Semantic Data

Practice 6 : Ontology Modeling

Jean-Louis Binot

Topics

1. Typical mistakes.
2. Expressing rules as general axioms
3. Designing and checking the class hierarchy.

To be read as documentation :

3. Term acquisition.
4. The quality property pattern.
5. Role composition.

Section 1: typical mistakes

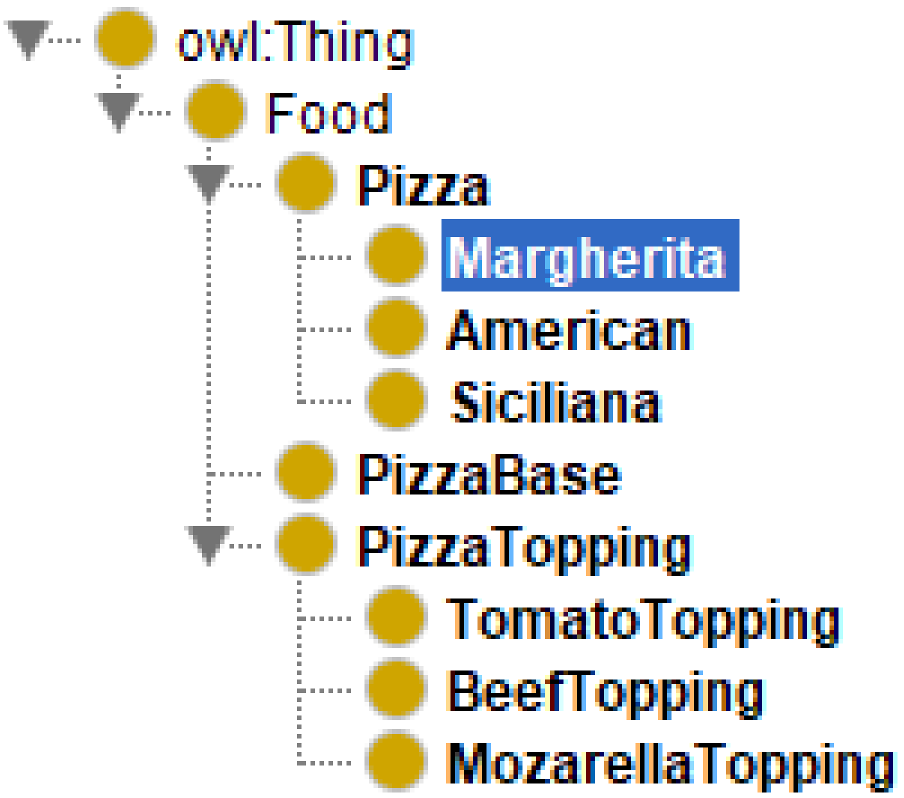
- Use of universals versus existential restrictions.
- Open world assumption and closure axioms.
- Understanding domain and range axioms.
- Expressing related information as part of name strings

This section is inspired from (Rector et al. 2004).

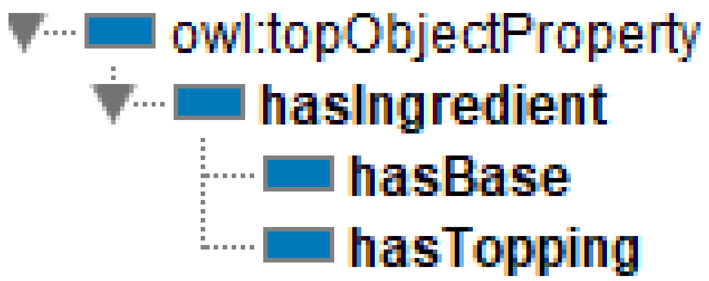
Section 1: typical mistakes

To get started :

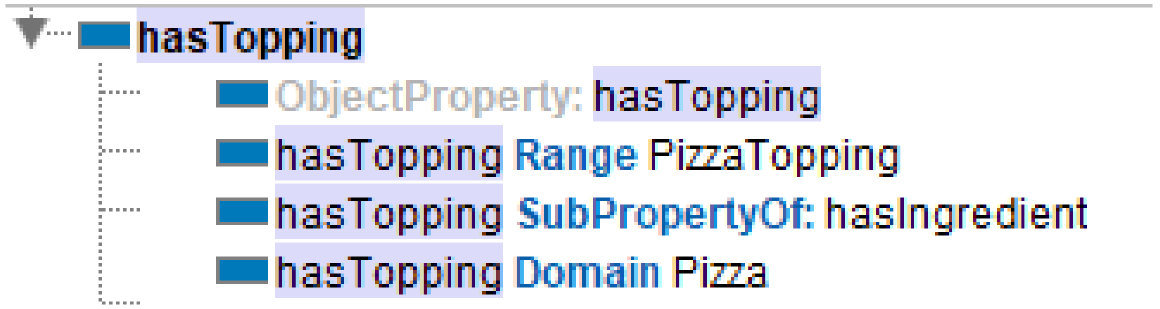
- Open a new ontology and save it (pizza.owl).
- Create the class hierarchy illustrated on the right.
- Do not forget to state that subclasses of Pizza are disjoint; same for PizzaTopping.
- Create object properties hasIngredient, hasBase and hasTopping as illustrated below.
- Save.



```
owl:Thing
  Food
    Pizza
      Margherita
      American
      Siciliana
    PizzaBase
    PizzaTopping
      TomatoTopping
      BeefTopping
      MozarellaTopping
```



```
owl:topObjectProperty
  hasIngredient
    hasBase
    hasTopping
```



```
hasTopping
  ObjectProperty: hasTopping
  hasTopping Range PizzaTopping
  hasTopping SubPropertyOf: hasIngredient
  hasTopping Domain Pizza
```

Use of universal versus existential restrictions

A Margherita is made with Mozzarella and Tomato (only).

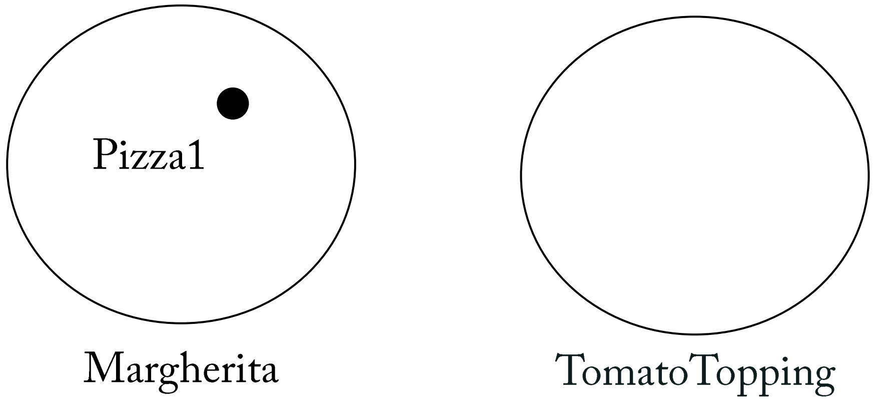
- Add to Margherita the restrictions :
  - hasTopping only MozzarellaTopping
  - hasTopping only TomatoTopping
- Create an instance :
  - Pizza1 of Margherita
- Ask in DLQuery :

Which pizzas have some tomato topping ?

- What happens ?

- No instance identified. Why ?

