Polytechnic University of Catalonia

Semantic Data Management

Lab 3: Knowledge Graphs

Author

Dai, Zhongkai
Moure, Ximena

Teachers

Romero, Oscar
Queralt, Anna
Flores, Javier

UPC

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

May 2023

Index

B. Ontology creation

B.1 TBOX definition

The TBOX plays a crucial role in defining the terminology component of knowledge bases. Within the TBOX, it is possible to provide a detailed description of the domain of interest along with its vocabulary, encompassing classes, properties, and relationships.

We created our TBOX in a programmatic manner using ***RDFLib*** from Python. To create the graphical representation of the model depicted below we used **Grafo**.

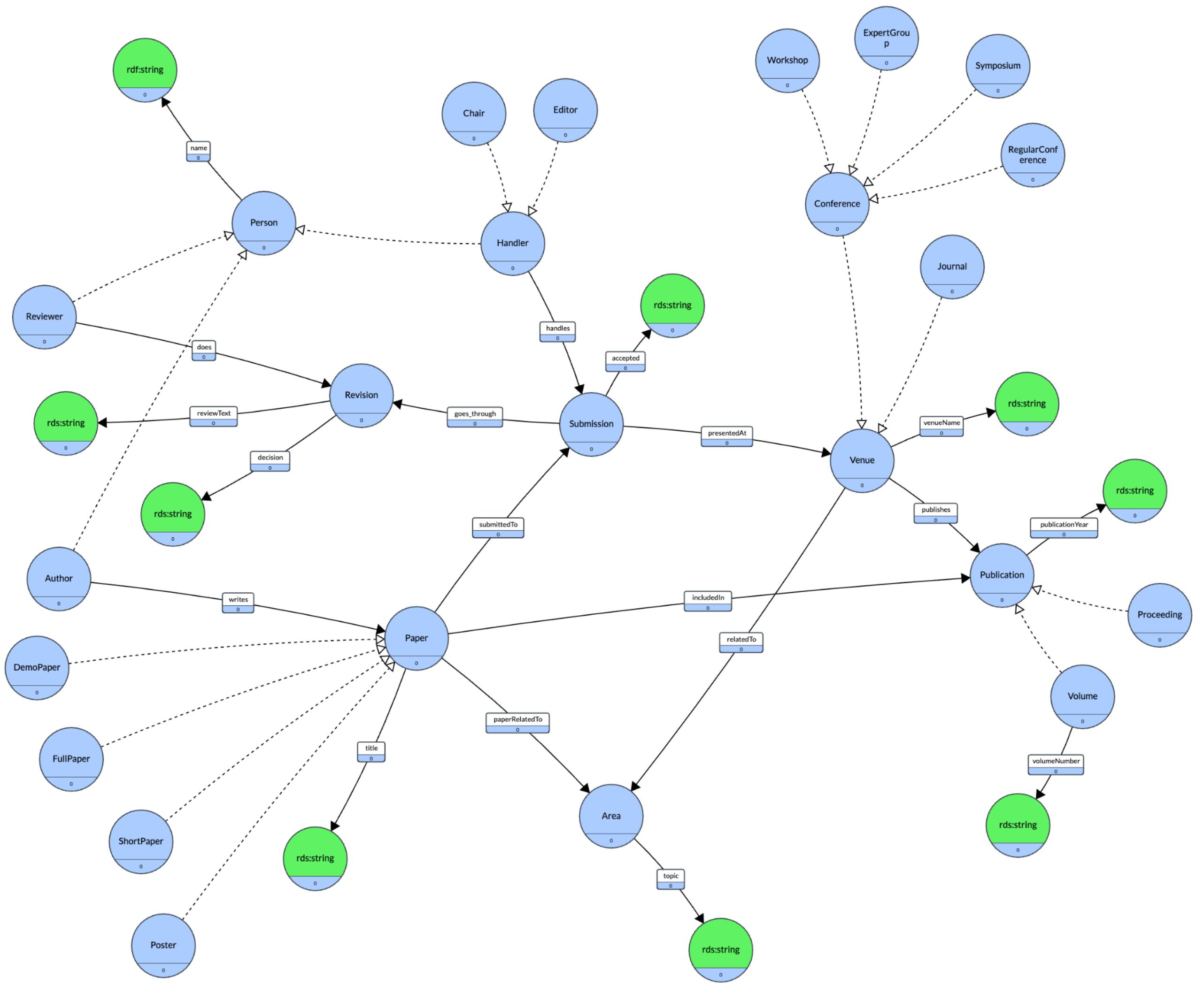

Figure 1: Graphical representation

To download the graph for better visualization, click the following link: tbox.png

