SPARQL Extensions for Processing Alignments

Jérôme Euzenat, *INRIA and Laboratoire d'Informatique de Grenoble*
Axel Polleres, *National University of Ireland, Galway*
François Scharffe, *University of Innsbruck*

Heterogeneity between ontologies is often handled by establishing correspondences between the ontologies' entities and transforming data according to these correspondences, whether for integrating heterogeneous data sources or exchanging messages between services. Relations between aligned entities can be very complex, so we developed an alignment language for expressing such complexities.[1] The language is independent of concrete knowledge-representation and processing languages, but transforming concrete data requires processing the correspondences expressed in the alignment language. In particular, it requires translating the source ontology's data instances to instances of the target ontology.

We expect this scenario to become more common as an increasing number of ontologies and data are developed and published on the Web using the Resource Description Framework (RDF). A query language is a natural choice for translating data because it allows both data extraction and data transformation. Hence, while RDF, RDF Schema, and the Web Ontology Language (OWL) are the standards for describing data and ontologies on the Web, the Simple Protocol and RDF Query Language (SPARQL) seems the natural candidate for expressing and processing ontological correspondences.[2] However, SPARQL in its current version isn't yet powerful enough to cover the full expressivity of the alignment language we developed. We therefore propose combining two recent SPARQL extensions to handle complex alignments:

- SPARQL++ provides aggregates, value-generating built-ins, and (possibly recursive) processing of mappings expressed in SPARQL,[3] and
- PSPARQL provides queries on path expressions by allowing regular expression patterns.[4]

We illustrate our proposal with a data-translation problem between two commonly used ontologies: friend-of-a-friend (FOAF, http://xmlns.com/foaf/0.1) and vCard (www.w3.org/2006/vcard/ns). Both vocabularies describe information about persons and organizations, and both are used extensively on the Web. They cover complementary as well as overlapping aspects of personal information.

Alignment Representation

The *alignment format* is an extensible format for expressing alignments in XML/RDF.[5] It supports interchange between alignments created using ontology-matching algorithms and native representation of simple correspondences between ontological entities. The format is associated with an Alignment API,[6] which is organized around a small set of constructs that let users describe alignments through sets of correspondences, together with related metadata such as the alignment's purpose or the way it was built. Each set of correspondences gives a description of the alignment entities. Figure 1 shows a sample correspondence, expressing the equivalence between a vCard and a FOAF person.

The expressive alignment language we developed extends the alignment format so that it can represent more elaborate correspondences.[1] In particular, it offers

- operators to relate an entity in one ontology to a combination of entities in the other,
- conditions to restrict an entity's scope, and
- transformations for property values such as aggregations, functions, and data-type conversions.

The language provides a high-level description of ontology alignments and a convenient exchange format for matching algorithms, GUIs, and mediation languages.

Grounding

Ontology mediation is a complex mediation process involving two main phases.[7]

First, the alignment is constructed at design time. Typically, ontology engineers use matching algorithms to automatically discover correspondences between ontologies. Graphical mapping interfaces assist the process of refining these correspondences, which eventually involve correspondence patterns.[8] An expressive exchange format must carry the precise meaning of the correspondences.

Second, at runtime, the previously built alignments are executed in a mediation task using a specific target formalism. *Grounding* is the term we use for transforming the alignment expressed in an alignment-representation formalism—such as our expressive alignment language—into the concrete language or algorithm executable on the particular knowledge representation.

When translating instance data in RDF, SPARQL has advantages compared with upcoming rules language standards such as the Rule Interchange Format. SPARQL is declarative and already widely used for querying RDF Web data. This makes SPARQL-based data translation a more natural tool for Semantic Web users than rule-based languages or XML-based extraction techniques at the moment.

We can illustrate the process of grounding to concrete SPARQL expressions for data translations by using the example FOAF-vCard correspondences in Figure 1.

Data Translation Using SPARQL

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

```
CONSTRUCT { result pattern }
FROM dataset
WHERE { graph pattern }
```

Answers to a SPARQL query Q rely on computing the set of possible homomorphisms from Q's basic graph pattern(s) into the RDF graph representing the knowledge base (that is, the dataset). The resulting variable bindings for instantiating the pattern in the WHERE part are then used to construct a new graph by instantiating the result pattern. So, if we want to reuse instance data described in one ontology when our application has been designed for another one, we can use such SPARQL CONSTRUCT queries as a translation mechanism. This is a natural mechanism for writing mapping rules between RDF vocabularies. For instance, the query in Figure 2(a) illustrates a CONSTRUCT query translating a `foaf:Person` into a `vc:VCard`. However, this query only covers the simple concept correspondence. We must complete this correspondence to additionally translate—possibly recursively—a person's properties, such as name, address, or tele-

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

Figure 1. A sample correspondence in the Alignment format.

phone number. CONSTRUCT queries can likewise be used for these subcorrespondences. For example, Figure 2(b) shows how we can extend the Figure 2(a) query to also map names in vCard addresses to FOAF. Figure 2(c) shows a CONSTRUCT statement that models a more complex submapping of the correspondence in Figure 1.

