NATURAL LANGUAGE INTERFACE FOR QUERYING LINKED DATA USING CASE-BASED REASONING

Hasna Boumechaal

LIRE Laboratory, abdelhamid mehri Constantine 2 University,
Constantine, Algeria
hasna.boumechaal@univ-constantine2.dz

Zizette Boufaida

LIRE Laboratory, abdelhamid mehri Constantine 2 University,
Constantine, Algeria
zizette.boufaida@univ-constantine2.dz

Abstract

With the emergence of Linked Data in RDF format, query systems should execute structured queries in SPARQL. Still, only expert programmers can specify their information needs and cope with Linked Data sources' diversity and heterogeneity. Therefore, understanding Natural Language (NL) queries to construct correct SPARQL queries is a great challenge for these query systems to access multiple heterogeneous semantic sources and Linked Data sets. This paper presents a new approach for querying Linked Data with NL queries. Our method identifies the topic and search criteria of the NL query based on the Case-Based Reasoning (CBR) technique. Next, we match the identified topic and search criteria to their corresponding entities in the dataset. Then, we represent the query as triples based on the semantic relations provided by the dataset. Finally, our system executes the SPARQL queries generated from the query triples, based on the proposed generation rules, to query the dataset and retrieve the relevant answers. Experiments showed that the proposed approach is efficient compared to the existing systems in terms of precision and recall.

***Keywords*: Semantic Web; Linked Data; Case-Based Reasoning (CBR); Question Answering Systems; SPARQL.**

1. Introduction

To query Linked Data [Bizer *et al.* (2009)] on the Web today, users need to understand the structure and vocabularies used in Linked Data sources and formulate the query using a structured query language syntax as SPARQL [Prud'hommeaux and Seaborne (2008)]. The latter is necessary when accessing information described in RDF (Resource Description Framework) format. In particular, RDF is the format used to enable improved and more intelligent use of Linked Data. RDF relates entities in the subject-predicate-object format where the subject and object are related to each other by the predicate. The problem is that users can not already construct the SPARQL query to suit their needs. Note that forming SPARQL queries to query RDF datasets is not the easiest task, even for expert users. If Natural Language (NL) queries can be translated into SPARQL queries, naive users can specify their information needs without learning SPARQL.

Therefore, it is necessary to define new approaches to query distributed Linked Data sources with NL queries. However, understanding NL queries to build correct formal queries such as SPARQL queries is a tremendous challenge for existing query systems to access multiple heterogeneous semantic and Linked Data sets. This work proposes a new approach for querying the DBpedia[a] ontology, one of the most well-known parts of the decentralized Linked Data effort. In particular, DBpedia offers information extracted from unstructured sources, such as Wikipedia. We need to produce correct answers to user queries, which are harder to find because some challenges need to be solved in NL query processing:

- First, we need to understand the user's query intent. This means filtering out relevant terms to disambiguate the meaning of NL queries.

[a]http://wiki.dbpedia.org/downloads-2016-10#dbpedia-ontology

- Next, we map these relevant terms to specific concepts in the queried datasets. This requires the mapping of natural language terms to specific concepts.
- Finally, we generate correct formal queries, which can be difficult as there are still many challenges in expressing NL queries in the syntax of formal languages.

The proposed system focuses on the following strategies to overcome these challenges:

(1) Each NL query has two main parts, the subject (topic) and the search criteria representing a description of that subject. Our system uses Case-Based Reasoning (CBR) [De Mántaras *et al.* (2005)] to identify all relevant terms in the NL query, such as the subject and the search criteria. First, we initialize the case base by different type of queries using Stanford Dependencies [De Marneffe *et al*, (2008)]. Then, the enrichment of the case base by the solved queries is done by learning new cases.

(2) We determine the meaning of a given query by representing that meaning as concepts and roles in datasets. This mapping of NL terms to corresponding entities is achieved using a developed disambiguation algorithm, the WordNet dictionary [Miller *et al.* (1990)] and user dialogues.

(3) We explore the richness of the semantic relationships provided by the datasets to represent queries in the form of valid triples. This is done using a special module that facilitates the formalization of queries based on the proposed generation rules.

The remainder of the paper is organized as follows: Section 2 presents existing research in the field of linked data querying. Section 3 discusses the proposed approach in which we describe the main modules of the proposed system. Then, a test is presented in Section 4. The conclusion and perspectives are discussed in Section 5.

2. Related Work

As Linked Data becomes a major component of the Semantic Web, the easiest way to make these data sets accessible to users is to develop new systems for querying Linked Data sources. In particular, these systems accept a natural language query, create a formal query from it, and execute it to retrieve the relevant answers. In this context, several systems for querying Linked Data by NL queries have been developed. [Damljanovic *et al.* (2011)] proposed the Natural Language Interface (NLI) FREyA to query Linked Open Data (LOD). FREyA delegates a large part of the semantic matching and disambiguation process to users. User choices are used to train the system to improve its performance over time. [Yahya *et al.* (2012)] proposed a question answering system based on an integer linear program to solve several disambiguation tasks jointly: segmenting questions into sentences, mapping sentences to entities, classes and semantic relations, and building SPARQL triple patterns.

The CASIA@V2 question answering system on Linked Data (DBpedia) was proposed by [Shizhu *et al.* (2014)] to translate natural language questions into structured SPARQL queries. This system is based on a joint learning framework using Markov Logic Network (MLN) for phrase detection, phrases mapping to semantic items and semantic items grouping. [Freitas and Curry (2014)] aimed to provide a NLI and an associated semantic index to support an increased level of vocabulary independency for queries on Linked Data/Semantic Web datasets distributional-compositional semantic approach. Xser [Xu et *al*. (2014)] is a Linked Data Question Answering System (DBpedia), converting users' natural language questions into structured queries. This work proposes an efficient pipeline framework to model a user's query intent as a sentence-level dependency DAG which is then instantiated according to a given KB to construct the final structured query. gAnswer [Zou et *al.* (2014)] is a question answering system that processes questions based on a semantic query graph to model the query intent in the natural language question structurally. First, based on the dependency analysis of the question, a graph is constructed that represents the semantic structure of the question. Then, this graph is mapped to sub-graphs in the RDF dataset.

