Query Rewriting for an Incremental Search in Heterogeneous Linked Data Sources

Ana I. Torre-Bastida[1], Jesús Bermúdez[2], Arantza Illarramendi[2], Eduardo Mena[3], and Marta González[1]

[1] Tecnalia Research & Innovation
{isabel.torre,marta.gonzalez}@tecnalia.com
[2] Departamento de Lenguajes y Sistemas Informáticos, UPV-EHU
{a.illarramendi,jesus.bermudez}@ehu.es
[3] Departamento de Informática e Ingeniería de Sistemas, Univ. Zaragoza
emena@unizar.es

Abstract. Nowadays, the number of linked data sources available on the Web is considerable. In this scenario, users are interested in frameworks that help them to query those heterogeneous data sources in a friendly way, so avoiding awareness of the technical details related to the heterogeneity and variety of data sources. With this aim, we present a system that implements an innovative query approach that obtains results to user queries in an incremental way. It sequentially accesses different datasets, expressed with possibly different vocabularies. Our approach enriches previous answers each time a different dataset is accessed. Mapping axioms between datasets are used for rewriting the original query and so obtaining new queries expressed with terms in the vocabularies of the target dataset. These rewritten queries may be semantically equivalent or they could result in a certain semantic loss; in this case, an estimation of the loss of information incurred is presented.

Keywords: Semantic Web, Linked Open Data Sources, SPARQL query, vocabulary mapping, query rewriting.

1 Introduction

In recent years an increasing number of RDF open data sources are emerging, partly due to the existence of techniques to convert non RDF datasources into RDF ones, supported by initiatives like the Linking Open Data (LOD)[1] with the aim of creating a "Web of Data". The Linked Open Data cloud diagram[2] shows datasets that have been published in Linked Data Format (around 338 datasets by 2013[3]), and this diagram is continuosly growing. Moreover, although those datasets follow the same representation format, they can deal with heterogeneous

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[2] http://lod-cloud.net/state/
[3] http://datahub.io/lv/group/lodcloud?tags% 3Dno-vocab-mappings

H.L. Larsen et al. (Eds.): FQAS 2013, LNAI 8132, pp. 13–24, 2013.

vocabularies to name the resources. In that scenario, users find difficulties in taking advantage of the contents of many of those datasets because they get lost with the quantity and the variety of them. For example, a user that is only familiar with BNE (Biblioteca Nacional de España)[4] dataset vocabularies could be interested in accessing BNB (Bristh National Bibliography)[5], DBpedia[6], or VEROIA (Public Library of Veroia)[7] datasets in order to find more information. However, not being familiar with the vocabularies of those datasets may dissuade him from trying to query them.

So, taking into account the significant volume of Linked Data being published on the Web, numerous research efforts have been oriented to find new ways to exploit this Web of Data. Those efforts can be broadly classified into three main categories: Linked Data browsers, Linked Data search engines, and domain-specific Linked Data appplications [1]. The proposal presented in this paper can be considered under the category of Linked Data search engines and more particularly, under human-oriented Linked Data search engines, where we can find other approaches such as Falcons[8] and SWSE[9], amongst others. Nevertheless, the main difference of our proposal with respect to existing engines lies in the fact that it provides the possibility of obtaining a broader response to a query formulated by a user by combining the following two aspects: (1) an automatic navigation through different datasets, one by one, using mappings defined among datasets; and (2) a controlled rewriting (generalization/specialization) of the query formulated by the user according to the vocabularies managed by the target dataset.

In summary, the novel contribution of this paper is the development of a system that provides the following main advantages:

- *A greater number of datasets at the users disposal.* Using our system the user can gain access to more datasets without bothering to know the existence of those datasets or the heterogeneous vocabularies that they manage. The system manages the navigation into different datasets.
- *Incremental answer enrichment.* By accessing different datasets the user can obtain more information of interest. For that, the system manages existing mapping axioms between datasets.
- *Exact or approximate answers.* If the system is not capable of obtaining a semantically equivalent rewriting for the query formulated by the user it will try to obtain a related query by generalizing/specializing that query and it will provide information about the loss in precision and/or recall with respect to the original query.

In the rest of the paper we present first some related works in section 2. Then, we introduce an overview of the query processing approach in section 3.

