



Web Ontology Language (OWL)

Máster Universitario en Inteligencia Artificial

Mikel Egaña Aranguren

3205 Facultad de Informática
Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo
28660 Boadilla del Monte
Spain

<http://www.oeg-upm.net>

megana@fi.upm.es
<http://mikeleganaaranguren.com>



<http://delicias.dia.fi.upm.es/wiki/index.php/MasterRD11-12>

Week 9. 8/11/2011. OWL (Mikel Egaña Aranguren)

<http://mikeleganaaranguren.wordpress.com/teaching/>

Introduction to OWL

OWL syntaxes

OWL semantics

Reasoning

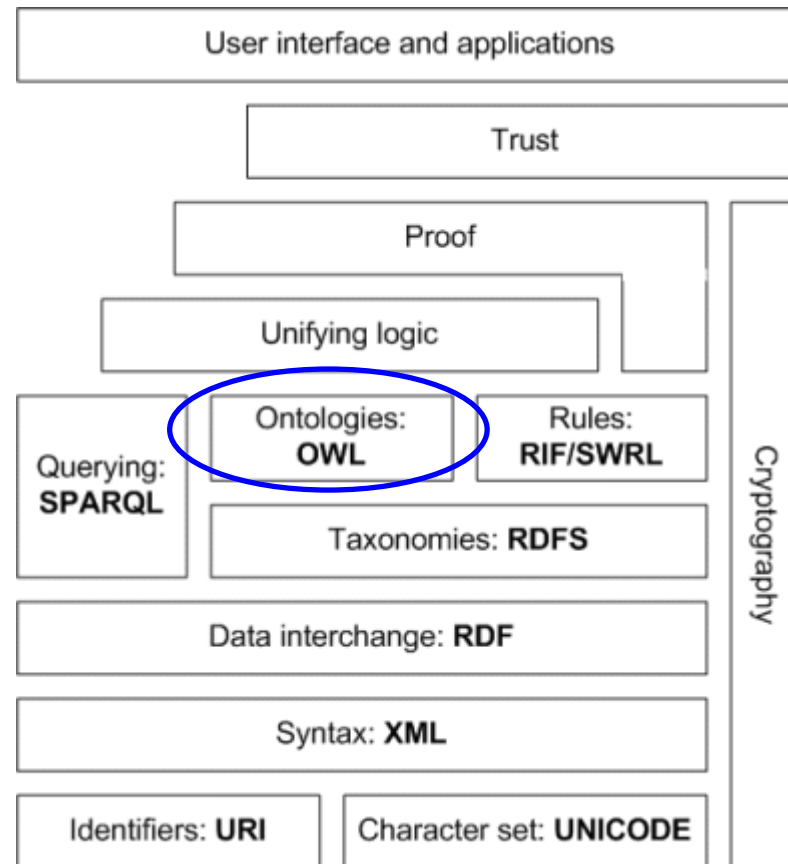
OWL tools

Assignment

(Comment on last assignment)

Introduction to OWL

OWL is a Knowledge Representation language proposed by the W3C as a standard to codify ontologies in a prospective Semantic Web



OWL is based in Description Logics

We can represent a knowledge domain computationally in an OWL ontology, in order to:

Apply automated reasoning: infer “new” knowledge, queries, consistency, classify entities against the ontology, ...

Integrate knowledge from different resources

Everything about OWL 2:

<http://www.w3.org/standards/techs/owl>

Document overview: <http://www.w3.org/TR/2009/REC-owl2-overview-20091027/>

Primer: <http://www.w3.org/TR/2009/REC-owl2-primer-20091027/>

Manchester OWL + Protégé tutorial (Copied some examples :-):

<http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/>

OWL versions:

“OWL 1”: OWL lite, OWL DL, OWL Full

OWL 1.1

OWL 2 profiles: OWL EL, OWL QL, OWL RL

OWL syntaxes

For computers: RDF/XML, OWL/XML, ...

RDF/XML:

```
<owl:Class rdf:about="#arm">  
  <rdfs:subClassOf>  
    <owl:Restriction>  
      <owl:onProperty rdf:resource="#part_of"/>  
      <owl:someValuesFrom rdf:resource="#body"/>  
    </owl:Restriction>  
  </rdfs:subClassOf>  
</owl:Class>
```

For humans: Manchester OWL Syntax, functional, ...

Manchester OWL Syntax: **arm** subClassOf **art_of** some **body**

http://www.co-ode.org/resources/reference/manchester_syntax/

OWL semantics

An OWL ontology comprises:

Entities: the named elements from the knowledge domain, created by the ontology creator. Entities are identified using URIs (To work in a web setting)

Axioms: axioms relate the entities to each other using the OWL logic vocabulary

An OWL ontology can import other ontologies ([owl:import](#)): the entities of the imported ontology can be referenced by axioms on our ontology

OWL is “Axiom-centric”

Entities only “exist” as part of axioms, and therefore the only way of creating an entity in an ontology is by adding an axiom that refers to it. We cannot create the class *A*, but we can state that *A subClassOf owl:Thing*

!!!

There are three types of entities in an OWL ontology:

Individuals

Properties

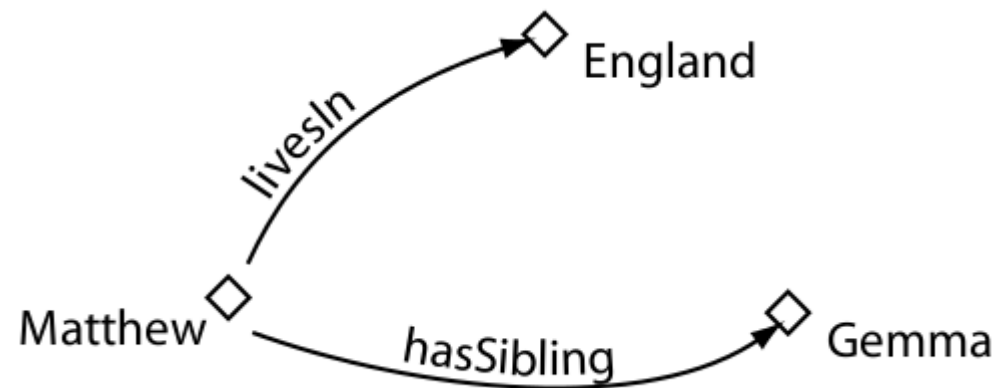
Classes

Individuals: the objects of the knowledge domain



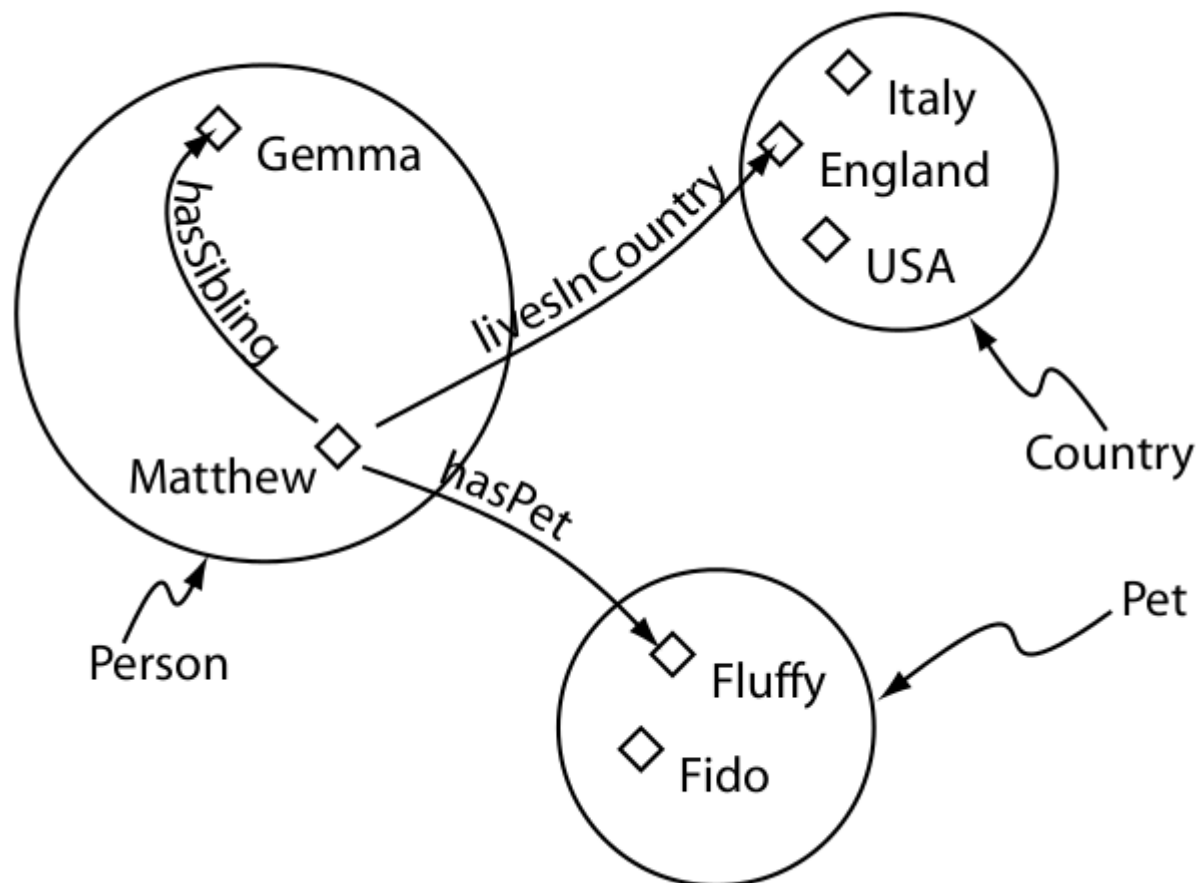
(Tutorial Manchester)

Properties: they can be used to link individuals in binary relations



(Tutorial Manchester)

Classes: sets of individuals with common characteristics



(Tutorial Manchester)

An OWL ontology with individuals and classes is a Knowledge Base

Knowledge Base (KB): Abox + Tbox

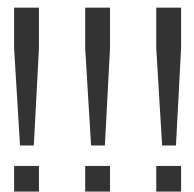
TBox (Terminological Box): ~schema (~ classes)

Abox (Assertional Box): ~data (~ individuals)

OWL works under the Open World Assumption (OWA)

Data Base (Closed World Assumption): the information not mentioned is false (Negation as Failure)

Knowledge Base (Open World Assumption): the information not mentioned is unknown (Can be true or false)



Pedro has spanish nationality

¿Does Pedro have british nationality?

