Semantic Data

Practice 4 : Ontology editing with Protégé

Jean-Louis Binot

Foreword

- This tutorial assumes that you have installed Protégé (instructions posted under the Project section of the course material).
- The example used in the tutorial is based on the following sources:
  - A tutorial from the course *IFT6282 Web Sémantique*, Lapalme, University of Montreal.
  - Using itself a small ontology from *(Grigoriou and van Harmelen 2004)*.

The example

Classes and properties are in italic.

a) A *Tree* is a subclass of *Plant*.

b) A *Branch* is (only) part of a *Tree*.

c) A *Leaf* is (only) part of a *Branch*.

d) An *Herbivore* is exactly an *Animal* eating only *Plants* or part of *Plants*.

e) A *Carnivore* is exactly an *Animal* eating *Animals*.

f) A *Giraffe* is an *Herbivore* and it is eating only *Leaves*.

g) A *Lion* is a *Carnivore* eating only *Herbivores*.

h) A *TastyPlant* is a *Plant* eaten by *Herbivores* and by *Carnivores*.

i) *Animal and Plant are disjoint.*

j) *eats* is the inverse of *isEatenBy*.

k) *part of is transitive.*

Such a description should be an outcome of the ontology modelling process !

The example expressed in description logic

1: need to define the hierarchy !

This was done in practice 3 (disjoint and transitivity axioms added).

$Tree \sqsubseteq Plant.$

$Branch \sqsubseteq \forall partOf.Tree$

$Leaf \sqsubseteq \forall partOf.Branch$

$Herbivore \equiv Animal \sqcap \forall eats.(Plant \sqcup \forall partOf.Plant)$

$Carnivore \equiv Animal \sqcap \exists eats.Animal$

$Giraffe \sqsubseteq Herbivore \sqcap \forall eats.Leaf$

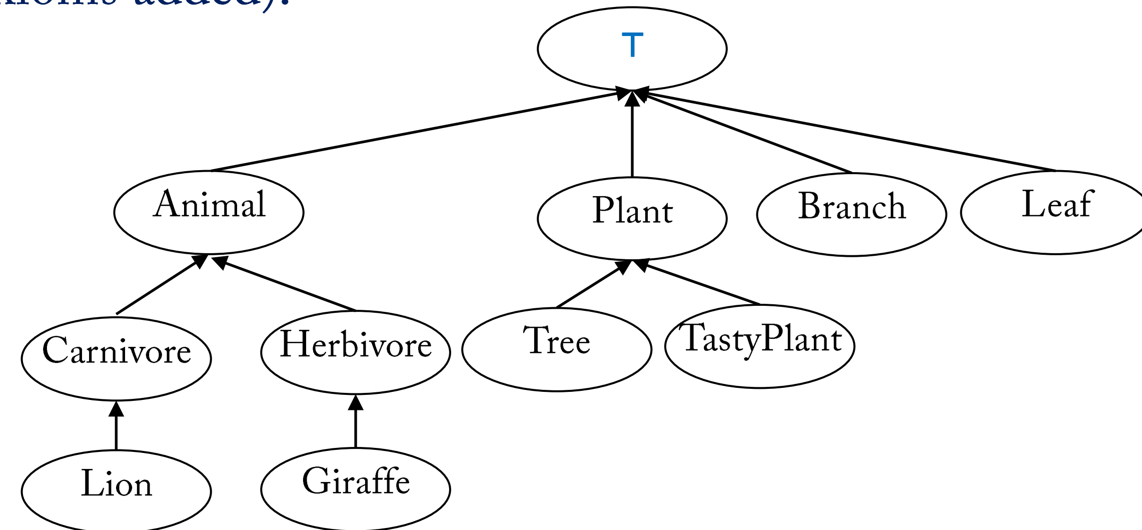
$Lion \sqsubseteq Carnivore \sqcap \forall eats.Herbivore$

$TastyPlant \sqsubseteq Plant \sqcap \exists isEatenBy.Herbivore \sqcap \exists isEatenBy.Carnivore$

$eats \equiv isEatenBy^{-}$

$Disjoint(Animal, Plant).$

$Trans(partOf)$

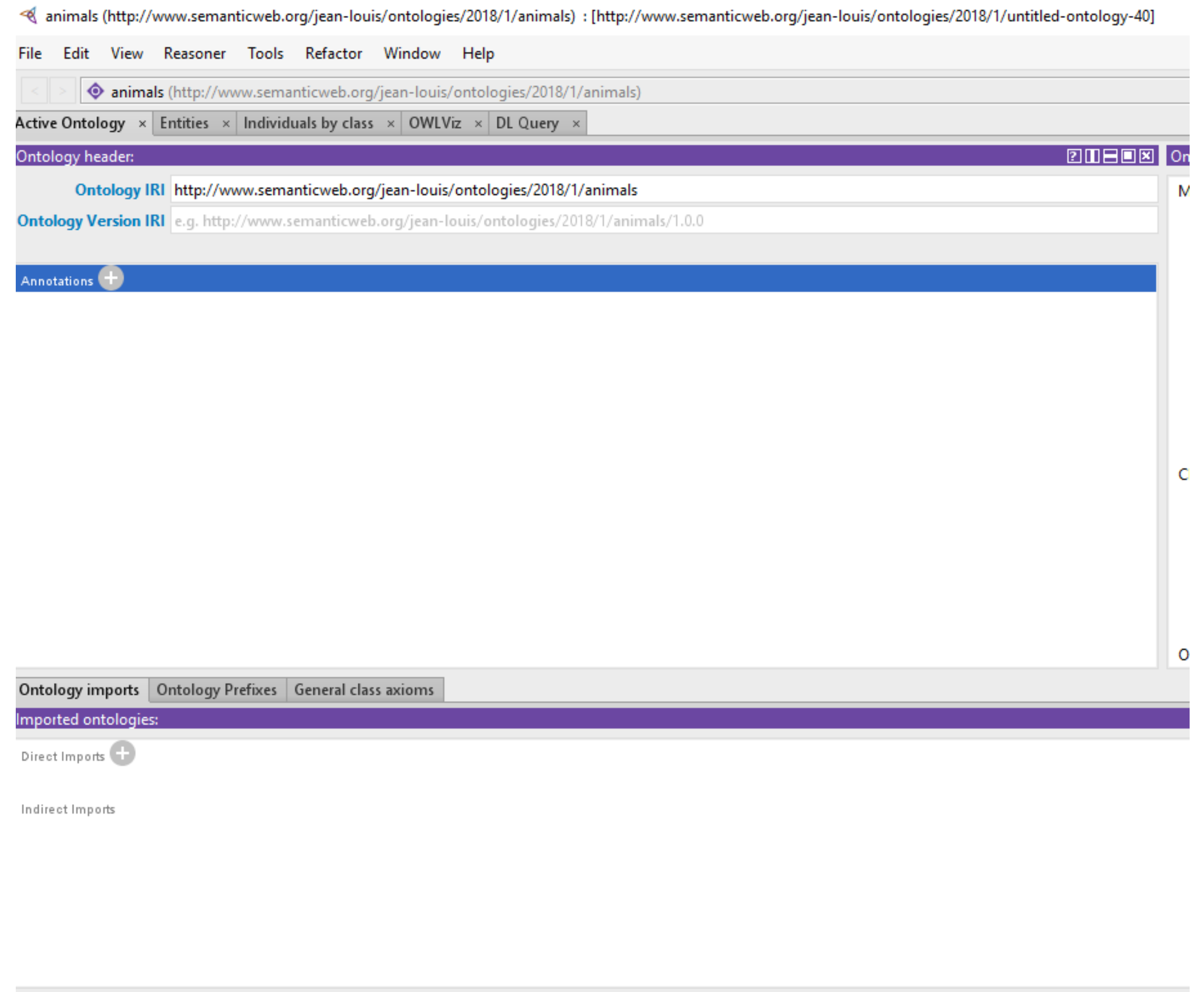


Start Protégé

- Open the editor
- File > New
- Give an URI to your ontology.

