Project

Web datamining & semantics Project

The Semantic Web project is a large and long practical exercise that consists in integrating all the pieces that have been seen during the first sessions into a consolidated application. To make sure you can advance sufficiently fast to cover everything, you will work by pair.

Main objectives

- **Design a food delivery discovery service** : We will describe different kinds of entities that are useful for the food delivery discovery service: food products, offers for the food items (including a single item or combos), restaurants and businesses that offer the food items; delivery services (deliveryman, or company: Delivroo, Uber Eats, ...) that work with the restaurants and business to deliver their food items; customers with their customers's preferences.
- Develop an application that provide, given a customer's description, the place where to order food at the time of request.

Pedagogical objectives

- Do a little software development, using Semantic Web programming frameworks
- Setup and interact with an RDF database
- Exploit multiple sources of heterogeneous data
- Present information online with rich metadata

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.

Indications :

1. Define different class and propertie hiearchies (at leat three hiearchies and three restrictions)
2. Define different class restrictions (eg. disjoint, existential and universal restrictions and other conditions on defined classes, ...) (at leat three)
3. Define at leat three different type of the properties (transitive, symmetric, inverseOf, etc.) if necessary.
4. You application should follows the Linked Data principles:

1. As a starting point, individuals are recommeded to be described as instances of classes in Schema.org schemaorg.owl. For instance, food products, be instances of `schema:Product`;
2. User preferences with respect to location, time, price range and, optionally, type of food should be stored in the Linked Data Platform. An example of preferences is provided in the `pref-charpenay` resource.
3. The data of the service should be available online and represented in a standard vocabulary that any application can process.

5. Check the consistency of your ontology with PELLET

Part II: Populating the ontology

You will populate you knowledge phase by :

- ***Manually:*** RDF instances that you could be manually created in Protégé
- ***Other sources:*** From non-RDF formats, you convert online available data (from open data sources, see bellow) into RDF, and load the resulting data. You can simply generate an RDF file that you load it manually, or (better) add the RDF programmatically using SPARQL Update queries. You need for instance to add Json Context to the different available Json files from open data sources, such us :
    - CoopCycle
        - CoopCycle is a federation of bike delivery coops, all over the world. Bike delivery coops are associations of cyclist who propose their service to restaurants and other businesses to deliver products to customers at home. From this page, you will have to find a JSON document that describes all the coops of this federation. Hint: use advanced capabilities of your browser.
        - From the JSON file found from the web page above, it is possible to find the URIs of each coop, usually associated with a city or agglomeration. From there, CoopCycle exposes information in JSON-LD about local businesses whose products are delivered by the coop. To collect that information, you will have to write an HTML parser that is able to navigate and/or to extract JSON-LD.

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
2. List the name of all Paris restaurants.
3. List the name of all vegetarian restaurant, for each one, display their delivery services
4. List the name of deliverymen older than 51 years and thay can deliver in Lyon in less that 15 minutes

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

Propose 5 SPARQL queries:

1. A query that contains at least 2 Optional Graph Patterns
2. A query that contains at least 2 alternatives and conjunctions
3. A query that contains a CONSTRUCT query form
4. A query that contains an ASK query form
5. A query that contains a DESCRIBE query form

Part IV: Manipulating the ontology using Jena

Using Jena develop the following functionalities that:

1. Loads the ontology and displays all the Persons (**without** using queries, **without** inference).
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.
3. Loads the ontology and displays all the Restaurants serving Italian food (**without** using queries, **using** inference).
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
5. Displays all entities that are restaurant and delivery compagny. Do this using a rule that defines a new class RestDelivery. The rule file must be saved in the data folder.
6. Specifies 3 different rules and implement them in a java program

These instructions assume that you are programming in Java, preferably with Eclipse, using the Apache Jena libraries. You may also use RDF4J in Java, RDFlib in Python, or Redland RDF libary in C, or dotNetRDF in C♯, or EasyRDF for PHP, or N3.js for JavaScript, or Ruby RDF for Ruby, or SWI-Prolog Semantic Web Library, etc.

Work to send:

You will be working on your project full time during the remaining sessions.
On your last course session, you will deliver all of your working files, and give a

presentation and a demo. You must additionally provide a written report explaining your choices, the functionalities, etc. A deadline will be specified later by your professors. Everything that comes after this deadline will be rejected as if nothing was delivered. Create an archive name1-name2-name3.zip with:

1. The *.owl file generated by Protégé (part I and II)

1. A (*.txt or *.doc) file containing the SPARQL queries (part III)

1. The eclipse src and data folders of (part IV and V)

1. The presentation file *.ppt

Upload you archive in DVO deposit;

Part V: Application (optional)

Make an application (Ideally a website, or a GUI application, or a terminal application) that will allow one to select places such as a city (in a list or on a map) and get the associated data. Ideally, provide links to nearby movies and theaters, or associated POI (Point of interests), see bellow links

The resulting lists and entities are recommender to be available in HTML with RDFa or JSON-LD. While the real time data may be generated on the fly, static data should be extracted from the triplestore using a SPARQL query.

