Table of Contents

Introduction

The goal of this coursework was to put into practice the ontology modelling and semantic data development skills, and to accomplish this I have defined an ontology, populated it from external source (basic task), added extra information into it from the second external source (bonus task) and provided a couple of queries for the ontology populated from the two different sources to demonstrate that it is working correctly and that it can provide the combined information.

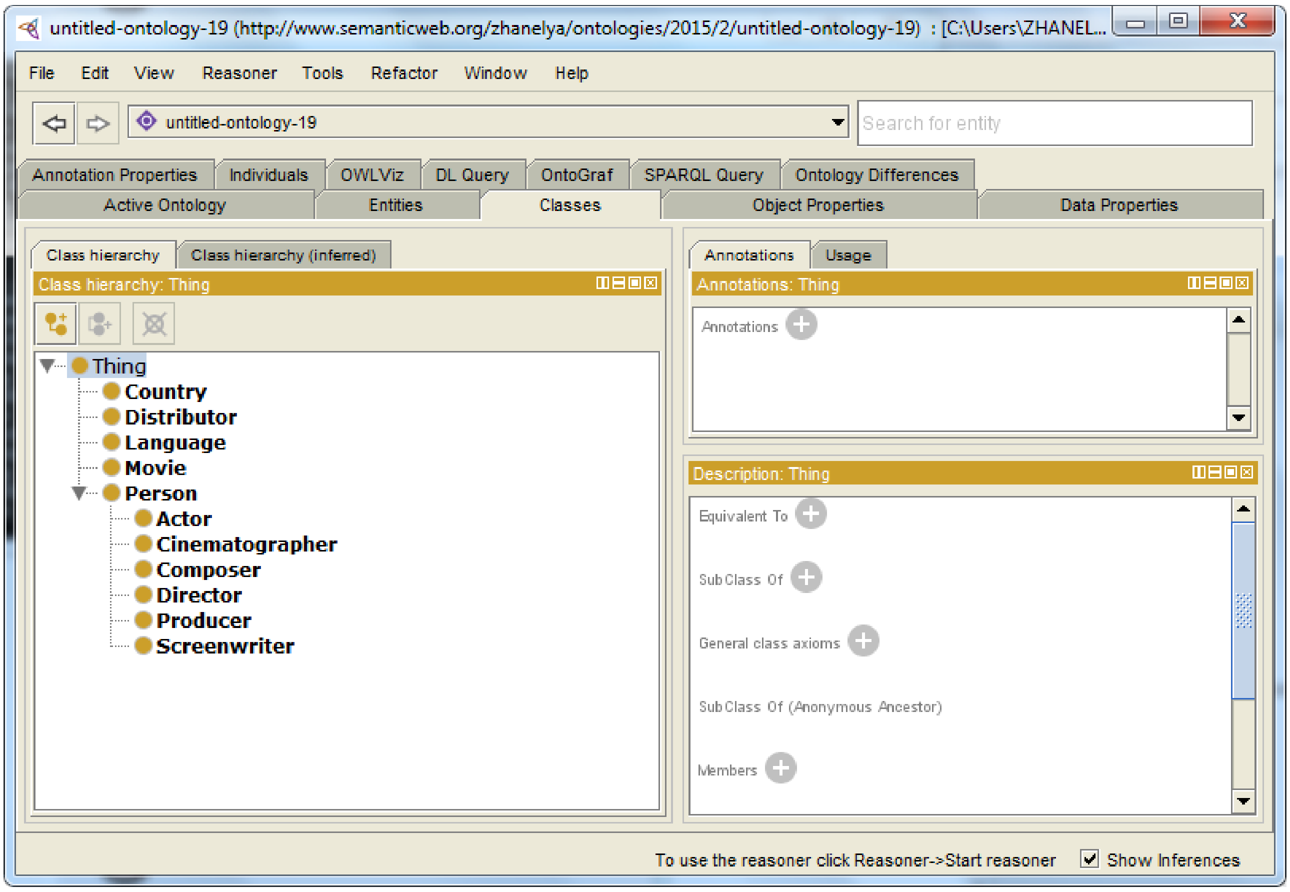
Constructing the Ontology

First, the purpose of the ontology needed to be established. The ontology that was created and populated from two external sources is covering the domain of movies and countries and allows such queries as "Find a movie filmed in countries with population more than 100M". Second step was to consider the ways of re-using the existing ontologies. In order to build and populate an ontology, I was using 2 endpoints, namely – DBpedia and Factbook (the corresponding links for these knowledge bases can be found under *Bibliography* section). Third, a list of terms and their internal organisation needed to be identified. In order to define the structure of the ontology I was using Protégé-OWL2. The resulting ontology can be found in *ontology.owl*. The process of defining the ontology is an iterative process, which is why I had to review my ontology multiple times (both when constructing the ontology structure and when attempting to populate it) in order to estimate if it can store sufficient information from the two endpoints, before I finally came up with the resulting model. In order to accomplish this I was studying several different individuals from the two knowledge bases, for instance I explored such movie entries from the DBpedia knowledge base as "Hysterical blindness" and "The Great Beauty", and such country entries from the Factbook ontology as "Afghanistan" and "Uzbekistan" (the corresponding links are provided under *Bibliography*).

- ***Class hierarchy***

Next, the schema and skeleton of the ontology had to be defined. Based on the structure of the information contained in the two endpoints, I created 11 classes for my ontology, which are shown in *Image 1*. *Country*, *Distributor*, *Language*, *Movie* and *Person* are the top level classes, whereas *Actor*, *Cinematographer*, *Composer*, *Director*, *Producer* and *Screenwriter* are the subclasses of the

Person class. One of the challenges while creating the ontology was to decide which information should be stored and how it should be contained in the ontology. For this purpose I used the movie (DBpedia) and country (Factbook) entries mentioned above in order to choose which properties of the movies and countries I want to represent as a class and in which way it would be more appropriate to store them.

