Knowledge Graph Lab

Semantic Data Management

Asha Seif
Kainaat Amjid

Submitted On 26th May, 2022

B.1 TBOX Definition

TBox statements are the terminology component of a knowledge graph that is used to describe a domain of interest by defining classes and properties as a domain vocabulary. TBOX is used as a schema or data model of a graph. In this lab, we have created a TBOX using **RDFLib**, a python package working for RDF, and using the **Grafo** tool for graphical representation of our model.

The model consists of the following components:

1. **Classes and their respective Subclasses**

Person: is a class created to describe the concept of a person, this class is further extended to generate a subclass of concept **Author**, **Chair**, **Reviewer, and Editor**. We have made the assumption that these subclasses are performing different roles which could be explained more in the object properties and data properties section.

The prefix definition used in all the code snippet is defined as follows:

```
prefix ns1: <http://exampleDb.org/> .
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
ns1:Person a rdfs:Class .
ns1:author rdfs:subClassOf  ns1:Person .
ns1:chair rdfs:subClassOf  ns1:Person .
ns1:reviewer rdfs:subClassOf  ns1:Person .
ns1:editor rdfs:subClassOf ns1:Person .
```

Paper: this class describes the concept of a paper that will be written by an author and can be published in a conference or journal and also can be reviewed by a reviewer. This class is further extended to subclasses; **Poster, Demo Paper, Short Paper, and Full Paper**.

```
ns1:Paper a rdfs:Class .
ns1:demoPaper rdfs:subClassOf  ns1:Paper .
ns1:poster rdfs:subClassOf  ns1:Paper .
ns1:shortPaper rdfs:subClassOf  ns1:Paper .
ns1:fullPaper rdfs:subClassOf ns1:Paper .
```

Venue: describes the concept of where the paper is submitted by an author, it extends to subclasses **Journal** and **Conference**. The subclass Conference is further extended to subclasses; **regular Conferences, workshops, expert Groups, and symposiums.**

```
ns1:venue a rdfs:Class .
ns1:conference rdfs:subClassOf  ns1:venue .
ns1:journal rdfs:subClassOf  ns1:venue .
ns1:regularConferences rdfs:subClassOf  ns1:conference .
```

```
ns1:symposiums rdfs:subClassOf ns1:conference .
ns1:expertGroups rdfs:subClassOf  ns1:conference .
ns1:workshops rdfs:subClassOf ns1:conference .
```

Publication: describes the concept of a paper published in either journal or conferences after being accepted for publication.

```
ns1:Publication a rdfs:Class .
```

Review: hold the concept of the reviews submitted by a reviewer of a given paper, with a decision of the reviewer whether the paper is being accepted or rejected for publication.

```
ns1:review rdfs:Class .
```

Area: hold the concept of in which area the paper is related, this class is extended to three subclasses which are **DB**, (related to databases contents), **ML** (related to machine learning contents), and **NLP** (related to natural language processing). Here we have made an assumption that a paper can be categorised as related to one or more subclasses of the area based on the keywords presented in the abstract of the paper.

```
ns1:area a rdfs:Class .
ns1:db rdfs:subClassOf  ns1:area .
ns1:ml rdfs:subClassOf  ns1:area .
ns1:nlp rdfs:subClassOf  ns1:area .
```

2. Object Properties with their respective Domain and Range

wrote: is the property which describes the relationships between an **author** (which is the domain) and **paper** (which is the range)

```
ns1:wrote a rdf:Property ;
        rdfs:domain ns1:author ;
        rdfs:range ns1:paper .
```

relatedto: is the property which describes the relationships between **paper** (domain) and **area** (range).

```
ns1:relatedto a rdf:Property ;
        rdfs:domain ns1:Paper ;
        rdfs:range ns1:area .
```

reviewAbout: is the property which describes the relationships between **review**(domain) and **area** (range).

```
ns1:reviewAbout a rdf:Property ;
        rdfs:domain ns1:review ;
        rdfs:range ns1:Paper .
```

