OWL Biomedical Ontology Tutorial (Protégé/OWL Version 3.0)

ALR

February 2005

1. Introduction

1.1 Goals and plan of tutorial

This tutorial is intended to take you through the basics of building an ontology in a biomedical area using the Protégé-OWL tools in the style suggested by the GALEN experience.

This tutorial assumes that you have already done at least the first part of the Protégé-OWL "Pizza Tutorial" [1]
http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf or one of its variants.

One of the hardest parts of building any ontology is getting started and choosing appropriate high level concepts. The tutorial starts with a ready built high level ontology. You can ignore the details, but the outline is consistent with what we recommend. The ontology provides medical concepts down to *Organs*, *OrganPart* and *Disorder*. It provides a few biological concepts including the notions of *Cell*, *CellularStructure*, *CellularProcess*, and *MembraneTransport*.

The single organ *Lung* is supplied along with the *MicroOrganism* categories *Bacterium*, *Virus*, and *Pneumococcus.*

Overall the goal of the first phase of the exercise is to construct representations for the notions of "Pneumococal Pneumonia" and "Enzyme for membrane transport".

The approach of the tutorial is first to work through the recommended solution of each issue and then to demonstrate the problems which occur if any of several alternatives are used. For some things, a simple version is given first and then a more sophisticated version later in the tutorial.

1.2 What is OWL?

1.2.1 What is it?

The language which has been known in various revisions as OIL, DAML+OIL, and now OWL is standard knowledge representation language of the Semantic Web community developed by W3C (the organisation that manages the web standards[1]). OWL is based description logics as a the next layer on top of node-and-arc style representations and a layer on top of RDFS. OWL, particularly in the Protégé-OWL editor, looks somewhat like a frame language. However, it has a formal logical semantics. There are classifiers available to check whether concepts and knowledge bases are consistent and to infer classification automatically. The concrete syntax underneath is in RDF/XML and not very useful directly. (In fact it is hideously ugly and to be avoided.) There is an abstract syntax and Protégé OWL provides several variant syntaxes. We strongly recommend that one of these be used in writing.

(For more detail see http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf)

1.2.2 What is in it?

The following should be familiar from the Protégé-OWL tutorial, but as a review, the basic constructs in OWL are:

- *Classes* – known in other systems as "concepts", "categories", or "types", *e.g.* "Person", "Diabetes", "Fracture of neck of left femur", etc. Classes come in two kinds
  - *Primitive classes* – classes for concepts which have no complete definition although they may be described and placed in a hierarchy
  - *Defined classes* – which are defined from other classes using the various operators in the language.

[1] see http://www.w3c.org/

- *Properties* – known in other systems as "slots", "relations", "attributes", or "roles"
- *Restrictions* – known in other systems as "filled slots", "statements", "relationships", "relation instances", or "criteria", or (confusingly) "properties – links between classes by means of properties. A quantified Class-property-class triple.
- *Axioms* – which provide additional information about classes

1.2.3 What can I do with it?

OWL allows the expression of an 'ontology' or logical model of a set of concepts (entities) and the relations amongst them in such a way that they can be tested for consistency and classified automatically. The logic used is a subset of first order logic that has been selected to be computationally tractable. (Proof in full first order logic is undecidable.)

1.2.4 If this isn't clear

Don't worry. The purpose of this tutorial is to demonstrate what we mean.

1.3 Notation and Conventions for this tutorial

In this tutorial the following conventions are used

- Phrases in English for concepts to be represented or English text versions of definitions are presented between double quotes "Enzyme for membrane transport"
- Things that appear on the screen are given in a bold sans-serif font like this: **TutorialTop-01**
- Classes (aka 'concepts') and property names (aka 'slot names', 'semantic links', 'roles') are written in 'camel back notation', *e.g.* **CellularStructure**, **hasLocus,** etc.[2]
- Class names always begin with an uppercase letter. Property names always begin with a lower case letter. It is a standard convention in English that *Classes* are always named with singular nouns[3].
- Technical terms are enclosed in single 'scare quotes' like this.
- Where there is a need to refer to a class or other ontological notion in the abstract rather than on the screen it is printed like this, *e.g. CellularStructure*

So given these conventions, the 'formal representation' of "pneumococcal pneumonia" is *PneumocccocalPneumonia*. which appears on the screen as **PneumococcalPneumonia.**

Note that OWL is case sensitive. "Pneumococcalpneumonia", pneumococcalPneumonia, and PneumococcalPneumonia are all different. OWL identifiers must begin with a letter, contain only letters, numerals, and the underscore character ('_'). They may not contain spaces.

Protégé OWL uses some unfamiliar symbols from logic and the "German notation" for description logics. Not all of these symbols are found in the standard Microsoft fonts. A list of the symbols and their interpretation is given below. If they do not print out correctly – even though we have tried to embed them in this document – then install the Zed font which can be downloaded from http://www.cs.kent.ac.uk/people/staff/rej/Zedfont/latest/. (or see the .pdf version of this document.)

We also give a list of paraphrases of OWL constructs and the link with the official OWL in the appendix.

[2] Note also that there is an ongoing discussion about whether "camel back" notation as used here or words separated by underlines is preferable. The custom is moving towards using underlines, *e.g.* "Pneumococcal_pneumonia" instead of "PneumococcalPneumonia". However, earlier versions of this tutorial used camel back because it avoided confusion when using the older OIL syntax. For now shall stick with that although OWL users tend to prefer the use of underlines. It makes no difference, provided one or the other is used consistently.

[3] In the tutorial I have preferred consistency with the actual class name over normal English usage. This occasionally leads to odd expressions such as "**Bacterium**s" when the formal name needs to be pluralised in text.

Symbol	Meaning	OWL
∃	"some"	*someValuesFrom*
∃ hasCause X	"hasCause SOME X"	*hasCause someValuesFrom(X)*
∀	"only"	*allValuesFrom*
∀ hasCause X	"hasCause ONLY X"	*hasCause allValuesFrom(X)*
⊓	"and"	*intersectionOf*
⊔	"or"	*unionOf*
≡	"are necessary & sufficient conditions for"	*equivalentClass*
⊑	"necessarily implies"	subclassOf

Note the shift in position between the ∃ and ∀ symbols and the "SOME" and "ONLY" that they represent. The qualifiers refer to the values (fillers).

1.4 Mechanics

The initial file is found in **TutorialTop-01.owl.**

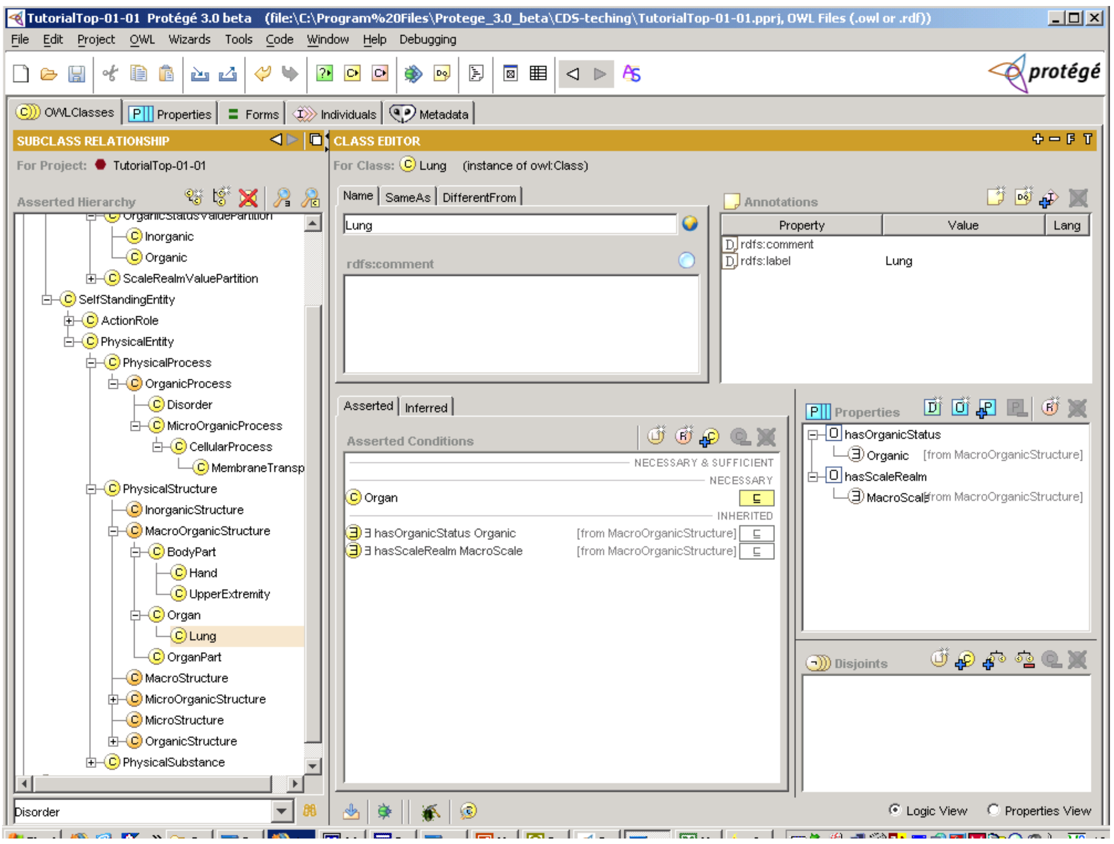
Steps to set up tutorial:

Initial Startup

- Start the **Racer** from the start menu. Where this is will depend on your machine. If in doubt look in C:\Program Files\Racer or C:\Program files\ Protege. Starting **Racer** will cause a black command screens to appear**.** You can minimise this window.
- Start **Protege** which should bring up a dialogue as indicated in the Protégé-OWL tutorial. When the dialogue appears, click **Build**, select **OWL files (.owl or .rdf)**, and navigate to **TutorialTop-01.** (You may have to download the tutorials files in which case they will be in athttp://www.cs.man.ac.uk/~rector/teaching/modules/cds/clinical-tutorial/
- Select **Save as** from the **File** menu and save the file immediately as **MyTutorial-01-01**—or any other name you choose ending in **01-01**. (*Save early! Save often! Always save to a new version number.* Protégé-OWL is not yet completely stable. Undo is difficult and bugs occasionally corrupt ontologies beyond retrieval. Save or be sorry!)
- Find **Lung** and **Disorder** (To find a concept, you can type it in the subpane at the bottom of the hierarchy window.)

The screen should now look approximately as shown below and you are ready to start. (You can adjust the exact size and position of the panes to suit in the usual way.)

(At this point you may want to browse around the hierarchy.)



2. Create the entity representing "Pneumonia"

2.1 Simple solution

Of the options given, it seems natural to consider representing "Pneumonia" as a *Disorder*. The simple solution is just to tell OWL that *Pneumonia* is a kind of disorder and then to say describe it.

First create it:

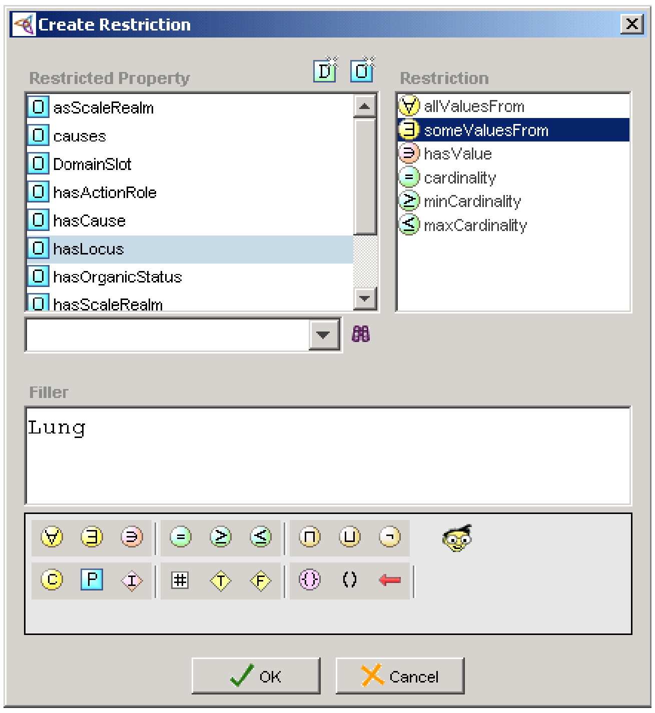
- Select **Disorder** and create a new subclass (either from the right mouse button menu or from the icon at the top of the **Asserted Hierarchy**)
- In the **Class Name** area, enter **Pneumonia.**
- **Pneumonia** will be added to the **Classes** hierarchy under **Disorder**.
- The **rdfs:comment** pane is blank. Enter something like "First attempt at defining *Pneumonia* simply" to remind you what you are doing and help anybody who comes along understand what they are seeing.

So far, this means that "*All* pneumonias *are also* disorders" or "*Pneumonia* is a kind of *Disorder*"

Then describe it:

What can we say about *Pneumonia?* Most obviously that it occurs in the lungs. To do this:

- In the **Asserted Conditions** pane, highlight **NECESSARY** and click the circled **R** icon
- When the **Create Restriction** pop-up appears choose **hasLocus** as the property in the upper left hand pane. (*hasLocus* is the 'property which links Disorders with anatomy.) Leave the qualifier set to **someValuesFrom.** Type Lung in the value area – or click the circled "c" icon in the icon tray and navigate to **Lung** in the pop-up.



What it means:

The restriction that appears will can be paraphrased as follows:

∃ **hasLocus Lung** "hasLocus SOME Lung"

Reminder: the prefix "∃" really indicates an infix operator "SOME". It is the **Lung** that exists, not the **Pneumonia**. (This is an oddity inherited from DL notation.)

Making **Pneumonia** a subclass of **Disorder** and adding the restriction as shown means:

"*All* pneumonias are located in *some* lung" **(1.)**
"*All* pneumonias *are also* disorders" **(2.)**

There are several other ways of saying "*All* pneumonias *are also* disorders":

"Pneumonia *is a kind of* disorder" **(3.)**
"*If* something is a pneumonia *then* it is a disorder" **(4.)**
"Being a pneumonia *implies* being a disorder" **(5.)**

"*Pneumonia is a subclass of Disorder*" ***(6.)***
"Disorder subsumes Pneumonia" ***(7.)***
Pneumonia → Disorder ***(8.)***

Being a kind of, or subclass of, something has a specific and very strong meaning in OWL:

"Everything that is a member of the subclass, without exception, is a member of the superclass" or

"Being a member of the subclass implies being a member of the superclass".

This strong logical meaning of "subclass" has two consequences:

- There can be no exceptions
- It is possible to prove things logically, including in many cases, whether one thing is a subclass of another – this is the key power of OWL.

2.2 Kinds of pneumonia – Make "Viral pneumonia" and "Bacterial pneumonia", "Pneumococcal pneumonia", and "Mixed Pneumonia"

2.2.1 Bacterial, Viral, and Pneumococcal Pneumonia – 'SubclassOf' and 'SameClassAs' – 'Primitives' and 'Definitions'

(This is a good point to save your work by selecting Save and then create a new file to continue with the next phase by doing a **Save as** to a new file **Mytutorial-02-01.** Save early; save often; Save to a new version.)

One way of classifying pneumonias is by their cause. Pneumonia can be caused either ‘viral’ or ‘bacterial’, *i.e* caused either by a virus or a bacterium or sometimes both.

The simplest way to represent “bacterial pneumonia”, is simply to define a subclass of *Pneumonia.*

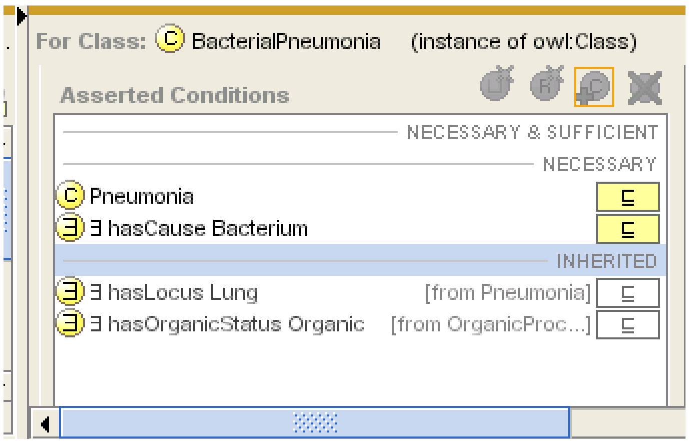
First create it:

- Select **Pneumonia**.
- Click the add-subclasses icon, OR use the right mouse button and select **Add subclass**
- Name the new subclass **Bacterial Pneumonia**

Then describe it:

- Add a restriction to say that it is caused by bacteria
  - Select **NECESSARY** in the **Asserted Conditions** pane.
  - Select **hasCause**, the name of the causal property, in the left hand pane of the pop-up
  - Type **Bacterium –** (note that the tab or ctrl-space will bring up a completer)
  - Click **OK**

The **Asserted Conditions** pane should now look like:



For Class: BacterialPneumonia (instance of owl:Class)

Asserted Conditions

NECESSARY & SUFFICIENT

NECESSARY

Pneumonia ⊑
∃ hasCause Bacterium ⊑

INHERITED

∃ hasLocus Lung [from Pneumonia] ⊑
∃ hasOrganicStatus Organic [from OrganicProc...] ⊑

The paraphrase for the new information is:

Pneumonia	“isKindOf Pneumonia”
∃ **hasCause Bacterium**	“hasCause SOME Bacterium”

What it means:

“*All* bacterial pneumonias are pneumonias”
“*All* bacterial pneumonia is caused by *some* bacteria” **(9.)**

Represented in simple logic notation as:

BacterialPneumonia → **Pneumonia** **(10.)**
BacterialPneumonia → **causedBy** some **Bacterium** **(11.)**

Or in German notation

BacterialPneumonia ⊑ **Pneumonia** **(12.)**
BacterialPneumonia ⊑ **causedBy** some **Bacterium** **(13.)**

2.2.2 Make it a *definition*:

Both statements above are undoubtedly true, but this is not enough to let the system recognise and classify other things, *e.g.* “pneumococcal pneumonia”, as kinds of “bacterial pneumonia”.

