RDF/SPARQL Design Pattern for Contextual Metadata

Olivier Corby, Catherine Faron Zucker

RDF/SPARQL Design Pattern for Contextual Metadata

Olivier Corby
INRIA Sophia Antipolis - Méditerranée
FR-06902 Sophia Antipolis cedex
olivier.corby@sophia.inria.fr

Catherine Faron-Zucker
Université de Nice-Sophia Antipolis, I3S,
CNRS FR-06903 Sophia Antipolis cedex
faron@polytech.unice.fr

Abstract

The basic principle of the Semantic Web carried by the RDF *data model is that a collection of* RDF *statements coexist all together and are universally true. However some case study imply contextual relevancy and truth. The* SPARQL *query language is provided with patterns enabling to choose the* RDF *dataset against which a query is executed. This is a first step to handle contextual metadata. Based on it, we present in this paper a design pattern to handle contextual metadata hierarchically organized. This is done by means of a* `subStateOf` *property reifying* RDF *entailment between contextual graphs. The* `subStateOf` *relation is modelled within* RDF *and therefore context hierarchies can be described and queried by means of* SPARQL *queries. We propose a slight syntactic extension to* SPARQL *to facilitate the query of context hierarchies, together with rewriting rules to return to standard* SPARQL.

1 Introduction

A basic principle of the Semantic Web is to make available a collection of RDF annotations of web ressources enabling their semantic retrieval. More complex scenarios implying contextual relevance or trust require more than the RDF data model where a set of statements coexist all together and are universally true.

The SPARQL query language for RDF is provided with named graph patterns enabling to choose the RDF dataset against which a query is executed. This is a first step to handle contextual metadata. It can be used to limit the scope of an RDF statement to the context in which it is relevant to query it. Furthermore, by naming contextualized RDF graphs, they can be themselves associated with RDF metadata, enabling querying on several levels of annotations [7].

In this paper we present an RDF/SPARQL design pattern to represent and query contextual metadata. We propose to organize RDF graphs into a hierarchical structure enabling to represent inclusion of contexts. This is done by means of a `subStateOf` property reifying RDF entailment between contextual graphs. The advantage of defining a `subStateOf` property within the RDF model is that context hierarchies can be described and queried with SPARQL. We propose an extension of SPARQL to lighten the syntax of queries exploiting hierarchies of RDF datasets.

The RDF/SPARQL design pattern and SPARQL extension we present in this paper are implemented in the Corese[1] RDF semantic search engine [2], [3]. CORESE has already proved itself in more than twenty semantic web applications; it is currently involved in several ongoing IST european and french projects, among which the e-WOK_HUB ANR french research project[2] for which the features relative to contextual metadata will be put to work.

In the next section we present SPARQL patterns on named graphs and their use to define the notion of contextual metadata and meta-metadata. We describe in section 3 the design pattern we propose to handle contextual metadata hierarchically organized. Section 4 is dedicated to the extension of SPARQL we propose to simply handle such metadata hierarchies and section 5 to the rewriting rules enabling to return to standard SPARQL. Finally we present some case studies of context hierarchies.

2 SPARQL Patterns on Named Graphs

The SPARQL query language is a W3C candidate recommendation for asking and answering queries against RDF data. It offers capabilities for querying by *graph patterns* and retrieval of solutions is based on graph matching.

A SPARQL query is executed against an RDF dataset which represents a collection of graphs: a default graph which does not have a name, and named graphs identified by URIs. *Patterns on named graphs* enable to query a set of RDF graphs named by their URIs.

For instance, the following SPARQL query asks both for a pattern in an RDF graph of the dataset whose URI is bound

[1] `http://www.inria.fr/acacia/soft/corese`
[2] `http://www.inria.fr/acacia/project/ewok`

to variable `?src` and for a triple pattern in the default graph of the dataset stating that there is an `c:author` property attached to this graph URI bound to `?src`.

```
SELECT * WHERE {
 ?src c:author ?person
 GRAPH ?src {
     ?doc c:author ?person
     ?doc c:date ?date}}
```

The GRAPH keyword is used to match patterns against named graphs in the query's RDF dataset; it can provide a URI to select one graph or use a variable which will range over the URIs of all named graphs in the dataset, as it is the case in our example. A SPARQL query may change the default RDF dataset to be used for matching. A new RDF dataset can be specified by using FROM NAMED clauses: it is the set of graphs referred to in the FROM NAMED clauses and the default graph is the RDF merge of these graphs.

