Ontology Web Language

Esempi e spiegazioni tratti da

https://www.w3.org/TR/owl2-primer/

OWL

OWL2 in generale

OWL 2 (Web Ontology Language) è un linguaggio per creare ontologie per il Web Semantico con un significato definito formalmente.

Le ontologie scritte in OWL 2 contengono classi, proprietà, individui, e letterali; sono scritte in documenti il cui formato è definito dal Web Semantico.

Le ontologie OWL 2 possono essere utilizzate in associazione con documenti RDF, anzi sono essere stesse normalmente codificate come documenti RDF.

Ragionamento automatico

- OWL si basa su logiche computazionali tali che la conoscenza espressa nell'ontologia può essere oggetto di ragionamento automatico da parte di software specifici (*reasoner*) che ne verificano la non contradditorietà e sono in grado di rendere esplicita la conoscenza implicita contenuta nell'ontologia

- La sintassi RDF/XML per OWL è normata da una specifica: "OWL 2 RDF Mapping". Ci sono molte altre sintassi ma questa è l'unica che qualsiasi strumento software per OWL deve essere in grado di gestire.

Struttura di un documento OWL

- OWL 2 non fornisce strumenti che descrivono in maniera prescrittiva la struttura di un documento OWL.

- In particolare, non c'è modo per specificare che una determinata informazione (per esempio, il codice fiscale di una persona) deve essere necessariamente presente.

Terminologia e asserzioni

Una parte tipicamente presente dell'ontologia consiste nella terminologia (o *vocabolario* o *T-box*) che costituisce la conoscenza generale sul dominio dato.

Accanto alla terminologia, l'ontologia può contenere un insieme di asserzioni (A-box) che descrivono entità concrete o specifiche del dominio dato.

Elementi dell'ontologia

- Entità: gli elementi che si riferiscono al mondo reale: il concetto di persona, ma anche John e Jack

- Assiomi: le asserzioni generali contenute nell'ontologia (per esempio, il concetto di persona è un concetto più specifico di quello di essere vivente)

- Espressioni: combinazioni di entità che vanno a formare entità più complesse
  - Relativi alle classi e alle proprietà

Tipi di entità

In OWL 2, si descrivono

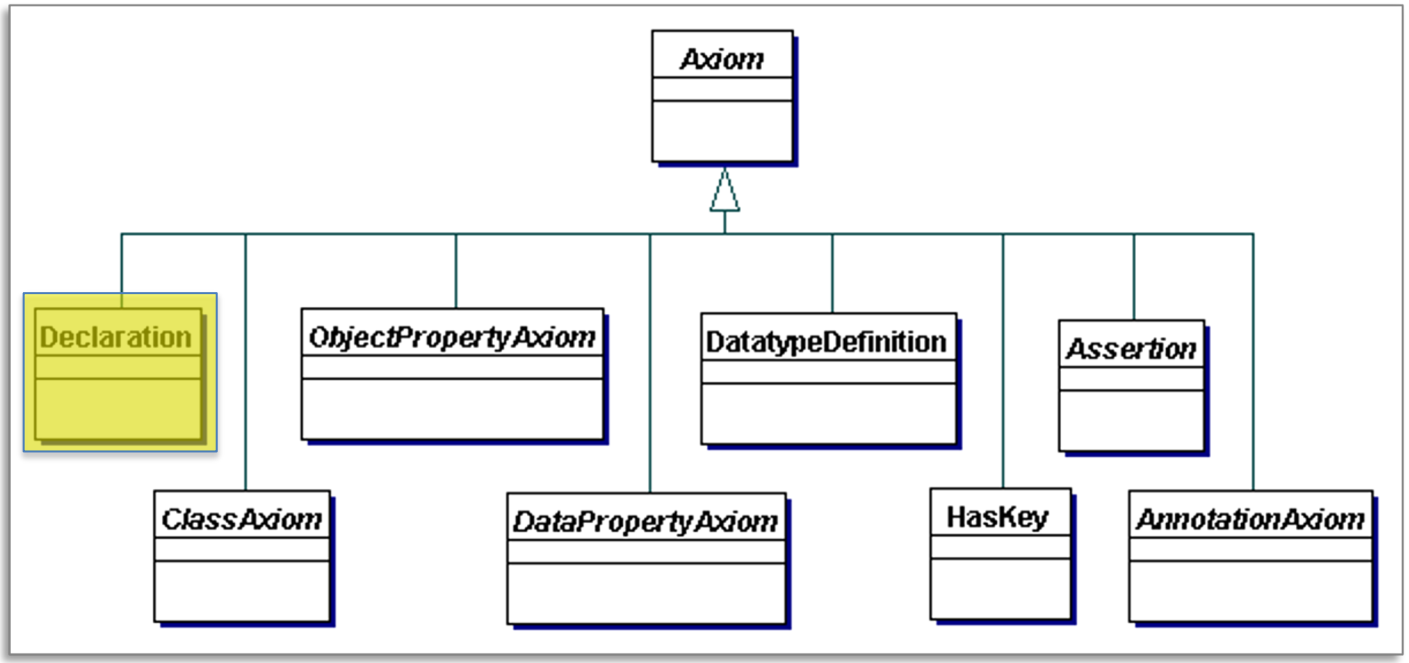
- Gli oggetti come **individui**
- Le categorie come **classi**
- Le relazioni come **proprietà**.

