protege-tutorial Documentation

Release 0.5

protege-tutorial

Sep 18, 2017

Contents:

Protégé 5 tutorial for GO Editors. V2

CHAPTER 1

Initial Preparation

GitHub Login

Obtain a login at GitHub (https://github.com/) and send your login name so you can be added to the GO Organization (https://github.com/geneontology). This is our repository for ontology files and issue trackers.

Clone this repository

Visit https://github.com/geneontology/protege-tutorial

click "Clone or Download"

This will give you a URL you can use

```
git clone https://github.com/geneontology/protege-tutorial.git
```

These files will be under `protege-tutorial` on your drive.

Install Protege 5 for Ontology Development

Download and install the latest version of Protege (5.1) for ontology development.

http://protege.stanford.edu/products.php#desktop-protege

The exercises assume that you have the tutorial directory checked out from GitHub. If for some reason you're unable to do this, you can download individual files from:

https://github.com/geneontology/protege-tutorial

CHAPTER 2

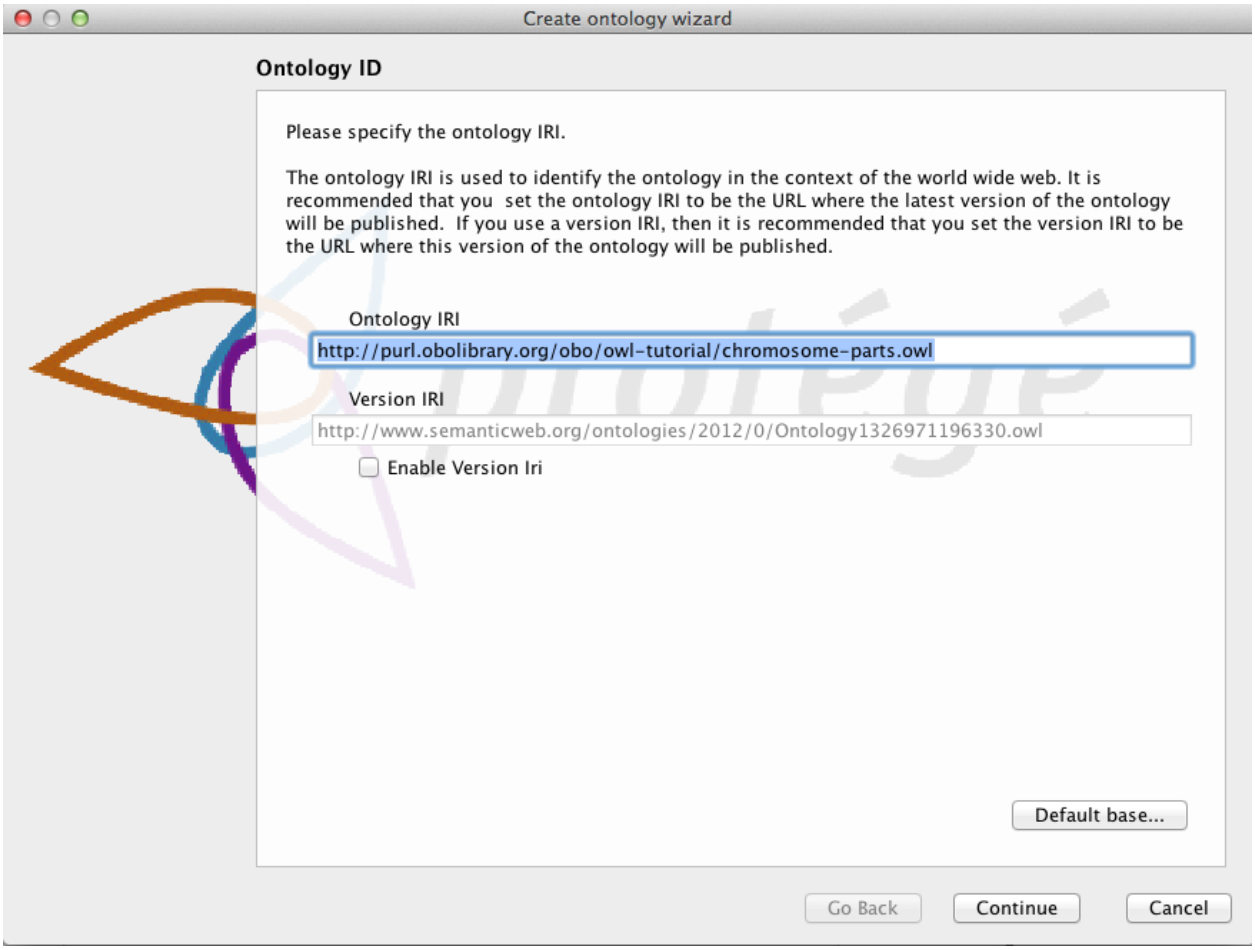
Starting Protégé

When you start Protégé a welcome dialog appears where you can choose to

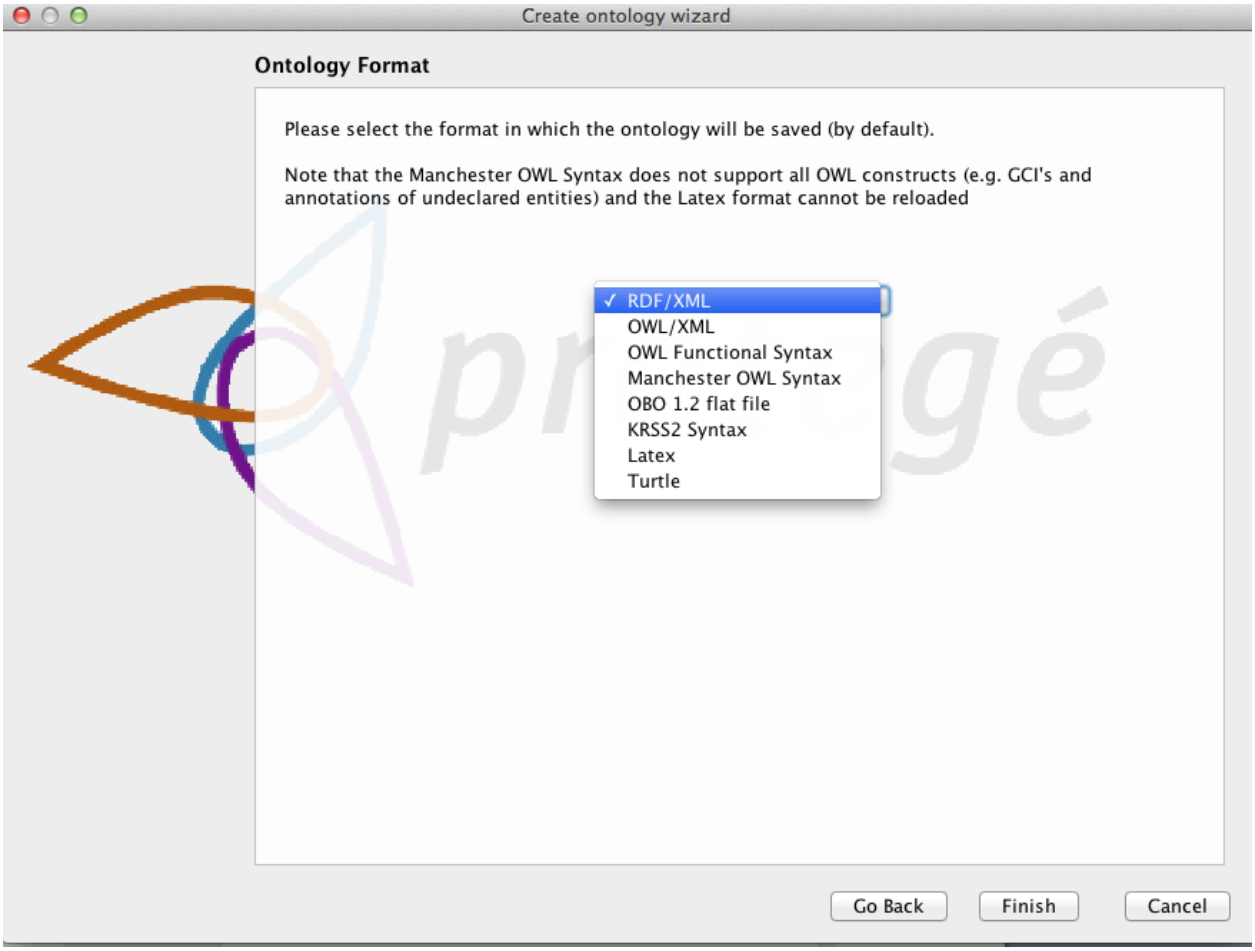
1. Create a new OWL ontology
2. Open an existing OWL ontology
3. Open an ontology from a particular URI (e.g http://purl.obolibrary.org/obo/go.owl).

We will begin by selecting “Create new Ontology”. In the next dialog we will create an IRI that can be used to

identify our ontology on the Web. You can set the IRI to anything you want at this stage, for this tutorial we will use http://purl.obolibrary.org/obo/owl-tutorial/chromosome-parts.owl.

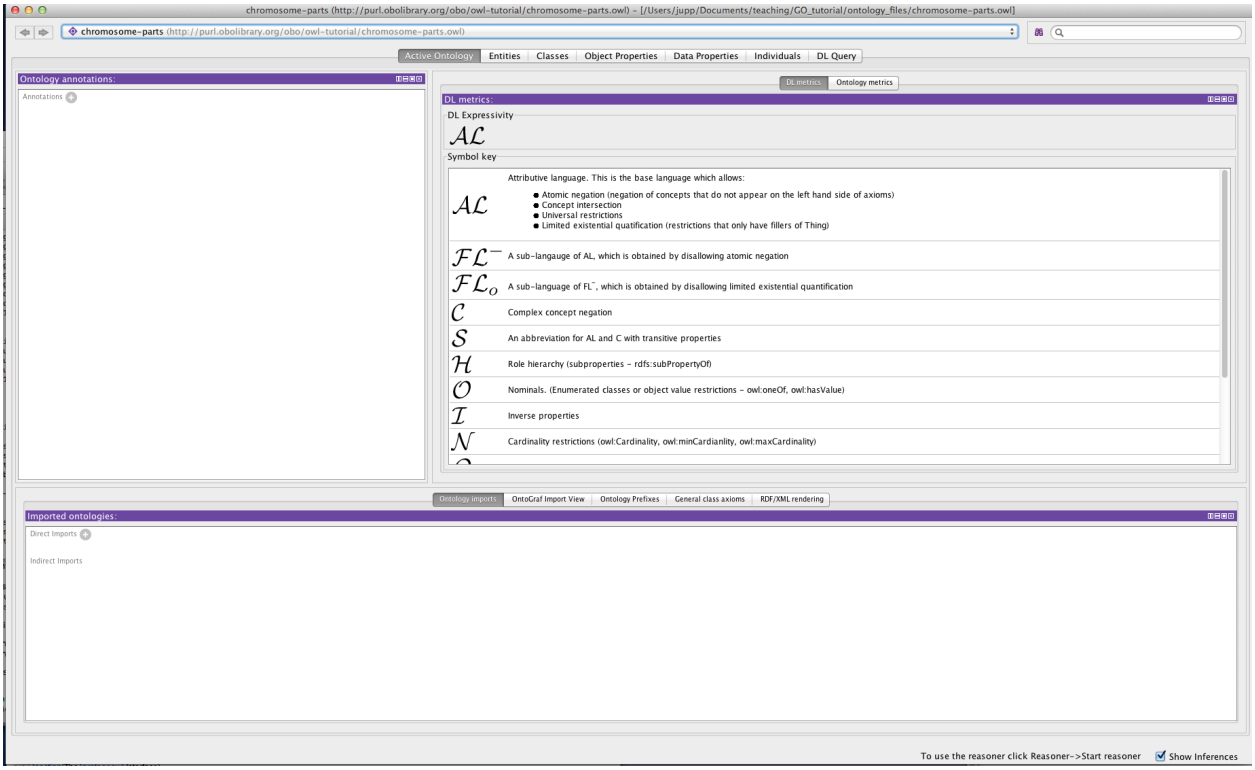


You will also want to save this ontology file to your hard disk. Use the next dialog to save the ontology file somewhere convenient. Finally you will want to choose a format for your ontology file. Protégé allows you to save your ontology in a variety of OWL formats, including the OBO file format. We recommend that you save your ontology in RDF/XML, as this is the most stable format to work with in Protégé. You can always choose to export your file in one of the other formats later.

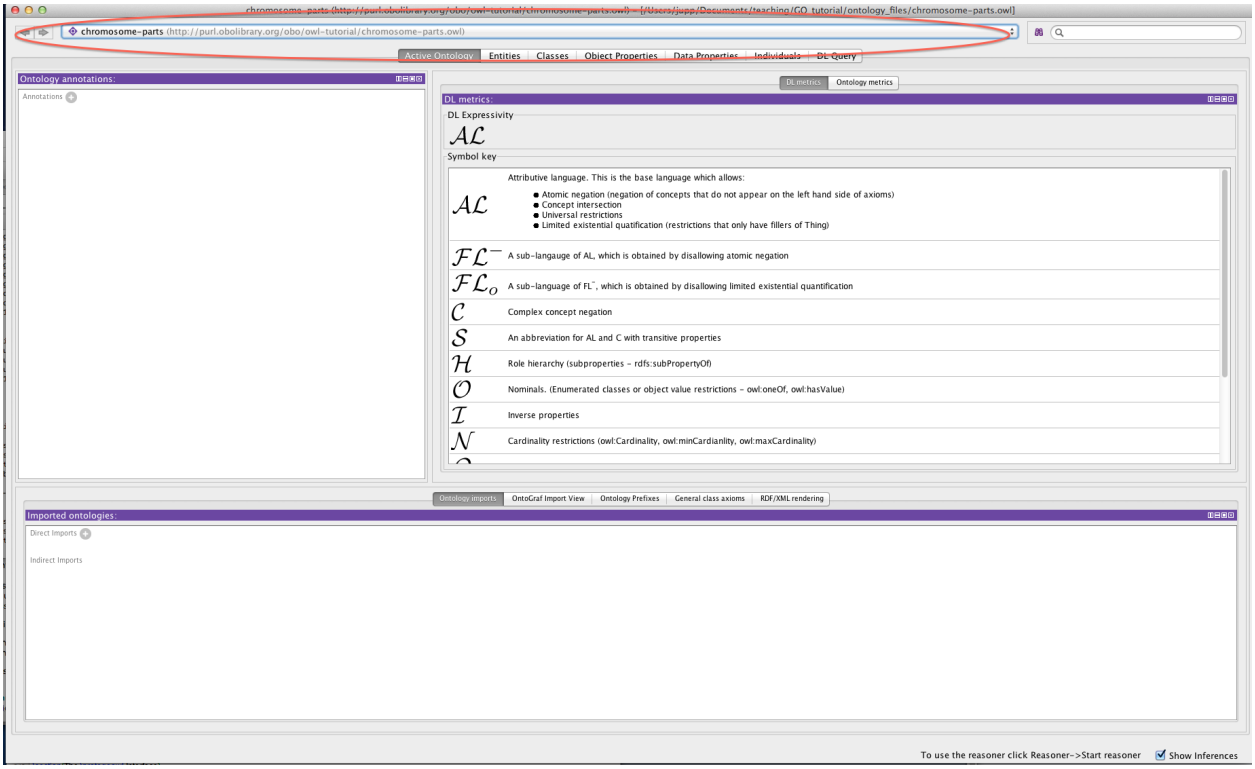


The Protégé UI

After a few seconds Protégé will launch into the main user interface. The protégé interface follows a basic paradigm of Tabs and Panels. The layout of tabs and panels is configurable by the user. By default protégé launches with 5 main tabs.



The first tab you see is the Active Ontology tab. Here you will find some basic meta-data about the ontology you are viewing. At the very top you see the IRI and file name of the active ontology you are viewing. Protégé allows you to work with multiple ontologies at once (See later), so this top bar is very important as it lets you know which ontology you are viewing and editing.

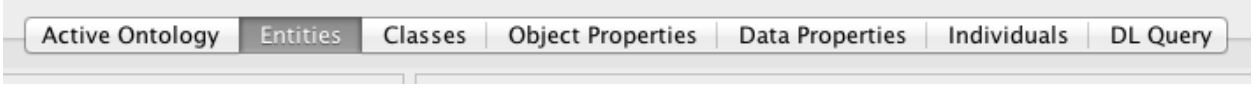


The panel on the left is the ontology annotations panel. You can use this panel to add basic meta-data to your ontology, such as the creation date, the authors and a short description.

CHAPTER 3

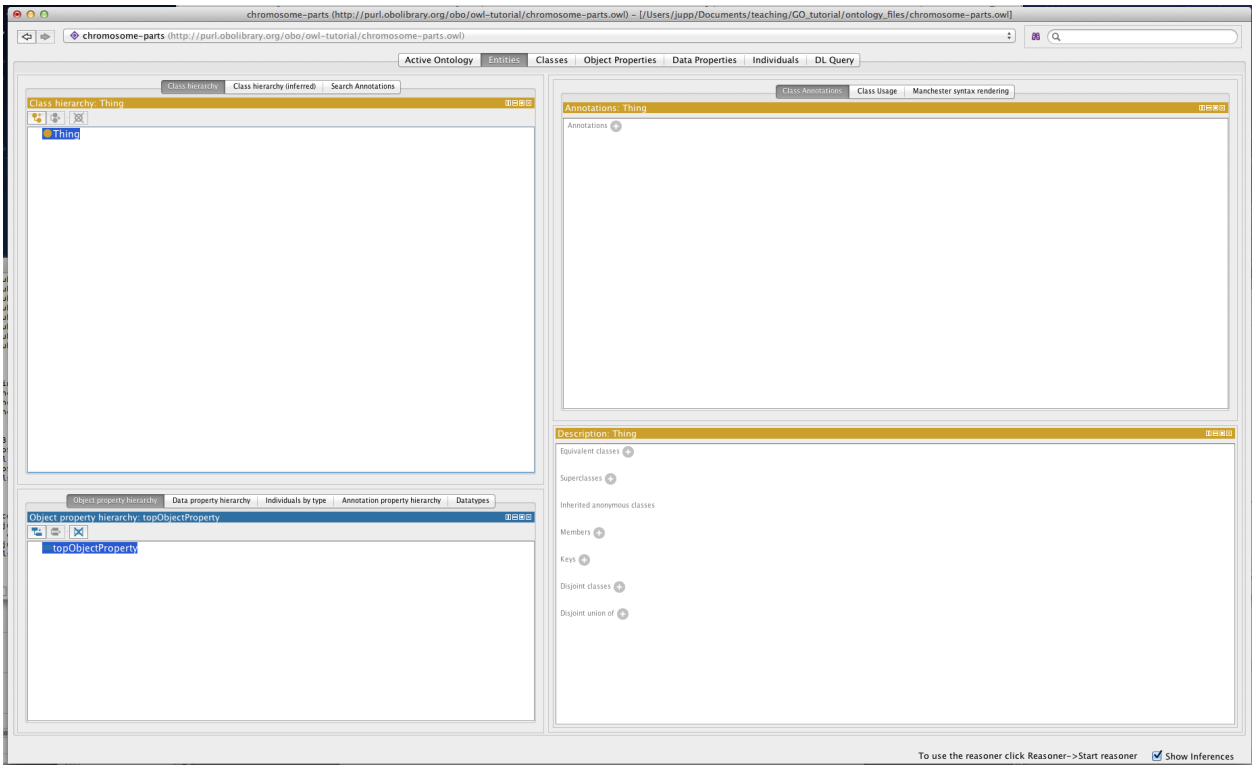
The entities tab

You will see along the top of the screen various tabs. Each tab provides a different perspective on the ontology. For example, the classes tab allows us to view and edit the classes in the ontology, and similarly the properties tab focuses on the properties in the ontology. The primary tab where you will spend most of your time is the entities tab.

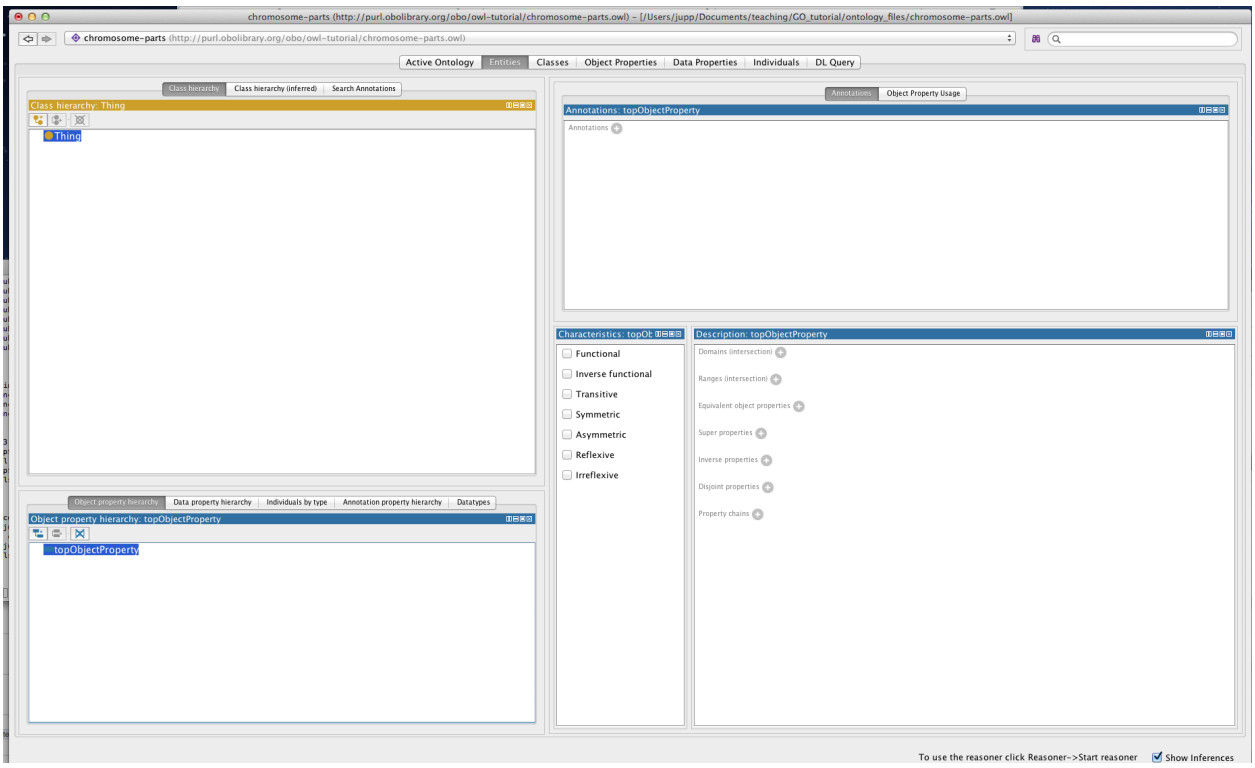


Select the entities tab and then select the Thing class. Thing is the root class for all OWL ontologies and it cannot be deleted in Protégé.

The entities tab is split into two halves. The left hand side provides a suite of panels for selecting various entities in your ontology. When a particular entity is selected the panels on the right hand side displays information about that entity. The entities panel is context specific, so if you have a class selected (like Thing) then the panels on the right are aimed at editing classes.

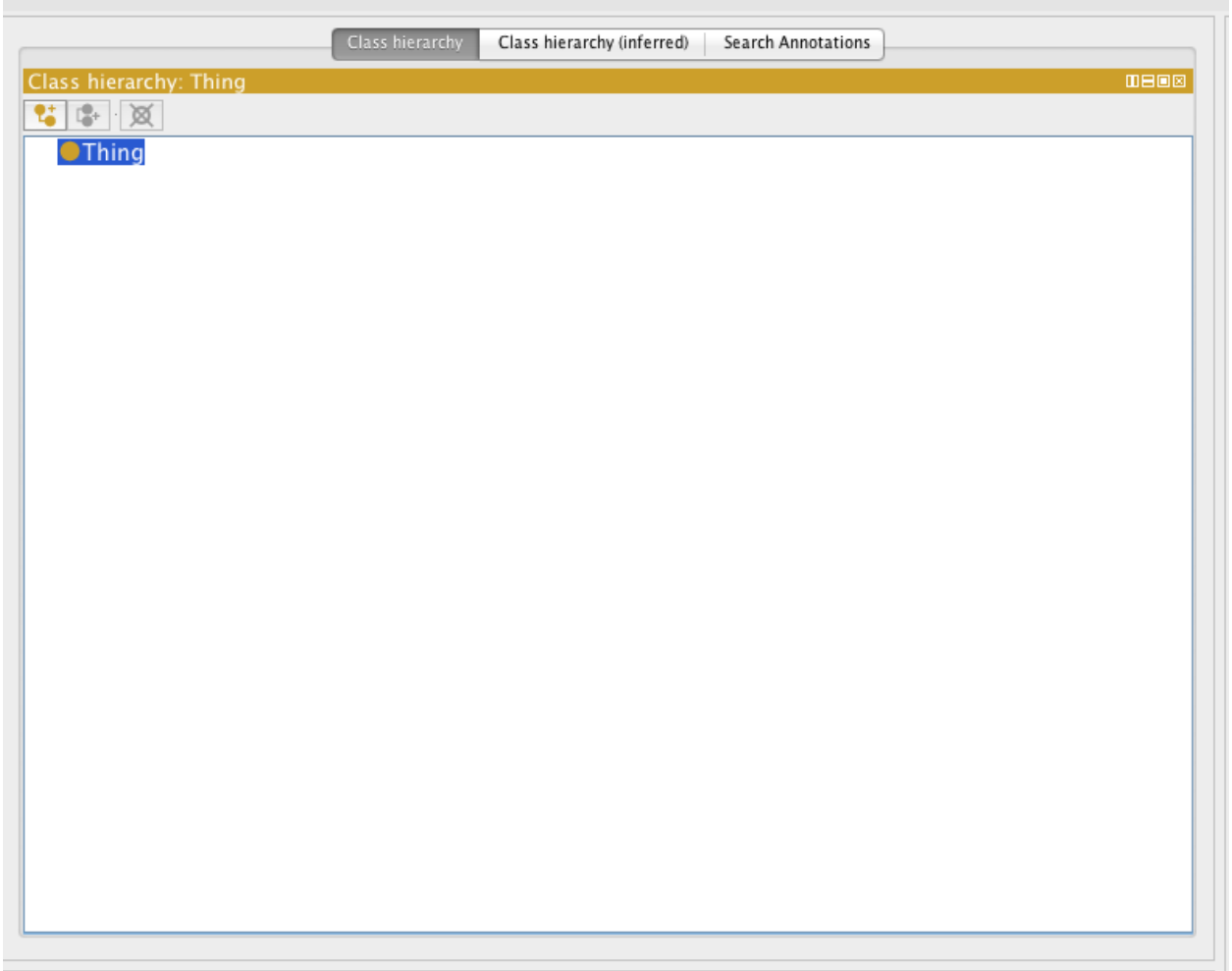


If you select the topObjectProperty property in the panel at the bottom, the right hand side panels will change context.