For instance, the following SPARQL query requires to match a graph pattern against two graphs, named `URI1` and `URI2`. In case a solution is retrieved, the name of the matching RDF graph will be provided through the binding of variable `?src1` and `?src2` used by the GRAPH keyword.

```
SELECT * FROM NAMED <URI1> FROM NAMED <URI2>
WHERE {
 graph ?src1 {
    ?doc c:author ?person
    ?doc c:date ?date}
 graph ?src2 {?src1 c:author ?person}}
```

Let us note in these two examples that a SPARQL query can match different parts of its pattern against different graphs of its RDF dataset. Let us then pay special attention to the two parts of the query's pattern in both examples: one of them involves the variable denoting the graph upon which the other is to be matched. The key point here is that graph URIs are resources that can therefore be part of the RDF dataset. Hence it is possible to write and query both metadata and meta-metadata, i.e. metadata about RDF graphs named by URIs. This is the basis to explicitly tie contextual information to RDF annotations in separate RDF (meta-) annotations.

As a first conclusion, named graphs can be used to represent different contexts within which alternative metadata can be described, and these contexts themselves (their URIs) can also be represented into separate named graphs.

3 Context Design Pattern

SPARQL semantics considers an unstructured collection of RDF graphs for datasets. However, in some applications, we would like to model the case where some metadata are considered as universally true and some other are concurrent hypothesis or interpretations that rely on the common metadata. This would be interesting when annotating experiments (e.g. in biology) or analysis (e.g. in geology).

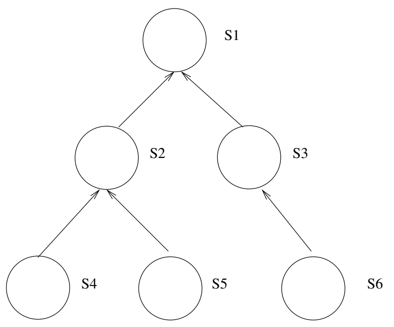

Figure 1. Example of a tree of contexts.

Hence we propose to hierarchically organize RDF datasets, based on RDFS entailment. When considering RDF datasets as contexts, the root of the hierarchy contains the common metadata that is true in any context, any other node of the hierarchy entails it. The other nodes of the hierarchy represent specific contexts; each one recursively inherits the triples of its ancestors and specializes the metadata embedded in the root context with concurrent sets of triples. To avoid the duplication of triples in this hierarchy, each node contains its own triples and inheritance is dynamically computed. This is close to the template of *genus/differentia* in Sowa's theory of conceptual graphs [6] for concept definition: a concept has the graph defining its ancestor as genus and the graph specializing it, and therefore differenciating it from its ancestor, as differentia.

In the Design Pattern we propose for handling contextual metadata, to reason with a context hierarchy, we reify RDFS entailment by a transitive and reflexive relation that is represented in RDF/OWL Lite by a `subStateOf` property between graph URIs. As a result, let us consider a tree structure of contextual metadata as depicted in figure 1. The RDF annotation below represents a snapshot of the hierarchical organization by means of occurrences of the `subStateOf` properties between graph URIs.

```
<rdf:Description rdf:about='&tree;s2'>
   <cos:subStateOf rdf:resource='&tree;s1'/>
</rdf:Description>
```

As stated before, each node contains its own triples and inheritance is dynamically computed. For instance, the context identified by `s2` comprises both the triples in `s2` and in `s1` since a `subStateOf` property holds between them.

We then base upon the SPARQL pattern on named graphs to query such a context hierarchy. Suppose that we want to match a query pattern against the `s2` context. We use graph variables `?s1` and `?s2` that go through all the contexts that `s2` is a substate of (by transitivity), including `s2` itself (by reflexivity) - in our example `s2` and `s1`. In the case of several triples in the query's pattern, one state variable per triple is needed, because each triple may come from a different state, all ancestors of `s2` in the context hierarchy:

```
SELECT * WHERE {
  GRAPH ?s1 {?doc c:author ?person} .
  tree:s2 cos:subStateOf ?s1 .
  GRAPH ?s2 {?doc c:date ?date} .
  tree:s2 cos:subStateOf ?s2}
```

We can see that for each target triple, we must add a `subStateOf` triple which double the size of the query, this notation becomes heavy for large graph patterns. Hence, we propose in the next section a slight syntactic extension to SPARQL to handle context hierarchies.

