Knowledge Representation Laboratory 8:
OWL

CITS3005

This laboratory will involve writing some basic OWL ontologies in Protégé . The following pages contain an example ontology taken from *Ontologies with Python Programming OWL 2.0 Ontologies with Python and Owlready2*, by Lamy Jean-Baptiste.

Protégé is a graphical ontology editor that can be downloaded from `https://protege.stanford.edu/`

1. Go through the getting started guide at `http://protegeproject.github.io/protege/getting-started/` to familiarise yourself with the key elements of the interface.

2. Build a pizza ontology, following the directions at `https://protegewiki.stanford.edu/wiki/Protege4Pizzas10Minutes`.

3. Run through see the Bacteria Ontology on the following pages and the attempt the following exercises:

   (a) In the bacteria ontology, add an individual of the Staphylococcus class having a rod shape. Run the reasoner; what do you observe?

   (b) Using the Protégé editor, extend the ontology of bacteria by adding the catalase test. This biological test helps to identify bacteria, and its result can be positive or negative. The catalase test is positive for Staphylococci and Pseudomonas, negative for Streptococci.

   (c) Using the Protégé editor, extend the bacteria ontology by adding the colour of the bacteria. Staphylococci are white or golden (this is the famous *Staphylococcus aureus*), Streptococci are translucent, and Pseudomonas are generally coloured (that is to say, not white).

   (d) Using the Protégé editor, add a new class of bacteria: *Mycobacterium leprae* (Hansen's bacillus, which causes leprosy). This species of bacteria is Gram positive, rod shape, and isolated or grouped in pairs. The catalase test is not relevant for this bacterium because it is very difficult to grow in vitro. The colour is yellow. Finally, all of these characteristics are sufficient to identify the bacteria.

   (e) In the Protégé editor, add an individual of the class Bacterium, rod shape, isolated, and yellow in colour. Check that this individual is properly classified as *Mycobacterium leprae*.

   (f) In the ontology of bacteria, add a disjunction between the different subclasses of Bacteria (Staphylococci, Streptococci, Pseudomonas, etc.). Does this change the result of the reasoning on the unknown bacterium?

3.3 Example: An ontology of bacteria

In order to illustrate the construction of an ontology and the possibilities it can offer, we will take as an example an ontology of bacteria. This ontology aims to describe bacteria and their physical and chemical characteristics. We will, however, limit ourselves to a few simple characteristics and a small number of species for obvious reasons of brevity. I apologize in advance to my biologist readers for the sometimes crude simplifications that we will have to carry out—the conception of a complete and exact ontology of bacteria would constitute a real research work in its own right!

We will only retain the following three characteristics for describing bacteria:

1. Their shape: Bacteria can be round or rod shaped (elongated shape).
2. Their grouping: Bacteria can be isolated from each other or grouped in pairs, in clusters, or in chains, which can be small or long chains.
3. Their Gram status: Gram + bacteria are colored by the Gram test, unlike Gram – bacteria.

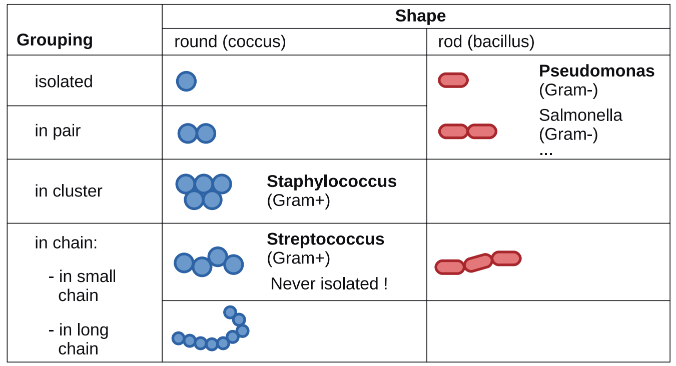
Figure 3-1 shows a classification of bacteria according to these characteristics. Round bacteria are called "coccus", and rod ones are called "bacillus".

In addition, we will only retain the following three families of pathogenic bacteria:

1. Staphylococcus: Round shape, grouped in clusters, Gram +
2. Streptococcus: Round shape, grouped in small chains but never isolated, Gram +

3. Pseudomonas: Rod shape, grouped in pairs or isolated, Gram –

Thereafter, we will consider that a bacterium can have several groupings: indeed, the observation never relates to a single bacterium but on several. It is therefore common to observe several groupings for the same species of bacteria: for example, Staphylococci which group in clusters may occasionally be present singly or in pairs. However, Streptococci are never isolated but always grouped (in pairs, in clusters, and, of course, preferably in chains).