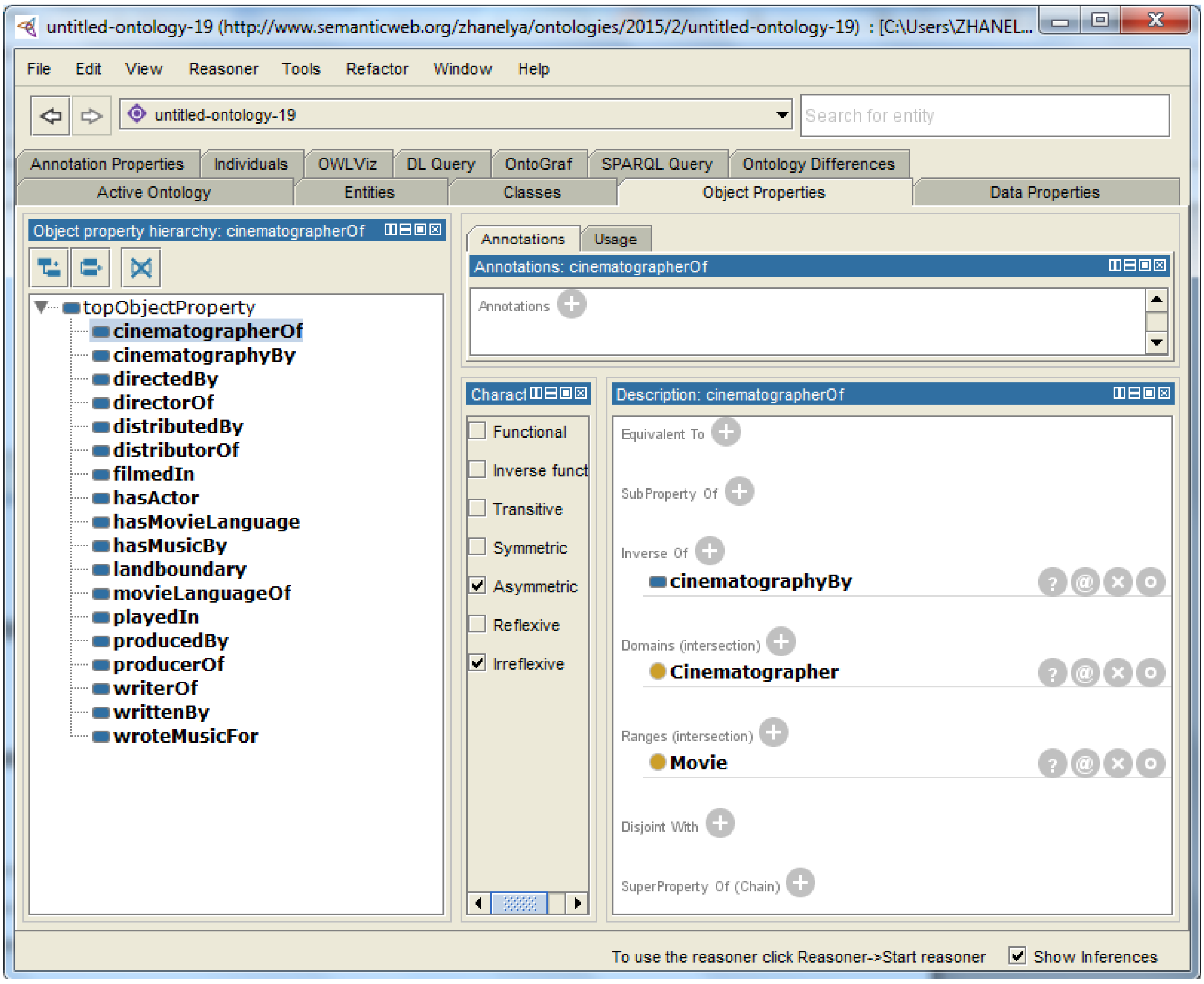

untitled-ontology-19 (http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19) : [C:\Users\ZHANEL...
File Edit View Reasoner Tools Refactor Window Help
untitled-ontology-19
Search for entity
Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences
Active Ontology Entities Classes Object Properties Data Properties
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Thing
Annotations Usage
Annotations: Thing
Annotations
Description: Thing
Equivalent To
SubClass Of
General class axioms
SubClass Of (Anonymous Ancestor)
Members
To use the reasoner click Reasoner->Start reasoner
Show Inferences

- Thing
  - Country
  - Distributor
  - Language
  - Movie
  - Person
    - Actor
    - Cinematographer
    - Composer
    - Director
    - Producer
    - Screenwriter

Image 1. Class hierarchy

- ***Object properties***

In order to enhance the ontology implementation, object properties had to be created to link different concepts together by forming reasonable relations among them. A full list of the object properties defined can be seen in *Image 2* below. For each object property, I defined whether it is functional, inverse functional, transitive, symmetric/asymmetric or reflexive/irreflexive. For instance, *cinematographerOf*, *cinematographyBy*, *directedBy*, *directorOf*, *distributedBy*, *distributorOf*, *filmedIn*, *hasActor*, *playedIn*, *hasMovieLanguage*, *movieLanguageOf*, *producedBy*, *producerOf*, *writtenBy*, *writerOf*, *hasMusicBy* and *wroteMusicFor* are asymmetric because they are not bi-directional (for instance, if a person is a cinematographer of a movie, the movie itself cannot be a cinematographer of the cinematographer), and irreflexive (for instance, director of a movie cannot be a director of him/herself). On the other hand, *landboundary* is symmetric and irreflexive property, as if a country has a land boundary with another country, then this relation holds in the opposite direction too, and because a country cannot have a land boundary with itself. Additionally, in order to produce a richer ontology, most of the asymmetric properties were defined in pairs, one being an inverse property of another – for example, *cinematographerOf* and *cinematographyBy*. Finally, every object property was given a domain and range restrictions to limit, correspondingly, the values that its subjects and objects can take. For example, the domain of the object property *directorOf* is the *Director* class and its range is the *Movie* class, whereas the *lanboundary* object property has *Country* as both its domain and range restrictions. However, it should be relatively intuitive from the object properties names which object property relates to which classes.



Object property hierarchy: cinematographerOf

- topObjectProperty
  - cinematographerOf
  - cinematographyBy
  - directedBy
  - directorOf
  - distributedBy
  - distributorOf
  - filmedIn
  - hasActor
  - hasMovieLanguage
  - hasMusicBy
  - landboundary
  - movieLanguageOf
  - playedIn
  - producedBy
  - producerOf
  - writerOf
  - writtenBy
  - wroteMusicFor

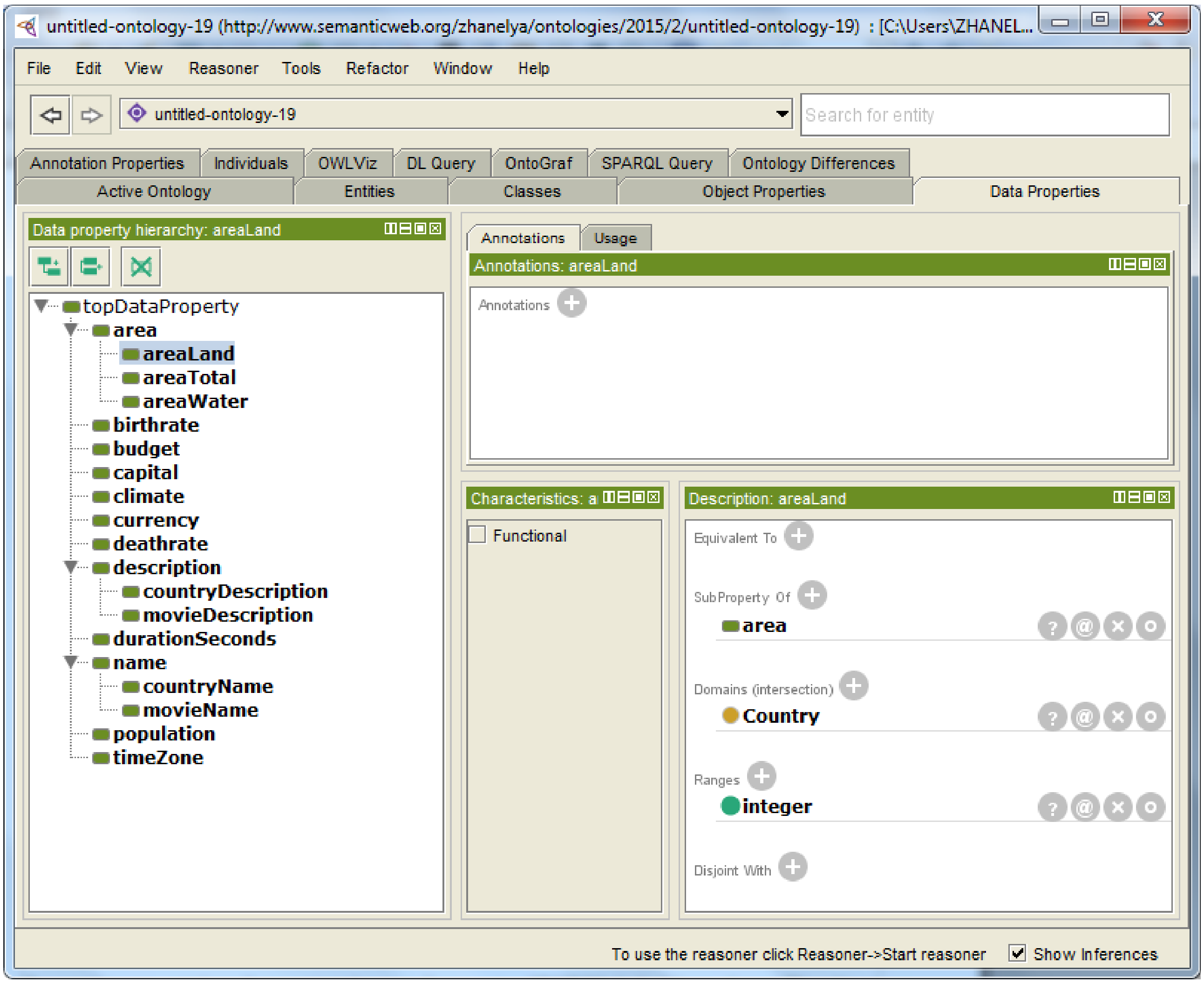
Functional [ ]
Inverse funct [ ]
Transitive [ ]
Symmetric [ ]
Asymmetric [✓]
Reflexive [ ]
Irreflexive [✓]

Equivalent To
SubProperty Of
Inverse Of
cinematographyBy
Domains (intersection)
Cinematographer
Ranges (intersection)
Movie
Disjoint With
SuperProperty Of (Chain)

Image 2. Object property hierarchy

- ***Data properties***

In order to cover atomic properties of movies and countries, I needed to add data properties to my ontology. A full hierarchy of the data properties can be found in *Image 3* below.