CWA (DB): No

OWA (OWL KB): We don't know (Pedro can have double nationality). Till we assert that Pedro can only have one nationality, OWL will assume he can have more than one

OWA advantage: we can add new knowledge (e.g. New nationalities) easily, we don't have to “change the schema”

OWA is good for settings in which our knowledge will always be incomplete: open systems like the (Semantic) Web

In OWL there is no Unique Name Assumption (UNA)

The fact that two entities have different URIs does not imply that they are different entities

We have to explicitly assert, if we want to, that two entities are different from each other

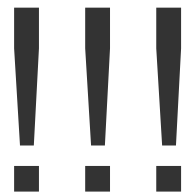
In the (Semantic) Web, different resources talk about the same entity



No UNA + OWA:

Building an ontology in OWL is like pruning a space in which by default everything is possible (OWA) and all the entities are the same (!UNA)

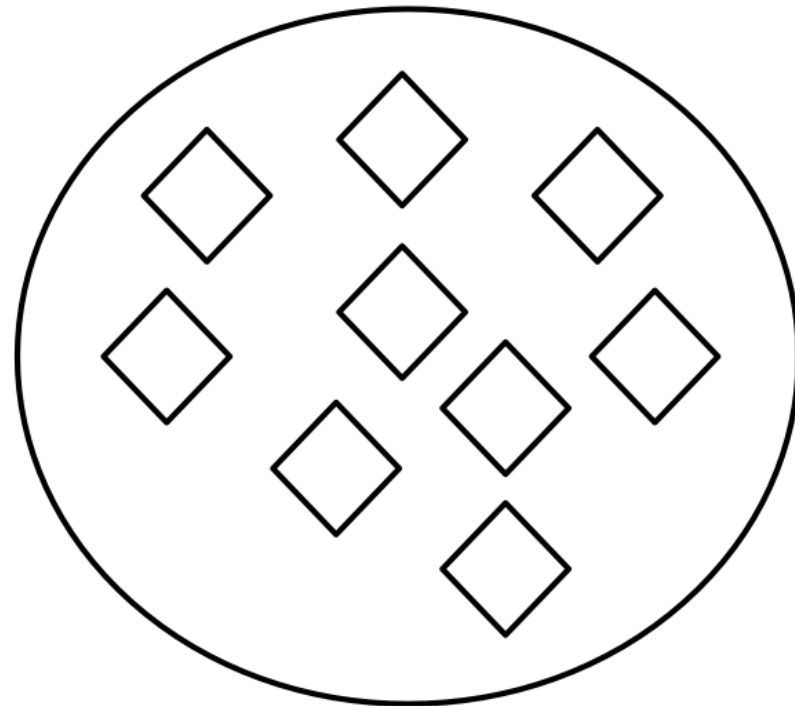
Such pruning is performed by adding axioms that limit the possible facts and make entities different to each other



