SENSE4US

SEVENTH FRAMEWORK PROGRAMME

Initial Investigation into Tools & Techniques for Distributed Searching of Open Data Repositories

Project acronym:	**SENSE4US**
Project full title:	**Data Insights for Policy Makers and Citizens**
Grant agreement no.:	**611242**
Responsible:	**University of Koblenz-Landau**
Contributors:	**Giuseppe Pirrò**
Document Reference:	**D4.3.1**
Dissemination Level:	**Internal**
Version:	**Final**
Date:	**29/01/2015**

SEVENTH FRAMEWORK PROGRAMME
European Commission
Project co-funded by the European Union under the Seventh Framework Programme

History

Version	*Date*	*Modification reason*	*Modified by*
0.1	15/12/2014	Initial Draft	Giuseppe Pirrò
0.2	6/1/2015	Comments	Steve Taylor
0.3	29/1/15	Final deliverable	Giuseppe Pirrò

Table of contents

Table of Contents

List of figures

List of abbreviations

<SPARQL>	**<SPARQL Protocol and RDF Query Language>**
<RDF>	**<Resource Description Framework >**
<RDFS>	**<Resource Description Framework Schema>**
<SKOS>	**<Simple Knowledge Organization System>**
<OWL>	**<Web Ontology Language>**
<BGP>	**<Basic Graph Pattern>**
<KG>	**<Knowledge Graph>**

Executive summary

This document accompanies the prototype (software) deliverable D4.3.1, which was delivered by the University of Koblenz to IT Innovation on 15 Dec 2014 for integration into the Sense4us prototype. This document is a brief overview of the prototype.

The aim of this prototype deliverable is to give some insights on the techniques and tools that have been developed in WP4 regarding the enrichment of entities, identified within policy documents, with structured descriptions available in Linked Open Data sources on the Web.

This will help the policy maker user of Sense4us by showing them concepts they may not know about that are related to concepts they know already, thus enriching their knowledge and providing links to new information that is relevant to their enquiry.

1. Introduction

This document accompanies the prototype (software) deliverable D4.3.1, which was delivered by the University of Koblenz to IT Innovation on 15 Dec 2014 for integration into the Sense4us prototype. This document is a brief overview of the prototype.

The aim of this prototype deliverable is to give some insights on the techniques and tools that have been developed in WP4 regarding the enrichment of entities, identified within policy documents, with structured descriptions available in Linked Open Data sources on the Web.

This prototype deliverable will help the policy maker user of Sense4us by showing them concepts they may not know about that are related to concepts they know already, thus enriching their knowledge and providing links to new information that is relevant to their enquiry.

1.1 Background

In the last years there has been an increasing trend in sharing and interlinking pieces of data on the Web. This (r)evolution is turning the classical Web, focused on hypertext documents and syntactic links among them, into a Web of Linked Data. In this new setting, Uniform Resource Identifiers (URIs) are used not only to identify Web documents and digital content, but also new kinds of resources such as real world objects (e.g., people, places, football teams) and abstract concepts (e.g., sport, philosophy, geography). Descriptions (or representations) for these resources can be obtained, in the same spirit of traditional documents, by dereferencing their associated URIs via the HTTP protocol.

Semantic links and descriptions are expressed by using a common data format, that is, the Resource Description Framework (RDF). RDF is a simple data model, which enables the making of statements about resources. Statements are triples in the form of subject, predicate and object, with the predicate expressing some relation between the subject and the object. Sets of RDF triples can be thought of as labelled graphs. RDF enables describing domain-specific entities and their relations by using taxonomies, vocabularies and ontologies expressed in formal languages such as RDF(S), SKOS and OWL.

Structured data are becoming a necessary support toward the next generation of search services. Indeed, major search engines such as Google and Yahoo! are complementing their indexes, build from unstructured documents (e.g., HTML pages), with structured. The main difference with respect to the Web of Linked Data is that data is not open and the querying capabilities are limited.

For Linked Open Data in the RDF format, there is a standard query language, called SPARQL. It resembles SQL plus a protocol to address the querying of structured data on the Web. Usually, Linked Open Data sources feature *SPARQL endpoints*, that is, network locations where SPARQL queries can be posed. This makes it possible to query in a structured way hundreds of RDF data sources covering diverse domain ranging from public sector data to general knowledge.

1.2 Application for Sense4us & Benefits for the Policy Maker End User

We will focus on the case of data sources that keep data in the RDF standard data format. The motivation is twofold; on one hand, there is an abundance of RDF data sources in the Linked

Open Data cloud that cover disparate domains; on the other hand, data in RDF can be conveniently queried upon by using the SPARQL standard query language for RDF.

