Protégé-OWL Tutorial

8th International Protégé Conference
Madrid
July 2005

Nick Drummond¹
Matthew Horridge¹
Holger Knublauch¹,²

Installation

- Download the “Full” version of Protege
- Select “Basic + OWL” in the installation Wizard
Overview

- Introduction
- RDF and OWL Lite
- OWL-DL
  - Classes and properties
  - Defined classes
  - Reasoning
- OWL-Full
- Other topics as time permits

The Semantic Web

- Platform for sharing domain models.
- Designed by humans
- “Understandable” for machines
- W3C standard languages
  - XML
  - RDF
  - OWL
  - ... SWRL (?)
RDF(S)

- Simple representation language for domain models / ontologies.
- Resources and links between them.
- Resource types:

  - Classes
  - Properties
  - Individuals

![Diagram showing RDF(S) concepts with classes, properties, and individuals.]

Creating a new RDF Project (1)

![Screenshots of creating a new RDF project, selecting project types and ontology imports.]

© Copyright The University of Manchester and Stanford University 2005
Creating a new RDF Project (2)

What is the default namespace?

Resources in your ontology can be grouped into namespaces. For example, all properties of the Dublin Core namespace have a URI starting with http://purl.org/dc/terms/. Such namespaces are typically abbreviated with acronyms such as "dc". The default namespace of the namespace that is used for resources without a prefix.

Which OWL dialect do you want to use?

You can select which elements of OWL and RDF you want to use in your project. You can change these settings later at any time, using the OWL Preferences... dialog. For example, if you select OWL Lite, then you cannot create ontologies with classes, and if you select pure RDF then you can only create individual Properties and role Classes.

Protégé User Interface
Properties Tab

Individuals Tab
Forms Tab

Optimising Forms
Saving Projects

Protégé project file (pprj) stores form customizations and some other settings

OWL/RDF file contains the actual model in “Semantic Web format”

RDF/OWL Files

<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns=http://www.owl-ontologies.com/areas.owl#"
    xmlns:base="http://www.owl-ontologies.com/areas.owl"
>  
  <owl:Ontology rdf:about="/"
    <rdfs:Class rdf:ID="Area"/>
    <rdfs:subClassOf rdf:resource="#Area"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="Country">
    <rdfs:subClassOf rdf:resource="#Area"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="City">
    <rdfs:subClassOf rdf:resource="#Area"/>
  </rdfs:Class>
  <rdf:Property rdf:ID="hasPart">
    <rdfs:domain rdf:resource="#Area"/>
    <rdfs:range rdf:resource="#Area"/>
  </rdf:Property>
...

Namespace declarations

Short notation for full URI
http://www.owl-ontologies.com/areas.owl#Area
Using RDF Files on the Web

Classes and properties

Australia
Germany
Spain

Individuals

http://www.owl-ontologies.com/areas.owl#Country


Classes and properties

Australia
Germany
Spain

RDF(S) Limitations

• Simple mechanism to formalise domains.
• Limited expressivity.
• Cannot be used to express things such as:
  • “Each are has at most one value for hasPopulation”
  • “Each Country must have at least one City”
  • “Countries cannot have countries as parts”
From RDF to OWL

OWL Dialects

- OWL-Lite
- OWL-DL
- OWL-Full
Building an OWL-DL Ontology

Why Pizzas?

- Fun
- Understood by a wide audience
- Not too controversial
- Simple - about the right numbers of classes and properties for a tutorial
- Comprehensive written version of the tutorial available
Components of an OWL Ontology

Components of an OWL Ontology

OWL Class Descriptions

- OWL is an ontology language that is primarily designed to describe and define classes. Classes are therefore the basic building blocks of an ontology.

- Classes are interpreted as sets of individuals.

- OWL supports six main ways of describing classes of individuals. The simplest of these is a Named Class. The other types of class descriptions are anonymous classes.
OWL Classes

- Named classes - create a class and assign a name to it. Two 'built in' named classes: owl:Thing and owl:Nothing.
- Anonymous classes - built up from class descriptions
  - Intersection, Union and Complement classes
  - Restriction classes - existential, universal, cardinality, hasValue
  - Enumeration classes
- Combinations of Named classes and anonymous classes are used to build up complex class descriptions.

Creating Named Classes

- Select the classes tab use the 'Create subclass' and 'Create sibling class' buttons to create Pizza, PizzaBase and PizzaTopping.
Creating Subclasses

• Create some *subclasses* of *PizzaTopping* to represent high level categorisations of pizza toppings.