GFMed [Marginean (2014)] follows a controlled natural language approach to answering biomedical questions. In particular, GFMed introduces grammars for a controlled natural language focused on biomedical information, and the corresponding controlled SPARQL language. The grammars are described in the grammatical framework, and they introduce linguistic and SPARQL sentences mainly concerning drugs, diseases, and the relationships between them. [Hamon *et al.* (2016)] propose translating natural language questions into SPARQL queries based on natural language processing tools, semantic resources, and RDF triple descriptions. [Diefenbach *et al.* (2017)] presented WDAqua as a new Question Answering (QA) component that the QA research community can easily use. It can be used to answer questions on DBpedia and Wikidata. DBpedia's language support is limited to English, whereas it can answer questions in 4 different languages on Wikidata, namely English, French, German and Italian. Moreover, it supports both entire natural language queries and keyword queries. Finally, [Zheng *et al.* (2018)] proposed a new approach to answering complex queries, namely using models to understand the input queries. Specifically, the authors decomposed the input question into a set of sub-questions via templates, where these sub-questions form a dependency graph. The model used in this work consists of two parts: the question pattern and the SPARQL query pattern.

However, to generate the SPARQL query, most existing methods have focused mainly on Natural Language Processing (NLP) tools to represent the NL query as a syntax tree. As a result, it will be challenging to produce a correct structured query from an incorrect syntax tree representation. Therefore, the Semantic Web research community needs flexible approaches that master the structure of heterogeneous and distributed data sources, identify the desire of a user query, and finally combine the query with the data sources to obtain those able to answer the query. To deal with the increasing heterogeneity of Linked Data on the Web and the alternative solutions to query them, we propose a new approach to facilitate understanding natural language queries to be translated into correct SPARQL queries. We suggest using CBR technique to identify all relevant terms constituting the NL query and their associated relations compared to the previously presented systems. The deployment of CBR solves many problems related to the interpretation of the NL queries intended meaning. In fact, our case base includes all the interpretations of treated and initial queries with different forms. When a new query is encountered, the system transforms or reuses the solutions of their similar queries stored in the case base to create a new proposed solution. The linguistic and semantic information is also exploited to identify the query triples, which are crucial to build the corresponding SPARQL queries. In particular, our proposal is based on two developed algorithms: the triplet extraction algorithm and the disambiguation algorithm to represent NL queries in a suitable form that facilitates the formalization of queries according to the proposed SPARQL generation rules.

3. The Proposed Approach

The aim is to develop a query system that allows the extraction of a relevant answer based on the semantic processing of NL queries. More precisely, the structure of our system consists of four main modules (Fig. 1), the query analyzer, the mapping module, the triples extractor, and the SPARQL queries generator.

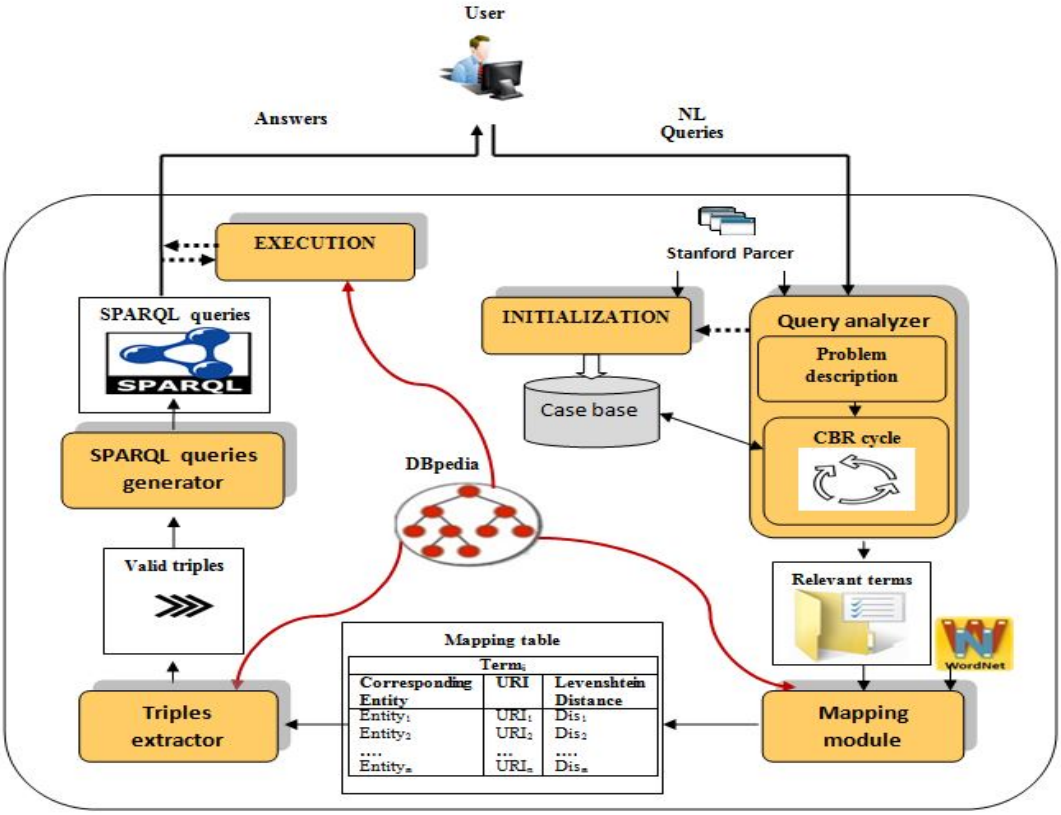

Fig. 1. System architecture.