We will describe a methodology to discover connectivity structures between a pair of entities given by the user. We will refer to such connectivity structures as *relatedness explanations*. An explanation is a graph whose structure represents the set of paths between a source and a target entity; it aims at providing some (visual) insight of why two entities are related, and what other entities are also related.

In order to uncover the nature of an explanation we will describe a methodology, which only requires access to a (remote) SPARQL query endpoint; no data pre-processing is needed. This makes our approach datasource-agnostic and readily available in many online RDF data sources. The methodology described in this document has been implemented and the source code has been delivered to the Sense4us coordinator and integrating partner, IT Innovation.

A case in point uses DBpedia[1], which is an RDF-encoded version of Wikipedia. A user may know some concepts that they wish to know more about, for example “Germany” and “Renewable Energy”. This prototype shows other entities in DBpedia (each entity corresponds to a Wikipedia page) that connect the two concepts. So in our example, we may find sources of renewable energy in Germany that have Wikipedia pages, German companies that specialise in technology to harness renewable energy, political issues surrounding renewable energy in Germany, and so on. The result is a graph of the paths arising from following links in the Wikipedia pages that connect the two concepts.

Therefore the key benefit of this work to the end user is that they can be shown entities (in our example Wikipedia pages) related to the user’s query of interest that they potentially did not know about, this increasing their knowledge. The user can put in as many pairs of entities as they like and the relationships between them will be shown.

The prototype is not limited to DBpedia – it is applicable to any data source encoded in RDF and with a SPARQL endpoint – both common features of semantically enriched data sources today.

[1] http://dbpedia.org/About

2 Conceptual Architecture

A high level overview of methodology is given in Figure 1. On the left hand side, the *Explanation Framework* represents the part of interest for the subsequent discussion. On the right hand side, the *Knowledge Base* is basically an RDF data source (e.g. available in the Linked Open Data Cloud) with a SPARQL endpoint that is used to pose queries in the SPARQL language for RDF.

Generally speaking, the explanation framework, which has been implemented as the software component of this deliverable, works as follows: the input consists in the URIs identifying a source (`Ws`) and a target entity (`Wt`), and an integer value K; this value represents the maximum length of paths considered between `Ws` and `Wt`. Although paths between entities can have an arbitrary length, in practice it has been shown that for Knowledge Graphs (KGs) like Facebook[2] the average distance between entities is bound by a value K<5. The choice of considering paths of length K in our approach is also reasonable on the light of the fact that we focus on providing explanations of manageable size that can be given visual representation and then interpreted by the user.

Note that the explanation framework does not require to have data locally; it only requires the availability of an endpoint on the knowledge base. The output produced by the explanation framework is a graph, that is, *Ge* in Figure 1. Such graph consists in the set of paths (of length at most K) connecting `Ws` and `Wt`. In the remainder of the document we give some more insights on the two main components of the explanation framework, that is, the *connectivity query builder* and the *explanation builder*.

