IN3067/INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 2: Creating (small) RDF-based Knowledge Graphs

Ernesto Jiménez-Ruiz

Academic course: 2023-2024
Updated: February 26, 2024

Contents

1 Git Repositories **2**

2 Creating an RDF graph **2**

3 Creating triples programatically **3**

4 First steps with the ontology editor Protégé **4**

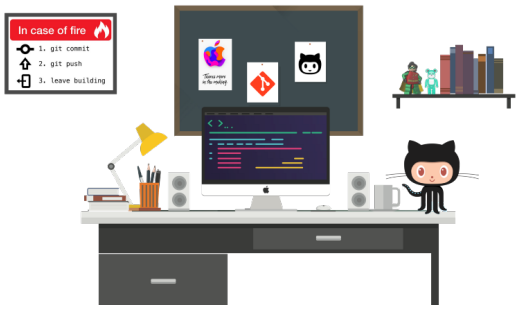
5 FOAF - Friend of a friend (Optional) **6**

6 Solutions **6**

1 Git Repositories

Support codes for the laboratory sessions are available in *github*. There are two repositories for 2024, one in Python and another in Java:

`https://github.com/city-knowledge-graphs`



2 Creating an RDF graph

There are different options to create RDF triples:

- Using your favourite text editor (this Section).
- Using an ontology editor like Protégé (see Section 4).
- Programmatically with Python's RDFlib or Java's Jena API (see Section 3).

Task 2.1: Use a text editor to create triples in Turtle format. Create:

- An entity that represents yourself of type `foaf:Person`.
- Triples for your name and surname.
- Triples for your city and country of birth.
- Triple(s) with the list of languages you speak.
- Triples describing your past or current employer/university, stating the date of start and end (if applicable).

Tips:

- Select a suitable namespace for your entities.
- Define prefixes.
- Reuse vocabulary if possible (*e.g.*, `http://dbpedia.org/resource/Spain`, `https://dbpedia.org/ontology/birthPlace`, `http://xmlns.com/foaf/0.1/name`).
- You may need to use reification or a n-ary relationship.
- Give a *.ttl* extension to your created file.

Task 2.2: Load the created .ttl file with RDFLib or Jena API and print the triples in the graph. This may give errors if the created triples have not been properly formatted. An example script is given. Tip: *See example code in GitHub*.

3 Creating triples programatically

Both RDFlib and Jena API provide methods to populate an empty graph or add triples to an existing one.

RDFLib documentation:

- `https://rdflib.readthedocs.io/en/stable/intro_to_creating_rdf.html`
- `https://rdflib.readthedocs.io/en/stable/rdf_terms.html`

JENA API documentation:

- `https://jena.apache.org/tutorials/rdf_api.html`
- `https://jena.apache.org/documentation/notes/`

Task 2.3: Repeat Task 2.1 with your favourite RDF library. Tip: *See examples in GitHub*.

Task 2.4: Transform Table 1 into triples.

- Try to first create with a text editor.
- Create the triples using RDFLib or Jena.
- Try to create triples by automatically processing the CSV file representing Table 1. Tip: *An example script to load a csv file is given in GitHub. Note that the coursework Part 2 aims at creating triples from a csv file.*

Table 1: Table about companies

Company	Founding year	Headquarters
OST	2017	Oxford
DeepReason.ai	2018	Oxford
Oxstem	2011	Oxford
Oxbotica	2014	Oxford
DeepMind	2010	London

4 First steps with the ontology editor Protégé

Protégé is an editor of OWL ontologies (`https://protege.stanford.edu/`). We will use the latest Desktop version. When opening `Protégé` you are offered to install additional functionalities (*i.e.*, plugins), you can ignore that for now. If you encounter issues installing `Protégé`, there is also a Web version called `WebProtégé` (`https://protegewiki.stanford.edu/wiki/WebProtege`); but the interface is a bit different.