Data property hierarchy: areaLand

```
topDataProperty
    area
        areaLand
        areaTotal
        areaWater
    birthrate
    budget
    capital
    climate
    currency
    deathrate
    description
        countryDescription
        movieDescription
    durationSeconds
    name
        countryName
        movieName
    population
    timeZone
```

Functional: ☐

SubProperty Of: area
Domains (intersection): Country
Ranges: integer

Image 3. Data property hierarchy

There are 19 data properties in total defined in the ontology. 12 of them are top level data properties. *Area, birthrate, deathrate, capital, climate, currency, population* and *timeZone* were created to describe countries, whereas *budget* and *durationSeconds* characterise movies. There are also 2 general data properties – *description* and *name*, each having the corresponding sub-properties for movies and countries, namely, *movieDescription* and *countryDescription*, *movieName* and *countryName*. There is also a list of area sub-properties, namely *areaLand*, *areaTotal* and *areaWater.* There are several data properties that are functional, for instance, *capital*, *countryName* and *movieName*, because they can uniquely identify a certain country/movie. Apart from that, every data property except such general properties as *name* and *description*, has domain and range restrictions, where domain specifies which class of individuals it can refer to, and range specifies the data type of the property. For instance, *areaLand* has a domain restriction of *Country* and a range restriction of *integer* as it is representing the land area of a country in numerical format, whereas *movieName* has domain restriction of *Movie* and range restriction of *string*.

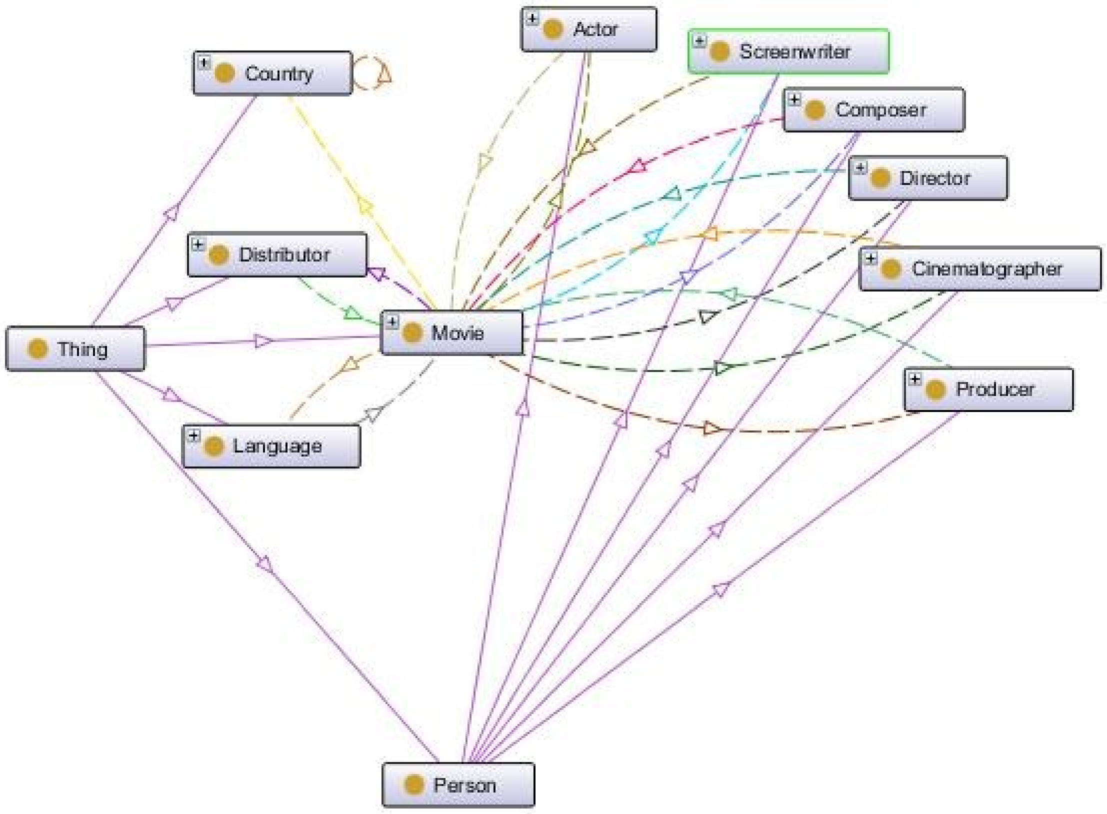


Image 4. OntoGraph ontology representation

A complete ontology represented in a form of an OntoGraph can be seen in *Image 4* above. The links connecting the classes together illustrate the subclass relationship (plain lines) and relations between different concepts (presented as dotted lines on the graph, defined as object properties in the ontology). The main challenge while creating an ontology was to decide which information can be represented as a class and its appropriate object properties and which information should rather be presented in a form of data properties of the existing classes. The assumption that I used at this stage was that basic information that can be stored in a form of literals or numbers (such as country surface area, or movie name) should be represented by data properties, whereas more complex entities such as *Actor* or *Composer*, which would potentially have their own properties, should be presented as a separate class with the appropriate object properties linking its individuals to some other classes, for example *Movie*. Apart from that, I needed to choose reasonable names for different classes and properties so that they would make sense when the ontology is constructed. For instance, *abstract* property of the movies from DBpedia was renamed into *movieDescription* property within my ontology to make it clearer for the user what kind of information it contains.

Populating the Ontology from DBpedia

Now, when the ontology skeleton was created, it needed to be populated with external information, as an ontology must be able to integrate and re-use available semantic data. All the concepts of my ontology T-Box were based on the information from DBpedia and Factbook, making it possible to populate its A-Box with already existing data. To populate the ontology from external source I was using SPARQL(1.1). I have chosen DBpedia as my first external ontology, as it contains various information about films. In order to populate the ontology, I produced a python script which can be found in *basic.py*. An outline of the created method will be described below.

First, the appropriate python libraries needed to be imported, such as rdflib, graph and SPARQL to enable rdf parsing, graph processing, and SPARQL queries. Logging module was also imported and configured to ease debugging process of the code.

```
import logging
import rdflib
from rdflib.graph import Graph, URIRef
from SPARQLWrapper import SPARQLWrapper, RDF
from rdflib.plugins.memory import IOMemory

logging.basicConfig()
…
```

Then, the SPARQL endpoint needed to be configured. For this purpose I was using DBpedia endpoint, which can be accessed through the http://dbpedia.org/sparql link.

```
...
sparql = SPARQLWrapper("http://dbpedia.org/sparql")
...
```

Query construction is one of the main parts of the ontology population. First, the prefixes were defined, which serve as short-cuts for complete ontology URLs, both from local and external knowledge bases. Then, within the CONSTRUCT statement, a list of movie properties that needed to be populated was provided together with the location where they should be stored in my local ontology, for instance, the name of the movie was stored under the *movieName* property. Movie attributes that have to be stored as separate class individuals, additionally needed a reference to the name of the class from the local ontology, for instance *filmedIn* has a reference to the *Country* class, as *filmedIn* represents a relation between *Movie* and *Country* classes.

