Rewriting SELECT SPARQL queries from 1:n complex correspondences

Élodie Thiéblin, Fabien Amarger, Ollivier Haemmerlé, Nathalie Hernandez, Cassia Trojahn

IRIT & Université de Toulouse 2 Jean Jaurès, Toulouse, France
elodie.thieblin@gmail.com,{fabien.amarger, ollivier.haemmerle, nathalie.hernandez,cassia.trojahn}@irit.fr

Abstract. This paper presents a mechanism for rewriting SPARQL queries based on complex ontology correspondences. While the usefulness of simple correspondences, involving single entities from both source and target ontologies, has long been recognized, query rewriting requires more expressive links between ontology entities expressing the true relationships between them. Here, complex correspondences, in the format 1:n, between overlapping ontologies are exploited for rewriting `SELECT` SPARQL queries, so that they can be expressed over different RDF data sets in the Linked Open Data. Our approach has been evaluated using two data sets, one from the agriculture domain and another based on a reduced set involving the ontologies from the OAEI Conference track.

1 Introduction

A SPARQL query is intrinsically related to the ontological model that describes the RDF source. To federate knowledge from different sources described by various ontologies, a SPARQL query must be adapted to each of the knowledge bases. In order to use the Linked Open Data potential at its best, it is important to bridge the gap of semantic heterogeneity between knowledge bases. Ontology matching [5] is a solution for finding correspondences (i.e., an alignment) between two ontologies. There are two types of correspondences : simple correspondences and complex correspondences. A simple correspondence matches an element from the first ontology to its semantically related ontological element in the second ontology. Nevertheless, simple correspondences cannot cover every case of use because of the model differences between ontological sources. Complex correspondences palliate the lack of expressiveness of simple alignments. They extend simple correspondences to correspondences between complex constructions of ontological entities of the two ontologies.

Simple correspondences can be easily used to transform SPARQL queries. The usual approach (integrated in the Alignment API [3]) is to replace the IRI of an ontological entity in the initial query by the IRI of its corresponding entity. This approach considers that the correspondence stands for an equivalence relation. However by using simple correspondences, not all SPARQL queries can be transformed. In this paper, an approach that exploits complex correspondences

for SPARQL query transformation is proposed. Even though there are only a few systems able to automatically generate them, manually drawing correspondences used in the proposed mechanism is likely a less fastidious task than manually rewriting every SPARQL query for each new RDF-triple store.

Our approach is based on a set of rules for rewriting a subset of `SELECT` SPARQL queries from complex correspondences involving an equivalence relation between ontology entities. These correspondences are expressed in EDOAL, a language proposed for representing complex correspondences. The approach has been validated on two data sets. The first one was built to meet the needs of agriculture experts willing to find cross knowledge about agronomic taxons between DBpedia and a dedicated knowledge base. The second data set was inspired from a subset of queries from the OAEI oa4qa[1] task data set and could be further developed in order to enrich this track.

The rest of the paper is structured as follows. §2 introduces ontology matching and the EDOAL syntax. §3 discusses related work and §4 presents the rewriting rules on which our approach is based. §5 discusses the validation of the approach and §6 concludes the paper and presents perspectives for future work.

2 Ontology matching

Matching two ontologies is the process of generating an alignment A between two ontologies $\mathcal{O}$ and $\mathcal{O}'$. A is directional, denoted $A_{\mathcal{O}\rightarrow\mathcal{O}'}$:

Definition 1 (Alignment). *An alignment $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ is a set of correspondences $A_{\mathcal{O}\rightarrow\mathcal{O}'} = \{c_1, c_2, ..., c_n\}$, where each c_i is a triple $\langle e_{\mathcal{O}}, e'_{\mathcal{O}}, r\rangle$:*

- *if the correspondence c_i is **simple**, both $e_{\mathcal{O}}$ and $e_{\mathcal{O}'}$ stand for one and only one entity (i.e., a class or a property) (1:1);*
- *if the correspondence is **complex**, at least one of $e_{\mathcal{O}}$ or $e_{\mathcal{O}'}$ involves one or more entities in a logical formulation. The correspondence is therefore (1:n), (m:1) or (m:n), where $e_{\mathcal{O}}$ refers to a subset of elements $\in \mathcal{O}$, and $e_{\mathcal{O}'}$ refers to a subset of elements $\in \mathcal{O}'$. The elements of $e_{\mathcal{O}}$, resp. $e_{\mathcal{O}'}$ form a logical construction using the constructors of a formal language (First-Order Logic or Description Logics);*
- *r is a relation, e.g., equivalence ($\equiv$), more general($\sqsupseteq$), more specific ($\sqsubseteq$), holding between $e_{\mathcal{O}}$ and $e_{\mathcal{O}'}$;*