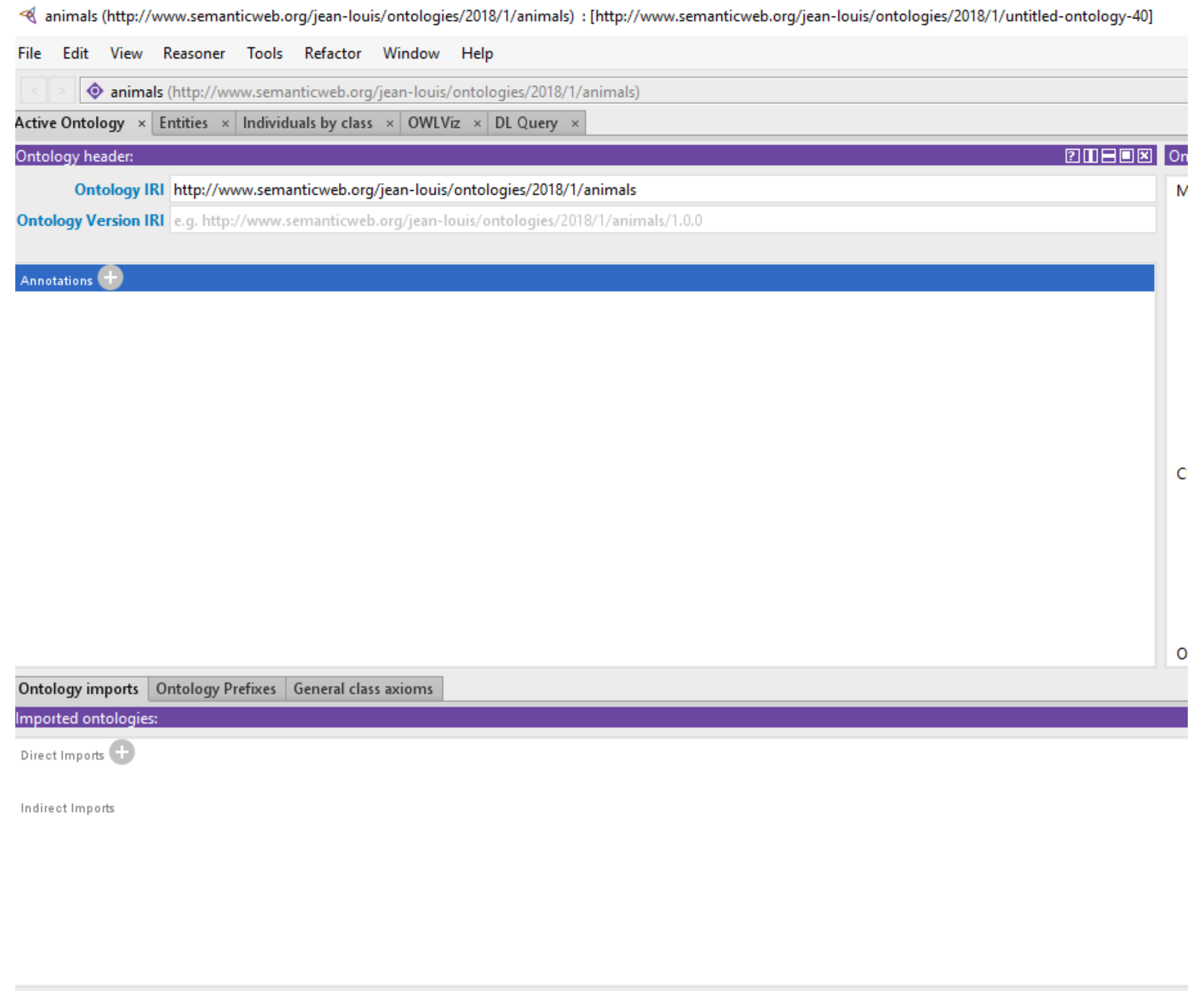
  (just keep the address proposed with a meaningful file name).
- Save it as an OWL ontology

  (e.g., Turtle syntax).



The main screen

- The screen is divided into views, depending on the selected tab.
- The views can be configured with the options from the Windows menu :
  - Windows > Views
  - Windows > Tabs
- More info on Protégé views.

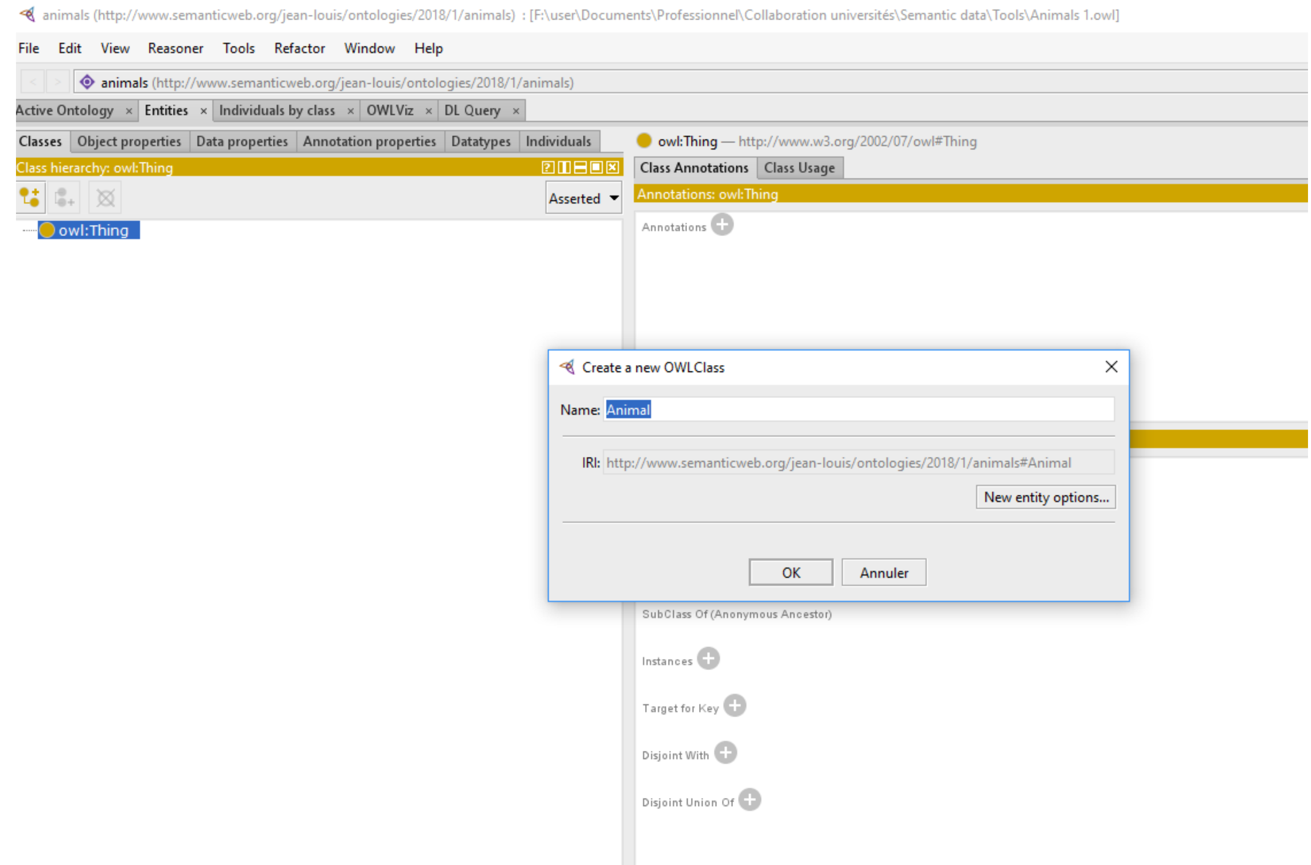


Creating classes

- Select the Entities Tab and the Class subtab.
- Select owl:Thing (root class)