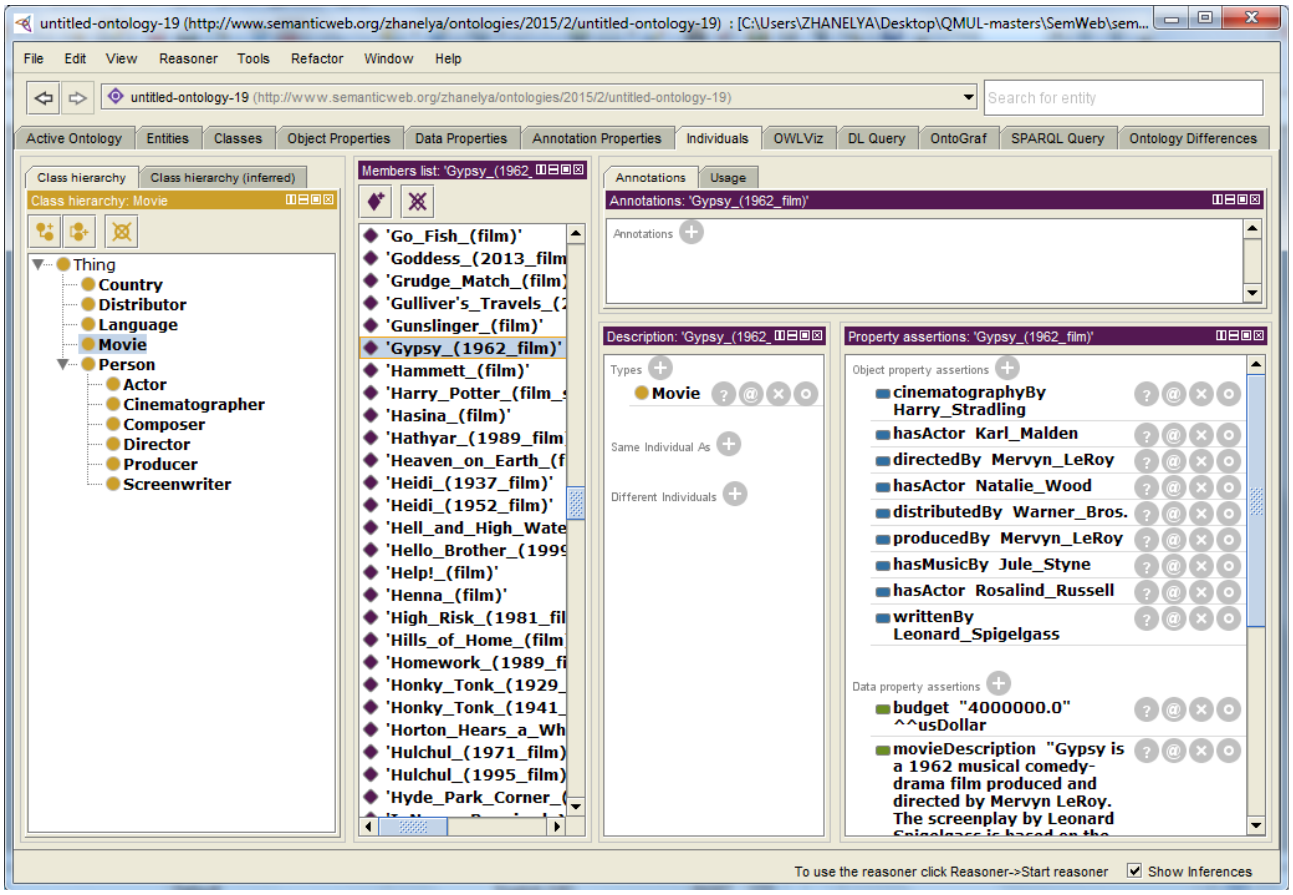
After that, within the WHERE statement, the corresponding properties from the DBpedia ontology were to be specified together with how they could be accessed. Some of the properties are optional, as quite often some non-key properties are missing from the external ontology. Finally, the result of the query was filtered to include only movies having their name and description written in English, and the number of results to be returned was limited to address memory usage/performance issues. The constructed query was then assigned to SPARQL endpoint, whose return format was set to RDF.

```
...
construct_query="""
    PREFIX movie: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
    ...
    PREFIX dbpprop: <http://dbpedia.org/property/>


    CONSTRUCT {
    ?film rdf:type movie:Movie .
    ?film movie:movieName ?name .
    ...
    ?film movie:filmedIn ?country .
    ?country rdf:type movie:Country .
    ...
    }
     WHERE{
     ?film rdf:type dbpedia-owl:Film .
     ?film foaf:name ?name .
     ...
     OPTIONAL {?film dbpedia-owl:country ?country}
     FILTER (LANG(?name)="en")
     FILTER (LANG(?abstract)="en")
    }
    LIMIT 100000 """
sparql.setQuery(construct_query)
sparql.setReturnFormat(RDF)
...
```

Finally, a graph was created and configured. The results retrieved from the endpoint were passed to this graph, and then merged with the local ontology, producing a new local ontology – *result_basic.owl*, serialised in XML format.

```
...
memory_store=IOMemory()
graph_id=URIRef("http://www.semanticweb.org/store/movie")
g = Graph(store=memory_store, identifier=graph_id)
rdflib.plugin.register('sparql', rdflib.query.Processor, 'rdfextras.sparql.processor', 'Processor')
rdflib.plugin.register('sparql', rdflib.query.Result, 'rdfextras.sparql.query', 'SPARQLQueryResult')


g = sparql.query().convert()
g.parse("ontology.owl")
g.serialize("result_basic.owl", "xml")
...
```

The complete ontology populated with movies information can be seen in *Image 5*. Each movie now has the corresponding information retrieved from external ontology, such as movie name, cinematographer, country it was filmed in, etc.


untitled-ontology-19 (http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19) : [C:\Users\ZHANELYA\Desktop\QMUL-masters\SemWeb\sem...
File Edit View Reasoner Tools Refactor Window Help
untitled-ontology-19 (http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19)
Search for entity
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Movie
Members list: 'Gypsy_(1962
Annotations Usage
Annotations: 'Gypsy_(1962_film)'
Description: 'Gypsy_(1962
Property assertions: 'Gypsy_(1962_film)'
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Thing
- Country
- Distributor
- Language
- Movie
- Person
  - Actor
  - Cinematographer
  - Composer
  - Director
  - Producer
  - Screenwriter

'Go_Fish_(film)'
'Goddess_(2013_film
'Grudge_Match_(film)
'Gulliver's_Travels_(
'Gunslinger_(film)'
'Gypsy_(1962_film)'
'Hammett_(film)'
'Harry_Potter_(film_
'Hasina_(film)'
'Hathyar_(1989_film
'Heaven_on_Earth_(f
'Heidi_(1937_film)'
'Heidi_(1952_film)'
'Hell_and_High_Wate
'Hello_Brother_(1999
'Help!_(film)'
'Henna_(film)'
'High_Risk_(1981_fil
'Hills_of_Home_(film
'Homework_(1989_fi
'Honky_Tonk_(1929_
'Honky_Tonk_(1941_
'Horton_Hears_a_Wh
'Hulchul_(1971_film)
'Hulchul_(1995_film)
'Hyde_Park_Corner_(

Annotations

Types
Movie

Same Individual As

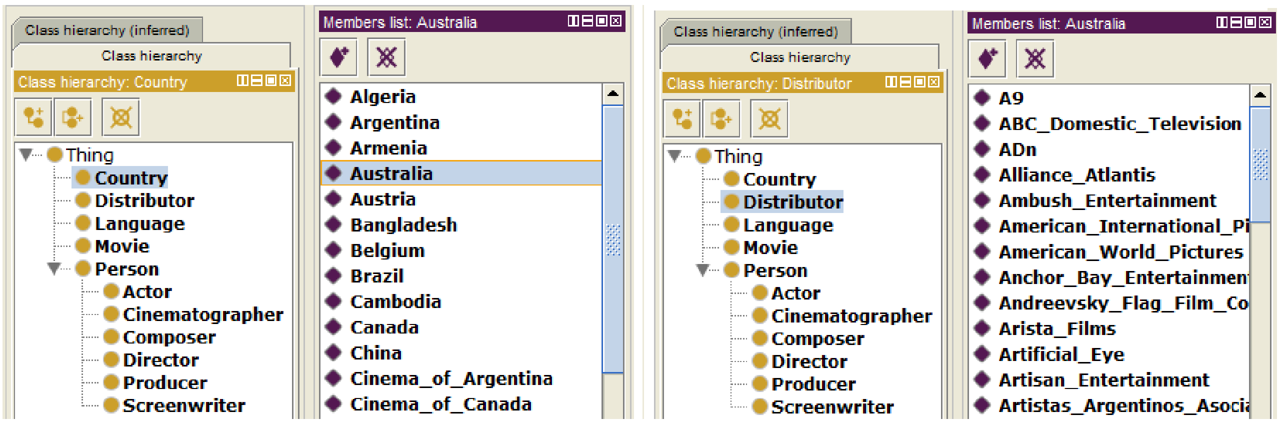
Different Individuals

Object property assertions
cinematographyBy Harry_Stradling
hasActor Karl_Malden
directedBy Mervyn_LeRoy
hasActor Natalie_Wood
distributedBy Warner_Bros.
producedBy Mervyn_LeRoy
hasMusicBy Jule_Styne
hasActor Rosalind_Russell
writtenBy Leonard_Spigelgass

Data property assertions
budget "4000000.0"^^usDollar
movieDescription "Gypsy is a 1962 musical comedy-drama film produced and directed by Mervyn LeRoy. The screenplay by Leonard

Image5. Populated movies ontology

However, *Movie* class is not the only class that had new individuals added from DBpedia. Other classes have also obtained new instances during the ontology population. However, each such individual does not have its own properties, as the information contained in these classes is not the main purpose of this ontology, and it was added only as these individuals are connected with some Movie instances. Examples of such complementary individuals can be seen in *Image 6* which illustrates part of the instances list for *Country* and *Distributor* classes.



