# Semantic Web & Ontologies

Pablo Mollá Chárlez

September 24, 2024

## Contents



## 1 Ontology Web Language

OWL (Ontology Web Language) is a powerful knowledge representation language designed for authoring ontologies. Ontologies provide formal definitions for the concepts, relationships, and entities within a specific domain, making it easier to reason about and infer new knowledge from the defined data.

OWL builds upon RDF and RDFS, which offer basic frameworks for describing resources and their relationships. While RDF and RDFS allow us to express simple relationships and class hierarchies, they lack the expressive power needed for more complex inferences. OWL fills this gap by enabling richer relationships, constraints and logical inferences.

## 1.1 Connecting OWL and RDF

Let's give a quick summary.

- **RDF** is a standard for data interchange on the web. It allows the creation of statements in the form subject-predicate-object triples. For instance, "John hasWife Mary".
- **RDFS** extends RDF by providing the means to define classes, subclasses, and properties, allowing us to express hierarchies and basic relationships. However, it is still limited in terms of expressing more complex relationships.
- **OWL** extends RDFS, providing a more expressive framework for defining relationships between entities. OWL enables logical reasoning through constructs like:

## 1.2 OWL Turtle Syntax

Let's explain via examples the different predefined properties of OWL.

### 1.2.1 differentFrom

Property to assert that two individuals are different from each other.

```
# Define properties
:John rdf:type :Person .
:Bill rdf:type :Person .
# Triple
:John owl:differentFrom :Bill .
```

### 1.2.2 sameAs

Property stating that 2 individuals refer to the same entity, even if they have different URIs.

```
# Define properties
:James rdf:type :Person .
:Jim rdf:type :Person .
# Triple
:James owl:sameAs :Jim .
```

### 1.2.3 Restrictions on property values

This datatype is used to specify that a value must be a non-negative integer.

```
# Define properties
:James rdf:type :Person .
# Triple
:James :hasAge "51"^^xsd:nonNegativeInteger .
```

### 1.2.4 Negative property assertions

This construct is used to assert that a specific property does not hold between two individuals.

- **sourceIndividual**: The individual for which the property does not hold.
- **assertionProperty**: The property that does not hold.

- **targetIndividual**: The individual with whom the relationship does not exist.

```
# Define properties
:Bill rdf:type :Person .
:Mary rdf:type :Person .

# Bill has not a wife called Mary
[] rdf:type owl:NegativePropertyAssertion ;
   owl:sourceIndividual :Bill ;
   owl: assertionProperty :hasWife ;
   pwl:targetIndividual :Mary .
```

### 1.2.5 equivalentClass

Property that defines 2 classes as being equivalent, meaning they have the same instances, they share the same individuals.

```
# Define properties
:Human owl:equivalentClass :Person .
```

### 1.2.6 AllDisjointClasses

Declares that several classes are mutually disjoint, meaning on individual can be a member of more than one of them.

```
[] rdf:type owl:AllDisjointClasses ;
   owl:members (:Cat :Dog :Bird) .
```

### 1.2.7 intersectionOf

Specifies that a class is the intersection of several other classes (individuals must belong to all these classes).

```
# Defining a class called PetAndMammal
:PetandMammal rdf:type owl:Class ;
   owl:intersectionOf (:Pet :Mammal) .
```

### 1.2.8 unionOf

Specifies that a class is the union of several other classes (individuals must belong to at least one of these classes).

```
# Defining a class called PetOrWildAnimal
:PetOrWildAnimal rdf:type owl:Class ;
   owl:unionOf (:Pet :WildAnimal) .
```

### 1.2.9 complementOf

Defines a class as the complement of another class (set of individuals not belonging to the given class).

```
# Defining a class called NotCat
:NotCat rdf:type owl:Class ;
   owl:complementOf :Cat .
```

### 1.2.10 Property Restrictions

Used in property restrictions to specify the property to which the restriction applies to.

```
# Person is a class in the ontology
:Person rdf:type owl:Class;
    # Person is a subclass of some anonymous class
    # defined by the restriction inside the brackets
    rdfs:subClassOf [
        # Setting the restriction
        rdf:type owl:Restriction ;
        # Property on which restriction is applied, i.e. people are related to
        # something via the hasPet property
        owl:onProperty :hasPet ;
        # Person must be related to at least one
        # individual from the class Dog
        owl:someValuesFrom :Dog
    ] .
```

In simple terms, this restriction is saying: "A person class includes individuals who have at least one pet, and that pet must be a dog."

### 1.2.11 Restriction Class & someValuesFrom

Used to specify the class on which a restriction applies.

```
# Setting the restriction class
:HasPetDog rdf:type owl:Restriction ;
    owl:onProperty :hasPet ;
    owl:onClass :Dog ;
    owl:someValuesFrom :Dog .
```

In simple terms, this restriction is saying that HasPetDog is a class where individuals must have at least one pet, and that pet must specifically be a dog.

