



Protégé

Classes, Properties and Instances

Susana R. de Novoa
UNIK4710

Protégé Tutorial : Overview

- Session 1: Basic Concepts
- Session 2: Tutorial Scenario
- Session 3: Exercises

Session 1: Basics Concepts

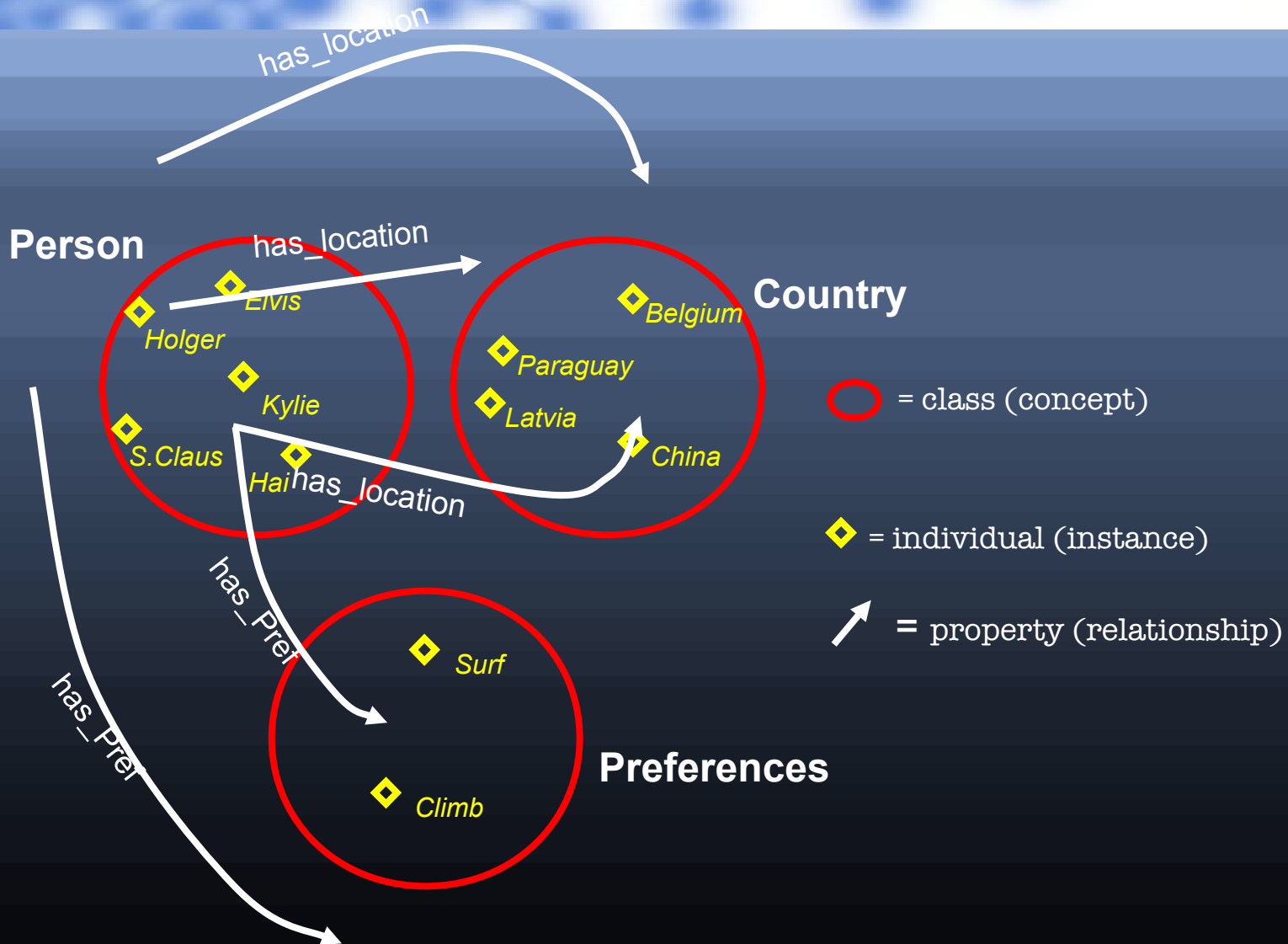
- Review of OWL
- Intro: Protégé

Review of OWL

What's inside an OWL ontology:

- Classes
 - Relations between classes (disjoints, equivalents...)
- Properties
 - Characteristics of properties (transitive, ...)
- Individuals

