



Centro de Estudios en
Microelectrónica y Sistemas
Distribuidos (CEMISID)



Construcción de una ontología OWL con protégé 4

Material: A Practical Guide To Building OWL Ontologies Using
Protégé 4. 2011

Tania Rodríguez
Jose Aguilar

Merida Junio 2012

CONSTRUCCIÓN DE ONTOLOGÍA OWL

- Las ontologías son usadas para capturar el conocimiento sobre algún dominio de interés.
- Una ontología describe los conceptos dentro del dominio y la relación que tiene entre esos conceptos.
- Un lenguaje estándar para hacer ontologías es OWL desarrollado por W3C.
- OWL permite describir conceptos y además cuenta con un conjunto de operadores (intercesión, unión, y negación).
- OWL esta basado en lógica descriptiva que permite el uso de un razonador.

COMPONENTES DE UNA ONTOLOGÍA OWL

Ontologías	OWL	PROTÉGÉ
Instancias	Individuos	Casos (instance)
Relaciones	Propiedades	Slots
Conceptos	Clases	Clases

INDIVIDUOS EN OWL

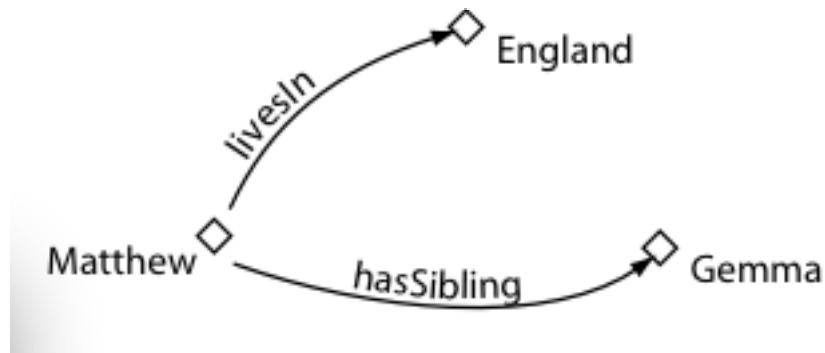
- Los individuos representan objetos del dominio de interés y son también conocidos como instancias.



Representación de Individuos

PROPIEDADES EN OWL

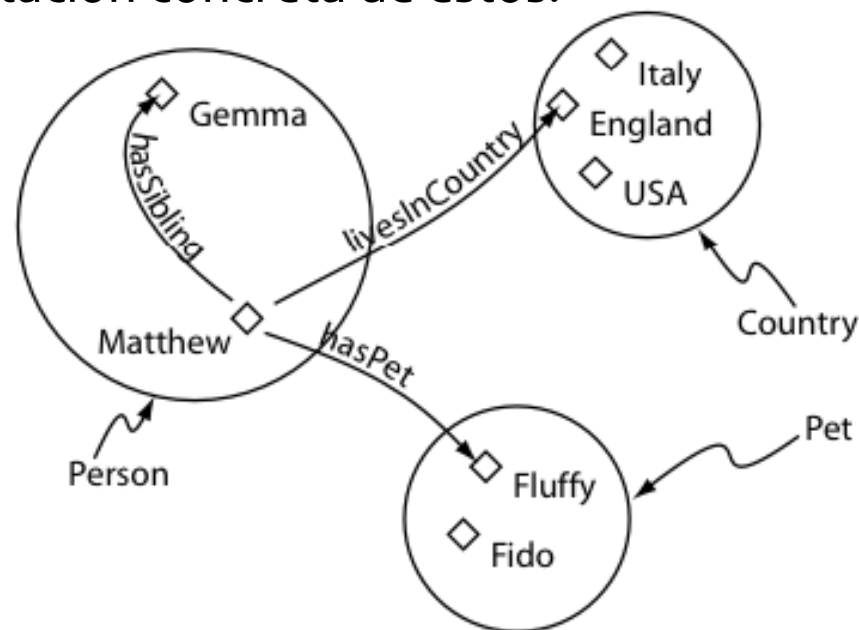
- Las propiedades son relaciones binarias sobre los individuos y pueden ser inversas, transitivas o simétricas.



Representación de Propiedades

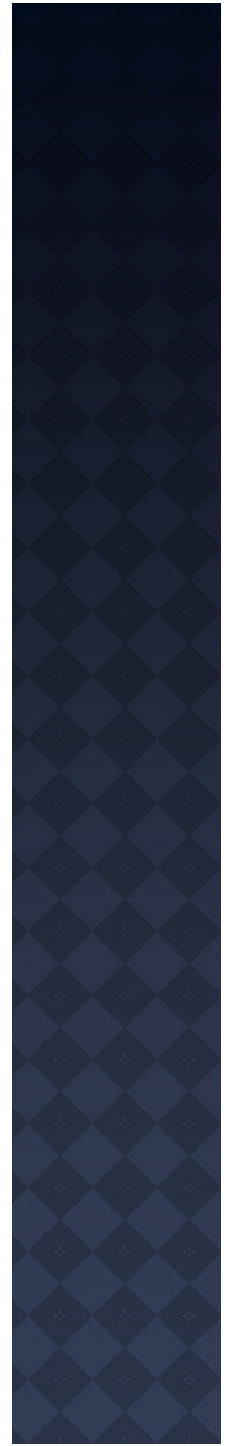
CLASES EN OWL

- Las clases OWL se entienden como conjuntos que contienen individuos y pueden ser organizadas dentro de una jerarquía de clases y subclases conocida como taxonomía. Las clases también son conocidas como conceptos, pues son una representación concreta de éstos.



Representación de clases

LÓGICA DESCRIPTIVA Y OWL



LÓGICA DE PRIMER ORDEN

- ⦿ El fundamento que garantiza la pureza lógica de la ontologías es la lógica de primer orden. Sobre ella se asienta las lógicas descriptivas (DL), así como OWL.
- ⦿ Porque usamos lógicas descriptivas:
 - Lógica de primer orden es indecidible (es fácil afirmar cosas de objetos, pero computacionalmente complejo)
 - Se requiere de un lenguaje formal para construir y combinar definiciones de categorías (p.ej. Relaciones de subconjunto y superconjunto)
 - Razonadores semánticos se basan en ella: FaCT++, Rancer, Pellet, ...

LÓGICA DESCRIPTIVA

- ⦿ Lenguajes de representación del conocimiento
- ⦿ DL se diseñó como una extensión de marcos y redes semánticas, equipada con semántica basada en lógica.
- ⦿ Características :
 - Un formalismo descriptivo: conceptos, roles (relaciones), individuos.
 - Un formalismo terminológico: axiomas que describen propiedades genéricas.
 - Un formalismo asertivo: introduce propiedades de individuos.

LÓGICA DESCRIPTIVA

⦿ Principales tareas de inferencia con lógica descriptiva:

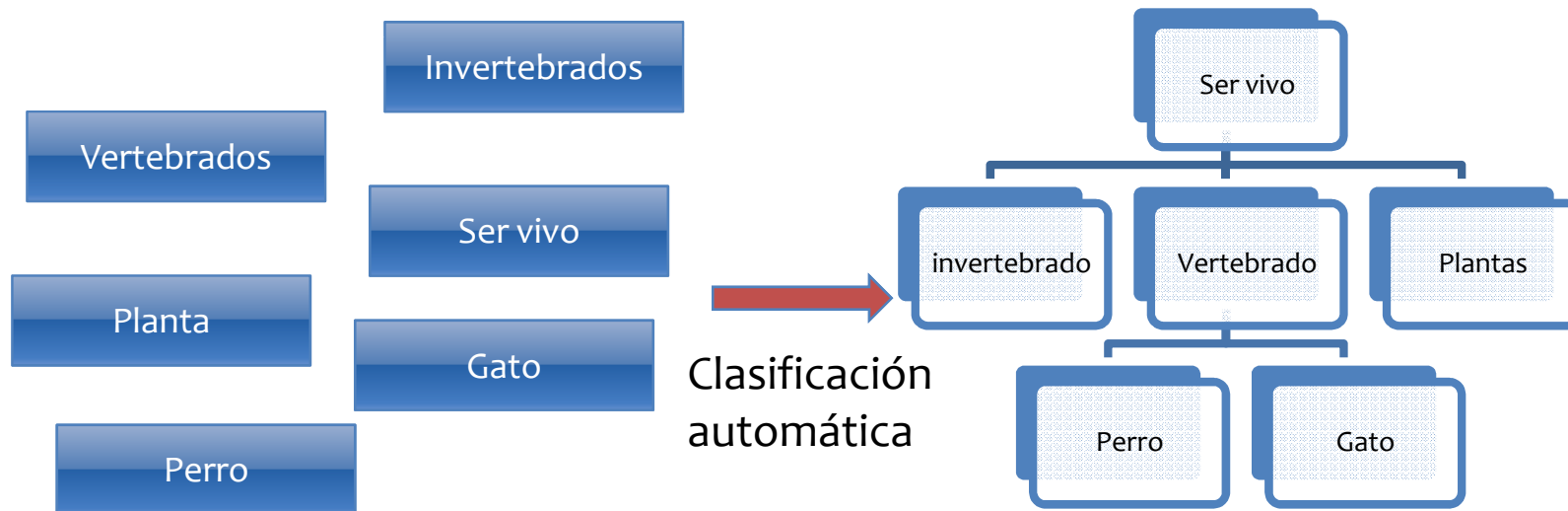
- Subsunción (comprobar si una categoría es subconjunto de otra)
- Clasificación (comprobar si un objeto pertenece a una categoría)

⦿ Ejemplo:

- Soltero = $Y(\text{NoCasado}, \text{Adulto}, \text{Masculino})$
- $\text{Soltero}(x) \Rightarrow \text{NoCasado}(x) \wedge \text{adulto}(x) \wedge \text{masculino}(x)$
(lógica de primer orden)

LÓGICAS DESCRIPTIVAS

- ◉ Ejemplo: realiza clasificación automática (realizada por el motor de inferencias del lenguaje-razonador) en tiempo de ejecución



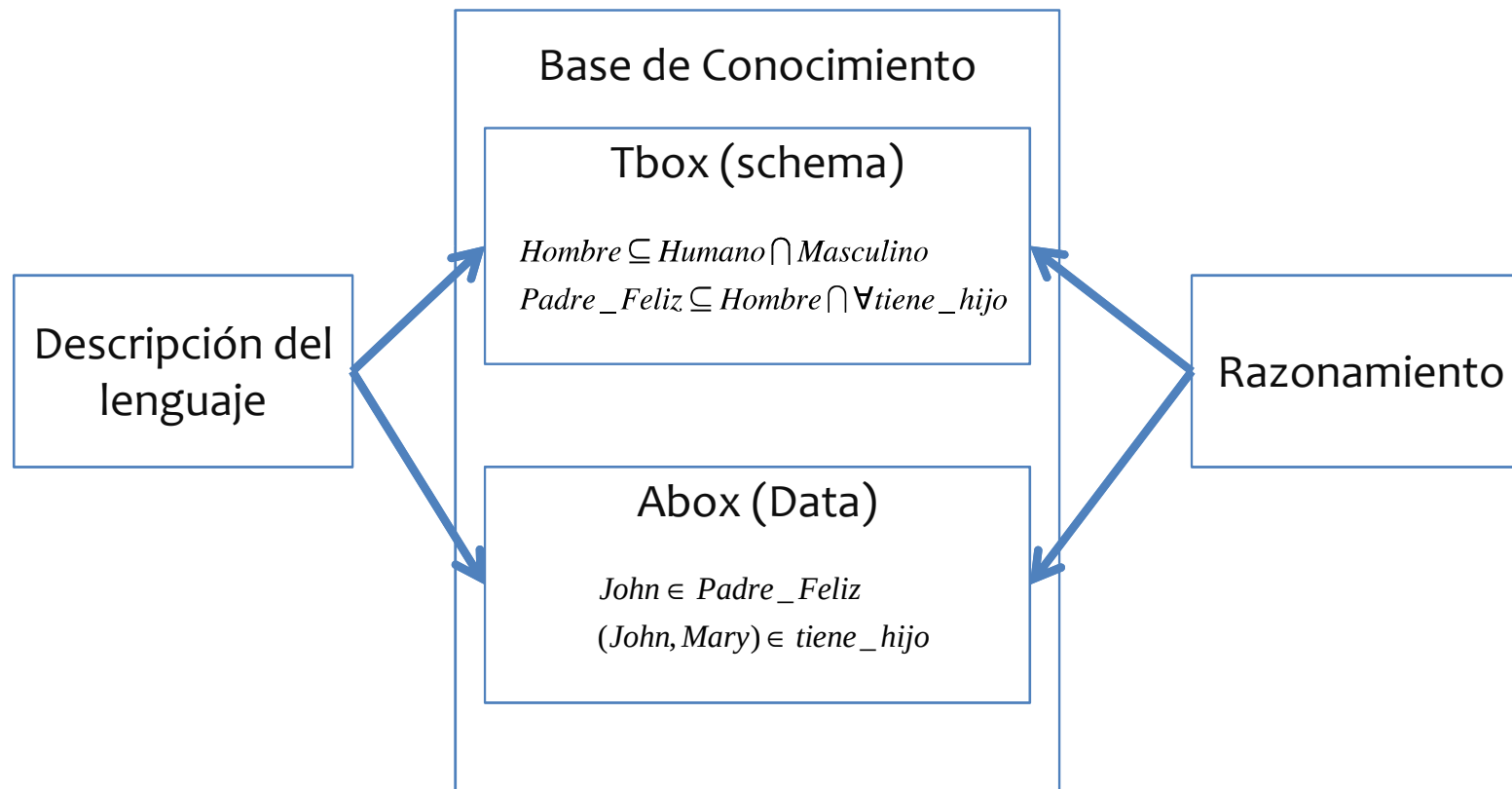
CORRESPONDENCIA ENTRE OWL Y DL

Constructor OWL	Representación DL	Ejemplo
owl:equivalentTo (C,D)	$C \equiv D (C \sqsubseteq D \text{ y } D \sqsubseteq C)$	<i>Persona</i> $\equiv$ <i>Humano</i>
rdfs:subClassOf (C,D)	$C \sqsubseteq D$	<i>Padres</i> $\sqsubseteq$ <i>Persona</i>
owl:complementOf (C,D)	$C \equiv \neg D (\text{negacion})$	<i>Varon</i> $\equiv \neg$ <i>Mujer</i>
owl:disjointWith (C,D)	$C \sqsubseteq \neg D$	<i>Padre</i> $\sqsubseteq \neg$ <i>Madre</i>
owl:intersectionOf (C,D)	$C \sqcap D (\text{conjuncion})$	<i>Padres</i> $\sqcap$ <i>Varon</i>
owl:unionOf (C,D)	$C \sqcup D (\text{disjuncion})$	<i>Padre</i> $\sqcup$ <i>Madre</i>
owl:oneOf (I1, I2)	$\{I_1\} \sqcup \{I_2\}$	$\{\text{Juan}\} \sqcup \{\text{Maria}\}$
owl:someValuesFrom(P,C)	$\exists P.C (\text{existencial})$	$\exists \text{tieneHijo.Hija}$
owl:allValuesFrom(P,C)	$\forall P.C (\text{universal})$	$\forall \text{tieneHijo.Hijo}$
owl:hasValue (P,I1)	$\exists P.\{I_1\}$	$\exists \text{tieneHijo.}\{\text{Juan}\}$
owl:cardinality(P,n)	$= n.P$	$= 2.\text{tienePadres}$
owl:minCardinality(P,n)	$\geq n.P$	$\geq 1.\text{tieneHija}$
owl:maxCardinality(P,n)	$\leq n.P$	$\leq 2.\text{tieneHijos}$

Un concepto en DL se refiere a una clase en OWL.

Un rol en DL es una propiedad en OWL.

ARQUITECTURA DL



LÓGICAS DESCRIPTIVAS: TBOX

- Tbox: contiene declaraciones terminológicas generales. Vocabulario de un dominio de aplicación en función de: Conceptos, Roles, etc. Son de dos tipos.

- Definición de concepto

$$A \equiv C$$

Ejemplo

$$\text{Mujer} \equiv \text{Persona} \cap \text{Femenino}$$

$$\text{Madre} \equiv \text{Mujer} \cap \exists \text{tiene_hijo. Persona}$$

- Axiomas descriptivos de roles, etc.

$$C_1 \subseteq C_2$$

Ejemplo

$$\exists \text{tiene_hijo. Persona} \subseteq \text{Persona}$$

LÓGICAS DESCRIPTIVAS: ABOX

- ⦿ Abox: contiene aserciones (instancias) sobre elementos y relaciones concretas del dominio. Es decir, son aserciones acerca de individuos usando vocabulario. Dos tipos:

- Instancias de conceptos

$$o \equiv C$$

Ejemplo

Moby – Dick $\equiv$ Ballena

Juan : Hombre $\cap \exists$ tiene – hijo

- Instancias de axiomas

$$(o_1, o_2) : R$$

Ejemplo

(Ana, Juan) : tiene_hijo

FORMALIZAR EN DL Y LUEGO EN OWL DL

- ◉ Definición de conceptos.
 - El pasto y los arboles son plantas. Las hojas son parte del árbol, pero existen otras partes de un árbol que no son hojas. Un perro debe comer al menos huesos. Una oveja es un animal solo debe comer pasto. Una jirafa es un animal que solo debe comer hojas. Las vacas locas solo se alimenta de cerebros que pertenecen a las ovejas.
- ◉ Restricciones:
 - Animales son disjuntos con plantas.
- ◉ Propiedades:
 - Comer es aplicado a los animales y su inverso es comido_por.
- ◉ Individuos
 - Tom
 - Flossie es una vaca
 - Rex es un perro y es una mascota de Mick
 - Fido es un perro
 - Tibbs es un gato

FORMALIZAR EN DL Y LUEGO EN OWL

DL

1. El pasto y los arboles son plantas.
2. Las hojas son parte del árbol, pero existen otras partes de un árbol que no son hojas.
3. Un perro debe comer al menos huesos.
4. Una oveja es un animal y solo debe comer pasto.
5. Una jirafa es un animal que solo debe comer hojas.
6. Una vaca loca es una vaca que se alimenta de cerebros que son parte de las ovejas.
7. Animales o parte de animales son disjuntos con plantas o parte de plantas.

⦿ Propiedades:

- Comer es aplicado a los animales y su inverso es comido_por.

1. $Pasto \subseteq Planta$
1. $Arboles \subseteq Planta$
2. $Hojas \subseteq Arboles$
3. $Perro \subseteq Animal \cap \exists come.Huesos$
4. $Oveja \subseteq Animal \cap \forall come.Pasto$
5. $Jirafa \subseteq Animal \cap \forall come.Hojas$
6. $VacaLocas \subseteq vacas \cap \forall come(cerebro \subseteq Oveja)$
7. $Animal \subseteq \neg Planta$

FORMALIZAR EN DL Y LUEGO EN OWL

1.1.(LD). $Pasto \subseteq Planta$.

(OWL) $rdf : subClassOf (Pasto, Planta)$

1.2.(LD). $Arboles \subseteq Planta$.

(OWL) $rdf : subClassOf (Arboles, Planta)$

2.(DL) $Hojas \subseteq Arboles$

(OWL) $owl : subClassOf (Hojas, Arboles)$

3.(DL) $Perro \subseteq Animal \cap \exists come.Huesos$

(OWL) $owl : subClassOf (Perro, (intersection(Animal, someValuesFrom(come, Huesos))))$

4.(DL) $Oveja \subseteq Animal \cap \forall come.Pasto$

(OWL) $owl : subClassOf (Animal, intersectionOf (Animal, owl : someValuesFrom(come, Pasto)))$

5.(DL) $Jirafa \subseteq Animal \cap \forall come.Hojas$

(OWL) $owl : subClassOf (Animal, intersection(Animal, owl : allValuesFrom(come, Hojas)))$

6.(DL) $VacaLocas \subseteq Vacas \cap \forall come(cerebro \subseteq Oveja)$

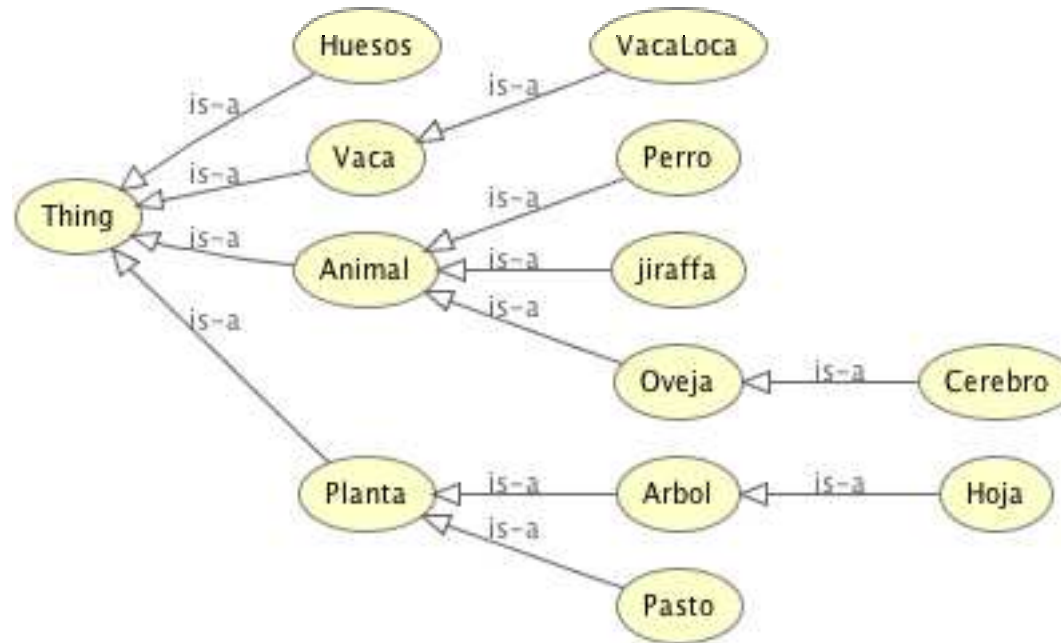
(OWL) $owl : subClassOf (Vacas, intersection(Vacas, owl : allValuesFrom(come, Cerebro)))$

$owl : subClassOf (Cerebro, Oveja)$

7.(DL) $Animal \subseteq \neg Planta$

(OWL) $owl : disjointWith(C, D)$

RESULTADOS



1. $Pasto \subseteq Planta$

1. $Arboles \subseteq Planta$

2. $Hojas \subseteq Arboles$

3. $Perro \subseteq Animal \cap \exists come.Huesos$

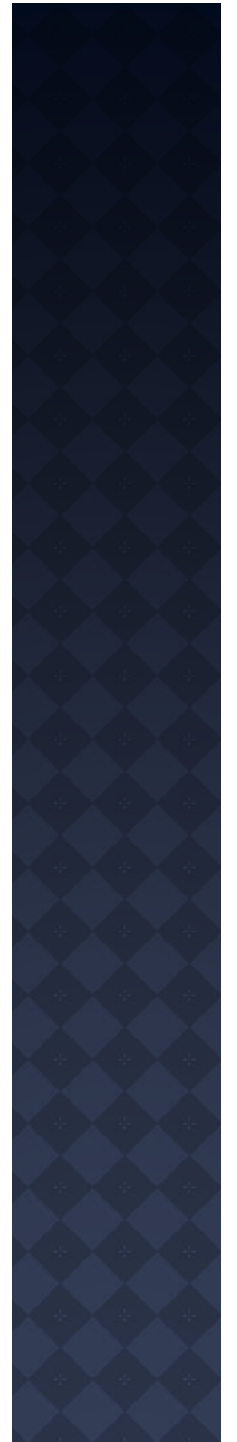
4. $Oveja \subseteq Animal \cap \forall come.Pasto$

5. $Jirafa \subseteq Animal \cap \forall come.Hojas$

6. $VacaLocas \subseteq vacas \cap \forall come(cerebro \subseteq Oveja)$

7. $Animal \subseteq \neg Planta$

CONSTRUCCION DE UNA ONTOLOGÍA OWL EN PROTEGE



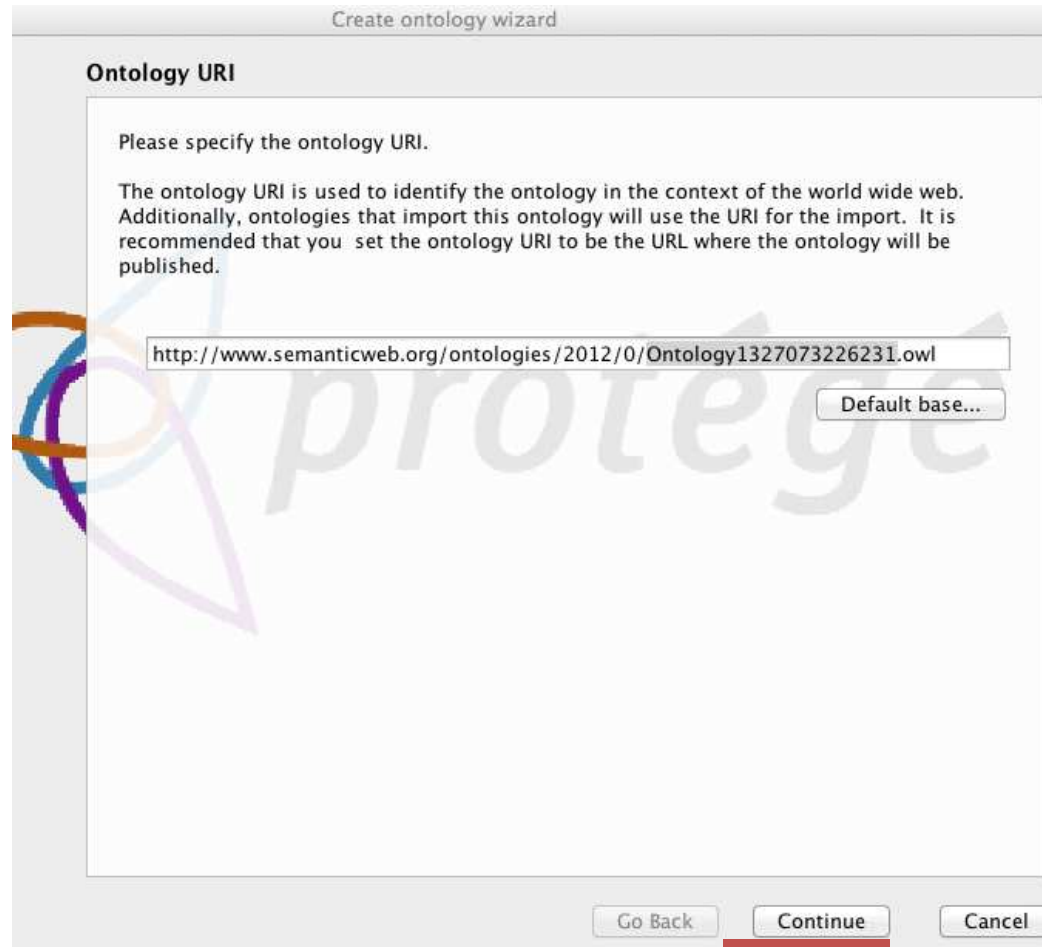
CREAR UNA NUEVA ONTOLOGÍA OWL

- Inicie la aplicación protégé.
- En la pantalla de bienvenida, seleccione “Create New OWL Ontology”



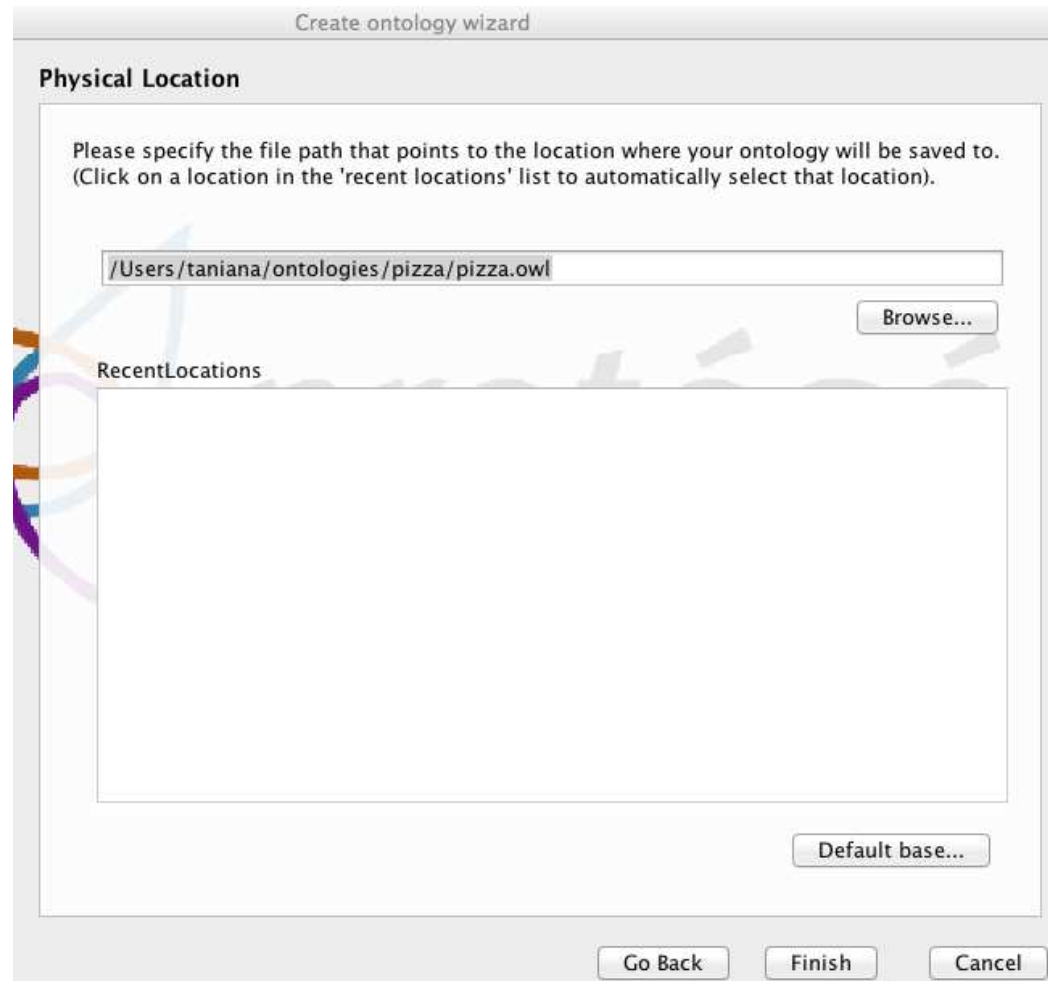
CREAR UNA NUEVA ONTOLOGÍA OWL

1. Toda ontología usa Unique Resource Identifier (URI)
2. Coloque el nombre de la ontología y presione Continue para seguir.
3. En nuestro caso colocaremos pizza.owl



CREAR UNA NUEVA ONTOLOGÍA OWL

1. Usted puede guardar la ontología en su pc
2. Presione Finish para seguir.



The screenshot shows a window titled "Create ontology wizard" with a tab labeled "Physical Location". The main text inside the window reads: "Please specify the file path that points to the location where your ontology will be saved to. (Click on a location in the 'recent locations' list to automatically select that location).". Below this text is a text input field containing the path "/Users/taniana/ontologies/pizza/pizza.owl". To the right of the input field is a "Browse..." button. Below the input field is a section titled "RecentLocations" which contains an empty list box. At the bottom right of the main content area is a "Default base..." button. At the very bottom of the window are three buttons: "Go Back", "Finish", and "Cancel".

INTERFAZ DEL PROTÉGÉ

pizza.owl (http://www.semanticweb.org/ontologies/2012/0/pizza.owl) - [/Users/taniana/ontologies/pizza/pizza.owl]

← → pizza.owl 🔍

Active Ontology | Entities | Classes | Object Properties | Data Properties | Individuals | OWLViz | DL Query

Ontology Annotations | Inferred Axioms

Ontology annotations: ⌵ ⌶ ⌵ ⌶

Annotations +

Ontology Imports | General axioms | RDF/XML Rendering ▶

Imported ontologies: ⌵ ⌶ ⌵ ⌶

Direct imports +

Indirect imports

Ontology metrics: ⌵ ⌶ ⌵ ⌶

Metrics

Class count	0
Object property count	0
Data property count	0
Individual count	0
DL expressivity	AL

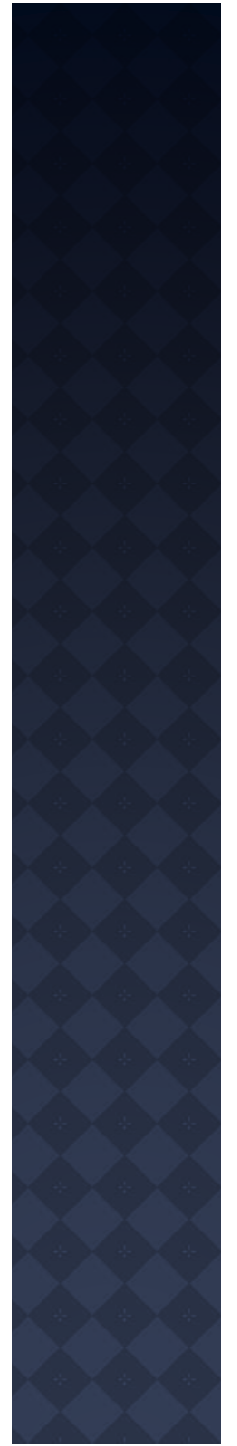
Class axioms

SubClass axioms count	0
Equivalent classes axiom...	0
Disjoint classes axioms c...	0
GCI count	0
Hidden GCI Count	0

Object property axioms

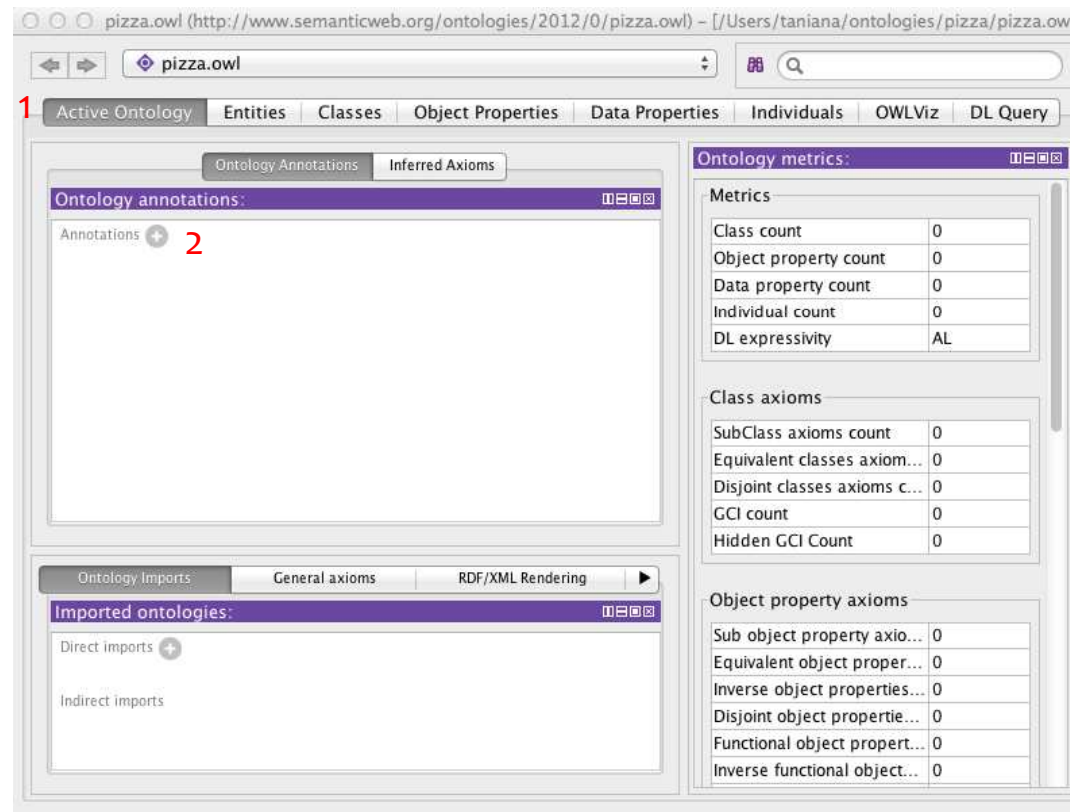
Sub object property axio...	0
Equivalent object proper...	0
Inverse object properties...	0
Disjoint object propertie...	0
Functional object propert...	0
Inverse functional object...	0

**AÑADIR UN COMENTARIO
EN LA ONTOLOGÍA**



AÑADIR UN COMENTARIO EN LA ONTOLOGÍA

1. Este seguro que este en Active Ontology.
2. En Ontology annotations: clic el icono (+)



AÑADIR UN COMENTARIO EN LA ONTOLOGÍA

1. Entre un comentario que describa la ontología que esta desarrollando.
2. En nuestro ejemplo puede colocar: Ontología de pizza que describe varias pizzas según los ingredientes.
3. Presione Ok

The screenshot shows a web browser window titled 'pizza.owl (http://www.semanticweb.org/ontologies/2012/0/pizza.owl)'. The interface is divided into two main sections. On the left, there is a list of 'Built in annotation URIs' with 'comment' selected. Below this list are 'Dublin Core annotation URIs'. On the right, there is a 'Value' field containing the text 'Ontología de pizza que describe varias pizzas según los ingredientes.' Below the value field are 'Type' and 'Lang' dropdown menus. At the bottom right, there are 'Cancel' and 'OK' buttons.

Annotation URI
backwardCompatibleWith
comment
deprecated
incompatibleWith
isDefinedBy
label
priorVersion
seeAlso
versionInfo

Dublin Core annotation URIs

contributor
coverage
creator
date
description
format
identifier
language
publisher
relation
rights
source

Value: Ontología de pizza que describe varias pizzas según los ingredientes.

Type: [] Lang: []

Cancel OK

AÑADIR UN COMENTARIO EN LA ONTOLOGÍA

The screenshot displays the Protege OWL editor interface for the 'pizza.owl' ontology. The browser address bar shows the URL: `http://www.semanticweb.org/ontologies/2012/0/pizza.owl`. The main menu includes 'Active Ontology', 'Entities', 'Classes', 'Object Properties', 'Data Properties', 'Individuals', 'OWLViz', and 'DL Query'. The 'Ontology Annotations' tab is active, showing a list of annotations. A new annotation named 'comment' has been added, with the text: "Ontología de pizza que describe varias pizzas según los ingredientes." The 'Ontology metrics' panel on the right shows various counts: Class count (0), Object property count (0), Data property count (0), Individual count (0), DL expressivity (AL), SubClass axioms count (0), Equivalent classes axioms count (0), Disjoint classes axioms count (0), GCI count (0), Hidden GCI Count (0), Sub object property axioms count (0), Equivalent object property axioms count (0), Inverse object properties axioms count (0), Disjoint object properties axioms count (0), Functional object property axioms count (0), Inverse functional object property axioms count (0), Transitive object property axioms count (0), and Symmetric object property axioms count (0).

Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query

Ontology Annotations Inferred Axioms

Ontology annotations:

Annotations +

comment

"Ontología de pizza que describe varias pizzas según los ingredientes."

