown in Table 3. When a list pattern is matched to *G*, all list variables _:x_i and _:x_j with i ≠ j *must* be matched to different nodes; furthermore, it *must not* be possible to match the list pattern to two maximal subsets of *G* such that some list variable in the first pattern instance is matched to the same node as some (possibly different) variable in the second pattern instance. This is necessary in order to detect malformed lists such as lists with internal cycles, lists that share tails, and lists that cross.

Table 3. Patterns Corresponding to RDF Lists

Sequence *S*	Triples Corresponding to *T(S)*	Main Node of *T(S)*
SEQ		*rdf:nil*
SEQ y	_:x *rdf:first* y . _:x *rdf:rest* *rdf:nil* .	_:x
SEQ y_1 ... y_n { n>1 }	_:x_1 *rdf:first* y_1 . _:x_1 *rdf:rest* _:x_2 . ... _:x_n *rdf:first* y_n . _:x_n *rdf:rest* *rdf:nil* .	_:x_1

3.1 Extracting Declarations and the IRIs of the Directly Imported Ontology Documents

This section specifies the result of step CP 2.2 of the canonical parsing process on an RDF graph *G*.

3.1.1 Resolving Included RDF Graphs

For backwards compatibility with OWL 1 DL, if *G* contains an *owl:imports* triple pointing to an RDF document encoding an RDF graph *G*' where *G*' does not have an ontology header, this *owl:imports* triple is interpreted as an *include* rather than an import — that is, the triples of *G*' are included into *G* and are not parsed into a separate ontology. To achieve this, the following transformation is applied to *G* as long as the following rule is applicable to *G*.

If *G* contains a pair of triples of the form

```
x rdf:type owl:Ontology .
x owl:imports *:y .
```

and the values for x and *:y have not already been considered, the following actions are performed:

1. The document accessible from the IRI *:y is retrieved using the augmented retrieval process from Section 3.2 of the OWL 2 Specification [*OWL 2 Specification*].
2. The document is parsed into an RDF graph *G*'.
3. If the parsing succeeds and the graph *G*' does not contain a triple of the form

   ```
   z rdf:type owl:Ontology.
   ```

   then *G*' is merged (as in the RDF Semantics [*RDF Semantics*]) into *G* and the triple x *owl:imports* *:y .

is removed from *G*.

3.1.2 Parsing of the Ontology Header and Declarations

Next, the ontology header is extracted from *G* by matching patterns from Table 4 to *G*. It *must* be possible to match exactly one such pattern to *G* in exactly one way. The matched triples are removed from *G*. The set *Imp(G)* of the IRIs of ontology documents that are directly imported into *G* contains exactly all *:z_1, ..., *:z_k that are matched in the pattern.

Table 4. Parsing of the Ontology Header

If *G* contains this pattern...	**...then the ontology header has this form.**

*:x *rdf:type owl:Ontology* .
[*:x *owl:versionIRI* *:y .]
*:x *owl:imports* *:z_1 .
...
*:x *owl:imports* *:z_k .
{ k ≥ 0 and
  the following triple pattern cannot be matched in *G*:
     u w *:x .
     u *rdf:type owl:Ontology* .
     w *rdf:type owl:OntologyProperty* .
}
	Ontology(*:x [*:y]
	    Import(*:z_1)
	    ...
	    Import(*:z_k)
	    ...
	)

_:x *rdf:type owl:Ontology* .
_:x *owl:imports* *:z_1 .
...
_:x *owl:imports* *:z_k .
{ k ≥ 0 and
  the following triple pattern cannot be matched in *G*:
     u w _:x .
     u *rdf:type owl:Ontology* .
     w *rdf:type owl:OntologyProperty* .
}
	Ontology(
	    Import(*:z_1)
	    ...
	    Import(*:z_k)
	    ...
	)

Next, for backwards compatibility with OWL 1 DL, certain redundant triples are removed from *G*. In particular, if the triple pattern from the left-hand side of Table 5 is matched in *G*, then the triples on the right-hand side of Table 5 are removed from *G*.

Table 5. Triples to be Removed for Backwards Compatibility with OWL 1 DL

If *G* contains this pattern...	**...then these triples are removed from *G*.**
x *rdf:type owl:Ontology* .	x *rdf:type owl:Ontology* .
x *rdf:type owl:Class* . x *rdf:type rdfs:Class* .	x *rdf:type rdfs:Class* .

W3C Recommendation

x *rdf:type rdfs:Datatype* . x *rdf:type rdfs:Class* .	x *rdf:type rdfs:Class* .
x *rdf:type owl:DataRange* . x *rdf:type rdfs:Class* .	x *rdf:type rdfs:Class* .
x *rdf:type owl:Restriction* . x *rdf:type rdfs:Class* .	x *rdf:type rdfs:Class* .
x *rdf:type owl:Restriction* . x *rdf:type owl:Class* .	x *rdf:type owl:Class* .
x *rdf:type owl:ObjectProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type owl:FunctionalProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type owl:InverseFunctionalProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type owl:TransitiveProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type owl:DatatypeProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type owl:AnnotationProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type owl:OntologyProperty* . x *rdf:type rdf:Property* .	x *rdf:type rdf:Property* .
x *rdf:type rdf:List* . x *rdf:first* y . x *rdf:rest* z .	x *rdf:type rdf:List* .

