Rapport Projet Web Datamining & semantics

Tom Havyarimana - Oussama El Atrache

Choix du dataset et contexte:

Depuis l'entretien, nous modifiés des éléments à notre projet. Lors de celui-ci nous avions une source de donnée. Nous en avons ajouté une autre, et également une API de météo. Nous avons également modifié les classes.

Nous avons choisi 2 sources de données. Le premier dataset contient les données des stations de vélos en libre service dans la ville de Lyon. Le deuxième contient les mêmes types de données mais dans la ville de Rennes. Ces données sont récupérées en temps réel et mises à jour plusieurs fois par jour. Ils nous renseignent sur l'état de fonctionnement de la station, sa localisation (latitude, longitude), sur le nombre de vélos et d'emplacements de vélo disponibles, ainsi que sur la date de la dernière mise à jour.

Afin de proposer une solution complète, nous avons décidé d'ajouter une information supplémentaire. En passant par une API, nous récupérons en temps réel les données météorologiques de Rennes et Lyon. Ils nous permettent de connaître le temps (par exemple: ciel dégagé), la température, ainsi que la température ressentie.

Afin de récupérer nos données en temps réel, on envoie une requête à chacune des API (celle pour les données des vélos de Lyon et celles pour les données des vélos de Rennes). Pour ce faire, on utilise un script python. A chaque fois que la page web est rechargée, les données sont elles aussi mises à jour. Il en va de même pour les informations météo de la ville de Rennes et de Lyon.

```
def requestAPI(url):
  req = requests.get(url)
  return req.json()
```

Pour commencer, on a la méthode pour l'envoi des requêtes à chaque API.

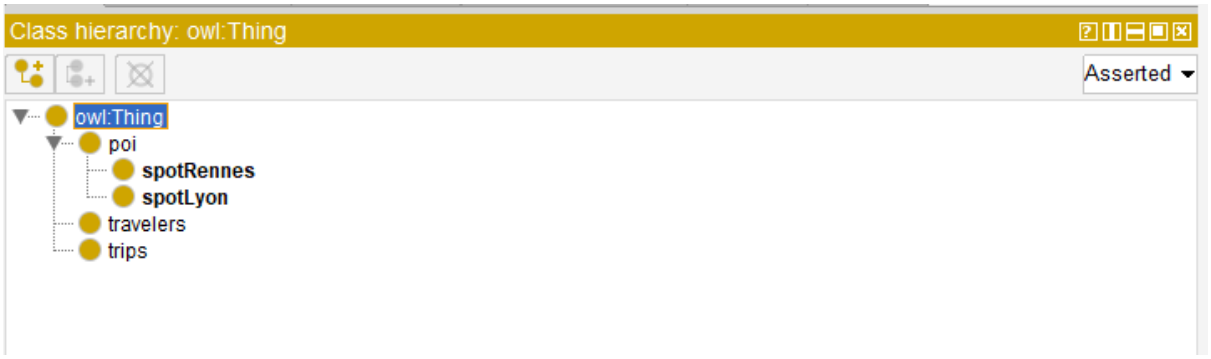
```python
def recup_datas_lyon(req_lyon):
  features = req_lyon["features"]

  stations = []

  for station in features:
    new_station = {}

    properties = station["properties"]

    new_station["nom"] = properties["name"]
    new_station["id_station"] = properties["number"]
    new_station["nb_emplacements"] = properties["bike_stands"]
    new_station["nb_emplacements_dispo"] = properties["available_bike_stands"]
    new_station["nb_velos_dispo"] = properties["available_bikes"]
    new_station["etat"] = properties["status"]
    new_station["latitude"] = properties["lat"]
    new_station["longitude"] = properties["lng"]
    new_station["derniere_maj"] = properties["last_update"]

    stations.append(new_station)

  return stations
```

```python
def recup_datas_rennes(req_rennes):
  records = req_rennes["records"]

  stations = []

  for station in records:
    new_station = {}

    fields = station["fields"]

    new_station["nom"] = fields["nom"]
    new_station["id_station"] = fields["idstation"]
    new_station["nb_emplacements"] = fields["nombreemplacementsactuels"]
    new_station["nb_emplacements_dispo"] = fields["nombreemplacementsdisponibles"]
    new_station["nb_velos_dispo"] = fields["nombrevelosdisponibles"]
    new_station["etat"] = fields["etat"]
    new_station["latitude"] = fields["coordonnees"][0]
    new_station["longitude"] = fields["coordonnees"][1]
    new_station["derniere_maj"] = fields["lastupdate"]

    stations.append(new_station)

  return stations
```