To allow the system to recognise classes as being kinds of “bacterial pneumonia” we need to change the class from being a *Primitive* to being *Defined.*

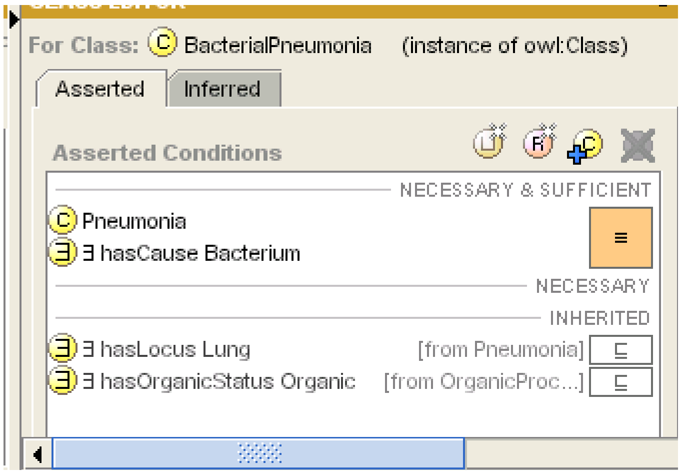
To say that a class is *Defined* is to say that it has at least one set of “necessary and sufficient” conditions, *i.e.* a set of conditions that taken together are sufficient to recognise the class.

- Drag and drop **Pneumonia** and ∃ **hasCause Bacterium** from the **NECESSARY** to the **NECESSARY & SUFFICIENT** subpanes

OR

- Select the class **BacterialPneumonia** and from the right mouse button menu select **Convert to defined class**.

The screen should now look like:



For Class: BacterialPneumonia (instance of owl:Class)

Asserted | Inferred

Asserted Conditions

NECESSARY & SUFFICIENT

Pneumonia
∃ hasCause Bacterium ≡

NECESSARY

INHERITED

∃ hasLocus Lung [from Pneumonia] ⊑
∃ hasOrganicStatus Organic [from OrganicProc...] ⊑

Note that in variant of standard description logic notation used in Protégé-OWL, the symbol "⊑" means "necessarily implies", and "≡"for means "if and only if" or "necessary and sufficient".

What it now means:

"*Any* pneumonia caused by a bacterium is a bacterial pneumonia" **(14.)**

or more in other words

"Something is a bacterial pneumonia *if and only if* it is a pneumonia and caused by a bacterium" **(15.)**

represented in logic notation as

BacterialPneumonia ↔ **Pneumonia & causedBy** some **Bacterium** *(***16.***)*

By putting the conditions in the **NECESSARY & SUFFICIENT** subpane, it indicates that the implication goes both ways as in (16). If they are the **NECESSARY** subpane, then the implication goes only one way as in (10)- (13).

Why the difference between *Defined* and *Primitive* classes matters (NECESSARY & SUFFICENT vs NECESSARY):

In order to demonstrate how important this distinction between 'defined' and 'primitive' classes is, perform the following experiment in which we will first make a class for "pneumococal pneumonia" and then see how the classifier treats the two classes depending on whether the class for "bacterial pneumonia" is defined or primitive.

Make **PneumococcalPneumona** as you made **BacterialPneumonia**

First create it:

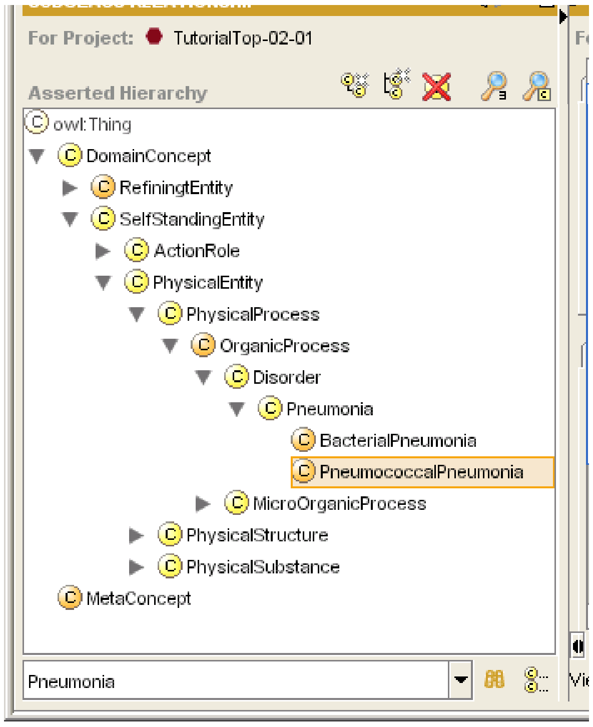
- Create a subclass of **Pneumonia** called **PneumococcalPneumonia**

Then describe it:

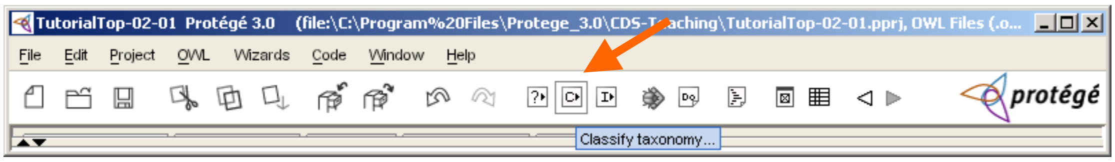
- Add a restriction to say that it is caused by bacteria
  - Under **Asserted Conditions** select **NECESSARY & SUFFICIENT** and then click the circled R icon
  - Select **hasCause**, the name of the causal property
  - Select **Pneumococcus**

Classify it to see the results:

- The **Asserted Hierarchy** window should now be expanded to look as shown below. This is the hierarchy *before* the Classifier has performed automatic classification. Note that **BacterialPneumonia** and **PneumococcalPneumonia** are siblings.

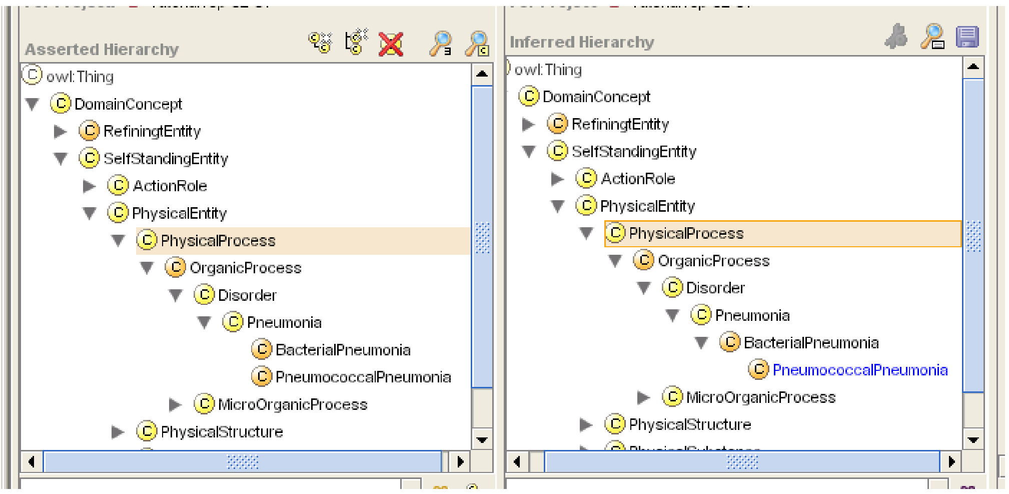


- Click classify icon as shown below (at end of red arrow).
- When the pop-up finishes, click **Close**


TutorialTop-02-01 Protégé 3.0 (file:\C:\Program%20Files\Protege_3.0\CDS-Teaching\TutorialTop-02-01.pprj, OWL Files (.o...
File
Edit
Project
OWL
Wizards
Code
Window
Help
protégé
Classify taxonomy...

A pane for "Inferred Hierarchy" will appear.

- Click on anything else and then back on **PneumococcalPneumonia** to synchornise the views. The two pans should now look as shown below. The blue colour of the PneumococcalPneumonia indicates that it has been moved by the classifier. (The move will also show up in the list of changes in the bottom pane.) .