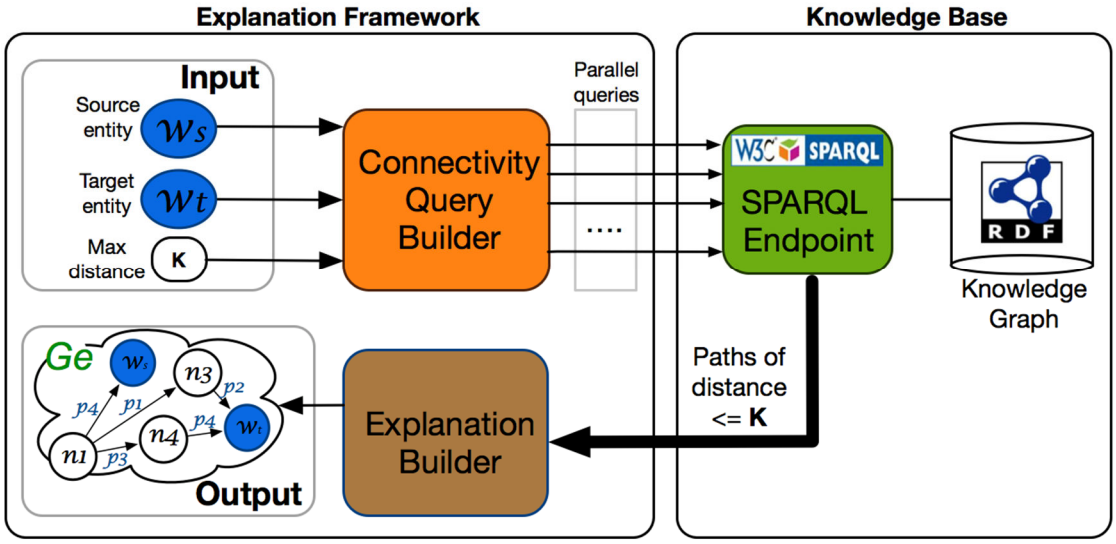

Figure 1. Overview of the Approach

2.1 The Connectivity Query Builder

This module takes as input the identifiers (i.e., URIs) of the source entity (`Ws`), the target entity (`Wt`); moreover, it also requires an integer value K. The output is a set of SPARQL queries. Such queries have the aim of enabling to retrieve the set of paths of length at most K connecting `Ws` and `Wt`. Before showing how the connectivity builder works we give a little bit of background on the SPARQL query language.

[2] https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859

2.1.1 The SPARQL query language: a brief overview

In what follows we focus on the subset of SPARQL that is of interest for the explanation framework. We are interested in the most basic form of SPARQL queries, that is, Basic Graph Patterns (BGPs); we shall also make usage of simple aggregate operators and in particular COUNT. Let V be a set of SPARQL variables, that is, strings starting with the `?` symbol, *U* a set of URIs and *L* a set of literals. A *triple pattern* P is a triple of the form:

$$P=(U \cup L \cup V) \times (U \cup V) \times (U \cup L \cup V)$$

BGPs are sets of triple patterns that can be combined via operators; we will make usage of the join operator (represented by the symbol. in the SPARQL syntax). To give an example of how a SPARQL query looks like, consider the RDF graph in Figure 2 and the SPARQL query

```
SELECT ?pl WHERE {:Fritz_Lang dbpo:birthPlace ?pl }
```

Such query selects the *bindings* of the variable ?pl. The triple pattern in the WHERE clause looks for triples of the form (`:Fritz_Lang dbpo:birthPlace ?pl`) and binds, in this case, `?pl` to *Vienna*; indeed this is the only triple in Figure 2 satisfying the triple pattern. On the other hand, the query

```
SELECT COUNT(?p) WHERE {:Fritz_Lang ?p ?o}
```

counts the number of triples in which the URI `:Fritz_Lang` is the subject (6 in this example).

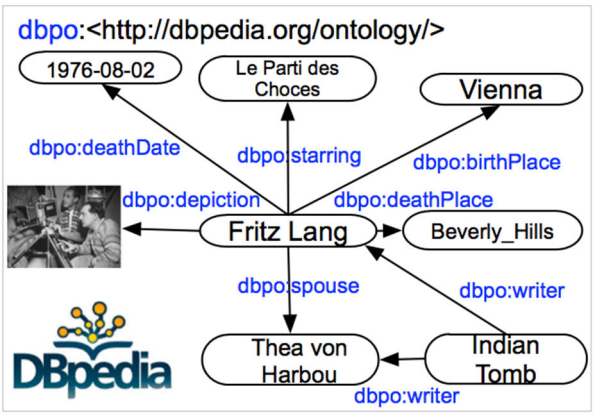

Figure 2. An example of RDF graph.

2.1.2 Finding Paths via SPARQL

As briefly mentioned before, SPARQL enables to retrieve information from an RDF graph. In what follows we discuss how to leverage SPARQL for the purpose of building explanations. In more detail, given a source entity `Ws`, a target entity `Wt` and a value `K` SPARQL enables to find paths of length K by leveraging BGPs. **Error! Reference source not found.** shows the general structure of a BGPs to retrieve paths.

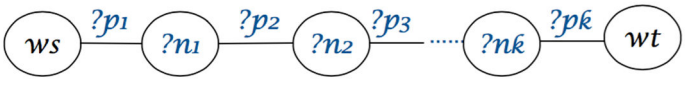

Figure 3. General structure of BGPs to retrieve paths.

As it can be observed, the endpoints of the BGP are `Ws` and `Wt` ; moreover note that exactly K variables representing edges (connections) are used. Note that in Figure 3 directions on the

edges have not been reported. This is because in order to get all the paths, all the combination of ingoing/outgoing edges must be considered. We will clarify this reasoning with an example.