We can then execute these queries on a set of instance data represented in RDF to yield the transformed ontology instances in the target ontology. However, in practical use cases, it turns out that the available constructs in SPARQL still aren't sufficient for an expressive mapping language as required by complex applications.

SPARQL Extensions for Accurate Translation

Three features that SPARQL currently lacks would be particularly useful for processing alignments—namely, aggregate computation, individual generation, and path expressions.

Aggregates. Definition of a Project (DOAP, http://trac.usefulinc.com/doap) is an open source project to create an XML/RDF vocabulary to describe software projects. The DOAP vocabulary contains a revision property—that is, version numbers of released project versions. With an aggregate function MAX, you could map DOAP information into the RDF Open Source Software Vocabulary (http://xam.de/ns/os), which provides a latest-release property, as Figure 3a shows. Other aggregates, such as count, average, or sum, might be needed for complex and complete mappings.

Individual generation. Completing the mapping between vCard and FOAF, if we try mapping from vc:workTel to foaf:phone, we observe that the former is a data-type property and the latter an object property in OWL/RDF. Basically, a mapping needs a conversion function, generating a new

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

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

CONSTRUCT { ?X rdf:type foaf:Person.
  ?X foaf:based_near "Grenoble"^^xsd:string. }
WHERE { ?X rdf:type v:VCard .
      OPTIONAL {?X v:workTel ?PH.}
  OPTIONAL {?X v:workAdr [v:locality ?L]}
  FILTER ( startsWith(?PH, "+33476")
      OR ?L = "Grenoble") }
(c)
```

Figure 2. SPARQL data-translation examples: (a) simple mapping from vCard to the FOAF person concept, (b) simple mapping of that concept's related properties, and (c) combined mapping including combination of properties (Examples omit FROM clauses.)

URI from a literal. Figure 3b shows such a mapping.

SPARQL doesn't allow such value generations at the moment, but they are defined and implemented in a recent extension called SPARQL++.[3] SPARQL++ also provides aggregates. Therefore, we consider SPARQL++ to be a valid basis for such a mapping language, but it doesn't yet address all the issues in complex relations. For example, RDF's blank nodes, which correspond to existential variables in CONSTRUCT queries, involve additional complications, which are discussed in more detail elsewhere.[3]

Paths. Another missing part for expressing complex mappings is path expressions over RDF graph patterns, which aren't expressible in SPARQL. This is fairly surprising for a language that claims to be a graph query language. Here, PSPARQL—another recent extension—SPARQL allows to replace the basic graph patterns—that is, RDF graphs with variables—by graphs with variables and regular path expressions in place of predicates.[4] We can view path expressions as complementary to aggregations: where aggregations join pieces together, path expressions extract them individually.

The example PSPARQL query in Figure 3c exhibits two path expressions in the WHERE clause using the indefinite composition (+) operator, extending SPARQL's

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

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

CONSTRUCT { ?X rdf:type ex:PotentialSalesPerson }
WHERE {?X foaf:knows+ [ foaf:worksFor
        [ vc:adr [ vc:city "Innsbruck"]]] }
(c)
```

Figure 3. Example SPARQL extensions for accurate translation: (a) using aggregates to map DOAP to the Open Source Software Vocabulary, (b) using value-generation in CONSTRUCTs to map vCard telephone numbers (represented as strings) to FOAF telephone numbers (represented as URIs), and (c) a complex mapping using regular path expressions. (Examples omit FROM clauses.)

existing simple bracketed path expressions. This query maps to the class of potential salespersons for Innsbruck, by picking persons that indirectly know someone working in for a company based in Innsbruck.

We demonstrated that a query language is an adequate means for transforming RDF data according to some ontology alignment. However, the current SPARQL specification isn't yet powerful enough for supporting this task with the complex mappings that are necessary for describing alignments between ontologies on the instance level. The combination of SPARQL extensions—SPARQL++ and PSPARQL—can serve as a basis to ground expressive ontology alignments in concrete executable mappings between data RDF graphs adhering to different, overlapping ontologies.

To implement a complete alignment framework, we propose two things: first, an implementation of a SPARQL data transformation engine integrating PSPARQL and SPARQL++ and, second, a grounding of an abstract, expressive alignment language to this new PSPARQL++. We are currently working

on reconciling the different proposed extensions toward a common prototype.

Acknowledgments

The authors thank Faisal Alkhateeb, the main designer of PSPARQL. Alex Polleres' work is supported by the Science Foundation Ireland under the Lion project (SFI/02/CE1/I131) and by the European Commission under project inContext (FP6 IST-034718). A longer version of this essay appeared in the 2008 Proc. 2nd Int'l Conf. on Complex, Intelligent, and Software-Intensive Systems (CISIS 08), IEEE CS Press, pp.913–917.