Le proprietà sono suddivise in.

- **Object properties** che collegano individui a individui (come una persona al suo coniuge)
- **Datatype properties** che assegnano un dato a un individuo (per esempio l'età a una persona)
- **Annotation properties** che contengono commenti e descrizioni sulle entità

Axioms

"Axioms in OWL 2 can be declarations, axioms about classes, axioms about object or data properties, datatype definitions, keys, assertions (sometimes also called *facts*), and axioms about annotations."



https://www.w3.org/TR/owl2-syntax/#Class_Expressions

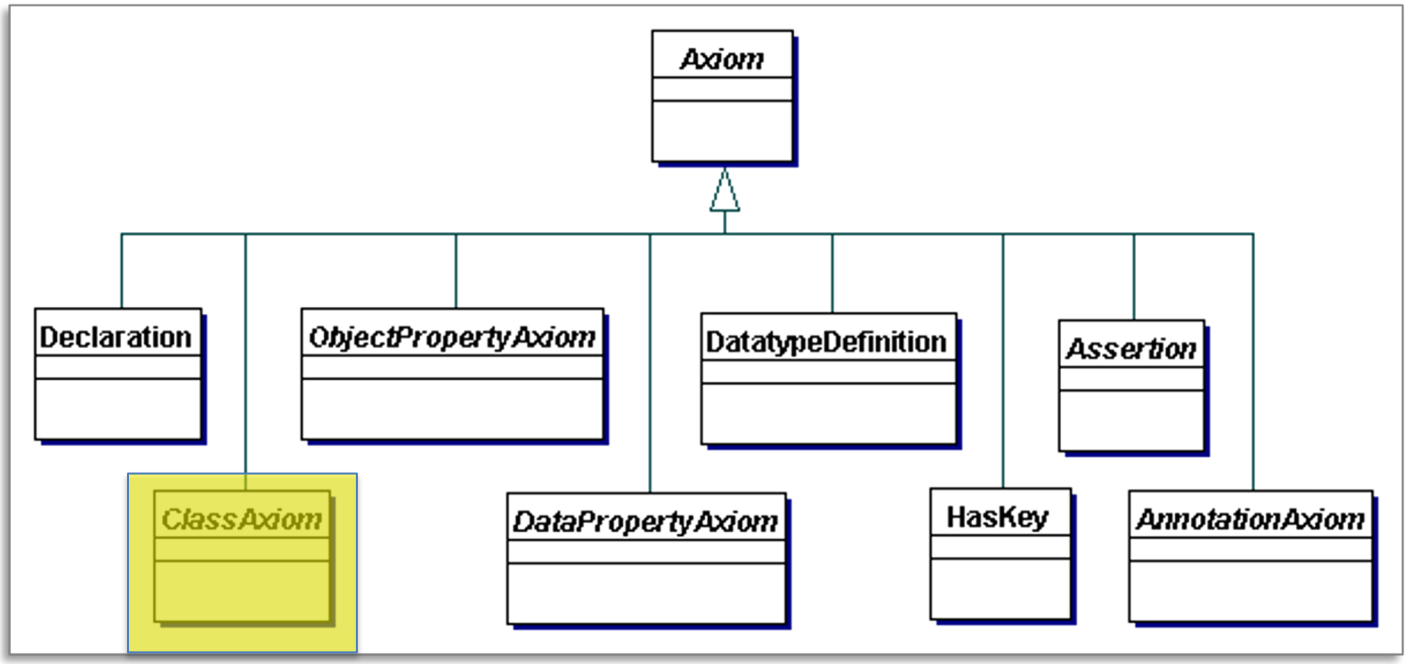
Declarations

"Each IRI used in an OWL 2 ontology needs to be declared in the ontology"