Grouping	Shape	
	round (coccus)	rod (bacillus)
isolated		**Pseudomonas** (Gram-) Salmonella (Gram-) ...
in pair		
in cluster	**Staphylococcus** (Gram+)	
in chain: - in small chain	**Streptococcus** (Gram+) Never isolated !	
- in long chain		

Figure 3-1. Simple classification of bacteria according to three criteria

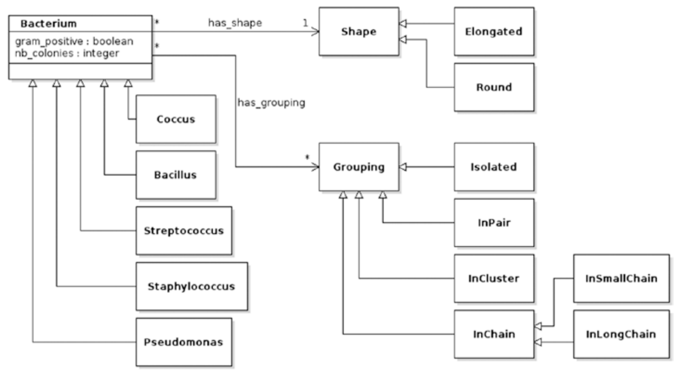


Figure 3-2. *UML class diagram of the bacteria ontology*

Figure 3-2 gives the class diagram in UML (Unified Modeling Language). Please note, however, that ontologies allow more information to be represented than what appears on the class diagram. For example, (practically) all Gram + bacteria of round form grouped in clusters are Staphylococci. For this species, it will therefore be possible to deduce the class of bacteria, its shape, grouping, and Gram status. On the contrary, Pseudomonas are not the only bacteria of rod shape, isolated, or in pairs. This is an important difference because it will impact automatic reasoning; however, a "classic" object model (like that of Python; see 2.9) does not allow taking it into account.

At the very beginning of this chapter, we defined an ontology as "as independent as possible from the intended application". For example, the ontology of bacteria could have multiple applications, such as:

- Create an encyclopedic website describing the properties of the different bacteria (see 4.12)
- Facilitate the entry or extraction of information on bacteria (see 5.14)

- Help identify an unknown bacterium (see 7.7)
- Enrich with information on bacteria already existing ontologies or resources, such as UMLS (see 9.10)
- Facilitate statistical studies in a hospital by allowing the grouping of similar bacteria (to answer questions such as "has the number of infections with anaerobic bacteria increased in the last month?")

Each of these applications could be achieved with a specific knowledge base. For example, the identification of bacteria could be done with a knowledge base composed of rules like the following one:

IF shape = round AND grouping = in cluster AND gram = '+'

THEN staphylococcus

However, an ontology is capable of achieving **all** these applications from the same source of knowledge, which greatly facilitates the maintenance and reuse of this knowledge.

In the following sections, we will build a (small) formal ontology in OWL from this classification of bacteria, using the Protégé editor.

3.4 Creating a new ontology

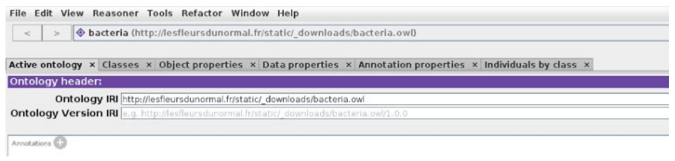
When you launch the Protégé editor, it automatically creates a new empty ontology. The editor includes several tabs; by default, the Active Ontology tab is displayed.

In this tab, we will define the IRI of our ontology. The IRI is the "name" of the ontology, and this name takes the form of an Internet address. Please note, however, the IRI must be in the form of an Internet address, but the ontology does not need to be available on the Internet at this address! It is thus usual to create ontologies whose IRI begins with "`http://www.semanticweb.org/`" or "`http://www.test.org/`", without holding the rights to these Internet domain names.

We will call our bacteria ontology:

`http://lesfleursdunormal.fr/static/_downloads/bacteria.owl`

(NB: This Internet address points to my personal site, on which you can actually download the full ontology). You can enter this IRI in the "Ontology IRI" field of Protégé, as shown in the following screenshot:



File Edit View Reasoner Tools Refactor Window Help

bacteria (http://lesfleursdunormal.fr/static/_downloads/bacteria.owl)

Active ontology × Classes × Object properties × Data properties × Annotation properties × Individuals by class ×

Ontology header:

Ontology IRI http://lesfleursdunormal.fr/static/_downloads/bacteria.owl

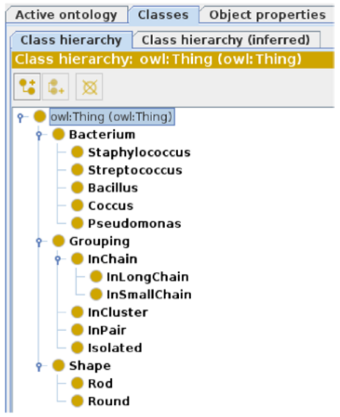
Ontology Version IRI e.g. http://lesfleursdunormal.fr/static/_downloads/bacteria.owl/1.0.0

Annotations

You can then save the ontology in RDF/XML format, in a file that you will call "bacteria.owl". Do not forget thereafter to regularly save the ontology during its edition.