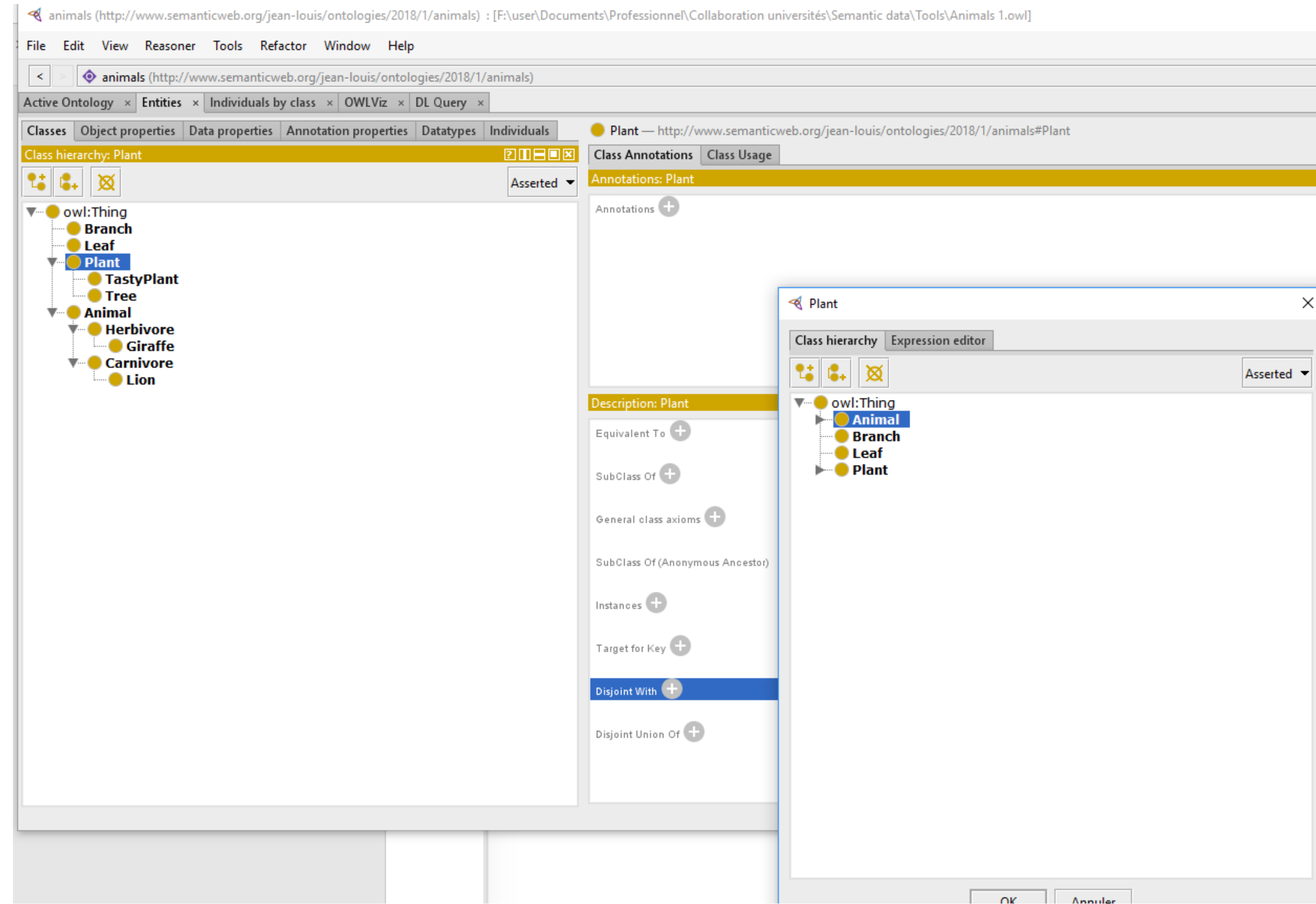
Equivalent to T in DL : the universal concept.

- The buttons allow you to :
  - Define a subclass;
  - Define a sibling class;
  - Suppress a class and her subtree.



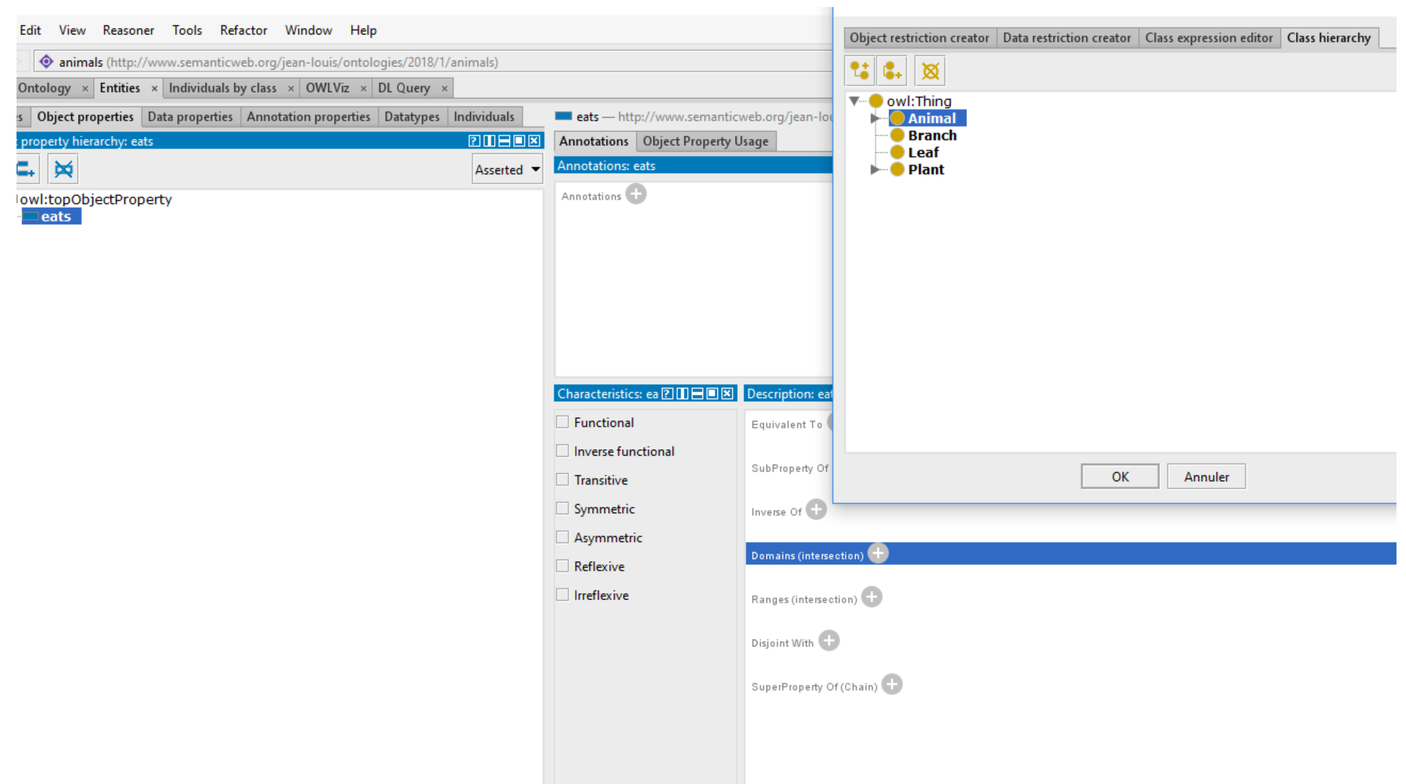
Creating classes

- Define all the classes.
- Indicate that Plant and Animal are disjoint by selecting Disjoint With in the description view and selecting the right class.
- You can check that the disjoint property is also recorded on the other class.
- You can correct mistakes with the Refactor (e.g., Rename entity).



Creating Properties

- In Entities, select the Properties subtab.
- You have now access to the hierarchy of properties.
- Define the property *eats*; specify its domain, the class Animal :
  - Select Domain in the Description view and use the Class hierarchy tab of the domain selection view.
  - Note : we will not specify the range; why ?



Creating Properties

- Add *isEatenBy* and indicates it is the inverse property of *eats*.
- Add *partOf* and select the characteristic transitive.
- Select eats and checks that the inverse property has been correctly set.
- The result should look like the picture on the right.

