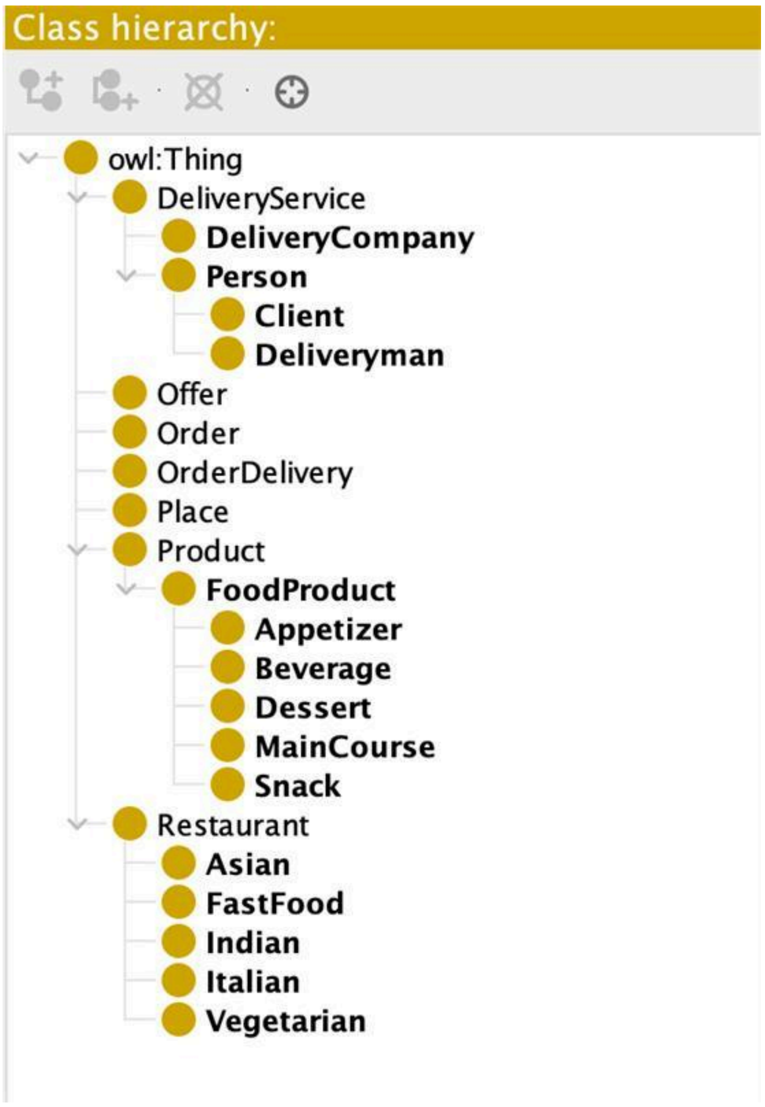
Web data mining & semantics Project

By : Ariane Rousseau & Capucine Foucher

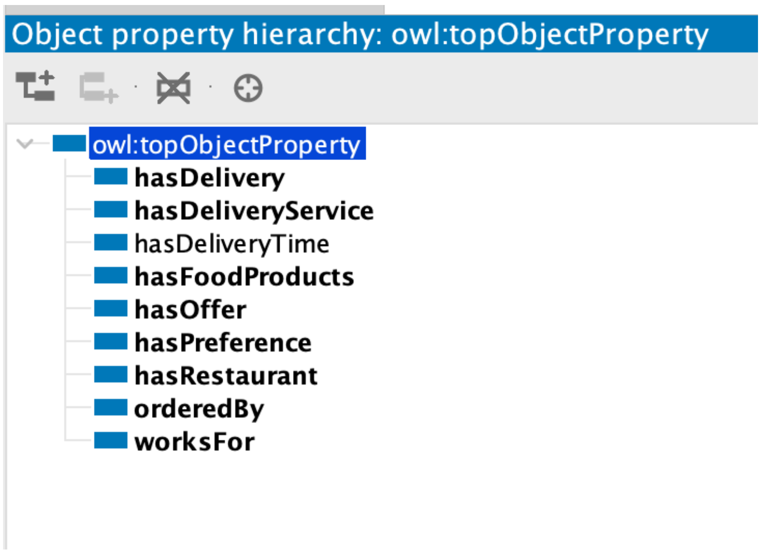
Part I: Modeling the ontology

In this part, we aim to create an ontology, using the Protégé editor, which models the food delivery discovery schema in terms of classes hierarchy, properties and restrictions.



```
Class hierarchy:
owl:Thing
    DeliveryService
        DeliveryCompany
        Person
            Client
            Deliveryman
    Offer
    Order
    OrderDelivery
    Place
    Product
        FoodProduct
            Appetizer
            Beverage
            Dessert
            MainCourse
            Snack
    Restaurant
        Asian
        FastFood
        Indian
        Italian
        Vegetarian
```