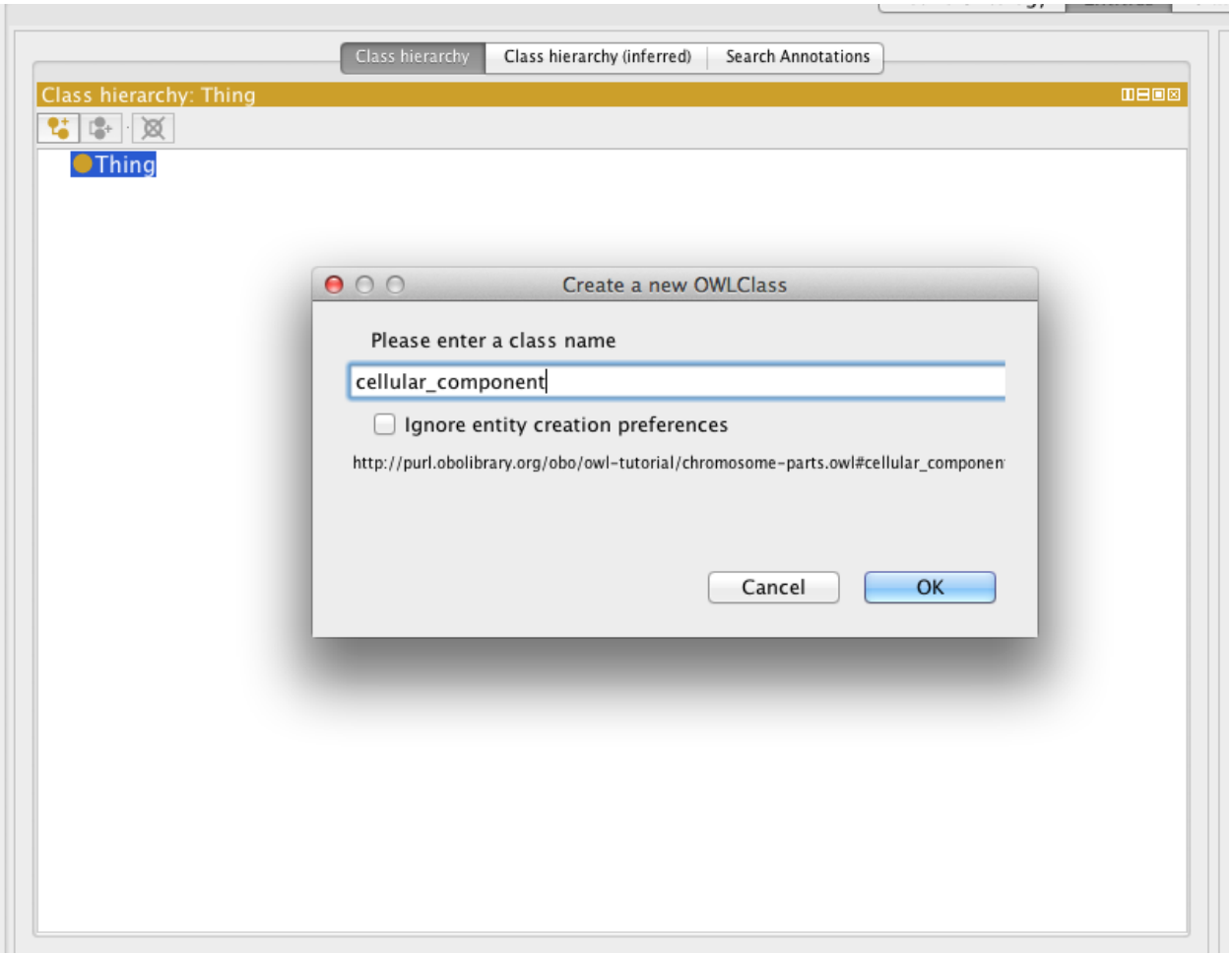


Creating your first class

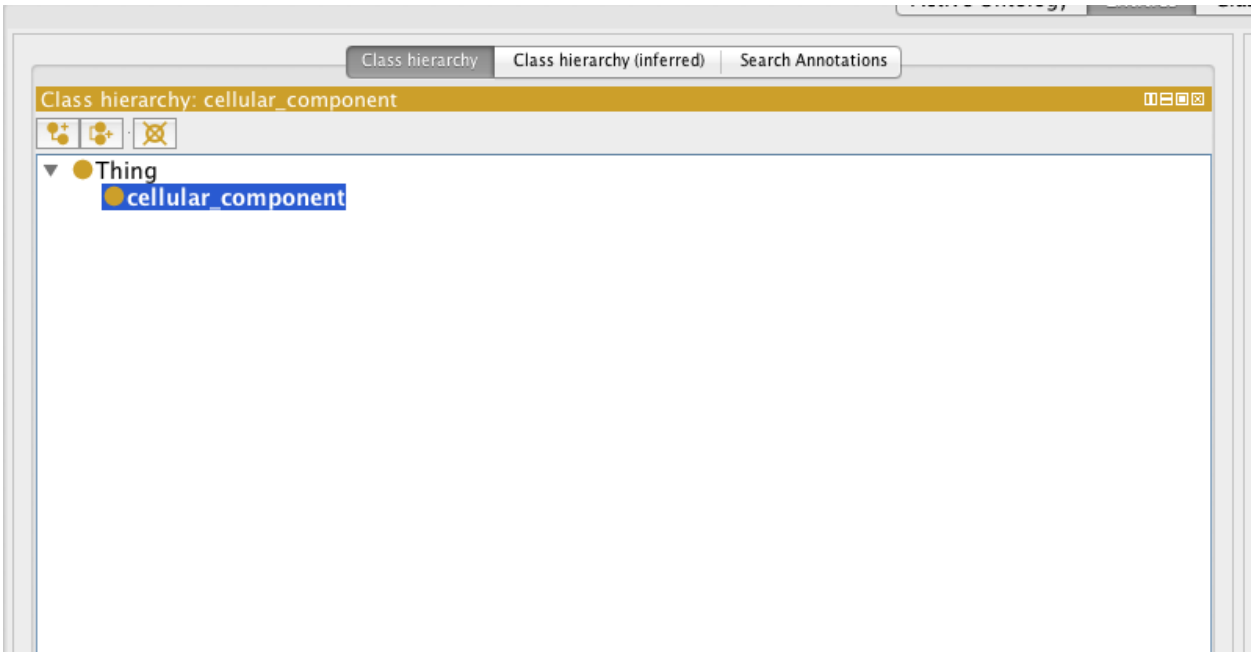
By far the most common panel for working with your ontology is the Class hierarchy panel.



There are three buttons at the top of the class hierarchy view. These allow you to add a subclass, add a sibling or delete a selected class. We will use the 'add subclass' button to add a child class to OWL thing. For now, simply name this class cellular_component.

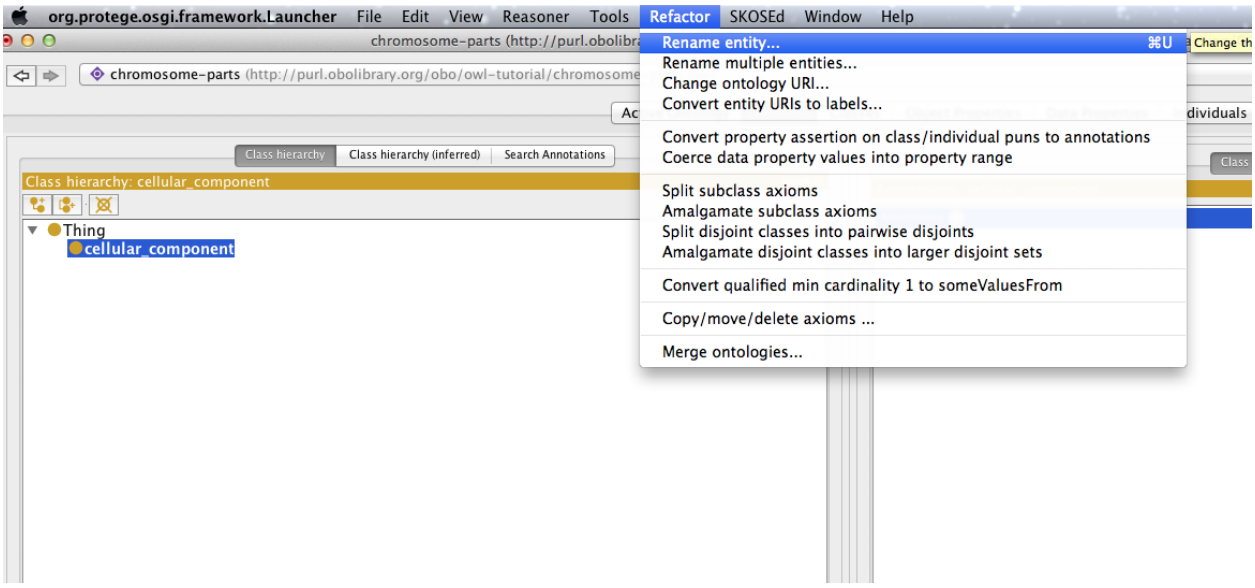


The class should have been created as follows. By default Protégé will use the ontology IRI, followed by a #, followed by your specified name (replacing spaces with underscores) as the the unique IRI for this entity. If you hover over this class with your mouse you will see the full IRI for this class

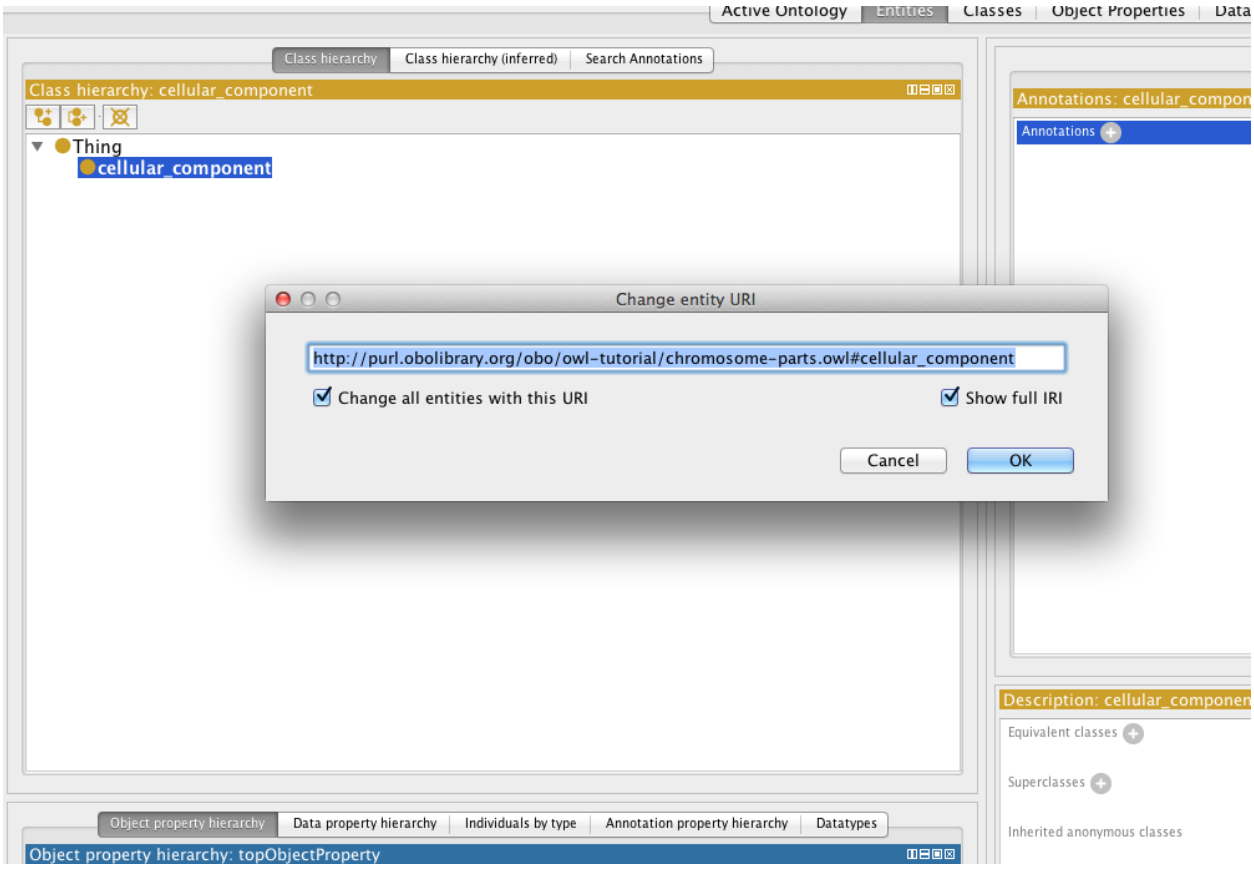


Renaming an entity

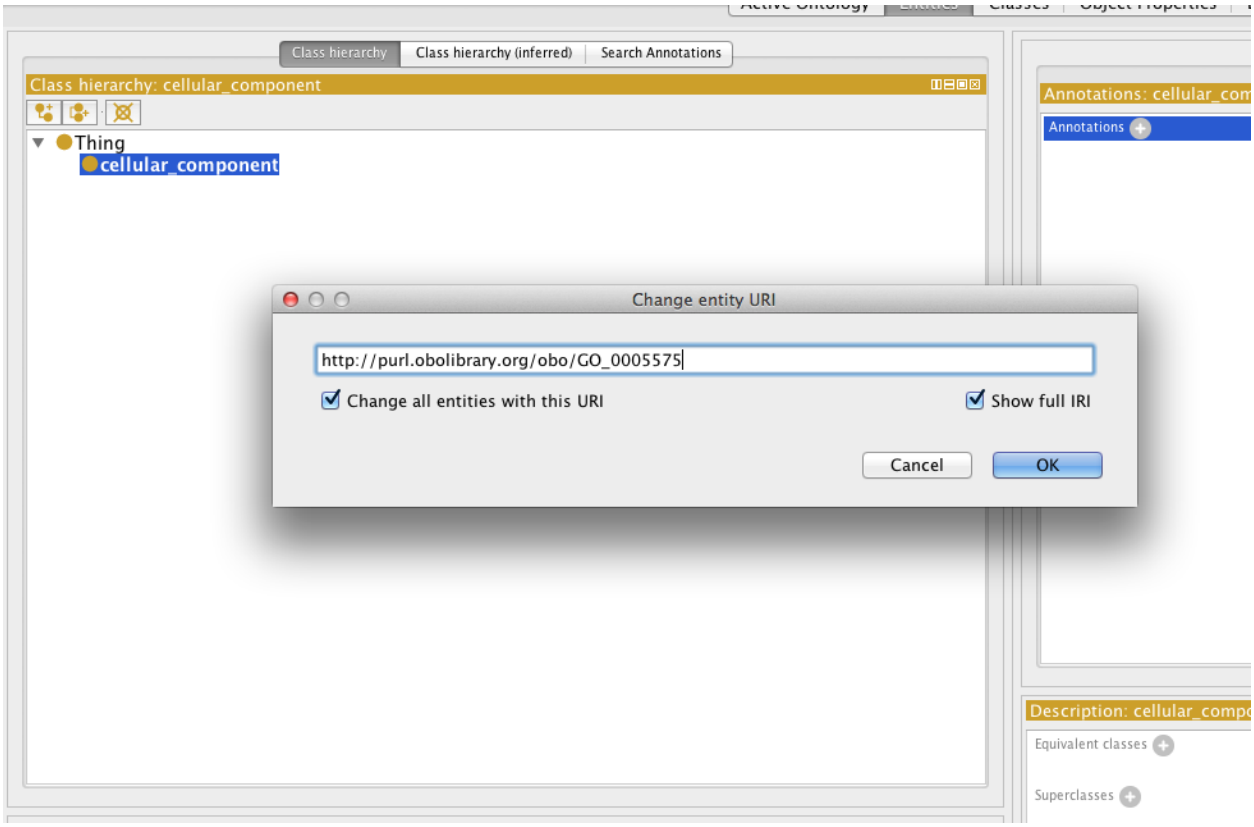
We can change the IRI for a concept using the rename function in the refactoring menu. Rename the cellular_component class to use its proper IRI from the Gene Ontology (http://purl.obolibrary.org/obo/GO_0005575)



Make sure the check the “Show full IRI” box so you can edit the full IRI.



And then paste or type in the correct GO URI.

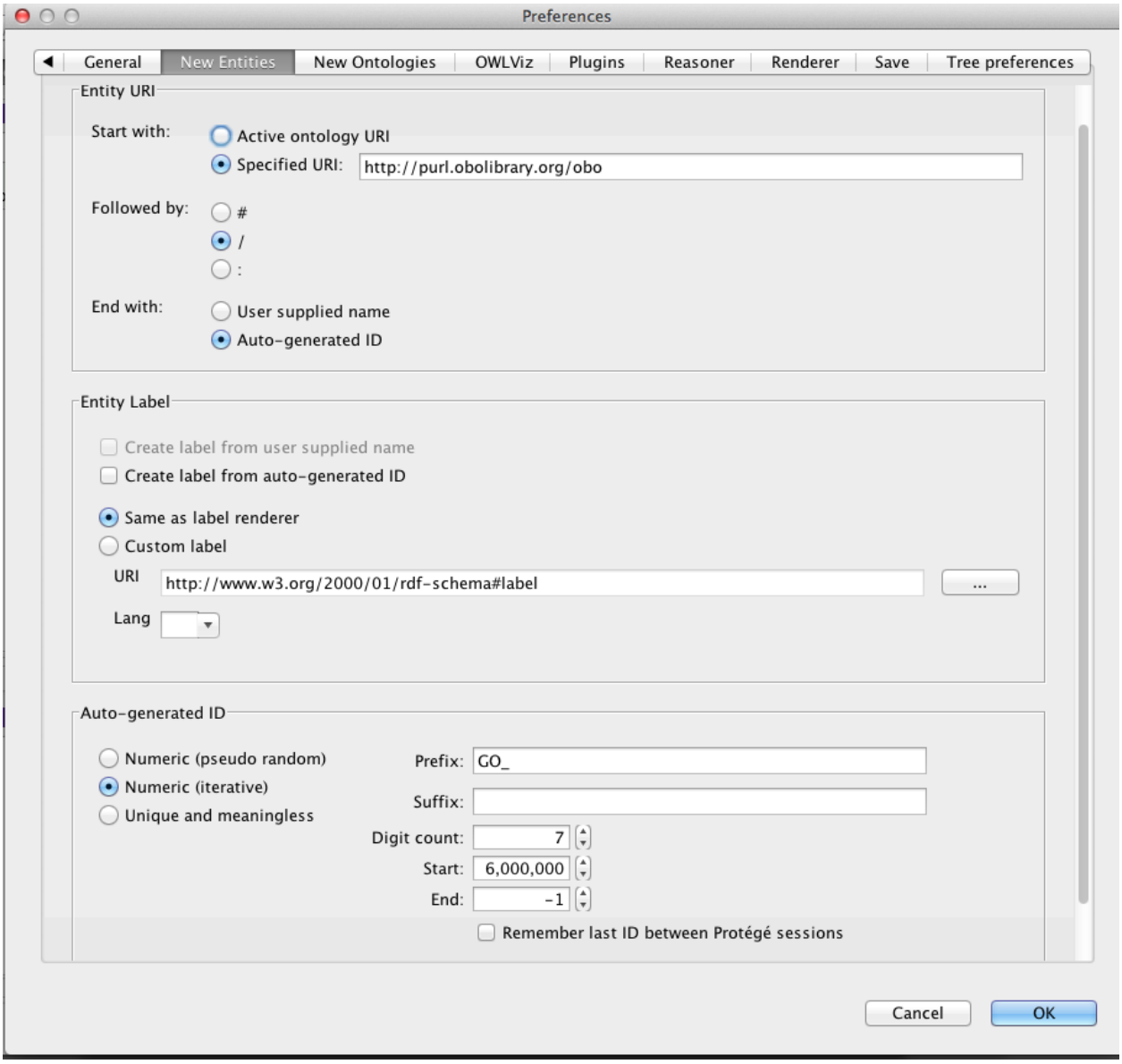


Now the correct GO URI appears in the ontology. Obviously you don't want to have to rename every entity you create when building your own ontology. Luckily Protégé provides a "New Entities" preferences panel where you can specify how new IRI should be created.

New entities

Terms in the ontologies we use have separate names and IDs. The names are annotation values (labels) and the IDs are represented using IRIs. The OBO foundry has a policy on IRI (or ID) generation. You can set an id strategy using the "New Entities" tab under the Protégé preferences.

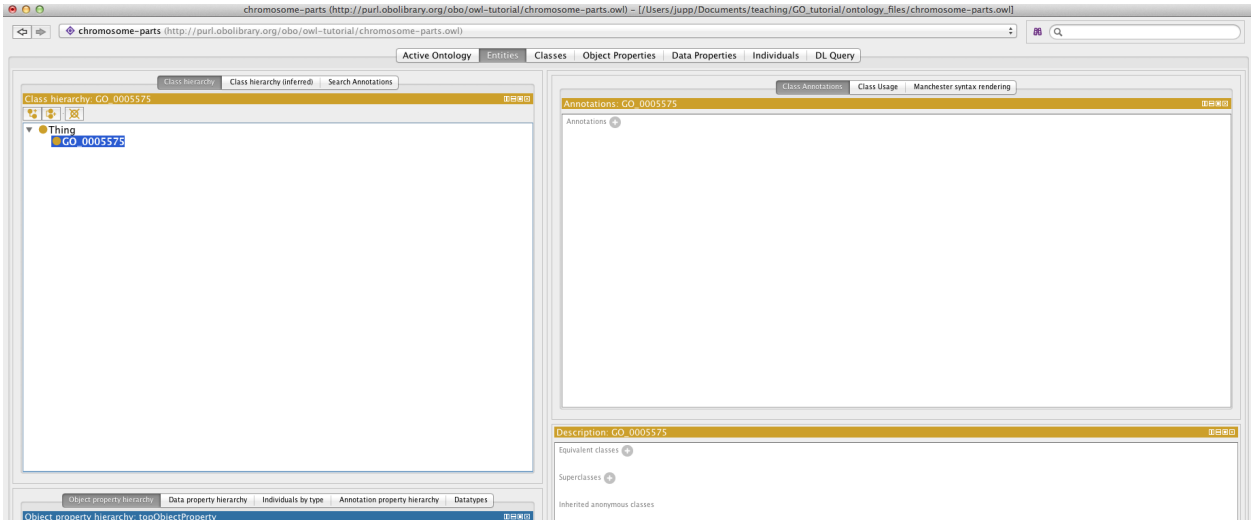
Set your new entity preferences as follows:



For ontologies other than GO, change the value of the prefix. Note that all OBO library ontologies should use the obolibrary.org Specified URI value.