Definition of role restrictions

- $Branch \subseteq \forall partOf.Tree$
  - Select Branch in the class hierarchy, and SubclassOf in description view.
  - In the view that opens, select object restriction creator, the partOf relationship, and the class Tree as restriction filler.
  - Select Only (universal) as restriction type.
- The result is shown in Manchester syntax in the description view of Branch :

  partOf only Tree
- Do the same for

  $Leaf \subseteq \forall partOf.Branch$

Definition of role restrictions

- $Herbivore \equiv Animal \sqcap \forall eats.(Plant \sqcup \forall partOf.Plant)$
  - Select Herbivore in the class hierarchy; in the Equivalent to option of the description view, select +.
  - Select the class expression editor, allowing to enter the concept expression in Manchester syntax.
  - Enter the restriction in Manchester syntax :

  Animal and eats only (Plant or partOf only Plant)

- To write concept expressions one may drag classes or properties and use tab for completion.

Definition of role restrictions

- $Carnivore \equiv Animal \sqcap \exists eats.Animal$
  - Add an Equivalence definition as follows :
    Animal and eats some Animal
- $Giraffe \sqsubseteq Herbivore \sqcap \forall eats.Leaf$
  - Edit the subclass to add the expression :
    eats only Leaf
- $Lion \sqsubseteq Carnivore \sqcap \forall eats.Herbivore$
  - Add a subclass for the expression :
    eats only Herbivore
- $TastyPlant \sqsubseteq Plant \sqcap \exists isEatenBy.Herbivore \sqcap \exists isEatenBy.Carnivore$
  - Add two subclasses for the expressions :
    isEatenBy some Carnivore
    isEatenBy some Herbivore

animals (http://www.semanticweb.org/jean-louis/ontologies/2018/1/animals) : [F:\user\Documents\Professionnel\Collaboration universités\Semantic data\T...

File Edit View Reasoner Tools Refactor Window Help

animals (http://www.semanticweb.org/jean-louis/ontologies/2018/1/animals)

Active Ontology | Entities | Individuals by class | OWLViz | DL Query

Annotation properties | Datatypes | Individuals | Classes | Object properties | Data properties

Class hierarchy: TastyPlant — Asserted

- owl:Thing
  - Giraffe
  - Animal
    - Herbivore
    - Carnivore
      - Lion
  - Branch
  - Leaf
  - Plant
    - TastyPlant
    - Tree

TastyPlant — http://www.semanticweb.org/jean-louis/ontologies/2018/1/animals#TastyPlant

Class Annotations | Class Usage

Annotations: TastyPlant

Annotations
rdfs:comment
A TastyPlan is a Plant which is eaten by a Carnivore and also by an Herbivore

Description: TastyPlant

Equivalent To

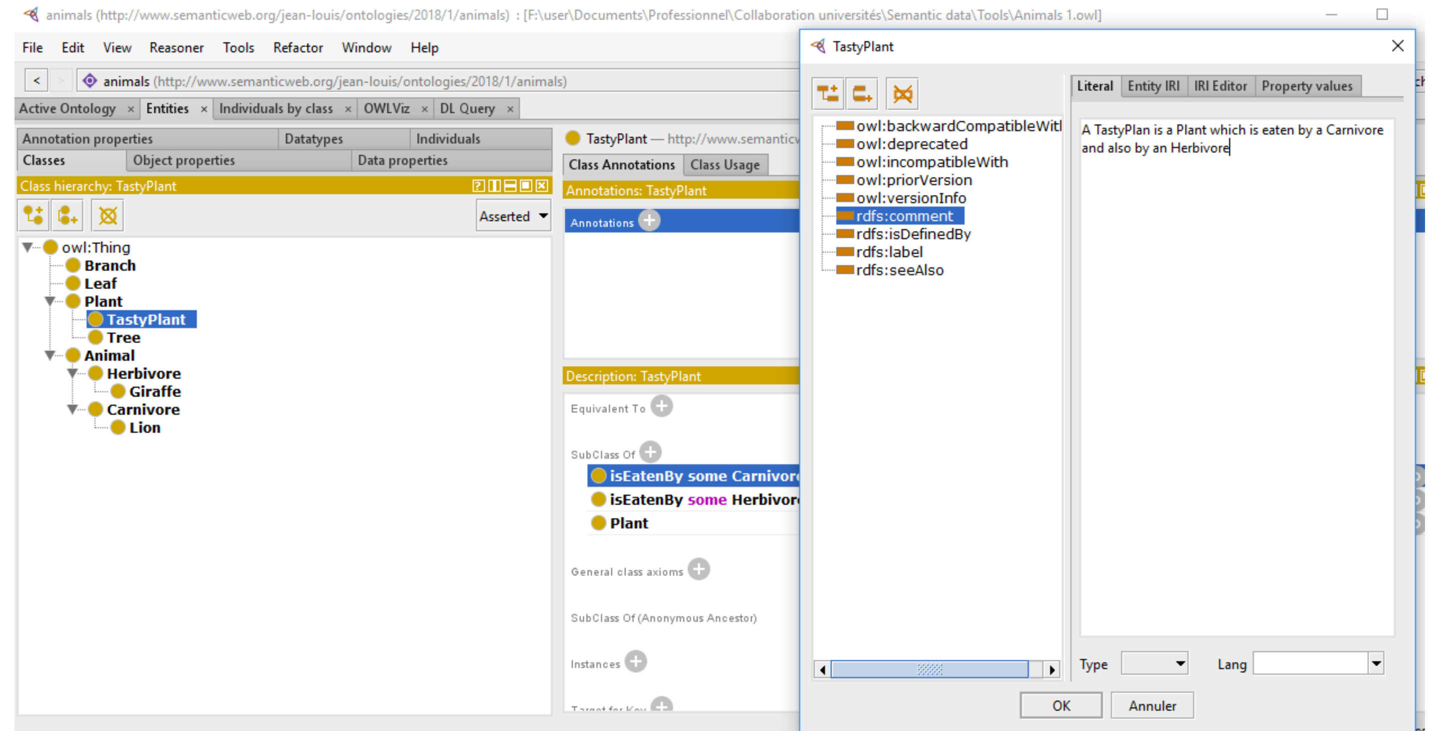
SubClass Of
- isEatenBy some Carnivore
- isEatenBy some Herbivore
- Plant

General class axioms

SubClass Of (Anonymous Ancestor)

Annotations

- When a description is complex, it is useful to document it.
- This is done with Annotations.
- Select the class TastyPlant and add a comment annotation in the class Annotations view.



Adding instances

- Let us add the following instances :

