

Protégé-OWL Tutorial

8th International Protégé Conference
Madrid
July 2005

Nick Drummond¹
Matthew Horridge¹
Holger Knublauch^{1,2}

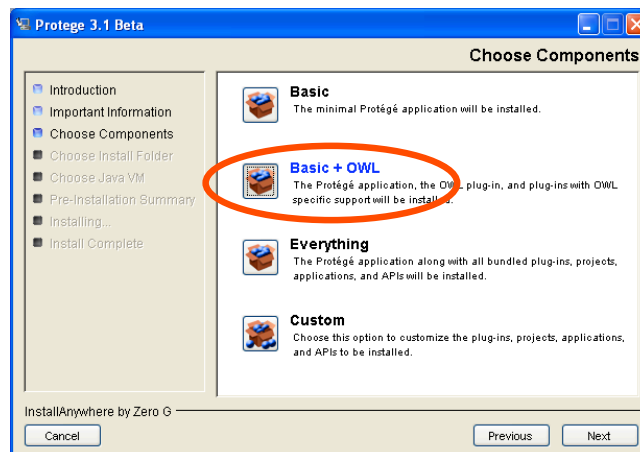


© Copyright The University of Manchester and Stanford University 2005

1

Installation

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



© Copyright The University of Manchester and Stanford University 2005

2

Overview

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

© Copyright The University of Manchester and Stanford University 2005

3

The Semantic Web

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

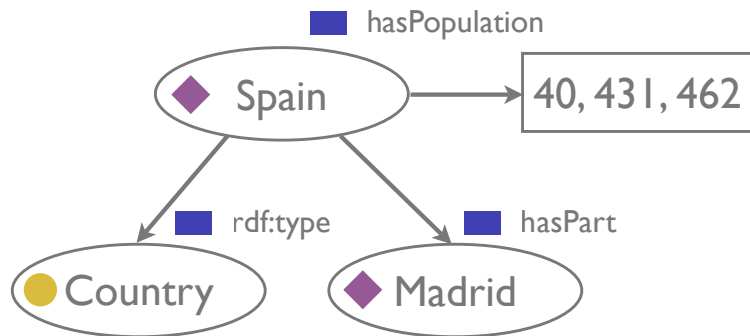
© Copyright The University of Manchester and Stanford University 2005

4

RDF(S)

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

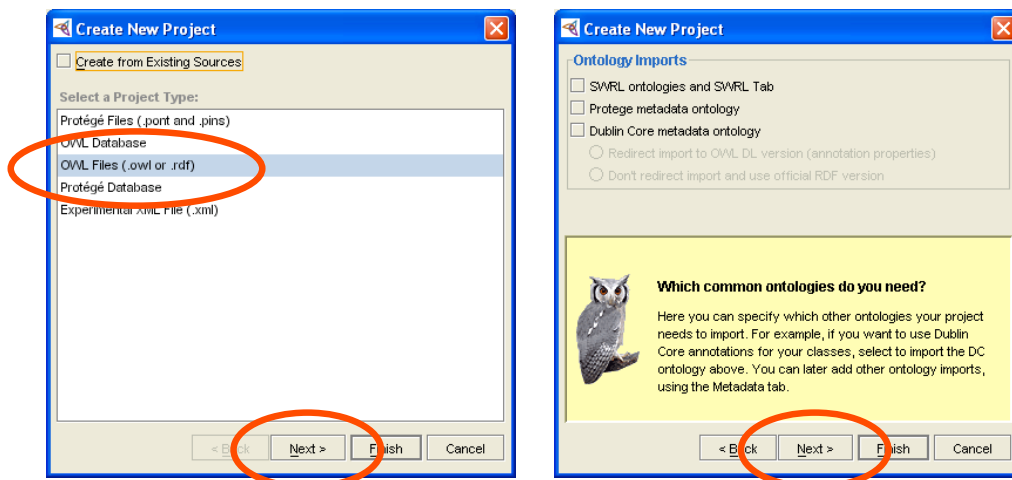
- Classes
- Properties
- ◆ Individuals



© Copyright The University of Manchester and Stanford University 2005

5

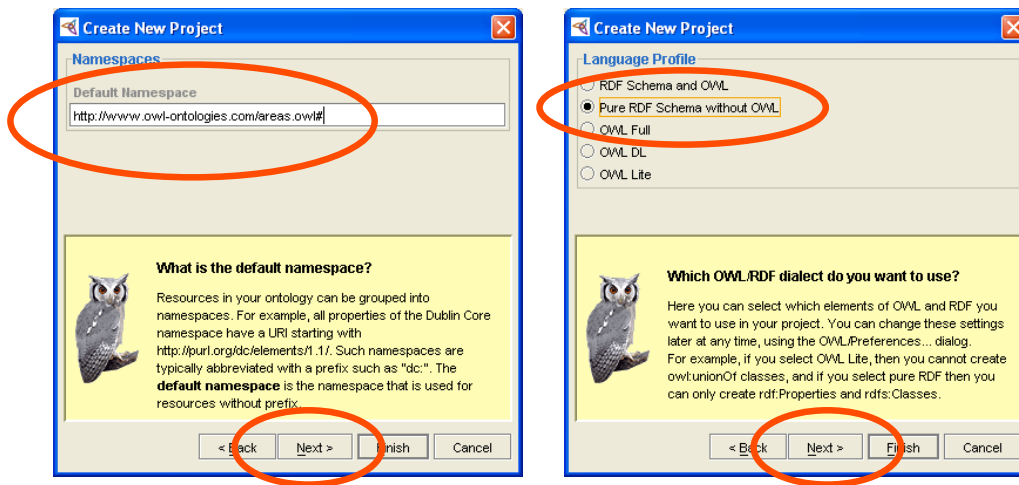
Creating a new RDF Project (I)



© Copyright The University of Manchester and Stanford University 2005

6

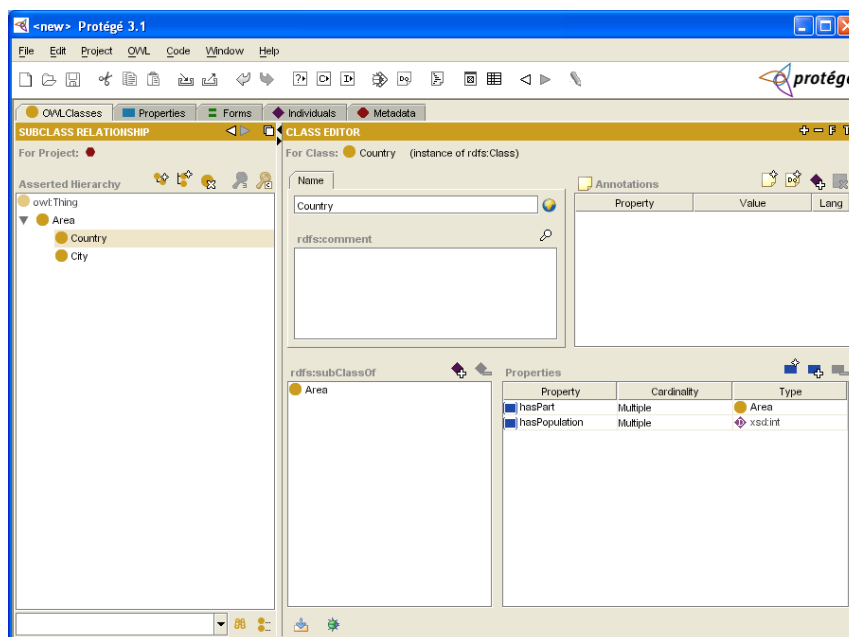
Creating a new RDF Project (2)