Thing
Country
Distributor
Language
Movie
Person
Actor
Cinematographer
Composer
Director
Producer
Screenwriter

Algeria
Argentina
Armenia
Australia
Austria
Bangladesh
Belgium
Brazil
Cambodia
Canada
China
Cinema_of_Argentina
Cinema_of_Canada

Thing
Country
Distributor
Language
Movie
Person
Actor
Cinematographer
Composer
Director
Producer
Screenwriter

A9
ABC_Domestic_Television
ADn
Alliance_Atlantis
Ambush_Entertainment
American_International_Pi
American_World_Pictures
Anchor_Bay_Entertainmen
Andreevsky_Flag_Film_Co
Arista_Films
Artificial_Eye
Artisan_Entertainment
Artistas_Argentinos_Asoci

Image 6. Complementary individuals populated from DBpedia

One of the principal challenges when populating the ontology was matching the concepts and properties from the local store with the DBpedia concepts and properties in order to store information appropriately. In order to accomplish that, it was important to understand clearly what information is represented by which names within the external ontology and at which address/location it can be accessed. Apart from that, the results were limited just to those represented in English, as the ontology end user in my case will most likely prefer the information to be presented in English.

Querying the Ontology populated from DBpedia

In order to test the ontology implementation, I produced a python script containing two queries for my local ontology – *query_basic.py*. After the appropriate libraries were imported and configured (logging and rdflib), and after the graph was created by parsing the local ontology resulting from populating my ontology with DBpedia (described above), I defined two queries for my ontology. Query1's main purpose is to return movies filmed in United Kingdom. To achieve that, after defining the prefixes for querying my local store, I selected distinct movie names and the corresponding countries that they were filmed in, limiting the results only to those, where the country name would contain the string "United Kingdom".

```
...
query1 = """
PREFIX movie: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?name ?country
WHERE { ?film rdf:type movie:Movie .
    ?film movie:movieName ?name .
    ?film movie:filmedIn ?country .
    FILTER regex(str(?country), "United_Kingdom")
  }
"""
results = g.query(query1)
...
```

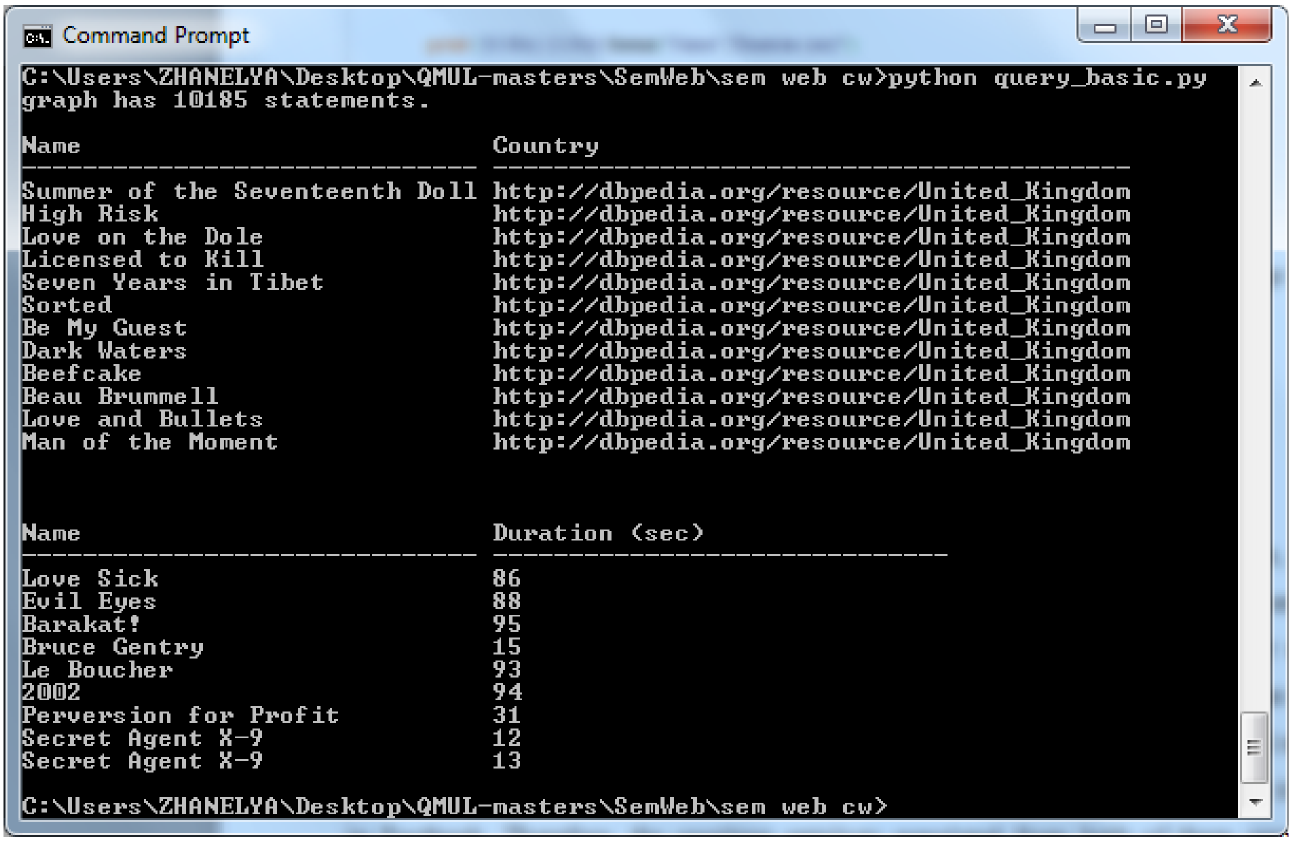
Query2 is aimed to retrieve movies that last less than 100 seconds. First, the appropriate prefixes were defined. Then, distinct names and their corresponding duration in seconds were selected. Finally, the results were limited to only those movies that have duration shorter than 100 seconds.

```
…
query2 = """
PREFIX movie: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?name ?durationSeconds
WHERE { ?film rdf:type movie:Movie .
    ?film movie:movieName ?name .
    ?film movie:durationSeconds ?durationSeconds .
    FILTER (?durationSeconds < 100.0)
  }
"""
results = g.query(query2)
…
```

After the queries were constructed, I needed to display the results back to the user in clear formatting. So, for the first query, I displayed the 2 resulting columns – Name and Country, using an I/O formatting in order to set the column width to 30 and 40 accordingly. The same way were formatted the results returned from the second query, but the column names and width were set differently.

```
...
print ('{0:30s} {1:40s}'.format("Name","Country"))
print ('{0:30s} {1:40s}'.format("------------------------------","----------------------------------------"))
for name,country in results:
        print ('{0:30s} {1:40s}'.format(name,country))
...
print ('{0:30s} {1:30s}'.format("Name","Duration (sec)"))
print ('{0:30s} {1:30s}'.format("------------------------------","------------------------------"))
for name,duration in results:
        print ('{0:30s} {1:30s}'.format(name,duration))
...
```

