Assignment 3

Introduction to Semantic Systems 2024W

Bosse Behrens, st.ID: 12347333 e12347333@student.tuwien.ac.at

1 CSV-Files Mappings

First I updated the ontology film.ttl before this step to include all new classes and properties and then imported the vocabulary into openRefine so I could map the classes and properties with the right names already suggested.

1.1 Movie Metadata

I started with the `1000_movies_metadata.csv`. First I loaded it into openRefine and cleaned the data a bit. The boolean in adult I set to lowercase so openRefine could detect it as boolean, the release date I formatted to only show the year so it matched with the ontology. I also removed 3 faulty entries that had strange data.
With the RDF extension I then started on the mappings. First I set the baseIRI to match the ontology. Then I set on the left side the identifying URI to id and added the "film", so each object of the class film is identifdied by its id. on the right side I added all the properties. Most are data properties, for example popularity, for which I set the data type to non-integer (translates to double). The only object property is hasIMDBResource which maps each film to its IMDB Resource. For this I added the property, set the values to the imdb id which is the identifier for the IMDBResource and then added the type of the IMDBResource class.

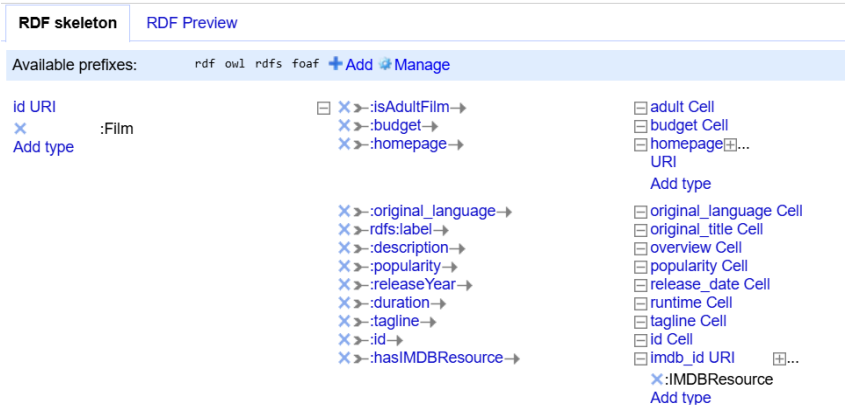

Figure 1: Mapping of the film class in 1000_movies_metadata.json

Then for the other class IMDBResource (didn't fit into one screenshot due to openRefines set window size) I did the same. I set the identifier URI to the IMDB id anmd added the three data properties.

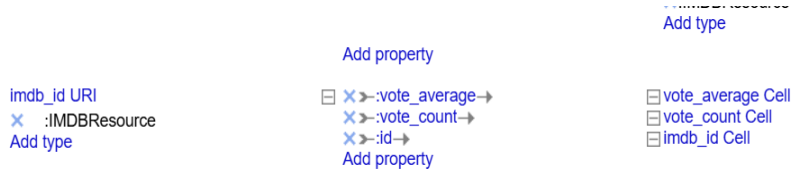

Figure 2: Mapping of the IMDBResource class in 1000_movies_metadata.csv

1.2 IMDB URLs

For the next csv file `1000_links.csv` I proceeded similarily. First I set the base IRI, then set the identifier URI to the IMDB id and added the data property of the IMDB URL. Here I used as a type the URI so it shows the actual link instead of just the string.

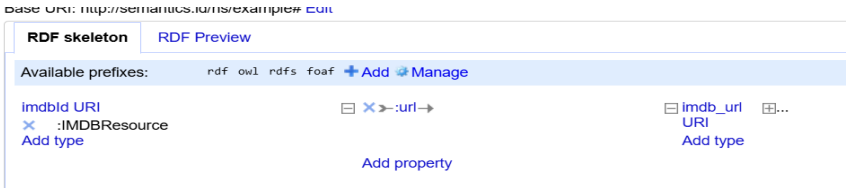

Figure 3: Mapping of the IMDBResource class in 1000_links.csv

1.3 Keywords

For the next csv file `1000_keywords.csv` I proceeded similarily. Here I first transformed the csv using a python script, so it ended up not having all keywords of a movie in the json format as a cell content, but one line for each pair of movie and keyword, so for each movie multiple rows. Then I again created he base IRI, set the identifier to the movie id and mapped the data property of the keyword string to it.