Next, for backwards compatibility with OWL 1 DL, *G* is modified such that declarations can be properly extracted in the next step. When a triple pattern from the first column of Table 6 is matched in *G*, the matching triples are *replaced in G* with the triples from the second column. This matching phase stops when matching a pattern and replacing it as specified does not change *G*. Note that *G* is a set and thus cannot contain duplicate triples, so this last condition prevents infinite matches.

Table 6. Additional Declaration Triples

If *G* contains this pattern...	...then the matched triples are replaced in *G* with these triples.
*:x *rdf:type owl:OntologyProperty* .	*:x *rdf:type owl:AnnotationProperty* .
*:x *rdf:type owl:InverseFunctionalProperty* .	*:x *rdf:type owl:ObjectProperty* . *:x *rdf:type owl:InverseFunctionalProperty* .

W3C Recommendation

*:x *rdf:type owl:TransitiveProperty* .	*:x *rdf:type owl:ObjectProperty* . *:x *rdf:type owl:TransitiveProperty* .
*:x *rdf:type owl:SymmetricProperty* .	*:x *rdf:type owl:ObjectProperty* . *:x *rdf:type owl:SymmetricProperty* .

Next, the set of declarations *Decl(G)* is extracted from *G* according to Table 7. The matched triples are not removed from *G* — the triples from Table 7 can contain annotations so, in order to correctly parse the annotations, they will be matched again in the step described in Section 3.2.5.

Table 7. Parsing Declarations in *G*

If *G* contains this pattern...	**...then this declaration is added to *Decl(G)*.**
*:x *rdf:type owl:Class* .	Declaration(Class(*:x))
*:x *rdf:type rdfs:Datatype* .	Declaration(Datatype(*:x))
*:x *rdf:type owl:ObjectProperty* .	Declaration(ObjectProperty(*:x))
*:x *rdf:type owl:DatatypeProperty* .	Declaration(DataProperty(*:x))
*:x *rdf:type owl:AnnotationProperty* .	Declaration(AnnotationProperty(*:x))
*:x *rdf:type owl:NamedIndividual* .	Declaration(NamedIndividual(*:x))
_:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* *:y . _:x *owl:annotatedProperty rdf:type* . _:x *owl:annotatedTarget owl:Class* .	Declaration(Class(*:y))
_:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* *:y . _:x *owl:annotatedProperty rdf:type* . _:x *owl:annotatedTarget rdfs:Datatype* .	Declaration(Datatype(*:y))
_:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* *:y . _:x *owl:annotatedProperty rdf:type* . _:x *owl:annotatedTarget owl:ObjectProperty* .	Declaration(ObjectProperty(*:y))
_:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* *:y . _:x *owl:annotatedProperty rdf:type* . _:x *owl:annotatedTarget owl:DatatypeProperty* .	Declaration(DataProperty(*:y))
_:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* *:y . _:x *owl:annotatedProperty rdf:type* .	Declaration(AnnotationProperty(*:y))

_:x *owl:annotatedTarget owl:AnnotationProperty* .	
_:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* *:y . _:x *owl:annotatedProperty rdf:type* . _:x *owl:annotatedTarget owl:NamedIndividual* .	Declaration(NamedIndividual(*:y))

Finally, the set RIND of blank nodes used in reification is identified. This is done by initially setting RIND = ∅ and then applying the patterns shown in Table 8. The matched triples are not deleted from *G*.

Table 8. Identifying Reification Blank Nodes

If *G* contains this pattern, then _:x is added to RIND.
_:x *rdf:type owl:Axiom* .
_:x *rdf:type owl:Annotation* .
_:x *rdf:type owl:AllDisjointClasses* .
_:x *rdf:type owl:AllDisjointProperties* .
_:x *rdf:type owl:AllDifferent* .
_:x *rdf:type owl:NegativePropertyAssertion* .

3.2 Populating an Ontology

This section specifies the result of step CP 3.3 of the canonical parsing process on an RDF graph *G*, the corresponding instance O_G of the **Ontology** class, and the set *AllDecl(G)* of all declarations for *G* computed as specified in step CP 3.1 of the canonical parsing process.

3.2.1 Analyzing Declarations

The following functions map an IRI or a blank node x occurring in *G* into an object of the structural specification. In particular,

- CE(x) maps x into a class expression,
- DR(x) maps x into a data range,
- OPE(x) maps x into an object property expression,
- DPE(x) maps x into a data property expression, and
- AP(x) maps x into an annotation property.