3.1. *Query Analyzer*

3.1.1. Initialization

In this step, we use Stanford Dependencies to extract the subject and the search criteria of different types of queries based on their word orders. We note that each query has a specific sequence of words that represent its word order. This latter belongs to a general query structure. For example, the query “Which books were written

by Danielle Steel" has the word order< WDT – NNS – VBD – VBN – IN – NNP – NNP> from the basic structure <Question word + Subject + Auxiliary + Verb + Object> . Otherwise, we consider all unstructured queries as keyword queries. Fig. 2 shows our classification of different types of NL queries.

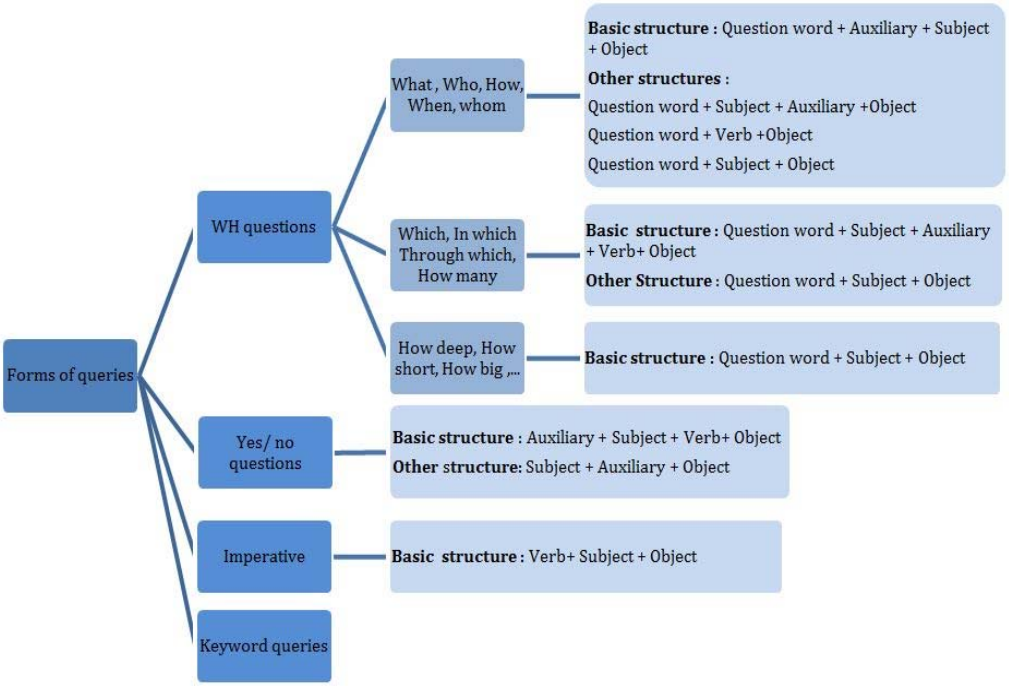

Fig. 2. Types of queries.

3.1.1.1. Identification of the subject

In particular, the subject can be a variable, a term, or some terms of the query. There are two main forms of NL queries: structured queries (NL sentences) characterized by a correct grammatical order, while keyword queries represent a random succession of keywords (words). The subject is the first name in the query in keyword queries, as it is the most important. However, in structured queries, the identification of the subject of the query is made by some proposed rules based on the Stanford Dependencies extracted from the query. For example, the query "Which books were written by Danielle Steel" has the following dependencies:

dobj (written-4 Which-1)
nsubjpass (written-4 books-2)
auxpass (written-4 were-3)
root (ROOT-0 written-4)
case (Steel-7 by-5)
compound (Steel-7 Danielle-6)
nmod:agent (written-4 Steel-7)

In this query, we use the first two dependencies to identify the subject of the query: books.

We note that each dependency is written as abbreviated-relation-name (governor, dependent) where governor and dependent are words in the sentence plus a number indicating the word's position. Table 1 lists some Stanford Dependencies:

The dependency's abbreviated name	Definition
acomp	adjectival complement
amod	adjectival modifier
cc	coordination
det	determiner
dobj	direct obj
neg	negation modifier
nsubj	nominal subject
nsubjpass	passive nominal subject
num	numeric modifier

Table 1. Example of Stanford Dependencies

3.1.1.2. Identification of search criteria

While the Stanford typed dependency representation allows users without linguistic expertise to understand the description of grammatical relations in a sentence, we use these dependencies to extract the different relations between query terms. Since the triple is a minimal representation of the information without losing the context, we represent each NL query as extracted triples in the form {subject; predicate; object} based on the algorithm developed below:

Triples-Extraction Algorithm

Input:
Query: the input query.
Begin
DEPENCIES=Stanford-dependencies (query); // each dependency has the form Dependency (head, dependent).
Ndeps : number of dependencies.
Extracted-triples: list of extracted triples
For (int i=0, i<ndeps,i++) do
// we treat $Dependency_i(head_i, dependent_i)$
Case 1: nominal subject
If ($Dependency_I$ =**"nsubj"**) then
For (int J=I+1, J<ndeps,J++) do
//we search the triple object
If(**dobj** ($heat_I$, $dependent_J$) then // $head_J$=$head_I$
Add($dependent_I$, $head_I$, $dependent_J$) to Extracted-triples;
Case 2: passive nominal subject
If ($Dependency_I$ = **"nsubjpass"**) then
// we complete the triple
For (int J=I+1, J<ndeps,J++) do
If(**nmod:agent** ($heat_I$, $dependent_J$) then // $head_J$=$head_I$
Add($dependent_I$, $head_I$, $dependent_J$) to Extracted-triples;
Else If(**nmod:by** ($dependent_I$, $dependent_J$) then // $dependent_I$ =$head_I$
Add($dependent_I$, $head_I$, $dependent_J$) to Extracted-triples;
Else If(**nmod:in** ($heat_I$, $dependent_J$) then // $head_J$=$head_I$
Add($dependent_I$, $head_I$, $dependent_J$) to Extracted-triple;
End for
Case 3 : compound noun
If ($Dependency_I$ = **"compound"**) then
Add($head_I$, - , $dependent_I$) to Extracted-triples;
Case 4: numeric modifier
If ($Dependency_I$ = **"mwe" and** $dependent_I$= **"than"**) then
If ($Dependency_{I+1}$ =**"nummod"**) then
Add($head_I$, $dependent_{I+1}$, $head_{I+1}$) to Extracted-triples;
Case 5: negation modifier
If ($Dependency_I$ = **"neg"**and$heat_I$**="not"**) then
Add("not, $heat_I$,-) to Extracted-triples;
Case 6: unknown predicate (subject, var, object)
Case 7: unknown object (subject, predicate, var)
Case 8: unknown subject (var, predicate, object)
End for
Output:
Extracted-triples