Ontology metrics:

Metrics

Class count	0
Object property count	0
Data property count	0
Individual count	0
DL expressivity	AL

Class axioms

SubClass axioms count	0
Equivalent classes axioms count	0
Disjoint classes axioms count	0
GCI count	0
Hidden GCI Count	0

Object property axioms

Sub object property axioms count	0
Equivalent object property axioms count	0
Inverse object properties axioms count	0
Disjoint object properties axioms count	0
Functional object property axioms count	0
Inverse functional object property axioms count	0
Transitive object property axioms count	0
Symmetric object property axioms count	0

Ontology Imports General axioms RDF/XML Rendering

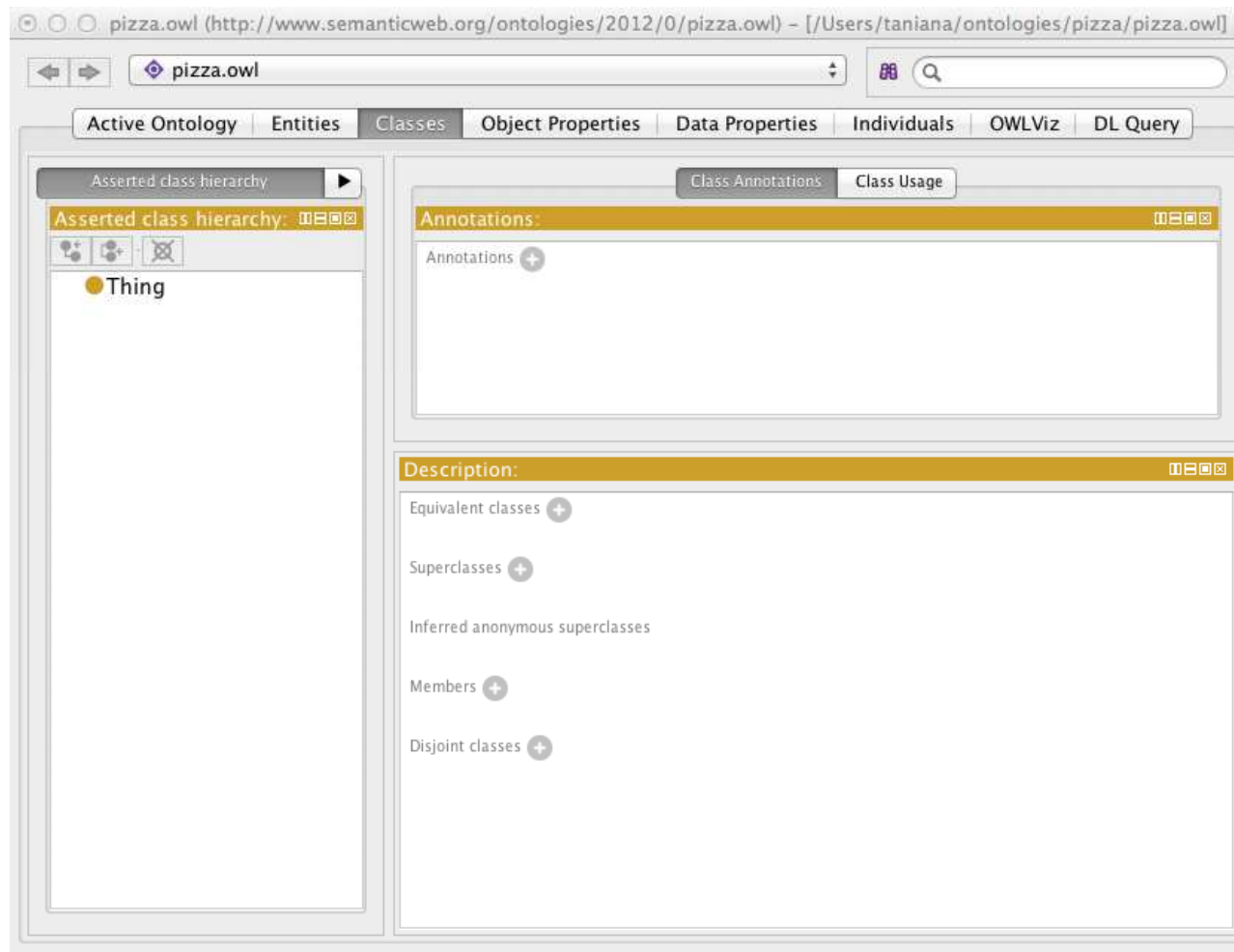
Imported ontologies:

Direct Imports +

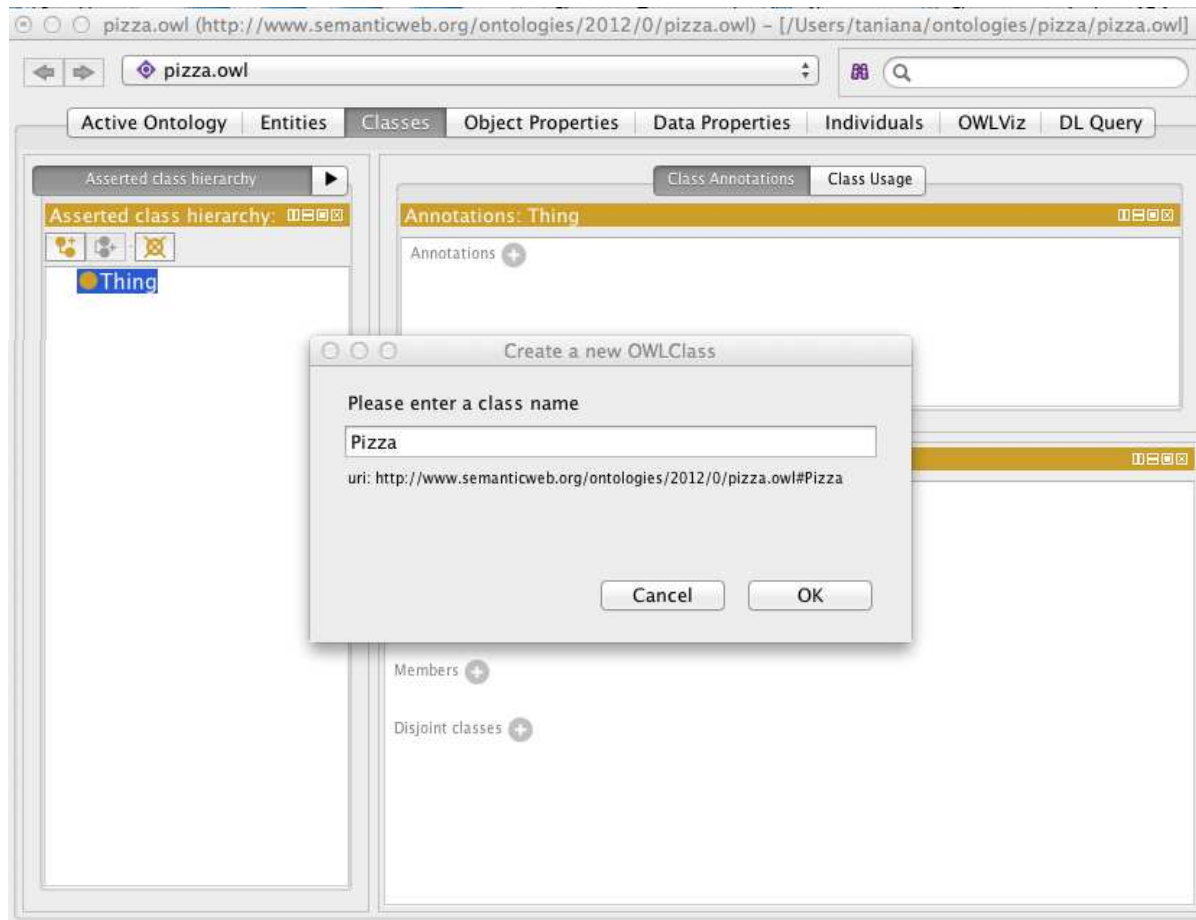
Indirect imports

CREAR LAS CLASES

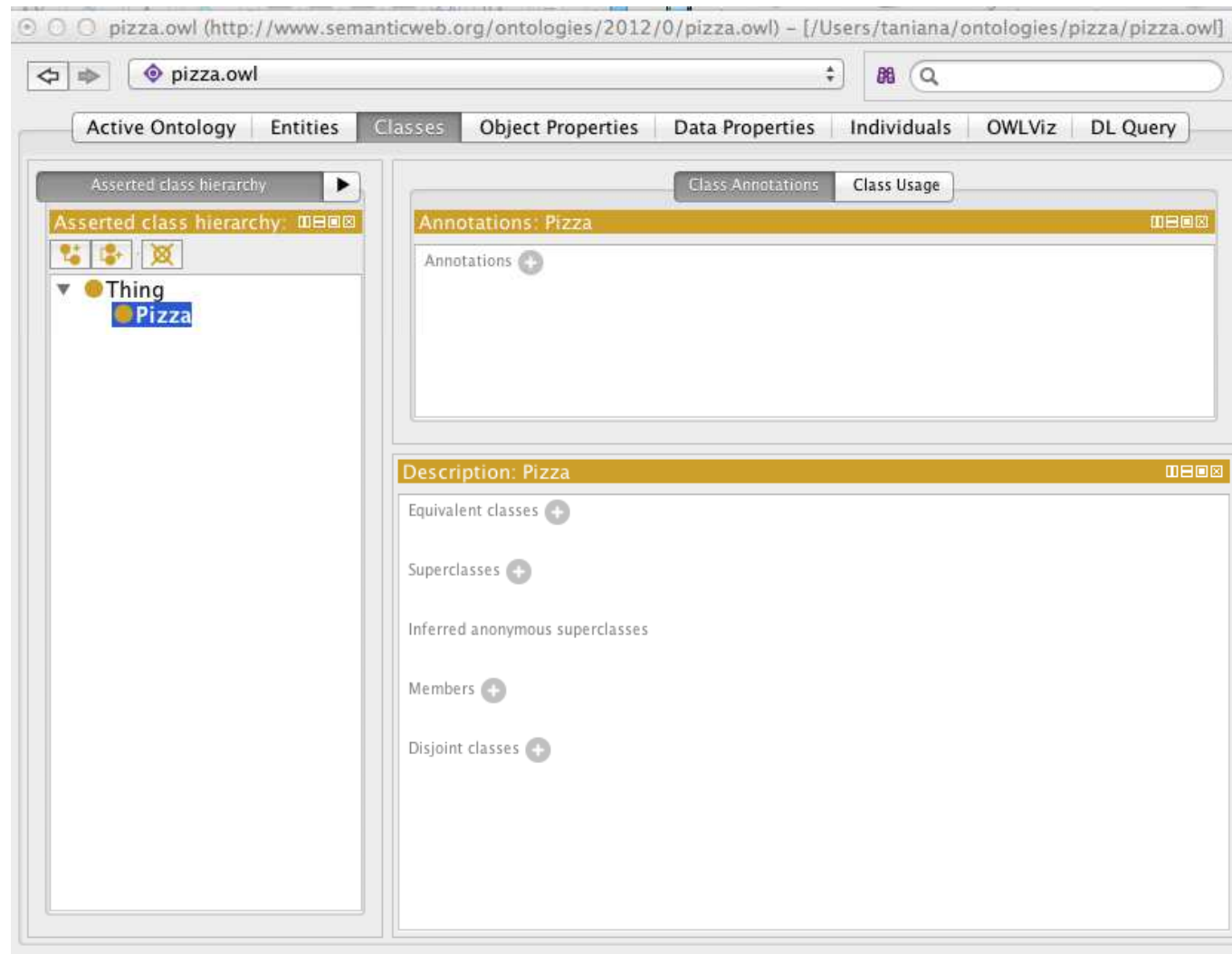
CLASES EN PROTÉGÉ



CLASES EN PROTÉGÉ

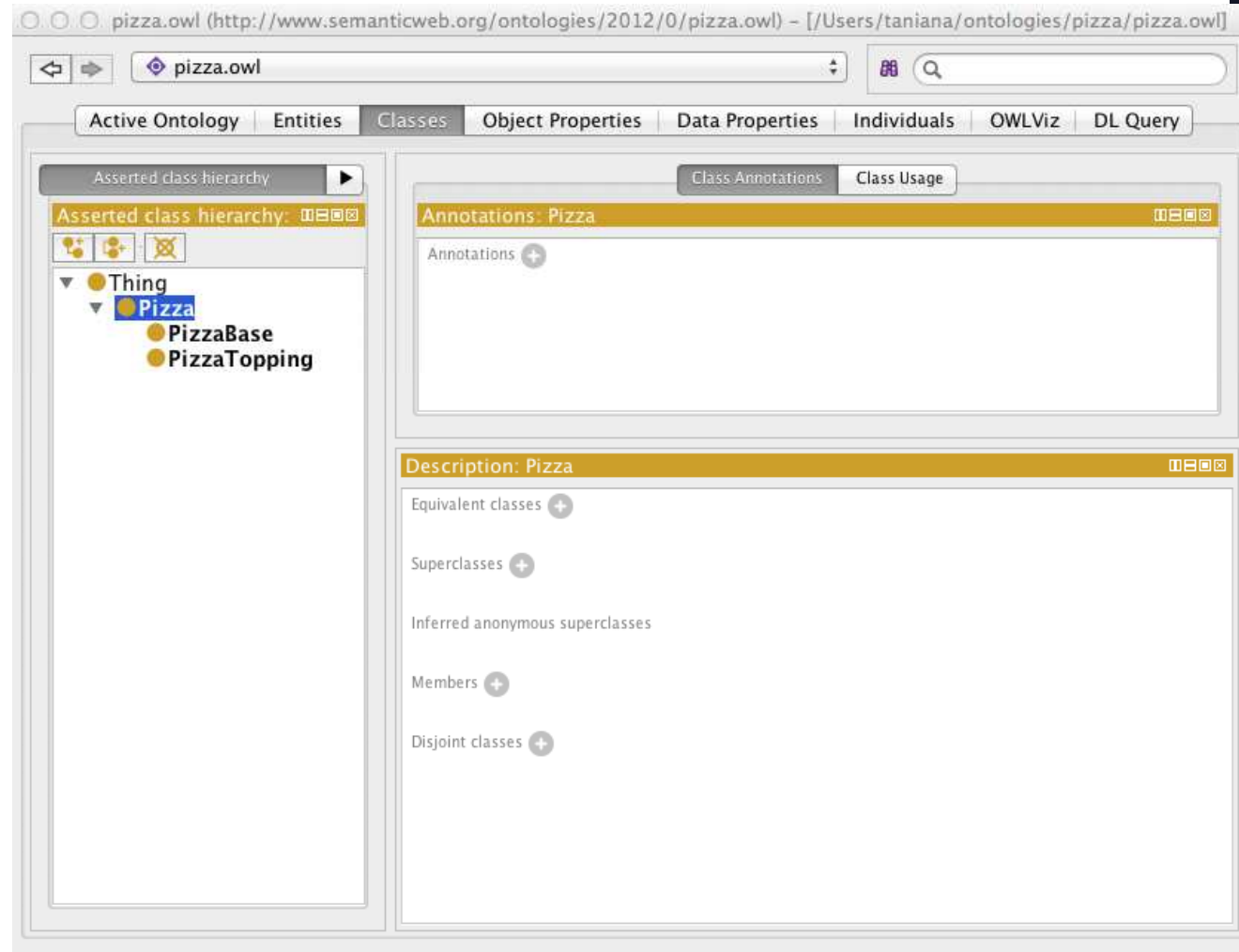


CLASES EN PROTÉGÉ

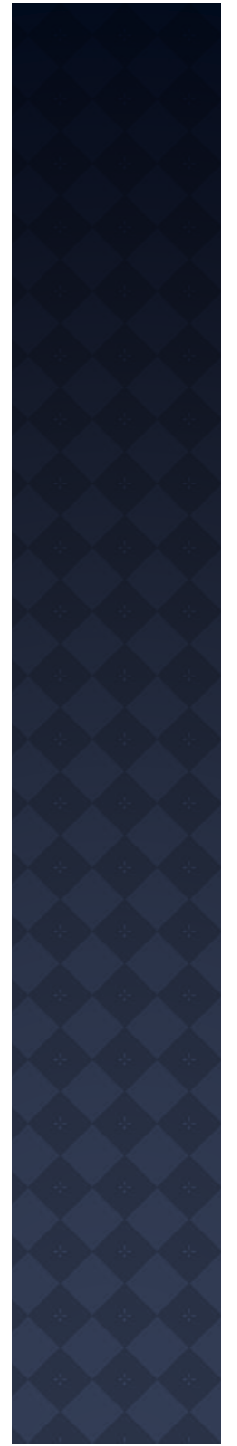


CLASES EN PROTÉGÉ

Repita los pasos anteriores para crear:
PizzaTopping
PizzaBase

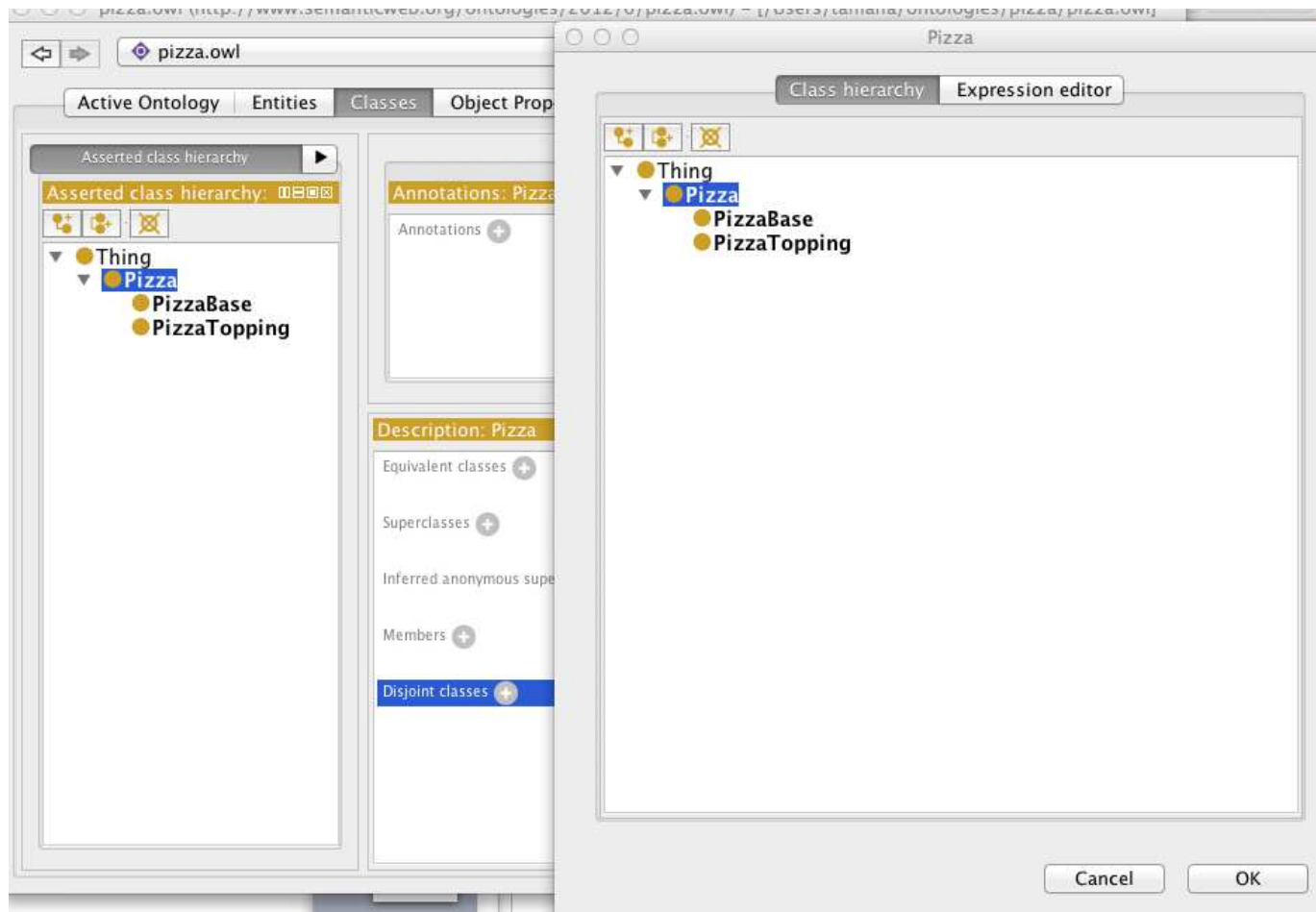


CLASES DISJUNTAS

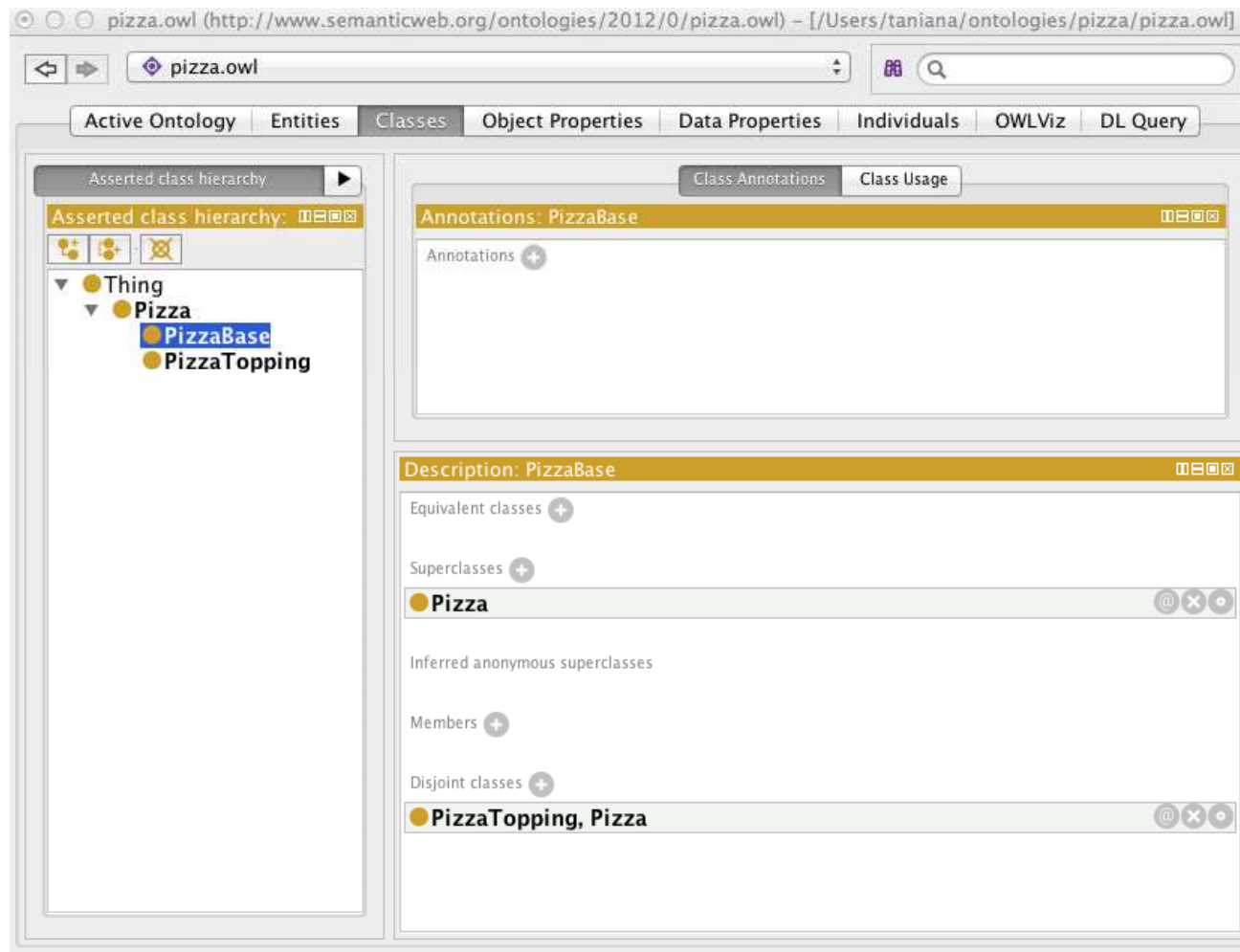


CLASES DISJUNTAS

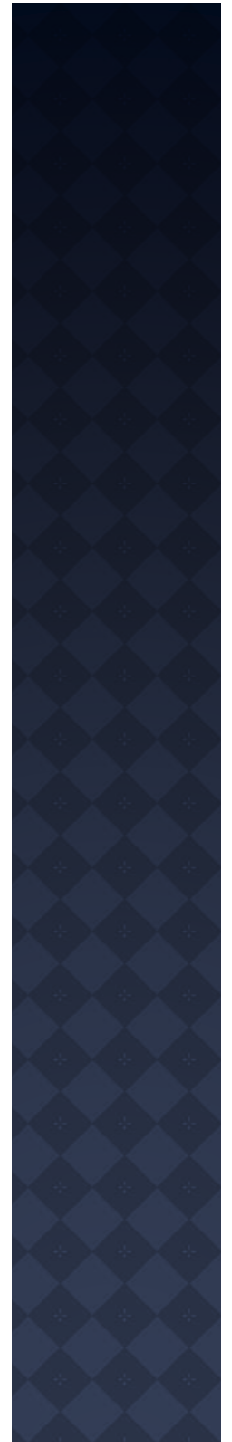
Después de adicionar varias clases en la jerarquía de la ontología, se requiere establecer clases disjuntas, que indican que un objeto o individuo no puede ser instancia de más de una de estas clases que se establecieron disjuntas



CLASES DISJUNTAS

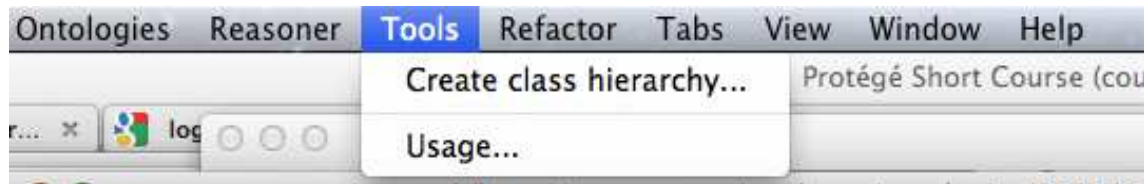


USANDO LA HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES



USANDO HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES

- Añadir algunas subclases de PizzaBase con la herramienta.



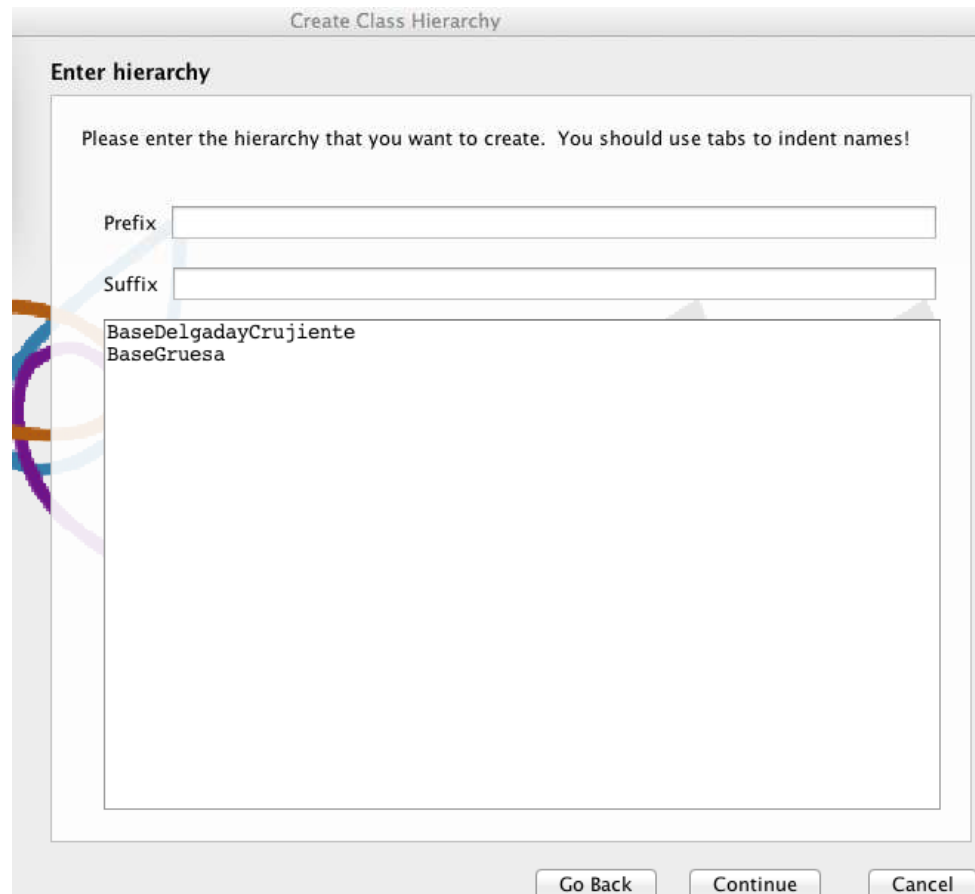
1. Seleccione la clase PizzaBase
2. Presione continue



USANDO HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES

- Añadir algunas subclases de PizzaBase con la herramienta.

1. Introducir:
 1. BaseDelgadayCrujiente
 2. Base Gruesa
2. Seleccionar Continue



Create Class Hierarchy

Enter hierarchy

Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix

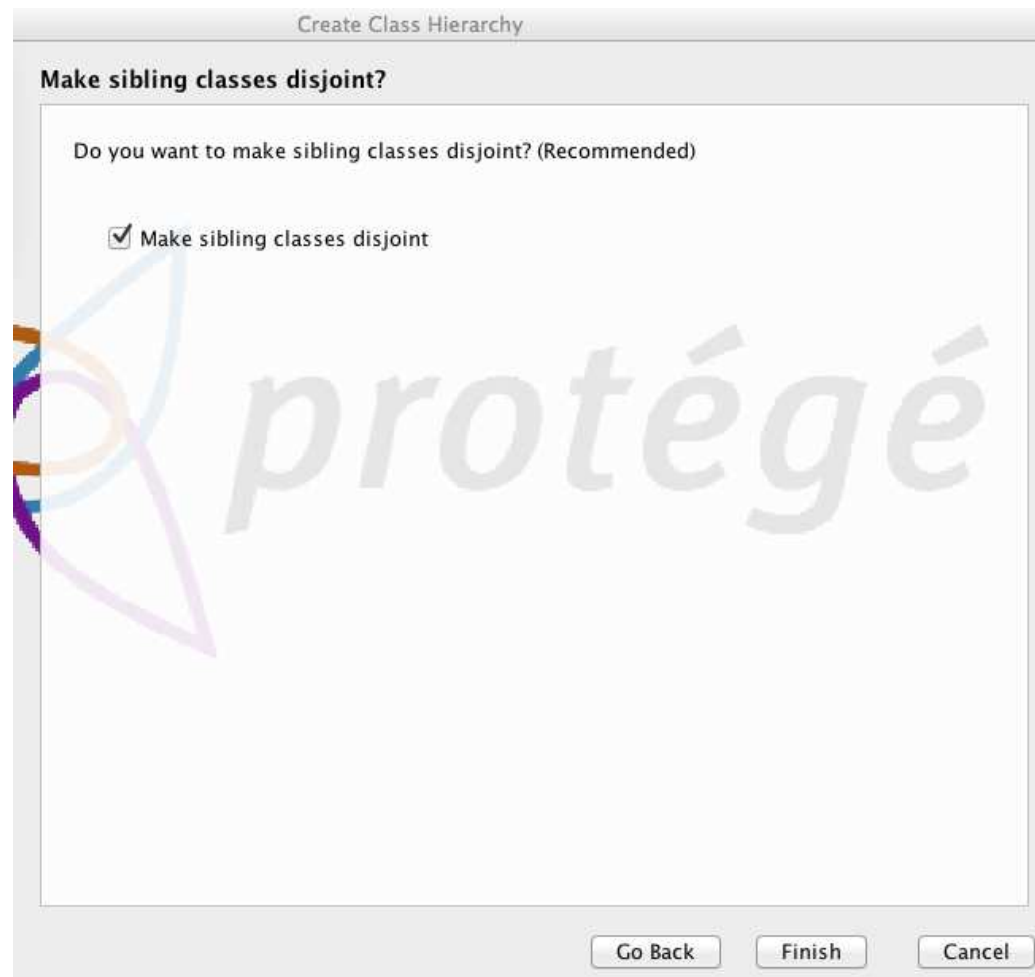
Suffix

BaseDelgadayCrujiente
BaseGruesa

Go Back Continue Cancel

USANDO HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES

- Decir que las clases creadas son disjuntas



USANDO HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES

- Añadir algunas subclases de PizzaTopping con la herramienta.

1. Realizar los pasos anteriores
2. Suffix Topping
3. Introducir:
 1. Queso
 - Mozzarella
 - Parmesano
 2. Carne
 - Jamón
 - Pepperoni
 - Salchichón
 3. Pescado
 - Anchoa
4. Seleccionar Continue

Create Class Hierarchy

Enter hierarchy

Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix

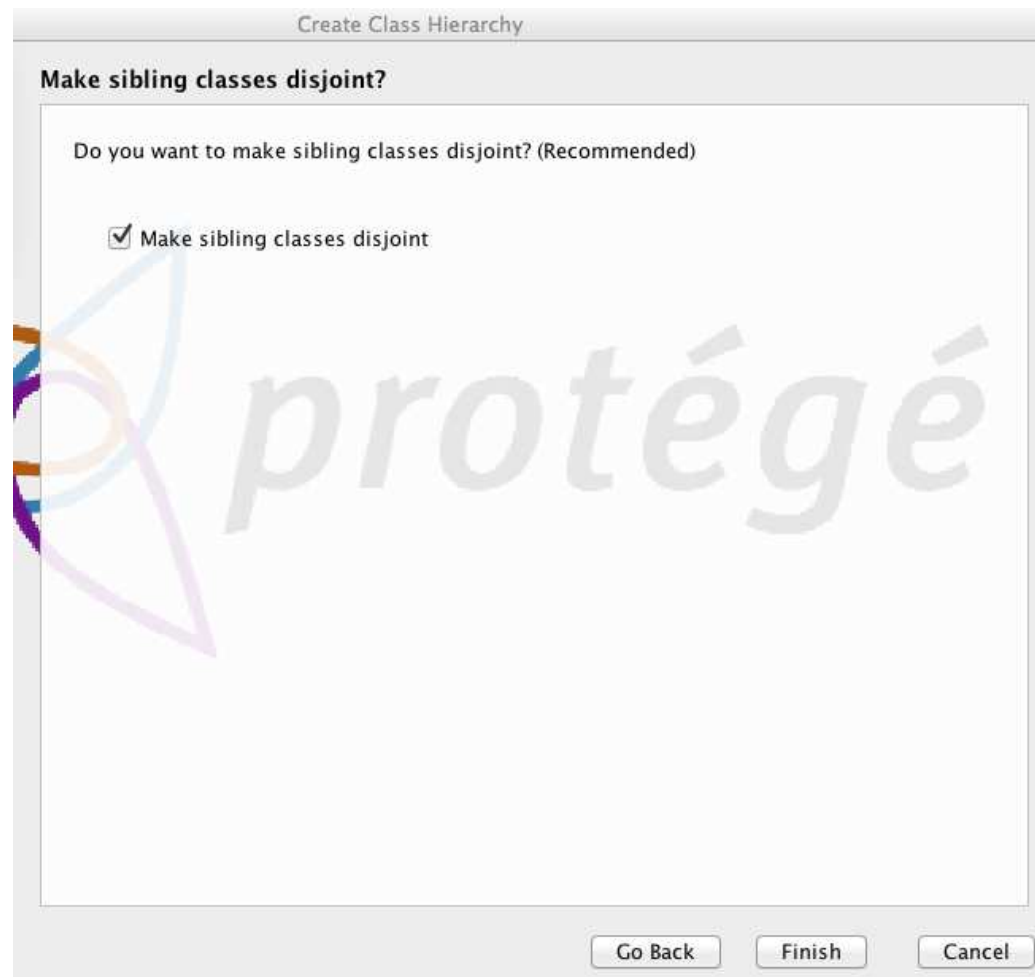
Suffix

```
Queso
  Mozzarella
  Parmesano
Carne
  Jamon
  Pepperoni
  Salchichon
Pescado
  Anchoa
  Camaron
  Atun
Vegetables
  Alcaparra
  Champinones
  Cebolla
  Aceituna
  Pimenton
    PimentonVerde
    PimentonRojo
Tomato
```

Go Back Continue Cancel

USANDO HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES

- Decir que las clases creadas son disjuntas



USANDO HERRAMIENTA CREATE CLASS HIERARCHY PARA CREAR LAS CLASES

⦿ Resultado



PROPIEDADES

PROPIEDADES OWL

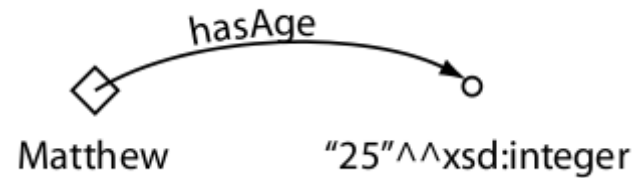
- ⦿ Las propiedades OWL representan relaciones entre dos objetos (**caso clásico de las relaciones en ontologías**) o individuos o propiedades de una clase (**esto es una diferencia con el uso habitual del termino “relación” en ontologías**).
- ⦿ Existen dos tipos de propiedades en OWL:
 - “ObjectProperties”, que permite relacionar un individuo con otro, y
 - “DatatypeProperties”, que relaciona un individuo con un XML Schema Datatype value o un literal RDF