The output returned from this script querying the local ontology can be seen in *Image 7* below.



```
C:\Users\ZHANELYA\Desktop\QMUL-masters\SemWeb\sem web cw>python query_basic.py
graph has 10185 statements.

Name                           Country
------------------------------ ----------------------------------------
Summer of the Seventeenth Doll http://dbpedia.org/resource/United_Kingdom
High Risk                      http://dbpedia.org/resource/United_Kingdom
Love on the Dole               http://dbpedia.org/resource/United_Kingdom
Licensed to Kill               http://dbpedia.org/resource/United_Kingdom
Seven Years in Tibet           http://dbpedia.org/resource/United_Kingdom
Sorted                         http://dbpedia.org/resource/United_Kingdom
Be My Guest                    http://dbpedia.org/resource/United_Kingdom
Dark Waters                    http://dbpedia.org/resource/United_Kingdom
Beefcake                       http://dbpedia.org/resource/United_Kingdom
Beau Brummell                  http://dbpedia.org/resource/United_Kingdom
Love and Bullets               http://dbpedia.org/resource/United_Kingdom
Man of the Moment              http://dbpedia.org/resource/United_Kingdom


Name                           Duration (sec)
------------------------------ ------------------------------
Love Sick                      86
Evil Eyes                      88
Barakat!                       95
Bruce Gentry                   15
Le Boucher                     93
2002                           94
Perversion for Profit          31
Secret Agent X-9               12
Secret Agent X-9               13

C:\Users\ZHANELYA\Desktop\QMUL-masters\SemWeb\sem web cw>
```

Image 7. Querying movies ontology

Populating the Ontology from Factbook

In order to enrich the ontology already populated with information about movies, and fulfil the requirements set by the bonus task, I had to populate the ontology from the second external semantic data repository. In order to achieve this I chose the Factbook ontology as my second source, because it is fully concentrated on information about countries and thus has more detailed data about each country instance. For example, such country information as birth rate (total number of live births per 1000 citizens) and land boundary cannot be retrieved from DBpedia, but it can be found in Factbook. Therefore, the resulting ontology populated from both of them would be able to answer such questions as "Which movies were filmed in the countries with birthrate between 15 and 20?", and "Which movies were filmed in countries bordering with Kazakhstan?", which would not be possible with either remote knowledge base alone. The python script used for this task – *bonus.py* – is briefly described below.

After the appropriate python libraries, such as rdflib, graph, SPARQL and logging, were imported and configured, the Factbook SPARQL endpoint was defined:

```
…
sparql = SPARQLWrapper("http://wifo5-04.informatik.uni-mannheim.de/factbook/sparql")

...
```

Then, I constructed the query to populate the local ontology from the external Factbook source. First, the prefixes were identified, both for local and remote ontologies. Next, inside the CONSTRUCT statement, a set of country properties to be populated, such as *countryName* and *capital*, were listed and their location within the local storage was identified. One of the most interesting attributes of the countries data set is the land boundary property, as it needs to have a reference to the Country class itself. Within the WHERE clause, the name and location of each country property to be populated was identified within the remote Factbook knowledge base in order to make sure that the data is stored in accordance with its remote structure. As this knowledge base seems to be reasonably stable, none of the properties were set to be optional. Finally, the query was assigned to SPARQL endpoint, and the return type was set to RDF.

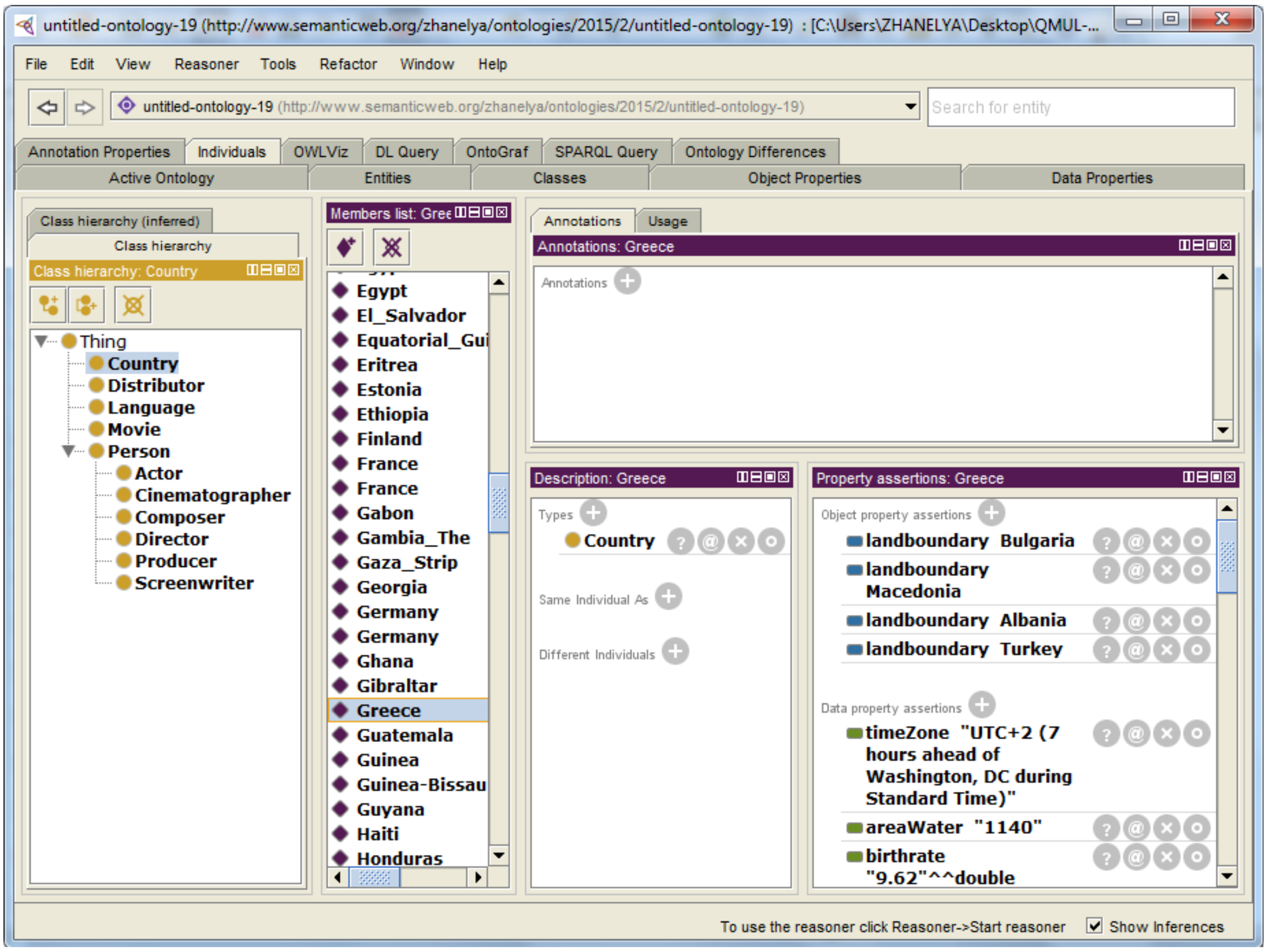
```
...
construct_query="""
    PREFIX mc: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
    ...
```

```
        PREFIX factbook: <http://wifo5-04.informatik.uni-mannheim.de/factbook/ns#>
        CONSTRUCT {
        ?country rdf:type mc:Country .
        ?country mc:countryName ?name .
        ...
        ?country mc:landboundary ?landboundary .
        ?landboundary rdf:type mc:Country .
                }
        WHERE{
        ?country rdf:type factbook:Country .
        ?country factbook:name ?name .
        ...
        ?country factbook:landboundary ?landboundary .
        }"""
    sparql.setQuery(construct_query)
    sparql.setReturnFormat(RDF)
    ...
```

In the end, a graph was defined and configured. The results obtained from the endpoint were assigned to this graph, which was then merged with the local ontology already populated with movies, resulting in a relatively complete local knowledge base about movies and countries, which can be found in *result_bonus.owl*, serialised in XML format.

```
    ...
    memory_store=IOMemory()
    graph_id=URIRef('http://www.semanticweb.org/store/movie_country')
    g = Graph(store=memory_store, identifier=graph_id)
    rdflib.plugin.register('sparql', rdflib.query.Processor, 'rdfextras.sparql.processor', 'Processor')
    rdflib.plugin.register('sparql', rdflib.query.Result, 'rdfextras.sparql.query', 'SPARQLQueryResult')


    g = sparql.query().convert()
    g.parse("result_basic.owl")
    g.serialize("result_bonus.owl", "xml")
    ...
```