In this algorithm, we try to extract all possible relations between the query terms based on eight different cases. In particular, the cases of the nominal subject and the passive nominal subject allow us to identify the correct triple in form (subject, predicate, object). Furthermore, cases 3, 4, and 5 give specific descriptions such as the compound noun, the superlative expression, and the negation. Finally, the other cases allow us to identify triplets whose subject, predicate, or object is unknown. The previous example has the following extracted triple, in which we use the cases 2, and 3: <books, written, Danielle Steel>.

3.1.2. New Queries as New Cases

In order to improve the performances of our system and optimize the pervious analysis of NL queries, we make use of case-based reasoning technique (CBR) to identify the subject and the search criteria of query.

In CBR, a case is defined as a description of a problem along with its solution. For this, the case representation is an important aspect in designing efficient CBR systems. In this context, we define the linguistic information such the word order of input query as an abstract representation of a problem.

Furthermore, we define the subject and the search criteria of query as a solution for a given problem (Fig 3.). This representation allows exceptional solved queries to be stored as cases and used in query solving.

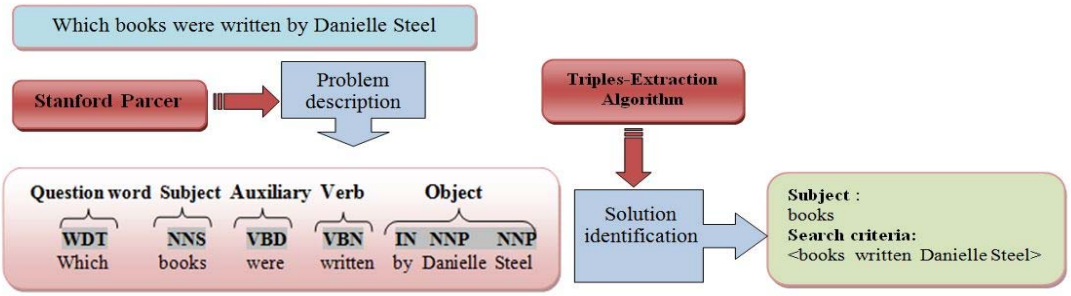

Fig. 3. Initialization of the query “Which books were written by Danielle Steel”

Specially, the system uses its memory of relevant past cases (solved queries) to interpret the new queries that allow finding similar cases and generating new solutions from them.

3.1.2.1 CBR Life Cycle

Following, we present a brief overview of the traditional problem-solving cycle in CBR applied by our system, to identify the subject and the search criteria of the query “Which countries are connected by the Rhine”. First, we identify the problem of query. For example, the pervious query has the following problem description:

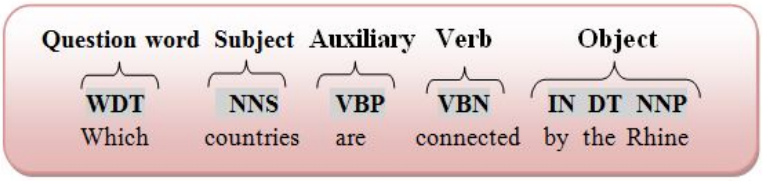

Fig. 4. The problem description.

- **Retrieve**

After identify the query type, we want to retrieve queries that have the same structure of the current problem in the case base. As shown in figure 5, cases 1, 2, and 3 are selected as similar cases of target case.

