MASWS Assignment 1 Part 2

Diljá Rudolfsdottir s0966087 & Steven Eardley s0934142

1. Tool Used to Run Queries

To run our SPARQL queries we used a java based tool named *twinkle*. It's useful because it provides a handy GUI (see Illustration 1) for writing and viewing results, but also support for both SPARQL and the more capable ARQ extended syntax. This allowed us to include functions like COUNT() for more interesting queries.

One disadvantage was found when using this tool: it was necessary to further decrease our data sample size. Our original data source was one year's worth of MOT test data, from which one month was converted. This required further reduction to just a single day, due to a *'Java heap space'* error message shown, causing failed queries (Illustration 2). This corresponds to 128, 775 MOT tests which is translated by our system into 2,833,056 lines of RDF. One day is a tiny portion of the original data set.

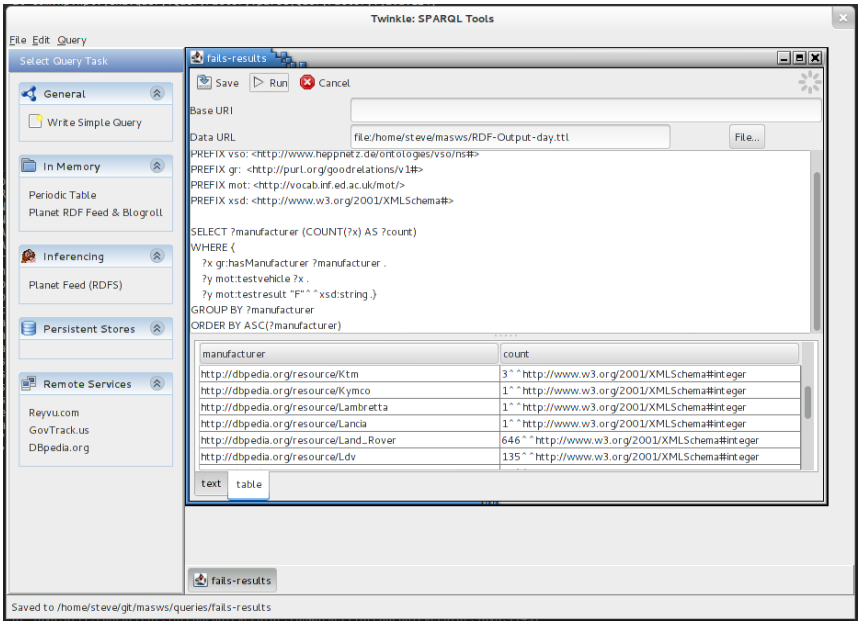

Illustration 1: User Interface for Twinkle

2. Changes to Conversion Code

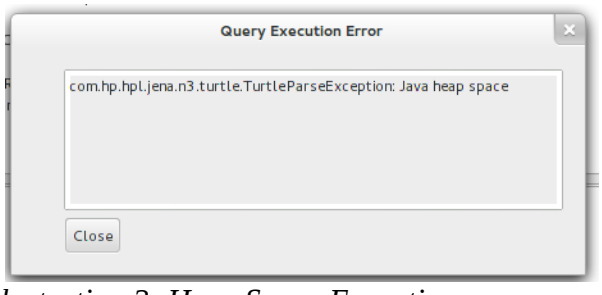

Illustration 2: Heap Space Exception

Initially the data failed to parse, requiring modifications to the java code submitted for part 1. The problems arised where vehicle manufacturer names were being translated into dbpedia URIs: while single-word manufacturers like 'RENAULT' were correctly converted to `dbpedia:Renault` (`http://dbpedia.org/resource/Renault`) but any manufacturer with more than one word, such as 'LAND ROVER' was converted to `dbpedia:Land rover.` This has now been changed to produce valid URIs more reliably: changing spaces for underscores and making the first letter of each word upper case, to produce e.g. `dbpedia:Land_Rover`, which correctly corresponds to `http://dbpedia.org/resource/Land_Rover`.

