Lab: Ontologías (I)

Sistemas Inteligentes Distribuidos

Sergio Alvarez
Javier Vázquez

Objetivos de la sesión

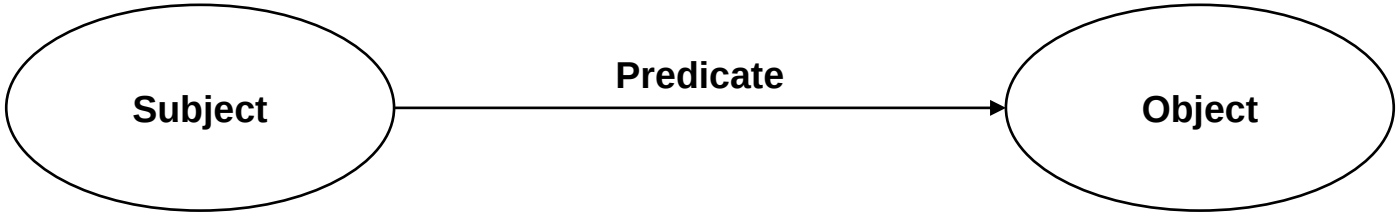
- Analizar ejemplos de cómo puede cambiar la axiomática soportada según la versión de OWL y cómo eso puede afectar al razonamiento ontológico
- Ver una propuesta de metodología de diseño de ontologías, viendo en cada fase qué partes de OWL son necesarias
- Aprender cómo aplicar todos estos conceptos en Protégé
- Enumerar los diferentes axiomas soportados en OWL, su sintaxis en Protégé y su correspondencia con la lógica descriptiva