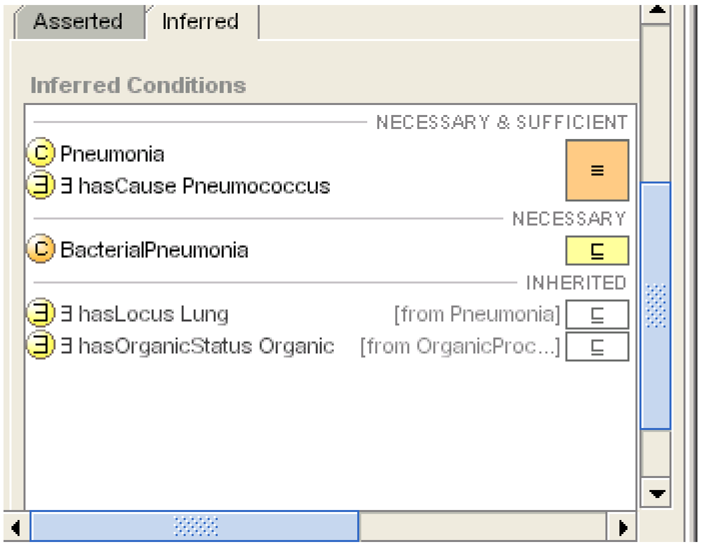


2.2.3 'Inferred conditions"

Click on the **Inferred Conditions** tab to see that PneumococcalPneumonia has now been inferred to have a necessary condition of being a BacterialPneumonia.

2.2.4 Inherited conditions

Notice the **INHERITED** conditions it has also inherited having the locus **Lung** and the status **Organic**. (The significance of **Organic** will be explained later.)



2.2.5 What happens if you forget to make it a definition?

A common error in OWL is to make conditions only **NECESSARY** rather than **NECESSARY & SUFFICIENT** – *i.e.* to leave a class as primitive when it should be defined. To see the effect of this mistake, select BacterialPneumonia, and on the right menu click **Convert to primitive class**. The **NECESSARY & SUFFICIENT** conditions will be moved down to the **NECESSARY** subpane. Now run the classifier again. Note that **PneumococcalPneumonia** is no longer classified under **BacterialPneumonia**

Nothing which is not Defined will ever have anything else classified underneath it by the classifier!

(In the absence of axioms or domain/range constraints, to be covered later)

The classifier can classify primitives under new definitions, but it cannot classify one primitive beneath another. That is why they are called "Primitive". That's the meaning of the difference between the single and double arrow in (14-16) above.

In OWL, to say that something is "primitive" is to say that we cannot (or do not choose to) give it a complete definition – sufficient as well as necessary but only choose to give it a 'description' by which it can be classified under other defined classes.

Put the knowledge base back to the correct format.

- Select BacterialPneumonia and use the right mouse button menu item **Convert to defined class** to reset the ontology to the correct form.

2.2.6 Mixed Pneumonia: Multiple values and the meaning of *someValuesFrom* ("some" / ∃ / existential restrictions)

(Be sure you have performed the last correction so that the knowledge base is correct)

Some Pneumonia has a mixture of viral and bacterial causes.

First define ViralPneumonia

Follow the same procedure to make **ViralPneumonia** as to make **BacterialPneumonia** only this time make the cause in the restriction **Virus.** Be sure you make the conditions necessary & sufficient. (An easy way to do this is to use the **Create clone** item on the right mouse button for the class and then edit the result.)

ViralPneumonia should now has the two necessary and sufficient conditions, with their paraphrases

Pneumonia	"isKindOf Pneumonia"
∃ **hasCause Virus**	"hasCause SOME virus"

Then define MixedPneumonia

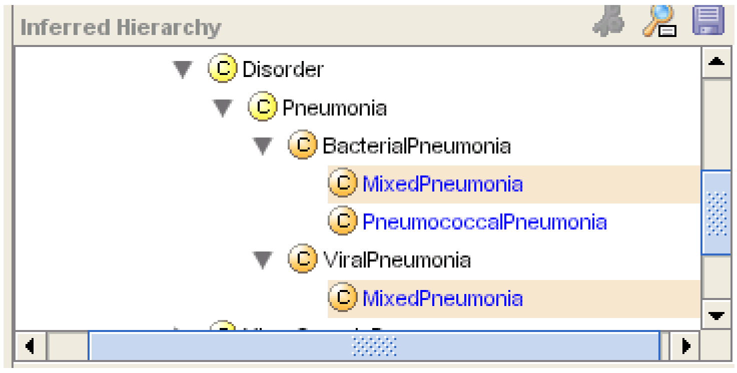
Define a new subclass of Pneumonia as **MixedPneumonia**

Pneumonia	"isKindOf Pneumonia"
∃ **hasCause Virus**	"hasCause SOME Virus"
∃ **hasCause Bactium**	"hasCause SOME Bacterium"

Classify it

Press the classify button and then when classification is finished, reselect **MixedPneumonia.**

MixedPneumonia will appear in the **Inferred Hierarchy** window as a subclass of both **BacterialPneumonia** and **ViralPneumonia**.



This is correct because t the restriction

∃ **hasCause Bacterium** "hasCause some Bacterium" **(17.)**

appears in both definitions. It means

"has *some* bacterium as its cause"

Nothing we have said up to this point prevents diseases from having more than one cause. The restriction does not indicate *the* cause, but *a* cause, possibly amongst many.

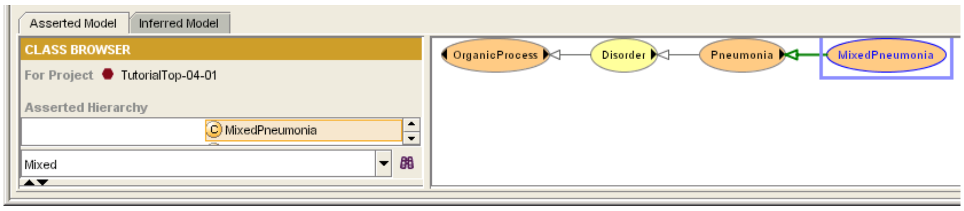
(There is a way to indicate that a property can only have one value, which is dealt with later)

2.2.7 Visualising it with OwlViz

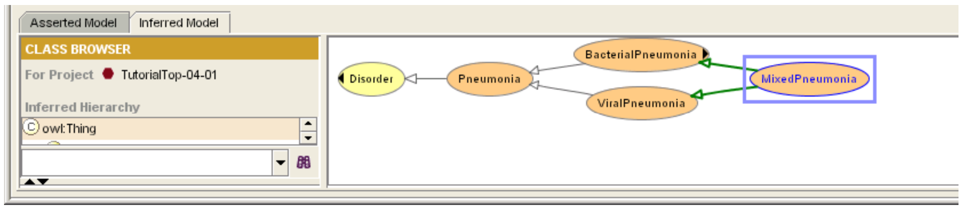
This is also a good point to demonstrate the use of OwlViz.

- From the **Project** menu select **Configure**
- Tick **OwlVizTab** and click **OK**.

- Select the **OwlViz** tab that has appeared at the right of the tabs.
- In the left-hand upper pane select **MixedPneumonia**. (You may have to drag down the bottom border of the pane to see it.)
- Click the **C** (class) Icon in the tab's icon row.
- Accept the defaults
- What you now see is the structure before classification



- Click the **Inferred Model** subtab. What you see now is the results of the classification showing **MixedPneumonia** with two parents.



Save at this point

This is another good point to save your work, and then create a new file to go on by doing a **Save as** to **MyTutorial-03-01**.

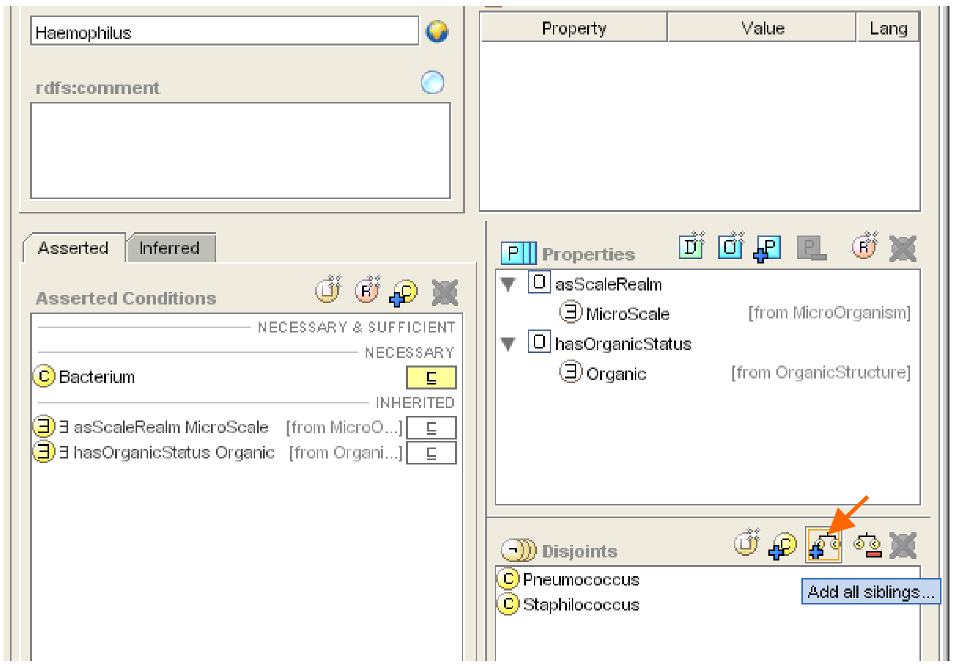
3. Creating new kinds of "Pneumonia": 'Self-standing concepts', 'Modifiers' and 'Value partitions"

3.1 Extending the causes of pneumonia: Disjointness axioms

At this stage there is only one bacterium, "pneumococcus". Obviously we would eventually want this to be a long list. For the time being we shall add just "staphylococcus" and "haemophilus" as primitive subclasses of **Bacterium**

Manually:

- Select **Bacterium** in the **Classes** pane
- Choose **add new subclass** from the right mouse button menu
- Name the subclass **Staphylococcus**
- Repeat for **Haemophilus**
- Double click on **Bacterium** and check the hierarchy to make sure you got it right
- Make the different kinds of bacteria different ("disjoint"). With any of the new classes selected , in the disjoint subpane at the lower right, click the all disjoint button as shown below, and click OK to the pop-up default **Mutually between all siblings.**



The screen should look as above (except for the arrow indicating the correct icon.)