4 SPARQL Extension for Context

4.1 STATE Keyword

We propose an extension of SPARQL with a STATE keyword whose grammar rule is similar to the GRAPH's one. The example above corresponds to the query shown below.

```
SELECT * WHERE {
  STATE tree:s2 {
     ?doc c:author ?person . ?doc c:date ?date}}
```

4.2 STATE and Variables

The STATE keyword, like the GRAPH keyword, can either provide a URI to select one state or use a state variable so that the query returns the URIs identifying the contexts for which the matching provides a solution.

```
SELECT * WHERE {
  STATE ?state {
     ?doc c:author ?person . ?doc c:date ?date}}
```

4.3 STATE and the FROM Clause

Like the GRAPH keyword, the STATE keyword can be associated with a FROM clause.

```
SELECT * FROM <URI1> WHERE {
   STATE ?state {?doc c:author ?person}}
```

is rewritten as:

```
SELECT * FROM <URI1> WHERE {
   GRAPH ?s {?doc c:author ?person} .
   ?state cos:subStateOf ?s}
```

Hence, the `subStateOf` triples are matched with those of the graph identified by the URI in the FROM clause. It is then possible to select the context hierarchies that we want to query. Furthermore, in order to be compliant with SPARQL, the STATE keyword can also be associated with a FROM NAMED clause. In that case, the hierarchy of contexts is reduced to the specific URIs given in the `from named`.

5 Compiling Patterns with STATE

We present rewriting rules for compiling state into standard SPARQL, using Gilles Kahn's Natural Semantics formalism [4]. In the following, each $?si$ represents a new allocated variable; the left part of the bottom of a rule is the pattern to rewrite, the right part of the bottom is the result of the rewriting and the top of the rule is the condition of the rewriting. A condition describes the rewriting of subparts of the pattern to rewrite; there may be no condition.

$$\frac{state\ S\ \{EXP\} \rightarrow EXP'\ \&\ state\ S\ \{REST\} \rightarrow REST'}{state\ S\ \{EXP\ .\ REST\}\ \rightarrow\ EXP'\ .\ REST'} \quad (1)$$

$$\frac{state\ S\ \{EXP_1\}\ \rightarrow\ EXP_1'\ \&\ state\ S\ \{EXP_2\}\ \rightarrow\ EXP_2'}{state\ S\ \{EXP_1\ optional\ EXP_2\}\ \rightarrow\ EXP_1'\ optional\ EXP_2'} \quad (2)$$

$$\frac{state\ S\ \{EXP_1\}\ \rightarrow\ EXP_1'\ \&\ state\ S\ \{EXP_2\}\ \rightarrow\ EXP_2'}{state\ S\ \{EXP_1\ union\ EXP_2\}\ \rightarrow\ EXP_1'\ union\ EXP_2'} \quad (3)$$

$$\frac{EXP\ \rightarrow\ EXP'}{\begin{array}{c} state\ S\ \{graph\ G\ \{EXP\}\}\ \rightarrow \\ S\ cos:subStateOf\ ?s_i\ .\ graph\ G\ \{EXP'\}\end{array}} \quad (4)$$

$$\frac{state\ S_1\ \{EXP\}\ \rightarrow\ EXP'\ \&\ state\ S_2\ \{EXP'\}\ \rightarrow\ EXP''}{state\ S_2\ \{state\ S_1\ \{EXP\}\}\ \rightarrow\ EXP''} \quad (5)$$

$$\frac{}{\begin{array}{c} state\ S\ \{TRIPLE\}\ \rightarrow \\ S\ cos:subStateOf\ ?s_i\ .\ graph\ ?s_i\ \{TRIPLE\}\end{array}} \quad (6)$$

For instance, let us detail the compiling of the following pattern:

```
STATE ?s {?x c:hasCreated ?doc .
          ?x c:isMemberOf ?org}
```

Rule (1) applies with: `EXP = ?x c:hasCreated ?doc` and `REST = ?x c:isMemberOf ?org`. According to the condition of rule (1), we must rewrite `state ?s {?x c:hasCreated ?doc}` and `state ?s {?x c:isMemberOf ?org}`. Rule (6) applies on these two patterns and generates: `graph ?si {?x c:hasCreated ?doc} . ?s cos:subStateOf ?si` and `graph ?sj {?x c:isMemberOf ?org} . ?s cos:subStateOf ?sj`. Regarding the result of the rewriting in rule (1), the initial pattern is then rewritten as the concatenation of these four patterns.