Task 2.5: Check the *short demo Protégé video* in moodle and try to follow these steps:

- Open Protégé (Protégé opens with an empty ontology).
- Change the default *Ontology IRI* in "Active ontology/Annotations". *e.g.*, `http://www.semanticweb.org/in3067inm713/2024/lab2`.
- Create *prefixes* in "Active ontology/Ontology Prefixes".
  - city: `http://www.example.org/university/london/city#`
  - foaf: `http://xmlns.com/foaf/0.1/`
  - lab2: `http://www.semanticweb.org/in3067inm713/2024/lab2`.
- Create *classes* Person and Module in Tab "Entities/Classes":
  - `city:Module`
  - `foaf:Person`
  - `city:Lecturer`
- Create a *subclass* relationship between `foaf:Person` and `city:Lecturer`.
- Create an *object property* `city:teaches` in Tab "Entities/Object properties" .
- Create a `rdfs:label` annotation for each of the entities (*e.g.*, `city:Lecturer rdfs:label` "Lecturer")
- Create a *data property* `foaf:name` in Tab "Entities/Data properties".
- Create *individuals* in Tab "Entities/Individuals".
  - `city:ernesto`
  - `city:inm713`
  - `city:inm373`
  - `city:in3067`
- Assign *types* to `city:ernesto` and `city:inm713`. Select a type from class hierarchy.

- Create *Object property assertions* (*e.g.*, triples with property `city:teaches`) associated to the individual `city:ernesto` and linking to `city:inm713` and `city:inm373`.
- Create a *Data property assertion* (*e.g.*, a triple with property `foaf:name`) associated to individual `city:ernesto`.
- Indicate that `city:inm713` is the same instance as (*i.e.*, `owl:samAs`) `city:in3067`.
- Indicate that `city:ernesto` is teaching `city:inm713` in 2024.
- Perform classification/reasoning.
- Save the created ontology into `turtle` format (*e.g.*, `.ttl`) and `rdf/xml` format (*e.g.*, `.owl`)

Task 2.6: Create the same triples as in Task 2.1 in Protégé, save the file in turtle format, and compare the generated file with yours from Task 2.1.

Tips:

- Protégé (and OWL) does not allow the creation of blank nodes. Solutions involving reification or n-ary relationships can use a named individual.
- Protégé can however annotate axioms with annotation properties (*i.e.*, no logical implications), see Figure 1 for example.
- In OWL, instead of having a list of values, one should create different property assertions.

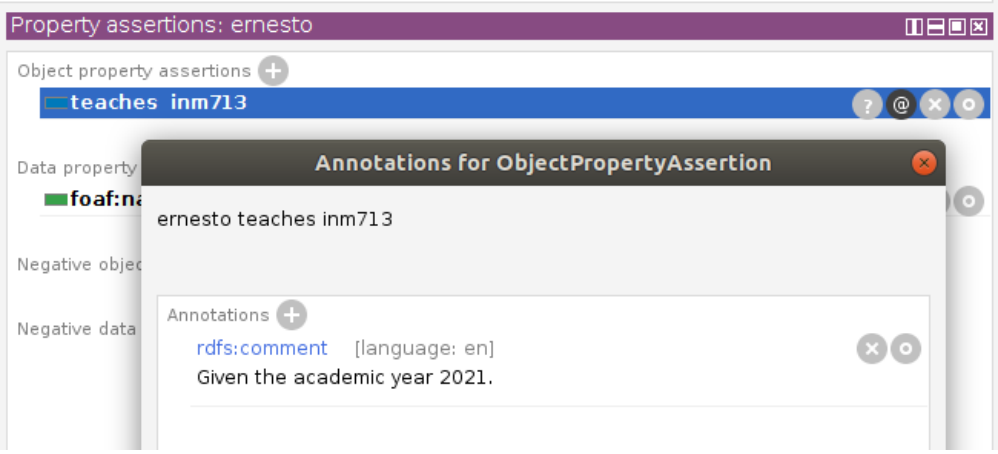


Figure 1: Creating annotation for a property assertion in Protégé.

5 FOAF - Friend of a friend (Optional)

The FOAF project is an old project where RDF was the core technology. FOAF aimed at being a simple technology to make it easier to share and use information about people and their activities. See `https://en.wikipedia.org/wiki/FOAF_(ontology)` for more information.

Task 2.7: (Optional) Go to the FOAF-a-Matic online service (`http://ldodds.com/foaf/foaf-a-matic.en.html`) and create your own FOAF file. Add myself as friend (*See Also* field; no strong commitments here :-)): `https://raw.githubusercontent.com/city-knowledge-graphs/foaf/main/ernesto_foaf.rdf`.