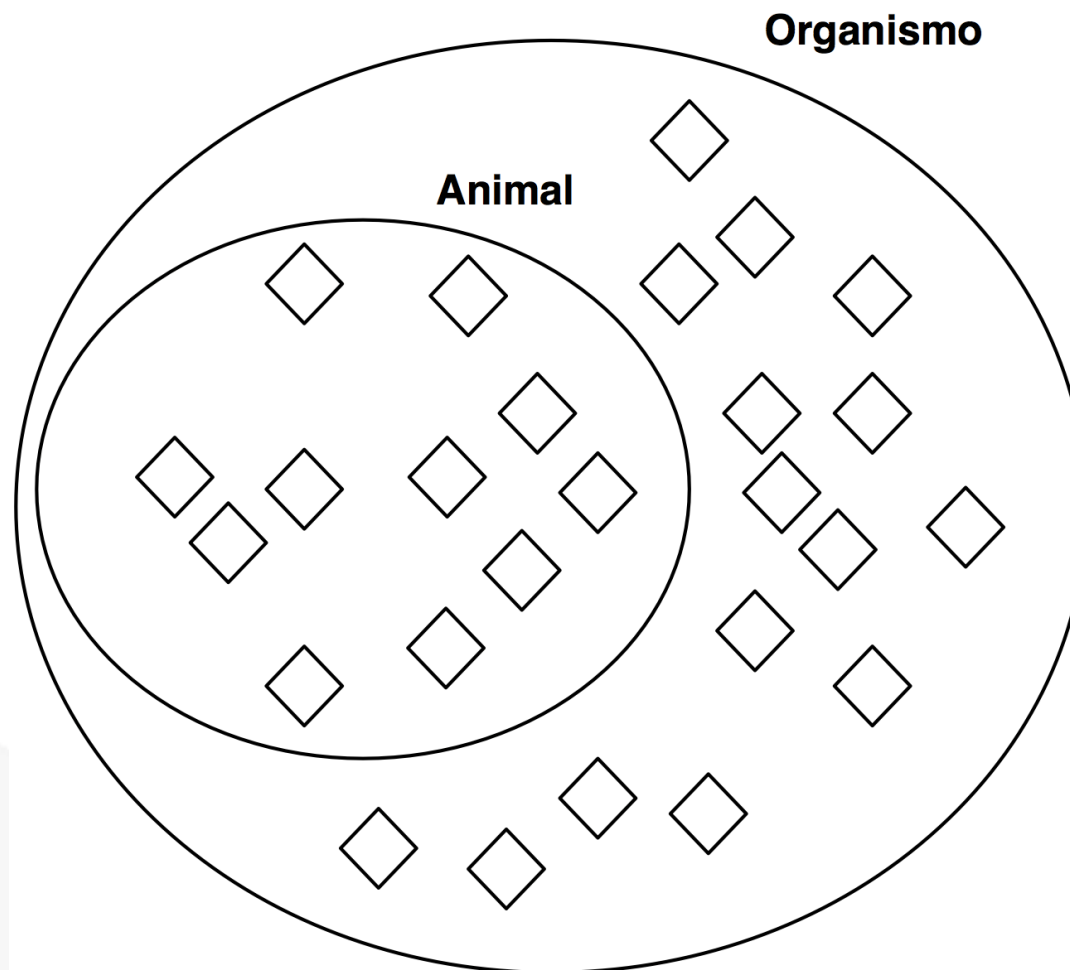
Classes

Classes: Sets of individuals

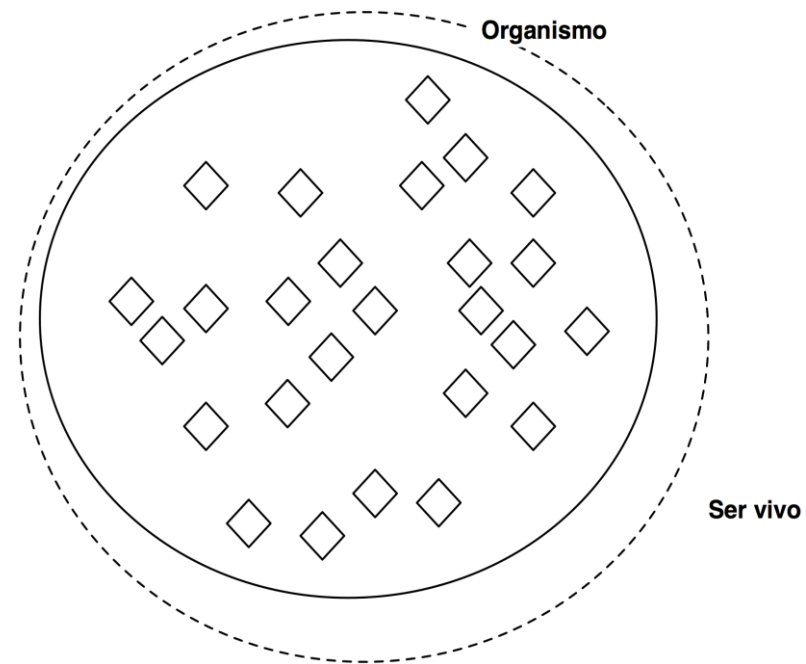
Organismo



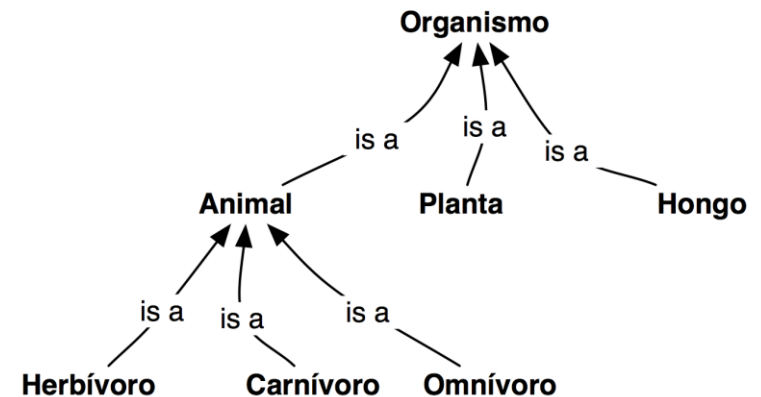
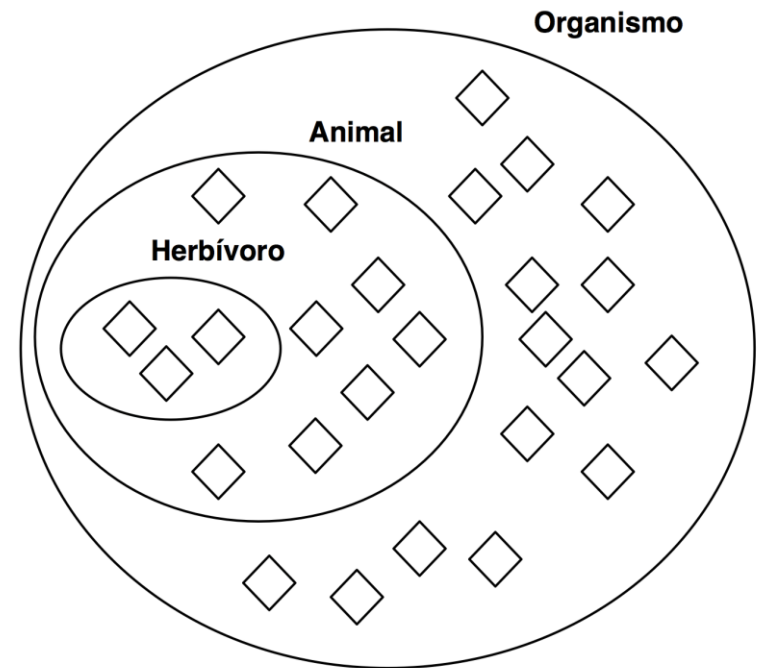
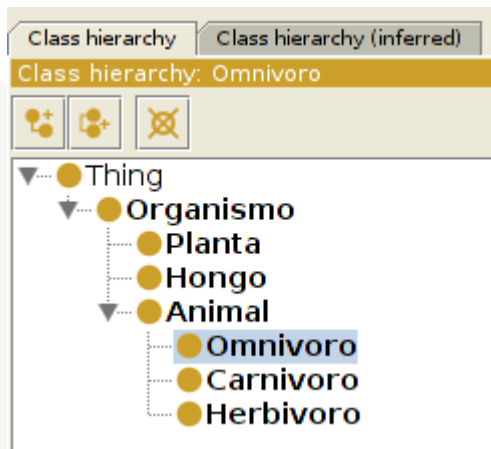
Classes can be subclasses of other classes: all the instances of the subclass are also instances of the superclass (But not the other way around)



Classes are equivalent if the extent of their sets is exactly the same: all the instances of A are also instances of B and the other way around



A taxonomy can be built combining different class-subclass axioms



In order to define the qualities that the individuals of a class must hold to be members of that class, *restrictions* on the number and type of binary relations are used

Thus, the restrictions define the conditions that must be fulfilled to be a member of a given class

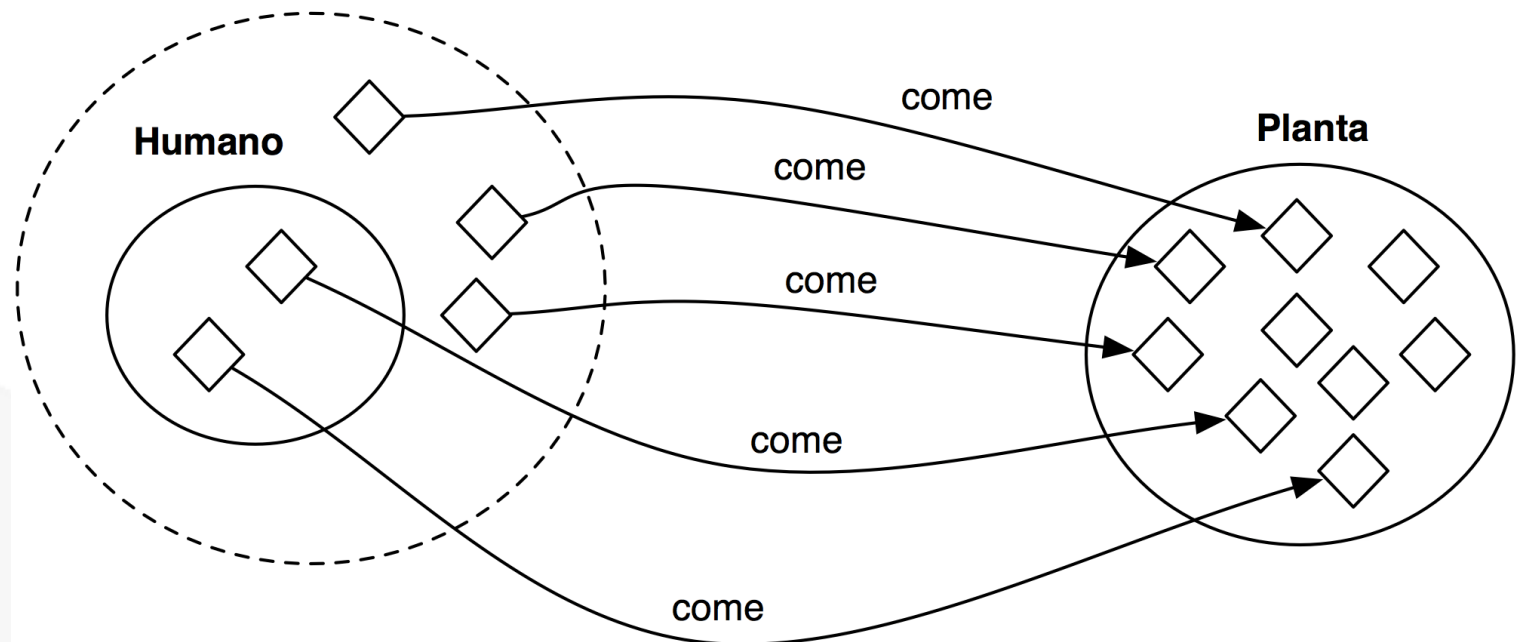
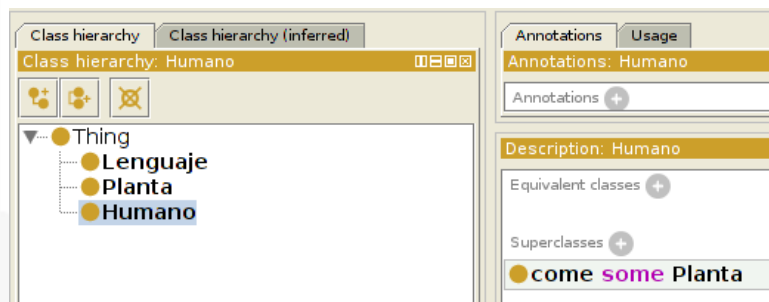
For example, we can state (In our ontology!) that in order to be human something must eat plants

Eating plants is a *necessary condition* to be human: all the humans eat plants, but there are other organisms that also eat plants that are not humans

