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 are allowed to work by pair.

Main objectives

- **Design a movie application** that follows the Linked Data principles: the application should be represented in a standard vocabulary that any application can process. As a starting point, movie instances should be described as instances of schema:Movie. Schema.org for Developers
- Schema.org en schemaorg.owl
- Develop an application to create, query and validate calendar events.

Resources

You will make use of data sets, in RDF and non-RDF formats, such us :

- the Open Movie Database and try the examples
- https://developer.imdb.com/
- https://data.culture.gouv.fr/explore/dataset/etablissements-cinematographiques/api/

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 movies. A movie has one or several directors, writers and actors. It also has a title, one or several genres, a year, a country and a language. To define the Genre of a movie, possible choices are: Thriller, Crime, Action, Drama or Comedy. Actors, directors and writers are persons. Persons have a gender (male or female), a name, an age and a nationality. Movies are scheduled in theaters that have locations.

Indications :

1. Define classes that are disjoint, restrictions and conditions on classes if necessary. Example of a restriction: an actor has the restriction: *isActor* of a Movie.
2. Define the types of the properties (transitive, symmetric, inverseOf, etc.) if necessary.
3. While defining a property, define its inverse (*hasActor* and isActorOf) if necessary.
4. While defining a property, define its domain and range.
5. Define any other concepts or properties if it is needed

6. Define or reuse a vocabulary for describing the resources (Schema.org, FoaF, etc.). You can make it more specific, but it should have generic concepts that can be reused for many things.
7. Check the consistency of your ontology with PELLET

Part II: Populating the ontology

Create some individuals to the Movie class such as:

- Pulp Fiction, Genre: Crime Thriller, 1994, USA, English.
- Kill Bill (volume 1), Genre: Action Crime Thriller, 2003, USA, English.

Create some individuals to the Person class such as:

- Quentin Tarantino, American, 53 years old, writer and director of Pulp Fiction and Kill Bill (volume1). He also played a role in that movie.
- John Travolta, American, 59 years old, actor in Pulp Fiction.
- Uma Thurman, 43 years old, actress in Pulp Fiction. She also participated as a writer in Kill Bill (volume1).

Create some individuals to the different classes: actors, theater locations, etc.

Convert static data (from open data sources, see above) 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.

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.

Setting up a triplestore

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.

Part III: Querying the ontology

Write SPARQL queries to response to the following:

1. List the instances of the class Actor
2. List the name of all Thriller movies. For each one, display its director.
3. List the name of all Crime Thriller movies.
4. List the name of Actors older than 51 years.
5. List of movies that are played in theater 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 Actors (**without** using queries, **using** inference).

4. Develops a program that :
    a. Reads a name of a movie
    b. If it doesn't exist displays an error message
    c. Else, display its year, country, genres and actors
    d. Display their program where and when
5. Displays all persons that are actors and directors. Do this using a rule that defines a new class ActorDirector. 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.

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

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

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)
2. A (***.txt or *.doc)** file containing the SPARQL queries (part III)
3. The eclipse **src** and **data** folders of (part IV and V)
4. The presentation file ***.ppt**

Upload you archive in DVO deposit, the title of the zip file **must** be: **SW-Project-name1_name2**