PROPIEDADES OWL

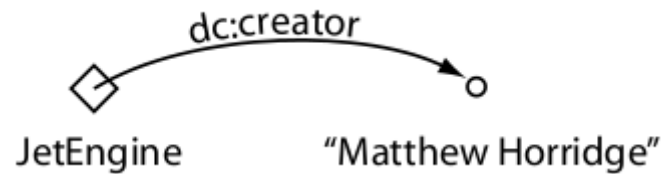
Object Properties



DataType Properties



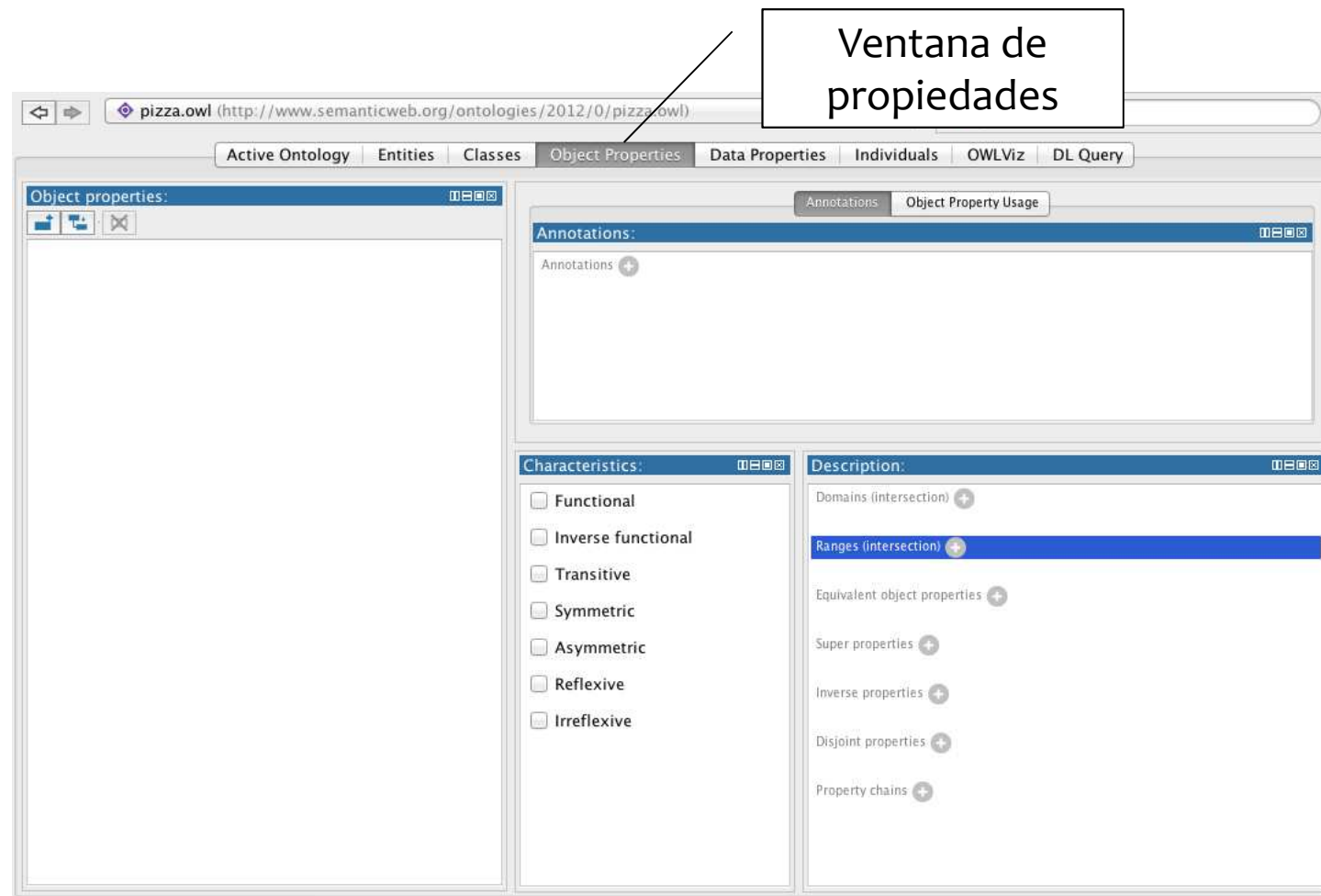
Annotation Properties*



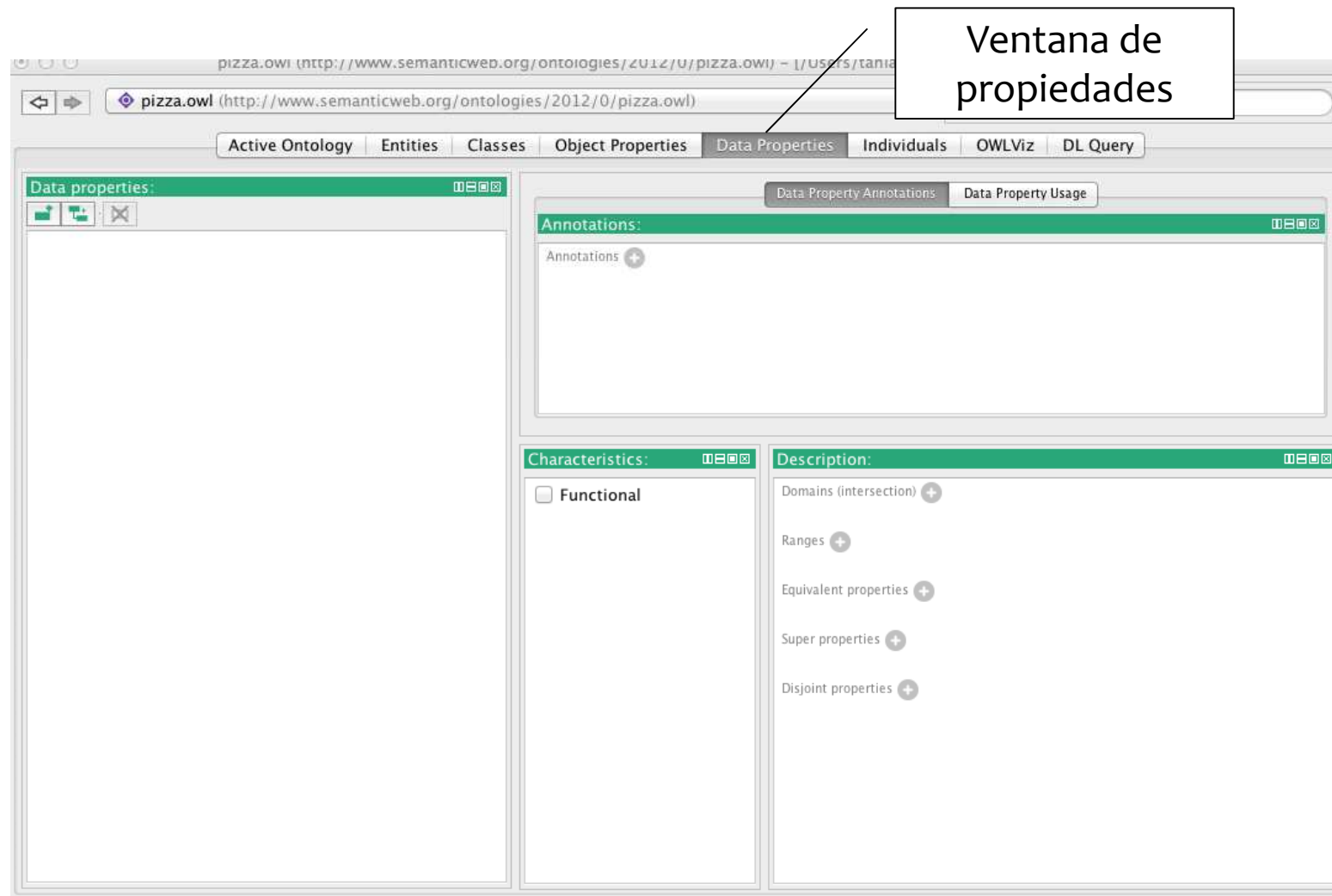
PROPIEDADES OWL

- ⦿ Los elementos que debe tener un ObjectProperty son:
 - nombre,
 - Dominio: hace referencia a la clase o clases iniciales y
 - Rango: hace referencia a la clase o clases finales.
- ⦿ Por ejemplo la relación **es tutor**
 - Nombre: **es_tutor**
 - Dominio: **Docente**
 - Rango: **Estudiante**

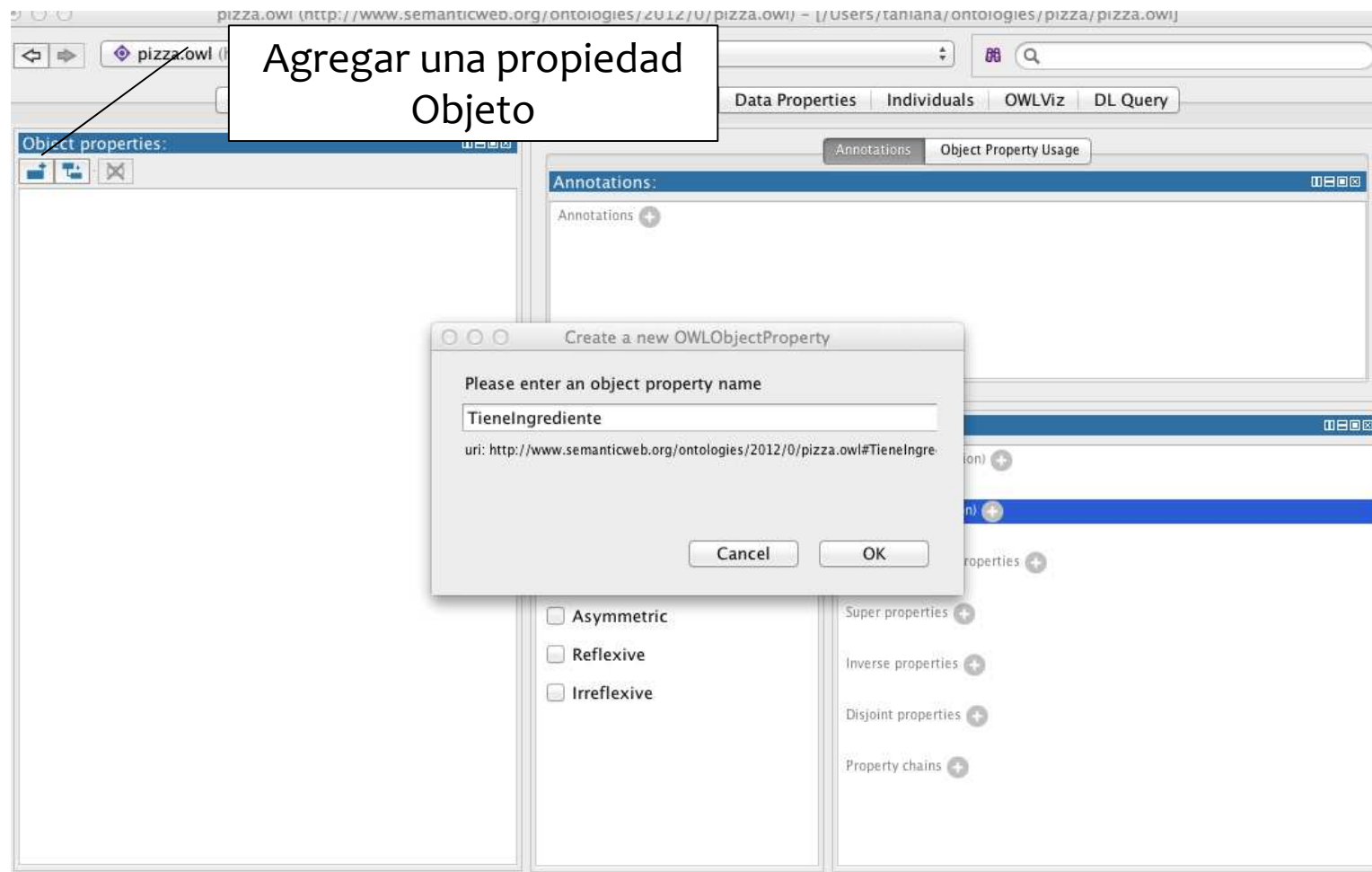
TAB DE PROPIEDADES OBJETOS



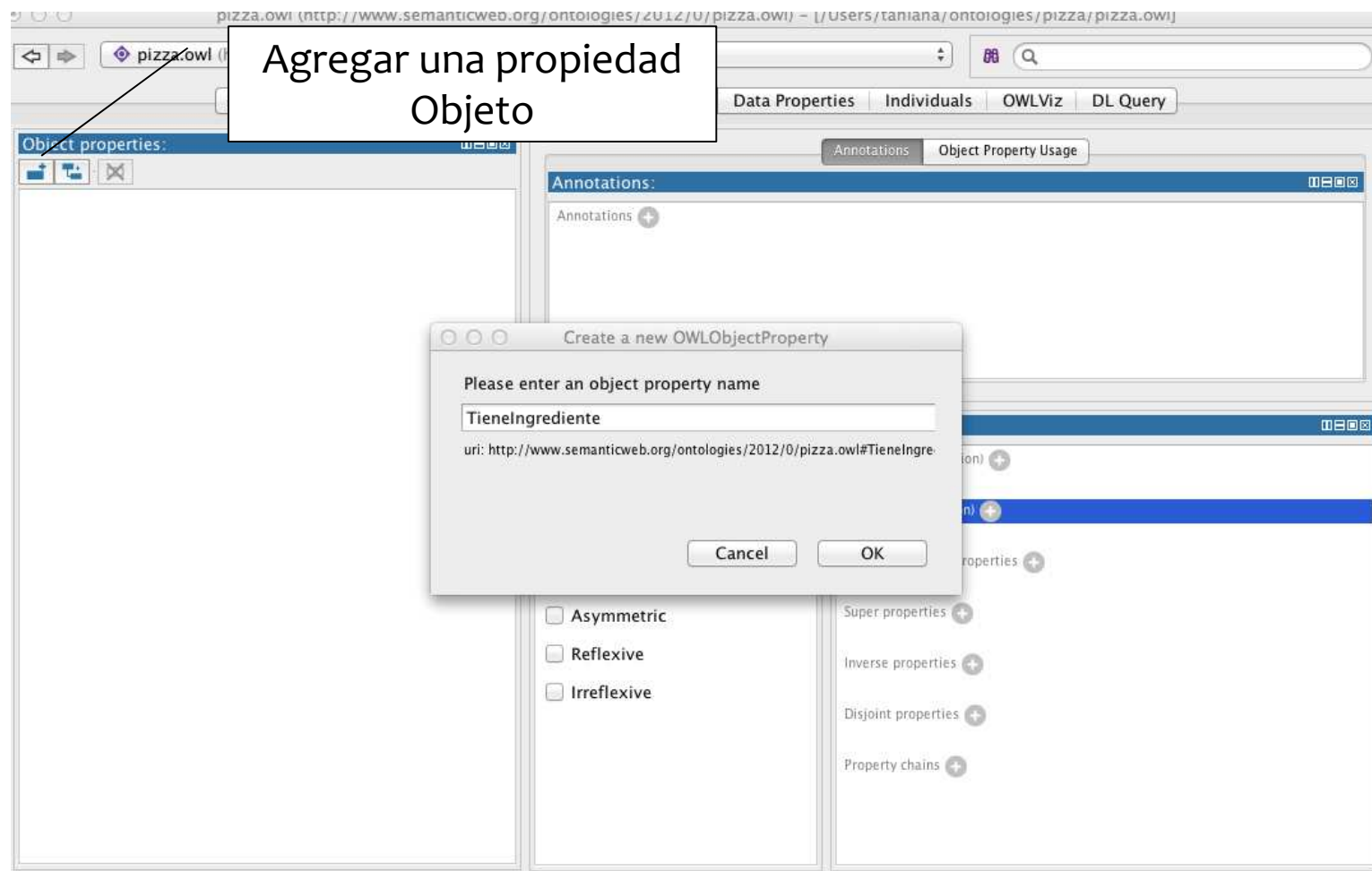
TAB DE PROPIEDADES DATOS



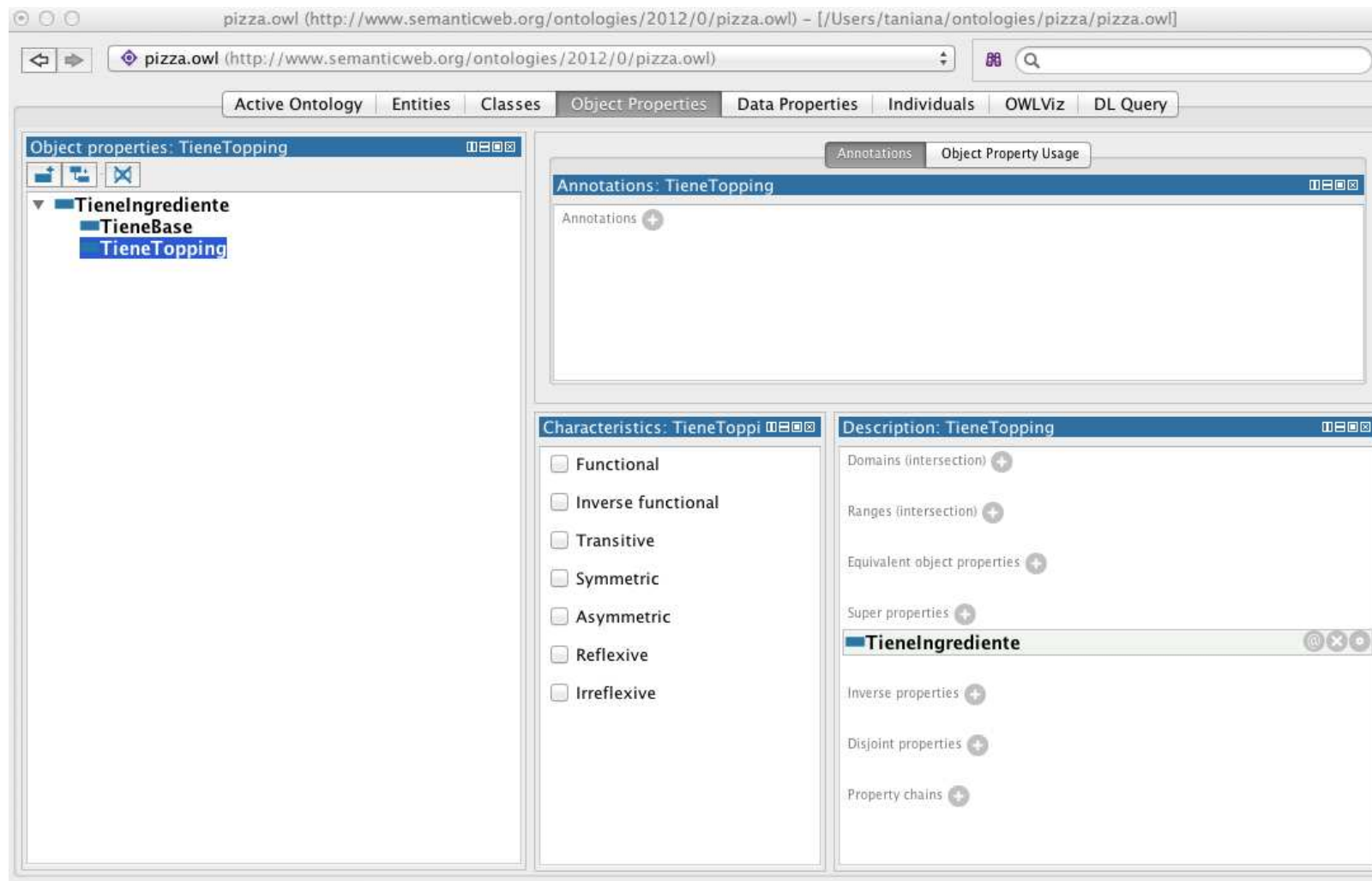
CREAR LA PROPIEDAD OBJETO LLAMADA TIENEINGREDIENTE



CREAR LA OTRAS PROPIEDADES OBJETO



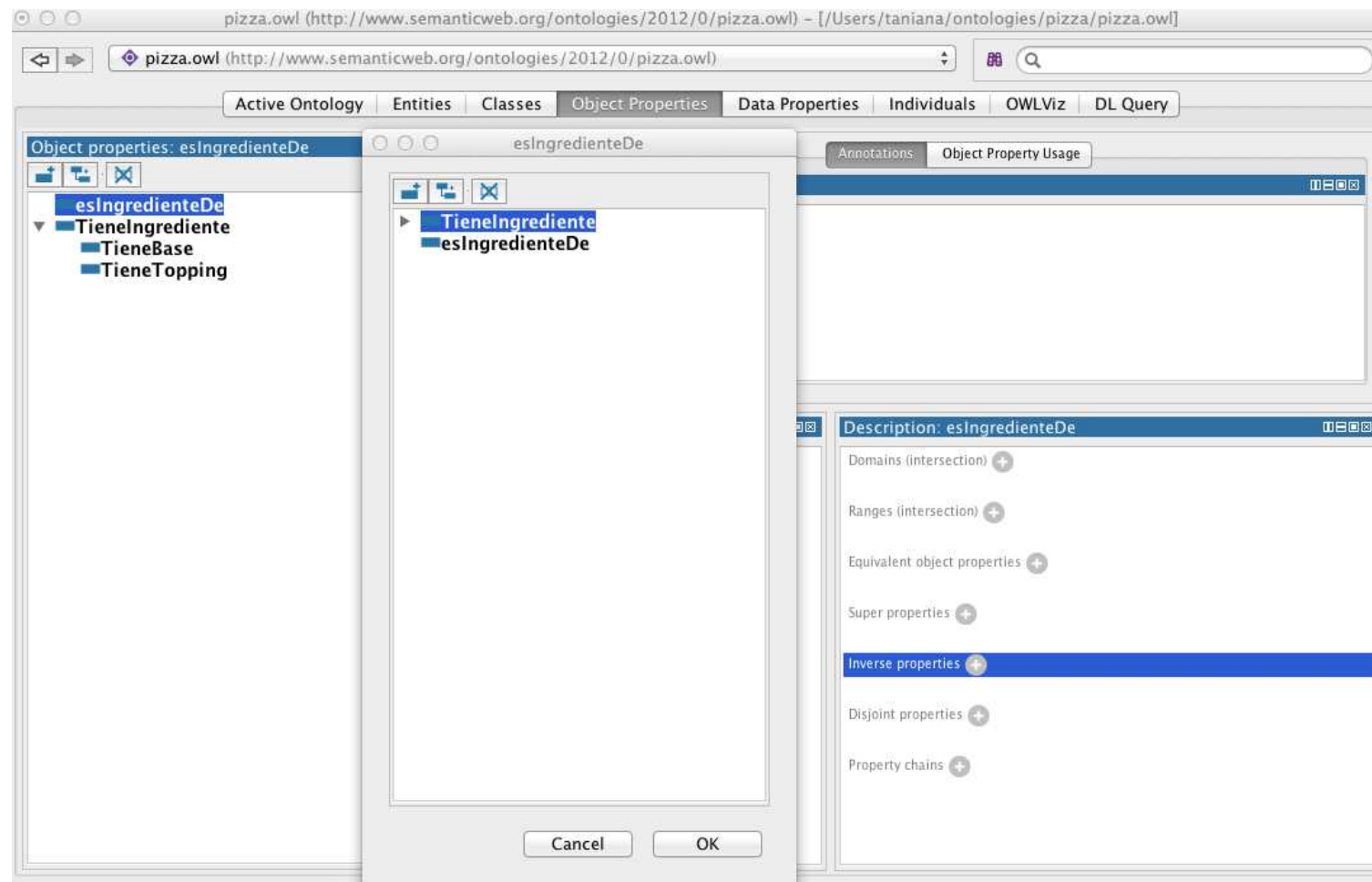
CREAR LA OTRAS PROPIEDADES OBJETO



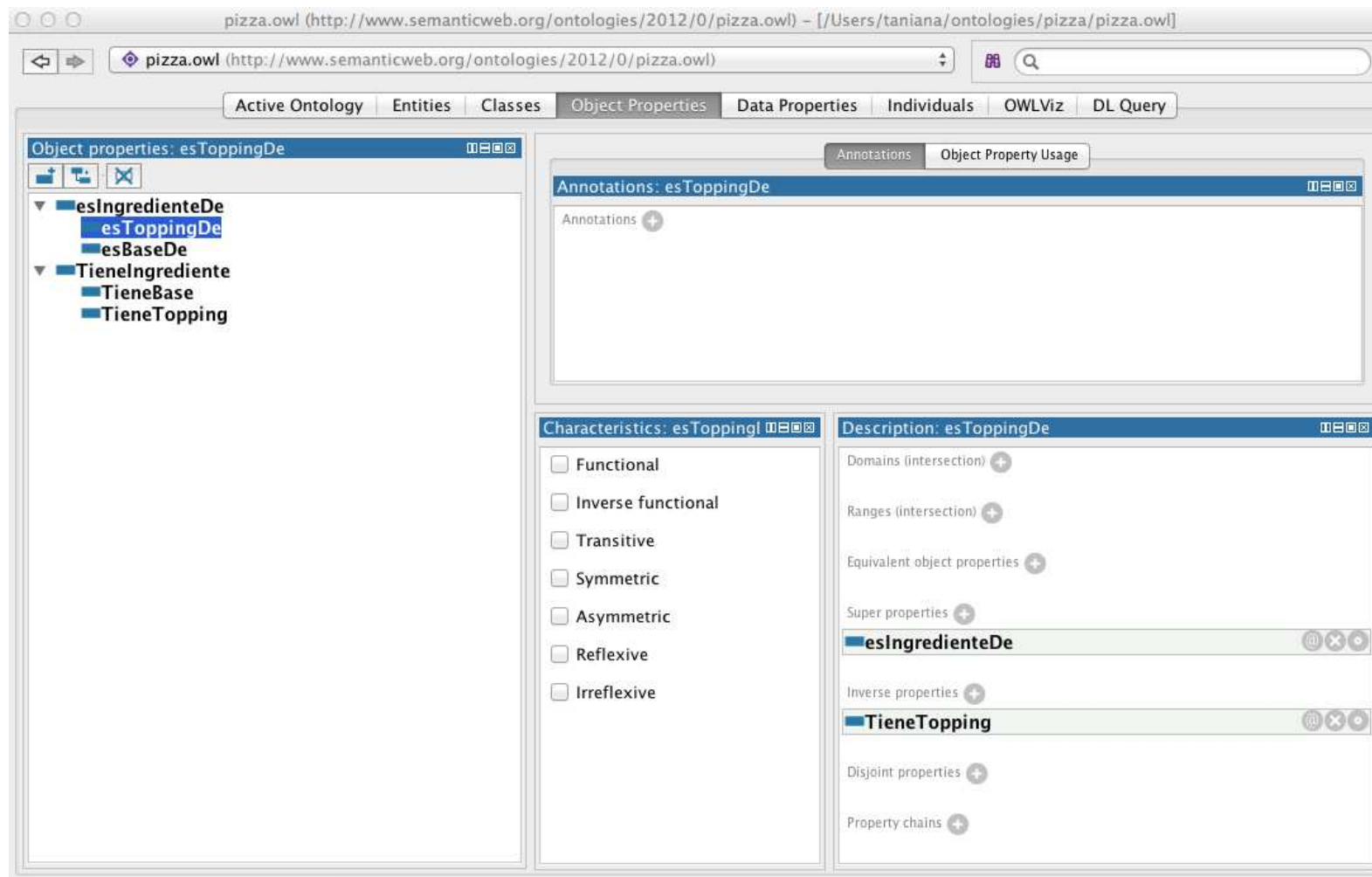
PROPIEDADES INVERSA

- ⦿ Cada `ObjectProperty` debe tener su correspondiente propiedad inversa. Si una propiedad enlaza un objeto A con otro B, entonces la propiedad inversa enlaza el objeto B con el A.

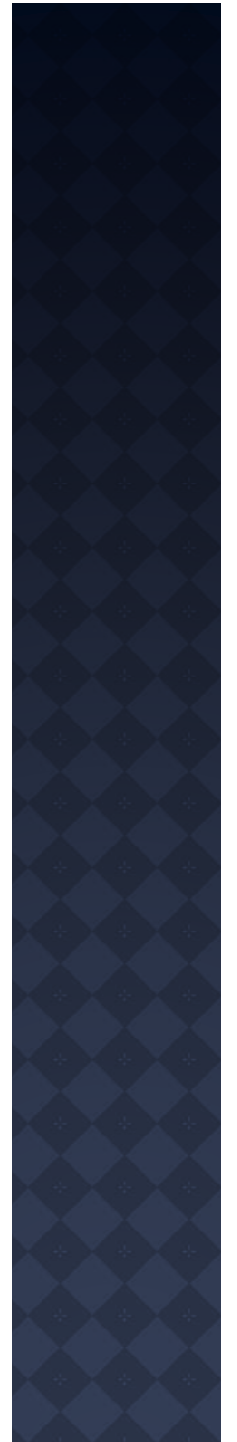
CREAR PROPIEDADES INVERSAS



CREAR OTRAS PROPIEDADES INVERSAS



CARACTERÍSTICAS DE LAS PROPIEDADES



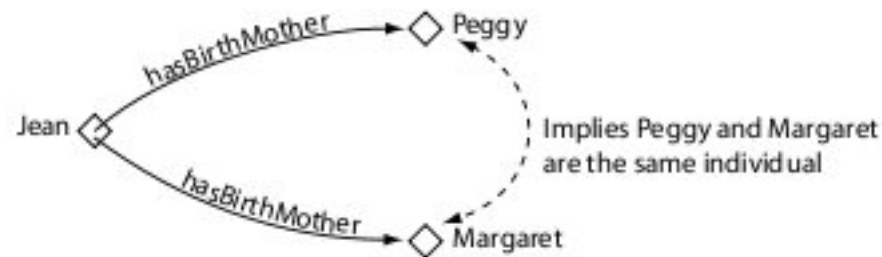
CARACTERÍSTICAS DE LAS PROPIEDADES

- ◉ OWL permite que el significado de las propiedades sea enriquecido con las características de las propiedades, que son:
 - Propiedades funcionales
 - Propiedades funcionales inversas
 - Propiedades transitivas
 - Propiedades simétricas

PROPIEDADES FUNCIONALES

- ⦿ Propiedades funcionales: Si una propiedad es funcional dado un objeto, puede estar a lo sumo un objeto relacionado con otro mediante esta propiedad.
- ⦿ Por ejemplo, si se tienen tres objetos que son A, B y C y se tiene una propiedad funcional tienePadre, entonces se podrían asociar los objetos A y B por medio de la propiedad y daría como resultado A tienePadre B. Igualmente se podrían asociar los objetos A y C por medio de la propiedad y daría como resultado A tienePadre C. Como tienePadre es propiedad funcional, se concluye que B y C son el mismo objeto.

PROPIEDAD FUNCIONAL



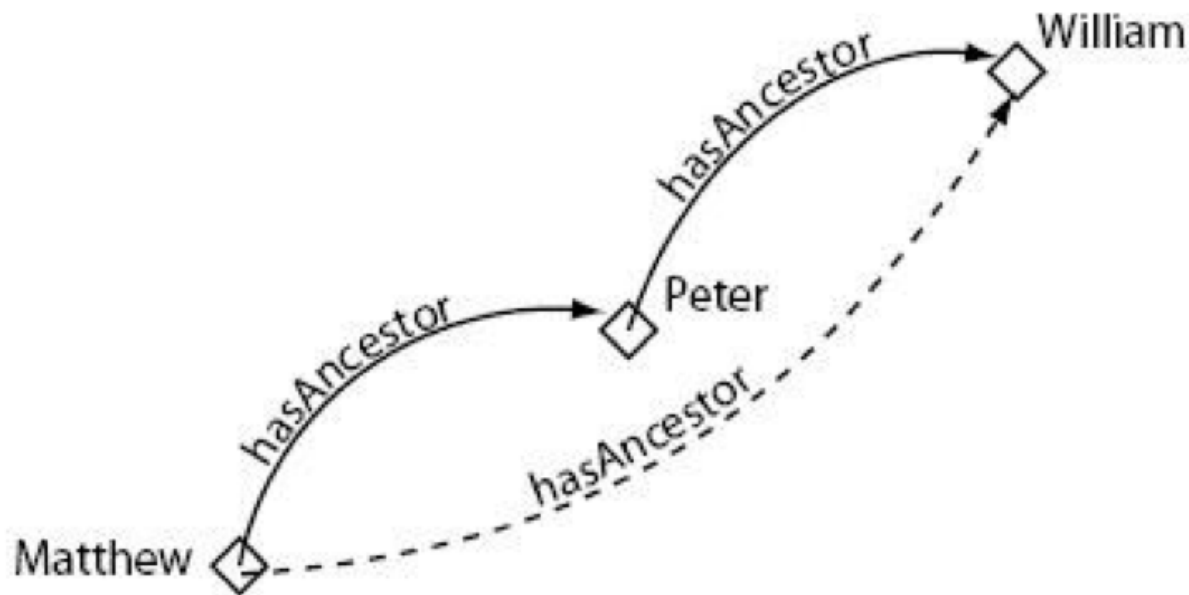
Si una propiedad es funcional, para un individuo determinado, no puede haber más de una persona que se relaciona con el individuo a través de la propiedad. En la figura se muestra un ejemplo de una propiedad funcional `hasBirthMother` - algo que sólo puede tener una madre biológica. Si decimos que el individuo Jean `hasBirthMother` Peggy y también se dice que el individuo Jean `hasBirthMother` Margaret, entonces porque `hasBirthMother` es una propiedad funcional, se puede inferir que Peggy y Margaret debe ser el mismo individuo. Cabe señalar, sin embargo, que si Peggy y Margaret se dice explícitamente que dos individuos diferentes entonces las declaraciones anteriores llevaría a una contradicción.

PROPIEDADES FUNCIONALES INVERSA

- ◉ Propiedades funcionales inversas: Si una propiedad es funcional inversa, indica que puede estar a lo sumo un objeto relacionado con otro mediante esta propiedad de forma inversa a la propiedad funcional original.
- ◉ Por ejemplo, si se tienen tres objetos que son A, B y C y se tiene una propiedad funcional esPadreDe, entonces se podría asociar el objeto B y A por medio de la propiedad y daría como resultado B esPadreDe A. Igualmente se podrían asociar los objetos C y A por medio de la propiedad y daría como resultado C esPadreDe A. Como esPadreDe es propiedad funcional inversa, se concluye que B y C son el mismo objeto.

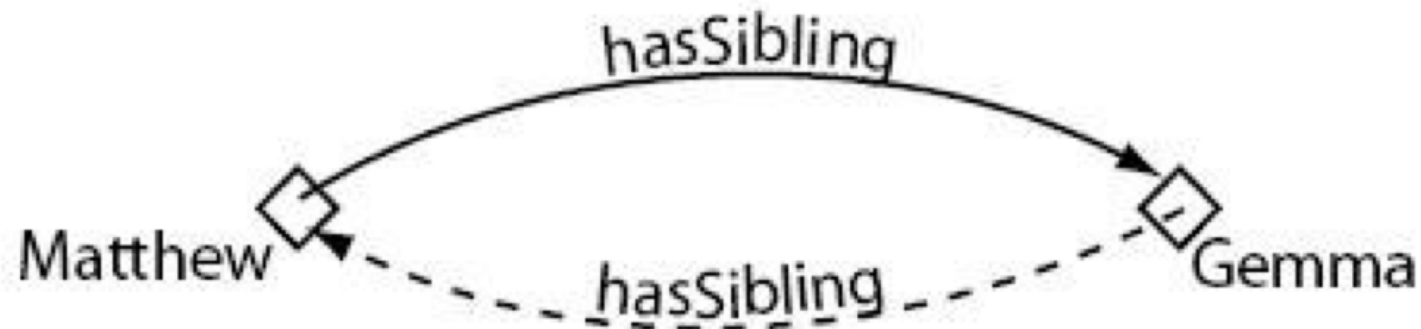
PROPIEDAD TRANSITIVA

Propiedades transitivas: Si una propiedad es transitiva y relaciona dos objetos A y B, y además hay una propiedad que relaciona al objeto B con otro C, entonces se puede inferir que el objeto A está relacionado con el objeto C mediante la propiedad transitiva.



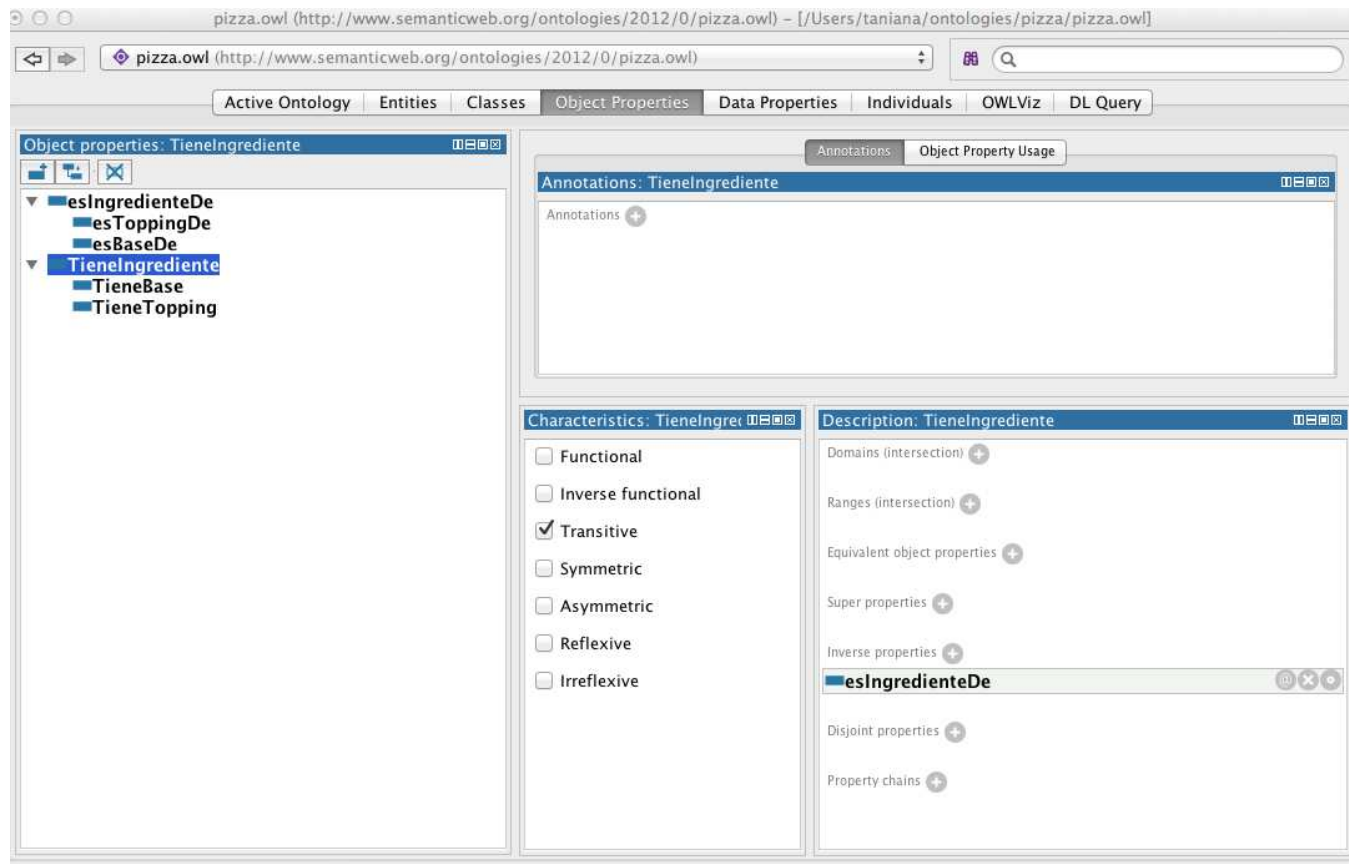
PROPIEDAD SIMÉTRICA

- Propiedades simétricas: Si una propiedad P es simétrica y la propiedad relaciona a los objetos A y B, entonces el objeto B es relacionado por medio de la propiedad P con el objeto A.