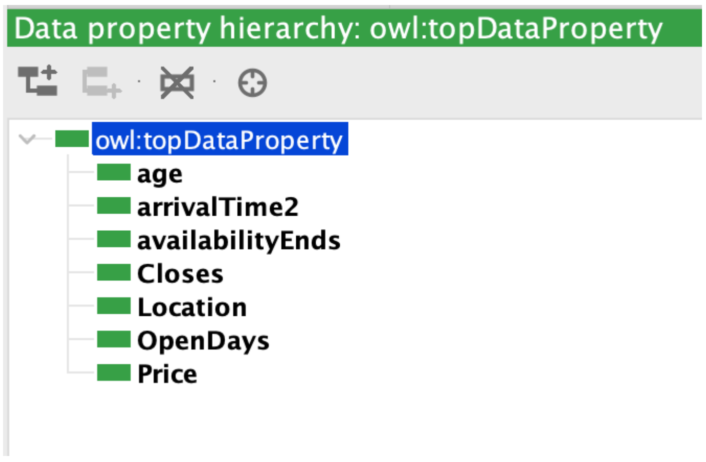
As it is possible to observe, we have created our ontology. We decided to implement 7 classes : DeliveryService, Offer, Order, OrderDelivery, Place, Product, Restaurant. Those classes are the main entities in our food delivery service. Some of these classes were imported with the base ontology schema.org. Also some of these classes have subclasses. They allow us to implement more details to our system. That is why a Product which comes from schema.org has a subclass FoodProduct that can be an Appetizer, a Beverage, a Dessert, MainCourse or a Snack. Moreover, the DeliveryService can either be the DeliveryCompany or a Person that can either be a Deliveryman or a Client. Lastly, we created different types of restaurants : Asian, FastFood, Indian, Italian, Vegetarian.


Object property hierarchy: owl:topObjectProperty

- owl:topObjectProperty
  - hasDelivery
  - hasDeliveryService
  - hasDeliveryTime
  - hasFoodProducts
  - hasOffer
  - hasPreference
  - hasRestaurant
  - orderedBy
  - worksFor

Moreover, we created object properties that allow us to establish relationships and create a structured knowledge base. Here is the list of said object properties : address, hasDelivery, hasDeliveryService, hasDeliveryTime, hasFoodProducts, hasOffer, hasPreference, hasPrice, hasRestaurant, orderedBy, and worksFor. They are linked through domain and range to the different classes.

- hasDelivery only Order (range) : OrderDelivery (domain)
- hasDeliveryService some DeliveryCompany (range): Restaurant, FoodProduct (domain)
- hasFoodProduct some FoodProduct (range) : Offer, Order (domain)
- hasOffer some offer (range) : Restaurant, FoodProduct (domain)
- hasPreference some FoodProduct and some Restaurant (range) : Client (domain)
- OrderedBy only Client (range) : Order, FoodProduct (domain)
- WorksFor only DeliveryCompany (range) : DeliveryMan (domain)
- hasRestaurant some Restaurant (range) : FoodProduct (domain)