[4] BNE - (`http://datos.bne.es/sparql`)
[5] BNB - (`http://bnb.data.bl.uk/sparql`)
[6] DBpedia - (`http://wiki.dbpedia.org/Datasets`)
[7] VEROIA - (`http://libver.math.auth.gr/sparql`)
[8] `http://ws.nju.edu.cn/falcons/objectsearch/index.jsp`
[9] `http://swse.deri.org/`

We follow with a detailed explanation of the query rewriting algorithm and with a brief presentation of how the information loss is measured in sections 4 and 5. Finally we end with some conclusions in section 6.

2 Related Works

According to the growth of the Semantic Web, SPARQL query processing over heterogeneous data sources is an active research field. Some systems (such as DARQ [10], FedX [12]) consider federated approaches over distributed data sources with the ultimate goal of virtual integration. One main difference with our proposal is that they focus on top-down strategies where the relevant sources are known while in our proposal the sources are discovered during the query processing. Nevertheless one main drawback for query processing over heterogeneous data is that existing mapping axioms are scarce and very simples (most of them are of the owl:sameAs type)

Even closer to our approach are the works related to SPARQL query rewriting. Some of them, such as [7] and [2], support the query rewriting with mapping axioms described by logic rules that are applied to the triple patterns that compose the query; [7] uses a quite expressive specific mapping language based on Description Logics and [2] uses less expressive Horn clause-like rules. In both cases, the mapping language is much more expressive than what is usually found in datasets metadata (for instance, VoID[10] linksets) and the approach does not seem to scale up well due to the hard work needed to define that kind of mapping.

Another approach to query rewriting is query relaxation, which consists of reformulating the triple patterns of a query to retrieve more results without excessive loss in precision. Examples of that approach are [6] and [3]. Each work presents a different methodology for defining some types of relaxation: [6] uses vocabulary inference on triple patterns and [3] uses a statistical language modeling technique that allows them to compute the similarity between two entities. Both of them define a ranking model for the presentation of the query results. Although our proposal shares with them the goal of providing more results to the user, they are focused more on generalizing the query while we are focused on rewriting the query trying to preserve the meaning of the original query as much as possible, and so generalizing or specializing parts of the query when necessary in the new context.

The query rewriting problem is also considered [5], but we differ in the way to face it. In our proposal we cope with existing mapping axioms, that relate different vocabularies, and we make the most of them in the query rewriting process. In contrast, [5] disregards such mapping axioms and looks for results in the target dataset by evaluating similarity with an Entity Relevance Model (ERM) calculated with the results of the original query. The calculation of the ERM is based on the number of word occurrences in the results obtained, which are later used as keywords for evaluating similarity. The strength of this method turns into its weakness in some scenarios because there are datasets that make

[10] `http://vocab.deri.ie/void`

abundant use of codes to identify their entities and those strings do not help as keywords.

Finally, query rewriting has also been extensively considered in the area of ontology matching [4]. A distinguising aspect of our system is the measurement of information loss. In order to compute it we adapt the approach presented in [11] and further elaborated in [9] to estimate the information loss when a term is substituted by an expression.

3 An Overview of the Query Processing Approach

In this section we present first some terminology that will be used throughout the rest of the paper. Then we show the main steps followed by the system to provide an answer to one query formulated by the user. Finally, we present a brief specification of the main components of the system that are involved in the process of providing the answer.

With respect to the terminology used, we consider datasets that are modeled as RDF graphs. An RDF *graph* is a set of RDF triples. An RDF *triple* is a statement formed by a *subject*, a *predicate* and an *object* [8]. Elements in a triple are represented by IRIs and objects may also be represented by literals. We use *term* for any element in a triple. Each dataset is described with terms from a declared vocabulary set. Let us use *target* dataset for the dataset over which the query is going to be evaluated, and we use *target* vocabulary set for its declared vocabulary set.

SPARQL queries[11] are made of graph patterns. A *graph pattern* is a query expression made of a set of triple patterns. A *triple pattern* is a triple where any of its elements may be a variable. When a triple pattern of a query is expressed with terms of the target vocabulary set we say that the triple pattern is *adequate* for the target dataset. When every triple pattern of a query is *adequate* for a target dataset, we say that the query is *adequate* for that target dataset.

