A RDF Graph generator for Data Stewards activities in NFDI4BIOIMAGE

Carsten Fortmann-Grote

[2025-03-28 Fri]

Layout the concept

The goal is to convert the Data Steward's (DaSt) activities spreadsheet into a a RDF graph.

Requirements

- Load all sheets in the DaSt spreadsheet as `pandas.DataFrame`.
- Generate a graph that contains all data stewards, their names and qualifications as property values.

Code skeleton

The code consists of module imports, namespace definitions and a number of functions. The `main()` function performs all steps to convert the sheets to graphs.

```
# Imports
<<imports>>

# Namespaces
<<namespaces>>

# Functions
<<functions>>

if __name__ == "__main__":
    main()
```

Imports

```
import pandas
import rdflib
```

```python
from rdflib import Graph, URIRef,  Literal
from rdflib import RDF, FOAF, RDFS
import pandas
from SPARQLWrapper import SPARQLWrapper, JSON
```

Namespaces

We need to define some additional namespaces

```python
dast = rdflib.Namespace("http://purl.nfdi4bi.org/rdf/dast/")
dastp = rdflib.Namespace("http://purl.nfdi4bi.org/rdf/dast#")
WD = rdflib.Namespace("http://www.wikidata.org/entity/")
WDT = rdflib.Namespace("http://www.wikidata.org/prop/direct/")
```

Functions

Load spreadsheet

Our first function loads the spreadsheet and returns the individual sheets as =pandas.DataFrame=s.

```python
def load_spreadsheet(path=None):
    """ Load spreadsheet at path into a pandas DataFrame.

    :param path: The filepath or URL of the spreadsheet to load.
    :type  path: str

    """

    sheets = pandas.read_excel('/home/grotec/GerBI-Cloud/NFDI4BIOIMAGE Consortium/DaSt Team/

    return sheets
```

Test

Let's test our new function.

```python
sheets = load_spreadsheet()
```

```python
assert "Helpdesk & TA Duties" in sheets.keys()
```

Data Stewards

The sheet "Expertise & assigment reque" lists all data stewards and their skills. We'll start with this sheet. The first column, starting from row 7, has the DaSt's names appended by their a

- Data Stewards are entered by their name, sometimes followed by affiliation. Turn every name and affiliation into a foaf:Person object. Add their wikidata and/or

orcid ids if available. Consider usinc vcard instead of foaf.

Function that cleans and returns the DaSts

```
def clean_dasts(sheets):
    dasts = sheets['Expertise & assignment of reque']

    dasts.columns = dasts.loc[6]

    return dasts.drop(axis=0, labels=range(6))
```

A function that returns a set of all DaSt names.

```
def dast_names(dasts_df):
    all_names = set(
        [nm for nm in dasts_df.iloc[:8,0].dropna().values]
    )

    return all_names
```

Test

We test that the function above returns 8 names.

```
sheets = load_spreadsheet()
dasts = clean_dasts(sheets)
all_names_set = dast_names(dasts)

assert len(all_names_set) == 8
```

Conversion to RDF

Now we'll convert the entries in the expertise sheet to triples. Each subject is an instance of `foaf:Person` and of `dast:DataSteward`. We also add a `wdt:P31 wd:Q5` statement and, if available, the wikidata subject URI corresponding to the person. Finally, we add a statement declaring that the person is a participant in NFDI4BIOIMAGE (`wdt:P1344 wd:Q113500855`).

We'll first implement a few functions to query the wikidata sparql endpoint for the wikidata URI given the name. Need a function to run a query on a given endpoint. The function `get_results()` runs a passed query on a given endpoint:

```
def get_results(endpoint_url, query):
    sparql = SPARQLWrapper(endpoint_url)
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)

    return sparql.query().convert()
```

With this, we can now code a function that queries wikidata for the subject of an item that is labelled with a given name. As additional constraints, we assert that the item must be a participant in NFDI4BIOIMAGE. If the query does not yield any results, we return None:

```
def get_wikidata_id(name):
    query = f"""PREFIX wdt: <http://www.wikidata.org/prop/direct/>
      PREFIX wd: <http://www.wikidata.org/entity/>

      select ?person ?personLabel where {{
        service wikibase:label {{bd:serviceParam wikibase:language "en" .}}
          ?person wdt:P31 wd:Q5;
                  wdt:P1344 | ^wdt:P710 wd:Q113500855;
                  rdfs:label ?name .
          filter(regex(?name, "{name}"))
    }}
    limit 1
    """


    endpoint_url = "https://query.wikidata.org/sparql"
    results = get_results(endpoint_url, query)

    if len(results["results"]["bindings"]) > 0:
        return URIRef(results["results"]["bindings"][0]['person']['value'])

    return None
```

Let's test this query execution: Passing a name that does not correspond to a NFDI4BIOIMAGE participant should return None, querying for a name that does indeed correspond to a participant, should return that person's wikidata item's URI:

```
assert get_wikidata_id("Ada Lovelace") is None
assert get_wikidata_id("Mohsen Ahmadi") == URIRef("http://www.wikidata.org/entity/Q91349605'
```

Finally, we have all components to code up the main workhorse function which converts a set of names to a graph which contains all Data Stewards and their statements.

```
def dast2rdf(dast_names):

    graph = Graph()
    graph.bind("wdt", str(WDT))
    graph.bind("wd", str(WD))
    graph.bind("", str(dast))
    graph.bind("this", str(dastp))
    graph.base = dast

    for name in sorted(dast_names):

        first_last = " ".join(name.split(" ")[:-1])
        subj = dast.term(f"DataSteward/{first_last.replace(' ', '_')}")

        graph.add((subj, RDF.type, FOAF.Person)) # Is a person.
        graph.add((subj, WDT.P31, WD.Q5))        # Is a human (wikidata)
        graph.add((subj, RDF.type, dast.term("DataSteward")))
        graph.add((subj, WDT.P1344, WD.Q113500855))        # Participant in nfdi4bioimage
        graph.add((subj, RDFS.label, Literal(f"{first_last}^^xsd:string")))

        wikidata_uri = get_wikidata_id(first_last)

        if wikidata_uri is not None:
            graph.add((subj, RDFS.seeAlso, wikidata_uri))
    return graph
```

We test the last function by passing a list of just one name and assert that `dast2rdf` returns a graph.

```
names = ["Jens Wendt"]
graph = dast2rdf(names)

assert len(graph) == 6
```

Now, we have everything together to get the graph for all Data Stewards.

```
def main():
  sheets = load_spreadsheet()
  dasts = clean_dasts(sheets)
  all_names_set = dast_names(dasts)

  graph = dast2rdf(all_names_set)

  graph.serialize("data_stewards--20250328.ttl")
```

We will now use our `main()` function to generate a graph of all Data Stewards:

```
main()
```