Defining OWL Axioms in Protégé

Nicolò Pratelli

Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche – Pisa

UNIVERSITÀ DI PISA

A.A. 2024-2025

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"

In the past seminar

In the last seminar, you learned how to:

- create an OWL ontology in Protégé
- set the ontology IRI and the prefixes
- create **classes** and their **subclasses**
- create **properties** and their **sub-properties**
- set property **domain** and **range**
- create **individuals** and set their class

OWL Axioms

Today we will see how to **define OWL axioms** in the desktop version of Protégé.

Class Expressions

Class expressions are used to define the **intension** of classes in a more specific way than was possible with RDFS

For example, the class **Researcher** could be defined as "**Person who has a contract with 1 research organization and has 1 researcher ID**"

Class expressions include:

- Set-theoretic expressions
- Property restrictions
- Property cardinality restrictions

Set-Theoretic Class Expressions

Set-theoretic class expressions are used to combine classes using set-theory operators.

In today's exercise we will use:

- **unionOf**: class A contains all individuals of B plus all individuals of C, and nothing else

  Example: class Animal is the union of Human and NonHumanAnimal

- **intersectionOf**: class A contains only the individuals that are at the same time in B and in C

  Example: class Smartwatch is the intersection of Watch and Computer

- **complementOf**: class A contains all individuals that are not in class B

  Example: class LivingThing is the complement of NonLivingThing

Class Expressions in Protégé

To define union, intersection, and complement in Protégé, you have to use the **class expression editor**:

- union with keyword **or**
- intersection with keyword **and**
- complement with keyword **not**

Property Restrictions

Property restrictions are class expressions that **restrict the values of a property** when it is applied to a certain class

In today's exercise we will use two types of property restrictions:

- **allValuesFrom** (universal quantifier, called **only** in Protégé)
- **someValuesFrom** (existential quantifier, called **some** in Protégé)

Property Restrictions Example

To state that "MargheritaPizza **must have** a RedBase **and two specific** toppings (MozzarellaTopping and BasilTopping)":

- MargheritaPizza hasBase only RedBase → *the base of MargheritaPizza can be RedBase and no other*
- MargheritaPizza hasTopping only (MozzarellaTopping or BasilTopping) → *the topping of MargheritaPizza can be MozzarellaTopping or BasilTopping, or both, and no other*
- MargheritaPizza hasTopping some MozzarellaTopping → *MargheritaPizza must have at least one MozzarellaTopping*
- MargheritaPizza hasTopping some BasilTopping → *MargheritaPizza must have at least one BasilTopping*

Cardinality Restrictions

- **owl:minCardinality**: when property P is applied to class A, each member of class A must be connected to at least n individuals using P

Example: Parent hasChild min1 Child

- **owl:maxCardinality**: when property P is applied to class A, each member of class A must be connected to at most n individuals using P

Example: NurserySchoolClass hasStudent max 28 Student

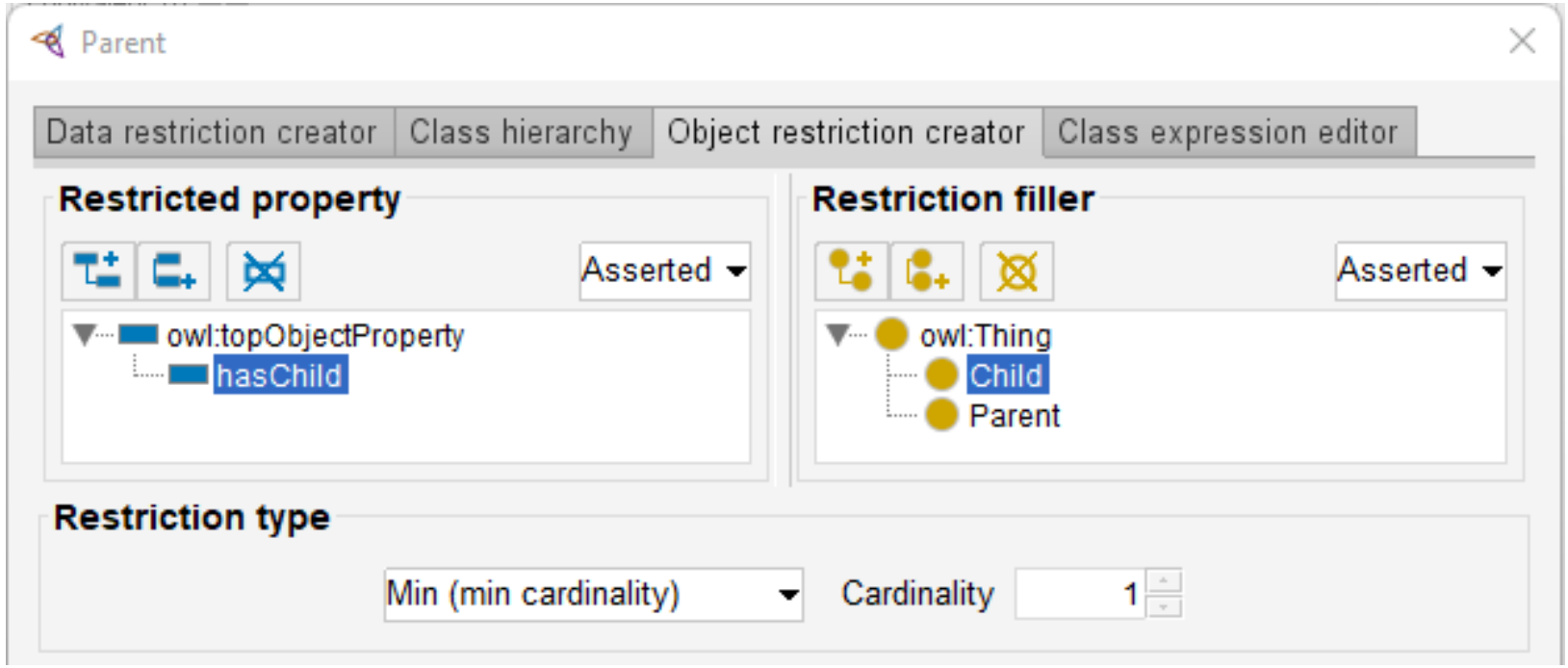
- **owl:cardinality**: when property P is applied to class A, each member of class A must be connected to exactly n individuals using P