  Leo: a Lion

  Sophie: a Giraffe

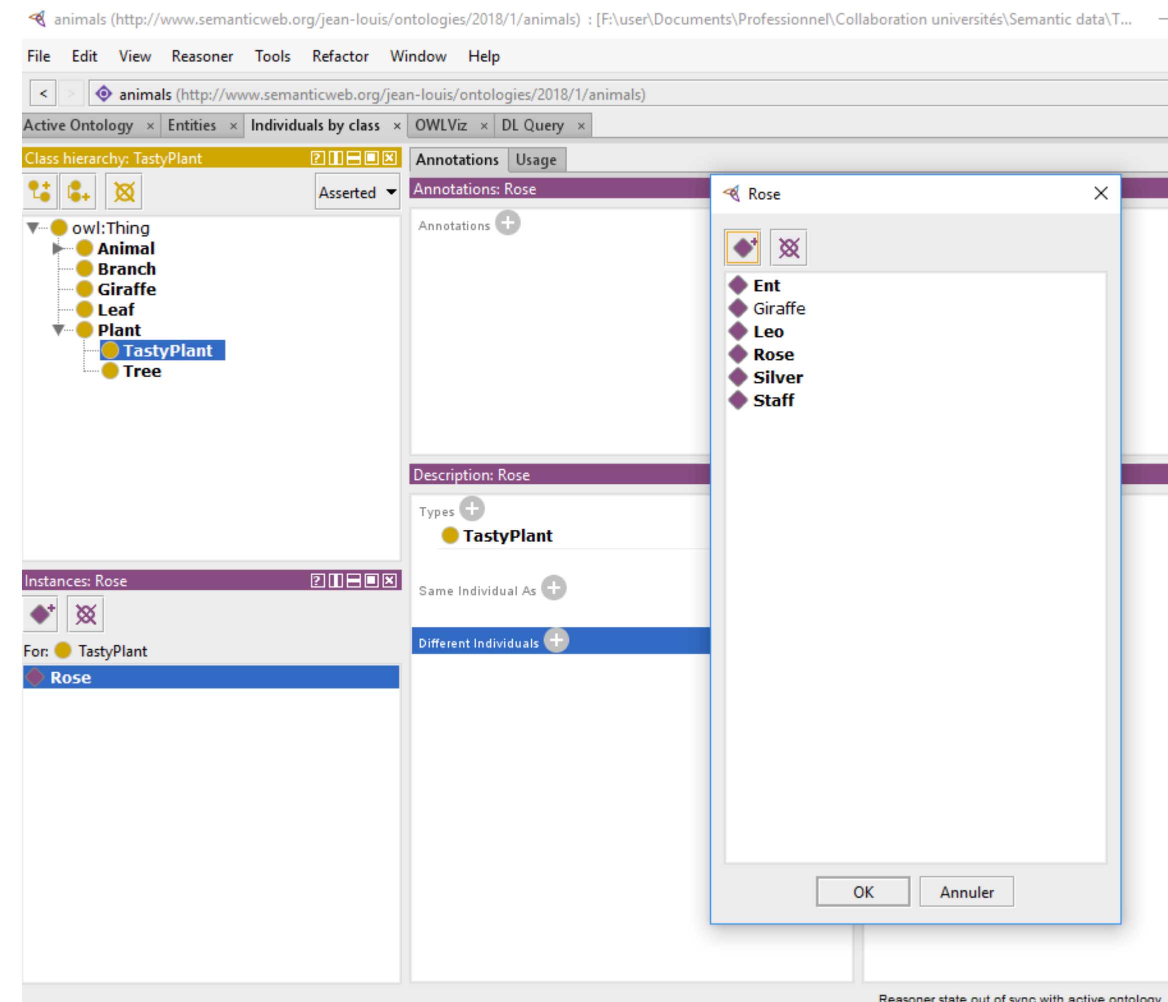
  Ent: a Tree

  Rose: a TastyPlant

  Staff: a Branch

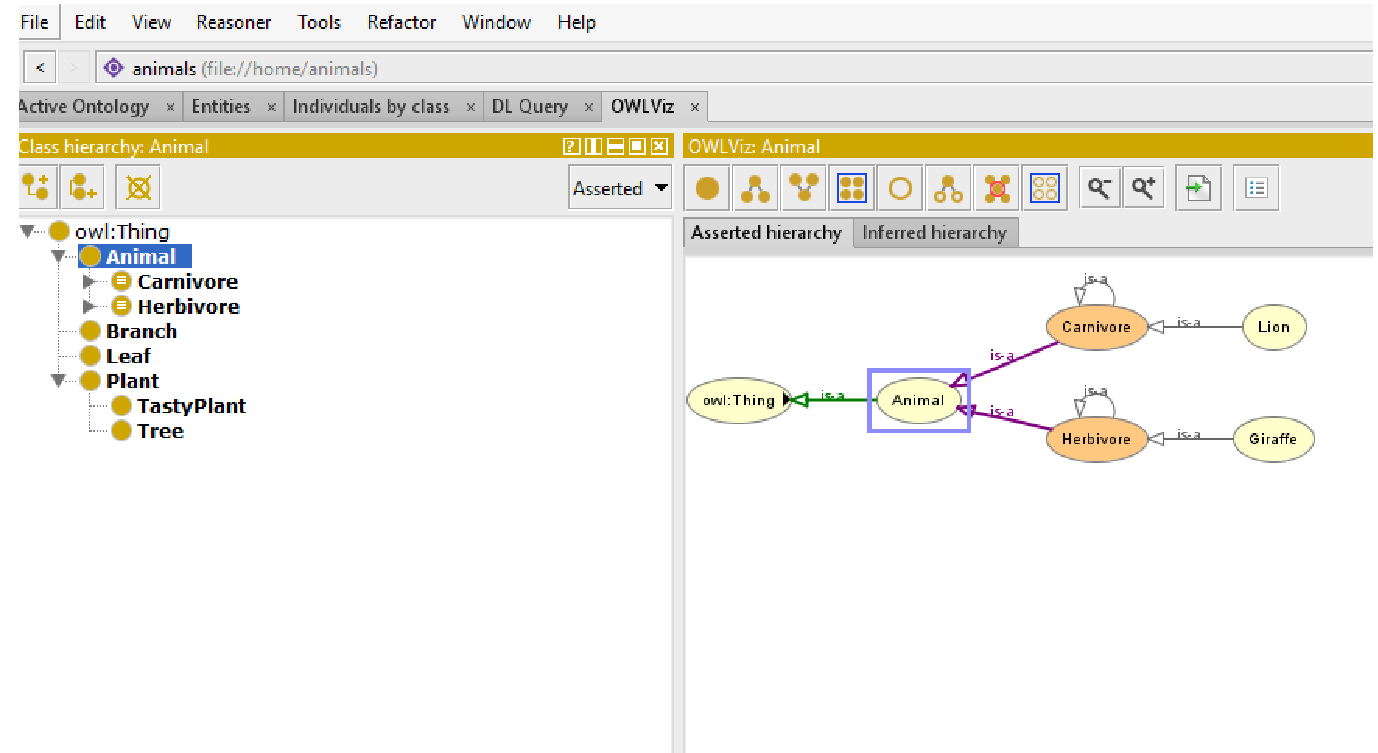
  Silver : a Leaf

- Enter instances by selecting the Individuals by class tab, and the direct instances pane.
- Specify that they are different : in the description view, select different individuals.
- Why ?
- Description logics do not support the unique name assumption !



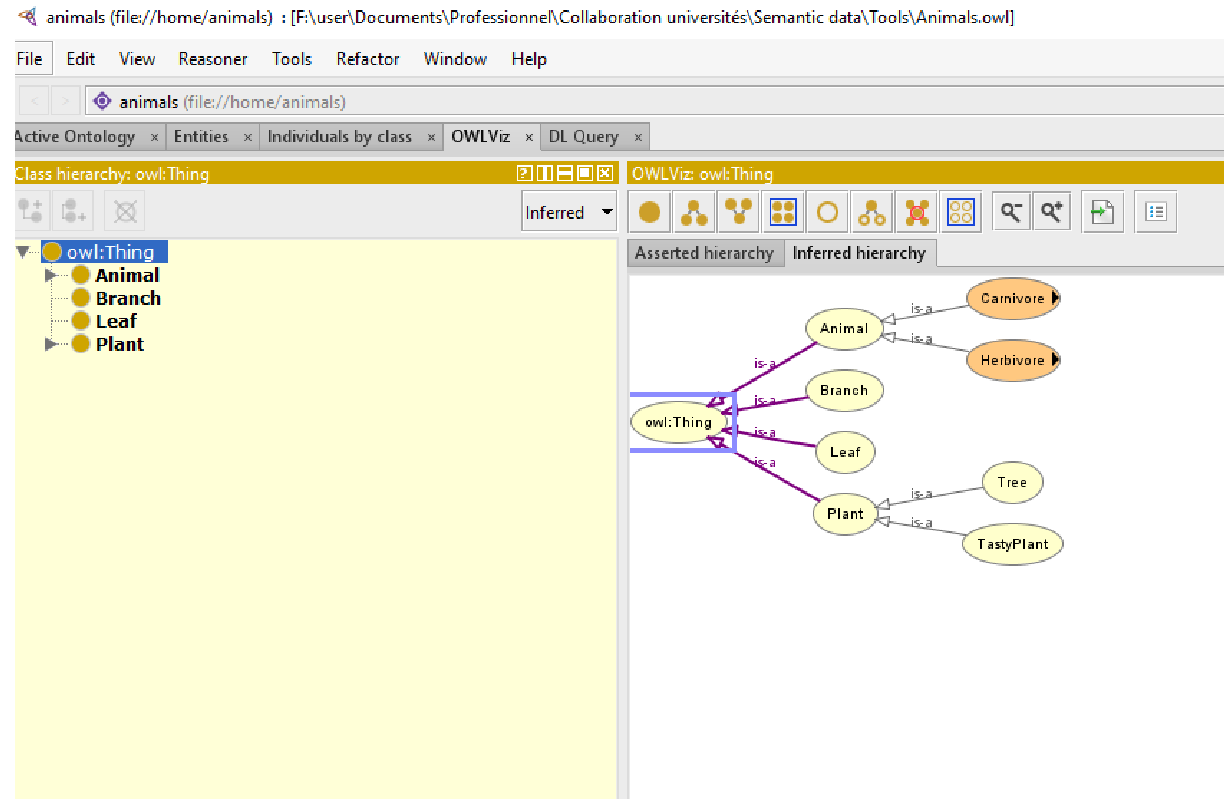
Visualizing the ontology

- If you have installed the OWLViz plugin you can visualize the ontology and browse through the hierarchy.
  - Defined classes are shown in orange, primitive classes in yellow.



