Documentation

Introduction

In this assignment, different techniques were applied for population knowledge graphs. More specifically, first the given film ontology was extended with necessary classes and properties and then an ontology was populated with instances from certain data sets. The types of data sets were CSV and JSON files. Regarding the CSV files, the mapping was done using the OpenRefine program with RDF extension that proposed during the lectures. Regarding the JSON files, the mapping was done with the help of the RML mapper provided from github (https://github.com/RMLio/rmlmapper-java). Finally, SPARQL queries were implemented on the final knowledge graphs in order to further enrich them.

Overview and high-level description of the RML mappings

The 2 JSON data sets (1000_movies_metadata and 1000_credits) contain information about the film ontology. In particular, the 1000_movies_metadata.json contains the id of the films, the imdb id, the original title of the movies, the type of genres of the movies including the id, the productions companies and countries and the spoken languages. Furthermore, the 1000_credits.json file contains information about the cast and the crew members of each film.

Firstly, regarding the RML mapping, the appropriate prefixes were defined. Below is presented the prefixes that were used.

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix map: <http://mapping.example.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix sd: <http://www.w3.org/ns/sparql-service-
description#> .
@prefix ex: <http://semantics.id/ns/example/film#> .
```

After that, the logical source was defined. The logical source mapping consists of a reference to input sources (rml:source "1000_movies_metadata.json" and rml:source "1000_credits.json"), the referenceFormulation to specify how to refer the data (rml:referenceFormulation ql:JSONPath) and finally the rml:iterator "$.[*]" which is the iterator that specifies how to iterate over the data.

Consequently, the subject map was defined using as template a URI pattern that describes how each triple's subject is produced. Also, the type of the URI was defined. Below is presented an example of the RML Subject Map.

```
rr:subjectMap [
   rr:template "http://semantics.id/ns/example/film#film_{id}";
   rr:class ex:Film;
 ]
```

Then, the triples are produced using the Predicate Object Map. The predicate map specifies how the triple's predicate is produced. The same thing applies for the object map regarding the triple's objects. Below is presented an example of the RML Predicate Object Map.

```
rr:predicateObjectMap [
    rr:predicate ex:idIMDB;
    rr:objectMap [
        rml:reference "imdb_id" ;
        rr:datatype xsd:string ;
    ]
```

After creating the RML mapping files, the command *"java -jar rmlmapper-6.1.3-r367-all.jar -m (name of the file).rml.ttl"* was used in order the triples to be created. To extract the triples from the mapping, the command *"java -jar rmlmapper-6.1.3-r367-all.jar -m (name of the file).rml.ttl -o (output file)"* was used. Finally, for the final knowledge graph in ttl format from the JSON files, the rdflib library was used, provided by python, in order to transform the triples into turtle format. In the next chapter, an example sub-graph from the JSON files is shown in ttl format.

Mapping explanation

Regarding the 1000_movies_metadata JSON file, the class instances for the class FilmStudio took the "name" and the "id" from the "production_companies". Also, the class instances for the class Genre took the "name" and the "id" from the "genres". Finally, for the class Film the originalTitle (String), hasSpokenLanguage (String) and hasProductionCountry (String) were mapped to "original_title", "iso_3166_1" and "iso_639_1", respectively. Furthermore, the genre of each film was connected to the class instances for the Genre class through the object property hasGenre.

Regarding the 1000_credits JSON file, the class instances for the class Actor took the fullName (String) and the gender (String) from the "cast" and more specific the "name" and "gender". Additionally, the class instances for the class Director took the fullName (String) and the gender (String) from the "crew" and more specific the "name" and "gender" where "department"=="Directing". The class instances for the class ScriptWriter took the fullName (String) and the gender (String) from the "crew" and more specific the "name" and "gender" where "department"=="Writing". The class instances for the class Cast were connected through the object property hasCastActor with the instances of the class Actor. Also, the hasCastCharacter (String) was mapped to "cast" and more specific the "character". Finally, the class instances for the class Film were connected through the object properties hasDirector, hasScriptWriter, hasActor and hasCast with the class instances of the classes Director, ScriptWriter, Actor and Cast, respectively.

Sub-graphs from JSON files

In this part of the documentation, an example sub-graph is presented from JSON files in TTL serialization containing an instance from each class that are connected to each other.

Example 1 from 1000_movies_metadata.json