Example: Human hasHeart exactly 1 Heart

Property Restrictions in Protégé

To define property restrictions in Protégé, you have to use the **object restriction creator** or the data restriction creator (depending on the type of property)

Select the property, the class of the object of the property, the restriction type, and for cardinality set the number



Class Expression Axioms

Class expression axioms establish relationships between class expressions, and they include:

- **subClassOf**: all individuals in class A are also in class B

  Example: class Pizza is subclass of class Food

- **equivalentClass**: all individuals in class A are also in class B, and all individuals in B are also in A

  Example: class Stream is equivalent to class Creek

- **disjointWith**: all individual in class A are not in B, and all individuals in B are not in A

  Example: class Mountain is disjoint with class Sea

- **disjointUnionOf**: class A contains all individuals of B plus all individuals of C, and B and C are disjoint

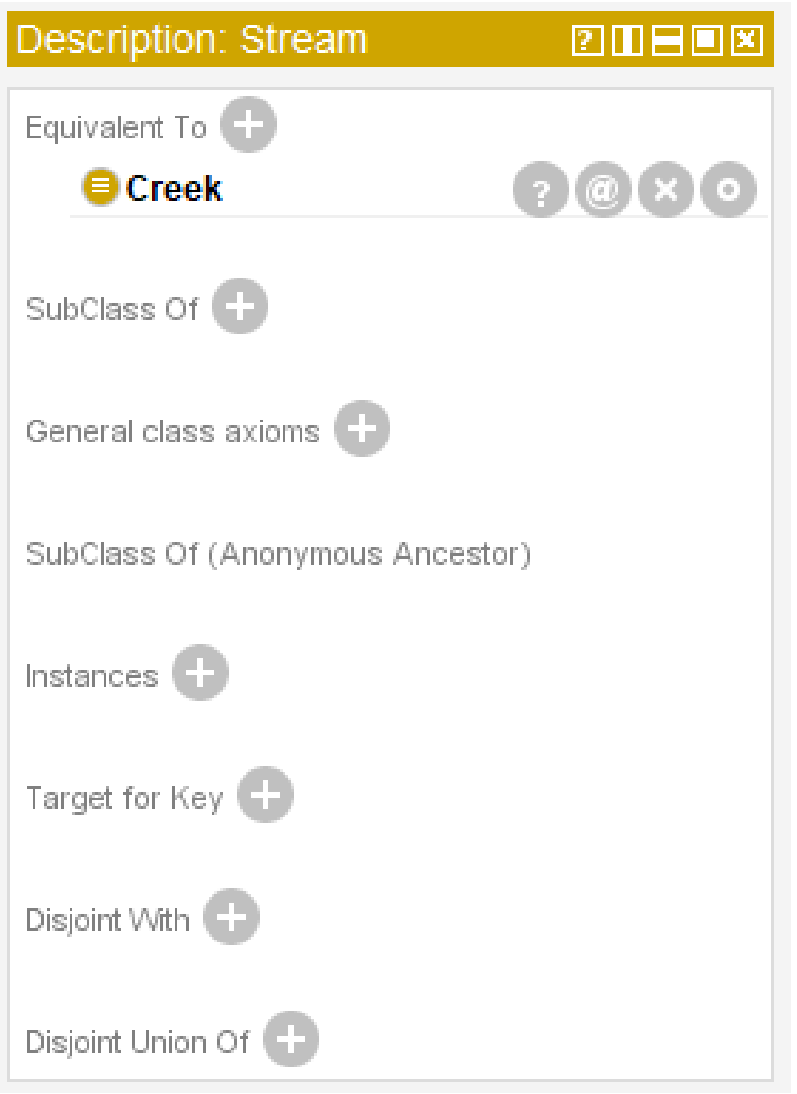
  Example: class Animal is the disjoint union of Human and NonHumanAnimal