To better understand our design we explain some of the main decisions we made:

- “Submission” class: we created this node that gathers everything with respect to a paper submission. It has a property “accepted” which represents if the paper was accepted to be published or not, this happens when the majority of reviewers do a positive revision for the paper. By having this node it is easier to know (easier to get the information) of which papers have been accepted at a venue and thus are published. Also by having this node we are able to connect the Handler (chairs and editors) with the entire process, there is no need to create a new relation between Handler and Revision since we already know that the Handler manages all the submission.
- We added “Person” as a super-class for “Author”, “Reviewer” and “Handler”. This last one is a super-class to “Chair” and “Editor”. With this hierarchy we established a clear relationship between the concepts. Also, it allows the subclasses to inherit properties from the super-class and one important fact is that it enables more expressive querying and reasoning capabilities.
- We added “Venue” as a super-class to “Conference” and “Journal”. The same advantages described before for defining hierarchies apply here.
- Once a paper is accepted it is Published either on a Proceedings (if it was presented at a Conference) or on a Volume (if it was presented at a Journal) and thus it becomes a Publication. This is why we decided to create a class Publication which is a super-class to “Proceeding” and “Volume” that is connected to Venue with “publishes”.
- Each reviewer can reject or accept a submission of a paper and give his/her opinion. This is why we say that a Submission “goes_through” different “Revision” which are done by a Reviewer and have a decision and a text associated with it.
- “Included_in" is to know which papers are included in a publication. Only accepted papers are included in publications.

Although RDFS is the chosen language for building our knowledge graph, it is worth noting that OWL offers more advanced features, particularly in terms of using cardinality constraints. For instance, in the project, we need to ensure that every paper is reviewed by a minimum of two reviewers. While this constraint cannot be directly implemented in RDFS, it is possible to do so in OWL. However, it's important to consider that OWL`s inference process is more complex and computationally expensive compared to RDFS.

The code is in the file ***11C-B.1-DaiMoure.py*** and the output file is *tbox.ttl*.

B.2 ABOX definition

We also used ***RDFLib*** from Python.

First of all, we looked at the csv files from the first project and decided which of them can be reused:

- conference.csv: it has one column with the name of the conferences.
- journal.csv: it has one column with the name of the journals.
- writes.csv: it has two columns with the paper title and the name of the author that wrote the paper.

All the other data was generated from scratch:

- Areas: we created three Area nodes with the topics machine learning, database and natural language processing.
- Conferences: from the csv file we created four types of conferences (Workshop, Symposium, Regular_conference, Expert_group) and assigned them an Area based on their number (Conference1 to machine learning, Conference2 to database, Conference3 to natural language processing, …).
- Journals: from the csv file we created the Journal nodes and assigned them an Area the same way we did with Conferences.
- From writes.csv we generated all the other data:
    - Papers: we created four types of papers (Short_paper, Poster_paper, Demo_paper, Full_paper) and from the first column of writes.csv we assigned its paperTitle. We also assigned an Area for each Paper the same way we did with Conferences and Journals, this way **papers on an Area can only be presentedAt or published by Conferences or Journals of the same Area.**
    - Submissions: we created the Submission nodes. If the node was odd ("Submission1"), we assigned "yes" to accepted, meaning that the Submission was accepted. Otherwise, we assigned "no" to accepted.
    - Chairs: we decided that Chairs handled Short_paper and Poster_paper. Both of them are presentedAt Conferences but only Short_paper is published and includedIn a Proceeding. **Here, we satisfy the fact that a Poster_paper can only be presentedAt a Conference from the project statement.**
    - Editors: we decided that Editors handled Demo_paper and Full_paper. Both of them are presentedAt Journal but only Demo_paper is published and includedIn a Volume
    - Revisions: for each Submission we assigned randomly from 2 to 5 Revisions. **Here, we satisfy the fact that Chairs (respectively, Editors) assign at least 2 reviewers to the submitted paper.**
    - Reviewers: for simplicity, each Reviewer does a Revision, so we created a new Reviewer node for each Revision. Only those Submissions with odd numbers are accepted ("Submission1"), otherwise it is rejected. This way, the accepted Submissions only have positive Revisions and the rejected Submissions only have negative Revisions.
    - Authors: we created the Author nodes and from the second column of writes.csv we assigned its personName.