Also note, this list of subclasses for **Bacterium** is not complete – the subclasses do not 'cover' the parent class Bacterium. This means that this list does not necessarily include all possible bacteria. This is in general true of 'Self-standing Concepts' – *i.e.* the things in the world like bacteria, people, bridges, diseases, etc. which have meaning on their own – in contrast to 'modifiers' such as "severe" which only take their meaning in combination with the thing modified.

Summary

For self-standing concepts, the primitive subclasses of each primitive class should be disjoint but do not cover the parent class.

3.2 Representing "Severe pneumonia": Properties, Value partitions and subclass covering axioms

We want to represent the notion of a modifier or refining value for "pneumonia" to be able to say "Severe pneumonia", "Mild pneumonia", "Moderately severe pneumonia" etc. This will take several steps:

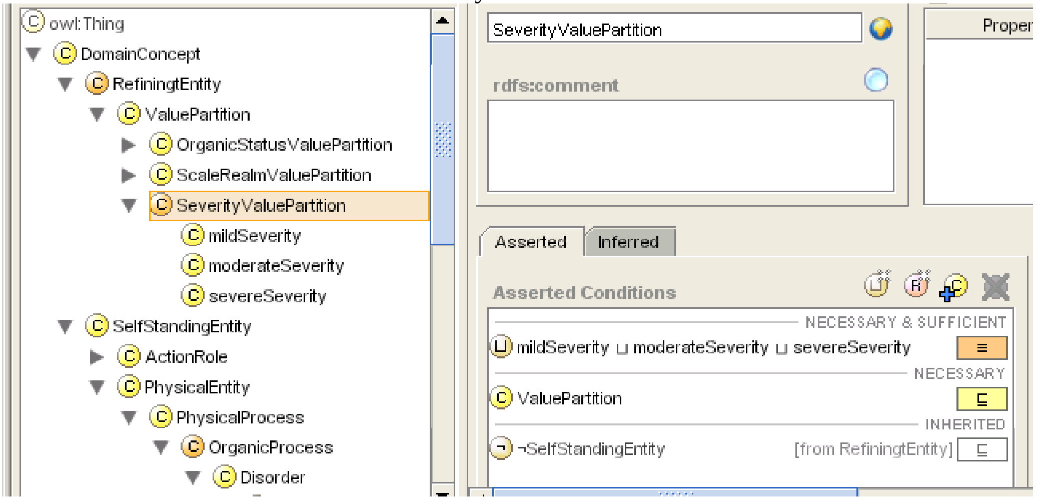
- Step 1: Create a new **ValuePartition**, **SeverityValuePartition;** create disjoint subclasses for "mild", "medium" and "severe", **mildSeverity**, **mediumSeverity**, **severeSeverity**. We shall be pedantic in the naming because "mild" might go with other value partitions. All of the subclasses need to be disjoint, and should "cover" the parent SeverityValuePartition, *i.e.* the class **SeverityValuePartition** is equal to **mildSeverity OR mediumSeverity OR severeSeverity**
- Step 2: Create a new property for **hasSeverity** and make it **functional** (single valued)
- Step 3: Define a new kind of Pneumonia using the **hasSeverity** property and the filler **severeSeverity**

Because there are several steps this is easiest to do using the Value Partition Wizard that can be found on the Wizard menu (or the tools menu if you have a later version of Protégé.) The Wizard steps through the process.

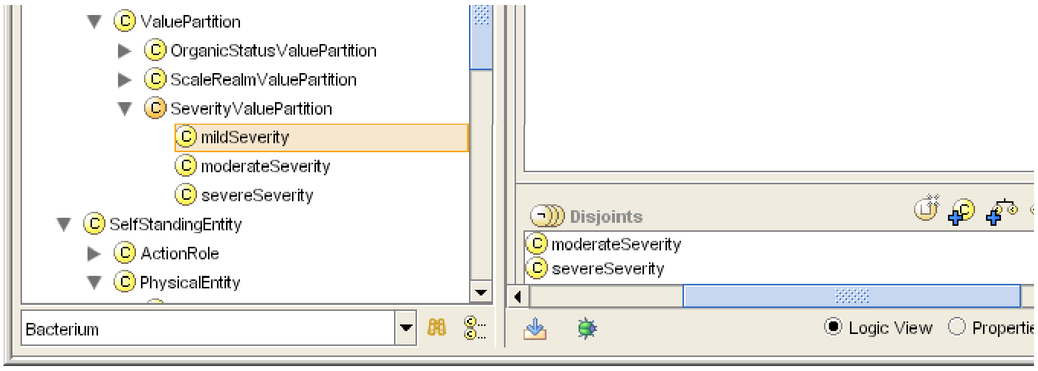
- Select **Create Value Partition** from the **Wizards** menu
- Enter the name **SeverityValuePartition**, and click next.
- Set the property name to **hasSeverity**, and then click next

- Click the **append text** box and enter **Severity**
- Enter the values **mild, moderate, severe** on separate lines, click next
- Check the names for typos, click next
- In annotations select **rdfs:comment** and add something like “severity for use with clinical conditions”, click next.
- Accept the default to keep this value partition under **ValuePartition** by clicking **Finish**.
- Go to the properties tab and set the domain of **hasSeverity** to **Disorder**. (In new versions of the widget this step is included in the Wizard.)

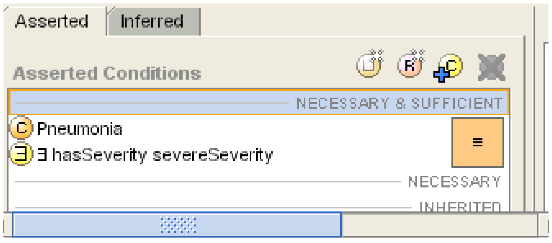
The results should be as shown below with three subclasses under SeverityValuePartition and SeverityValuePartition defined as being “*mildSeverity OR moderateSeverity OR severeSeverity*”. This type of definition is sometimes called a “covering axiom” and can be expressed separately from the class axiom in the OWL Abstract syntax.



The subclasses are all disjoint as shown below.



We now have the machinery to define a **SeverePneumonia** as “any **Pneumonia** that **hasSeverity** some **severeSeverity**”.



Some readers may find this formulation of "value partitions" surprising in that it uses classes rather than individuals to specify values. A more detailed discussion can be found the ProtegeOwl (Pizza) tutorial[4] or on the Semantic Web Best Practice Working Group page.[5]

3.3 Improving the definition of "Pneumonia"

When we first created **Pneumonia** it was a s a primitive class. All that is represented in the class is that it is a **Disorder** and **hasLocus Lung**. That might be enough for some applications, but for any very extensive ontology of diseases we will probably want more information.

The simple definition in a dictionary for "pneumonia" is an "Inflammation of the lung"[6]. There are many "inflammations" – of almost any organ – so that part of the definition seems worth capturing.

What about "inflammation" should that be primitive or defined. Depending on the dictionary, the definition will be something about a "morphology characterised by redness, heat, infiltration by leukocytes…" Nothing like as simple as "Inflammation of the Lung".

In general, "inflammation" is probably best thought of as what philosophers call a 'natural kind' and left as a primitive. Any definition is likely to be inadequate. More importantly, the individual characteristics are almost never going to be used so the ontology will never need recognise an "inflammation"—in fact to do well requires quite different reasoning techniques. There are many other similar notions include "tumour", "infection", "fibrosis" "fracture" and almost all named anatomical structures – "lung", "liver", "heart" etc. By contrast, "Inflammation of the lung" is a simple definition that seems to mean what it says.

