Processing ontology alignments with SPARQL (Position paper)

Jérôme Euzenat
INRIA & LIG
Grenoble, France
Email: jerome.euzenat@inrialpes.fr

Axel Polleres Digital Enterprise Research Institute
National University of Ireland, Galway
Email: axel.polleres@deri.org

François Scharffe University of Innsbruck
Innsbruck, Austria
Email: francois.scharffe@uibk.ac.at

Abstract

Solving problems raised by heterogeneous ontologies can be achieved by matching the ontologies and processing the resulting alignments. This is typical of data mediation in which the data must be translated from one knowledge source to another. We propose to solve the data translation problem, i.e. the processing part, using the SPARQL query language. Indeed, such a language is particularly adequate for extracting data from one ontology and, through its CONSTRUCT statement, for generating new data. We present examples of such transformations, but we also present a set of example correspondences illustrating the needs for particular representation constructs, such as aggregates, value-generating built-in functions and paths, which are missing from SPARQL. Hence, we advocate the use of two SPARQL extensions providing these missing features.

1 Introduction

Dealing with heterogeneity between ontologies is very often achieved by establishing correspondences between entities found in these ontologies and transforming data according to these correspondences, be it for integrating heterogeneous data sources or exchanging messages between services. Relations between entities featured in alignments may be very complex. For expressing them, we have developed an expressive alignment language independent from knowledge representation and processing languages [1].

When one wants to actually transform data, the correspondences expressed in this language must be processed. In particular, data translation is needed when instances, described using a source ontology, have to be provided as instances of a target ontology. This scenario is expected to become common as more ontologies are developed and used to describe RDF data. A natural choice for translating data would be to use a query language because they allow extracting and transforming data. Hence, when buying into RDF(S) and OWL being the standards for describing ontologies and data, SPARQL [2] is a natural candidate as a language for expressing and processing the correspondences. However, SPARQL is not powerful enough for covering the full complexity of the expressive language in particular because it lacks the possibility to express aggregates, value-generating functions and paths. Thus, we propose to combine two recent extensions of SPARQL in order to be able to transform data according to complex alignments: SPARQL++ [3] provides aggregates, value-generating built-ins and (possibly recursive) processing of mappings expressed in SPARQL and PSPARQL [4] provides queries on path expressions (made from regular expression patterns) which are sufficient for expressing those of the expressive language.

Below, we illustrate this paper with a data translation problem between the FOAF [5] and vCard[1] ontologies. Both vocabularies describe information about persons and organizations, both are extensively used, and they cover complementary as well as overlapping aspects.

2 Alignment Representation

The Alignment format [6] allows the representation of simple correspondences between ontological entities. It provides an interchange format between alignments created by ontology matching algorithms [7]. It is organized around a small set of constructs: an alignment is described through

[1] http://www.w3.org/2006/vcard/ns

a set of correspondences, together with related metadata such as the aligned ontologies, its purpose, or the way it was built. Alignments are made of a set of correspondences giving a description of the alignment entities. Each correspondence describe the relation between two ontological entities. A measure of confidence in the correspondence is given.

A sample correspondence, expressing the equivalence between a vCard and a person is described as follows:

```
<Cell>
  <entity1 rdf:resource="&foaf;Person"/>
  <entity2 rdf:resource="&vc;VCard"/>
  <measure rdf:datatype="&xsd;float">1.0</measure>
  <relation>equivalence</relation>
</Cell>
```

The Alignment format format however only permits the representation of one-to-one correspondences between homogeneous ontological entities. It is thus not able to represent complex correspondences such as the one introduced below. Particularly, it would need the following constructs:

- operators to relate an entity in one ontology to a combination of entities in the other.
- conditions restricting entities scope
- transformation of properties values such as aggregates functions or datatype conversions.