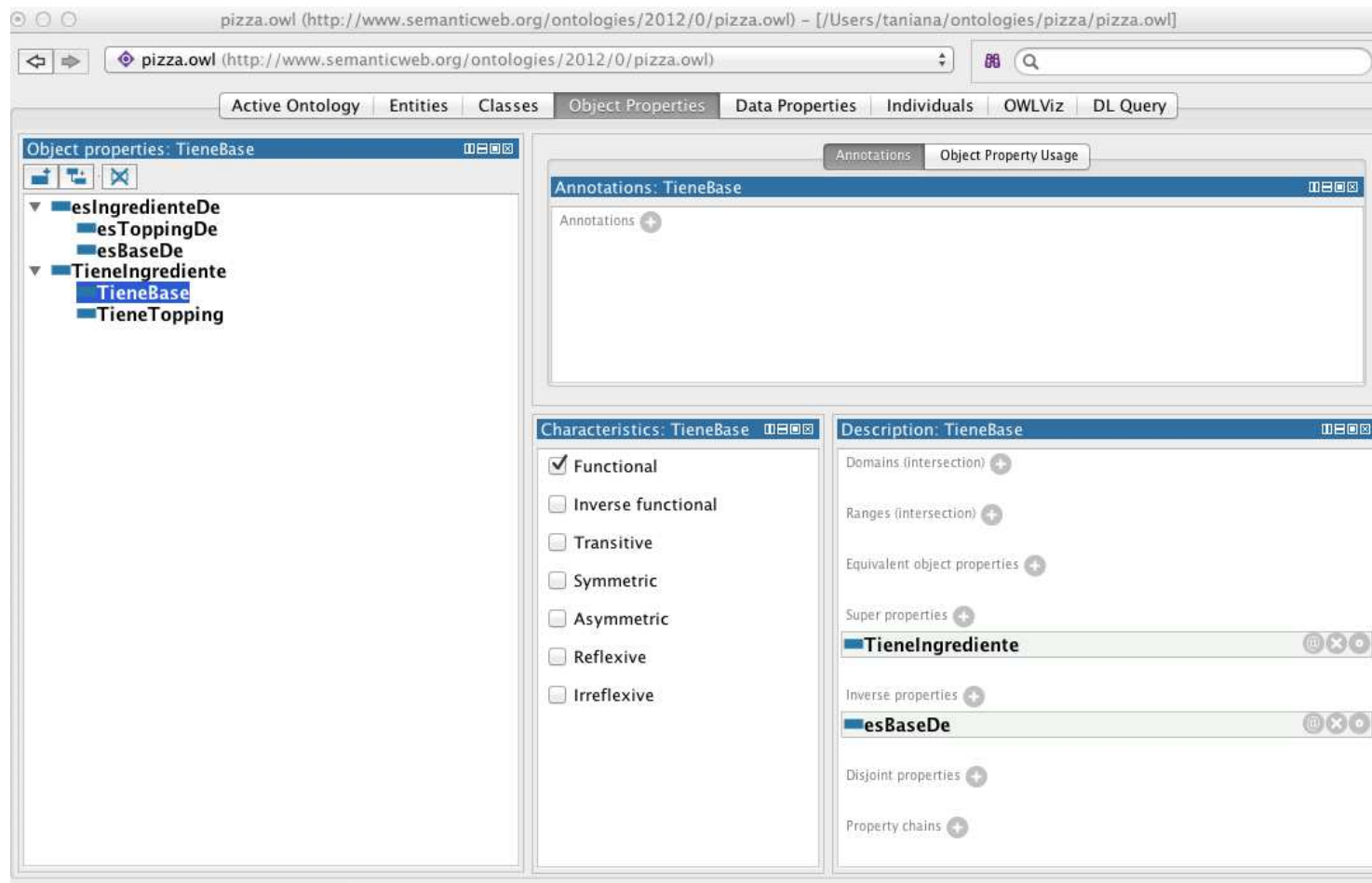
PROPIEDAD TRANSITIVA

Crear la propiedad tieneIngrediente propiedad Transitiva

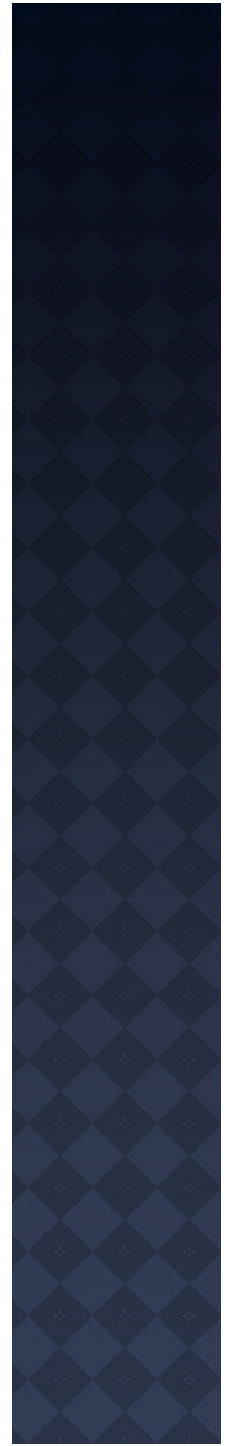


PROPIEDAD FUNCIONAL

- ◉ Crear TieneBase como propiedad funcional

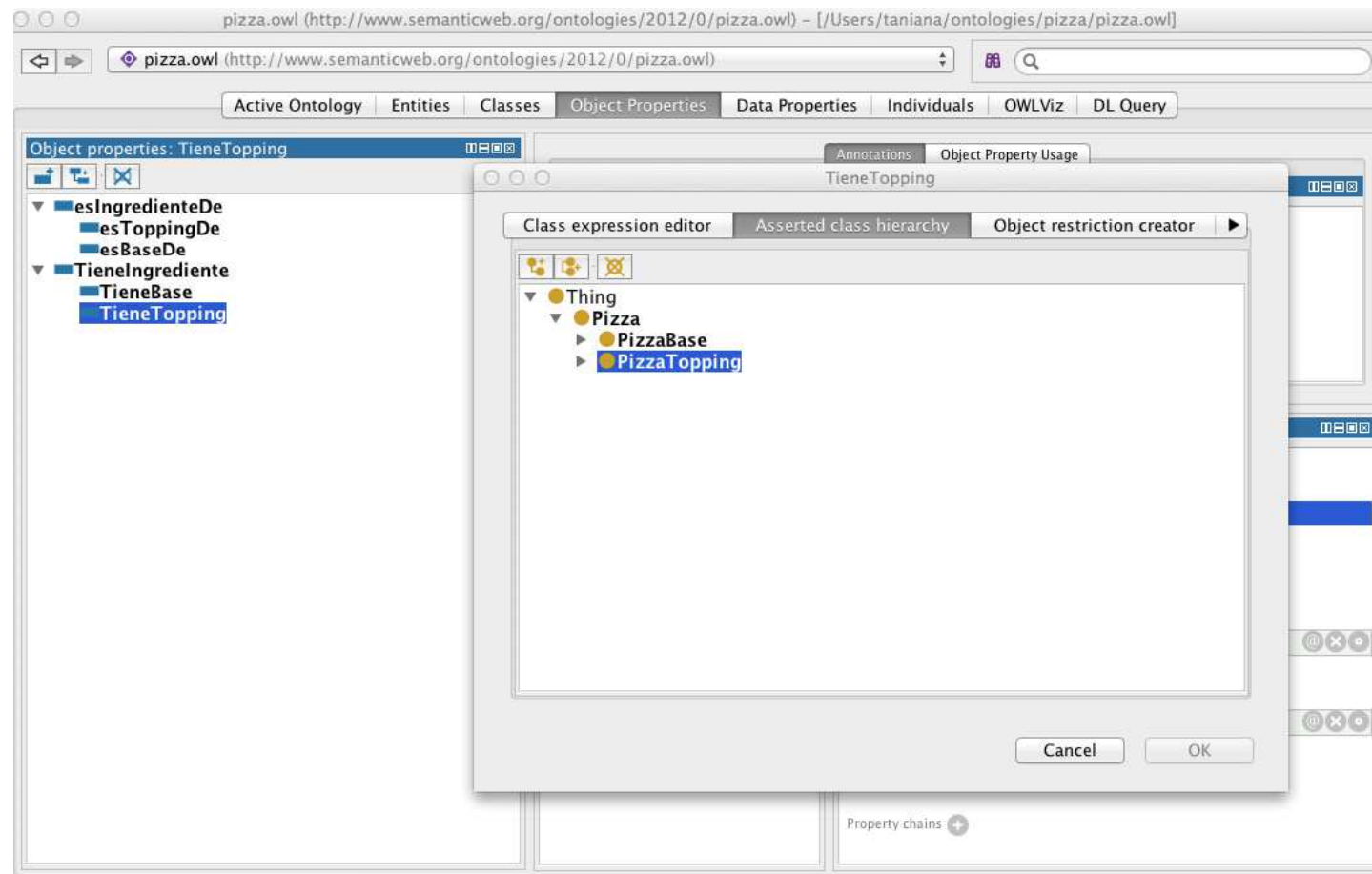


PROPIEDADES DE DOMINIO Y RANGO



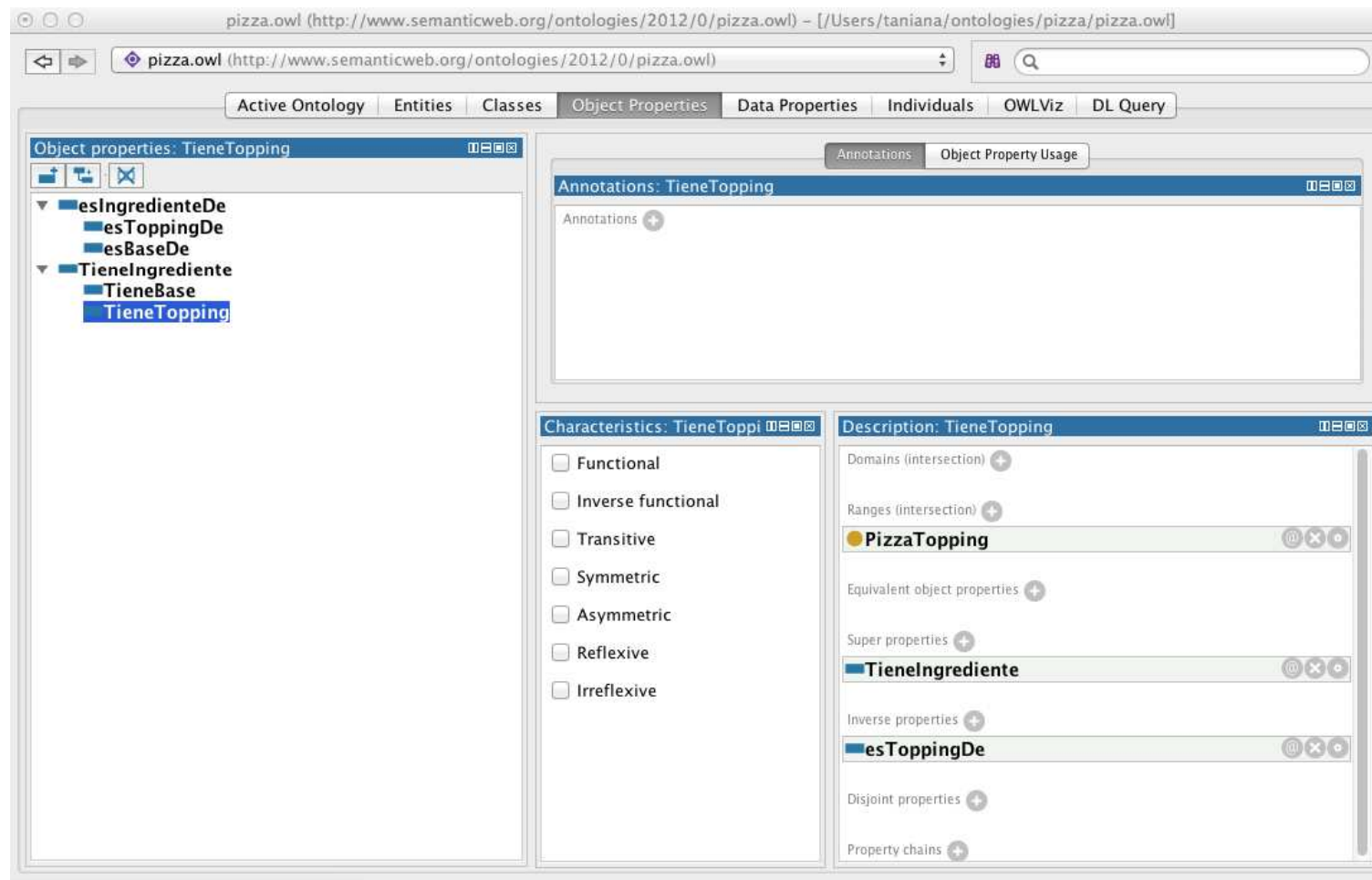
ESPECIFICAR EL RANGO

Especificar el rango de la propiedad tieneTopping



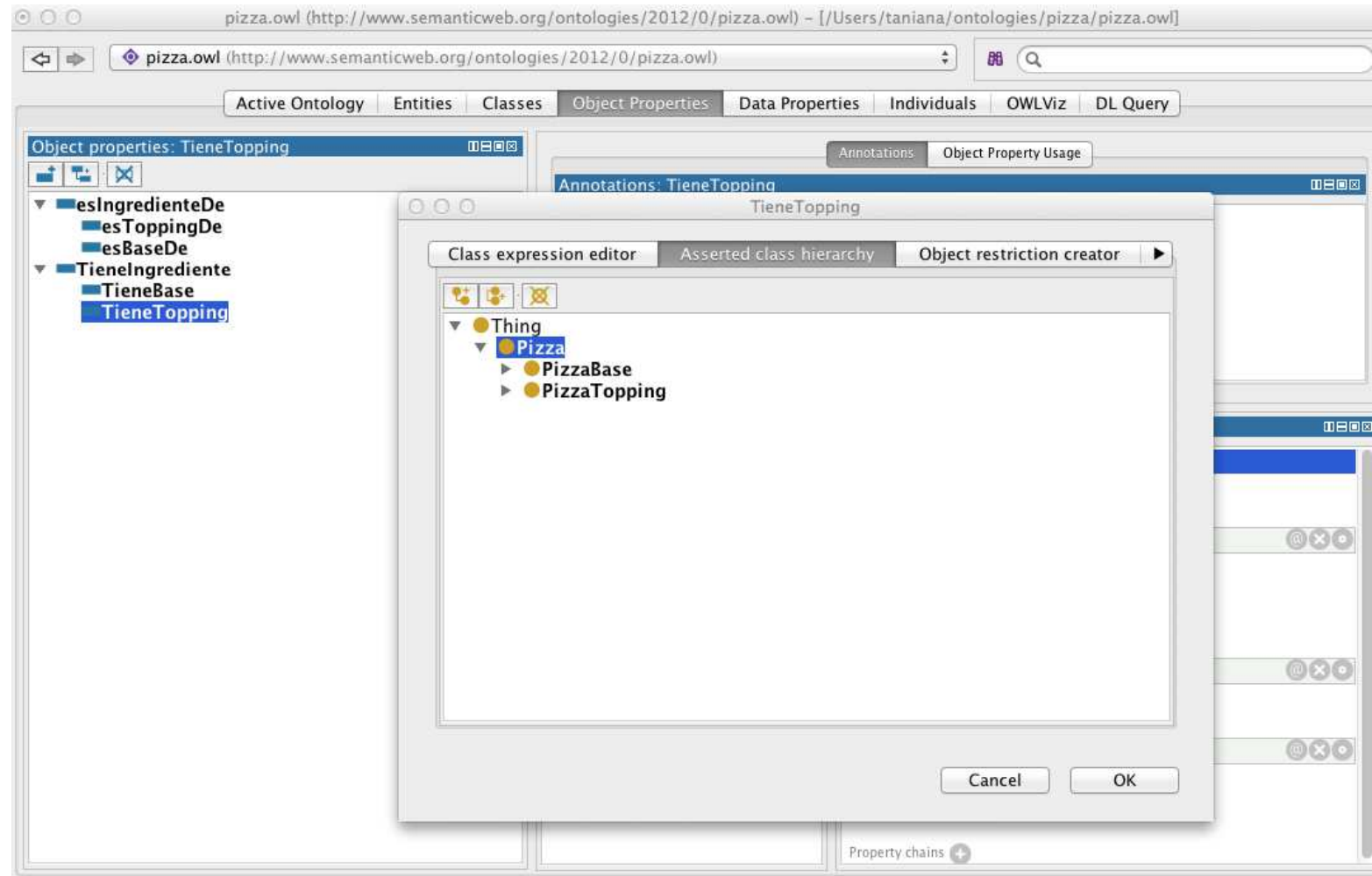
ESPECIFICAR EL RANGO

Especificar el rango de la propiedad tieneTopping



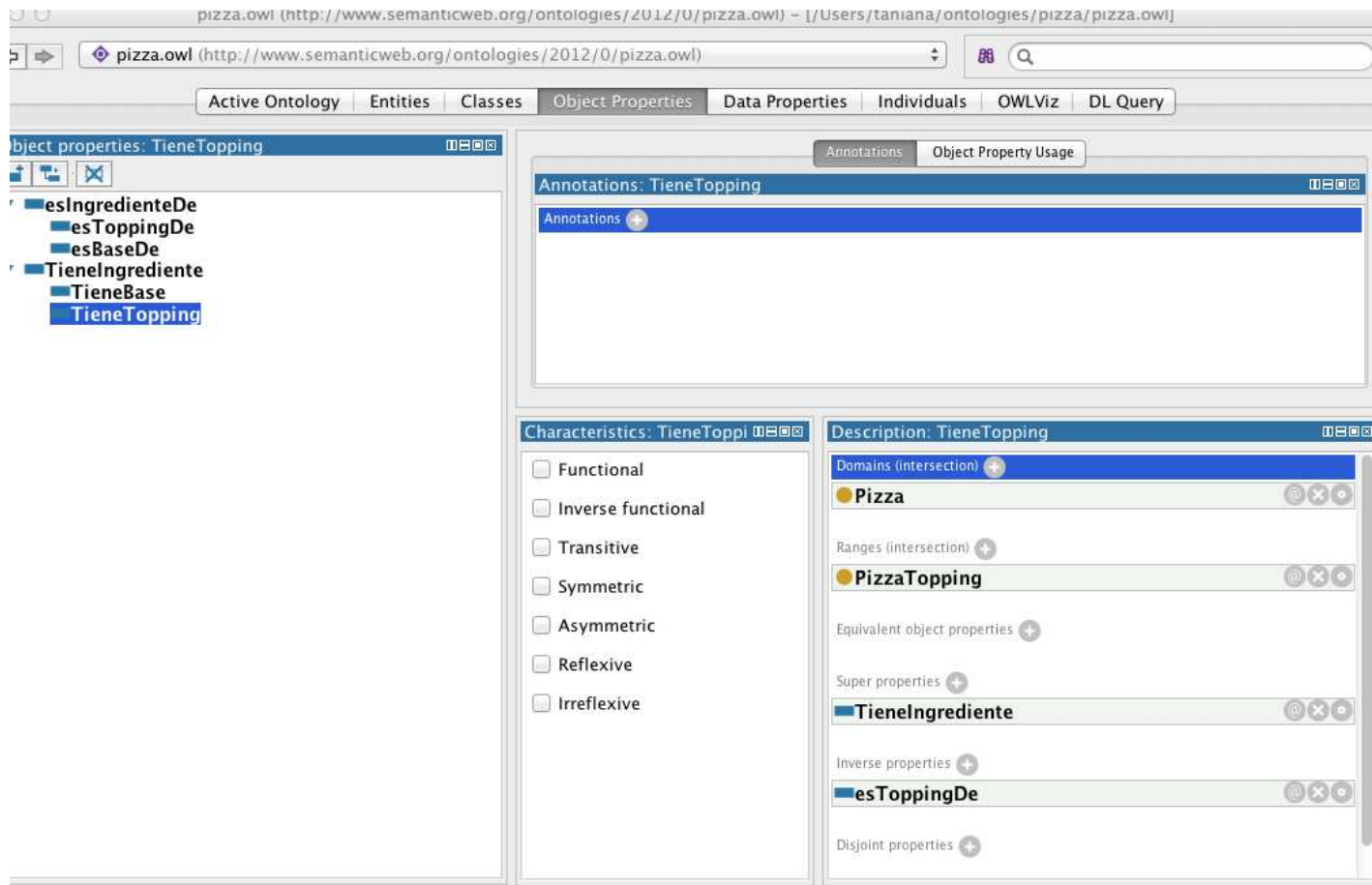
ESPECIFICAR EL DOMINIO

Especificar el dominio de la propiedad tieneTopping



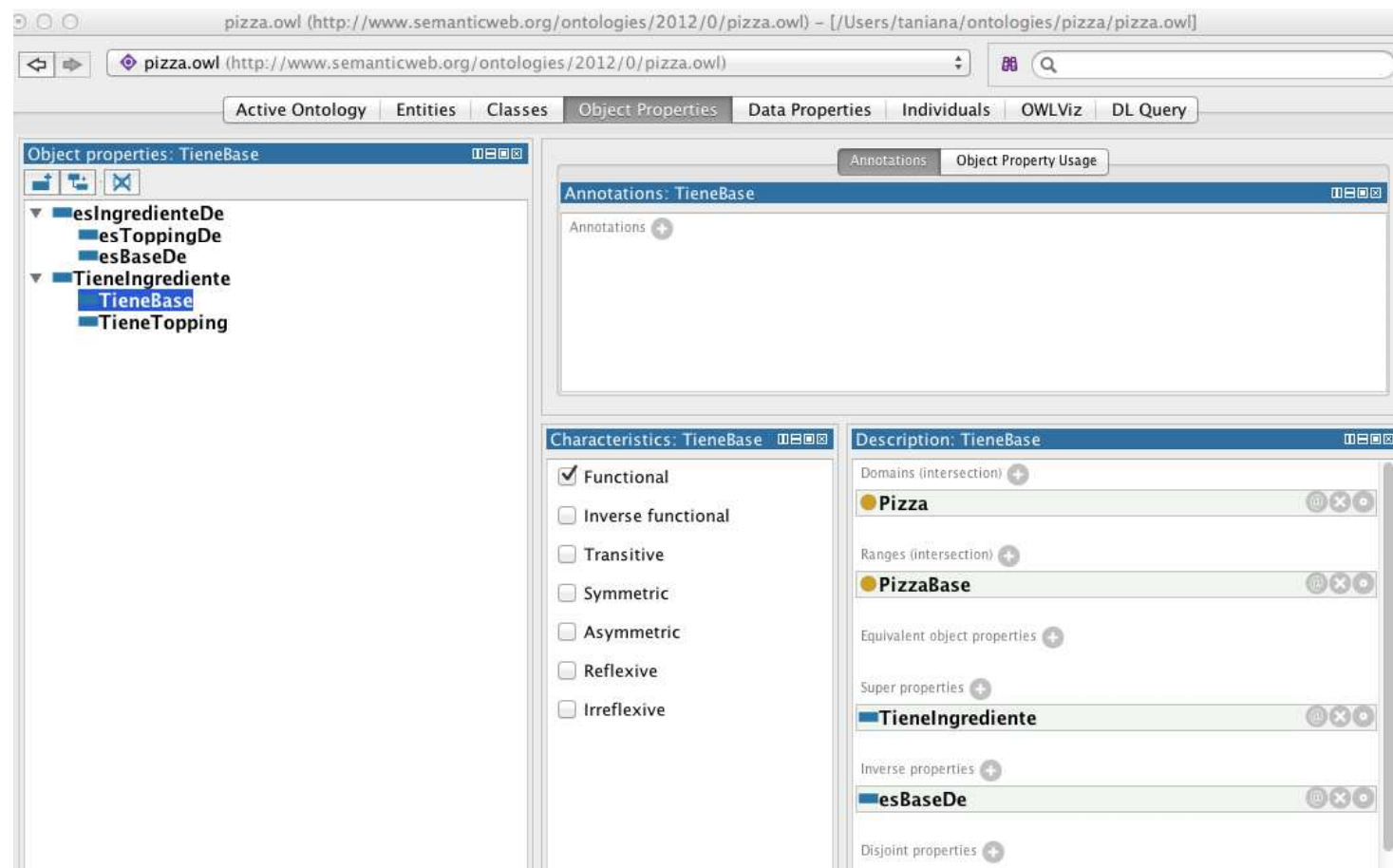
ESPECIFICAR EL DOMINIO

Especificar el dominio de la propiedad tieneTopping



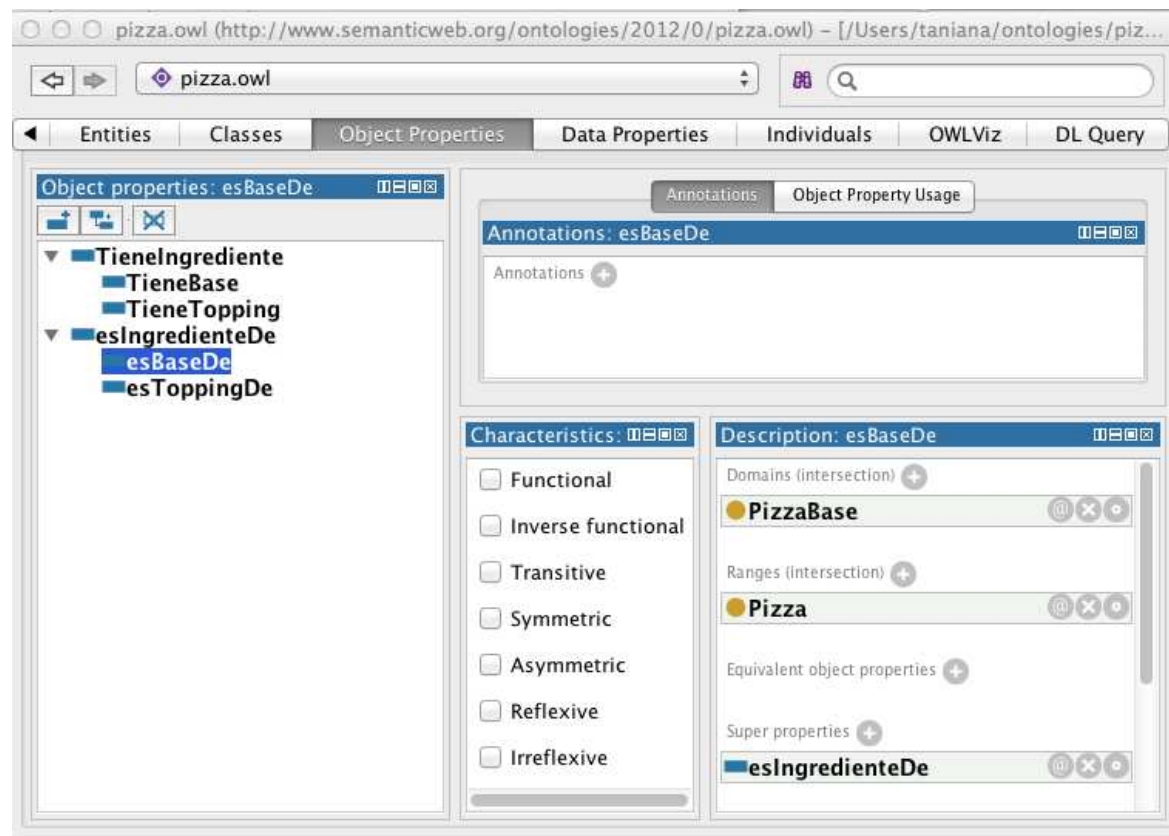
ESPECIFICAR EL DOMINIO Y RANGO

Especificar el dominio y rango de la propiedad tieneBase y su inversa de la propiedad esBaseDe

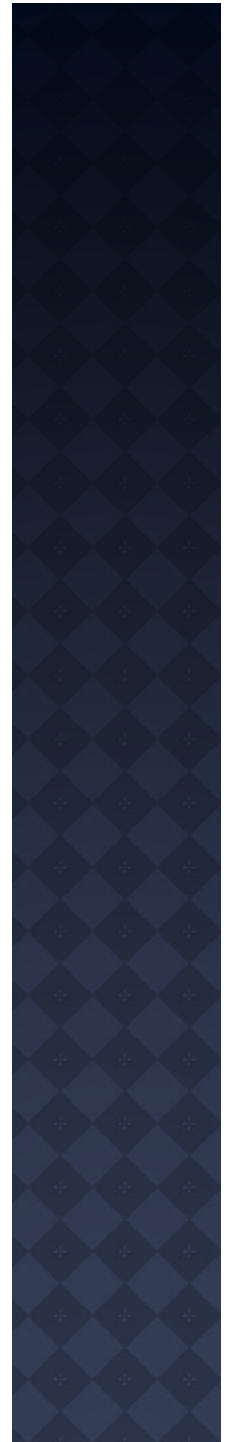


ESPECIFICAR EL DOMINIO Y RANGO

Especificar el dominio y rango de la propiedad tieneBase y su inversa de la propiedad esBaseDe



DESCRIBIENDO Y DEFINIENDO LAS CLASES



OWL: CONSTRUCTORES SOBRE LA DEFINICIÓN DE CLASES Y AXIOMAS PARA LAS CLASES, PROPIEDADES Y INDIVIDUOS

Intersection:	$C_1 \cap \dots \cap C_n$	intersectionOf	$\text{Human} \cap \text{Male}$
Union:	$C_1 \cup \dots \cup C_n$	unionOf	$\text{Doctor} \cup \text{Lawyer}$
Negation:	$\neg C$	complementOf	$\neg \text{Male}$
Nominals:	$\{x_1\} \cup \dots \cup \{x_n\}$	oneOf	$\{\text{john}\} \cup \dots \cup \{\text{mary}\}$
Universal restriction:	$\forall P.C$	allValuesFrom	$\forall \text{hasChild}.\text{Doctor}$
Existential restriction:	$\exists P.C$	someValuesFrom	$\exists \text{hasChild}.\text{Lawyer}$
Maximum cardinality:	$\leq nP$	maxCardinality	$\leq 3 \text{hasChild}$
Minimum cardinality:	$\geq nP$	minCardinality	$\geq 1 \text{hasChild}$
Specific Value:	$\exists P.\{x\}$	hasValue	$\exists \text{hasColleague}.\{\text{Matthew}\}$
Subclass	$C1 \subseteq C2$	subClassOf	$\text{Human} \subseteq \text{Animal} \cap \text{Biped}$
Equivalence	$C1 \equiv C2$	equivalentClass	$\text{Man} \equiv \text{Human} \cap \text{Male}$
Disjointness	$C1 \cap C2 \subseteq \perp$	disjointWith	$\text{Male} \cap \text{Female} \subseteq \perp$
Subproperty	$P1 \subseteq P2$	subPropertyOf	$\text{hasDaughter} \subseteq \text{hasChild}$
Equivalence	$P1 \equiv P2$	equivalentProperty	$\text{cost} \equiv \text{price}$
Inverse	$P1 \equiv P2^-$	inverseOf	$\text{hasChild} \equiv \text{hasParent}^-$
Transitive	$P^+ \subseteq P$	TransitiveProperty	$\text{ancestor}^+ \subseteq \text{ancestor}$
Functional	$T \subseteq \leq 1P$	FunctionalProperty	$T \subseteq \leq 1 \text{hasMother}$
InverseFunctional	$T \subseteq \leq 1P^-$	InverseFunctionalProperty	$T \subseteq \leq 1 \text{hasPassportID}^-$
Equivalence	$\{x1\} \equiv \{x2\}$	sameIndividualAs	$\{\text{oeg:OscarCorcho}\} \equiv \{\text{img:Oscar}\}$
Different	$\{x1\} \equiv \neg \{x2\}$	differentFrom, AllDifferent	$\{\text{john}\} \equiv \neg \{\text{peter}\}$

DESCRIBIENDO Y DEFINIENDO LAS CLASES

- ⦿ Una vez creadas varias propiedades, se pueden utilizar para definir y describir el comportamiento de las clases.
- ⦿ Restricciones de propiedades
 - Las propiedades son utilizadas para crear restricciones en las clases en una ontología OWL. Usualmente el nombre de la propiedad debería sugerir las restricciones impuestas a los objetos de la clase. Las restricciones OWL se presentan en las siguientes tres categorías:
 - ⦿ Restricciones de cuantificación.
 - ⦿ Restricciones de cardinalidad.
 - ⦿ Restricciones de valor.

RESTRICCIONES CUANTIFICACIÓN

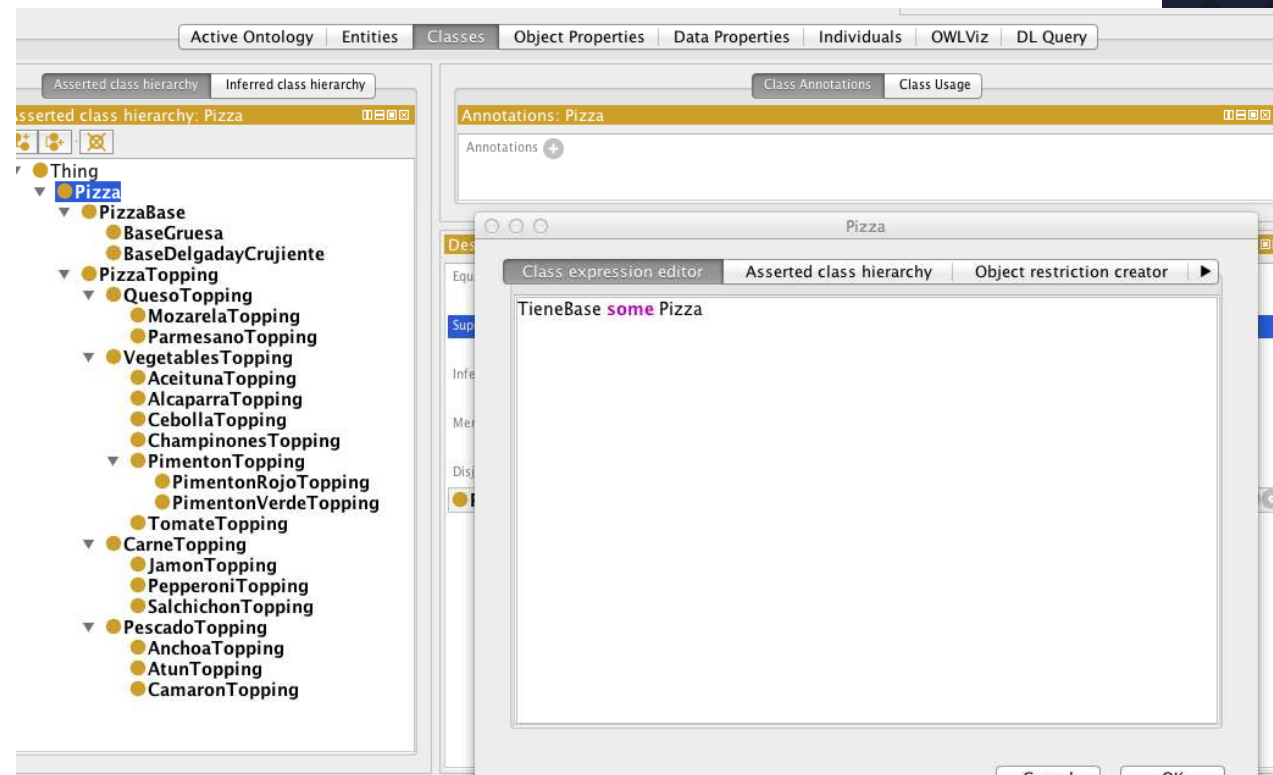
DE

- ⦿ Las restricciones de cuantificación se componen de los siguientes elementos:
 - Cuantificador existencial ($\exists$), el cual permite indicar la existencia de al menos un objeto. En protégé 4 la palabra clave **some** es usado para denotar $\exists$
 - Cuantificador universal ($\forall$), el cual permite indicar la existencia de todos los objetos. En protégé 4. la palabra clave es **only** es usado para denotar $\forall$

CUANTIFICADOR UNIVERSAL

Por ejemplo la restricción para pizza que especifica que una pizza debe tener una PizzaBase (**Existencial**)

1. Seleccione Pizza
2. Seleccione en el icon de agregar al lado de Superclasse.
3. En la ventana Class expression editor. Coloque TieneBase some Pizza

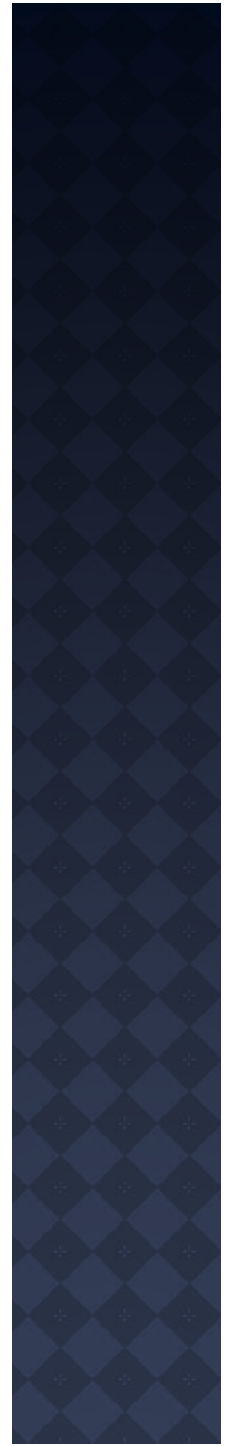


CUANTIFICADOR UNIVERSAL

Por ejemplo la restricción para pizza que especifica que una pizza debe tener una PizzaBase

The screenshot displays a software interface with two main panels. The left panel, titled 'Asserted class hierarchy' and 'Inferred class hierarchy', shows a tree structure of classes. The 'Pizza' class is highlighted, and its subclasses are listed: 'PizzaBase', 'BaseGruesa', 'BaseDelgadayaCrujiente', 'PizzaTopping', 'QuesoTopping', 'MozarelaTopping', 'ParmesanoTopping', 'VegetablesTopping', 'AceitunaTopping', 'AlcaparraTopping', 'CebollaTopping', 'ChampiononesTopping', 'PimentonTopping', 'PimentonRojoTopping', 'PimentonVerdeTopping', 'TomateTopping', 'CarneTopping', 'JamonTopping', 'PepperoniTopping', 'SalchichonTopping', 'PescadoTopping', 'AnchoaTopping', 'AtunTopping', and 'CamaronTopping'. The right panel, titled 'Class Annotations' and 'Class Usage', shows the 'Annotations: Pizza' section with a plus icon for adding annotations. Below this, the 'Description: Pizza' section shows 'Equivalent classes' with a plus icon, 'Superclasses' with a plus icon, and a list of superclasses including 'TieneBase some Pizza'. The 'Inferred anonymous superclasses' section is empty. The 'Members' section has a plus icon. The 'Disjoint classes' section has a plus icon. The 'Class Usage' section shows a list of classes including 'PizzaTopping, PizzaBase'.

CREANDO ALGUNAS DIFERENTES CLASES DE PIZZAS



CREANDO ALGUNAS CLASES DE PIZZA

Crear una subclase de Pizza llamada NombrePizza y una subclase de pizza llamada PizzaMargarita

The screenshot displays a software interface for managing a class hierarchy. On the left, the 'Asserted class hierarchy' pane shows a tree structure starting with 'Thing', followed by 'Pizza'. Under 'Pizza', 'NombrePizza' is highlighted, and below it, 'PizzaMargarita' is listed. Other classes like 'PizzaBase', 'BaseGruesa', 'BaseDelgadoCrujiente', and various 'Topping' classes are also visible. On the right, the 'Class Annotations' pane for 'NombrePizza' is active. It shows 'Annotations' (empty), 'Description' (empty), 'Equivalent classes' (empty), 'Superclasses' (listing 'Pizza'), 'Inferred anonymous superclasses' (listing 'TieneBase some Pizza'), 'Members' (empty), and 'Disjoint classes' (empty).