Checking the consistency of the ontology

- A strength of Protégé is the capability to call a description logic reasoner to verify the consistency of the ontology.
  - Select the reasoner HermiT in Reasoner menu;
  - Do CTR-R to start the reasoner.
- One of the first tasks of the reasoner is to classify the Ontology.
  - It will place every class description in its proper place in the hierarchy and add inferred subclass links if needed.
  - It will also show inconsistencies in red.
  - The inferred hierarchy can be seen in OWLViz.
- In this case no new subclass and no inconsistency !



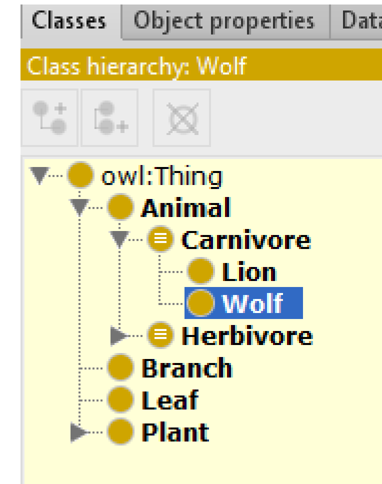
Adding a class and classifying it

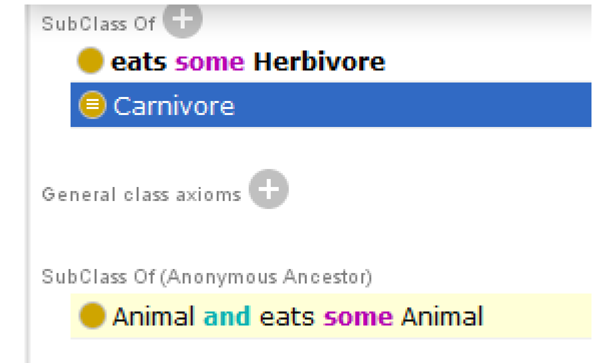
- Let us add the following class under owl :Thing *(go back to asserted view)* :

$\text{Wolf} \subseteq \exists eats.Herbivore$

- And then reclassify (*CTRL-R* or *Reasoner > Synchronize*)…

- We see that Wolf is reclassified as a Carnivore. The " ?" sign provides an explanation :
  - A Wolf is an Animal due to the domain restriction of eats;
  - An Herbivore is an Animal, so a Wolf is an Animal eating some Animal.





Explanation for Wolf SubClassOf Carnivore

- Show regular justifications
- Show laconic justifications
- All justifications
- Limit justifications to 2

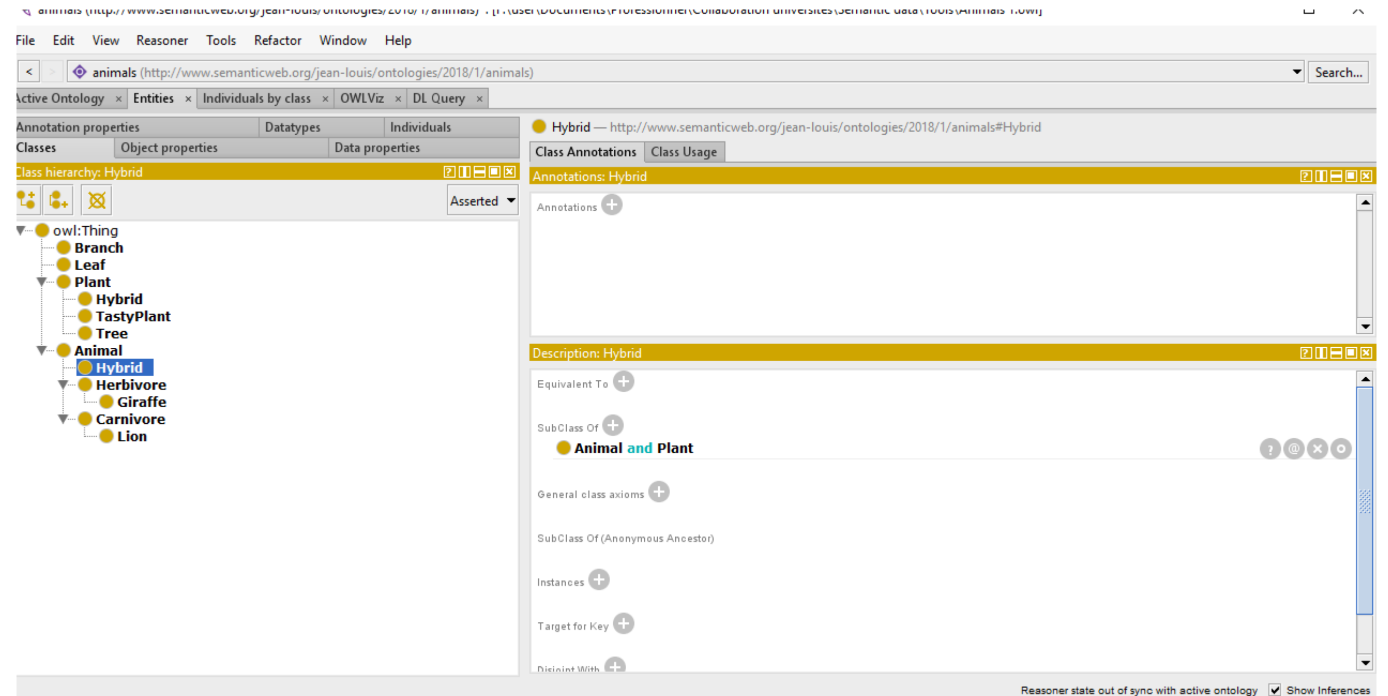
Explanation 1 — Display laconic explanation

Explanation for: Wolf SubClassOf Carnivore

	Axiom	
1)	Wolf **SubClassOf** eats **some** Herbivore	In ALL other justifications
2)	eats **Domain** Animal	In ALL other justifications
3)	Herbivore **SubClassOf** Animal	In NO other justifications
4)	Carnivore **EquivalentTo** Animal **and** (eats **some** Animal)	In ALL other justifications

Checking the consistency of the ontology

- Let us create a mistake !
- Add a new class Hybrid which is both an Animal and a Plant :
  - Define Hybrid as a subclass of OWL:Thing and a subclass of the expression Animal and Plant.