Class Expression Axioms in Protégé

In Protégé, you can use buttons in the class description view to define axioms of:

- **subclass**
- **equivalence**
- **disjointness**
- **disjoint union**

Then, select the appropriate class in the class hierarchy or use the expression editor or restriction creator

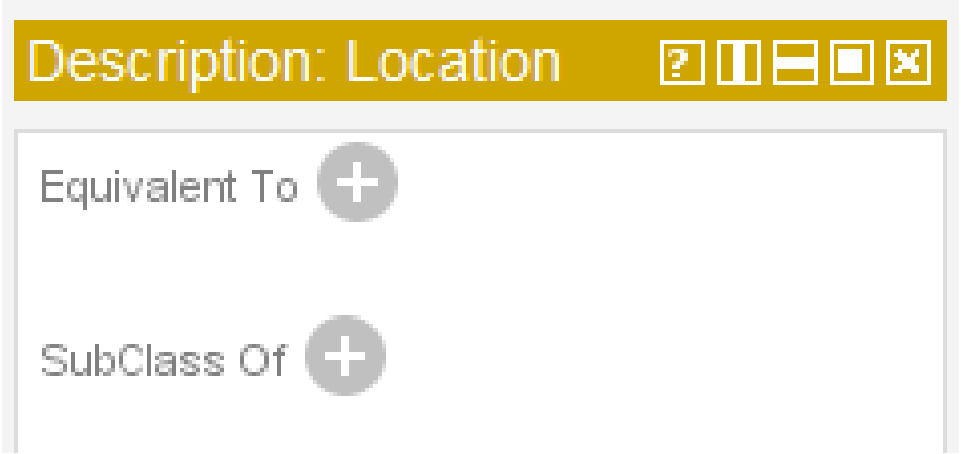


Property Restrictions in Protégé

Property restrictions can be applied using **subClassOf** axioms or **equivalentClass** axioms

An axiom of equivalentClass is much more stringent, so it is easier to make errors (e.g. if there are edge cases)

In general, unless the exercise explicitly states (or strongly suggests) equivalence, use subClassOf axioms for your restrictions



Property Axioms

Today we will see the following property axioms:

- **inverseOf**
- **property characteristics**

The inverseOf axioms allows you to define the inverse of a property

For example:

parent1 isParentOf child1

child1 isChildOf parent1

isParentOf inverseOf isChildOf

Property Characteristics (1)

- **Functional**: property P is functional if it can have at most one value. If multiple individuals are specified as values for the property, they will be inferred to denote the same object

  Example: luisaVerdi hasFiscalCode "ABCDEF12G34H567I"

- **Inverse Functional**: the inverse of property P is functional

  Example: universityOfPisa hasDean riccardoZucchi

- **Transitive**: if individual x is related to individual y via property P, and individual y is related to individual z via P, then individual x is related to individual z via P

Example: europe hasPart france, france hasPart provence → europe hasPart provence

Property Characteristics (2)

- **Symmetric**: property P has itself as an inverse, so if individual x is related to individual y, then y is related to x

  Example: marioRossi isRelativeOf pieroBianchi

- **Asymmetric**: if individual x is related to individual y via P, then y is not related to x via P