We have developed an expressive alignment language [1] extending this format to overcome the aforementionned limitations. This language extends the cell description of the alignment format with new constructs. First, it distinguishes four types of ontology entities: Classes, Relations, Properties and Instances. Relations relate two classes, while properties relate classes to datatypes. Heterogeneous correspondences such as class to relation can be specified. Entities can be grouped together via set operators. Specific conditions are available for each kind of entities in order to restrict their scope. The scope of a class can therefore be restricted by the value, type or occurrence of one of its attributes. Relations can be restricted on the type of their domain or range, and attributes can be restricted on their value, or domain type. Finally, it is possible to specify transformations on attributes values by referring to methods or web services. We give in the following an example cell including attribute value conditions in order to restrict the scope of classes (people having a work phone number starting with +33476 are based near Grenoble).

```
<Cell>
 <entity1>
  <Class rdf:about="&foaf;Person">
    <attributeValueCondition>
      <Restriction>
       <onProperty
         rdf:resource="&foaf;based_near"/>
       <comparator
         rdf:datatype="&xsd;string">
          xsd:equals</comparator>
       <value rdf:datatype="&xsd;string">
         Grenoble</value>
      </Restriction>
    </attributeValueCondition>
  </Class>
 </entity1>
 <entity2>
  <Class rdf:about="&v;VCard">
   <attributeValueCondition>
     <Restriction>
      <onProperty
        rdf:resource="&v;workTel"/>
      <comparator
        rdf:datatype="&xsd;string">
         xsd:startsWith</comparator>
      <value rdf:datatype="&xsd;string">
        +33476</value>
     </Restriction>
    </attributeValueCondition>
  </Class>
 </entity2>
 <measure RDF:datatype='&BSD;float'>1.0
 </measure>
 <relation>equivalence</relation>
</Cell>
```

This language provides a high level description of ontology alignments. It can be conveniently used as an exchange format between matching algorithms, graphical user interfaces and mediation languages. Let see how it can be processed when one needs to transfer data from one ontology to another.

3 Grounding

Ontology mediation is a complex process involving two main phases [9]. At design time, the alignment is constructed: matching algorithms are used to automatically discover correspondences, and graphical mapping interfaces assist the process of refining these correspondences, eventually using correspondence patterns [10]. An expressive exchange format, such as the one presented above, is required in order to carry the meaning of the correspondences.

At run time, the previously built alignments are executed in a particular mediation task such as merging two ontologies in a new one, translating a query addressed to a source ontology into a query addressed to a target ontology, or translating the data described under a source ontology into instance data described according to a target ontology. For each task, the best adapted formalism should be used. When

merging ontologies, the alignment should be expressed as axioms in the ontology language. When translating queries, a rule language might be best appropriate. When translating instance data, a query language such as SPARQL seems to be best appropriate. SPARQL has the advantage to be widely used for querying RDF data on the web. This makes SPARQL-based data translation more usable for semantic-web users in comparison to rule languages or XML-based extraction techniques.

We call grounding the process of transforming the alignment expressed in an alignment representation formalism into the knowledge representation formalism executable for a particular task. For the data translation task presented in this paper, we need to specify a grounding from the expressive alignment format to a proposed executable form.

The example correspondence betwen Foaf files and vCards in Section 2 will be by this process translated in a corresponding SPARQL expression.

4 Data translation using SPARQL

SPARQL [2] is a W3C candidate recommendation for querying RDF. Typically, queries in SPARQL are used to select bindings of RDF-Terms to *variables* from a set of source RDF graphs (also called the *dataset*[2]) according to a *graph pattern*, i.e. in a slightly simplified view such a query follows the general structure:

```
SELECT variables
FROM dataset
WHERE { graph pattern }
```

Answers to a SPARQL query Q rely on computing the set of possible homomorphisms from the basic graph pattern(s) of Q into the RDF graph representing the knowledge base.
In the patterns SPARQL queries support features such as filter expressions, unions, i.e., disjunction, or optional query parts to allow sophisticated queries. For instance, to select all addresses and their full names from a vCard file using vCard's RDF representation [11] one could use the following query

```
SELECT ?X ?FN
WHERE { ?X vc:FN ?FN .
    FILTER isLiteral(?FN) }
```