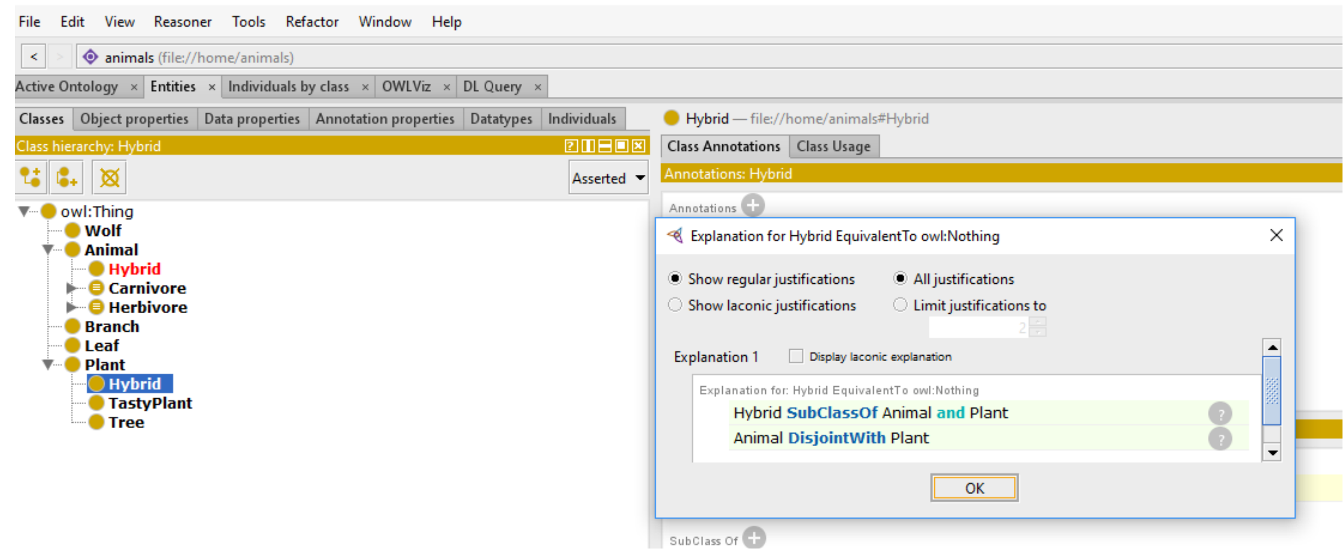


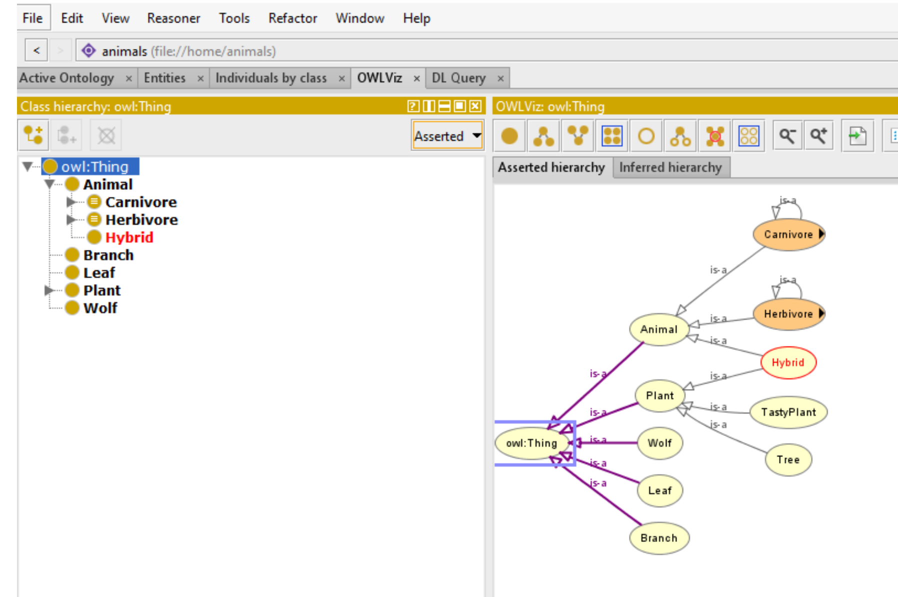
Checking the consistency of the ontology

- Launch the reasoner again.
- Hybrid appears in red and is indicated as the equivalent of owl:Nothing.

  => the concept cannot be satisfied.
- Explanation : Animal and Plant are disjoint !
- The mistake is also shown in OWLViz.
- To eliminate the mistake, suppress the class Hybrid *(asserted view, Entity/Classes tab, Delete button)* and run the reasoner again.

Note : the OWLViz tab may have to be closed and reopened to refresh the view.



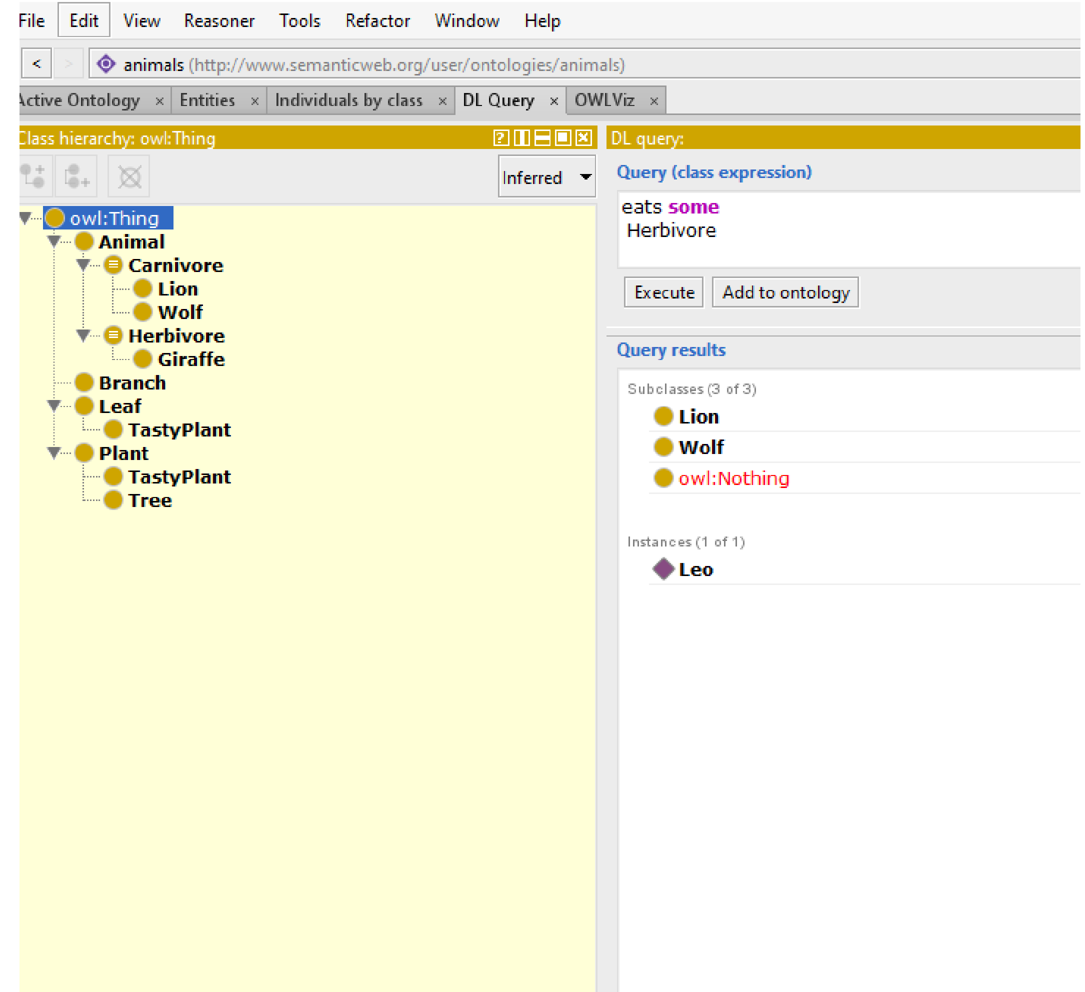


Querying with DL query tab

- When the ontology is completed, it can be tested by querying it with the DL Query view.
- The DL query use the Manchester syntax; the documentation is here.
- This tab is only usable if the ontology has been classified with a reasoner.
- Having a consistent ontology, let us ask DL queries :
- **Query 1** : eat some Herbivore ?
  - Answer : *Lion, Wolf, owl:NOTHING*
- Why is *owl:NOTHING* always a subclass?

The empty set is always a subset. To avoid displaying it you may uncheck it in the DL Query panel.