The following could hold :



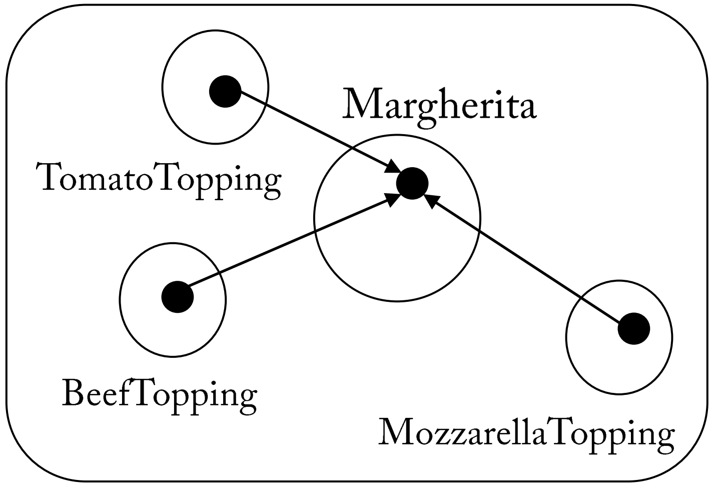
A universal restriction does not state "at least one" : it is trivially satisfied by an individual if that individual has no role corresponding to that restriction.

Open world assumption and closure axioms

- Transform the universal restrictions into existential ones.
  - hasTopping some MozarellaTopping and hasTopping some TomatoTopping
  - Test previous query (should work).
- Create some instances :
  - Tt of TomatoTopping
  - Mt of MozarellaTopping
  - Bt of BeefTopping

  Do not forget to state that individuals are different !
- Add individual object property assertions :
  - Pizza1 hasTopping Tt
  - Pizza1 hasTopping Mt
  - Pizza1 hasTopping Bt

- Ask in DLQuery : Margherita
  - Pizza1 identified (should be rejected). Why ?
- The following model is a model of our existential restriction :



- Open World Assumption : information non specified is not false, but unknown.

Open world assumption and closure axioms ./.

- Add a closure axiom by adding to the existential restriction a universal restriction :

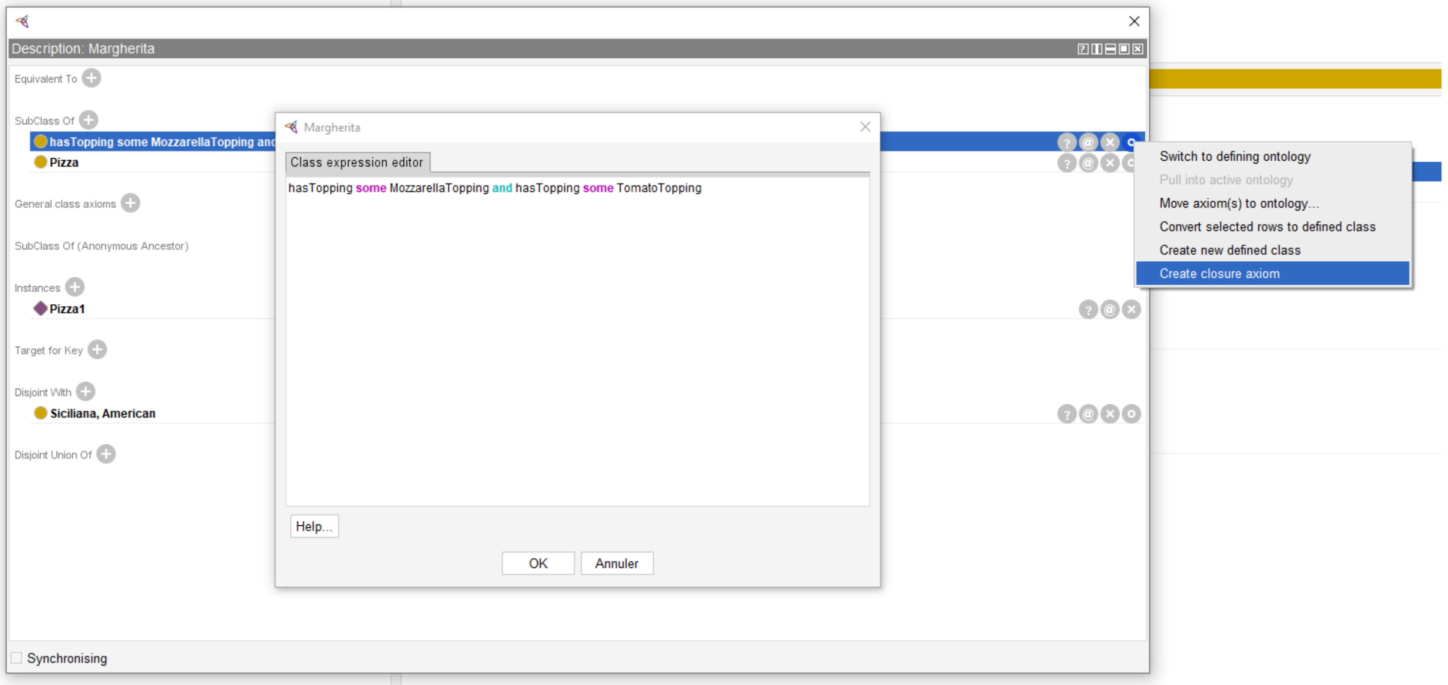
  In addition to : hasTopping some MozarellaTopping and hasTopping some TomatoTopping

  Add : hasTopping only (MozzarellaTopping or TomatoTopping)

- Result : inconsistent ontology (this is what was intended).
- Fix it by removing the beef topping assertion for pizza1.

Adding a closure axiom is an important logical design pattern in ontology modeling.

This important operation has a special menu feature in Protege: select a restriction, right-click on the edit button and select Create closure axiom



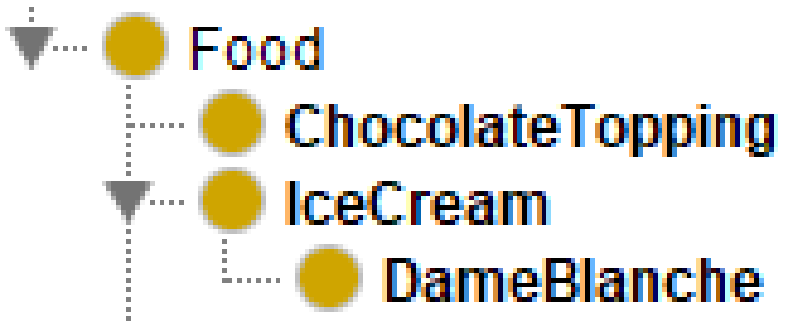
hasTopping some MozzarellaTopping and hasTopping some TomatoTopping

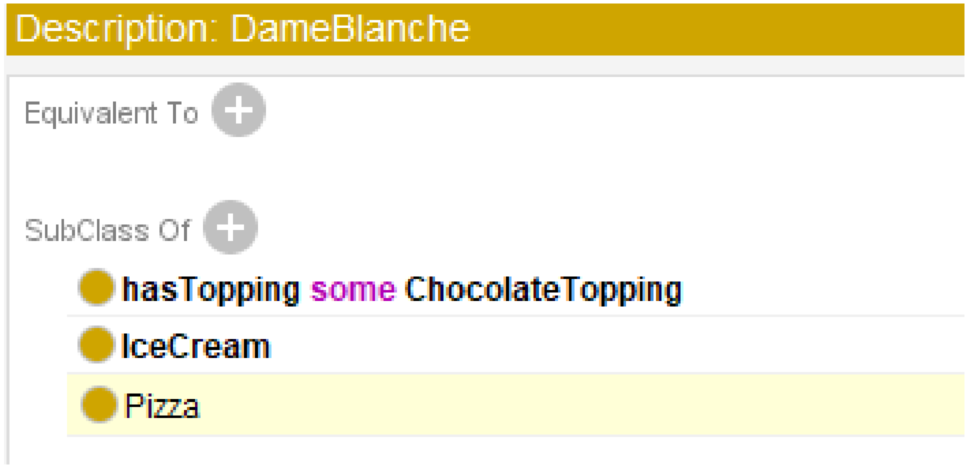
Understanding domain and range axioms

- Domain and range are axioms, not constraints.
  They are used to make inferences.
- To illustrate :
  - Add the classes on the right to your hierarchy.
  - Insert a restriction for DameBlanche :

  hasTopping some ChocolateTopping

  - Start the reasoner.
- What happens ?
  - DameBlanche is inferred as being a pizza, because the domain of hasTopping is Pizza
  - Possible solution : use distinct relations hasIceTopping and hasPizzaTopping with different domains
    (can be sub-relations of hasTopping)