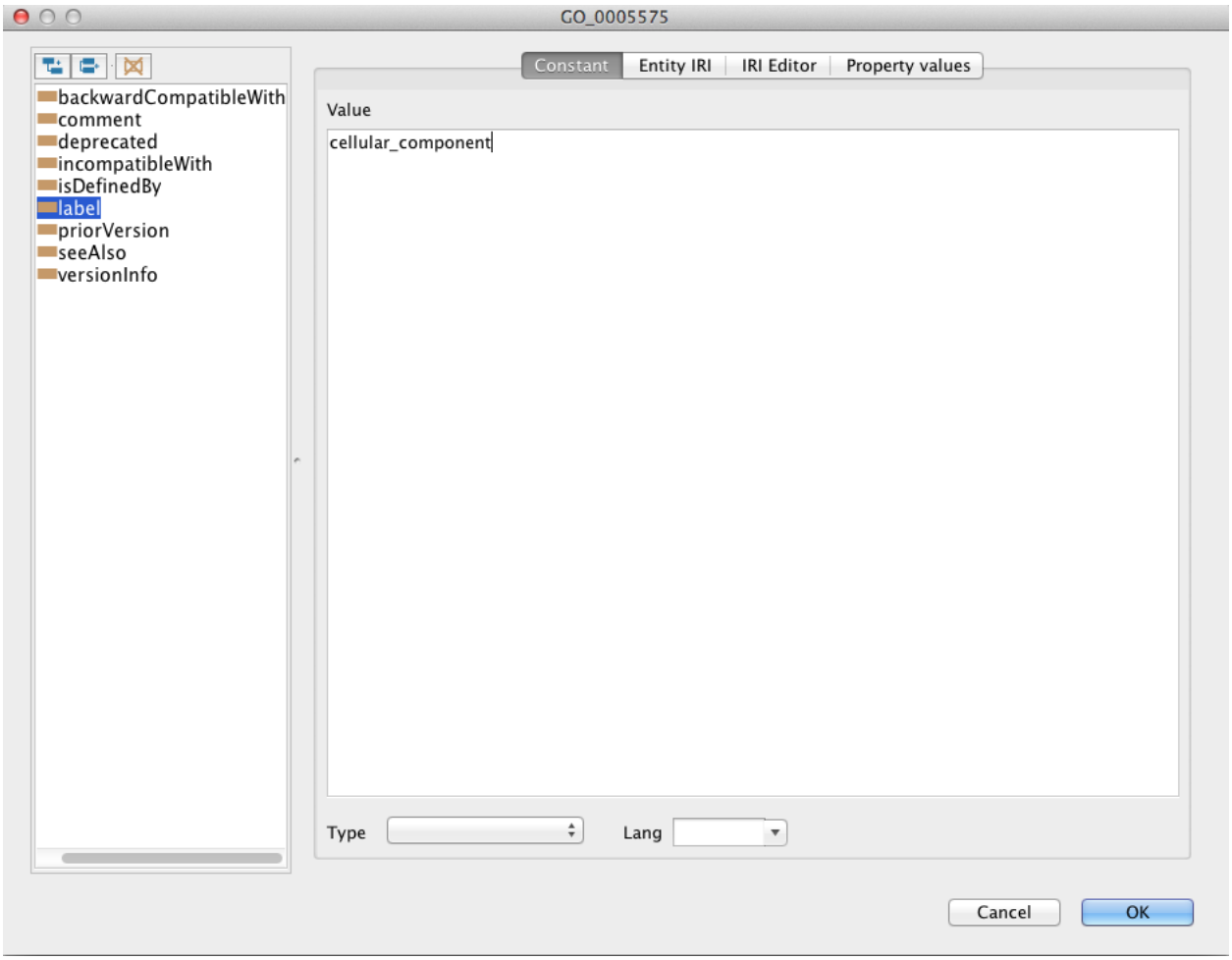
Adding annotations properties

You can add annotations (such as labels, descriptions, xrefs, etc.) to any OWL entity using Protégé. The panel on the right, named Annotations is where these annotations are added. Use this panel to add a "cellular_component" label to the class you created previously.



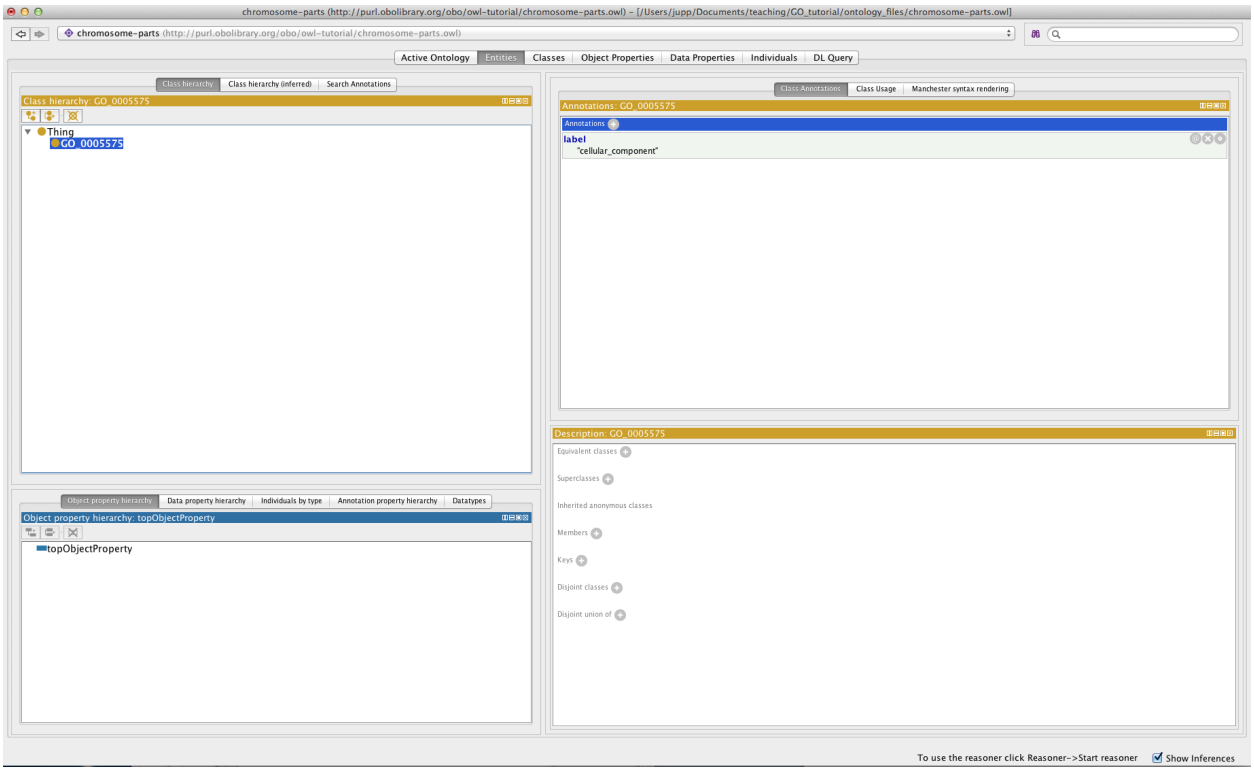
Select the + button to add an annotation to the selected entity. Protégé has a set of built in annotation properties, such as label and comment – add a comment such as "created during Berkeley tutorial". You can also create your own annotation properties.

Note that for practical purposes you will start from an existing OWL file that will include a pre-declared set of annotation properties such as 'has exact synonym' and 'definition'. You may never need to create your own annotation properties.

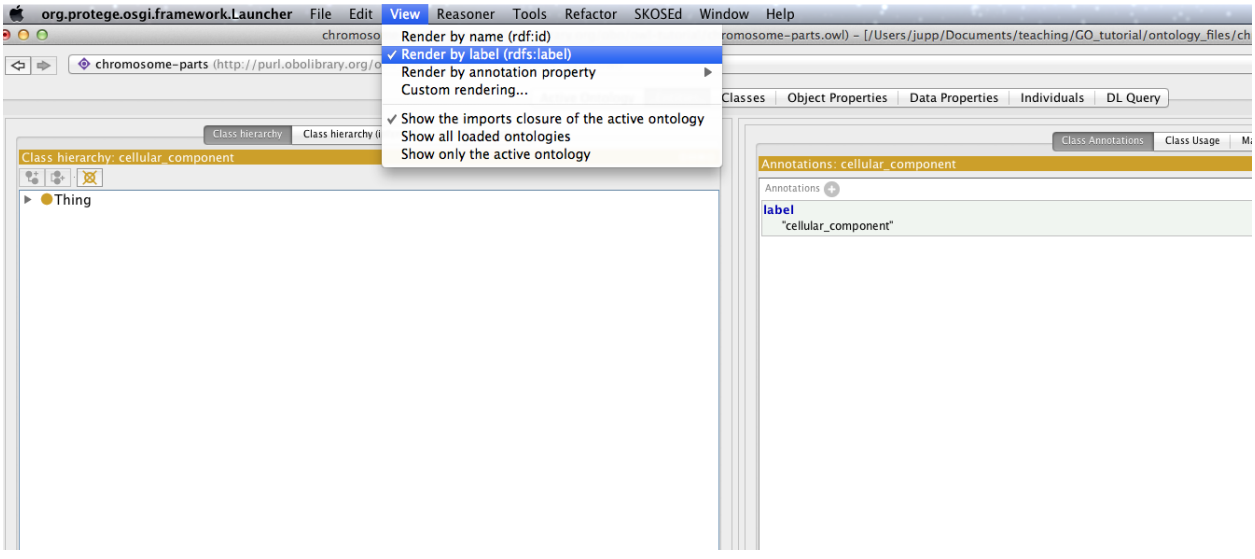


Setting label rendering

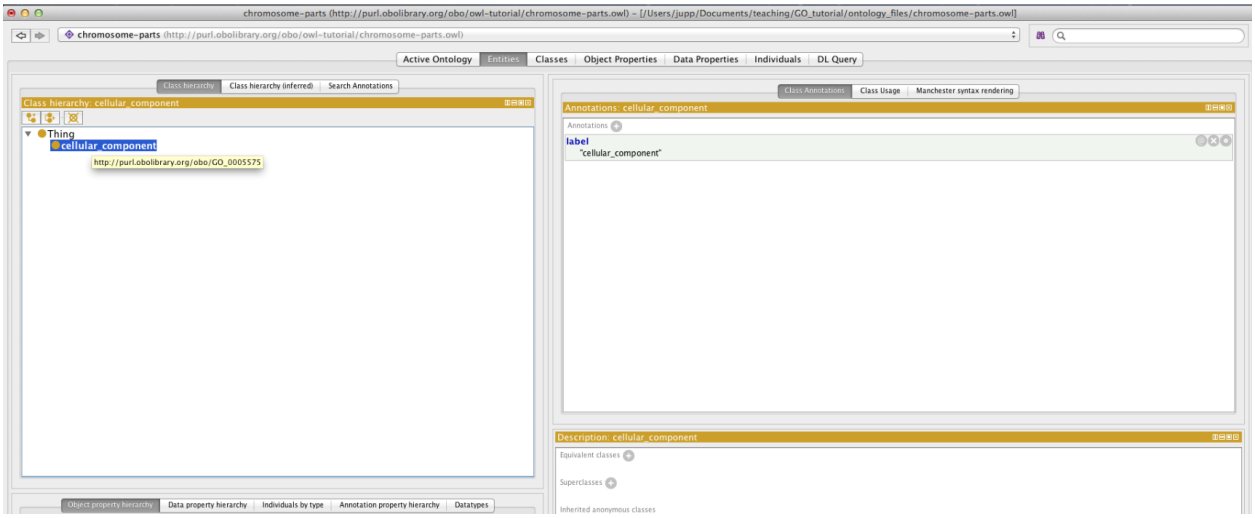
You can change how Protégé renders entities. It is common to want to view entities by the their label, rather than identifiers. You can tell Protégé to render on any annotation property you choose. Lets render all entities by their class label.



In the view menu choose render by label



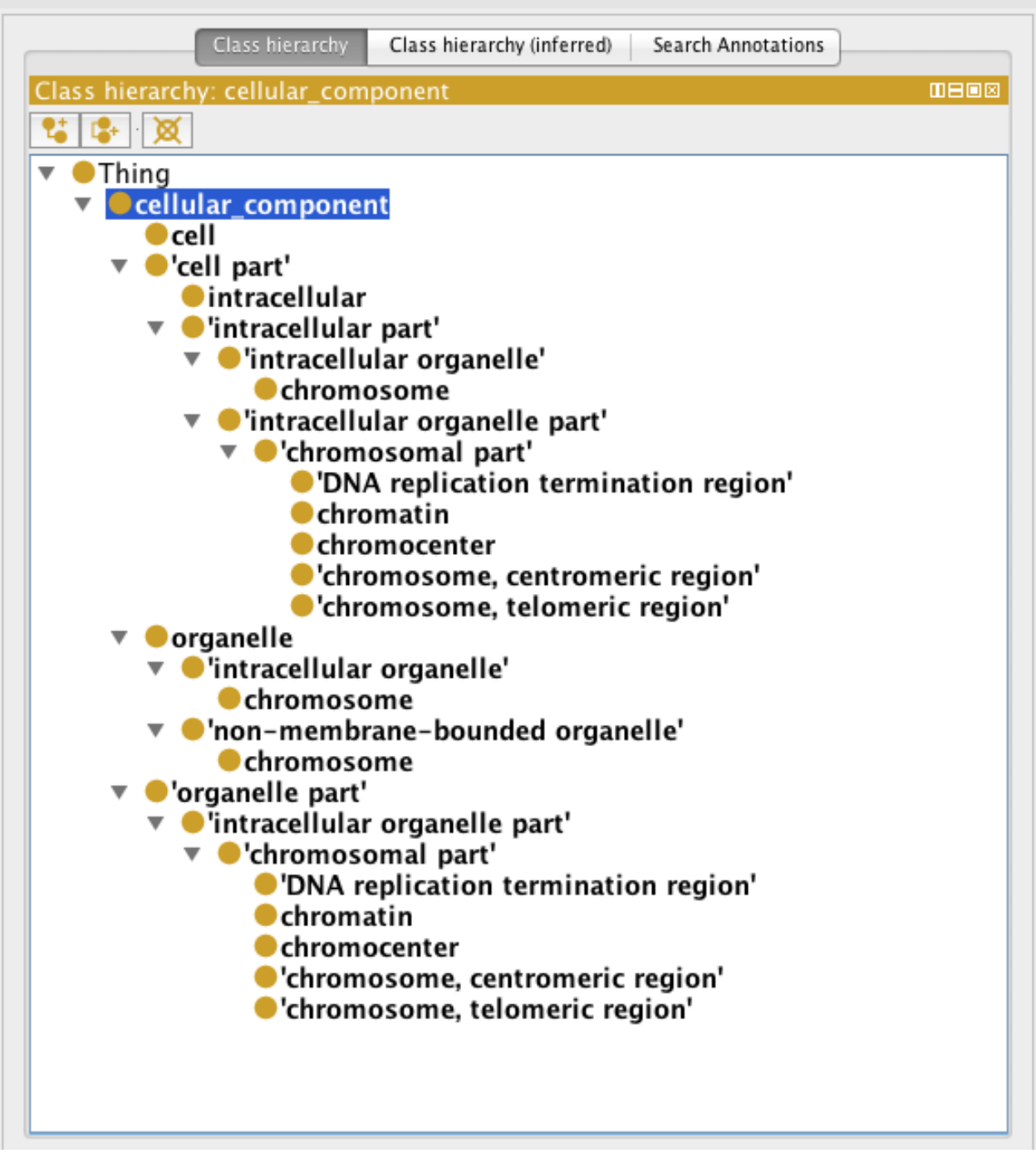
The cellular_component class will now render in the hierarchy view using the value of the label annotation property.



Creating the class hierarchy

We will now create a simple class hierarchy. In Protégé 'class hierarchy' typically refers to a sub/superclass hierarchy (also known as an is_a hierarchy in OBO-Edit). We will return to relations such as 'part of' later on in this tutorial. For now, we will take advantage of the fact that the GO cell component ontology allows us to bypass this for now by means of classes such as 'cell part' and 'nuclear part'.

Using the class hierarchy view create a small section of the cellular component branch of the GO as shown in the following screenshot. Play around with add subclass, add sibling and the drag and drop functionality.



Don't bother to add textual definitions, synonyms, etc. at this stage.

After you have become familiar, you can save your efforts or discard them – you won't need this ontology from here on.

EXERCISE: Basic Subclass Hierarchy

Go to the directory basic-subclass in the tutorial folder and open chromosome-parts.owl

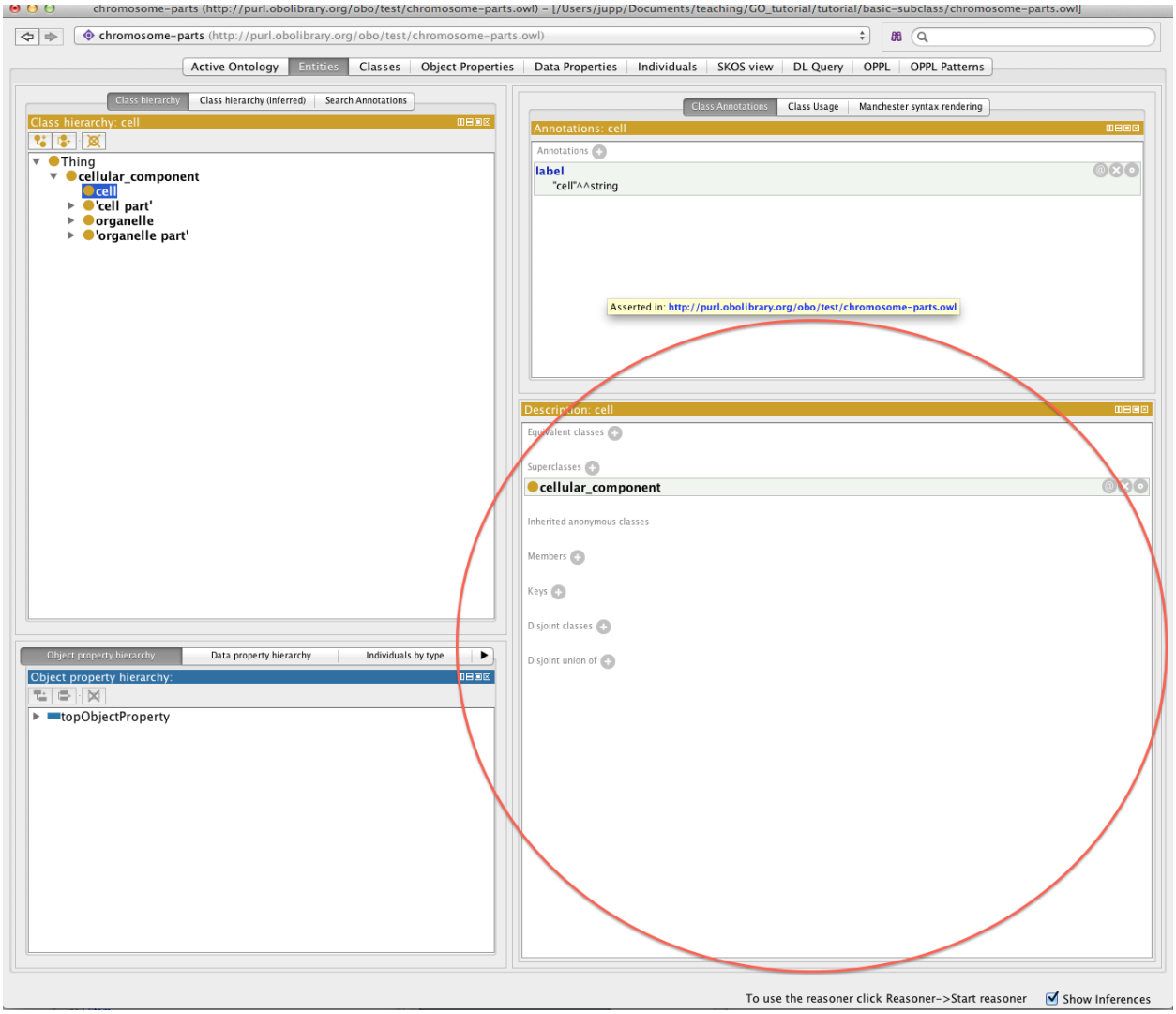
Follow the instructions in the README file

CHAPTER 4

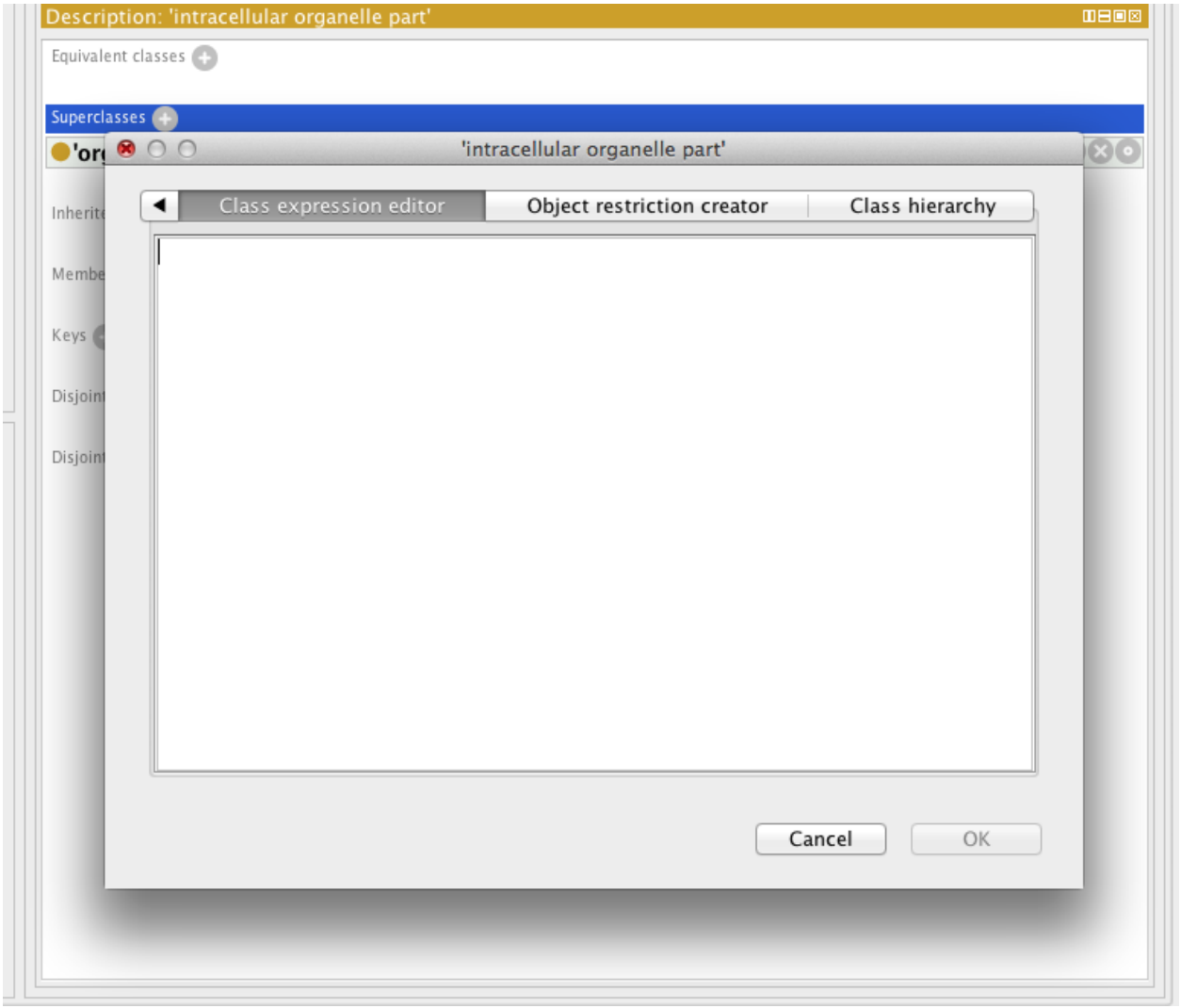
Class description view

We have seen how to add subclasses in the class hierarchy. Another way to do the same thing is via the Class description view.

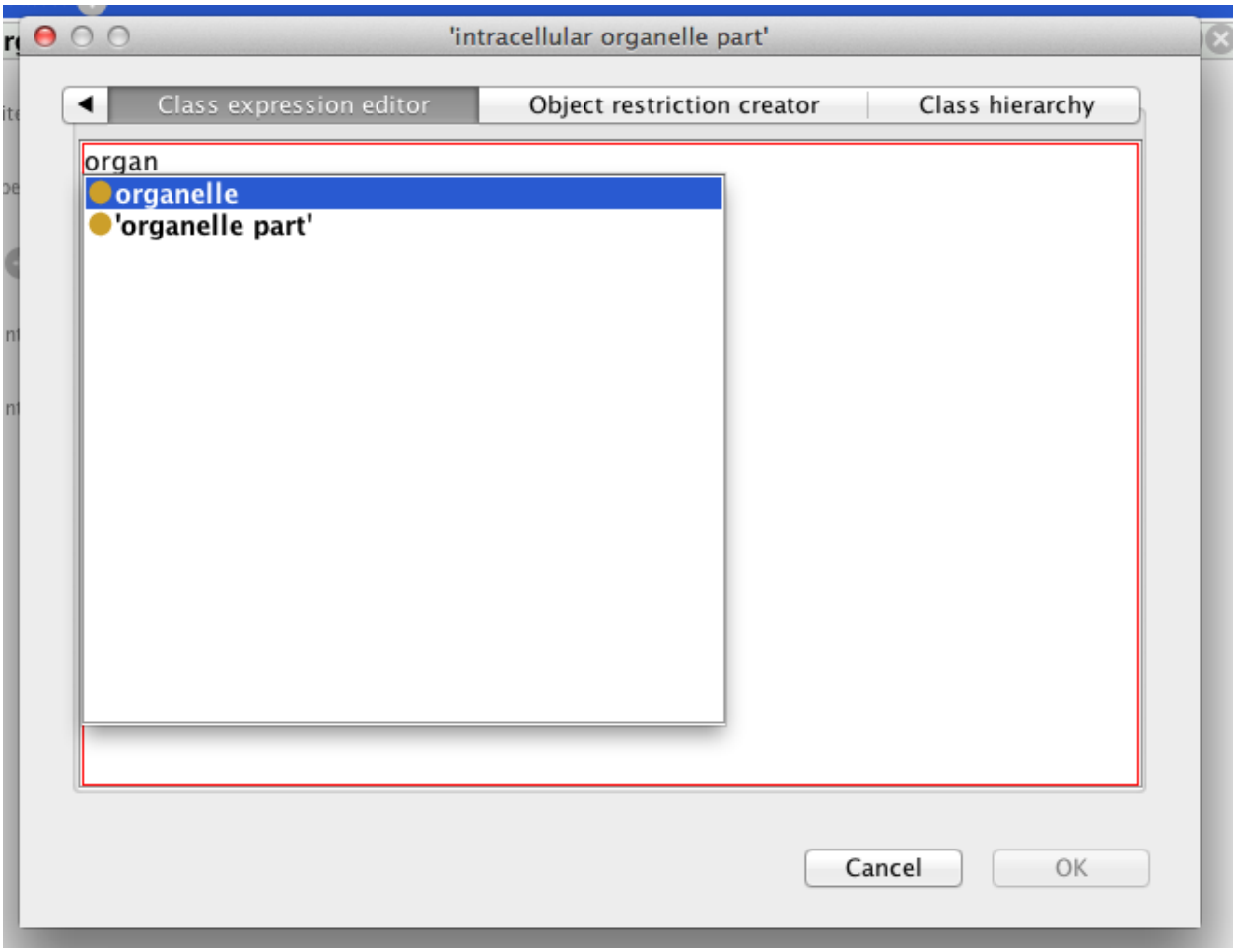
When an OWL class is selected in the entities view, the right hand side of the tab shows the class description panel. If we select the cell class we see in the class description view that this class has a superclass (cellular_component). Using the + button we could add more superclasses to the cell class.