The *original* query is expressed with terms from a *source* vocabulary set. Let us call $\mathcal{T}$ the set of terms used by the original query. As long as any term in $\mathcal{T}$ belongs to a vocabulary in the target vocabulary set, the original query is adequate for the target dataset and the query can be properly processed over that dataset. However, if there were terms in $\mathcal{T}$ not appearing in the target vocabulary set, triple patterns of the original query including any such term should be rewritten into appropriate graph patterns, with terms taken from the target vocabulary set, in order to become an adequate query for the target dataset.

Terms in $\mathcal{T}$ appearing in synonymy mapping axioms (i.e. expressed with any of the properties `owl:sameAs`, `owl:equivalentClass`, `owl:equivalentProperty`) with a term in the target vocabulary set can be directly replaced by the synonym term. Those terms in $\mathcal{T}$ not appearing in the target vocabulary set and not appearing in synonymy mapping axioms with terms in the target vocabulary set are called *conflicting* terms. Since there is no guarantee for enough

[11] `http://www.w3.org/TR/rdf-sparql-query/`

synonym mapping axioms between source and target vocabulary sets that allow a semantic preserving rewriting of the original query into an adequate query for the target vocabulary, we must cope with query rewritings with some loss of information. The goal of the query rewriting algorithm is to replace every triple pattern including conflicting terms with a graph pattern adequate for the target dataset.

3.1 Main Query Processing Steps

The query which we will use as a running example is "Give me resources whose author is Tim Berners-Lee". The steps followed to answer that query are presented next:

1. The user formulates the query dealing with a provided GUI. For that, he uses terms that belong to a vocabulary that he is familiar with (for example, DBLP and FOAF vocabularies in this case). Notice that it is not required that the user knows the SPARQL language for RDF, he should only know the terms `dblp:Tim_Berners-Lee`, and `foaf:maker` from the DBLP and FOAF vocabularies. The system produces the following query:

   ```
   PREFIX foaf: <http://xmlns.com/foaf/0.1/>
   PREFIX dblp: <http://dblp.l3s.de/d2r/resource/authors/>
   {?resource foaf:maker dblp:Tim_Berners-Lee>}
   ```

2. The system asks the user for a name of a dataset in which he is interested in finding the answer. If the user does not provide any specific name, then the system shows the user different possible datasets that belong to the same domain (e.g., bibliographic domain). If the user does not select any of them then the system selects one. Following the previous example, we assume that the user selects DBpedia dataset among those presented by the system.
3. The system first tries to find the query terms in the selected dataset. If it finds them, it runs the query processing. Otherwise the system tries to rewrite the query formulated by the user into another equivalent query using mapping axioms. At this point two different situations may happen:
   (a) The system finds synonymy mapping axioms, defined between the source and target vocabularies, that allows it to rewrite each term of the query into an equivalent term in the target vocabulary (for instance, mapping axioms of the type `dblp:Tim_Berners-Lee owl:sameAs dbpedia: Tim_Berners-Lee`). Following the previous example, the property `foaf:maker` is replaced with `dbpedia-owl:author`. The rewritten query is the following:

   ```
   PREFIX dbpedia: <http://dbpedia.org/resource/>
   PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
   {?resource dbpedia-owl:author dbpedia:Tim_Berners-Lee>}
   ```

   Then the system obtains the answer querying the DBpedia dataset and shows the answer to the user through the GUI. The results obtained by the considered query are:

`http://dbpedia.org/resource/Tabulator`
`http://dbpedia.org/resource/Weaving_the_Web:_The_Original_Design_and_Ultimate_Destiny_of_the_World_Wide_Web_by_its_inventor`

(b) The system does not find synonymy mapping axioms for every term in the original query. In this case, the triple including the conflicting term is replaced with a graph pattern until an adequate query is obtained. In sections 4 and 5 we present the algorithm used for the rewriting and an example that illustrates the behaviour, respectively.

4. The system asks the user if he is interested in querying another dataset. If the answer is *No* the process ends. If the answer is *Yes* the process returns to step 2.