Description: DameBlanche

Equivalent To

SubClass Of

hasTopping some ChocolateTopping

IceCream

Pizza

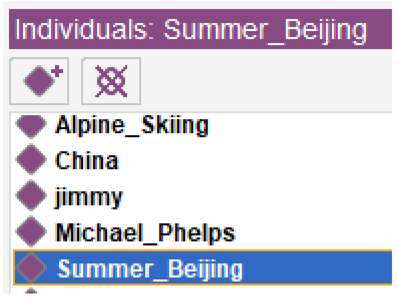
(Protégé file : pizzasimple.owl).

Avoid expressing related information as part of name strings

- Example 1 : location

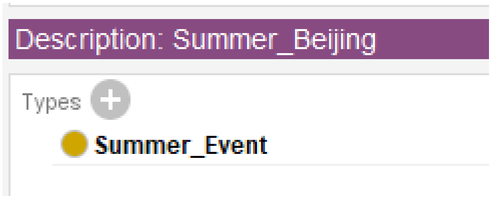
  City only present in string name of object (Olympics).

  - Maybe understandable by human beings but a program must extract the city name from the string !
  - Instead use a topology model : a game takes place in a city which is a part of a country.



Individuals: Summer_Beijing

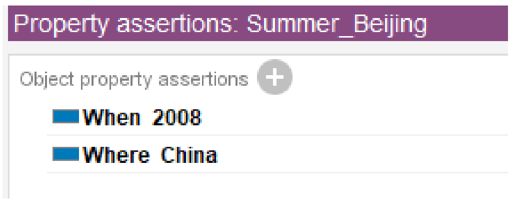
- Alpine_Skiing
- China
- jimmy
- Michael_Phelps
- Summer_Beijing



Description: Summer_Beijing

Types

- Summer_Event



Property assertions: Summer_Beijing

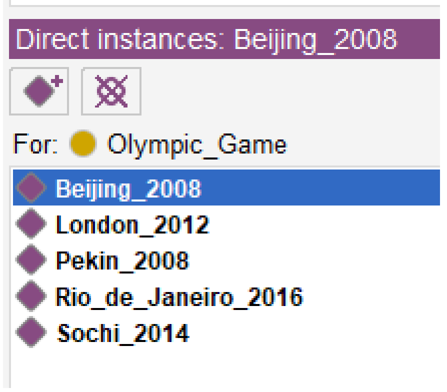
Object property assertions

- When 2008
- Where China

- Example 2 : time

  Date only present in string name of object (Olympics).

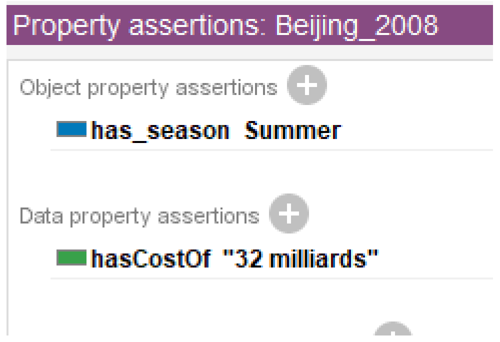
  - Again, maybe understandable by human beings but a program must extract the Olympics date from the string !
  - Instead use an explicit relation to some time information.



Direct instances: Beijing_2008

For: Olympic_Game

- Beijing_2008
- London_2012
- Pekin_2008
- Rio_de_Janeiro_2016
- Sochi_2014



Property assertions: Beijing_2008

Object property assertions

- has_season Summer

Data property assertions

- hasCostOf "32 milliards"

Section 2 : expressing rules as DL axioms

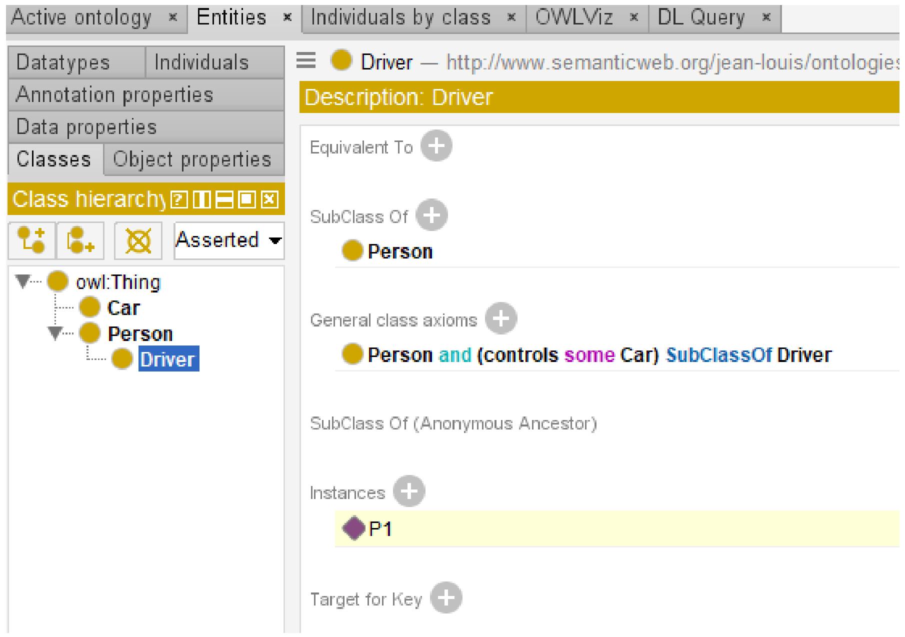
- It is possible to express (definite Horn clauses $^{(*)}$) rules as DL axioms (with limitations).

  A person who controls a car is a driver.

  - Horn rule :
    Person(?p) ^ controls(?p, ?c) ^ car(?c) -> Driver(?p)
  - FOL equivalent :
    $\forall x\ (\ (Person(x) \wedge \exists y\ controls(x, y)) \rightarrow Driver(x))$
  - General DL axiom :
    $Person \sqcap \exists controls.Car \sqsubseteq Driver$

- Create class hierarchy, add a general class axiom, add instances P1, P2 of Person, C1 of Car, and property assertion P1 controls C1. When you ask the DL query Driver, answer should be P1.

- We will look at further extensions (SWRL) in chapter 10.



