IN3067/INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 3: Creating (small) RDF-based Knowledge Graphs

Ernesto Jiménez-Ruiz

Academic course: 2021-2022
Updated: February 22, 2022

Contents

1 Git Repositories

Support codes for the laboratory sessions are available in *github*. There are two repositories for 2022, 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.
- Using an ontology editor like Protégé (as we did in Week 2).
- Programmatically with RDFlib (Python) or Jena API (Java).

Task 3.1: 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.

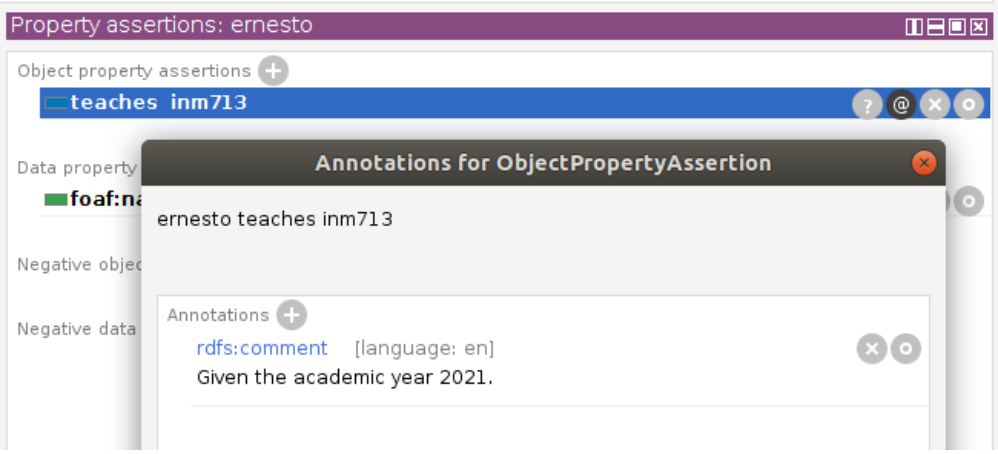

Figure 1: Creating annotation for a property assertion in Protégé.

Task 3.2: 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.

Task 3.3: Create the same triples in Protégé, save the file in turtle format, and compare the generated file with yours.

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.

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/`

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

Task 3.4: Repeat Task 3.1 with your favourite RDF library. Tip: *See examples in GitHub*.

Task 3.5: Transform Table 1 into triples. You can *hard-code* the triples as in Task 3.4 or read directly from the given CSV file. An example script to load a csv file is given.

4 FOAF - Friend of a friend

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 2.1: 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-2022/main/ernesto_foaf.rdf`.

Task 2.2: Upload your generated FOAF RDF file to `https://github.com/city-knowledge-graphs/foaf-2022` via a pull-request. Alternatively, send me the generated FOAF RDF file. Use the FOAF visualiser (`https://foaf-visualizer.gnu.org.ua/`) with your FOAF file (*e.g.*, `https://foaf-visualizer.gnu.org.ua/?uri=https://raw.githubusercontent.com/city-knowledge-graphs/foaf-2022/main/ernesto_foaf.rdf`).

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`.

5 Solutions

Task3.1: I have added in the GitHub repository a file `Solution_Task3.1.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 rdf:value "70"8sd: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_Task3.1.ttl`.

Noe 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/`.

Task3.2: Use the provided script to load a graph to check if the created file is correct.

Task3.3: I have added in the GitHub repositories the model solution generated with Protégé: `Solution_Task3.3_Protege.ttl`.

Task3.4: The model solutions are in the respective GitHub repositories.
Python: `Solution_Task3.4.py` and `Solution_Task3.4.ipynb`.
Java: `Solution_Task3_4.java`.

Task3.5: The model solution assumes the existence of manual or automatic mapping of the CSV file (Table 1) entities to a KG like DBpedia. 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/`). 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 3.5:

- 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 this task, 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*). The model solutions are available in the respective GitHub repositories.
Python: `Solution_Task3.5.py` and `Solution_Task3.5.ipynb`.
Java: `Solution_Task3_5.java`.