The alignment $A_{O\rightarrow O'}$ is said *complex* if it contains at least one complex correspondence. A correspondence c_i, can also be noted $e_{\mathcal{O}}\ r\ e_{\mathcal{O}'}$, as for the complex correspondences in the following.

$$Chairman \equiv Demo\_Chair \sqcup OC\_Chair \sqcup PC\_Chair \sqcup Session\_Chair \sqcup Tutorial\_Chair \sqcup Workshop\_Chair \quad (1)$$

$$Accepted\_Paper \equiv Paper \sqcap \exists hasDecision.Acceptance \quad (2)$$

[1] `http://oaei.ontologymatching.org/2015/oa4qa/index.html`

Example 1 expresses a complex correspondence between entities from Cmt[2] and Ekaw[3] ontologies. It states that the concept *Chairman* of Cmt is equivalent to the union of the concepts *Demo_Chair*, *OC_Chair*, *PC_Chair*, *Session_Chair*, *Tutorial_Chair* and *Workshop_Chair* of Ekaw. Example 2 expresses a complex correspondence, where the concept *Accepted_Paper* of Ekaw is equivalent to the concept *Paper* of Cmt for which the domain of the object property *hasDecision* is restricted to an individual of the type *Acceptance*.

For representing complex correspondences, the EDOAL language has been proposed [6, 3]. It is fit to express simple and complex matching of cardinality (1:1), (1:n), (n:1) and (n:m). The entities $e_{\mathcal{O}}$ and $e_{\mathcal{O}'}$ are represented by expressions (class, relation or property expressions) that can be an ID (or IRI), a construction or a restriction. For a detailed description of EDOAL syntax the reader can refer to [6]. We illustrate this syntax with the following examples of 1:n complex correspondences given above. We use the prefixes `ekaw:<http://ekaw#>`, `cmt:<http://cmt#>`. Example 1 presents a *class expression* involving a class construction built with a union of concepts, while Example 2 expresses a *class expression* involving an attribute domain restriction.

Example 1

```
<entity1>
  <edoal:Class rdf:about="&cmt;Chairman"/>
</entity1>
<entity2>
<edoal:Class>
  <edoal:or rdf:parseType="Collection">
    <edoal:Class rdf:about="&ekaw;Demo_Chair"/>
    <edoal:Class rdf:about="&ekaw;OC_Chair"/>
     <edoal:Class rdf:about="&ekaw;PC_Chair"/>
     <edoal:Class rdf:about="&ekaw;Session_Chair"/>
    <edoal:Class rdf:about="&ekaw;Tutorial_Chair"/>
    <edoal:Class rdf:about="&ekaw;Workshop_Chair"/>
  </edoal:or>
</entity2>
<measure rdf:datatype="&xsd;float">1.0</measure>
<relation>Equivalence</relation>
```

Example 2

```
<entity1>
  <edoal:Class rdf:about="&ekaw;Accepted_Paper"/>
</entity1>
<entity2>
  <edoal:Class>
    <edoal:and rdf:parseType="Collection">
      <edoal:Class rdf:about="&cmt;Paper"/>
       <edoal:AttributeDomainRestriction>
         <edoal:onAttribute>
           <edoal:Relation rdf:about="&cmt;hasDecision"/>
         </edoal:onAttribute>
         <edoal:class>
           <edoal:Class rdf:about="&cmt;Acceptance"/>
         </edoal:class>
       </edoal:AttributeDomainRestriction>
    </edoal:and>
  </edoal:Class>
</entity2>
<measure rdf:datatype="&xsd;float">1.0</measure>
<relation>Equivalence</relation>
```

Additional examples of correspondences expressed in EDOAL are presented in examples 3 and 4. Example 3 shows a correspondence where the relation *writtenBy* of Ekaw is equivalent to *a relation expression* constructed with the inverse of the relation *writePaper* in Cmt. Example 4, involving the ConfOf[4] ontology (prefix `confOf:<http://confOf#>`), gives an example of a *class expression* constructed with an attribute value restriction stating that in Ekaw an *Early-Registered_Participant* is a *Participant* for which the value of the data property *earlyRegistration* is equal to *true* in ConfOf.