CREANDO ALGUNAS CLASES DE PIZZA

Ya teniendo creada la clase PizzaMargarita Necesitamos especificar que tipo de topping tiene. Por lo tanto necesitamos dos restricciones que diga que tiene MozarellaTopping y otra que diga que tiene tomatesTopping

The screenshot displays the Protégé ontology editor interface. The top navigation bar includes tabs for 'Active Ontology', 'Entities', 'Classes', 'Object Properties', 'Data Properties', 'Individuals', 'OWLviz', and 'DL Query'. The 'Classes' tab is active, showing the 'Asserted class hierarchy' for 'PizzaMargarita' on the left and the 'Class Annotations' and 'Class Usage' panels on the right.

Asserted class hierarchy: PizzaMargarita

- Thing
 - Pizza
 - NombrePizza
 - PizzaMargarita**
 - PizzaBase
 - BaseGruesa
 - BaseDelgadayCrujiente
 - PizzaTopping
 - QuesoTopping
 - MozarellaTopping
 - ParmesanoTopping
 - VegetablesTopping
 - AceitunaTopping
 - AlcaparraTopping
 - CebollaTopping
 - ChampinonesTopping
 - PimentonTopping
 - PimentonRojoTopping
 - PimentonVerdeTopping
 - TomateTopping
 - CarneTopping
 - JamonTopping
 - PepperoniTopping
 - SalchichonTopping
 - PescadoTopping

Class Annotations: PizzaMargarita

Annotations +

comment

"Una pizizza que solo contiene queso mozzarella y tomate de toppinn"

Description: PizzaMargarita

Equivalent classes +

Superclasses +

- NombrePizza
- TieneTopping **some** MozarellaTopping
- TieneTopping **some** TomateTopping

Inferred anonymous superclasses

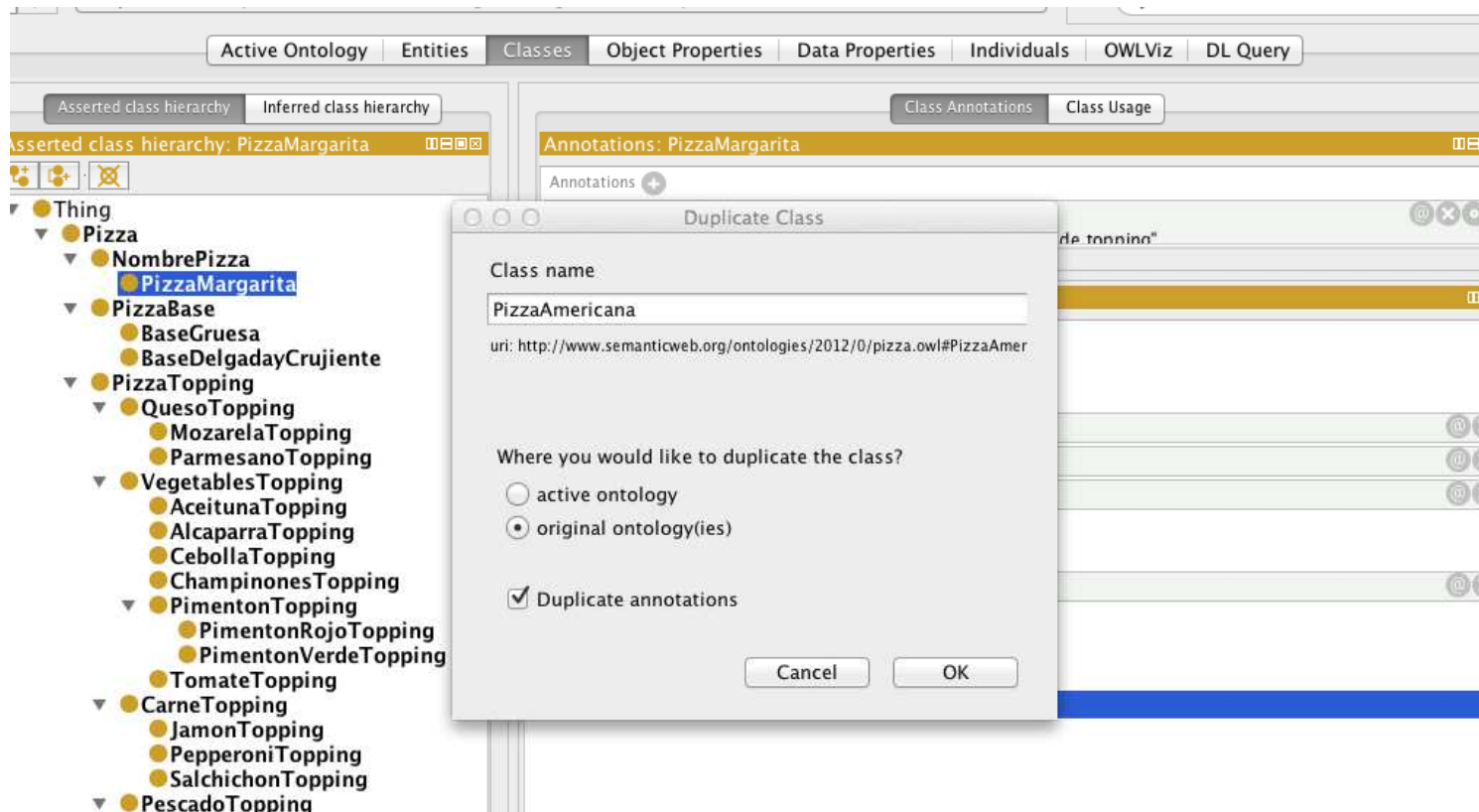
- TieneBase **some** Pizza

Members +

Disjoint classes +

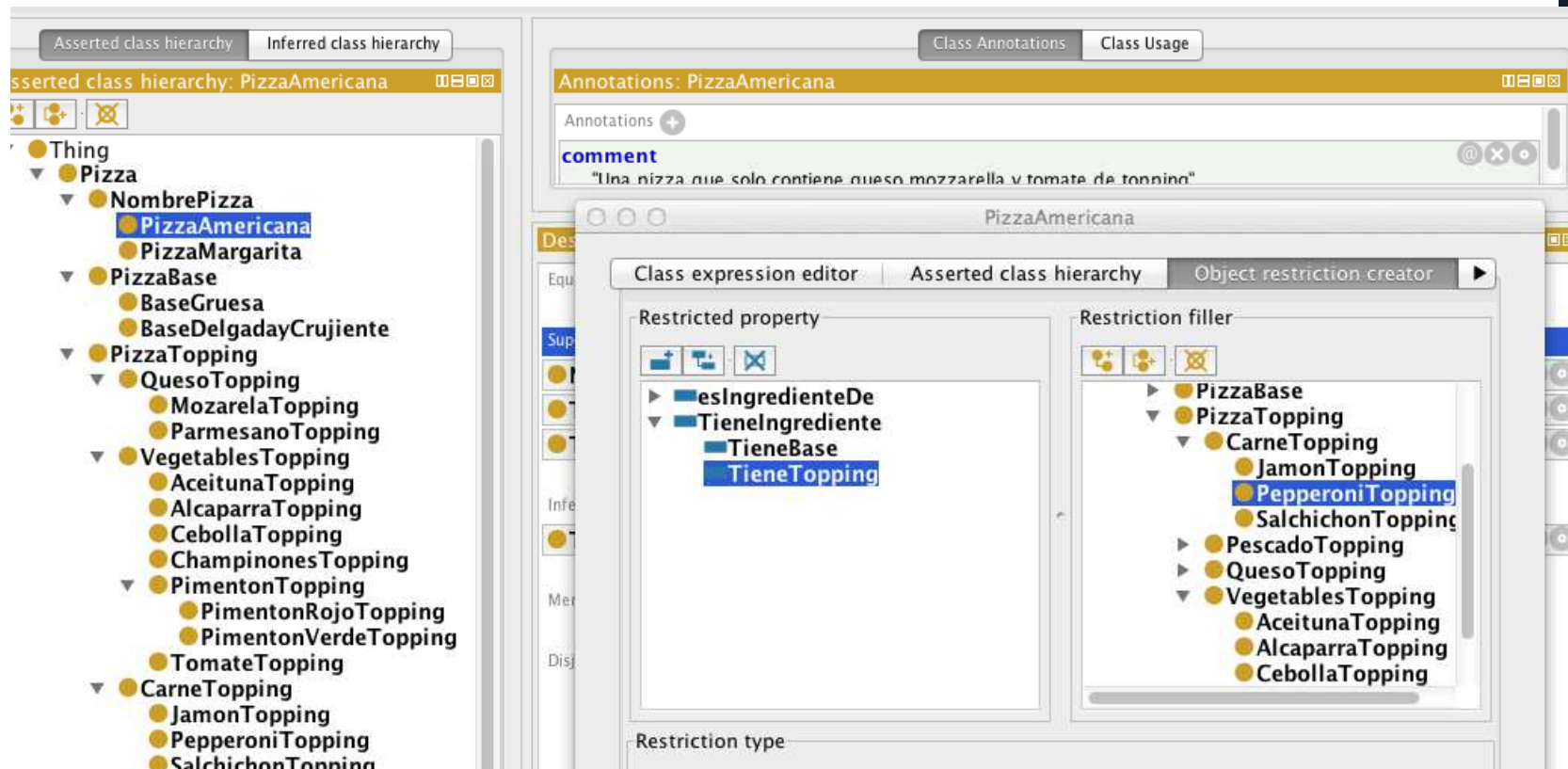
CREANDO ALGUNAS CLASES DE PIZZA

Ahora vamos a crear una clase que representa la PizzaAmericana, la cual tiene el PaperoniTopping, MozarellaTopping y tomateTopping. Debido a que la pizzaAmericana es parecida a PizzaMozzarella vamos a clonarla



CREANDO ALGUNAS CLASES DE PIZZA

Ahora vamos a crear una clase que representa la PizzaAmericana, la cual tiene el PaperoniTopping, MozarellaTopping y tomateTopping. Debido a que la pizzaAmericana es parecida a PizzaMozzarella vamos a clonarla



CREANDO ALGUNAS CLASES DE PIZZA

Ahora vamos a crear una clase que representa la PizzaAmericana, la cual tiene el PaperoniTopping, MozzarellaTopping y tomateTopping. Debido a que la pizzaAmericana es parecida a PizzaMozzarella vamos a clonarla

The screenshot displays the Protégé ontology editor interface. The top navigation bar includes tabs for 'Active Ontology', 'Entities', 'Classes', 'Object Properties', 'Data Properties', 'Individuals', 'OWL Viz', and 'DL Query'. The 'Classes' tab is active.

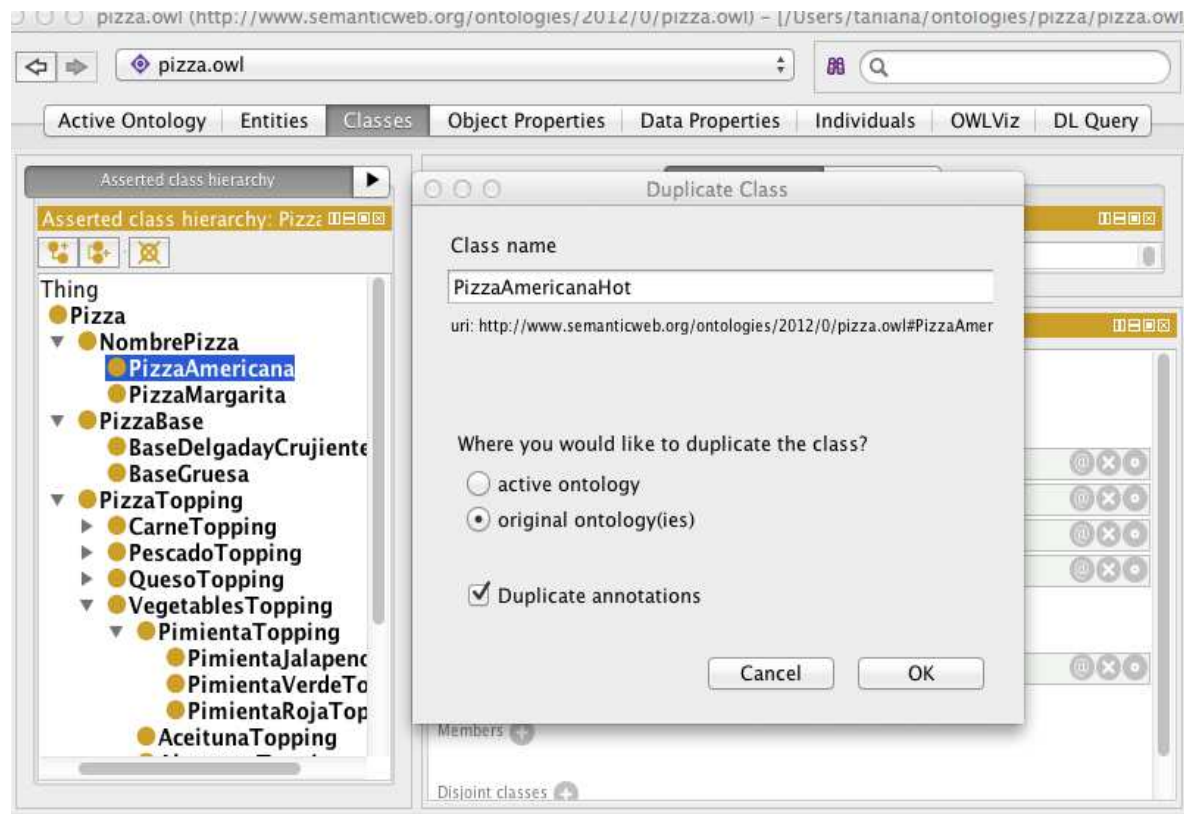
On the left, the 'Asserted class hierarchy' pane shows a tree structure. The 'Pizza' class is expanded, revealing subclasses: 'NombrePizza' (which includes 'PizzaAmericana' and 'PizzaMargarita'), 'PizzaBase' (with subclasses 'BaseGruesa' and 'BaseDelgadoCrujiente'), and 'PizzaTopping' (which includes 'QuesoTopping', 'VegetablesTopping', and 'PimentonTopping').

The right pane shows the details for the 'PizzaAmericana' class. It includes a 'comment' with the text: "Una pizza que solo contiene queso mozzarella y tomate de topping". The 'Description' section lists the following superclasses:

- NombrePizza
- TieneTopping some MozzarellaTopping
- TieneTopping some PepperoniTopping
- TieneTopping some TomateTopping
- Inferred anonymous superclasses: TieneBase some Pizza

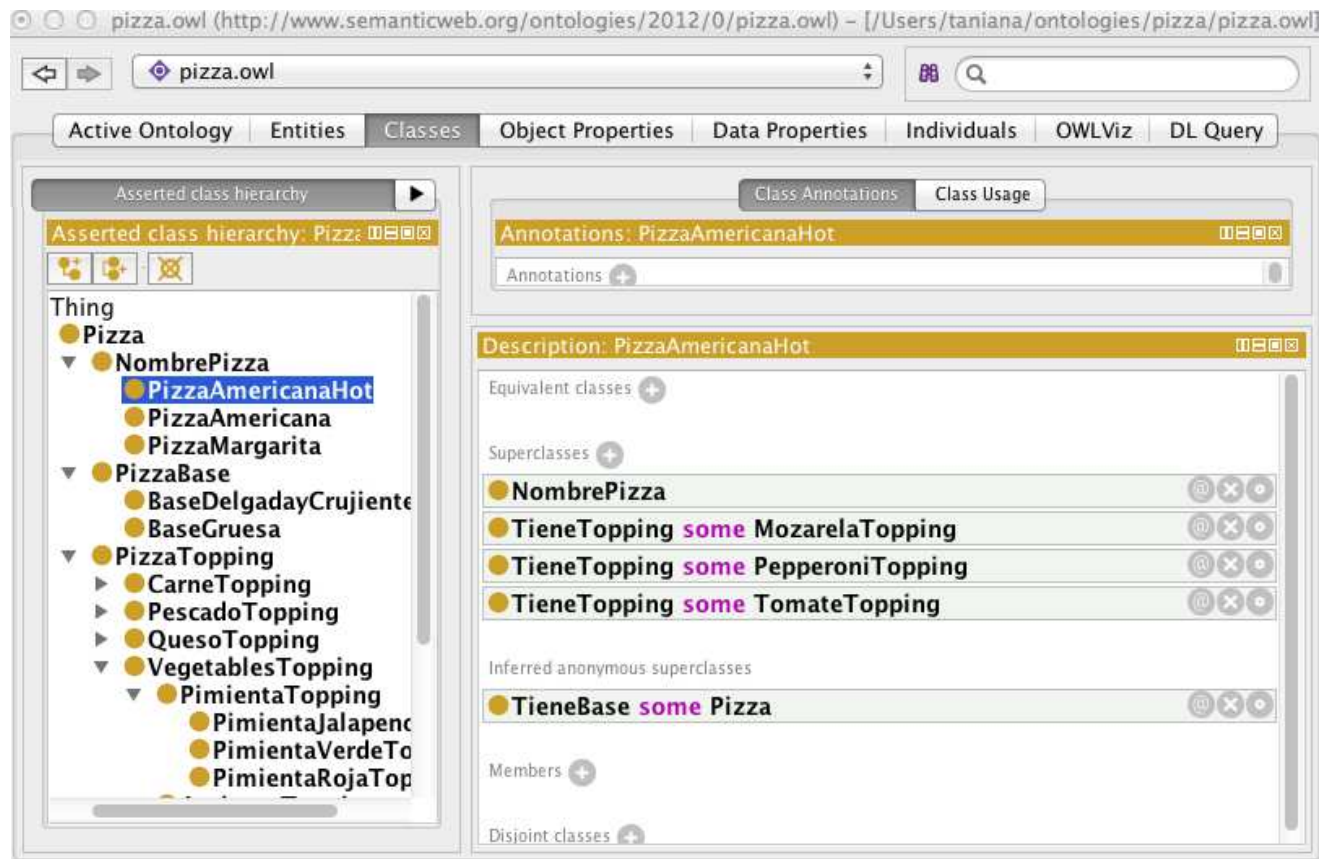
CREANDO ALGUNAS CLASES DE PIZZA

Crear una pizza PizzaAmericanaHot que es parecida a la PizzaAmericana, pero tiene PimiemientaJalapenoTopping, utilizando el procedimiento de clonar la PizzaAmericana



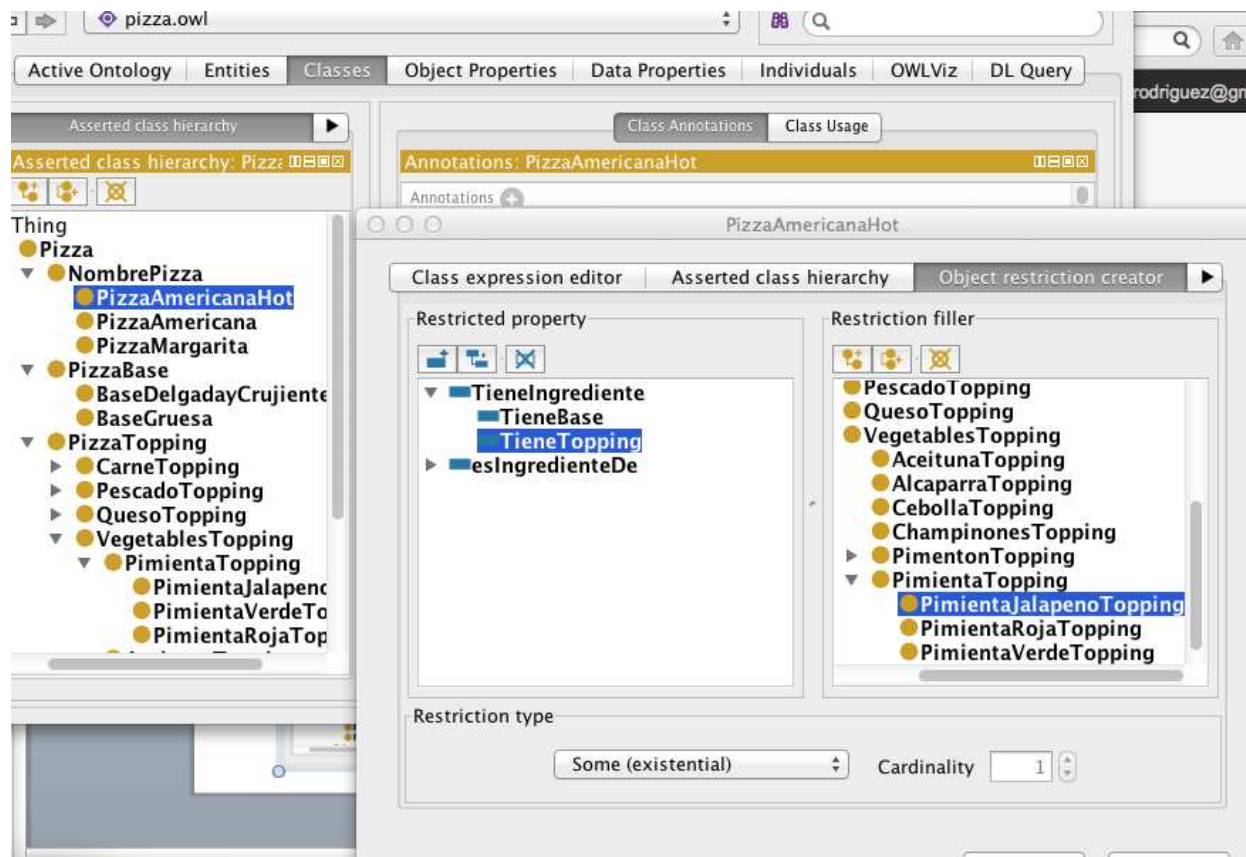
CREANDO ALGUNAS CLASES DE PIZZA

Crear una pizza `PizzaAmericanaHot` que es parecida a la `PizzaAmericana`, pero tiene `PimiemientaJalapenoTopping`, utilizando el procedimiento de clonar la `PizzaAmericana`



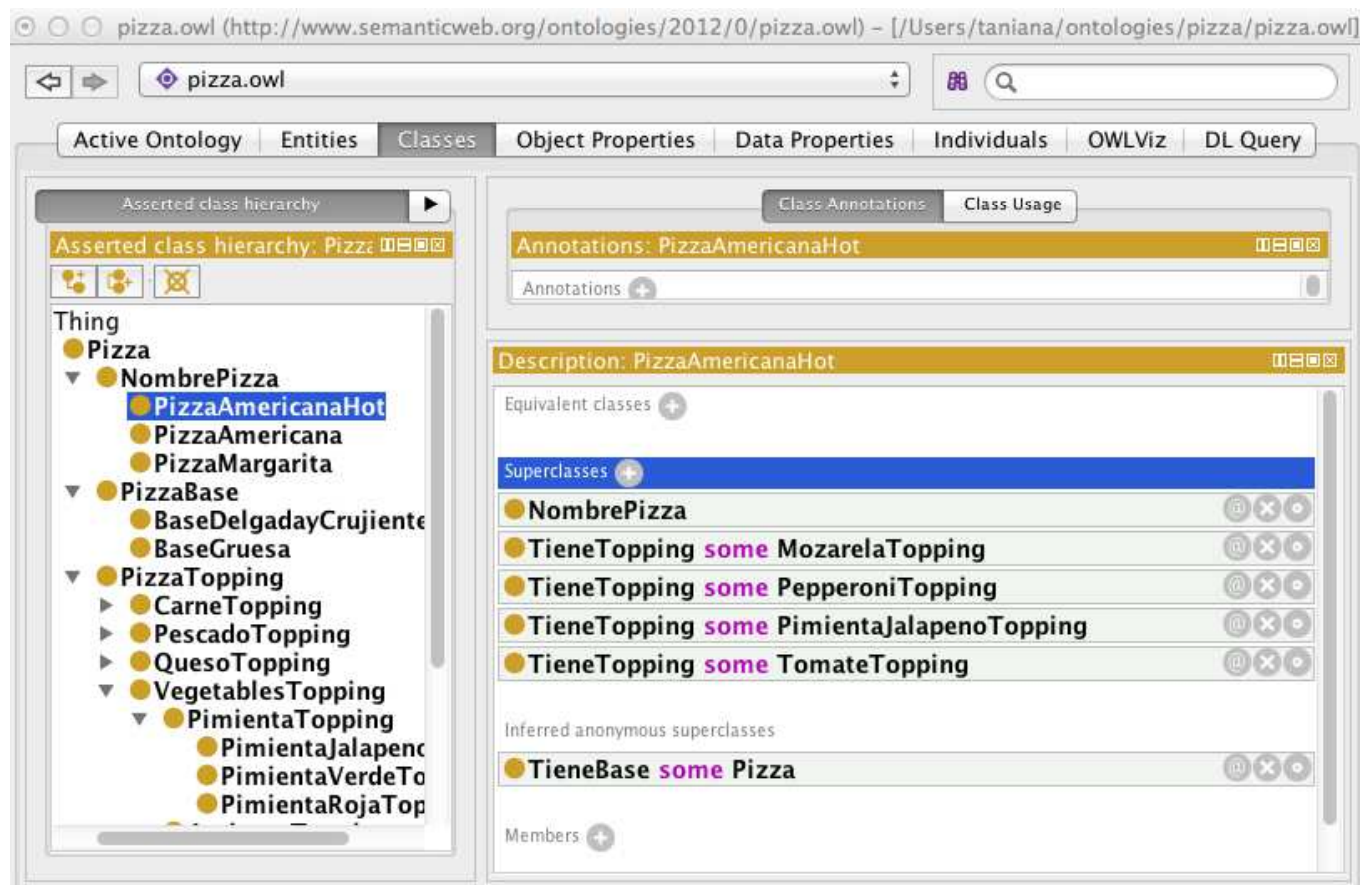
CREANDO ALGUNAS CLASES DE PIZZA

Crear una pizza PizzaAmericanaHot que es parecida a la PizzaAmericana, pero tiene PimiemientaJalapenoTopping, utilizando el procedimiento de clonar la PizzaAmericana



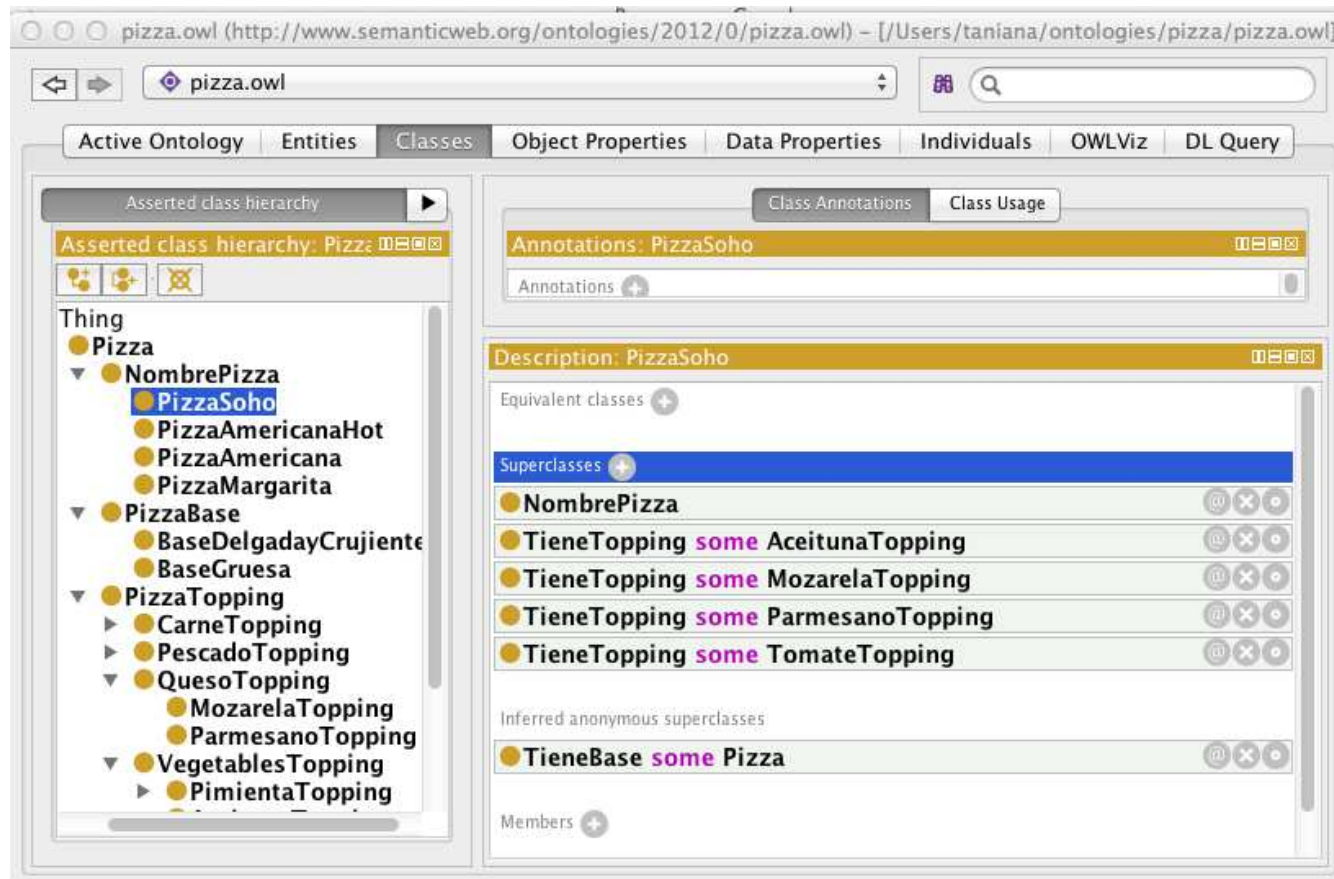
CREANDO ALGUNAS CLASES DE PIZZA

Crear una pizza PizzaAmericanaHot que es parecida a la PizzaAmericana, pero tiene PimiemientaJalapenoTopping, utilizando el procedimiento de clonar la PizzaAmericana



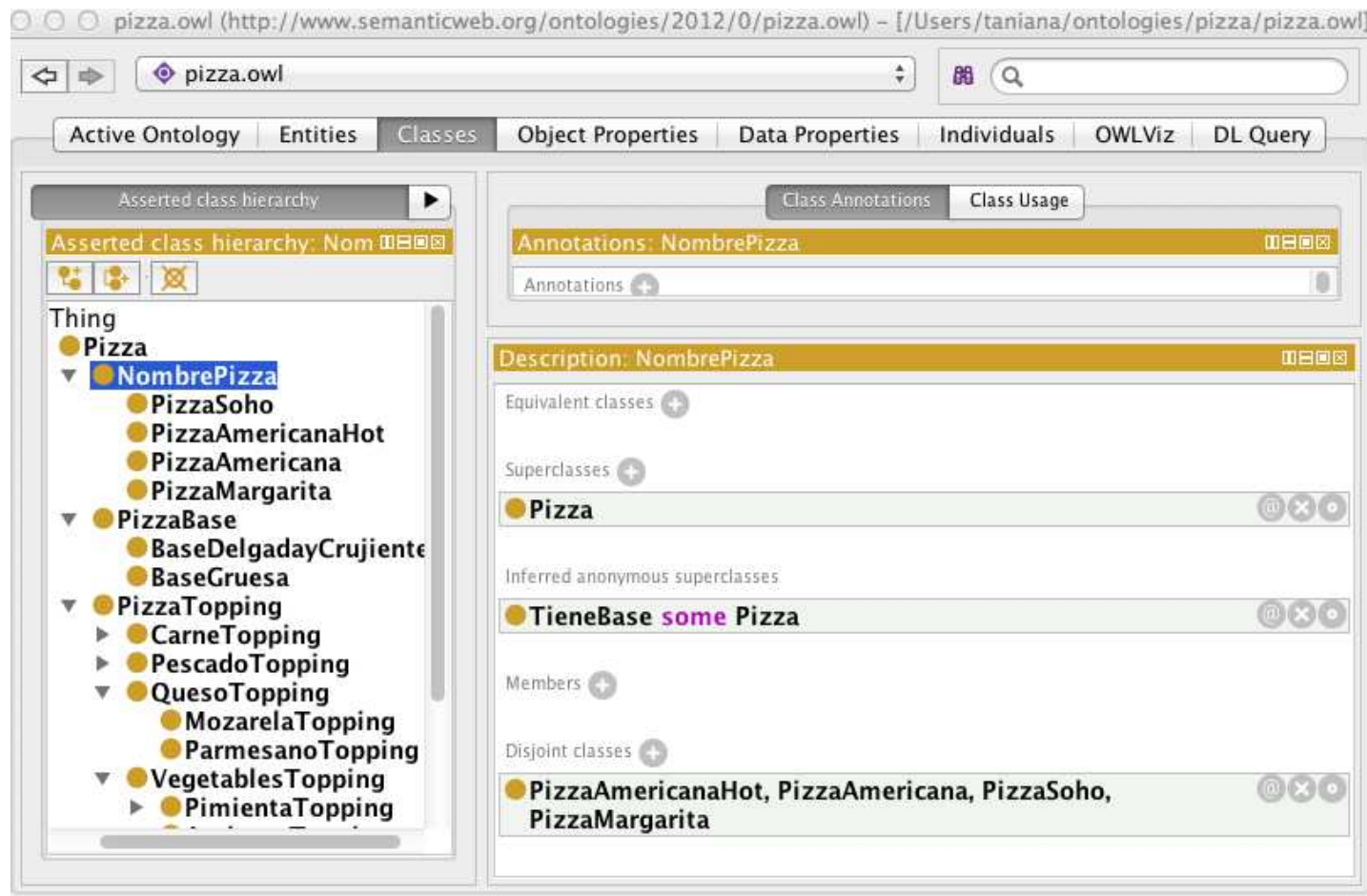
CREANDO ALGUNAS CLASES DE PIZZA

Como los pasos para crear una pizza PizzaAmericanaHot vamos a crear la PizzaSoho que es parecido a la PizzaMargarita pero tiene adicional AceitunasTopping y ParmezanoTopping



CREANDO ALGUNAS CLASES DE PIZZA

Por Ultimo realizar la subclases de NombrePizzas Disjunta



USANDO RAZONADOR

USANDO E INVOCANDO EL RAZONADOR

Una característica clave de las ontologías que son descritas usando OWL-DL es que pueden ser procesadas por un razonador.