3.2 System Modules

In order to accomplish the steps presented in the previous subsection the system handles the following modules:

- *Input/Output Module*. This module manages a GUI that facilitates, on the one hand, the task of querying the datasets using some predefined forms; and, on the other hand, presents the obtained answer with a friendly appearance.
- *Rewriting Module*. This module is in charge of two main tasks: *Query analysis* and *Query rewriting*. The *Query analysis* consists of parsing the query formulated by the user and obtaining a tree model. For this task, the *Query Analyzer* module implemented with ARQ[12] is used. In this task the datasets that belong to the domain considered in the query are also selected. Concerning *Query rewriting*, we have developed an algorithm (explained in section 4) that rewrites the query expressed using a source vocabulary into an adequate query. The algorithm makes use of mapping axioms expressed as RDF triples and which can be obtained through SPARQL endpoints or RDF dumps.
  The mapping axioms we are considering in this paper are those triples whose subject and object are from different vocabularies and the predicate is one of the following terms: `owl:sameAs`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `owl:equivalentClass`, and `owl:equivalentProperty`. Future work will consider a broader set of properties for the mapping axioms.
- *Evaluation Module*. Taking into account that different rewritings could be possible for a query, the goal of this module is to evaluate those different rewritings and to select the one that incurs the least information loss. For that it handles some defined metrics (see section 5.1) and the information stored in the VoID statistics of the considered datasets.
- *Processing Module*. Once the best query rewriting is selected, this module is in charge of obtaining the answer for the query by accessing the corresponding dataset.

[12] Apache Jena/ARQ (`http://jena.apache.org/documentation/query/index.html`)

4 Query Rewriting Algorithm

In this section we present the query rewriting algorithm. Its foundation is a graph traversing algorithm looking for the nearest terms (that belong to the target dataset) of a conflicting term.

We follow two guiding principles for the replacement of conflicting terms: (1) a term can be replaced with the conjunction of its directly subsuming terms. (2) a term can be replaced with the disjunction of its directly subsumee terms. These guiding principles are recursively followed until adequate expressions are accomplished.

A distinguishing feature of our working scenario is that source and target vocabulary sets are not necessarily fully integrated. Notice that datasets are totally independent from one another and our system is only allowed to access them by their particular web services (SPARQL endpoint or programmatic interface). Therefore, our system depends only on the declared vocabulary sets and the published mapping axioms. Infered relationships between terms are not taken into account unless the target system provides them. We are aware of the limitations of that consideration, but we think that it is quite a realistic scenario nowadays.

In the following, we present the algorithm that obtains an adequate query expression for the target dataset with the minimum loss of information with respect to the original query Q measured by our proposed metrics.

First of all, the original query Q is decomposed into triple patterns which in turn are decomposed into the collection of terms $\mathcal{T}$. This step is represented in line 4 in the displayed listing of the algorithm. Notice that variables are not included in $\mathcal{T}$. Variables are maintained unchanged in the rewritten query. Neither literal values are included in $\mathcal{T}$. Literal values are processed by domain specific transformer functions that take into account structure, units and measurement systems.

Then, for each term in $\mathcal{T}$, a collection of expressions is constructed and gathered with the term. Each expression represents a possible substitution of the triple pattern including the conflicting term for a graph pattern adequate for the target dataset. See lines 5 to 10 in the algorithm. Considering these expressions associated with each term, the set of all possible adequate queries is constructed (line 12) and the information loss of each query is measured and the query with the least loss is selected (line 14).

The core of the algorithm is the REWRITE routine (line 7) which examines source and target vocabularies, with their respective mapping axioms, in order to discover possible substitutions for a given term in a source vocabulary. Let us consider terms in a vocabulary as nodes in a graph and relationships between terms (specifically `rdfs:subClassOf`, `rdf:subPropertyOf`, `owl:equivalentClass`, `owl:equivalentProperty`, and `owl:sameAs`) as directed labeled edges between nodes. Notice that, due to mapping axioms between two vocabularies, we can consider those vocabularies as parts of the same graph. REWRITE routine performs a variation of a Breadth First Search traverse from a conflicting term, looking for its *frontier* of terms that belong to a target vocabulary. A term f belongs to the *frontier* of a term t if it satisfies the

following three conditions: (a) f belongs to a target vocabulary, (b) there is a *trail* from t to f, and (c) there is not another term g (different from f) belonging to a target vocabulary in that trail. A *trail* from a node t to a node f is a sequence of edges that connects t and f independent of the direction of the edges. For instance, t `rdfs:subClassOf` r, f `rdfs:subClassOf` r is a trail from t to f.

Although a trail admits the traversing of edges in whatever direction, our algorithm keeps track of the pair formed by each node in the trail and the direction of the edge followed during the traverse since that is crucial information for producing the adequate expressions for substitution. Notice that we are interested in obtaining a conjunction expression with the directly subsuming terms, and a disjunction expression with the directly subsumee terms. For that reason, different routines are used to traverse the graph. In line 28 of the algorithm, directSuper(t) is the routine in charge of traversing the edges leaving t. In line 30 of the algorithm, directSub(t) is the routine in charge of traversing the edges entering t. Whenever a synonym to a term in a target vocabulary is found (line 25), such information is added to a queue (line 26) that stores the result of the REWRITE routine.