Suppose that we are interested in finding paths of length K≤2 connecting `Ws=:Fritz_Lang Wt=:Thea_von_Harbou`. In order to consider all the possible cases, the following set of SPARQL queries must be used.

```
SELECT * WHERE{:Fritz_Lang ?p1 :Thea_von_Harbou}
SELECT * WHERE{:Thea_von_Harbou ?p1 :Fritz_Lang}
SELECT * WHERE{:Fritz_Lang ?p1 ?n1. ?n1 ?p2 :Thea_von_Harbou}
SELECT * WHERE{:Fritz_Lang ?p1 ?n1. :Thea_von_Harbou ?p2 ?n1}
SELECT * WHERE{?n1 ?p1 :Fritz_Lang. :Thea_von_Harbou ?n1 ?p2}
SELECT * WHERE{?n1 ?p1 :Fritz_Lang. ?n1 ?p2 :Thea_von_Harbou}
```

Figure 4. SPARQL queries to find paths of length K≤ 2 .

As it can be observed, in Figure 4, the first two queries consider paths of length one; since a path may exist in two directions, two queries are required. Indeed, note that the URI `:Fritz_Lang` appears in the first position of the triple pattern while in the second query it appears as object. The retrieval of paths of length 2 requires 4 queries. In general, given a value K, to retrieve paths of length K, 2^k queries are required.

Note that to make efficient the retrieval of paths from a SPARQL endpoint, queries can be sent in parallel by the *Connectivity Query Builder* (see Figure 1). In order to achieve this goal, our implementation leverages multithreading.

The set of paths retrieved via the will be then passed to the *Explanation Builder* described in the next section.

2.2 The Explanation Builder

This module takes in input a set of paths obtained from a SPARQL endpoint and generates a graph including these paths by merging them. The merging phase consists of generating a single node for the same URI, which can potentially appear in more than one path. From an implementation point of view, the merging will be handled via the Jena library. We describe the organization of the source code implementing the explanation framework next.

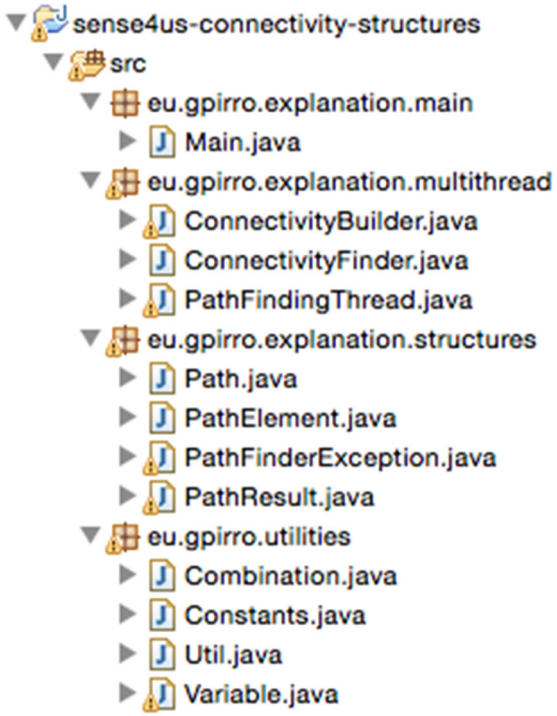

Figure 5. Structure of the implementation

Figure 5 describes the structure of the Java classes implemented. In order to test the application, there is a Main class in the package **eu.gpirro.explanation.main**.

Parameter values that control the execution (e.g., value of K, number of thread, endpoint address) are set in the class **eu.gpirro.utilities.Constants**. The package **eu.gpirro.explanation.multithread** controls the multithread execution of the set of queries.

In particular, the class PathFinding Thread represents the concrete implementation of a Thread, which receives a query as input, executes it and insert results in the shared data structure handled by the *monitor* class ConnectivityFinder. Moreover, the last thread is responsible for notifying the termination of the path finding phase (via a call to the method *shutdown* in the class ConnectifityFinder).

The second phase of the algorithm is handled by the class **eu.gpirro.explanation.multithread ConnectivityBuilder**. This class receives as input the set of paths and generates the explanation as the merge of all paths. This latter class also provides a method to save the results on disk. Results can be saved in different RDF serializations (e.g., N-TRIPLE, XML/RDF).

3 Conclusions and Next Steps

The concepts described in this document have been implemented in Java, tested and released to the Sense4us consortium ready for integration in the first prototype. The input consists of the Ws and Wt URIs, and a maximum path length K. The output is a set of RDF triples that describe the graph, and these can be visualised using off the shelf RDF visualisation tools, and this will be explored in WP7.

The code is tested and when given the DBpedia SPARQL endpoint, confirmed to deliver the connections found in Wikipedia, as described in the example above.

The next work planned is to refine the connectivity structures produced by the tool according to different strategies among which:

- Informativeness: selecting a subset of paths according to the relative importance of the predicates in these paths
- Diversity: selecting a sufficiently heterogeneous (in terms of predicates) set of paths
- Combination of informativeness and diversity.