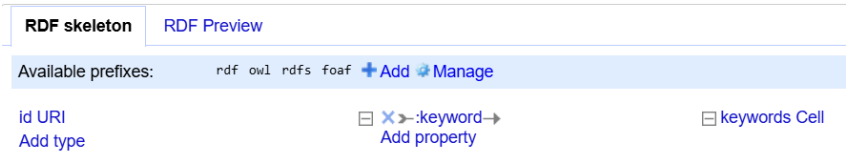

Figure 4: Mapping of the IMDBResource class in 1000_keywords.csv

1.4 Subgraph of combined Csv-Files

The following is a resulting subgraph from the combined ttl in TTL-serialization.

```
<http://semantics.id/ns/example#film_862>
    rdf:type            :Film;
    :isAdultFilm        false;
    :budget
    ↪  "30000000"^^<http://www.w3.org/2001/XMLSchema#double>;
    :homepage           <http://toystory.disney.com/toy-story>;
    :original_language  "en";
    rdfs:label          "Toy Story";
    :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.";
    :popularity
    ↪  "21.946943"^^<http://www.w3.org/2001/XMLSchema#double>;
    :releaseYear        "1995"^^<http://www.w3.org/2001/XMLSchema#int>;
    :duration           "81"^^<http://www.w3.org/2001/XMLSchema#int>;
    :id                 "862";
    :hasIMDBResource  <http://semantics.id/ns/example#tt0114709>;
    :keyword  "jealousy";
    :keyword  "toy";
    :keyword  "boy";
    :keyword  "friendship";
    :keyword  "friends";
    :keyword  "rivalry";
    :keyword  "boy next door";
    :keyword  "new toy";
    :keyword  "toy comes to life" .
```

```
<http://semantics.id/ns/example#tt0114709>
    rdf:type       :IMDBResource;
    :vote_average  "7.7"^^<http://www.w3.org/2001/XMLSchema#double>;
    :vote_count    "5415"^^<http://www.w3.org/2001/XMLSchema#int>;
    :id            "tt0114709";
    :url      <https://www.imdb.com/title/tt0114709>.
```

