Université de Montpellier
Faculté des Scicences
30 Place E.Bataillon, 34095 Montpellier

Année 2023-2024

Rapport de Travaux Pratiques

TP n°2
RDF Schema et OWL

par

Romain GALLERNE

Encadrant de TP : M Frederico Ulliana

Responsable du module : M Frederico Ulliana

Table des matières

Données, Vocabulaire et Ontologies

1.1 Identifier et lister l'ensemble des vocabulaires utilisées dans les données RDF

Un vocabulaire est composé de classes et de relations. Afin d'identifier clairement le vocabulaire, nous avons schématisé le document. Voici le schéma obtenu :

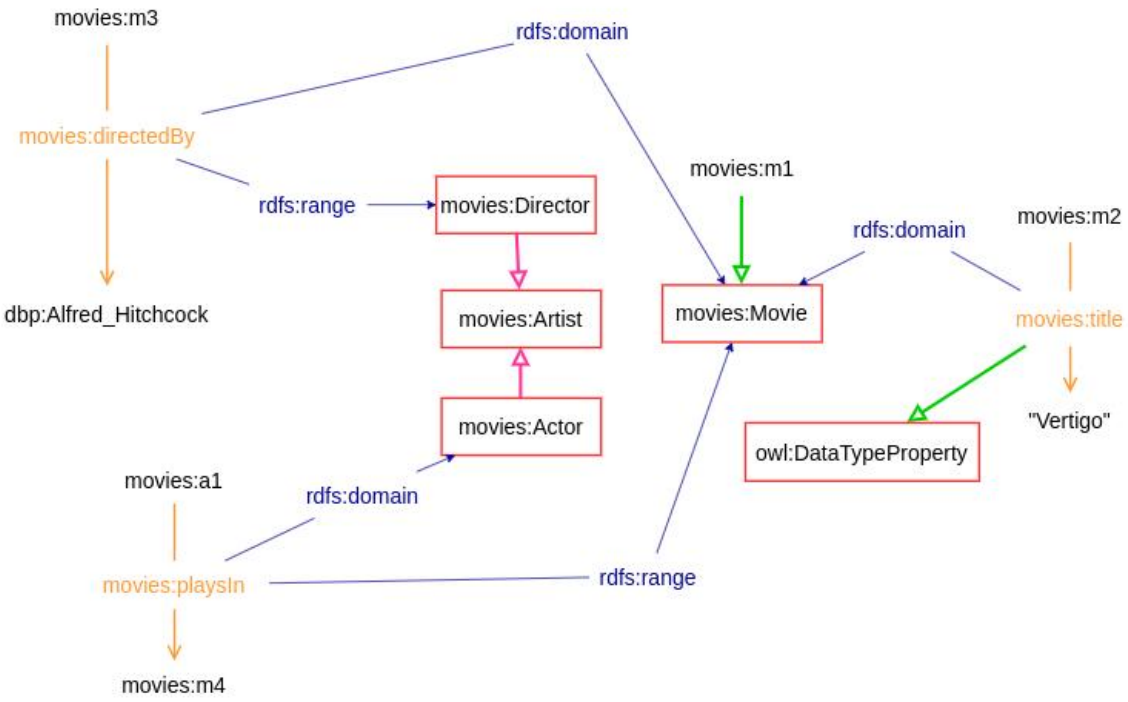

Figure 1 : Schématisation du document de l'énoncé

À l'aide du schéma obtenu, on peut donc identifier clairement les vocabulaires. Le vocabulaire principal, "Movies" composé des classes :

— "Director"
— "Artist"
— "Actor"
— "Movie"

et des propriétés :

— "directedBy"
— "playsIn"
— "title"

Les autres vocabulaires présents ici sont "rdfs" qui permet de structurer les ontologies. On a également le vocabulaire owl qui porte lui aussi sur la structuration des ontologies et enfin dbpedia qui porte sur une large variété de connaissances généralistes.

1.2 Séparer les triplets contenant des connaissances ontologiques des triples représentants des données.

On distingue deux types de triplets dans un document RDF. Les triplets ontologiques représentent les conaissances du domaine qu'on obtient le plus souvent aurpès des experts (exemple : un film est dirigé par un directeur). Les triplets des données représentent les données que l'on souhaite enregistrer (exemple : Le film "Star wars" a été dirigé par "Georges Lucas").