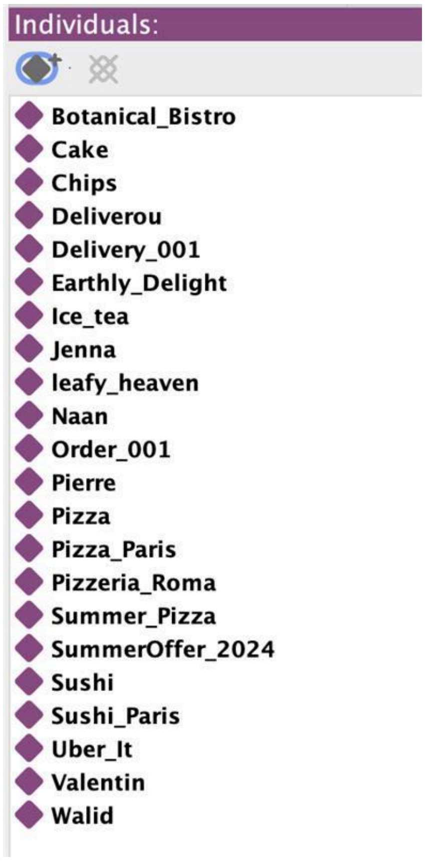

Data property hierarchy: owl:topDataProperty

- owl:topDataProperty
  - age
  - arrivalTime2
  - availabilityEnds
  - Closes
  - Location
  - OpenDays
  - Price

Lastly, we implemented data properties to represent attribute and link data. Those properties are age(int), arrivalTime2(int), availabilityEnds(int), Closes (string), Location(string), OpenDays(string), Price(int).

Part II: Populating the ontology

- We have created some individuals manually to populated our classes :


Individuals:

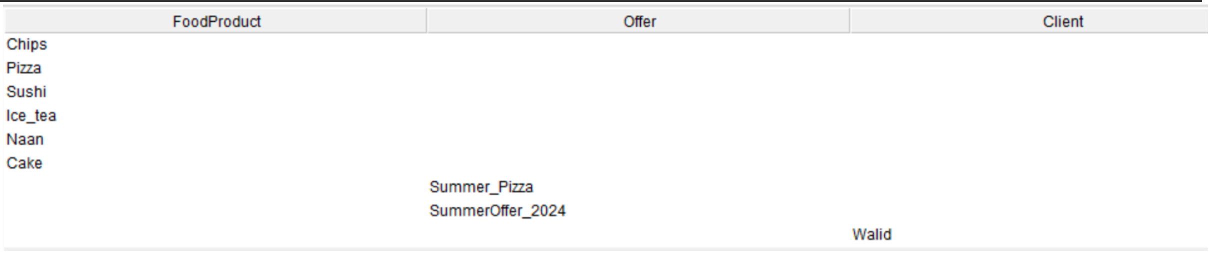
- Botanical_Bistro
- Cake
- Chips
- Deliverou
- Delivery_001
- Earthly_Delight
- Ice_tea
- Jenna
- leafy_heaven
- Naan
- Order_001
- Pierre
- Pizza
- Pizza_Paris
- Pizzeria_Roma
- Summer_Pizza
- SummerOffer_2024
- Sushi
- Sushi_Paris
- Uber_It
- Valentin
- Walid

Part III: Querying the ontology

Write SPARQL queries to response to the following:

1. List the instances of the class food products, offers and customers

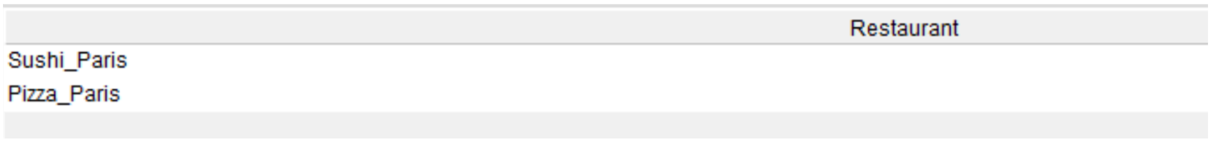
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:
<http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#
>

SELECT ?foodProduct ?offer ?client
WHERE {
  {
       ?foodProduct rdf:type/rdfs:subClassOf* ont:FoodProduct .
  }
  UNION
  {
       ?offer rdf:type ont:Offer .
  }
  UNION
  {
       ?client rdf:type ont:Client .
  }
}
```



FoodProduct	Offer	Client
Chips		
Pizza		
Sushi		
Ice_tea		
Naan		
Cake		
	Summer_Pizza	
	SummerOffer_2024	
		Walid

2. List the name of all Paris restaurants.

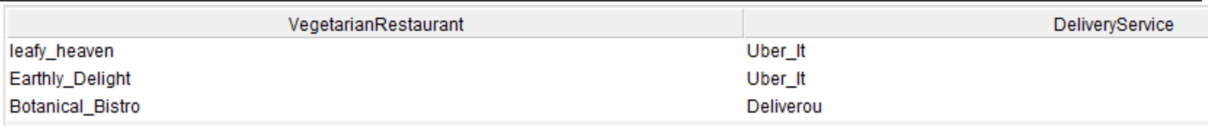
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#>
SELECT ?Restaurant
WHERE {
?Restaurant rdf:type/rdfs:subClassOf* ont:Restaurant .
?Restaurant ont:Location "Paris" .
}
```



