Práctica 3: Ontologías

Sistemas de Información Distribuidos

Pedro Allué Tamargo (758267) Juan José Tambo Tambo (755742)

16 de marzo de 2021

Índice

1. Esfuerzos

- **Pedro**:
  - **Tiempo invertido:** 3:30 horas.
- **Juan José**:
  - **Tiempo invertido:** 3:30 horas.

2. Ejercicio 1

Para el desarrollo del primer ejercicio de la práctica, se ha utilizado la herramienta *Protegé*, mediante la cual se ha establecido la ontología indicada en el guión. Siguiendo paso a paso cada uno de los puntos que se indican en la práctica, se han creado los siguientes individuos, clases y propiedades de objeto y sus respectivas relaciones de herencia:

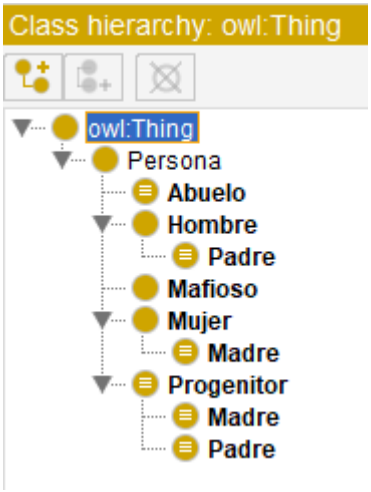

Figura 1: Estructura de las clases

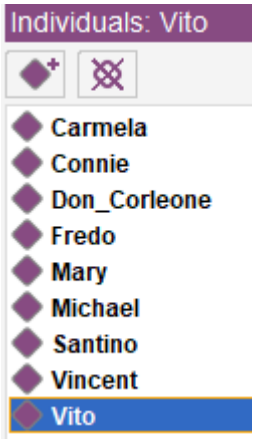

Figura 2: Estructura de los individuos

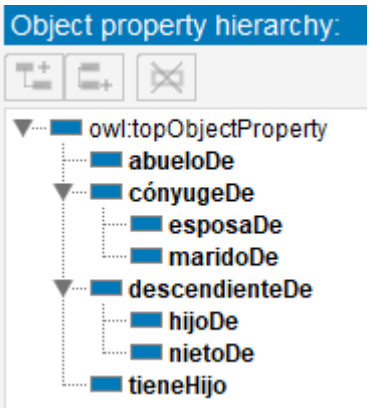

Figura 3: Estructura de las propiedades

Para las acciones como hacer clases disjuntas, asignar una clase a un individuo, hacer dos individuos idénticos (o diferentes) entre sí o hacer propiedades inversas, no se va a entrar en detalles de su realización ya que son acciones básicas que la herramienta permite realizar sin ninguna complicación.

En lo relacionado a las clases, cabe destacar que para cumplir con los apartados `t, u` relacionados con *Padre* y *Madre*, se debe hacer que estas clases estén relacionadas con la clase *Progenitor* (de la cual son subclases directas) y con las clases *Hombre* y *Mujer* respectivamente. Para ello, para cada una de estas clases se debe indicar la relación `[Hombre|Mujer] and Progenitor` en el apartado *Equivalent To*[1].

En la Figura 1, aparecen como subclases de *Hombre* y *Mujer* pero esto se debe a la relación de equivalencia especificada. Además, para que se cumpla el apartado `s`, desde el apartado *Equivalent To* en la clase *Progenitor*, se debe añadir `tieneHijo some Persona`. De esta manera se indica que para pertenecer a esta clase, debe relacionarse con alguna persona mediante esta propiedad. Las clases *Madre* y *Padre* heredan este comportamiento.

En cuanto a las *Object properties*, caben destacar *maridoDe* y *esposaDe*, las cuales además de ser inversas entre sí, se ha asignado un dominio para poder satisfacer los apartados `p` y `q`. Para *maridoDe* se ha indicado que el dominio es la clase *Hombre*, mientras que para *mujerDe* se ha asignado *Mujer*. De esta manera se restringe a que solo pueden tener esta propiedad los individuos pertenecientes a estas clases.

Por último, a los individuos se les han asignado las clases y las propiedades indicadas en el guión, de tal manera que hasta el momento solo son utilizadas las clases *Hombre*, *Mujer* y *Mafioso* y las propiedades *hijoDe*, *maridoDe*

[1] *Hombre* para la clase *Padre* y *Mujer* para *Madre*

y *mujerDe*.