Review of OWL

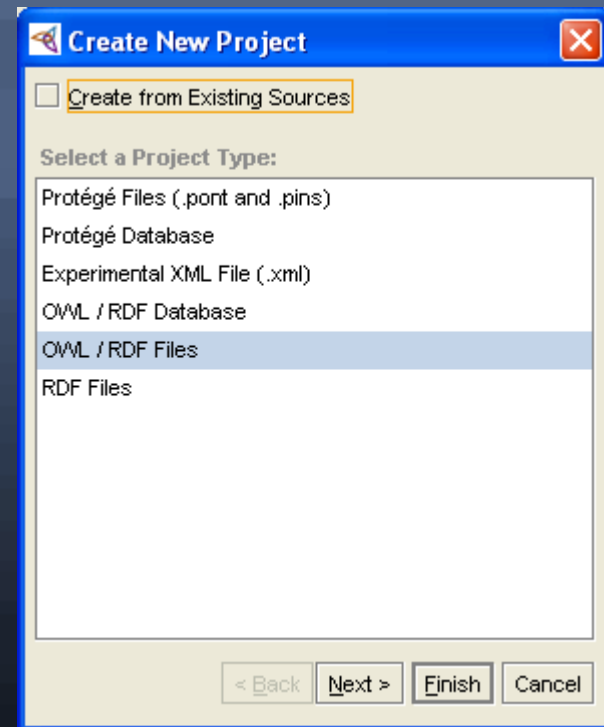
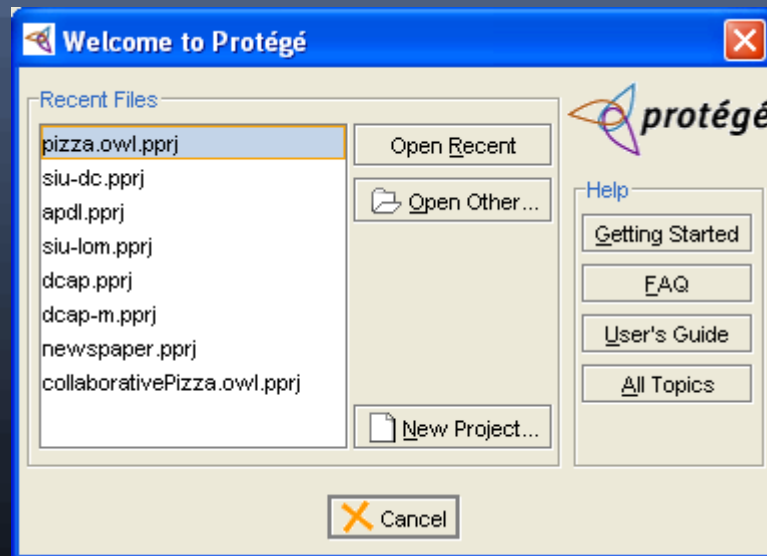


Intro: Protégé

- *Go to:*
<http://protege.stanford.edu/download/registered.html>
- *Download full Protégé 3.5 Beta*
- *Install the software*