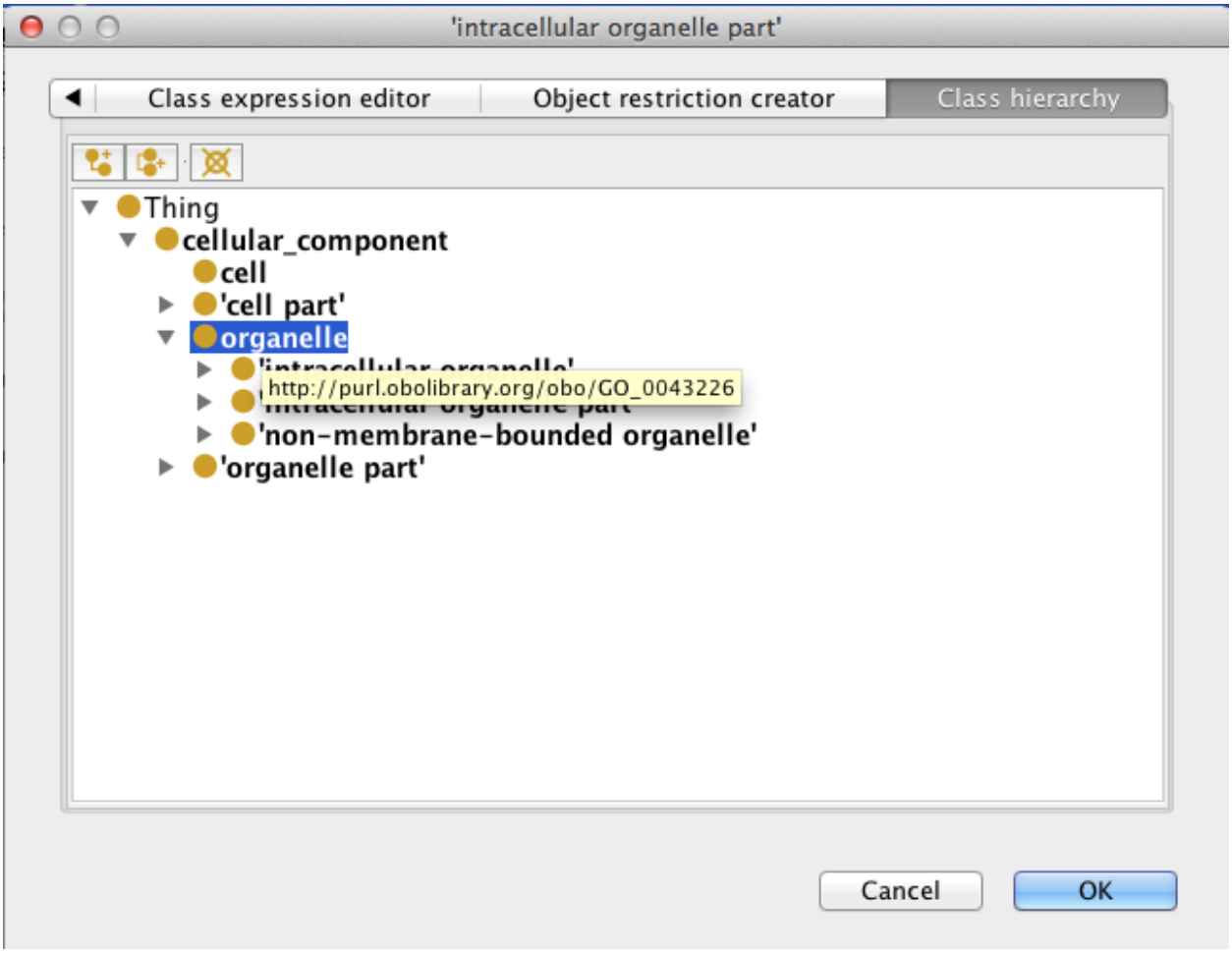
Select the ‘intracellular organelle part’ class in your ontology. Using the superclasses + button, add the ‘organelle’ class as a super class.



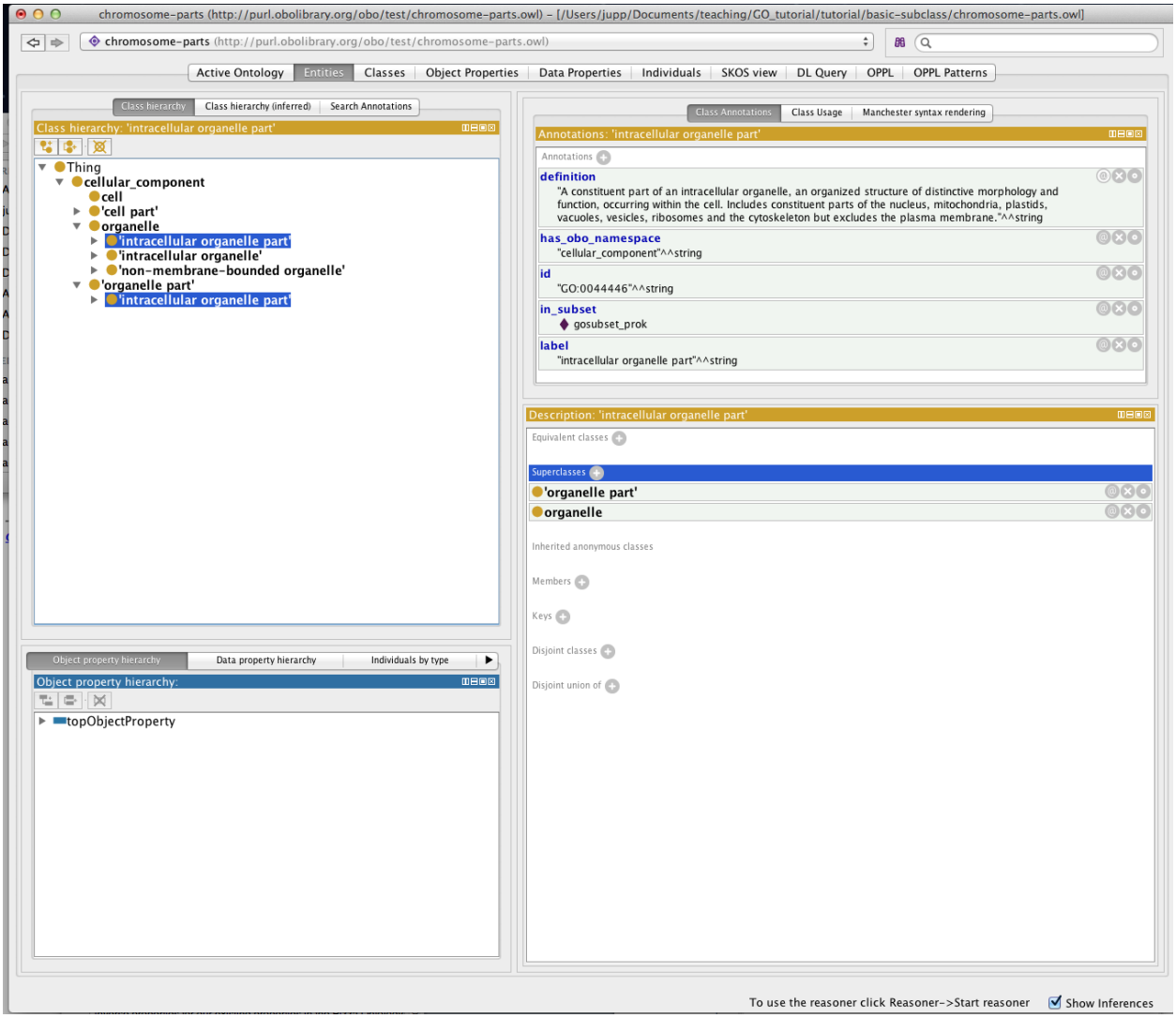
There are various ways to assert a superclass. The simplest it to just type in the class expression editor. Hint: Pressing CTRL + SPACE allows you to autocomplete on a term.



You can also use the class hierarchy tab here to search, browse and select the appropriate class.



The 'intracellular organelle part' class will now have two parents asserted in the class hierarchy. You will also be able to see both parents in the class description view.

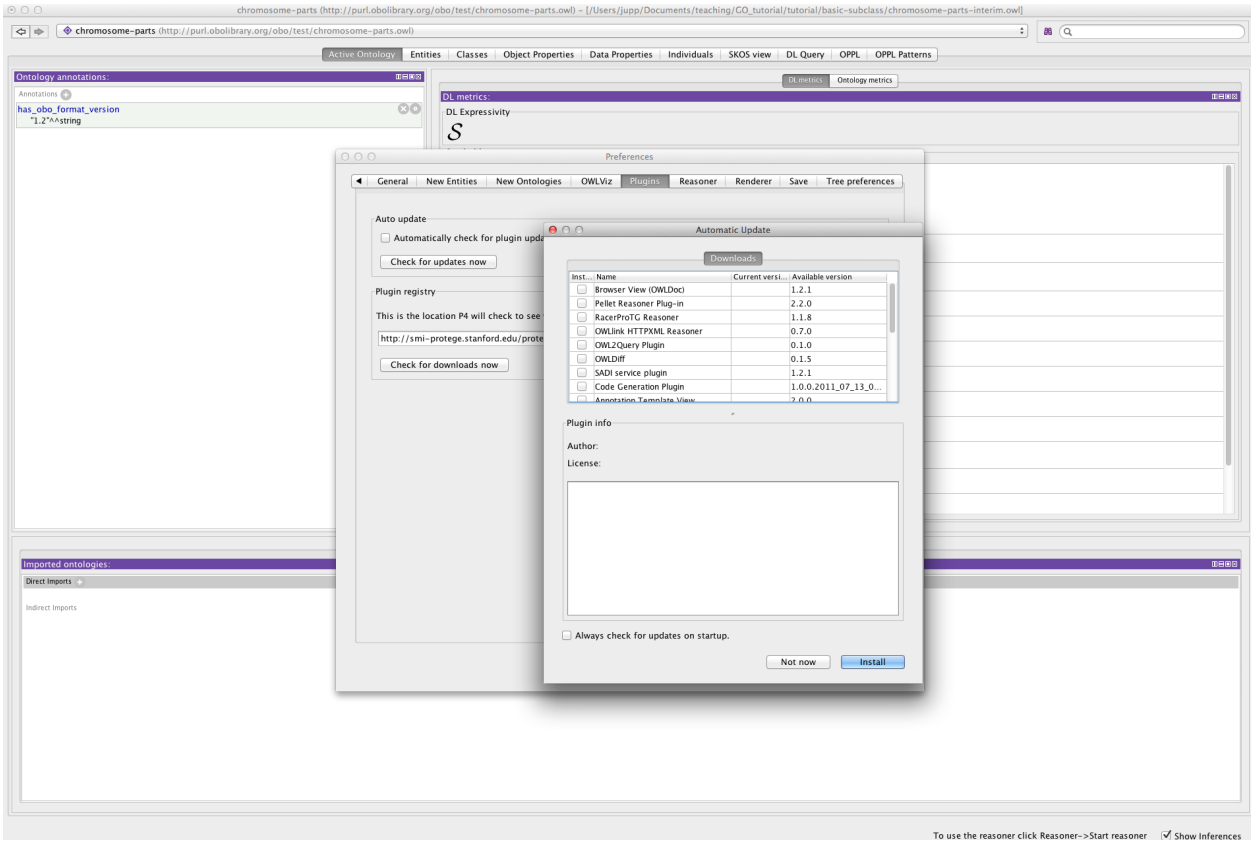

chromosome-parts (http://purl.obolibrary.org/obo/test/chromosome-parts.owl) - [/Users/jupp/Documents/teaching/GO_tutorial/tutorial/basic-subclass/chromosome-parts.owl]
chromosome-parts (http://purl.obolibrary.org/obo/test/chromosome-parts.owl)
Active Ontology
Entities
Classes
Object Properties
Data Properties
Individuals
SKOS view
DL Query
OPPL
OPPL Patterns
Class hierarchy
Class hierarchy (inferred)
Search Annotations
Class hierarchy: 'intracellular organelle part'
Thing
cellular_component
cell
'cell part'
organelle
'intracellular organelle part'
'intracellular organelle'
'non-membrane-bounded organelle'
'organelle part'
'intracellular organelle part'
Class Annotations
Class Usage
Manchester syntax rendering
Annotations: 'intracellular organelle part'
Annotations
definition
"A constituent part of an intracellular organelle, an organized structure of distinctive morphology and function, occurring within the cell. Includes constituent parts of the nucleus, mitochondria, plastids, vacuoles, vesicles, ribosomes and the cytoskeleton but excludes the plasma membrane."^^string
has_obo_namespace
"cellular_component"^^string
id
"GO:0044446"^^string
in_subset
gosubset_prok
label
"intracellular organelle part"^^string
Description: 'intracellular organelle part'
Equivalent classes
Superclasses
'organelle part'
organelle
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
Object property hierarchy
Data property hierarchy
Individuals by type
Object property hierarchy:
topObjectProperty
To use the reasoner click Reasoner->Start reasoner
Show Inferences

CHAPTER 5

Protégé plugins

Protégé is built on a plugin architecture. There is an activity community of developers writing plugin extensions to Protégé. There is a plugin library in Protégé that allows you to pick and install plugins. You may also find plugins elsewhere on the web that must be installed manually.

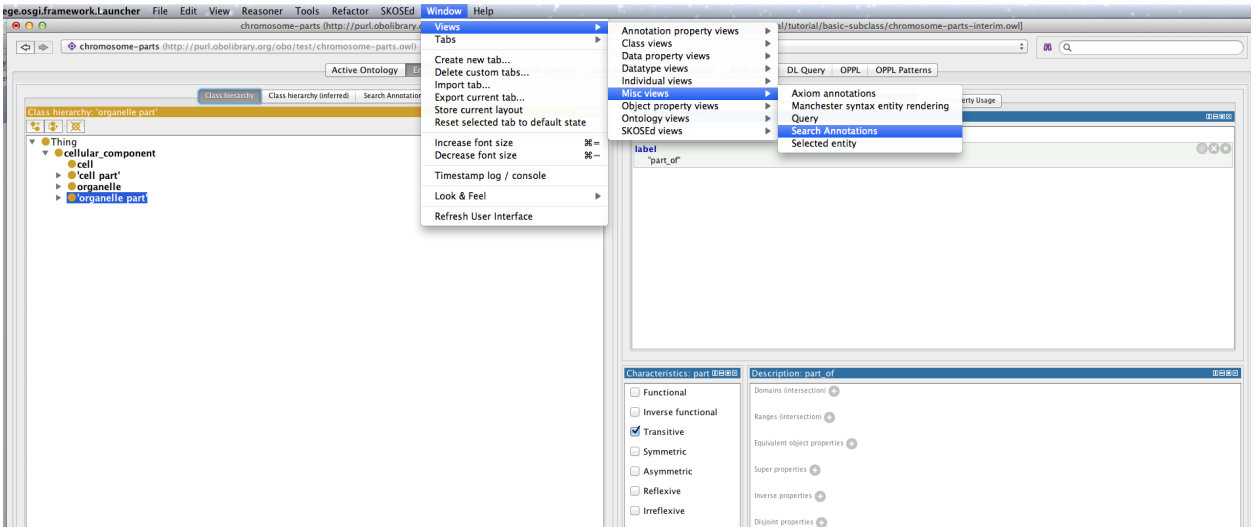
You can find the plugin library in the Protégé preferences. Select the check for downloads button to see the list of available plugins.



Install the Annotation Search View and the Existential tree view

Annotation search plugin

Most plugins are either tabs, panels or menu items. The annotations search plugin provides a new panel that can be used to search through OWL annotations (such as labels and definitions). Tabs and panels can be found in the Window menu. Under Window -> Views -> Misc views -> annotation search. Once selected you can choose to drop this panel over any existing panel in Protégé. We recommend that you drop this panel to the right of the class hierarchy view, on top of the existing annotation view panel.



You can use the annotation search panel to search through all annotation, or restrict it to individual annotations, such as the label. The annotation view also supports regular expression queries.

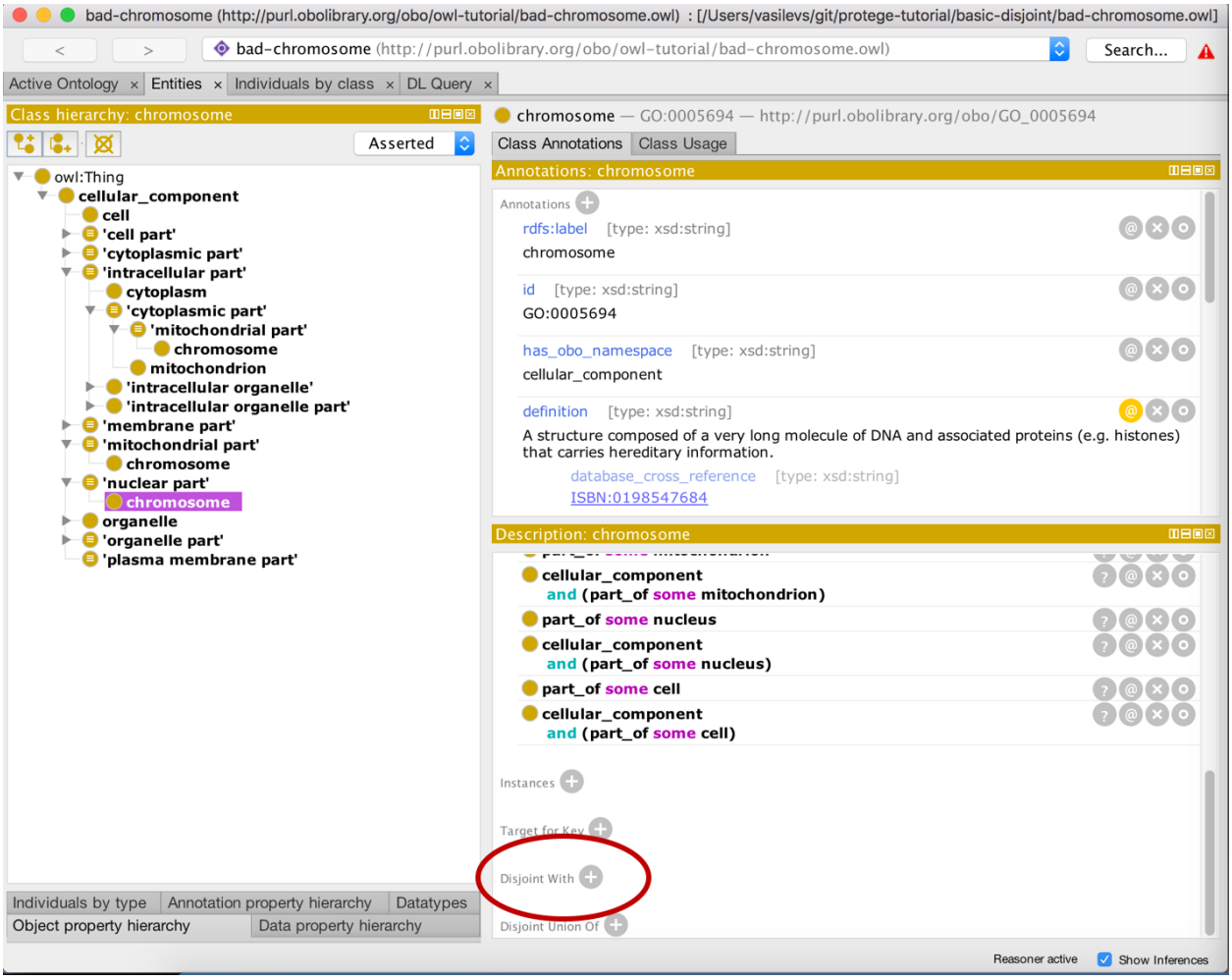
CHAPTER 6

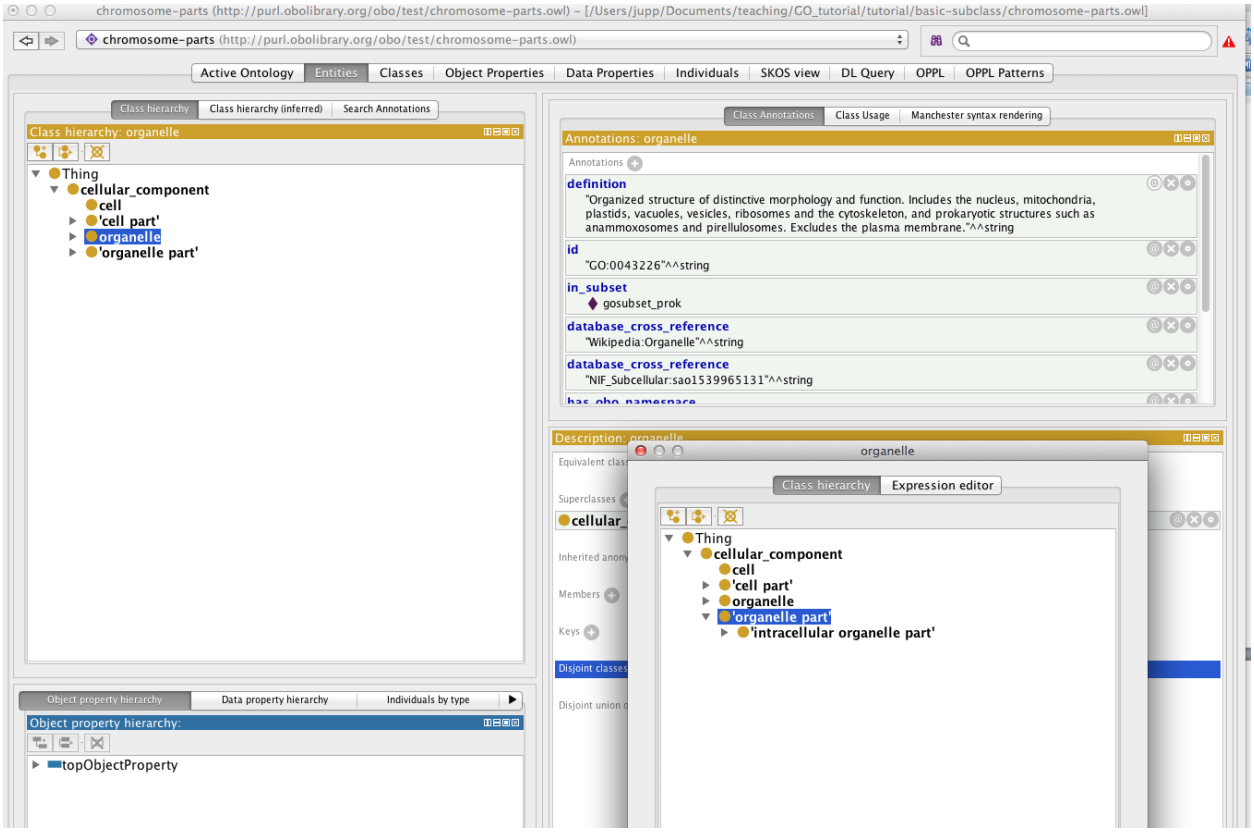
Disjointness

[the instructors will describe the concept of disjoint classes here]

At the top of our class hierarchy we have cell, cell part, organelle and organelle part. By default OWL assumes that these classes can overlap, i.e. there are individuals who can be instances of more than one of these classes. We want to create a restriction on our ontology that states these classes are different and that no individual can be a member of more than one of these classes. We can say this in OWL by creating a *disjoint classes* axiom.