Una vez se tiene configurado, se utiliza el razonador *HermiT* desde `Reasoner` → `HermiT` → `Start reasoner` para que calcule todas las inferencias. Al utilizarlo, se observa que se asignan correctamente a los individuos las clases y propiedades no especificadas explícitamente de forma previa. Por ejemplo, *Vito* previamente pertenecía solo a la clase *Hombre* y la única propiedad que tenía era *maridoDe Carmela*. Tras el razonador, pertenece a las clases *Hombre, Abuelo, Mafioso y Padre* y tiene las siguientes relaciones:

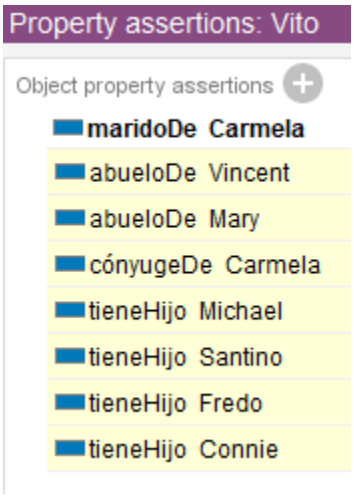

Figura 4: Propiedades de *Vito*

3. Ejercicio 2

Para el desarrollo de este ejercicio se han descargado las librerías (ficheros *JAR*) propuestas en el enunciado y se han incluido en el directorio `lib` del proyecto.
Para llevar acabo las tareas propuestas en el enunciado se ha utilizado la plantilla adjunta al mismo.

Para que el programa en *Java* leyera correctamente el archivo `owl` generado desde *Protegé*, se debe extraer la ontología desde esta herramienta con `File` → `Gather Ontologies` y seleccionar el formato RDF/XML Syntax o OWL/XML Syntax, ya que, en caso contrario aparecen errores a la hora de leer el archivo como "*inconsistent ontology*".

Para añadir individuos a la ontología con la clase *Mafioso* se debe obtener la clase con el método `getOWLClass` y crear los individuos correspondientes mediante el método `getOWLNamedIndividual`. Seguidamente, para indicar que el individuo pertenece a esa clase, se crea un axioma mediante `getOWLClassAssertionAxiom` y se añade a la ontología. Finalmente, para que esta relación se refleje en la ontología resultante, se debe llamar al método `applyChange` del *manager*.

Para obtener todos los individuos que pertenecen a una clase determinada (*Hombre*, *Mafioso*) se debe obtener la clase correspondiente mediante `getOWLClass` y se utiliza el razonador con su método `getInstances` el cual devuelve todos los individuos pertenecientes a esa clase.

En el caso de las consultas que involucran propiedades (*esposa de Vito*, *descendientes de Vito*, *padre de Michael* y *abuela de Vincent*) se ha seguido el mismo patrón. Para las dos primeras, se obtiene la propiedad que interesa para la consulta y el individuo correspondiente. Seguidamente, mediante el método `getOWLObjectHasValue()`, se obtiene un objeto perteneciente a la clase `OWLClassExpression` que contiene todas las relaciones que tiene el individuo indicado con esa propiedad.

A la hora de obtener *el padre de Michael* se añade una restricción adicional ya que la propiedad inversa de *hijoDe* es *tieneHijo* y por lo tanto hay que concretar que se ha pedido un padre (individuo perteneciente a la clase

Padre). Por lo tanto se ha utilizado la clase `OWLDataFactory` con el método `getOWLObjectIntersectionOf()` que devuelve una expresión con la intersección de las dos expresiones pasadas como parámetros. Estas expresiones son las relaciones que se obtienen con *Michael* y la propiedad *tieneHijo* junto a la clase padre, indicando así que solo interesan las relaciones cuyo objeto pertenezca a las clase *Padre*. Además es importante tener en cuenta que las instancias de `OWLClass` son instancias de `OWLCLassExpression` y por lo tanto simplifica el proceso de obtener los individuos pertenecientes a una clase dada usando el método `getInstances()` del razonador.

Por último, para la consulta de *obtener la abuela de Vincent*, se intentó seguir la misma lógica que la consulta anterior pero no se obtenía ningún resultado. Esto es debido a que esta consulta se realiza sobre *factory*, es decir, aún no se ha aplicado el razonador, por lo que las relaciones *abueloDe* aún no aparecen aplicadas a ningún individuo. Por ello, se ha realizado una consulta muy parecida a la anterior pero utilizando métodos del razonador. De esta manera, para obtener todos los abuelos de *Vincent*, se ha utilizado el método `getObjectPropertyValues()`, el cual funciona de forma parecida al `getOWLObjectHasValue()` de *factory* con la diferencia que devuelve directamente el conjunto de los individuos en vez de un `OWLClassExpression`. Cabe destacar que la propiedad usada ha sido *abueloDe* y se ha obtenido su inversa (que es la que interesa con respecto a *Vincent*) mediante `getInverseProperty()`. Se podría haber utilizado directamente la propiedad *nietoDe* pero se ha decidido hacer de esta forma.

Hasta el momento se habrían obtenido todos los abuelos de *Vincent*, pero interesa la abuela en concreto. Se ha intentado hacer una intersección como en la consulta anterior indicando que los resultados tenían que pertenecer a la clase *Mujer* pero no se ha encontrado ningún método, por lo que para cada uno de sus abuelos, se comprueba si pertenece a la clase *Mujer* mediante `esMujer.getIndividuals(ont).contains(individual)`[2]. y se muestra por pantalla en caso afirmativo.

4. Urls de interés

- Documentación *Javadoc* de OWLAPI (*enlace*)

[2]esMujer hacer referencia a la clase *Mujer*