submitted: is the property which describes the relationships between **reviewer**(domain) and **review**(range).

```
ns1:submitted a rdf:Property ;
        rdfs:domain ns1:reviewer ;
        rdfs:range ns1:review .
```

submittedIn: is the property which describes the relationships between **Paper**(domain) and **venue**(range).

```
ns1:submittedIn a rdf:Property ;
        rdfs:domain ns1:Paper ;
        rdfs:range ns1:venue .
```

acceptedfor: is the property which describes the relationships between **Paper**(domain) and **Publication**(range).

```
ns1:acceptedfor a rdf:Property ;
        rdfs:domain ns1:Paper ;
        rdfs:range ns1:Publication .
```

handledByC: is the property which describes the relationships between **conference**(domain) and **chair**(range).

```
ns1:handledByC a rdf:Property ;
        rdfs:domain ns1:conference ;
        rdfs:range ns1:chair .
```

handledByE: is the property which describes the relationships between **journal**(domain) and **editor**(range).

```
ns1:handledByE a rdf:Property ;
        rdfs:domain ns1:journal ;
        rdfs:range ns1:editor .
```

3. Some of Data Properties with their respective Domain and Range

In this part we will explain some data properties of the class concept, all other descriptions are available in the rdfs files available in a source_code folder. Some code snippet used to define this property is as follows:

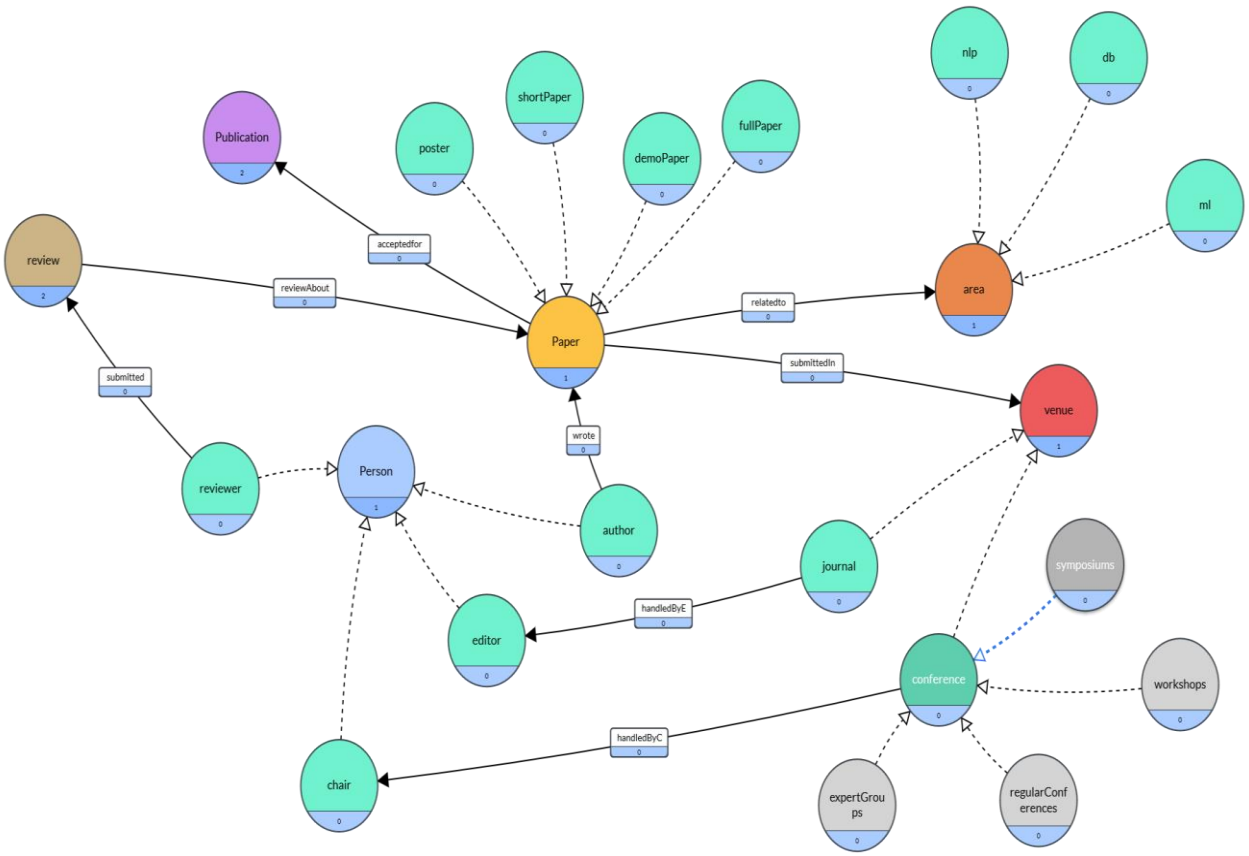
```
ns1:personName a rdf:Property ;
        rdfs:domain ns1:person ;
        rdfs:range xsd:String .
ns1:year a rdf:Property ;
        rdfs:domain ns1:Publication ;
        rdfs:range xsd:Integer .
```