Restaurant
Sushi_Paris
Pizza_Paris

3. List the name of all vegetarian restaurant, for each one, display their delivery services

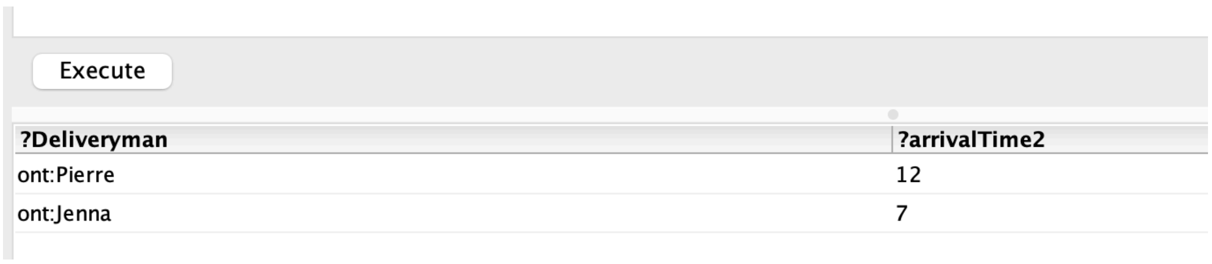
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:
<http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#
>

SELECT ?VegetarianRestaurant ?DeliveryService
WHERE {
?VegetarianRestaurant rdf:type ont:Vegetarian .
?VegetarianRestaurant ont:hasDeliveryService ?DeliveryService .
}
```



VegetarianRestaurant	DeliveryService
leafy_heaven	Uber_It
Earthly_Delight	Uber_It
Botanical_Bistro	Deliverou

4. List the name of delivery men older than 51 years and they can deliver in Lyon in less that 15 minutes

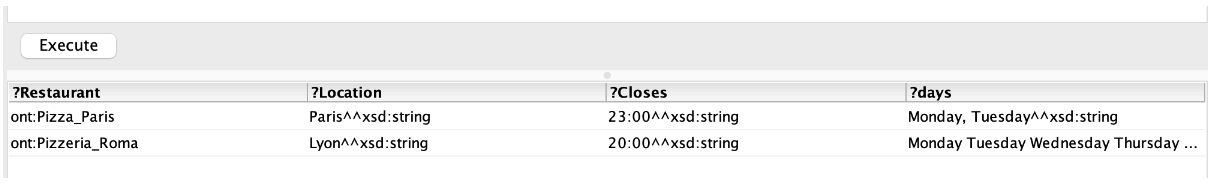
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <htpp://www.w3.org/2001/XMLSchema#>
PREFIX ont:
<http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#
>
SELECT ?Deliveryman ?arrivalTime2
WHERE {
  ?Deliveryman rdf:type ont:Deliveryman .
  ?Deliveryman ont:age ?age .
  ?Deliveryman ont:arrivalTime2 ?arrivalTime2 .
  ?Deliveryman ont:Location "Lyon" .
  FILTER (?age > 51) .
  FILTER (?arrivalTime2 < 20) .
}
```