OWL

Ontologías

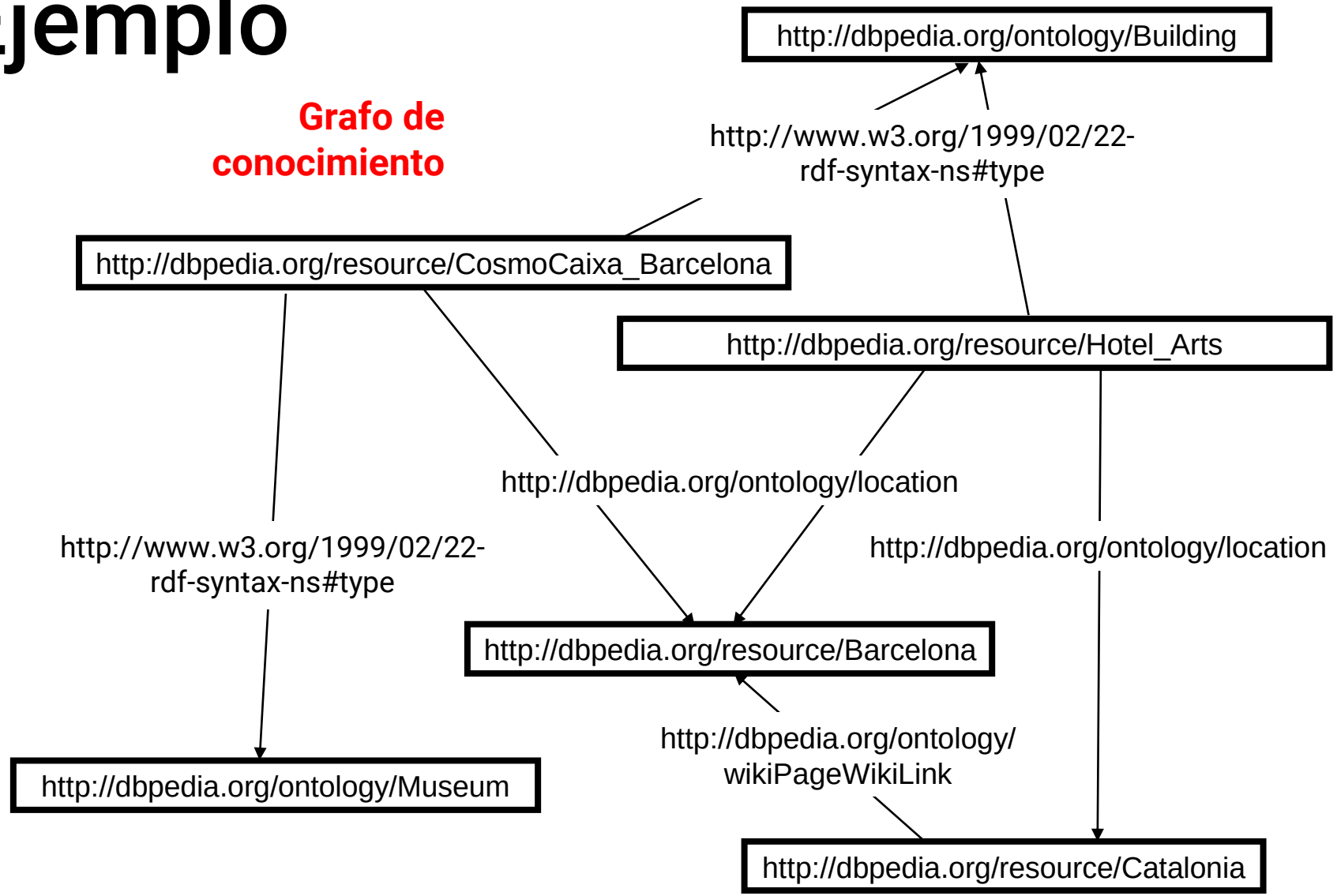
RDF

- Las ontologías y grafos de conocimiento (*knowledge graphs*) se suelen representar como **tripletas** (*triples*) **<sujeto, predicado, objeto>**



- El formato estándar más común es **RDF** (Resource Description Framework)
- En RDF, cada uno de los tres elementos es una URI, e.g.
  - **S: <http://dbpedia.org/resource/Tetris>**
  - **P: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>**
  - **O: <http://dbpedia.org/ontology/VideoGame>**

RDF: Ejemplo


Grafo de conocimiento
http://dbpedia.org/ontology/Building
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://dbpedia.org/resource/CosmoCaixa_Barcelona
http://dbpedia.org/resource/Hotel_Arts
http://dbpedia.org/ontology/location
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://dbpedia.org/ontology/location
http://dbpedia.org/resource/Barcelona
http://dbpedia.org/ontology/wikiPageWikiLink
http://dbpedia.org/ontology/Museum
http://dbpedia.org/resource/Catalonia

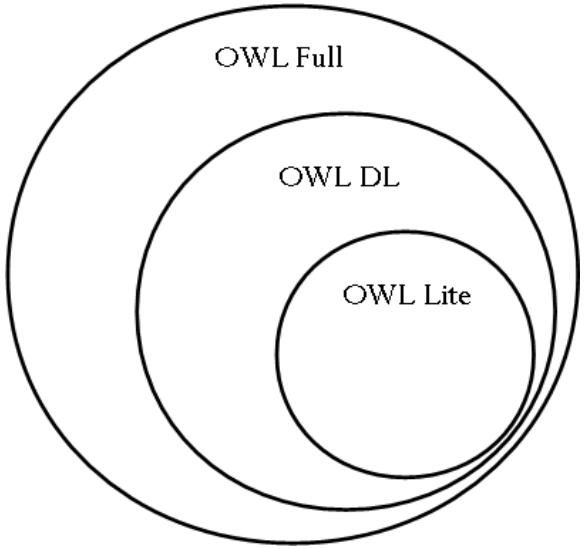
RDF Schema

- **Classes**
  - rdfs:Resource
  - rdfs:Class
  - rdfs:Literal
  - rdfs:Datatype
  - rdf:langString
  - rdf:HTML
  - rdf:XMLLiteral
  - rdf:Property
- **Properties**
  - rdfs:range
  - rdfs:domain
  - rdf:type
  - rdfs:subClassOf
  - rdfs:subPropertyOf
  - rdfs:label
  - rdfs:comment

https://www.w3.org/TR/rdf-schema

Las distintas versiones de OWL

- OWL-FULL (OWL 1)
  - Sin restricciones
  - Permite bucles en las relaciones
  - Algunas fórmulas pueden llevar tiempo infinito en resolverse
- OWL-DL (OWL 1) & OWL 2
  - Termino medio
  - Basado en lógica descriptiva
  - Contiene algunas restricciones
- OWL-LITE (OWL 1)
  - Básico
  - Taxonomía con restricciones simples
- Complejidad de la lógica descriptiva: http://www.cs.man.ac.uk/~ezolin/dl/