3. Query 1

The first query returns the model of each electric vehicle in the data file. This may be useful for people interested in finding out more about electricity as a fuel for transport.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX vso: <http://www.heppnetz.de/ontologies/vso/ns#>
PREFIX mot: <http://vocab.inf.ed.ac.uk/mot/>

SELECT  ?vehiclemodel

WHERE
      { ?x vso:fuelType dbpedia:Electricity .
        ?x mot:vehiclemodel ?vehiclemodel . }
```

This produces the following result:

```
-------------------------------------------------------------------------
| vehiclemodel                                                          |
=========================================================================
| "G-WIZ"^^<http://www.w3.org/2001/XMLSchema#string>                    |
| "PRIUS T SPIRIT VV-I AUTO"^^<http://www.w3.org/2001/XMLSchema#string> |
| "RX400 H SE CVT"^^<http://www.w3.org/2001/XMLSchema#string>           |
| "UNCLASSIFIED"^^<http://www.w3.org/2001/XMLSchema#string>             |
| "PRIUS T SPIRIT VV-I AUTO"^^<http://www.w3.org/2001/XMLSchema#string> |
| "PRIUS T3 VVT-I AUTO"^^<http://www.w3.org/2001/XMLSchema#string>      |
| "CIVIC EXECUTIVE IMA"^^<http://www.w3.org/2001/XMLSchema#string>      |
| "INSIGHT"^^<http://www.w3.org/2001/XMLSchema#string>                  |
| "UNCLASSIFIED"^^<http://www.w3.org/2001/XMLSchema#string>             |
| "PRIUS T3 VVT-I AUTO"^^<http://www.w3.org/2001/XMLSchema#string>      |
...
```

4. Query 2

Secondly, we find the top ten vehicles with the highest mileages on the odometer. This shows extracting deeper relations from the data (the float from the mileage object), and performing useful operations such as sorting the values.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX vso: <http://www.heppnetz.de/ontologies/vso/ns#>
PREFIX gr:  <http://purl.org/goodrelations/v1#>
PREFIX mot: <http://vocab.inf.ed.ac.uk/mot/>

SELECT ?manufacturer ?vehiclemodel ?value
```

```
      WHERE
            { ?x gr:hasManufacturer ?manufacturer .
              ?x vso:mileageFromOdometer ?mileage .
              ?x mot:vehiclemodel ?vehiclemodel .
                 ?mileage gr:hasValueFloat ?value }
      ORDER BY DESC(?value)
      LIMIT 10
-------------------------------------------------------------------------------
| manufacturer                          | vehiclemodel               | value
|
===============================================================================
| <http://dbpedia.org/resource/Renault>    | "CLIO 565 DCI"             | "999985" |
| <http://dbpedia.org/resource/Volkswagen> | "PASSAT S"                 | "990317" |
| <http://dbpedia.org/resource/Honda>      | "CR-V I-VTEC SE"           | "985870" |
| <http://dbpedia.org/resource/Skoda>      | "OCTAVIA CLASIC TDI"       | "979667" |
| <http://dbpedia.org/resource/Land_Rover> | "DEFENDER 110 COUNTY TD5"  | "978140" |
| <http://dbpedia.org/resource/Ford>       | "UNCLASSIFIED"             | "973313" |
| <http://dbpedia.org/resource/Ford>       | "UNCLASSIFIED"             | "973313" |
| <http://dbpedia.org/resource/Citroen>    | "SAXO VTR"                 | "960005" |
| <http://dbpedia.org/resource/Renault>    | "MASTER LM35 DCI"          | "941340" |
| <http://dbpedia.org/resource/Saab>       | "9-3 SE TURBO"             | "939661" |
-------------------------------------------------------------------------------
```

The types (e.g. `"999985"^^<http://www.w3.org/2001/XMLSchema#float>`) are omitted here for clarity.