3.4.1 Classes

In Protégé, the "Classes" tab allows you to navigate through existing classes and to create new classes. The buttons and allow you to create a new daughter or sister class of the selected class, respectively. Using these buttons, we can create a class hierarchy corresponding to our previous UML model, as in the following screenshot:



Active ontology Classes Object properties

Class hierarchy Class hierarchy (inferred)

Class hierarchy: owl:Thing (owl:Thing)

```
owl:Thing (owl:Thing)
  Bacterium
    Staphylococcus
    Streptococcus
    Bacillus
    Coccus
    Pseudomonas
  Grouping
    InChain
      InLongChain
      InSmallChain
    InCluster
    InPair
    Isolated
  Shape
    Rod
    Round
```

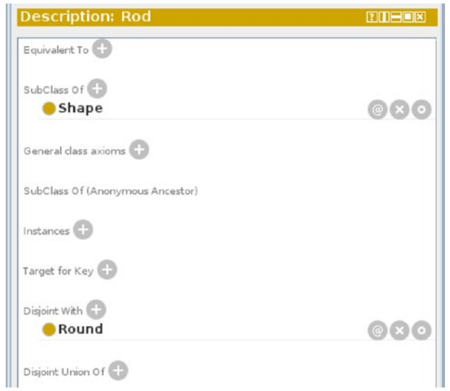
In ontologies, inheritance is also called "is-a relationship": for example, we can say that a Pseudomonas *is a* Bacterium.

3.4.2 Disjoints

An important difference between an ontology and an object model is as follows: in an ontology, an individual can belong to *several* classes. Therefore, a given shape could very well be *both* round and rod! The Open-World assumption allows this type of interpretation: anything that is not formally prohibited is considered possible.

In our ontology of bacteria, we want to prohibit this: a given shape is either round or rod, but cannot be both at the same time. For this, we must declare the two classes Round and Rod as *disjoint*. Two disjoint classes cannot have individuals in common.

The disjoint classes are declared in the "Description" panel of the "Classes" tab. We will select the Rod class and then click the "+" button to the right of the "Disjoint with" section and choose the Round class in the "Class hierarchy" tab of the dialog box. You should get the following result:

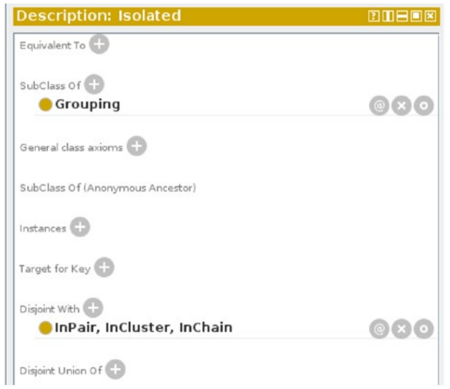

Description: Rod
Equivalent To
SubClass Of
Shape
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
Target for Key
Disjoint With
Round
Disjoint Union Of

The two classes are now disjoint. Note that it is not necessary to declare the second class (Round) disjoint from the first (Rod): this is automatically deduced from the previous declaration.

In the same way, the InSmallChain class must be declared disjoint from the InLongChain class.

The Isolated, InPair, InCluster, and InChain classes must be declared as *pairwise disjoint*: that is to say that any pair made up of two classes from this list are disjoint. To do this, simply select one of the classes

(e.g., Isolated), click the "+" button to the right of "Disjoint with", and select the other three classes simultaneously (by pressing the control key, not by clicking three times the "+" button!). The result should be as follows:


Description: Isolated
Equivalent To
SubClass Of
Grouping
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
Target for Key
Disjoint With
InPair, InCluster, InChain
Disjoint Union Of

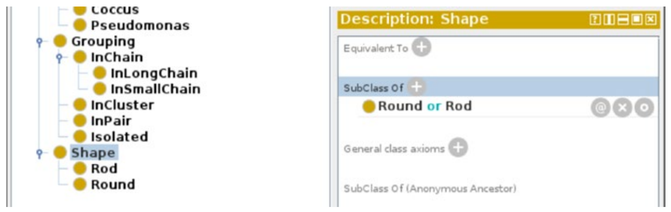
Attention, concerning the subclasses of Grouping, the disjoint does not mean that a given bacterium cannot be observed with two different groupings (e.g., Isolated or InPair, like Pseudomonas). The disjoint only means that a given grouping cannot be both Isolated and InPair, but it does not prohibit a bacterium from having two distinct groupings, one of the class Isolated and the other of the class InPair.

In the same way, the classes Bacteria, Shape, and Grouping must be declared disjoint: for example, a geometric shape cannot be the same thing as a bacterium! It may seem obvious to a human, but remember that it is not to a machine. Ontologies seek to formalize knowledge comprehensively, including the most obvious piece of knowledge.

3.4.3 Partitions

We have defined two classes of shapes, Round and Rod, which are now disjoint. However, we have not excluded the existence of other shapes, for example, triangular. Again, the Open-World assumption makes such interpretations possible. However, there are only two possible shapes for a bacterium: Round or Rod. We must declare that all Shape is either Round or Rod: it is a partition (we will say that the classes Round and Rod constitute a partition of the class Shape).

To do this, we select the Shape class, and, in the “Description” panel, we click the “+” to the right of “SubClass Of”. This “+” button allows you to add superclasses to the class; these can be named classes, but also OWL logical constructors, like here. In the dialog box that appears, we select the “Class expression editor” tab, and we enter the constructor “Round or Rod”. You should obtain the following result:



This constructor “or” allows two classes to be linked with a logical OR (also called a *union*, when we think in set logic). It means that the Shape class is a subclass of the union of the Rod and Round classes. Consequently, any shape is now either round or rod, and there are therefore no other possible shapes.