© Copyright The University of Manchester and Stanford University 2005

7

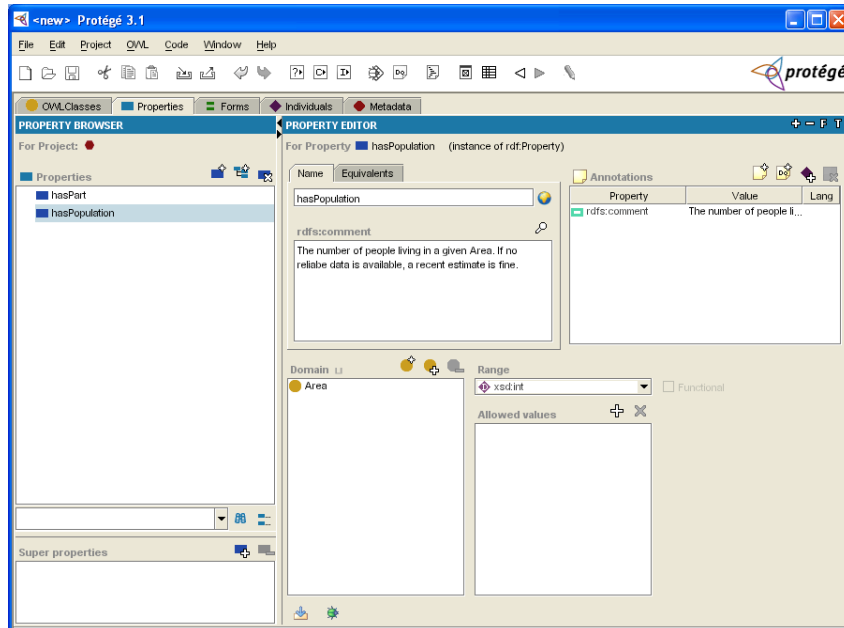
Protégé User Interface



© Copyright The University of Manchester and Stanford University 2005

8

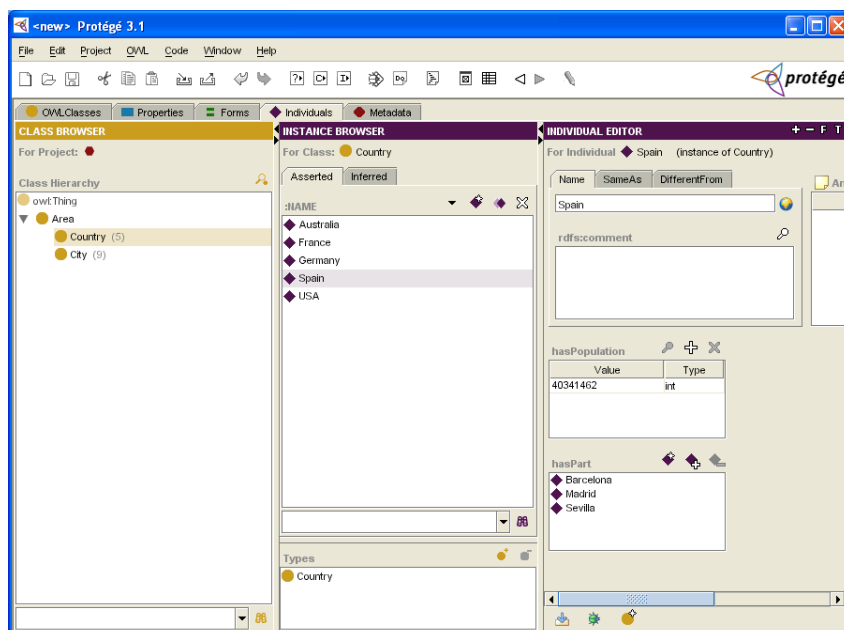
Properties Tab



© Copyright The University of Manchester and Stanford University 2005

9

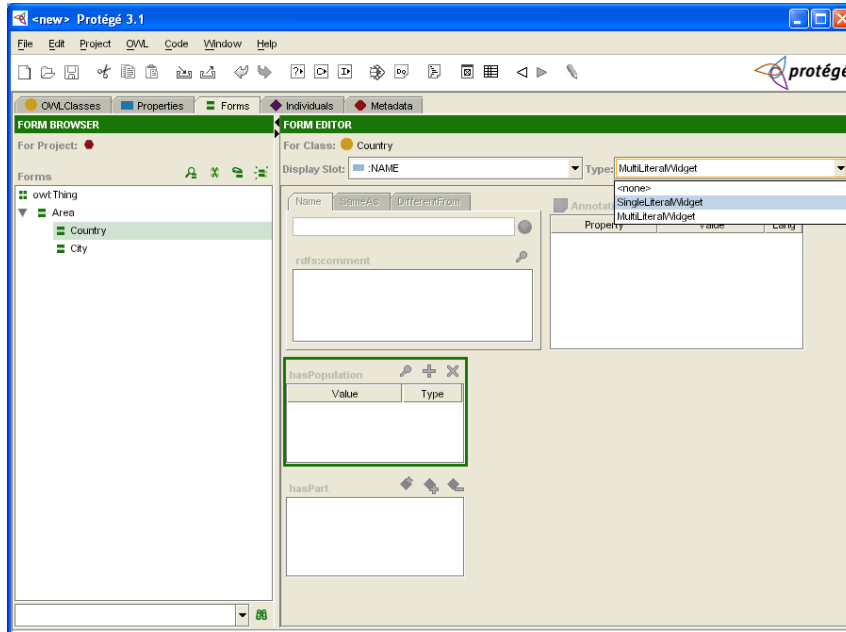
Individuals Tab



© Copyright The University of Manchester and Stanford University 2005

10

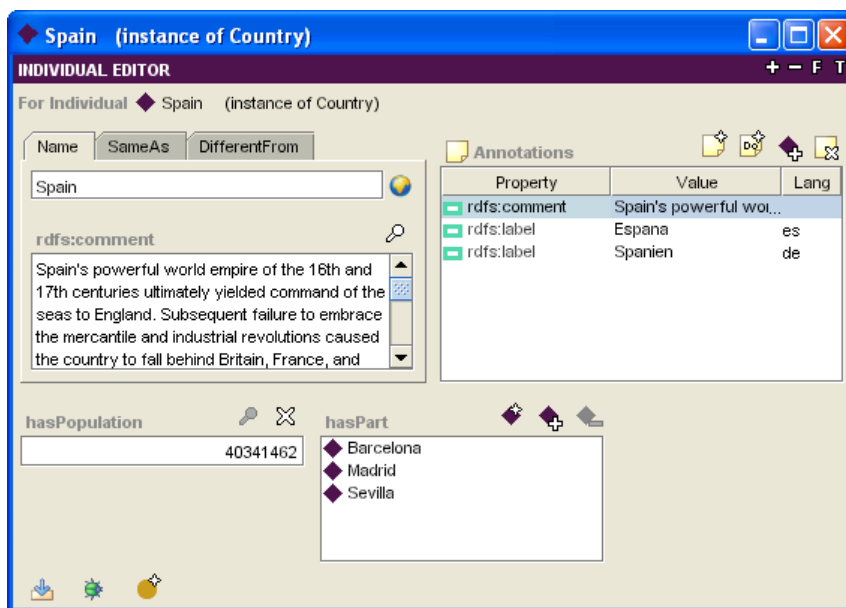
Forms Tab



© Copyright The University of Manchester and Stanford University 2005

11

Optimising Forms

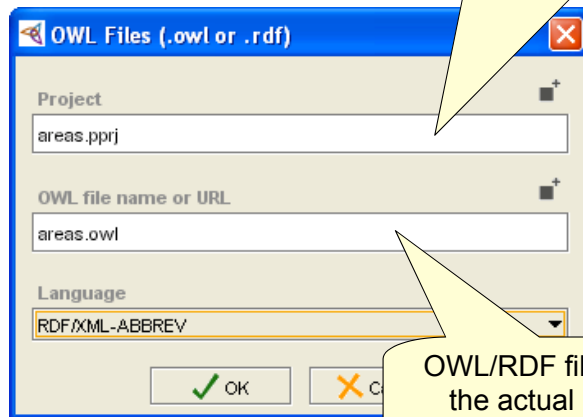


© Copyright The University of Manchester and Stanford University 2005

12

Saving Projects

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



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

© Copyright The University of Manchester and Stanford University 2005

13

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#"
  xml:base="http://www.owl-ontologies.com/areas.owl">
  <owl:Ontology rdf:about="" />
  <rdfs:Class rdf:ID="Area" />
  <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

© Copyright The University of Manchester and Stanford University 2005

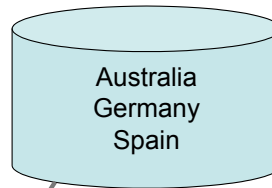
14

Using RDF Files on the Web

Classes and properties

```
<?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#"
  xml:base="http://www.owl-ontologies.com/areas.owl#"
  <owl:Ontology rdf:about="">
  <rdfs:Class rdf:ID="Area"/>