Initially, these functions are undefined for all IRIs and blank nodes occurring in *G*; this is written as CE(x) = ε, DR(x) = ε, OPE(x) = ε, DPE(x) = ε, and AP(x) = ε. The functions are updated as parsing progresses. All of the following conditions *must* be satisfied at any given point in time during parsing.

- For each x, at most one of OPE(x), DPE(x), and AP(x) is defined.
- For each x, at most one of CE(x) and DR(x) is defined.

Furthermore, the value of any of these functions for any x *must not* be redefined during parsing (i.e., if a function is not undefined for x, no attempt should be made to change the function's value for x).

Functions CE, DR, OPE, DPE, and AP are initialized as shown in Table 9.

Table 9. Initialization of CE, DR, OPE, DPE, and AP

If *AllDecl(G)* contains this declaration...	...then perform this assignment.
`Declaration( Class( *:x ) )`	CE(*:x) := a class with the IRI *:x
`Declaration( Datatype( *:x ) )`	DR(*:x) := a datatype with the IRI *:x
`Declaration( ObjectProperty( *:x ) )`	OPE(*:x) := an object property with the IRI *:x
`Declaration( DataProperty( *:x ) )`	DPE(*:x) := a data property with the IRI *:x
`Declaration( AnnotationProperty( *:x ) )`	AP(*:x) := an annotation property with the IRI *:x

3.2.2 Parsing of Annotations

The annotations in *G* are parsed next. The function ANN assigns a set of annotations ANN(x) to each IRI or blank node x. This function is initialized by setting ANN(x) = ∅ for each each IRI or blank node x. Next, the triple patterns from Table 10 are matched in *G* and, for each matched pattern, ANN(x) is extended with an annotation from the right column. Each time one of these triple patterns is matched, the matched triples are removed from *G*. This process is repeated until no further matches are possible.

Table 10. Parsing of Annotations

If *G* contains this pattern...	...then this annotation is added to *ANN(x)*.
x *:y xlt . { AP(*:y) ≠ ε and there is no blank node _:w such that G contains the following triples: _:w rdf:type owl:Annotation . _:w owl:annotatedSource x . _:w owl:annotatedProperty *:y . _:w owl:annotatedTarget xlt . }	Annotation(*:y xlt)
x *:y xlt . _:w rdf:type owl:Annotation . _:w owl:annotatedSource x . _:w owl:annotatedProperty *:y . _:w owl:annotatedTarget xlt . { AP(*:y) ≠ ε and no other triple in G contains _:w in subject or object position }	Annotation(ANN(_:w) *:y xlt)

W3C Recommendation

3.2.3 Parsing of Ontology Annotations

Let x be the node that was matched in *G* to `*:x` or `_:x` according to the patterns from Table 4; then, `ANN(x)` determines the set of ontology annotations of O_G.

3.2.4 Parsing of Expressions

Next, functions OPE, DR, and CE are extended as shown in Tables 11, 12, and 13, as well as in Tables 14 and 15. The patterns in the latter two tables are not generated by the mapping from Section 2, but they can be present in RDF graphs that encode OWL 1 DL ontologies. Each time a pattern is matched, the matched triples are removed from *G*. Pattern matching is repeated until no triple pattern can be matched to *G*.

Table 11. Parsing Object Property Expressions

If *G* contains this pattern...	...then *OPE(_:x)* is set to this object property expression.
:x *owl:inverseOf* *:y . { OPE(:x) = ε and OPE(*:y) ≠ ε }	ObjectInverseOf(OPE(*:y))

Table 12. Parsing of Data Ranges

If *G* contains this pattern...	...then *DR(_:x)* is set to this data range.
_:x *rdf:type* *rdfs:Datatype* . _:x *owl:intersectionOf* T(SEQ y_1 ... y_n) . { n ≥ 2 and DR(y_i) ≠ ε for each 1 ≤ i ≤ n }	DataIntersectionOf(DR(y_1) ... DR(y_n))
_:x *rdf:type* *rdfs:Datatype* . _:x *owl:unionOf* T(SEQ y_1 ... y_n) . { n ≥ 2 and DR(y_i) ≠ ε for each 1 ≤ i ≤ n }	DataUnionOf(DR(y_1) ... DR(y_n))
_:x *rdf:type* *rdfs:Datatype* . _:x *owl:datatypeComplementOf* y . { DR(y) ≠ ε }	DataComplementOf(DR(y))
_:x *rdf:type* *rdfs:Datatype* . _:x *owl:oneOf* T(SEQ lt_1 ... lt_n) . { n ≥ 1 }	DataOneOf(lt_1 ... lt_n)
_:x *rdf:type* *rdfs:Datatype* . _:x *owl:onDatatype* *:y . _:x *owl:withRestrictions* T(SEQ _:z_1 ... _:z_n) . _:z_1 *:w_1 lt_1 . ... _:z_n *:w_n lt_n . { DR(*:y) is a datatype }	DatatypeRestriction(DR(*:y) *:w_1 lt_1 ... *:w_n lt_n)