### 1.2.12 Restriction Class & allValuesFrom

Defines a restriction stating that all values for a given property must come from a specific class.

```
:Zoo rdf:type owl:Class ;
    rdfs:subClassOf [
        rdf:type owl:Restriction ;
        owl:onProperty :hostsAnimal ;
        owl:allValuesFrom :Animal
    ] .
```

In simple terms, the restriction is saying that the class Zoo includes (hostsAnimal) just individuals that are animals.

### 1.2.13 Restriction Class & hasValue

Defines a restriction where the property (predicate) must have a specific value.

```
:PersonWithDog rdf:type owl:Class ;
    rdfs:subClassOf [
        rdf:type owl:Restriction ;
        owl:onProperty :hasPet ;
        owl:hasValue :Doggy
    ] .
```

In simple terms, the restriction is saying that the class PersonWithDog includes (hasPet) just individuals that with the specific value Doggy.

#### 1.2.14 Self-Restriction Class

Defines a self-restriction, where the property must point to the individual itself.

```
:SelfLover rdf:type owl:Class ;
    rdfs:subClassOf [
        rdf:type owl:Restriction ;
        owl:onProperty :loves ;
        owl:hasSelf true
    ] .
```

In simple terms, the restriction is saying that the class SelfLover has individuals who love themselves.

#### 1.2.15 Restriction Class & minQualifiedCardinality

Defines the minimum number of values a property can have, with the restriction applied to a specific class.

```
:Parent rdf:type owl:Class ;
    rdfs:subClassOf [
        rdf:type owl:Restriction ;
        owl:onProperty :hasChild ;
        owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
        owl:onClass :Person
    ] .
```

In simple terms, the restriction is saying that the class Parent must have at least ”1” child, and the child must be a ”Person”.

#### 1.2.16 Restriction Class & minQualifiedCardinality

Defines the maximum number of values a property can have, with the restriction applied to a specific class.

```
:OneDogOwner rdf:type owl:Class ;
    rdfs:subClassOf [
        rdf:type owl:Restriction ;
        owl:onProperty :hasPet ;
        owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
        owl:onClass :Dog
    ] .
```

In simple terms, the restriction is saying that the class OneDogOwner must have at most ”1” pet, and the pet must be a ”Dog”.

### 1.2.17 Class Extension: oneOf

Defines a class extension explicitly with a list of its individuals.

```
:WeekDays rdf:type owl:Class ;
    owl:oneOf (:Monday :Tuesday :Wednesday :Thursday :Friday) .
```

In simple terms, this defines the class WeekDays as a class with specific individuals (the days of the week).

### 1.2.18 propertyDisjointWith

Defines 2 properties that cannot have the same values for any pair of individuals.

```
:hasMother owl:propertyDisjointWith :hasFather .
```

In simple terms, the class Parent must have at least "1" child, and the child must be a "Person".

### 1.2.19 inverseOf

Defines 2 properties that are inverses of each other.

```
:hasChild owl:inverseOf :hasParent .
```

In simple terms, this declares that "hasChild" and "hasParent" are inverse properties (if A has a child B, then B must have a parent A).

### 1.2.20 SymmetricProperty

States that a property is symmetric (if A relates to B, then B related to A).

```
:isMarriedTo rdf:type owl:SymmetricProperty .
```

In simple terms, if John isMarriedTo Mary, then Mary isMarriedTo John.

### 1.2.21 AsymmetricProperty

States that a property is asymmetric (if A relates to B, then B doesn't relate to A).

```
:isTallerThan rdf:type owl:AsymmetricProperty .
```

In simple terms, if John isTallerThan Mary, then Mary can't be taller than John.

### 1.2.22 FunctionalProperty

Specifies that a property is functional (i.e. an individual can have at most one values for this property).

```
:hasSSn rdf:type owl:FunctionalProperty .
```

### 1.2.23 InverseFunctionalProperty

Specifies that the inverse of a property is functional (i.e. a value can only have one source).

```
:hasSSn rdf:type owl:InverseFunctionalProperty .
```

This means that each SSN is assigned to only one individual.

#### 1.2.24 TransitiveProperty

Declares that a property is transitive (if A relates to B, and B relates to C, then A relates to C).

```
:ancestorOf rdf:type owl:TransitiveProperty .
```

This means that if "John ancestorOf Bill" and "Bill ancestorOf Sam", then "John ancestorOf Sam".

#### 1.2.25 propertyChainAxiom

Specifies that a property can be defined as the composition of other properties.

```
:hasUncle owl:propertyChainAxiom (:hasParent :hasBrother) .
```

This states that if someone has a parent and that parent has a brother, then the original person has an uncle. The "hasUncle" relationship is inferred by combining "hasParent" and "hasBrother".

#### 1.2.26 hasKey

Specifies that certain properties uniquely identify an individual in the class.

```
:Person rdf:type owl:Class ;
    owl:hasKey (:hasSSN :hasEmail) .
```

This states that individuals in the Person class can be uniquely identified by their SSN and Email.