OWL Full

- `owl:Class = rdfs:Class`
  - Todos los documentos RDF son documentos OWL-FULL
- `owl:Thing = rdfs:Resource`
- `owl:ObjectProperty = rdf:Property`
  - `owl:DatatypeProperty` hereda de `owl:ObjectProperty`
- Un recurso puede ser a la vez clase e instancia
  - `Boeing-747` is an instance of the class `AirplaneType`
  - `airplane1` is an instance of the class `Boeing-747`
- Permite el uso de todas las restricciones definidas en OWL
- Sin restricciones sobre los axiomas (pueden ser inconsistentes)

OWL Full

https://www.w3.org/TR/owl-ref

owl:allValuesFrom
owl:SomeValuesFrom
owl:hasValue
owl:maxCardinality
owl:minCardinality
owl:cardinality
owl:intersectionOf
owl:unionOf
owl:complementOf
rdfs:subClassOf
owl:equivalentClass
owl:disjointWith

rdfs:subPropertyOf
rdfs:domain
rdfs:range
owl:equivalentProperty
owl:inverseOf
owl:FunctionalProperty
owl:InverseFunctionalProperty
owl:TransitiveProperty
owl:SymmetricProperty
owl:sameAs
owl:differentFrom
owl:AllDifferent

OWL DL

- Permite todos los tipos de restricciones, con limitaciones
  - Restricciones numéricas no soportadas en propiedades transitivas
  - `owl:DatatypeProperty` no hereda de `owl:ObjectProperty`
    - Propiedades como `inverseOf`, `SimmetricProperty` o `TransitiveProperty` no se pueden aplicar a `DatatypeProperties`
  - Restricciones en las anotaciones (annotation property)
    - Sólo soporta datos, literales, URI o Instancias
    - Una annotation property no puede tener sub-propiedades
- Los axiomas definidos han de ser jerárquicos, correctos y consistentes (e.g. no depender de clases no declaradas)
  - Los axiomas sobre igualdad o diferencia han de referirse a las instancias (*individuals*)

OWL Lite

- Restricciones de cardinalidad `0..1`
- No permite algunas restricciones importantes como `disjointWith`
- Casi tan complejo como OWL-DL/OWL 2 de modo que apenas se usa

Ciclo de desarrollo de una Ontología

Ontologías

Ejercicio: PizzaOntology I

- Descargad la ontología:
  - https://raw.githubusercontent.com/owlcs/pizza-ontology/master/pizza.owl
  - Aseguraos que se guarda con nombre de fichero `pizza.owl`, sin extensión `.html` o `.txt` (si hace falta, renombradlo)
  - Si no podéis guardarlo, copiad todo el xml y guardadlo a mano
- Cargad la ontología en Protégé
- Añadid algún comentario (Annotations)

- Cuál es el dominio que intentamos cubrir con la Ontología
  - Problema de la granularidad
  - Problema de la omnisciencia
- Para qué vamos a usar la Ontología
- Identificar las *Competency Questions*
  - Para qué tipos de preguntas debe darnos respuesta la información que contiene la Ontología
- Las decisiones no son finales, pueden cambiar durante el ciclo de desarrollo de la Ontología

- ¿Por qué?
  - Eficiencia, menor coste de desarrollo
  - Integración directa con sistemas que usen esa Ontología
  - Uso de Ontologías que han sido validadas en casos de uso prácticos (aplicaciones)

https://lod-cloud.net

- Cuáles son los términos sobre los que vamos a hablar
- Cuáles son las propiedades de esos términos
- Qué queremos decir sobre esos términos

- Primer paso:
  - No organizar los términos, hacer una lista con lo que queremos incluir en la Ontología

- Definir las clases y la taxonomía
  - Una clase es un concepto del dominio, no un objeto
    - ¡No sólo entidades, pueden ser propiedades!
  - Una clase es un conjunto de elementos con propiedades similares
    - ¿Y qué es cada elemento entonces?
- La taxonomía es la jerarquía de clases
  - ¿Cuándo agrupamos dos clases en la misma superclase?
    - La respuesta la tenéis en esta transparencia