If we want to exploit instance data described under one ontology while our application has been designed for the other, we need a translation mechanism. Through its CONSTRUCT statement, SPARQL also provide the possibility to construct an RDF graph as the result of a query over another graph. This is a natural mechanism for writing mapping rules between RDF vocabularies. For instance, the following query illustrates a CONSTRUCT query translating a `foaf:Person` into a `vc:VCard`.

[2]The FROM clause will be omitted in subsequent examples.

```
CONSTRUCT { ?x rdf:type vc:VCard }
WHERE { ?x rdf:type foaf:Person }
```

This simple example needs to be completed in order to additionally translate – possibly recursively – a person's properties, like name, address, or telephone number.

CONSTRUCT queries can be applied the same selections as the other queries: The query above can be modified to map the names in vCard addresses to FOAF as follows:

```
CONSTRUCT { ?X foaf:name ?FN . }
WHERE { ?X vc:FN ?FN .
    FILTER isLiteral(?FN) }
```

Particularly, the mapping from Section 2 above can likewise be modeled by a simple CONSTRUCT statement:

```
CONSTRUCT { ?X a foaf:Person.
    ?X foaf:based_near "Grenoble"^^xsd:string. }
WHERE { ?X a v:VCard .
        ?X v:workTel ?PH.
    FILTER startsWith(?PH, "+33476") }
```

and executing it on a set of instance data represented in RDF would yield the transformed ontology instances in the target ontology.

However, it turns out that the available constructs are not sufficient for a fully-fledged mapping language.

5 SPARQL extensions for accurate translation

We introduce below three features that SPARQL lacks, though they would be particularly useful for processing alignments. These features are aggregate computation, individual generation and path expressions.

5.1 Aggregates

The DOAP vocabulary [12] contains revision, i.e., version numbers of released versions of projects. With an aggregate function `MAX`, one can map DOAP information into the RDF Open Source Software Vocabulary [13], which talks about the latest release of a project, by picking the maximum value (numerically or lexicographically) of the set of revision numbers specified by a graph pattern as follows:

```
CONSTRUCT { ?P os:latestRelease
                MAX(?V : ?P doap:release ?R.
                         ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project . }
```

Other aggregates, such as count, average, or sum, might be needed for complex and complete mappings.

5.2 Individual generation

Completing the mapping betwen vCard and FOAF, if we try mapping from `vc:homeTel` to `foaf:phone`, we observe that the former is a datatype property and the latter an object property. Basically, a mapping needs a conversion function, generating a new URI

```
CONSTRUCT {?X foaf:phone
  xsd:anyURI(
     fn:concat("tel:",fn:encode-for-uri(?T))).}
WHERE { ?X vc:tel ?T . }
```

Such value generations are not allowed in SPARQL at the moment, but defined and implemented in an extended version of SPARQL, called SPARQL++ [3], which we consider a valid basis for a mapping language, but there are still more issues missing. Blank nodes, which correspond to existential variables in mapping rule heads, involve some more complications, which are discussed in more detail in [3].

5.3 Paths

Another missing part is path expressions, which are not expressible in SPARQL, a fairly surprising fact for a language which claims to be a graph query language.

PSPARQL [4] extends SPARQL by replacing the atomic SPARQL graph patterns, i.e., RDF graphs with variables, by RDF graphs with variables and paths expressions in place of relations. Paths can be seen as complementary with aggregations: where aggregation join pieces together, paths extract them individually.