Table 13. Parsing of Class Expressions

If *G* contains this pattern...	...then *CE(_:x)* is set to this class expression.
_:x *rdf:type owl:Class* . _:x *owl:intersectionOf* T(SEQ y_1 ... y_n) . { $n \geq 2$ and $CE(y_i) \neq \varepsilon$ for each $1 \leq i \leq n$ }	ObjectIntersectionOf(CE(y_1) ... CE(y_n))
_:x *rdf:type owl:Class* . _:x *owl:unionOf* T(SEQ y_1 ... y_n) . { $n \geq 2$ and $CE(y_i) \neq \varepsilon$ for each $1 \leq i \leq n$ }	ObjectUnionOf(CE(y_1) ... CE(y_n))
_:x *rdf:type owl:Class* . _:x *owl:complementOf* y . { $CE(y) \neq \varepsilon$ }	ObjectComplementOf(CE(y))
_:x *rdf:type owl:Class* . _:x *owl:oneOf* T(SEQ *:y_1 ... *:y_n) . { $n \geq 1$ }	ObjectOneOf(*:y_1 ... *:y_n)
_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:someValuesFrom* z . { $OPE(y) \neq \varepsilon$ and $CE(z) \neq \varepsilon$ }	ObjectSomeValuesFrom(OPE(y) CE(z))
_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:allValuesFrom* z . { $OPE(y) \neq \varepsilon$ and $CE(z) \neq \varepsilon$ }	ObjectAllValuesFrom(OPE(y) CE(z))
_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:hasValue* *:z . { $OPE(y) \neq \varepsilon$ }	ObjectHasValue(OPE(y) *:z)
_:x *rdf:type owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:hasSelf* "true"^^*xsd:boolean* . { $OPE(y) \neq \varepsilon$ }	ObjectHasSelf(OPE(y))
_:x *rdf:type owl:Restriction* . _:x *owl:minQualifiedCardinality* NN_INT(n) . _:x *owl:onProperty* y . _:x owl:onClass z . { $OPE(y) \neq \varepsilon$ and $CE(z) \neq \varepsilon$ }	ObjectMinCardinality(n OPE(y) CE(z))
_:x *rdf:type owl:Restriction* . _:x *owl:maxQualifiedCardinality* NN_INT(n) . _:x *owl:onProperty* y . _:x owl:onClass z . { $OPE(y) \neq \varepsilon$ and $CE(z) \neq \varepsilon$ }	ObjectMaxCardinality(n OPE(y) CE(z))

W3C Recommendation

_:x *rdf:type* *owl:Restriction* . _:x *owl:qualifiedCardinality* NN_INT(n) . _:x *owl:onProperty* y . _:x owl:onClass z . { OPE(y) ≠ ε and CE(z) ≠ ε }	ObjectExactCardinality(n OPE(y) CE(z))
_:x *rdf:type* *owl:Restriction* . _:x *owl:minCardinality* NN_INT(n) . _:x *owl:onProperty* y . { OPE(y) ≠ ε }	ObjectMinCardinality(n OPE(y))
_:x *rdf:type* *owl:Restriction* . _:x *owl:maxCardinality* NN_INT(n) . _:x *owl:onProperty* y . { OPE(y) ≠ ε }	ObjectMaxCardinality(n OPE(y))
_:x *rdf:type* *owl:Restriction* . _:x *owl:cardinality* NN_INT(n) . _:x *owl:onProperty* y . { OPE(y) ≠ ε }	ObjectExactCardinality(n OPE(y))
_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:hasValue* lt . { DPE(y) ≠ ε }	DataHasValue(DPE(y) lt)
_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:someValuesFrom* z . { DPE(y) ≠ ε and DR(z) ≠ ε }	DataSomeValuesFrom(DPE(y) DR(z))
_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperties* T(SEQ y_1 ... y_n) . _:x *owl:someValuesFrom* z . { n ≥ 1, DPE(y_i) ≠ ε for each 1 ≤ i ≤ n, and DR(z) ≠ ε }	DataSomeValuesFrom(DPE(y_1) ... DPE(y_n) DR(z))
_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperty* y . _:x *owl:allValuesFrom* z . { DPE(y) ≠ ε and DR(z) ≠ ε }	DataAllValuesFrom(DPE(y) DR(z))
_:x *rdf:type* *owl:Restriction* . _:x *owl:onProperties* T(SEQ y_1 ... y_n) . _:x *owl:allValuesFrom* z . { n ≥ 1, DPE(y_i) ≠ ε for each 1 ≤ i ≤ n, and DR(z) ≠ ε }	DataAllValuesFrom(DPE(y_1) ... DPE(y_n) DR(z))
_:x *rdf:type* *owl:Restriction* . _:x *owl:minQualifiedCardinality* NN_INT(n) . _:x *owl:onProperty* y . _:x *owl:onDataRange* z . { DPE(y) ≠ ε and DR(z) ≠ ε }	DataMinCardinality(n DPE(y) DR(z))
_:x *rdf:type* *owl:Restriction* . _:x *owl:maxQualifiedCardinality* NN_INT(n)	DataMaxCardinality(n DPE(y) DR(z))

