Lenguaje de ontologías

Ingeniería de Sistemas

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

OWL

- Web Ontology Language o Lenguaje de Ontologías para la Web es un lenguaje de etiquetado semántico para publicar y compartir ontologías en la Web.
- Es una recomendación del W3C, y puede usarse para representar ontologías de forma explícita. Permite definir el significado de términos en vocabularios y las relaciones entre aquellos términos.
- Es una extensión del lenguaje RDF y emplea las tripletas de RDF, pero tiene más poder expresivo.

OWL

- Es un lenguaje diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, a diferencia de situaciones donde el contenido solamente necesita ser presentado.
- Surge a partir del lenguaje DAML-OIL y es mucho más potente.
- Al igual que OIL, se estructura en capas que difieren en la complejidad y puede ser adaptado a las necesidades de cada usuario, al nivel de expresividad que se precise y a los distintos tipos de aplicaciones existentes (motores de búsqueda, agentes, etc.)

OWL

- Clases y propiedades
- Condiciones necesarias y suficientes
- Constructores de clases
  - IntersectionOf
  - UnionOf
  - ComplementOf
  - somevaluesFrom
  - allValuesFrom
  - minCardinality
  - maxCardinality
  - Cardinality
  - one of
  - hasValue
- Axiomas de clases
  - SubClassOf
  - sameClassAs
  - disjointWith
- Axiomas individuales
  - SameIndividualAs
  - DifferentindividualFrom
- Axiomas sobre propiedades
  - SubPropertyOf
  - Domain
  - Range
  - InverseOf
  - FunctionalProperty
  - TransitiveProperty
  - SymmetricProperty
  - InverseFunctionalProperty

Clases y propiedades

- Las taxonomías se constituyen principalmente mediante relaciones subClassOf.
- Las propiedades son relaciones binarias entre la clase que la utiliza y el valor que contiene:
  - Object property: relaciones entre elementos de dos clases (entre instancias)
  - Datatype property: relaciones entre elementos de clases y valores concretos
- La utilización de razonadores permitirá deducir información y/o relaciones adicionales a las expresadas explícitamente (nuevas relaciones taxonómicas, equivalencias entre clases o inconsistencias).

Condiciones necesarias y suficientes

- Condiciones necesarias de pertenencia a una clase. Son propiedades que un individuo debe poseer si es miembro de la clase.

```
SubClassOf (Dog Mammal)
<owl:Class rdf:ID="Dog">
  <rdfs:subClassOf rdf:resource="#Mammal"/>
</owl:Class>
```

- Un individuo de la clase Dog debe serlo también de la clase Mammal.

Condiciones necesarias y suficientes

- Condiciones suficientes de pertenencia a una clase. Son propiedades que un individuo debe tener para que pueda ser reconocido como miembro de una clase.
- Uso de sameClassAs

```
EquivalentClasses (CarOwner intersectionOf(Person restriction(owns someValuesFrom =
    Car)))
<owl:Class rdf:ID="CarOwner">
<owl:sameClassAs>
 <owl:Class>
  <owl:intersectionOf rdf:parsetype="owl:collection">
   <owl:Class rdf:about="Person"/>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#owns"/>
      <owl:someValuesFrom rdf:resource="#Car"/>
   </owl:Restricton/>
  </owl:intersectionOf>
 </owl:Class>
</sameClassAs>
<owl:Class>
```

- Todos los miembros de la clase Carowner deben ser personas que posean un coche, y viceversa.

Clases definidas

- Son clases que contienen condiciones necesarias y suficientes.
- Facilitan la utilización de razondores para inferir nueva información.
- Por ejemplo, si incluimos:

```
SubClassOf (CarOwner Adult)
<owl:Class rdf:ID="CarOwner">
        <owl:subClassOf rdf:resource="#Adult">
</owl:Class>
```

- Todos los Carowners son adultos. No es condición suficiente de pertenencia, así que no tendríamos que saber apriori que un individuo es adulto para poder reconocerlo como Carowner. Pero si un individuo es reconocido como Carowner, se puede deducir que es adulto.