  <rdfs:ClassOf rdf:resource="#Area"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="City">
  <rdfs:subClassOf rdf:resource="#Area"/>
  </rdfs:Class>
  <rdfs:Property rdf:ID="hasPart">
  <rdfs:domain rdf:resource="#Area"/>
  <rdfs:range rdf:resource="#Area"/>
  </rdfs:Property>
  ...
</rdf:RDF>
```

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



Individuals



<http://www.cia.gov/cia/publications/factbook/maps/sp-map.gif>

© Copyright The University of Manchester and Stanford University 2005

15

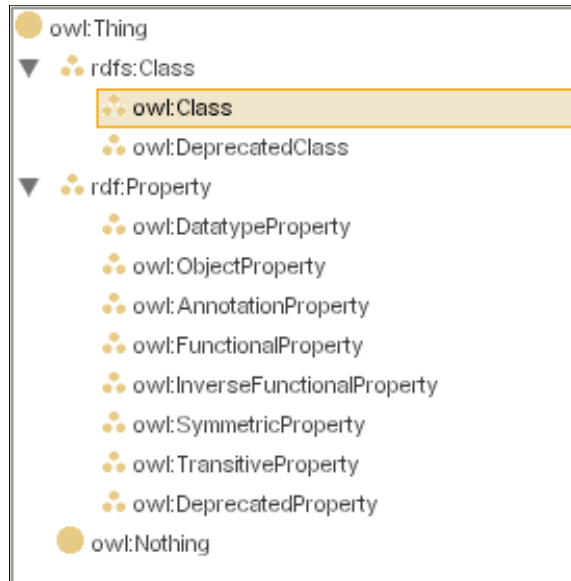
RDF(S) Limitations

- 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”

© Copyright The University of Manchester and Stanford University 2005

16

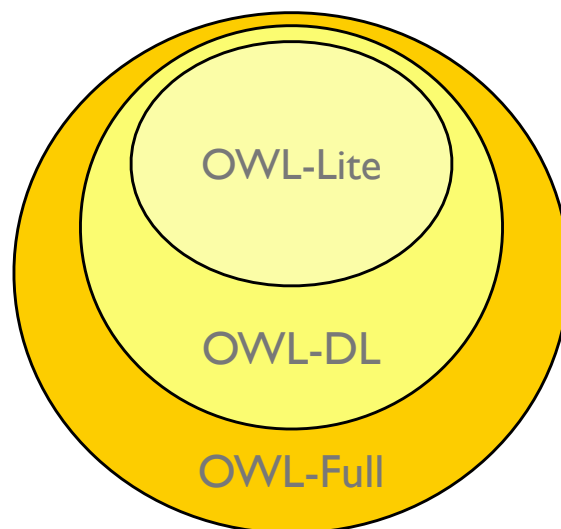
From RDF to OWL



© Copyright The University of Manchester and Stanford University 2005

17

OWL Dialects



© Copyright The University of Manchester and Stanford University 2005

18

Building an OWL-DL Ontology

19

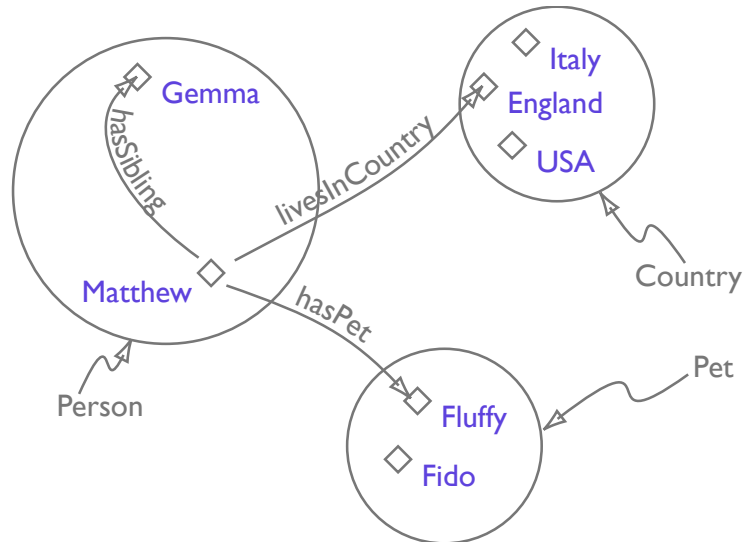
Why Pizzas?

- 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

© Copyright The University of Manchester and Stanford University 2005

20

Components of an OWL Ontology



© Copyright The University of Manchester and Stanford University 2005

21

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**.

© Copyright The University of Manchester and Stanford University 2005

22

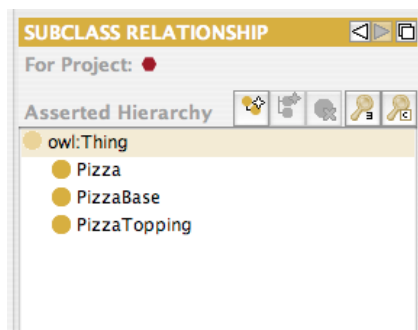
OWL Classes

- 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`.

