

TALLER No. 3

OWL Y PROTÉGÉ

Profesor: Jaime Alberto Guzmán Luna

Contenido del taller:

1. Manejo de protege
2. OWL

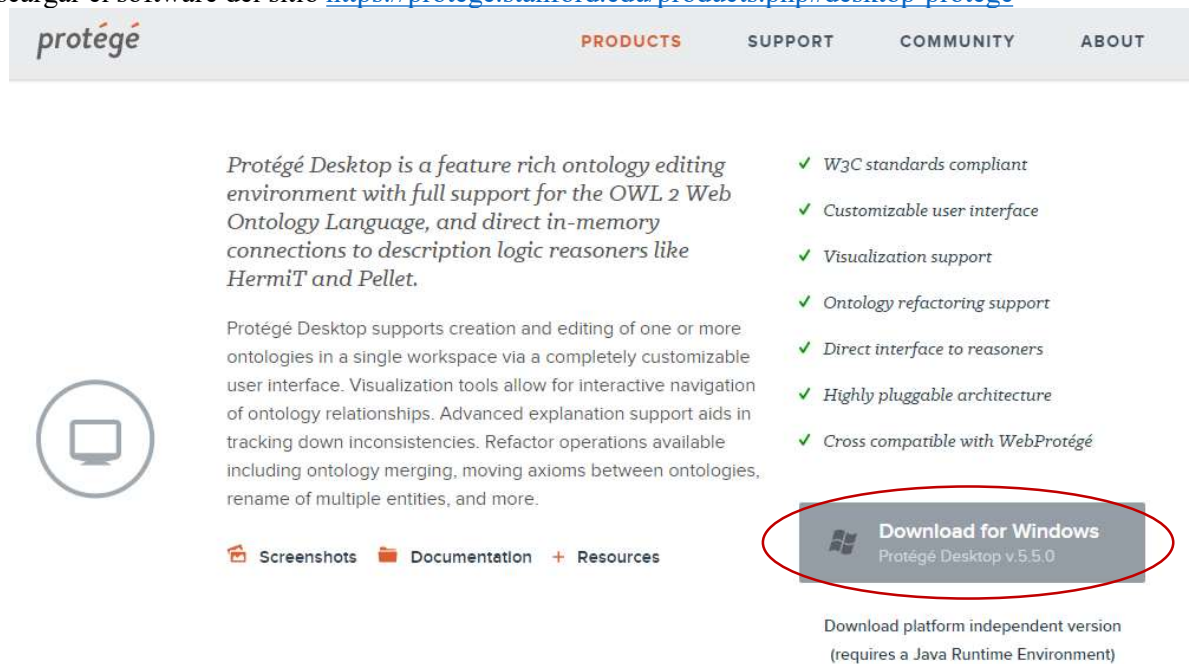
Tutorial para construir una ontología OWL en Protégé 5.5

I. Instalar Java en su versión 8

Para esto descargue e instale la versión del JDK acorde a su sistema operativo en el siguiente enlace <https://github.com/frekele/oracle-java/releases>

II. Cómo instalar Protégé 5.5 ?

Descargar el software del sitio <https://protege.stanford.edu/products.php#desktop-protege>



The screenshot shows the Protégé website with the following content:

- protégé** logo and navigation links: PRODUCTS, SUPPORT, COMMUNITY, ABOUT.
- Protégé Desktop** description: "Protégé Desktop is a feature rich ontology editing environment with full support for the OWL 2 Web Ontology Language, and direct in-memory connections to description logic reasoners like HermiT and Pellet."
- Features list:**
 - ✓ W3C standards compliant
 - ✓ Customizable user interface
 - ✓ Visualization support
 - ✓ Ontology refactoring support
 - ✓ Direct interface to reasoners
 - ✓ Highly pluggable architecture
 - ✓ Cross compatible with WebProtégé
- Download for Windows** button (highlighted with a red circle): "Protégé Desktop v.5.5.0"
- Download platform independent version** (requires a Java Runtime Environment)
- Links:** Screenshots, Documentation, Resources.

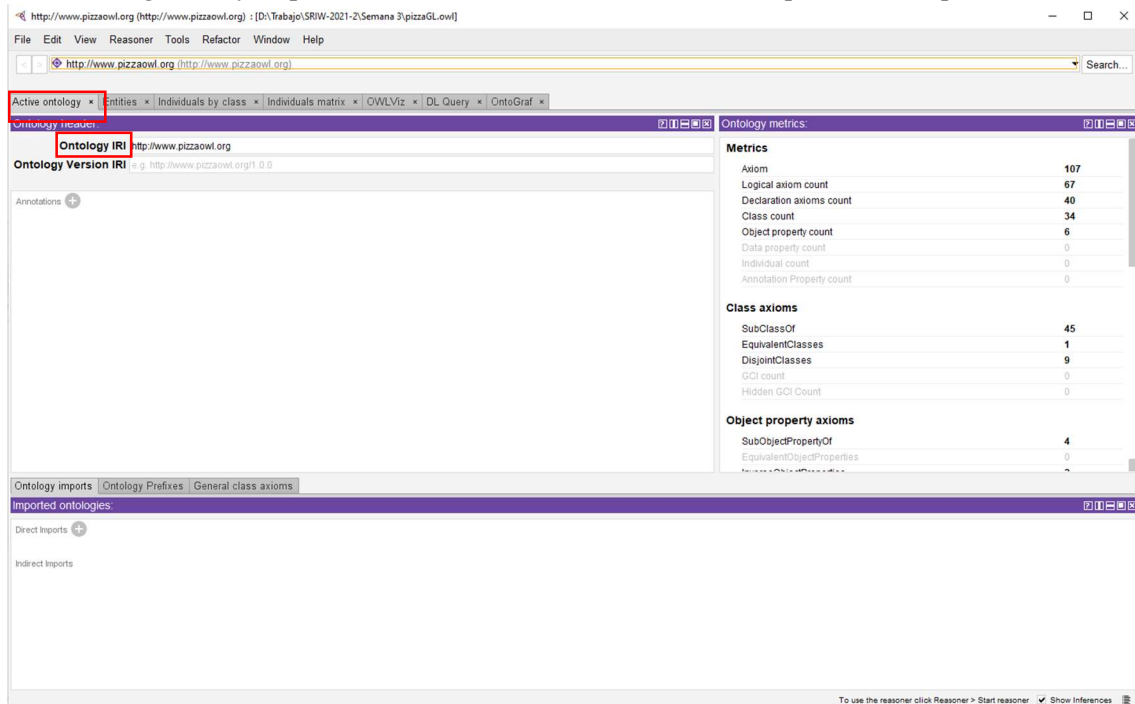
NOTA: Esta versión requiere tener instalado el “Java Runtime Environment”.

Es bueno registrarse, si el software lo solicita.

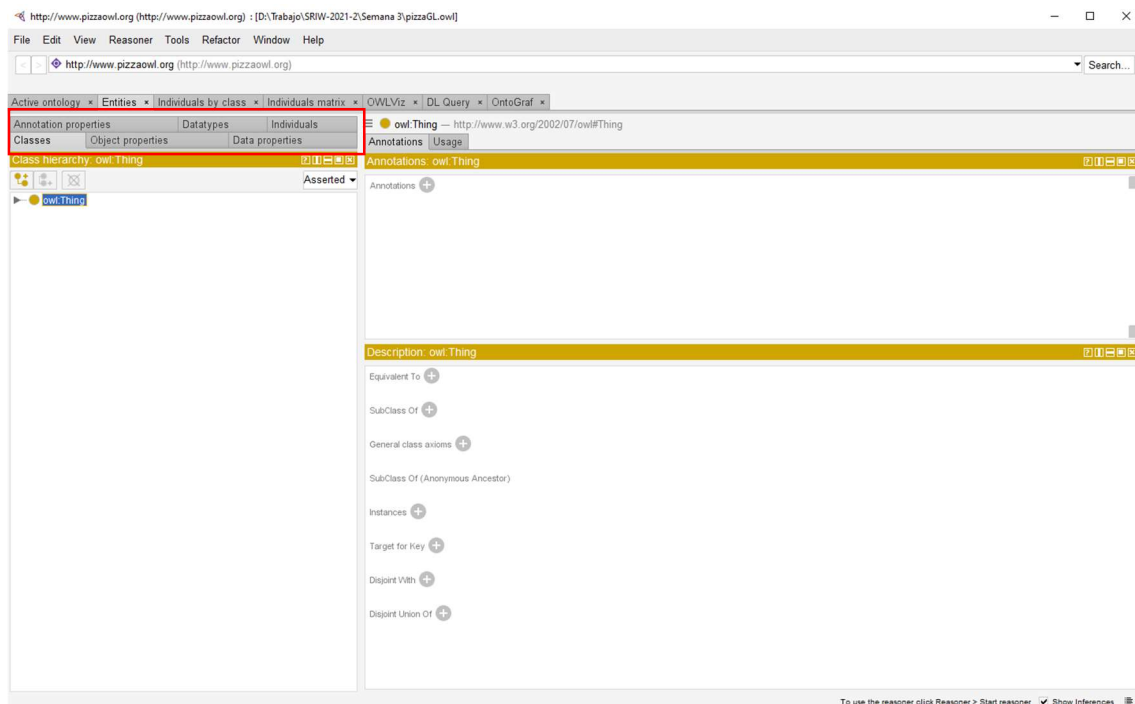
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

II. Construir una ontología

1. Ejecutar Protégé.exe en la carpeta donde se descargó el software. Aparece la ventana siguiente y se puede cambiar el IRI, dándole el nombre que nosotros queramos



Podemos observar varias pestañas. Vamos a la pestaña *Entities*. Podemos apreciar la siguiente ventana:



Vemos otro conjunto de pestañas. Entre ellas la pestaña *Classes*. Allí podemos crear la jerarquía de clases. La ontología vacía contiene la clase « *Thing* », que es raíz de todas las clases de la ontología.

También están las pestañas para las propiedades. Ellas son la *Object properties*, *Data properties* y *Annotation properties*.

La pestaña para los individuos es *Individuals*

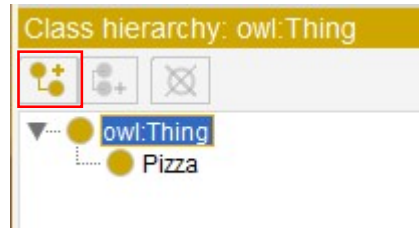
LAS CLASES OWL

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

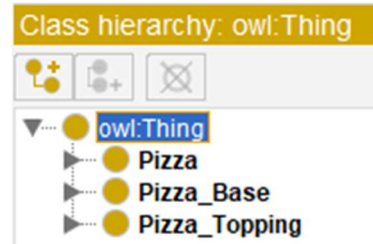
Vamos a crear las clases *Pizza*, *Pizza_Topping* y *Pizza_Base* como sub-clases de la clase *owl:Thing*

Para ello, debemos estar dentro de las pestañas *Entities/Classes*

El primer icono, permite crear subclases:

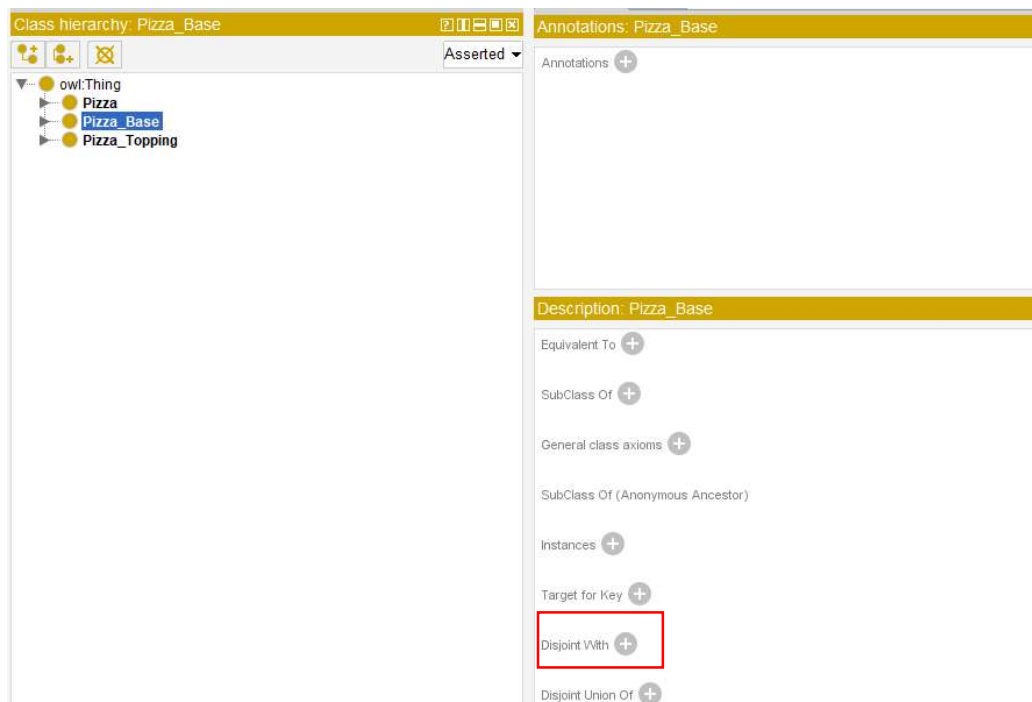


La jerarquía de clases debe quedar así:



En OWL se supone que las clases se traslapan. No se puede suponer que un individuo no es miembro de una clase particular simplemente porque no se ha dicho que lo es. Con el fin de separar un grupo de clases, se deben definir disjuntas las unas de las otras.