The resulting ontology populated from the two remote ontologies is illustrated in *Image 8*. It now accommodates appropriate data about each country together with the information about movies.



untitled-ontology-19 (http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19) : [C:\Users\ZHANELYA\Desktop\QMUL-...

untitled-ontology-19 (http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19)

Class hierarchy: Country

- Thing
  - Country
  - Distributor
  - Language
  - Movie
  - Person
    - Actor
    - Cinematographer
    - Composer
    - Director
    - Producer
    - Screenwriter

Members list: Gree

Egypt
El_Salvador
Equatorial_Gui
Eritrea
Estonia
Ethiopia
Finland
France
France
Gabon
Gambia_The
Gaza_Strip
Georgia
Germany
Germany
Ghana
Gibraltar
Greece
Guatemala
Guinea
Guinea-Bissau
Guyana
Haiti
Honduras

Annotations: Greece

Annotations

Description: Greece

Types
Country

Same Individual As

Different Individuals

Property assertions: Greece

Object property assertions
landboundary Bulgaria
landboundary Macedonia
landboundary Albania
landboundary Turkey

Data property assertions
timeZone "UTC+2 (7 hours ahead of Washington, DC during Standard Time)"
areaWater "1140"
birthrate "9.62"^^double

Image 8. Populated movies-countries ontology

It was quite challenging to map the land boundary property from the Factbook remote ontology onto Country class individuals in the local store, as it was supposed to be referring to the Country class itself. However, after assuming that this type of relation does not really need any specific approach apart from defining it as being symmetric, I simply modelled it the same way that I was modelling a similar relation between two distinct classes.

Querying the Ontology populated from DBpedia and Factbook

Now, when the ontology was complete, I needed to test its implementation. In order to accomplish that, a python script was developed – *query_bonus.py* – containing four queries for my local ontology. First, the appropriate libraries were imported and configured (logging and rdflib), then, I created a graph by parsing the local ontology populated from two different sources. Finally, 4 queries were defined for my ontology.

Query1 is aimed to retrieve 10 movies filmed in countries with population more than 100M. Once the prefixes were defined for the local store, I selected distinct movie names and the corresponding countries that they were filmed in. In order to merge the information about movies and countries data, I selected the countries that have the same name as the countries where the movies were filmed in: `FILTER regex(str(?countryName_movie), str(?countryName_country))`. Finally, the movies-countries list was limited to those entries that only have movies filmed in countries with population over 100M: `FILTER (?population > 100000000)`, and the results were limited to 10 entries. The query results were then displayed in the terminal and the column names and dimensions were appropriately formatted. Any of the following queries were displayed in a similar manner.

```
...
query1="""
PREFIX mc: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?movieName ?countryName_country ?population
WHERE { ?film rdf:type mc:Movie .
    ?film mc:movieName ?movieName .
    ?film mc:filmedIn ?countryName_movie .
    ?country rdf:type mc:Country .
    ?country mc:countryName ?countryName_country .
    ?country mc:population ?population .
    FILTER regex(str(?countryName_movie), str(?countryName_country))
    FILTER (?population > 100000000)
    }LIMIT 10
    """
print ('{0:40s} {1:20s} {2:10s}'.format("Title","Filmed in","Population"))
print ('{0:40s} {1:20s} {2:10s}'.format("----------------------------------------","--------------------","------------------"))
for x,y,z in g.query(query1):
   print ('{0:40s} {1:20s} {2:10s}'.format(x.encode('utf-8'),y.encode('utf-8'),z))
...
```

Query2's main purpose is to return movies filmed in countries that have a capital “Buenos Aires”, i.e. in Argentina. After the appropriate prefixes were defined, I selected distinct movie names and the corresponding countries they were filmed in. Next, the countries information was merged with movies information by matching the country name with the country that each movie was filmed in. Finally, the results were limited to those that only have movies filmed in countries with a capital "Buenos Aires".

```
...
query2="""
PREFIX mc: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?movieName ?countryName_country ?capital
WHERE { ?film rdf:type mc:Movie .
    ?film mc:movieName ?movieName .
    ?film mc:filmedIn ?countryName_movie .
    ?country rdf:type mc:Country .
    ?country mc:countryName ?countryName_country .
    ?country mc:capital ?capital .
    FILTER regex(str(?countryName_movie), str(?countryName_country))
    FILTER regex(str(?capital),"Buenos Aires")
    }
...
```

Query3 was developed to retrieve movies filmed in countries with birthrate between 15 and 20. First, the appropriate prefixes were defined, then distinct movie names and the countries they were filmed in were selected. Next, movies and countries data was merged together and the results were limited to movies filmed in only those countries having their birthrate in a specified range.

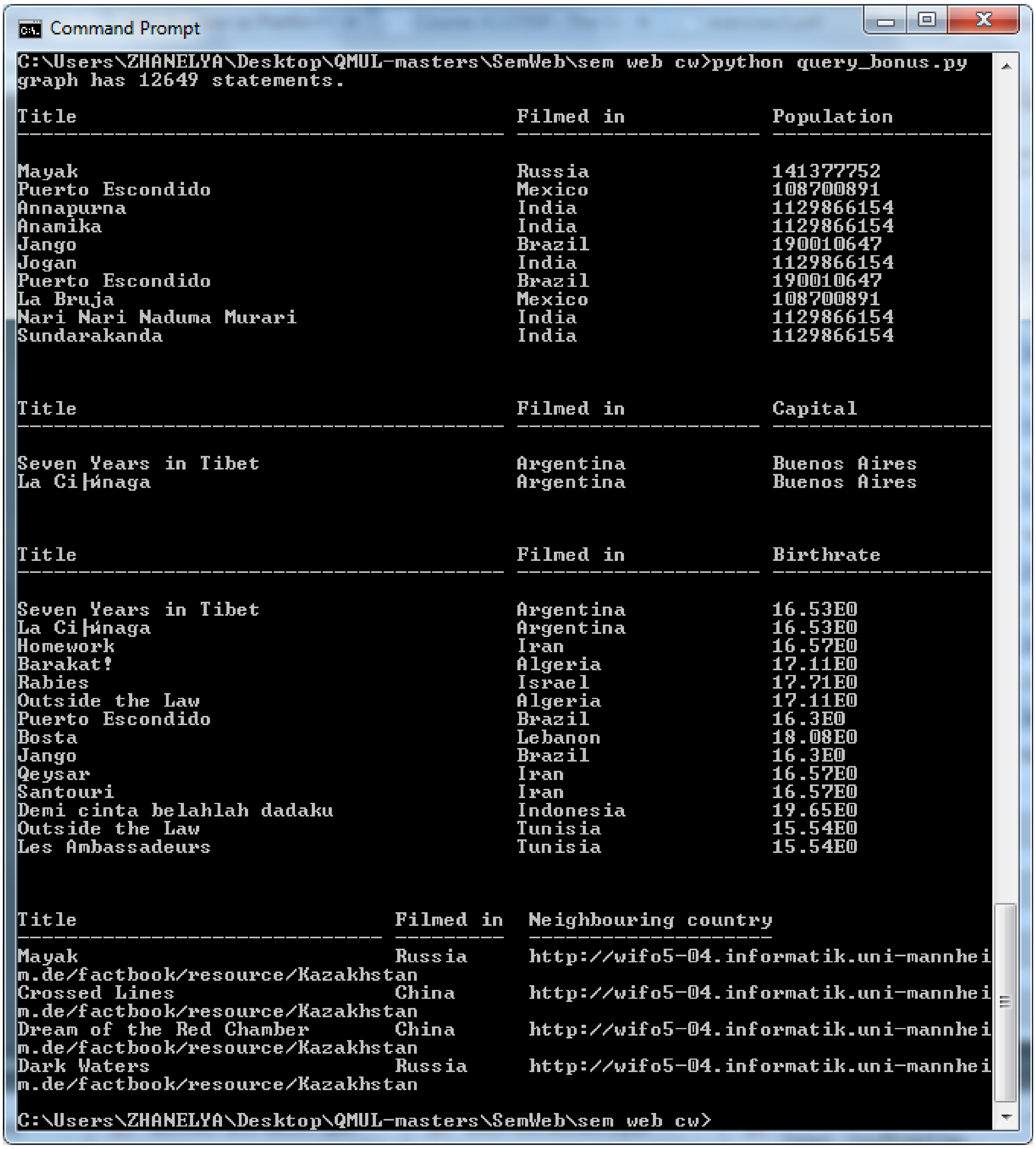
```
...
query3="""
PREFIX mc: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?movieName ?countryName_country ?birthrate
WHERE { ?film rdf:type mc:Movie .
    ?film mc:movieName ?movieName .
```

```
        ?film mc:filmedIn ?countryName_movie .
        ?country rdf:type mc:Country .
        ?country mc:countryName ?countryName_country .
        ?country mc:birthrate ?birthrate .
        FILTER regex(str(?countryName_movie), str(?countryName_country))
        FILTER (?birthrate > 10.00 && ?birthrate < 20.00)
        }
        """
...
```