Soient les triplets représentant les connaissances ontologiques :

— movies :directedBy rdfs :domain movies :Movie .
— movies :title rdfs :domain movies :Movie .
— movies :directedBy rdfs :range movies :Director .
— movies :playsIn rdfs :domain movies :Actor .
— movies :playsIn rdfs :range movies :Movie .
— movies :Actor rdfs :subClassOf movies :Artist .
— movies :Director rdfs :subClassOf movies :Artist .
— movies :title rdf :type owl :DataTypeProperty .

Les triplets représentants les données :

— movies :m2 movies :title ”Vertigo” .

— movies :m1 rdf :type movies :Movie .

— movies :m3 movies :directedBy dbp :Alfred_Hitchcock .

— movies :a1 movies :playsIn movies :m4 .

1.3 Donner la liste des triplets qu'on peut inférer grâce aux connaissances ontologiques.

De part les relations ici présentes, on peut inférer d'autres triplets, notamment grâce aux règles de subsumption et de transitivité, bien que la transitivité ne soit pas indispensable. Cela permet d'obtenir les règles suivantes :

— movies :directedBy rdfs :range movies :Artist .

— movies :playsIn rdfs :domain movies :Artist .

— movies :m3 rdfs :type movies :Movie .

— movies :m4 rdfs :type movies :Movie .

— movies :m2 rdfs :type movies :Movie .

— dbp :Alfred_Hitchcock rdfs :type movies :Directeur .

— dbp :Alfred_Hitchcock rdfs :type movies :Artist .

— movies :a1 rdfs :type movies :Actor .

— movies :a1 rdfs :type movies :Artist .

1.4 Créer un modèle Jena

1.4.1 Modèle Jena

On commence ici par la création d'un modèle Jena ”basique” comme on l'a fait lors du premier TP.

```
// Define namespace prefixes
String MOV_namespace = "http://www.lirmm.fr/ulliana/movies";
String FOAF_namespace = "http://xmlns.com/foaf/0.1/";
String DBP_namespace = "http://dbpedia.org/";
model.setNsPrefix("movies", MOV_namespace);
model.setNsPrefix("foaf", FOAF_namespace);
model.setNsPrefix("dbpedia", DBP_namespace);

// Propriétés et ressources
Resource director = model.createResource(MOV_namespace + "Director");
Resource actor = model.createResource(MOV_namespace + "Actor");
Resource artist = model.createResource(MOV_namespace + "Artist");
Resource movie = model.createResource(MOV_namespace + "Movie");

Property directedBy = model.createProperty(MOV_namespace + "directedBy");
Property playsIn = model.createProperty(MOV_namespace + "playsIn");
Property title = model.createProperty(MOV_namespace + "title");

//Hierarchie du vocabulaire des films
model.add(director, RDFS.subClassOf, artist);
model.add(actor, RDFS.subClassOf, artist);
model.add(title, RDF.type, OWL.DatatypeProperty);
model.add(directedBy, RDFS.range, director);
model.add(directedBy, RDFS.domain, movie);
model.add(playsIn, RDFS.range, movie);
model.add(playsIn, RDFS.domain, actor);
model.add(title, RDFS.domain, movie);
```

```
// Film 1
Resource m1 = model.createResource(MOV_namespace + "m1");
model.add(m1, RDF.type, movie);

// Film 2
Resource m2 = model.createResource(MOV_namespace + "m2");
model.add(m2, title, model.createLiteral("Vertigo"));

// Film 3
Resource m3 = model.createResource(MOV_namespace + "m3");
Resource ah = model.createResource(DBP_namespace + "Alfred_Hitchcock");
model.add(m3, directedBy, ah);

// Film 4
Resource a1 = model.createResource(MOV_namespace + "a1");
Resource m4 = model.createResource(MOV_namespace + "m4");
model.add(a1, playsIn, m4);
```

Figure 1 : Premier modèle Jena

On remarque que ce modèle basique n'utilisant que le ModelCreator ne réalise pas les inférences. On va donc explorer une méthodologie différentes en utilisant un résonneur OWL pour obtenir les inférences.