Active ontology × | Entities × | Individuals by class × | OWLViz × | DL Query ×

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

Class hierarchy

Asserted

- owl:Thing
  - Car
  - Person
    - Driver

Driver — http://www.semanticweb.org/jean-louis/ontologies

Description: Driver

Equivalent To

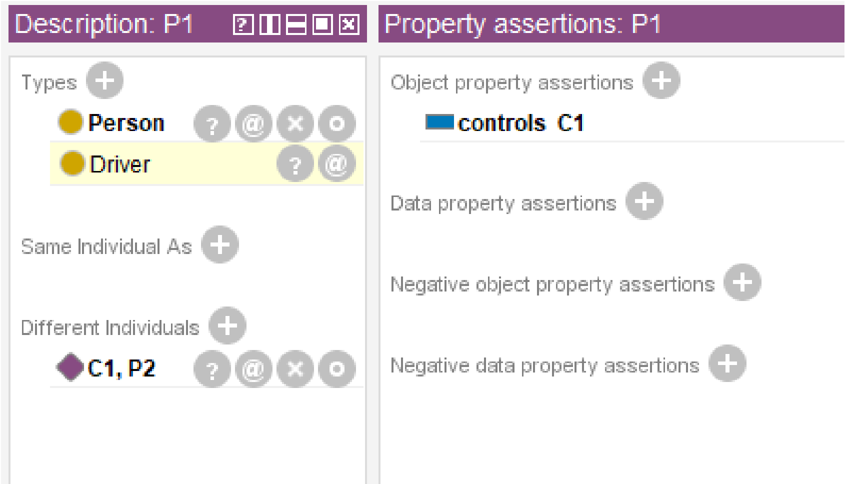
SubClass Of
Person

General class axioms
Person and (controls some Car) SubClassOf Driver

SubClass Of (Anonymous Ancestor)

Instances
P1

Target for Key



Description: P1

Types
Person
Driver

Same Individual As

Different Individuals
C1, P2

Property assertions: P1

Object property assertions
controls C1

Data property assertions

Negative object property assertions

Negative data property assertions

* : disjunctions of literals of which exactly one is positive.

Section 3 : designing and checking the class hierarchy

This section documents design criteria and quality checks for the class hierarchy :

- Basic design decision : class versus instance.
- General criteria to design and check the class hierarchy
  (reminder; was reviewed in chapter 5b. Ontology Modeling).
- Formal criteria to check the class hierarchy.

(the OntoClean part of this section is inspired from the course Semantic Web Technologies, H. Paulheim, Universität Manheim, and the paper on OntoClean from Guarino and Welty).

Class or instance ?

- In order to use a class, the following two relations must make sense :
  - Instantiation : are there situations like "X is a C" (x is a dog; x is a Pluto) ?
  - Subsumption : can we say "every C1 is a C2" (every Pluto is a dog) ?
- Instances define the most specific level of your model.

Romanée-Conti is a wine domain in Burgundy. In a wine ontology, will it be a class or an instance ?

- If we want only to talk about great wines, Romanée-Conti can be an instance.

Romanée-Conti is a GreatWine; a GreatWine is a Wine.

- If we want to discuss specific years as instances, we need Romanée-Conti as class.

Romanée-Conti-1961 is a Romanée-Conti.

- If we want to model a wine cellar, we may need to consider each bottle as an instance.

I have 3 bottles of Romanée-Conti-1961.

General criteria to design and check the class hierarchy

Reminder from chapter 5b : Ontology Modeling.

- Reuse the broad categories (e.g., objects, animates, events…) of an upper ontology.
- Respect the meaning of the subsumption relation (e.g., do not mix it with part_of).
- Check for synonyms : they normally have the same extension and not should not be distinct.
- Avoid cycles !
- Keep siblings at the same level of generality :
  - *WhiteWine* and *Chardonnay* should not be both direct subclasses of *Wine*. *WhiteWine* is more general.
- Have a reasonable span for the hierarchy structure at each level :
  - Too few subclasses (1 or 2) may indicate an incomplete ontology or a modeling problem.
  - Too many subclasses: some meaningful intermediate levels may be missing.

Formal criteria to check the class hierarchy

In addition to the basic criteria discussed above, three formal criteria to check the class hierarchy have been developed by Guarino and Welty as part of the OntoClean research *(Guarino and Welty 2009)* :

- ❑ Rigidity.
- ❑ Identity.
- ❑ Unity.

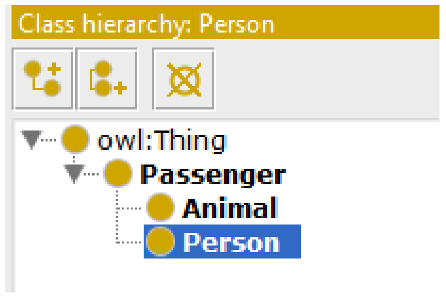
They are covered next.

Formal criteria 1 : class rigidity - example

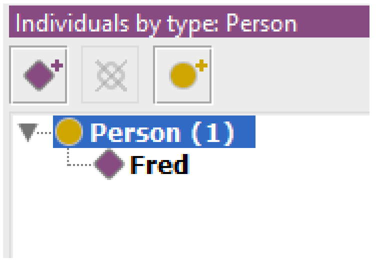
- Create a small ontology for public transport:
  - *Persons and animals can be passengers.*
  - *Fred is a Person.*
- Do a query on Passenger.
- Does the answer include Fred ?
  If so is that normal ?

  No; nothing says Fred should be a passenger.
  What is going on ?

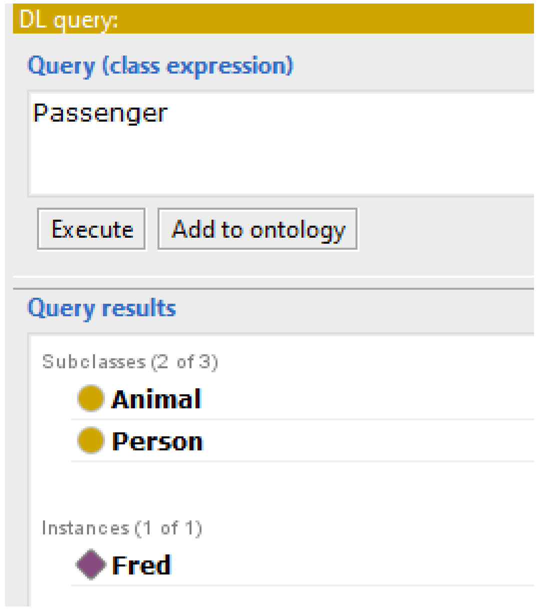
(Protégé file : passenger1.owl)


Class hierarchy: Person

```
owl:Thing
  Passenger
    Animal
    Person
```


Individuals by type: Person

```
Person (1)
  Fred
```


DL query:

Query (class expression)

```
Passenger
```

Query results

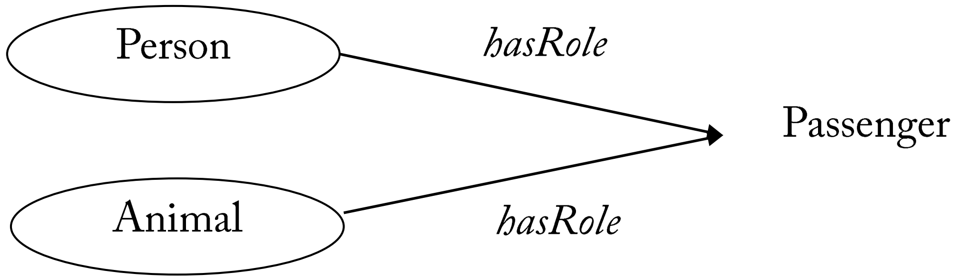
```
Subclasses (2 of 3)
  Animal
  Person

Instances (1 of 1)
  Fred
```

Class rigidity

- OntoClean distinguishes rigid and non rigid classes :
  - Membership in a rigid class must be permanent.