Intro: Protégé

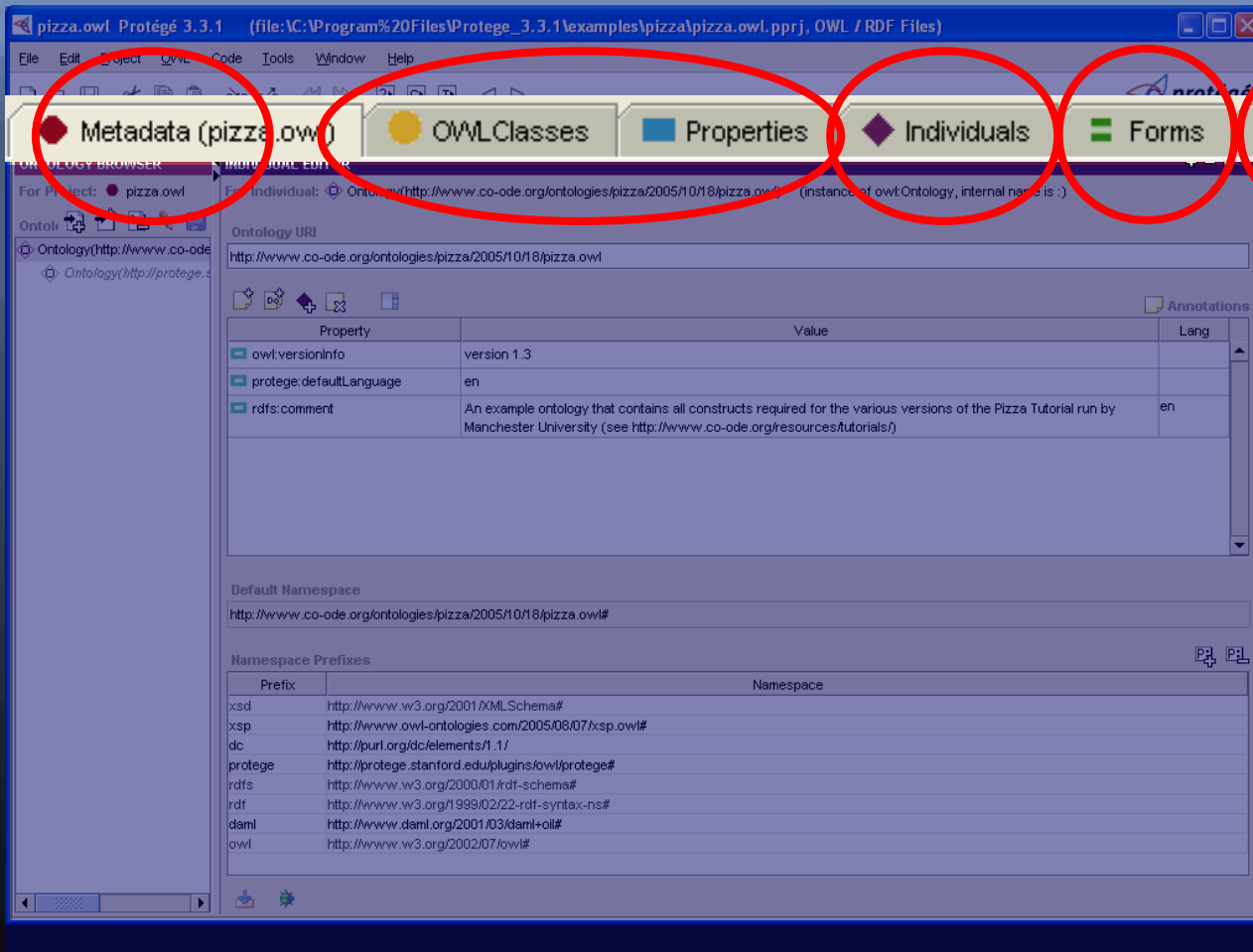
1. Select “New Project...”
2. Select “OWL/RDF Files”



Session 2: Tutorial Scenario

- Protégé Tabs
- Interface: Creating Classes
 - Concept: Disjointness
- Interface: Creating Properties
 - Concept: Characteristics of properties
 - Concept: Describing Classes
- Interface: Creating Conditions
 - Concept: Characteristics of conditions

Protégé Tabs



Create basic Ontology

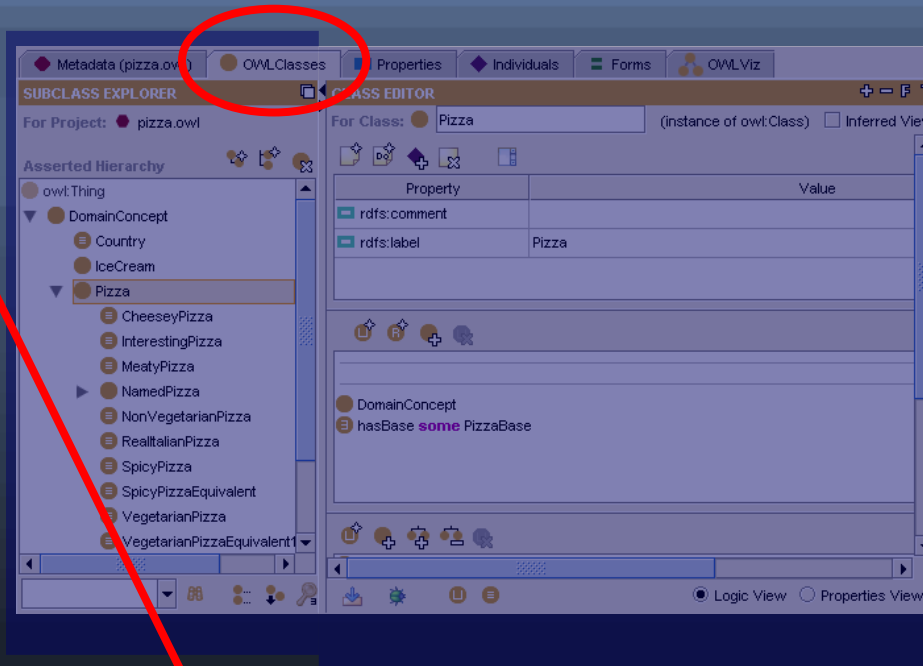
Changing the GUI

Populating the model

Top-level functionality

Extensions (visualisation)