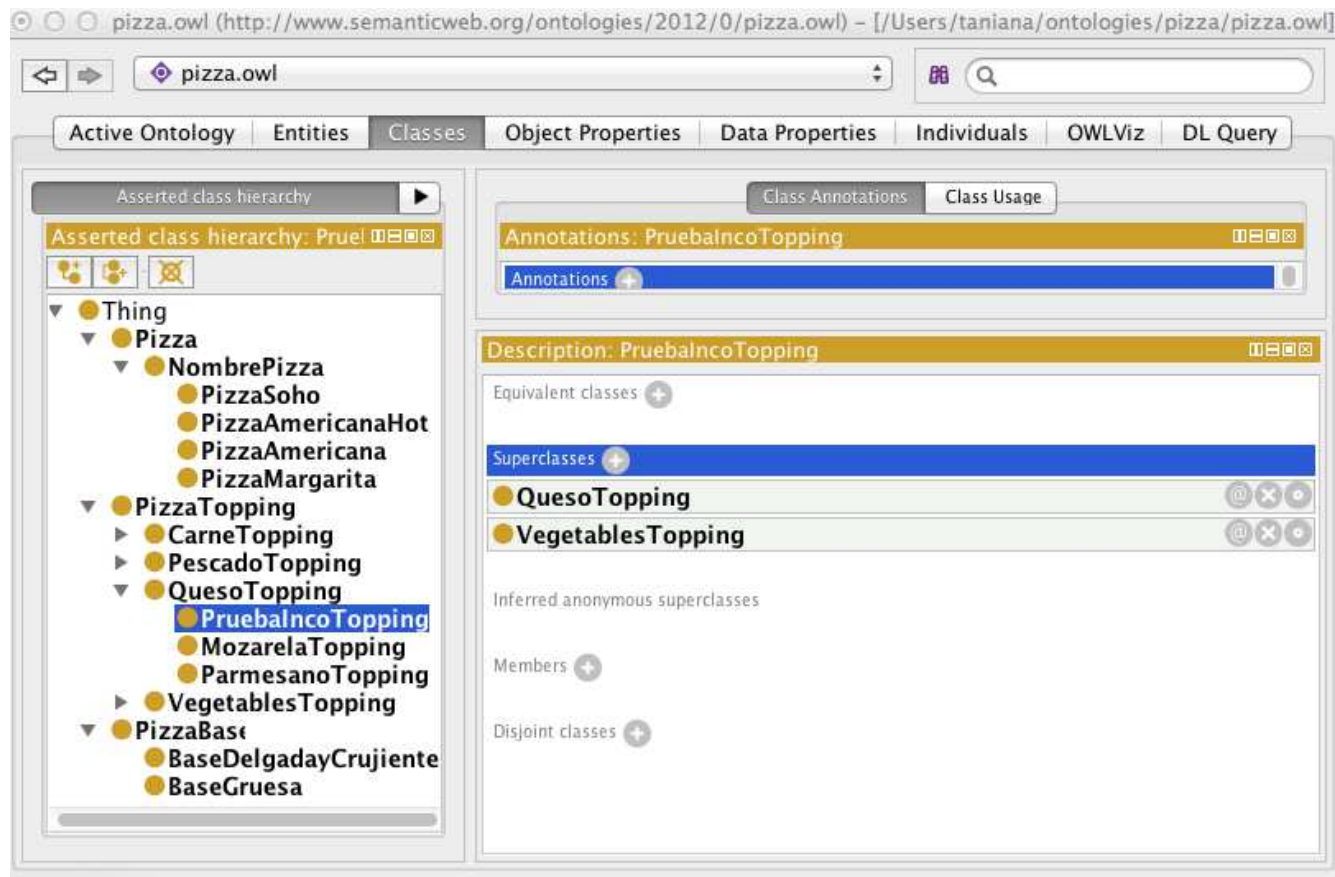
Los razonadores, llamados clasificadores, también pueden chequear consistencia

Invocando el razonador



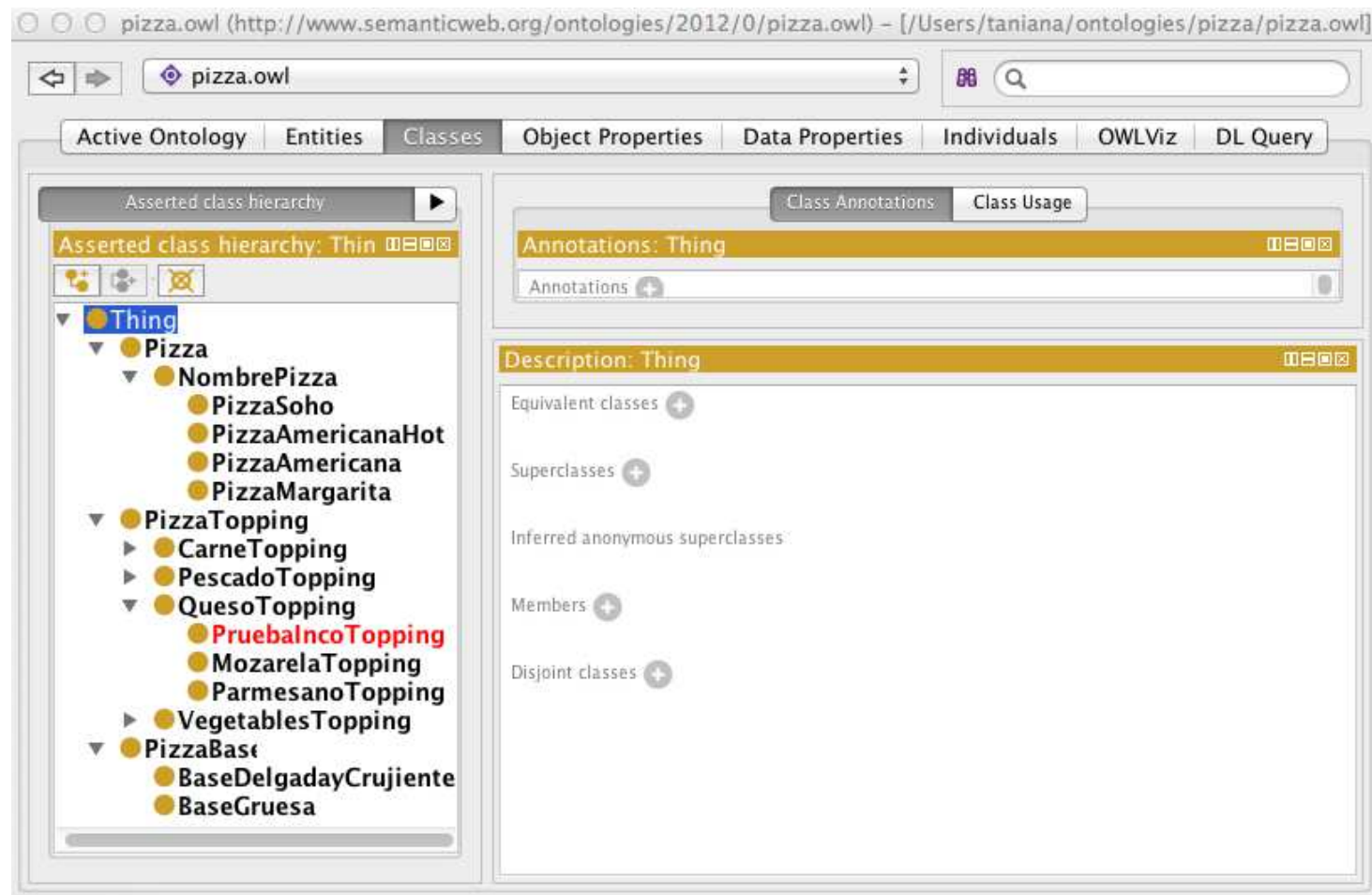
CLASE INCONSISTENTE

Para demostrar el uso de razonador que detecte clases inconsistentes en la ontología, vamos a crear una clase que es subclase de QuesoTopping y también de VegetableTopping

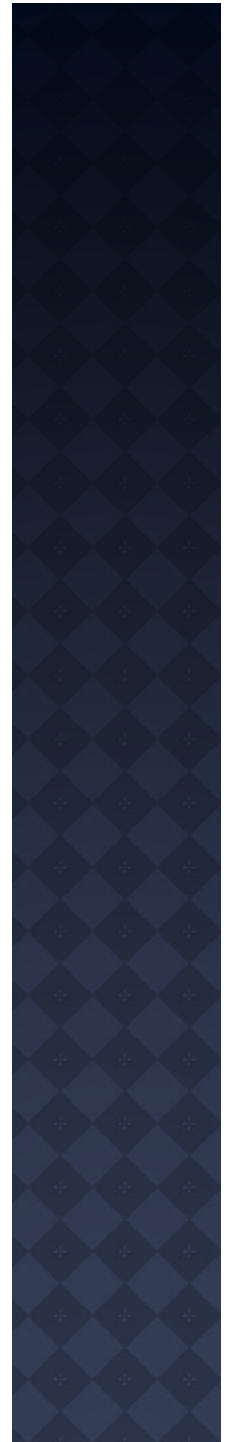


CLASE INCONSISTENTE

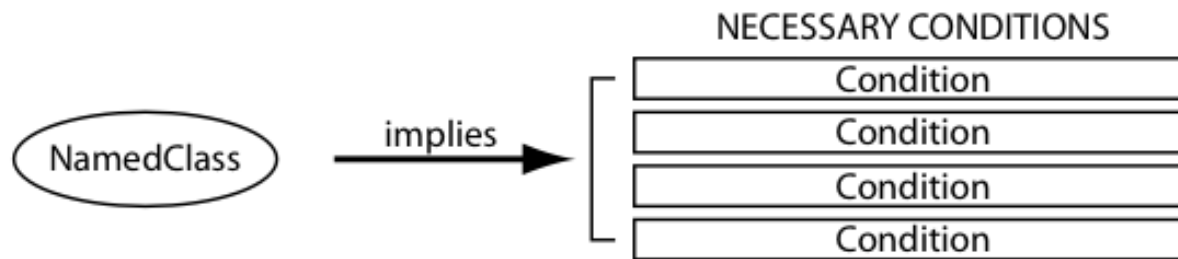
Clasifique utilizando el razonador pellet



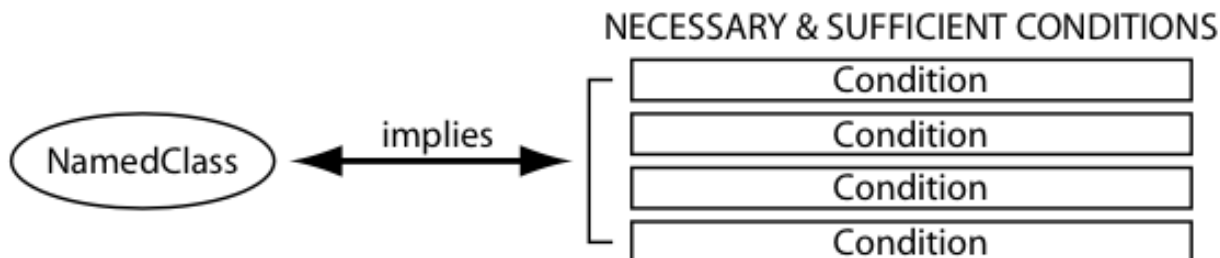
**CONDICIÓN NECESARIA Y
SUFICIENTE**



CONDICIÓN NECESARIA Y SUFICIENTE

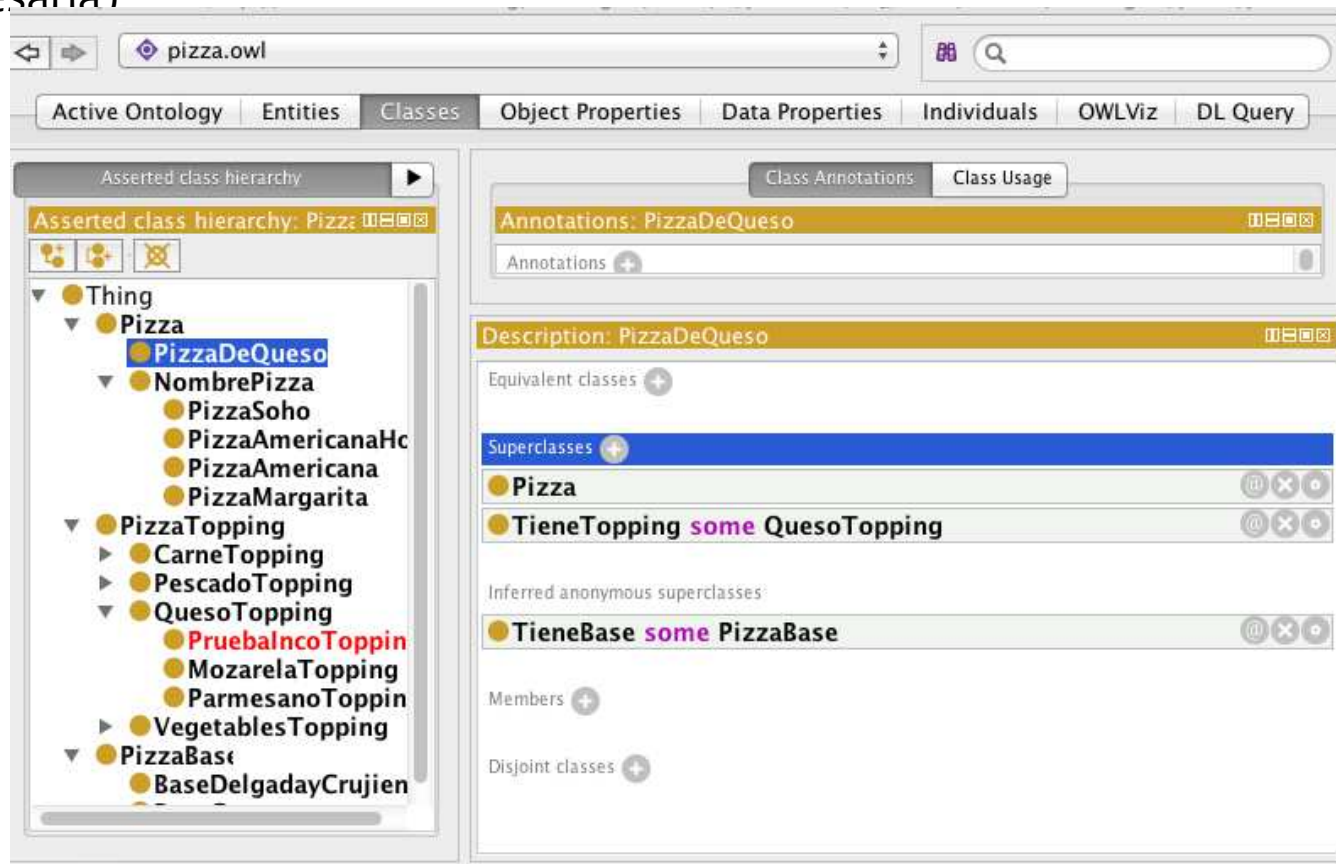


If an individual is a member of 'NamedClass' then it must satisfy the conditions. However if some individual satisfies these necessary conditions, we cannot say that it is a member of 'Named Class' (the conditions are not 'sufficient' to be able to say this) - this is indicated by the direction of the arrow.



CONDICIÓN NECESARIA Y SUFICIENTE

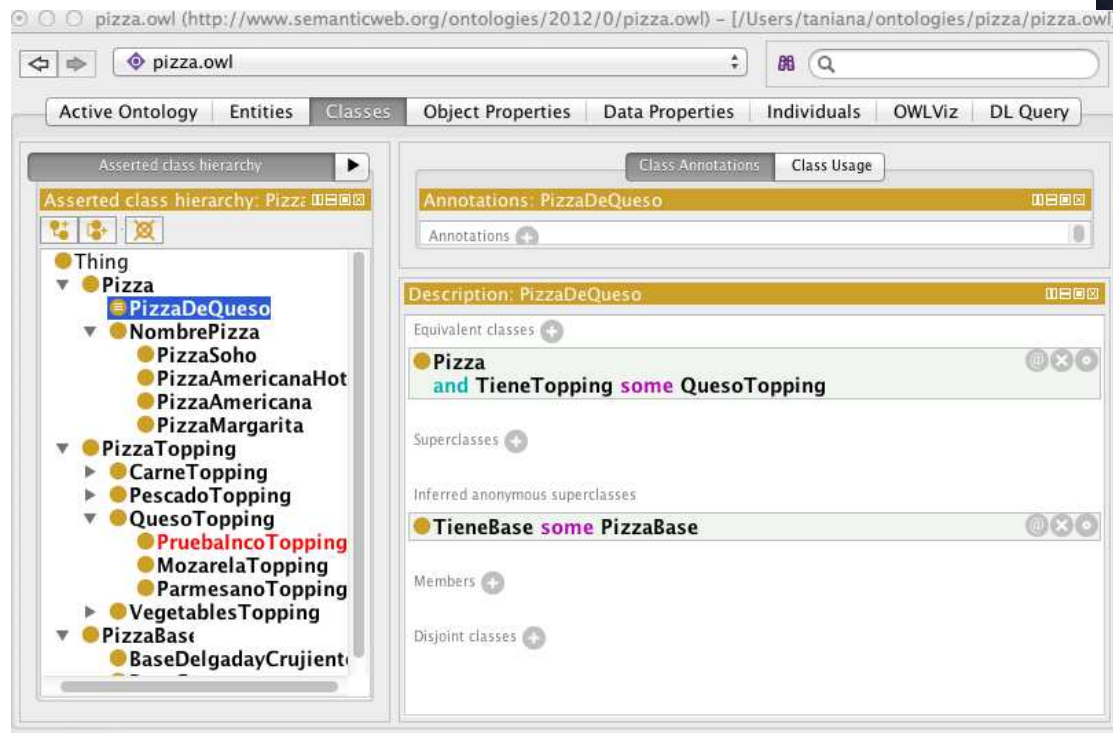
Vamos a crear una subclase de Pizza llamada PizzadeQueso que es una pizza que por lo menos tiene una clases de QuesoTopping (condicion necesaria)



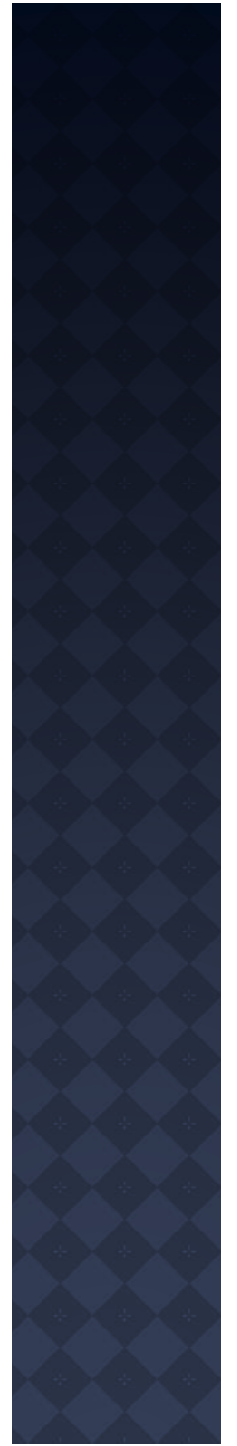
CONDICIÓN NECESARIA Y SUFICIENTE

Vamos crear condición necesaria y suficiente también conocido como Defined Class y en protege clases equivalentes

1. Seleccione PizzaDeQueso.
2. En edit seleccione en el menu “Convert to define class”

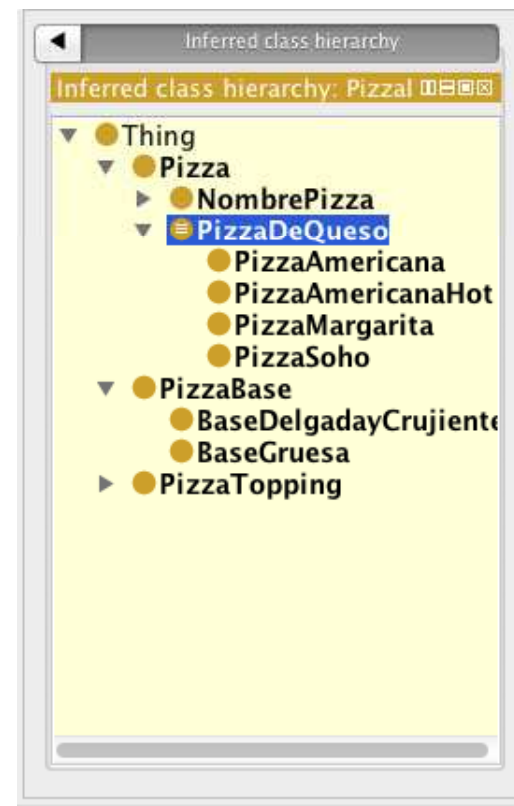
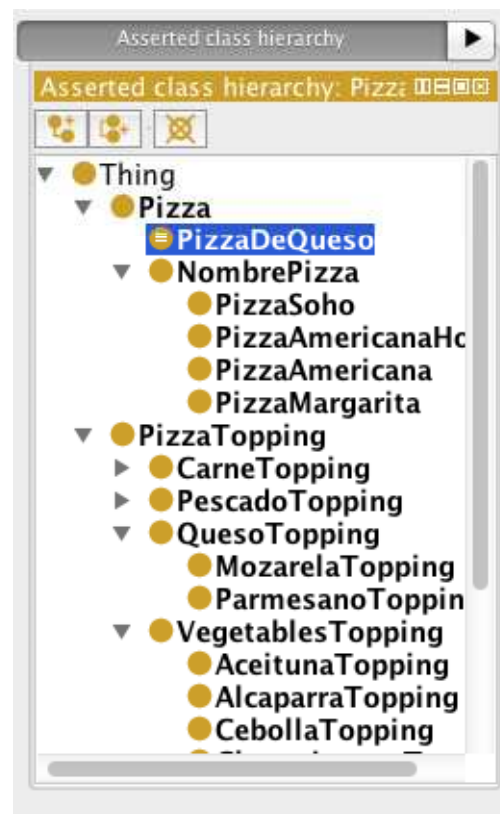
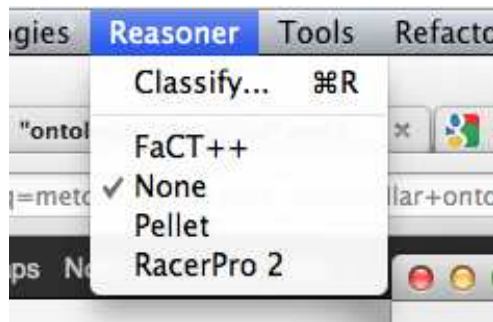


RAZONAMIENTO AUTOMÁTICO

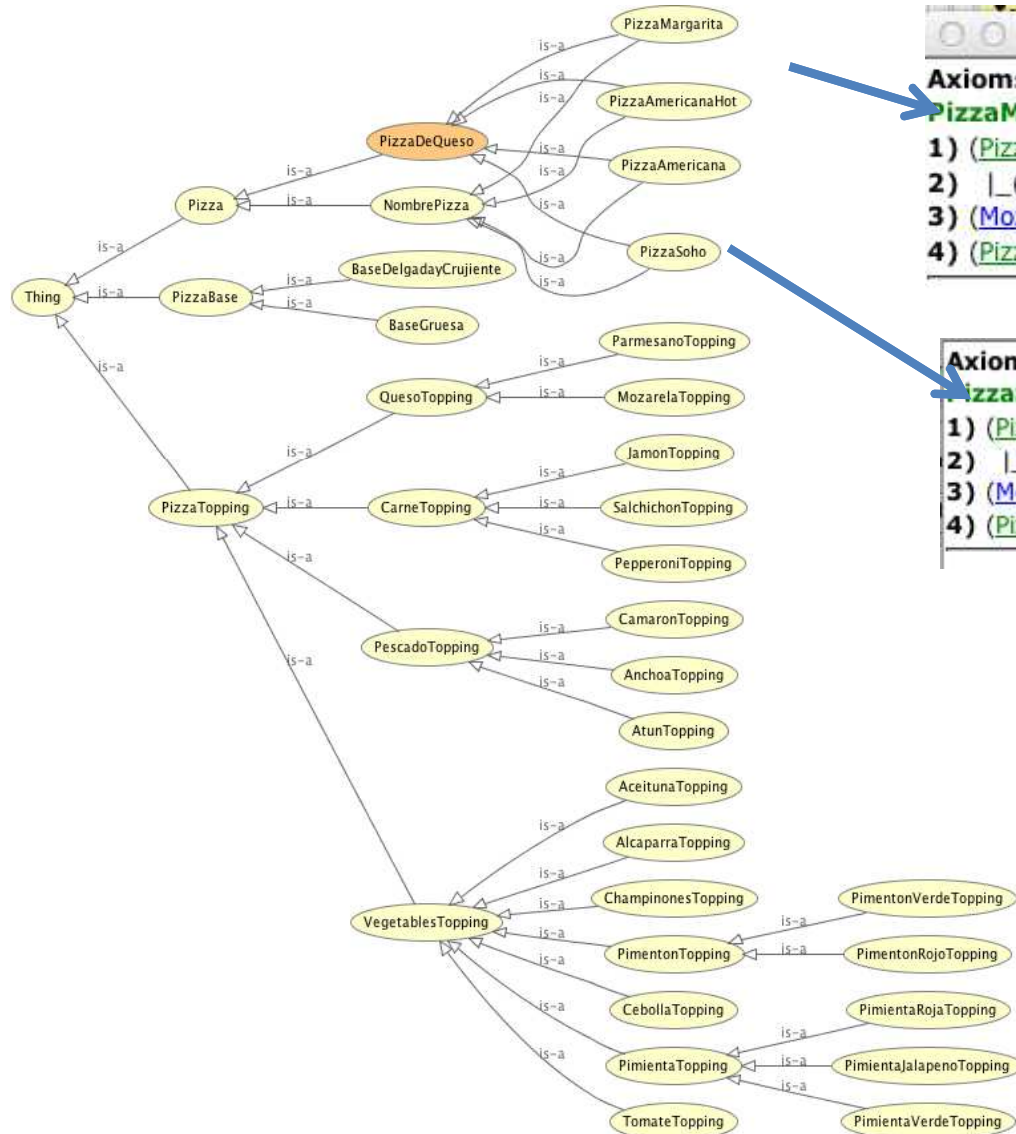


RAZONAMIENTO AUTOMÁTICO

Ya teniendo creada la definicion de una PizzaDeQueso se puede usar el razonador para calcular la subclases PizzaDeQueso



RAZONAMIENTO AUTOMÁTICO



Axioms causing the inference

PizzaMargarita $\subseteq$ PizzaDeQueso:

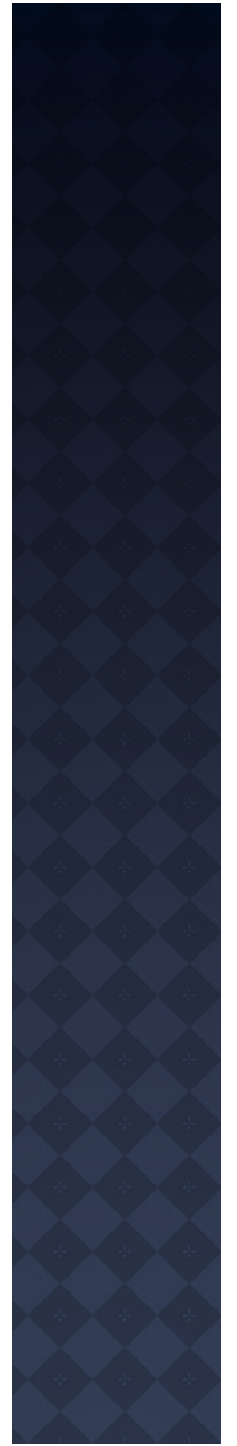
- 1) $(\text{PizzaDeQueso} = ((\exists \text{TieneTopping} . \text{QuesoTopping}) \cap \text{Pizza}))$
- 2) $\neg (\text{TieneTopping} \text{ domain } \text{Pizza})$
- 3) $(\text{MozarelaTopping} \subseteq \text{QuesoTopping})$
- 4) $(\text{PizzaMargarita} \subseteq (\exists \text{TieneTopping} . \text{MozarelaTopping}))$

Axioms causing the inference

PizzaSoho $\subseteq$ PizzaDeQueso:

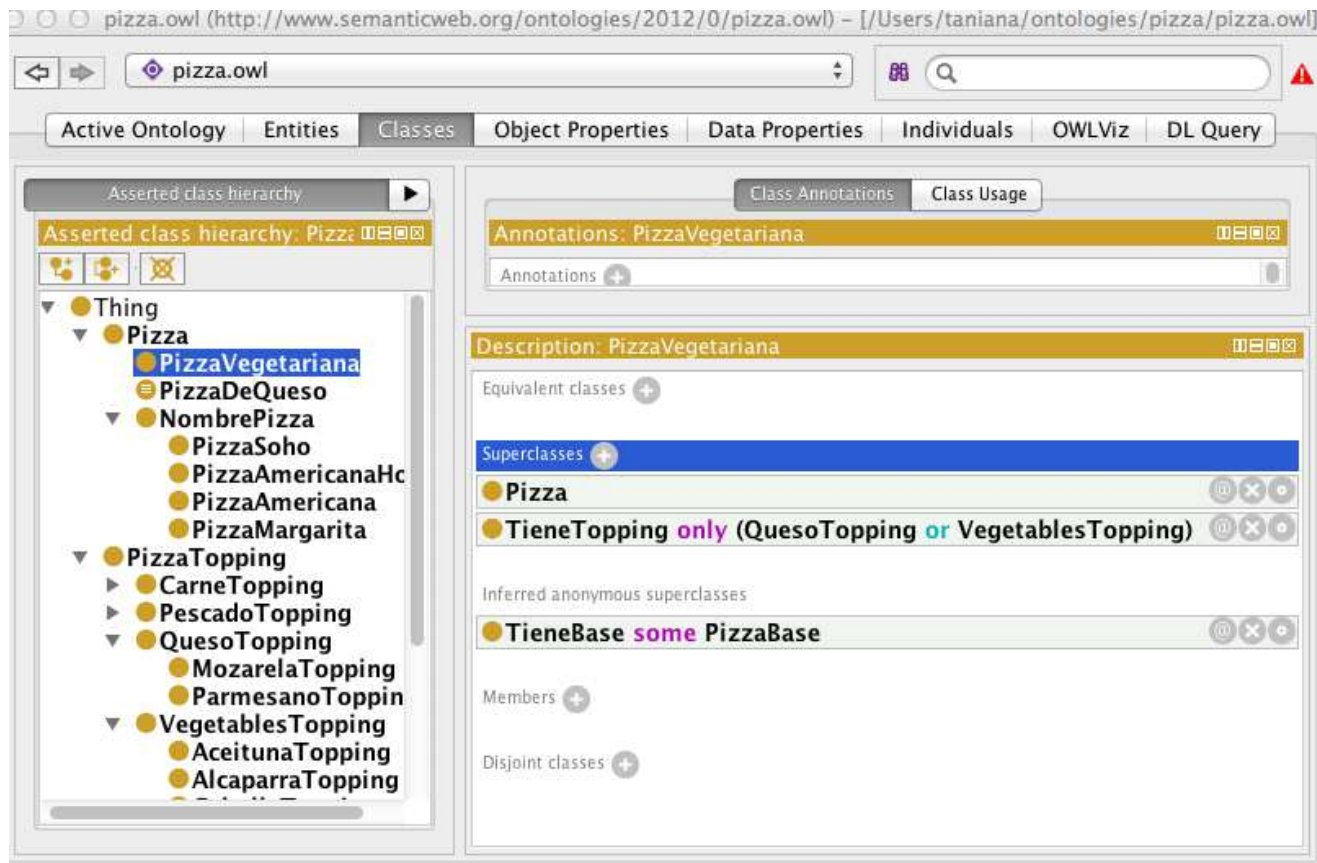
- 1) $(\text{PizzaDeQueso} = ((\exists \text{TieneTopping} . \text{QuesoTopping}) \cap \text{Pizza}))$
- 2) $\neg (\text{TieneTopping} \text{ domain } \text{Pizza})$
- 3) $(\text{MozarelaTopping} \subseteq \text{QuesoTopping})$
- 4) $(\text{PizzaSoho} \subseteq (\exists \text{TieneTopping} . \text{MozarelaTopping}))$

RESTRICCIÓN UNIVERSAL



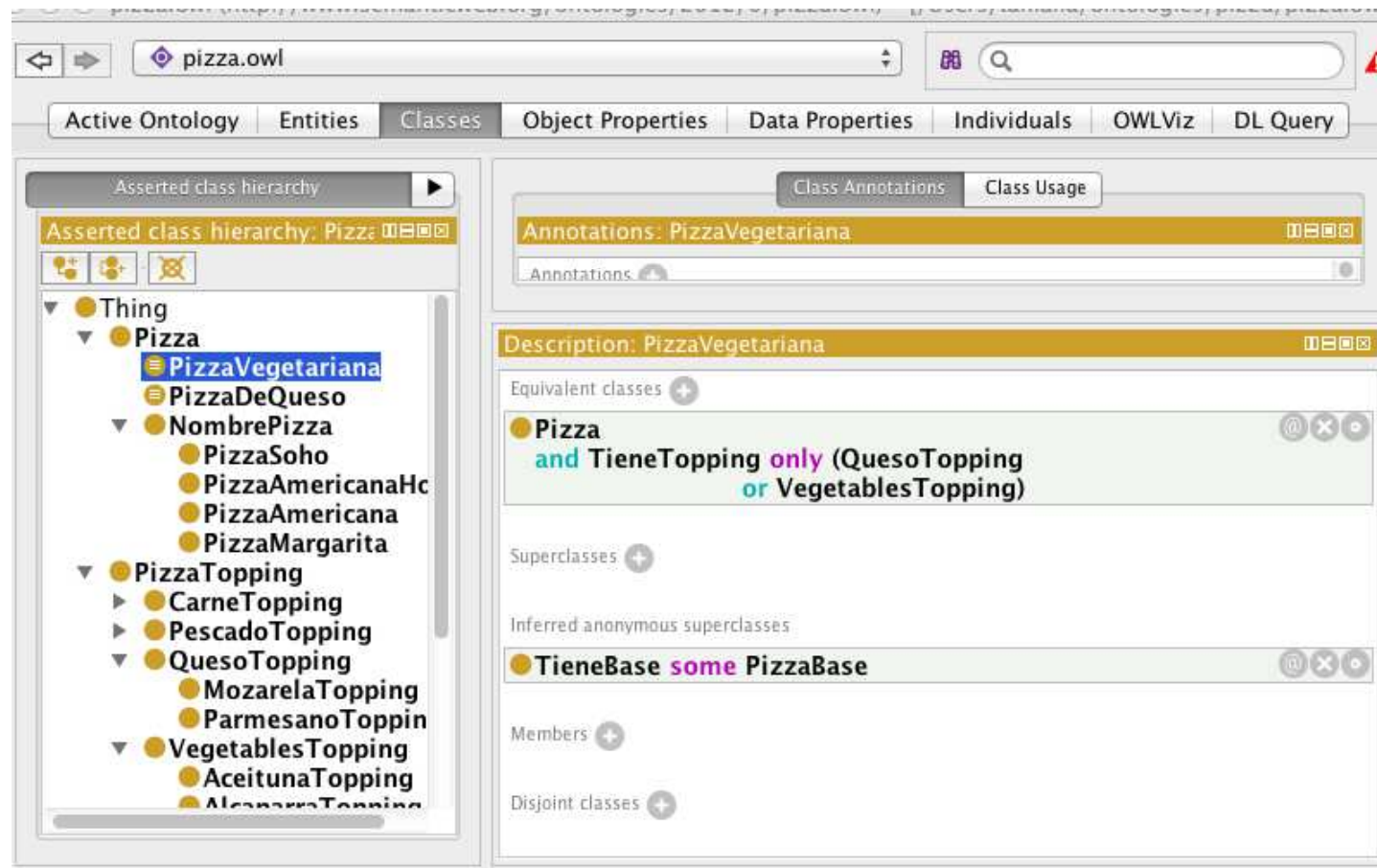
RESTRICCIÓN UNIVERSAL

Ahora vamos a crear una PizzaVegetariana. Los individuos que solo son miembros de esta clases son QuesoTopping o VegetablesTopping. Por lo tanto vamos a usar la restricción universal

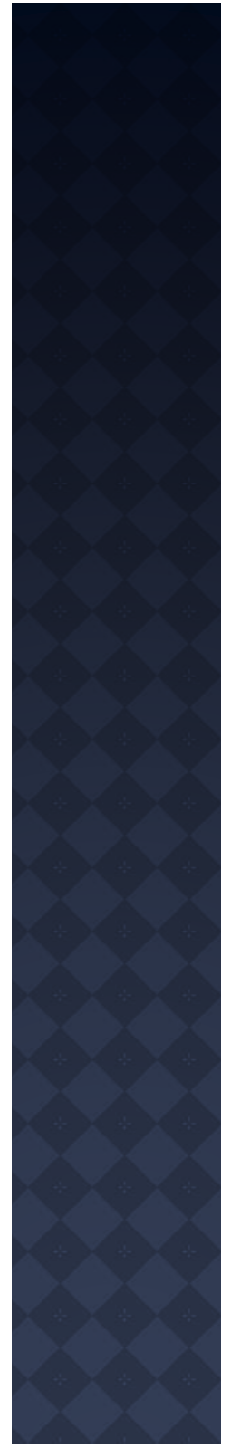


RESTRICCIÓN UNIVERSAL

Ahora vamos a convertirla en necesaria y suficiente

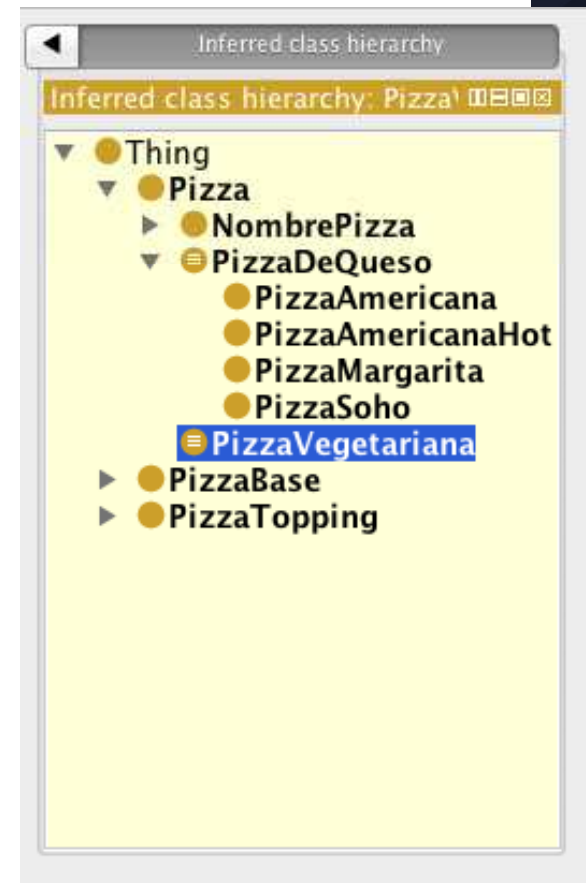
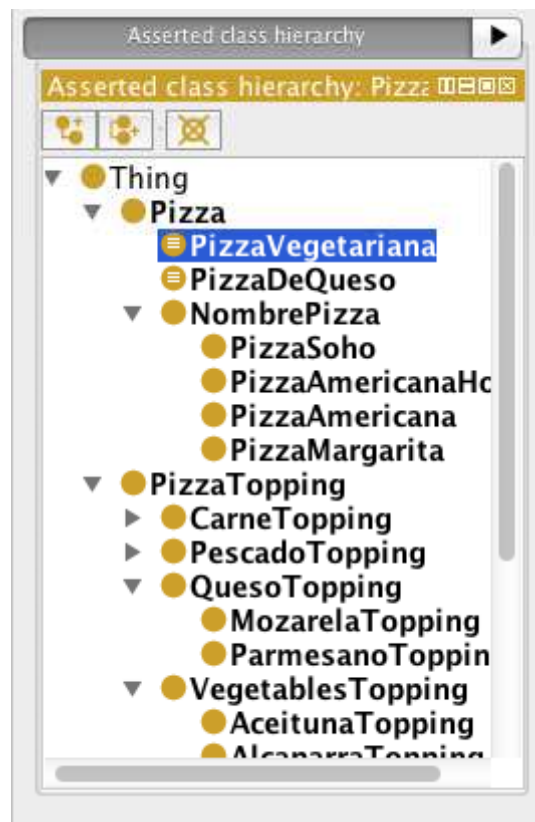
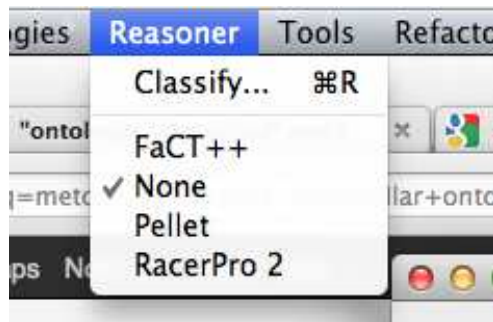


RAZONAMIENTO AUTOMÁTICO

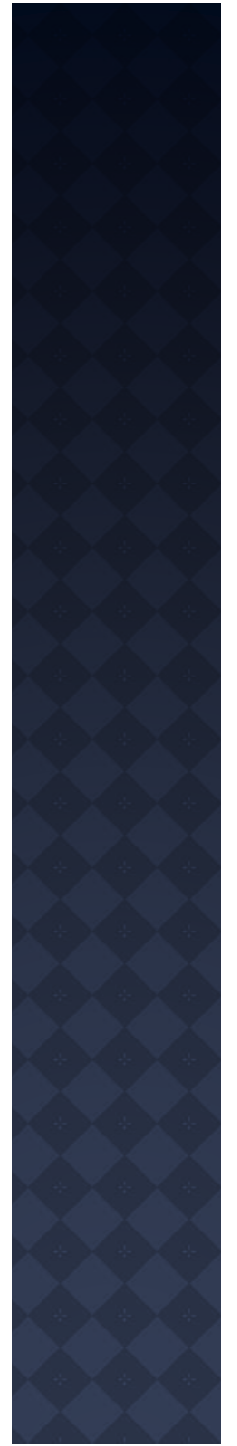


RAZONAMIENTO AUTOMÁTICO

Ya teniendo creada la definición de una PizzaVegetariana Usamos el razonador para ver que pasa