```
ex:FilmStudio_10038 a ex:FilmStudio ;
    rdfs:label "Square Enix" ;
    ex:idFilmStudio "10038" .
```

Example 2 from 1000_movies_metadata.json

```
ex:film_124 a ex:Film ;
    ex:hasGenre ex:genre_14,
        ex:genre_18 ;
    ex:hasProductionCountry "PL" ;
    ex:hasSpokenLanguage "en",
        "pl" ;
    ex:idIMDB "tt0086961" ;
    ex:originalTitle "Bez Konca" .
```

Example 3 from 1000_movies_metadata.json

```
ex:genre_37 a ex:Genre ;
    rdfs:label "Western" ;
    ex:idGenre "37" .
```

Example 1 from 1000_credits.json

```
ex:film_100 a ex:Film ;
    ex:hasActor ex:Actor_1116,
        ex:Actor_115787,
        ex:Actor_1473866,
        ex:Actor_17290,
        ex:Actor_236388,
        ex:Actor_4317,
        ex:Actor_4318,
        ex:Actor_47632,
        ex:Actor_973,
        ex:Actor_974,
        ex:Actor_975,
        ex:Actor_976,
        ex:Actor_978,
        ex:Actor_980,
        ex:Actor_982,
        ex:Actor_984,
        ex:Actor_985,
        ex:Actor_986,
        ex:Actor_987,
        ex:Actor_988,
        ex:Actor_989,
        ex:Actor_990,
        ex:Actor_991,
        ex:Actor_992,
        ex:Actor_993,
        ex:Actor_994,
        ex:Actor_995 ;
    ex:hasCast ex:Cast_12,
        ex:Cast_13,
```

```
        ex:Cast_14,
        ex:Cast_15,
        ex:Cast_16,
        ex:Cast_17,
        ex:Cast_18,
        ex:Cast_19,
        ex:Cast_20,
        ex:Cast_21,
        ex:Cast_22,
        ex:Cast_23,
        ex:Cast_24,
        ex:Cast_25,
        ex:Cast_26,
        ex:Cast_27,
        ex:Cast_28,
        ex:Cast_29,
        ex:Cast_30,
        ex:Cast_31,
        ex:Cast_32,
        ex:Cast_35,
        ex:Cast_36,
        ex:Cast_37,
        ex:Cast_38,
        ex:Cast_39,
        ex:Cast_40 ;
    ex:hasDirector ex:director_956
;
    ex:hasScriptWriter
ex:writer_956 .
```

Example 2 from 1000_credits.json

```
ex:director_9181 a ex:Director ;
    ex:fullName "Edward Zwick" ;
    ex:gender "2"
```

Example 3 from 1000_credits.json

```
ex:writer_10230 a ex:ScriptWriter ;
    ex:fullName "Hannes Stöhr" ;
    ex:gender "2" .
```

Example 4 from 1000_credits.json