Constructores de clases

- intesectionOf: Combina dos clases formando su intersección.

```
intersectionOf (Woman Driver)
<owl:Class>
   <owl:intersectionOf rdf:parseType="owl:collection">
     <owl:Class rdf:about="Woman"/>
     <owl:Class rdf:about="Driver"/>
   </owl:intersectionOf>
</owl:Class>
```

Constructores de clases

- unionOf: Combina dos clases formando su unión.

```
unionOf (Cat Dog)
<owl:Class>
  <owl:unionOf rdf:parseType="owl:collection">
       <owl:Class rdf:about="Cat"/>
       <owl:Class rdf:about="Dog"/>
  </owl:unionOf>
</owl:Class>
```

- Son miembros de esta clase los individuos que son miembros de la clase Gato o de la clase Perro.

Constructores de clases

- complementOf: Describe la colección de individuos de los que se sabe que no son instancias de la clase.

```
restriction (eats allValuesFrom = complementOf (AnimalProduct))
<owl:Restriction>
    <owl:onProperty rdf:resource="#eats"/>
    <owl:allValuesFrom>
      <owl:Class>
               <owl:complementOf rdf:resource="#AnimalProduct"/>
      </owl:Class>
    </owl:allValuesFrom>
</owl:Restriction>
```

Constructores de clases

- OWL asume una teoría de suposición de mundo abierto, “el que no podamos demostrar que un individuo sea una instancia de X no implica que no sea una instancia de X”.

Constructores de clases: Restricciones

- someValuesFrom: Describe aquellos individuos que tienen una relación con otros individuos de una clase determinada.

```
restriction (owns someValuesFrom = Car)
<owl:Restriction>
 <owl:onProperty rdf:resource="#owns"/>
 <owl:someValuesFrom rdf:resource="#Car"/>
</owl:Restriction>
```

- Los individuos de esta clase son los que están relacionados con al menos un individuo de la clase Car mediante la propiedad owns.
- Este constructor no restringe otras características que pudieran estar presentes (ej. Un individuo podría tener varios coches, y también poseer otros objetos).

Constructores de clases: Restricciones

- allValuesFrom: Son aquellos individuos con relaciones que están restringidas a una clase determinada.

```
restriction (owns allValuesFrom = Car)
<owl:Restriction>
    <owl:onProperty rdf:resource="#owns"/>
    <owl:allValuesFrom rdf:resource="#Car"/>
</owl:Restriction>
```

- Los individuos de esta clase sólo pueden relacionarse con objetos de la clase Car cuando usan la relación owns.
- No exige la existencia de la relación. Sólo que si la propiedad existe, entonces el valor de la propiedad debe ser un objeto de la clase especificada.

Constructores de clases: Restricciones

- minCardinality

```
restriction (children minCardinality(2))
 <owl:Restriction>
    <owl:onProperty rdf:resource="#children"/>
    <owl:minCardinality>2</owl:minCardinality>
</owl:Restriction/>
```

- maxCardinality

```
restriction (children maxCardinality(2))
<owl:Restriction>
    <owl:onProperty rdf:resource="#children"/>
    <owl:maxCardinality>2</owl:maxCardinality>
</owl:Restricton/>
```

- cardinality

```
restriction (children cardinality(2))
<owl:Restriction>
    <owl:onProperty rdf:resource="#children"/>
    <owl:cardinality>2</owl:cardinality>
</owl:Restricton/>
```

Constructores de clases: Restricciones

- oneOf: Facilita la definición de clases por extensión, mediante la enumeración de las instancias de la clase.

```
EnumeratedClass (DaysOfTheWeek Mon Tue Wed Thu Fri
Sat Sun)
<owl:Class rdf:ID="DaysOfTheWeek">
   <owl:sameClassAs>
    <owl:oneOf rdf:parseType="owl:collection">
        <owl:Thing rdf:about="#Mon"/>
        <owl:Thing rdf:about="#Tue"/>
        <owl:Thing rdf:about="#Wed"/>
        <owl:Thing rdf:about="#Thu"/>
        <owl:Thing rdf:about="#Fri"/>
        <owl:Thing rdf:about="#Sat"/>
        <owl:Thing rdf:about="#Sun"/>
    </owl:oneOf>
   </owl:sameClassAs>
</owl:Class>
```

Constructores de clases: Restricciones

- hasValue: Describe aquellos individuos que tienen un determinado valor en una propiedad concreta.

```
EquivalentClasses (
        Italian
        intersectionOf (Person restriction (bornIn value = Italy)))
<owl:Class rdf:ID="Italian">
        <owl:sameClassAs>
          <owl:Class>
                        <owl:intersectionOf rdf:parsetype="owl:collection">
                                <owl:Class rdf:about="Person"/>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#bornIn"/>
                                        <owl:hasValue rdf:resource="#Italy"/>
                                </owl:Restriction/>
                        </owl:intersectionOf>
          </owl:Class>
        </sameClassAs>
<owl:Class>
```