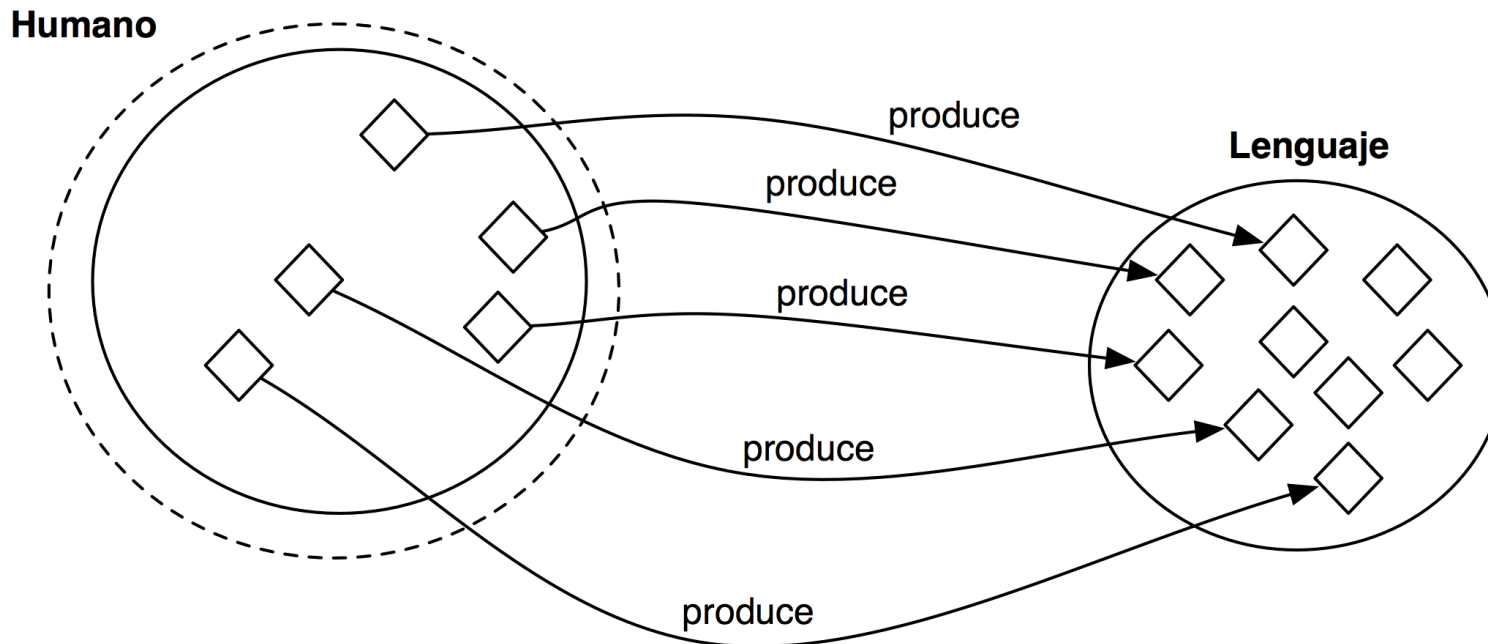
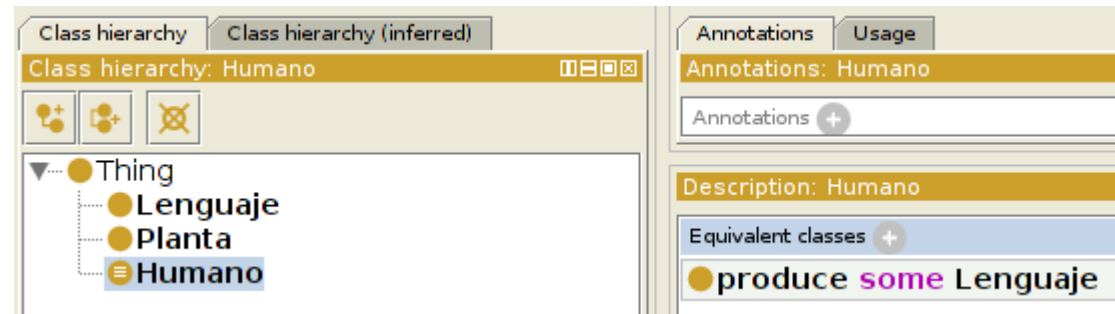
We can also define a *necessary and sufficient* condition: producing language is a unique quality of humans: if we find an individual (Organism) capable of producing language we can infer that is human, since no other organism does it

Conditions are anonymous classes: the named class we are defining with such conditions can be a subclass (Necessary) or equivalent class (Necessary and sufficient) to the anonymous class

The class **Humano** is a subclass (N) of the anonymous class comprised of the individuals that have at least one **come** binary relation with an individual of the class **Planta**



The class **Humano** is equivalent (N+S) to the anonymous class comprised of the individuals that have at least on relation with the property **produce** with and individual of the class **Lenguaje**

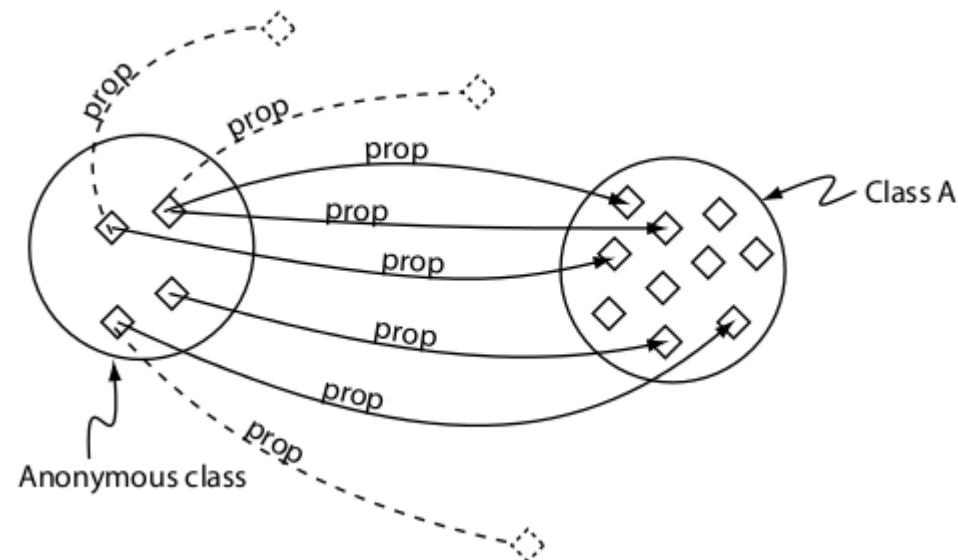


The classes with necessary and sufficient conditions are *defined* classes, and they are exploited for automated reasoning

The classes with only necessary conditions are *primitive* classes

Existential restrictions

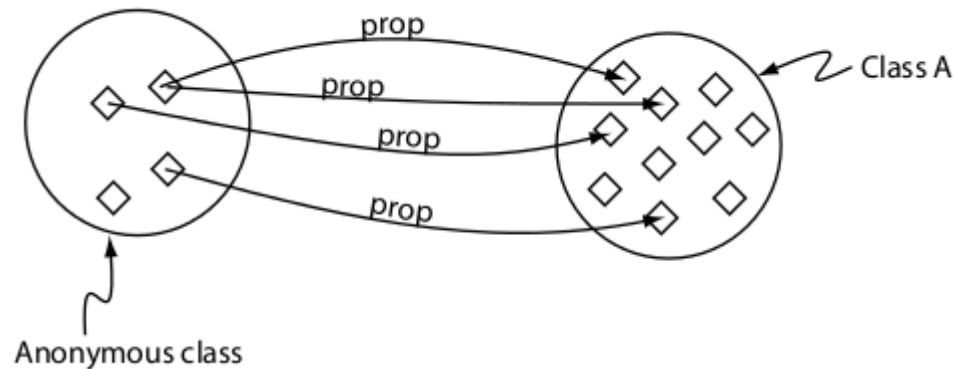
owl:someValuesFrom: the anonymous class comprised of the individuals that, amongst other things, have at least one relation to an individual of a given class with a given property: [humano subClassOf come some Planta](#)



(Tutorial Manchester)

Universal restriction

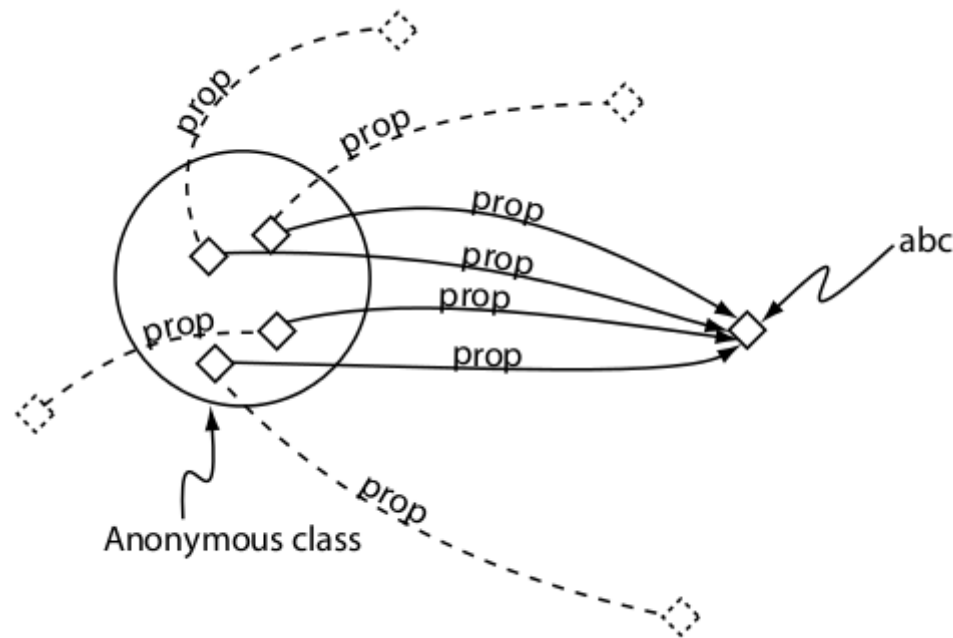
owl:allValuesFrom: the anonymous class comprised of the individuals that, if having a relation with a given property, must be to an individual of a concrete class or *none*: `humano subClassOf come only Organismo`