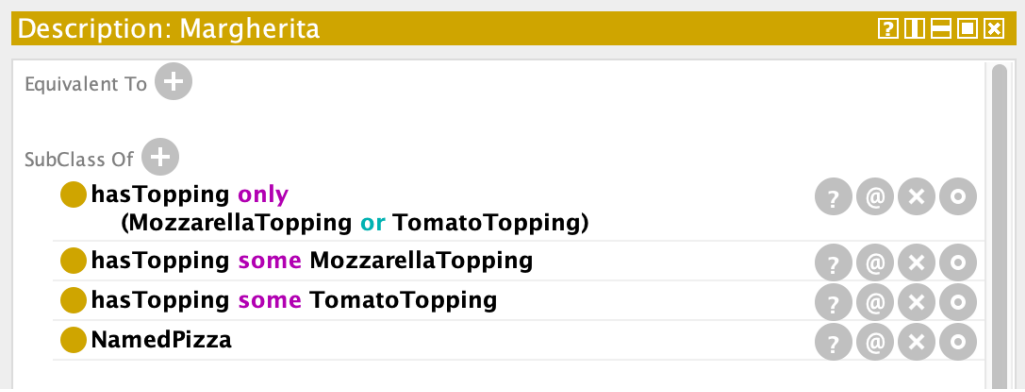
- Definir las clases y la taxonomía
  - Top-Down: Definir primero los conceptos más generales, y luego especializar
    - ¿Y cuál es el concepto más general?
- Bottom-up: Definir los conceptos más específicos y luego agruparlos en clases más generales
  - ¿Y cuándo los agrupamos?
- Combinación: Definir los conceptos más importantes y luego generalizarlos y especificarlos en paralelo
  - Útil si aplicamos otra de las dos técnicas y nos quedamos atascados
- Clases disjuntas
  - Una instancia no puede pertenecer a ambas clases a la vez

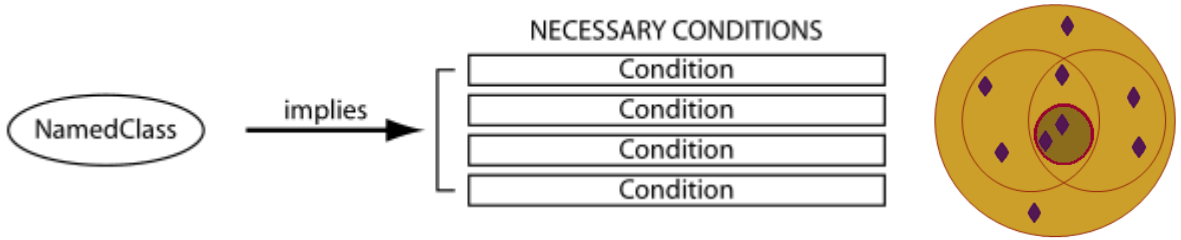
Ejercicio: PizzaOntology II

- Cread una subclase de `Thing` (Entities)
- Cread una clase hermana (sibling) y una subclase de esta clase
- Cread otra hermana
- Borrad la segunda hermana
- Haced que la clase y su hermana sean disjuntas (Disjoint With)
  - ¿Hace falta hacerlo para las dos?

Tipos de clases: Primitiva

- **Condiciones necesarias (inclusión)**
  - Si algo reúne las condiciones no es necesariamente obligatorio que sea un miembro de la clase
  - **PERO:** Un elemento escogido al azar que sabemos que es miembro de la clase, sabemos que reúne las condiciones
  - ¿Qué ocurre con un elemento escogido al azar que sabemos que reúne las condiciones?