Axiomas de clases

- Los axiomas de clases establecen relaciones entre clases.
- subClassOf: Indican que todas las instancias de una clase deben serlo también de la otra.

```
SubClassOf (Dog Mammal)
<owl:Class rdf:ID="Dog">
   <rdfs:subClassOf rdf:resource="#Mammal"/>
</owl:Class>
```

Axiomas de clases

```
SubClassOf (
   restriction (drives someValuesFrom = Bus)
   restriction (reads allValuesFrom = Tabloid)
)
<owl:Restriction>
   <owl:onProperty rdf:resource="#drives"/>
   <owl:someValuesFrom rdf:resource="#Bus"/>
   <rdfs:subClassOf>
    <owl:Restriction>
           <owl:onProperty rdf:resource="#reads"/>
           <owl:allValuesFrom rdf:resource="#Tabloid"/>
    </owl:Restricton/>
   </rdfs:subClassOf>
</owl:Restricton/>
```

Axiomas de clases

- sameClassAs: Establece la equivalencia entre dos clases.
- Igual que con subClassOf las clases pueden ser primitivas o descripciones complejas.

```
EquivalentClasses (Man Bloke)
<owl:Class rdf:ID="Man">
   <owl:sameClassAs rdf:resource="#Bloke"/>
</owl:Class>
```

- disjointWith: Establece que las clases implicadas son disjuntas.

```
DisjointClasses (Cat Dog)
<owl:Class rdf:ID="Cat">
   <owl:disjointWith rdf:resource="#Dog"/>
</owl:Class>
```

Axiomas sobre individuos

- sameIndividualAs: Nos permite referirnos al mismo individuo utilizando nombres distintos.

```
SameIndividual (SeanKBechhofer SeanBechhofer)
<owl:Individual rdf:ID="SeanKBechhofer">
   <owl:sameIndividualAs rdf:about="#SeanBechhofer"/>
</owlIndividual>
```

- differentIndividualFrom: Establece que dos individuos son distintos.

```
DifferentIndividuals (Tibbs Ginger)
<owl:Individual rdf:ID="Tibbs">
   <owl:differentIndividualFrom rdf:about="#Ginger"/>
</owlIndividual>
```

Axiomas sobre propiedades

- Los axiomas sobre propiedades establecen propiedades generales sobre propiedades como las jerarquías de propiedades, rangos y dominios.
- subPropertyOf: Establece que una propiedad es una subpropiedad de otra.

```
ObjectProperty (hasPet super = owns)
<owl:ObjectProperty rdf:ID="hasPet">
   <rdfs:subPropertyOf rdf:resource="#owns"/>
</owl:ObjectProperty>
```

Axiomas sobre propiedades

- Domain: Indica la(s) clase(s) que pueden incluir esta propiedad.

```
ObjectProperty (owns domain = Person)
<owl:ObjectProperty rdf:ID="owns">
   <rdfs:domain rdf:resource="#Person"/>
</owl:ObjectProperty>
```

- Si ObjectProperty (owns domain = Person) y Individual(Mike (owns Tibbs)) → Mike is an instance of Person (no se deduciría un error a pesar de que no hayamos indicado explícitamente que Mike es una Persona.

Axiomas sobre propiedades

- Range: Indica la clase a la que debe pertenecer el valor de la propiedad.

```
ObjectProperty (eats range = Food)
<owl:ObjectProperty rdf:ID="eats">
   <rdfs:range rdf:resource="#Food"/>
</owl:ObjectProperty>
```

- inverseOf

```
ObjectProperty (owns inverseOf = ownedBy)
<owl:ObjectProperty rdf:ID="owns">
   <owl:inverseOf rdf:resource="#ownedBy"/>
</owl:ObjectProperty>
```

Axiomas sobre propiedades

- FunctionalProperty: Si una propiedad es funcional cualquier individuo que la use, el valor debe ser único. Es equivalente a decir que la cardinalidad máxima es 1 y la mínima 0.

```
ObjectProperty (ownedBy Functional)
<owl:ObjectProperty rdf:ID="ownedBy">
   <rdf:type>
    <owl:FunctionalProperty/>
   </rdf:type>
</owl:ObjectProperty>
```

- Si (individual (Tibbs (ownedBy Fred)) y Individual (Tibbs (ownedBy Jim))) entonces se puede deducir que Fred y Jim son el mismo individuo.