    If the individual is no longer in the class, it ceases to exist.
  - Person is a rigid class.
  - Passenger is not a rigid class : one may stop being a passenger.
- OntoClean rule : rigid classes must not be subclasses of non rigid classes.
- Solution : introduce a property linking the rigid class to the non-rigid one.



- Suggested exercise : implement that solution in Protégé *(Protégé file : passenger2.owl)*.

Formal criteria : class unity - example

- Create a small ontology for hydrology :

  Water is matter; an ocean is water; the Atlantic Ocean (instance) is an ocean.

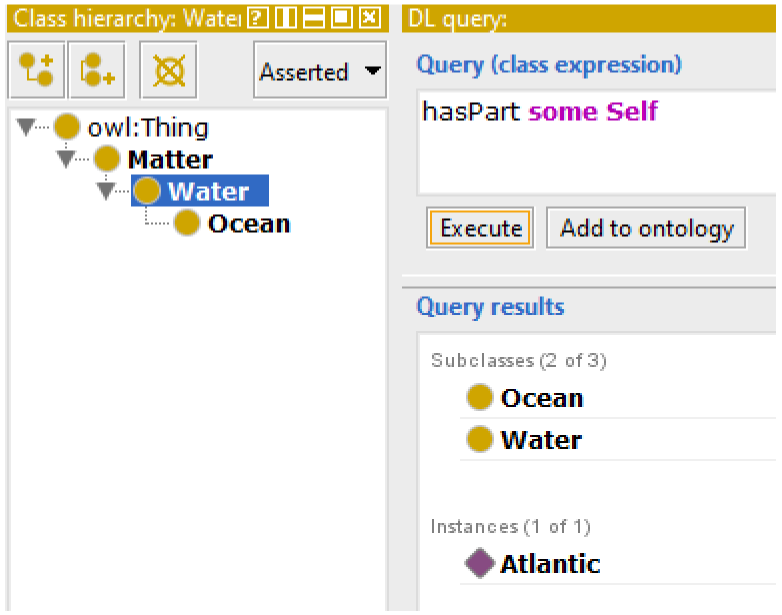
  Any part of water is water (mereological invariance).

  - For the second sentence, use an object property hasPart and add for Water the restriction subclass of hasPart some Self.

- Do a query on hasPart some Self. What happens ?

The Atlantic is part of the answer, and so considered as mereologically invariant. This is not normal.

(Protégé File : ocean1.owl).



```
hasPart some Self
```

Subclasses (2 of 3)
- Ocean
- Water

Instances (1 of 1)
- Atlantic

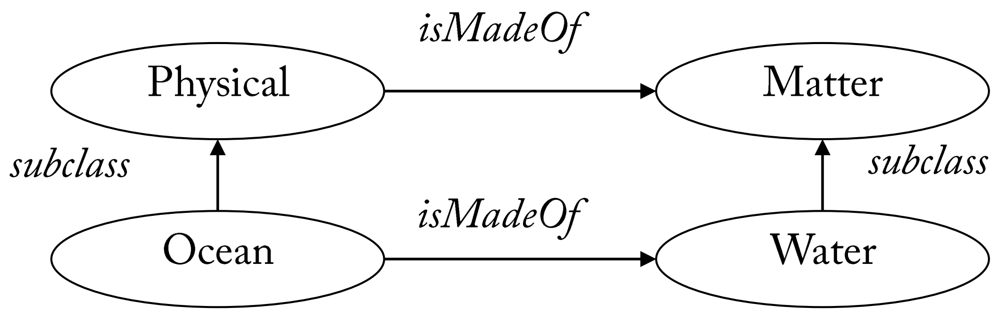
Class unity

- There is a distinction between amount of matter and physical objects (cf. chapter 5).

  Objects have individuation (belong to unity classes in OntoClean).

  Matter resists individuation (belong to anti-unity classes in OntoClean).

- OntoClean rule :
  - Unity classes may only have unity subclasses.
  - Anti-unity classes may only have anti-unity subclasses.
  - Solution : again introduce an additional relation.



- Suggested exercise : implement that solution in Protégé

  (Protégé file : ocean2.owl).

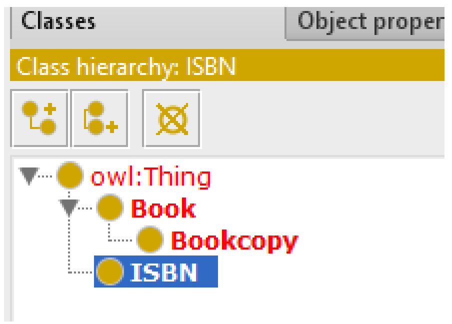
Formal criteria : class identity - example

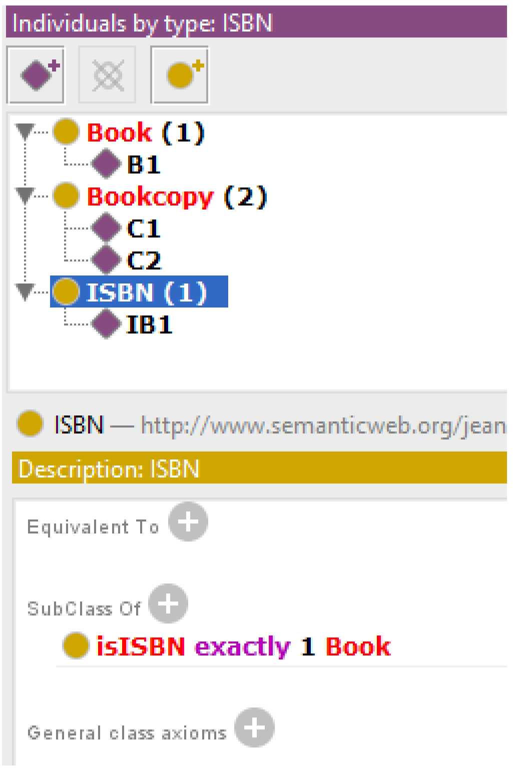
- Create a small ontology for a library :
  - *Each book has an ISBN. The library contains copies of books.*

    An ISBN is the ISBN of only one book.
  - Create classes, a hierarchy and instances of Book, Bookcopy, and ISBN. Do not forget to make all individuals different.
  - Link an instance of Book to an instance of ISBN through an object property hasISBN. Use the same property to link the same ISBN to an instance of Bookcopy to indicate that it is a copy of that book.
  - Define property isISBN as the inverse of hasISBN.

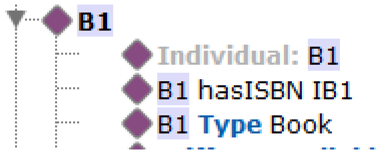
    To capture that *An ISBN is the ISBN of only one book*, use a restriction subclass : *isISBN exactly 1 Book*.
- Run a reasoner. What happens ?