[2] `http://oaei.ontologymatching.org/2015/conference/data/cmt.owl`

[3] `http://oaei.ontologymatching.org/2015/conference/data/ekaw.owl`

[4] `http://oaei.ontologymatching.org/2015/conference/data/confof.owl`

Example 3

```
<entity1>
  <edoal:Relation rdf:about="&ekaw;writtenBy"/>
</entity1>
<entity2>
  <edoal:Relation>
    <edoal:inverse>
      <edoal:Relation rdf:about="&cmt;writePaper"/>
    </edoal:inverse>
  </edoal:Relation>
</entity2>
<measure rdf:datatype="&xsd;float">1.0</measure>
<relation>Equivalence</relation>
```

Example 4

```
<entity1>
  <edoal:Class rdf:about="&ekaw;Early-Registered_Participant"/>
</entity1>
<entity2>
  <edoal:Class>
    <edoal:and rdf:parseType="Collection">
      <edoal:Class rdf:about="&confOf;Participant"/>
      <edoal:AttributeValueRestriction>
        <edoal:onAttribute>
          <edoal:Property rdf:about="&confOf;earlyRegistration"/>
        <edoal:onAttribute>
        <edoal:comparator rdf:resource="&edoal;equals"/>
        <edoal:value>
          <edoal:Literal edoal:type="xsd;boolean" edoal:string="true"/>
        </edoal:value>
      </edoal:AttributeValueRestriction>
    </edoal:and>
  </edoal:Class>
</entity2>
<measure rdf:datatype="&xsd;float">1.0</measure>
<relation>Equivalence</relation>
```

3 Related Work

A naive approach for rewriting SPARQL queries consists in replacing the IRI of an entity of the initial query by the corresponding IRI in the alignment, using simple correspondences. This approach is integrated in the Alignment API [3]. However, it does not take into account the specific kind of relation expressed in the correspondence (e.g., generalisation or specialization). The approach in Euzenat *et al.* [4] aims at writing `CONSTRUCT` SPARQL queries from complex alignments. A new knowledge base expressed with the source ontology vocabulary is populated with the instances of the target knowledge base. A rewriting approach not limited to queries of type `CONSTRUCT` and that takes advantage of complex (1:n) alignments has been proposed by Correndo *et al.* [1]. It applies a declarative formalism for expressing alignments between RDF graphs. In [2], a subset of EDOAL expressions are transformed into a set of rewriting rules. The expressions involving the restrictions on concepts and properties and the restrictions on property occurrences and values are not featured in the rewriting rules. Makris *et al.* [9, 8] present the SPARQL-RW rewriting framework that applies a set of predefined rules for (complex) correspondences. They define a set of correspondence types on which the rewriting process is based (i.e., *Class Expression*, *Object Property Expression*, *Datatype Property*, and *Individual*). Zheng *et al.* [14] propose a rewriting algorithm that serves the purpose of context (i.e, units of measure) interchange for interoperability. Finally, Gillet *et al.* [7] propose an approach for rewriting query patterns that describe query families, using complex alignments. In this paper, we propose a set of rules for automatically rewriting SPARQL queries based on complex alignments. Differently from [4], our approach rewrites SPARQL queries instead of writing them from a complex alignment. Unlike [2], the proposed mechanism can handle restrictions on concepts and relations. In comparison with the [9] approach, EDOAL is an alternative to represent alignments in a more expressive (thus complete) way. For

instance, we propose occurrence and property datatype restrictions translation rules. However, our approach is limited to (1:n) complex alignments and does not handle initial SPARQL queries containing filters, unions, or other SPARQL options. The approach is based on the assumption that the queries to be transformed aim at retrieving new instances to meet a certain need. This is why only $\mathcal{T}box$ elements are taken into account. [14] focuses on context correspondences while our approach intends to translate all $\mathcal{T}box$ elements of a query. Finally, the proposal of [7] rewrites query patterns while we are interested in rewriting SPARQL queries, in a different level of abstraction. Although our approach relies on complex correspondences, their generation is out of the scope of this paper. The reader can refer to [13, 11] on the generation of complex correspondences based on patterns, linguistic approaches [12], or query mining [10].

4 SPARQL queries reformulation approach

Our approach focuses on the reformulation of a subset of SELECT SPARQL queries. We consider initial queries, which are to be rewritten, of the type:

$$Q_{\mathcal{O}} \; = \; SELECT \; DISTINCT? \; (\; Var+ \; | \; '*' \;) \; WHERE \; \{ \; T_{Q_{\mathcal{O}}} \; \}$$

where Var corresponds to the set of variables used as projection attributes and T_{Q_O} stands for the query pattern made of triples expressed using the source ontology $\mathcal{O}$. A triple t of T_{Q_O} is composed of a subject s, a predicate p and an object o. $\forall t \in T_{Q_{\mathcal{O}}}, \quad t = \langle s, p, o \rangle$. We only consider triples where s is a variable.

