Semantic Data Management
Master in Data Science
Project 3: Knowledge Graph

Camila Perez…..……………………camila.perez@estudiantat.upc.edu
Meritxell Arbiol López……………meritxell.arbiol@estudiantat.upc.edu

B1. TBOX definition

- ### TBOX Creation

To define the TBOX for the research publication domain related to Lab 1, we used an open source ontology editor and a framework for building intelligent systems called Protégé, developed by the Center for Biomedical Informatics Research at Stanford University School of Medicine.

When choosing which language to select we have taken into account that RDF is a language that allows expressing facts, and that although RDFS is the schema language for RDF that allows not only to detail constraints on individuals and relations used in RDF triples but also to express semantic relations between classes and properties, it turns out to be too broad at the same time. Therefore, for defining constraints, OWL offers much more possibilities by allowing us to use semantic constructs, making it essential to derive more meaning (reasoning and inference) from just a few facts.

Since Protégé fully supports the World Wide Web Consortium's OWL 2 Web Ontology Language and RDF specifications, we began the creation of our TBOX. Using the "class hierarchy" view, we created the necessary classes and subclasses, along with the relevant class expressions, including "disjoint with". We then add the properties of the objects and data in the respective views, indicating their domain and range.

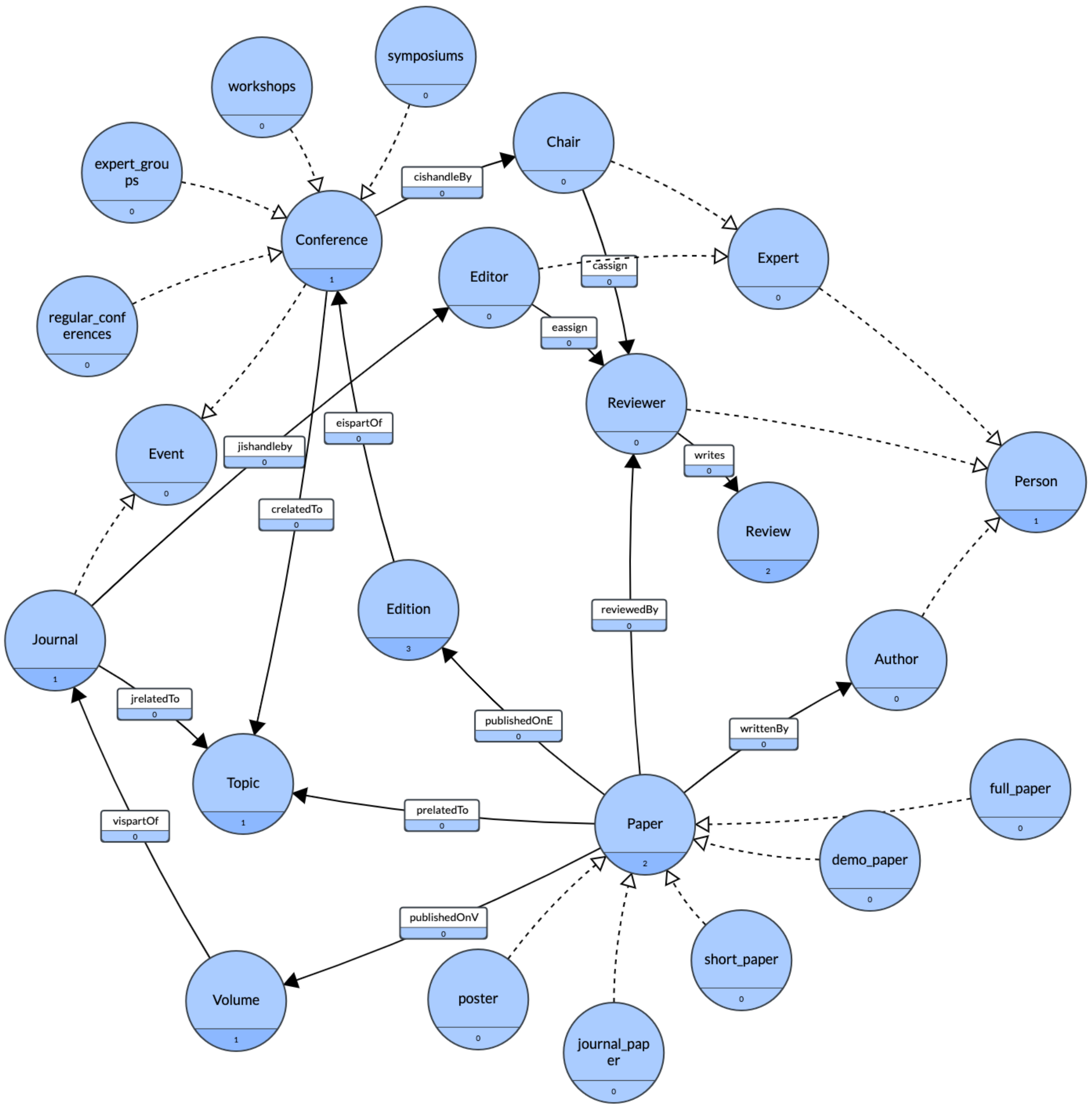
We defined 7 classes, corresponding to Paper, Person, Event, Edition, Volume, Topic and Review. We also added 5 subclasses for the different types of Paper, 3 subclasses of Person to identify author, reviewer and expert, which also has two more subclasses for chair and editor; and 2 subclasses of Event; Journal and Conference, with the latter having 4 more subclasses referring to the different types of conferences. We also incorporated 14 object properties, and 12 data properties to fully exploit semantics. We have defined at least one attribute for each class, there are some attributes that are defined at the class level, and there are others that are at the subclass level.