. _:x *owl:onProperty* y . _:x *owl:onDataRange* z . { DPE(y) ≠ ε and DR(z) ≠ ε }	
_:x *rdf:type owl:Restriction* . _:x *owl:qualifiedCardinality* NN_INT(n) . _:x *owl:onProperty* y . _:x *owl:onDataRange* z . { DPE(y) ≠ ε and DR(z) ≠ ε }	DataExactCardinality(n DPE(y) DR(z))
_:x *rdf:type owl:Restriction* . _:x *owl:minCardinality* NN_INT(n) . _:x *owl:onProperty* y . { DPE(y) ≠ ε }	DataMinCardinality(n DPE(y))
_:x *rdf:type owl:Restriction* . _:x *owl:maxCardinality* NN_INT(n) . _:x *owl:onProperty* y . { DPE(y) ≠ ε }	DataMaxCardinality(n DPE(y))
_:x *rdf:type owl:Restriction* . _:x *owl:cardinality* NN_INT(n) . _:x *owl:onProperty* y . { DPE(y) ≠ ε }	DataExactCardinality(n DPE(y))

Table 14. Parsing of Data Ranges for Compatibility with OWL 1 DL

If *G* contains this pattern...	...then *DR(_:x)* is set to this object property expression.
_:x *rdf:type owl:DataRange* . _:x *owl:oneOf* T(SEQ lt$_1$ ... lt$_n$) . { n ≥ 1 }	DataOneOf(lt$_1$ ... lt$_n$)
_:x *rdf:type owl:DataRange* . _:x *owl:oneOf* T(SEQ) .	DataComplementOf(*rdfs:Literal*)

Table 15. Parsing of Class Expressions for Compatibility with OWL 1 DL

If *G* contains this pattern...	...then *CE(_:x)* is set to this class expression.
_:x *rdf:type owl:Class* . _:x *owl:unionOf* T(SEQ) .	*owl:Nothing*
_:x *rdf:type owl:Class* . _:x *owl:unionOf* T(SEQ y) . { CE(y) ≠ ε }	CE(y)
_:x *rdf:type owl:Class* . _:x *owl:intersectionOf* T(SEQ) .	*owl:Thing*
_:x *rdf:type owl:Class* . _:x *owl:intersectionOf* T(SEQ y) . { CE(y) ≠ ε }	CE(y)
_:x *rdf:type owl:Class* . _:x *owl:oneOf* T(SEQ) .	*owl:Nothing*

3.2.5 Parsing of Axioms

Next, O_G is populated with axioms. For clarity, the axiom patterns are split into two tables.

- Table 16 presents the patterns for axioms without annotations.
- Annotated axioms are parsed as follows:
  - In case of the patterns for *owl:AllDisjointClasses*, *owl:AllDisjointProperties*, *owl:AllDifferent*, and *owl:NegativePropertyAssertion*, axiom annotations are defined by `ANN(_:x)`.
  - For all other axioms, axiom annotations are obtained by additionally matching patterns from Table 17 in *G* during axiom matching.

The axioms in *G* are parsed as follows:

- All annotated axioms are parsed first.
- Only when no pattern for annotated axioms can be matched in *G*, then the patterns for axioms without annotations are matched.

In either case, each time a triple pattern is matched, the matched triples are removed from *G*.

Table 16. Parsing of Axioms without Annotations

If *G* contains this pattern...	**...then the following axiom is added to O_G.**
`*:x` *`rdf:type owl:Class`* `.`	`Declaration( Class( *:x ) )`
`*:x` *`rdf:type rdfs:Datatype`* `.`	`Declaration( Datatype( *:x ) )`
`*:x` *`rdf:type owl:ObjectProperty`* `.`	`Declaration( ObjectProperty( *:x ) )`
`*:x` *`rdf:type owl:DatatypeProperty`* `.`	`Declaration( DataProperty( *:x ) )`
`*:x` *`rdf:type owl:AnnotationProperty`* `.`	`Declaration( AnnotationProperty( *:x ) )`
`*:x` *`rdf:type owl:NamedIndividual`* `.`	`Declaration( NamedIndividual( *:x ) )`
`x` *`rdfs:subClassOf`* `y .` `{ CE(x) ≠ ε and CE(y) ≠ ε }`	`SubClassOf( CE(x) CE(y) )`
`x` *`owl:equivalentClass`* `y .` `{ CE(x) ≠ ε and CE(y) ≠ ε }`	`EquivalentClasses( CE(x) CE(y) )`
`x` *`owl:disjointWith`* `y .` `{ CE(x) ≠ ε and CE(y) ≠ ε }`	`DisjointClasses( CE(x) CE(y) )`
`_:x` *`rdf:type owl:AllDisjointClasses`* `.` `_:x` *`owl:members`* T(SEQ y_1 ... y_n) . { n ≥ 2 and CE(y_i) ≠ ε for each 1 ≤ i ≤ n }	DisjointClasses(CE(y_1) ... CE(y_n))
`*:x` *`owl:disjointUnionOf`* T(SEQ y_1 ... y_n) . { n ≥ 2,	DisjointUnion(CE(*:x) CE(y_1) ... CE(y_n))