Task 2.8: (Optional) Upload your generated FOAF RDF file to `https://github.com/city-knowledge-graphs/foaf` via a pull-request. Alternatively, send me the generated FOAF RDF file.

FOAF was an interesting project, but there are currently more recent efforts. For example, Wikidata (`https://www.wikidata.org/`) is a community driven knowledge graph including both manually and automatically created entries. Check my entry: `https://www.wikidata.org/wiki/Q56614973`.

6 Solutions

Task 2.1: I have added in the GitHub repository a file `Solution_Task2.1.ttl` with the model solution. I have reused some predicates and resources from DBpedia. *e.g.*,:

- `http://dbpedia.org/resource/City,_University_of_London`
- `http://dbpedia.org/resource/Italian_language`
- `http://dbpedia.org/ontology/birthPlace`

Tip: to get candidate DBpedia entities, google the entity name followed by DBpedia (*e.g.*,. "Oxford DBpedia") and get the URI from the suggested DBpedia page. In Week 5 we will use the DBpedia look-up to do this programmatically. This service provides a fuzzy search functionality to match candidate entities to a given input string.

Alternative to reification. As we saw in the lecture notes, one can also create a new instance that tries to cover the n-ary relationship. For example, when dealing with measurements like "`city:ernesto dbp:weight 70`", if we would like to say that 70 represents *kilogramns*, we could also create a new instance and associate to that instance the `value` and the `units`:

- `city:ernesto dbp:weight city:weight_ernesto .`
- `city:weight_ernesto qudt:value "70"^^xsd:integer .`

- `city:weight_ernesto qudt:hasUnit unit:CentiM .`
- `city:weight_ernesto rdf:type city:Measurement`

Instead of the named individual `city:weight_ernesto` one could also use a blank node. See `Solution_Task2.1.ttl`.

Note that we reuse vocabulary from QUDT (`http://www.qudt.org/`), a semantic specifications for units of measure, quantity kind, dimensions and data types.

Additional examples can be found here: `https://www.w3.org/TR/swbp-n-aryRelations/`.

Task 2.3: The model solutions are in the respective GitHub repositories. Python: `Solution_Task2.3.py` and `Solution_Task2.3.ipynb`.
Java: `Solution_Task2_3.java`.

Task 2.4: The model solution assumes the existence of manual or automatic mapping of the entities in the CSV file (Table 1) to a KG like DBpedia. Instead of DBpedia one could use a custom ontology as we will do in the Part 2 of the coursework.

Manual annotation is time-consuming for very large datasets and KGs. There are however systems that try to (semi)automate the process (*e.g.*, systems participating in the SemTab challenge: `https://www.cs.ox.ac.uk/isg/challenges/sem-tab/`).[1] Automatic systems typically find the semantic type of a column, the relation between columns, and correspondences between cells and entities in the KG. For example, for the Table 1 of Task 2.4:

- Elements in Column 0 are of type `dbo:Company`
- Elements of Column 2 are of type `dbo:City`
- Columns 0 and 1 are related via the predicate `dbo:foundingYear`
- Columns 0 and 2 are related via the predicate `dbo:headquarter`
- Some cells can also be matched to KG entities:
  - `dbr:Oxford`
  - `dbr:London`
  - `dbr:DeepMind`
  - `dbr:Oxbotica`

In Week 5 we will implement a basic system that finds candidate KG entities for a given cell using the DBpedia look-up service.

For the task in this lab, the model solution implements a very basic look-up relying on a very small dictionary with the above entities. A fresh URI is created for the cells without a KG corresponce (*e.g.*, *OST*, *DeepReason.ai*, *Oxstem*). Note that in the Part 2 of the coursework, we will need to create many fresh URIs, only for a subset of the CSV cells we will be able to find an entity in an external KG like DBpedia.

[1] A system for the SemTab could be a good MSc project.

The model solutions are available in the respective GitHub repositories.
Python: `Solution_Task2.4_table.py` and `Solution_Task2.4_table.ipynb`.
Java: `Solution_Task2_4_table.java`.

Task 2.5: I have added in the GitHub repositories the generated ontology from the Protégé demo: `Solution_Task2.5_Protege.ttl` and `Solution_Task2.5_Protege.owl`.

Task 2.6: I have added in the GitHub repositories the model solution generated with Protégé: `Solution_Task2.6_Protege.ttl`.