Data Semantics - Final Coursework

Mini project

Primenta Aikaterini Maria
ID: 240447658
ec24786

Ontology Description

The main goal of this project is to build a clear and organised semantic model of airports and flights that allows for advanced searching across different aviation data sources. This ontology connects information such as airport locations, flight details, and service data in a meaningful way. By combining data that would normally be separate, it helps users ask more complex questions and discover new insights. The structure of the ontology is as follows:

Classes

1. *Airline*: Represents airline companies.

2. *Airport*: Represents airports. The Airport class has 3 subclasses:
   (a) *Hub Airport:* Represents hub airports, defined as airports with a large number of flights.
   (b) *International Airport*: Represents international airports, defined as airports that accommodate international flights, meaning flights connecting airports located in different countries.
   (c) *Major Airport:* Represents major airports, defined as airports that have at least one runway longer than 3000 meters.

3. *Country*: Represents countries that have airports.

4. *City* (subclass of Country): Represents cities where airports are located.

5. *Runway*: Represents airport runways.

6. *Flight*: Represents flights. The Flight class has 2 subclasses:
   (a) *Domestic Flight*: Represents flights whose departure and arrival airports are located in the same country.
   (b) *Long Flight*: Represents flights with a duration of more than 6 hours (360 minutes).

Object Properties

1. *hasArrivalAirport*: A functional object property with Flight as the domain and Airport as the range. It defines the relationship between a flight and its arrival airport.

2. *hasDepartureAirport*: A functional object property with Flight as the domain and Airport as the range. It defines the relationship between a flight and its departure airport.

3. *hasHub*: An object property with Airline as the domain and Airport as the range. It represents airports that serve as hubs for an airline, meaning the airline operates frequent flights from these airports.

4. *hasRunway*: An object property with Airport as the domain and Runway as the range. It defines the relationship between airports and their runways.

5. *isLocatedIn*: A functional object property with Airport as the domain and Country as the range. It defines the relationship between an airport and the country in which it is located.

6. *servesCity* (subproperty of isLocatedIn): A functional object property with Airport as the domain and City as the range. It specifies the relationship between an airport and the city it serves.

7. *operatedBy*: A functional object property with Flight as the domain and Airline as the range. It indicates which airline operates a given flight.

Data Properties

1. *hasArrivalTime*: A data property with Flight as the domain and xsd:String as the range. It denotes the time of arrival of a flight.

2. *hasDepartureTime*: A data property with Flight as the domain and xsd:String as the range. It denotes the time of departure of a flight.

3. *hasCoordinates*: A data property with Airport as the domain and xsd:String as the range. It represents the coordinates of a given airport.

4. *hasDistance*: A data property with Flight as the domain and xsd:Integer as the range. It represents the distance in miles between the origin and destination airports.

5. *hasDuration*: A data property with Flight as the domain and xsd:Integer as the range. It represents the duration of a flight (air time) in minutes.

6. *hasFlightNumber*: A data property with Flight as the domain and xsd:Integer as the range. It denotes the number of the flight.

7. *hasIATACode*: A data property with Airport as the domain and xsd:String as the range. It indicates the International Air Transport Association (IATA) code of a given airport.

8. *hasICAOCode*: A data property with Airport as the domain and xsd:String as the range. It indicates the International Civil Aviation Organization (ICAO) code of a given airport.

9. *hasName*: A data property with Country as the domain and xsd:String as the range. It represents the name of a country.

10. *hasRunwayLen*: A data property with Runway as the domain and xsd:Decimal as the range. It denotes the length of a runway.

All the above classes and properties were designed based on the available data. Some were also introduced to support SWRL rules that infer new individuals for specific classes and properties. The SWRL rules used are described as follows:

1. *Inferring international airports*: Two airports are considered international if they are connected by a flight between different countries.

2. *Inferring domestic flights*: A flight is considered domestic if both its departure and arrival airports are located in the same country.

3. *Inferring major airports*: Inferring major airports: An airport is classified as a major airport if it has at least one runway longer than 3,000 meters.

4. *Inferring long flights*: A flight with a duration greater than 360 minutes (6 hours) is classified as a long-haul flight.

5. *Inferring hub airports*: An airport with more than four departing flights is considered a hub airport.

Data Sources

For the basic task, the data were retrieved using a SPARQL query from the Wikidata open knowledge base. The dataset included airport names, their codes (IATA and ICAO), the countries and cities where they are located, their coordinates, runways, and runway lengths. For the intermediate task, I downloaded a dataset in CSV format from kaggle (https://www.kaggle.com/datasets/mahoora00135/flights) which contains detailed information about flights.

Queries

To test querying the ontology, I created several example SPARQL queries for both the basic task (which includes only airport data) and the intermediate task (which also includes flight data). All query results are limited to 30 entries for readability. The queries include:

1. List of airports
2. List of airports by country
3. List of airports with IATA code starting with “A”
4. Count airports by country
5. List of flights
6. List of flights by airline
7. List of morning flights (before 12pm)
8. List of airports with the most connecting flights

Challenges

The biggest challenge in this project was obtaining a sufficient amount of data from semantic web sources using SPARQL. DBpedia provided only a limited dataset compared to Wikidata, and both sources had restricted topic coverage and frequently returned errors when executing more complex queries. This limitation made it difficult to build a more sophisticated and interesting ontology with a deeper class hierarchy and a wider variety of object properties. However, by focusing on the data that was reliably available, I was able to develop a meaningful and coherent ontology.

The second major challenge arose during the advanced task involving SWRL rules. Unfortunately, Protégé frequently crashed on my computer when running SWRL rules—even when working with smaller datasets or executing just a single rule. This made it difficult to verify inferred axioms and populate new classes or object properties with the results of rule-based reasoning.

Guide to run the code

This project is divided into two connected parts: the basic task and the intermediate task. Below is a step-by-step guide explaining the functionality of each file and how to execute the code.

Protégé ontology files

airports_ontology.rdf: The initial ontology that defines the necessary classes, object properties, and data properties. It serves as the input for the basic task, where airport data is fetched and populated using a SPARQL query to Wikidata.

populated_airports.owl: The output ontology of the basic task. It contains the full class/property structure along with the populated individuals retrieved from Wikidata.

populated_flights.owl: The final ontology output of the intermediate task. It includes all airport data (from the basic task) and adds flight-related individuals populated from a second data source: a CSV file (flights.csv).

Steps

1. First run the populate_basic.py using the command:

   ```
   python populate_basic.py
   ```

   This script retrieves airport data from Wikidata via a SPARQL query and populates the ontology, producing the output file populated_airports.owl.

2. To verify the ontology, run the query_basic.py using the command:

   ```
   python query_basic.py
   ```

   This script allows querying the airport ontology to check if the data was populated correctly.

3. Then, for the intermediate task, run the populate_intermediate.py as:

   ```
   python populate_intermediate.py
   ```

   This script reads flight information from the CSV file and adds it to the existing ontology, resulting in populated_flights.owl.

4. Again, to verify the ontology, run the query_intermediate.py using:

   ```
   python query_intermediate.py
   ```

   This script enables querying both airport and flight data from the populated ontology.

5. For the advanced task, open the populated_flights.owl in protege and run the SWRL rules in the SWRLTab. If it becomes unresponsive, try running one rule at a time to avoid overloading the reasoner.