To implement this definition:

Make Inflammation, Infection, and Fibrosis

- Select Disorder; create a subclasses **Inflammation**, **Fibrosis**, and **Infection**, and make them disjoint with their siblings. (This can also be done with the **Create group of classes** wizard.)
- Add comments, something like "Primitive notion of Inflammation", see dictionary definition "a morphology characterised by redness, heat, infiltration by leukocytes…", etc.

Edit the definition of Pneumonia

- Select **Pneumonia**
- Check the disjoints window to be sure that pneumonia is not disjoint with any other classes. If it is, remove the disjoints using the remove disjoints button and selecting the option **Only between this class and its siblings** from the pop-up. (Defined classes should almost never have disjoints. It is easier and safer to remove them first.)
- Drag and drop the **NECESSARY** conditions to **NECESSARY & SUFFICIENT**
  OR select **Convert to defined class** from the right mouse button menu on the class.
- Classify to make sure the results still work and are satisfiable. If some **Disorders** are unsatisfiable, this is almost certainly because the disjoints are set incorrectly.

3.4 Normalising Ontologies - The principle of orthogonal taxonomies

The above principles can be elaborated as the "principle of orthogonal taxonomies"or "Normalising ontologies" see [2][7]

For self-standing concepts, it is important for modularity that each primitive class have only one primitive superclass. The taxonomy of primitive classes forms the 'skeleton' or 'backbone' of the ontology.

Keeping taxonomies of primitives independent means that each one can be modified separately. For example, we can add new kinds of disorder or new kinds of micro-organism separately.

---

[4] http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf

[5] http://www.w3.org/TR/swbp-specified-values/

[6] We will ignore any difference between "pneumonia" and "pneumonitis" for the purposes of the examples in this tutorial – a topic on which dictionaries disagree.

[7] Available from http://www.cs.man.ac.uk/~rector/papers/rector-modularisation-kcap-2003-distrib.pdf

This is most easily seen in examples from either biology or organisations. Consider the example of. It seems most natural to make the primary primitive classification of organic substances on the basis of their structure: "steroid", "protein", "inorganic ion", "peptide", etc. However, functional roles cut across all of these, so that "testosterone" is a "steroid hormone" and "Insulin" a protein hormone. Another application might be more interested in which organs secreted which hormone.

It is important that it be possible to modify the hierarchies for structure, role, and origin separately. The best way to achieve this is to keep separate taxonomies for '*roleSpecifiers*' or '*actionRoleSpecifiers*' (not to be confused with the use of the word "role" for semantic link in description logics"). Entire subtaxonomies of different roles can be built up and manipulated independently. For now we shall just create the minimum.

To illustrate this we shall create simplified classes for notions of **Hormone, Neurotransmiter, Steroid, Protein, AminoAcid, and Testosterone, Insulin, and AcetylCholine.**

Create RoleSpecifiers for Hormone and Neurotransmitter

- Select **OrganicActionRole** and make the two disjoint primitive subclasses **HormoneActionRole** and **NeurotransmitterActionRole**[8]**.** (NB, do not make them covering (using a "covering axiom" as defined above) because the list is obviously incomplete, and would probably, in principle, always be incomplete.)

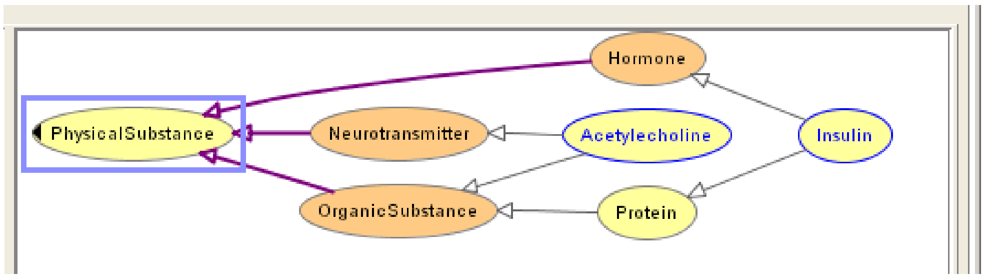
Define Hormone and Neurotransmitter

- Select **PhysicalSubstance** and create two defined subclasses for **Hormone** and **Neurotransmitter** using the property **hasActionRole** and the two **OrganicActionRole**s you have just created. (Note we create it at this level because there is no requirement that a hormone or neurotransmitter be an organic substance, even if its action is biological), *e.g.*
  Hormone ≡

PhysicalSubstance	"is kind of Physical substance"
∃ hasActionRole HormoneActionRole	"has action some Hormone action role"

- Under **OrganicSubstance** create a subclass for **Protein** and under that a substance for **Insulin**, both primitive, and use the property **hasActionRole** to give **Insulin** the **ActionRole HormoneActopmRole**. Do analogously for **Acetylcholine** and **NeurotransmitterActionRole**.

Run the classifier again to check that it comes out as you expect. Check it with OwlViz (remember to click the **Inferred Model** subtab.)



3.4.1 Principle: Untangling Taxonomies – "Self-standing concepts" vs "Value partitions"

Untangling taxonomies so that they are independent is equivalent to normalising databases. If the individual taxonomies of primitive concepts are not independent, then one is likely to introduce anomalies when updating the taxonomy or adapting it to a different purpose.

Typically existing classifications from other sources are tangled – *i.e* not normalised – and come with several different 'axes' mixed together in a single taxonomy – *e.g.* the axes "chemical structure", "action", and "use" are entangled in typical drug classifications and must be entangled to form a normalised logical ontology. Similarly, the axes "morphology", "anatomy" and "aetiology" are entangled in typical disease classifications.

[8] Where such roles go in the overall high level ontology is somewhat arbitrary, but for now it is convenient to think of them as a shorthand for more complex processes which we don't want to specify in more detail

A major benefit of using a logic based classifier is that these tangled taxonomies can be untangled and then reconstructed as sets of definitions which the classifier can maintain automatically. It is almost impossible to maintain large multiple taxonomies manually.

In untangling taxonomies, it is important to distinguish self-standing concepts from value partitions. Lists of self-standing primitive concepts are 'open', *i.e* incomplete, and so do not 'cover' the parent primitive concept. By contrast, lists of the values subsumed by a value Zartition are (almost always) complete by definition. (However, the **ValuePartition**s themselves need not be disjoint – *e.g.* a disease can be both serious and chronic. More on this point later)

Summary: Principles for Self-standing Taxonomies

For primitive classes representing self-standing concepts:

No primitive class should have more than one primitive superclass (parent). (If there seems to be a need for a second primitive parent, create a type of 'role specifier' instead or reorganise the hierarchy in some other way.)

The primitive subclasses of a class should be disjoint but should not covering – i.e the list of primitive subclasses should be assumed incomplete.

For classes representing value partitions

The primitive subclasses should be both disjoint and (usually) covering – the lists are usually complete by definition.

For defined classes

There should be no disjoints specified. Disjointness (or not) will be inferred by the classifier if the above rules (and a few others) are followed.

In a designing large ontology project, selecting which axes will be primitive and which constructed from definitions is a key task. For some implementation purposes, it may be desirable to re-tangle the taxonomies just as may be appropriate to de-normalise a database for efficiency. But the ontology design should be untangled.

3.5 Knowledge is fractal: How much detail should be modelled?

All ontologies are approximations. Building an ontology involves many decisions such as deciding # whether a concept such as "pneumonia" should be left as a primitive or defined. There is no simple rule. Knowledge is fractal –it is always possible to model in more detail, and modelling is seductive – it is always possible to model in more detail than necessary.

However, there are a few guidelines:

- *Natural kinds should be modelled as primitives* – things that take a long time to define in a dictionary or are most easily pointed to or given by examples are likely to have to be modelled as primitives – species, parts of the body, basic disease processes "inflammation", are all probably best treated as natural kinds. Most natural kinds are expressed by a single word, but sometimes it takes a phrase or compound to name them. Sometimes the distinction is subtle. For example, the distinction between "black bird" and "blackbird", or in biology between "artery serving the liver" and "hepatic artery" – the first describes a class of arteries; the second a specific named artery in that class.
- *Categories that "mean what they say" such as "Disorders of the Lung" which will be used to query the knowledge base should usually be defined* – because the classifier can only classify things under defined classes. In fact, defined classes can be thought of as predefined queries. Such categories are sometimes called "analytic concepts" or "analytic kinds" because they can be analysed into their different aspects.

- *If the description does not involve axes that are critical to 'normalising' ('untangling') the taxonomies, then may be easier to model them as primitives*. You can always reconsider later whether it is worth defining them as the design for the ontology develops.

There are two practical matters that cannot be discussed fully within the scope of this short tutorial but should be mentioned.

- Definitions are more expensive than primitives computationally because the classifier has to consider what things should be classified under them as well as where they themselves should be classified. There may be practical reasons in large ontologies to leave some things primitive even though they could be defined.

Summary

Natural kinds should be represented as primitives

Other concepts may be expressed as primitives either because their definition is not needed in the applications or as a temporary expedient while the ontology is developed.