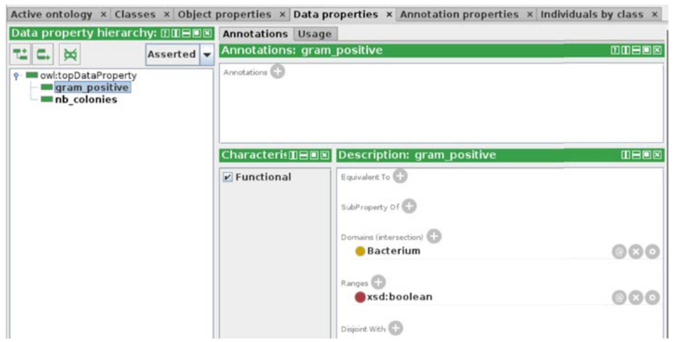
In the same way, we must partition InChain (SubClass Of “InSmallChain or InLongChain”) and Grouping (SubClass Of “Isolated or InPair or InCluster or InChain”).

3.4.4 Data properties

We will now deal with the properties. In ontologies, unlike object-oriented programming, properties are defined independently of classes. OWL considers three categories of properties: data properties whose values are data (numbers, texts, dates, Booleans, etc.), object properties whose values are entities (i.e., ontology individuals), and annotation properties which do not intervene in semantics or reasoning and can therefore mix data and entities without restriction.

In Protégé, the “Data Properties” tab allows you to create data properties. OWL supports inheritance between properties, in addition to inheritance between classes; however, we will not use it here. Using the and buttons, which work similarly to those for classes, we will create two new data properties called “gram_positive” and “nb_colonies”. This last property will not be really useful to describe bacteria, but it will serve as an example of numeric data property.

You should arrive at the following result:



Each data property can be configured by specifying:

- Its *domain* ("Domains (intersection)" in Protégé): This is the class for which the property is defined.
- Its *range* ("Ranges"): This is the associated datatype. It can be an integer or a real number, Boolean, character string, date, and so on. Please note: to work with Python and Owlready afterward, it is preferable to use the types integer for integer numbers and decimal for real numbers (refer to Table 4-1 for more information). Attention, the range of an OWL property has nothing to do with the Python `range()` function which allows you to create lists of numbers (see 2.6).
- Its *functional* status ("Functional" checkbox): When a property is functional, a given individual can have (at most) only one value for this property. On the contrary, if the property is not functional, a given individual can have several values.

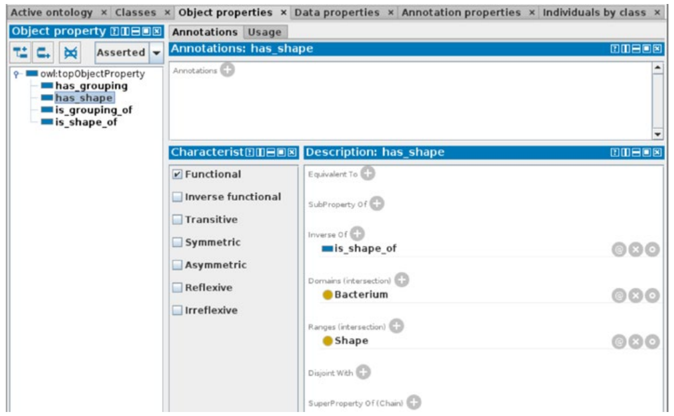
Domain and range are optional. It is possible to define several domains and ranges; however, it is the **intersection** of the different domains/ranges that is considered and not their union, which is often not the desired result. For example, consider the property "has_shape" and two classes, Bacteria and Viruses, of which individuals can have a shape. If we define two domains, Bacteria and Virus, only individuals belonging to *both* the Bacteria class and the Viruses class can have a shape! If one wants to say that all Viruses and all Bacteria may have a shape, it is necessary to define the domain as being the *union* of classes, that is to say, "Bacterium or Virus".

Here, we will configure our two data properties as follows:

- gram_positive: Functional (check the box), domain: Bacteria, range: Boolean
- nb_colonies: Functional (check the box), domain: Bacteria, range: integer

3.4.5 Object properties

In Protégé, the "Object Properties" tab allows you to create object properties. Using the and buttons, we create four new object properties called "has_shape", "has_grouping", "is_shape_of", and "is_grouping_of", as in the following screenshot:



Each object property can be configured by specifying:

- Its *domain* ("Domains (intersection)" in Protégé): This is the class for which the property is defined.

- Its *range* (“Ranges (intersection)”): This is the class of associated objects.

  As before, if several domains or ranges are indicated, it is their intersection that is considered.

- Its *inverse property* (“Inverse Of”): The inverse property corresponds to existing relationships when the property is read backward; if a property exists between A and B, then its inverse property exists between B and A. For example, the property “is_shape_of” is the inverse of “has_shape”: if a bacterium X has the shape A, then A is the shape of X. These inverse properties will be useful in Python to navigate using the relation has_shape/is_shape_of in both directions.
- Its *functional* status (“Functional” checkbox): When a property is functional, a given individual can have (at most) only one value for this property. On the contrary, if the property is not functional, a given individual can have several values.
- Its *inverse functional* status (“Inverse functional” checkbox): A property is inverse functional if the inverse property is functional. For example, the property is_father_of is inverse functional: a man A can be the father of several children B, C, D, and so on, but for each of these children, A is their only father.
- Its *transitive* status (“Transitive” checkbox): A property is transitive if it is possible to “chain” this property on several objects. For example, the property “is_larger_than” is transitive: if an individual A is larger than B and if B is himself larger than C, then we can deduce that A is larger than C.