Termination of our algorithm is guaranteed because the graph traverse prevents the processing of a previously visited node (avoiding cycles) and furthermore a natural threshold parameter is established in order to limit the maximum distance from the conflicting term of a visited node in the graph.

```
1  //Returns an adequate query for onto_target,
2  //produced by a rewriting of Q with the least loss of information
3  QUERY_SELECTION(Q, ontoSource, ontoTarget) return Query
4   terms = DecomposeQuery(Q); // terms is the set of terms in Q
5   for each term in terms do
6   {
7     rewritingExpressions = REWRITE(term, ontoSource, ontoTarget);
8     //stores the term together with its adequate rewriting expressions
9     termsRewritings.add(term, rewritingExpressions);
10  }
11  //Constructs queries from the expressions obtained for each term
12  possibleQueries = ConstructQuery(Q, termsRewritings);
13  //Selects and returns the query that provides less loss of information
14  return LeastLoss(Q, possibleQueries);
15
16
17 //Constructs a queue of adequate expressions for term in onto_target
18 REWRITE(term, ontoSource, ontoTarget) return Queue<Expression>
19  resultQueue = new Queue();
20  traverseQueue = new Queue();
21  traverseQueue.add(term);
22  while  not traverseQueue.isEmpty() do
23  {
24    t = traverseQueue.remove();
25    if has_synonym(t, ontoTarget) then
26       resultQueue.add(map(t, ontoTarget));
27    else //t is a conflicting term
28       {        ceiling = directSuper(t);
29                traverseQueue.enqueueAll(ceiling);
30                floor = directSub(t);
31                traverseQueue.enqueueAll(floor);       }
32  }
33  return resultQueue;
```

5 Estimation of Information Loss

In this section we describe how we measure the loss of information caused by the rewriting of the original query. Also we explain in detail a use case that needs these rewritings to achieve an adequate query.

5.1 Measuring the Loss of Information

The system measures the loss of information using a composite measure adapted from [11]. This measure is based on the combination of the metrics *precision* and *recall* from Information Retrieval literature. We measure the proportion of retrieved data that is relevant (*precision*) and the proportion of relevant data that is retrieved (*recall*).

To calculate these metrics, we use datasets metadata published as VoID statistics. There are VoID statements that inform us of the number or entities of a class or the number of pairs of resources related by a property in a certain dataset. For instance, in `:DBpedia` dataset, the class `dbpedia:Book` has 26198 entities and there are 4102 triples with the property `dbpedia:notableWorks`.

```
:DBpedia a void:Dataset;
    void:classPartition [          void:propertyPartition [
        void:class dbpedia:Book;       void:property dbpedia:notableWorks;
        void:entities 26198;   ];      void:triples 4102;  ];
```

Given a conflicting term *ct*, we define *Ext(ct)* as the extension of *ct*; that is the collection of relevant instances for that term. Let us call *Rewr(ct)* to an expression obtained by the rewriting of a conflicting term *ct*, and *Ext(Rewr(ct))* to the extension of the rewritten expression, that is the retrieved instances for that expression.

We define $\sharp$*Ext(ct)* as the number of entities (resp. triples) registered for *ct* in the dataset (this value should be obtained from the metadata statistics). In the case of *Ext(Rewr(ct))*, we cannot expect a registered value in the metadata. Instead we calculate an estimation for an interval of values [$\sharp$*Ext(Rewr(ct).low)*, $\sharp$*Ext(Rewr(ct).high)*] which bound the minimum and the maximum cardinality of the expression extension. Those values are used for the calculation of our measures of precision and recall. However, due to the lack of space and the intricacy of the different cases that must be taken into account, we will not to present a detailed explanation for the calculation here.