The purpose of our approach is to produce the set $T_{Q_{O'}}$ that contains the triples expressed according to entities of the ontology O', from T_{Q_O}, by using the complex alignment $A_{\mathcal{O} \rightarrow \mathcal{O}'}$. The approach is limited to complex correspondences (1:n) establishing an equivalence relation between entities of same nature. Such correspondences involve EDOAL expressions, as follows :

$\langle ClassID, ClassID | ClassConstruction | ClassConstruction, \equiv \rangle$
$\langle RelationID, RelationID | RelationConstruction | RelationRestriction, \equiv \rangle$
$\langle PropertyID, PropertyID | PropertyConstruction | PropertyRestriction, \equiv \rangle$

We also make the assumption that the alignment is complete and covers all the correspondences required to transform the entities of $T_{Q_{\mathcal{O}}}$. We define rules that take the set of triples $T_{Q_{\mathcal{O}}}$ as input and generate a SPARQL query. Three types of triples in $T_{Q_{\mathcal{O}}}$ are considered : *Class Object Triples*, *Predicate Triples* and *Other Triples*.

Algorithm 1 depicts the SPARQL query rewriting process. The **rewriteClassObject** and **rewritePredicate** functions apply the rules described in the following sections. These functions are recursive and can call each other. If a triple is not a *Class Object Triples* or a *Predicate Triples*, it means that its subject s is a variable, its predicate is an object property or a data property for which no correspondence is needed and its object is either a literal or a variable. This kind of triple does not need any transformation and is directly added to the final query. An example of such triple is `?s rdfs:label "a literal"`.

Algorithm 2 Rewriting mechanism process

```
new_query ← " "
for all triple t = ⟨s, p, o⟩ in query do
   if t is a Class Object Triple then
      new_query ← new_query + rewriteClassObject(s,p, o_O')
   else if t is a Predicate Triple then
      new_query ← new_query + rewritePredicate(s,p_O', o)
   else
      new_query ← new_query + t
   end if
end for
return new_query
```

4.1 Class Object Triples

Class object triples, denoted $T^{Class}_{Q_\mathcal{O}}$, are structured as

$$\forall t \in T^{Class}_{Q_\mathcal{O}}, t = \langle s, p, o_\mathcal{O} \rangle \quad \text{, where} \begin{cases} s \text{ is a variable} \\ p \text{ is rdf:type} \\ o_\mathcal{O} \text{ is a } \textit{ClassID} \\ \exists < o_\mathcal{O}, o_{\mathcal{O}'}, \equiv > \ \in A_{\mathcal{O}\to\mathcal{O}'} \end{cases}$$

A class triple is identified if its object $o_\mathcal{O}$ is a *ClassID* and if there is a correspondence linking $o_\mathcal{O}$ to a class expression $o_{\mathcal{O}'}$ in $A_{\mathcal{O}\to\mathcal{O}'}$. In the transformation of a class triple t, its subject s and its predicate p remain the same. Only its object $o_\mathcal{O}$ is transformed according to its equivalent element $o_{\mathcal{O}'}$ in the alignment. The transformation rules of the **rewriteClassObject** function depend on the nature of the expression $o_{\mathcal{O}'}$, as follows:

1. *ClassID*: The expression $o_{\mathcal{O}'}$ is a*ClassID*. The transformed triple return by the function is: s p `IRI`$(o_{\mathcal{O}'})$.
2. *ClassConstruction*: $o_{\mathcal{O}'}$ is a class construction between two or more class expressions denoted by: $e^1_{\mathcal{O}'}$, $e^2_{\mathcal{O}'}$, $e^n_{\mathcal{O}'}$. The transformation rule depends on the construction operator.
   (a) AND: transforming an intersection consists in rewriting each triplet having as subject s, as predicate p and as object a distinct $e^i_{\mathcal{O}'}$ expression:
   rewriteClassObject$(s, p, e^1_{\mathcal{O}'})$ + **rewriteClassObject**$(s, p, e^2_{\mathcal{O}'})$ + ... + **rewriteClassObject**$(s, p, e^n_{\mathcal{O}'})$
   (b) OR: transforming a union consists in using the SPARQL keyword "UNION" between the rewriting of each triplet having as subject s, as predicate p and as object a distinct $e^i_{\mathcal{O}'}$:[5]
   `{ +` **rewriteClassObject**$(s, p, e^1_{\mathcal{O}'})$ `+ } UNION { +` **rewriteClassObject**$(s, p, e^2_{\mathcal{O}'})$ `+ } +...+ UNION { +` **rewriteClassObject**$(s, p, e^n_{\mathcal{O}'})$ `+}`
   Table 1 shows an example of query rewriting based on the complex correspondence of Example 1, involving a *class construction with OR*.

[5]For sake of clarity and simplicity, we do not represent string delimiters.