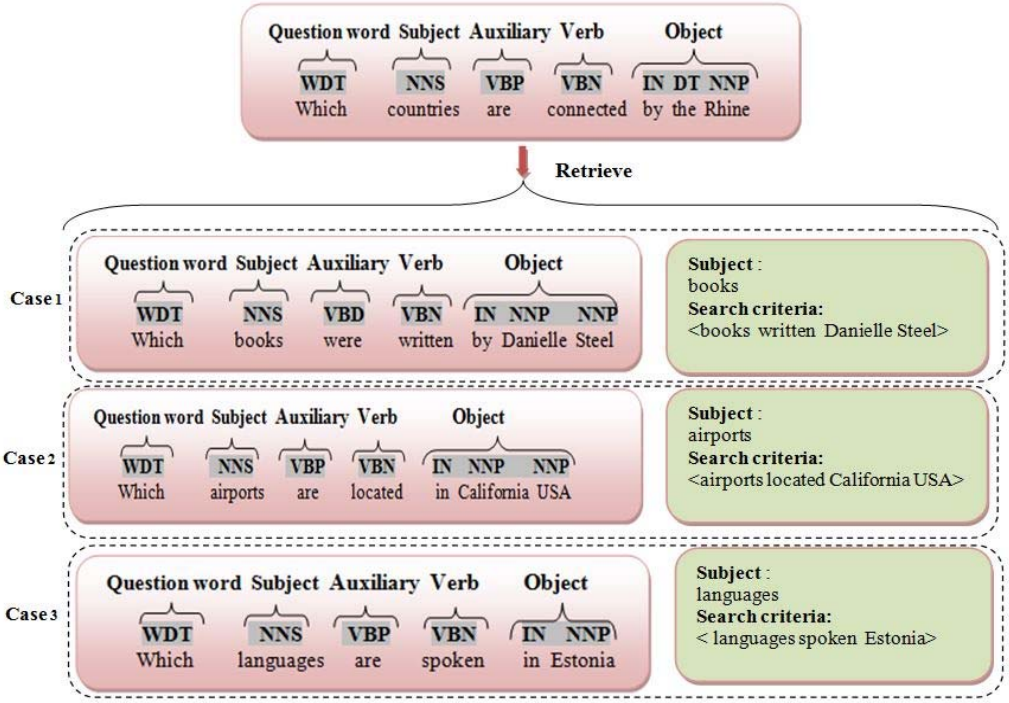

Figure 5. Similar cases

For identifying the most similar case, it is necessary to measure the similarities between the structure of the target case and those of similar cases. This is done based on the vector space model [Raghavan, and Wong (1986)]. The idea is that we represent the cases as vectors of part-of-speech (POS), and compare cases by measuring the similarity between these vectors based on the Cosine similarity [Garcia (2015)].

Case	Query	POS vector
Target case	Which countries are connected by the Rhine	WDT – NNS – VBP – VBN – IN – DT – NNP
Case1	Which books were written by Danielle Steel	WDT – NNS – VBD – VBN – IN – NNP – NNP
Case2	Which airports are located in California, USA	WDT – NNS – VBP – VBN – IN – NNP – NNP
Case3	Which languages are spoken in Estonia	WDT – NNS – VBP – VBN – IN – NNP

Table 2. Queries POS vectors

The cosine similarity between two vectors on the vector space is a measure that calculates the cosine of the angle between them. Cosine similarity will generate a metric that says how related are two vectors by looking at the angle instead of magnitude, this is the cosine similarity formula:

$$SIM_{\cos ine}(C_1, C_2) = \frac{\overrightarrow{c_1 * c_2}}{\left\| \overrightarrow{c_1} \right\| . \left\| \overrightarrow{c_2} \right\|} \qquad (1)$$

In fact, for each POS in the NL query, we calculate the term frequency (TF) to build the TF vectors as shown in Table 3:

POS	WDT	NNS	VBP	VBD	VBN	IN	DT	NNP
Target case	1/8	1/8	1/8	0	1/8	1/8	1/8	1/8
Case1	1/8	1/8	0	1/8	1/8	1/8	0	2/8
Case2	1/8	1/8	1/8	0	1/8	1/8	0	2/8
Case3	1/8	1/8	1/8	0	1/8	1/8	0	1/8

Table 3. TF vectors

Then, we calculate the Cosine similarity between the TF vector of target case and those of similar cases in order to find the most similar case:

Cosine Similarities	Measurement
SIMcosine(Target case, Case1)	0,75
SIMcosine(Target case, Case2)	0,75
SIMcosine(Target case, Case3)	0,92

Table 4. Cosine Similarities

In above example, it is observed that case 3 is the most similar case to target case.

- **Reuse**

Because new solutions rarely match old ones exactly, the solution obtained from the case base must be adapted for resolving the target case. This adaptation processes done based on proposed adaptation rules that focus more on the similarities in the basic structure of queries (Fig. 6).

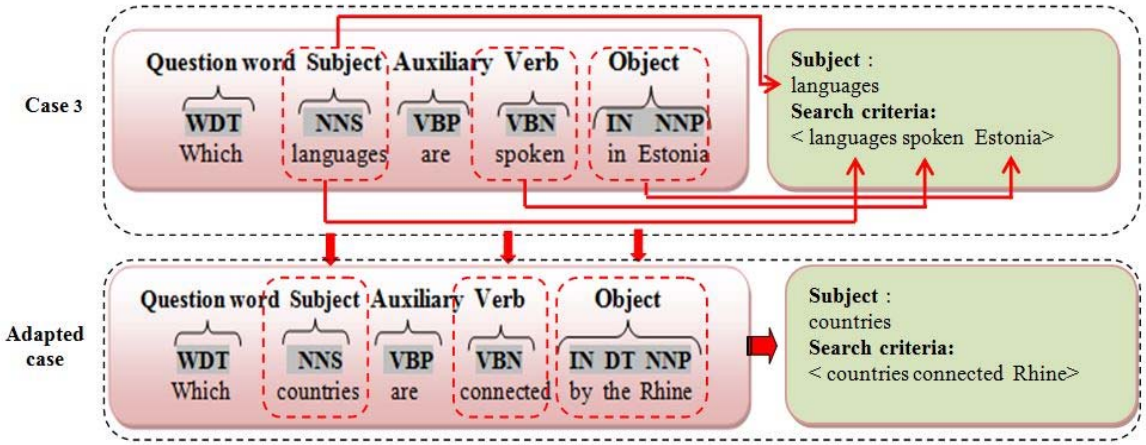

Figure 6.Adapted solution

We take into account the differences in problem description of queries such as the structure of the subject and the object. For instance, the subject may be a variable, simple, or composed nouns. Moreover, the object may be direct, indirect, verbal, or object of prepositions. In this context, the solution is adapted to better fit the user's query.

- **Revise**

Then, the solution proposed by the system is evaluated by being applied to the initial problem or assessed by a domain expert. If the solution is considered to be correct, the system produces it as new case. Otherwise, the CBR system turns back to the reuse or the initialization phase and searches for other solutions.

- **Retain**.

Finally, the problem description and its solution can be retained as a new case, and the system learns it to solve new problems. Thus, learning new cases allows the enrichment of the case base over the time. We note that each query stored in the case base must has a word order different to the others.