References

1. J. Euzenat, F. Scharffe, and A. Zimmermann, *Expressive Alignment Language and Implementation*, tech. report D2.2.10, Knowledge Web Network of Excellence (EU-IST-2004-507482), 2007.
2. E. Prud'hommeaux and A. Seaborne, eds., *SPARQL Query Language for RDF*, W3C Recommendation, 2008.
3. A. Polleres, F. Scharffe, and R. Schindlauer, "SPARQL++ for Mapping between RDF Vocabularies," *Proc. 6th Int'l Conf. Ontologies, DataBases, and Applications of Semantics* (ODBASE 07), LNCS 4803, Springer, 2007, pp. 878–896.
4. F. Alkhateeb, J.-F. Baget, and J. Euzenat, *Extending SPARQL with Regular Expression Patterns*, Tech. report 6191, Institut National de Recherche en Informatique et en Automatique (INRIA), 2007.
5. J. Euzenat and P. Shvaiko, *Ontology Matching*, Springer, 2007.
6. J. Euzenat, "An API for Ontology Alignment," *Proc. 3rd Int'l Semantic Web Conf.*, Springer, 2004, pp. 698–712.
7. F. Scharffe et al., *Analysis of Knowledge Transformation and Merging Techniques and Implementations*, tech. Report D2.2.7, Knowledge Web Network of Excellence (EU-IST-2004-507482), 2007.
8. F. Scharffe et al., "Correspondence Patterns for Ontology Mediation," Ontology Matching Workshop, CEUR-WS Vol. 304, 2007, pp. 296-300.

Jérôme Euzenat is a senior research scientist at INRIA Grenoble Rhone-Alpes and Laboratoire d'Informatique de Grenoble. Contact him at jerome.euzenat@inrialpes.fr.

Axel Polleres is a lecturer and project leader at the Digital Enterprise Research Institute at the National Unviersity of Ireland, Galway. Contact him at axel.polleres@deri.org.

François Scharffe is a researcher at the Semantic Technology Institute and PhD candidate at the University of Innsbruck. Contact him at francois.scharffe@uibk.ac.at.

Ontology Matching: Status and Challenges

Konstantinos Kotis,
University of the Aegean
Monika Lanzenberger,
Vienna University of Technology

Ontology matching is a significant Semantic Web research topic and a critical operation in many domains—for example, heterogeneous systems interoperability, data warehouse integration, e-commerce query mediation, and semantic-services discovery. The matching operation takes two (sometimes more) ontologies as input, each consisting of a set of discrete entities and axioms, and outputs the relationships between the entities, such as equivalence or subsumption.

Researchers have proposed many solutions to the ontology-matching problem.[1,2] Although it's a different problem from schema matching, the techniques developed for each of them has benefitted the other. Most research has focused on specific criteria for evaluating and distinguishing between matching approaches according to

- *algorithm input*—for example, entity labels, internal structures, attribute types, and relationships with other entities;
- *matching-process characteristics*—for example, the approximate or exact nature of its computation or the way it interprets input data (syntactic, external, or semantic); and
- *algorithm output*—for example, a one-to-one matching or a one-to-many or many-to-many correspondence.

Other significant distinctions in the output results include the confidence and probability percentages of the mapping results and the kinds of relations provided (equivalence, subsumption, incompatibility, and so on).

Human involvement during the ontology-matching process is usually a trade-off with the results' precision and recall percentages. Fully automated tools are still looking for higher accuracy. International contests such as the Ontology Alignment Evaluation Initiative (http://oaei.ontologymatching.org) provide a forum and benchmarks for this task.[3] Still, both automated and semiautomated tools need to improve their performance. For instance, most of them can't handle large real-domain ontologies such as those in medicine and biology, although the research community has developed more and more realistic testbeds to evaluate tool solutions to the scalability problem.

Beyond ontology-matching methods, tools, and evaluation initiatives, recent efforts have focused on design frameworks for ontology-matching tools, such as AUTOMS-F.[4] Such frameworks let developers use APIs to not only develop ontology-matching methods but also, and more important, synthesize these methods into robust tools for producing more accurate mappings. Beyond this, existing methods need more work to improve their matching quality.

Dilemmas and Critical Questions

The research community's efforts to provide diverse solutions to the matching problem by developing a variety of tools haven't yet generated a dominant set of methods that can serve as a benchmark for designing other matching tools. This might reflect the variety of domain-specific user or application needs. In fact, we conjecture that the community couldn't nominate "the best tool" because so many critical questions arise within a specific problem-solving context.

For example, should the tool be fully or semiautomated? To answer this question satisfactorily requires knowing the extent of human involvement, if any, and how this influences the accuracy of mapping results. How much time must users spend validating mapping results? Can we ensure that mapping results are valid without users' involvement? Ultimately, the questions resolve to what is most critical for their application: investing in human involvement to validate resources or automating the ontology-matching tool? If user interaction is essential, you must provide the means to analyze the matching results and understand the source ontologies' characteristics.

Another important question is whether the tool should provide very high precision and indifferent recall or vice versa. What balance between these parameters does the user's application require? What are the trade-offs? Is there an optimal trade-off, and can users tune the methods to achieve it?

Performance involves another set of questions. Does the application call for a tool that supports high computational complexity and rather slow execution time, or are rapid results more important? What percentage of precision and recall are users willing to sacrifice to speed up the ontol-