PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
MAESTRÍA EN INFORMÁTICA

INGENIERÍA Y GESTIÓN DEL CONOCIMIENTO
Laboratorio 2 - RDFS
(Segundo Semestre del 2020)

Objetivos de aprendizaje:

- Manipular estructuras RDFS.
- Ejecutar inferencias en modelos RDF.

---

1. Introducción

En este laboratorio se busca que el alumno manipule estructuras RDFS para resolver problemas de la Ingeniería del Conocimiento. Se busca que el alumno pueda crear, salvar, leer, consultar y ejecutar inferencias en modelos RDFS usando un lenguaje de programación, aplicando de esta manera los conceptos vistos en la sesión teórica.

2. Métodos y procedimientos

Se utilizarán las siguientes herramientas:

- Como lenguaje de programación se utilizará Java.
- Como Entorno de Desarrollo Integrado (IDE) se utilizará Netbeans 8.2 el cual puede ser descargado a partir de la URL `https://netbeans.org/downloads/8.2/`.
- Para la manipulación del RDF y RDFS se utilizará el API de Apache Jena, el cual se encuentra en la URL `https://jena.apache.org/`. El alumno podrá encontrar un tutorial de RDF usando Jena en la URL `https://jena.apache.org/tutorials/rdf_api.html`.

Parte I
Inferencia en RDFS - `subclassOf`

3. Creación de esquema RDFS

Para probar la inferencia en RDFS, se creará primero un esquema como el que se presenta en la figura 1, el cual ha sido tomado del texto guía. Para facilitar la creación de los recursos se ha implementado un método en Java que facilita esta tarea. Este método se encuentra en el programa 1.

Programa 1: Creando recursos en RDF en Jena

```
1 ...
2 public class Ejemplo01 {
3
4     private static Resource crearRecurso(String uri, String id, Model model) {
5         return model.createResource(uri + id);
6     }
7 ...
```

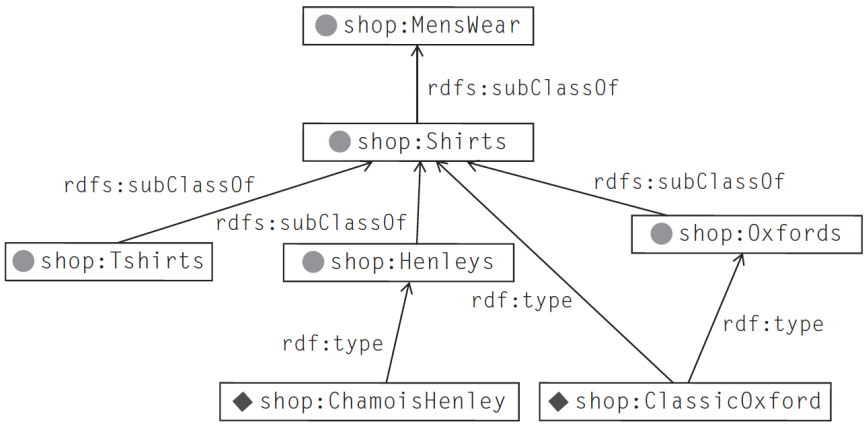

Figura 1: Estructura RDF.

Usremos este método para crear las diferentes recursos que conforman el esquema. En el programa 2 se puede apreciar la creación de todos los recursos. Se ha usado la cadena `http://www.pucp.edu.pe/informatica#` como base para las URI de este modelo. Este valor es referencial y puede ser cambiado sin mayor problema.