- To see instances in the answer, check it in the DL Query panel.



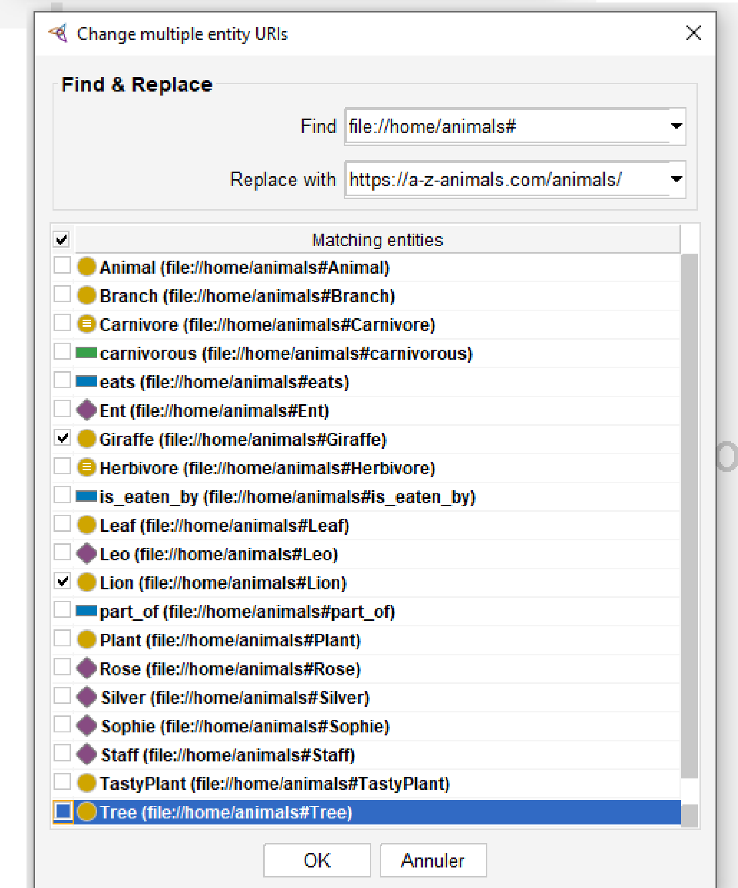
Querying with DL query tab ./.

- Query 2 : eat only Herbivore ?
  - Answer : *Lion* but also *Plant*, *TastyPlant*, *Tree* !
  - Why ?
- Look at the explanation :
  - The domain of eats is Animal; Plant is disjoint with Animal; hence no plant is eating in that ontology.
  - The universal restriction is true if it is applied to an empty domain !
- This is logically correct but a sign that the query or the ontology can be improved.
  - Can we make the query more specific ?
    Yes: eats some and eats only Herbivore.
    (owl :Thing does not need to be stated after some).
  - Does it make sense to specify what plants eat ?
    Possible (plants could eat nutrients, a subclass of "matter", but there are also carnivorous plants).

Manipulating URIs

- Full URI is displayed when hovering mouse over object.
- To change ontology URI :
  - Refactor>Change ontology URI

  Change ontology URI to https://a-z-animals.com/animals/.
  - Add a new class : Tiger. New names have the new ontology URI
- To change existing names to desired URIs
  - Use Refactor>Rename multiple entities; rename Lion and Giraffe.
- To check URIs : right click on object>show URI in web browser.
- Other classes may refer to a different URI base.
  - E.g., give Carnivore the URI https://en.wikipedia.org/wiki/Carnivore; use a similar base for Herbivore.
- Use File>Preferences>Renderer to control names rendering.
  - If the URI is not easily readable, you may use Render by annotation and give a name through the label annotation.



An exercise

- To complete this practice session :

Try to extend your ontology by adding carnivorous plants.

First obvious try : conflict !

- First obvious try : add a class Carnivorous plant with a role restriction eats some animal :

Animal
  Carnivore: eats some Animal
    Lion: eats only Herbivore
Plant
  CarnivorousPlant : eats some animal

- Starting the reasoner shows that this concept is not satisfiable !
- Why ?
- The domain of eats is Animal.

- Possible solutions ?

1. Remove or modify the domain restriction on eats.

How can it be modified ?

- Use a disjunction as domain : Animal or CarnivorousPlant

Is that the best option ?

It suppresses useful inferences.

We have added Wolf with restriction eats some Animal.
Using the domain restriction, Wolf was reclassified as Carnivore. That would no longer be possible.

- What are other options ?

2. Introduce a property, e.g. carnivorous.
3. Consider another classification axis : diet.

The carnivorous property version

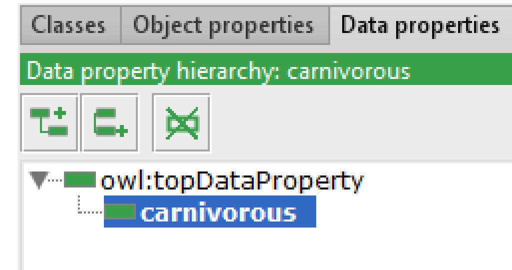
- Option 1 : formulate *carnivorous* as a boolean datatype property.
- We want to state the equivalent of the FOL formula:

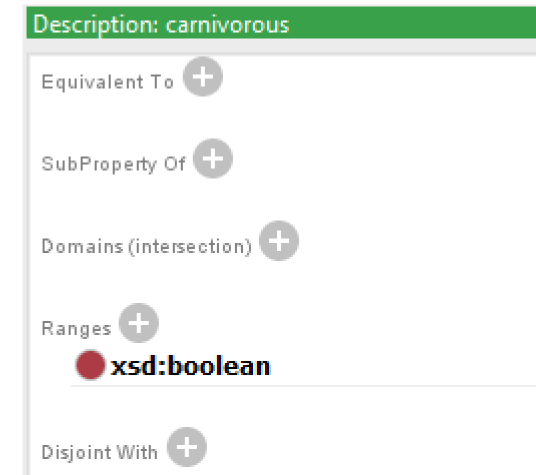
  carnivorous(CarnivorousPlant, true).

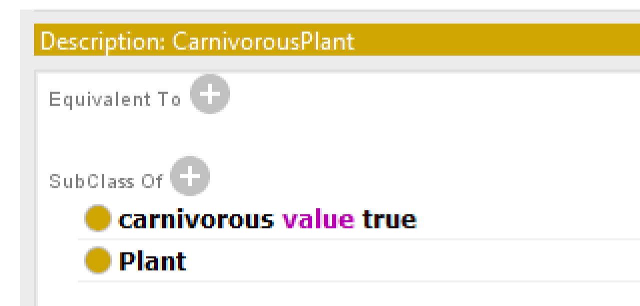
  and adapt our ontology.

1. Define carnivorous as a datatype property with a range *xsd:boolean*.
2. Add a value (owl:hasValue) constraint on the class CarnivorousPlant (using class expression editor) :

   carnivorous value true .







The carnivorous property version ./.

4. Do the same for Wolf then do a query to check behavior :

carnivorous value true ?

- One should have Wolf and CarnivorousPlant as answers.

5. The carnivorous property can be inherited.
   - Add the property carnivorous value true to Carnivore.
   - Run the same query.
   - One should have Carnivore (obviously), Lion and the instance Leo as additional answers.

The alternate classification axis version

- Option 2 : create a separate classification axis Diet with subclasses Carnivore and Herbivore.
- Motivations :
  - *Carnivore*, *Herbivore*, *Omnivore* is a distinction only based on diet, while *Mammal*, *Bird*… is based on common features of the species.
  - *Carnivore* can possibly apply to plants while *Herbivore* is only applied to animals.
- This requires a few modifications in the ontology and is left as an exercise.

Class or property, which version is best ?

- Some criteria :
  - Property related information is basically limited to property hierarchies, domains and range. If this is satisfactory for your modeling, the property version may be enough.
  - Classes may be involved in other relationships than hierarchies. If the concept needs to be reused / referenced in other relations or restrictions the class version is probably better.
  - The reasoning services include identification of instances. If the concept will have specific instances that need to be identified, a class-based approach is again better.

THANK YOU