AXIOMAS DE CLAUSURA

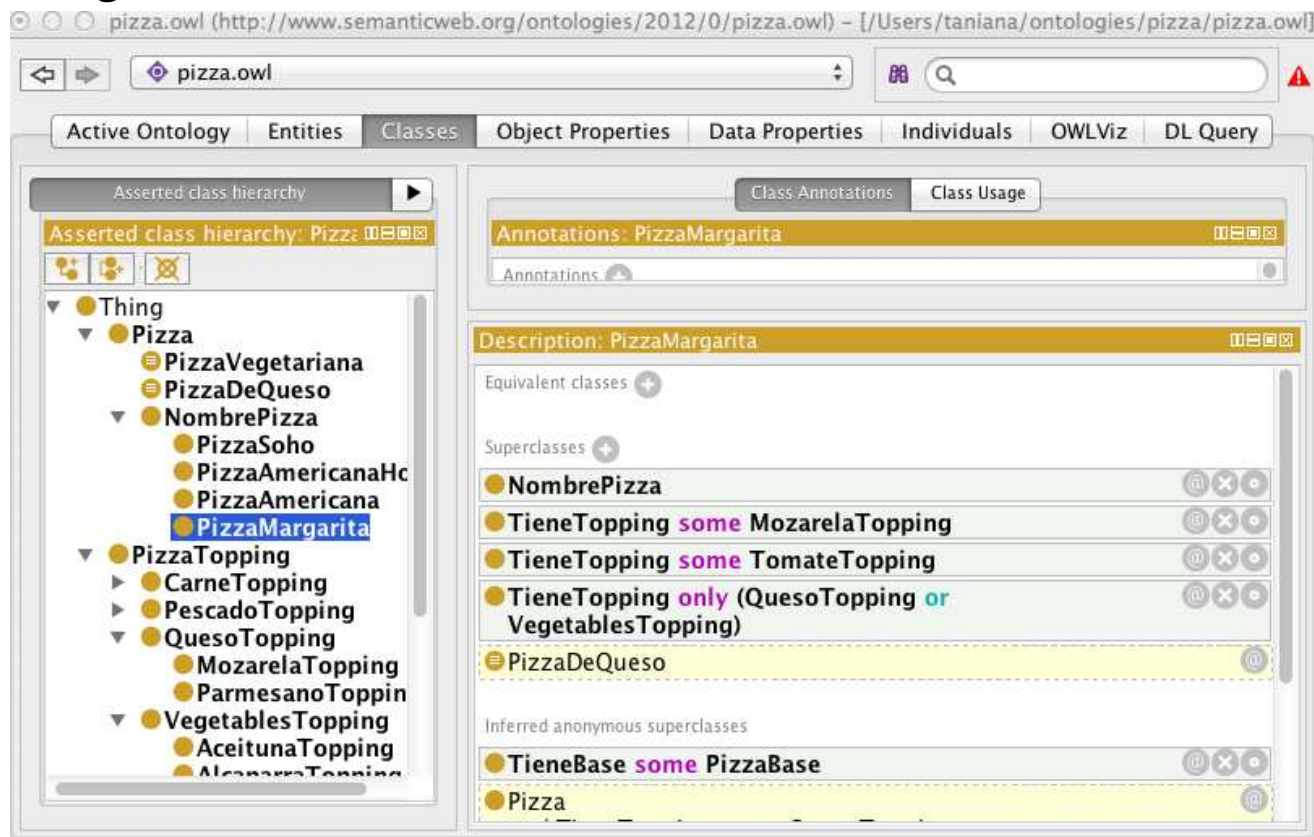


AXIOMA DE CLAUSURA

Un axioma de clausura por ejemplo sobre la propiedad tieneTopping seria

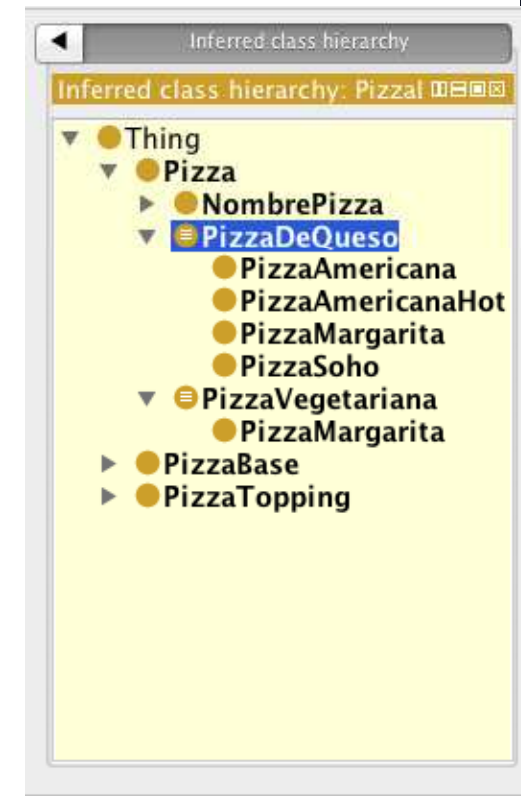
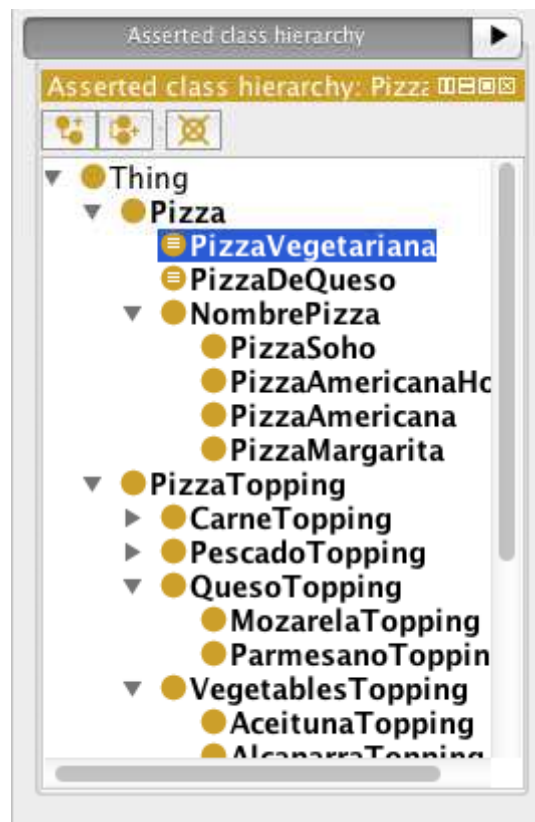
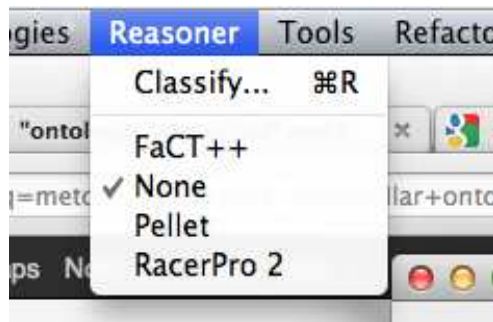
$$\forall \text{tieneTopping}(\text{MozarelaTopping} \cup \text{QuesoTopping})$$

Añadimos el axioma de clausura sobre la propiedad tieneTopping para la PizzaMargarita

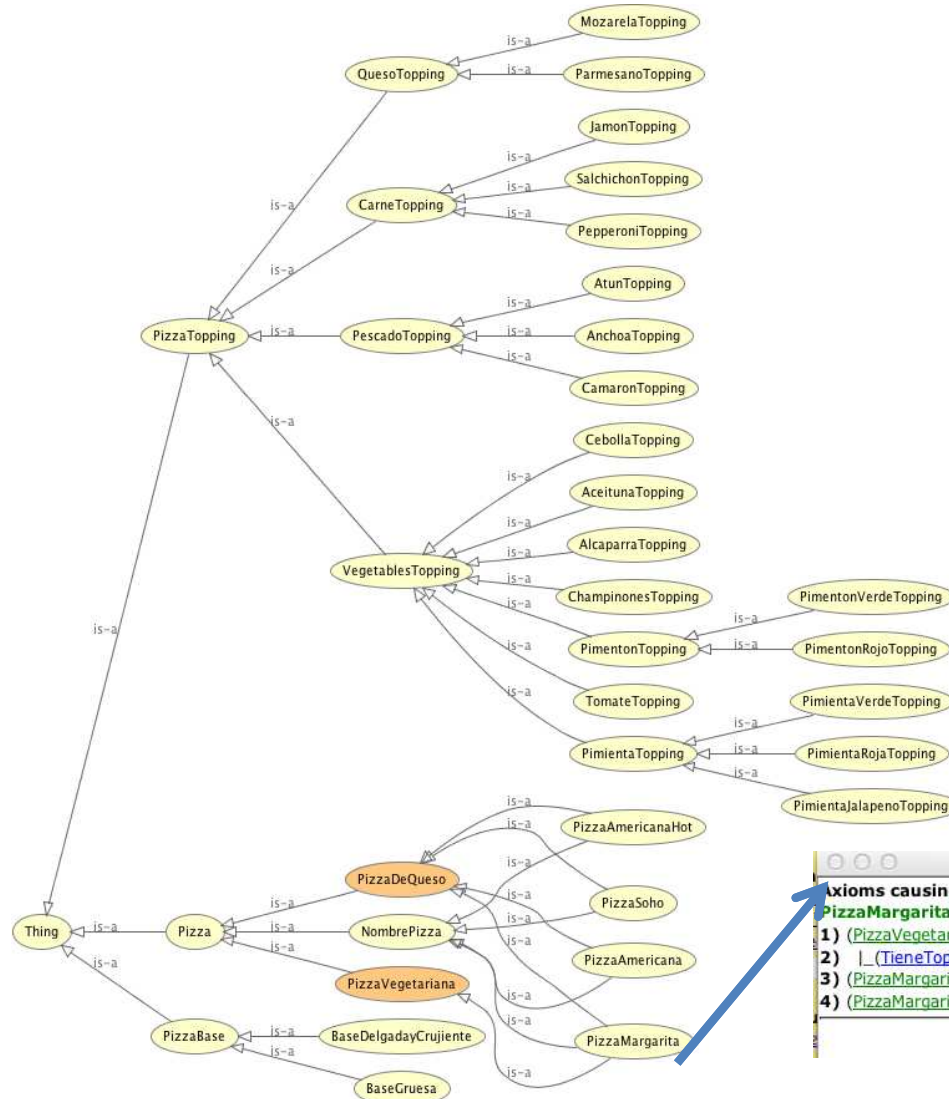


RAZONAMIENTO AUTOMÁTICO

Ya teniendo creada la definición de una PizzaVegetariana Usamos el razonador para ver que pasa



RAZONAMIENTO AUTOMÁTICO



Explanation	
Axioms causing the inference	
PizzaMargarita $\subseteq$ PizzaVegetariana:	
1)	$(\text{PizzaVegetariana} = (\text{Pizza} \cap (\forall \text{TieneTopping} . (\text{QuesoTopping} \cup \text{VegetablesTopping}))))$
2)	$\neg (\text{TieneTopping} \text{ domain Pizza})$
3)	$(\text{PizzaMargarita} \subseteq (\exists \text{TieneTopping} . \text{TomateTopping}))$
4)	$(\text{PizzaMargarita} \subseteq (\forall \text{TieneTopping} . (\text{QuesoTopping} \cup \text{VegetablesTopping})))$

AXIOMA DE CLAUSURA

Vamos hacer lo mismo para PizzaSoho, PizzaAmericanaHot y PizzaAmericana y luego utilizamos el razonador a ver que pasa

The screenshot shows the Protégé ontology editor interface. The left pane displays the 'Asserted class hierarchy' for the 'pizza.owl' ontology. The hierarchy is as follows:

- Thing
 - Pizza
 - PizzaVegetariana
 - PizzaDeQueso
 - NombrePizza
 - PizzaSoho**
 - PizzaAmericanaHot
 - PizzaAmericana
 - PizzaMargarita
 - PizzaTopping
 - CarneTopping
 - PescadoTopping
 - QuesoTopping
 - MozarelaTopping
 - ParmesanoTopping
 - VegetablesTopping
 - AceitunaTopping
 - AlcaparraTopping

The right pane shows the 'Class Annotations' and 'Class Usage' for 'PizzaSoho'. The 'Description: PizzaSoho' section lists the following axioms:

- Equivalent classes: +
- Superclasses: +
- NombrePizza
- TieneTopping some AceitunaTopping
- TieneTopping some MozarelaTopping
- TieneTopping some ParmesanoTopping
- TieneTopping some TomateTopping
- TieneTopping only (AceitunaTopping or MozarelaTopping or ParmesanoTopping or TomateTopping)
- PizzaDeQueso

AXIOMA DE CLAUSURA

Vamos a hacer lo mismo para PizzaSoho, PizzaAmericanaHot y PizzaAmericana y luego utilizamos el razonador a ver que pasa

The screenshot shows the Protégé web interface for the ontology `pizza.owl` (http://www.semanticweb.org/ontologies/2012/0/pizza.owl). The interface is divided into several panes:

- Active Ontology**: Shows the current ontology being viewed.
- Entities**: A tab for viewing entities.
- Classes**: The active tab, showing the class hierarchy.
- Object Properties**: A tab for viewing object properties.
- Data Properties**: A tab for viewing data properties.
- Individuals**: A tab for viewing individuals.
- OWL Viz**: A tab for visualizing the ontology.
- DL Query**: A tab for entering DL queries.

The **Classes** pane is further divided into **Asserted class hierarchy** and **Inferred class hierarchy**. The **Asserted class hierarchy** shows the following structure:

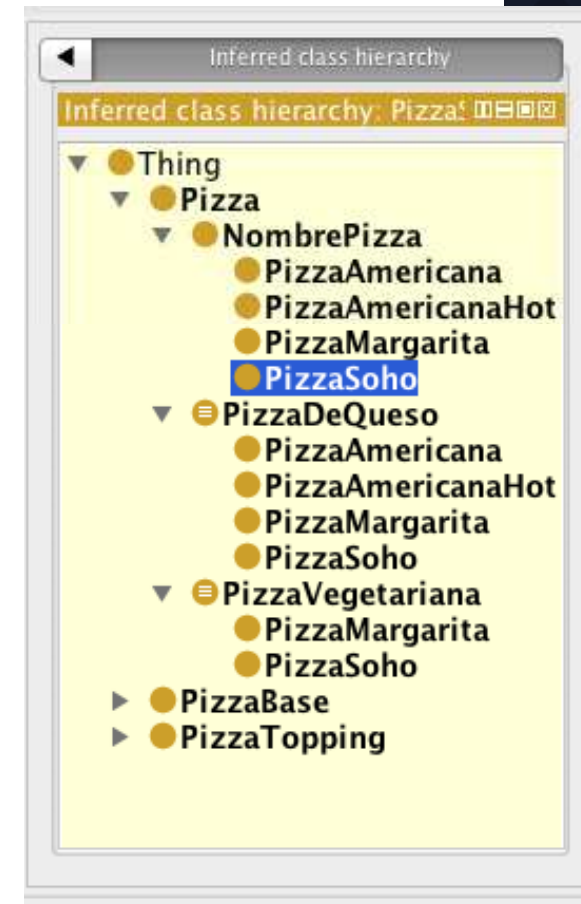
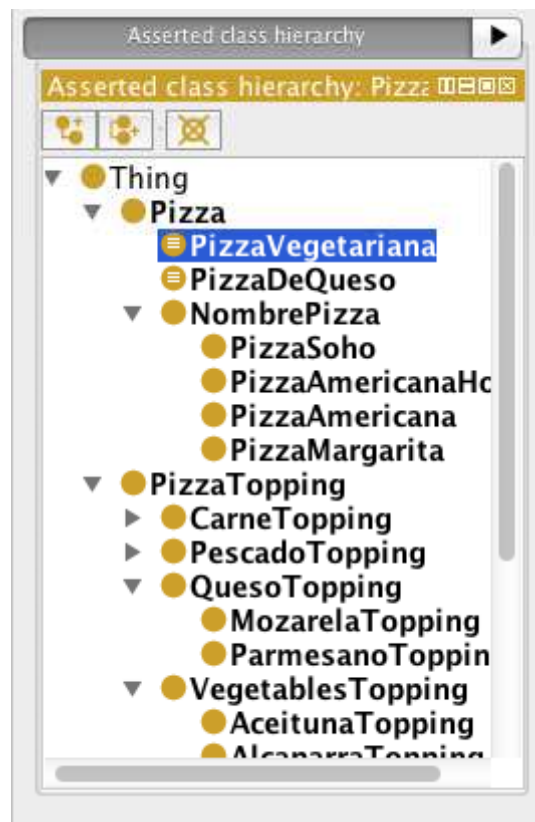
- Thing
 - Pizza
 - NombrePizza
 - PizzaAmericana
 - PizzaAmericanaHot** (highlighted)
 - PizzaMargarita
 - PizzaSoho
 - PizzaCaloriasAltas
 - PizzaCaloriasBaja
 - PizzaDeQueso
 - PizzaInteresante
 - PizzaVegetariana
 - PizzaBase
 - BaseDelgadayCrujiente
 - BaseGruesa
 - PizzaTopping

The **Inferred class hierarchy** pane shows the following information for **PizzaAmericanaHot**:

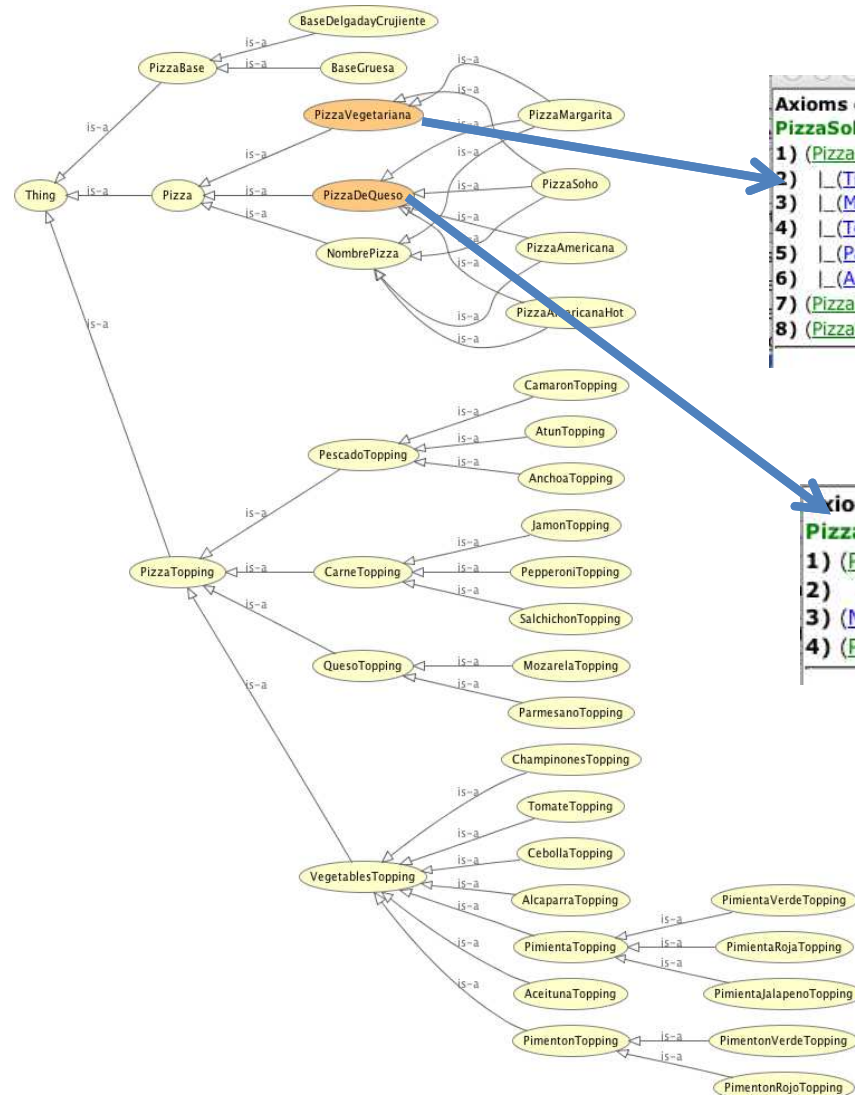
- Annotations**: A section for class annotations.
- Description**: A section for the class description.
- Equivalent classes**: A list of equivalent classes.
- Superclasses**: A list of superclasses, including:
 - NombrePizza
 - TieneTopping **some** MozzarellaTopping
 - TieneTopping **some** PepperoniTopping
 - TieneTopping **some** PimientaJalapenoTopping
 - TieneTopping **some** TomateTopping
 - TieneTopping **only** (MozarellaTopping or PepperoniTopping or PimientaTopping or TomateTopping)
- Inferred anonymous superclasses**: A list of inferred anonymous superclasses, including:
 - TieneCalorias **some** integer
 - TieneBase **some** PizzaBase

RAZONAMIENTO AUTOMÁTICO

Clasificamos nuevamente



RAZONAMIENTO AUTOMÁTICO



Axioms causing the inference

PizzaSoho $\subseteq$ PizzaVegetariana:

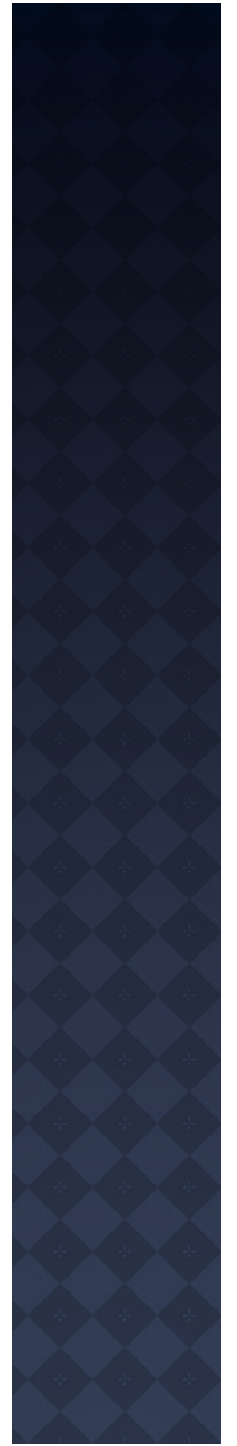
- 1) $(\text{PizzaSoho} \subseteq (\forall \text{TieneTopping} . (\text{MozarelaTopping} \cup \text{TomateTopping} \cup \text{ParmesanoTopping} \cup \text{AceitunaTopping})))$
- 2) $\perp (\text{TieneTopping} \text{ domain } \text{Pizza})$
- 3) $\perp (\text{MozarelaTopping} \subseteq \text{QuesoTopping})$
- 4) $\perp (\text{TomateTopping} \subseteq \text{VegetablesTopping})$
- 5) $\perp (\text{ParmesanoTopping} \subseteq \text{QuesoTopping})$
- 6) $\perp (\text{AceitunaTopping} \subseteq \text{VegetablesTopping})$
- 7) $(\text{PizzaSoho} \subseteq (\exists \text{TieneTopping} . \text{AceitunaTopping}))$
- 8) $(\text{PizzaVegetariana} = (\text{Pizza} \cap (\forall \text{TieneTopping} . (\text{QuesoTopping} \cup \text{VegetablesTopping}))))$

Axioms causing the inference

PizzaSoho $\subseteq$ PizzaDeQueso:

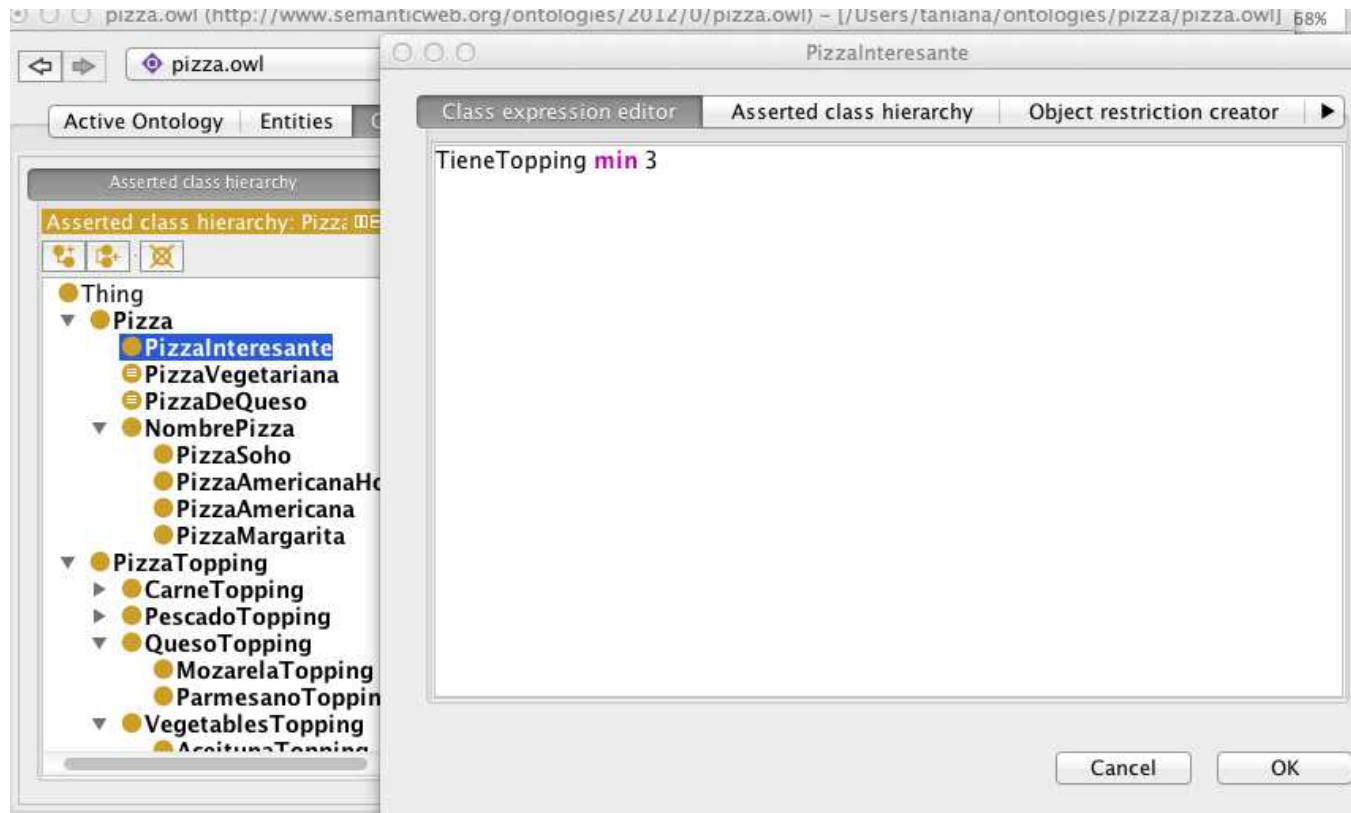
- 1) $(\text{PizzaDeQueso} = ((\exists \text{TieneTopping} . \text{QuesoTopping}) \cap \text{Pizza}))$
- 2) $\perp (\text{TieneTopping} \text{ domain } \text{Pizza})$
- 3) $(\text{MozarelaTopping} \subseteq \text{QuesoTopping})$
- 4) $(\text{PizzaSoho} \subseteq (\exists \text{TieneTopping} . \text{MozarelaTopping}))$

RESTRICCIÓN DE CARDINALIDAD



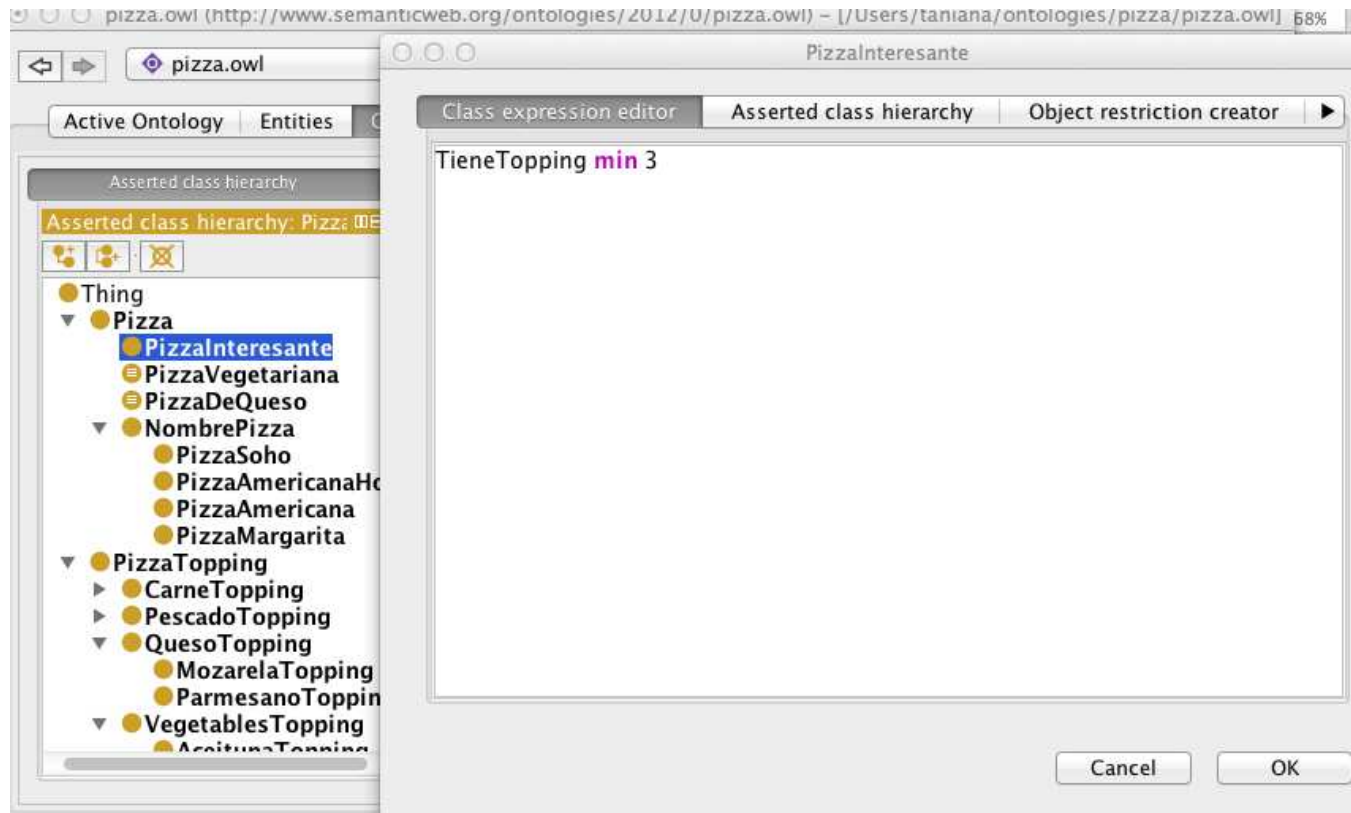
RESTRICCIÓN DE CARDINALIDAD

Vamos añadir una restricción de cardinalidad, creamos una subclases de Pizza llamada pizza interesante, donde vamos a definir que tiene 3 o mas toppings



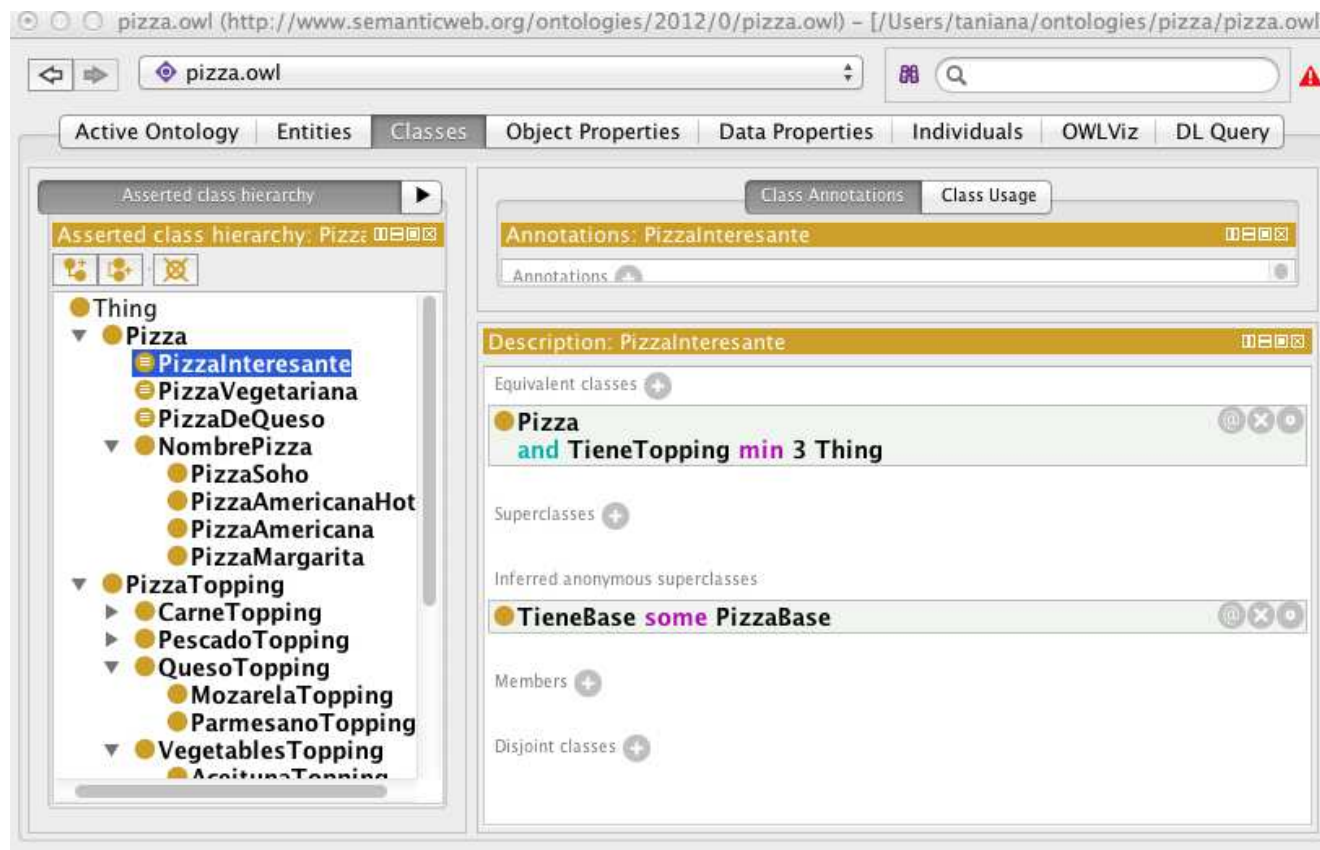
RESTRICCIÓN DE CARDINALIDAD

Vamos añadir una restricción de cardinalidad, creamos una subclases de Pizza llamada pizza interesante, donde vamos a definir que tiene 3 o mas toppings y seleccione Ok.



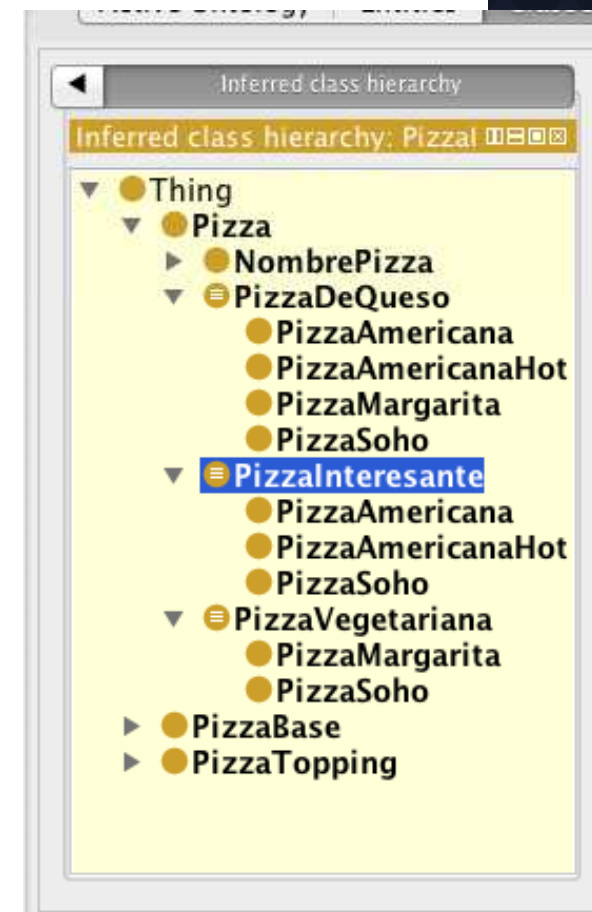
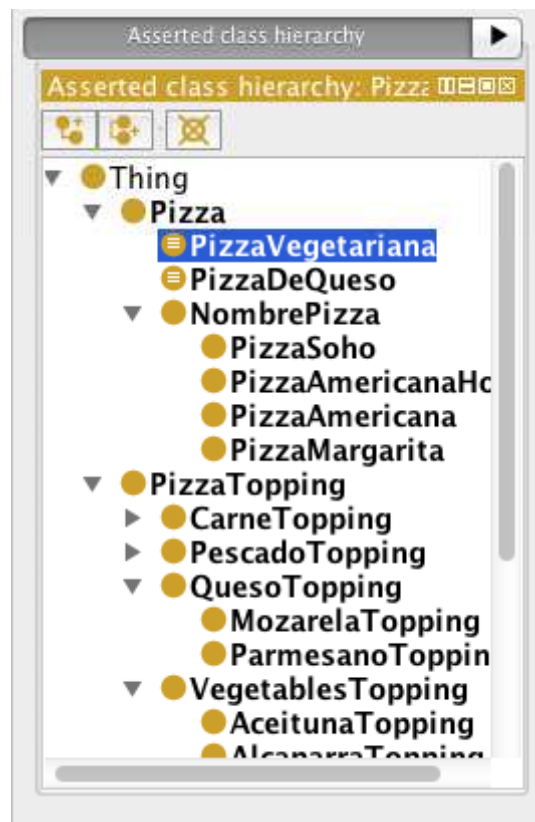
RESTRICCIÓN DE CARDINALIDAD

Luego seleccione “Convert to defined class” y la superclase debe quedar sin nada y se debe cambiar a la clase equivalente



RAZONAMIENTO AUTOMÁTICO

Clasificamos nuevamente

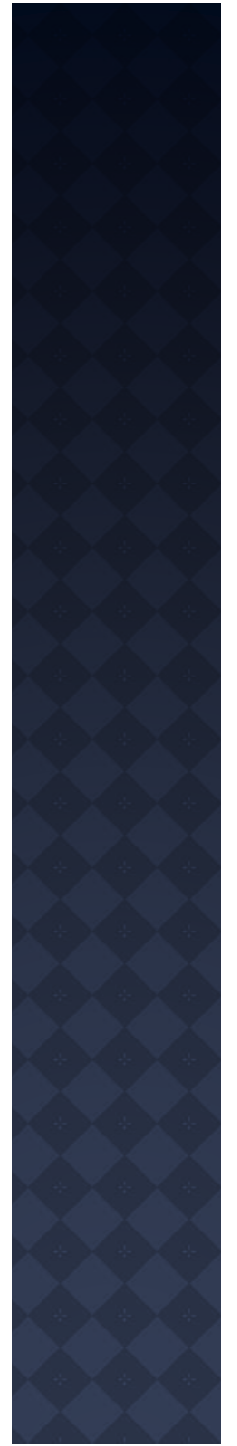


RAZONAMIENTO AUTOMÁTICO



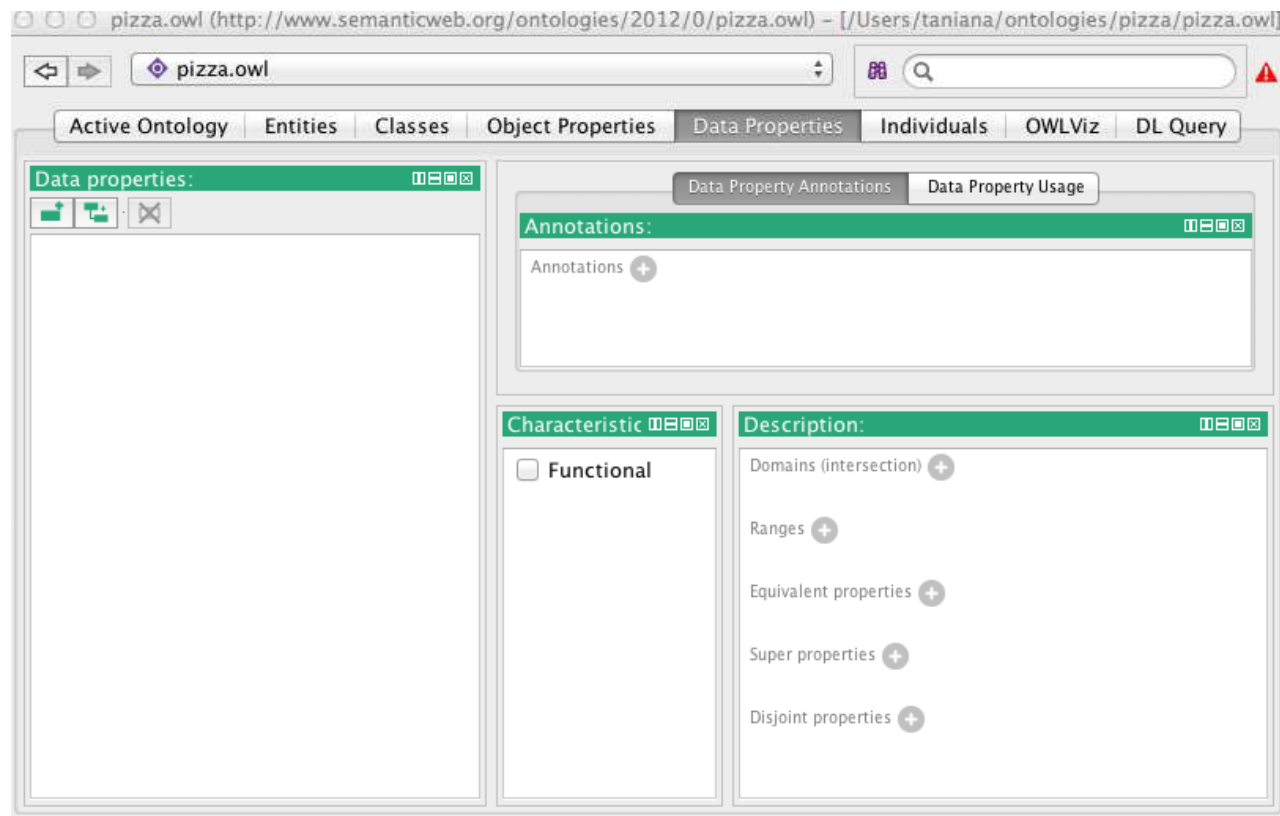
Axioms causing the inference
PizzaInteresante $\subseteq$ **Pizza**:
1) (**PizzaInteresante** = ((≥ 3 TieneTopping) $\cap$ **Pizza**))