?Deliveryman	?arrivalTime2
ont:Pierre	12
ont:Jenna	7

5. List of restaurant that serve Italian food for a specific day and where and until when

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <htpp://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#>
SELECT ?Restaurant ?Location ?Closes ?days
WHERE {
  ?Restaurant rdf:type ont:Italian .
  ?Restaurant ont:Location ?Location .
  ?Restaurant ont:OpenDays ?days .
  ?Restaurant ont:Closes ?Closes .
  FILTER(CONTAINS(?days, "Tuesday"))  # Replace "specific_day" with the actual day you
are interested in
}
```



?Restaurant	?Location	?Closes	?days
ont:Pizza_Paris	Paris^^xsd:string	23:00^^xsd:string	Monday, Tuesday^^xsd:string
ont:Pizzeria_Roma	Lyon^^xsd:string	20:00^^xsd:string	Monday Tuesday Wednesday Thursday ...

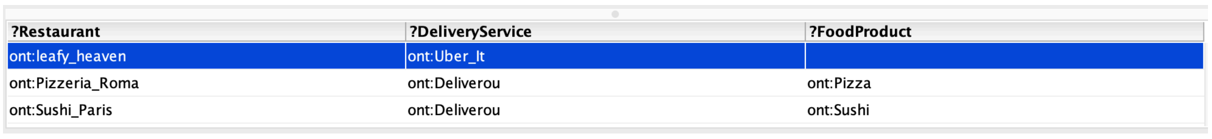
Propose 5 SPARQL queries:

1. A query that contains at least 2 Optional Graph Patterns

Let's create a query to list restaurants and optionally get their delivery services and the food products they offer.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <htpp://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#>
```

```
SELECT ?Restaurant ?DeliveryService ?FoodProduct
WHERE {
  ?Restaurant rdf:type ont:Restaurant .
  OPTIONAL {
    ?Restaurant ont:hasDeliveryService ?DeliveryService .
  }
  OPTIONAL {
    ?Restaurant ont:hasOffer ?Offer .
    ?Offer ont:hasFoodProducts ?FoodProduct .
  }
}
```

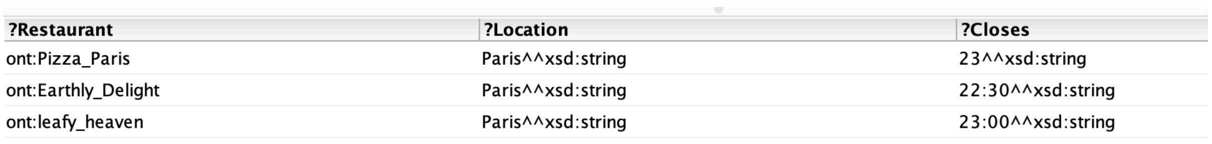


?Restaurant	?DeliveryService	?FoodProduct
ont:leafy_heaven	ont:Uber_It	
ont:Pizzeria_Roma	ont:Deliverou	ont:Pizza
ont:Sushi_Paris	ont:Deliverou	ont:Sushi

2. A query that contains at least 2 alternatives and conjunctions

Let's say we want to find all restaurants that are either Italian or Vegetarian, and for each one, we display their location and closing time if they are located in Paris and open on Tuesdays.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <htpp://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#>
SELECT ?Restaurant ?Location ?Closes
WHERE {
  {
    ?Restaurant rdf:type ont:Italian .
  } UNION {
    ?Restaurant rdf:type ont:Vegetarian .
  }
  ?Restaurant ont:Location ?Location .
  ?Restaurant ont:OpenDays ?OpenDays .
  ?Restaurant ont:Closes ?Closes .
  FILTER(?Location = "Paris" && CONTAINS(?OpenDays, "Tuesday"))
}
```

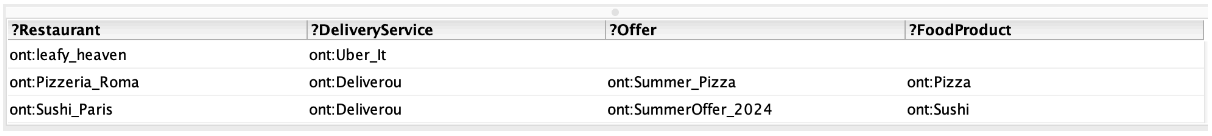


?Restaurant	?Location	?Closes
ont:Pizza_Paris	Paris^^xsd:string	23^^xsd:string
ont:Earthly_Delight	Paris^^xsd:string	22:30^^xsd:string
ont:leafy_heaven	Paris^^xsd:string	23:00^^xsd:string

3. A query that contains a CONSTRUCT query form

Let's construct a graph that includes restaurants along with their associated delivery services and food products.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <htpp://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#>
CONSTRUCT {
  ?Restaurant rdf:type ont:Restaurant .
  ?Restaurant ont:hasDeliveryService ?DeliveryService .
  ?Restaurant ont:hasOffer ?Offer .
  ?Offer ont:hasFoodProducts ?FoodProduct .
}
WHERE {
  ?Restaurant rdf:type ont:Restaurant .

  OPTIONAL {
    ?Restaurant ont:hasDeliveryService ?DeliveryService .
  }

  OPTIONAL {
    ?Restaurant ont:hasOffer ?Offer .
    ?Offer ont:hasFoodProducts ?FoodProduct .
  }
}
```



?Restaurant	?DeliveryService	?Offer	?FoodProduct
ont:leafy_heaven	ont:Uber_It		
ont:Pizzeria_Roma	ont:Deliverou	ont:Summer_Pizza	ont:Pizza
ont:Sushi_Paris	ont:Deliverou	ont:SummerOffer_2024	ont:Sushi

4. A query that contains an ASK query form

Let's write an ASK query to check if there are any Italian restaurants in Paris that are open on Tuesdays.