Table showing the data properties of some concept

Data Properties	Domain	Range
personName	Person	String data
paperName	Paper	String data
venueName	venue	String data
year	Publication	Integer data
comments	review	String data
decision	review	String data

The graphical representation of the schema

This is the visual graph of the data model created in the Grafo tool, all the files generated by this tool are available in a folder named **grafo_files** which contain the image file named **schema_model.png** and the rdfs files named as **grafo.ttl.**



B.2 ABOX Definition

ABOX statements are the assertion components which are the facts associated with the TBOX conceptual model. These statements must be compliant with the vocabulary defined in the TBOX, therefore we have defined the ABOX based on the TBOX vocabulary as explained in **B.1** above. To generate the assertions statements we use csv datasets available in the folder named **processed_data** and the **RDFLib** python packages.

The following are some of the python snippet code used to create ABOX and connecting them with tbox statements of some classes concepts

Authors

We use authors.csv file to add the data of authors, we divide the data in the authors four (4) sections to load them as abox statements. To generate different nodes we use authorID as a base URI references in the format ***author+authorID*** examples *author10141271600* as shown in the code below:

```
for index, row in auth_sixty.iterrows():
    #connecting instance with tbox
    g.add((author+"/"+str(row['ID']),RDF.type,author) )
    #connecting person node with person name
    g.add((author+"/"+str(row['ID']),person_name, Literal(row['name'],
datatype=XSD.string)) )
```

In above code this line of code

```
    g.add((author+"/"+str(row['ID']),RDF.type,author) )
```

is connecting author instance in abox with tbox.

Similarly we loaded data in abox for reviewer,chairs,editors and connected them with tbox.

Paper

Paper has 4 subclasses: fullPaper, demoPaper, short paper and poster. First we connect it to tbox then load the data of fullPaper using the property pname(has name) as predicate and literal as an object in the graph. This is an example of code which adds fullPaper data, but all the other subclasses of paper were added in this way.

```
for index, row in Name_articles[Name_articles['Papertype'] ==
'fullPaper'].iterrows():
         #connecting instance with tbox
        g.add((fullPaper+"/"+str(row['ID']),RDF.type,fullPaper) )
         #loading name of paper
        g.add((fullPaper+"/"+str(row['ID']),pname,
Literal(row['title'], datatype=XSD.string)) )
```

Paper written by an authors (wrote property)

To load data into abox we connect already loaded authors in a graph with already loaded articles in the graph with the property “wrote ".

```
for index, row in NArt[NArt['Papertype'] == 'fullPaper'].iterrows():
g.add((author+"/"+str(row['ID_x']),wrote,fullPaper+"/"+str(row['article
_ID'])))
```

Publication

We are first connecting the instance of Publication with tbox. We are storing the publication name and year with URIRef of Publication plus ID of publication with the predicate of pub_name and pub_year respectively and entering data in literal form.

```
for index,row in pN.iterrows():
    #connecting instance with tbox
    g.add((Publication+"/"+str(row['ID']),RDF.type,Publication) )
    g.add((Publication+"/"+str(row['ID']),pub_name,
Literal(row['name'], datatype=XSD.string)))
    g.add((Publication+"/"+str(row['ID']),pub_year,
Literal(row['year'], datatype=XSD.decimal)))
```