On récupère ensuite les informations qui nous intéressent dans chaque dataset renvoyé par les APIs. Il suffit ensuite de créer les ontologies à partir de ces données.

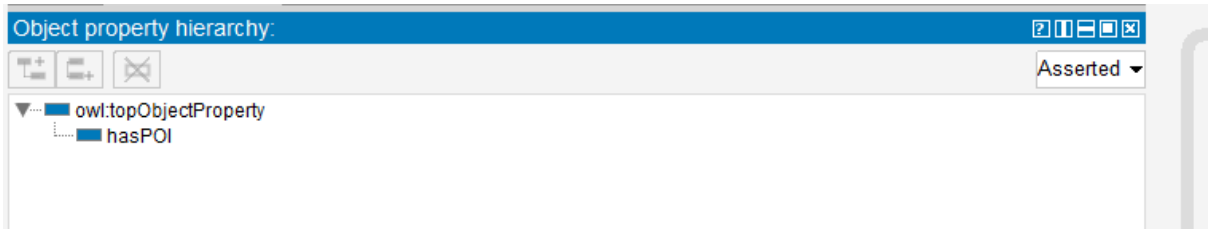
Ontologie:

Classes



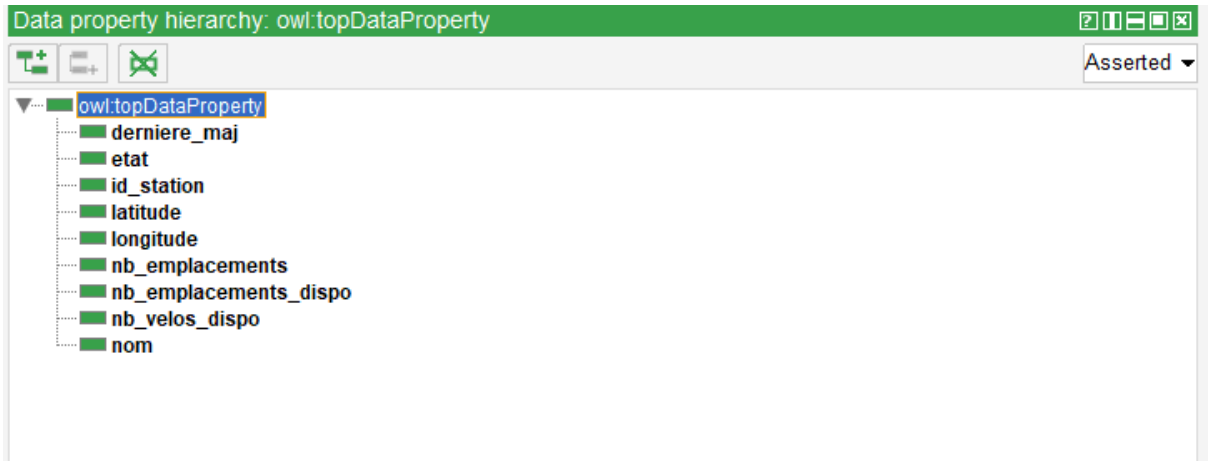
Nous avons créé 5 classes. 2 sous-classes de la classe POI (Point Of Interest) appelés spotRennes et spotLyon. Elles contiennent les informations des stations de vélos. Ensuite, une classe appelée traverlers, contenant les informations des voyageurs, et une autre, trips, contenant les informations des voyages de ces voyageurs à travers les différentes stations de vélos.