3.2. *Mapping module*

This module allows finding the entities of the DBpedia ontology that correspond to the terms of the query. In particular, each term in the search criteria will be associated with its corresponding entities in DBpedia with their associated similarities. First, for each query term, we execute the SPARQL query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbr: http://dbpedia.org/resource/>
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#/>
SELECT ?URI where { ?URI <http://www.w3.org/2000/01/rdf-schema#label> "term"@en. }
```

This query checks whether this term exists in DBpedia. If not, we propose an algorithm to identify its corresponding entities in DBpedia. In the following, we describe the pseudo-code of the disambiguation algorithm to query the DBpedia ontology:

Disambiguation Algorithm

Input:
n: number of query terms;
T= {T_1, T_2,..,T_n}: list of query terms;
WN: WordNet;
MAPPING (AT, WN)
i=1;
Boolean Exist=false;
While (i<=n) do
MT_i: mapping table of the term T_i;
DBpediaLookupClient (T_i)
OE= {OE_1, OE_2,…,OE_m};// list of corresponding entities of T_i in DBpedia
If OE≠φthen
Exist=true;
For each OE_j∈OE **do** //j∈{1,2,..,m}
LD_{ij}=LD(T_i,OE_j); Levenshtein Distance between the initial term and its corresponding in DBpedia
Add (OE_j, LD_{ij}) to MT_i ;
End for
Classification of MT_i;
if (LD(T_i,OE_1)> 5then // OE_1 is the first nearest corresponding to T_i in DBpedia
MT_i=φ
Else if (OE=φ || MT_i=φ)
// we must look for synonyms of Ti in WordNet
WS_i: list of synonyms of T_i in WordNet;
For (each ws_k∈WS_i) **do** // k∈{1,2,..,l}
DBpediaLookupClient (ws_k)
OE_k= {OE_{k1}, OE_{k2},…,OE_{km}};// list of corresponding entities of ws_k in DBpedia
If OE_k≠φ
Exist=true;
For (each OE_{kj}∈OE_k) **do** // j∈{1,2,..,m}
LD_{kj}= LD(T_i,OE_{kj});
Add (OE_{kj}, LD_{kj}) to MT_i;
End for
Classification of MT_i;
End if
End for
End if
If (not Exist || LD(T_i,OE_1)>5) then // OE_1 is the first nearest corresponding to T_i in DBpedia**) then**
/*use the triple extractor*/
/*Dialogue with user*/
/*Replace the ambiguous term T_i*/
Else
If (Exist) then
i=i+1;
Exist= false;
End if
End if
End while
Output:
MT= {MT_1, MT_2,…,MT_n}: mapping table of ambiguous terms

In this algorithm, the mapping of NL terms to DBpedia entities is done as follows: for each query term, we use the DBpediaLookupClient[b] class to identify its corresponding entities in DBpedia. Then we rank these

[b] https://github.com/mark-watson/java_practical_semantic_web/blob/master/src/com/knowledgebooks/info_spiders/DBpediaLookupClient.java

matching entities by the Levenshtein distances between the query term and these entities. If the first closest matching entity of a given query in DBpedia has a Levenshtein distance greater than 5 or if this term has no matching entities in DBpedia, we have to search for its synonyms in WordNet. If this term or at least one of its synonyms has matching entities in DBpedia, the list of matching entities consists of the mapped entities ranked with the Levenshtein distances. Otherwise, if a query term or its synonyms in WordNet do not have matching entities in DBpedia, we use the triple extractor to find a missing component (property, or argument) in a query triple. Finally, if the triples extractor failed to identify that component, we establish a dialogue with the user to get a clarification. For example, in the previous query "Which books were written by Danielle Steel", we obtained the following list of matching terms:

Query terms	Corresponding in DBpedia	Uri	Levenshtein Distance
books	Book	http://dbpedia.org/ontology/Book	2
written	written	http://dbpedia.org/ property/ written	0
Danielle Steel	Danielle_Steel	http://dbpedia.org/resource/Danielle_Steel	1

Table 5. Mapping of NL terms to DBpedia entities

Because we use the prefixes:
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbr: http://dbpedia.org/resource/>
We obtain the query triple: <dbo: book, dbp: written, res: Danielle_Steel>
Therefore, we used the following module to identify possible relationships between the concepts dbo: book,dbp: written, and res: Danielle_Steel in DBpedia.

3.3. *Triples Extractor*

Each triple of the query has two arguments; the first is called the domain, and the second is called range. In particular, an argument can be a class, an instance, or a literal. In addition, the relation can be an object_property or a data_type_property. In this context, it would be more useful to construct correct triples based on the relationships found between the NL query terms in DBpedia. First, we look for the relationships between each pair of arguments in the triple <Argument1,unknown, Argument2>. In this case, we complete the triple with the found relationship. Therefore, we execute the following SPARQL query to find the missing property between two arguments Argument1, and Argument2:

```
Select distinct ?prop where {
?uri rdf:type prefix: Argument1.
?uri ?prop prefix: Argument2 .
}
```

For example, in the previous query, we execute the SPARQL query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select distinct ?prop where {
?uri rdf:type dbo:Book .
?uri ?prop res:Danielle_Steel.
}
```

In this case, we confirm that the two arguments have a relationship and complete the triple with it (prop).e.g.<dbo: Book, dbo:author, res: Danielle_Steel>. Otherwise, we essentially examined each triple of the form argument-relation <argument, relation, unknown> to check whether the argument is the domain or the range of the relation and enrich the query with the missing argument, which can be a class, instance, or literal.

In particular, we run the following SPARQL queries to find the missing argument:

```
Select distinct ?Arg2 where {
?uri rdf:type prefix: Argument1.
?uri prefix :prop ?Arg2.
}
```

And

```
Select distinct ?Arg1 where {
?uri rdf:type prefix: Argument2 .
?Arg1 prefix :prop ?uri.
}
```

If the first query has a result, we confirm that the argument is the domain of the relation and complete the triple with it (Arg2). Otherwise, we execute the second query. If it has a result, we confirm that the argument is the range of the relation, and we complete the triple with it (Arg1).