Area

Area has three subclasses ml, nlp, db. We are loading names of these subclasses.

```
#assigning name to subclasses of class area
g.add((ml,a_name, Literal('Machine_learning', datatype=XSD.string)))
g.add((db,a_name, Literal('Databases', datatype=XSD.string)))
g.add((nlp,a_name, Literal('Natural_language_Processing',
datatype=XSD.string)
```

Assigning area to papers randomly (with relatedto Predicate)

We created a list named area_list which includes URI’s of subclasses of areas. And then randomly assign it to papers

```
for index, row in NPub[NPub['Papertype'] == 'fullPaper'].iterrows():
g.add((fullPaper+"/"+str(row['article_ID']),relatedto,URIRef(np.random.choi
ce(area_list))) )
```

Paper accepted in publication

We have papers and publications in graphs already. Now we are connecting them with the predicate “accepted”.

```
for index, row in NPub[NPub['Papertype'] == 'fullPaper'].iterrows():
g.add((fullPaper+"/"+str(row['article_ID']),accepted,Publication+"/"+str(ro
w['publisher_ID'])))
```

Conference

Conference has 4 subclasses: workshops, symposiums, expertGroups and regularConferences. We are first connecting the instance of workshop subclass with tbox, then adding name of workshop with property “vname” and workshop is handled by which chair using property “chandles”.

```
for index, row in conf[conf['Conferencetype'] ==
'workshops'].iterrows():
    g.add((workshops+"/"+str(row['ID']),RDF.type,workshops) )
    g.add((workshops+"/"+str(row['ID']),vname, Literal(row['name'],
datatype=XSD.string)) )
g.add((workshops+"/"+str(row['ID']),chandles,chair+"/"+str(row['chairID
']) ))
```

We have used the same properties to load data in other subclasses of conference.

Paper submitted in conference

Paper and conference instances are already loaded in the graph. We will connect which paper is submitted in which conference using property “submittedIn”

```
for index, row in All.loc[(All['Papertype'] == 'fullPaper') &
(All['Conferencetype'] == 'workshops')].iterrows():
g.add((fullPaper+"/"+str(row['article_ID']),submittedIn,workshops+"/"+s
tr(row['ID'])))
```

Similarly for all other subclasses of both paper and conference.

Journals handled by editors

Here we are loading data in a graph for Journals and connecting it to tbox. Editors handle journals, we already have loaded instances of editor in abox. We will connect them to journals they handle.

```
for index, row in jour.iterrows():
    g.add((journal+"/"+str(row['ID']),RDF.type,journal))
g.add((journal+"/"+str(row['ID']),vname, Literal(row['name'],
datatype=XSD.string)))
    g.add((journal+"/"+str(row['ID']),ehandles,
editor+"/"+str(row['editorID'])))
```

Paper submitted In journals

Here we are connecting papers submittedIn journals. We are connecting paper instances in abox graph with journal instances in abox with a predicate “submittedIn”.

```
for index, row in All_j[All_j['Papertype'] == 'fullPaper'].iterrows():
g.add((fullPaper+"/"+str(row['article_ID']),submittedIn,journal+"/"+str
(row['ID'])))
```

Reviews
Reviews are submitted by reviewers. And one paper has at least 2 reviewers. We made this sure in data, that every paper has 2 reviewers.
Here we are first connecting each instance with tbox. Then connect the reviewer with their review. After that we are connecting the review with the decision and text that the reviewer submitted.

```
for index, row in rev.iterrows():
    g.add((review+"/"+str(index),RDF.type,review))
g.add((reviewer+"/"+str(row['author_ID']),R_submitted,review+"/"+str(in
dex)))
      g.add((review+"/"+str(index),r_decision,Literal(row['decision'],
datatype=XSD.string)))
    g.add((review+"/"+str(index),r_comments,Literal(row['description'],
datatype=XSD.string)))
g.add((review+"/"+str(index),r_about,URIRef(uri+"/"+str(row['Papertype'
])+"/"+str(row['publisher_ID']))))
```

B.3 Create the final ontology

To create the final ontology we use the RDFLib python library to link the TBOX and ABOX before loading to the GraphDB tool as explained in **B.2** above .