© Copyright The University of Manchester and Stanford University 2005

23

Creating Named Classes

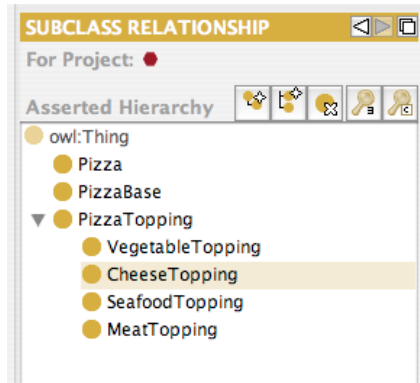


- Select the classes tab use the 'Create subclass' and 'Create sibling class' buttons to create `Pizza`, `PizzaBase` and `PizzaTopping`.

© Copyright The University of Manchester and Stanford University 2005

24

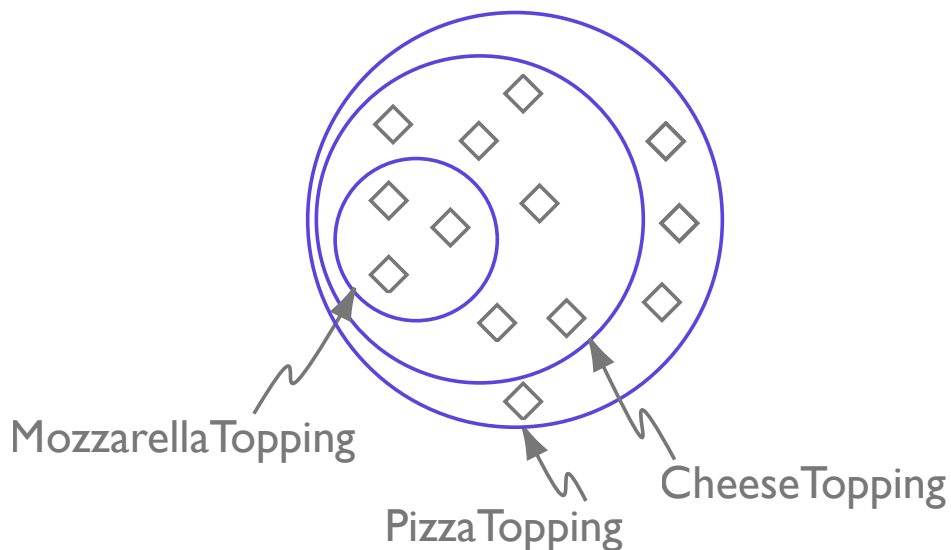
Creating Subclasses



- 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



Create some classes to describe different pizzas

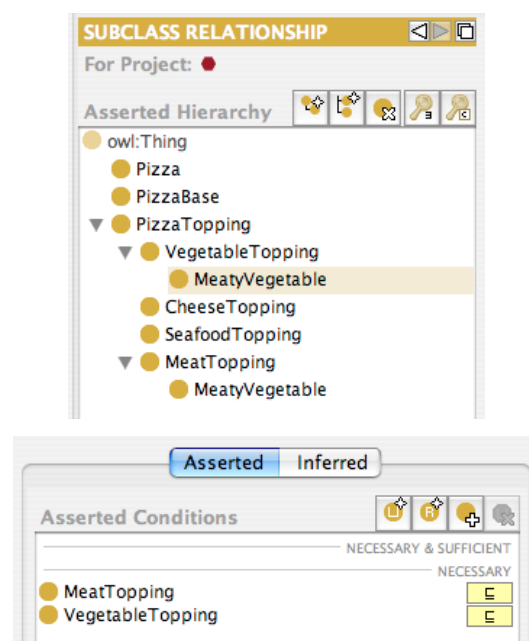
- Create a subclass of Pizza called 'NamedPizza'.
- Create 'MargheritaPizza', 'AmericanaPizza', 'SpicyBeefPizza' as subclasses of NamedPizza.

© Copyright The University of Manchester and Stanford University 2005

27

Multiple Inheritance

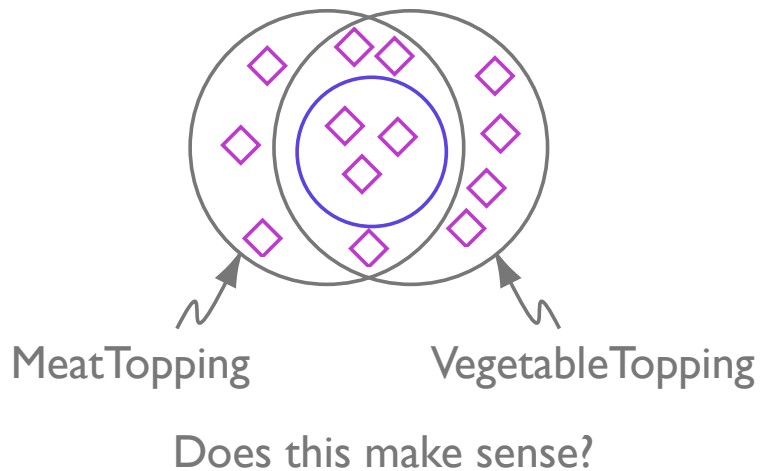
- OWL allows us to specify multiple named superclasses for any OWL class.
- Create a class called **MeatyVegetable** as a subclass of **VegetableTopping**.
- Use the **Conditions Widget** to add **MeatTopping** as an extra superclass to **MeatyVegetable**



© Copyright The University of Manchester and Stanford University 2005

28

What is a MeatyVegetable?



© Copyright The University of Manchester and Stanford University 2005

29

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**.

© Copyright The University of Manchester and Stanford University 2005

30

Reasoning

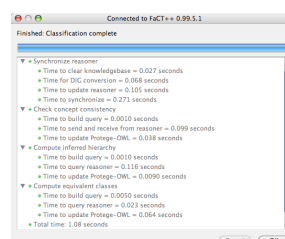
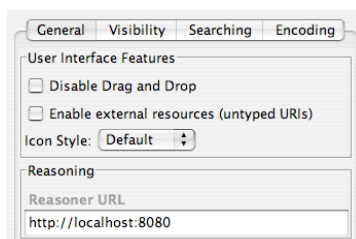
- For an ontology that falls into the scope of **OWL-DL**, we can use a **DL Reasoner** to **infer** information that isn't explicitly represented in the ontology. Standard 'reasoning services' are:
 - **Subsumption** checking
 - **Equivalence** checking
 - **Consistency** checking
 - **Instantiation**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

31
© Copyright The University of Manchester and Stanford University 2005

31

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.



© Copyright The University of Manchester and Stanford University 2005

32

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**.

© Copyright The University of Manchester and Stanford University 2005

33

Add disjoint axioms and reclassify the ontology

- Add a disjoint axiom to make VegetableTopping disjoint from MeatTopping.
- Reclassify the ontology.

The screenshot shows the Protege-OWL interface. On the left, a 'Check concept consistency' window displays the following information:

- Time to build query = 0.0080 seconds
- Time to send and receive from reasoner = 0.08 sec
- Inconsistent concepts
 - MeatyVegetable is inconsistent
- Time to update Protege-OWL = 0.38 seconds

On the right, a class hierarchy is shown:

- PizzaTopping
 - VegetableTopping
 - MeatyVegetable
 - CheeseTopping
 - SeafoodTopping
 - MeatTopping
 - MeatyVegetable

Where else do we need disjoint axioms?