Protegé comes with several reasoners, we use HermiT and the ontology debugging tool to validate the coherence and consistency of this, resulting in a positive test.

The format we use to store our Protege OWL ontology is Turtle Syntax, as it is a widely supported format. However, Protege supports several other formats for uploading and downloading ontologies (RDF/XML, OWL/XML, OBO, and others).

Finally, to obtain the following graphical representation of our TBOX created in Protegé, we used “Gra.fo”, a well-known tool for visual modeling of knowledge graphs.

- Graphical representation



B2. ABOX Definition

As mentioned in the project statement, we have used much of the data we had prepared for the first project we did. Even so, some things needed to be added to have all the information needed according to the statement. We have added, for example, the concept of Chairs and Editors, as well as the fact that there should be at least 2 reviewers for each submitted paper. In our case, we have used data from papers downloaded from DBLP, where there are papers that have only been published, that is why in our case, all reviews are "Accept". We have also added the concepts of the different types of papers and the different types of conferences that may be available.

So we have put all the data together in a single CSV. We have decided to simplify it as much as possible so that we have data for everything but few, that is why we have decided to work with 10 papers. This way we can easily check if the generated ABOX file is correct by importing it into GraphDB.

To create the ABOX file we have done it using the RDFLib Python library. Once the ABOX is generated, we have parsed it to save it into the graph that we are going to use to generate our ABOX.
We have to be careful with the namespaces we use. So we are going to take into account the namespace where we have defined our Ontology, which in our case is *"http://localhost:7200/publications/"*, and we are going to create a new namespace where we define the different instances, which we call it *"http://localhost:7200/publications/data/"*.

Having these spaces defined, we only have to parse each one of our CSV instances, and add them to the graph using the relationship that we already had defined in our Ontology (our ABOX). Here is a short example of how we have parsed the graph, created the two namespaces, parsed the instances *Author* and *Paper* from the CSV, and added them to our graph through the "*writtenBy*" relationship.

```
g.parse("Ontology.owl")
NS =  ("http://localhost:7200/publications/")
VOCAB = Namespace("http://localhost:7200/publications/data/")
paper = row['IDpaper'].replace(' ', '_').replace('(',
'').replace(')', '')
author = row['IDauthor'].replace(' ', '_')
g.add((VOCAB[paper], NS["writtenBy"], VOCAB[author]))
```

Continuing with the previous example, by saying that between the parsed elements of the CSV *paper* and *author* there is the relationship *"writtenBy"*, it is already inferred in our graph that the *paper* element is of type *Paper* and the *author* of type *Author*, since this is defined in our ontology. The same happens for most of the classes of our network, which do not need to be defined in our ABOX, since only the relation is enough, since the type is defined in our TBOX.