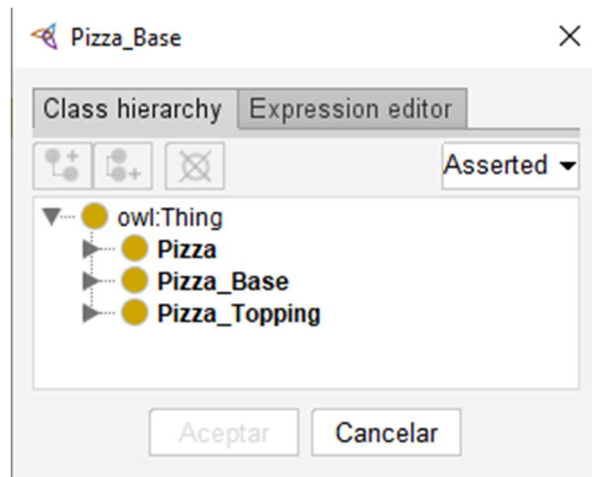
Entonces vamos a definir que esas tres clases que acabamos de crear son disjuntas entre ellas.



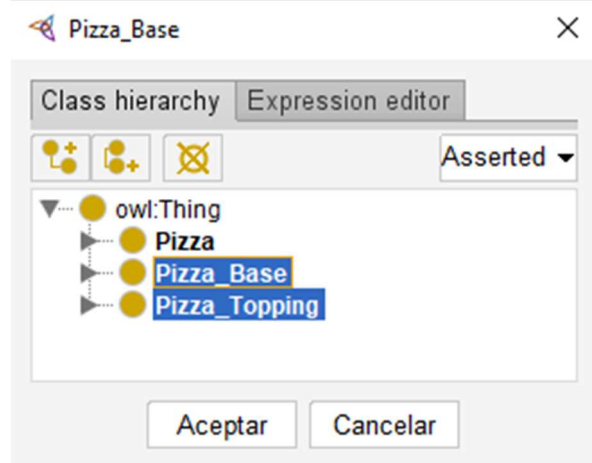
Seleccionando la clase *Pizza* vamos a utilizar la opción *Disjoint with* para decir que todas las hermanas de la clase *Pizza* son clases disjuntas entre sí.

Si hacemos click sobre el signo de “+” al lado de “*Disjoint with*” aparece la siguiente ventana:

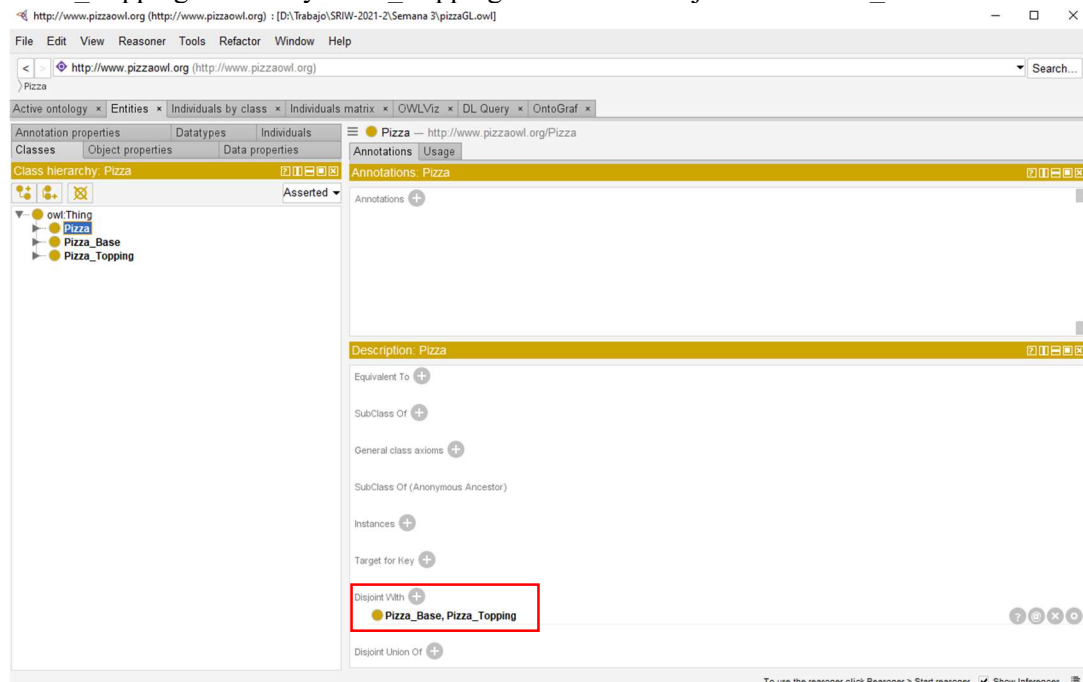
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Damos click en Pizza_Base y con la tecla Ctrl presionada seleccionamos las hermanas de Pizza y luego damos click en Aceptar



Note que la ventana de las clases disjuntas muestra ahora las clases Pizza_Base y Pizza_Topping como clases disjuntas de Pizza. Además, crea automáticamente las disyunciones entre las otras clases. Es decir, Pizza_Base y Pizza como clases disjuntas de la clase Pizza_Topping. Y Pizza y Pizza_Topping como clases disjuntas de Pizza_Base.

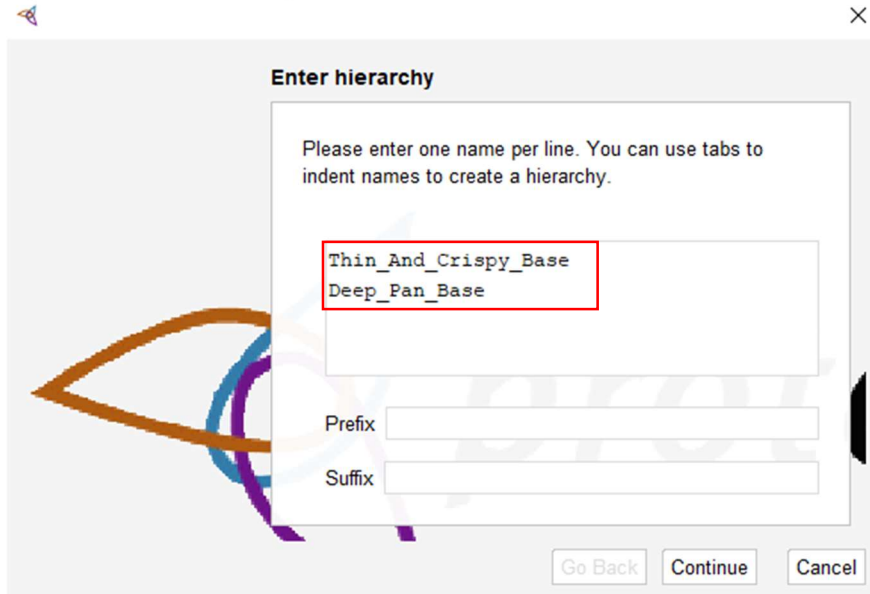


SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

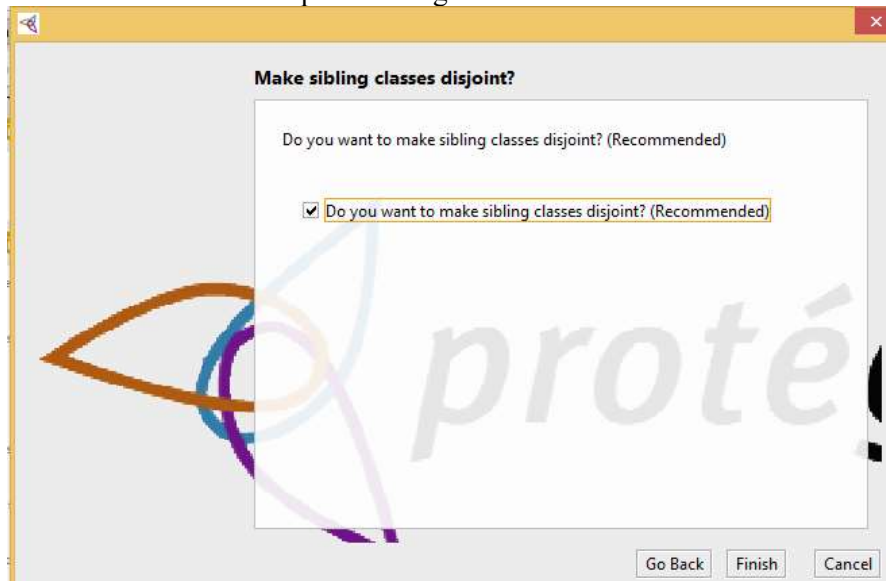
CREAR MÚLTIPLES SUBCLASES

Una manera de crear sub clases rápidamente es la siguiente: en el menú de la parte superior de la ventana Protégé, escoger « **Tools** », y luego « **Create class hierarchy** ». Vamos a crear las subclases **Thin_And_Crispy_Base** y **Deep_Pan_Base** de la clase **Pizza_Base**

Aparecerá la ventana siguiente y escribimos los nombres de las subclases de **Pizza_Base**:

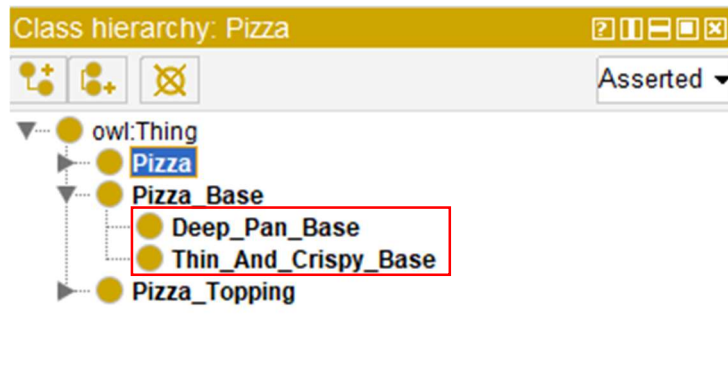


Click en « Continue ». Aparece la siguiente ventana:



Verificamos que esté activada la opción de poner disjuntas las clases que se acabaron de crear. Click en « Finish ». Ahora, tenemos la clase **Pizza_Base** con las subclases: **Thin_And_Crispy_Base**, **Deep_Pan_Base**. Ver figura siguiente:

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Ahora, vamos a crear las subclases de algunas clases:

La clase **Pizza_Topping** tiene como subclases:

- Meat_Topping**
- Vegetable_Topping**
- Cheese_Topping**
- Seafood_Topping**

La clase **Meat_Topping** tiene las subclases:

- Spicy_Beef_Topping**
- Pepperoni_Topping**
- Salami_Topping**
- Ham_Topping**

La clase **Vegetable_Topping** tiene las subclases:

- Tomato_Topping**
- Olive_Topping**
- Mushroom_Topping**
- Pepper_Topping**
- Onion_Topping**
- Caper_Topping**

La clase **Pepper_Topping** tiene las subclases:

- Red_Pepper_Topping**
- Green_Pepper_Topping**
- Jalapeno_Pepper_Topping**

La clase **Cheese_Topping**, tiene las subclases

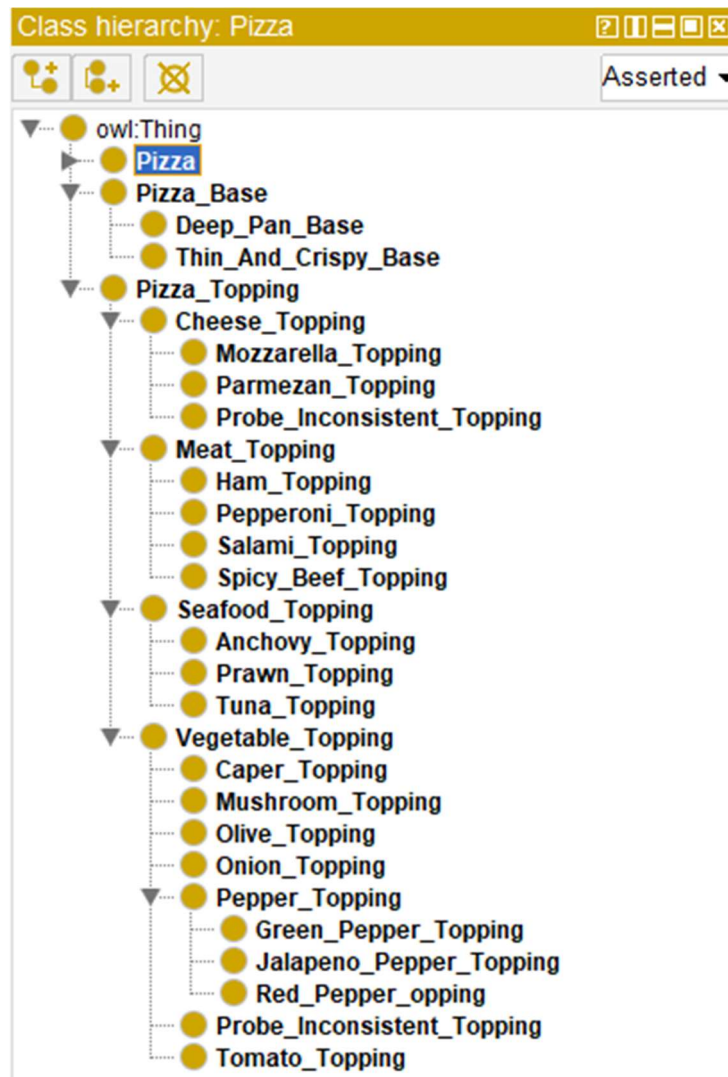
- Mozzarella_Topping**
- Parmesan_Topping**

La clase **Seafood_Topping** tiene las subclases

- Tuna_Topping**
- Anchovy_Topping**
- Prawn_Topping**

La jerarquía de clases debe parecerse a la de la figura siguiente:

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

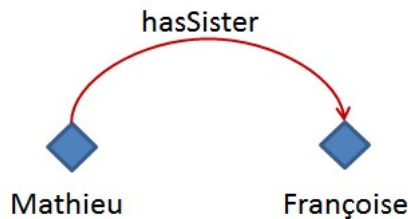


SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

Las propiedades OWL

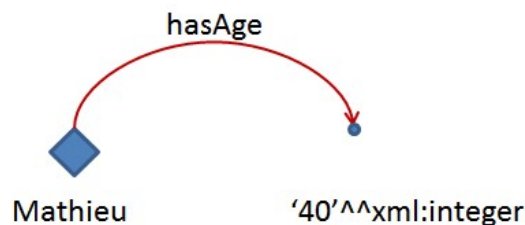
Las propiedades OWL representan las relaciones entre dos individuos. Existen dos principales tipos de propiedades: las propiedades de tipo Objeto («*Object Properties*») las propiedades de tipo Dato («*Data Properties*»).

Las «*Object Properties*» ligan un individuo a otro individuo.



La «*Object Properties*» **hasSister** ligando el individuo « **Mathieu** » al individuo « **Françoise** »

Propiedades de tipo «*Data Properties*» ligando un individuo a un valor de un tipo dado, por ejemplo: string, integer, etc.

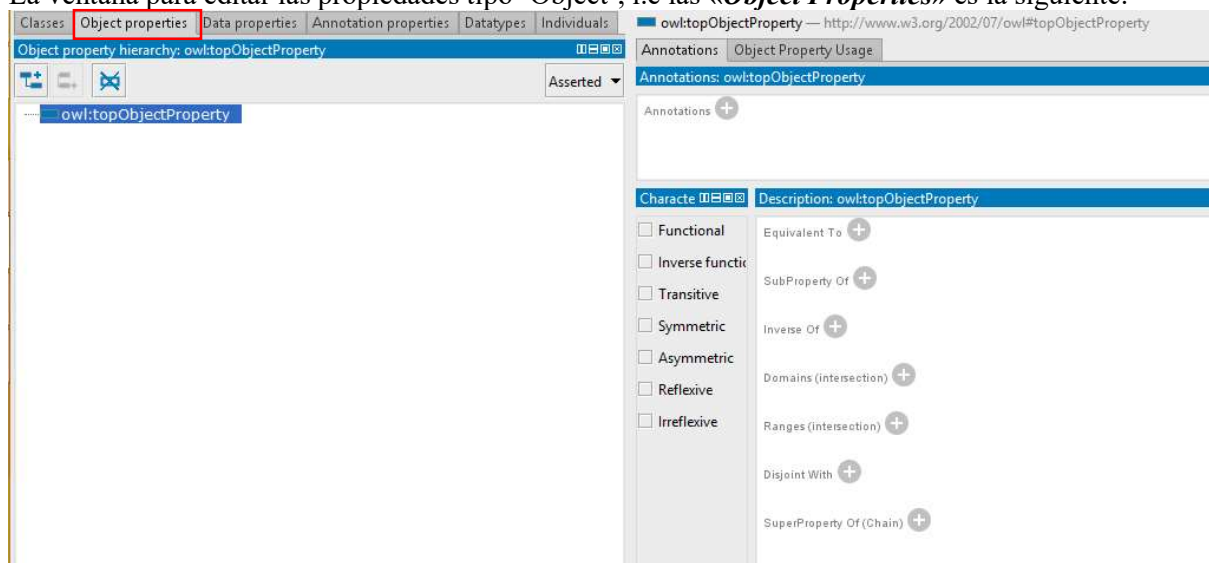


La «*Data properties*» (**hasAge**) ligando el individuo « **Matthieu** » al literal '**40**' que es de tipo entero

OWL tiene un tercer tipo de propiedad: las propiedades de anotación. Las propiedades de anotación pueden ser utilizadas para adicionar información a las clases (metadatos), los individuos y las propiedades 'Data properties'.

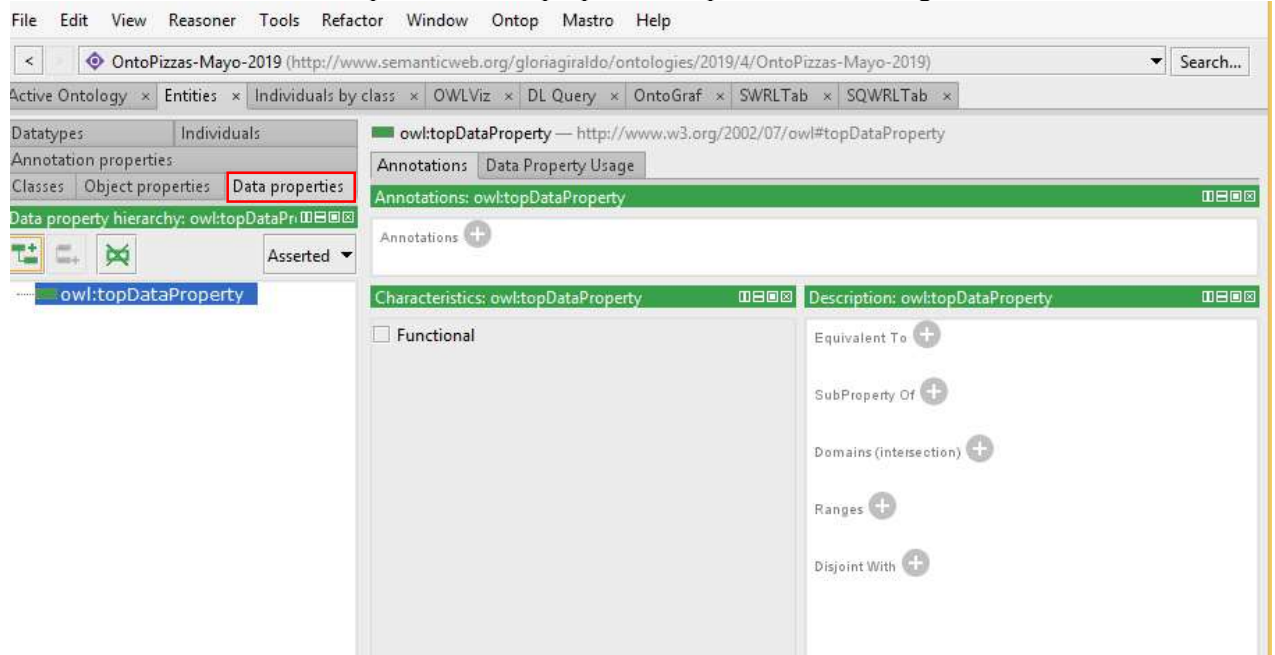
Aunque no existe ninguna convención estricta para darle nombre a las propiedades, es recomendable que los nombres de las propiedades comiencen por una letra minúscula, sin espacios y el resto de las palabras la primera en mayúscula. Es igualmente recomendable que las propiedades lleven como prefijo la palabra «has» o la palabra «is», o su equivalente según el idioma de la ontología. Por ejemplo 'hasPart', 'isPartOf', 'hasManufacturer', 'isProducerOf'. Esta convención, que es próxima al lenguaje natural, ayuda a hacer la propiedad más clara.

La ventana para editar las propiedades tipo 'Object', i.e las «*Object Properties*» es la siguiente:



SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

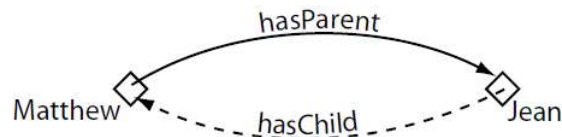
De manera similar, la ventana para editar las propiedades tipo ‘Data » es la siguiente:



Vamos a crear las propiedades tipo “Object”: **has_Ingredient**, **has_Topping** y **has_Base**:
En OWL, las propiedades pueden tener sub propiedades, de tal manera que es posible formar jerarquías de propiedades. En nuestro caso de las Pizzas, es mejor si **has_Base** y **has_Topping** son sub propiedades de **has_Ingredient**, se puede entonces deslizar con el ratón **has_Base** y **has_Topping** hacia el interior de **has_Ingredient**.

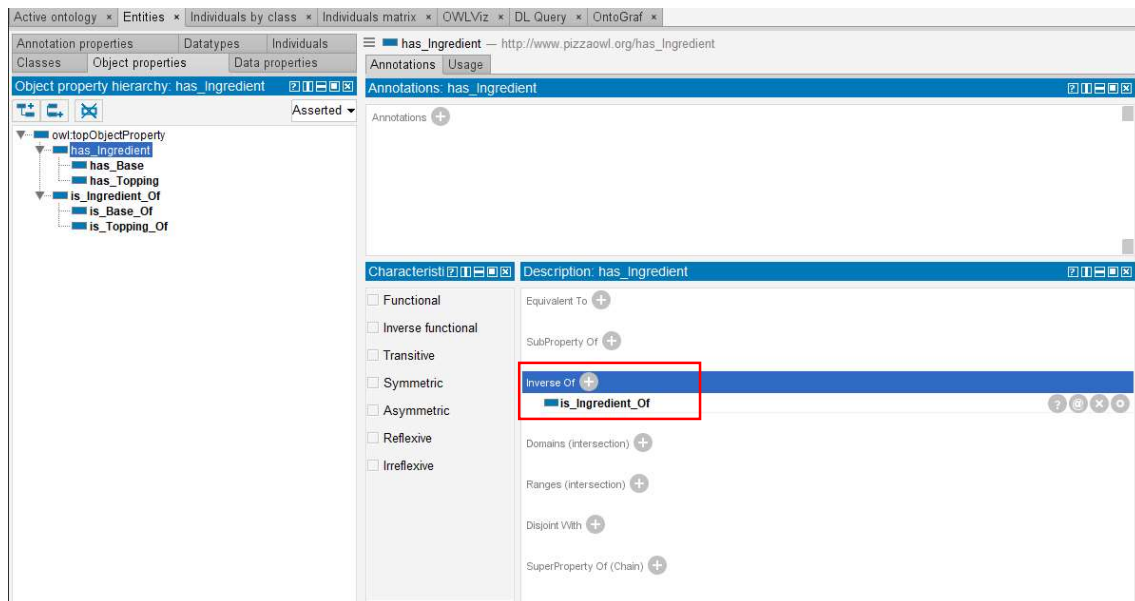
PROPIEDADES INVERSAS

Cada propiedad puede tener su propiedad inversa correspondiente. Por ejemplo:



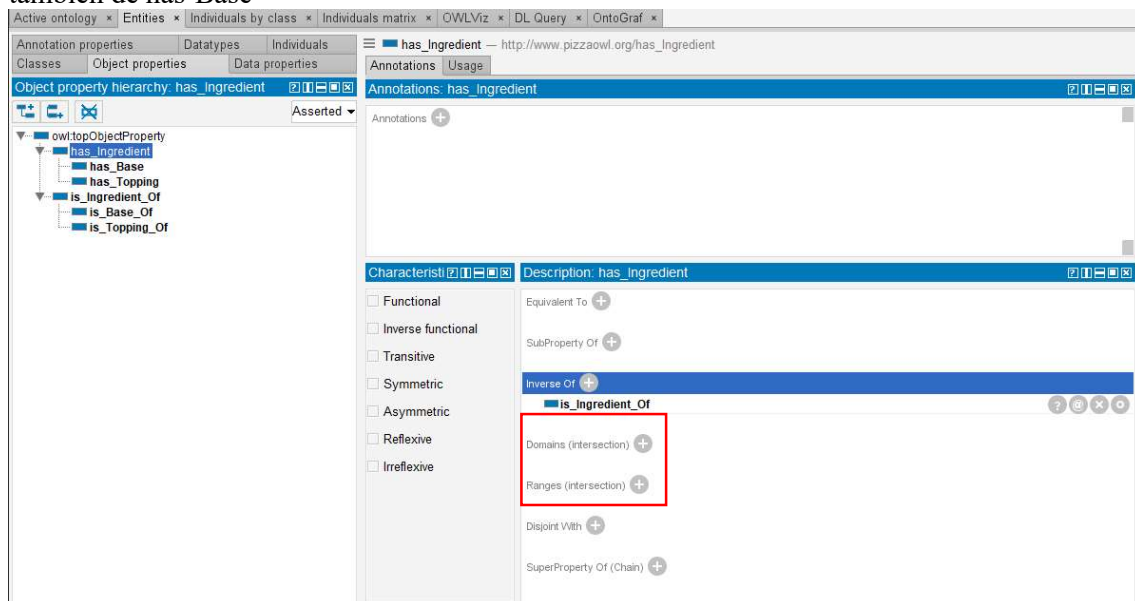
Se pueden definir propiedades inversas utilizando la siguiente interfaz. Por ejemplo, se puede definir que la propiedad inversa de ‘has_Ingredient’ es ‘is_Ingredient_Of’. La inversa de ‘has_Base’ es ‘is_Base_Of’ y la inversa de ‘has_Topping’ es ‘is_Topping_Of’. Para ello, vamos a crear la propiedad ‘is_Ingredient_Of’ y las sub propiedades ‘is_Base_Of’ y ‘is_Topping_Of’. Ahora, situándonos en cada propiedad, definimos las propiedades inversas con el icono « Set Inversa Property ».

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

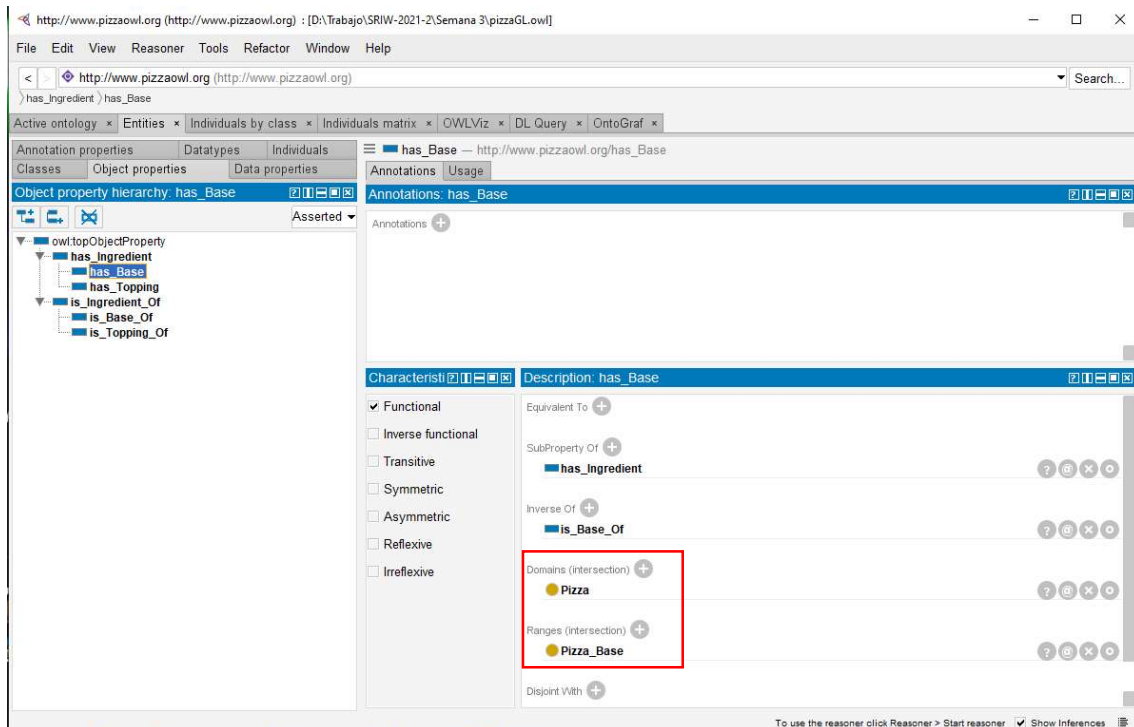
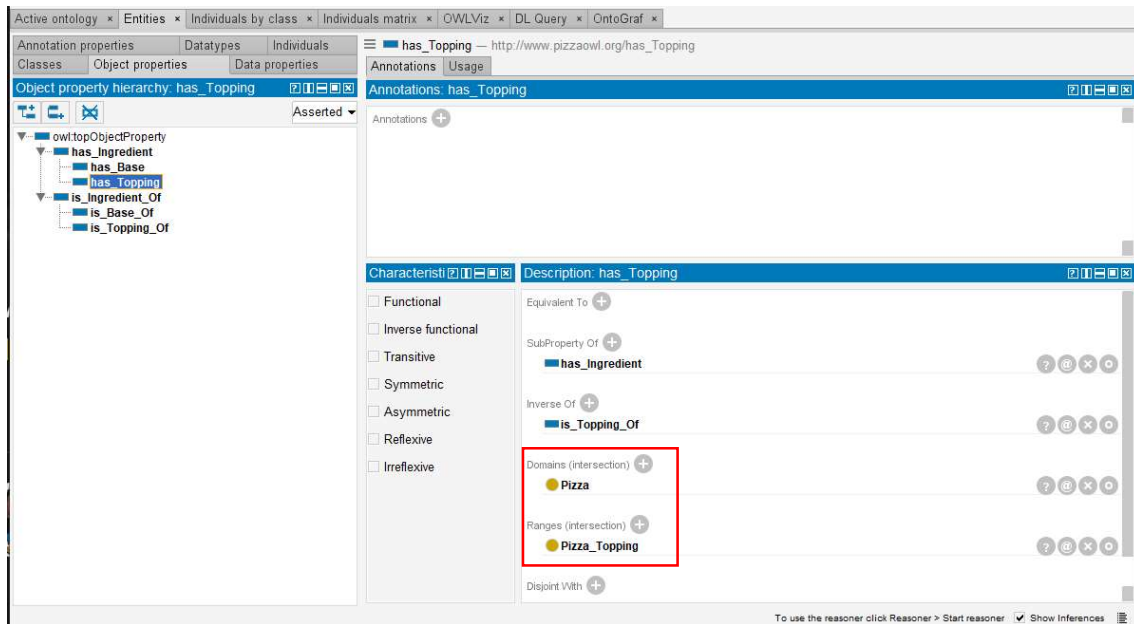


El dominio y el rango de las propiedades

Las propiedades ligan individuos de un dominio con individuos de un rango. Por ejemplo, la propiedad 'has_Topping' liga individuos de la clase 'Pizza' a individuos de la clase 'Pizza_Topping'. En ese caso el dominio de la propiedad 'has_Topping' es 'Pizza' y el rango es 'Pizza_Topping'. Ahora definamos el dominio y el rango de la propiedad 'has_Topping' y también de has-Base



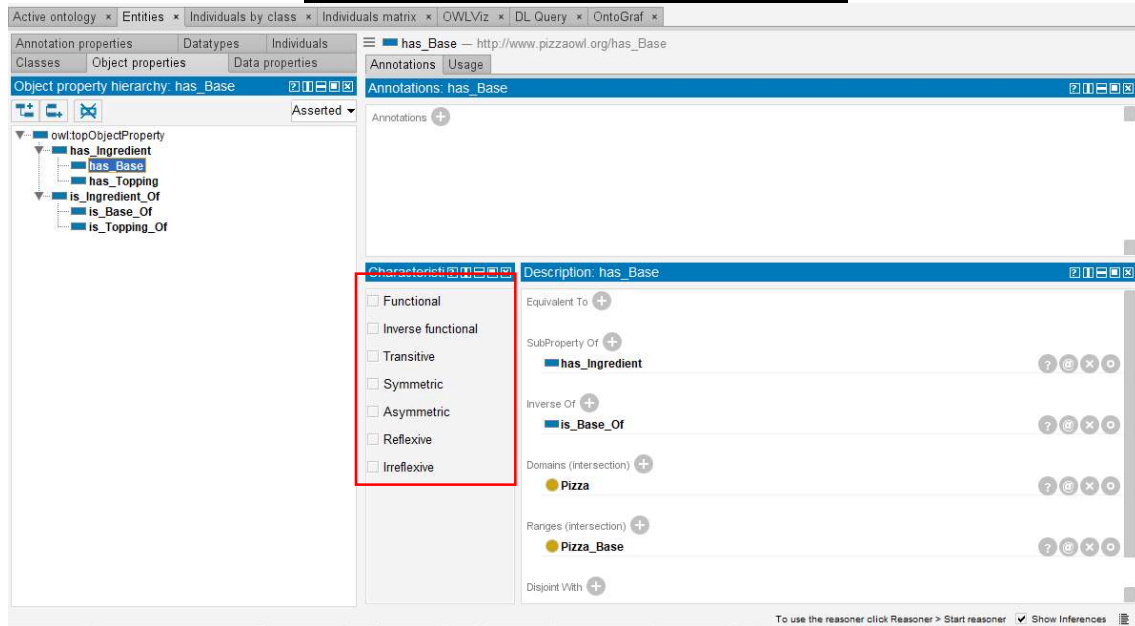
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



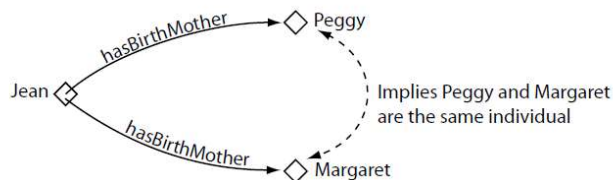
Es importante resaltar que en OWL el dominio y el rango son utilizados como «axiomas», no como restricciones a verificar. Por ejemplo, si la propiedad ‘has_Topping’ tiene como dominio ‘Pizza’ y aplicamos la propiedad ‘has_Topping’ a los individuos de la clase ‘Ice_Cream’, eso no sería un error. Ello sería utilizado por el razonador para deducir que la clase ‘Ice_Cream’ debe ser una sub clase de la clase **Pizza**! Sin embargo, si declaramos ‘Ice_Cream’ y ‘Pizza’ como clases disjuntas, **entonces sí habría un error!!!**

Es posible especificar varias clases como el rango de una propiedad. Si en Protégé-OWL varias clases son especificadas como el rango de una propiedad, entonces el rango es interpretado como la unión de las clases.