CE(x) ≠ ε, and CE(y_i) ≠ ε for each 1 ≤ i ≤ n }	
x *rdfs:subPropertyOf* y . { OPE(x) ≠ ε and OPE(y) ≠ ε }	SubObjectPropertyOf(OPE(x) OPE(y))
x *owl:propertyChainAxiom* T(SEQ y_1 ... y_n) . { n ≥ 2, OPE(y_i) ≠ ε for each 1 ≤ i ≤ n, and OPE(x) ≠ ε }	SubObjectPropertyOf(ObjectPropertyChain(OPE(y_1) ... OPE(y_n)) OPE(x))
x *owl:equivalentProperty* y . { OPE(x) ≠ ε and OPE(y) ≠ ε }	EquivalentObjectProperties(OPE(x) OPE(y))
x *owl:propertyDisjointWith* y . { OPE(x) ≠ ε and OPE(y) ≠ ε }	DisjointObjectProperties(OPE(x) OPE(y))
_:x *rdf:type owl:AllDisjointProperties* . _:x *owl:members* T(SEQ y_1 ... y_n) . { n ≥ 2 and OPE(y_i) ≠ ε for each 1 ≤ i ≤ n }	DisjointObjectProperties(OPE(y_1) ... OPE(y_n))
x *rdfs:domain* y . { OPE(x) ≠ ε and CE(y) ≠ ε }	ObjectPropertyDomain(OPE(x) CE(y))
x *rdfs:range* y . { OPE(x) ≠ ε and CE(y) ≠ ε }	ObjectPropertyRange(OPE(x) CE(y))
x *owl:inverseOf* y . { OPE(x) ≠ ε and OPE(y) ≠ ε }	InverseObjectProperties(OPE(x) OPE(y))
x *rdf:type owl:FunctionalProperty* . { OPE(x) ≠ ε }	FunctionalObjectProperty(OPE(x))
x *rdf:type owl:InverseFunctionalProperty* . { OPE(x) ≠ ε }	InverseFunctionalObjectProperty(OPE(x))
x *rdf:type owl:ReflexiveProperty* . { OPE(x) ≠ ε }	ReflexiveObjectProperty(OPE(x))
x *rdf:type owl:IrreflexiveProperty* . { OPE(x) ≠ ε }	IrreflexiveObjectProperty(OPE(x))
x *rdf:type owl:SymmetricProperty* . { OPE(x) ≠ ε }	SymmetricObjectProperty(OPE(x))
x *rdf:type owl:AsymmetricProperty* . { OPE(x) ≠ ε }	AsymmetricObjectProperty(OPE(x))
x *rdf:type owl:TransitiveProperty* . { OPE(x) ≠ ε }	TransitiveObjectProperty(OPE(x))
x *rdfs:subPropertyOf* y . { DPE(x) ≠ ε and DPE(y) ≠ ε }	SubDataPropertyOf(DPE(x) DPE(y))

W3C Recommendation

x *owl:equivalentProperty* y . { DPE(x) ≠ ε and DPE(y) ≠ ε }	EquivalentDataProperties(DPE(x) DPE(y))
x *owl:propertyDisjointWith* y . { DPE(x) ≠ ε and DPE(y) ≠ ε }	DisjointDataProperties(DPE(x) DPE(y))
_:x *rdf:type owl:AllDisjointProperties* . _:x *owl:members* T(SEQ y_1 ... y_n) . { n ≥ 2 and DPE(y_i) ≠ ε for each 1 ≤ i ≤ n }	DisjointDataProperties(DPE(y_1) ... DPE(y_n))
x *rdfs:domain* y . { DPE(x) ≠ ε and CE(y) ≠ ε }	DataPropertyDomain(DPE(x) CE(y))
x *rdfs:range* y . { DPE(x) ≠ ε and DR(y) ≠ ε }	DataPropertyRange(DPE(x) DR(y))
x *rdf:type owl:FunctionalProperty* . { DPE(x) ≠ ε }	FunctionalDataProperty(DPE(x))
*:x *owl:equivalentClass* y . { DR(*:x) ≠ ε amd DR(y) ≠ ε }	DatatypeDefinition(DR(*:x) DR(y))
x *owl:hasKey* T(SEQ y_1 ... y_k) . { CE(x) ≠ ε, and the sequence y_1 ... y_k can be partitioned into disjoint sequences z_1 ... z_m and w_1 ... w_n such that m > 0 or n > 0 (or both) and OPE(z_i) ≠ ε for each 1 ≤ i ≤ m and DPE(w_j) ≠ ε for each 1 ≤ j ≤ n }	HasKey(CE(x) (OPE(z_1) ... OPE(z_m)) (DPE(w_1) ... DPE(w_n)))
x *owl:sameAs* y .	SameIndividual(x y)
x *owl:differentFrom* y .	DifferentIndividuals(x y)
_:x *rdf:type owl:AllDifferent* . _:x *owl:members* T(SEQ x_1 ... x_n) . { n ≥ 2 }	DifferentIndividuals(x_1 ... x_n)
_:x *rdf:type owl:AllDifferent* . _:x *owl:distinctMembers* T(SEQ x_1 ... x_n) . { n ≥ 2 }	DifferentIndividuals(x_1 ... x_n)
x *rdf:type* y . { CE(y) ≠ ε }	ClassAssertion(CE(y) x)
x *:y z . { OPE(*:y) ≠ ε }	ObjectPropertyAssertion(OPE(*:y) x z)
_:x *rdf:type owl:NegativePropertyAssertion* . _:x *owl:sourceIndividual* w . _:x *owl:assertionProperty* y .	NegativeObjectPropertyAssertion(OPE(y) w z)