Query for Cmt	Transformed query for Ekaw

SELECT ?z WHERE {
?z rdf:type cmt:Chairman.
}
	SELECT ?z WHERE {
	{?z rdf:type ekaw:Demo_Chair. }
	UNION {?z rdf:type ekaw:OC_Chair. }
	UNION {?z rdf:type ekaw:PC_Chair. }
	UNION {?z rdf:type ekaw:Session_Chair. }
	UNION {?z rdf:type ekaw:Tutorial_Chair. }
	UNION {?z rdf:type ekaw:Workshop_Chair. }}

Table 1. Transformation of a class triple based on the correspondence of Example 1 between a *classID* and a *class construction* using the OR rule.

(c) NOT: finding the negation of a class expression consists in finding the set of triples $\langle s, p, v\rangle$, where v is an intermediate variable, and from which the triples $\langle s, p, e^1_{\mathcal{O}'}\rangle$ are removed :

s p v . MINUS { + **rewriteClassObject**$(s, p, e^1_{\mathcal{O}'})$ + }

3. *ClassRestriction* : Restriction on class expressions takes into account relation or property expressions noted $relation(o_{\mathcal{O}'})$ or $property(o_{\mathcal{O}'})$. The transformation of the class triple depends on the nature of the restriction:
   (a) *TypeRestriction*: this restriction applies to a property expression stated in $o_{\mathcal{O}'}$ that limits the datatype of the property to a given *type*. The transformation rule consists in using an intermediate variable v that becomes the object of a new triple (that will keep on being rewritten according to the nature of $property(o_{\mathcal{O}'})$). The type restriction is applied to v with the use of a SPARQL `FILTER` and the `datatype(`v`)` function:
   rewritePredicate$(s, property(o_{\mathcal{O}'}), v)$ + FILTER (datatype(v) = *type*)
   (b) *DomainRestriction*: this restriction limits the range of a relation expression stated in $o_{\mathcal{O}'}$ to a class expression $range(o_{\mathcal{O}'})$ also stated in $o_{\mathcal{O}'}$. The **rewritePredicate** function is called with the relation $relation(o_{\mathcal{O}'})$ between the subject s and an intermediate variable v. The **rewriteClassObject** function is called to assert that v is an instance of the $range(o_{\mathcal{O}'})$ class expression : **rewritePredicate**$(s, relation(o_{\mathcal{O}'}), v)$ + **rewriteClassObject**$(v, \texttt{rdf:type}, range(o_{\mathcal{O}'}))$
   Table 2 presents a query transformation example based on the correspondence of Example 2 involving this kind of restriction.

Query for Ekaw	Transformed query for Cmt

SELECT ?z WHERE {
?z rdf:type ekaw:Accepted_Paper.
}
	SELECT ?z WHERE {
	?z rdf:type cmt:Paper.
	?z cmt:hasDecision ?var_temp.
	?var_temp rdf:type cmt:Acceptance. }

Table 2. Transformation of a class triple based on the correspondence on Example 2 between a *classID* and a *class expression* using the *DomainRestriction* rule.

(c) *ValueRestriction*: this restriction applies to a relation or property expression. The **rewritePredicate** function is called between the subject s, the $relation(o_{\mathcal{O}'})$ or $property(o_{\mathcal{O}'})$ and an intermediate variable v.

To restrain the values that can be taken by v, a SPARQL "FILTER" is used to compare v to a *value* given in the class expression. In the actual implementation, the stated value *value* can only be a literal or an instance. The comparator *cp* used in the SPARQL `FILTER` is one of the comparators provided by the EDOAL syntax : "=", ">" and "<". **rewritePredicate**$(s, relation/property(o_{\mathcal{O}'}), v)$ `+ FILTER (`v cp $value$`)` where $cp \in \{=, <, >\}$. For a "=" comparator, the resulting query is not optimal in terms of performance. The rewriting rule exception could be : **rewritePredicate**$(s, relation/property(o_{\mathcal{O}'}), value)$ instead of using an intermediate variable and a `FILTER` that applies to it. Table 3 presents a transformation example based on the correspondence of Example 4.

Query for Ekaw	Transformed query for ConfOf

`SELECT ?z WHERE {`
`?z rdf:type ekaw:Early-Registered_Participant .`
`}`
	`SELECT ?z WHERE {`
	`?z rdf:type confOf:Participant.`
	`?z confOf:earlyRegistration ?var_temp.`
	`FILTER(?var_temp="true"^^ xsd:boolean).}`

Table 3. Transformation of a triple using the correspondence of Example 4 between a class ID and a class expression using the *ValueRestriction* rule.