ASK isn’t recognized by Protege.

5. A query that contains a DESCRIBE query form

Let's write a DESCRIBE query to return information about Italian restaurants in Paris.

DESCRIBE isn’t recognized by Protege.

Part IV: Manipulating the ontology

1. Loads the ontology and displays all the Persons (without using queries, without inference).

```
# Look for all instances of Client in the graph
clients = g.subjects(RDF.type, mon_ns.Client)

# Display Clients
print("Clients found in the RDF graph:")

for client in clients:
    print(client)

# Look for all instances of Deliveryman in the graph
deliverymen = g.subjects(RDF.type, mon_ns.Deliveryman)

# Display Deliveryman
print("\nDeliverymen found in the RDF graph:")

for deliveryman in deliverymen:
    print(deliveryman)
```

```
Clients found in the RDF graph:
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Walid

Deliverymen found in the RDF graph:
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Jenna
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pierre
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Valentin
```

2. Loads the ontology and displays all the Persons (using a query, without inference). Create the used query in text file under the data folder.

```
# Path to txt file
query_file = 'data/query_persons.txt'

# Load SPARQL query from file
with open(query_file, 'r', encoding='utf-8') as f:
    query_str = f.read()

# Prepae SPARQL query
query = prepareQuery(query_str)

# Execute query and get result
results = g.query(query)

# Display result
```

```python
print("Clients and Deliverymen found in the RDF graph:")

for row in results:
    print(row.person)
```

```
Clients and Deliverymen found in the RDF graph:
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Walid
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Jenna
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pierre
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Valentin
```

3. Loads the ontology and displays all the Restaurants serving Italian food (without using queries, using inference).

```python
# Look for all instances of Italian Restaurants in the graph
Italian_Restaurants = g.subjects(RDF.type, mon_ns.Italian)

# Display Italian Restaurants
print("Italian Restaurant found in the RDF graph:")

for Italian in Italian_Restaurants:
    print(Italian)
```

```
Italian Restaurant found in the RDF graph:
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizza_Paris
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizzeria_Roma
```

4. Develops a program that :
    1. Reads a name of a food
    2. If it doesn't exist displays an error message
    3. Else, display its restaurants, delivery services, and offers
    4. Display their availability where and when

```python
# function to get food uri
def get_food_uri(food_name):
    for food in g.subjects (RDF.type, mon_ns.FoodProduct):
        if (food, None, Literal (food_name)) in g:
            return food
        else :
            return None

# Retrieve associated data for Food name
def get_food_details(food_uri, food_name):
    # Retrieve restaurants serving this food
    restaurants = g.objects(food_uri, mon_ns.hasRestaurant)
    print(f"\nRestaurants serving '{food_name}':")
    for restaurant in restaurants:
        print(restaurant)

    # Retrieve delivery services offering this food
    delivery_services = set(g.objects(food_uri,
mon_ns.hasDeliveryService))
    print(f"\nDelivery services offering '{food_name}':")
    for delivery_service in delivery_services:
        print(delivery_service)

    # Retrieve offers for this food
    offers = set(g.objects(food_uri, mon_ns.hasOffer))
    print(f"\nOffers for '{food_name}':")
    for offer in offers:
        print(offer)
        # Retrieve availability details for each offer
        availability_ends = list(g.objects(offer,
mon_ns.availabilityEnds))
        for availability_end in availability_ends:
            print(f" - Available until: {availability_end}")

    #Retrieve Location for this food
    locations = set(g.objects(food_uri, mon_ns.Location))
    print(f"\nLocation '{food_name}':")
    for location in locations:
        print(location)

# Use the function
food_name = input("Enter the name of a food: ")
food_uri = get_food_uri(food_name)
get_food_details(food_uri, food_name)
```

```
Enter the name of a food:  Pizza

Restaurants serving 'Pizza':
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizza_Paris
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizzeria_Roma

Delivery services offering 'Pizza':
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Uber_It
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Deliverou

Offers for 'Pizza':
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Summer_Pizza
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#SummerOffer_2024

Location 'Pizza':
Lyon
Rome
Paris
```