(Tutorial Manchester)

hasValue

the anonymous class comprised of the individuals that have a relation to a concrete individual humano subClassOf come value este tomate



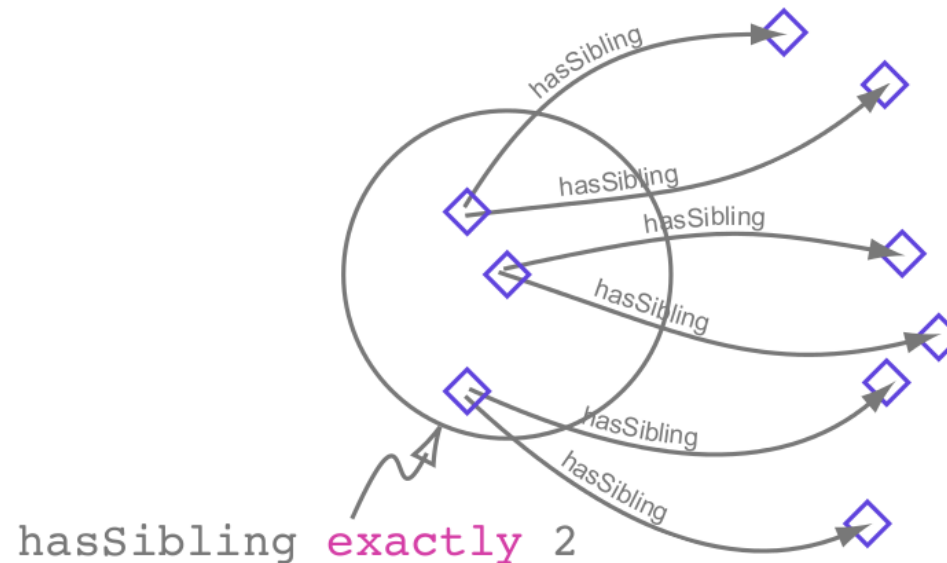
(Tutorial Manchester)

Cardinal restrictions:

Min: `humano subClassOf come min 1`

Max: `humano subClassOf come max 5`

Exactly: `humano subClassOf come exactly 3`



(Tutorial Manchester)

QCR (Qualified Cardinality Constraint):

Min: `humano subClassOf come min 1 Planta`

Max: `humano subClassOf come max 5 Planta`

Exactly: `humano subClassOf come exactly 3 Planta`

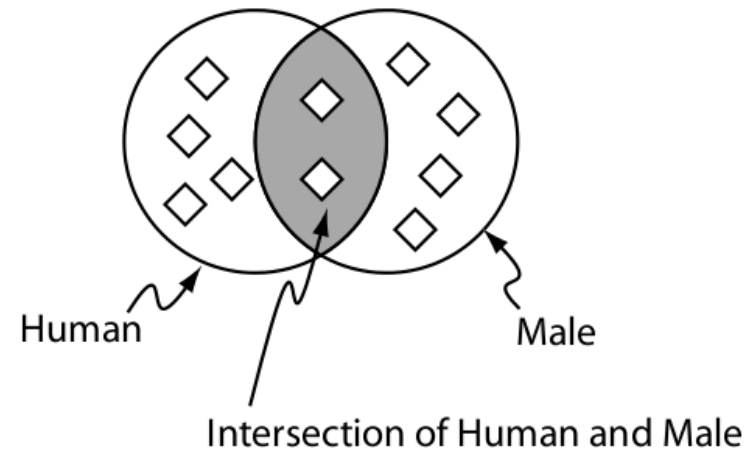
We can state that a class is different to other class (They don't have any individual in common) using disjointFrom: `humano disjointFrom planta`

We can state that two classes are the same (They have the same extent of individuals) using equivalentTo: `humano equivalentTo persona`

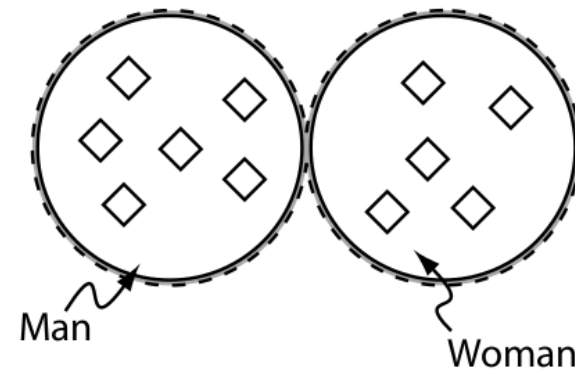
Booleans

Not: `humano subClassOf not (come some electrodomestico)`

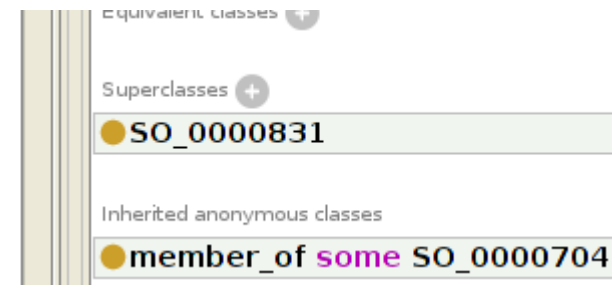
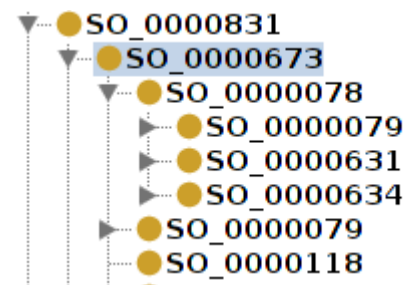
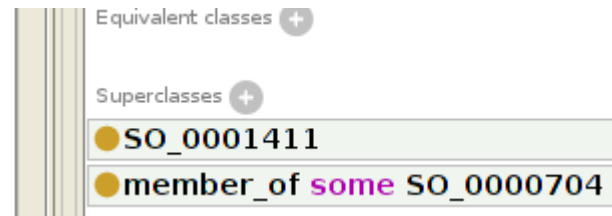
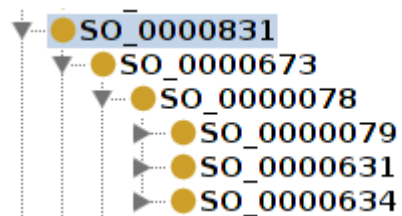
And (Intersection):
`man equivalentTo human and male`



Or (Union):
`human equivalentTo woman or man`



In a class hierarchy, the subclass “inherits” the conditions of the superclass: it can have further conditions but not a condition that conflicts with the conditions of the superclass



Conditions can be very complex, combining different OWL elements

The screenshot displays a web ontology editor interface. On the left, a 'Class hierarchy' panel shows a tree structure starting from 'Thing'. The class 'Hypothesis_MYB_AP1_UP' is selected and highlighted. The right panel shows the 'Annotations' tab for this class. It includes a 'Description' field with a complex logical expression and a list of 'Equivalent classes'.

Class hierarchy: Hypothesis_MYB_AP1_UP

Annotations: Hypothesis_MYB_AP1_UP

Description: Hypothesis_MYB_AP1_UP

Equivalent classes:

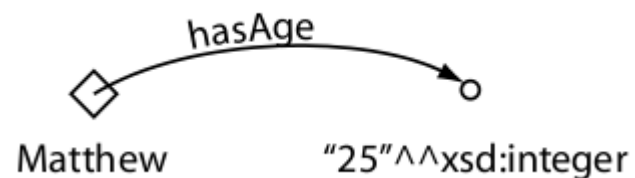
- transcription_factor **exactly** 1 (PRO_000009232 and (located_in_cellular_component **some** ((ECO_0000033 and GO_0005654) or (GO_0000790 and (evidence_code **some** ECO_0000203)))))
- target_gene **exactly** 1 (PRO_000010799 and (participates_in **some** (MI_0931 and (detected_by **some** MI_0438) and (has_participant **only** PRO_000009232))))
- hypothesis_entity **only** (PRO_000009232 or PRO_000010799)
- regulation **some** UP

Properties

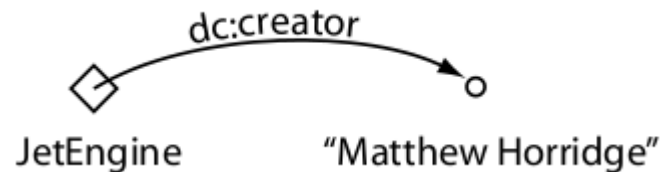
Object Properties



DataType Properties



Annotation Properties*



Object Properties

Property hierarchy:

Sub/SuperProperties

$p \text{ SubPropertyOf } q$

If $A \text{ p } B$, $A \text{ q } B$

But if $D \text{ q } F$, not $D \text{ p } F$

Equivalent Properties

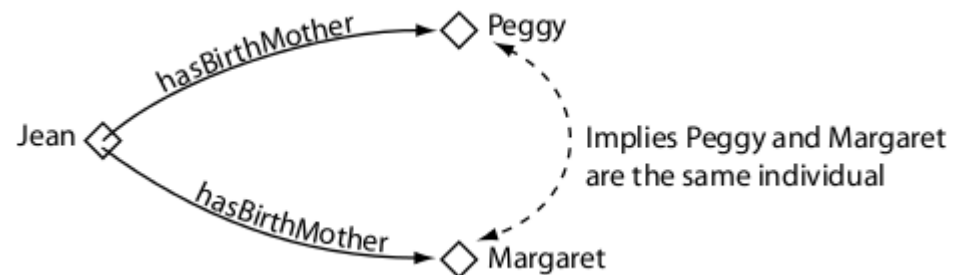
Disjoint Properties

The screenshot displays the Protégé OWL editor interface. The top menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. Below the menu is a toolbar with navigation icons and a dropdown menu showing 'Ontology1301827823935 (http://www.semanticweb.org/ontologies/2011/3/Ontology1301827823935.owl)'. The main workspace is divided into several panes:

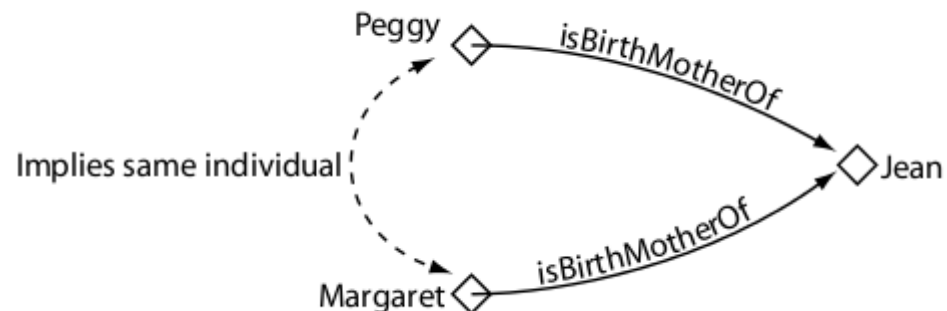
- Left Pane:** 'Object property hierarchy: interacciona_con'. It shows a tree structure with properties: topObjectProperty, es_localizacion_de, tiene_parte, componente_de, localizado_en, parte_de, interacciona_con (selected), mata_a, and estrangula_a.
- Top Right Pane:** 'Annotations: interacciona_con'. It has tabs for 'Annotations' and 'Usage'. The 'Annotations' tab is active, showing a list of annotations with a '+' icon to add more.
- Bottom Left Pane:** 'Characteristics: interacciona_con'. It lists various property characteristics with checkboxes:
 - ☐ Functional
 - ☐ Inverse functional
 - ☐ Transitive
 - ☐ Symmetric
 - ☐ Asymmetric
 - ☐ Reflexive
 - ☐ Irreflexive
- Bottom Right Pane:** 'Description: interacciona_con'. It lists various property descriptions with '+' icons to expand:
 - Domains (intersection) +
 - Ranges (intersection) +
 - Equivalent object properties +
 - Super properties +
 - Inverse properties +
 - Disjoint properties +
 - Property chains +

At the bottom of the interface, there is a status bar with the text: 'To use the reasoner click Reasoner->Start Reasoner' and a checked checkbox for 'Show Inferences'.

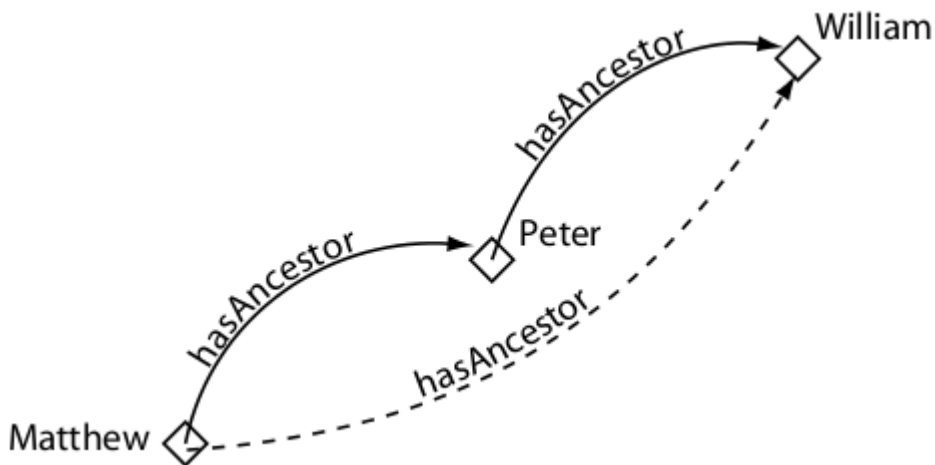
Functional



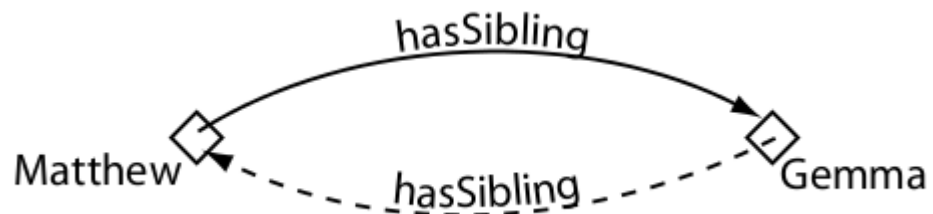
Inverse functional



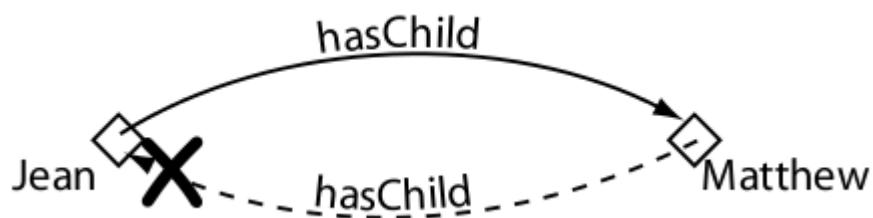
Transitive



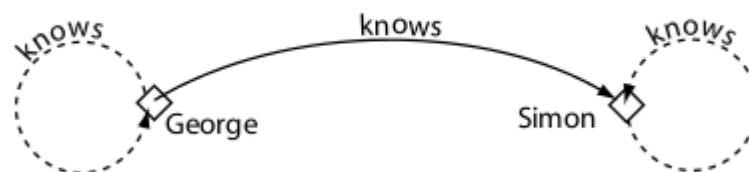
Symmetric



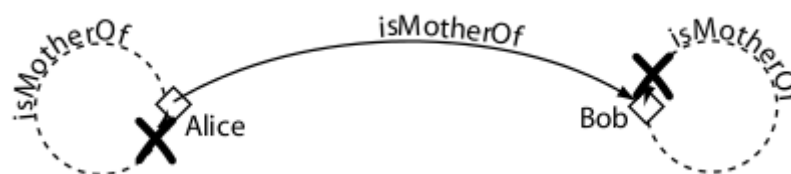
Antisymmetric*



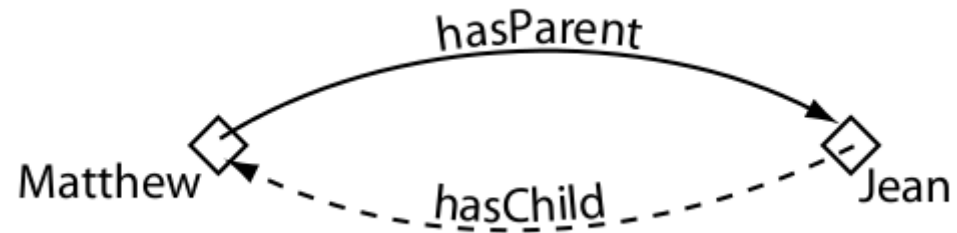
Reflexive



Irreflexive*



Inverse properties



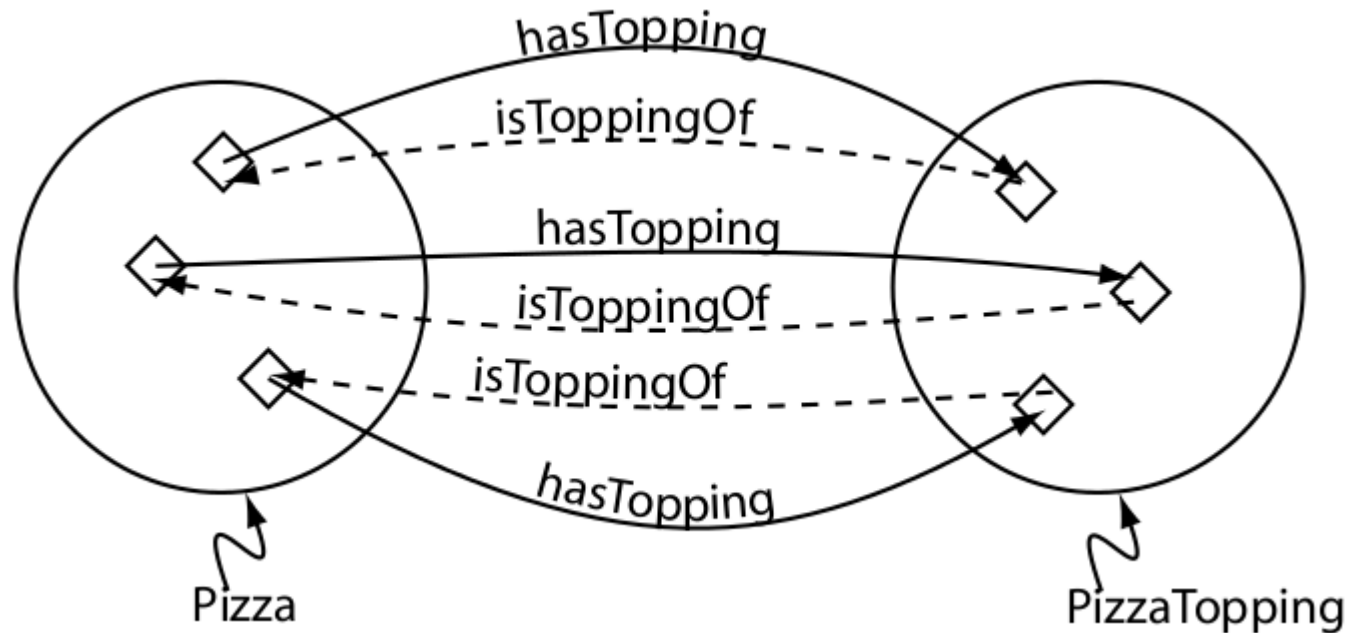
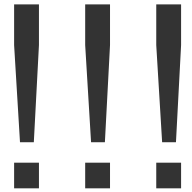
(Tutorial Manchester)