The result of this step is queries that contain all valid triples to generate correct SPARQL queries.

3.4. *SPARQL Queries Generator*

Structured queries are essentially built from two components: the syntax of the query language and the elements (entities and relationships) of the data model behind the dataset. Briefly, the SPARQL query has the general format **< prefix declarations> SELECT <query-objects> WHERE <query-body><Query modifiers>.** The prefix declarations represent the abbreviations of the URIs. In addition, the query objects represent the query head which may contain variables prefixed with either "?" or "$." It identifies the variables that will appear in the query results. The body of the query consists of a series of triplet patterns, expressed using the Turtle syntax. These triples together form what is known as a graph pattern. Finally, the query modifiers represent the restrictions of the query.

In the following, we give an overview of the proposed rules for building correct SPARQL queries:

(1) If the query object is a class in ontology, we interpret it in the head as a variable prefixed with "?" .g. (?className). Also, we interpret it in the query body as a triple pattern <?className rdf:type ClassName> after the "WHERE" clause. For example, the following query returns all instances of the class Book:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT  ?book WHERE {
?book rdf:type dbo:Book .
}
```

(2) If the query object is an entity of type RDF Literal, we interpret it directly as a variable, e.g."?L"after SELECT. Similarly, we interpret it in the query body as a triple pattern <?x, DataProperty, ?L> after the "WHERE" clause.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?title WHERE {
?book rdf:type dbo:Book .
?book rdfs:label ?title }
Order By ?title
Limit 20
```

The comparative change is also interpreted with the Order By and Limit instructions.

(3) Each matching triple is directly interpreted as a triple pattern after "WHERE"in the query body. We note that instances and properties are directly interpreted as their URIs in the DBpedia. In contrast, we interpret classes or RDF Literal Type entities as variables according to the rules above.

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?film WHERE {
?film rdf:type dbo:Film  .
?film dbo:starring res:Tom_Cruise
} Limit 20
```

(4) SPARQL also supports ASK queries which intuitively correspond to a Yes/No question in conversational language. An ASK query is a check to see if any resources in the dataset match the search pattern; the

answer is either true or false. For example, the following query corresponds to a Yes/No question. If this query is submitted as described above, the answer given will be "true."

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
ASK
{ res:Amazon_River dbp:length ?amazon .
res:Nile dbp:length ?nile .
FILTER(?amazon > ?nile) .
}
```

Constraint queries that satisfy a specific condition can be interpreted with the SPARQL language. Specifically, SPARQL FILTER functions can test the values of RDF literal strings. SPARQL FILTERS can also restrict numeric values by arithmetic expressions.

(5) If the query contains multiple triples, all triples in the query are linked by the conjunctive "and"by default. SPARQL also supports compound queries using the "or" operator with UNION. SPARQL contains a UNION pattern; this is formed by placing the UNION keyword between two subsets of statements, each subset delimited by square brackets. The meaning is that variable bindings should be retrieved if they satisfy either the left or right-hand statements (or both).

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?personne
WHERE {
{?personne dbp:birthplace res:Paris}
UNION {
?personne dbp:deathPlace res:Paris
} }
```

(6) SPARQL also supports negation by providing operator "!" and instruction " NOT EXISTS ".

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?personne
WHERE {
?personne dbp:birthplace res:Paris.
FILTER NOT EXISTS {?personne dbp:deathPlace res:Paris} }
```