Allow us to say that precision and recall of a rewriting of a conflicting term *ct* will be measured with an interval [*Precision(ct).low*, *Precision(ct).high*] where *Precision(ct).low* $= \mathcal{L}(\sharp Ext(ct),\ \sharp Ext(Rewr(ct).low),\ \sharp Ext(Rewr(ct).high))$ and *Precision(ct).high* $= \mathcal{H}(\sharp Ext(ct),\ \sharp Ext(Rewr(ct).low),\ \sharp Ext(Rewr(ct).high))$ are functional values calculated after a careful analysis of the diverse semantic relationships between *ct* and *Rewr(ct)*. Offered only as a hint, consider that the functions are variations on the following formulae, presented in [9]:

$$Precision(ct) = \frac{\sharp(Ext(ct) \cap Ext(Rewr(ct)))}{\sharp Ext(Rewr(ct))}; Recall(ct) = \frac{\sharp(Ext(ct) \cap Ext(Rewr(ct)))}{\sharp Ext(ct)}$$

In order to provide the user with a certain capacity for expressing preferences on precision or recall, we introduce a real value parameter α ($0 \leq \alpha \leq 1$) for tuning the function to calculate the loss of information due to the rewriting of a conflicting term. Again, this measure is expressed as an interval of values:

$$Loss(ct).low = 1 - \frac{1}{\alpha(\frac{1}{Precision(ct).high}) + (1-\alpha)(\frac{1}{Recall(ct).high})} \quad (1)$$

$$Loss(ct).high = 1 - \frac{1}{\alpha(\frac{1}{Precision(ct).low}) + (1-\alpha)(\frac{1}{Recall(ct).low})} \quad (2)$$

Finally, many functions can be considered for the calculation of the loss of information incurred for the rewriting of the entire original query Q. We are aware that more research and experimentation is needed to select the most appropriate ones for our task. Nevertheless, for the sake of this paper, let us use a very simple and effective one such as the maximum among the set of values that represent the losses.

$$Loss(Q).low = max\{Loss(ct).low \mid ct \text{ conflicting term in } Q\} \quad (3)$$

$$Loss(Q).high = max\{Loss(ct).high \mid ct \text{ conflicting term in } Q\} \quad (4)$$

5.2 Rewriting Example

This section describes in detail an example of the process followed by our system in the case that loss of information is produced during the rewriting process. Consider that the system is trying to answer the original query shown in figure 1, which is expressed with terms in the proprietary *bdi* vocabulary, and that the user decides to commit the query to the DBpedia dataset. Some of the mapping axioms at the disposal of the system are as follows:

```
bdi:Document rdfs:subClassOf dbpedia:Work .
bdi:Publication rdfs:subClassOf bdi:Document .
dbpedia:WrittenWork rdfs:subClassOf bdi:Publication .
dbpedia:Website rdfs:subClassOf bdi:Publication .
dbpedia:Miguel_de_Cervantes owl:sameAs bdi:Miguel_de_Cervantes.
dbpedia:notableWork owl:sameAs bdi:isAuthor .
```

During the process, two possible rewritings are generated, as shown in figure 1. The one on the left is due to the pair of mapping axioms that specify that `dbpedia:Work` is a superclass of the conflicting term `bdi:Publication`; and, the one on the right is due to a pair of mapping axioms that specify that `dbpedia:WritenWork` and `dbpedia:Website` are subclasses of `bdi:Publication` (see those terms in the shaded boxes of figure 1).

The calculation of the loss information for each rewriting is as follows. Notice that the only conflicting term, in this case, is ($bdi : Publication$). Firstly, the extension of the conflicting term and the rewriting expresssions are calculated.

Ext(bdi:Publication) = 503;
Ext(dbpedia:Work)= 387599;

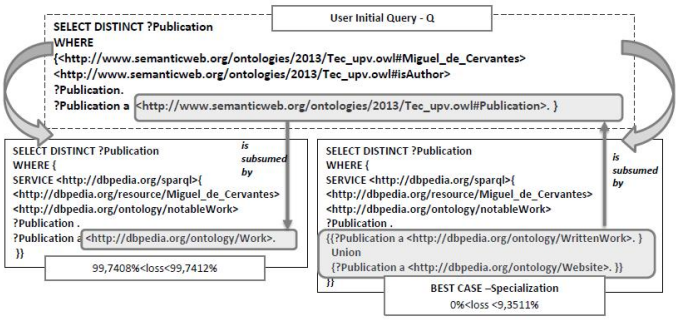

Fig. 1. Rewriting expressions generated

Ext(dbpedia:WrittenWork ∪ dbpedia:Website).low = min[40016, 2438] = 2438;
Ext(dbpedia:WrittenWork ∪ dbpedia:Website).high = 40016+2438 = 42454.