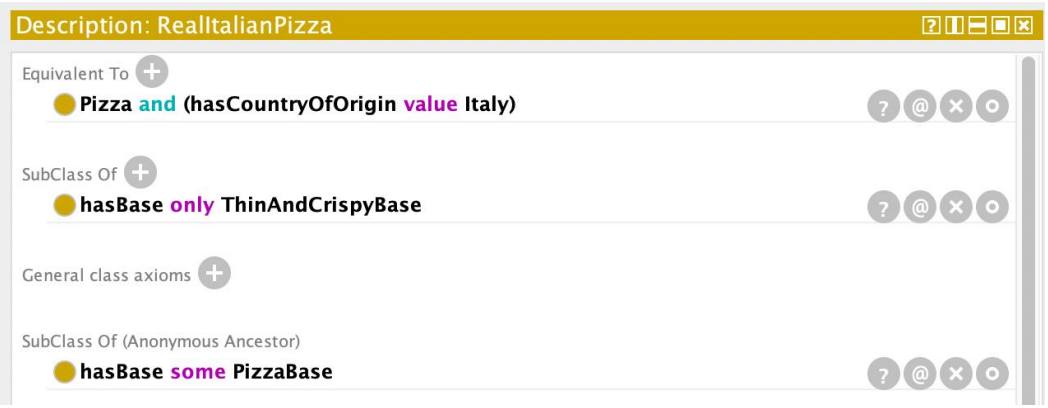


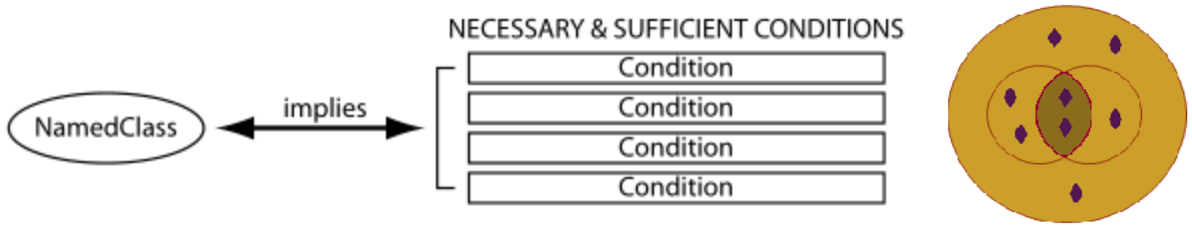


$$\mathbf{MargheritaPizza} \sqsubseteq \mathbf{NamedPizza} \sqcap \forall \mathbf{hasTopping}.\{\mathbf{MozzarellaTopping}, \mathbf{TomatoTopping}\} \sqcap \exists \mathbf{hasTopping}.\mathbf{MozzarellaTopping} \sqcap \exists \mathbf{hasTopping}.\mathbf{TomatoTopping}$$

Tipos de clases: Equivalente

- **Condiciones necesarias y suficientes (equivalencia)**
  - Si algo reúne las condiciones es suficiente para decir que es un miembro de la clase
  - Un elemento escogido al azar que sabemos que es miembro de la clase, sabemos que reúne las condiciones
  - ¿Qué ocurre con un elemento escogido al azar que sabemos que reúne las condiciones?





$$\mathbf{RealItalianPizza} \equiv \mathbf{Pizza} \sqcap \exists \mathbf{hasCountryOfOrigin}.\{\mathbf{Italy}\}$$

$$\mathrm{RealItalianPizza} \sqsubseteq \forall \mathrm{hasBase}.\,\mathrm{ThinAndCrispyBase}$$

Ejercicio: PizzaOntology III

- Observad la diferencia entre `NamedPizza` y `RealItalianPizza`
  - ¿Cuál es primitiva y cuál equivalente?
  - ¿Cuáles son las condiciones suficientes para que una pizza sea `RealItalianPizza`?
  - ¿Qué condiciones necesarias añade a las suficientes?

- Asociadas a la clases (Dominio-Rango):
  - Si el rango **no** es una clase:
    - **Data Properties**
  - Si el rango es una clase:
    - Relaciones a otras instancias de la clase
    - **Object Properties**

- Las restricciones definen el conjunto de valores posibles para una propiedad
- Las restricciones más comunes son:
  - Dominio
  - Rango
- En realidad, no son restricciones de tipos concretos a comprobar con métodos específicos: se reducen a **axiomas**

Ejercicio: PizzaOntology IV

- Arrancad el Reasoner incluído en Protégé
  - Se puede configurar para ampliar/acotar el ámbito de lógica
- Analizad inconsistencias
  - Buscad las inconsistencias (en rojo)
  - ¿A qué se deben estas inconsistencias?
  - Usad el símbolo de interrogación para obtener explicaciones
- Analizad la clasificación del razonador
  - Buscad las inferencias de clasificación (en amarillo)
  - Usad el símbolo de interrogación para obtener explicaciones
- Parad el Reasoner

Ejercicio: PizzaOntology V

- Cread una `ObjectProperty` para expresar en qué país se vende
  - Asignad dominio y rango
- Cread una subpropiedad de `hasIngredient` para poder representar el relleno del borde
  - Asignad dominio y rango
  - Asignad alguna restricción, como por ejemplo `inverse of`
- Cread una `DataProperty` para poder representar el precio
  - Asignad dominio y rango

- Crear instancias de las clases
  - La clase se convierte en un tipo directo de la instancia
  - Las superclases del tipo directo son tipos de la instancia
- Asignar valores a las propiedades
  - Los valores asignados deben cumplir las restricciones impuestas
  - Se puede usar razonadores para comprobar que las restricciones se cumplan