B3. Create the final Ontology

We have created the links between TBOX and ABOX programmatically with Python and using the same library as in the previous point: RDFLib.

To link the TBOX and the ABOX we have done it in the same python script in which we have generated the ABOX, called *"CamilaAndMeritxell-SECTIONb.2_b.3-Perez_Arbiol.py"*.
Continuing with what we had already commented in the previous section, to link the TBOX and the ABOX we have done it by parsing the graph of our TBOX file and taking into account the namespace that we have used to define the schema in the TBOX.
Doing this, there are already many inferences because part of what we are going to generate for the ABOX is already defined as the type of what it is in our TBOX.

When specifying the relationship (properties) that exists between the different parsed elements (domain and range), most elements do not need to specify what type they are.

The elements that have the inferred type are: *Paper, Author, Reviewer, Review, Topic, PersonName, TitlePaper, AstractPaper, ReviewDecission, ReviewComment, TopicName, Edition, Conference, Chair, ConferenceName, EditionName, EditionCity, EditionYear, ChairName, Volume, Journal, Editor, JournalTtitle, VolumeYear, EditorName*

Another thing to keep in mind is that when we have a class with several subclasses inside, for example we have the Person class, and inside this we have different subclasses (*Reviewer, Author, Chair* and *Editor*), if we have a property, for example *"personhasaName "*, linked directly to the *"Person"* class, we can declare this relationship with each of the subclasses in our ABOX, here is an example:

```
g.add((VOCAB[paper], NS["writtenBy"], VOCAB[author]))
g.add((VOCAB[author], NS["personhasaName"], VOCAB[author_name]))
```

In the case of the previous example we are inferring that *"author_name"* is a label of *"Person"*, since the property is *"personhasaName"*, and in addition, with the property *"writtenBy"* we are also saying that this person is of type *"Author"* (and in our TBOX we have said that the range of the property *"writtenBy"* is *"Author"*, and the domain of *"personhasaName"* is *"Person"*).

For some elements the inference is not implicit, since we see that it is necessary to specify the type of the element. This happens when there are classes and subclasses, and the range or domain property has been defined in the large class.

In our case this happens for the *TypePaper*, since a "*Paper"* has several subclasses depending on the type that it is, and the properties that we have in the domain or the range have the *"Paper"* class, and not to the different subclasses that compose it. Because of this, it is necessary to specify for each of the *"Paper"* elements that we parse, to declare which type of *"TypePaper"* it is.

Here is a small example of how we have done it, where *"paper"* is the paperID directly parsed from our CSV, and the *"typepaper"* is a string that matches one of the *"Paper"* subclasses that are defined inside our TBOX:

```
g.add((VOCAB[paper], RDF.type, NS[typepaper]))
```

The same happens with the "*TypeConference"*, where all the relationships have been declared with the "*Conference"* class, and not with each of the subclasses. So we have specified in our ABOX each of our conferences to which subclass it belongs in a similar way to the example shown above.