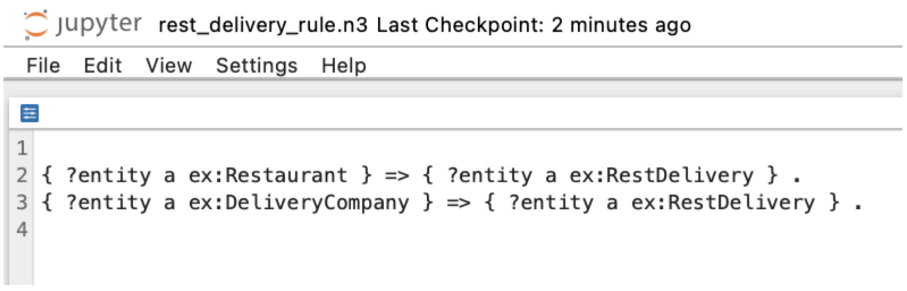
5. Displays all entities that are restaurants and delivery companies. Do this using a rule that defines a new class RestDelivery. The rule file must be saved in the data folder.

```
# function to get subclasses of a class
def get_all_subclasses(graph, class_uri):
    subclasses = set()
    for subclass in graph.transitive_subjects(RDFS.subClassOf,
class_uri):
        subclasses.add(subclass)
    return subclasses

# use function to get subclasses of Restaurant
restaurant_subclasses = get_all_subclasses(g, mon_ns.Restaurant)

# Store all entities of type RestDelivery
rest_deliveries = set()
for subclass in restaurant_subclasses:
    for entity in g.subjects(RDF.type, subclass):
        rest_deliveries.add(entity)
        g.add((entity, RDF.type, mon_ns.RestDelivery))

# Get all entities of type DeliveryCompany
for entity in g.subjects(RDF.type, mon_ns.DeliveryCompany):
    rest_deliveries.add(entity)
    g.add((entity, RDF.type, mon_ns.RestDelivery))

# Define rules of format N3 to mark entities as RestDelivery
rule_content = """
{ ?entity a ex:Restaurant } => { ?entity a ex:RestDelivery } .
{ ?entity a ex:DeliveryCompany } => { ?entity a ex:RestDelivery } .
"""
```

```
# Write rules in file rest_delivery_rule.n3
with open("data/rest_delivery_rule.n3", "w") as file:
    file.write(rule_content)

# Displays entities of type RestDelivery
print("RestDelivery entities:")
for rest_delivery in rest_deliveries:
    print(rest_delivery)
```

```
RestDelivery entities:
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Sushi_Paris
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Botanical_Bistro
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Uber_It
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Earthly_Delight
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizzeria_Roma
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizza_Paris
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#leafy_heaven
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Deliverou
```

As we can see the rules have been written in the file :


jupyter rest_delivery_rule.n3 Last Checkpoint: 2 minutes ago
File Edit View Settings Help

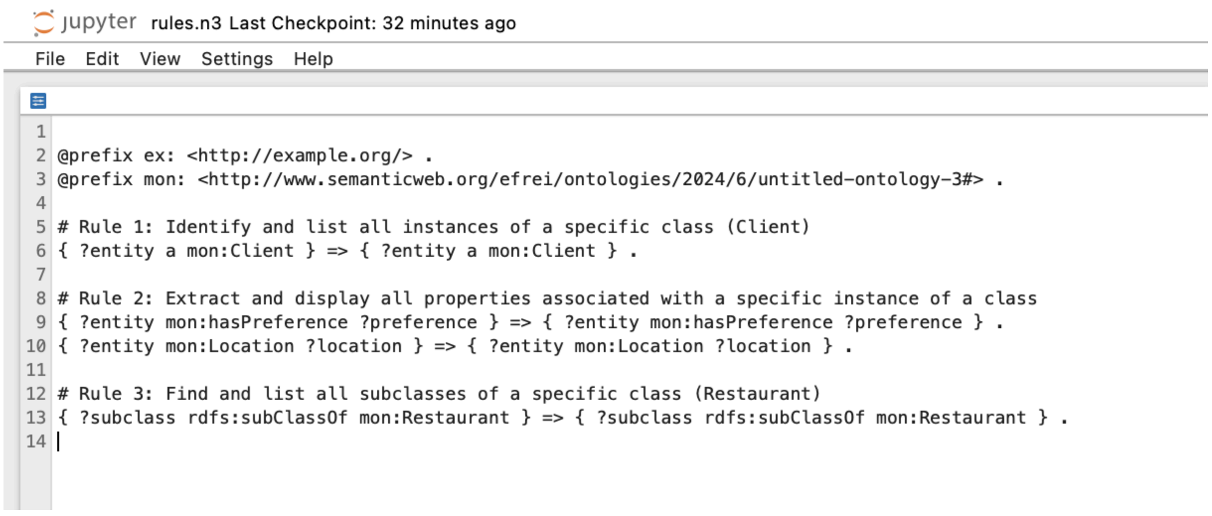
```

{ ?entity a ex:Restaurant } => { ?entity a ex:RestDelivery } .
{ ?entity a ex:DeliveryCompany } => { ?entity a ex:RestDelivery } .