Object property



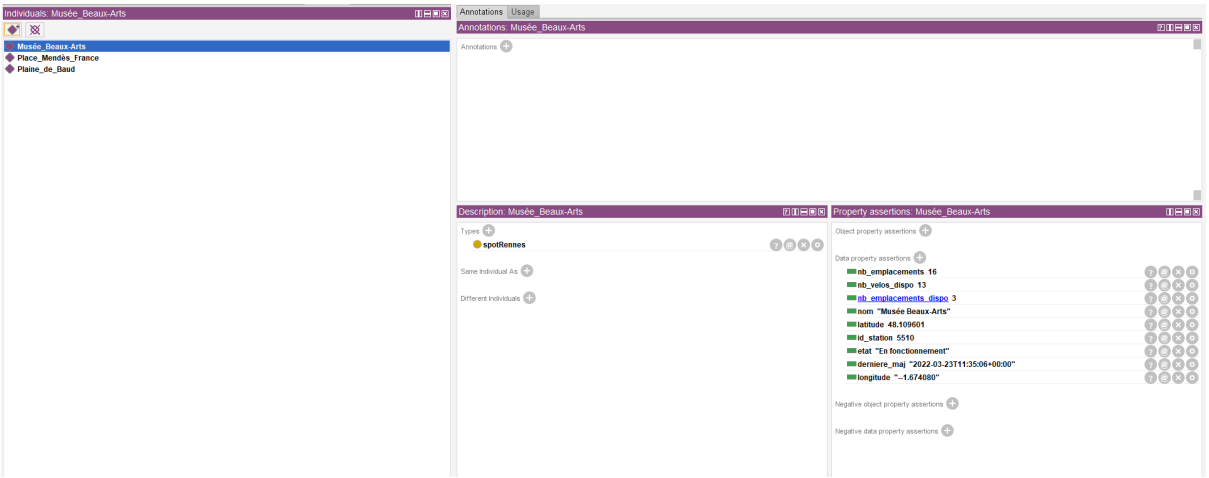
Un object property lié à la classe Point of interest.

Data property



Autant de data property que de colonnes dans notre dataset. Chacune de ces data property à une restriction sur son type (string, int, float, ou datetime).

Individuals



Voici à quoi ressemble un individu. Ici une station de vélo à Rennes appelée Musée Beaux-Arts.
A ce stade, 3 individus ont été ajoutés à la main afin de connaître la forme (le schéma) de sa représentation RDF, dans le but d'automatiser cette génération grâce à un script python.

Peuplement de l'ontologie:

Un fois qu'on a eu un exemple de la manière dont est crée l'ontologie, nous avons créé un script python afin d'automatiser la création des individus.

```
URL = "http://www.semanticweb.org/havyt/ontologies/2022/2/untitled-ontology-14"
URL_TYPE = "http://www.w3.org/2001/XMLSchema"
```

```
def create_ontologies(datas, subclass):

  individuals = []

  for i in range(datas.shape[0]):
    line = datas.iloc[i, :]

    line['nom'] = line['nom'].replace("&", "et")
    name = line['nom'].replace("/", "")

    elements_name = name.split()
    name = "_".join([element.capitalize() for element in elements_name])

    individual = ""
    individual += f"  <owl:NamedIndividual rdf:about=\"{URL}#{name}\">\n"
    individual += f"      <rdf:type rdf:resource=\"{URL}#{subclass}\"/>\n"
    individual += f"      <derniere_maj>{line['derniere_maj']}</derniere_maj>\n"
    individual += f"      <etat>{line['etat']}</etat>\n"
    individual += f"      <id_station rdf:datatype=\"{URL_TYPE}#integer\">{line['id_station']}</id_station>\n"
    individual += f"      <latitude rdf:datatype=\"{URL_TYPE}#decimal\">{line['latitude']}</latitude>\n"
    individual += f"      <longitude rdf:datatype=\"{URL_TYPE}#decimal\">{line['longitude']}</longitude>\n"
    individual += f"      <nb_emplacements rdf:datatype=\"{URL_TYPE}#integer\">{line['nb_emplacements']}</nb_emplacements>\n"
    individual += f"      <nb_emplacements_dispo rdf:datatype=\"{URL_TYPE}#integer\">{line['nb_emplacements_dispo']}</nb_emplacements_dispo>\n"
    individual += f"      <nb_velos_dispo rdf:datatype=\"{URL_TYPE}#integer\">{line['nb_velos_dispo']}</nb_velos_dispo>\n"
    individual += f"      <nom>{line['nom']}</nom>\n"
    individual += "   </owl:NamedIndividual>\n"

    individuals.append(individual)

  return individuals
```

```
individuals_lyon = create_ontologies(lyon_stations, "spotLyon")
individuals_rennes = create_ontologies(rennes_stations, "spotRennes")

individuals_lyon = "\n\n".join(individuals_lyon)
individuals_rennes = "\n\n".join(individuals_rennes)
```