6 Use Cases of Contexts

It is possible to join two contexts into a subcontext that will inherit triples from both contexts. This enables to test if the union of two contexts contains a contradiction or enables to solve a problem :

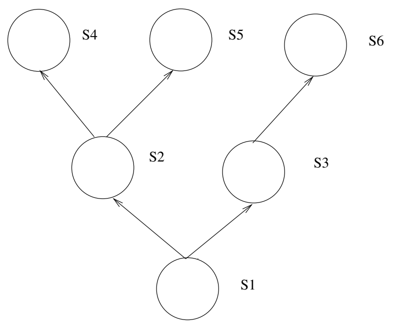

Figure 2. Example of context partonomy.

```
<rdf:Description rdf:about='&tree;s7'>
    <cos:subStateOf rdf:resource='&tree;s5'/>
    <cos:subStateOf rdf:resource='&tree;s6'/>
</rdf:Description>
```

Context Partonomy In the previous part we have shown hierarchies of contexts where a node represents the common knowledge that is recursively inherited by its descendants. It is also possible to consider hierarchies where a node represents the union of its sons. In that case, the root context represents the union of all the triples of the hierarchy. Our state pattern enables to implement this use case as shown in figure 2. The state pattern below goes through all states because `tree:s1` inherits the whole hierarchy that can be considered as a partonomy:

```
SELECT * WHERE {
   STATE tree:s1 { PATTERN }}
```

However, it is not intuitive from a user point of view to describe such a partonomy upside down. Hence, we introduced a second property named `subPartOf` defined as the inverse property of `subStateOf`.

7 Related Work

[1] address the problem of provenance and trust on the web and propose an extension of RDF to handle RDF graphs named by URIs, enabling RDF statements describing RDF graphs. The authors apply their work to Semantic Web publishing where named graphs allow publishers to communicate assertional intent, and to sign their graphs. Graphs are then trusted depending on their content, information about the graph, and the task the user is performing.

[7] use the notion of context to separate statements that refer to different contextual information. They describe a practical solution to explicitly tie contextual information to RDF statements. They identify SPARQL as the query language satisfying their requirements with its patterns on named graphs, however they do not propose any extension of RDF or SPARQL.

8 Conclusion and on going Work

In this paper we have presented an RDF/SPARQL design pattern to represent contextual metadata using a hierarchy of contexts represented as named graphs and ordered by RDFS entailment. We have proposed an operational mechanism for expressing and handling contextual metadata through the extension of SPARQL with a STATE pattern and its compilation into standard SPARQL.

This SPARQL extension has been implemented in the Corese semantic search engine and will be used in the e-WOK_HUB ANR french research project dedicated to geo sciences problems where alternative interpretations of geological data must be represented and reasoned about.

We argue that contextual reasoning is a key issue for intelligent web systems, especially for social networks, intelligent human-web interaction, web trust, etc.. Many applications show us the benefits that would provide the extension of RDF and SPARQL to define, exchange and query identified sources of RDF statements.

Acknowledgement We would like to cite the former Smeci project from INRIA which introduced state trees for rule based reasoning and acknowledge Pierre Haren for these fructuous ideas as well as Rose Dieng for metareasoning on state trees.

References

[1] J. Carroll, C. Bizer, P. Hayes and P. Stickler. Named Graphs, Provenance and Trust. 14th World Wide Web Conference, Chiba, Japan, pages 613–622, 2005.

[2] O. Corby, R. Dieng, C. Faron-Zucker, F. Gandon. Searching the Semantic Web: Approximate Query Processing based on Ontologies, IEEE Intelligent Systems 21(1), 2006.

[3] O. Corby, C. Faron-Zucker. Implementation of SPARQL based on Graph Homomorphism. 15th International Conference on Conceptual Structures, Sheffield, UK, 2007.

[4] G. Kahn. Natural Semantics. RR-0601 INRIA, 1987.

[5] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, W3C Candidate Recommendation, 2007. `http://www.w3.org/TR/rdf-sparql-query`

[6] J.F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984.

[7] H. Stoermer, I. Palmisano, D. Redavid, L. Iannone, P. Bouquet and G. Semeraro. RDF and Contexts: Use of SPARQL and Named Graphs to Achieve Contextualization. 1st Jena User Conference, Bristol, UK, 2006.