2 JSON-Files RML-Mappings

2.1 Credits JSON

For this file I first preprocessed it with a python script. The transformed json file is also in the submission, called `1000_credits_flattened.json`. The parser and RML mapper had problems with the nested arrays for the jsonpath, so I flattened the structure so each crew and cast entry also had the movie id, instead of the id being higher up in the structure where it had to be retrieved. I then wrote the YARRRML file which was then first parsed by the YARRRML-Parser and then mapped by the RMLMapper. THis is the YARRRML script:

```
prefixes:
  ex: "http://semantics.id/ns/example#"
  film: "http://semantics.id/ns/example/film#"
  rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  rdfs: "http://www.w3.org/2000/01/rdf-schema#"
  xsd: "http://www.w3.org/2001/XMLSchema#"

mappings:
  actors:
    sources:
      - ['1000_credits_flattened.json~jsonpath', '$[?(@.type == "cast")]']
    s: ex:actor_$(id)
    po:
      - [a, film:Actor]
      - [film:fullName, $(name)]
      - [film:gender, $(gender)]
      - [film:filmID, $(filmID)]

  directors:
    sources:
      - ['1000_credits_flattened.json~jsonpath', '$[?(@.type == "crew" &&
        ↪  @.role == "Director")]']
    s: ex:director_$(id)
    po:
      - [a, film:Director]
      - [film:fullName, $(name)]
      - [film:gender, $(gender)]
      - [film:filmID, $(filmID)]

  scriptwriters:
    sources:
      - ['1000_credits_flattened.json~jsonpath', '$[?(@.type == "crew" &&
        ↪  @.role == "Screenplay")]']
    s: ex:scriptwriter_$(id)
    po:
      - [a, film:ScriptWriter]
```

```
    - [film:fullName, $(name)]
    - [film:gender, $(gender)]
    - [film:filmID, $(filmID)]

  editors:
    sources:
      - ['1000_credits_flattened.json~jsonpath', '$[?(@.type == "crew" &&
      ↪  @.role == "Editor")]']
    s: ex:editor_$(id)
    po:
      - [a, film:Editor]
      - [film:fullName, $(name)]
      - [film:gender, $(gender)]
      - [film:filmID, $(filmID)]

  cast:
    sources:
      - ['1000_credits_flattened.json~jsonpath', '$[?(@.type == "cast")]']
    s: ex:cast_$(id)
    po:
      - [a, film:Cast]
      - [film:hasCastActor, 'ex:actor_$(id)~iri']
      - [film:hasCastCharacter, $(character)]

  films:
    sources:
      - ['1000_credits_flattened.json~jsonpath', '$[*]']
    s: ex:film_$(filmID)
    po:
      - [a, film:Film]
      - [film:hasActor, 'ex:actor_$(id)~iri']
      - [film:hasDirector, 'ex:director_$(id)~iri']
      - [film:hasScriptWriter, 'ex:scriptwriter_$(id)~iri']
      - [film:hasEditor, 'ex:editor_$(id)~iri']
      - [film:hasCast, 'ex:cast_$(id)~iri']
```

First it identifies actors which all have the type cast. The subject is actor_id. The properties are then mapped. First the class/rdf:type Actor, then the data properties fullName and gender and finally the object proeprty filmID where the actor is linked to the movies he starred in. For directors, editors and scriptwriters it is similar, though they are identified in the json by the type crew with the respective roles. Then they get mapped similar properties. For the Cast then it is also mapped for the rdf:type film:Cast for each movie its actor via the actor_id and then as a data property the string that is the name of the character they play in the respective movie. Finally in the other direction each rdf:type :film is mapped to its cast/crew by the object properties hasActor, hasDirector, etc.

2.2 Movie metadata JSON

For the second json-file following is the YARRRML script:

```
prefixes:
  ex: "http://semantics.id/ns/example#"
  film: "http://semantics.id/ns/example/film#"
  rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  rdfs: "http://www.w3.org/2000/01/rdf-schema#"
  xsd: "http://www.w3.org/2001/XMLSchema#"
```

```
mappings:
  films:
    sources:
      - ['1000_movies_metadata.json~jsonpath', '$[*]']
    s: ex:film_$(id)
    po:
      - [a, film:Film]
      - [film:id, $(id)]  # Added this line for film ID
      - [film:originalTitle, $(original_title)]
      - p: film:hasSpokenLanguage
        o: $(spoken_languages[*].name)
      - p: film:hasProductionCountry
        o: $(production_countries[*].name)
      - [film:hasFilmStudio, 'ex:studio_$(production_companies[*].id)~iri']
      - [film:hasGenre, 'ex:genre_$(genres[*].id)~iri']

  genres:
    sources:
      - ['1000_movies_metadata.json~jsonpath', '$[*].genres[*]']
    s: ex:genre_$(id)
    po:
      - [a, film:Genre]
      - [film:id, $(id), xsd:integer]
      - [rdfs:label, $(name)]

  studios:
    sources:
      - ['1000_movies_metadata.json~jsonpath', '$[*].production_companies[*]']
    s: ex:studio_$(id)
    po:
      - [a, film:FilmStudio]
      - [film:id, $(id), xsd:integer]
      - [rdfs:label, $(name)]
```

Here first for the rdf:type film:Film it maps to each movie some data properties like hasSpokenLanguage or hasProductionCountry. Then it also maps the two object proeprties hasFilmStudio and hasGenre to the classes Genre and FilmStudio. For these other two classes of rdf:type film:Genre/Filmstudio it maps the subject of the genre_/studio_id and proceeds to givge the data properties of id and rdfs:label (name).