```

6. Specifies 3 different rules and implement them in a java program

```
def get_all_subclasses(graph, class_uri):
    subclasses = set()
    for subclass in graph.transitive_subjects(RDFS.subClassOf,
class_uri):
        subclasses.add(subclass)
    return subclasses


# Rule 1: Identify and list all instances Client class
client_instances = set()
for entity in g.subjects(RDF.type, mon_ns.Client):
    client_instances.add(entity)
    g.add((entity, RDF.type, mon_ns.Client))

# Rule 2: Extract and display all properties of Client "Walid"
walid_properties = {}
walid_instance = URIRef(mon_ns + "Walid")
for p, o in g.predicate_objects(subject=walid_instance):
    walid_properties[p] = o

# Rule 3: Find and list all subclasses of class Restaurant
restaurant_subclasses = get_all_subclasses(g, mon_ns.Restaurant)

rule_content = """
@prefix ex: <http://example.org/> .
@prefix mon:
<http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#
> .

# Rule 1: Identify and list all instances of a specific class (Client)
{ ?entity a mon:Client } => { ?entity a mon:Client } .

# Rule 2: Extract and display all properties associated with a specific
instance of a class
{ ?entity mon:hasPreference ?preference } => { ?entity mon:hasPreference
?preference } .
{ ?entity mon:Location ?location } => { ?entity mon:Location ?location }
.

# Rule 3: Find and list all subclasses of a specific class (Restaurant)
{ ?subclass rdfs:subClassOf mon:Restaurant } => { ?subclass
rdfs:subClassOf mon:Restaurant } .
"""

with open("data/rules.n3", "w") as file:
    file.write(rule_content)


print("Client Instances:")
for client in client_instances:
    print(client)

print("\nProperties of 'Walid':")
for prop, value in walid_properties.items():
    print(f"{prop}: {value}")

print("\nSubclasses of 'Restaurant':")
for subclass in restaurant_subclasses:
    print(subclass)
```

```
Client Instances:
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Walid

Properties of 'Walid':
http://www.w3.org/1999/02/22-rdf-syntax-ns#type: http://www.semanticweb.org/efrei/ontologies/2024/6/un
titled-ontology-3#Client
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#hasPreference: http://www.seman
ticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Pizzeria_Roma
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Location: Lyon

Subclasses of 'Restaurant':
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Restaurant
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Italian
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Indian
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#FastFood
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Asian
http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#Vegetarian
```

As we can see the rules have been written in the file :


jupyter rules.n3 Last Checkpoint: 32 minutes ago
File Edit View Settings Help

```

@prefix ex: <http://example.org/> .
@prefix mon: <http://www.semanticweb.org/efrei/ontologies/2024/6/untitled-ontology-3#> .

# Rule 1: Identify and list all instances of a specific class (Client)
{ ?entity a mon:Client } => { ?entity a mon:Client } .

# Rule 2: Extract and display all properties associated with a specific instance of a class
{ ?entity mon:hasPreference ?preference } => { ?entity mon:hasPreference ?preference } .
{ ?entity mon:Location ?location } => { ?entity mon:Location ?location } .

# Rule 3: Find and list all subclasses of a specific class (Restaurant)
{ ?subclass rdfs:subClassOf mon:Restaurant } => { ?subclass rdfs:subClassOf mon:Restaurant } .

```