1.4.2 Inférences

On crée donc deux fichier **owlMovieData.tt** et **owlMovieSchema.tt**. Dans le premier fichier on insère toutes les données sur notre modèle, dans le second on insère les connaissances ontologiques comme détaillées dans la partie **1.2**. On execute alors le programme et on obtient le résultat suivant :

```
==========RDF Triples before inferences========================================

http://www.lirmm.fr/ulliana/movies#m1 @rdf:type http://www.lirmm.fr/ulliana/movies#Movie
http://www.lirmm.fr/ulliana/movies#m2 @http://www.lirmm.fr/ulliana/movies#title "Vertigo"
http://www.lirmm.fr/ulliana/movies#a1 @http://www.lirmm.fr/ulliana/movies#playsIn http://www.lirmm.fr/ulliana/movies#m4
http://www.lirmm.fr/ulliana/movies#m3 @http://www.lirmm.fr/ulliana/movies#directedBy http://dbpedia.org/Alfred_Hitchcock


==========RDF Triples after inferences=========================================

http://dbpedia.org/Alfred_Hitchcock @rdf:type http://www.lirmm.fr/ulliana/movies#Director
http://dbpedia.org/Alfred_Hitchcock @rdf:type http://www.lirmm.fr/ulliana/movies#Artist
http://www.lirmm.fr/ulliana/movies#m2 @rdf:type http://www.lirmm.fr/ulliana/movies#Movie
http://www.lirmm.fr/ulliana/movies#m3 @rdf:type http://www.lirmm.fr/ulliana/movies#Movie
http://www.lirmm.fr/ulliana/movies#m4 @rdf:type http://www.lirmm.fr/ulliana/movies#Movie
http://www.lirmm.fr/ulliana/movies#a1 @rdf:type http://www.lirmm.fr/ulliana/movies#Actor
http://www.lirmm.fr/ulliana/movies#a1 @rdf:type http://www.lirmm.fr/ulliana/movies#Artist


================================================================================
```

Figure 2 : Résultat obtenu après exécution du résonneur OWL.

On constate que tous les triplés de données que nous avions inférés manuellement sont bien présents à la suite de l'exécution du résonneur OWL.

De RDF-Schema à OWL

2.1 Extensions du modèle précédent

2.1.1 Symétrie

Pour ajouter une propriété de symétrie, on définit une relation **collegue** tel que le collegue de quelqu'un a pour collegue cette même personne :

— movies :collegue rdf :type owl :SymmetricProperty .
— movies :a1 movies :collegue movies :a2 .
Cela permet donc d'inférer la connaissance suivante :
— movies :a2 movies :collegue movies :a1 .

2.1.2 Fonctionelle

Pour la propriété fonctionelle, c'est à dire la propriété d'unité de l'objet, on ajoute les triplets suivant :
— movies :directedBy rdf :type owl :FunctionalProperty .
— movies :m3 movies :directedBy movies :AH .
On obtient donc deux nouveau triplets inférés :
— movies :Alfred_Hitchcock owl :sameAs movies :AH .
— movies :AH owl :sameAs movies :Alfred_Hitchcock .

2.1.3 Inverse

Pour la propriété inverse permettant d'inférer une nouvelle propriété en inversant le sujet et l'objet d'une propriété pré-existente, on ajoute les triplets suivant :

— movies :playsIn owl :inverseOf movies :asActor .

Cela permet d'obtenir le triplet :

— movies :m4 movies :asActor movies :a1 .

2.1.4 Transitif

Pour la propriété transitive permettant d'inférer de nouveaux triplets par transitivité sur des URI A en relation avec B et B en relation avec C donnant A en relation avec C, on ajoute les triplets suivant en réutilisant la relation "collegue" :

— movies :collegue rdf :type owl :TransitiveProperty .

— movies :a1 movies :collegue movies :a2 .

— movies :a2 movies :collegue movies :a3 .

Cela permet d'obtenir :

— movies :a1 movies :collegue movies :a3 .

— movies :a3 movies :collegue movies :a1 . (par transitivité ET symétrie)

2.1.5 Chaîne de propriété

Pour la chaine de propriété permettant d'inférer de nouveau triplets sur deux URI A et B s'il existe un chemin de propriété P1, P2... Pn de A vers B. On écrit les triplets suivants :

— movies :playedUnderDirector rdf :type owl :ObjectProperty ; owl :PropertyChainAxiom (movies :playsIn movies :directedBy) .

— movies :a2 movies :playsIn movies :m3 .

On infère ainsi les triplets :

— movies :a2 movies :playedUnderDirector movies :Alfred_Hitchcock .

— movies :a2 movies :playedUnderDirector movies :AH .

2.2 Implémentation en Jena

L'implémentation en Jena se fait simplement en indiquant les nouveaux triplés cités précedemment dans les fichiers schéma et données des ressources du résonneurs. On obtient bien les triplets inférés donnés ci-dessus.