2.3 Subgraph of combined Json-Files

Following is a subgraph of the resulting combined TTL-serialized file.
The order of the merging did not make lines with the same movie id after each other, so the following does not match one movie, but are excerpts from multiple as can be seen by the different film_id.

```
<http://semantics.id/ns/example#actor_100>
        <http://semantics.id/ns/example/film#filmID>  "20";
        <http://semantics.id/ns/example/film#filmID>  "277";
        <http://semantics.id/ns/example/film#filmID>  "834";
        <http://semantics.id/ns/example/film#fullName>  "Scott Speedman";
        <http://semantics.id/ns/example/film#gender>  "2";
        a       <http://semantics.id/ns/example/film#Actor> .
```

```
<http://semantics.id/ns/example#director_10>
        <http://semantics.id/ns/example/film#filmID>  "609";
        <http://semantics.id/ns/example/film#fullName>  "Tobe Hooper";
        <http://semantics.id/ns/example/film#gender>  "2";
        a       <http://semantics.id/ns/example/film#Director> .


<http://semantics.id/ns/example#editor_131>
        <http://semantics.id/ns/example/film#filmID>  "120";
        <http://semantics.id/ns/example/film#fullName>  "John Gilbert";
        <http://semantics.id/ns/example/film#gender>  "2";
        a       <http://semantics.id/ns/example/film#Editor> .

<http://semantics.id/ns/example#scriptwriter_126>
        <http://semantics.id/ns/example/film#filmID>  "847";
        <http://semantics.id/ns/example/film#fullName>  "Bob Dolman";
        <http://semantics.id/ns/example/film#gender>  "2";
        a       <http://semantics.id/ns/example/film#ScriptWriter> .

<http://semantics.id/ns/example#film_871>
        <http://semantics.id/ns/example/film#hasCast>
        ↪  <http://semantics.id/ns/example#cast_636>;
        <http://semantics.id/ns/example/film#hasDirector>
        ↪  <http://semantics.id/ns/example#director_13246>;
        <http://semantics.id/ns/example/film#hasEditor>
        ↪  <http://semantics.id/ns/example#editor_13255>;
      <http://semantics.id/ns/example/film#hasScriptWriter>
      ↪  <http://semantics.id/ns/example#scriptwriter_13248>;

<http://semantics.id/ns/example#cast_100>
        <http://semantics.id/ns/example/film#hasCastActor>
        ↪  <http://semantics.id/ns/example#actor_100>;
        <http://semantics.id/ns/example/film#hasCastCharacter>  "Don";
        <http://semantics.id/ns/example/film#hasCastCharacter>  "Michael
        ↪  Corvin";
        a       <http://semantics.id/ns/example/film#Cast> .


<http://semantics.id/ns/example#studio_766>
        <http://semantics.id/ns/example/film#id>  766;
        a       <http://semantics.id/ns/example/film#FilmStudio>;
        <http://www.w3.org/2000/01/rdf-schema#label>  "Blue Tulip Productions"
        ↪  .

<http://semantics.id/ns/example#genre_35>
        <http://semantics.id/ns/example/film#id>  35;
        a       <http://semantics.id/ns/example/film#Genre>;
        <http://www.w3.org/2000/01/rdf-schema#label>  "Comedy" .


<http://semantics.id/ns/example#film_100>
        <http://semantics.id/ns/example/film#hasFilmStudio>
        ↪  <http://semantics.id/ns/example#studio_13419>;
```

```
        <http://semantics.id/ns/example/film#hasFilmStudio>
        ↪  <http://semantics.id/ns/example#studio_1382>;
        <http://semantics.id/ns/example/film#hasFilmStudio>
        ↪  <http://semantics.id/ns/example#studio_146>;
        <http://semantics.id/ns/example/film#hasFilmStudio>
        ↪  <http://semantics.id/ns/example#studio_21920>;
        <http://semantics.id/ns/example/film#hasFilmStudio>
        ↪  <http://semantics.id/ns/example#studio_491>;
        <http://semantics.id/ns/example/film#hasGenre>
        ↪  <http://semantics.id/ns/example#genre_35>;
        <http://semantics.id/ns/example/film#hasGenre>
        ↪  <http://semantics.id/ns/example#genre_80>;
        <http://semantics.id/ns/example/film#hasProductionCountry>  "United
        ↪  Kingdom";
        <http://semantics.id/ns/example/film#hasSpokenLanguage>  "English";
        <http://semantics.id/ns/example/film#id>  "100";
        <http://semantics.id/ns/example/film#originalTitle>  "Lock, Stock and
        ↪  Two Smoking Barrels";
        a       <http://semantics.id/ns/example/film#Film> .
```