- Its *symmetric* status (“Symmetric” checkbox): A property is symmetrical if it can be read indifferently in both directions (it is thus its own inverse). For example, the property “is_married_to” is symmetrical: if person A is married to person B, then B is married to A.
- Its *asymmetric* status (“Asymmetric” checkbox): A property is asymmetrical if it is never symmetrical. For example, the property “has_father” is asymmetric: if A has for father B, then it is not possible that B has for father A.
- Its *reflexive* status (“Reflexive” checkbox): A property is reflexive if it always applies between any object and itself. For example, the property “knows” is reflexive: each person X knows himself.
- Its *irreflexive* status (“Irreflexive” checkbox): A property is irreflexive if it is never reflexive. For example, the property “is_married_to” is irreflexive: one cannot be married to him/herself.

Here, we will configure our object properties as follows:

- has_shape: Functional (check the box), domain: Bacteria, range: Shape
- has_grouping: Nonfunctional (do not check the box), domain: Bacteria, range: Grouping
- is_shape_of: Nonfunctional, domain: Form, range: Bacterium, inverse: has_shape
- is_grouping of: Nonfunctional, domain: Grouping, range: Bacteria, inverse: has_grouping

Note that it is enough to define the inverse property of only one of the two properties of the couple: for example, here, we do not need to specify that has_shape has for inverse is_shape_of. This can be easily deduced from the inverse property of is_shape_of.

3.4.6 Restrictions

Now that we have created the properties, we can go back to the classes and add restrictions, based on these properties.

The restrictions are added in the "Classes" tab of Protégé, by clicking the "+" button to the right of "SubClass Of" in the "Description" section. "SubClass Of" allows you to add superclasses to the class; it can be an OWL named class created as before but also constructors, such as partitions (see 3.4.3) but also restrictions.

For example, the bacterium Pseudomonas has a Gram negative staining. This results in OWL by the following restriction: the Boolean property "gram_positive" must have the false value. This restriction is assimilated to a class: it is the class of bacteria having the false value for the "gram_positive" property. We can therefore define the Pseudomonas class as a subclass of this restriction class.

OWL offers several categories of restrictions. The following restrictions are used to model the relationships between two classes:

- Existential restriction (*some*): It represents the class of individuals who have at least one relation of a certain property with an individual belonging to a certain class.

  This restriction is written "property **some** class" in Protégé. For example, we have seen (Figure 3-1) that Pseudomonas all have a Rod shape. Rod is a class, which means that there might be several subtypes

  of the Rod shape (e.g., we could distinguish regular and irregular rod shapes). This restriction will therefore be written "has_shape some Rod".

- Cardinality restrictions (*exactly*, *min*, *max*): It represents the class of individuals who have a certain number of relations of a certain property with an individual belonging to a certain class. The number can be exact (*exactly*) or a minimum (*min*) or maximum (*max*) value.

  These restrictions are written "property **exactly** number class", "property **min** number class", or "property **max** number class" in Protégé. It is a more specific version of the existential restriction: an existential restriction is equivalent to a restriction of cardinality "min 1".

- Universal restriction (*only*): It represents the class of individuals who have only a relation of a certain property with one (or more) individuals belonging to a certain class (including its subclasses).

  This restriction is written "property **only** class" in Protégé. For example, the Pseudomonas is observed only with a Rod shape, which we will write "has_shape only Rod".

  Be careful not to confuse the universal restriction "has_shape only Rod" with the previous existential restriction, "has_shape some Rod". The existential restriction states that all Pseudomonas have at least one Rod shape, while the universal restriction states that all Pseudomonas have no other shape than

Rod. It is quite common to combine two similar restrictions, one universal and the other existential, with the same target class.

On the other hand, we will not use a universal restriction for grouping, because we have seen previously that bacteria can occasionally present other groups than their typical grouping.

The following restriction makes it possible to model a relation between a class and an individual or a datatype value:

- Value restriction (*value*, sometimes called *role-filler*): It represents the class of individuals who have a certain value for a certain property.

  This restriction is written "property **value** individual/datatype" in Protégé. For example, Pseudomonas is always associated with Gram negative staining. This restriction will be written "gram_positive value false".

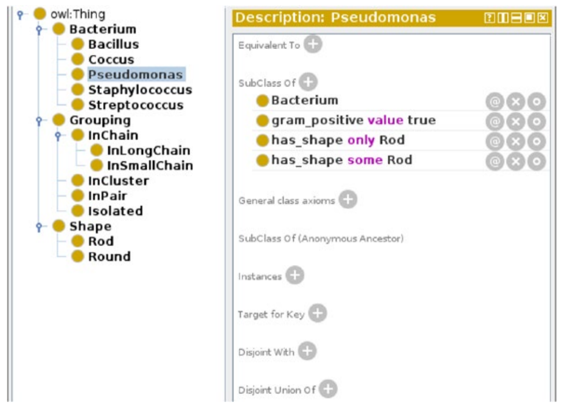
To add restrictions in Protégé, after clicking the "+" button, you can:

- Either manually enter the restriction in the "Class expression editor" tab (tip: the tabulation key allows you to complete a partial entry, e.g., "Bact" for "Bacteria"),
- Or use the "Object restriction creator" or "Data restriction creator" tab (depending on the type of property) and choose the values from the drop-down lists.