Analytic kinds that involve major axes of the ontology should be defined.

4. Locations and Parts

Part-whole relations and their interaction with the locus of diseases and procedures is one of the major topics of any medical ontology. We have already met the hasLocus attribute. Here we go on to provide an introduction to part-whole relations. Beware. The study of part-whole relations ('partonomies' is a field in its own right, known to philosophers, linguists, and mathematicians as 'mereology'. What follows barely scratches the surface.[9]

4.1 Lobar Pneumonia

The other important way to classify pneumonia is by where it appears in the lung. The goal of this section is

- To create the subdivisions of the lung into lobes
- To modify the definition of **Pneumonia** so that the classification works for lobar pneumonias to be kinds of pneumonias.

Note that the lobes of the lung are not kinds of Lung but rather parts of the lung. Kinds are not parts. This is one of the common and pernicious errors of ontology development.

Create the properties for subdivision

We want **isSubdivisionOf** to be transitive – subdivisions of subdivisions are subdivisions. Therefore we will create two roles, **isSubdivisionOf** and a child role **isSubdivisionOfDirectly** which is not transitive. At the moment having the two roles has little benefit, but in more complex applications it is much easier to put constraints on the non-transitive child role and it is often useful to be able to ask the question – what are the immediate children of a thing along a particular axis.

- Click on the **Properties** tab, select **RelationProperty** from the menu and **add subproperty** from the right mouse button menu.
- Name the new property **isSubdivisisonOf** and tick the **Transitive** box at the bottom.[10]
- Click the **O** (for object property) button by inverses and in the pop up type **hasSubdivision**.

[9] See also draft in preparation for the Semantic Web Best Practices Working Group, http://www.cs.man.ac.uk/~rector/swbp/simple-part-whole/simple-part-whole-relations-v0-2.html

[10] Setting inverses, domains, and ranges can have serious effects on performance so will be postponed for this tutorial.

- Add some documentation such as "*e.g.* the relation between lobes and organs, divisions and wholes, etc."
- Select the new **isSubdivisionOf** property from the Properties list and make a new subproperty **isSubdivisionOfDirectly**.
- Do *not* tick the transitive box.

Create the lobes of the lung

- Go back to the **Classes** window by clicking the **OWL Classes** tab.
- Select **OrganPart** (founder **MacroOrganicStructure**) and create a new primitive subclass **Lobe**
- Select **Lobe** and create a new defined subclass **LobeOfLung**
- Add the restriction ∃ **isSubdivisionOfDirectly Lung**

Define two different versions of "lobar pneumonia"

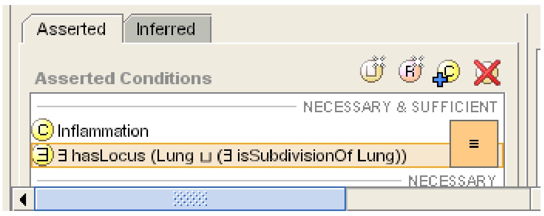
- Select **Pneumonia** and create a defined subclass of **Pneumonia** named **LobarPneumoniaX** which ∃ **hasLocus LobeOfLung**
- Select **Inflammation** and create a defined subclass of **Inflammation** named **LobarPneumoniaY** which ∃ **hasLocus** LobeOfLung
- Run the classifier to see the results by clicking on classify icon.
- Select each of **LobarPneumoniaX** and **LobarPneumoniaY** to see the results

LobarPneumoniaX is classified under **Pneumonia** because that was part of the definition. But look at **LobarPneumoniaY.** Clearly it ought to fit the definition, it is an "inflammation of the lung", but the problem is that it is of a part of the lung rather than the whole lung. (Note also that LobarPneumoniaX classifies under LobarPneumoniaY. Why?)

To fix the classification of LobarPneumoniaY requires changing the definition of **Pneumonia** and making it an expression equivalent to "inflammation located in the lung or any of its subdivisions". This requires the use of the expression editor that we will introduce here briefly by example.

Edit the definition of Pneumonia

- Select **Pneumonia** and the restriction ∃ **hasLocus Lung**
- Using the icons or keyboard shortcuts[11] edit the restriction until it looks like:



Inflammation
∃ hasLocus (Lung ⊔ (∃ isSubdivisionOf Lung))

This means

"Pneumonia is *any* Inflammation which has *some* location either in the Lung or in *some* subdivision of the lung"

Continue

- Reclassify the ontology

LobarPneumoniaX and **LobarPneumoniaY** should now be logically equivalent, as shown by: **LobarPneumoniaX** ≡ **LobarPneumoniaY**.

[11] You can also use the typing shortcut, "some" for ∃, "or" for ⊔, "and" for ⊓, etc.

5. Reciprocals: The asymmetry of statements in OWL.

5.1 Asymmetry of statements and reciprocals

Go back to the **LobeOfLung**. It has the property of being a lobe of the lung. Making this definition says nothing about either lungs or lobes in general, just that there might be a class of things which were lobes of lungs.

All lungs have lobes, but not all lobes are lobes of lungs. We might want to add this to the description of lung, but of course not to the definition of Lobe. For this reason, restrictions in OWL are normally asymmetrical.

We will come back to the right and left lung eventually, but leave it at that for now.

5.2 Reciprocal Statements

All “hands” are subdivisions of “some upper extremity”, and all “upper extremities” have some subdivision “hand” (barring some nasty problems with congenital abnormalities which we will leave aside for now). However, although **isSubdivisionOf** and **hasSubdivision** are inverses, these two statements are not equivalent. Therefore, in some situations we need both.[12]

To show this requires a *Restriction* on both *Hand* and *UpperExtremity.*

- Add a new restriction ∃ **isSubdivisionOfDirectly UpperExtremity** to **Hand**
- Create a new defined class for “**BodyPart**s which are subdivisions of **UpperExtremity**” Note that in general in definitions we use the transitive parent **isSubdivisisonOf** rather than the non-transitive child **isSubdivisionOfDirectly**.)
- Classify the model and be sure that **Hand** is classified under the new defined class.
- Create a new defined class for “*BodyPart*s which hasSubdivision *Hand*”
- Classify the model and see that nothing appears under the new concept
- Add a new restriction ∃ **hasSubdivisionOfDirectly Hand** to **UpperExtremity**
- Reclassify the model again and see that **UpperExtremity** is classified under the new concept.

6. Adding additional necessary conditions to defined concepts

So far we have been adding additional restrictions to the description of *Primitive Concepts, i.e.* the **SubclassOf** button has been pressed. As long as we are dealing with *Primitive Concepts* it is simply a matter of adding new restrictions to their description.

However, when we are dealing with *Defined Concepts,* we have to be more careful about which things are necessary *and sufficient* and which are just necessary. Consider what would happen if we want to say that “BacterialPneumonia” has a potential treatment “Antibiotic”. We don’t want the *definition* of *BacterialPneumonia* to include that it is potentially treatable by antibiotics. If we did, then *BacterialPneumonia* would be defined as

> “Any *Inflammation* which *hasLocus Lung, hasCause Bacteria, and hasPotentialTreatment Antibiotic”*

In order to decide if something were *BacterialPneumonia* we would have to determine if it had the potential treatment of *Antibiotic*. But we probably want to find out if something is a bacterial pneumonia precisely to determine if it might be treatable by antibiotics. We would have to specify in advance precisely what we want to infer.

Formally, in OWL syntax, this process is quite involved. The Protégé OWL tool was specifically designed to make it easy.

To define Drug and Antibiotic

- Select **OrganicActionRole** from the **OwlClasses** list
- Create a new primitive subclass named **DrugActionRole**
- Create a primitive subclass of **DrugActionRole** name **AntibioticActionRole**
- Select **PhysicalSubstance** from the **Classes** list.

[12] Care is needed when using the classifier with reciprocal statements, as it is easy to produce ontologies which take for ever to classifier even though they are correct.

- Create a new **Defined** subclass named **Drug,** defined as any **Substance** that has an **DrugActionRole**
- Repeat for **Antibiotic** with **DrugActionRole**

To define a property for isPotentialTreatmentFor

- Click to Properties tab to go to the properties window
- Select **RelationProperty**
- Add a subproperty named **isPotentialTreatmentFor** (In a real ontology, this property would probably be part of a hierarchy of related properties, but we will keep it simple for now). Make its inverse **hasPotentialTreatment**. (Do not make it functional or transitive or symmetric.) Make the domain and range **Drug** and **Disorder** respectively
- Select **BacterialPneumonia;** currently it should have a set of N**ECESSARY & SUFFICIENT** conditions and no **NECESSARY** conditions.
- Highlight the **NECESSARY** line. and add a restriction ∃ **hasPotentialTreatment Antibiotic**. ("has potential treatment some antibiotic").

What it means

"*All* bacterial pneumonias (as defined) have a potential treatment *some* antibiotic"[13].