Interface: Creating Classes



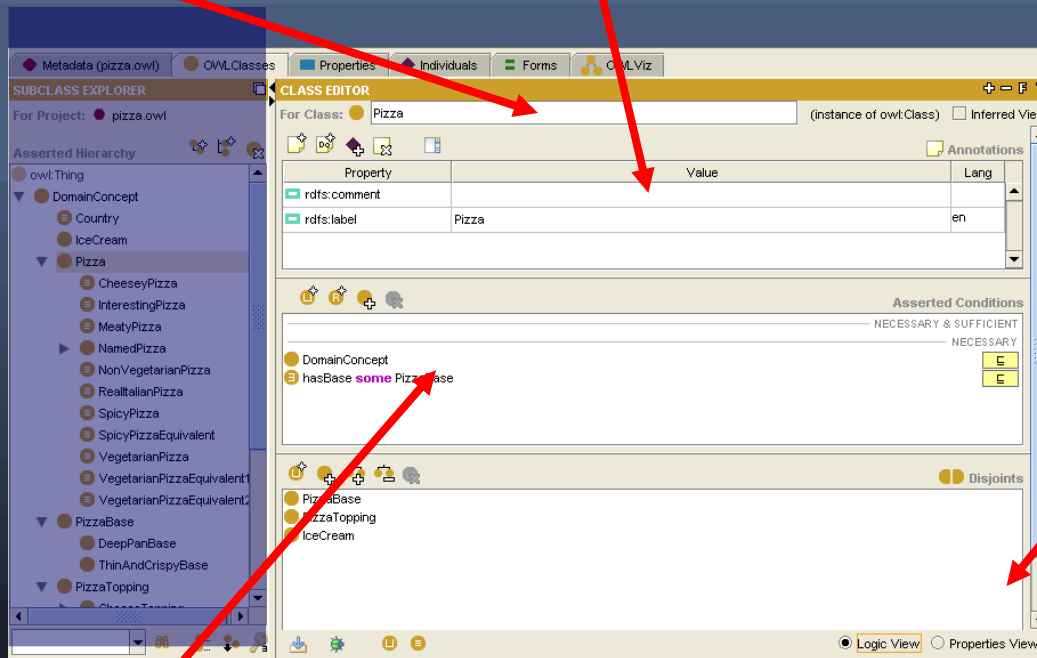
Create and Delete classes

Search for class

Interface: Creating Classes

Class name and documentation

Class annotations



Disjoints widget

Conditions Widget

Concept: Disjointness

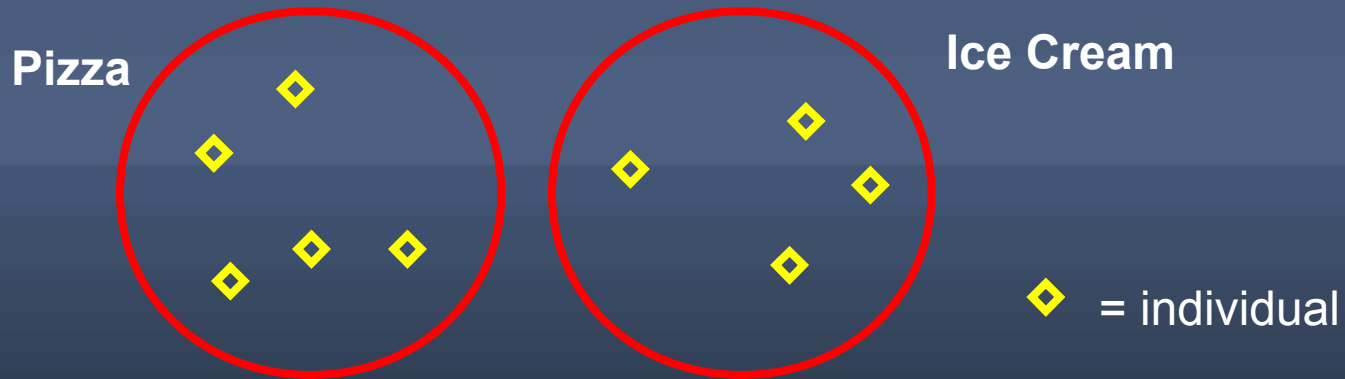
- All classes could potentially overlap



- This means an individual could be both a **Pizza** and a **Ice Cream** at the same time.