For example, the SPARQL queries generator associates the subject and triples of the previous query "Which books were written by Danielle Steel"to build a correct SPARQL query based on the proposed generation rules as shown below:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT ?uri WHERE { ?uri rdf:type dbo:Book.
?uri dbo:author dbr:Danielle_Steel }
```

Finally, the execution of SPARQL queries is handled by the org.apache.jena.query package[c] which relies on the Query class that contains all the details of the SPARQL query.

[c] https://jena.apache.org/documentation/query/app_api.html

4. Experimental Results

In this section, we evaluate the performance of our system on QALD-8 [Usbeck *et al.* (2018a)] and QALD-9 [Usbeck et *al.* (2018b)]. The QALD (Question Answering over Linked Data) challenge is aimed at all researchers and practitioners working on Linked Data querying, natural language processing for question answering, multilingual information retrieval and related topics. In particular, the QALD challenge aims at providing a set of multilingual natural language questions, corresponding SPARQL queries and answers.

The initial version of our case base consists of 40 cases of queries selected from QALD-9. In this experiment, we evaluate the effectiveness of our approach. For each system response, we check whether the answer is correct or not by comparing the output with the generated SPARQL query of QALD-8 and QALD-9.

Experiments showed that the proposed approach is efficient compared to the state-of-the-art methods in terms of precision and recall. Table 6 shows the comparison of our approach with some previous systems.

	Macro Recall		**Macro Precision**		**Macro F-measure**	
QALD	QALD-8	QALD-9	QALD-8	QALD-9	QALD-8	QALD-9
wdaqua-core 0/1	0,4065	0.267	0,3912	0.261	0,3872	0.250
gAnswer	0,3902	0.327	0,3862	0.293	0,388	0.298
The proposed system	0 ,4255	0,2836	0,4545	0,2433	0 ,4333	0,220

Table 6. Evaluation results of the test dataset of QALD-8 and QALD-9.

However, some NL queries such as “Which US states are in the same time zone as Utah?” and “What was the name of the famous battle in 1836 in San Antonio?” do not have correct corresponding SPARQL queries because we cannot link specific terms to the corresponding entities. Therefore, correct mappings could only be placed if we were more familiar with the knowledge structure inherent in DBpedia. In addition, we have some difficulties in expressing these NL queries in SPARQL syntax. This implies that the ambiguity of the NL queries and the formal language syntax reduce the quality of the answers, which may produce irrelevant answers.

5. Conclusion

The need to access data contained in Linked Data through NL queries is obvious in the Semantic Web. However, this creates significant problems such as the ambiguity and complexity of processing NL queries and translating them into correct formal queries to provide the right answers. This paper aims to present the developed system that provides a relevant answer to NL queries posed by users. In particular, CBR technique and a triplet extraction algorithm based on Stanford dependencies are used to identify the subject and search criteria of the NL query. Thus, we combined the developed disambiguation algorithm, the WordNet dictionary, and the user dialogues for query disambiguation. We also enriched the queries with new relations and concepts from the dataset to build correct triples. Finally, we generate formal queries from these triples to interrogate the dataset and return the relevant answer.

As the number of large datasets available on the SW has increased and many efforts are underway to research and build applications that exploit the Web of Data, we plan to enhance our system to query other LOD (Linked Open Data) resources and handle more complex queries.

References

[1] Bizer, C., Heath, T., Berners-Lee, T. (2009): “Linked Data – The Story So Far”. Journal of Semantic Web and Information Systems, Vol. 5 No. 3, pp.1-22.

[2] De Marneffe M,C., Christopher D., Manning, (2008) :”The Stanford typed dependencies representation”, In Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation, Manchester, Kingdom — United, pp. 1-8.

[3] Diefenbach D., Deep Singh K., Maret P., (2017): “Wdaqua-core0: A question answering component for the research community”, In SemWebEval 2017, Portoroz, Slovenia, pp. 84-89.

[4] Damljanovic, D., Agatonovic, M., Cunningham, H., (2011): “FREyA: An Interactive Way of Querying Linked Data Using Natural Language”, In Proceedings of the 1st Workshop on Question Answering over Linked Data (ESWC 2011). Heraklion, Greece, pp 125-138.

[5] De Mántaras R. L.,Mcsherry D., Bridge D., Leake D., , Smyth B., Craw S., Faltings B., Maher M. L. , Cox M. T. , Forbus K., Keane M., Aamodt A., &Watson I., (2005): “Retrieval, reuse, revision, and retention in case based reasoning”, Journal of The Knowledge Engineering Review, Vol. 20 No. 3, pp.215-240.

[6] Freitas, A., and Curry, E., (2014): “Natural Language Queries over Heterogeneous Linked Data Graphs: A Distributional-Compositional Semantics Approach”, In Proceedings of the 19th international conference on Intelligent User Interfaces (IUI ‘14), New York, NY, USA, pp. 279–288.

[7] Garcia E., (2015): “A Tutorial on Distance and Similarity”, available at http://www.minerazzi.com/tutorials/distance-similarity-tutorial.pdf (accessed 13 February 2015).

[8] Hamon T., Grabar N., Fleur Mougin F., (2016): “Querying Biomedical Linked Data with Natural Language Questions”, Journal of Semantic Web, Vol. 8 No. 4, pp. 581-599.

[9] Marginean A., (2014): "GFMed: Question answering over biomedical linked data with grammatical framework", In Proceedings of the 1st International Workshop on Natural Language Interfaces for Web of Data (NLI-WoD 2014) co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, pp.1224-1235
[10] Miller G. A., Beckwith R., Fellbaum C., Gross D., Miller K., (1990): "Introduction to WordNet: an on-line lexical database", International Journal of Lexicography, Vol. 3 No. 4, pp.235- 244.
[11] Prud'hommeaux, E.,and Seaborne, A., (2008): "SPARQL Query Language for RDF", available at http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ (Accessed 26 March 2013).
[12] Shizhu H., Yuanzhe Z., Liu K., Zhao J., (2014): "CASIA@V2: A MLN-based question answering system over linked data", In Working Notes for {CLEF} 2014 Conference (CLEF 2014). Sheffield, UK, pp. 1249-1259.
[13] Raghavan V. V., Wong S. K. M., (1986): "A critical analysis of vector space model for information retrieval", Journal of the American Society for Information Science, Vol. 37 No. 5, pp.279-87.
[14] Usbeck R., Gusmita R. H., Ngomo A. N., Saleem M., (2018): "9th Challenge on Question Answering over Linked Data (QALD-9)" (invited paper), in Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, pp. 58–64.
[15] Usbeck R., Ngomo A. N., Ricardo Usbeck1, ConradsF., Roder1 M., Napolitano G., (2018):"8th Challenge on Question Answering overLinked Data (QALD-8)", In Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, pp. 51-57
[16] Xu K., Feng Y., Zhao D., (2014): "Xser@QALD-4: Answering natural language questions via phrasal semantic parsing", In CLEF2014 Working Notes Papers, pp. 15-18.
[17] Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V. &Weikum, G., (2012): "Natural Language Questions for the Web of Data", In Proceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP CoNLL 2012). Jeju Island, Korea, pp. 379–390
[18] Zheng W., Xu Yu J., Zou L., Cheng H., (2018): "Question Answering Over Knowledge Graphs: Question Understanding Via Template Decomposition". Proceedings of the VLDB Endowment, Vol. 11, No. 11, pp.1373-1386.
[19] Zou L., Huang R., Wang H., Xu Yu J., He W., Zhao D., (2014): "Natural langauge question answering over RDF a graph data driven approach 2014", In Proceedings of SIGMOD '14:International Conference on Management of Data. New York, United States, pp. 313-324.

Authors Profile

Hasna boumechaal, is an teaching assistant at Higher Normal School of Constantine, Constantine, Algeria, and she is also an attached member of the SI & BC research group at Lire Laboratory at University of Abdelhamid Mehri-Constantine 2, Constantine, Algeria. She is currently a PhD student and her research interests include Natural Language interfaces, Semantic Web; and Linked Data.

Zizette Boufaida, is a professor of Computer Science and is currently dean at University of Abdelhamid Mehri Constantine 2, Ali Mendjeli, Constantine, Algeria.

Her research areas include knowledge representation and reasoning, formal knowledge representation for Semantic Web and ontology development.