• Add additional classes to represent different kinds of pizza toppings.

  Tomato, Parmesan, Mushroom, Pepperoni, Anchovy, Ham, Mozzarella, SpicyBeef

Meaning of Subclass

MozzarellaTopping
PizzaTopping
CheeseTopping
Create some classes to describe different pizzas

- Create a subclass of Pizza called ‘NamedPizza’.

Multiple Inheritance

- OWL allows us to specify multiple named superclasses for any OWL class.
- Create a class called MeatyVegetable as a subclass of VegetableTopping.
- Use the Conditions Widget to add MeatTopping as an extra superclass to MeatyVegetable.
What is a MeatyVegetable?

Does this make sense?

Checking an Ontology

- We've just created a very strange class - intuitively, it should not be possible for individuals that are both a kind of MeatTopping and a kind of VegetableTopping to exist.

- We know that having individuals that are kinds of MeatyVegetables doesn't make sense from a modelling point of view, but can these individuals exist from a logical point of view?

- Ideally, we would like to automatically check our ontology to ensure that the logical meaning corresponds to the intended meaning. To do this, we can use a reasoner.
### Reasoning

- For an ontology that falls into the scope of OWL-DL, we can use a **DL Reasoner** to **infer** information that isn’t explicitly represented in the ontology. Standard ‘reasoning services’ are:
  - **Subsumption** checking
  - **Equivalence** checking
  - **Consistency** checking
  - **Instantiation** checking

### Using a reasoner to check class consistency

- Protege-OWL can be used with any DIG compliant reasoner.
- Communication with the reasoner takes place via HTTP.
- Ensure a reasoner is running and press the check consistency button to check the consistency of named classes in the ontology.
Disjoint Axioms

- Having used a reasoner to check the consistency of named classes in the ontology we notice that MeatyVegetable is consistent - in other words, it’s possible for individuals that are both MeatTopping and CheeseTopping to exist!

- OWL classes ‘overlap’ unless they are explicitly stated to be disjoint with each other, or they are inferred to be disjoint with each other.

- We need to specify that if something is a MeatTopping it cannot be a VegetableTopping. To do this we use disjoint axioms.

Add disjoint axioms and reclassify the ontology

- Add a disjoint axiom to make VegetableTopping disjoint from MeatTopping.

- Reclassify the ontology.

Where else do we need disjoint axioms?
Properties

- OWL has two main types of properties: Object properties and Datatype properties.
- Object properties relate an individual to an individual.
- Datatype properties link an individual to a data value.
- A third type of property, Annotation properties, can be used to attach 'meta-data' to classes, properties and individuals.

Property Hierarchies

- OWL supports the specification of a property hierarchy.
- We can specify that a property has a super-property. In fact, for any given property we can specify multiple super properties.
- In OWL-DL, object properties may only have object properties as super-properties, and datatype properties may only have datatype properties as super-properties.
Create Properties to Describe Pizzas

- Create an object property called ‘hasIngredient’.
- Create an object property called ‘hasTopping’ as a sub-property of hasIngredient.
- Create an object property called ‘hasBase’ as a sub-property of hasIngredient.

Property Characteristics

- We can specify additional property characteristics by typing properties as:
  - Functional
  - InverseFunctional
  - Symmetric
  - Transitive
  - Beware! Certain combinations of the above characteristics can cause the ontology to be OWL-Full.
Augment the properties with characteristics

- A pizza only has one base - make the hasBase property functional.
- We want to capture the fact that the ingredients that make up the pizza toppings also make up the pizza - make the hasIngredient property transitive.

Restrictions

- Restrictions describe a class of individuals that is determined by the type and possibly the number of relationships that they participate in.
- Restrictions can be grouped into three main categories:
  - Quantifier restrictions (Existential ∃, Universal ∀)
  - Cardinality restrictions (Min ≥, Equal =, Max ≤)
  - HasValue restrictions (∉)
Existential Restrictions

- The most common type of restriction that we will use is an existential restriction, which has the symbol $\exists$ (backwards E).
- The existential restriction means 'some values from', or 'at least one'.
- An existential restriction describes the class of individuals that have at least one relationship along a specified property to an individual that is a member of a specified class.