Axiomas sobre propiedades

- TransitiveProperty: Por ejemplo, cualquier ascendiente de mis ascendientes también son mis ascendientes.

```
ObjectProperty (ancestor Transitive)
<owl:ObjectProperty rdf:ID="ancestor">
   <rdf:type>
    <owl:FunctionalProperty/>
   </rdf:type>
</owl:ObjectProperty>
```

- Para mantener la decibilidad del lenguaje (y que se pueda utilizar un razonador), las propiedades transitivas (ni las superpropiedades de una propiedad transitiva) no pueden ser utilizadas en restricciones de cardinalidad.

Axiomas sobre propiedades

- SymmetricProperty: Si x se relaciona con y mediante la relación simétrica r, entonces también se verifica que y está relacionada con x vía r.

```
ObjectProperty (friend Symmetric)
<owl:ObjectProperty rdf:ID="friend">
   <rdf:type>
    <owl:SymmetricProperty/>
   </rdf:type>
</owl:ObjectProperty>
```

Axiomas sobre propiedades

- InverseFunctionalProperty: Para una propiedad que sea funcional inversa, su inversa es funcional. Significa que conociendo el valor de la propiedad podemos identificar unívocamente al individuo.

```
ObjectProperty (isMotherOf InverseFunctional)
<owl:DataProperty rdf:ID="isMotherOf">
   <rdf:type>
    <owl:InverseFunctionalProperty/>
   </rdf:type>
</owl:DataProperty>
```

- Por ejemplo, si hasMother es la inversa de isMotherOf, has Mother es Functional.
- Funcional Inversa es equivalente a decir que su propiedad inversa es funcional.

OWL Y DL (Lógica descriptiva)