Concept: Disjointness

- If we state that classes are disjoint



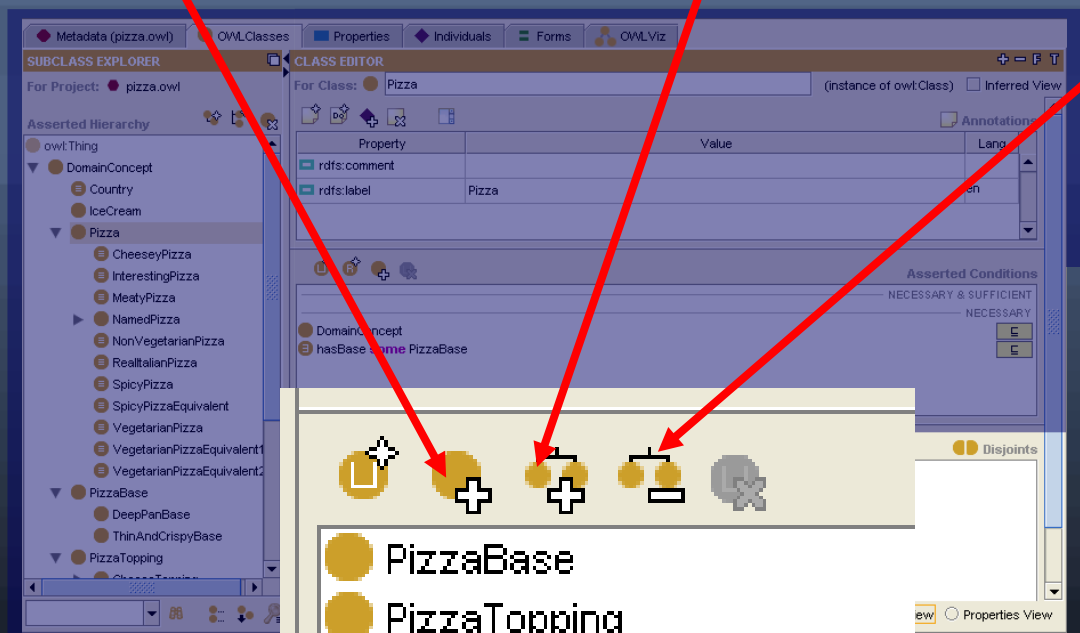
- This means an individual cannot be both a **Pizza** and a **Ice Cream** at the same time
- We must do this explicitly in the interface