(d) *AttributeOccurrenceRestriction*: this restriction restrains the number of occurrences of a relation or a property expression. In order to count this number of occurrences, a SPARQL `SELECT` is imbricated to link the subject s to the count $count_v$ of an intermediate variable v. The value of $count_v$ is calculated thanks to the SPARQL `COUNT` function. The graph pattern in the imbricated `SELECT` is represented by the call of **rewritePredicate**$(s, relation/property(o_{\mathcal{O}'}), v)$. After the imbricated `SELECT`, a `FILTER` limits the value of $count_v$ to the restriction value val_{rest} with the comparator *cp* (both stated in the class restriction).

`{{SELECT` s `(COUNT(`v`) AS` $count_v$`) WHERE`
`{ +` **rewritePredicate**$(s, relation/property(o_{\mathcal{O}'}), v)$ `+ }`
`GROUP BY` s`.}`
`FILTER (`$compte_v$ cp val_{rest}`)}` , where $cp \in \{=, <, >\}$

Here, the resulting query could be optimized for a relation or a property occurring at least once ($count > 0$) . Instead of having an imbricated `SELECT`, the rewriting rule could be: **rewritePredicate**$(s, relation/property(o_{\mathcal{O}'}), v)$ with v a temporary variable.

4.2 Predicate Triples

Predicate triples, denoted by $T_{Q\mathcal{O}}^{Predicate}$ have the following structure :

$$\forall t \in T_{Q\mathcal{O}}^{Predicate} t = \langle s, p_{\mathcal{O}}, o \rangle \quad \text{, where } \begin{cases} s \text{ is a variable} \\ p_{\mathcal{O}} = \text{ a RelationId or PropertyId} \\ o \text{ is a variable, an instance or a literal} \\ \exists < p_{\mathcal{O}}, p_{\mathcal{O}'}, \equiv > \in A_{\mathcal{O} \to \mathcal{O}'} \end{cases}$$

In predicate triples, $p_{\mathcal{O}}$ is either a *RelationId* or a *PropertyId* and $p_{\mathcal{O}'}$ is respectively a relation expression or a property expression. A relation triple is transformed according to the nature of the expression $p_{\mathcal{O}'}$.

1. *RelationId* or *PropertyId*: the following triple is added to $T_{Q_{\mathcal{O}'}}$: s `IRI`$(p_{\mathcal{O}'})$ o.
2. *RelationConstruction* or *PropertyConstruction*: $p_{\mathcal{O}'}$ is a construction between relation or property expressions designated by $p^1_{\mathcal{O}'}$, $p^2_{\mathcal{O}'}$, $p^n_{\mathcal{O}'}$. The transformation of the relation triple depends on the operator of the construction.
   (a) AND: this construction can be between two or more expressions (relation expressions resp. property expressions). The **rewritePredicate** function is called as follows:
   rewritePredicate$(s, p^1_{\mathcal{O}'}, o)$ `+` **rewritePredicate**$(s, p^2_{\mathcal{O}'}, o)$ `+ ...` `+` **rewritePredicate**$(s, p^n_{\mathcal{O}'}, o)$
   (b) OR: this construction can be between two or more expressions (relation expressions resp. property expressions). A SPARQL `UNION` links the calls to **rewritePredicate**:
   `{ +` **rewritePredicate**$(s, p^1_{\mathcal{O}'}, o)$ `+} UNION { +` **rewritePredicate**$(s, p^2_{\mathcal{O}'}, o)$ `+ } + ... + UNION { +` **rewritePredicate**$(s, p^n_{\mathcal{O}'}, o)$ `+ }`
   (c) NOT: the negation of a relation is the subset of all relations minus this relation. To represent all relations an intermediate variable v is introduced. The negation will be done using a SPARQL `MINUS`:
   s v o `. MINUS { +` **rewritePredicate**$(s, p^1_{\mathcal{O}'}, o)$ `+ }`
   (d) COMPOSE: a relation composition is a relation chain. Intermediate variables v_1, v_2, etc. are introduced to complete the chain between the subject s and the object o. If the relation expression $p_{\mathcal{O}'}$ is a *RelationExpression*, all the expressions of the chain will be relation expressions. If $p_{\mathcal{O}'}$ is a *PropertyExpression*, all the expressions of the chain will be relation expressions except the last one that will be a property expression. We assume that a composition imbrication or the use of a negation inside a composition is a modeling problem in the alignment itself.
   rewritePredicate$(s, p^1_{\mathcal{O}'}, v_1)$ `+` **rewritePredicate**$(v_1, p^2_{\mathcal{O}'}, v_2)$ `+ ... +` **rewritePredicate**$(v_{n-1}, p^n_{\mathcal{O}'}, o)$
   (e) INVERSE : this construction only applies to a *RelationExpression*. Inverting a relation consists in switching its subject and its object in a triple. **rewritePredicate**$(o, p^1_{\mathcal{O}'}, s)$ Table 4 gives an example of a triple transformation based on the correspondence of Example 3.
   (f) REFLEXIVE : this construction only applies to a *RelationExpression*. This operator is used to specify that a relation links its subject s to itself.
   rewritePredicate $(s, p^1_{\mathcal{O}'}, s)$
   (g) SYMMETRIC : this construction only applies to a *RelationExpression*. This operator is used to specify that a relation is used both ways : it is the intersection of a relation and its inverse.
   rewritePredicate$(s, p^1_{\mathcal{O}'}, o)$ + **rewritePredicate**$(o, p^1_{\mathcal{O}'}, s)$
3. *RelationDomainRestriction* or *PropertyDomainRestriction*: these restrictions limit the domain of a relation or property to a class expression $domain(p^1_{\mathcal{O}'})$ stated in the relation or property expression.