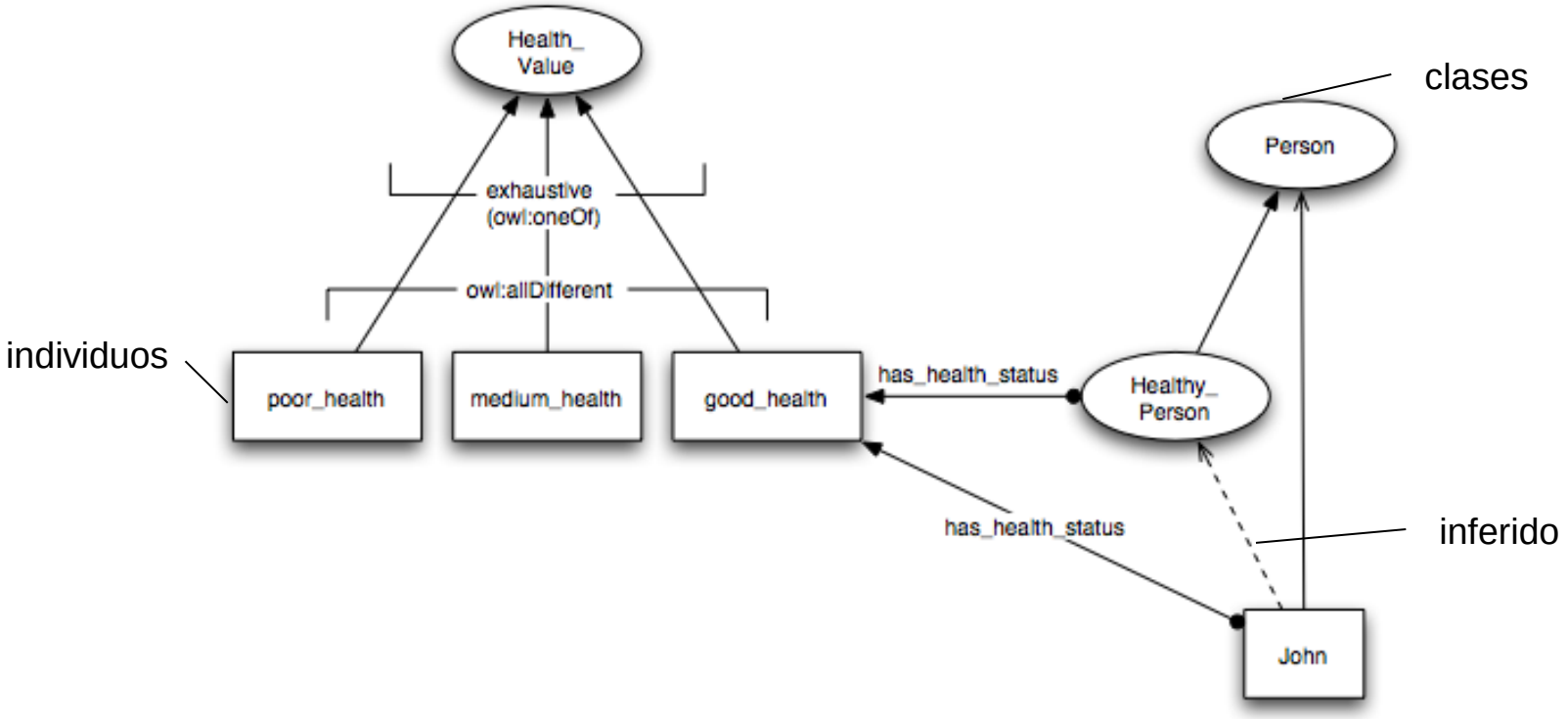
Constructor OWL	Representación DL	Ejemplo
owl:equivalentTo (C,D)	$C \equiv D (C \sqsubseteq D \; y \; D \sqsubseteq C)$	$Persona \equiv Humano$
rdfs:subClassOf (C,D)	$C \sqsubseteq D$	$Padres \sqsubseteq Persona$
owl:complementOf (C,D)	$C \equiv \neg D (negacion)$	$Varon \equiv \neg Mujer$
owl:disjointWith (C,D)	$C \sqsubseteq \neg D$	$Padre \sqsubseteq \neg Madre$
owl:intersectionOf (C,D)	$C \sqcap D (conjuncion)$	$Padres \sqcap Varon$
owl:unionOf (C,D)	$C \sqcup D (disjuncion)$	$Padre \sqcup Madre$
owl:oneOf (I1, I2)	$\{I_1\} \sqcup \{I_2\}$	$\{Juan\} \sqcup \{Maria\}$
owl:someValuesFrom(P,C)	$\exists P.C (existencial)$	$\exists tieneHijo.Hija$
owl:allValuesFrom(P,C)	$\forall P.C (universal)$	$\forall tieneHijo.Hijo$
owl:hasValue (P,I1)	$\exists P.\{I_1\}$	$\exists tieneHijo.\{Juan\}$
owl:cardinality(P,n)	$= n.P$	$= 2.tienePadres$
owl:minCardinality(P,n)	$\geq n.P$	$\geq 1.tieneHija$
owl:maxCardinality(P,n)	$\leq n.P$	$\leq 2.tieneHijos$

"value partitions“ y "value sets"