To further describe the Pseudomonas class, we will add the following restrictions:

- "has_shape some Rod"
- "has_shape only Rod"
- "gram_positive value false"[1]

Note that we have used an existential and a universal restriction for the shape, since Rod is a class and not an individual or a data, and on the contrary a value restriction for the Gram coloring, because false is a datatype value.



Bacterium
gram_positive value true
has_shape only Rod
has_shape some Rod

[1]Attention, in OWL, false and true are written without capital letters, while in Python they are written with (i.e., False and True; see 2.4.2).

3.4.7 Union, intersection, and complement

OWL also allows the use of logical operators as constructors. These operators have different names depending on whether they are considered from a logical point of view or from a set theory point of view; however, it is indeed the same thing. Three operators are available:

- **Logical AND** or intersection: These are individuals belonging to several classes at the same time.

  The intersection is written “class1 **and** class2” in Protégé. Of course, more classes can be included in the intersection, for example, “class1 **and** class2 **and** class3”.

- **Logical OR** or union: These are individuals belonging to a class among several.

  The union is written “class1 **or** class2” in Protégé. Similarly, unions are not limited to two classes, for example, “class1 **or** class2 **or** class3”. For example, the Pseudomonas can have two groupings: Isolated and InPair. We can therefore build the union of these two classes, which will be written “Isolated or InPair”.

  Furthermore, we have already used the union previously, to express the partitions (see 3.4.3).

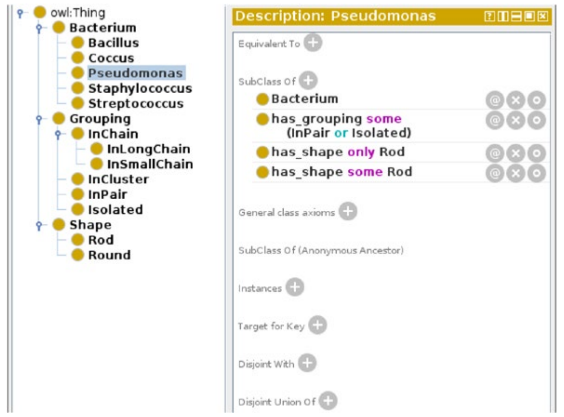
- **Logical NOT** or complement: These are individuals who do not belong to a given class. The complement is written “**not** class” in Protégé.

OWL also allows you to combine logical operators with restrictions and classes, by grouping the different elements in parentheses.

In order to refine the Pseudomonas class, we will add the following superclass:

- “has_grouping some (Isolated or InPair)”

This restriction states that all Pseudomonas have at least one Isolated or InPair grouping.



```
owl:Thing
  Bacterium
    Bacillus
    Coccus
    Pseudomonas
    Staphylococcus
    Streptococcus
  Grouping
    InChain
      InLongChain
      InSmallChain
    InCluster
    InPair
    Isolated
  Shape
    Rod
    Round
```

Description: Pseudomonas

Equivalent To

SubClass Of
- Bacterium
- has_grouping some (InPair or Isolated)
- has_shape only Rod
- has_shape some Rod

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With

Disjoint Union Of

3.4.8 Definitions (equivalent-to relations)

In the previous two sections, we used restrictions and constructors to describe the properties of the class. However, this is not a definition in the formal sense, because we have not fully and uniquely described the class. For example, all Pseudomonas have a Rod shape, but not all bacteria with a Rod shape are Pseudomonas!

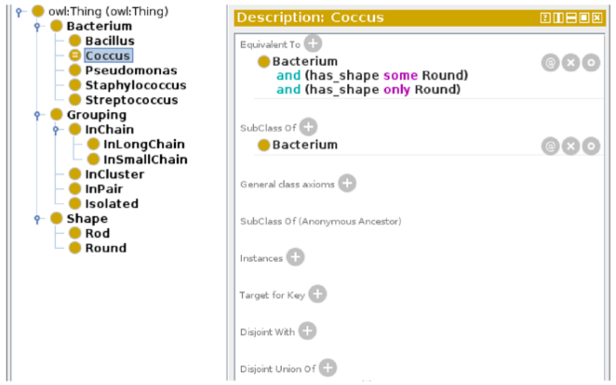
OWL allows you to give a class a formal equivalence definition, *via* an equivalence relation. Then, the defined classes allow reclassifying individuals during automatic reasoning (which we will see later in section 3.5 and in Chapter 7).

For example, the Coccus class is the class of bacteria with a Round shape (i.e., at least one Round shape and only Round shape). We can therefore define it as follows:

- Coccus: "Bacterium and (has_shape some Round)
  and (has_shape only Round)"

Note that, unlike the restrictions and constructors that we used previously as a superclass for Pseudomonas, equivalences must be defined "in one piece". We cannot divide the definition into three parts "Bacteria", "has_shape some Round", and "has_shape only Round" unless we change its meaning entirely!