Between all the nodes, we also created the properties in order to satisfy the TBOX graph.

In most cases we only included one attribute per concept but this was just to keep it simple when randomly generating the data. If we wanted to add more attributes it would be simple to do it, for instance we could have added an abstract to every paper by just adding the following to the tbox and then creating the instances with it:

g.add((ns.abstract, RDF.type, RDF.Property))

g.add((ns.abstract, RDFS.domain, ns.Paper))

g.add((ns.abstract, RDFS.range, RDFS.Literal))

The code is in the file ***11C-B.2-DaiMoure.py*** and the output file is *abox.ttl*.

Take into account that due to the fact that some of the data is randomly generated if you run the abox script you will get a different result from it than the one we provided in the project. The abox.ttl file you get will have different data than the one we provided, so the queries on the last section will give different results.

B.3 Create the final ontology

Due to the fact that we used ***RDFLib*** to generate both the TBOX and the ABOX, when creating the instances for the ABOX, the ABOX is already linked with the TBOX via rdf:type. For example, in the TBOX we have:

```python
# Define the classes
g.add((ns.Person, RDF.type, RDFS.Class))
g.add((ns.Author, RDF.type, RDFS.Class))

# Define the subclasses
g.add((ns.Author, RDFS.subClassOf, ns.Person))
```

In the ABOX we have: (ns.Author1 is linked with ns.Author via rdf:type)

```python
g.add((ns.Author1, RDF.type, ns.Author))
```

We considered RDFS entailment, which is a basic form of inference that infers additional triples based on the RDFS vocabulary. RDFS entailment allows for reasoning about subclass and subproperty relationships, as well as inferring that every instance of a class is also an instance of its superclasses.

To link the ABOX with the TBOX via rdf:type, we saved the following rdf:type links to explicitly generate thanks to reasoning from the previous example:

- Every instance of "Author" is also an instance of "Person" and "RDFS.Class".

When importing both the Tbox and Abox to our repository in GraphDB, we configured it with the RDFS(Optimized) ruleset. According to GraphDB, this ruleset enables the application of standard model-theoretic RDFS semantics. It includes support for inferencing related to subClassOf relationships, type inference as well as subPropertyOf relationships. In the image below the class hierarchy is depicted.