Create some existential restrictions

- We can specify that every pizza must have at least one base by creating an existential restriction along the hasBase property with a filler of PizzaBase.
- To do this in Protege-OWL we use the conditions widget.

$$\exists \text{ hasBase PizzaBase}$$
More Existential Restrictions

- Create restrictions to say that MargheritaPizzas have at least one mozzarella topping and at least one tomato topping.
- Create restrictions to describe the fact that AmericanaPizzas have toppings of mozzarella, tomato and pepperoni.
- Create restrictions to represent SpicyBeefPizzas having toppings of mozzarella, tomato and spicy beef.

Existential Restrictions

∃ hasTopping MozzarellaTopping
Necessary Conditions

• So far, all the conditions that we have used in class descriptions have been \textit{necessary} conditions.

• For a given class, \textit{necessary} conditions are the conditions that an individual \textit{must} fulfil if it is a member of that class.

• For example, recall our description of a MargheritaPizza...

If an individual is a member of MargheritaPizza, it is \textit{necessarily} a NamedPizza, and it \textit{necessarily} has at least one mozzarella topping, and it \textit{necessarily} has at least one tomato topping.

Necessary Conditions

• For a given class, the \textit{necessary} conditions represent \textit{superclasses} of that class.

• Recall our description of a pizza...

© Copyright The University of Manchester and Stanford University 2005
Necessary Conditions on Pizza

Necessary & Sufficient Conditions

- With Necessary conditions, if we know that an individual is a member of a given class, we also know that it must fulfil the Necessary conditions on that class.

- What about going 'the other way round'? e.g. Given an individual that fulfils some conditions, what classes is it a member of?

- OWL also supports Necessary & Sufficient conditions, which allow us to determine that any individual that satisfies the conditions can be inferred to be a member of the class that the conditions are on.
Necessary & Sufficient Conditions

- Recall that **ClassA** is a subclass of **ClassB** if all individuals in **ClassA** are also in **ClassB**.

- Therefore, if all of the individuals in **ClassB** fulfil the necessary & sufficient conditions on **ClassA**, all of them must also be members of **ClassA**, and we can infer that **ClassB** is a subclass of **ClassA**.

Necessary & Sufficient Conditions Example

- Create a **CheesyPizza** class, which is a subclass of **Pizza** and also has at least one **CheeseTopping**.

- Create the conditions under the Necessary & Sufficient header in the conditions widget.
Necessary & Sufficient Conditions Example

- Classify the ontology...

Terminology

- Classes with only Necessary conditions are also known as Primitive classes.
- Classes with at least one set of Necessary & Sufficient conditions are known as Defined classes.
- Classes with only Necessary conditions are said to have a Description. Classes with at least one set of Necessary & Sufficient conditions are said to have a Definition.
- A distinction can be made between asserted and inferred information.
Add another defined class

- Create another defined class called **MeatyPizza** to specify that any pizza with at least one meat topping must be a **MeatyPizza** and then reclassify the ontology.

Multiple Parents

- Take a look at our **asserted** class hierarchy. Notice that each class only has **one primitive parent** - the class hierarchy is a **tree**.

- Notice that the **inferred** class hierarchy is a **lattice** - classes have **multiple parents**.

- We let the reasoner take care of maintaining a multi-parent hierarchy - the asserted hierarchy is maintained as a tree.
Vegetarian Pizzas

- We want to define what it means to be a VegetarianPizza, so that we can use the reasoner determine which of our pizzas are vegetarian pizzas.
- How should we define a VegetarianPizza?

Vegetarian Pizzas

- We will define a VegetarianPizza to be any pizza that only has vegetarian toppings.
- To do this we need to decide what a vegetarian topping is...
Vegetarian Toppings

- A VegetarianTopping is any pizza topping that is either a CheeseTopping or VegetableTopping.
- We can use a Union Class to specify this.
- Create VegetarianTopping as a subclass of PizzaTopping, and add a necessary & sufficient condition to specify that VegetarianToppings are either CheeseTopping or VegetableTopping.

Union Classes

CheeseTopping

VegetableTopping

CheeseTopping \cup VegetableTopping
Universal Restrictions

- We want to be able to say that VegetarianPizzas only have toppings that are VegetarianToppings.
- In order to do this we can use a Universal Restriction.
- Universal restrictions are written using the symbol $\forall$ (upside down A).
- Universal restrictions mean ‘all values from’, or ‘only’.
- A universal restriction describes the class of individuals that for a given property, only have relationships to individuals from a specified class.