To add the restriction in Protégé, click the "+" button to the right of "Equivalent To", then manually enter the restriction in the "Class expression editor" tab (again, you can use the tabulation key for completion).



Protégé marks the defined classes with a different icon: a brown circle in which appears the symbol "≡" which means "equivalent to" in description logics.

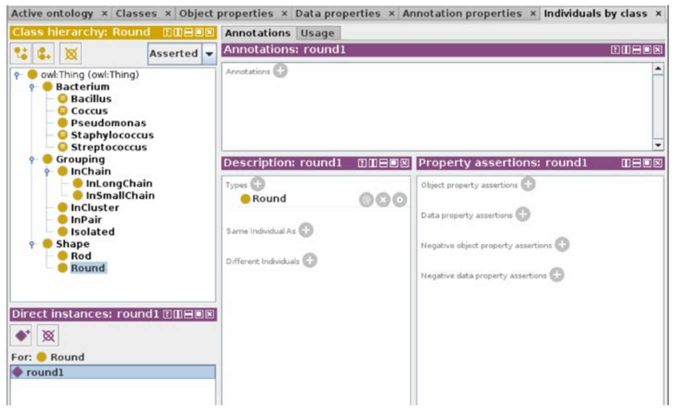
Similarly, we will define the Bacillus, Staphylococcus, and Streptococcus classes as follows:

- Bacillus: "Bacterium and (has_shape some Rod)
  and (has_shape only Rod)"
- Staphylococcus: "Bacterium and (has_shape some Round)
  and (has_shape only Round)
  and (has_grouping some InCluster)
  and (gram_positive value true)"
- Streptococcus: "Bacterium and (has_shape some Round)
  and (has_shape only Round)
  and (has_grouping some InSmallChain)
  and (has_grouping only (not Isolated))
  and (gram_positive value true)"

For Streptococcus, the restriction "has_grouping only (not Isolated)" indicates that Streptococcus can only have groupings other than Isolated: it is never observed isolated.

3.4.9 Individuals

The "Individuals" tab of Protégé allows you to navigate through individuals and create new ones. In order to test our ontology, we will create a few individuals. To do this, select the class in the "Class hierarchy" panel and then click the ◆ button in the "Members list" panel (this panel lists the individuals belonging to the class). We will first select the Round class and create a shape that we will call "round1", as in the following screenshot:



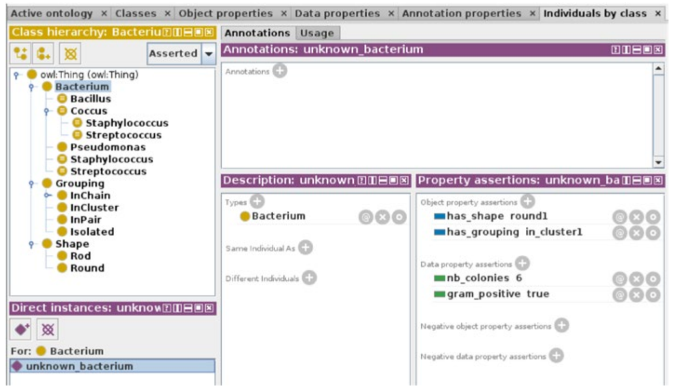
In the same way, we create an individual called "in_cluster1", belonging to the class "InCluster".

Then, we create an individual called "unknown_bacterium", belonging to the "Bacterium" class. Finally, in the "Property assertions" panel, we enter the relationships of this individual by clicking the "+" buttons to the right of "Object property assertions" and "Data property assertions". We will enter the following relationships:

- Object property:
  - has_shape: round1
  - has_grouping: in_cluster1

- Data property:
  - gram_positive: true
  - nb_colonies: 6

The following screenshot shows the expected result:



3.4.10 Other constructs

OWL and Protégé also offer other constructors, of less frequent use.

- The **set of individuals** (also called *one of*) allows creating a class restricted to a set of individuals. It is written between braces: "{individual1, individual2, ...}". It can also be used to transform an individual into a class (also called the *singleton* class because it has only one instance/individual), as follows: "{individual}".

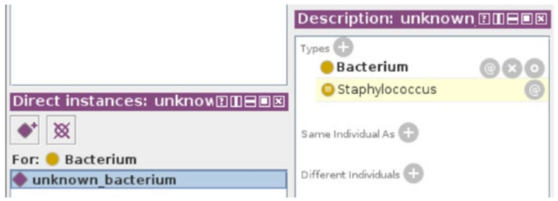
- The **inverse** of a property is written “inverse (property)”. For example, “inverse (has_shape)” is equivalent to “is_shape_of” in our ontology of bacteria. This constructor is especially useful when the ontology does not define named inverse properties.
- A property chain is written “property1 o property2” (the circle corresponds to the lowercase letter “o”). They are also called the property *composition*. They make it possible to “chain” several properties, for example, “is_shape_of o has_grouping” to pass directly from a shape to the groupings of bacteria having this shape.