Ci sono due tipi di dichiarazioni:

- Che un certo IRI fa parte dell'ontologia
- A che tipo di entità appartiene l'IRI
  - class
  - datatype
  - object property
  - data property
  - annotation property
  - an individual (NamedIndividual)

Axioms



https://www.w3.org/TR/owl2-syntax/#Class_Expressions

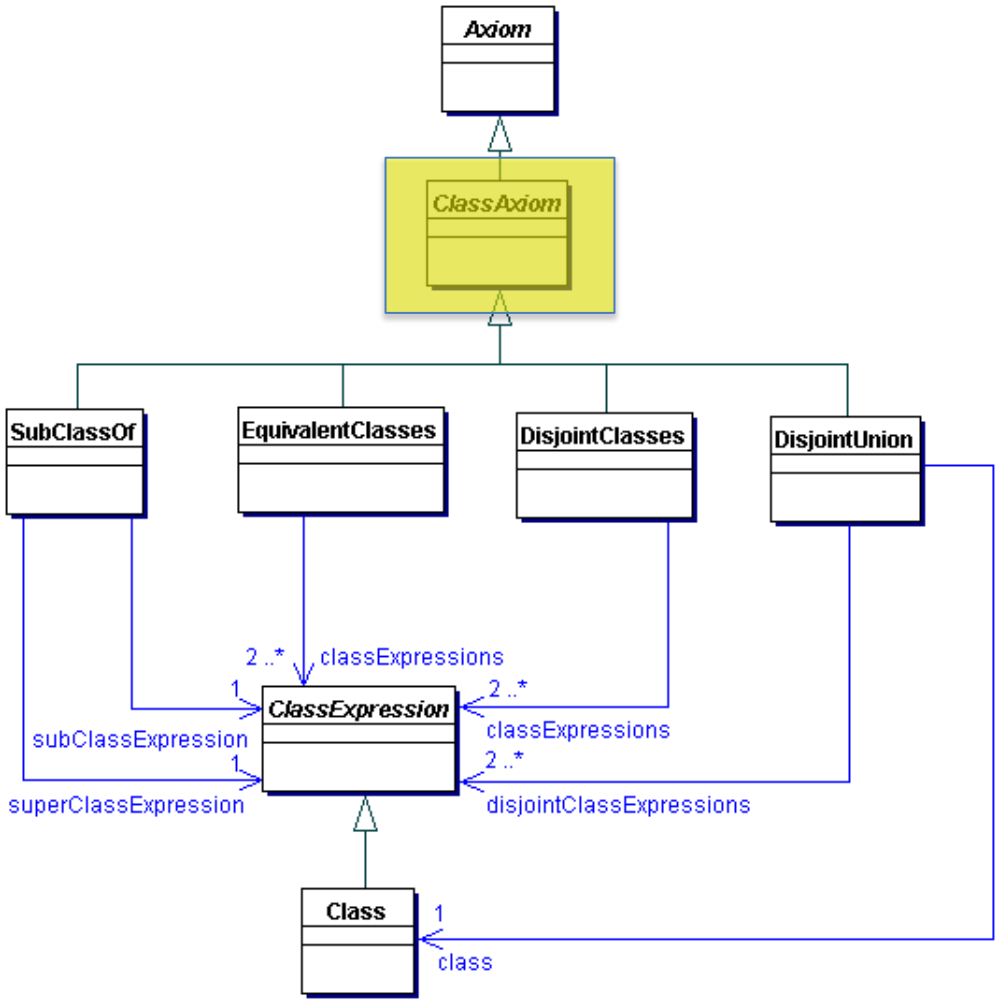
Class axioms: relazioni tra classi

«OWL 2 provides axioms that allow **relationships** to be established between class expressions:»

- The SubClassOf axiom allows one to state that each instance of one class expression is also an instance of another class expression, and thus to construct a hierarchy of classes.
- The EquivalentClasses axiom allows one to state that several class expressions are equivalent to each other.
- The DisjointClasses axiom allows one to state that several class expressions are pairwise disjoint — that is, that they have no instances in common.
- The DisjointUnion class expression allows one to define a class as a disjoint union of several class expressions and thus to express covering constraints.