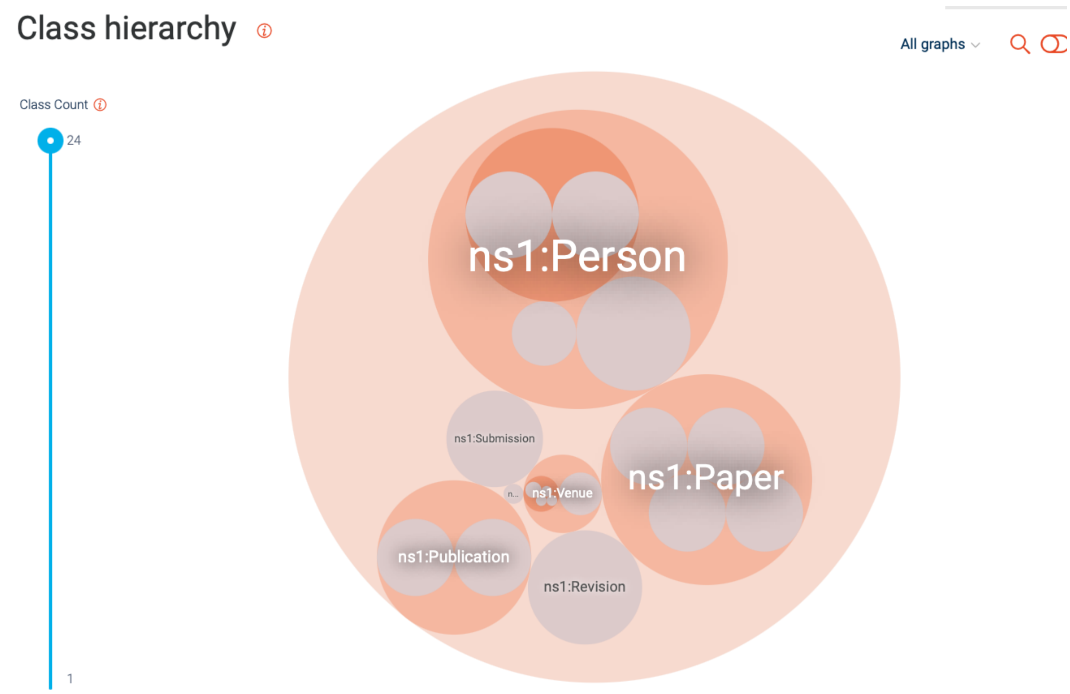

Figure 2: Class Hierarchy

When looking at the overall statistics of our repository, we observe an expansion ratio of 1.4, which is determined by the number of implicit instances compared to the total statements. Additionally, the repository includes a total of 15777 inferred statements. This can be seen in the image below.

Repository P3	
Location:	Local
Type:	Graphdb
Access:	Read/write
Total statements:	54,193
Explicit:	38,416
Inferred:	15,777
Expansion ratio (total/explicit):	1.41

Figure 3: Overall statistics

The following are some basic statistics about the resulting knowledge graph.

Number of classes (not including classes like rdfs:Class, rdfs:Literal, etc): 24

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ns1: <http://example.org/>
SELECT (COUNT(DISTINCT ?class) AS ?numClasses) WHERE {
  ?class a rdfs:Class .
  FILTER(isUri(?class) && STRSTARTS(STR(?class), STR(ns1:)))
}
```

Number of classes (including everything): 38

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT (count(distinct ?class) as ?numClasses) WHERE {
     ?class a rdfs:Class .
}
```

Number of properties: 24

```
SELECT (COUNT(DISTINCT ?p) AS ?n_properties) WHERE { ?s ?p ?o}
```

Number of triplets: 45715

```
SELECT (COUNT(*) AS ?n_triplets) where{?s?p?o }
```

Number of instances: 10753

```
PREFIX ns1: <http://example.org/>
SELECT (count(distinct ?instance) as ?numInstances) WHERE {
    ?instance ?property ?class .
FILTER(isUri(?class) && STRSTARTS(STR(?class), STR(ns1:))) }
```

Number of triplets that use the main properties: 27608

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

How many times each property has been used:

```
SELECT ?property (COUNT(?property) AS ?Total) WHERE { ?s ?property
?o . }
GROUP BY ?property
ORDER BY DESC(?Total)
```

	property	Total
1	rdf:type	"17955"^^xsd:integer
2	ns1:personName	"4641"^^xsd:integer
3	ns1:does	"3541"^^xsd:integer
4	ns1:decision	"3541"^^xsd:integer
5	ns1:reviewText	"3541"^^xsd:integer
6	ns1:goesThrough	"3541"^^xsd:integer
7	ns1:writes	"1000"^^xsd:integer
8	ns1:handles	"1000"^^xsd:integer
9	ns1:paperRelatedTo	"1000"^^xsd:integer
10	ns1:paperTitle	"1000"^^xsd:integer
11	ns1:submittedTo	"1000"^^xsd:integer
12	ns1:accepted	"1000"^^xsd:integer
13	ns1:presentedAt	"1000"^^xsd:integer
14	ns1:includedIn	"500"^^xsd:integer
15	ns1:publicationYear	"500"^^xsd:integer
16	ns1:publishes	"500"^^xsd:integer
17	ns1:volumeNumber	"250"^^xsd:integer
18	rdfs:subClassOf	"69"^^xsd:integer
19	rdfs:subPropertyOf	"33"^^xsd:integer
20	rdfs:domain	"25"^^xsd:integer
21	rdfs:range	"25"^^xsd:integer
22	ns1:venueName	"25"^^xsd:integer
23	ns1:venueRelatedTo	"25"^^xsd:integer
24	ns1:topic	"3"^^xsd:integer