Características de las propiedades en OWL

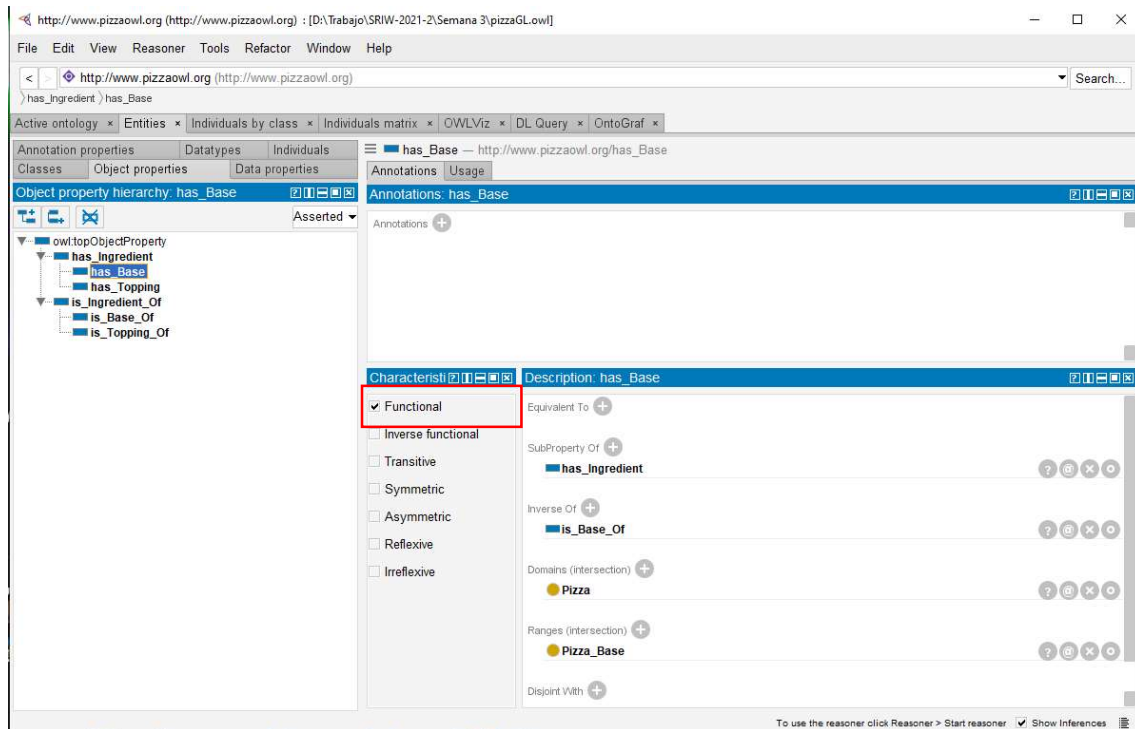


Propiedad Funcional : si una propiedad es funcional, para un individuo dado, puede haber como máximo un individuo que está ligado a otro individuo vía esta propiedad. Por ejemplo, una persona solamente puede tener una madre de nacimiento. Si decimos que Jean ‘hasBirthMother’ Peggy y Jean ‘hasBirthMother’ Margaret, eso implica que Peggy y Margaret son el mismo individuo, de otra manera eso sería un error. Las propiedades funcionales son conocidas como propiedades de valor único.

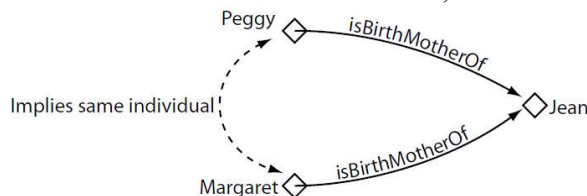


Ahora vamos a definir que una Pizza puede solamente tener una base, para ello nosotros definimos la propiedad **has_Base** como funcional.

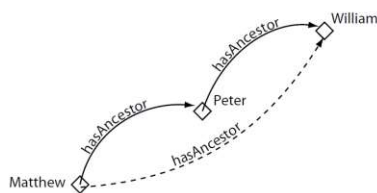
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Propiedad Funcional Inversa : si una propiedad es funcional Inversa, eso significa que su propiedad Inversa es funcional. Por ejemplo, si ‘**isBirthMotherOf**’ es la propiedad Inversa de ‘**hasBirthMother**’ la cual es funcional, entonces ‘**isBirthMotherOf**’ es Inversa funcional.

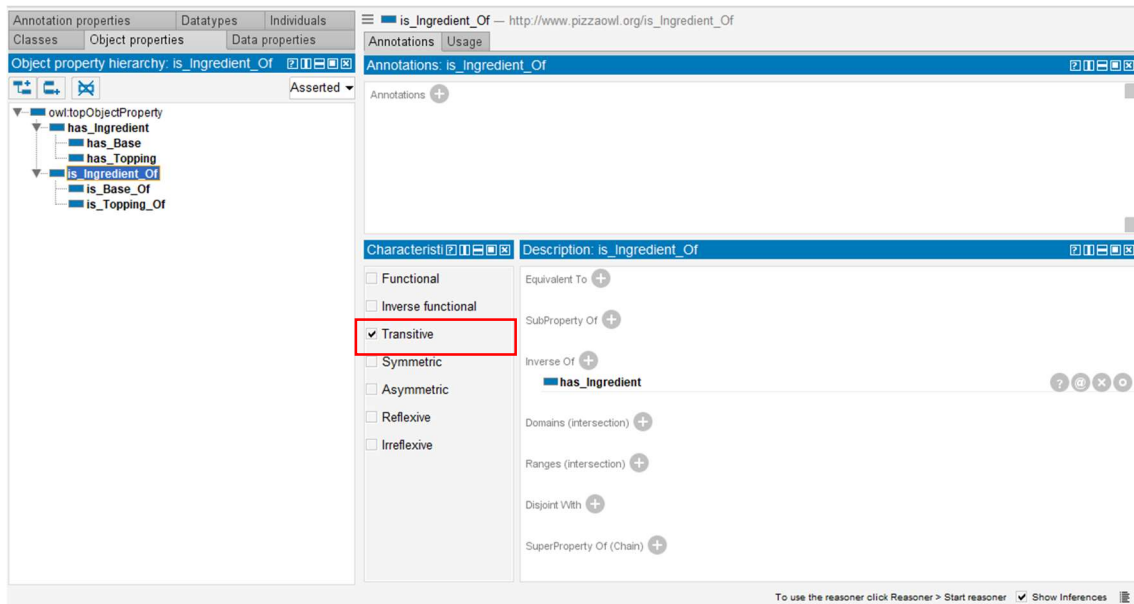


Propiedades transitivas : Si una propiedad **P** es transitiva, y **P** liga el individuo **A** al individuo **B**, y también el individuo **B** al individuo **C**, entonces podemos deducir que el individuo **A** está ligado al individuo **C** vía la propiedad **P**. Por ejemplo, si tenemos la propiedad **hasAncestor** como transitiva y si el individuo **Matthew** tiene como ancestro al individuo **Peter** y **Peter** tiene como ancestro al individuo **William**, entonces podemos decir que **Matthew** tiene como ancestro a **William**.

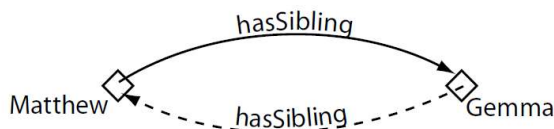


Vamos a definir la propiedad **is_Ingredient_Of** como transitiva.

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Propiedades simétricas: Si una propiedad **P** es simétrica, y esta propiedad liga el individuo **A** al individuo **B**, entonces el individuo **B** está igualmente ligado al individuo **A** por la propiedad **P**.



OBSERVACIÓN:

Si una propiedad es transitiva entonces su propiedad Inversa es también transitiva.

Si una propiedad es transitiva entonces ella no puede ser funcional.

OWL-DL no permite **Data properties** transitivas, ni simétricas, ni inversas.

Descripción y definición de clases

Ya tenemos algunas propiedades que nos permitirán definir algunas clases de nuestra ontología de Pizzas. En OWL las propiedades son utilizadas para crear restricciones sobre los individuos pertenecientes a las clases. Las restricciones son de tres tipos:

Restricciones de Cuantificadores

Restricciones de Cardinalidad

Restricciones de 'hasValue'.

Las restricciones de cuantificadores están compuestas por un cuantificador, una propiedad y un «filler». Se pueden utilizar los cuantificadores: Existencial (\exists) y universal (\forall).

El **cuantificador existencial** (\exists) puede ser leído como «para ciertos valores de». En protégé es llamado some.

El **cuantificador universal** (\forall) puede ser leído como «para todos los valores de». En protégé es llamado only.

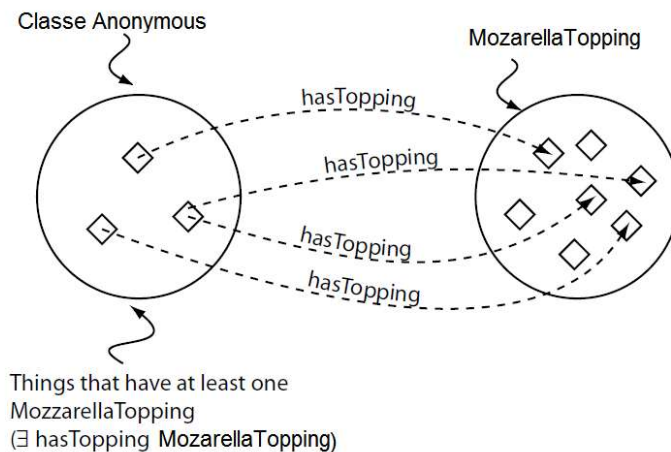
Por ejemplo, la restricción:



Esta restricción describe la clase anónima de los individuos que tienen al menos un topping que

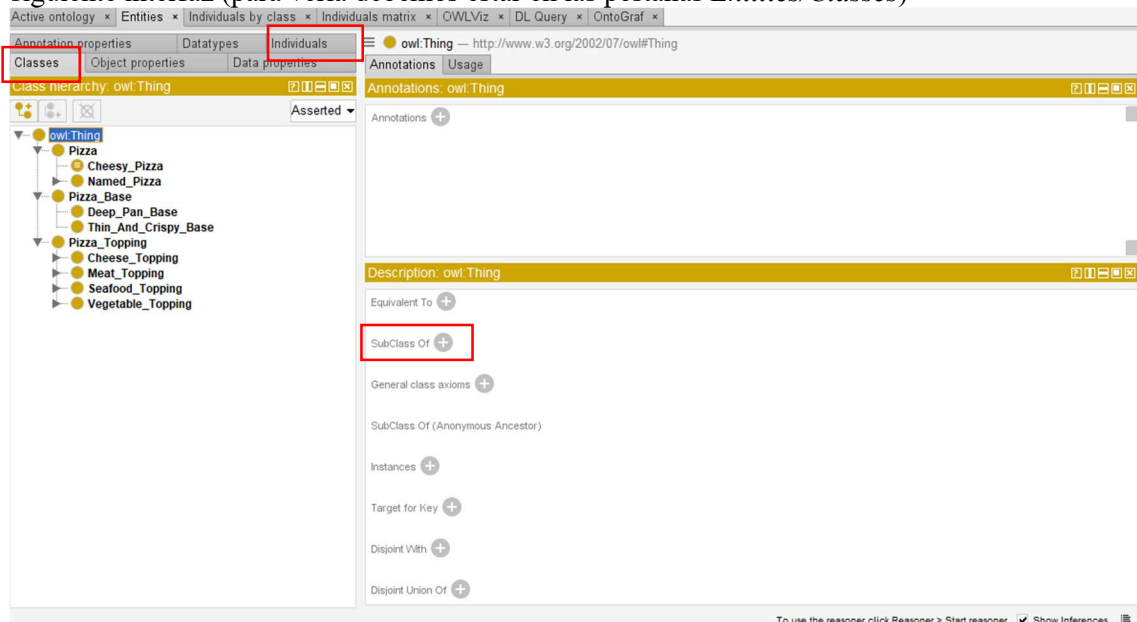
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

es **Mozarella_Topping**. Gráficamente eso sería:



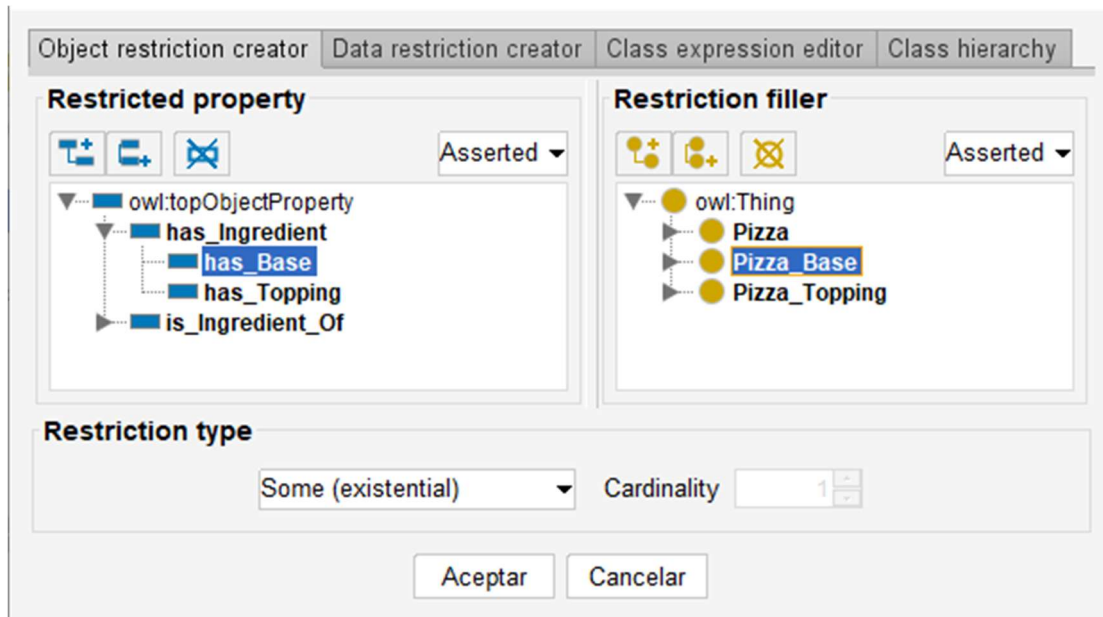
La clase anónima contiene todos los individuos que satisfacen esta restricción.