https://www.w3.org/TR/owl2-syntax/#Class_Expressions

Class axioms: diagramma



https://www.w3.org/TR/owl2-syntax

Class Axioms

- Sottoclasse:
  - SubClassOf(*a:Boy a:Child*)
  - Boy è sottoclasse di Child
- Classi equivalenti:
  - EquivalentClasses(*a:CatOwner a:PadroneDiGatti*)
  - le due classi sono entrambe sottoclasse dell'altra
- Disgiunzione:
  - DisjointClasses(*a:Cat a:Dog*)
  - Cat e Dog sono classi disgiunte
- Disjoint Union:
  - DisjointUnion(*a:Person a:Child a:Adult*)
  - Person ha come sottoclassi le due classi disgiunte Child e Adult

functional style syntax

Espressioni

- Le entità possono essere combinate a formare espressioni usando dei costruttori di classi

  «In OWL 2, classes and property expressions are used to construct class expressions, sometimes also called descriptions, and, in the description logic literature, complex concepts.»

  «Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties; individuals satisfying these conditions are said to be *instances* of the respective class expressions»

- Per esempio tramite congiunzione il concetto di professoressa può essere costruito dai concetti di “docente” e “donna”

- Cf. linguaggi BACK, CLASSIC

Class expressions

- *La classe è la forma più semplice di class expression*

- OWL2 fornisce un insieme di operatori per definire le classi:
  - connettivi booleani and, or e not
  - quantificazione universale e esistenziale
  - restrizioni numeriche
  - enumerazione di individui

Costruire Class Expressions in OWL2

Connettivi

- ObjectIntersectionOf — intersezione
- ObjectUnionOf — unione
- ObjectComplementOf — complemento
- ObjectOneOf — enumerazione

Proprietà (Object properties)

- ObjectSomeValuesFrom — quantificazione esistenziale
- ObjectAllValuesFrom — quantificazione universale
- ObjectMinCardinality — restrizione su cardinalità minima
- ObjectMaxCardinality — restrizione su cardinalità massima
- ObjectExactCardinality — restrizione su cardinalità esatta

Data properties (come sopra)

- DataSomeValuesFrom
- DataAllValuesFrom
- DataMinCardinality
- DataMaxCardinality
- DataExactCardinality

Sintassi funzionale

In sintesi

In OWL si distinguono *classi*, *individui* e *proprietà*

Class expressions (e property expressions) permettono di descrivere classi (e proprietà)

mediante operatori booleani, quantificazione, restrizioni su proprietà, ecc.

Gli *assiomi di classe* permettono di stabilire relazioni tra classi

sottoclasse

equivalenza

unione

partizione

(con effetti rilevanti per il ragionamento automatico)

Il linguaggio OWL

Passo a passo tramite esempi
(sintassi Turtle e RDF/XML)

Classi

- Si definisce una classe dichiarando che essa appartiene al tipo Classe di OWL

:Person rdf:type owl:Class ;

Individui e classi

Mary appartiene alla classe Person

:Mary rdf:type :Person .

John appartiene alla classe Father

:John rdf:type :Father .

Individui e classi (RDF/XML)

Mary appartiene alla classe Person

```
<ClassAssertion>
   <Class IRI="Person"/>
   <NamedIndividual IRI="Mary"/>
 </ClassAssertion>
```

John appartiene alla classe Father

```
<owl:NamedIndividual rdf:about="John”>
    <rdf:type rdf:resource="Father"/>
</owl:NamedIndividual>
```

Un individuo può appartenere a più classi

Classi e sottoclassi

Assiomi di sottoclasse:

:Woman rdfs:subClassOf :Person .