3.5 Automatic reasoning

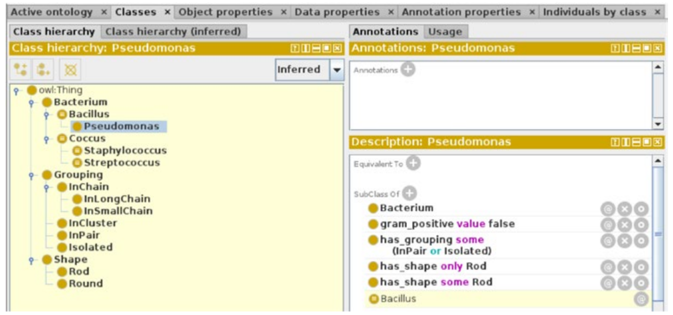
Now our bacteria ontology is ready!

To verify the absence of inconsistency in the ontology and test the automatic reasoning, we can use the “Reasoner ➤ Start reasoner” menu which will execute the automatic reasoner. Several reasoners are available; I recommend the use of HermiT.

Once the reasoning has been carried out, individuals are reclassified in Protégé. For example, the individual “unknown_bacterium” that we had created was of the class Bacteria. We can see that it has been reclassified into a new class: Staphylococcus (the new classes appear on a yellow background in Protégé). Indeed, this bacterium satisfies the conditions to be a Staphylococcus (round shape, grouped in clusters, Gram + status).



In addition, the reasoner also reorganized the classes. To observe this, we will return to the Classes tab and click “Class hierarchy (inferred)”. The class tree has been changed. We can see, for example, that the Pseudomonas class has been reclassified as a subclass of the Bacillus class. Indeed, all the individuals of this class satisfy the definition of the Bacillus class, since Pseudomonas all have a Rod shape.



You may also try the following two experiences:

1. Create an individual of the Bacterium class, with a Rod shape, grouped in pairs and/or isolated, and a Gram negative status. This individual will be reclassified in the Bacillus class, but not in Pseudomonas. Indeed, we have not given a formal

definition of the Pseudomonas class; the reasoner cannot therefore deduce that such a bacterium is a Pseudomonas. The absence of definition was a desired choice when designing the ontology, because Pseudomonas are not the only bacteria with a rod shape, isolated or in pairs, and Gram negative (see Figure 3-1).

2. Create an individual of the Bacterium class, with a round shape, grouped in small chain, and having a Gram positive status. This individual will be reclassified in the Coccus class, but not in Streptococcus. However, this class does contain a definition! However, the individual we just created does not fully meet the definition of the Streptococcus class.

   In fact, in the definition, we indicated "has_grouping only (not Isolated)". In the individual, we indicated an InSmallChain grouping; however, the property "has_grouping" is not functional, and therefore several values are possible. The Open-World assumption implies that the reasoner cannot exclude the existence of another grouping, not mentioned in the ontology, which could be Isolated.

   Consequently, to be able to deduce that our individual is a Streptococcus, it would be necessary to indicate in the ontology that the individual has no other groupings than those explicitly mentioned or that he has no grouping of the class Isolated.

   On the other hand, in the formal definitions, we also used universal constraints ("only") on the "has_shape" property. However, this does not prevent the classification of individuals in the Coccus, Bacillus, and Staphylococcus classes. Why? Because the property is functional and the Round and Rod classes are disjoint. Therefore, when a bacterium has a Rod shape, it is impossible for it to have a Round shape, and vice versa. On the contrary, the property "has_grouping" is not functional, and therefore this reasoning is no longer possible.

   We will come back to this problem, and the solution will be provided in 7.3.

3.6 Modeling exercises

Here are some exercises to train you in ontology modeling:

1. In the bacteria ontology, add an individual of the Staphylococcus class having a rod shape. Run the reasoner; what do you observe?
2. Using the Protégé editor, extend the ontology of bacteria by adding the catalase test. This biological test helps to identify bacteria, and its result can be positive or negative. The catalase test is positive for Staphylococci and Pseudomonas, negative for Streptococci.
3. Using the Protégé editor, extend the bacteria ontology by adding the color of the bacteria. Staphylococci are white or golden (this is the

famous *Staphylococcus aureus*), Streptococci are translucent, and Pseudomonas are generally colored (that is to say, not white).

4. Using the Protégé editor, add a new class of bacteria: *Mycobacterium leprae* (Hansen's bacillus, which causes leprosy). This species of bacteria is Gram positive, rod shape, and isolated or grouped in pairs. The catalase test is not relevant for this bacterium because it is very difficult to grow in vitro. The color is yellow. Finally, all of these characteristics are sufficient to identify the bacteria.
5. In the Protégé editor, add an individual of the class Bacterium, rod shape, isolated, and yellow in color. Check that this individual is properly classified as *Mycobacterium leprae*.
6. In the ontology of bacteria, add a disjunction between the different subclasses of Bacteria (Staphylococci, Streptococci, Pseudomonas, etc.). Does this change the result of the reasoning on the unknown bacterium?
7. An OWL ontology was carried out to structure the drug interactions. This ontology is intended to automatically detect interactions within prescriptions prescribed by doctors, using a reasoner. Could the Open-World assumption pose a problem during the reasoning?
8. Using the Protégé editor, build an ontology describing the books, the authors, and the editors. You take inspiration from the object model presented in 2.9.