Concept: Disjointness

Add siblings as disjoint

Add new disjoint

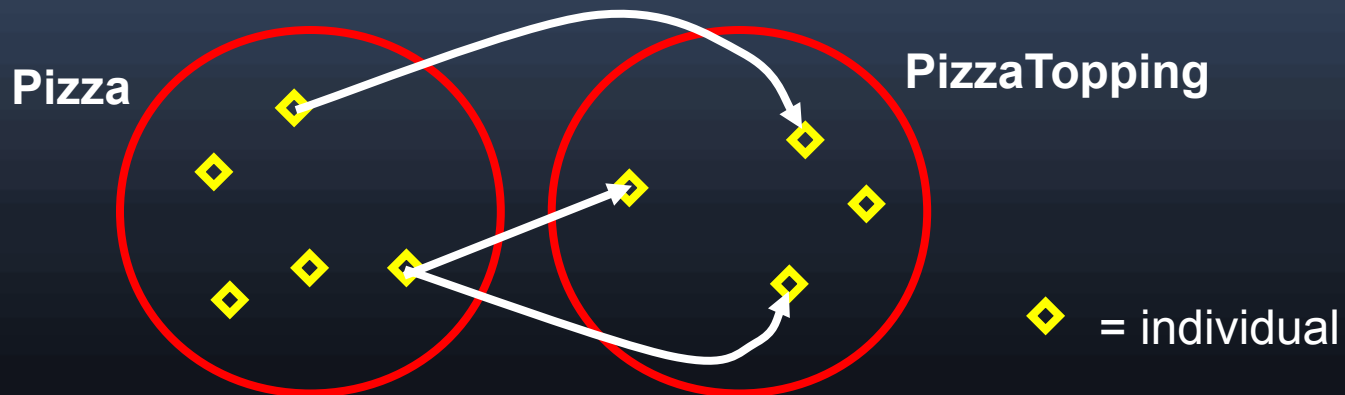
Remove disjoint siblings



List of disjoint classes

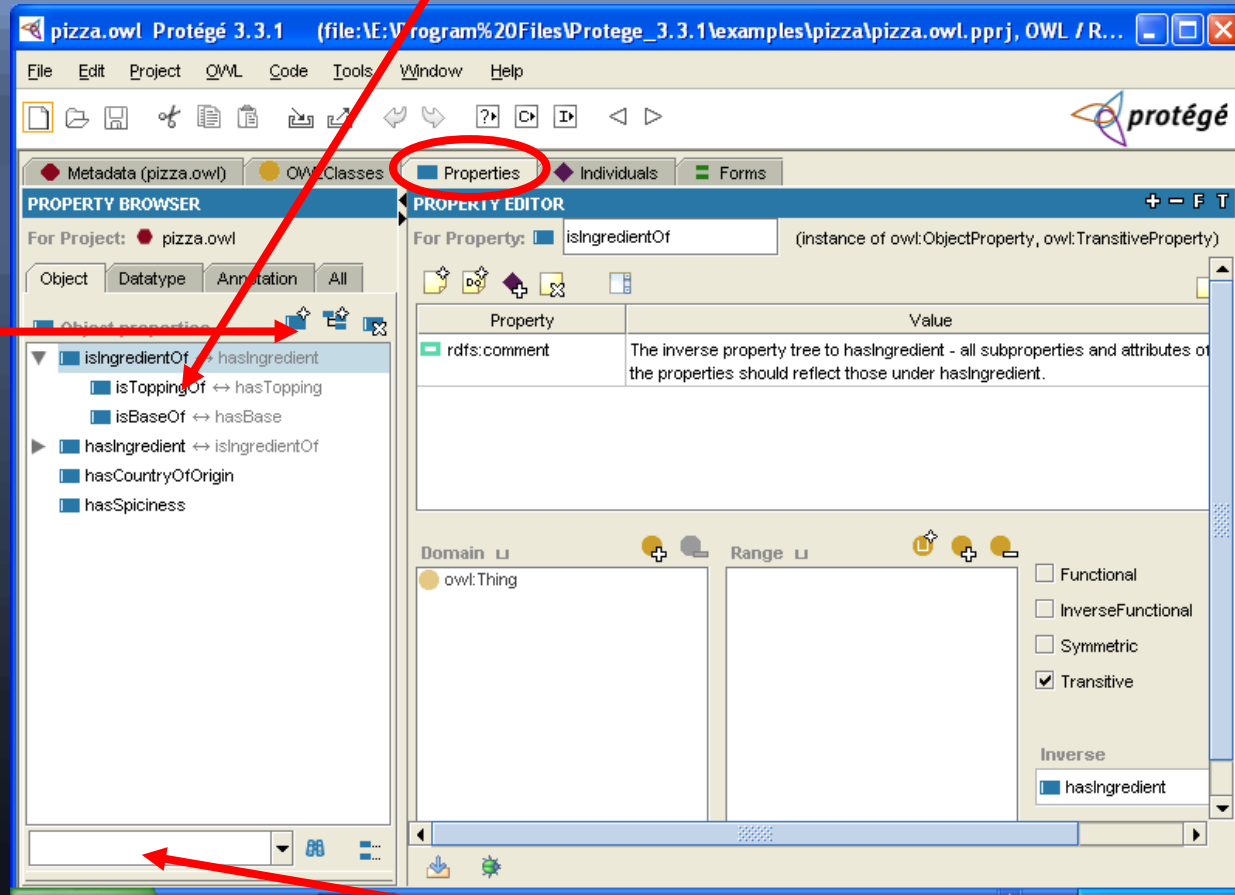
Interface: Creating Properties

- We want to say more about **Pizza** individuals, such as their relationship with other individuals
- We can do this with properties



Interface: Creating Properties

Properties can be in a hierarchy



Create property

Search for property

Concept: Characteristics of properties

Functional

There can be at most one individual (range) that is related to the domain individual via the property: ***“hasMother”***

The inverse property is functional:

“hasMother → hasChild”

InverseFunctional

$P(A,B) \rightarrow P(B,A)$

If a property **P** is symmetric, and the property relates individual A to individual B : ***“hasSibling”***

Symmetric

Transitive

$P(A,B) \text{ and } P(B,C) \rightarrow P(A,C)$

If a property **P** is transitive, and the property relates individual A to individual B, and also individual B to individual C: ***“hasAncestor”***

Concept: Describing Classes

PROPERTY BROWSER
For Project: pizza.owl

Object Datatype Annotation All

Object properties

- hasCountryOfOrigin
- hasIngredient ↔ isIngredientOf
 - hasBase ↔ isBaseOf
 - hasTopping ↔ isToppingOf
- hasSpiciness
- isIngredientOf ↔ hasIngredient

PROPERTY EDITOR for hasBase (instance of owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:ObjectPro... + - F T)
For Property: <http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl#hasBase>

Property	Value	Lang
rdfs:comment		

Domain: Pizza

Range: PizzaBase

Annotations: Functional, InverseFunctional, Symmetric, Transitive

Inverse: isBaseOf

```
graph LR; subgraph Pizza; direction TB; SpicyPizza; VegetarianPizza; end; subgraph PizzaBase; direction TB; DeepPanBase; ThinAndCrispyBase; end; Pizza -- hasBase --> PizzaBase; PizzaBase -- isBaseOf --> Pizza;
```