Ejercicio: PizzaOntology VI

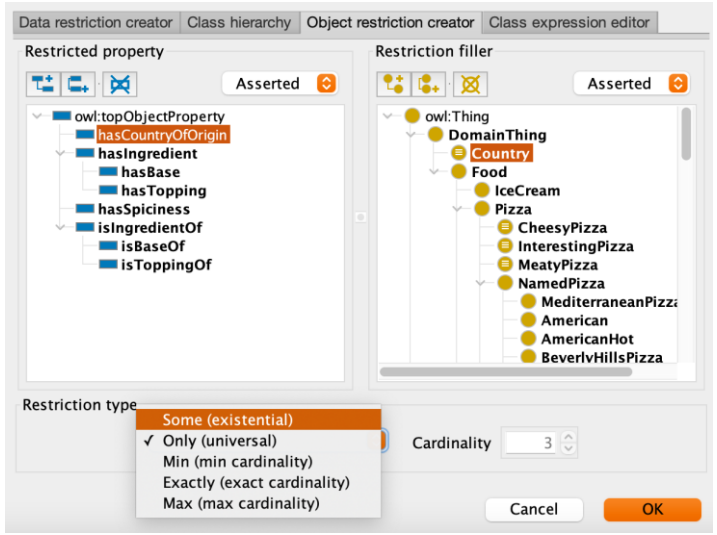
- Cread una instancia de `DeepPanBase` (Individuals)
- Cread una instancia de `Pizza` con las siguientes propiedades:
  - Tiene como país de origen `Italy`
  - Tiene como base la instancia de `DeepPanBase` que habéis creado
- En el menú, arrancad el Reasoner
  - ¿Es inconsistente? ¿Por qué?
- Borrad estas instancias y volved a sincronizar el Reasoner

Axiomas

Ontologías

Interfaz de Protégé

- Algunos axiomas están disponibles en el editor de Protégé



- Guía para editar en texto libre (Manchester OWL Syntax):
  - http://protegeproject.github.io/protege/class-expression-syntax/

Restricciones: axiomas de clase

- Enumeraciones de individuos: permiten definir una clase por su conjunto de instancias, sin necesidad de nombrarla
  - En OWL: **`oneOf`**
  - En Manchester OWL: **{x_1, …, x_n}**
  - En DL: $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\}$
- Clases excluyentes entre sí: indican cuando una instancia no puede pertenecer a dos clases a la vez
  - En OWL: **`disjointWith`**
  - En Manchester OWL: **`not (C and D)`**
  - En DL: $\boldsymbol{C} \sqsubseteq \neg \boldsymbol{D}$

Restricciones: axiomas de clase

- Clases equivalentes entre si
  - En OWL: **`sameClassAs`**
  - Sin correspondencia en DL, ¡no confundir con `equivalentClass` ($C \equiv D$)!
  - Todas las instancias de C son instancias de D: útil para enlazar ontologías
  - No todos los razonadores realizan la inferencia (ineficiente)

Restricciones de cuantificación

- Existencial: instancias que tienen una relación P con al menos una instancia de la clase C
  - En OWL: **`someValuesFrom`**
  - En Manchester OWL: `P` **`some`** `C`
  - En DL: $\exists \boldsymbol{P}.\boldsymbol{C}$
- Universal: no es necesario que las instancias estén relacionadas por P, pero si lo están, lo están con instancias de C
  - En OWL: **`allValuesFrom`**
  - En Manchester OWL: `P` **`only`** `C`
  - En DL: $\forall \boldsymbol{P}.\boldsymbol{C}$

Ejercicio: PizzaOntology VII

- Cread un topping `TurtleTopping`
- Cread una `Pizza` llamada `SuperMarioPizza` con condiciones necesarias y suficientes: tener como ingredientes champiñones y tortugas
- Cread una instancia de una pizza de tipo `Pizza` con ingredientes instancias de champiñones y tortugas
- Sincronizad el Razonador y observad cómo se clasifica
- Cambiad `SuperMarioPizza` para que las condiciones sean sólo necesarias y observad la diferencia tras sincronizar