We want to assert that organelle and 'organelle part' are disjoint. To do this first select the cell class. In the class description view select the + button next to disjoints. You can use CTRL to select multiple classes.


bad-chromosome (http://purl.obolibrary.org/obo/owl-tutorial/bad-chromosome.owl) : [/Users/vasilevs/git/protege-tutorial/basic-disjoint/bad-chromosome.owl]
bad-chromosome (http://purl.obolibrary.org/obo/owl-tutorial/bad-chromosome.owl)
Search...
Active Ontology
Entities
Individuals by class
DL Query
Class hierarchy: chromosome
Asserted
owl:Thing
cellular_component
cell
'cell part'
'cytoplasmic part'
'intracellular part'
cytoplasm
'cytoplasmic part'
'mitochondrial part'
chromosome
mitochondrion
'intracellular organelle'
'intracellular organelle part'
'membrane part'
'mitochondrial part'
chromosome
'nuclear part'
chromosome
organelle
'organelle part'
'plasma membrane part'
Individuals by type
Annotation property hierarchy
Datatypes
Object property hierarchy
Data property hierarchy
chromosome — GO:0005694 — http://purl.obolibrary.org/obo/GO_0005694
Class Annotations
Class Usage
Annotations: chromosome
Annotations
rdfs:label [type: xsd:string]
chromosome
id [type: xsd:string]
GO:0005694
has_obo_namespace [type: xsd:string]
cellular_component
definition [type: xsd:string]
A structure composed of a very long molecule of DNA and associated proteins (e.g. histones) that carries hereditary information.
database_cross_reference [type: xsd:string]
ISBN:0198547684
Description: chromosome
cellular_component
and (part_of some mitochondrion)
part_of some nucleus
cellular_component
and (part_of some nucleus)
part_of some cell
cellular_component
and (part_of some cell)
Instances
Target for Key
Disjoint With
Disjoint Union Of
Reasoner active
Show Inferences

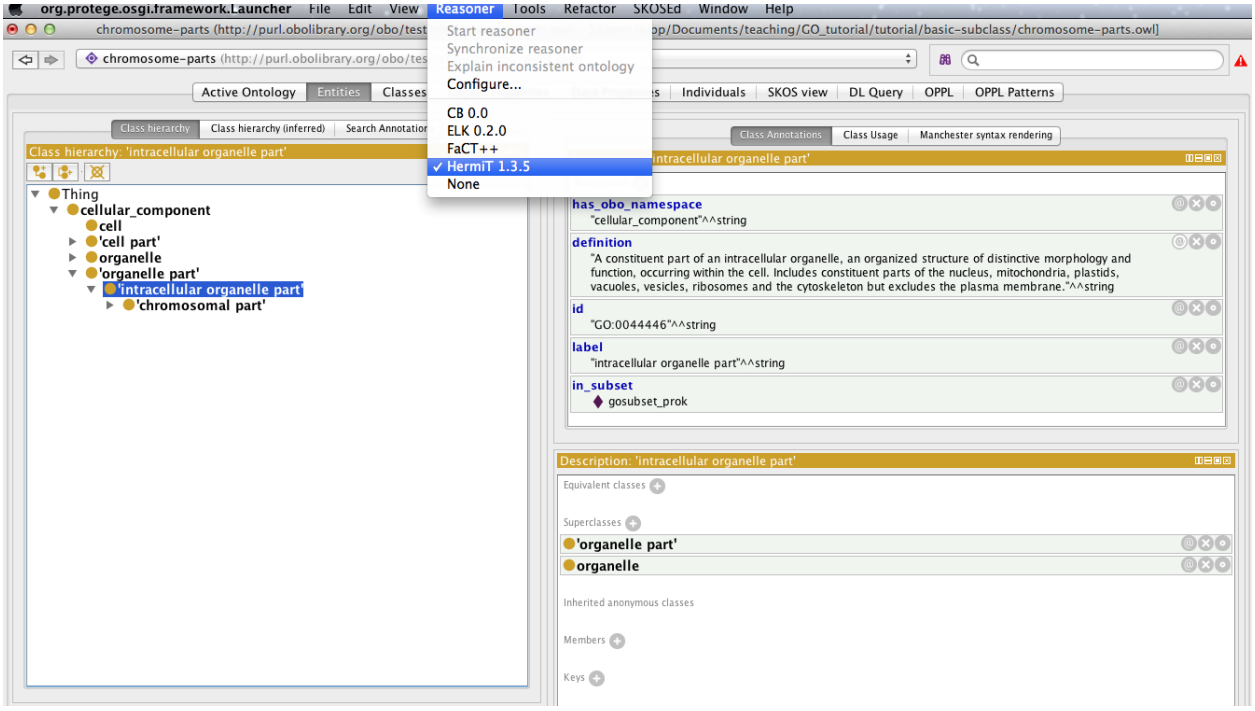


Note that the directionality is irrelevant.

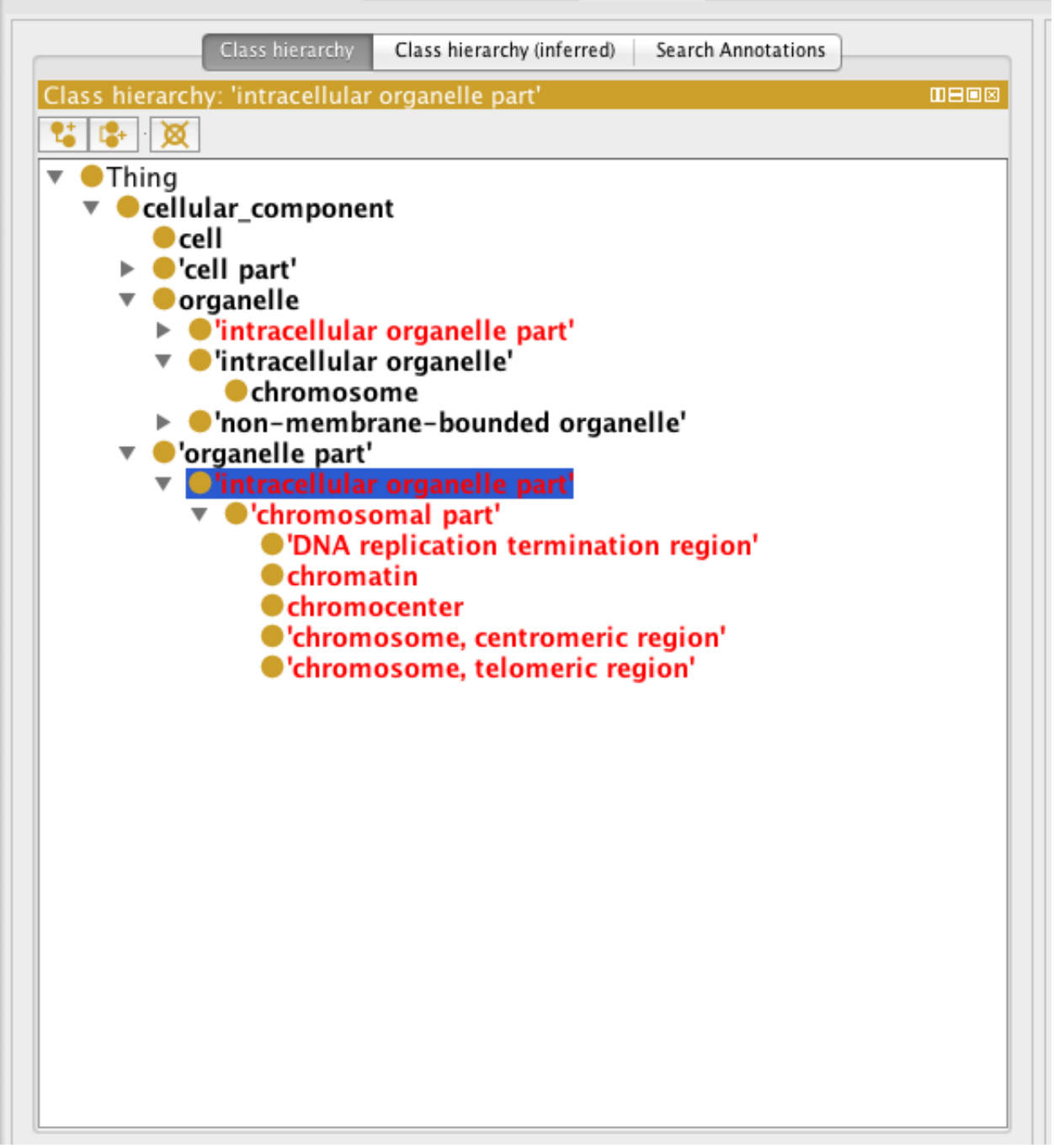
Reasoning and inconsistency checking

We have introduced a deliberate mistake into the ontology. We previously asserted that ‘intracellular organelle part’ is a subclass of both ‘organelle part’ and ‘organelle’. We have now added an axiom stating that ‘organelle’ and ‘organelle part’ are all disjoint. We can use the reasoner to check the consistency of our ontology. The reasoner should detect our contradiction.

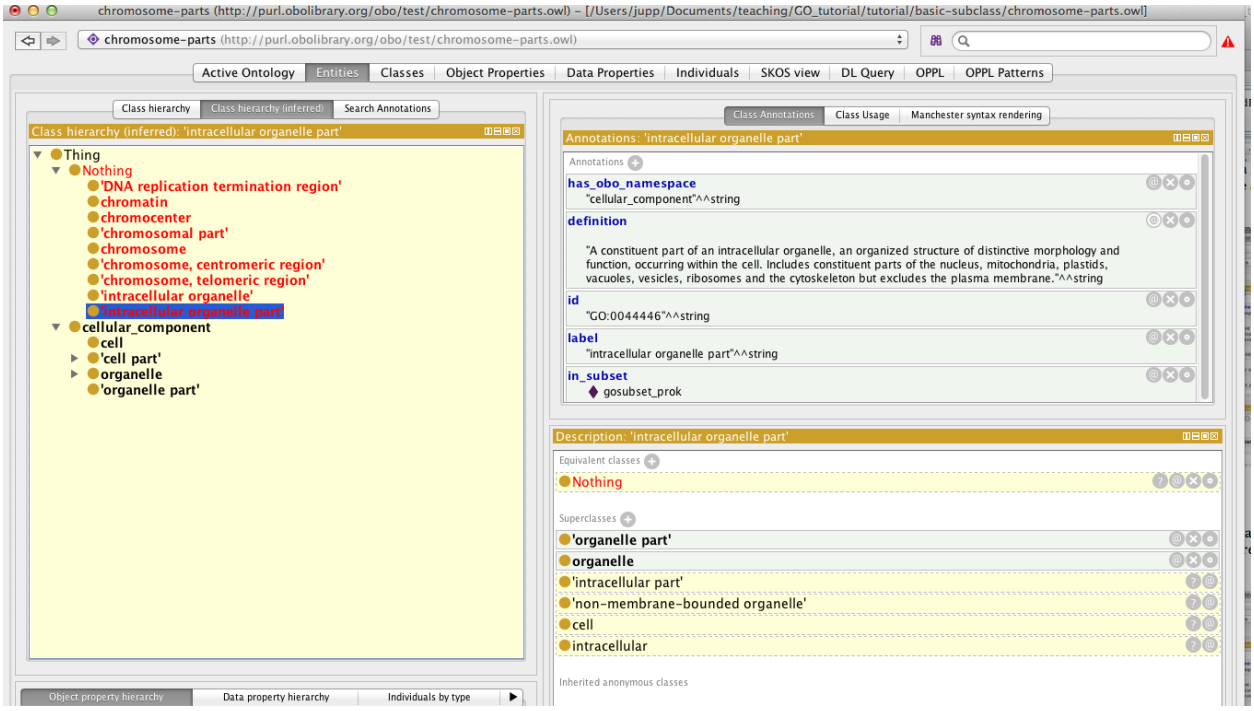
Protégé comes with several reasoners, and more can be installed via the plugins mechanism (see plugins chapter). Select a reasoner from the Reasoner menu (HermiT, Pellet or Fact++ will work). Once a reasoner is highlighted, select “Start reasoner” from the menu.



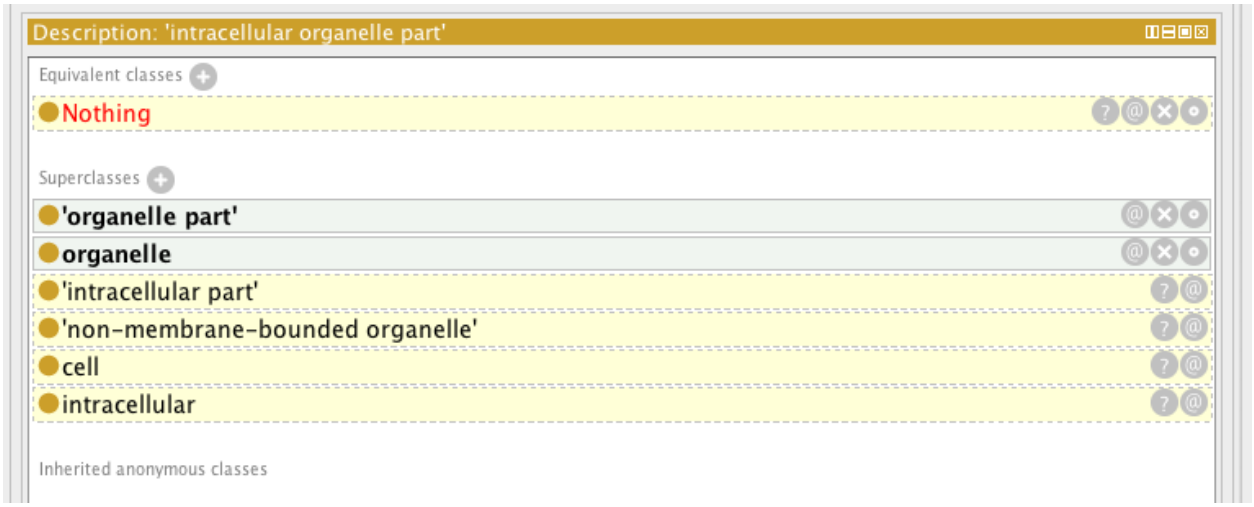
A progress bar will indicate when classification is complete. The 'intracellular organelle part' class will have changed to red indicating that the class is now *unsatisfiable*.



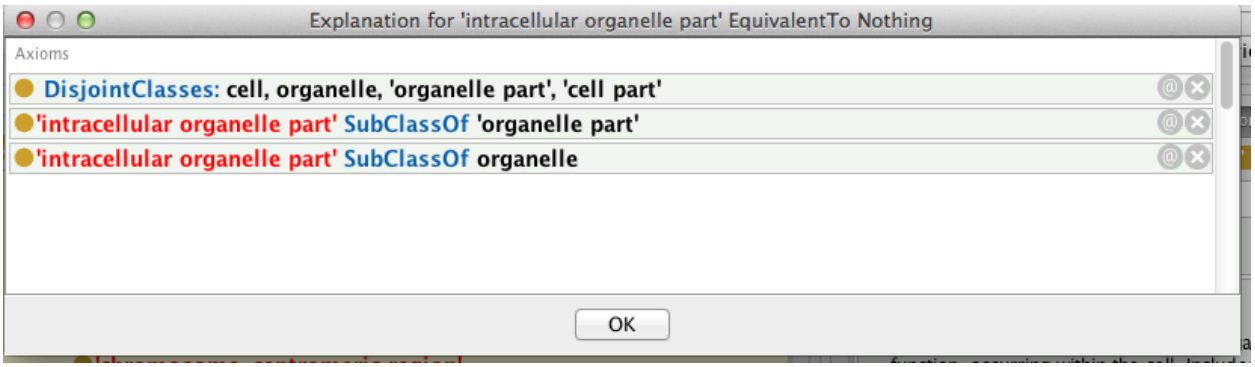
You can also see unsatisfiable classes using the "Class hierarchy (inferred)" panel next to the "Class hierarchy" panel. (Note - in Protege 5.1, the toggle the "Asserted button" to switch to the inferred view.) Here you will a special class called Nothing. When we previously said that all OWL classes are sublasses of OWL Thing. OWL Nothing is a leaf class or bottom class of your ontology. Any classes that are deemed unsatisfiable by the reasoner are shown as sublasses or equivalent to OWL Nothing. The "class hierarchy (inferred)" view will show you all subclasses of Nothing.



Once the ontology is classified inferred statements or axioms are shown in the various panels with a light yellow shading. The class description for 'intracellular organelle part' should look something like the following screen shot. You will see that the class has been asserted equivalent to the Nothing class. Inside this statement a small question mark icon appears, clicking this will get an explanation from the reasoner for this inconsistency.



Select the ? icon to get an explanation for this inconsistency. The explanation shows the axioms involved. We see the disjoint class axiom alongside the two subclass axioms are causing the inconsistency. We can simply repair this ontology by removing the 'intracellular organelle part' subClassOf 'organelle' axiom.



Remove the axiom, and resynchronise the reasoner from the reasoner menu.

EXERCISE: Basic Disjoint

Go to the basic-disjoint folder in the tutorial directory and follow the instructions in the README

CHAPTER 7

Making Ontologies from Spreadsheets and templates (Optional)

Go to the template-examples folder

Open “test1.csv” in Excel. This has a sample set of plant traits

A1 iri

	A	B	C	D	E	F	G
1	iri	label	entity	entity label	quality	quality label	
2	DEMOTRAIT_0000002	seed morphology	PO_0009010	seed	PATO_0000051	morphology	
3	DEMOTRAIT_0000003	seed shape	PO_0009010	seed	PATO_0000052	shape	
4	DEMOTRAIT_0000004	seed size	PO_0009010	seed	PATO_0000117	size	
5	DEMOTRAIT_0000005	seed weight	PO_0009010	seed	PATO_0000128	weight	
6	DEMOTRAIT_0000007	morphology	PO_0009011	plant structure	PATO_0000051	morphology	
7	DEMOTRAIT_0000008	shape	PO_0009011	plant structure	PATO_0000052	shape	
8	DEMOTRAIT_0000009	size	PO_0009011	plant structure	PATO_0000117	size	
9	DEMOTRAIT_0000010	weight	PO_0009011	plant structure	PATO_0000128	weight	
10	DEMOTRAIT_0000012	leaf morphology	PO_0009025	vascular leaf	PATO_0000051	morphology	
11	DEMOTRAIT_0000013	leaf shape	PO_0009025	vascular leaf	PATO_0000052	shape	
12	DEMOTRAIT_0000014	leaf size	PO_0009025	vascular leaf	PATO_0000117	size	
13	DEMOTRAIT_0000015	leaf weight	PO_0009025	vascular leaf	PATO_0000128	weight	
14	DEMOTRAIT_0000017	flower morphology	PO_0009046	flower	PATO_0000051	morphology	
15	DEMOTRAIT_0000018	flower shape	PO_0009046	flower	PATO_0000052	shape	
16	DEMOTRAIT_0000019	flower size	PO_0009046	flower	PATO_0000117	size	
17	DEMOTRAIT_0000020	flower weight	PO_0009046	flower	PATO_0000128	weight	
18	DEMOTRAIT_0000022	perianth morphology	PO_0009058	perianth	PATO_0000051	morphology	
19	DEMOTRAIT_0000023	perianth shape	PO_0009058	perianth	PATO_0000052	shape	
20	DEMOTRAIT_0000024	perianth size	PO_0009058	perianth	PATO_0000117	size	
21	DEMOTRAIT_0000025	perianth weight	PO_0009058	perianth	PATO_0000128	weight	
22							

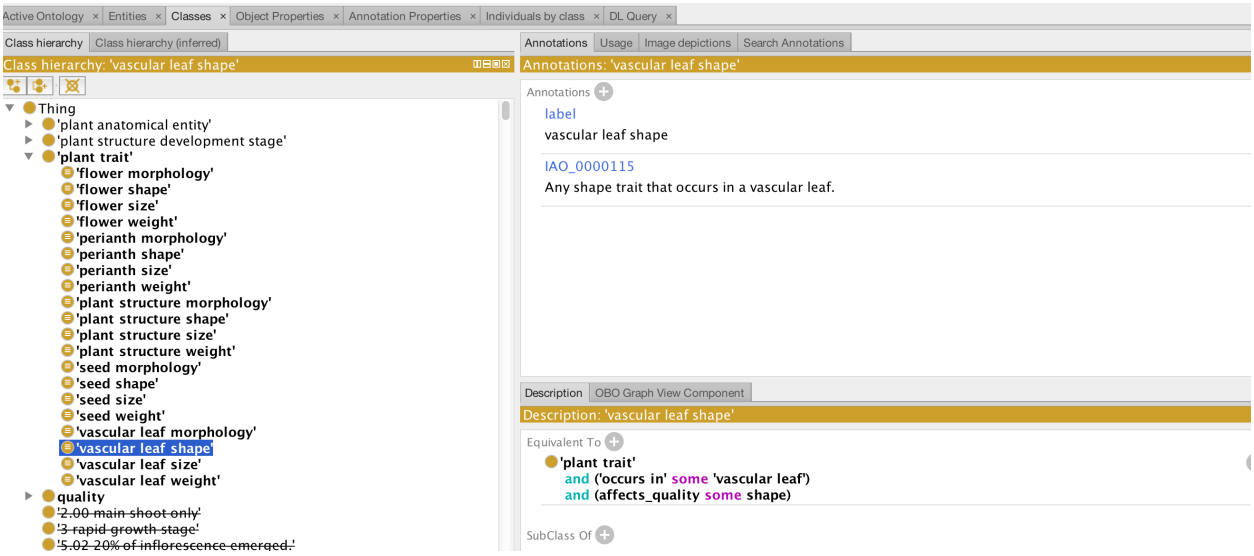
If you have python installed, you can convert this to OWL:

./apply-pattern.py -p simple_measurable.yaml –i test1.csv > test1.owl

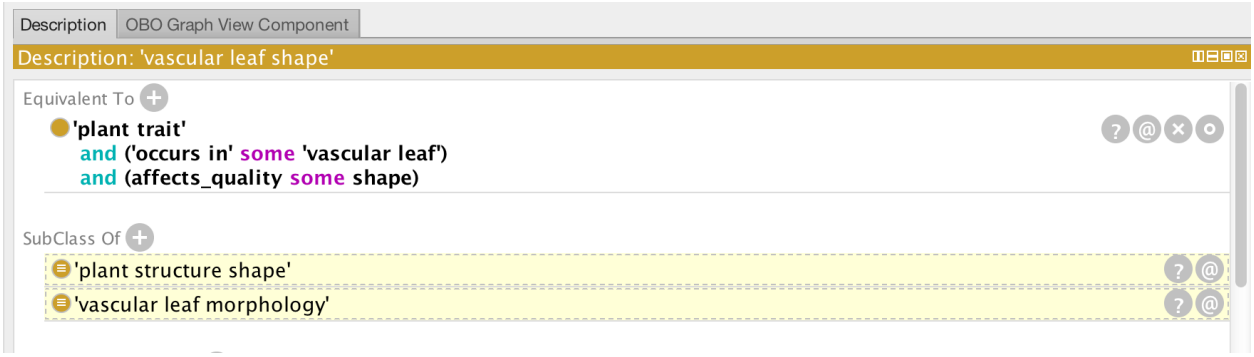
Otherwise, just use the existing “test1.owl” file in the directory.

Open this in Protégé and look under plant trait. You should see a flat list of terms. You can navigate to one and look at the description tab. As you can see, the values from the spreadsheet have been slotted in as specific object property

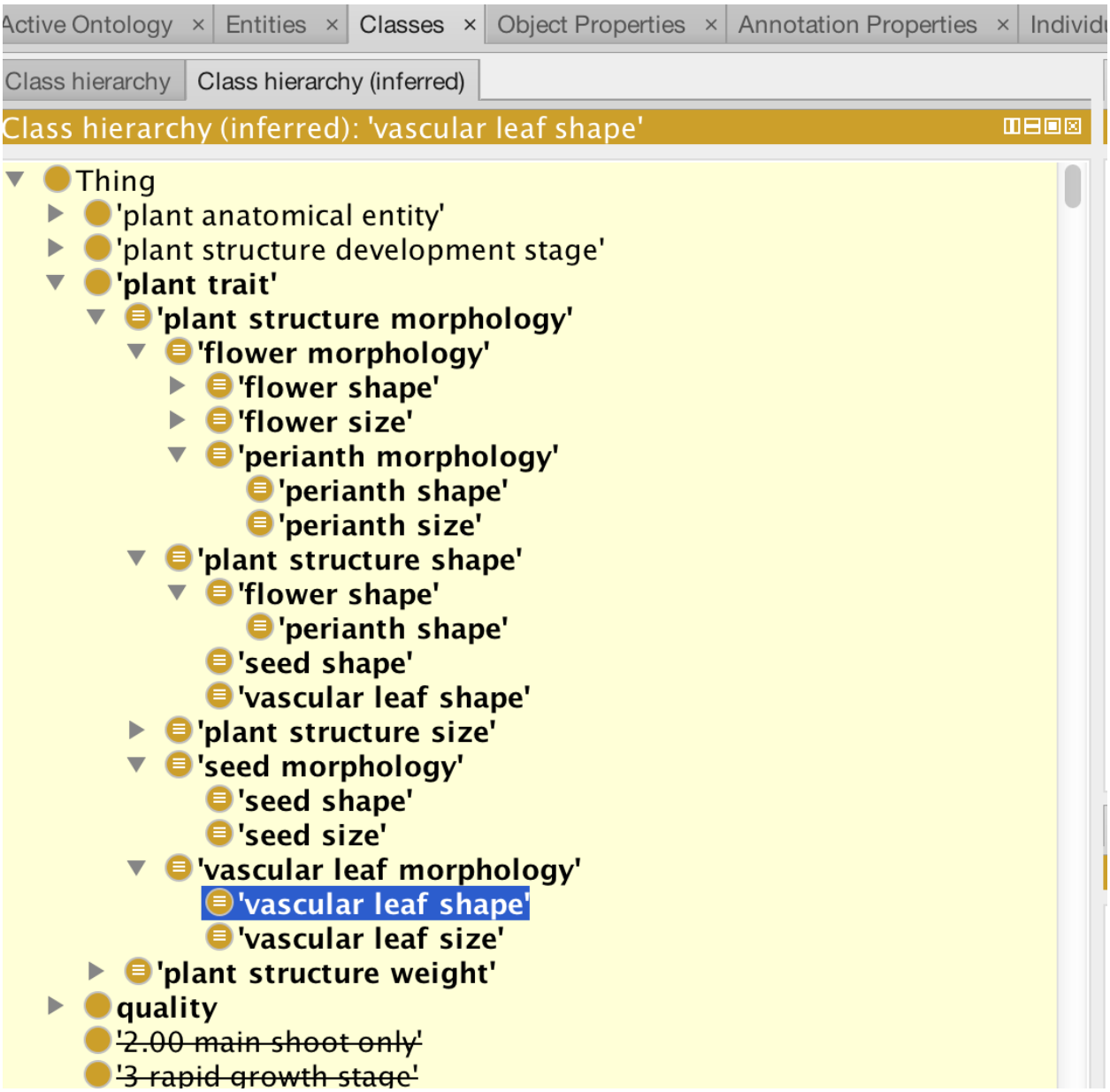
restrictions:



Next, turn on the reasoner. You will see superclasses show up in yellow. These are **inferred superclasses**.