  The ontology is not satisfiable. *(Protégé file : bookidentity1.owl)*





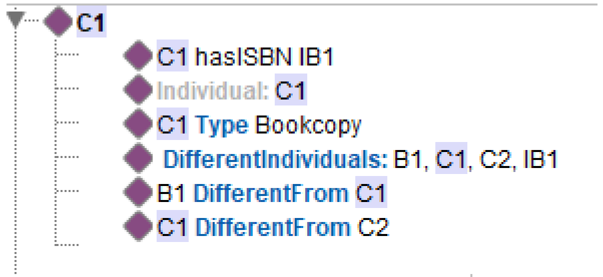
isISBN exactly 1 Book


B1

Individual: B1

B1 hasISBN IB1

B1 Type Book


C1

C1 hasISBN IB1

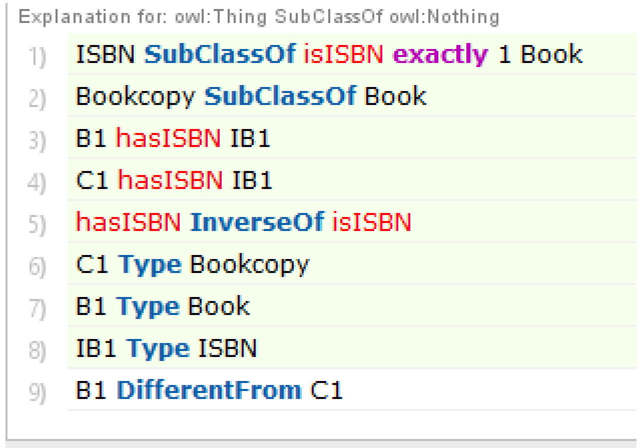
Individual: C1

C1 Type Bookcopy

DifferentIndividuals: B1, C1, C2, IB1

B1 DifferentFrom C1

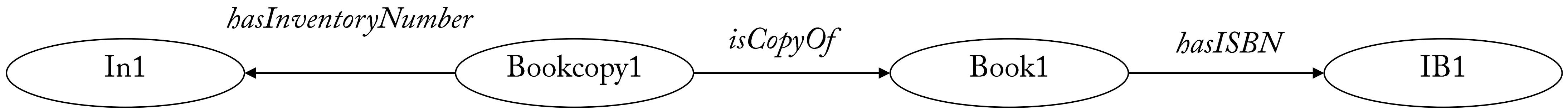
C1 DifferentFrom C2


Explanation for: owl:Thing SubClassOf owl:Nothing

1) ISBN SubClassOf isISBN exactly 1 Book
2) Bookcopy SubClassOf Book
3) B1 hasISBN IB1
4) C1 hasISBN IB1
5) hasISBN InverseOf isISBN
6) C1 Type Bookcopy
7) B1 Type Book
8) IB1 Type ISBN
9) B1 DifferentFrom C1

Class identity

- Books are identified by ISBNs, but book copies must be identified by more than just the ISBN, to make each copy distinct : e.g., a library inventory number.
- Fine, but it does not solve everything :
  - At the level of the class BookCopy the instances are different.
  - At the level of the class Book they are the same : they concern the same book.
- OntoClean rule : if p is a subclass of q, p cannot have identity criteria that q does not has.
  - Otherwise, the subclass would contain more instances that the superclass, which is nonsensical.
- Solution : introduce a property to link the book to the book copy.



- Suggested exercise : implement that solution in Protégé *(file bookidentity2.owl).*

Annexes: documentation to be read

- Section 3 : term acquisition
- Section 4 : the entity property quality pattern
- Section 5 : role composition

Section 3 : term acquisition (to read)

This section documents useful techniques to start the first phase of an ontology project.

- Steps involved (covered in next slides) :
  - 1 : Term extraction.
  - 2 : Term grouping.
  - 3 : Term normalization.
  - 4 : Term organisation.

(this section is based on a tutorial from Bechhofer and Sattler, COMP62342, University of Manchester)

Term acquisition – step 1 : term extraction

The aim is to build a small ontology of animals, domesticated or wild.

There are several sorts of domesticated animals, though by far the most are mammals (like us!). For example, our faithful pets, cats and dogs, are clearly domesticated (or we would not keep such dangerous carnivores in our homes), as is the delicious yet docile cow which is farmed in ever increasing numbers.

Step 1 : term extraction.

- *Identify (a) suitable text(s) for the targeted domain, such as the one shown on the left.*
- *Highlight the seemingly relevant terms in the text(s).*

Term acquisition – step 2 : grouping

- domesticated
- animals
- mammals
- us
- pets
- cats
- dogs
- dangerous
- carnivores
- homes
- delicious
- docile
- cow
- farmed
- increasing
- numbers

Step 2 : term grouping
- *Group the terms by relevant categories.*
- *Eliminate terms which do not seem part of the target domain.*

- Base animal categories (noun terms)
  - animals
  - cats
  - dogs
  - mammals
  - cow
  - us
- Ways an animal can be (adjective terms)
  - domesticated
  - pets
  - dangerous
  - carnivores
  - delicious
  - docile
  - farmed
- ~~Other stuff~~
  - ~~homes~~
  - ~~increasing~~
  - ~~numbers~~

Term acquisition - step 3 : normalize terms

- Base animal categories (noun terms)
  - animals
  - cats
  - dogs
  - mammals
  - cow
  - us
- Ways an animal can be (adjective terms)
  - domesticated
  - pets
  - dangerous
  - carnivores
  - delicious
  - docile
  - farmed

Step 3 : term normalization
- *Unify grammatical form and spelling.*
- *Find good names.*
- *Add terms from background knowledge.*

- Base animal categories (noun terms)
  - Animal
  - Cat
  - Dog
  - Mammal
  - Cow
  - Human
- Ways an animal can be (adjective terms)
  - domesticated
  - wild
  - pet
  - dangerous
  - carnivorous
  - omnivorous
  - herbivorous
  - delicious
  - docile
  - farmed

Term acquisition - step 4 : organize and define terms

- Base animal categories
  - Animal
  - Mammal
  - Cat
  - Dog
  - Cow
  - Human
- Ways an animal can be
  - Domesticated
  - wild
  - dangerous
  - carnivorous
  - omnivorous
  - herbivorous
  - delicious
  - docile
  - pet
  - farmed

- Base animal categories
  - Animal - eats some things
  - Mammal - has (as parts) mammal glands
  - Cat
  - Dog
  - Cow - eats only grass
  - Human - is an omnivore
- Ways an animal can be (adjective terms)
  - Domesticated
  - wild
  - dangerous
  - carnivorous - eats only meat
  - omnivorous - eats meat and plants
  - herbivorous - eats only plants
  - delicious - tastes good
  - docile
  - pet - lives with human
  - farmed - is eaten by human

Step 4 : term organisation
- *Reorder from general to particular.*
- *Start building definitions in natural language.*
- *Next step : use the ontology editor to formalize.*

Section 4 : the entity property quality pattern (to be read)

- Example : try to create a small ontology about physical objects and measure units.

  Physical objects have a weight.

  Object O1 weights 5,5 Kg.

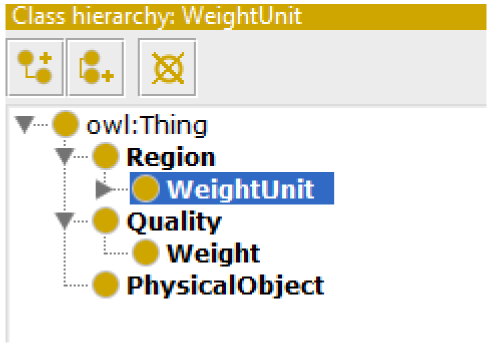
  Refer to the ideas concerning qualities of the DOLCE top ontology.

- A possible solution in triples :

  O1 a PhysicalObject .
  O1 hasWeight W1 .
  W1 a Weight .
  W1 hasUnit Kg .
  Kg a WeightUnit .
  W1 hasValue 5,5 .