La classe Woman è una sottoclasse della classe Person

→ Tutti gli individui della classe Woman sono anche membri della classe Person (inclusione insiemistica)

RDF/XML

```
<owl:Class rdf:about="Woman">
  <rdfs:subClassOf rdf:resource="Person"/>
 </owl:Class>
```

Gerarchia delle classi

Nelle ontologie OWL, le relazioni di tipo classe-sottoclasse vengono utilizzate per creare delle vere e proprie gerarchie di classi

Non solo Woman è una sottoclasse di Person, *ma Mother è una sottoclasse di Woman*

```
<owl:Class rdf:about="Mother">
  <rdfs:subClassOf rdf:resource="Woman"/>
</owl:Class>
```

→ Ogni classe è sottoclasse di se stessa

Turtle

```
:Mother rdfs:subClassOf :Woman .
```

Classi equivalenti

Due classi possono essere dichiarate equivalenti

Person e Human sono classi equivalenti

```
<owl:Class rdf:about="Person">
  <owl:equivalentClass rdf:resource="Human"/>
</owl:Class>
```

Turtle

```
:Person owl:equivalentClass :Human .
```

E classi disgiunte

Un insieme di classi possono essere dichiarate disgiunte

Uomini e donne sono classi disgiunte

```
[]  rdf:type     owl:AllDisjointClasses ;
    owl:members  ( :Woman  :Man ) .
```

→ Due classi disgiunte non possono avere individui in comune

RDF

```
<owl:AllDisjointClasses>
  <owl:members rdf:parseType="Collection">
    <owl:Class rdf:about="Woman"/>
    <owl:Class rdf:about="Man"/>
  </owl:members>
 </owl:AllDisjointClasses>
```

Object properties

- Si definisce una Object Property asserendo che essa appartiene al tipo ObjectProperty di OWL

```
:hasSpouse rdf:type owl:ObjectProperty ;
```

Object properties

La proprietà hasWife ha come domain la classe Man e come target la classe Woman

:hasWife rdfs:domain :Man ;
rdfs:range :Woman .

→ Se non specificato diversamente, la proprietà si applica a tutte le classi (sia per domain sia per range)

RDF/XML

```
<owl:ObjectProperty rdf:about="hasWife">
    <rdfs:domain rdf:resource="Man"/>
    <rdfs:range rdf:resource="Woman"/>
  </owl:ObjectProperty>
```

Utilizzo delle proprietà

Le proprietà di tipo Object Property rappresentano relazioni tra classi (e quindi, si asseriscono degli individui)

John è sposato (hasWife) con Mary

:John :hasWife :Mary

```
<rdf:Description rdf:about="John">
  <hasWife rdf:resource="Mary"/>
 </rdf:Description>
```

(utilizzo della proprietà)

Gerarchie di proprietà

E’ possibile descrivere una proprietà come sottoproprietà di un’altra

La proprietà hasWife è una sottoproprietà di hasSpouse

```
:hasWife rdfs:subPropertyOf :hasSpouse .
```

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
 </owl:ObjectProperty>
```

→ se una proprietà vale tra due individui, vale anche la proprietà più generale

Individui: identità e differenza

James e Jim sono lo stesso individuo

```
:John owl:differentFrom :Bill .
```

```
<rdf:Description rdf:about="James">
  <owl:sameAs rdf:resource="Jim"/>
</rdf:Description>
```

Mentre *John e Bill sono individui diversi*

```
:John owl:sameAs :Bill .
```

```
<rdf:Description rdf:about="John">
  <owl:differentFrom rdf:resource="Bill"/>
</rdf:Description>
```

→ Same as viene usato per allineare basi di conoscenza diverse

Data properties

Le data property hanno come domain una classe e come range un tipo di dato

```
:hasAge rdf:type owl:DatatypeProperty ;
```

La proprietà hasAge collega una persona alla sua età, espressa come intero.

John ha 51 anni

```
:hasAge  rdfs:domain  :Person ;
         rdfs:range   xsd:nonNegativeInteger .

:John  :hasAge  51 .
```

(abbreviato per "51"^^xsd:integer)

RDF/XML

```
<owl:DatatypeProperty rdf:about="hasAge">
  <rdfs:domain rdf:resource="Person"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
 </owl:DatatypeProperty>


<Person rdf:about="John">
  <hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">51</hasAge>
 </Person>
```

Restrizioni

- Le restrizioni sono uno dei meccanismi principali per definire nuove classi a partire da quelle esistenti

- Due tipi di restrizioni:

  - Restrizioni su **classi** mediante operatori insiemistici (intersezione-and, unione-or, complemento-not)

  - Restrizioni poste su **proprietà** (esistenziale, universale, sulla cardinalità)

Intersezione

L'intersezione permette di definire una classe come intersezione di due classi