Navigate to “Class hierarchy (inferred)”



You will see the entire polyhierarchy is inferred automatically

CHAPTER 8

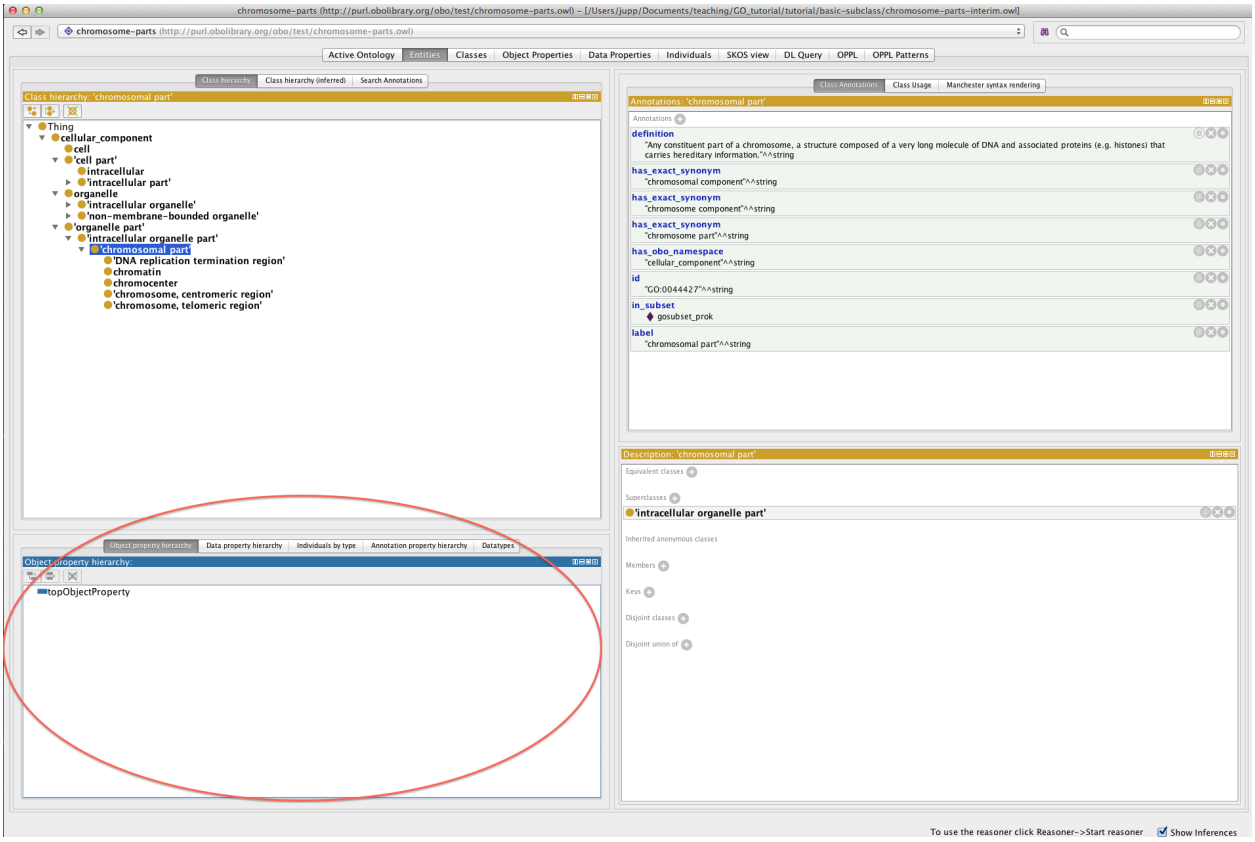
Object properties

At this point load: chromosome-parts-interim.owl form the tutorial directory 'basic-restriction'

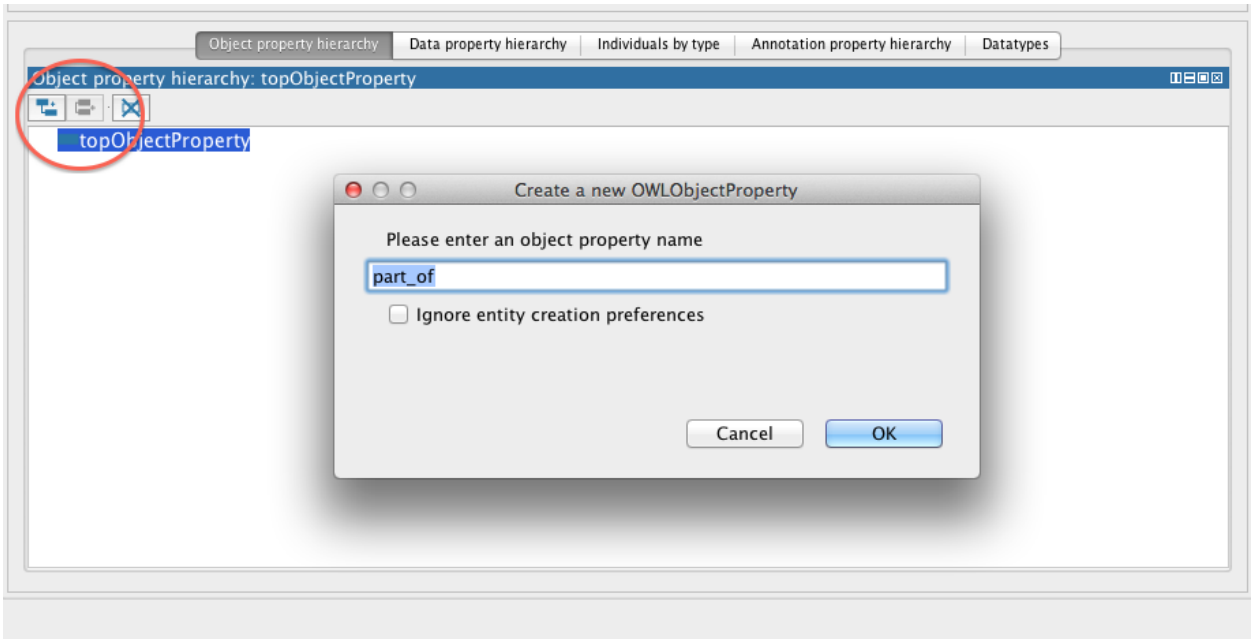
We will now create an object property and use this to add some restriction onto classes. In OWL properties are used to assert relationships between individuals (or instance). Properties in OWL can have characteristics such as being transitive or symmetric. We can assert additional information about properties such their domain and range, along with defining inverse properties.

Create an object property

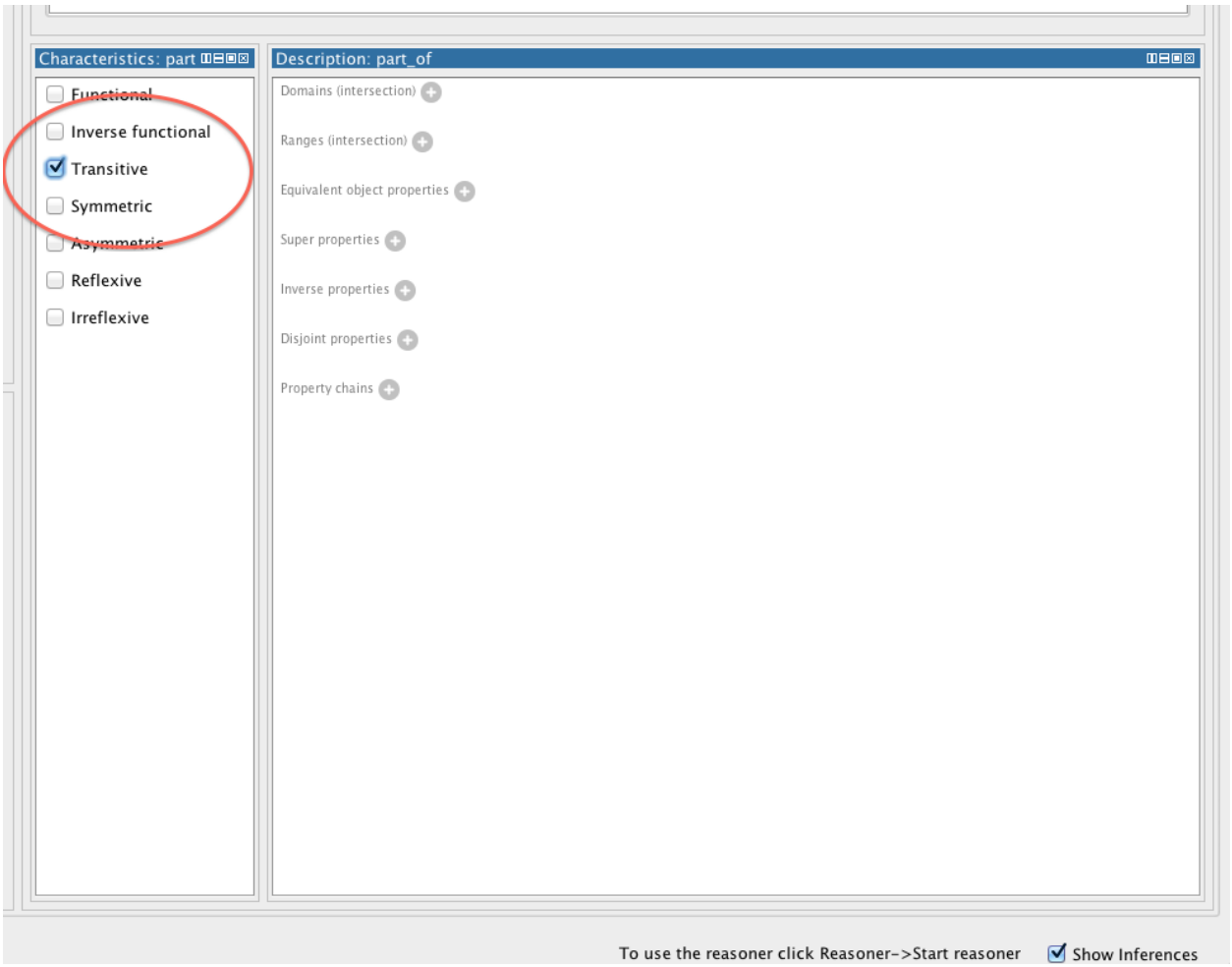
We will use the object property view circled below to create a part_of property. In OWL all properties are a sub property of topObjectProperty.



Select the "add sub property button" circled below and name the property part_of.



We can use the property description view shown below to make assertions about this property. We want to state that the part_of property has the characteristic of being transitive. If a property is transitive, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P. A good example of a transitive property is the geneological 'ancestor of' relationship. We can make a property transitive in Protege by simply selecting the transitive check box.


Characteristics: part
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: part_of
Domains (intersection)
Ranges (intersection)
Equivalent object properties
Super properties
Inverse properties
Disjoint properties
Property chains
To use the reasoner click Reasoner->Start reasoner
Show Inferences

CHAPTER 9

OWL class restrictions

As previously stated, in OWL we use object property to describe binary relationships between two individuals (or instances). We can also use the properties to describe new classes (or sets of individuals) using *restrictions*. A restriction describes a class of individuals based on the relationships that members of the class participate in. In other words a restriction is a kind of class, in the same way that a named class is a kind of class.

For example, we can use a named class to capture all the individuals that are chromosome parts. But we could also describe the class of chromosome parts as all the instances that are '*part of'* a chromosome.

In OWL there are three main types of restrictions that can be placed on classes. These are **quantifier restriction**, **cardinality restrictions** and **hasValue** restriction. In this tutorial will initially focus on quantifier restrictions.

Quantifier restriction are further categorised into two types, the **existential** and the **universal** restriction.

- **Existential** restrictions describe classes of individuals that participate in at least one relationship along a specified property to individuals that are members of a specified class. For example, "the class of individuals that have at least one (some) 'part of' relationship to members of the 'Chromosome class". In Protégé 4 the keyword 'some' is used to denote existential restrictions.
- **Universal** restrictions describe classes of individuals that for a given property only have relationships along this property to individuals that are members of a specified class. For example, we can say a cellular component is capable of many functions using the existential quantifier, however, OWL semantics assume that there could be more. We can use the universal quantifier to add closure to the existential. That is we can assert that a cellular component is capable of these function, and is only capable of those function and no other. Another example is that the process of hair growth is found **only** in instances of the class Mammalia. In Protégé the keyword "only" is used.

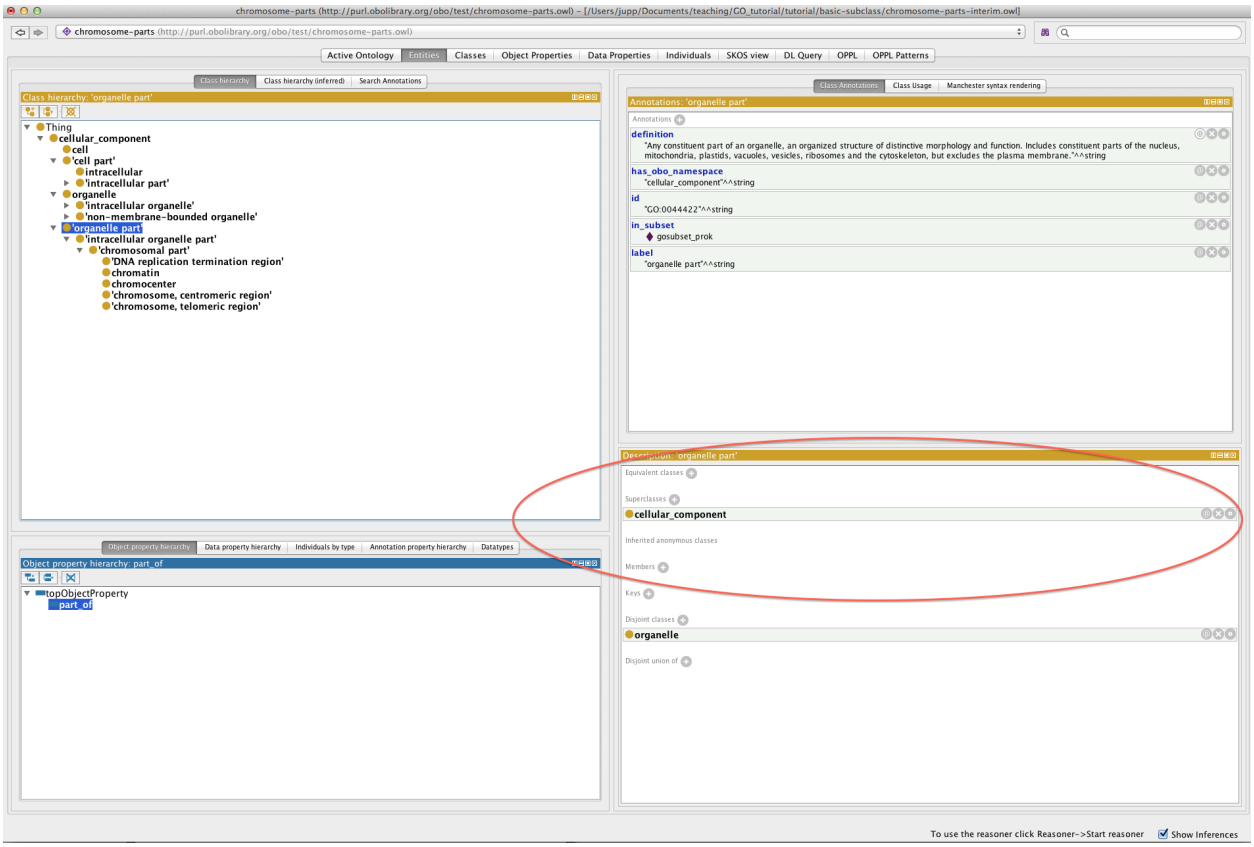
In this tutorial we will deal exclusively with the existential (some) quantifier. Note that in OBO-Format, all relationships are implicitly existentially qualified.

Superclass restrictions

In OBO-Edit you will be familiar with creating relationships between classes. Strictly speaking in OWL you don't make relationships between classes, however, using OWL restrictions we essentially achieve the same thing.

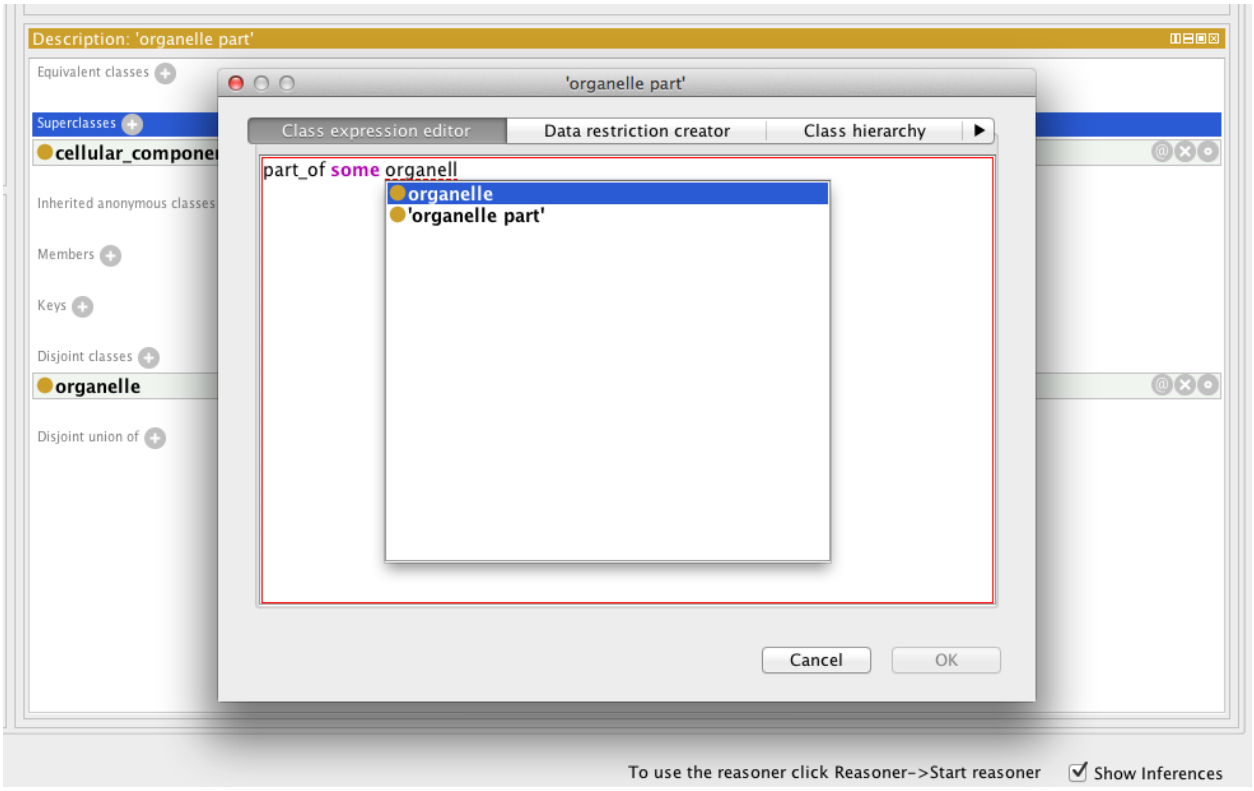
We want to capture the knowledge that the named class 'organelle part' is part of an organelle. In OWL speak, we want to say that every instance of an 'organelle part' is also an instance of the class of things that have at least one 'part of' relationship to an 'organelle'. In OWL we do this by creating an existential restriction on the 'organelle part' class.

Select 'organelle part' in the class hierarchy and look at it's current class description. At the top of this view there are two slots for defining equivalent classes and superclasses. 'organelle part' already has one superclass named cellular_component.

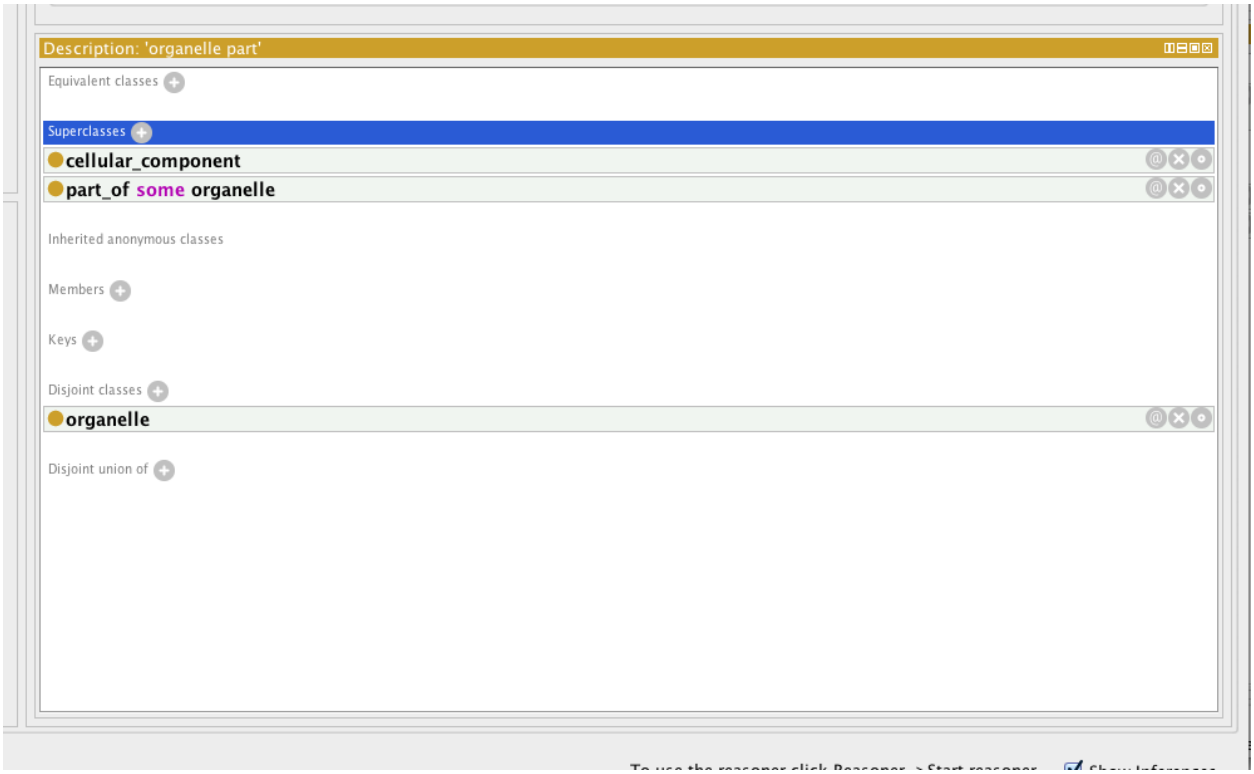


We will create a restriction on 'organelle part' stating 'organelle part' has a *'part of'* relationship to some 'organelle'. Select the + icon next to the superclasses slot. We will define this anonymous superclass in Manchester OWL syntax as

'part of' some 'organelle'.



The class restriction will be shown in the superclasses slot as follows.



Using Protégé create some of your own part_of restriction the ‘cell part’, ‘intracellular part’ and ‘chromosomal part’ classes.

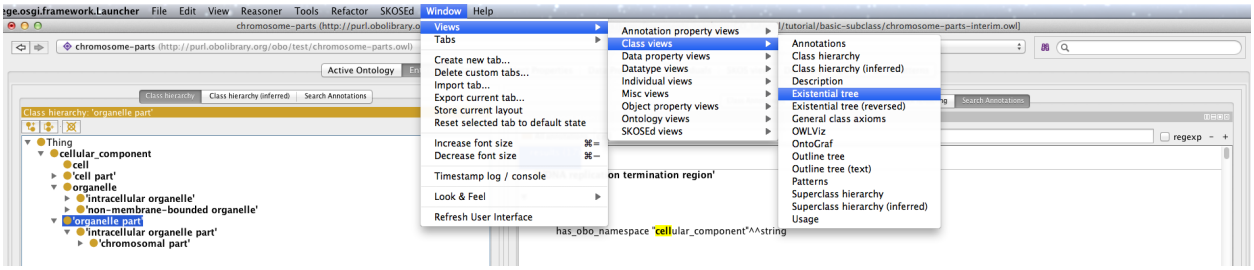
[the instructors may elucidate more on the nature of these class restrictions here]

NOTE: After each edit to the ontology you might want to synchronize the reasoner to make sure you didn't introduce any inconsistencies into your ontology.

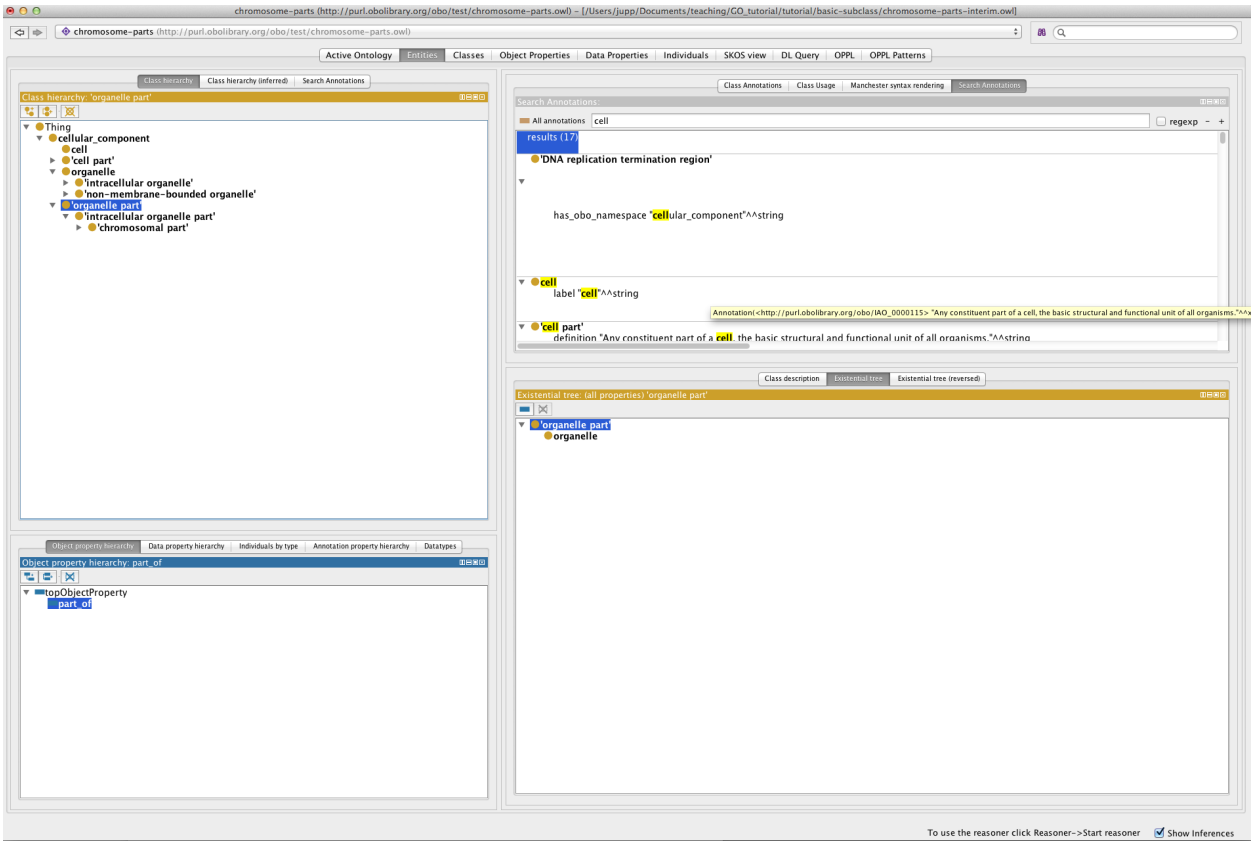
Existential tree plugin

The class hierarchy view in protégé shows subclass/superclass[2] relationships between classes. The default class hierarchy view is restricted to showing strict is-a, or sub/super class relationships. The existential tree is an alternate class hierarchy view that organise classes into hierarchies based on existential restriction. For example, viewing a partonomy along the 'part of' existential restriction.