Interface: Creating Conditions

Conditions asserted by the ontology engineer

Add different types of condition

The screenshot shows an ontology editor interface. On the left, a tree view displays the class hierarchy for 'Pizza', including subclasses like 'VegetarianPizza' and 'PizzaBase'. The main window shows the 'CLASS EDITOR' for 'Pizza'. A pop-up window titled 'Asserted / Inferred' is overlaid on the editor, showing a list of 'ASSERTED CONDITIONS' for the class 'Pizza'. The conditions are: 'Pizza' (Necessary & Sufficient), '∃ hasTopping CheeseTopping' (Necessary), '∃ hasGreasiness GreaseLevel' (Necessary), and '∃ hasBase PizzaBase' (Inherited from Pizza). A red circle highlights the 'Asserted' and 'Inferred' tabs in the pop-up window. Red arrows point from the text labels to the corresponding elements in the interface.

Definition of the class

Description of the class

Conditions inherited from superclasses

Interface: Creating Conditions

Restricted Property

Filler Expression

Restriction Type

RESTRICTED PROPERTY:

- hasBase
- hasGreasyness
- hasTopping

RESTRICTION:

- allValuesFrom
- someValuesFrom
- hasValue
- cardinality
- minCardinality
- maxCardinality

FILLER:

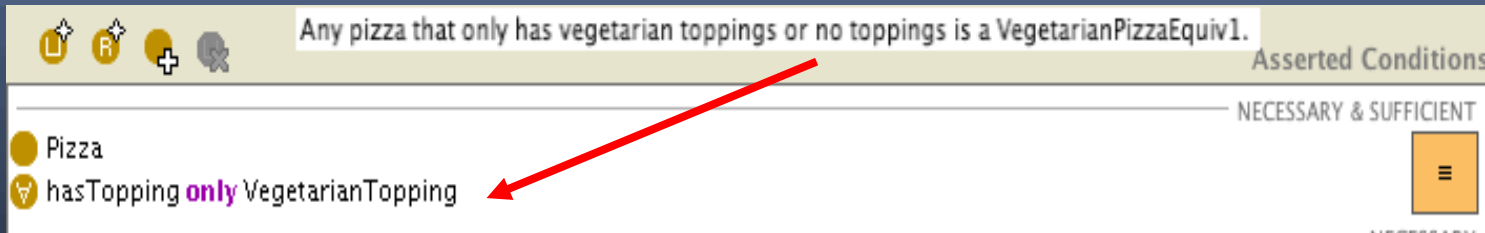
PizzaBase

OK Cancel

Concept: Characteristics of conditions

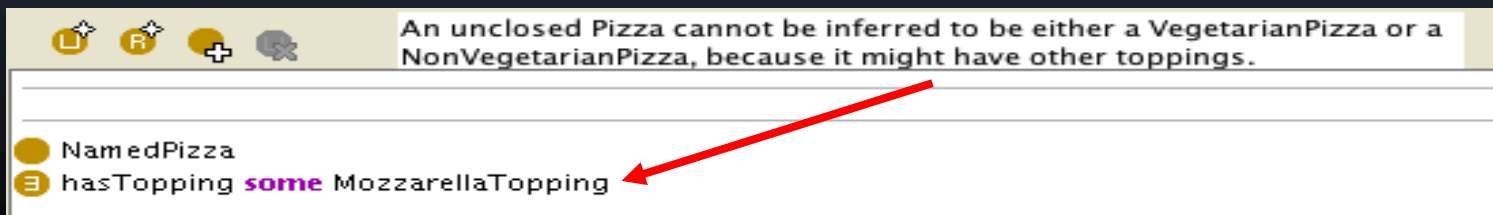
Define a condition for property values:

- 👉 AllValuesFrom: All values of the property must be of a certain type.



A screenshot of a condition editor interface. At the top, a text box contains the condition: "Any pizza that only has vegetarian toppings or no toppings is a VegetarianPizzaEquiv1." Below this, a list of conditions is shown. The first condition is "Pizza" with a yellow circle icon. The second condition is "hasTopping **only** VegetarianTopping" with a yellow circle icon containing a downward arrow. A red arrow points from the text box to the "only" keyword in the second condition. On the right side, there are labels "Asserted Conditions" and "NECESSARY & SUFFICIENT".