Inference regime entailment

We have used RDF entailment and RDFS entailment. From that we used the following classes:
RDF.type to indicate that this URIRef is of type class or property, also used to link abox with tbox.
RDFS.Class to indicate that this URIRef is a class.
RDFS.subClassOf to define subclasses of a class.
RDF.Property to indicate the predicate/property.
RDFS.domain to indicate the domain of a property.
RDFS.range to indicate the range of a property.

Example of abox statements used to insert data of an author with the paper he wrote

```
ns1:author5455950435748 a ns1:author ;
        ns1:personName “Chu H. ”^^xsd:string ;
        ns1:wrote ns1:fullPaper153 .
```

Summary of table instances

The final ontology was generated in the format of a turtle file with the extensions of **.ttl**. After generating this file we create a repository in GraphDB and then import the file to store the data and compute the statistics of the instances as summarised in the table below:

Sample of SPARCL statements used to compute the statistics

```
SELECT (COUNT(*) as ?Triples)
WHERE
  {
      { ?s ?p ?o }
  }
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?p (COUNT(?p) as ?pCount)
WHERE
{
  ?s ?p ?o .
}
```

Table Of Instances Statistics

Components	Measures
Number of classes (main classes +subclasses)	23
Number of properties	44
Number of instances of main classes	12097
Number of triples using main properties	42748

B.4 Querying the ontology

After creating the final ontology we use SPARQL feature available in GraphDB to query the ontology, the following are the SPARQL statements used to :

1. **Find all Authors**

To find the information about authors we query from the concept author and retrieve all the names of authors available in the ontology.

```
PREFIX ns1: <http://exampleDb.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?authorName
WHERE {
      ?author rdf:type ns1:author .
      ?author ns1:personName ?authorName .
}  ORDER BY ?authorName
```

2. **Find all properties whose domain is Author**

The properties associated with the domain author without the properties inherited from it superclass Person is obtained by the following query

```
PREFIX ns1: <http://exampleDb.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
WHERE {
      ?propertiesOfAuthor rdfs:domain ns1:author .
}
```

To get all the properties whose domain is author class and with the one inherited from it superclass Person we use UNION clauses

```
PREFIX ns1: <http://exampleDb.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?propertiesOfAuthor
WHERE {
      {
         ?propertiesOfAuthor rdfs:domain ns1:author .
      }
UNION
      {
        ns1:author rdfs:subClassOf ?base .
        ?propertiesOfAuthor rdfs:domain ?base .
      }
}
```

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

To get the properties whose domain is either conference or journal we use UNION clauses to combine the statements which retrieve the domain of conference and journal.

```
PREFIX ns1: <http://exampleDb.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT  *
WHERE {{
      ?properties rdfs:domain ns1:journal .
      }
UNION{
      ?properties rdfs:domain ns1:conference .
      }
}
```

4. **Find all the papers written by a given author that was published in the database conferences.**

Here we made the assumption that the paper is connected to the related area while it is accepted for publication, therefore to find the one which is related to the database conference we have just queried from an author (“Fullwood D.”) whose paper which is published in a database conference.

```
PREFIX ns1: <http://exampleDb.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?title ?authorName ?conferenceName
WHERE {
      ?paper rdf:type ns1:Paper .
      ?author rdf:type ns1:author .
      ?conference rdf:type ns1:conference.
      ?author ns1:personName ?authorName.
FILTER ( ?authorName = "Fullwood D." ) .
      ?author ns1:wrote ?paper .
      ?paper ns1:paperName ?title .
      ?paper ns1:relatedto ns1:db .
      ?paper ns1:submittedIn ?conference .
      ?conference ns1:venueName ?conferenceName .
}
```

REFERENCES

1. https://rdflib.readthedocs.io/en/stable/
2. https://gra.fo/
3. https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/