```
ex:Actor_10055 a ex:Actor ;
    ex:fullName "Actor Sergio Kato";
    ex:gender "2".
```

Example 5 from 1000_credits.json

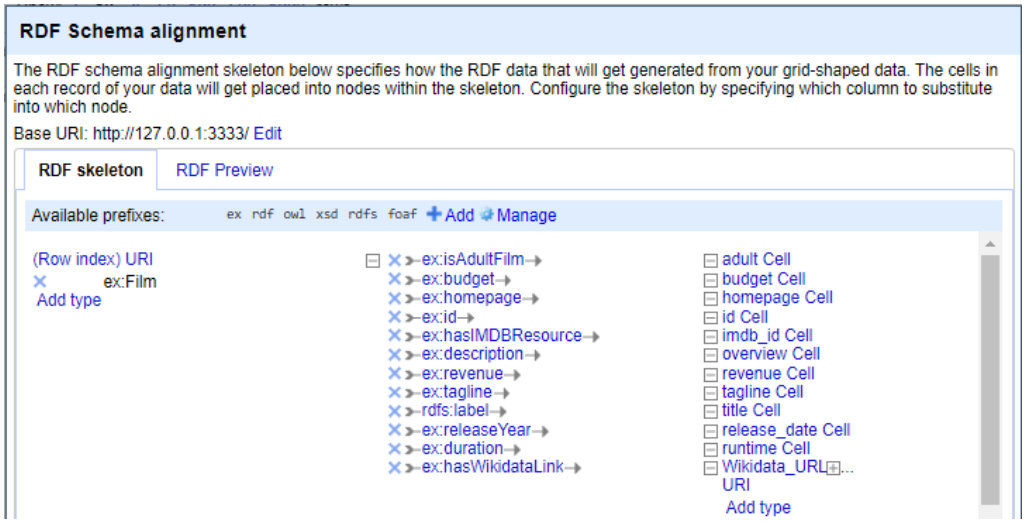
The last example refers to the class Cast. Due to the fact that the example was long for the documentation, an incomplete example was used below for demonstration purposes only.

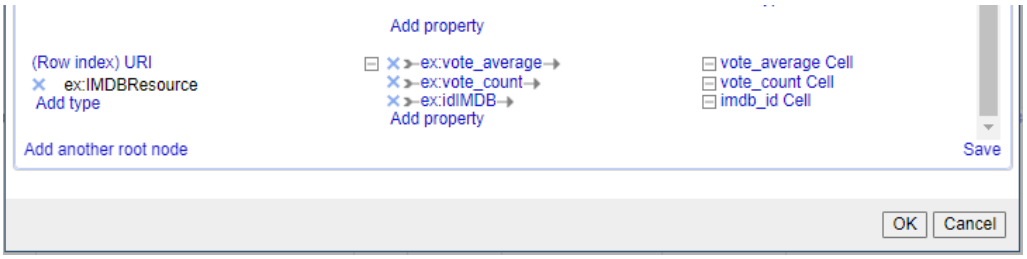
```
ex:Cast_1 a ex:Cast ;
   ex:hasCastActor ex:Actor_1,
       ex:Actor_100,
       ex:Actor_100031,
       …
  ex:hasCastCharacter "Adele Ekstrom",
       "Adjutant von General Weidling",
       "Admiral Boom",
      "Admiral Chuck Farrell",
       …
```

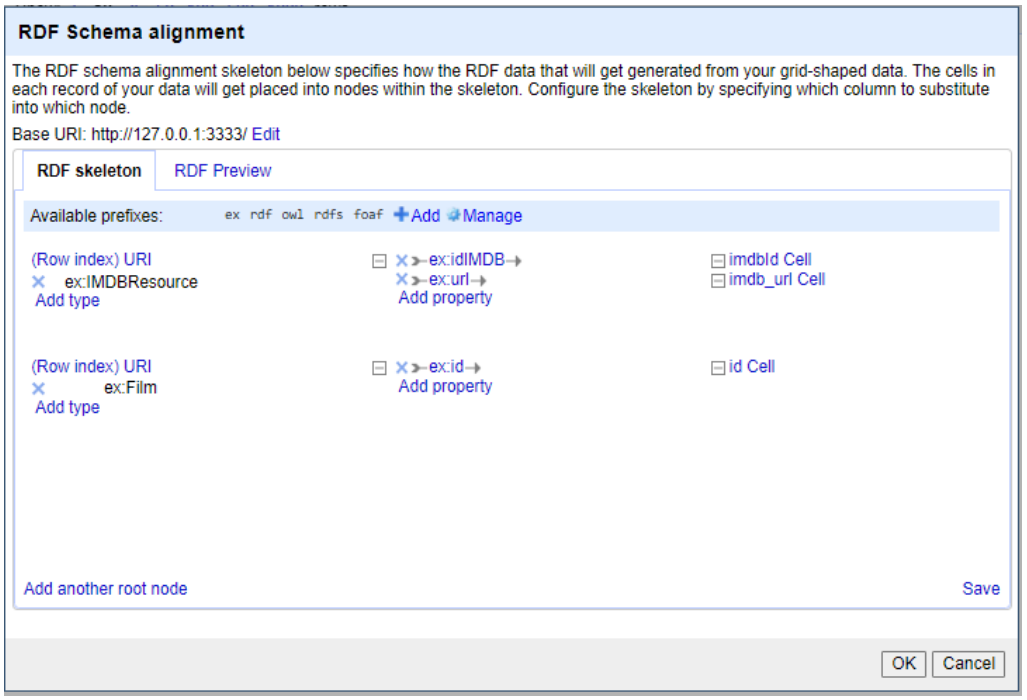
RDFRefine mappings and high-level description

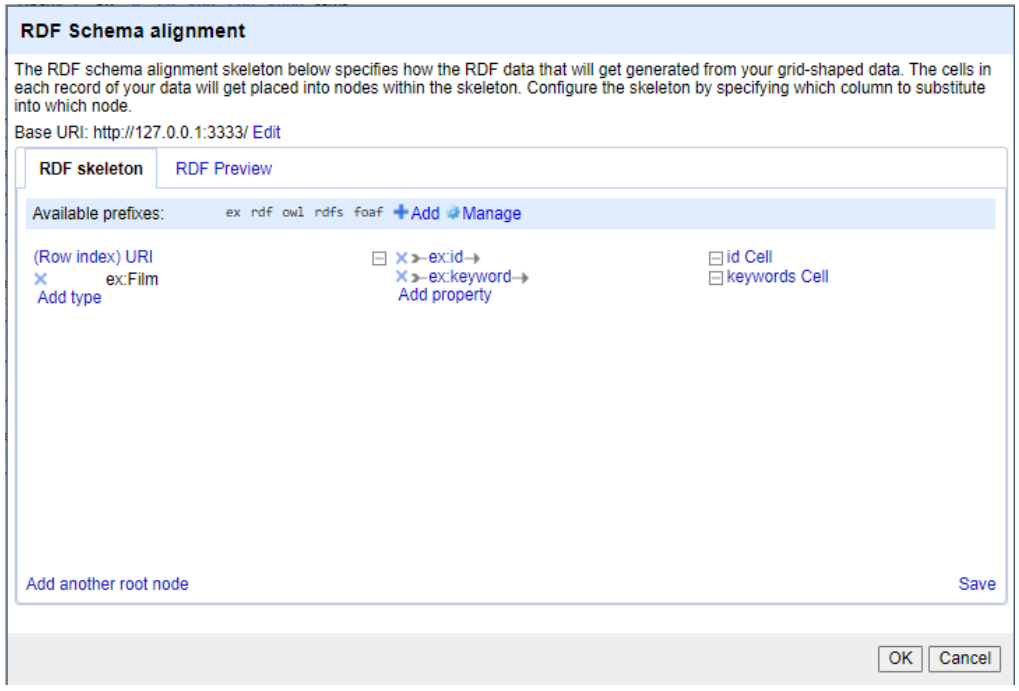
In the screenshots below are presented RDFRefine mappings that were done on the 3 CSV files. The first screenshot is related to the 1000_movies_metadata CSV file. Based on the screenshot can be seen the mapping that was done in the extended film ontology. On the left of the RDF schema, the two classes, Film and IMDBResource, are observed. In the middle are the data properties, where some of them belong to a previous version of the film ontology and some were added to the extended one. Finally, on the right are the columns, with their names on them, of the CSV file where they represent the values of the instances. It is worth noting that the reconciliation was used on the original titles in order to match the film titles with the corresponding films from the wikidata. After successfully reconcile the column original_title, we matched each cell to its best candidate and then a new column ("Wikidata_URL") was created. This column contains the url from the wikidata of each film.