- Summary Table

Metrics	Count
1. Number of classes	23
2. Number of properties	40
3. Number of instances for the main class: “Class:Person”	57
4. Number of triples using the main properties	743
5. Most used property : “rdf: type”	157

Required Queries

1. Number of classes

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ( count(?class) as ?n_classes ) { ?class a owl:Class }
```

2. Number of properties

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

3. Number of instances for the main class

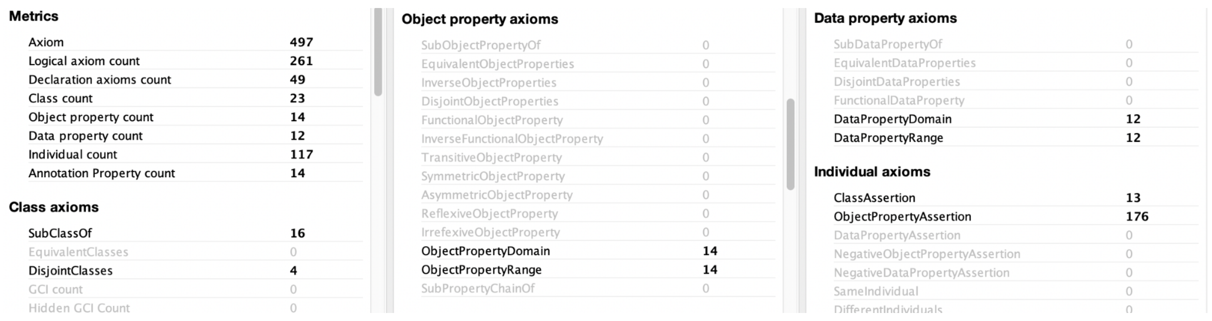
```
PREFIX : <http://localhost:7200/publications/>
SELECT (COUNT(?s) AS ?n_instances)
WHERE {?s a :Person .}
```

4. Number of triplets
```
SELECT (COUNT(*) AS ?n_triplets)
where { ?s ?p ?o  }
```

5. How many times it has been used each property

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

- Metrics provided by Protegé platform



Metrics

Axiom	497
Logical axiom count	261
Declaration axioms count	49
Class count	23
Object property count	14
Data property count	12
Individual count	117
Annotation Property count	14

Class axioms

SubClassOf	16
EquivalentClasses	0
DisjointClasses	4
GCI count	0
Hidden GCI Count	0

Object property axioms

SubObjectPropertyOf	0
EquivalentObjectProperties	0
InverseObjectProperties	0
DisjointObjectProperties	0
FunctionalObjectProperty	0
InverseFunctionalObjectProperty	0
TransitiveObjectProperty	0
SymmetricObjectProperty	0
AsymmetricObjectProperty	0
ReflexiveObjectProperty	0
IrrefexiveObjectProperty	0
ObjectPropertyDomain	14
ObjectPropertyRange	14
SubPropertyChainOf	0

Data property axioms

SubDataPropertyOf	0
EquivalentDataProperties	0
DisjointDataProperties	0
FunctionalDataProperty	0
DataPropertyDomain	12
DataPropertyRange	12

Individual axioms

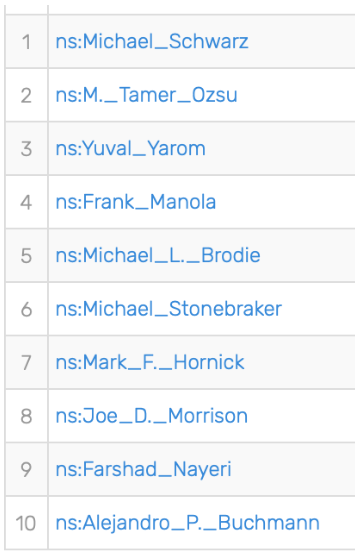
ClassAssertion	13
ObjectPropertyAssertion	176
DataPropertyAssertion	0
NegativeObjectPropertyAssertion	0
NegativeDataPropertyAssertion	0
SameIndividual	0
DifferentIndividuals	0

B4. Querying the Ontology

1. Find all Authors.

```
PREFIX : <http://localhost:7200/publications/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?authorname
WHERE {
    ?person :personhasaName ?authorname .
    ?person rdf:type :Author .
}
```

As expected, the query returns the name of the 10 authors that we have defined in our ABOX:



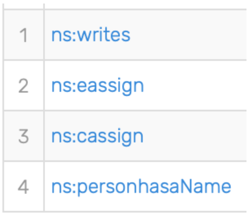
1	ns:Michael_Schwarz
2	ns:M._Tamer_Ozsu
3	ns:Yuval_Yarom
4	ns:Frank_Manola
5	ns:Michael_L._Brodie
6	ns:Michael_Stonebraker
7	ns:Mark_F._Hornick
8	ns:Joe_D._Morrison
9	ns:Farshad_Nayeri
10	ns:Alejandro_P._Buchmann

2. Find all properties whose domain is Author.