Restricciones cualificadas de cardinalidad

- Mínimo número de relaciones
  - En OWL: **`minCardinality`**
  - En Manchester OWL: P **min** n C
  - En DL: $\geq \boldsymbol{nP.C}$
- Máximo número de relaciones
  - En OWL: **`maxCardinality`**
  - En Manchester OWL: P **max** n C
  - En DL: $\leq \boldsymbol{nP.C}$
- Número exacto de relaciones
  - En OWL: **`cardinality`**
  - En Manchester OWL: P **exactly** n C
  - En DL: $\leq \boldsymbol{nP.C} \sqcup \geq \boldsymbol{nP.C}$

Restricciones cualificadas por valor

- Restricción del tipo de valor: enumeraciones de instancias
  - En OWL: **`someValuesFrom/allValuesFrom`** combinados con **`oneOf`**
  - En Manchester OWL: `P` **`value`** $\{x_1,\ldots,x_n\}$
  - En DL: $\exists P.\{x_1,\ldots,x_n\}$ o $\forall P.\{x_1,\ldots,x_n\}$
  - Por ejemplo:

Description logics:

$$\text{PizzaEscandinava} \equiv \text{Pizza} \sqcap \text{hasCountryOrigin}.\{\text{Denmark}, \text{Norway}, \text{Sweden}\}$$

Manchester OWL Syntax:

```
Pizza and (hasCountryOfOrigin value {Denmark, Norway, Sweden})
```

Restricciones: operaciones de conjuntos

- Intersección de dos clases
  - En OWL: **intersectionOf**
  - En Manchester OWL: C **and** D
  - En DL: $\boldsymbol{C} \sqcap \boldsymbol{D}$
- Unión de dos clases
  - En OWL: **unionOf**
  - En Manchester OWL: C **or** D
  - En DL: $\boldsymbol{C} \sqcup \boldsymbol{D}$
- Complemento de una clase: todas las instancias que no pertenecen a esa clase
  - En OWL: **complementOf**
  - En Manchester OWL: **not** C
  - En DL: $\neg \boldsymbol{C}$

Ejercicio: PizzaOntology VIII

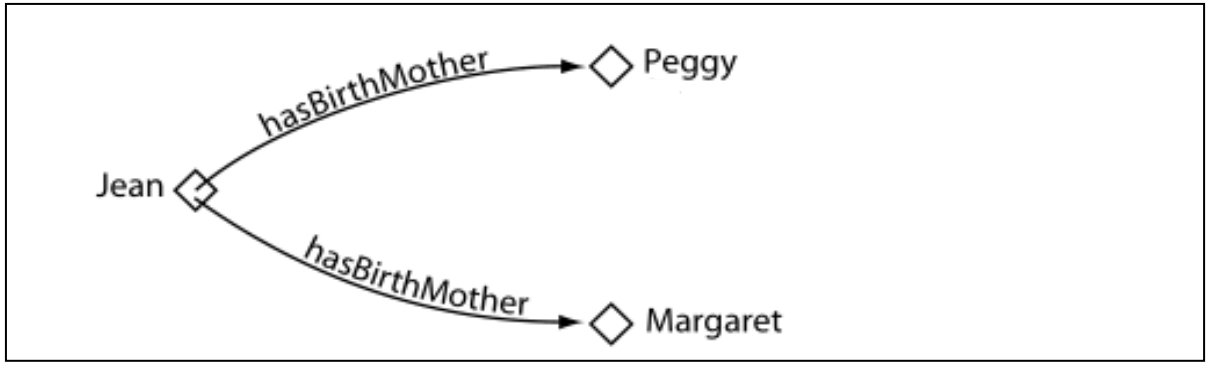
- Cread las siguientes pizzas:
  - Pizza con marisco: contiene como mínimo marisco
  - Pizza de marisco: todos los ingredientes son de marisco
  - Pizza ecléctica: mínimo 10 ingredientes
  - Pizza de oferta: máximo 2 ingredientes
  - Pizza binaria: exactamente 2 ingredientes
  - Pizza triqueso: exactamente 3 ingredientes, todos de queso
  - Pizza escandinava
    - Tendréis que editar en texto libre (`Class Expression Editor`) y posiblemente también editar la clase `Country`
    - Utilizad **or** y **value** (tenéis ejemplos en `American` y en la guía)
  - Pizza aburrida especial: pizzas que no sean `InterestingPizza`, pero que estén en la unión entre las `MeatyPizza` y las `CheeseyPizza`