Secondly, precision and recall taking into account the relationships between the conflicting term and its rewriting expressions are calculated.

With respect to Rewr(bdi:Publication) = dbpedia:Work
[Precision.low = 0,0012960; Precision.high = 0,0012977; Recall = 1]
With respect to Rewr(bdi:Publication) = db:WrittenWork ∪ db:Website
[Precision = 1; Recall.low = 0,828969; Recall.high=1]

Then, the loss of information interval for $bdi : Publication$ with a parameter $\alpha = 0.5$ (meaning equal preference on precision and recall) is calculated.

With respect to Rewr(bdi:Publication) = dbpedia:Work
[Loss(bdi:Publication).low = 0,997408; Loss(bdi:Publication).high= 0,997412]
With respect to Rewr(bdi:Publication) = db:WrittenWork ∪ db:Website
[Loss(bdi:Publication).low = 0; Loss(bdi:Publication).high= 0.093511]

Considering the above information loss intervals, the system will choose the second option (replacing `bdi:Publication` with `db:WrittenWork` ∪ `db:Website`) as the loss of information is estimated to be between 0% and 9% (i.e., very low even with the posibility of being 0%, that is no loss of information). However, the first option (replacing `bdi:Publication` with `dbpedia:Work`) is estimated to incur in a big loss of information (about 99.7%), which is something that could be expected: `dbpedia:Work` references many works that are not publications. Anyway, in absence of the second option, the first one (despite returning many references to works that are not publications) also returns the publications included in `dbpedia:Work` which could satisfy the user. The alternative, not dealing with imprecise answers, would return nothing when a semantic preserving query into a new dataset cannot be achieved.

6 Conclusions

For this new era of Web of Data we present in this paper a proposal that offers the users the possibility of querying heterogeneous Linked Data sources in a friendly way. That means that users do not need to take notice of technical details

associated with the heterogeneity and variety of existing datasets. The proposal gives the opportunity to enrich the answer of the query incrementally, by visiting different datasets one by one, without needing to know the particular features of each dataset. The main component of the proposal is an algorithm that rewrites queries formulated by the users, using preferred vocabularies, into other ones expressed using the vocabularies of the datasets visited. This algorithm makes an extensive use of mapping axioms already defined in the datasets. To rewrite preserving query semantics may be difficult many times, for that reason the algorithm also handles rewritings with some loss of information.

Experiments are being carried out for tuning the estimation loss formulae.

Acknowledgements. This work is together supported by the TIN2010-21387-CO2-01 project and the Iñaki Goenaga (FCT-IG) Technology Centres Foundation.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) 5(3), 1–22 (2009)
2. Correndo, G., Salvadores, M., Millard, I., Glaser, H., Shadbolt, N.: Sparql query rewriting for implementing data integration over linked data. In: Proceedings of the 2010 EDBT/ICDT Workshops, p. 4. ACM (2010)
3. Elbassuoni, S., Ramanath, M., Weikum, G.: Query relaxation for entity-relationship search. In: The Semanic Web: Research and Applications, pp. 62–76 (2011)
4. Euzenat, J., Shvaiko, P.: Ontology matching, vol. 18. Springer, Heidelberg (2007)
5. Herzig, D., Tran, T.: One query to bind them all. In: COLD 2011, CEUR Workshop Proceedings, vol. 782 (2011)
6. Hurtado, C., Poulovassilis, A., Wood, P.: Query relaxation in rdf. Journal on Data Semantics X, 31–61 (2008)
7. Makris, K., Gioldasis, N., Bikakis, N., Christodoulakis, S.: Ontology mapping and sparql rewriting for querying federated rdf data sources. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6427, pp. 1108–1117. Springer, Heidelberg (2010)
8. Manola, F., Miller, E., McBride, B.: Rdf primer w3c recommendation (February 10, 2004)
9. Mena, E., Kashyap, V., Illarramendi, A., Sheth, A.: Imprecise answers on highly open and distributed environments: An approach based on information loss for multi-ontology based query processing. International Journal of Cooperative Information Systems (IJCIS) 9(4), 403–425 (2000)
10. Quilitz, B., Leser, U.: Querying distributed rdf data sources with sparql. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)
11. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of. Addison-Wesley (1989)
12. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization techniques for federated query processing on linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 601–616. Springer, Heidelberg (2011)