_:x *owl:targetIndividual* z . { OPE(y) ≠ ε }	
x *:y lt . { DPE(*:y) ≠ ε }	DataPropertyAssertion(DPE(*:y) x lt)
_:x *rdf:type* *owl:NegativePropertyAssertion* . _:x *owl:sourceIndividual* w . _:x *owl:assertionProperty* y . _:x *owl:targetValue* lt . { DPE(y) ≠ ε }	NegativeDataPropertyAssertion(DPE(y) w lt)
*:x *rdf:type owl:DeprecatedClass* .	AnnotationAssertion(*owl:deprecated* *:x "true"^^*xsd:boolean*)
*:x *rdf:type owl:DeprecatedProperty* .	AnnotationAssertion(*owl:deprecated* *:x "true"^^*xsd:boolean*)
*:x *rdfs:subPropertyOf* *:y . { AP(*:x) ≠ ε and AP(*:y) ≠ ε }	SubAnnotationPropertyOf(AP(*:x) AP(*:y))
*:x *rdfs:domain* *:y . { AP(*:x) ≠ ε }	AnnotationPropertyDomain(AP(*:x) *:y)
*:x *rdfs:range* *:y . { AP(*:x) ≠ ε }	AnnotationPropertyRange(AP(*:x) *:y)

Table 17. Parsing of Annotated Axioms

If *G* contains this pattern...	**...then the following axiom is added to O_G.**
s *:p xlt . _:x *rdf:type owl:Axiom* . _:x *owl:annotatedSource* s . _:x *owl:annotatedProperty* *:p . _:x *owl:annotatedTarget* xlt . { s *:p xlt . is the main triple of an axiom according to Table 16 and *G* contains possible necessary side triples for the axiom }	The result is the axiom corresponding to s *:p xlt . (and possible side triples) that additionally contains the annotations ANN(_:x).

Next, for each blank node or IRI x such that x ∉ RIND, and for each annotation Annotation(annotation_1 ... annotation_n AP y) ∈ ANN(x) with n possibly being equal to zero, the following annotation assertion is added to O_G:

AnnotationAssertion(annotation_1 ... annotation_n AP x y)

Finally, the patterns from Table 18 are matched in *G* and the resulting axioms are added to O_G. These patterns are not generated by the mapping from Section 2, but they can be present in RDF graphs that encode OWL 1 DL ontologies. (Note that the patterns from the table do not contain triples of the form *:x *rdf:type owl:Class* because such triples are removed while parsing the entity declarations, as specified in Section 3.1.2.) Each time a triple pattern is matched, the matched triples are removed from *G*.

W3C Recommendation

Table 18. Parsing of Axioms for Compatibility with OWL 1 DL

If *G* contains this pattern...	**...then the following axiom is added to O_G.**
*:x *owl:complementOf* y . { CE(*:x) ≠ ε and CE(y) ≠ ε }	EquivalentClasses(CE(*:x) ObjectComplementOf(CE(y)))
*:x *owl:unionOf* T(SEQ) . { CE(*:x) ≠ ε }	EquivalentClasses(CE(*:x) *owl:Nothing*)
*:x *owl:unionOf* T(SEQ y) . { CE(*:x) ≠ ε and CE(y) ≠ ε }	EquivalentClasses(CE(*:x) CE(y))
*:x *owl:unionOf* T(SEQ y_1 ... y_n) . { n ≥ 2, CE(*:x) ≠ ε, and CE(y_i) ≠ ε for each 1 ≤ i ≤ n }	EquivalentClasses(CE(*:x) ObjectUnionOf(CE(y_1) ... CE(y_n)))
*:x *owl:intersectionOf* T(SEQ) . { CE(*:x) ≠ ε }	EquivalentClasses(CE(*:x) *owl:Thing*)
*:x *owl:intersectionOf* T(SEQ y) . { CE(*:x) ≠ ε and CE(y) ≠ ε }	EquivalentClasses(CE(*:x) CE(y))
*:x *owl:intersectionOf* T(SEQ y_1 ... y_n) . { n ≥ 2, CE(*:x) ≠ ε, and CE(y_i) ≠ ε for each 1 ≤ i ≤ n }	EquivalentClasses(CE(*:x) ObjectIntersectionOf(CE(y_1) ... CE(y_n)))
*:x *owl:oneOf* T(SEQ) . { CE(*:x) ≠ ε }	EquivalentClasses(CE(*:x) *owl:Nothing*)
*:x *owl:oneOf* T(SEQ *:y_1 ... *:y_n) . { n ≥ 1 and CE(*:x) ≠ ε }	EquivalentClasses(CE(*:x) ObjectOneOf(*:y_1 ... *:y_n))