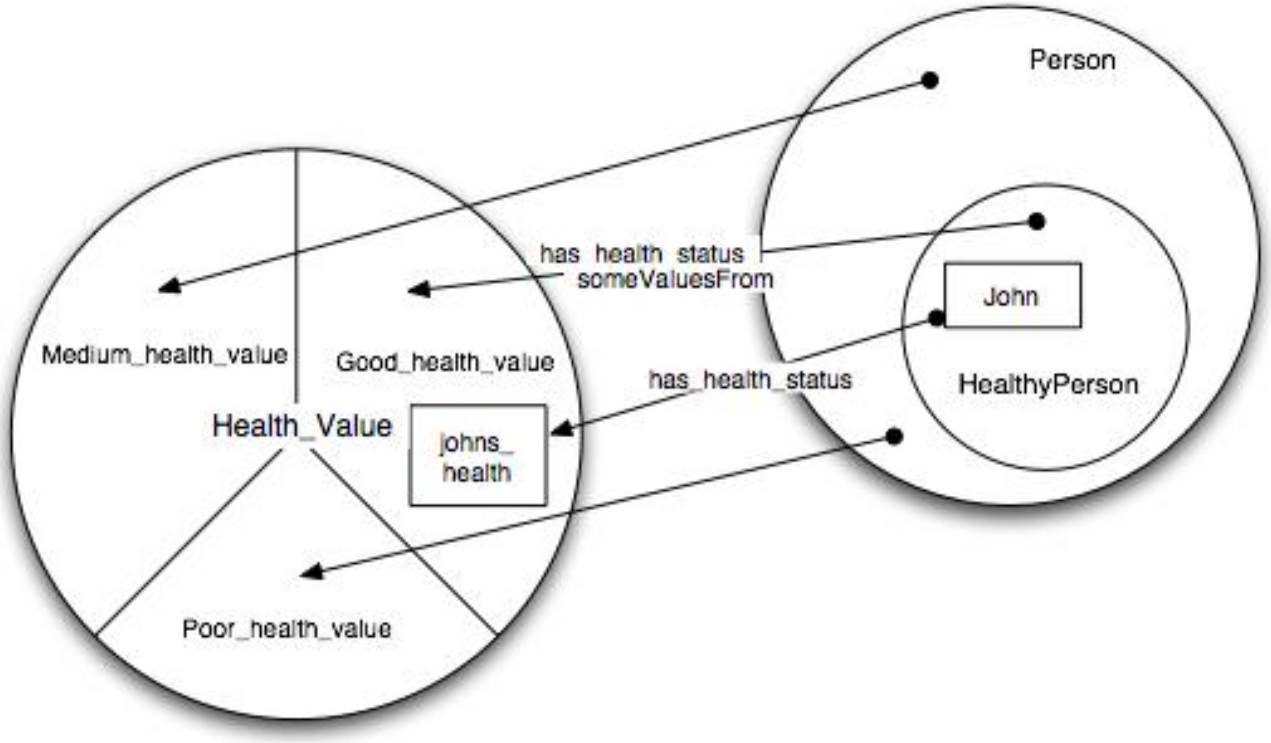
- Es habitual que las ontologías incluyan atributos con listas de valores ‘permitidos’ que se corresponderían con tipos de datos enumerados (ej. Tamaño: grande, mediano, pequeño)

- Opciones:
  - Valores como individuos de una misma clase que refleja el ‘tipo’ del dato
  - Definiendo una subclase para cada valor del ‘tipo’ del dato
  - Con un ‘dataType’ de OWL