PROPIEDADES DATATYPE



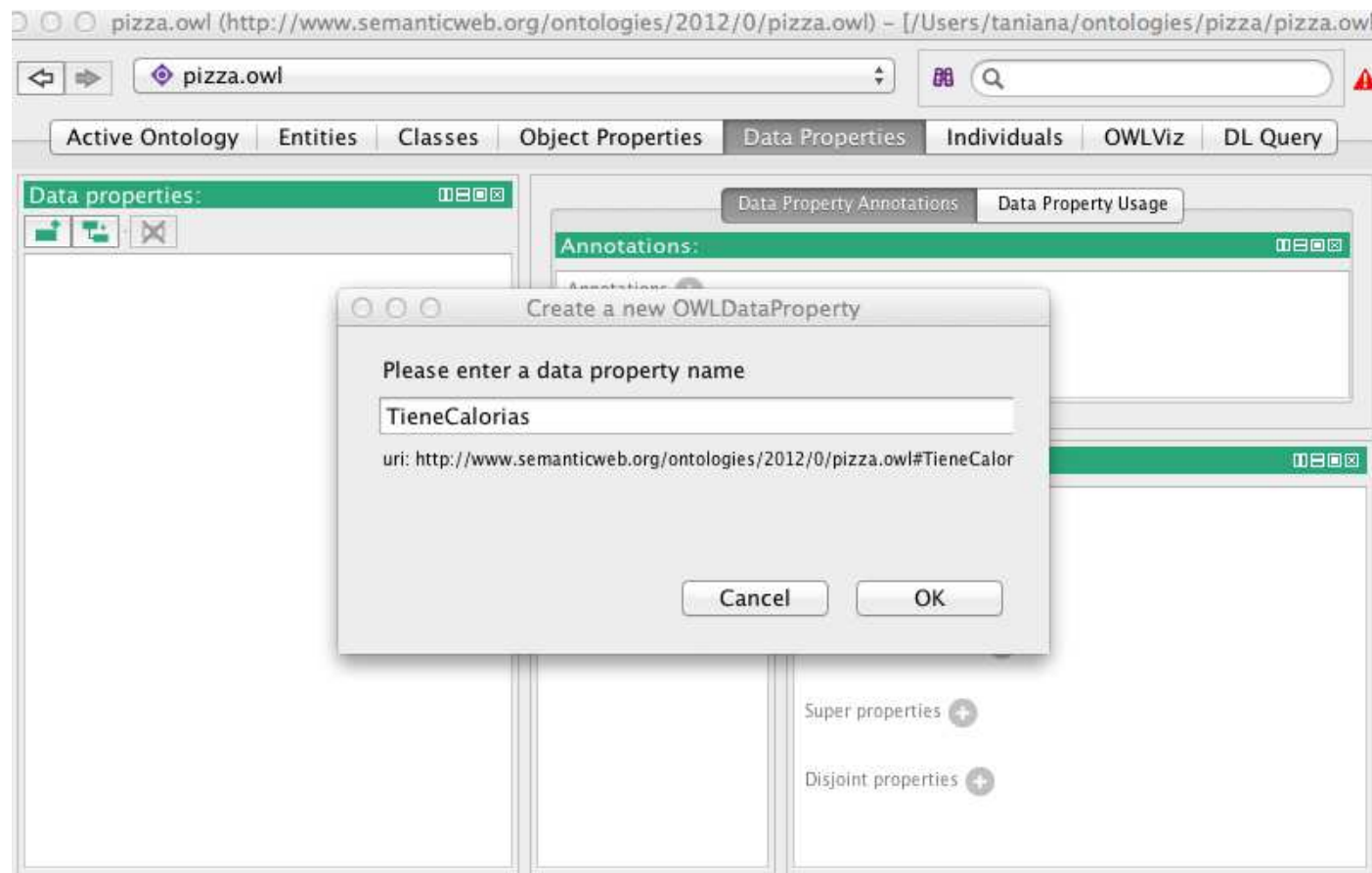
PROPIEDADES DATATYPE

- ◉ Usaremos las propiedad Datatype para describir las calorías de las pizzas



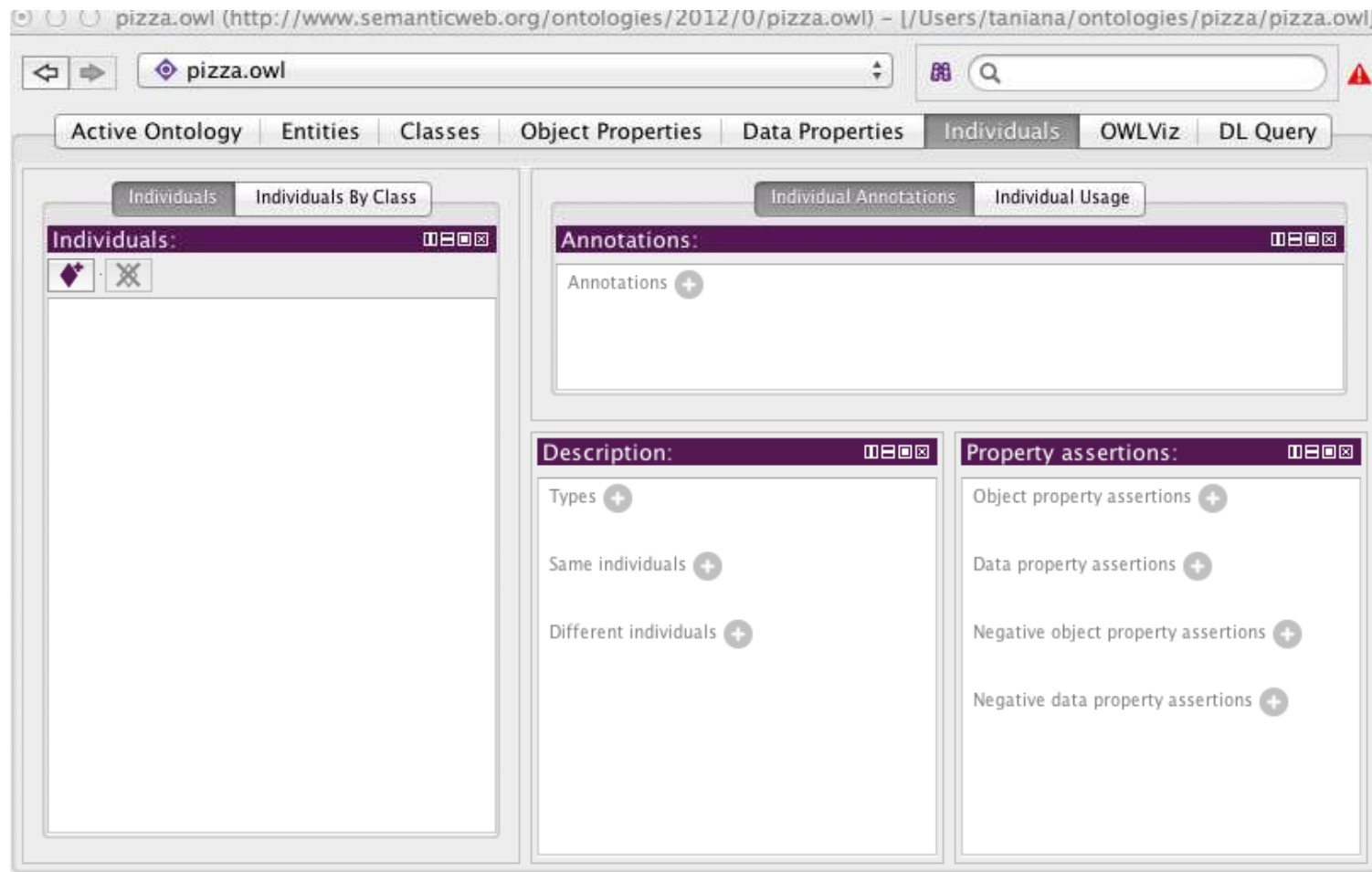
PROPIEDADES DATATYPE

- ◉ Creamos un datatype llamado tieneCalorias



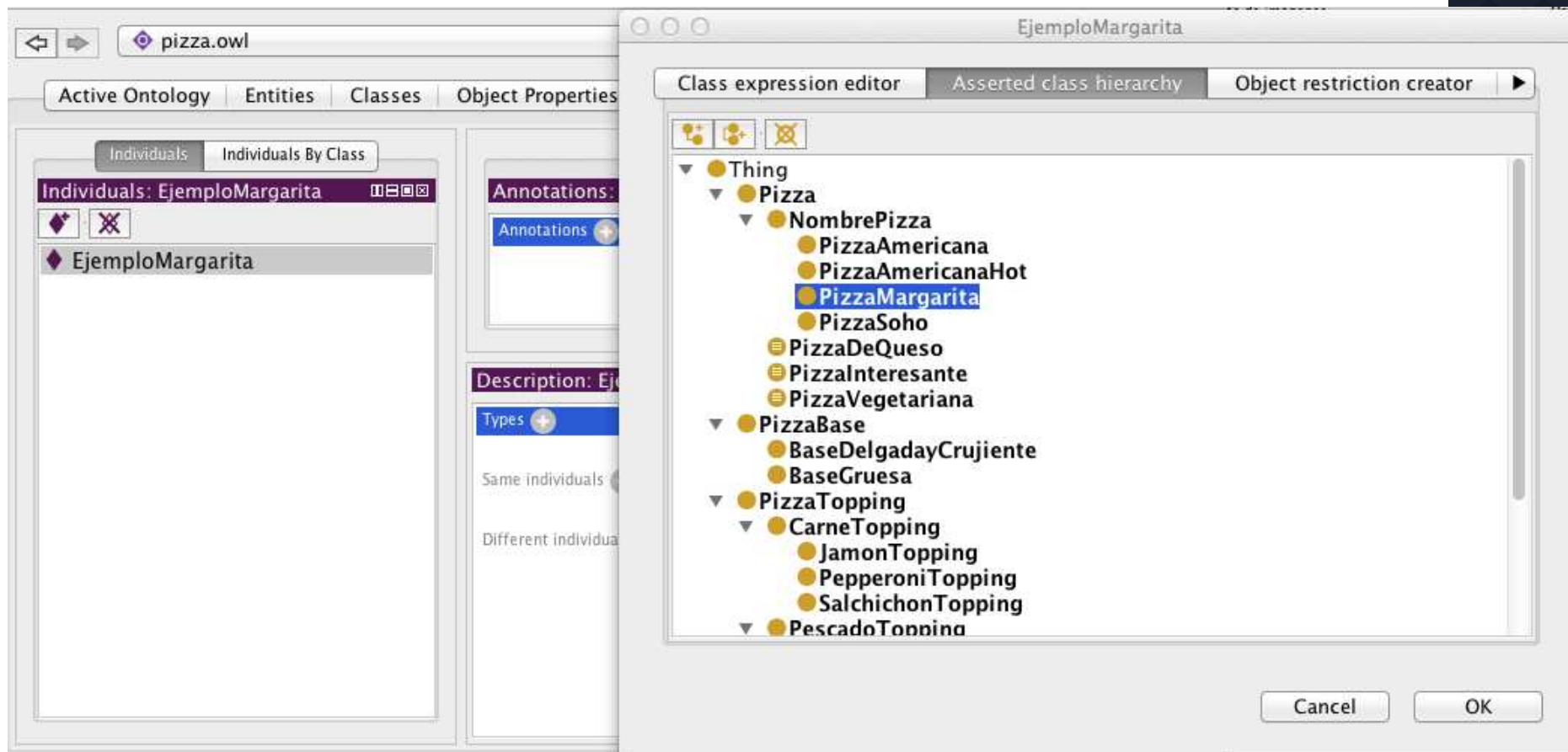
PROPIEDADES DATATYPE

- Crear un ejemplo de individuo de pizza



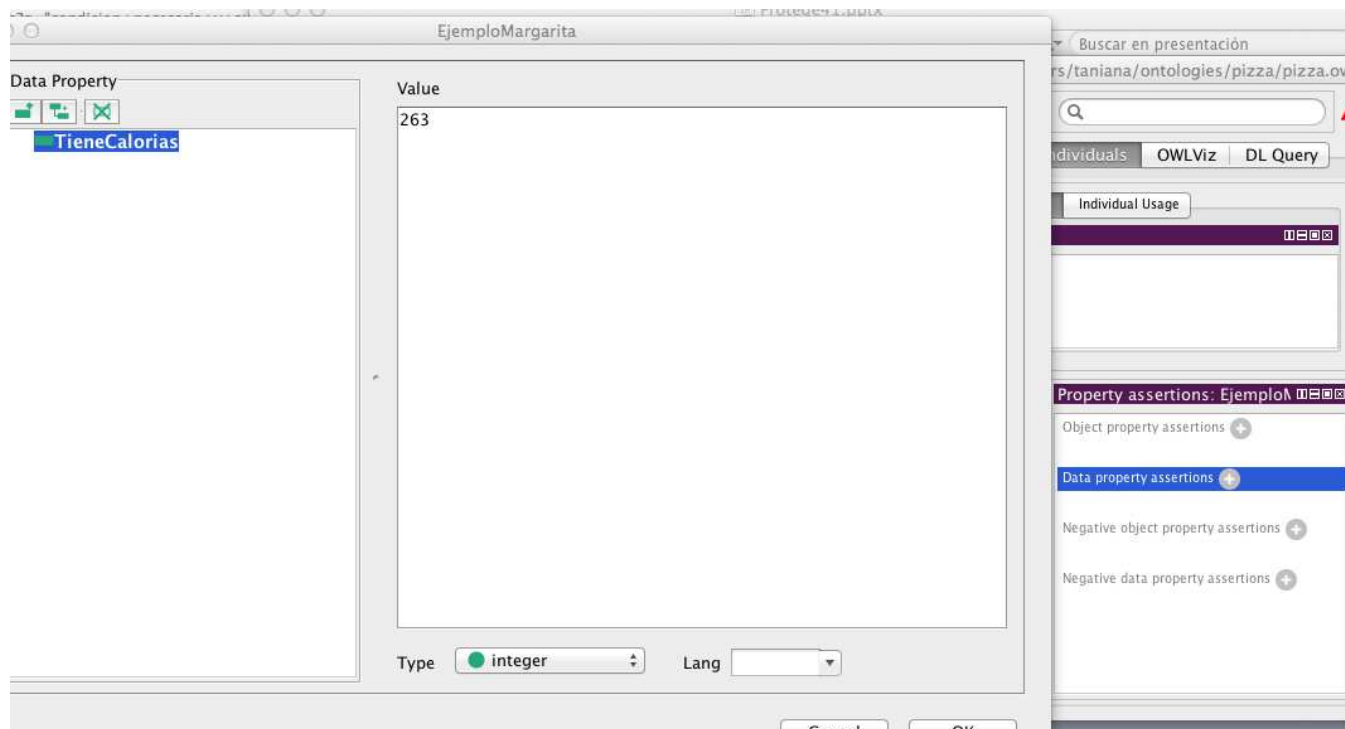
PROPIEDADES DATATYPE

- En la ventana Individual Description añade el tipo de PizzaMargarita y seleccione ok



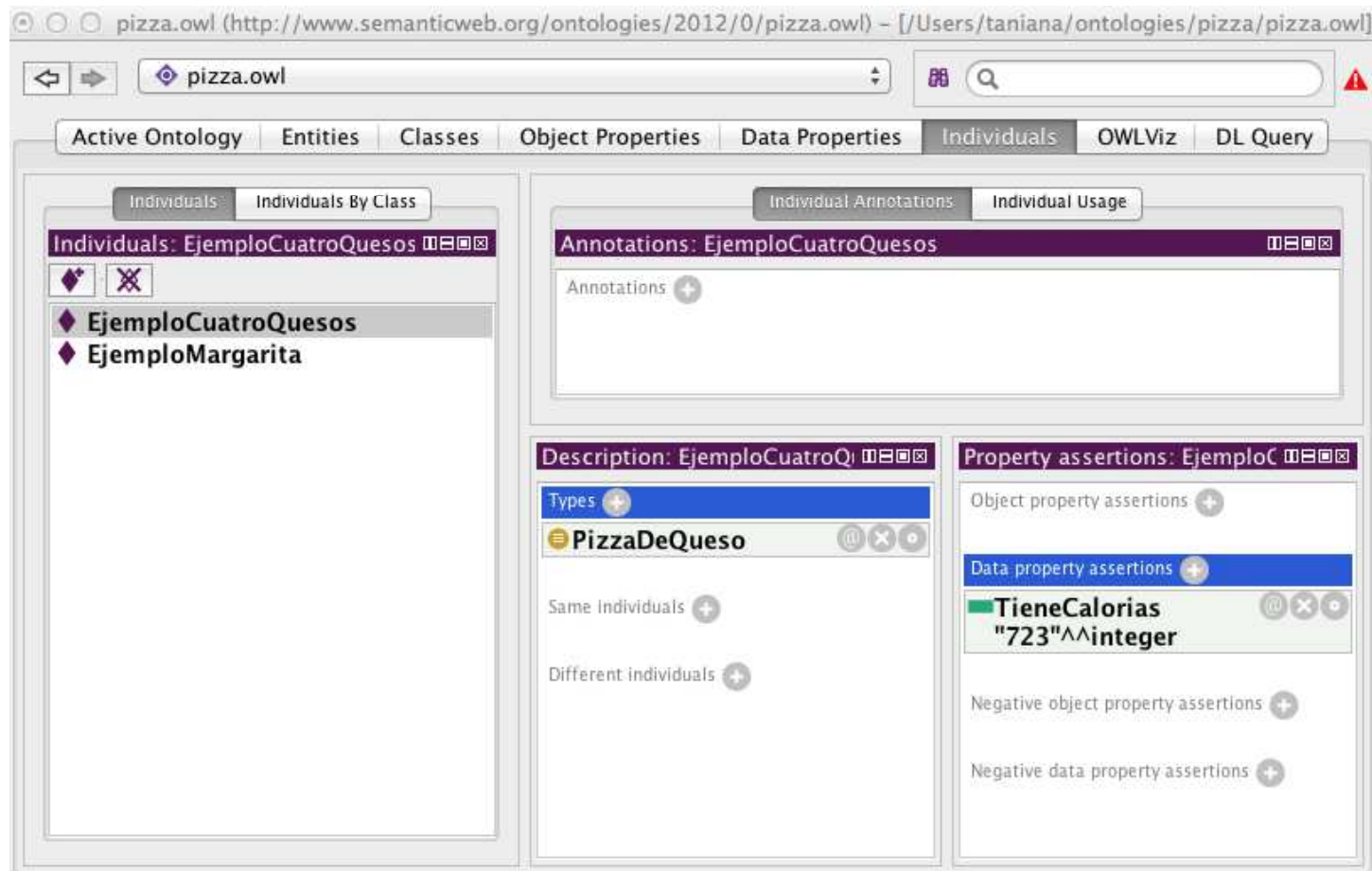
PROPIEDADES DATATYPE

- En la ventana Property assertions y añade en Data Property assertion seleccione TieneCalorias y defínalo entero y valor 263



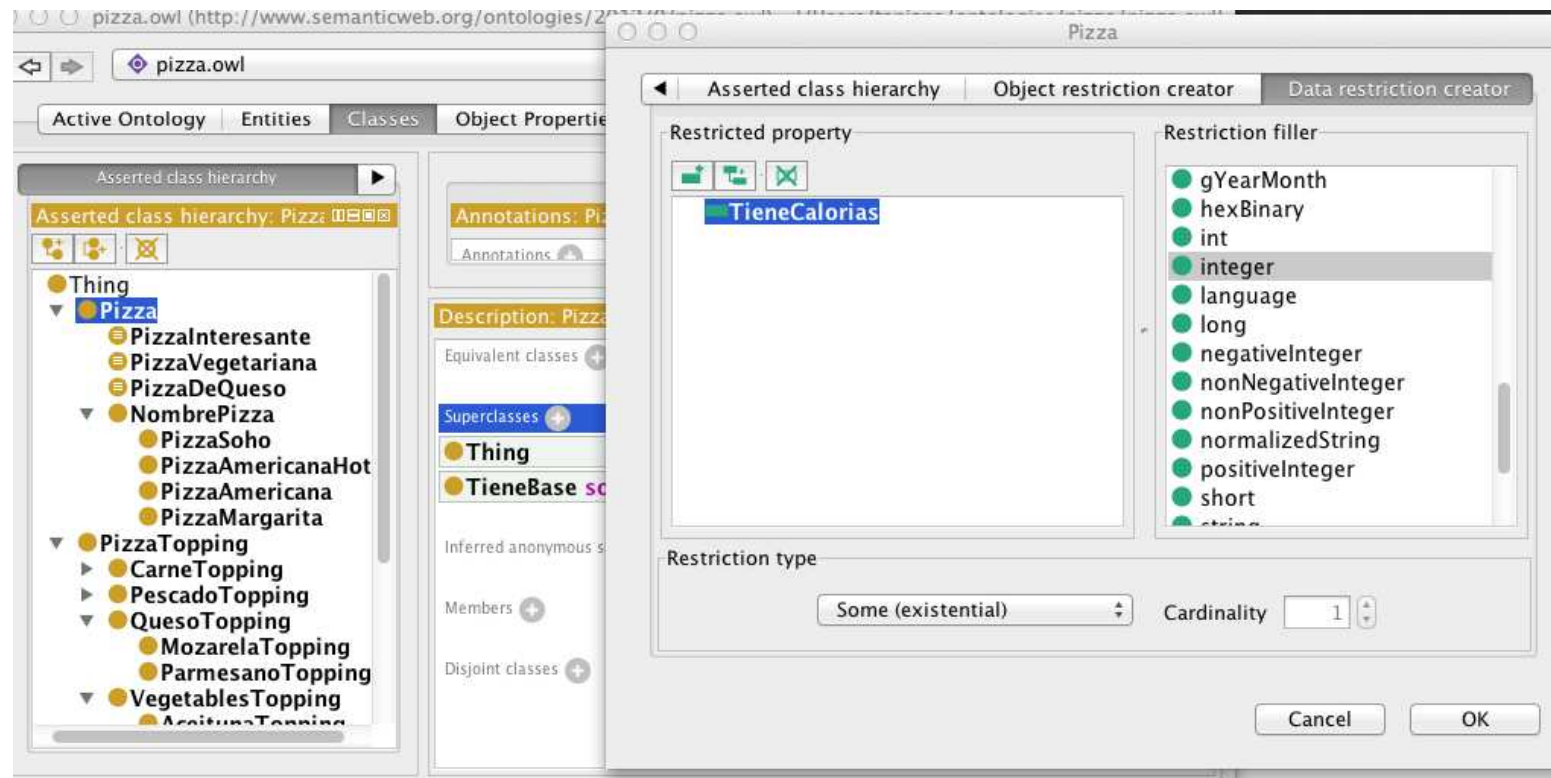
PROPIEDADES DATATYPE

- Crear un ejemplo de individuo
EjemploCuatroQuesos con 723



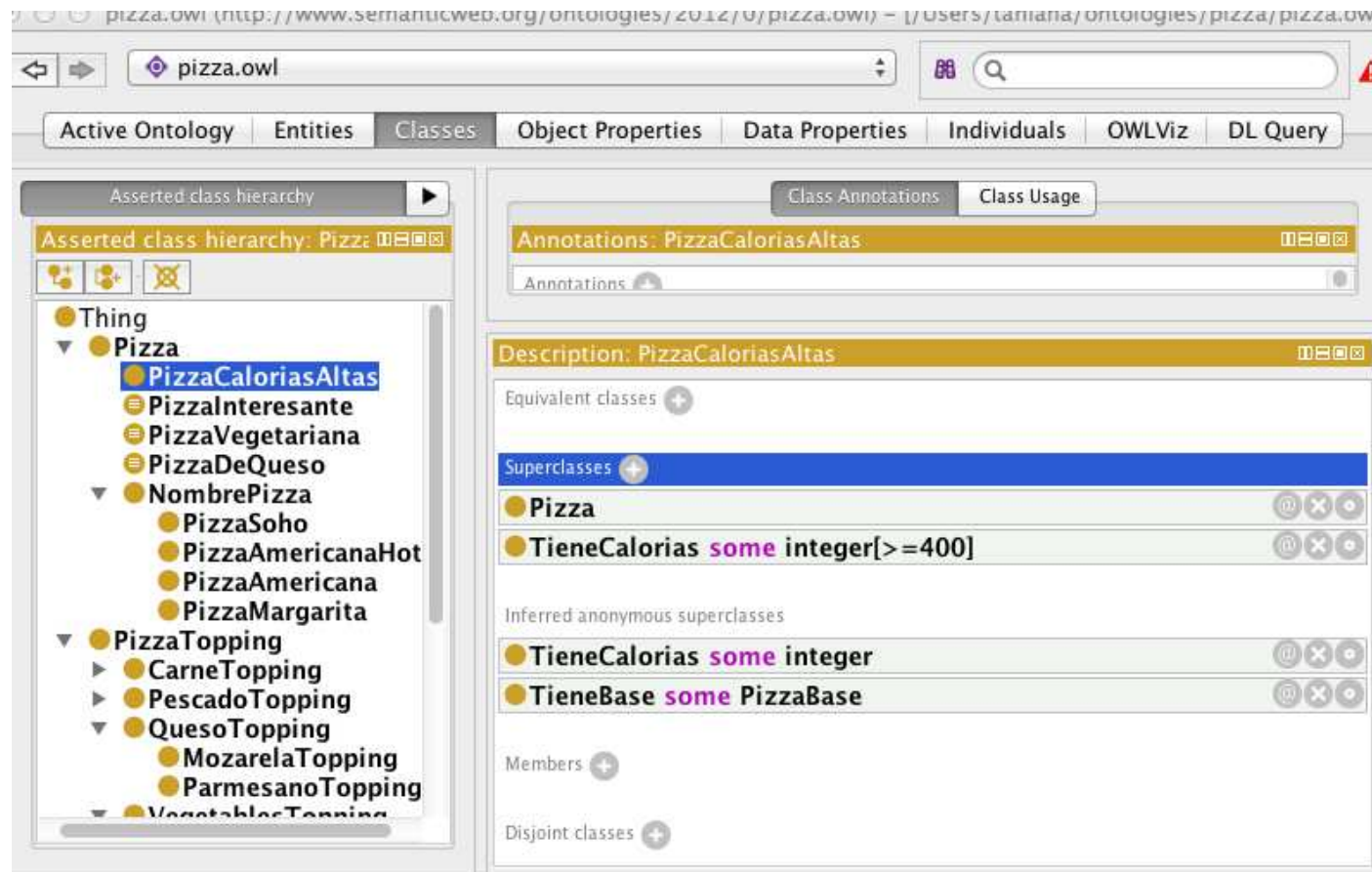
PROPIEDADES DATATYPE

- Seleccione pizza y añada una superclase y asegure que la restriction esta en some y presione Ok



PROPIEDADES DATATYPE

- Crear una Subclase PizzaCaloriasAltas de Pizza que tiene calorías igual o mayor que 400.



PROPIEDADES DATATYPE

- Crear la clase equivalente

The screenshot shows the Protege OWL editor interface. The browser address bar displays the URL: `http://www.semanticweb.org/ontologies/2012/0/pizza.owl`. The main window has several tabs: "Active Ontology", "Entities", "Classes", "Object Properties", "Data Properties", "Individuals", "OWLViz", and "DL Query". The "Classes" tab is active, showing the "Asserted class hierarchy: Pizza". The hierarchy lists the following classes:

- Thing
 - Pizza
 - PizzaCaloriasAltas
 - PizzaInteresante
 - PizzaVegetariana
 - PizzaDeQueso
 - NombrePizza
 - PizzaSoho
 - PizzaAmericanaHot
 - PizzaAmericana
 - PizzaMargarita
 - PizzaTopping
 - CarneTopping
 - PescadoTopping
 - QuesoTopping
 - MozarelaTopping
 - ParmesanoTopping
 - VegetablesTopping

The right-hand pane shows the "Class Annotations" and "Class Usage" tabs. The "Class Annotations" tab is active, displaying the "Annotations: PizzaCaloriasAltas". Below this, the "Description: PizzaCaloriasAltas" is shown, which includes the following information:

- Equivalent classes:
 - `Pizza and TieneCalorias some integer[>= "400"^^integer]`
- Superclasses:
 - `TieneCalorias some integer`
 - `TieneBase some PizzaBase`
- Inferred anonymous superclasses:
 - `TieneCalorias some integer`
 - `TieneBase some PizzaBase`
- Members:
 - `+`
- Disjoint classes:
 - `+`

PROPIEDADES DATATYPE

- Crear PizzaCaloriasBaja menor 400

The screenshot shows the Protégé OWL editor interface. The browser address bar displays the URL: `http://www.semanticweb.org/ontologies/2012/0/pizza.owl`. The main window has several tabs: **Active Ontology**, **Entities**, **Classes** (selected), **Object Properties**, **Data Properties**, **Individuals**, **OWLviz**, and **DL Query**.

On the left, the **Asserted class hierarchy** is shown. It lists the following classes under **Thing**:

- Pizza**
 - PizzaCaloriasBaja** (highlighted)
 - PizzaCaloriasAltas**
 - PizzaInteresante**
 - PizzaVegetariana**
 - PizzaDeQueso**
 - NombrePizza**
 - PizzaSoho**
 - PizzaAmericanaHot**
 - PizzaAmericana**
 - PizzaMargarita**
 - PizzaTopping**
 - CarneTopping**
 - PescadoTopping**
 - QuesoTopping**
 - MozarelaTopping**
 - BermecanoTopping**

On the right, the **Class Annotations** and **Class Usage** tabs are visible. The **Annotations: PizzaCaloriasBaja** section shows no annotations. The **Description: PizzaCaloriasBaja** section shows the following definition:

Equivalent classes +

- Pizza** and **TieneCalorias** some integer[< "400"^^integer]

Superclasses +

- TieneCalorias** some integer
- TieneBase** some **PizzaBase**

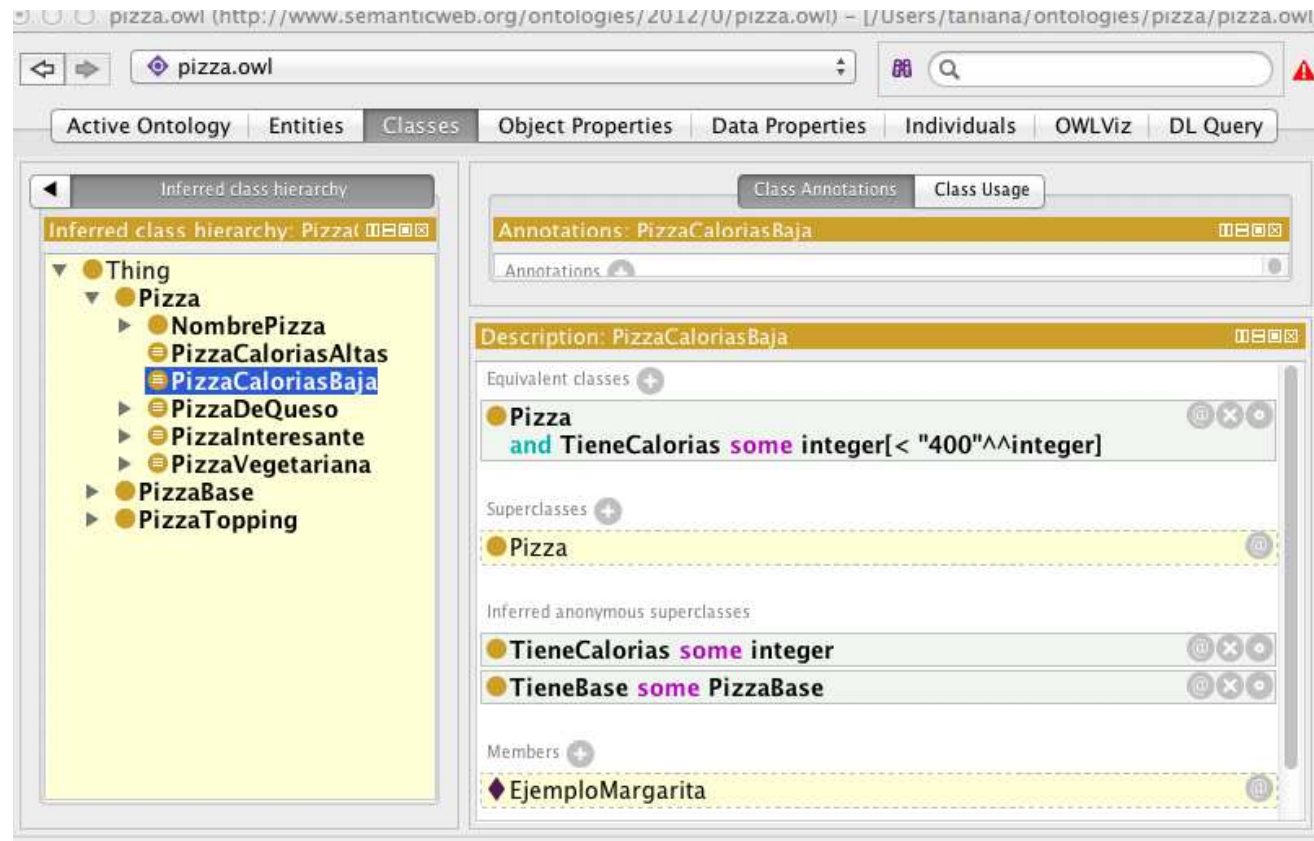
Inferred anonymous superclasses

Members +

Disjoint classes +

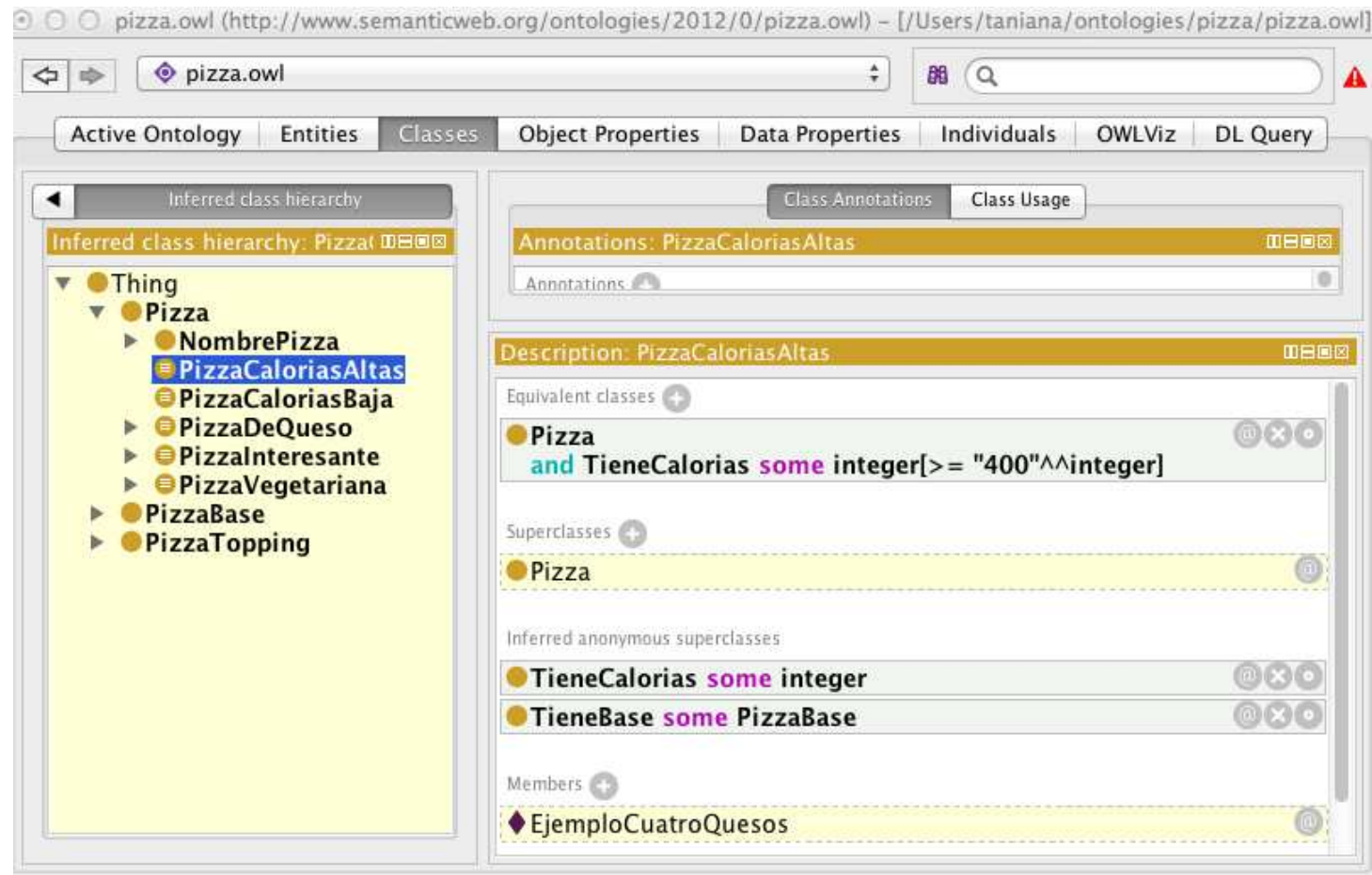
PROPIEDADES DATATYPE

- ◉ Usamos el razonador



PROPIEDADES DATATYPE

- Usamos el razonador



ACTIVIDAD EXTRA

Realizar la Ontología de la Universidad de Los Andes

Realizar el cookbook de prortégé 4.2.