```
PREFIX : <http://localhost:7200/publications/>
PREFIX ns: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?property
WHERE {
     ?property ns:domain :Author
}
```

This query does not return any results since we do not have properties defined in the Author domain, since we have decided to define them at the Person level. If instead of Author we look for the properties defined in the Person domain, we do get a list of properties:

```
PREFIX : <http://localhost:7200/publications/>
PREFIX ns: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?property
WHERE {
     ?property ns:domain :Person
}
```



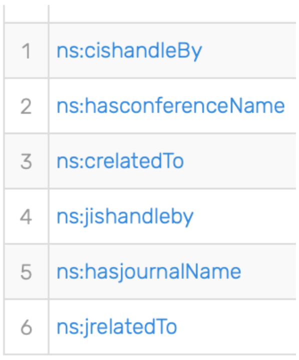
1	ns:writes
2	ns:eassign
3	ns:cassign
4	ns:personhasaName

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

```
PREFIX : <http://localhost:7200/publications/>
PREFIX ns: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?property
WHERE {
    {?property ns:domain :Conference}
    UNION
    {?property ns:domain :Journal}
}
```

For this query we get a total of 6 properties:

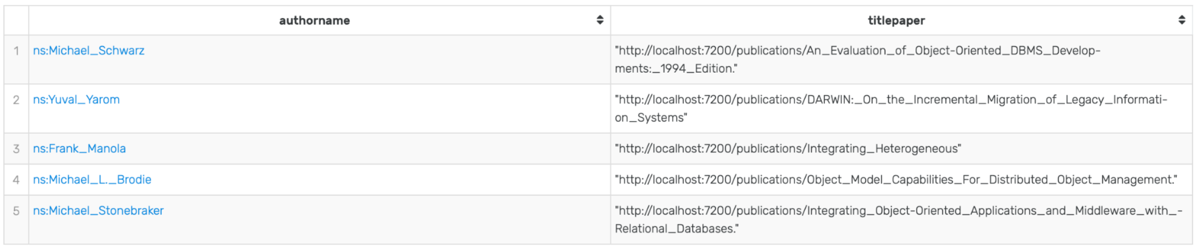


1	ns:cishandleBy
2	ns:hasconferenceName
3	ns:crelatedTo
4	ns:jishandleby
5	ns:hasjournalName
6	ns:jrelatedTo

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

```
PREFIX : <http://localhost:7200/publications/>
SELECT ?authorname (group_concat(?titlepaper) as ?titlepaper)
WHERE {
    ?paper :writtenBy ?author .
    ?author :personhasaName ?authorname .
    ?paper :hasaName ?titlepaper .
    ?paper :PublishenOnE ?edition .
}
group by ?authorname
```

We see that we have a total of 5 different papers that have been published in Conferences, along with the name of the author who has written them.



	authorname	titlepaper
1	ns:Michael_Schwarz	"http://localhost:7200/publications/An_Evaluation_of_Object-Oriented_DBMS_Develop-ments:_1994_Edition."
2	ns:Yuval_Yarom	"http://localhost:7200/publications/DARWIN:_On_the_Incremental_Migration_of_Legacy_Informati-on_Systems"
3	ns:Frank_Manola	"http://localhost:7200/publications/Integrating_Heterogeneous"
4	ns:Michael_L._Brodie	"http://localhost:7200/publications/Object_Model_Capabilities_For_Distributed_Object_Management."
5	ns:Michael_Stonebraker	"http://localhost:7200/publications/Integrating_Object-Oriented_Applications_and_Middleware_with_-Relational_Databases."