The existential tree view can be found under Window -> Views -> Existential Tree



Drop the view over the class description view.



Select the `part of` property to render your partonomy in the existential tree panel.

EXERCISE: Basic Restrictions

Stay in the "basic-restriction" folder in the tutorial directory and follow the instructions in the README.txt

The instructors may demonstrate some of the additional options for navigating the ontology at this point:

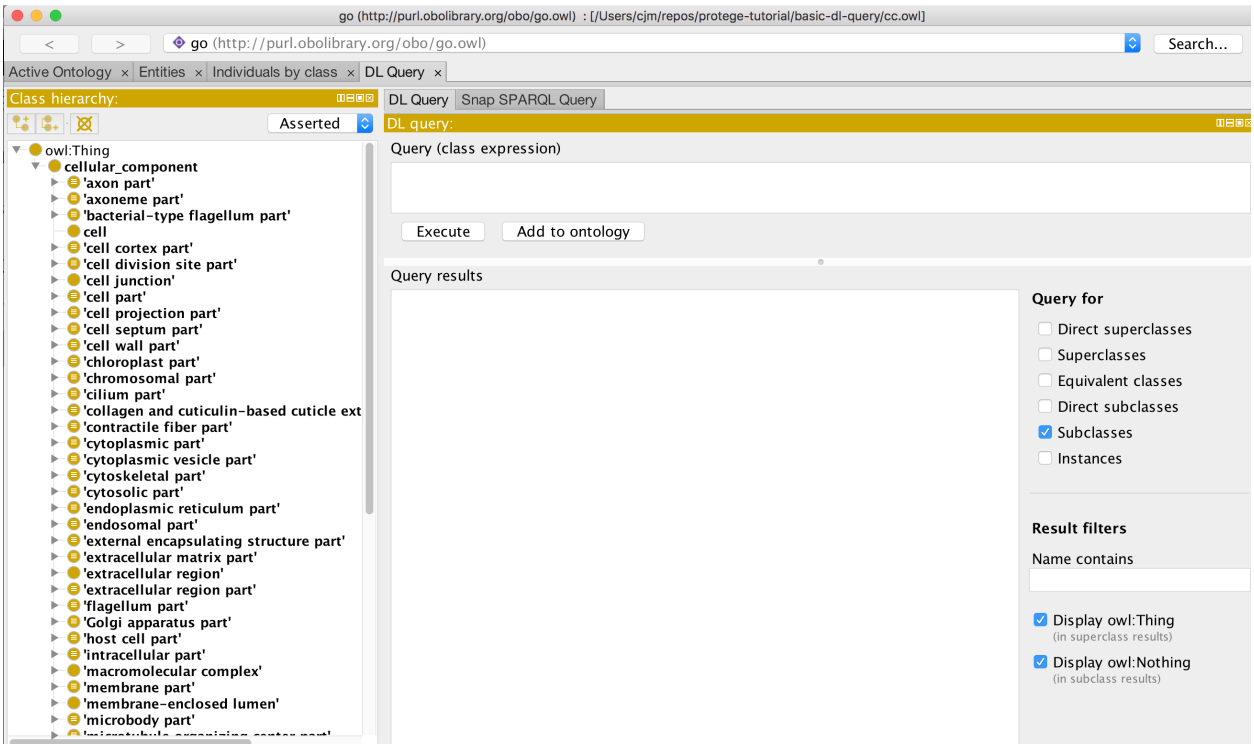
- Existential Tree Plugin
- OntoGraf

CHAPTER 10

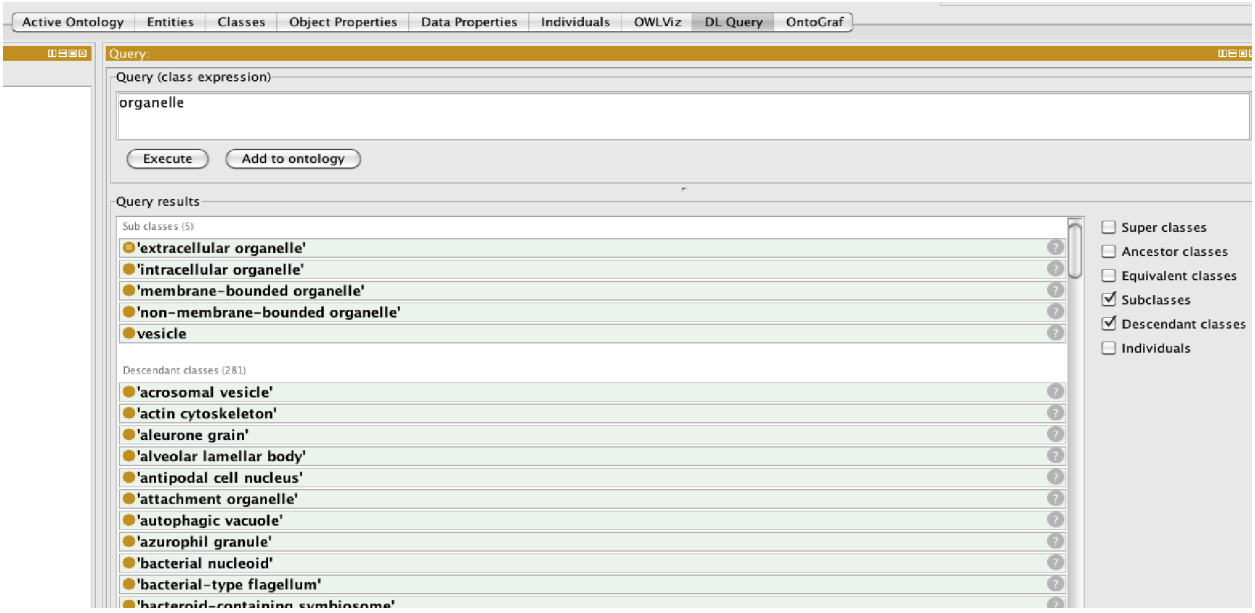
DL query tab

The DL query tab shown below provides an interface for querying and searching an ontology. The ontology must be classified by a reasoner before it can be queried in the DL query tab.

Go to the "basic-dl-query" folder and open "cc.owl". Navigate to the DL Query tab.

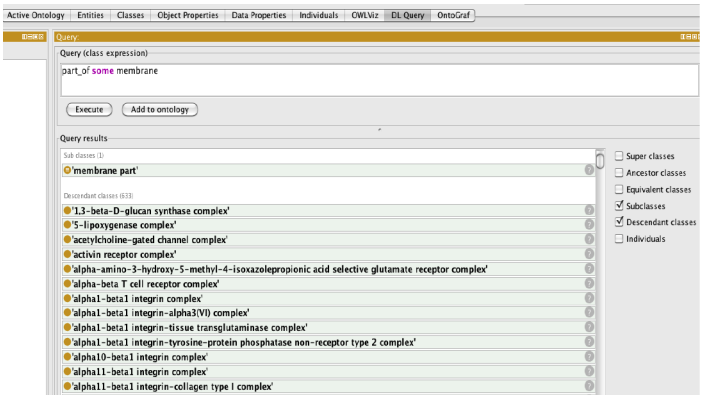


Type "organelle" into the box, and make sure "subclasses" and "descendent classes" are ticked. "subclasses" is the direct subclasses calculated by the reasoner (which may often but not always be the same as the asserted subclasses). "descendants" is all subclasses.

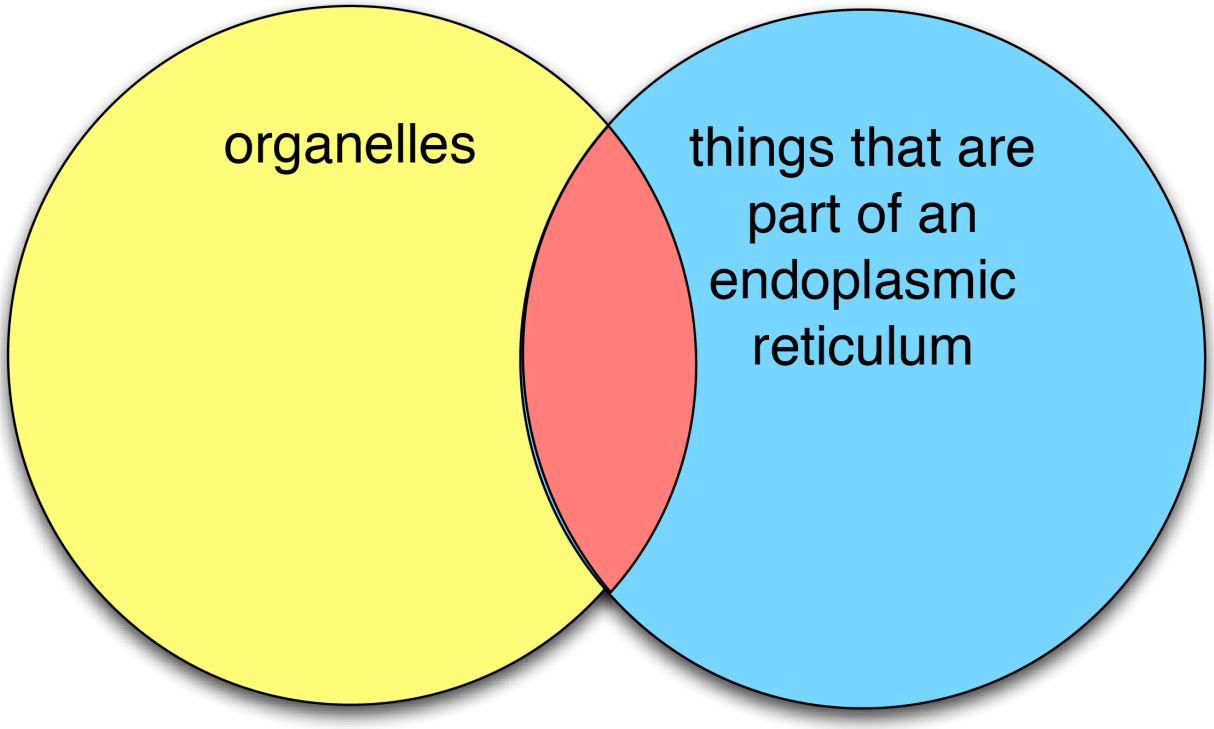


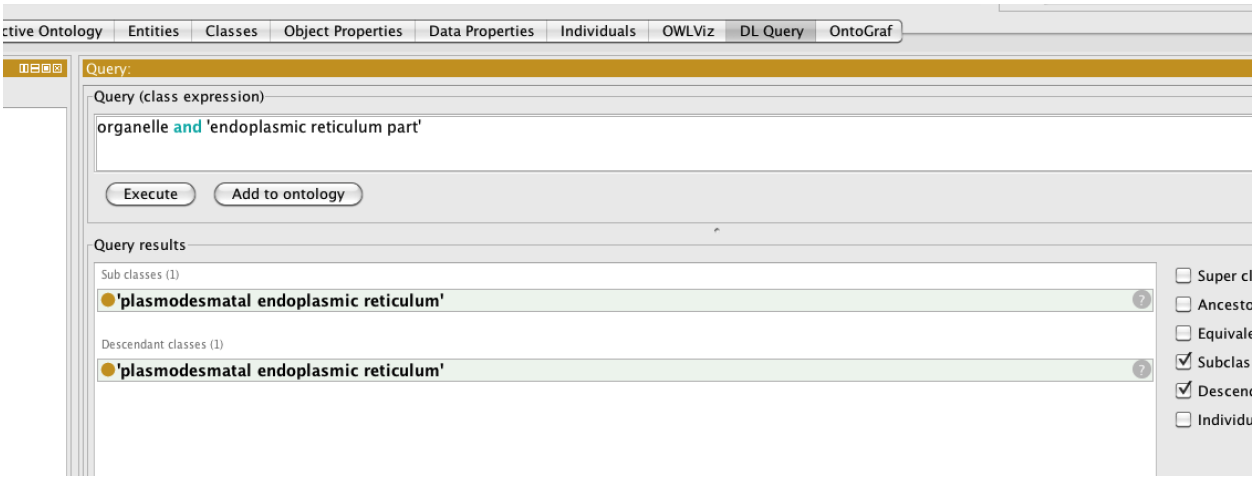
In general for GO you should never need the “individuals” box ticked.

You can type any valid OWL class expression into the DL query tab. For example, to find all classes whose members are part_of a membrane, type “part_of some membrane”.

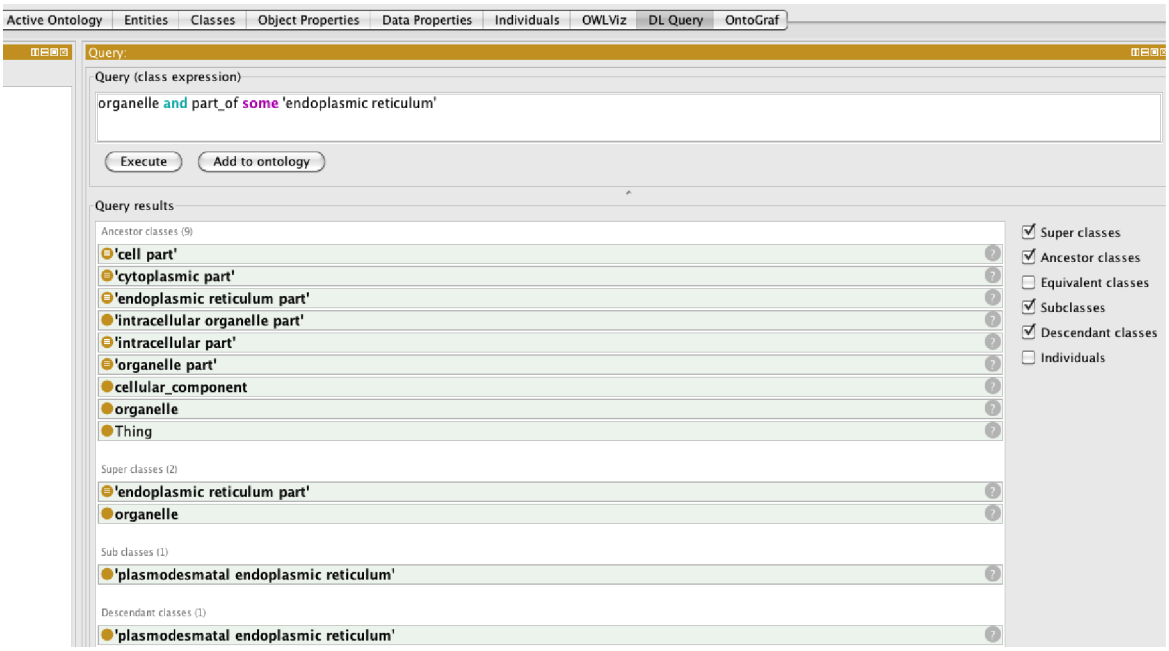


The OWL keyword “and” can be used to make a class expression that is the intersection of two class expressions. For example, to find the classes in the red area below, we want to find subclasses of the intersection of the class ‘organelle’ and the class ‘endoplasmic reticulum part’

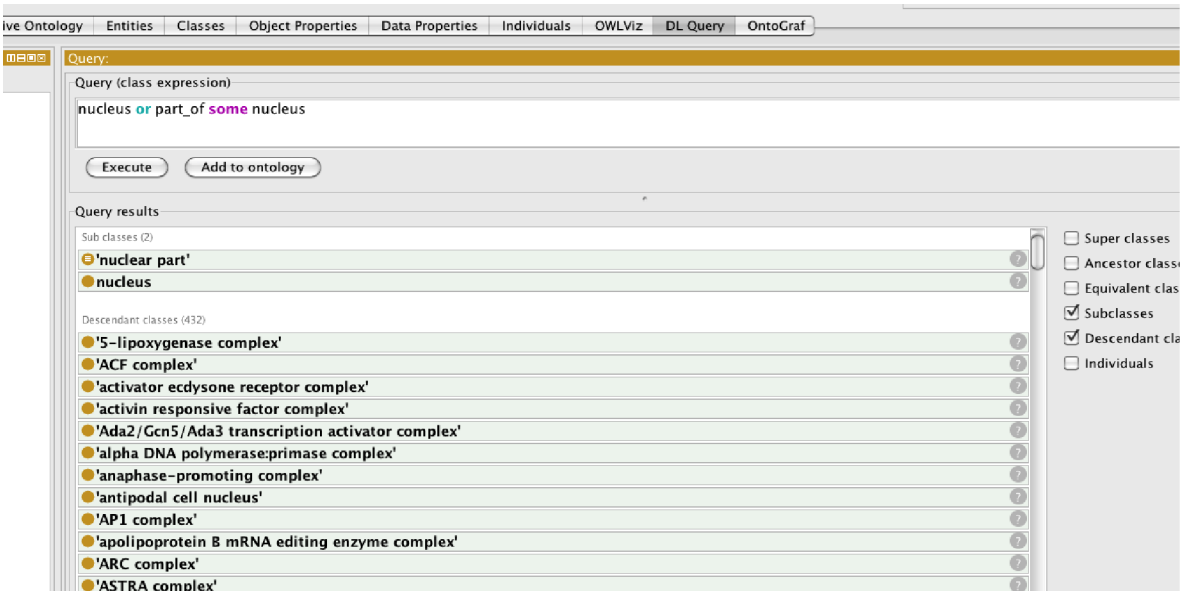




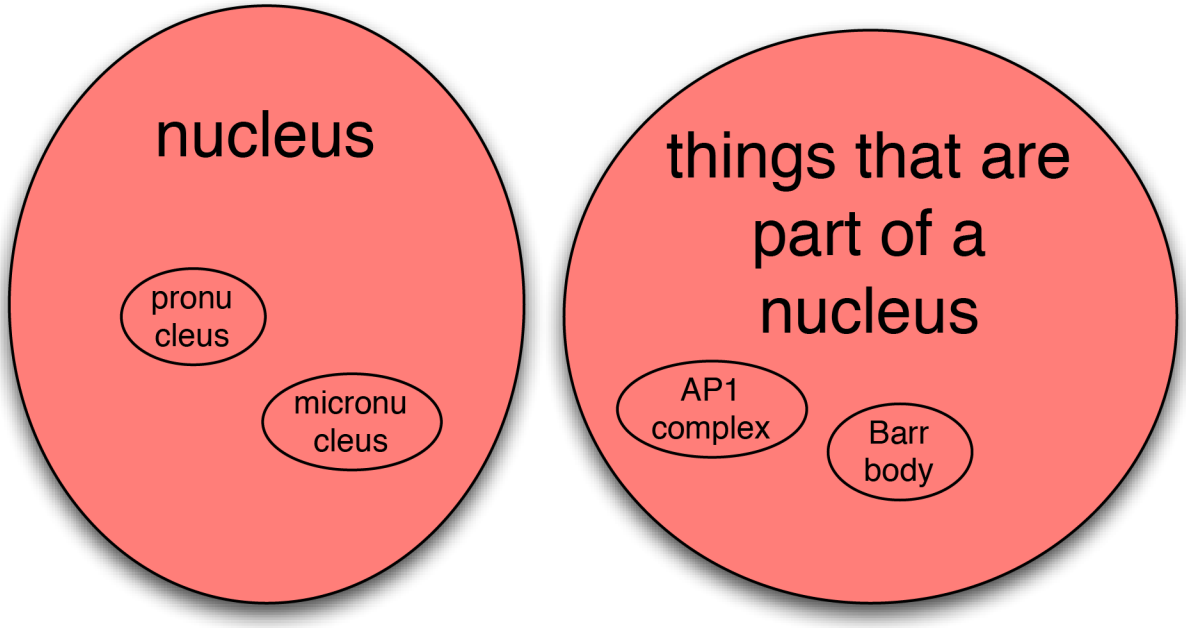
Note that we do not need to use the "part" grouping classes in GO. The same results can be obtained by querying for the intersection of the class "organelle" and the restriction "part_of some ER" – try this and see. We can also ask for superclasses by ticking the boxes above:



The 'or' keyword is to used to create a class expression that is the union of two class expressions. For example:



This is illustrated by the red area in the following Venn diagram:



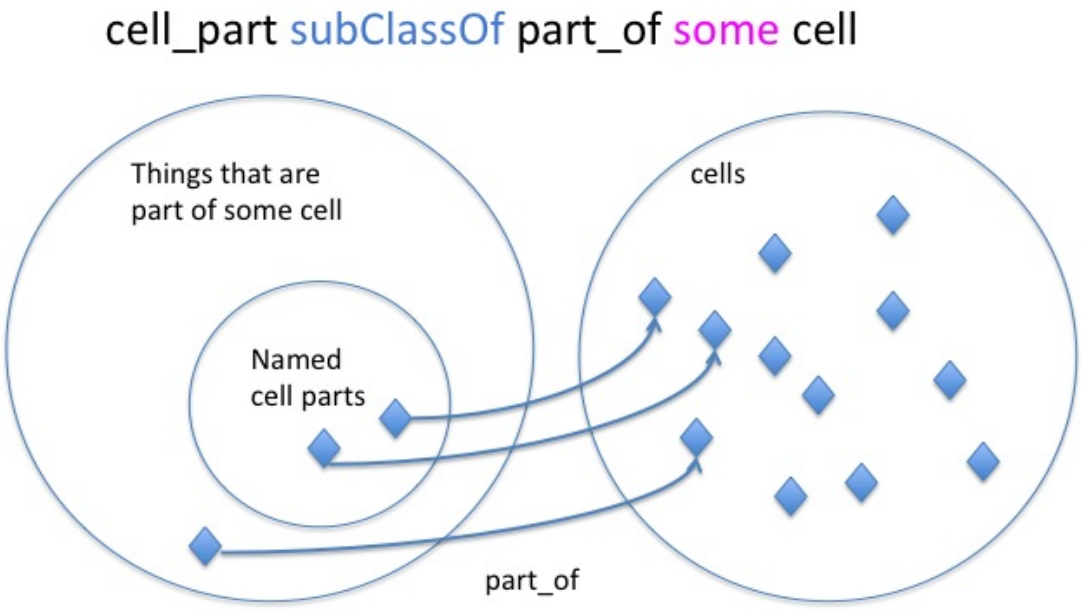
EXERCISE: Basic DL Queries

Go to the basic-dl-query folder in the tutorial directory and follow the instructions in the README.txt