5. Query 3

Here we are interested in reliability per manufacturer: producing a list, ordered alphabetically by car company, which shows how many of their vehicles failed MOTs. This demonstrates using variables `?x` and `?y` to relate tests and vehicles in a query. The use of COUNT requires switching the querying tool from SPARQL to ARQ syntax.

```
      PREFIX dbpedia: <http://dbpedia.org/resource/>
      PREFIX gr:  <http://purl.org/goodrelations/v1#>
      PREFIX mot: <http://vocab.inf.ed.ac.uk/mot/>
      PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

      SELECT ?manufacturer (COUNT(?x) AS ?count)
      WHERE {
              ?x gr:hasManufacturer ?manufacturer .
              ?y mot:testvehicle ?x .
              ?y mot:testresult "F"^^xsd:string .}
      GROUP BY ?manufacturer
      ORDER BY ASC(?manufacturer)
```

```
-----------------------------------------------------------
| manufacturer                                | count |
===========================================================
| <http://dbpedia.org/resource/Aixam>         | 2     |
| <http://dbpedia.org/resource/Ajs>           | 1     |
| <http://dbpedia.org/resource/Alfa_Romeo>    | 112   |
| <http://dbpedia.org/resource/Aprilia>       | 6     |
| <http://dbpedia.org/resource/Aston_Martin>  | 5     |
| <http://dbpedia.org/resource/Audi>          | 717   |
| <http://dbpedia.org/resource/Austin>        | 17    |
| <http://dbpedia.org/resource/Austin_Morris> | 5     |
| <http://dbpedia.org/resource/Auto-trail>    | 3     |
| <http://dbpedia.org/resource/Barossa>       | 2     |
...
```

6. Query 4

The fourth query demonstrates another possible use for our data set – environmental interest. It groups and counts the vehicles by fuel type.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX vso: <http://www.heppnetz.de/ontologies/vso/ns#>

SELECT ?fuel (COUNT(distinct ?x) AS ?count)
WHERE { ?x vso:fuelType ?fuel .}
GROUP BY ?fuel
ORDER BY ?count
LIMIT 10
```

```
-------------------------------------------------------------------
| fuel                                                  | count |
===================================================================
| <http://dbpedia.org/resource/Liquefied_petroleum_gas> | 120   |
| <http://dbpedia.org/resource/Alternative_fuel_vehicle> | 28    |
| <http://dbpedia.org/resource/Diesel_fuel>             | 38598 |
| <http://dbpedia.org/resource/Electricity>             | 59    |
| <http://dbpedia.org/resource/Gasoline>                | 78985 |
-------------------------------------------------------------------
```

7. Query 5

This query supposes a car club is interested in creating their own data set of Land Rovers with classic car status – older than 1973. Using the CONSTRUCT statement, the data from our set can be gathered using a different ontology: this time using `vso:` the Vehicle Sales Ontology. The result is a valid RDF data set in XML format.

```
      PREFIX dbpedia: <http://dbpedia.org/resource/>
      PREFIX vso: <http://www.heppnetz.de/ontologies/vso/ns#>
      PREFIX gr:  <http://purl.org/goodrelations/v1#>
      PREFIX mot: <http://vocab.inf.ed.ac.uk/mot/>
      PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

      CONSTRUCT  { ?x a vso:Automobile .
                   ?x mot:vehiclemodel ?vehiclemodel .
                   ?x vso:color ?colour .
                   ?x vso:fuelType ?fueltype .
                   ?x vso:engineDisplacement ?disp .
                   ?x vso:firstRegistration ?regdate}