rewriteClassObject$(s, \texttt{rdf:type}, domain(p_{\mathcal{O}'}))$

4. *RelationCoDomainRestriction*: this restriction restrains the range of a *RelationExpression* to a class expression $range(p_{\mathcal{O}'})$.
   rewriteClassObject$(o, \texttt{rdf:type}, range(p_{\mathcal{O}'}))$
5. *PropertyTypeRestriction*: this restriction limits the datatype of a property to a given type *type* in the property expression $_{\mathcal{O}'}$. A SPARQL `FILTER` with the `datatype(`o`)` function is used. `FILTER (datatype(`o`) =` *type*`)`

Query for Ekaw	Transformed query for Cmt
SELECT ?z WHERE {	SELECT ?z WHERE{
?paper :writtenBy ?author. }	?author `cmt:writePaper` ?paper. }

Table 4. Transformed of a triple using the correspondence of Example 3 between a relation ID and a relation construction with the INVERSE constructor.

5 Validation

As far as we know, there is no available data set consisting of two knowledge bases, a complex and complete alignment between two ontologies and corresponding SPARQL queries for both bases. In the context of the OAEI oa4qa[6], a data set involving simple alignments is available. In order to fill this gap, we have manually created two data sets, following the principle of the oa4qa task. The validation of our mechanism checks that the translated query retrieves the same results as the reference query. Although these data sets only contain a small number of queries, it serves as a basis for a first validation of our approach.

Knowledge bases and SPARQL queries. The first data set was built during a project aiming at collecting knowledge about plant taxonomy. To meet this need, the knowledge bases Agronomic Taxon[7] and DBpedia have been considered. The task consists of retrieving answers to the following queries:

- $qa1$: which are the taxa of type species ?
- $qa2$: which are the taxa having for higher taxonomic rank a family taxon ?
- $qa3$: which are the taxa of taxonomical rank kingdom ?
- $qa4$: which are the taxa of taxonomical rank order ?
- $qa5$: which are the taxa of taxonomical rank genus ?

In order to build this data set, reference SPARQL queries corresponding to the natural language description above have been written manually for each knowledge base. The same approach was followed to construct the second data set. It aims at interrogating a subset of the OAEI 2015 ontologies about conference organization[8]. Three ontologies of this data set were considered (Cmt, ConfOf and Ekaw). We have defined the SPARQL queries answering the following queries :

- $qb1$: which are the reviewers of accepted papers ? (Ekaw to Cmt)

[6] http://oaei.ontologymatching.org/2015/
[7] http://ontology.irstea.fr/AgronomicTaxon
[8] http://oaei.ontologymatching.org/2015/conference/index.html

– $qb2$: which are authors of long submissions ? (Ekaw to Cmt)
– $qb3$: which are the chairmen who have submitted a paper ? (Cmt to Ekaw)
– $qc1$: which are the early registered participants who authored a submitted paper ? (Ekaw to ConfOf)
– $qc2$: which are the late registered participants who wrote a poster ?(Ekaw to ConfOf)

The three ontologies were populated with instances meeting these needs. Simultaneously, the queries were transformed into SPARQL queries specifically written for each of the knowledge bases.

Complex alignments. 10 complex correspondences (and 1 simple) have been manually produced between Agronomic Taxon and DBpedia. 8 simple and 6 complex correspondences have been manually produced between the three ontologies of the Conference data set. The alignments are available online[9].

Discussion. Our validation is based on the manual comparison of the set of results returned from the automatically rewritten query with respect to the results of the reference query. Even though the reference query and the rewritten one differ in terms of syntax, they retrieve the same set of instances. For example, Table 5 shows the queries considered for the need $qc1$ described above. The initial SPARQL query for Ekaw was transformed by using the complex correspondences of Example 4 and the simple correspondence $ekaw : authorOf \equiv confOf : writes$. As stated above, although the generated query is not syntactically equivalent to the reference query for ConfOf, they retrieve the same set of instances. The whole set of rewritten queries is available online[10].