Domain and Range:

Usually classes or class unions

But any anonymous expression class can be used

They are not constraints, they are axioms



Data Type Properties

The screenshot displays the Protégé OWL editor interface. The top menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. Below the menu is a toolbar with navigation icons and a dropdown menu showing the current ontology: "Ontology1301827823935 (http://www.semanticweb.org/ontologies/2011/3/Ontology1301827823935.owl)".

The main workspace is divided into several panes:

- Left Pane:** Displays the "Data property hierarchy: topDataPrope...". It shows a tree structure where "topDataProperty" is expanded, revealing "cantidad_de_soldados", "distancia", and "distancia_de_ataque". "distancia" is further expanded to show "alcance_de_misil" and "alcance_de_cañon".
- Right Pane:** Contains three sub-panes:
 - Annotations:** Shows "Annotations: topDataPrope..." with a "+" button to add annotations.
 - Characteristics:** Shows "Characteristics: topDat..." with a "Functional" checkbox.
 - Description:** Shows "Description: topDataPrope..." with buttons for "Domains (intersection)", "Ranges", "Equivalent properties", "Super properties", and "Disjoint properties".

At the bottom of the interface, a status bar reads: "To use the reasoner click Reasoner->Start Reasoner" followed by a checked checkbox labeled "Show Inferences".

Equivalent / sub-super / disjoint

Only Functional (No transitive, inverse functional, ...)

Domain: ~ Object Properties

Range:

Built-in datatypes

Data range expression

Annotation Properties

Add non-semantic annotations in natural language to entities, axioms or the ontology

rdfs:label, rdfs:comment, ...

Dublin Core (<http://dublincore.org/>)

Custom annotation properties

Language (en, es, ...) and type (xsd:string, ...)



Individuals

An individual can be a member of one or more anonymous or named classes (**Types**)

An individual can be the same as other individual (**SameAs**)

An individual can be different from another individual (**DifferentFrom**)

Individuals can be related in binary relations (Object Properties):

my_wheel part_of my_car
my_wheel not part_of your_car

Individuals can be related with data (Data Type properties):

my_car has_power "90"^^xsd:positiveInteger
my_car not has_power "90"^^xsd:positiveInteger

The screenshot displays an OWL editor interface with three main panels. The left panel, titled 'Members list', shows a single member 'Mi_retrovisor'. The middle panel, titled 'Description: Mi_retrovisor', lists the types 'Recambio' and 'Retrovisor', and the individuals 'Retrovisor_de_mi_novia' and 'retrovisor_de_pedro'. The right panel, titled 'Property assertions: Mi_retrovisor', shows object property assertions 'parte_de mi_coche' and 'parte_de coche_de_pedro', and data property assertions 'tiene_precio 20' and 'tiene_precio 30'.

Members list

- Mi_retrovisor

Description: Mi_retrovisor

- Types
 - Recambio
 - Retrovisor
- Same individuals
 - Retrovisor_de_mi_novia
- Different individuals
 - retrovisor_de_pedro

Property assertions: Mi_retrovisor

- Object property assertions
 - parte_de mi_coche
 - parte_de coche_de_pedro
- Data property assertions
 - tiene_precio 20
 - tiene_precio 30

Some extra constructs

OWL oneOf

The screenshot displays a web ontology editor interface. On the left, the 'Class hierarchy' tab is active, showing a tree structure where 'Thing' is the root, 'Familia' is a subclass, and 'FamiliaMikel' is a subclass of 'Familia'. On the right, the 'Annotations' and 'Usage' tabs are visible. The 'Annotations' tab shows 'Annotations: FamiliaMikel' and a button to add annotations. The 'Usage' tab shows 'Description: FamiliaMikel' and several sections: 'Equivalent classes' with a button to add, a list containing '{Iker , JuanRamon , Mikel}', 'Superclasses' with a button to add, a list containing 'Familia', 'Inherited anonymous classes', and 'Members' with a button to add. The 'Members' section lists four individuals: Iker, JuanRamon, Mikel, and Pedro, each preceded by a diamond icon.

Class hierarchy: FamiliaMikel

- Thing
 - Familia
 - FamiliaMikel**

Annotations: FamiliaMikel

Annotations +

Description: FamiliaMikel

Equivalent classes +

- {Iker , JuanRamon , Mikel}

Superclasses +

- Familia

Inherited anonymous classes

Members +

- Iker
- JuanRamon
- Mikel
- Pedro

Role chains

The screenshot shows the OntoGraf web application interface. The top navigation bar includes tabs for Active Ontology, Entities, Classes, Object Properties, Data Properties, Individuals, OWLViz, DL Query, and OntoGraf. The 'Object Properties' tab is active.

On the left, the 'Object property hierarchy: ti' is displayed, showing a tree structure with 'topObjectProperty' as the root, and 'tiene_padre', 'tiene_primo', and 'tiene_sobrino' as its children.

The main area is divided into two panels. The left panel, titled 'Characteristics', contains a list of checkboxes for property characteristics: Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive. The right panel, titled 'Description: tiene_primo', contains a list of buttons for property relationships: Domains (intersection), Ranges (intersection), Equivalent object properties, Super properties, Inverse properties, Disjoint properties, and Property chains.

At the bottom of the right panel, a text box displays the property chain configuration: `tiene_padre o tiene_sobrino SubPropertyOf tiene_primo`.

OWL Self



OWL keys

<http://www.w3.org/TR/2009/REC-owl2-primer-20091027/#Keys>

~ “datatype inverse functional”

The screenshot shows a web ontology editor interface. On the left, the 'Class hierarchy' panel displays a tree structure with 'Thing' as the root and 'Persona' as a subclass. On the right, the 'Annotations: Persona' panel shows various properties and their values. The 'Description: Persona' panel lists 'Equivalent classes', 'Superclasses', 'Inherited anonymous classes', 'Members', and 'Keys'. The 'Members' list includes 'Maria', 'Mikel', and 'Oscar'. The 'Keys' list includes 'numero_seguridad_social'.

numero_seguridad_social "7"^^xsd:integer
 numero_seguridad_social "8"^^xsd:integer
 numero_seguridad_social "7"^^xsd:integer

Reasoning

Reasoning is performed by using a reasoner: a reasoner infers the axioms implied by the axioms we have stated in the ontology

Thus, the reasoner generates the *inferred* axioms from the *asserted* axioms

The reasoner makes *all* the implied axioms explicit, including the ones that would be missed by a human because of the complexity/size of the ontology