The second and the third screenshots are related to the 1000_links and 1000_keywords CSV files, respectively. The same thing, as mentioned for the first screenshot, applies to these two data sets.




Add property
(Row index) URI
ex:IMDBResource
Add type
ex:vote_average
ex:vote_count
ex:idIMDB
Add property
vote_average Cell
vote_count Cell
imdb_id Cell
Add another root node
Save
OK
Cancel


RDF Schema alignment
The RDF schema alignment skeleton below specifies how the RDF data that will get generated from your grid-shaped data. The cells in each record of your data will get placed into nodes within the skeleton. Configure the skeleton by specifying which column to substitute into which node.
Base URI: http://127.0.0.1:3333/ Edit
RDF skeleton
RDF Preview
Available prefixes: ex rdf owl rdfs foaf Add Manage
(Row index) URI
ex:IMDBResource
Add type
ex:idIMDB
ex:url
Add property
imdbId Cell
imdb_url Cell
(Row index) URI
ex:Film
Add type
ex:id
Add property
id Cell
Add another root node
Save
OK
Cancel


RDF Schema alignment
The RDF schema alignment skeleton below specifies how the RDF data that will get generated from your grid-shaped data. The cells in each record of your data will get placed into nodes within the skeleton. Configure the skeleton by specifying which column to substitute into which node.
Base URI: http://127.0.0.1:3333/ Edit
RDF skeleton
RDF Preview
Available prefixes: ex rdf owl rdfs foaf Add Manage
(Row index) URI
ex:Film
Add type
ex:id
ex:keyword
Add property
id Cell
keywords Cell
Add another root node
Save
OK
Cancel

Sub-graphs from CSV files

In the next 3 examples are presented an instance from each class that are to each other

Example 1 from 1000_movies_metadata.csv