```
:Mother   owl:equivalentClass  [
              rdf:type  owl:Class ;
              owl:intersectionOf  ( :Woman :Parent )
 ] .
```

La classe Mother è l'intersezione di Woman e Parent

→ Tutti gli individui della classe Mother appartengono anche alle classi Woman e Parent

RDF/XML

```
<owl:Class rdf:about="Mother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Woman"/>
        <owl:Class rdf:about="Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
```

Complemento

Una classe può essere definita come complemento di un'altra

La classe ChildlessPerson è complemento della classe Parent

```
:ChildlessPerson  owl:equivalentClass  [
   rdf:type          owl:Class ;
   owl:intersectionOf  ( :Person
                  [ rdf:type        owl:Class ;
                    owl:complementOf  :Parent ] )
 ] .
```

→ Un individuo non può appartenere a una classe E al suo complemento

RDF/XML

```
<owl:Class rdf:about="ChildlessPerson">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Person"/>
        <owl:Class>
          <owl:complementOf rdf:resource="Parent"/>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Restrizioni fuori da equivalenza tra classi

```
<owl:Class rdf:about="Grandfather">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Man"/>
        <owl:Class rdf:about="Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

```
:Grandfather  rdfs:subClassOf  [
  rdf:type          owl:Class ;
  owl:intersectionOf  ( :Man  :Parent )
] .
```

Manca owl:equivalentTo!

Quantificatori e proprietà

someValuesFrom esprime la quantificazione esistenziale

```
:Parent  owl:equivalentClass  [
    rdf:type          owl:Restriction ;
    owl:onProperty      :hasChild ;
    owl:someValuesFrom  :Person
 ] .
```

→ allValuesFrom = quantificatore esistenziale

RDF/XML

```
<owl:Class rdf:about="Parent">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasChild"/>
      <owl:someValuesFrom rdf:resource="Person"/>
    </owl:Restriction>
  </owl:equivalentClass>
 </owl:Class>
```

Restrizioni numeriche su proprietà

```
:John  rdf:type  [
  rdf:type                        owl:Restriction ;
  owl:maxQualifiedCardinality  "4"^^xsd:nonNegativeInteger ;
  owl:onProperty          :hasChild ;
  owl:onClass                :Parent
 ] .
```

→ **minQualifiedCardinality** permette di porre un vincolo sulla cardinalità minima

RDF/XML

```
rdf:Description rdf:about="John">
  <rdf:type>
   <owl:Restriction>
    <owl:maxQualifiedCardinality rdf:datatype="http://www.w3.org/2001/
    XMLSchema#nonNegativeInteger">
     4
    </owl:maxQualifiedCardinality>
    <owl:onProperty rdf:resource="hasChild"/>
    <owl:onClass rdf:resource="Parent"/>
   </owl:Restriction>
  </rdf:type>
 </rdf:Description>
```

Necessario e sufficiente

Le classi definite come equivalenti a un certo insieme di restrizioni sono denominate classi definite

Le restrizioni individuano le condizioni *necessarie* e *sufficienti* per l'appartenenza alla classe

Senza l'utilizzo del costrutto EquivalentTo si possono associare a una classe solo proprietà necessarie ma *non sufficienti*

Solo le classi definite permettono determinate forme di ragionamento!

Classi enumerate

E’ possibile definire una classe come enumerazione di individui

```
:MyBirthdayGuests  owl:equivalentClass  [
  rdf:type   owl:Class ;
  owl:oneOf  ( :Bill  :John  :Mary )
] .
```

La classe MyBirthdayGuests è definita come insieme degli individui Bill, John e Mary

RDF/XML

```
<owl:Class rdf:about="MyBirthdayGuests">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:about="Bill"/>
        <rdf:Description rdf:about="John"/>
        <rdf:Description rdf:about="Mary"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
```

Descrivere le proprietà

- Simmetriche
  - Es. *coniugeDi*
- Funzionali
  - Es. *haPadre*
- Inverse
  - Es. *figlioDi*
- Riflessive
  - Es. *conosce*
- Transitive
  - Es. *contiene*

Modellazione: prime osservazioni

- Non confondere la relazione di sottoclasse con la relazione mereologica part-of!
  - Si deve usare una proprietà per rappresentare le relazioni parte-tutto, non un assioma di sottoclasse

- L'ontologia deve essere leggibile
  - Organizzare le classi in una *gerarchia*
  - Attribuire alle entità *nomi* che hanno senso secondo il senso comune o per l'esperto
  - Associare *domain* e *range* alle proprietà