SDM

Lab 3 - Knowledge Graphs

Diogo Repas

diogo.repas@estudiantat.upc.edu

Universitat Politècnica De Catalunya

Niccolò Morabito

niccolo.morabito@estudiantat.upc.edu

Universitat Politècnica De Catalunya

Section A

For section A, the goal was to load an excerpt of the DBpedia TBOX and ABOX to GraphDB. We followed the instructions as stated without any inconvenience and explored its contents to get acquainted with the interface.

Section B

B.1. TBOX definition

The TBOX defines the terminology component of knowledge bases. In it, the domain of interest and its vocabulary can be detailed as classes, properties, and relationships. For the purposes of this laboratory assignment, the following graph representation of the TBOX was devised.

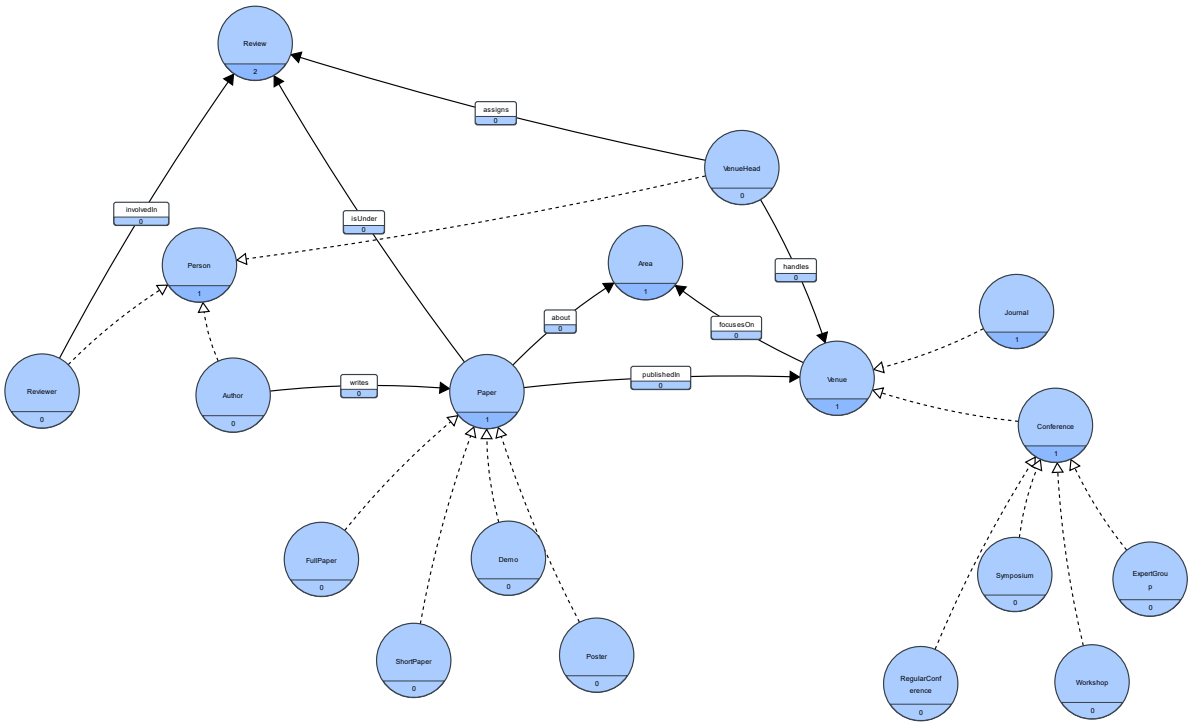

Figure 1. TBOX schema

One relevant aspect of this exercise is to decide what knowledge graph language to use: RDF, RDFS or OWL. We decided to use OWL for its flexibility in describing the underlying data, it is also one of the languages exported by gra.fo, which we used to design the schema above.

Note that there are a few restrictions that we were not able to implement in the given time frame, including restricting Papers of type Poster to Conferences, or the minimum assignment of 2

Reviewers to a Review. Both can be modelled using OWL and were a factor in the knowledge graph language decision.

B.2. ABOX definition

The ABOX is the assertion component of knowledge bases. For its generation, the Jena framework, namely its RDF API, was used. This library can read a document containing RDF statements defining a TBOX to create an ontology model such as the one in Figure 1. After that, it is possible to create the triples representing the assertions in the ABOX that can be finally exported into a file.

Firstly, instances of the main concepts were created from non-semantic data: Author, Paper, Area, Review, VenueHead, Conference and Journal. For these, it was enough to read the same csv files used in the first laboratory assignment and, for each row, create an instance in RDF.

The following method was used to create each instance:

```
model.createIndividual(URI, class);
```

The Universal Resource Identifier (`URI`) is composed of the Universal Resource Locator (URL) and the resource name (which we defined as the concatenation of the class name [className] and the unique identifier [id]). The `class` is a Resource object that represents the class in the TBOX.

All the unique identifiers were taken directly from the corresponding tabular file except for Reviews, since a Review in the first laboratory assignment was a bridge table between paper and reviewer. To overcome this issue, the identifier used in the resource name takes the form: authorId + "@" + paperId.

For the VenueHead concept to represent a chair or an editor, some data had to be generated. For this purpose, we used an online generator[1] to generate a csv file with two columns: an alphanumeric id, and a venueHeadName column. 300 instances were generated with this schema.