The following example of a PSPARQL query exhibits in the second and third lines of the WHERE clause, two path expressions made from the indefinite composition (+) and composition (.) operators.

```
SELECT ?X, ?Y
WHERE { ?Z foaf:name ?X.
        ?Z foaf:knows+ . foaf:worksFor ?Y
        ?Y vc:adr . vc:city "Innsbruck". }
```

The above query returns pairs of person and company such that the person indirectly knows someone working in this company and this company is based in New-York. PSPARQL offers other operators such as disjunction (—), Kleene closure (*) and atomic negation (!) but only composition is currently used in the expressive alignment language presented above. Otherwise, all SPARQL is preserved. In [4], it is shown that the complexity of SPARQL is preserved by the extension and provides algorithms for answering PSPARQL queries.

6 Conclusion: putting them together

In summary, we claim that a query language is an adequate means for transforming data according to some alignment. However, the current specification of SPARQL is not powerful enough for supporting this task with expressive alignments which are necessary for carefully describing relations between ontologies. We claim that the combination of SPARQL extensions, namely SPARQL++ and PSPARQL, can serve as a fruitful basis to ground expressive ontology alignments into concrete executable mappings between data RDF graphs adhering to different, overlapping ontologies.

Thus, in order to implement a complete alignment framework, we propose two things: (1) an implementation of a SPARQL data transformation engine integrating PSPARQL [4] and SPARQL++ [3], and (2), a grounding of an abstract, expressive alignment language to this new PSPARQL++.

The authors are currently working on reconciling their different proposed extensions towards a common prototype.

References

[1] J. Euzenat, F. Scharffe, and A. Zimmermann, "Expressive alignment language and implementation," Knowledge Web Network of Excellence (EU-IST-2004-507482), Tech. Rep. Project Deliverable D2.2.10, 2007.

[2] E. Prud'hommeaux and A. S. (eds.), "SPARQL query language for RDF," Nov. 2007, w3C Proposed Recommendation, available at http://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/.

[3] A. Polleres, F. Scharffe, and R. Schindlauer, "SPARQL++ for mapping between RDF vocabularies," in *OTM 2007, Part I : Proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007)*, ser. Lecture Notes in Computer Science, vol. 4803. Vilamoura, Algarve, Portugal: Springer, Nov. 2007, pp. 878–896. [Online]. Available: http://www.polleres.net/publications/poll-etal-2007.pdf

[4] F. Alkhateeb, J.-F. Baget, and J. Euzenat, "Extending SPARQL with regular expression patterns," Institut National de Recherche en Informatique et Automatique (INRIA), Tech. Rep. 6191, May 2007.

[5] D. Brickley and L. Miller, "FOAF vocabulary specification," Jul. 2005, http://xmlns.com/foaf/0.1/.

[6] J. Euzenat, "An API for ontology alignment," in *Proc. 3rd international semantic web conference, Hiroshima (JP)*, 2004, pp. 698–712.

[7] J. Euzenat and P. Shvaiko, *Ontology matching*. Springer, 2007.

[8] M. Klein, "Combining and relating ontologies: an analysis of problems and solutions," in *Workshop on Ontologies and Information Sharing*, 2001.

[9] F. Scharffe, J. Euzenat, C. Le Duc, A. Mocan, and P. Schvaiko, "Analysis of knowledge transformation and merging techniques and implementations," Knowledge Web Network of Excellence (EU-IST-2004-507482), Tech. Rep. Project Deliverable D2.2.7, 2007.

[10] F. Scharffe, J. Euzenat, Y. Ding, and D. Fensel, "Correspondence patterns for ontology mediation," in *Proceedings of the Ontology Matching Workshop at ISWC*, ISWC. Busan, Korea: CEUR, November 2007.

[11] R. Iannella, "Representing vCard objects in RDF/XML," Feb. 2001, w3C Note, available at http://www.w3.org/TR/vcard-rdf.

[12] E. Dumbill, "DOAP: Description of a project," http://xam.de/ns/os/.

[13] M. Völkel, "RDF (open source) software vocabulary," http://xam.de/ns/os/.