Define VegetarianPizza

- Create a subclass of Pizza called VegetarianPizza. Add a universal restriction to specify that a VegetarianPizza only has toppings that are VegetarianToppings. Make VegetarianPizza a defined class.
Universal Restrictions

∀ hasTopping VegetarianTopping

VegetarianTopping

Subclasses of VegetarianPizza

• Use the reasoner to classify the ontology... what are the subclasses of VegetarianPizza?

• Is our definition of VegetarianPizza correct?
The Open World Assumption

• None of the pizzas have been classified as subclasses of VegetarianPizza.

• Let’s revisit the description of MargheritaPizza...

![Image of Asserted and Inferred conditions for a pizza]

The Open World Assumption

• Just because something hasn’t been stated doesn’t mean that it isn’t true. Contrast this with a database.

• For example we haven’t stated that a Margherita pizza has some pepperoni topping, but because of the open world assumption, it could have some.

• In Open World Reasoning, something isn’t assumed to be false unless it is explicitly stated to be false.
Closure

- For all our pizzas we need to say something along the lines of “these kinds of pizzas have these toppings and only these toppings”.
- For example, a MargheritaPizza has some mozzarella topping and has some tomato topping and only has mozzarella topping or tomato topping.
- In other words, for any given pizza, we want to ‘close off’ the possible toppings.

Closure Axioms

- To close off the possible toppings that a MargheritaPizza can have, we need to use a universal restriction to say that a MargheritaPizza can only have toppings that are MozzarellaTopping or TomatoTopping.

The universal restriction is known as a closure axiom.
Add more closure axioms

- Modify the descriptions of the other pizzas by adding closure axioms along the hasTopping property.
- The general pattern for a closure axiom is to create a universal restriction along the property being closed, that has a filler which is the union of the fillers of the existential restrictions for that property.
- After adding the closure axioms, classify the ontology.

The asserted and inferred hierarchies

![Diagram showing the asserted and inferred hierarchies of pizza types]
Summary

- Classes are the building blocks of an OWL ontology - OWL has two main types of class description: Named classes and anonymous classes.
- OWL distinguishes between necessary, and necessary & sufficient conditions.
- OWL-DL is underpinned by a description logic. For ontologies that fall into the scope of OWL-DL we can use a reasoner to automatically check the consistency of classes, and take what we have explicitly stated in the ontology and use it to infer new information.
- OWL makes the Open World Assumption and uses Open World Reasoning.

Namespace and Prefixes

- Each ontology should have a unique default namespace.
- Imported concepts are prefixed.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
<th>Imports</th>
<th>Alias URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl</td>
<td><a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dc</td>
<td><a href="http://purl.org/dc/elements/1.1">http://purl.org/dc/elements/1.1</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>protege</td>
<td><a href="http://protege.stanford.edu/protege/">http://protege.stanford.edu/protege/</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w3c</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xhtml</td>
<td><a href="http://www.w3.org/2001/xmlns#">http://www.w3.org/2001/xmlns#</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dcterms</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xml</td>
<td><a href="http://www.w3.org/2000/01/xhtml4/other#">http://www.w3.org/2000/01/xhtml4/other#</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ontology Imports (1)

- Allows ontology A to access all the resources from ontology B.

Ontology Imports (2)
Multi-File Projects

- One ontology is “active”.
- Changes will only be applied to the active ontology.
Import Redirection

- To be sharable on the web, ontologies should import other ontologies via http: addresses.
- However, at development time ontologies are local files (file://... address).
- Ont-Policy file defines a mapping between
  - Logical address: http://...
  - Physical address: http://...

Namespace Pitfalls

- One file can define resources from multiple namespaces.
- The physical location of a file does not have to match with the default namespace.
- Different files can access the same namespace using different prefixes.
- Good practice: Keep it simple!
Using Metaclasses

- Make rdfs:Class (or owl:Class) visible.
- Create Metaclass (subclass of rdfs:Class).
- Add properties to metaclass.
- Change type of classes:
  - Right click on class.
  - Go to Metaclasses/Change metaclass...
  - Select new type.
Classes as Values (1)

- All classes are instances of a metaclass.
- Properties can take classes as values (rdfs:range = rdfs:Class).
- For example, a city may be famous for a kind of pizza.

```
isFamousFor
  ◆ Naples  ➔  ◇ Napolitana
```

Classes as Values (2)
Classes as Values (3)

Links

- Protege-OWL web site:
  
  http://protege.stanford.edu/plugins/owl

- CO-ODE web site:
  
  http://www.co-ode.org