3 SPARQL Queries

3.1 Name Update Query

First I ran the query to insert my own name and replace a director. I chose Steven Spielberg. Therefore now all movies he did should show up as directed by my name and student id.

```
PREFIX film: <http://semantics.id/ns/example/film#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

DELETE {
  <http://semantics.id/ns/example#director_488> film:fullName "Steven
  ↪  Spielberg" .
}
INSERT {
  <http://semantics.id/ns/example#director_488> film:fullName
  ↪  "YourFullName_1234567" .
}
WHERE {
  <http://semantics.id/ns/example#director_488> film:fullName "Steven
  ↪  Spielberg" .
}
```

3.2 Actor Count Property

In this query I am adding the data property to each object of the class Film that is the number of actors starring in it. This can be helpful when quickly wanting to identify a big or small production and then not having to do the whole normal query.

```
PREFIX film: <http://semantics.id/ns/example/film#>
PREFIX ex: <http://semantics.id/ns/example#>

INSERT {
  ?film film:numberOfActors ?actorCount .
}
WHERE {
```

```
  SELECT ?film (COUNT(?actor) AS ?actorCount)
  WHERE {
    ?film a film:Film ;
          film:hasActor ?actor .
  }
  GROUP BY ?film
}
```

3.3 Cast Summary

In this query I am adding the data property to each object of the class Film that is a summary of all names of actors that worked on a movie. With this it is possible to immediately return all names of actors that starred in the movie. This might give a better overview for movies with many actors.

```
PREFIX film: <http://semantics.id/ns/example/film#>

INSERT {
  ?film film:castSummary ?castSummary .
}
WHERE {
  {
    SELECT ?film (GROUP_CONCAT(?actorName; separator=", ") AS ?castSummary)
    WHERE {
      ?film film:hasActor ?actor .
      ?actor film:fullName ?actorName .
    }
    GROUP BY ?film
  }
}
```

3.4 Female to Male Cast Ratio

This query adds the data property to the Film object that is the ration of the cast of Female to Male, meaning how many women per man worked on the movie. This information might be helpful in identifying historical trends or the status quo in the movie industry.

```
PREFIX film: <http://semantics.id/ns/example/film#>

INSERT {
  ?film film:femaleToMaleRatio ?femaleToMaleRatio .
}
WHERE {
  {
    SELECT ?film
           (COUNT(?maleActor) AS ?maleCount)
           (COUNT(?femaleActor) AS ?femaleCount)
           (COUNT(?femaleActor) / COUNT(?maleActor) AS ?femaleToMaleRatio)
    WHERE {
      ?film film:hasActor ?actor .
      OPTIONAL {
        ?actor film:gender "2" .  # Male actors
        BIND(?actor AS ?maleActor)
      }
      OPTIONAL {
        ?actor film:gender "1" .  # Female actors
        BIND(?actor AS ?femaleActor)
```

```
      }
    }
    GROUP BY ?film
    HAVING (?maleCount > 0)  # Ensure no division by zero
  }
}
```