Propiedades

- Propiedades y subpropiedades
  - ¿Cuándo agrupamos propiedades en subpropiedades?
  - En OWL: **subPropertyOf**
  - En DL: $\boldsymbol{P_1} \sqsubseteq \boldsymbol{P_2}$
- Domino y rango de la propiedad
  - ¡Recordad que las propiedades son tratadas como axiomas!
  - En OWL: **rdfs:domain**, **rdfs:range**
  - En DL: $\exists \boldsymbol{P}.\top \sqsubseteq \boldsymbol{C}$ (dominio), $\top \sqsubseteq \forall \boldsymbol{P}.\boldsymbol{C}$ (rango)
- Propiedad Inversa, del tipo `hasComponent` vs `isComponentOf`
  - El dominio y el rango se intercambian
  - En OWL: **inverseOf**
  - En DL: $\boldsymbol{P_1} \equiv \boldsymbol{P_2^-}$

Propiedades

- Propiedad funcional: Cuando C y D están relacionados mediante una propiedad funcional sólo una instancia de D puede estar relacionada con cada instancia de C
  - En OWL: **FunctionalProperty**
  - En DL: $\top \sqsubseteq \leq 1P$
  - ¿Qué ocurre si más de una instancia de D está relacionada con la misma instancia de C?



Propiedades

- Propiedad funcional inversa: sólo puede haber una instancia de C para cada instancia de D
  - Por ejemplo, el número de serie de un portátil
  - En OWL: **InverseFunctionalProperty**
  - En DL: $\top \sqsubseteq \leq \mathbf{1}\boldsymbol{P}^{-}$
- Propiedad transitiva: si una instancia de A se relaciona con una de B y una de B con una de C, la instancia de A se relaciona con la de C
  - En OWL: **TransitiveProperty**
  - En DL: $\boldsymbol{P}^{+} \sqsubseteq \boldsymbol{P}$, e.g. $\mathrm{descendant}^{+} \sqsubseteq \mathrm{descendant}$
  - En DL (sintaxis equivalente con el operador *composición* $\circ$): $\boldsymbol{P} \circ \boldsymbol{P} \sqsubseteq \boldsymbol{P}$

Propiedades

- Propiedad simétrica: si una instancia de C se relaciona con una de D, la de D se relaciona con la de C
  - En OWL: **SymmetricProperty**
  - En DL: $\boldsymbol{P} \equiv \boldsymbol{P}^{-}$
- Propiedad asimétrica (antisimétrica): si una instancia de C se relaciona con una de D, la de D no se puede relacionar con la de C
  - En OWL: **AsymmetricProperty**
  - En DL: $\boldsymbol{P} \sqsubseteq \neg \boldsymbol{P}^{-}$ (la propiedad es disjunta con respecto de su inversa)
  - ¿Y qué pasa si se relaciona?

Propiedades

- Propiedad reflexiva: si P es reflexiva y una instancia de C se relaciona por P, esa relación es con esa misma instancia de C
  - En OWL: **ReflexiveProperty**
  - En DL: $\top \sqsubseteq \exists \boldsymbol{P}.\boldsymbol{Self}$
- Propiedad irreflexiva: si P es reflexiva y una instancia de C se relaciona por P, P esa relación no puede ser con esa misma instancia de C
  - ¿Y qué pasa si se aplica?
  - En OWL: **IrreflexiveProperty**
  - En DL: $\top \sqsubseteq \neg\exists \boldsymbol{P}.\boldsymbol{Self}$

Ejercicio: PizzaOntology IX

- Cread una propiedad funcional que asigne un creador a una `NamedPizza` (tendréis que crear clases)
  - Asignad dos creadores a una instancia de `NamedPizza`
  - Sincronizad el Razonador
  - ¿Qué inferencia ha hecho?
  - Identificad los creadores como Different Individuals y resincronizad
- Cread una propiedad transitiva que permita representar que la creación de una `NamedPizza` está influenciada por otra
- Cread una propiedad simétrica que permita expresar que dos ingredientes combinan bien

Referencias

- DAML
  - http://www.daml.org/
- OIL
  - http://www.cs.vu.nl/~frankh/postscript/IEEE-IS01.pdf
- RDF
  - http://www.xml.com/pub/a/2001/01/24/rdf.html
- OWL
  - http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial
- Protégé
  - http://protege.stanford.edu
- Ontologia de pizzas
  - www.co-ode.org/ontologies/pizza/