© Copyright The University of Manchester and Stanford University 2005

34

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.

© Copyright The University of Manchester and Stanford University 2005

35

Property Hierarchies

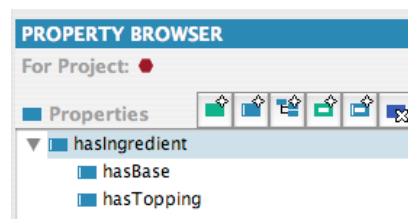
- OWL supports the specification of a **property hierarchy**.
- We can specify that a property has a **super-property**. In fact, for any given property we can specify multiple super properties.
- In OWL-DL, object properties may only have object properties as super-properties, and datatype properties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

© Copyright The University of Manchester and Stanford University 2005

36

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.



© Copyright The University of Manchester and Stanford University 2005

37

Property Characteristics

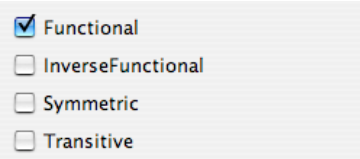
- We can specify additional property characteristics by typing properties as:
 - **Functional**
 - **InverseFunctional**
 - **Symmetric**
 - **Transitive**
- **Beware!** Certain combinations of the above characteristics can cause the ontology to be **OWL-Full**.

© Copyright The University of Manchester and Stanford University 2005

38

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**.



Functional
 InverseFunctional
 Symmetric
 Transitive

© Copyright The University of Manchester and Stanford University 2005

39

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 \exists , Universal \forall)
 - Cardinality restrictions (Min \geq , Equal $=$, Max \leq)
 - HasValue restrictions (\ni)

© Copyright The University of Manchester and Stanford University 2005

40

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 mean ‘**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**.

© Copyright The University of Manchester and Stanford University 2005

41

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 hasBase PizzaBase

Property Filler

© Copyright The University of Manchester and Stanford University 2005

42

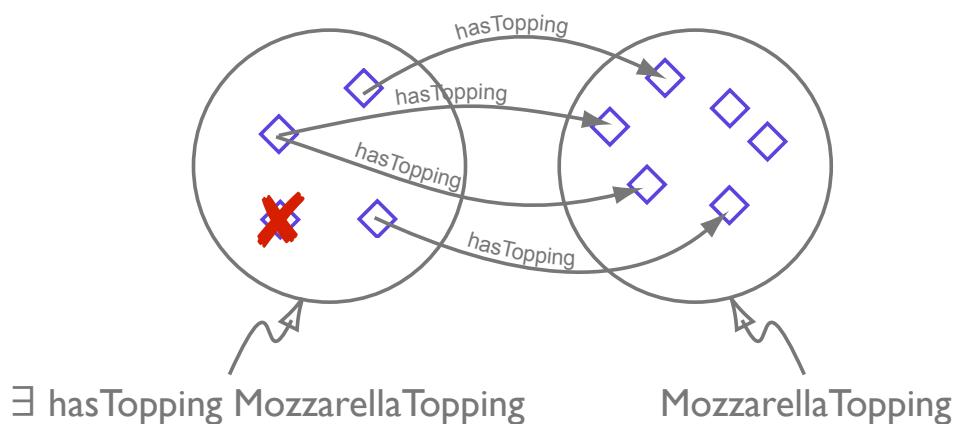
More Existential Restrictions

- 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.

© Copyright The University of Manchester and Stanford University 2005

43

Existential Restrictions

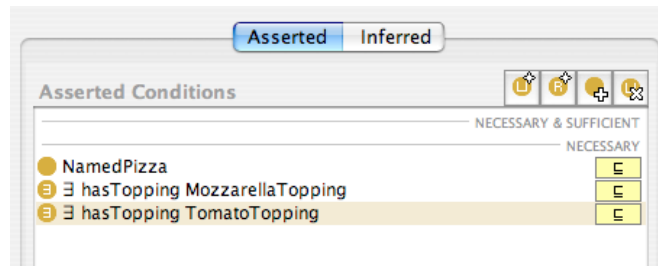


© Copyright The University of Manchester and Stanford University 2005

44

Necessary Conditions

- So far, all the conditions that we have used in class descriptions have been 'necessary' conditions.
- For a given class, necessary conditions are the conditions that an individual **must fulfil** if it is a member of that class.
- For example, recall our description of a **MargheritaPizza**...



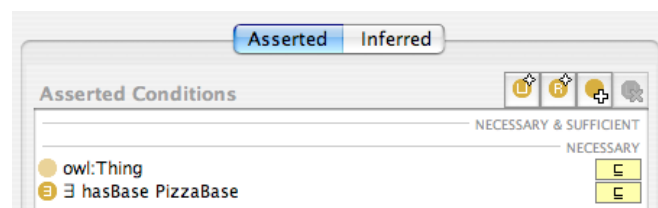
If an individual is a member of MargheritaPizza, it is **necessarily** a NamedPizza, and it **necessarily** has at least one mozzarella topping, and it **necessarily** has at least one tomato topping.

© Copyright The University of Manchester and Stanford University 2005

45

Necessary Conditions

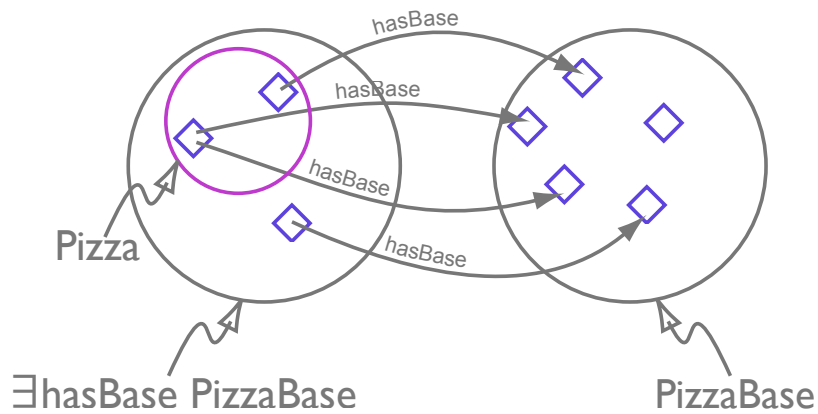
- For a given class, the necessary conditions represent **superclasses** of that class.
- Recall our description of a pizza...



© Copyright The University of Manchester and Stanford University 2005

46

Necessary Conditions on Pizza



© Copyright The University of Manchester and Stanford University 2005

47

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.

© Copyright The University of Manchester and Stanford University 2005

48

Necessary & Sufficient Conditions

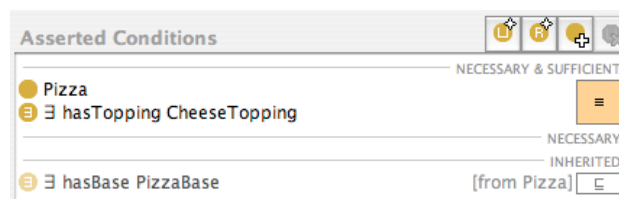
- Recall that **ClassA** is a subclass of **ClassB** if **all** individuals in **ClassA** are also in **ClassB**.
- Therefore, if **all** of the individuals in **ClassB** fulfil the **necessary & sufficient** conditions on **ClassA**, **all** of them must also be **members of ClassA**, and we can infer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

© Copyright The University of Manchester and Stanford University 2005

49

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.