Las restricciones de las clases son editadas utilizando la opción “SubClass Of” que aparece en siguiente interfaz (para verla debemos estar en las pestañas *Entities/Classes*)

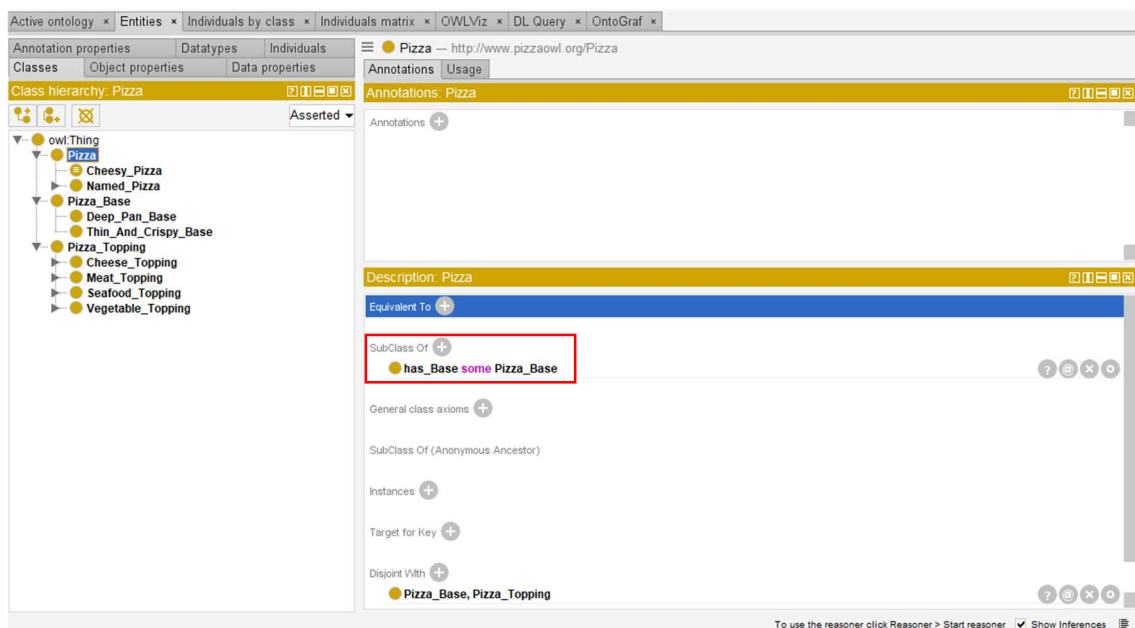


Haciendo click en el signo “+” a la derecha de “SubClass Of”, aparece la siguiente interfaz. La cual tiene una pestaña para crear restricciones sobre los objetos, restricciones sobre los datos, una pestaña que permite editar las expresiones de clases y una pestaña para ver la jerarquía de clases.

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

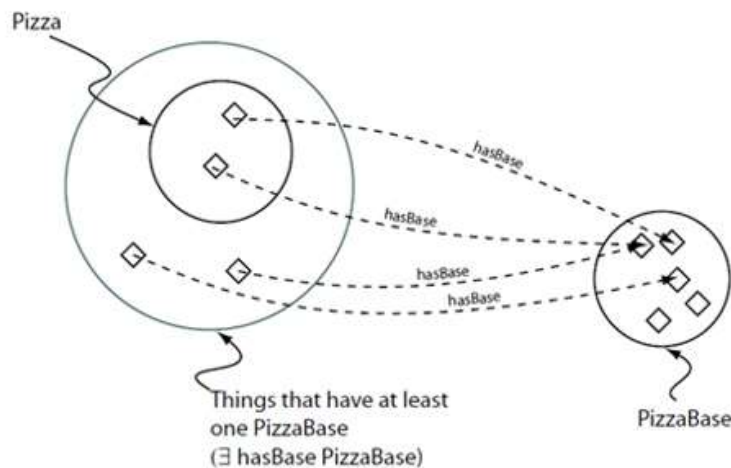


Ahora, vamos a adicionar una restricción para decir que una **Pizza** debe tener una **Pizza_Base**. En la ventana de las clases se debe seleccionar la clase **Pizza**, y luego se da click en “+” que aparece al lado de “SubClass Of”. Elegimos la primera pestaña “*Object restriction Creator*”. En la lista de las “*Restricted property*” se elige “has-base” en el “*Restriction Filler*” se escoge “**Pizza_Base**” y en el “*Restriction type*” se selecciona ‘**Some (existential)**’. Luego click en Aceptar, entonces debemos ver la siguiente expresión:



SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

Lo cual significa que **Pizza** tiene una **Pizza_Base**. Gráficamente sería:

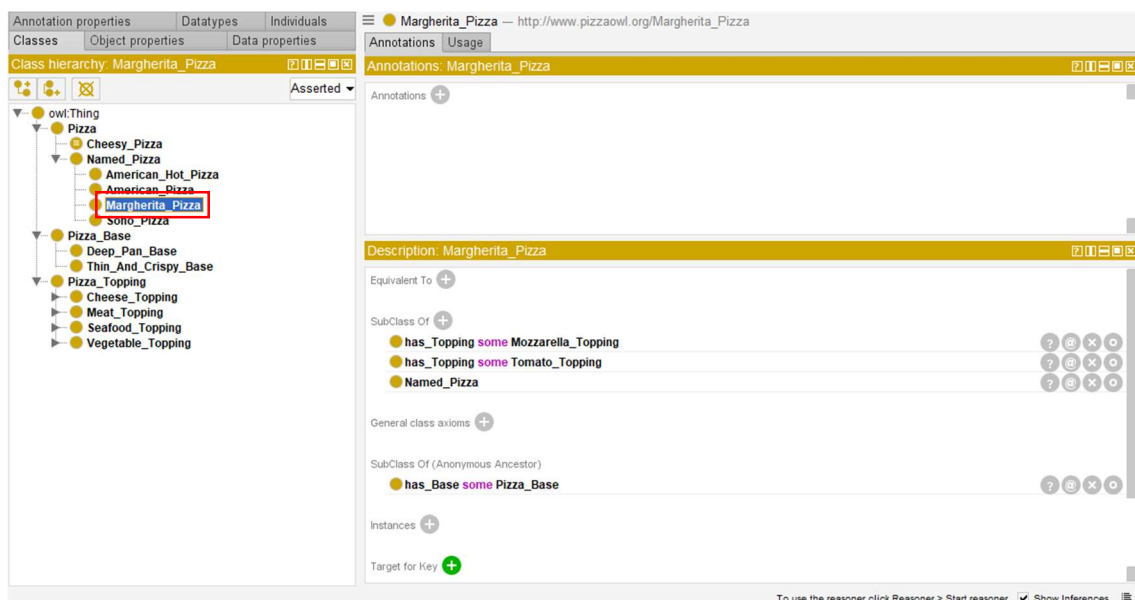


Observe que esas son condiciones necesarias, eso quiere decir que para que una cosa sea una **Pizza**, es necesario que sea miembro de la clase **owl:Thing** (en OWL, todo es un miembro de la clase **owl:Thing**) y además, es necesario que tenga al menos una **Pizza_Base**.

Más formalmente, para que una cosa sea una **Pizza** es necesario que tenga al menos una relación vía la propiedad **has_Base**, con un individuo de la clase **Pizza_Base**.

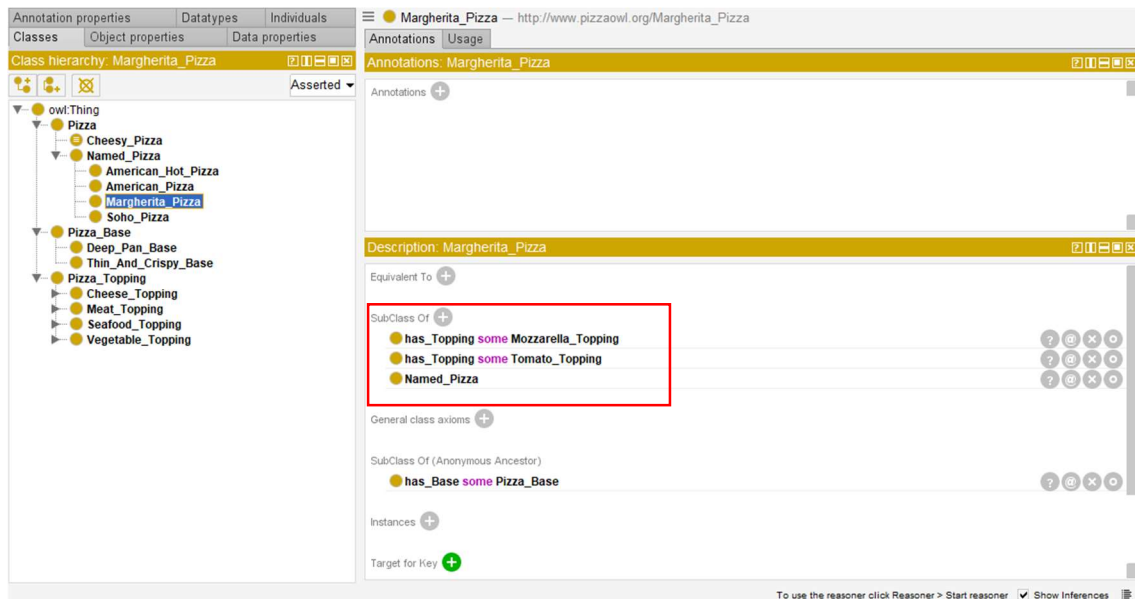
Ahora, vamos a crear diferentes tipos de **Pizza**. Por ejemplo **Margherita_Pizza** que es una pizza que tiene toppings de mozzarella y de tomate.

Con el propósito de tener nuestra ontología bien organizada, vamos a agrupar los nuevos tipos de pizzas en la clase '**Named_Pizza**', subclase de la clase **Pizza**. Y luego creamos **Margherita_Pizza** como una subclase de '**Named_Pizza**'. Vamos a especificar los toppings de **Margherita_Pizza** utilizando la ventana de las restricciones.



Vamos a adicionar las restricciones a **Margherita_Pizza** para que quede explícito que una **Margherita_Pizza** es una **Named_Pizza** la cual tiene al menos un topping de **Mozzarella_Topping** y al menos un topping de **Tomato_Topping**.

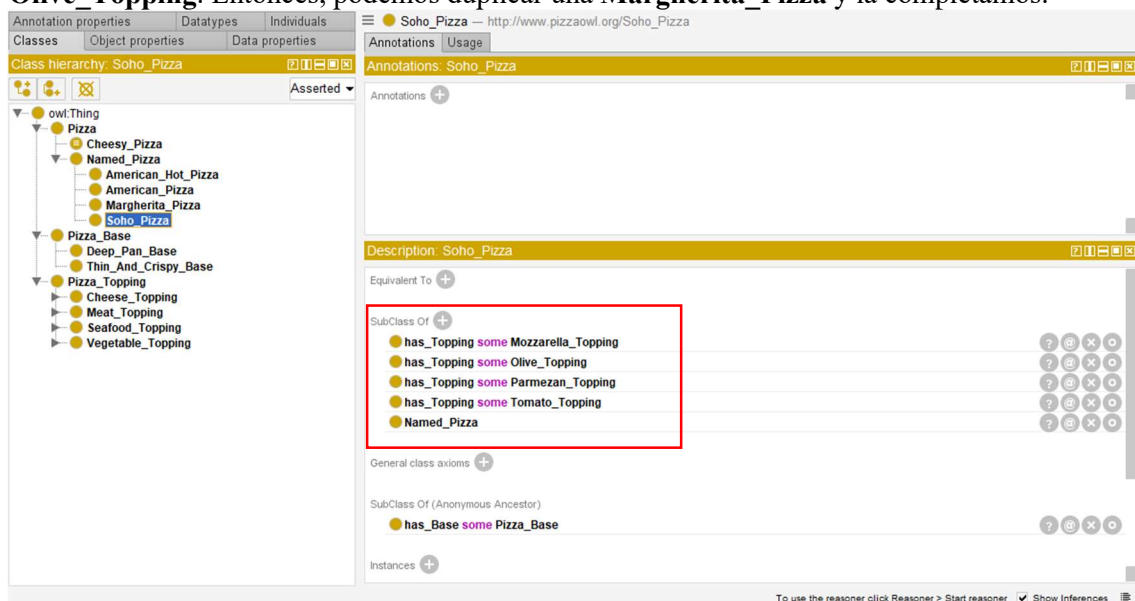
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Ahora vamos a crear una clase **American_Pizza**, la cual es similar a **Margherita_Pizza** pero con un topping adicional que es el **Pepperoni_Topping**. Entonces, vamos a crear un clone de Margherita_Pizza (Botón derecho del ratón y escoger *Duplicate Class*) y luego lo completamos con el otro topping.