Moreover, it is possible to specify the properties (instance's attributes) using the method:

```
instance.addProperty(property, base.createTypedLiteral(row[n]));
```

that will create a triple `<instance URI>, <property URI>, literal .`, where `literal` is the n-th value of the current row.

As required, at least one property is defined for every concept:

```
Author -> personalfullname

Paper -> papertitle

Area -> areaname

VenueHead -> venuename

Conference -> numseats

Journal -> numrewards
```

Journal and Conference inherit `venuename` from the superclass Venue and they have an additional numerical attribute each. The data used in Lab1 contained a `numReviewers` column for both journals and conferences; it could, therefore, have been added to the superclass Venue as a

[1] https://generatedata.com/

common attribute. However, since we already had a common property `venuename` we decided to give a new meaning to the integer value according to the class (number of awards for Journal and number of seats for Conference).

For simplicity, all the instances of Paper and Conference are instances of the corresponding main concept, and we did not generate instances for subclasses (FullPaper, Demo, Symposium, Workshop, etc.). We could have just randomly generated a type for each paper/conference and simply initialise an instance using the corresponding class instead of Paper/Conference, but it was not necessary for the purposes of this project.

For the properties that connect two concepts, we had a few files from the first laboratory assignment that worked as bridge tables, while others had to be generated or processed. Since for the purpose of this project we do not need to model volumes for journals, nor editions for conferences, the data for the property `publishedIn` between Paper and Journal or Paper and Conference are obtained simply by joining two files respectively from the original data.

Randomly selected VenueHead identifiers were used to associate with each review (for the `assigns` relationship), and each Journal or Conference (for the `handles` property). Similarly, for the `focusesOn` property we had to randomly associate n areas to each Venue, where n was a random number between 1 and 10.

The code for the changes or the generation explained in this section can be found in the file BDMA-G12-B-B2-MorabitoRepas.py. The Java code for the ABOX definition using the Jena framework can be found in the file BDMA-G12-B-B2-MorabitoRepas.java.

B.3. Create the final ontology

The connection between TBOX and ABOX is managed by the Jena framework, for instance, when creating an individual, its resource type can be specified so that the triple (<URI-individual> rdf:type <URI-resource>) can be generated automatically. The line of code responsible for this implicit mapping is explained in the previous section for individual creation and can be found in the file BDMA-G12-B-B2-MorabitoRepas.java.

An entailment regime is an extension of the SPARQL semantics that not only specifies which entailment relation is used, but also which graphs and queries are to be considered “well-formed”. For example, using the RDFS entailment regime, a vocabulary entailment stronger than the RDF entailment regime, we are able to infer the rdf:type of sub-classes, making the ontology much more usable. An example of this would be finding all Venues. With RDF entailment, no values will be returned since all instances in our ontology are either a Conference or a Journal. On the other hand, with RDFS entailment, the rdf:type is inferred implicitly. To set the entailment regime, the GraphDB ruleset of RDFS Plus was selected.

The following are some simple statistics about the generate knowledge graph and the queries used to obtain them. We assume that the main classes and properties are those in the namespace prefixed by `http://www.sdm.com/schema/`.

Number of classes: 34

```
SELECT (count(distinct ?class) as ?numClasses) WHERE {
      ?class a rdfs:Class .
}
```

Number of properties: 34

```
SELECT (count(distinct ?property) as ?numProperties) WHERE {
      ?property a rdf:Property .
}
```

Number of instances of the main classes: 606544

```
PREFIX : <http://www.sdm.com/schema/>
SELECT (count(distinct ?instance) as ?numInstances) WHERE {
    ?instance ?property ?class .
    FILTER(isUri(?class) && STRSTARTS(STR(?class), STR(:)))
}
```

Number of triples that use the main properties: 2691024

```
PREFIX : <http://www.sdm.com/schema/>
SELECT (count(?instance) as ?numTriples) WHERE {
    ?instance ?property ?class .
    FILTER(isUri(?property) && STRSTARTS(STR(?property), STR(:)))
}
```

B.4. Querying the ontology

Task 1 – Find all Authors.

```
PREFIX : <http://www.sdm.com/schema/>
select DISTINCT ?s where {
      ?s a :Author .
}
```

Task 2 – Find all properties whose domain is Author.

```
PREFIX : <http://www.sdm.com/schema/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select DISTINCT ?s where {
      ?s rdfs:domain :Author .
}
```

Task 3 – Find all properties whose domain is either Conference or Journal.

```
PREFIX : <http://www.sdm.com/schema/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select DISTINCT ?s where {
      {?s rdfs:domain :Conference .}
      UNION
```

```
      {?s rdfs:domain :Journal .}
}
```

Task 4 – Find all the Papers written by a given Author that were published in database Conferences.

```
PREFIX : <http://www.sdm.com/schema/>
select DISTINCT ?paper where {
      ?author a :Author ;
            :writes ?paper ;
            :personfullname "Hui Xu" .

    ?paper a :Paper;
           :publishedin ?venue .

    ?venue a :Conference;
           :focuseson ?area .

    ?area a :Area ;
          :areaname ?areaname .

    FILTER regex(?areaname, "database")
}
```