Programa 2: Creando recursos en RDF en Jena

```
...
public static void main(String[] args) {
        Model model = ModelFactory.createDefaultModel();
        String uri = "http://www.pucp.edu.pe/";
        String ns = "pucp";
        model.setNsPrefix(ns, uri);

        Resource mensWear = crearRecurso(uri, "MensWear", model);
        Resource shirts = crearRecurso(uri, "Shirts", model);
        Resource tshirts = crearRecurso(uri, "Tshirts", model);
        Resource henleys = crearRecurso(uri, "Henleys", model);
        Resource oxfords = crearRecurso(uri, "Oxfords", model);
        Resource chamoisHenley = crearRecurso(uri, "ChamoisHenley", model);
        Resource classicOxford = crearRecurso(uri, "ClassicOxford", model);
    }
...
```

Para crear los subconjuntos se usa la primitiva `RDFS.subClassOf` de RDF. Dentro de los vocabularios que ofrece Jena, como es de esperarse, se encuentra una para el soporte a RDFS. Usando el código `RDFS.subClassOf` se puede obtener la propiedad correspondiente. En el programa 3 se puede apreciar como se realiza esta creación de subconjuntos y cómo se agregan estos en el modelo.

Programa 3: Creando Creando subconjuntos en RDFS en Jena

```
...
        model.add(shirts, RDFS.subClassOf, mensWear);
        model.add(tshirts, RDFS.subClassOf, shirts);
        model.add(henleys, RDFS.subClassOf, shirts);
        model.add(oxfords, RDFS.subClassOf, shirts);
...
```

Finalmente para agregar las instancias se utiliza la primitiva `RDF.type`. Note que `type` es parte de las primitivas de RDF y no de RDFS. En el programa 4 podrá encontrar un ejemplo de cómo se hace la creación de estas instancias.

Programa 4: Creando instancias en RDFS en Jena

```
...
```

```
        model.add(chamoisHenley, RDF.type, henleys);
        model.add(classicOxford, RDF.type, shirts);
        model.add(classicOxford, RDF.type, oxfords);
...
```

Para poder usar este esquema RDFS, se procede a guardarlo. En el programa 5 podrá ver el código que permite realizar este tarea.

Programa 5: Grabando el esquema RDFS en Jena

```
...
        FileOutputStream output = null;
        try {
            output = new FileOutputStream("camisas.rdf");
        } catch (FileNotFoundException e) {
            System.out.println("Ocurrió un error al crear el archivo.");
        }
        model.write(output, "RDF/XML-ABBREV");
...
```

4. Consultando el esquema RDFS

En el programa 6 se encuentra el código en Java que permite consultar las subclases a las que pertenece el recurso `Henleys`. Puede usar como modelo de datos el archivo `camisas.rdf` que se encuentra en la plataforma Paideia, o el archivo que se encuentra en el URL `https://raw.githubusercontent.com/andres-melgar/rdf/master/camisas.rdf`. Como sabemos, se ha usado en prefijo `pucp` para los URI en el modelo RDF. Para expandir y obtener el URI completo, se usará el método `expandPrefix` de la clase `Model`. Este método recibe un *qname* y retorna la URI que le corresponde.

Programa 6: Consultando recurso en RDFS en Jena

```
...
        String inputFileName = "camisas.rdf";
        Model model = ModelFactory.createDefaultModel();
        model.read(inputFileName);

        String resourcedURI = model.expandPrefix("pucp:Henleys");
        Resource resource = model.getResource(resourcedURI);
        Selector selector = new SimpleSelector(resource, RDFS.subClassOf,
            (RDFNode) null);
        StmtIterator iter = model.listStatements(selector);
        while (iter.hasNext()) {
            System.out.println(iter.nextStatement().toString());
        }
...
```

Para poner en práctica

Ejecute el programa y analice el resultado:

- ¿Qué resultado retornó el programa?
- ¿Se consigue inferir todos las clases a los que pertenece el recurso?
- ¿Qué otra clase debió retornar?