Requêtes SPARQL:

Afficher tous les spots de vélos et leur localisation

```
SELECT ?nom ?latitude ?longitude
WHERE {
       ?p ns:nom ?nom .
       ?p ns:latitude ?latitude .
       ?p ns:longitude ?longitude .}
```

Afficher les spots avec les plus grands nombres de vélos disponibles

```
 SELECT ?p ?nb_velos_dispo
WHERE { ?p ns:nb_velos_dispo ?nb_velos_dispo}
ORDER BY DESC(?nb_velos_dispo)
```

Afficher la localisation des spots avec plus 5 vélos disponibles

```
SELECT ?p ?nom ?nb_velos_dispo
WHERE { ?p ns:nom ?nom .
      ?p ns:nb_velos_dispo ?nb_velos_dispo .
      FILTER (?nb_velos_dispo > 5)}
```

Création d'un nouveau Triple (en utilisant CONSTRUCT)

```
#SELECT ?latitude ?longitude ?localisation
CONSTRUCT {?p ns:loc ?localisation}
WHERE {
      ?p ns:latitude ?latitude .
       ?p ns:longitude ?longitude .
      BIND(concat(?latitude,";",?latitude) AS ?localisation)
      }
```

Y-a-t-il un spot de vélos appelé Place Ampère? (en utilisant ASK)

```
ASK { ?p ns:nom  "Place Ampère"}
```

Description d'un individu à partir de son identifiant (en utilisant DESCRIBE)

```
DESCRIBE ?p WHERE { ?p ns:id_station "7055" }
```

Trouver les spots en détresses, car plus de vélos disponibles ou HS (en utilisant OPTIONAL)

```
SELECT ?p ?etat ?nb_velos_dispo
WHERE  {
         OPTIONAL { ?p ns:nb_velos_dispo 0 .
                        ?p ns:nb_velos_dispo ?nb_velos_dispo .
                            ?p ns:etat ?etat . } .

         OPTIONAL { ?p ns:nb_velos_dispo ?nb_velos_dispo .
                            ?p ns:etat "CLOSED" .
                        ?p ns:etat ?etat . } .

         OPTIONAL { ?p ns:nb_velos_dispo ?nb_velos_dispo .
                            ?p ns:etat ?etat .
                              ?p ns:etat "En panne" .} .
       }
```

Règles SWRL faites sur nos anciennes données

```
[rule1: (?p rdf:type ns:Sans_TPE) (?p ns:type ?type) equals(?type,
"AVEC TPE")    ->  (?p rdf:type ns:Avec_TPE)]
[rule2: (?p rdf:type ns:Avec_TPE) (?p ns:type ?type) equals(?type,
"SANS TPE")    ->  (?p rdf:type ns:Sans_TPE)]
```

Solution finale:

Nous avons utilisé l'API streamlit pour la présentation de notre projet. Il permet de faire facilement une belle page web. Le plus grand challenge a été de rendre nous requêtes interactive. Toutes les requêtes tournent correctement sur le site, sauf celle contenant DESCRIBE, car elle n'est pas encore implémenté dans la librairie rdflib. Egalement une autre ne fonctionne pas, nous comprenons pas encore l'origine de l'erreur. Pour la carte interactive, nous avons utilisé l'API openrouteservice. Elle donne un résultat très satifaisant.

Travelers' trip and directions on interactive map

ID of spot departure:

1001

ID of spot arrival:

1002