Initial query for Ekaw (a)	Reference query for ConfOf	Generated query for ConfOf

`SELECT ?person WHERE {`
`?person :authorOf ?paper.`
`?paper a :Paper.`
`?person rdf:type`
`:Early-Registered_Participant.}`
	`SELECT ?person WHERE{`
	`?person`
	`:earlyRegistration true.`
	`?person :writes ?papier.`
	`?papier a :Paper.}`
		`SELECT ?person WHERE {`
		`?person :writes ?paper.`
		`?paper a :Paper.`
		`?person rdf:type :Participant.`
		`?person :earlyRegistration`
		`?v_temp0.`
		`FILTER(?v_temp0 =`
		`"true"^^xsd:boolean).}`

Table 5. Transformation of an initial query in comparison to its reference.

6 Conclusion and perspectives

In this paper, we have presented an approach to rewrite SELECT SPARQL queries formulated for a particular ontology to interrogate a knowledge base based on a second ontology using (1:n) complex correspondences. The proposed approach has been validated on two data sets manually created. There are many improvements to make to this mechanism. Indeed, the approach is limited to

[9] https://www.irit.fr/recherches/MELODI/telechargements/alignements.zip
[10] https://www.irit.fr/recherches/MELODI/telechargements/requetesgenerees.zip

formatted queries composed of triples whose subject is a variable. Instance alignments are not considered yet. We do not consider as well (n:m) correspondences. Proposals on complex graph pattern recognition in SPARQL queries would be interesting to take into account in order to address that matter. Another point is that we do not distinguish the kind of relation of a correspondence (subsumption and equivalence). Moreover, some EDOAL syntax of concepts have not been implemented, such as functions on literal (string concatenation, arithmetic operations, etc. that could be used in particular for value restrictions). Finally, property value restrictions is another EDOAL expression that was not implemented because it is more likely to be found in (n:m) correspondences. We plan to address all these points in future work.

References

1. Correndo, G., Salvadores, M., Millard, I., Glaser, H., Shadbolt, N.: SPARQL Query Rewriting for Implementing Data Integration over Linked Data. In: 1st International Workshop on Data Semantics (DataSem 2010) (March 2010)
2. Correndo, G., Shadbolt, N.: Translating expressive ontology mappings into rewriting rules to implement query rewriting. In: 6th Workshop on Ontology Matching (2011)
3. David, J., Euzenat, J., Scharffe, F., Trojahn, C.: The Alignment API 4.0. Semantic Web 2(1), 3–10 (2011)
4. Euzenat, J., Polleres, A., Scharffe, F.: Processing ontology alignments with sparql. In: International Conference on Complex, Intelligent and Software Intensive Systems. pp. 913–917 (2008)
5. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer-Verlag, Berlin, Heidelberg (2007)
6. Euzenat, J., Scharffe, F., Zimmermann, A.: Expressive alignment language and implementation. Tech. rep., INRIA (2007), `http://hal.inria.fr/hal-00822892/`
7. Gillet, P., Trojahn, C., Haemmerlé, O., Pradel, C.: Complex correspondences for query patterns rewriting. In: 8th Workshop on Ontology Matching (2013)
8. Makris, K., Bikakis, N., Gioldasis, N., Christodoulakis, S.: SPARQL-RW: transparent query access over mapped RDF data sources. In: 15th International Conference on Extending Database Technology. pp. 610–613. ACM (2012)
9. Makris, K., Gioldasis, N., Bikakis, N., Christodoulakis, S.: Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources. In: OTM Confederated International Conferences. pp. 1108–1117 (2010)
10. Qin, H., Dou, D., LePendu, P.: Discovering executable semantic mappings between ontologies. In: OTM International Conference. pp. 832–849 (2007)
11. Ritze, D., Meilicke, C., Sváb-Zamazal, O., Stuckenschmidt, H.: A pattern-based ontology matching approach for detecting complex correspondences. In: 4th Workshop on Ontology Matching (2009)
12. Ritze, D., Völker, J., Meilicke, C., Sváb-Zamazal, O.: Linguistic analysis for complex ontology matching. In: 5th Workshop on Ontology Matching (2010)
13. Scharffe, F., Fensel, D.: Correspondence patterns for ontology alignment. In: Knowledge Engineering: Practice and Patterns, pp. 83–92. Springer (2008)
14. Zheng, X., Madnick, S.E., Li, X.: SPARQL Query Mediation over RDF Data Sources with Disparate Contexts. In: WWW Workshop on Linked Data on the Web (2012)