Tal como está el programa, no se le ha dado la capacidad de hacer inferencias al momento de realizar la consulta. Esto es así pues, hacer inferencias a cada momento consume mucho procesamiento computacional. Para indicarle a Jena que se desea hacer inferencias al momento de realizar las consulta, hay que usar un modelo adicional. El modelo adicional debe tener la capacidad de hacer inferencias, esto se

consigue en Jena con la clase `InfModel`. Se usa el método `createRDFSModel` de la clase `ModelFactory` para instanciar un modelo de inferencia. Como parámetro del método `createRDFSModel` se usa el modelo leído previamente.

En el programa 7 se encuentra el código en Java que permite consultar las subclases a las que pertenece el recurso `Henleys`. Es bastante similar pero en este caso usa un modelo de inferencia.

Programa 7: Consultando recurso con modelo de inferencia en Jena

```
...
        String inputFileName = "camisas.rdf";
        Model model = ModelFactory.createDefaultModel();
        model.read(inputFileName);
        InfModel inf = ModelFactory.createRDFSModel(model);

        String resourcedURI = model.expandPrefix("pucp:Henleys");
        Resource resource = model.getResource(resourcedURI);
        Selector selector = new SimpleSelector(resource, RDFS.subClassOf,
            (RDFNode) null);
        StmtIterator iter = inf.listStatements(selector);
        while (iter.hasNext()) {
            System.out.println(iter.nextStatement().toString());
        }
...
```

Para poner en práctica

Ejecute el programa y analice el resultado:

- ¿Qué diferencia existe en los resultados de las consultas?
- ¿Para qué sirve la clase `InfModel`?
- ¿Qué nuevas tripletas retorna ahora la consulta?
- ¿El resultado de esta consulta es coherente con el patrón de inferencia de pertenencia de conjuntos?
- ¿Hubo necesidad de alterar la consulta o el modelo de datos?

5. Consultando tipos

Se desea ahora conocer las instancias de las camisas, es decir, aquellos recursos que se han definido como de tipo camisa. Probaremos primero sin usar el motor de inferencias de Jena. En el programa 8 se puede apreciar el código que realiza dicha consulta.

Programa 8: Consultando camisas en Jena

```
...
        String inputFileName = "camisas.rdf";
        Model model = ModelFactory.createDefaultModel();
        model.read(inputFileName);

        String resourcedURI = model.expandPrefix("pucp:Shirts");
        Resource resource = model.getResource(resourcedURI);
        Selector selector = new SimpleSelector(null, RDF.type, (RDFNode)
            resource);
        StmtIterator iter = model.listStatements(selector);
        while (iter.hasNext()) {
            System.out.println(iter.nextStatement().toString());
        }
...
```

Para poner en práctica

Ejecute el programa y analice el resultado:

- Según lo visto en clase, ¿qué otra instancia debería retornar la consulta?
- ¿Por qué no la ha retornado?

En el programa 9 se puede apreciar un código similar al anterior pero usando un modelo de inferencias.

Programa 9: Consultando camisas con modelo de inferencia en Jena

```
...
        String inputFileName = "camisas.rdf";
        Model model = ModelFactory.createDefaultModel();
        model.read(inputFileName);
        InfModel inf = ModelFactory.createRDFSModel(model);

        String resourcedURI = model.expandPrefix("pucp:Shirts");
        Resource resource = model.getResource(resourcedURI);
        Selector selector = new SimpleSelector(null, RDF.type, (RDFNode)
           resource);
        StmtIterator iter = inf.listStatements(selector);
        while (iter.hasNext()) {
            System.out.println(iter.nextStatement().toString());
        }
...
```

Para poner en práctica

Ejecute el programa y analice el resultado:

- ¿Qué diferencia existe en los resultados de las consultas?
- ¿Qué nuevas tripletas retorna ahora la consulta?
- ¿El resultado de esta consulta es coherente con el patrón de inferencia de pertenencia de conjuntos?
- ¿Hubo necesidad de alterar la consulta o el modelo de datos?
- ¿Existe diferencia entre las afirmación que se han realizado con las tripletas y los resultados inferidos en la consulta?

San Miguel, 16 de septiembre de 2020.