Igualmente, podemos crear la clase **American_Hot_Pizza** duplicando la clase **American_Pizza** y luego adicionándole el topping **Jalapeno_Pepper_Topping**.

Una **Soho_Pizza** es muy parecida a una **Margherita_Pizza** pero con **Parmesan_Topping** y **Olive_Topping**. Entonces, podemos duplicar una **Margherita_Pizza** y la completamos.



Todas las subclases de **Named_Pizza** deben ser disjuntas. Entonces situándose sobre la clase **Margherita_Pizza** se selecciona el icono '+' junto a "*Disjoint with*" y se seleccionan todas las clases hermanas de Margherita_Pizza.

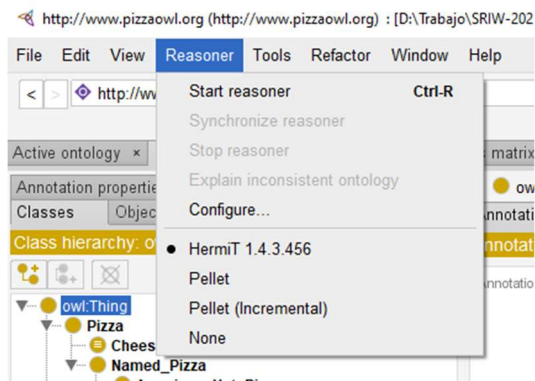
Utilizando un razonador

Una característica de las ontologías OWL-DL es que pueden ser tratadas por un razonador. Uno de los servicios más importantes que el razonador brinda es el de probar la subsunción. Como resultado de este servicio el razonador construye una nueva jerarquía llamada Jerarquía Inferida. Otro servicio estándar ofrecido por el razonador es probar la consistencia de la ontología. El razonador toma como base las condiciones definidas, para probar si es posible que una clase tenga

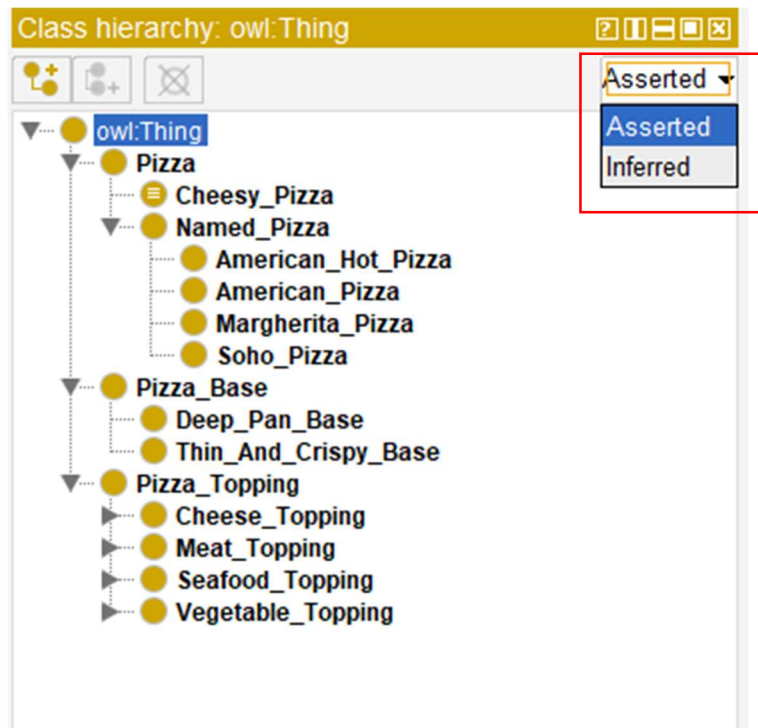
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

instancias o no. Una clase es considerada inconsistente si no es posible que tenga al menos una instancia.

Para chequear los razonadores disponibles en Protégé 5.5 entramos por la opción “Reasoner” situada en el menú superior. Allí seleccionamos un razonador.



En Protégé OWL la jerarquía construida de manera manual es llamada ‘**asserted hierarchy**’. La jerarquía de clases que es automáticamente calculada por el razonador es llamada ‘**inferred hierarchy**’.



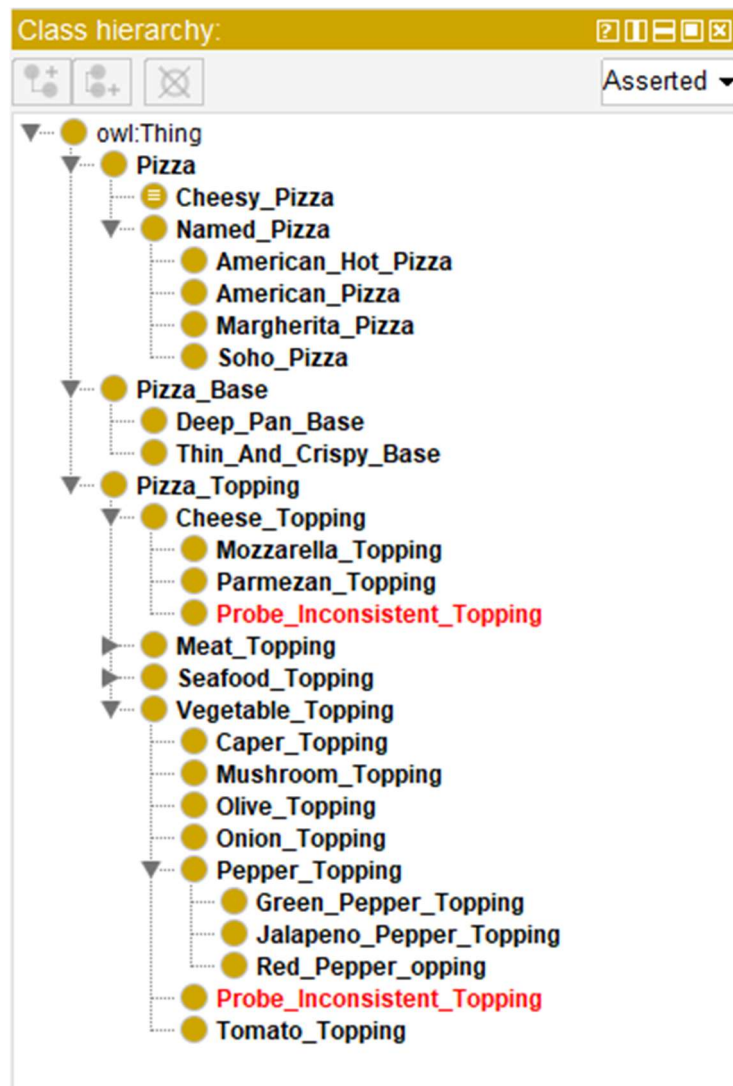
Las clases inconsistentes:

Para mostrar cómo el razonador encuentra clases inconsistentes en la ontología, vamos a adicionar una clase de prueba llamada ‘**Probe_Inconsistent_Topping**’, la cual va a servir para probar la integridad de nuestra ontología. Creemos la clase ‘**Probe_Inconsistent_Topping**’ como subclase de la clase ‘**Cheese_Topping**’ y de la clase ‘**Vegetable_Topping**’.

Entonces, para hacer eso, primero nos situamos sobre la clase **Cheese_Topping** y le creamos una subclase **Probe_Inconsistent_Topping**. Luego situados sobre la clase **Probe_Inconsistent_Topping** seleccionamos el “+” junto a la opción “*Subclass Of*” y seleccionamos la clase ‘**Vegetable_Topping**’.

Entonces aparece la clase **Probe_Inconsistent_Topping** como subclase de las dos clases: **Cheese_Topping** y **Vegetable_Topping**, las cuales son disjuntas. Podemos observar cómo aparece en rojo la clase ‘**Probe_Inconsistent_Topping**’, mostrando que hay una inconsistencia.

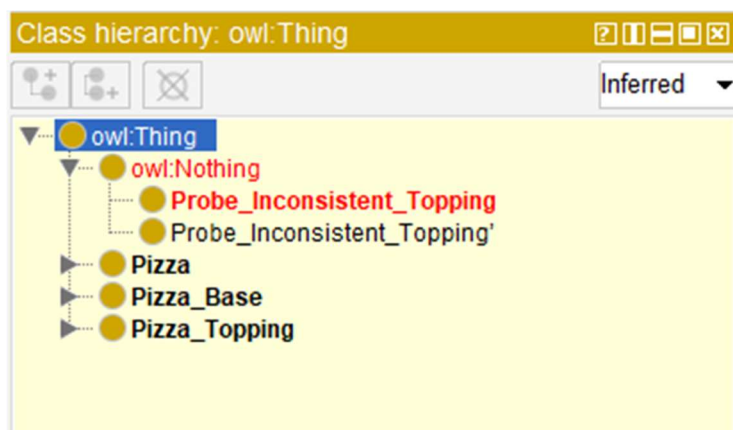
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Todos los individuos que son miembros de la clase **Probe_Inconsistent_Topping** son también necesariamente miembros de la clase **Cheese_Topping** y son también miembros de la clase **Vegetable_Topping**.

De manera intuitiva se diría que la clase **Probe_Inconsistent_Topping** es inconsistente porque una cosa no puede ser a la vez un **Cheese_Topping** y un **Vegetable_Topping**.

Además, la jerarquía inferida también muestra esa inconsistencia de la siguiente manera:



Para nosotros es evidente que una cosa no puede ser al mismo tiempo un **Cheese_Topping** y un **Vegetable_Topping**. Pero para el razonador, eso no es evidente. La verdadera razón por la cual

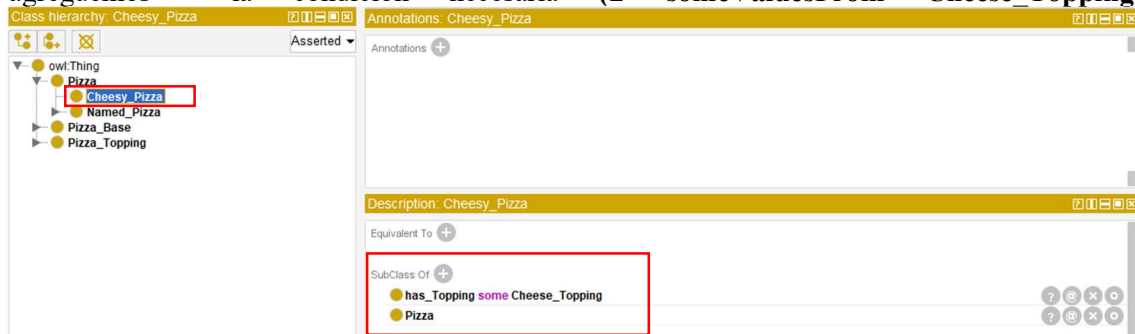
SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

el razonador detecta que la clase **Probe_Inconsistent_Topping** es inconsistente es que nosotros definimos que las clases **Cheese_Topping** y **Vegetable_Topping** eran disjuntas.