Total Number of properties, object and subjects:

```
SELECT (COUNT(DISTINCT ?subject) AS ?totalsubject) (COUNT(DISTINCT
?predicate) AS ?totalpredicate) (COUNT(DISTINCT ?object) AS
?totalobject)
WHERE{ ?subject ?predicate ?object}
```

	totalsubject	totalpredicate	totalobject
1	"10783"^^xsd:integer	"24"^^xsd:integer	"11834"^^xsd:integer

Number of instances grouped by class:

```
SELECT ?class (COUNT(?instance) AS ?numInstances) WHERE {
  ?instance a ?class .
}
GROUP BY ?class
```

	class	numInstances
1	rdf:Property	"31"^^xsd:integer
2	rdfs:Class	"38"^^xsd:integer
3	rdf:List	"1"^^xsd:integer
4	rdfs:Datatype	"3"^^xsd:integer
5	rdfs:ContainerMembershipProperty	"1"^^xsd:integer
6	ns1:Author	"100"^^xsd:integer
7	ns1:Person	"4641"^^xsd:integer
8	ns1:Short_paper	"250"^^xsd:integer
9	ns1:Paper	"1000"^^xsd:integer
10	ns1:Poster_paper	"250"^^xsd:integer
11	ns1:Demo_paper	"250"^^xsd:integer
12	ns1:Full_paper	"250"^^xsd:integer
13	ns1:Chair	"500"^^xsd:integer
14	ns1:Handler	"1000"^^xsd:integer
15	ns1:Submission	"1000"^^xsd:integer
16	ns1:Editor	"500"^^xsd:integer
17	ns1:Journal	"20"^^xsd:integer
18	ns1:Venue	"25"^^xsd:integer
19	ns1:Area	"3"^^xsd:integer
20	ns1:Reviewer	"3541"^^xsd:integer
21	ns1:Revision	"3541"^^xsd:integer
22	ns1:Proceeding	"250"^^xsd:integer
23	ns1:Publication	"500"^^xsd:integer
24	ns1:Volume	"250"^^xsd:integer
25	ns1:Workshop	"2"^^xsd:integer
26	ns1:Conference	"5"^^xsd:integer
27	ns1:Symposium	"1"^^xsd:integer
28	ns1:Expert_group	"1"^^xsd:integer
29	ns1:Regular_conference	"1"^^xsd:integer

B.4 Querying the ontology

B.4.1 Find all Authors

Unset
```
SELECT ?author WHERE {
      ?author a ns1:Author .
}
```

	author
1	ns1:Author1
2	ns1:Author10
3	ns1:Author100
4	ns1:Author11
5	ns1:Author12

We just show the first five results.

B.4.2 Find all properties whose domain is Author

Unset
```
SELECT ?property
WHERE {
  ?property rdfs:domain ns1:Author .
}
```

	property
1	ns1:writes

B.4.3 Find all properties whose domain is either Conference or Journal

Unset
```
SELECT ?property
WHERE {
  ?property rdfs:domain ns1:Venue
}
```

	property
1	ns1:publishes
2	ns1:venueName
3	ns1:venueRelatedTo

B.4.4 Find all the papers written by a given author that where published in database conferences

Unset
```
SELECT ?paper
WHERE {
    ?paper ns1:includedIn ?publication .
    ?author ns1:writes ?paper .
    ?author ns1:personName "John T. Fredricksen" .
    ?venue ns1:publishes ?publication .
    ?venue a ns1:Conference .
    ?venue ns1:venueRelatedTo ?area .
    ?area ns1:topic "database" .
}
```

We assumed that the given author's name is John T. Fredricksen.

	paper
1	ns1:Paper305
2	ns1:Paper5
3	ns1:Paper637