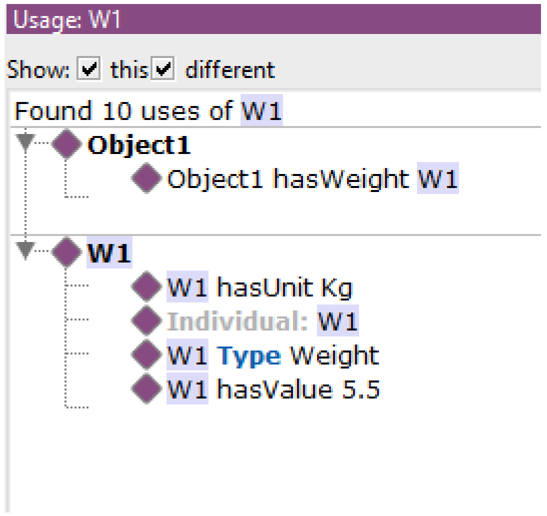
- The solution in Protégé :

  (File : measureunits.owl).



Class hierarchy: WeightUnit

- owl:Thing
  - Region
    - WeightUnit
  - Quality
    - Weight
  - PhysicalObject

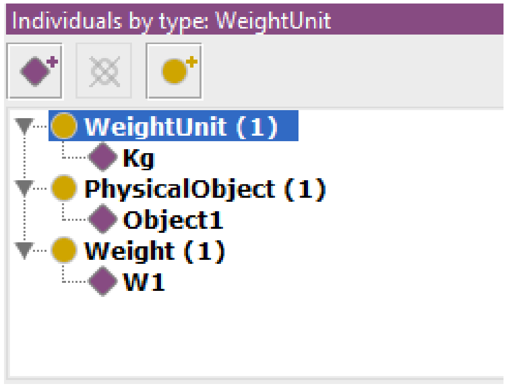


Usage: W1

Show: ✓ this ✓ different

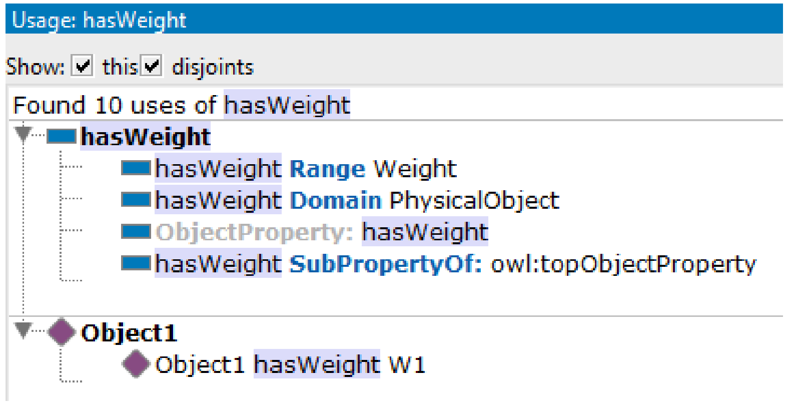
Found 10 uses of W1

- Object1
  - Object1 hasWeight W1
- W1
  - W1 hasUnit Kg
  - Individual: W1
  - W1 Type Weight
  - W1 hasValue 5.5



Individuals by type: WeightUnit

- WeightUnit (1)
  - Kg
- PhysicalObject (1)
  - Object1
- Weight (1)
  - W1



Usage: hasWeight

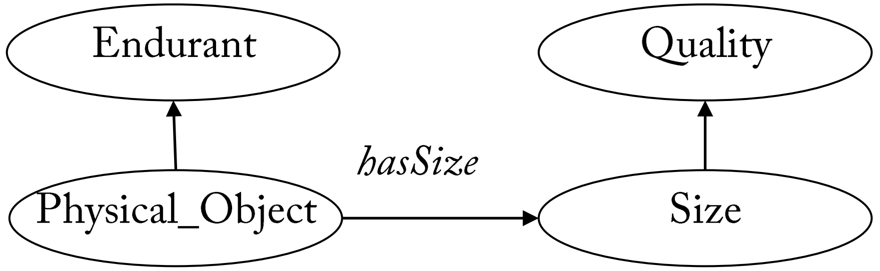
Show: ✓ this ✓ disjoints

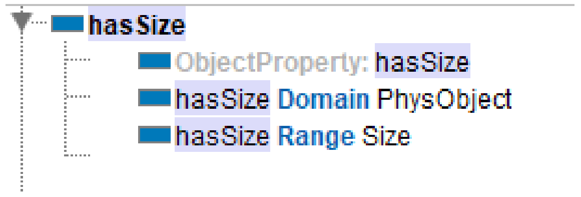
Found 10 uses of hasWeight

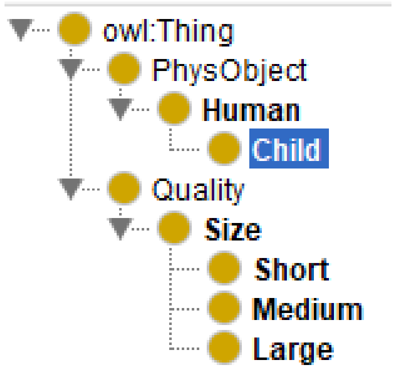
- hasWeight
  - hasWeight Range Weight
  - hasWeight Domain PhysicalObject
  - ObjectProperty: hasWeight
  - hasWeight SubPropertyOf: owl:topObjectProperty
- Object1
  - Object1 hasWeight W1

The entity property quality pattern ./.

- Used to model descriptive features of things.
- Elements:
  - For each feature or quality such as *size*, *weight*, etc. :
  - Use a functional property, e.g., *hasSize;*
  - Use a class for its values, e.g., *Size;*
  - State that the class is the **range** of the property;
  - State to which classes these qualities apply via the **domain** of the property.
- Using classes for descriptive features allows to make sub-partitions :
  - E.g., small, medium, large …
  - May overlap or be disjoint.
  - May have further properties.




hasSize
ObjectProperty: hasSize
hasSize Domain PhysObject
hasSize Range Size


owl:Thing
PhysObject
Human
Child
Quality
Size
Short
Medium
Large

Class: Child SubClassOf: Human, hasSize only Small
Class: Human SubClassOf hasSize some Size
DisjointClasses: Large, Medium, Small
Class Size: DisjointUnionOf: Large, Medium, Small

Section 5 : role composition – to be read

This example illustrates how to use composition of roles.