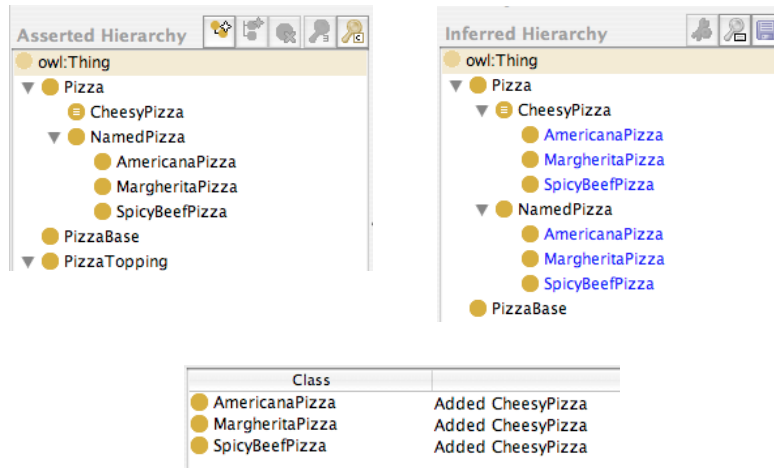


© Copyright The University of Manchester and Stanford University 2005

50

Necessary & Sufficient Conditions Example

- Classify the ontology...



© Copyright The University of Manchester and Stanford University 2005

51

Terminology

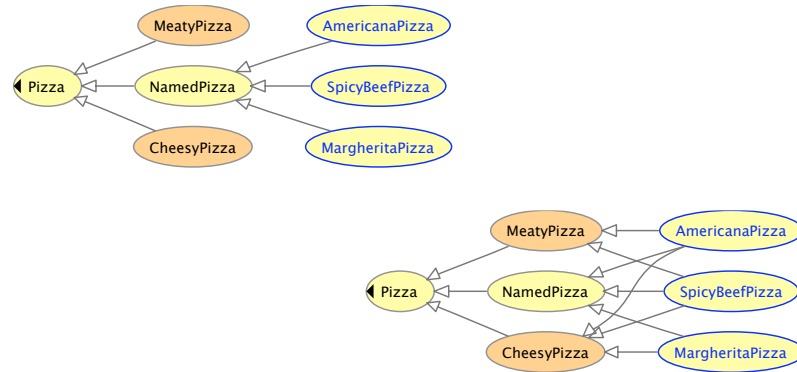
- Classes with only **Necessary** conditions are also known as **Primitive** classes.
- Classes with at least one set of **Necessary & Sufficient** conditions are known as **Defined** classes.
- Classes with only **Necessary** conditions are said to have a **Description**. Classes with at least one set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

© Copyright The University of Manchester and Stanford University 2005

52

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.



© Copyright The University of Manchester and Stanford University 2005

53

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.

© Copyright The University of Manchester and Stanford University 2005

54

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`?

© Copyright The University of Manchester and Stanford University 2005

55

Vegetarian Pizzas

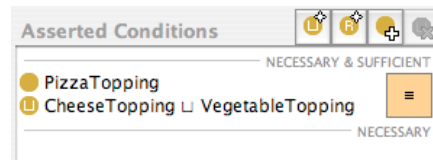
- We will define a `VegetarianPizza` to be any pizza that **only** has vegetarian toppings.
- To do this we need to decide what a vegetarian topping is...

© Copyright The University of Manchester and Stanford University 2005

56

Vegetarian Toppings

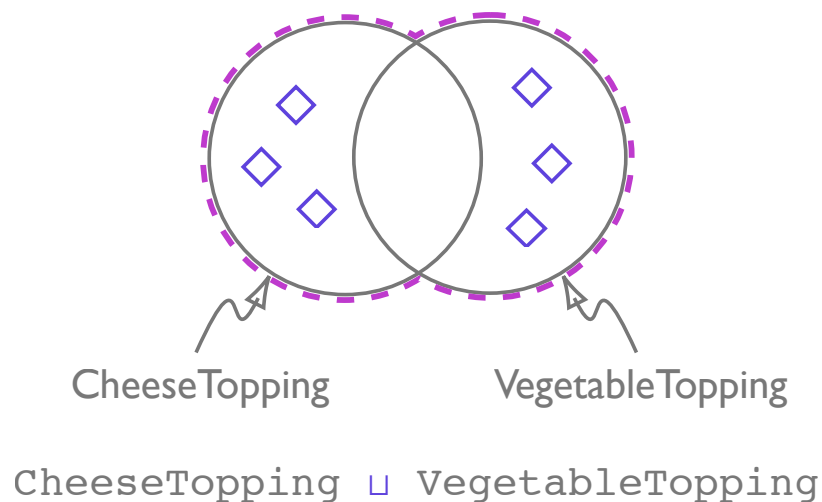
- A **VegetarianTopping** is any pizza topping that is either a **CheeseTopping** or **VegetableTopping**.
- We can use a **Union Class** to specify this.
- Create **VegetarianTopping** as a subclass of **PizzaTopping**, and add a necessary & sufficient condition to specify that **VegetarianToppings**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



© Copyright The University of Manchester and Stanford University 2005

57

Union Classes



© Copyright The University of Manchester and Stanford University 2005

58

Universal Restrictions

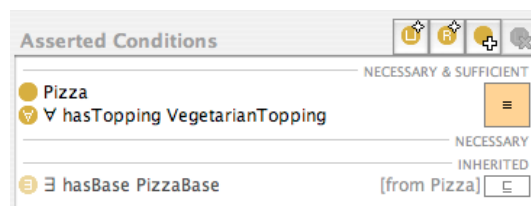
- We want to be able to say that `VegetarianPizzas` **only** have toppings that are `VegetarianToppings`.
- In order to do this we can use a **Universal Restriction**.
- Universal restrictions are written using the symbol \forall (upside down A).
- Universal restrictions mean 'all values from', or 'only'.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

© Copyright The University of Manchester and Stanford University 2005

59

Define VegetarianPizza

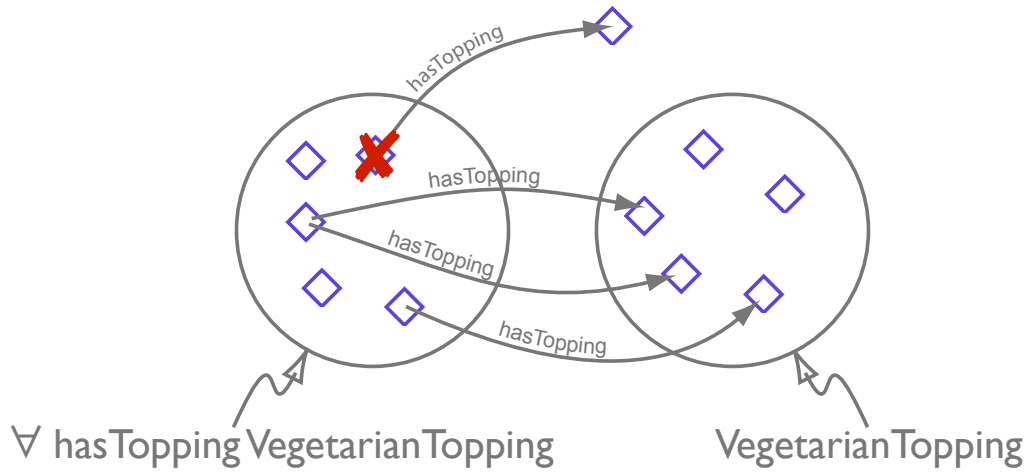
- Create a subclass of `Pizza` called `VegetarianPizza`. Add a **universal** restriction to specify that a `VegetarianPizza` **only** has toppings that are `VegetarianToppings`. Make `VegetarianPizza` a **defined** class.