Note that this is essentially a rule, "If something can be inferred to be a pneumonia, then it has a potential treatment some antibiotic". Taken with the definition of "bacterial pneumonia", it says "If an inflammation of the lung is caused by bacteria, then it is potentially treatable by antibiotics".

This is *not* to say that OWL is fundamentally a rule language. To the contrary, various query and rule languages are being designed on top of OWL. However, it is often important to create such simple rules, and it is important to understand the dual role played by additional necessary statements on defined classes.

Test it out

- Create a defined subclass of **Pneumonia** which has a potential treatment by Antibiotic, named **PneumoniaPotentiallyAntibioticTreatable,** with the restriction **has-class hasPotentialTreatment Antibiotic**
- Run the classifier
- Check to make sure that **BacterialPneumonia** is classified under **PneumoniaPotentiallyAntibioticTreatable**

Enter the "reciprocal"

So far we have said nothing about antibiotics, only about Pneumonias. However, it is probably more important to say something about the fact that antibiotics are useful for treating BacterialPneumonia

- Repeat the above to produce the axioms to say that all antibiotics are potentially treatment for some pneumonia. (Again, don't worry too much about the meaning of "potentially".)

Test it out

- Create a defined subclass of **PhysicalSubstance** which is a potential treatment for Pneumonia, named **PotentialPneumoniaTreatmentSubstance,** with the restriction ∃ **isPotentialTreatmentFor Pneumonia** ("isPotentialTreatment for some Pneumonia")
- Run the classifier.
- Check to make sure that **Antibiotic** is classified under **PotentialPneumoniaTreatmentSubstance**

6.1.1 The problem with generalisations – why the above is not really what we want to say

The statement "all antibiotics are potential treatments for pneumonia" is undoubtedly false, certainly in any clinical sense. The above statement is therefore two strong. What we want to say is that "*Some* antibiotics are treatments for some *pneumonias*". But our basic rules say that the only things we can say about a class are things that are true of all its individuals. On the other

[13] This statement depends on the interpretation of "potential", but such statements are highly useful in searching for potential treatments.

hand we want to be able to find Antibiotics by searching for “potential treatments for pneumonia”. How to cope?

There is no completely satisfactory solution in OWL.

One solution is to use “non-standard” reasoning simply to look at the inherited properties for BacterialPneumonia and note that one of them is “hasPotentialTreatment some Antibiotic”. This works well but does not actually produce a class of such treatments as would “standard reasoning”.

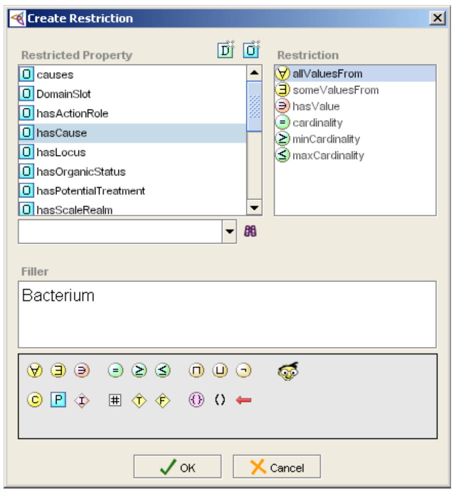
Another solution is to accept the overstatement as convenient, or to use a “kluge” to indicate when antibiotics are not really appropriate for pneumonia.

Other solutions involve extensions for defaults and exceptions and/or probabilities that are beyond the scope of this tutorial.

7. PureBacterialPneumonia and Universal (“only”/∀) restrictions

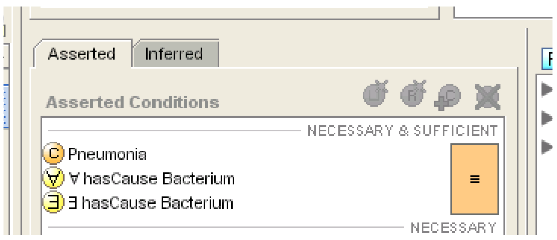
If we want to say that there is a kind of pneumonia which is caused by bacteria and only bacteria, then we have to add another restriction to say that it is caused by bacteria and *only* bacteria. For this we need an *allValuesFor* (“universal”/only/∀) restriction.

- In the **OwlClasses** tab select **BacterialPneumonia.**
- From the right mouse button menu select **Create clone** and name the new class **PureBacterialPneumonia.** Delete the restriction ∃ **hasPotentialTreatment Antibiotic**.
- In the **Asserted conditions** pane, select any of the restrictions in the **NECESSARY & SUFFICIENT** subpane. (Do *not* select the **NECESSARY & SUFFICIENT** line itself. This will start a new definition rather than add to the existing one.)
- Click the R (restrictions) icon and select the **allValuesFrom** in the right-hand upper pane; Select **hasCause** in the left hand upper pane, and enter **Bacterium** in the Filler pane.



Bacterium

The **Asserted Conditions** pane should now look like:



Pneumonia
∀ hasCause Bacterium
∃ hasCause Bacterium

This means[14]:

"hasCause only **Bacterium**" and
"hasCause some **Bacterium**s"

Or in otherwords

"Pneumonias which are caused by bacteria and only bacteria"

- Classify it:
- Double click on **PureBacterialPneumonia** to see the classification.

The results should show **PureBacterialPneumonia** classified under **BacterialPneumonia** and a sibling to **MixedPneumonia** and **PneumococcalPneumonia.**

7.1.1 Why both Restrictions are necessary: "Only does not imply some, nor vice versa"

Both Restrictions are necessary. To see why: =

- Create a clone of **PureBacterialPneumonia** and name it **PurePureBacterialPneumonia**
- Select the existential **(∃ someValuesFrom)** condition and delete it.
- Reclassify

PureBacterialPneumoniaXX is not classified as a kind of **BacterialPneumonia.** And if you click on the + sign to open the tree you will see that **PureBacterialPneumonia** is classified as a subclass of**Pure BacterialPneumoniaXX.** Why?

That **PureBacterialPneumonia** should be classified as a kind of **PureBacterialPneumoniaXX** is easy to see. **PureBacterialPneumonia** has the same constraints as **PureBacterialPneumoniaXX** plus one more; therefore anything that satisfies the constraints of **PureBacterialPneumonia** must satisfy the constraints for **PureBacterialPneumoniaXX**. That is the definition of subclass.

That **PureBacterialPneumoniaXX** should not classified as a kind of **BacterialPneumonia** is subtler but very important.

The restriction:

∀ **hasCause Bacterium**

means

"Causes are *only* **Bacterium**s" [15]

However, that the causes are *only* bacteria merely means that there are no causes which are *not* bacteria. It does not imply that there are any causes at all. Therefore an "uncaused" pneumonia counts, trivially.

7.1.2 Open World Reasoning and allValuesfrom

A further question to think about. Why is **PneumococcalPneumonia** not classified under **PureBacterialPneumonia**? Looking at the definition, the only cause given is **Pneumococcus** which clearly classified as a **Bacterium.** Why is that not sufficient to satisfy the definition?

Anyone used to dealing with database queries or logic languages such as Prolog may be particularly puzzled by this behaviour. Databases and logic programming use a 'closed world assumption', *i.e.* they assume that everything is known in advance The reason is that unliked databases and logic programming, OWL uses 'open world reasoning". Negation in OWL means "impossible" or "would cause an inconsistency" or "provably false" – in any world consistent with what we already know. To be false means that no new information could ever make it true. Most database systems use a much weaker version of negation known as 'negation as failure'.

Negation in database or logic programming uses a 'closed world' assumption in which negation is failure, *i.e.* if you can't find it to be true, it must be false.

[14] or put another way—which is the reason for the notation—

"S*ome* cause is a **Bacterium**" and
"All causes are **Bacterium**s"

The use of "all" indicated by the logical symbol "∀", can be confusing in OWL and related representations. It is generally much easier to treat **to-class** constraints as meaning "only"

[15] or equivalently as often put in logic textbooks

"All causes are **Bacterium**s"

Since OWL is open world, **PneumococcalPneumonia** subsumes any **Pneumonia** that has **Pneumococcus** as one of its causes, just as **BacterialPneumonia** subsumes anything which has a bacteria as at least one of its causes, including **MixedPneumonia**. That **MixedPneumonia** is consistent with **PneumococcalPneumonia** shows that it is not true to say that all possible **PneumococcalPneumonias** are **PureBacterialPneumonias** – some might be mixed.

7.1.3 Summary

∃ **property Value** means:
"**property** has *some* value from the class **Value**"
(and may have other values unless the property is 'functional')

∀ **propertyName Value** means:
"**property** can *only* have the values from the class **Value"**
(but does not imply that **property** has *any* value)

References

1. Horridge, M., Drummond, N., Knublauch, H., Rector, A., Stevens, R., Wang, H. and Wroe, C. Building Ontologies with Protege-OWL Plugin, 2004,http://www.co-ode.org/resources/
2. Rector, A., Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. in *Knowledge Capture 2003*, (Sanibel Island, FL, 2003), ACM, 121-128.