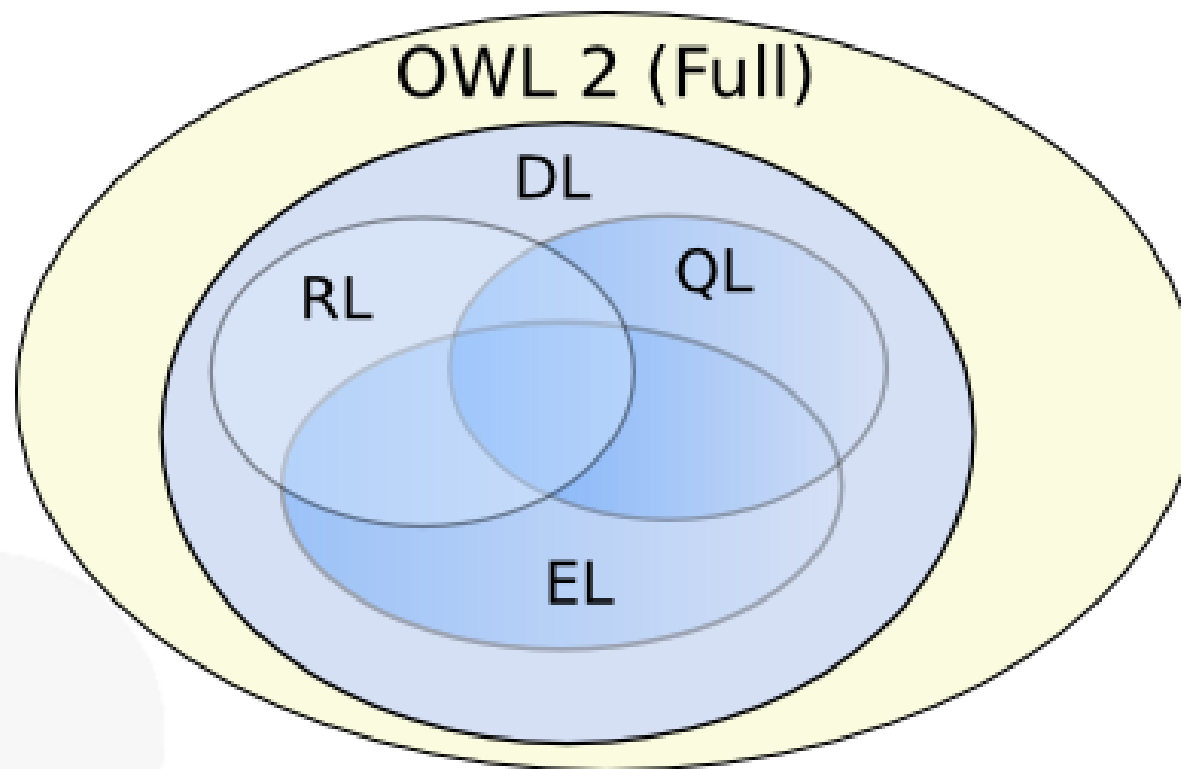
Therefore, a reasoner helps us deal with complex knowledge

OWL offers *sound and complete* reasoning if we don't use OWL full constructs (e.g. make an object property functional and transitive, ...)

That is the theory. In practice there can be efficiency problems. Reasoners are improving fast and OWL 2 offers different profiles optimized for different kinds of reasoning

OWL profiles

<http://www.w3.org/TR/owl2-profiles/>



Reasoning can be used to:

Maintain a class hierarchy

Check consistency of the ontology

Classify an entity against the ontology

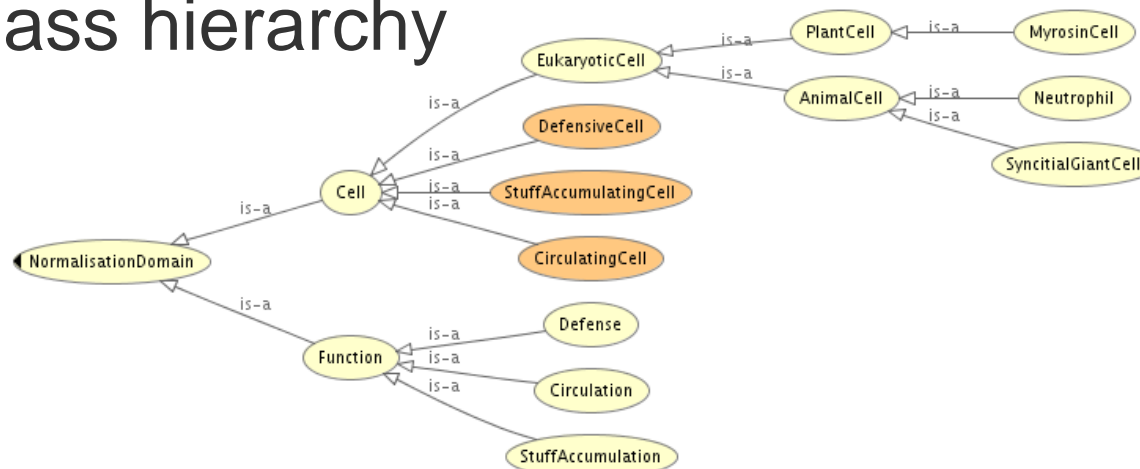
Make queries against the ontology

Use reasoning every time you change your ontology

Be aware of OWA and lack of UNA

!!!

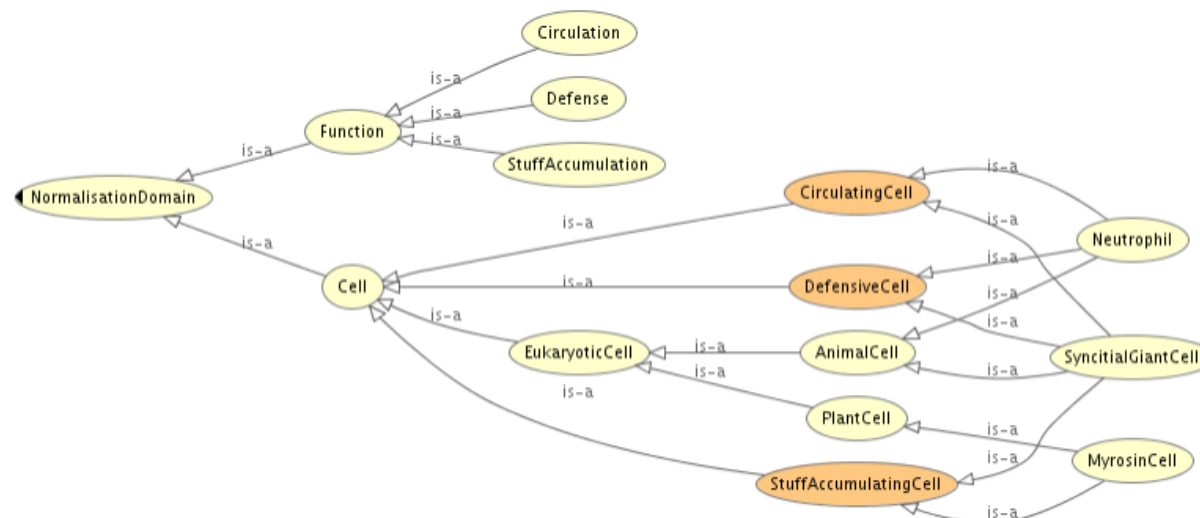
Maintain a class hierarchy



<http://www.gong.manchester.ac.uk/odp/html/Normalisation.html>



CLASSIFY



Check consistency of an ontology

Not satisfiable classes cannot have any individual (There is no individual that can satisfy the axioms)

An ontology becomes *inconsistent* if we state that a not satisfiable class has an individual

In an inconsistent ontology, not satisfiable classes are subclasses of **owl:Nothing**

Automated reasoning cannot be performed in an inconsistent ontology

An inconsistent ontology usually means that we have modelled something wrong

Check consistency of an ontology

The screenshot displays an ontology editor interface with three main panels:

- Class hierarchy:** Shows a tree structure starting with 'Thing'. Under 'Thing' are 'Coche' and 'Componente'. 'Componente' has subclasses 'audi', 'cilindro', and 'motor'. 'fabricante' has subclasses 'audi', 'skoda', and 'volkswagen'.
- Annotations:** Shows the 'audi' class with its description. Under 'Equivalent classes', 'Nothing' is listed. Under 'Superclasses', 'Componente' and 'fabricante' are listed.
- Axioms:** A list of axioms for the 'audi' class:
 - Componente DisjointWith fabricante**
 - audi SubClassOf Componente**
 - audi SubClassOf fabricante**

An arrow points from the 'Nothing' class in the 'Equivalent classes' section to the 'audi' class in the 'Axioms' section, highlighting the inconsistency.

Classify new entities against the ontology

Individuals: `types`

Classes: `subClassOf`, `equivalentTo`

Queries against the ontology

A query is an anonymous class

We ask the reasoner how the entities of the ontology relate to such class
(type, subclass, ...)

Defined classes can also be regarded as queries

Reasoning

The screenshot displays the Protégé OWL editor interface. The top menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. The address bar shows the ontology URL: <http://www.semanticweb.org/ontologies/2011/3/Ontology1301763636618.owl>.

The left sidebar, titled "Class hierarchy", shows a tree structure:

- Thing
 - CocheAudi
 - audi
 - cilindro
 - motor
 - skoda
 - volkswagen

The main workspace is divided into two tabs: "DL Query" and "Rules". The "DL Query" tab is active, showing a query editor with the text: `fabricado_por some (audi or skoda)`. Below the query editor are buttons for "Execute" and "Add to ontolo...".

The "Query results" section displays the following information:

- Equivalent classes (0)
- Ancestor classes (1)
 - Thing
- Super classes (1)
 - Thing
- Sub classes (1)
 - CocheAudi
- Descendant classes (1)
 - CocheAudi
- Instances (0)

On the right side of the query results, there is a list of checkboxes for filtering the results:

- ☒ Super classes
- ☒ Ancestor classes
- ☒ Equivalent classes
- ☒ Subclasses
- ☒ Descendant classes
- ☒ Individuals

The status bar at the bottom right indicates "Reasoner active" and "Show Inferences".

OWL tools

Ontology editors:

Protégé: <http://protege.stanford.edu/>

TopBraid composer:

http://www.topquadrant.com/products/TB_Composer.html

NeOn toolkit: <http://neon-toolkit.org>

APIs:

OWL API: <http://owlapi.sourceforge.net/>

Reasoners:

Pellet: <http://clarkparsia.com/pellet/>

HermiT: <http://hermit-reasoner.com/>

FaCT++: <http://code.google.com/p/factplusplus/>

Racer: <http://www.racer-systems.com/>