© Copyright The University of Manchester and Stanford University 2005

60

Universal Restrictions



© Copyright The University of Manchester and Stanford University 2005

61

Subclasses of VegetarianPizza

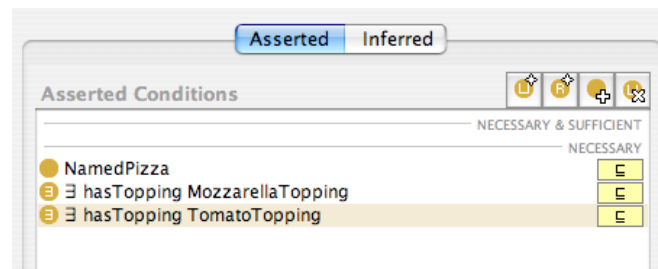
- Use the reasoner to classify the ontology.... what are the subclasses of VegetarianPizza?
- Is our definition of VegetarianPizza correct?

© Copyright The University of Manchester and Stanford University 2005

62

The Open World Assumption

- None of the pizzas have been classified as subclasses of VegetarianPizza.
- Let's revisit the description of MargheritaPizza...



© Copyright The University of Manchester and Stanford University 2005

63

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.

© Copyright The University of Manchester and Stanford University 2005

64

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.

© Copyright The University of Manchester and Stanford University 2005

65

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 screenshot shows a window titled "Asserted Conditions" with a toolbar containing icons for undo, redo, add, and delete. The window is divided into two sections: "NECESSARY & SUFFICIENT" and "NECESSARY". Under "NECESSARY & SUFFICIENT", there is a list of conditions: "NamedPizza", "∀ hasTopping (MozzarellaTopping ⊔ TomatoTopping)", "∃ hasTopping MozzarellaTopping", and "∃ hasTopping TomatoTopping". Each condition has a yellow button with a minus sign to its right. Under "NECESSARY", there is a condition "∃ hasBase PizzaBase" with a note "[from Pizza]" and a yellow button with a minus sign to its right.

The universal restriction is known as a **closure axiom**

© Copyright The University of Manchester and Stanford University 2005

66

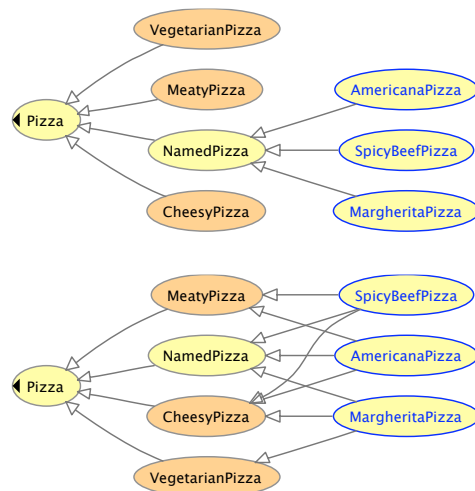
Add more closure axioms

- 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.

© Copyright The University of Manchester and Stanford University 2005

67

The asserted and inferred hierarchies



© Copyright The University of Manchester and Stanford University 2005

68

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**.

© Copyright The University of Manchester and Stanford University 2005

69

Namespace and Prefixes

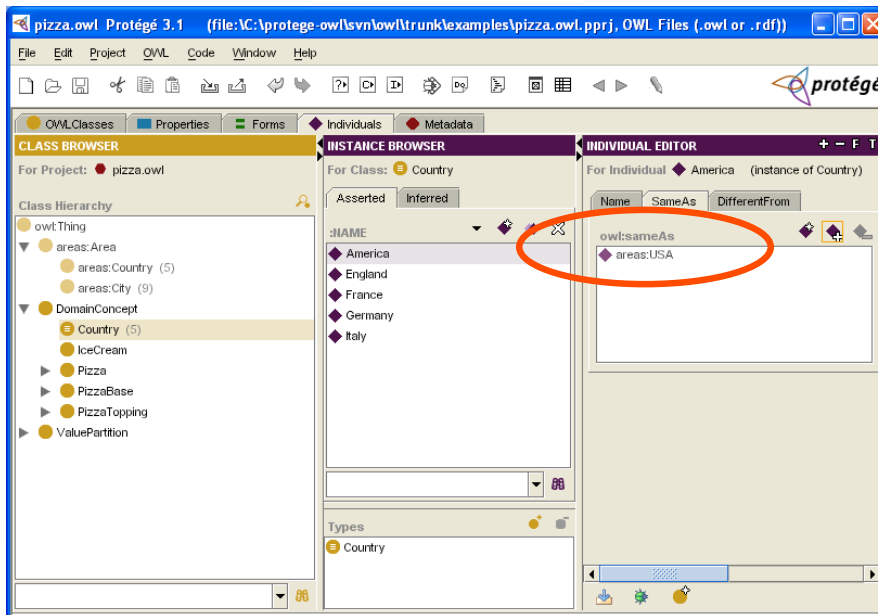
- Each ontology should have a unique default namespace.
- Imported concepts are prefixed.

Default Namespace			
<input type="text" value="http://www.co-ode.org/ontologies/pizza/2005/05/16/pizza.owl#"/>			
Namespace Prefixes		Imports	
Prefix	Namespace	Imported URI	Alias URL
xsd	http://www.w3.org/2001/XMLSchema#	http://protege.stanford.edu/plugins/owl/protege	
areas	http://www.owl-ontologies.com/areas.owl#	http://www.owl-ontologies.com/areas.owl	file:/C:/protege-owl/owl/areas.owl
dc	http://purl.org/dc/elements/1.1/		
protege	http://protege.stanford.edu/plugins/owl/protege#		
rdfs	http://www.w3.org/2000/01/rdf-schema#		
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#		
daml	http://www.daml.org/2001/03/daml+oil#		
owl	http://www.w3.org/2002/07/owl#		

© Copyright The University of Manchester and Stanford University 2005

70

Ontology Imports (3)

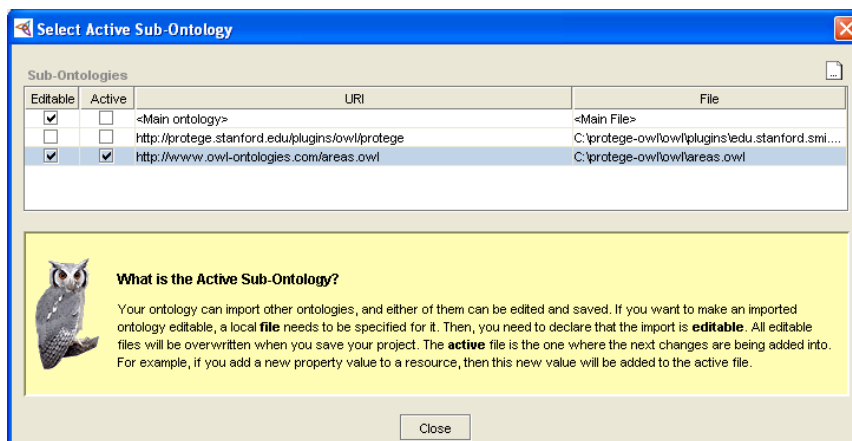


© Copyright The University of Manchester and Stanford University 2005

73

Multi-File Projects

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



© Copyright The University of Manchester and Stanford University 2005

74

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://...