Las condiciones necesarias y suficientes (Clases primitivas y clases definidas)

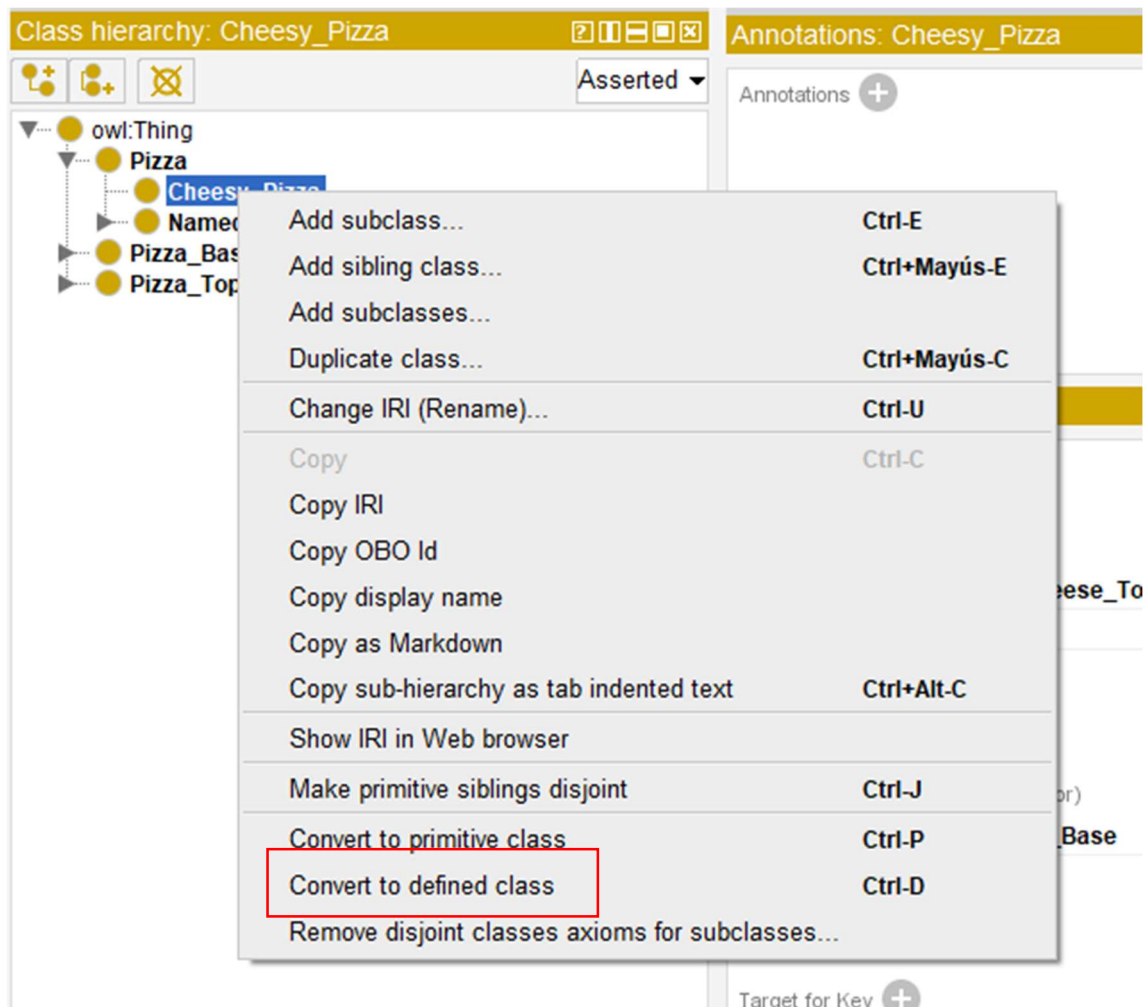
Hasta ahora hemos definido todas las clases con condiciones necesarias (opción Subclass Of +). Las condiciones necesarias se pueden leer como «si alguna cosa es miembro de esta clase, entonces es necesario que cumpla esas condiciones». Una clase que cumple solamente las condiciones necesarias es llamada **clase primitiva o clase parcial**.

Ilustremos eso con un ejemplo. Creemos una subclase **Cheesy_Pizza** de la clase **Pizza**. Y luego agreguemos la condición necesaria (\exists **someValuesFrom** **Cheese_Topping**)

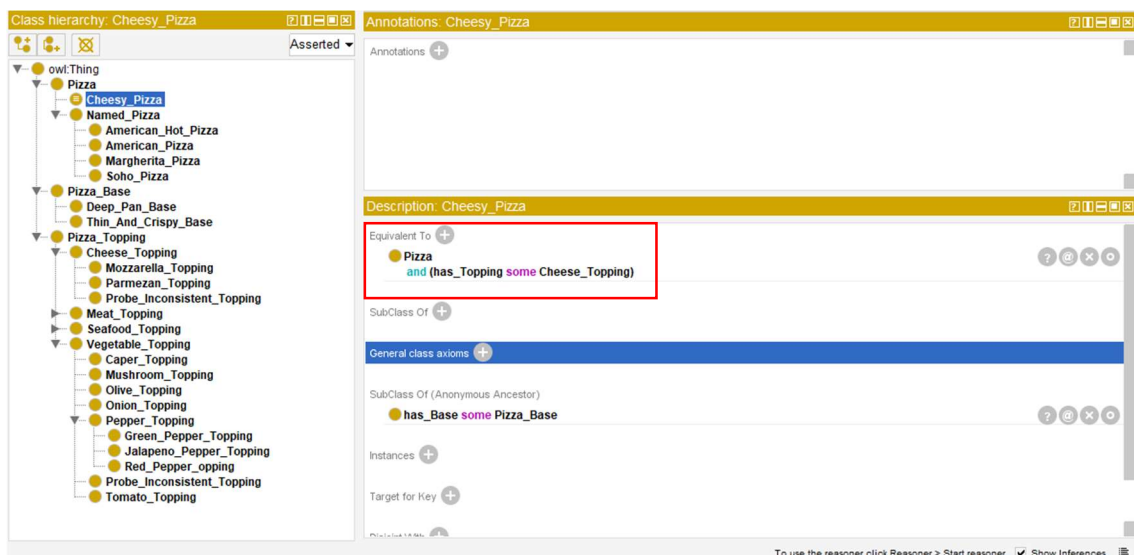


Así, nuestra descripción de la clase **Cheesy_Pizza** dice que si alguna cosa es un miembro de la clase **Cheesy_Pizza**, es necesario que sea miembro de la clase **Pizza** y es necesario que tenga al menos un topping que sea miembro de la clase **Cheese_Topping**. Nosotros utilizamos las condiciones necesarias para decir eso. Ahora, vamos a tomar un individuo aleatoriamente. Supongamos que sabemos que el individuo es miembro de la clase **Pizza** y que tiene al menos un topping de la clase **Cheese_Topping**. Sin embargo, el conocimiento que tenemos a partir de nuestra descripción de la clase **Cheesy_Pizza**, no es suficiente para determinar que el individuo es miembro de la clase **Cheesy_Pizza**. Para poder hacer esto, nosotros debemos cambiar las condiciones necesarias por condiciones necesarias y suficientes. Esto se hace dando click derecho sobre la clase y escogiendo la opción de “*Convert to defined class*”.

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



Entonces, apreciamos que la expresión que estaba en “SubClass of” pasa a la parte denominada “Equivalent to”, como puede apreciarse en la siguiente imagen:



NOTA: Una clase que tenga al menos una condición necesaria y suficiente es llamada **clase definida o clase completa**.

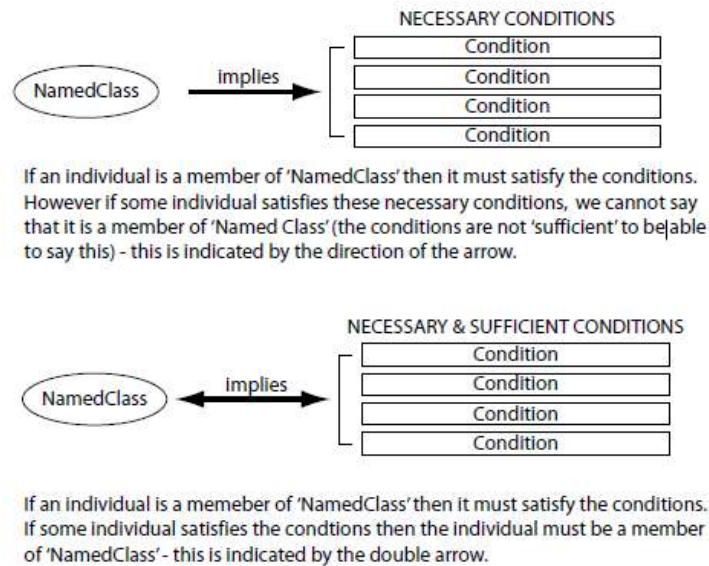
Entonces, hemos convertido una descripción de la clase Cheesy_Pizza en una definición.

NOTA: Definir dos condiciones NECESSARY & SUFFICIENT en lugar de una condición compuesta por dos expresiones, es una definición incorrecta de la clase Cheesy_Pizza, porque de

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB

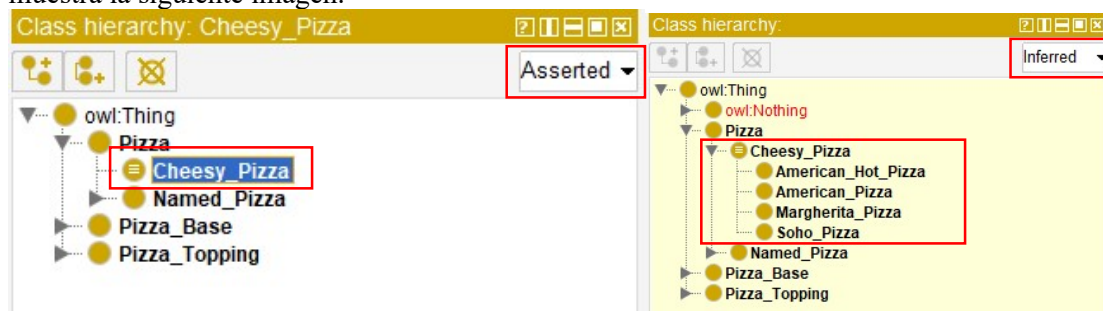
esta manera se estaría diciendo que **Cheesy_Pizza** es la unión de las condiciones necesarias y suficientes, y eso no es correcto.

En resumen, si la clase A es descrita utilizando condiciones necesarias, entonces podemos decir que, si un individuo es miembro de la clase A, él debe satisfacer las condiciones. NO podemos decir que un individuo cualquiera que satisfaga esas condiciones debe ser miembro de la clase A. Sin embargo, si la clase A es definida con condiciones necesarias y suficientes, se puede decir que un individuo cualquiera (aleatorio) que cumpla esas condiciones, debe ser miembro de la clase A. Así, las condiciones no solamente son condiciones necesarias para ser miembro de A, sino que ellas son suficientes para determinar que, si el individuo cumple esas condiciones, entonces él es miembro de A.



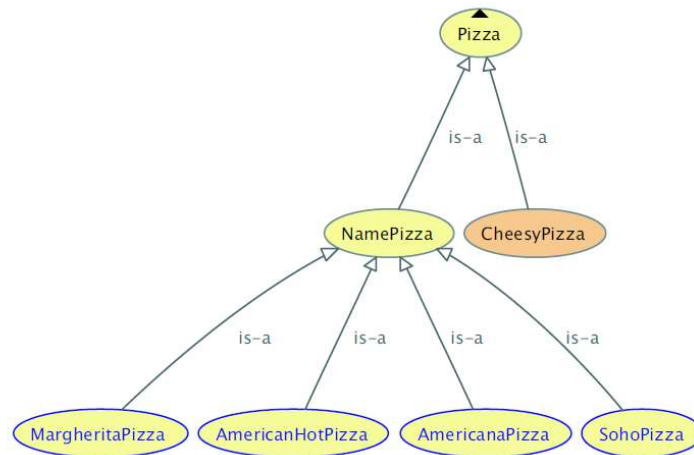
Clasificación automática

La ventaja más importante de tener una ontología OWL-DL es la capacidad de construir la taxonomía automáticamente a partir de las condiciones necesarias y suficientes. Es muy importante sobre todo cuando se trata de una ontología con millares de clases. Por ejemplo, si ejecutamos la opción de clasificar la ontología, se puede ver la « *inferred hierarchy* » como lo muestra la siguiente imagen:

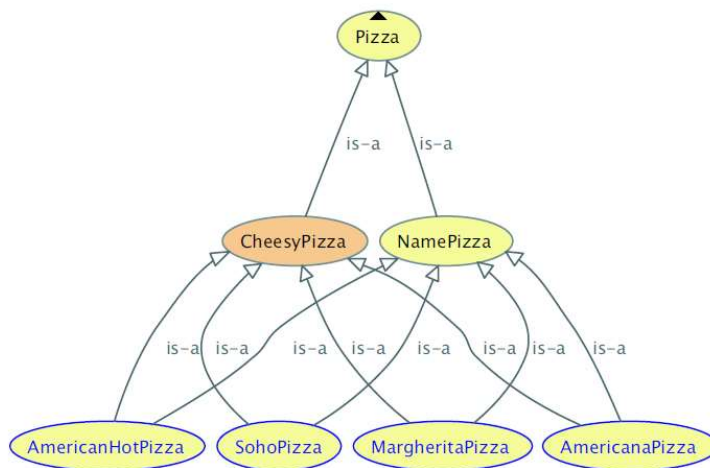


De manera esquemática, la jerarquía construida manualmente (**asserted hierarchy**) es:

SISTEMAS DE RECUPERACIÓN DE INFORMACIÓN EN LA WEB



La jerarquía inferida por el razonador es:



Referencias

- Tutorial-Ontologia-OWL-Pizzas-Español [Gloria Lucia Giraldo Gómez]
- Protégé OWL Tutorial (Pizza Ontology)
<http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/protg-owl-tutorial/>