  Example: isChildOf

- **Reflexive**: every individual is related to itself via property P

  Example: lauraRossi knows lauraRossi

- **Irreflexive**: no individual is related to itself via property P

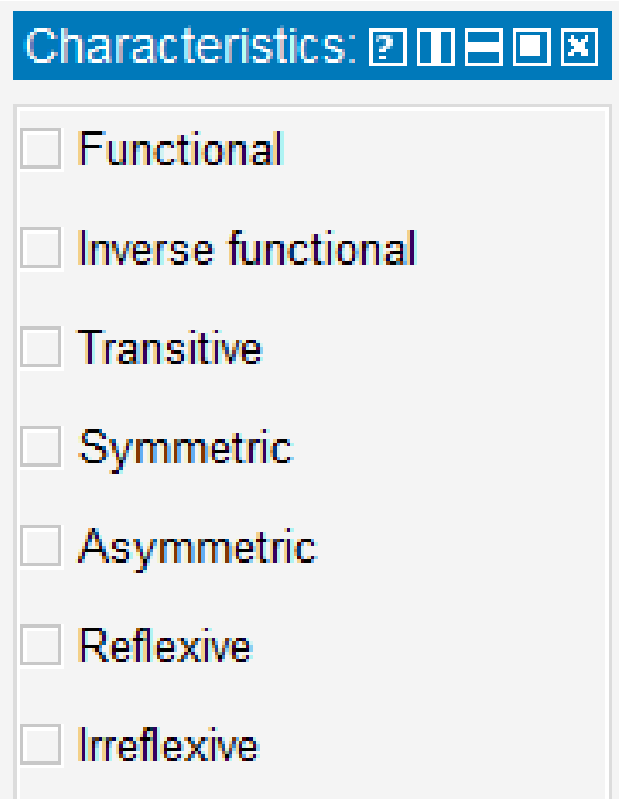
  Example: getMarried

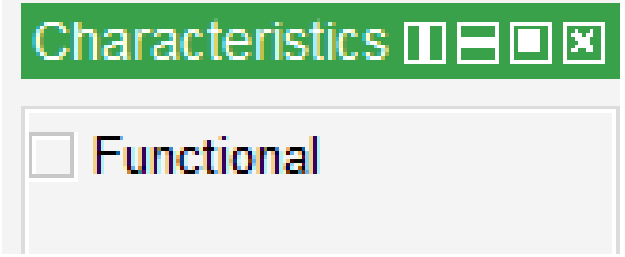
Property Characteristics in Protégé

To define property characteristics in Protégé, select a property and then use the Characteristics view

By selecting each checkbox you can activate the corresponding characteristic

Remember that object properties can have 7 characteristics, while data properties can have only 1 (functional)





Instructions for the Exercise

- Today we will add several axioms to the ontology about pizza that we created in the previous seminar (based on the original Pizza Ontology by Drummond, Horridge, Wroe & Steven)
- Open the ontology from the last seminar and define the
- axioms that you can find in the following slides, using:
  - Set-theoretic class expressions
  - Property restrictions (including cardinality)
  - Class expression axioms
  - Property axioms
- Save the resulting knowledge base in Turtle format and ask us to verify its correctness after you have finished

Exercise (1/4)

- Define three new **subclasses** of *Pizza*: *VegetarianPizza*, *NonVegetarianPizza*, *TastyPizza*
- Define the following OWL axioms:
  - *Pizza* is disjoint with *IceCream*, *Salad*, *PizzaBase*, and *PizzaTopping*
  - All subclasses of *NamedPizza* are disjoint with each other
  - All subclasses of *PizzaTopping* are disjoint with each other
  - *RedBase* is disjoint with *WhiteBase*
  - Each *Pizza* must have one and only one *PizzaBase*

Exercise (2/4)

- Define the following OWL axioms:
  - *MargheritaPizza* has a *RedBase* and the following toppings: *MozzarellaTopping*, *BasilTopping*
  - *MarinaraPizza* has a *RedBase* and the following toppings: *GarlicTopping*, *BasilTopping*
  - *HamMushroomPizza* has a *RedBase* and the following toppings: *MozzarellaTopping*, *HamTopping*, *MushroomTopping*
  - *PisanPizza* has a *RedBase* and the following toppings: *ParmigianoTopping*, *AnchoviesTopping*, *CaperTopping*

Exercise (3/4)

- Define the following OWL axioms:
  - *FourCheesePizza* has a *WhiteBase* and 4 *CheeseTopping*
  - *FourSeasonPizza* has a *RedBase* and the following toppings: *MozzarellaTopping*, *MushroomTopping*, *ArtichokeTopping*, *HamTopping*, *OliveTopping*
  - *NonVegetarianPizza* is equivalent to *Pizza* that has *MeatTopping* or *FishTopping*
  - *VegetarianPizza* is equivalent to a Pizza that is not a *NonVegetarianPizza*

Exercise (4/4)

- Define the following OWL axioms:
  - *Pizza* is the disjoint union of *VegetarianPizza* and *NonVegetarianPizza*
  - *TastyPizza* is equivalent to a *Pizza* with minimum 3 *PizzaTopping*
- Define a new data property called **hasPrice**, with domain *Pizza* and range *xsd:integer*
- For each object property and each data property, set the appropriate characteristics
- For each object property, define a new property and set it as its inverse