© Copyright The University of Manchester and Stanford University 2005

75

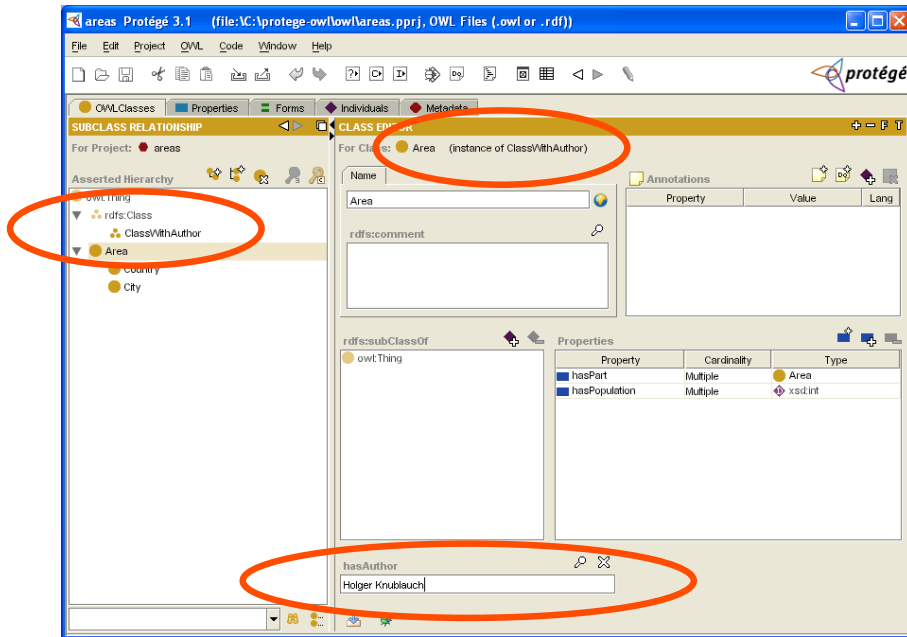
Namespace Pitfalls

- 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!

© Copyright The University of Manchester and Stanford University 2005

76

OWL-Full: Metaclasses



© Copyright The University of Manchester and Stanford University 2005

77

Using Metaclasses

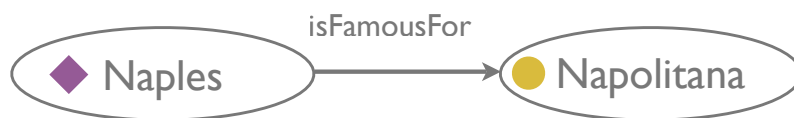
- Make rdfs:Class (or owl:Class) visible.
- Create Metaclass (subclass of rdfs:Class).
- Add properties to metaclass.
- Change type of classes:
 - Right click on class.
 - Go to Metaclasses/Change metaclass...
 - Select new type.

© Copyright The University of Manchester and Stanford University 2005

78

Classes as Values (1)

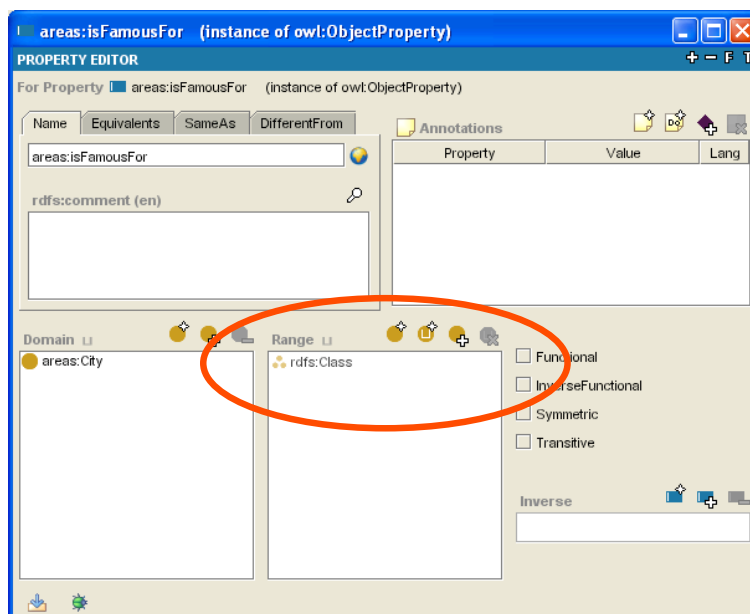
- All classes are instances of a metaclass.
- Properties can take classes as values (rdfs:range = rdfs:Class).
- For example, a city may be famous for a kind of pizza.



© Copyright The University of Manchester and Stanford University 2005

79

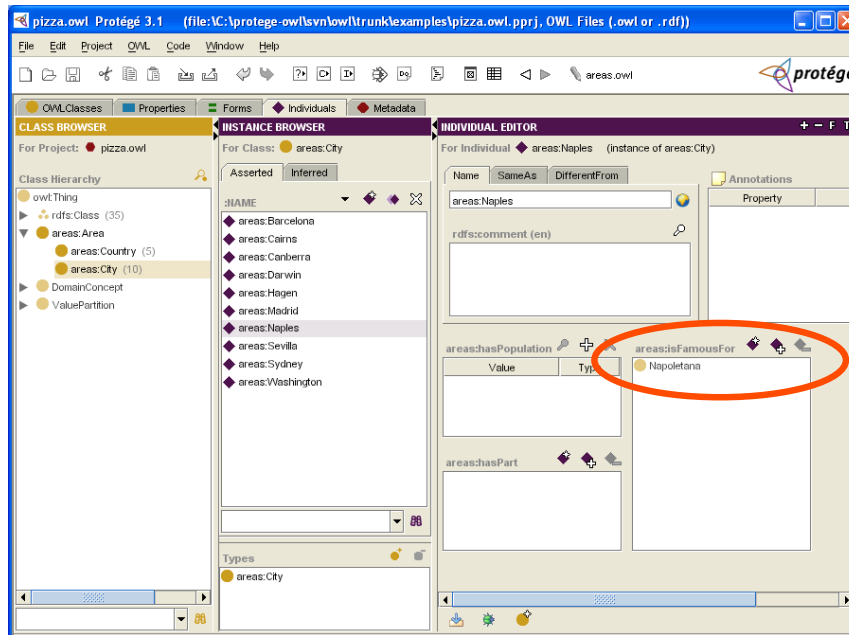
Classes as Values (2)



© Copyright The University of Manchester and Stanford University 2005

80

Classes as Values (3)



© Copyright The University of Manchester and Stanford University 2005

81

Links

- Protege-OWL web site:

<http://protege.stanford.edu/plugins/owl>

- CO-ODE web site:

<http://www.co-ode.org>

© Copyright The University of Manchester and Stanford University 2005

82