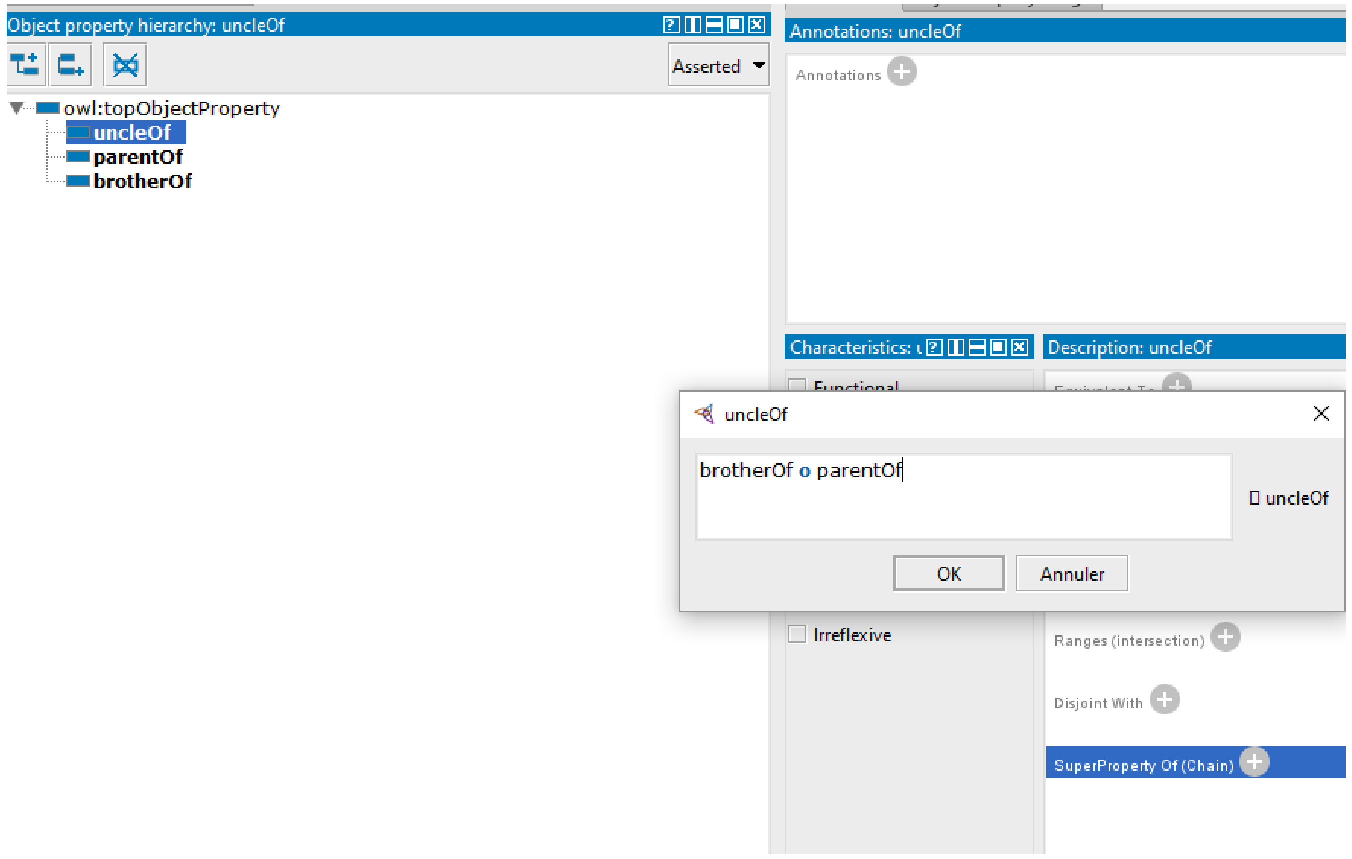
- Create a class Person
- Create 3 instances: *John*, *Marc*, *Fred*
- Define 3 properties at same hierarchy level

  uncleOf, *brotherOf*, *parentOf*

- Define the axiom

  $\mathit{hasParent} \circ \mathit{hasBrother} \subseteq \mathit{hasUncle}$

  Use SubProperty of(chain) and a lowercase o for role composition.



brotherOf o parentOf

□ uncleOf

Section 5 : role composition – to be read ./.

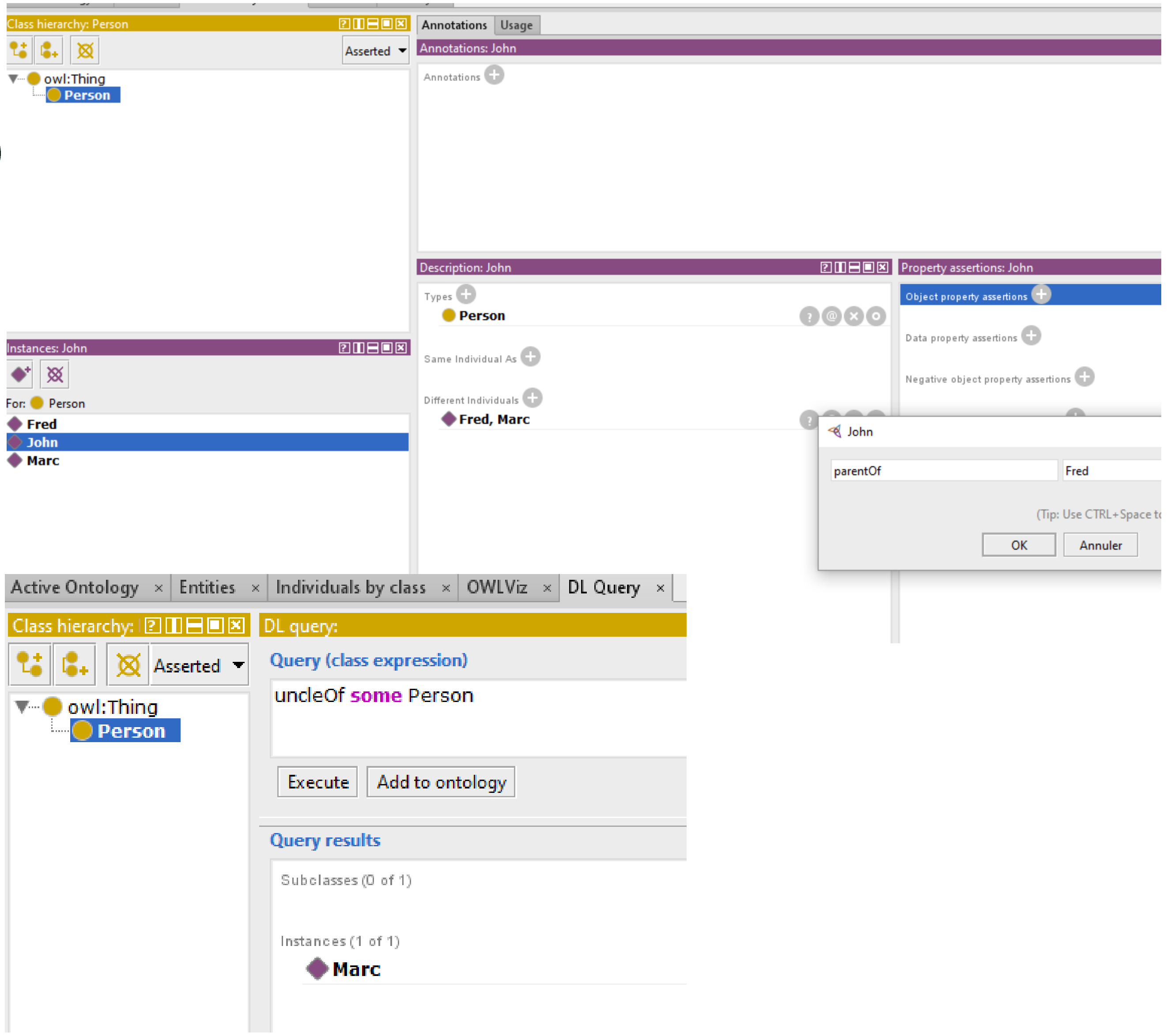
- ❑ Add instance axioms (object property assertions)

  Marc brotherOf John, *John parentOf Fred*

- ❑ Ask query *uncleOf some Person*

  Answer should be *Marc*.

(Protégé File : uncle.owl).



```
uncleOf some Person
```

Subclasses (0 of 1)

Instances (1 of 1)

Marc

References

- *[Guarino and Welty 2009] : Guarino N. and Welty C.*, An overview of OntoClean, *in (Staab and Studer, eds.), Handbook on Ontologies 2nd edition, Springer, 2009.*

- *[Rector A. et al. 2004]: Rector A. et al.*, OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns, *in (Motta E., Shadbolt N.R., Stutt A., Gibbins N. (eds)) Engineering Knowledge in the Age of the Semantic Web, EKAW 2004. Lecture Notes in Computer Science, vol 3257. Springer, Berlin, Heidelberg, 2004.*

THANK YOU