You can then try to extend your application by integrated other features. For example, you may:

1. extend the number of sources used;
2. extend the types of resources integrated in the application;
3. add more information, such as prices for train tickets, touristic data, street addresses, etc.;
4. store the history of dynamic data, and provide statistics;
5. use weather or air quality data as well;
6. define complex queries that bring interesting information not easily found manually;
7. define a complex ontology that expends beyond the core features of the application;
8.

Part VI: Triple store (optional)

Setup a triplestore. The simplest is to use files, and the best use stores such as: Apache Jena Fuseki, but you may also install a OpenLink's Virtuoso server (triplestore used by DBpedia in its backend) or Blazegraph (triplestore used by Wikidata) or Stardog (another commercial triplestore that has a free version) or GraphDB (yet another commercial triplestore with a free 60-day licence). A list of triplestores is available on Wikipedia.

There are many triplestores. The simplest to set up is probably Fuseki.

- Download the archive for Apache Jena Fuseki from Jena download page.
- In the archive, there is an executable: `fuseki-server.bat` for Windows systems, `fuseki-server` for Unix-based systems. Execute it. The server will be running in the background.
- With your Web browser, go to `http://localhost:3030`. This interface allows you to manage your data.
- Go to "manage datasets". Create a new dataset and make it persistent.
- Upload an RDF file of your choice.

In the exercise of the first part, you can generate all the data at once in a large Jena Model and serialise it as RDF, or you can fill in a triplestore little by little. If you want to add data to a triplestore such as Jena Fuseki, you can send update queries like this:

```
Model model = ModelFactory.createDefaultModel();
// ... build the model
String datasetURL = "http://localhost:3030/dataset";
String sparqlEndpoint = datasetURL + "/sparql";
String sparqlUpdate = datasetURL + "/update";
String graphStore = datasetURL + "/data";
RDFConnection conneg =
RDFConnectionFactory.connect(sparqlEndpoint,sparqlUpdate,graphStore);
conneg.load(model); // add the content of model to the triplestore
conneg.update("INSERT DATA { <test> a <TestClass> }"); // add the triple to the
triplestore
```

If you finish fast, you can then try to define a vocabulary for GTFS and transform all the SNCF data to RDF. Your vocabulary can also distinguish between train stations and coach stations, relate stations (StopArea) to more specific locations (StopPoint), etc.

Open Data for POI resources

Open data about train stations and train lines can serve as your first data source.

Public transport networks (trains, buses, tramways)

France stations and train lines:

SNCF (GTFS data)

High-speed train stations in France:

Gare TGV en France

Real time data for trains in France:

API SNCF

Additional static data about train stations

SNCF Open data about stations

Saint-Étienne buses and tramways:

Données ouvertes de la STAS (GTFS data)

Bicycle-sharing systems

Saint-Étienne:

Données ouvertes de Saint-Étienne Métropole

Lyon:

Real-time available. See also Grand Lyon data portal.

Rennes:

Real time data. See also Rennes Métropoloe data portal.

Montpellier:

Real time data. See also OpenData Montpellier Métopole.

Strasbourg:

Real time data. See also Strasbourg open data.

Paris:

Real time data. See also Paris open data portal.

You can find other data sets for bicycle-sharing systems all over the world by consulting the list of bicycle-sharing systems from Wikipedia. For France specifically, there is a list that contains more details. The open data portal of the French government has data

about many bicycle-sharing systems that you can get by going to https://transport.data.gouv.fr/gbfs/**city_name_lowercase**/station_status.json. For instance https://transport.data.gouv.fr/gbfs/toulouse/station_status.json for Toulouse. Bicycle-sharing systems that are operated by company JC Decaux have data available using a single API with URLs of the form https://api.jcdecaux.com/vls/v1/stations?contract=**Cityname**&apiKey=**your_API_key**. You need to register an API key to use it.

Parking places and parking lots

Argenteuil:

Static description of parking lots

Caen la Mer:

Buildings: Hospitals, universities, schools, etc.

Other geospatial data

Weather data API:

OpenWeather API (requires an API key)

Air quality API:

Air Quality Programmatic APIs

Electric vehicle charger in Saint-Étienne:

Infrastructure de Recharge Véhicules électriques- Saint Etienne Métropole

RDF for geospatial data:

Linked Geo Data (with a SPARQL endpoint)

API for getting geocoordinates from street addresses:

API Adresse from French government Web site.

RDF data from INSEE

Publication de données géographiques au format RDF

Inspired by Antoine Zimmermann course.