```
<http://127.0.0.1:3333/0> a ex:Film;
ex:isAdultFilm "False"^^xsd:boolean;
ex:budget "30000000"^^xsd:double;
ex:homepage "http://toystory.disney.com/toy-story";
ex:id "862";
ex:hasIMDBResource "0114709";
ex:description "Led by Woody, Andy's toys live happily in his room until Andy's birthday brings Buzz Lightyear onto the scene. Afraid of losing his place in Andy's heart, Woody plots against Buzz. But when circumstances separate Buzz and Woody from their owner, the duo eventually learns to put aside their differences.";
ex:revenue "373554033"^^xsd:int;
rdfs:label "Toy Story";
ex:releaseYear "1995-10-30"^^xsd:int;
ex:duration "81.0"^^xsd:int;
ex:hasWikidataLink <https://www.wikidata.org/wiki/Q171048>;
a ex:IMDBResource;
ex:vote_average "7.7"^^xsd:double;
ex:vote_count "5415"^^xsd:int;
ex:idIMDB "0114709" .
```

Example 2 from 1000_links.csv

```
<http://127.0.0.1:3333/0> a ex:IMDBResource;
ex:idIMDB "0114709";
ex:url "https://www.imdb.com/title/tt0114709";
a ex:Film;
ex:id "862" .
```

Example 3 from 1000_keywords.csv

```
<http://127.0.0.1:3333/0> a ex:Film;
ex:id "862";
ex:keyword "[{'id': 931, 'name': 'jealousy'}, {'id': 4290, 'name': 'toy'}, {'id': 5202, 'name': 'boy'}, {'id': 6054, 'name': 'friendship'}, {'id': 9713, 'name': 'friends'}, {'id': 9823, 'name': 'rivalry'}, {'id': 165503, 'name': 'boy next door'}, {'id': 170722, 'name': 'new toy'}, {'id': 187065, 'name': 'toy comes to life'}]" .
```