      WHERE
            { ?x mot:vehiclemodel ?vehiclemodel .
              ?x gr:hasManufacturer dbpedia:Land_Rover .
              ?x vso:color ?colour .
              ?x vso:fuelType ?fueltype .
              ?x vso:engineDisplacement ?disp .
              ?x vso:firstRegistration ?regdate .
              FILTER (?regdate < "1973-01-01"^^xsd:date) }
```

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vso="http://www.heppnetz.de/ontologies/vso/ns#"
    xmlns:gr="http://purl.org/goodrelations/v1#"
    xmlns:dbpedia="http://dbpedia.org/resource/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:mot="http://vocab.inf.ed.ac.uk/mot/" >
  <rdf:Description rdf:nodeID="A0">
    <vso:engineDisplacement rdf:nodeID="A1"/>
    <vso:color>GREEN</vso:color>
    <rdf:type rdf:resource="http://www.heppnetz.de/ontologies/vso/ns#Automobile"/>
    <vso:fuelType rdf:resource="http://dbpedia.org/resource/Gasoline"/>
    <vso:firstRegistration rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1958-01-
01</vso:firstRegistration>
    <mot:vehiclemodel
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">UNCLASSIFIED</mot:vehiclemodel>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A2">
    <vso:firstRegistration rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1965-09-
07</vso:firstRegistration>
    <rdf:type rdf:resource="http://www.heppnetz.de/ontologies/vso/ns#Automobile"/>
    <mot:vehiclemodel rdf:datatype="http://www.w3.org/2001/XMLSchema#string">88 - 4
CYL</mot:vehiclemodel>
```

```
    <vso:engineDisplacement rdf:nodeID="A3"/>
    <vso:color>BLUE</vso:color>
    <vso:fuelType rdf:resource="http://dbpedia.org/resource/Diesel_fuel"/>
  </rdf:Description>
...
</rdf:RDF>
```

8. Benefits of SPARQL

While the syntax of SPARQL seems in some way similar to that of XML and SQL, for example using the SELECT and UNION operators, the way it works is completely different. When querying XML the query is a serialisation. That is, it is a path that tells you exactly what you are looking for and in what location. This location goes all the way from the root of the document to the node. When using SPARQL to query RDF however the approach is completely different: you do not have to know the structure of the entire serialisation, as in XML, but instead you need to know the structure of the RDF graph; which parts are connected and what connects them. When you know this you can use SPARQL's idea of "pattern matching" - you can specify the triples you are looking for and then simply search the graph for instances that match those triples. You can use graph expressions to make more complicated queries and solution modifiers to select how you want the output to be presented but in its simplest form, its all about triples. When using SPARQL therefore, you do not need to worry about what follows what, such as in the serialisation form of XML. Instead you just have to know what connect to what, specify what you'd want to know and which triples you need in order to get this information. To explain this further, let us consider a very small, simplified, subset of our source dataset:

```
7172659|P|PE|YAMAHA|YQ 50 AEROX|WHITE|P|
9974237|P|SO|PEUGEOT|SPEEDFIGHT|BLUE|LPG|
16674952|F|E|VOLKSWAGEN|SHARAN SL TDI AUTO|SILVER|D|
```

In our simplified dataset the last column represents the fuel type used by the car. In the examples above these are P (petrol), LPG (liquefied petroleum gas) and D (diesel). If you'd want to find all of the models of cars which use diesel fuel you could use queries such as:

XML

```
/VehicleModel/FuelType/Diesel
```

SPARQL

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX vso: <http://www.heppnetz.de/ontologies/vso/ns#>
PREFIX mot: <http://vocab.inf.ed.ac.uk/mot/>
```

```
SELECT  ?vehiclemodel

WHERE
      { ?x mot:vehiclemodel ?vehiclemodel.
        ?x vso:fuelType dbpedia:Diesel_fuel. }
```

Let us now consider that we merge our dataset with another dataset, which tells us which fuel types are environmentally friendly and which aren't. This simple dataset is shown below:

```
Environmental|LPG |
Environmental|Electric |
Environmental|Steam |
Environmental|CNG |
Environmental|LNG |
Environmental|Fuel_Cells |
NonEnvironental|Petrol |
NonEnvironmental|Diesel |
```

If we imagine that we would somehow manually merge this new dataset so it would slot in the XML serialisation, then the query that searches for models of diesel cars will have changed for XML such that:

XML

```
/VehicleModel/FuelType/NonEnvironmental/Diesel
```

How about SPARQL then? In this case the SPARQL query will be exactly the same as above. This small example shows in practice the big benefits of SPARQL over XML and other querying languages. SPARQL doesn't have to concern itself every single time a dataset it merged with a new one. While XML has to change the path of its queries every single time new data is added, SPARQL does not have to do this - unless in fact it aims to query some of this added data as well.

Consider the scenario of adding new data to our source dataset but in this case there's an added column which tells us which fuel types are environmentally friendly and which aren't. An example of this is below:

```
21095460|ABR|B|RENAULT|CLIO RN|GREEN|NonEnvironmental|P|
32853419|ABR|B|PEUGEOT|206 LX D|GREEN|NonEnvironmental|D|
1195906|F|BD|VAUXHALL|ZAFIRA ELEGANCE 16V|BLACK|Environmental|E|
```

This could for example be more recent data than the one in the format above, but we want to be able to merge them all together because - really - the only difference is the extra information about the environmental/non environmental fuel. Easy right? Not for XML. Merging the six samples above would give us the following dataset:

```
7172659|P|PE|YAMAHA|YQ 50 AEROX|WHITE|P|
9974237|P|SO|PEUGEOT|SPEEDFIGHT|BLUE|LPG|
16674952|F|E|VOLKSWAGEN|SHARAN SL TDI AUTO|SILVER|D|
21095460|ABR|B|RENAULT|CLIO RN|GREEN|NonEnvironmental|P|
32853419|ABR|B|PEUGEOT|206 LX D|GREEN|NonEnvironmental|D|
1195906|F|BD|VAUXHALL|ZAFIRA ELEGANCE 16V|BLACK|Environmental|E|
```

If we now wanted to know which models of cars are diesel fuelled we would have to use two different queries for XML, one for each format or:

```
/VehicleModel/FuelType/Diesel
```

and

```
/VehicleModel/FuelType/NonEnvironmental/Diesel
```

Again, how about SPARQL? How does SPARQL diesel query change with the addition of a slightly different data? Answer: it doesn't! This feature is absolutely crucial for the Semantic Web because it relies on being able to integrate data from many different sources, which may not always have the exact same format. Therefore there is no need to change any old queries previously made to fit the new merged dataset, unless of course you mean to query some of the new additions. For XML however all your old queries would most likely have to be changed to fit the new structure of the serialisation.

How does this relate to the query results we acquired? Since our data is rather straightforward, we would most likely have been able to get the same results shown above using XML queries if the original dataset had been formatted into XML. When it comes to adding new data however, XML would have ran into trouble when querying while our RDF graph does not. To prove this we merged a small subset of dataset with the very small new dataset of environmental fuels mentioned above. The new RDF graph looks the same as before but has the following additions:

```
_:1 mot:environmental dbpedia:Liquefied_petroleum_gas.
_:2 mot:nonEnvironmental dbpedia:Diesel_fuel.
_:3 mot:nonEnvironmental dbpedia:Gasoline.
_:4 mot:environmental dbpedia:Electricity .
_:5 mot:environmental dbpedia:Steam .
_:6 mot:environmental dbpedia:Fuel_cell .
_:7 mot:environmental dbpedia:Compressed_natural_gas .
_:8 mot:environmental dbpedia:Liquefied_natural_gas .
```

The new RDF graph was therefore simply just copied into the same file as the original one and no further actions were required to merge the two. To test this new database we created a SPARQL query, designed to return all of the vehicle models which used non environmental fuel. The query is below:

SPARQL

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX vso: <http://www.heppnetz.de/ontologies/vso/ns#>
PREFIX mot: <http://vocab.inf.ed.ac.uk/mot/>

SELECT  ?vehiclemodel

WHERE
      {?x mot:vehiclemodel ?vehiclemodel.
       ?x vso:fuelType ?type.
       ?y mot:nonEnvironmental ?type. }
```

As we can see from this example the new query does not require us to make any drastic changes to the format, it is in fact exactly like the query above but in this case the fuelType we are searching for is not diesel but instead any non environmental fuel. SPARQL can do that very easily on the new merged data.

How about if we wanted to do the same for XML? Without being big XML experts, we know that merging the two datasets would require a lot more work than the amount shown above - which simply consists of adding one more triple and making sure it matched one of the previous ones. XML would require us to try and slot the new information into some appropriate place in the serialisation and defining for every single known vehicle whether their fuel type is environmental or not. Instead SPARQL can just take the fuel type we already know each vehicle has and match it with the new information which tells us which types of fuel are environmental and which aren't.