At the end of this process, the graph *G must* be empty.

4 Appendix: Change Log (Informative)

4.1 Changes Since Recommendation

This section summarizes the changes to this document since the Recommendation of 27 October, 2009.

- With the publication of the XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes Recommendation of 5 April 2012, the elements of OWL 2 which are based on

W3C Recommendation

XSD 1.1 are now considered required, and the note detailing the optional dependency on the XSD 1.1 Candidate Recommendation of 30 April, 2009 has been removed from the "Status of this Document" section.
- Minor typographical errors were corrected as detailed on the OWL 2 Errata page.

4.2 Changes Since Proposed Recommendation

This section summarizes the changes to this document since the Proposed Recommendation of 22 September, 2009.

- The two arguments in the ClassAssertion axiom in Table 16 were swapped to bring the axiom in line with the functional-style syntax.

4.3 Changes Since Candidate Recommendation

This section summarizes the changes to this document since the Candidate Recommendation of 11 June, 2009.

- Two minor bugs were fixed in the reverse mappings of inverseOf and hasKey.

4.4 Changes Since Last Call

This section summarizes the changes to this document since the Last Call Working Draft of 21 April, 2009.

- The RDF vocabulary for annotations was changed: owl:subject, owl:predicate and owl:object became, respectively, owl:annotatedSource, owl:annotatedProperty and owl:annotatedTarget.
- Several lists of syntax were updated to track a previous change in Structural Specification and Functional-Style Syntax.
- Two of the examples were fixed.
- Some minor editorial changes were made.

5 Acknowledgments

The starting point for the development of OWL 2 was the OWL1.1 member submission, itself a result of user and developer feedback, and in particular of information gathered during the OWL Experiences and Directions (OWLED) Workshop series. The working group also considered postponed issues from the WebOnt Working Group.

This document has been produced by the OWL Working Group (see below), and its contents reflect extensive discussions within the Working Group as a whole. The editors extend special thanks to Markus Krötzsch (FZI), Alan Ruttenberg (Science Commons), Uli Sattler (University of Manchester), Michael Schneider (FZI) and Evren Sirin (Clark & Parsia) for their thorough reviews.

The regular attendees at meetings of the OWL Working Group at the time of publication of this document were: Jie Bao (RPI), Diego Calvanese (Free University of Bozen-Bolzano), Bernardo Cuenca Grau (Oxford University Computing Laboratory), Martin Dzbor (Open University), Achille Fokoue (IBM Corporation), Christine Golbreich (Université de Versailles St-Quentin and LIRMM), Sandro Hawke (W3C/MIT), Ivan Herman (W3C/ERCIM), Rinke Hoekstra (University of Amsterdam), Ian Horrocks (Oxford University Computing Laboratory), Elisa Kendall (Sandpiper Software), Markus Krötzsch (FZI), Carsten Lutz (Universität Bremen), Deborah L. McGuinness (RPI), Boris Motik (Oxford University Computing Laboratory), Jeff Pan (University of Aberdeen), Bijan Parsia (University of Manchester), Peter F. Patel-Schneider (Bell Labs Research, Alcatel-Lucent), Sebastian Rudolph (FZI), Alan Ruttenberg (Science Commons), Uli Sattler (University of Manchester), Michael Schneider (FZI), Mike Smith (Clark & Parsia), Evan Wallace (NIST), Zhe Wu (Oracle Corporation), and Antoine Zimmermann (DERI Galway). We would also like to thank past members of the working group: Jeremy Carroll, Jim Hendler, and Vipul Kashyap.

6 References

[OWL 2 Specification]
: *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition)* Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. Latest version available at http://www.w3.org/TR/owl2-syntax/.

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

[RDF Semantics]
: *RDF Semantics*. Patrick Hayes, ed., W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Latest version available as http://www.w3.org/TR/rdf-mt/.

[RFC 2119]
: *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. Network Working Group, S. Bradner. IETF, March 1997, http://www.ietf.org/rfc/rfc2119.txt