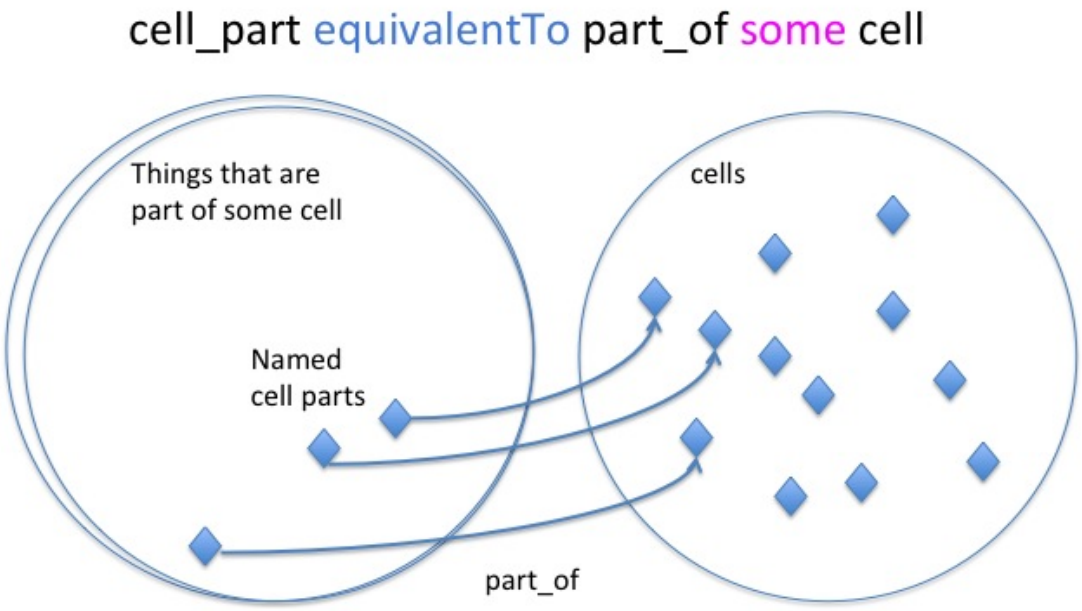
CHAPTER 11

Equivalent classes

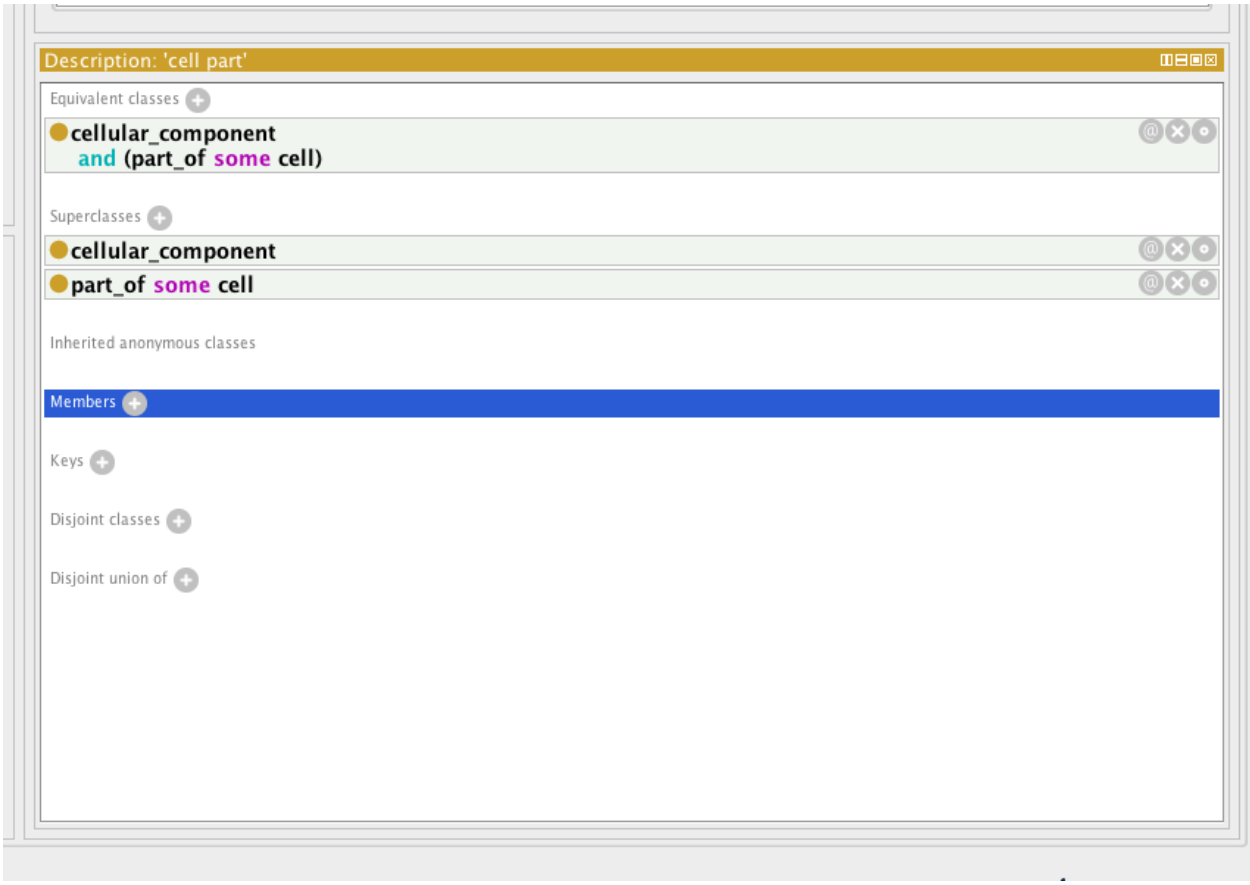
The previous example showed the creation of a class restriction. These restrictions were asserted as superclass restrictions, and are sometimes known as *necessary conditions*. That is, if an individual is a member of the ‘cell part’ then it is necessary for it to also be related to a ‘cell’ along the ‘part of’ property.



Necessary conditions alone mean that individuals can exist that are part of a cell, but are not a type of 'cell part'. In OWL we can make an even stronger statement and define the 'cell part' class as being equivalent to 'part of' some cell. This is known as a necessary and sufficient condition.



In Protégé we can create an equivalent class restriction inside the "Equivalent class' slot of the class description view.


Description: 'cell part'
Equivalent classes
cellular_component
and (part_of some cell)
Superclasses
cellular_component
part_of some cell
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of

CHAPTER 12

Automatic classification

EXERCISE: Basic classification

Go to the basic-classification folder and follow the instructions in the README.txt

EXERCISE: Classification using Unions (Optional)

Optional additional exercise if time permits:

Go to taxon-union folder

This introduces classification using “or” and “not”

Futher reading

owl primer - advanced class relationships

Introduction to property axioms

Property heirarchy:

Navigate to ‘occurs in’ (object property tab) in Protege to see this.

Excercise:

- make a git branch of the go-ontology repo
- open go-edit.obo in Protege

- GO has the class 'regulation of cvt pathway'
- Create a new class, using the same pattern, for 'negative regulation of cvt pathway' (Hint - you can use duplicate to do this, editing the newly created class).
- Run the reasoner. Do you understand the inferred classifications? (Use the explanation plugin to check).

Property chains:

Navigate to 'occurs in' (object property tab) in Protege to see this.

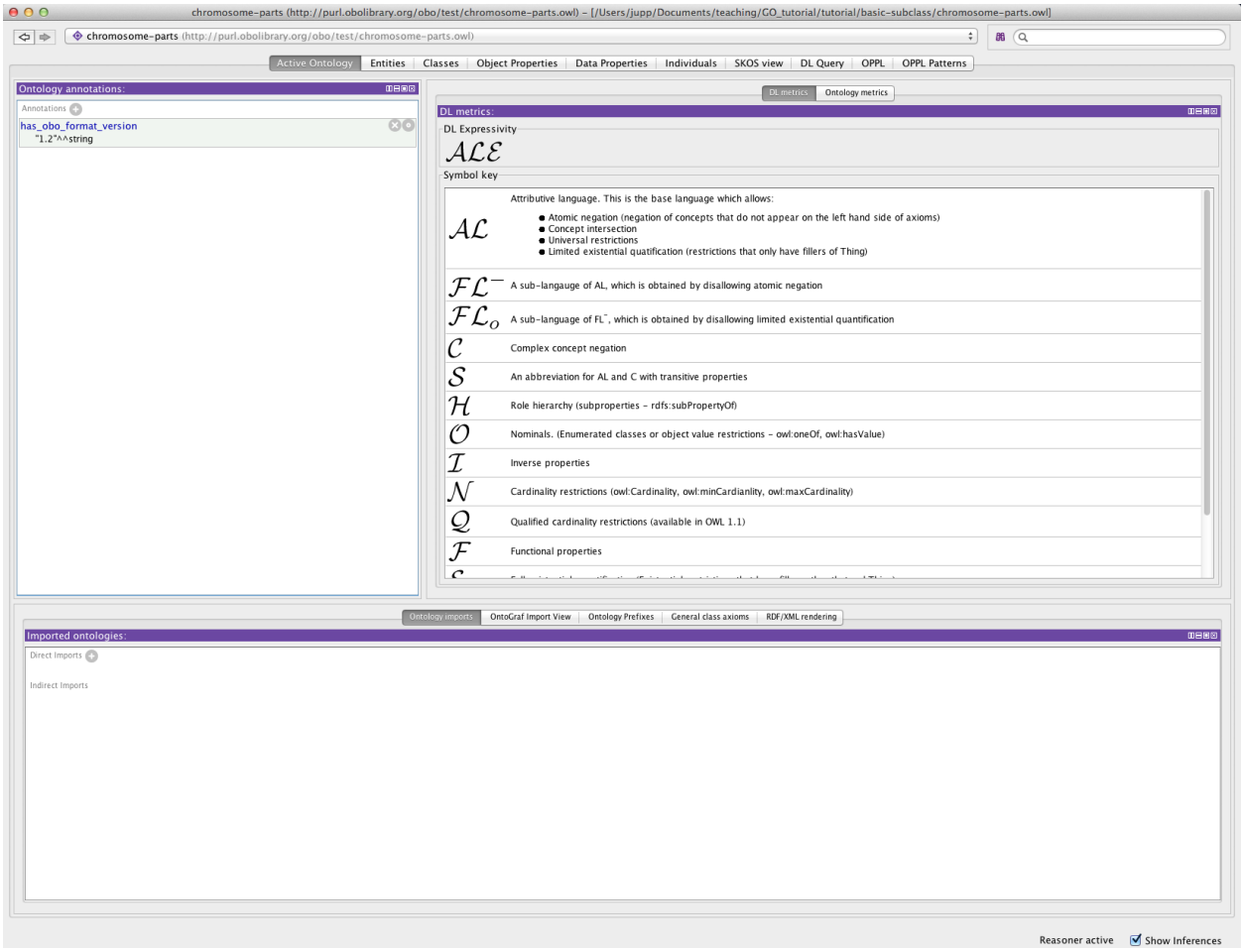
EXERCISE: transport & property chains

Go to the transport property chain exercise folder and follow the instructions in the README

CHAPTER 13

Imports

OWL ontologies may import one or more other OWL ontologies. Ontology imports are managed using the ontology URI (or IRI). Protégé allows you to import ontologies from both the web and your local files system. The imports panel is found in the Active Ontology tab by default.



When a file is imported into an ontology, only the IRI of the imported ontology is stored. Protégé uses the IRI to try and locate the imported ontology the next time you open the file in Protégé. Most OBO ontologies have an IRI that will refer to document via a URL on the web e.g. the Gene Ontology IRI is http://purl.obolibrary.org/obo/go.owl[4]. Navigating to this URL in a web browser will retrieve the latest gene ontology. If your ontology imports an ontology that does not resolve to a web URL, or you are not connected to the internet, then Protégé will prompt you to resolve the ontology IRI to a file on your system.

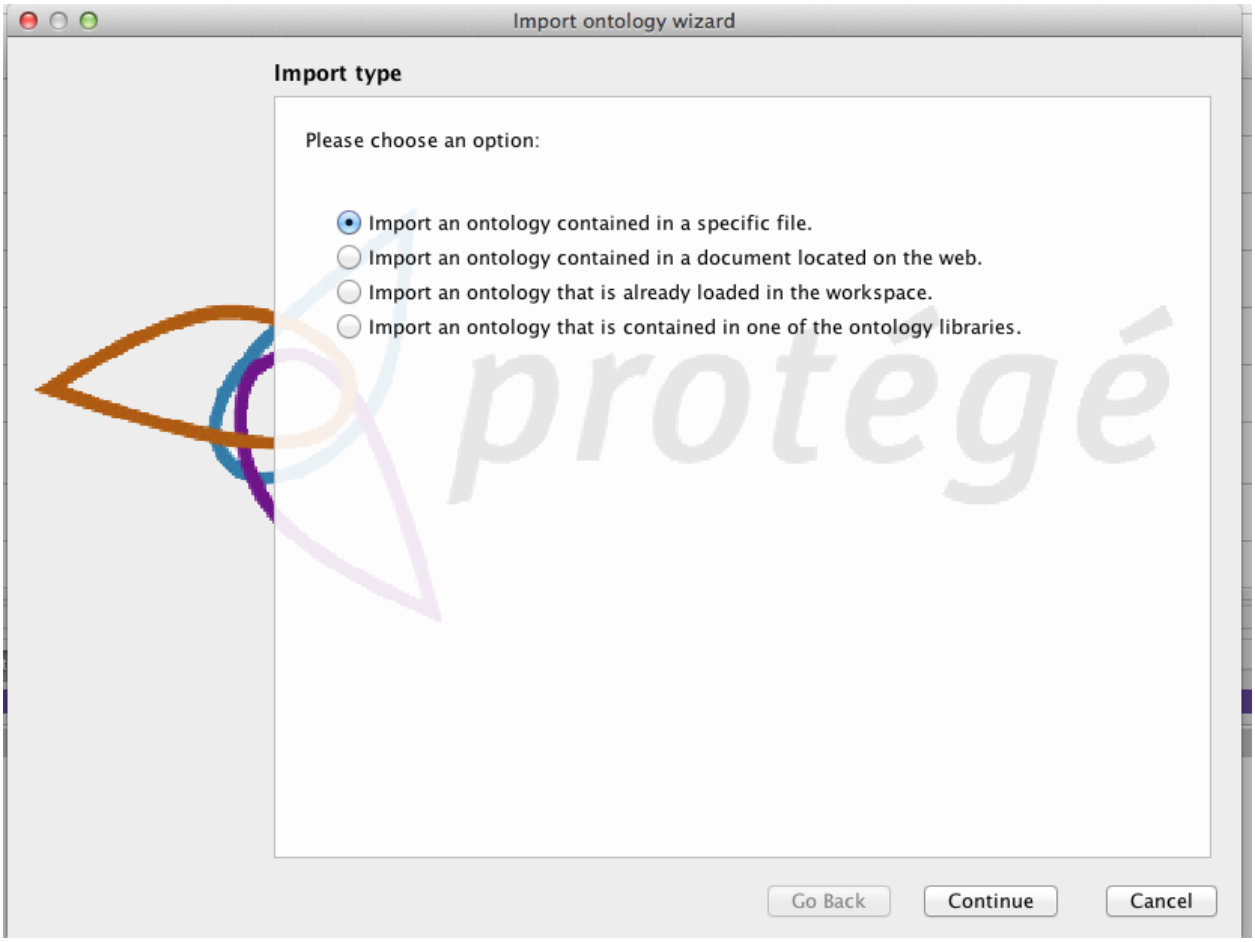
Some examples of imports can be found in

http://wiki.geneontology.org/index.php/Ontology_extensions

EXERCISE: response to stimulus

Go to the response-to-stimulus directory and follow the README.txt

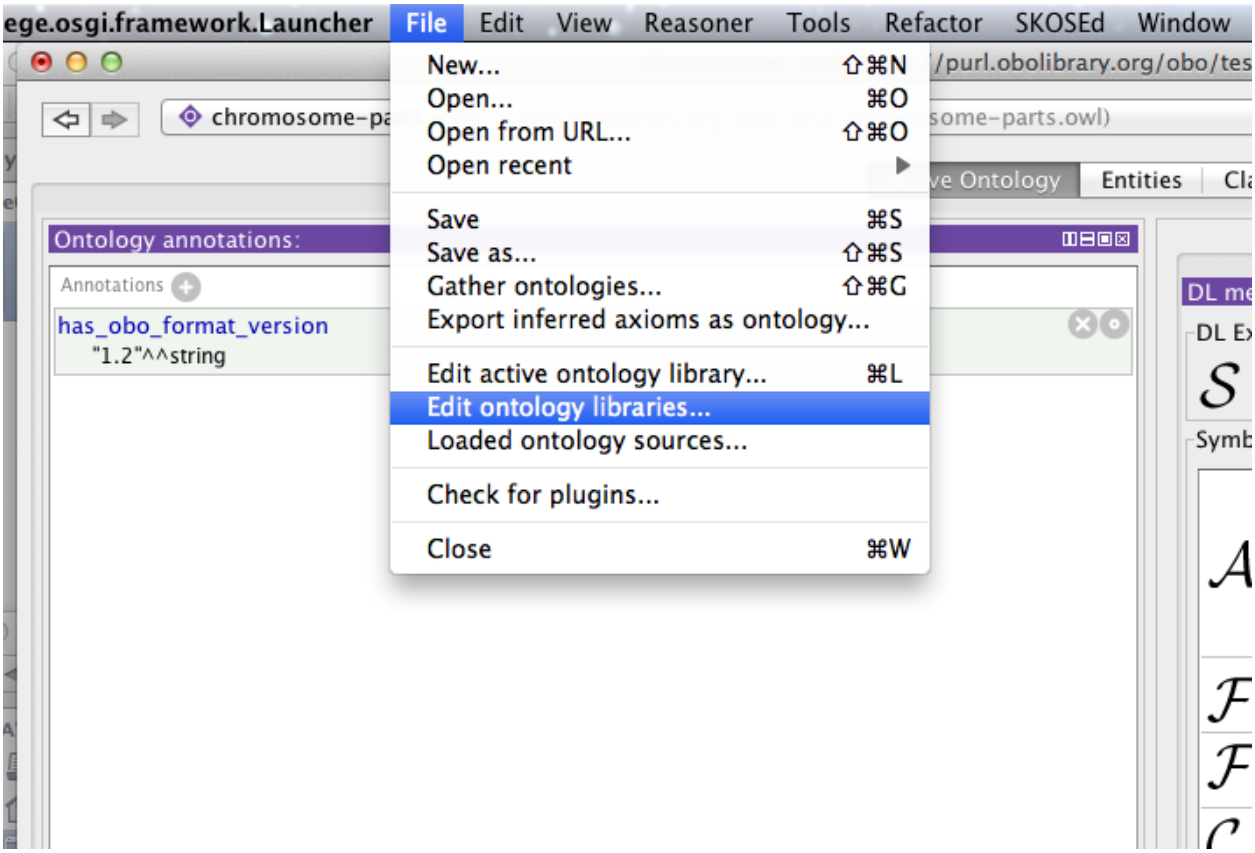
This example makes use of a small ontology called STIMO – the stimulus ontology.



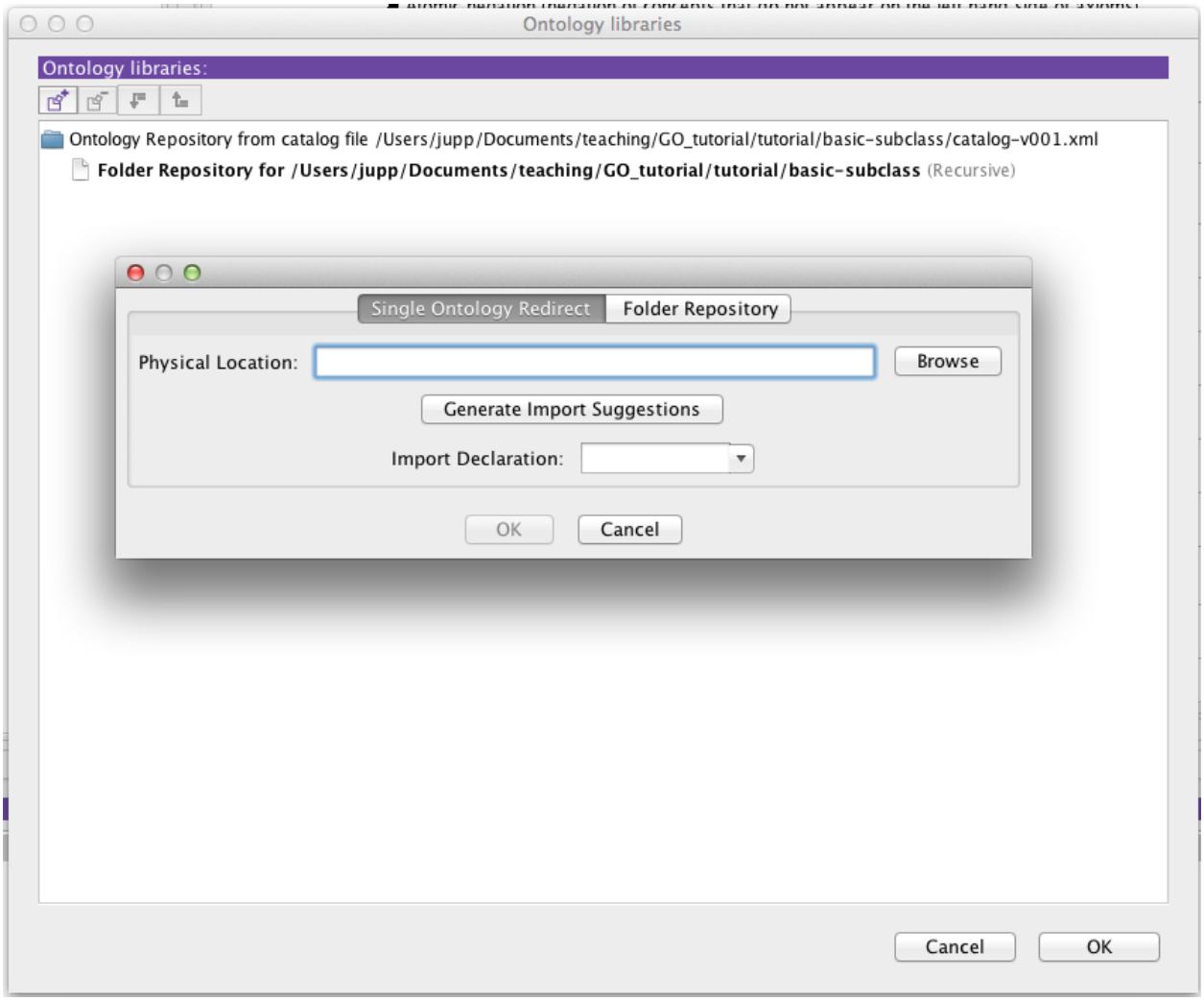
Ontology libraries

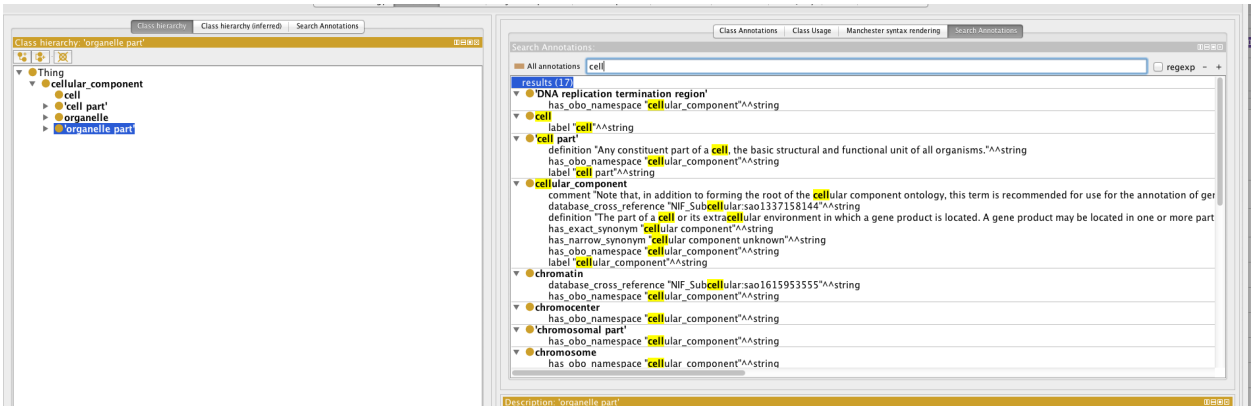
Having to wait for Protégé to download ontologies from the web can be rather time consuming, so luckily Protégé has a mechanism for you define ontology libraries that enable you to store mapping between ontology IRIs and files on your filesystem. You can create edit your ontology library in the File > Edit ontology libraries… menu item.

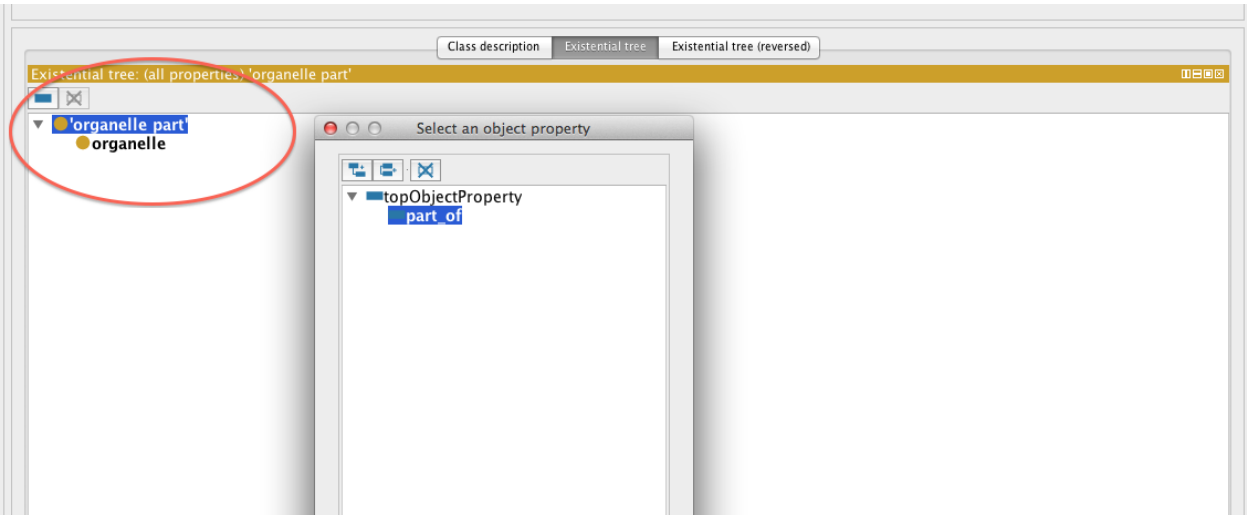
CHAPTER 14



Ontology libraries are stored in a file on your system called catalog-001.xml. When you start working with Protégé you will begin to notice many of these files cropping up on your file system. Chose a catalog-001.xml file to edit and add ontology files to your library.


Ontology libraries
Ontology libraries:
Ontology Repository from catalog file /Users/jupp/Documents/teaching/GO_tutorial/tutorial/basic-subclass/catalog-v001.xml
Folder Repository for /Users/jupp/Documents/teaching/GO_tutorial/tutorial/basic-subclass (Recursive)
Single Ontology Redirect
Folder Repository
Physical Location:
Browse
Generate Import Suggestions
Import Declaration:
OK
Cancel
Cancel
OK





Catalog files

TODO