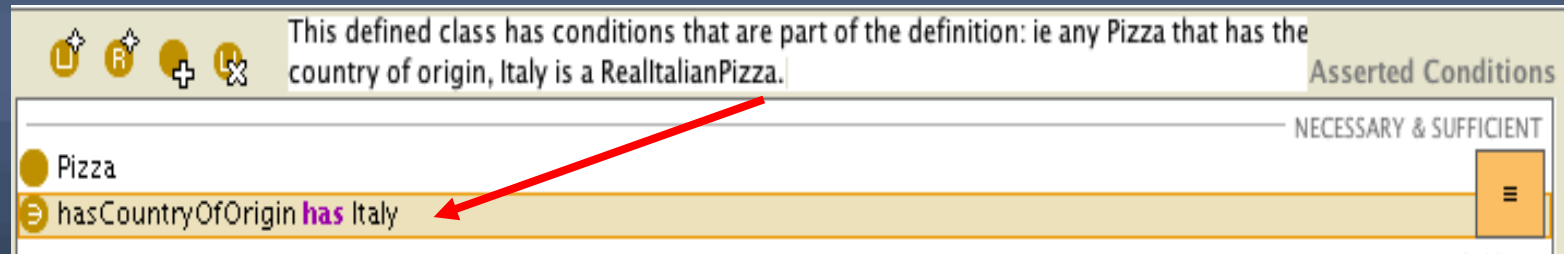
- 👉 SomeValuesFrom: At least one value of the property must be of a certain type



A screenshot of a condition editor interface. At the top, a text box contains the condition: "An unclosed Pizza cannot be inferred to be either a VegetarianPizza or a NonVegetarianPizza, because it might have other toppings." Below this, a list of conditions is shown. The first condition is "NamedPizza" with a yellow circle icon. The second condition is "hasTopping **some** MozzarellaTopping" with a yellow circle icon containing a downward arrow. A red arrow points from the text box to the "some" keyword in the second condition.

Concept: Characteristics of conditions

- HasValue : At least one of the values of the property is a certain value.



This defined class has conditions that are part of the definition: ie any Pizza that has the country of origin, Italy is a RealltalianPizza. Asserted Conditions

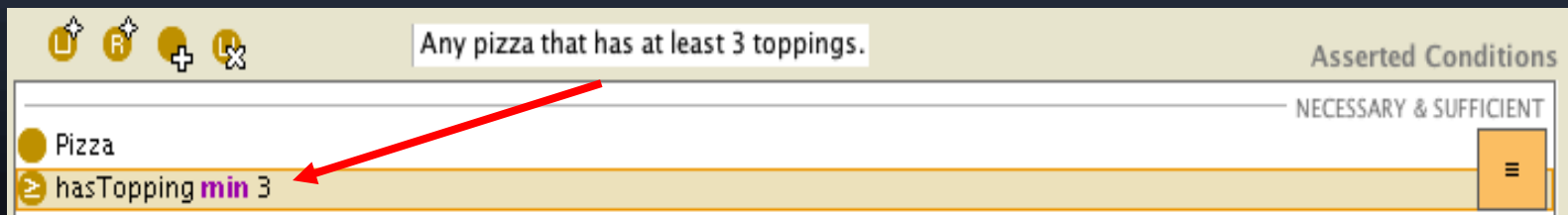
NECESSARY & SUFFICIENT

Pizza

hasCountryOfOrigin has Italy

A red arrow points from the text "any Pizza that has the country of origin, Italy is a RealltalianPizza." to the condition "hasCountryOfOrigin has Italy".

- Cardinality: The property must have at least/at most/exactly x values



Any pizza that has at least 3 toppings. Asserted Conditions

NECESSARY & SUFFICIENT

Pizza

hasTopping min 3

A red arrow points from the text "Any pizza that has at least 3 toppings." to the condition "hasTopping min 3".

Summary

You should now be able to:

- Identify components of the Protégé-OWL Interface
- Create Primitive Classes
- Create Properties
- Create some basic Restrictions and Conditions on a Class
- Next lesson: Properties & Rules.

Session 3: Exercises

Create a MargheritaPizza

Start with your existing ontology

1. Create a subclass of **Pizza** called **NamedPizza**
2. Create a subclass of **NamedPizza** called **MargheritaPizza**
3. Create a restriction to say that:
“Every MargheritaPizza must have at least one topping from TomatoTopping”
4. Create another restriction to say that:
“Every MargheritaPizza must have at least one topping from MozzarellaTopping”

Session 3: Exercises

Create other pizzas

Start with your existing ontology

1. *Add more topping ingredients as subclasses of `PizzaTopping`*
Use the hierarchy, but be aware of disjoints
2. *Create more subclasses of **NamedPizza***
3. *Create a restrictions on these pizzas to describe their ingredients*
4. *Save this for the next session*

Thank You

- Feedback on tutorial appreciated
- Software / resources / community at:
 - <http://www.co-ode.org/>
 - <http://protege.stanford.edu/>