The last query, Query4 was designed to return movies filmed in countries having common land boundary with Kazakhstan. In a similar manner, after prefixes were defined and distinct movie names and corresponding countries were selected, the data from two different ontologies stored now in a local store were merged together. Finally, the results were limited to those movies that were filmed in only those countries that are neighbouring with Kazakhstan.

```
...
query4="""
PREFIX mc: <http://www.semanticweb.org/zhanelya/ontologies/2015/2/untitled-ontology-19#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?movieName ?countryName_country ?landboundary
WHERE { ?film rdf:type mc:Movie .
        ?film mc:movieName ?movieName .
        ?film mc:filmedIn ?countryName_movie .
        ?country rdf:type mc:Country .
        ?country mc:countryName ?countryName_country .
        ?country mc:landboundary ?landboundary .
        FILTER regex(str(?countryName_movie), str(?countryName_country))
        FILTER regex(str(?landboundary),"Kazakhstan")
        }
        """
...
```

The output returned from this script querying the local ontology can be seen in *Image 9* below.



```
C:\Users\ZHANELYA\Desktop\QMUL-masters\SemWeb\sem web cw>python query_bonus.py
graph has 12649 statements.

Title                                    Filmed in            Population
---------------------------------------- -------------------- ------------------

Mayak                                    Russia               141377752
Puerto Escondido                         Mexico               108700891
Annapurna                                India                1129866154
Anamika                                  India                1129866154
Jango                                    Brazil               190010647
Jogan                                    India                1129866154
Puerto Escondido                         Brazil               190010647
La Bruja                                 Mexico               108700891
Nari Nari Naduma Murari                  India                1129866154
Sundarakanda                             India                1129866154



Title                                    Filmed in            Capital
---------------------------------------- -------------------- ------------------

Seven Years in Tibet                     Argentina            Buenos Aires
La Ci├ínaga                              Argentina            Buenos Aires



Title                                    Filmed in            Birthrate
---------------------------------------- -------------------- ------------------

Seven Years in Tibet                     Argentina            16.53E0
La Ci├ínaga                              Argentina            16.53E0
Homework                                 Iran                 16.57E0
Barakat!                                 Algeria              17.11E0
Rabies                                   Israel               17.71E0
Outside the Law                          Algeria              17.11E0
Puerto Escondido                         Brazil               16.3E0
Bosta                                    Lebanon              18.08E0
Jango                                    Brazil               16.3E0
Qeysar                                   Iran                 16.57E0
Santouri                                 Iran                 16.57E0
Demi cinta belahlah dadaku               Indonesia            19.65E0
Outside the Law                          Tunisia              15.54E0
Les Ambassadeurs                         Tunisia              15.54E0



Title                          Filmed in  Neighbouring country
------------------------------ ---------  --------------------
Mayak                          Russia     http://wifo5-04.informatik.uni-mannhei
m.de/factbook/resource/Kazakhstan
Crossed Lines                  China      http://wifo5-04.informatik.uni-mannhei
m.de/factbook/resource/Kazakhstan
Dream of the Red Chamber       China      http://wifo5-04.informatik.uni-mannhei
m.de/factbook/resource/Kazakhstan
Dark Waters                    Russia     http://wifo5-04.informatik.uni-mannhei
m.de/factbook/resource/Kazakhstan

C:\Users\ZHANELYA\Desktop\QMUL-masters\SemWeb\sem web cw>
```

Image 9. Querying movies-countries ontology

One of the main challenges at this step was to match the information about two different concepts in appropriate manner. To accomplish that, a common property was found for these concepts, namely the country name. Most *Movie* individuals have the *filmedIn* property which contains the country reference. All the *Country* individuals have the *countryName* property storing the country name. By matching these two properties it became possible to merge the data populated into ontology from two different sources. Another difficulty arose from the windows terminal specificity, as it cannot properly display non-utf8 symbols, for instance, symbols from languages other than English. To solve this issue, I was using y.encode('utf-8') for those entries that could potentially cause the problem when formatting the results and displaying them to the screen.

List of Files

#	**File Name**	**Description**
1	*SemanticWeb_CW_Zhanelya_Subebayeva.pdf*	Main report describing the process of building, populating and querying the ontology
2	*ontology.owl*	An initial Protégé-OWL ontology built
3	*catalog-v001.xml*	Catalogue generated automatically by Protégé for the ontology built
4	*result_basic.owl*	The ontology resulting from populating the original ontology from the first source (DBpedia)
5	*result_bonus.owl*	The resulting ontology after populating the ontology from the second source (Factbook) and combining it with the ontology populated from DBpedia
6	*basic.py*	The python script used to populate the ontology from the first SPARQL endpoint (DBpedia)
7	*bonus.py*	The python script used to populate the ontology from the second SPARQL endpoint (Factbook)
8	*query_basic.py*	The python script to query the local store, which is the ontology populated from the first SPARQL endpoint (result_basic.owl)
9	*query_bonus.py*	The python script to query the local store, which is the ontology populated from both SPARQL endpoints (result_bonus.owl)

How to run the code

I was running the code under the current settings of my machine, which is running Windows 7. When building the ontology I was using Protégé version 5.0.0 and Protégé default reasoner – HermiT 1.3.8.3. The python version that I was using for populating and querying my ontology is Python 2.7.5. There were some minor issues with Pyparsing version conflict, which was resolved by installing a specific version of Pyparsing – Pyparsing 1.5.7.

To run the python code, I used Windows Command Prompt – for instance entering *python basic.py* into the terminal populates the ontology stored in *ontology.owl* with movies information from the first source (Dbpedia), storing the resulting ontology in *result_basic.owl*; *python bonus.py* populates it from the second source (Factbook) storing the outcome in *result_bonus.owl*. Using *python query_basic.py* and *python query_bonus.py* will return the query results from querying the basic (populated from first source) and bonus (populated from 2 sources) ontologies correspondingly and display them on the terminal screen.

Bibliography

1. About: Histerical Blindness. The DBpedia Data Set (2014). Retrieved March 19, 2015, from http://dbpedia.org/page/Hysterical_Blindness_(film)
2. About: The Great Beauty. The DBpedia Data Set (2014). Retrieved March 19, 2015, from http://dbpedia.org/page/The_Great_Beauty
3. Afganistan. D2R Server for the CIA Factbook. (n.d.). Retrieved March 19, 2015, from http://wifo5-04.informatik.uni-mannheim.de/factbook/page/Afghanistan
4. Antoniou, G. and vanHarmelen, F. (2004). A Semantic Web Primer. MIT Press, Cambridge, MA, USA
5. D2R Server for the CIA Factbook. (n.d.). Retrieved March 19, 2015, from http://wifo5-04.informatik.uni-mannheim.de/factbook/
6. Thibodeau, Jr., T. (2015, January 8). The DBpedia Data Set (2014). Retrieved March 19, 2015, from http://wiki.dbpedia.org/Datasets#h434-2
7. Uzbekistan. D2R Server for the CIA Factbook. (n.d.). Retrieved March 19, 2015, from http://wifo5-04.informatik.uni-mannheim.de/factbook/page/Uzbekistan