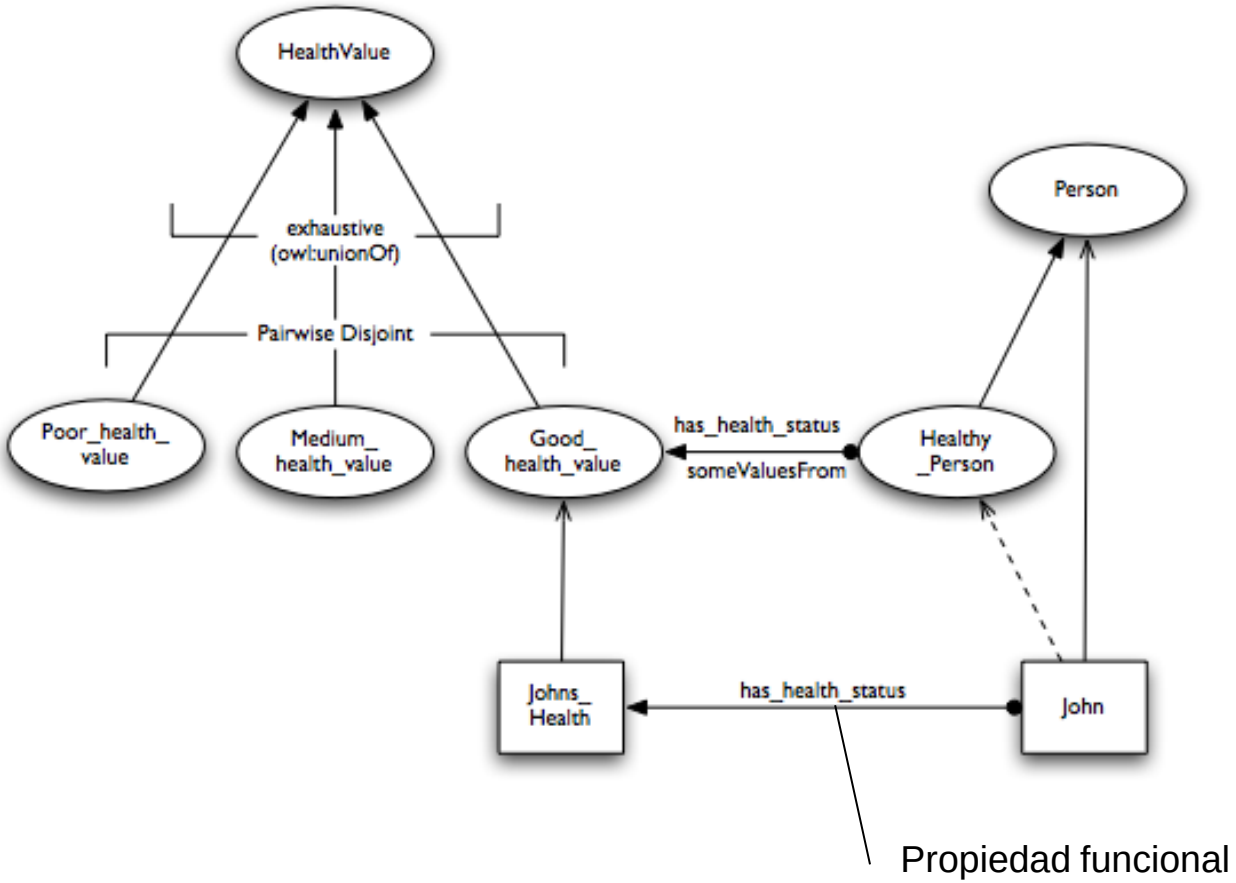
Valores como conjuntos de individuos



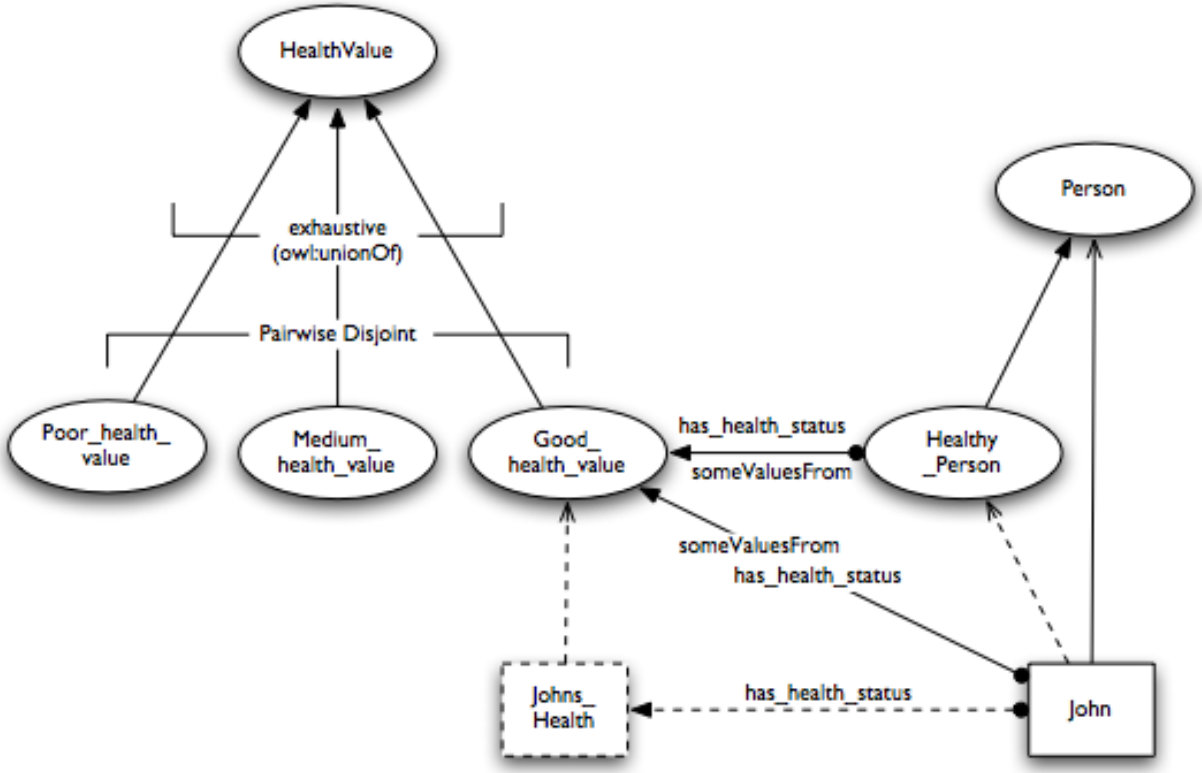
Valores como subclases



Valores como subclases



Valores como subclases



Protege OWL API

- Tutorial y guía de programación en
  - http://protege.stanford.edu/plugins/owl/api/guide.html