XML and Semantic Web Technologies

III. Semantic Web / 3. SPARQL Query Language for RDF

Prof. Dr. Dr. Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

III. Semantic Web / 3. SPARQL Query Language for RDF

1. Basic SPARQL queries

2. More on Queries Returning Tuples

3. Queries Returning RDF

SPARQL Specification

The SPARQL specification consists of the following parts:

- SPARQL Query Language for RDF (WD 2005/04/19)
- SPARQL Query Results XML Format (WD 2005/05/27)
- SPARQL Protocol for RDF (WD 2005/05/27)

as well as a further document on use cases and requirements.

SPARQL is a query language for RDF that

- has a non-XML syntax,
- makes use of (parts of) XPath as expression language,
- makes use of N3 notation

A simple SPARQL query

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <http://www.cgnm.de/rdf/sokrates.rdfs#> .
:Mortal rdf:type rdfs:Class .
:Human rdf:type rdfs:Class .
:Human rdfs:subClassOf :Mortal .
:Sokrates rdf:type :Human .
```

Figure 1: A sample RDF data file (here in N3, but any notation will do).

```
select ?c
where { <http://www.cgnm.de/rdf/sokrates.rdfs#Sokrates>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        ?c }
```

Figure 2: A simple SPARQL query.

Executing SPARQL queries

SPARQL is implemented in ARQ, the Jena query processor (by HP).

sparql –data sokrates.n3 –query sokrates-simple.sq

```
-----------------------------------------------
| c                                           |
===============================================
| <http://www.cgnm.de/rdf/sokrates.rdfs#Human> |
-----------------------------------------------
```

Figure 3: The result of the simple SPARQL query.

SPARQL operates on a RDF graph / set of triples

- explicitly materialized or
- specified implicitely by a an explicitely materialized RDF graph and a set of inference rules.

The ready-to-use commandline tools of ARQ do not support inference yet.

Graph Pattern

A SPARQL query basically consists of a query graph pattern
(plus some additional information).

The simplest form of a graph pattern is a sequence of triples in N3 notation, i.e.,

syntax	meaning
< ⟨*URI*⟩ >	relative URI reference
" ⟨*string*⟩ "	untyped literal
" ⟨*string*⟩ "@@<⟨*URI*⟩>	typed literal
_: ⟨*NCName*⟩	anonymous node name
⟨*integer*⟩	= " ⟨*integer*⟩ " ^^xsd:integer
⟨*double*⟩	= " ⟨*double*⟩ " ^^xsd:double
true, false	= " true " ^^xsd:boolean
? ⟨*NCName*⟩	variable

Graph Pattern Matching

The basic operation is to retrieve all substitutions of the variables s.t. the query graph pattern after substitution is a subgraph of the source graph.

In SPARQL, URIs could be abbreviated either

- by declaring a namespace prefix and using ⟨*QNames*⟩ (as in N3)
- or by declaring a base URI and using relative URIs.

```
prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <http://www.cgnm.de/rdf/sokrates.rdfs#>
select ?c
where { :Sokrates rdf:type ?c }
```

Figure 4: Simple SPARQL query using namespace prefixes.

```
base <http://www.cgnm.de/rdf/sokrates.rdfs>
prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?c
where { <#Sokrates> rdf:type ?c }
```

Figure 5: Simple SPARQL query using namespace prefixes and a base URI.

SPARQL query syntax

```
⟨Query⟩ := ( base ⟨QuotedURIref⟩ )?
   ( prefix ⟨NCNAME_PREFIX⟩? : ⟨QuotedURIref⟩ )*

   ( ask
   | select distinct? (⟨Var⟩+ | ⋆)
   | construct ⟨ConstructTemplate⟩
   | describe ( (⟨Var⟩ | ⟨URI⟩)+ | ⋆) )

   ( from named? ⟨URI⟩ )*
   ( where ⟨GraphPattern⟩ )?
   ( order by ⟨OrderCondition⟩+ )?
   ( limit ⟨INTEGER⟩ )?
   ( offset ⟨INTEGER⟩ )?
```

There is at most one unnamed `from` clause allowed (background graph).
Comments start with `#`.

```
⟨URI⟩ := ⟨QuotedURIref⟩ | ⟨QName⟩
```

Kinds of SPARQL Queries / Outputs

SPARQL supports $3\frac{1}{2}$ different query types:

1. `ask` returns `true`, if there is at least one substitution, else `false`,
2. `select` returns a the set of substitution tuples (like SQL),
3. `construct` returns a RDF graph build from the substition tuples and a template (eventually a subgraph of the original graph or a newly constructed graph),
4. `describe` also returns a RDF graph with some implementation-dependent contents.

III. Semantic Web / 3. SPARQL Query Language for RDF

1. Basic SPARQL queries

2. More on Queries Returning Tuples

3. Queries Returning RDF

A Fresh Example

```
@prefix xs:   <http://www.w3.org/2001/XMLSchema#> .
@prefix : <http://www.cgnm.de/rdf/family#> .
:Anne :age "45"^^xs:integer; :marriedTo :Bert ; :motherOf :Clara, :Dennis .
:Bert :age "49"^^xs:integer; :marriedTo :Anne ; :fatherOf :Clara, :Dennis .
:Clara :age "24"^^xs:integer; :marriedTo :Emil; :motherOf :Fred, :Gisa .
:Dennis :age "22"^^xs:integer.
:Emil :age "27"^^xs:integer; :marriedTo :Clara; :fatherOf :Fred, :Gisa .
:Fred :age "2"^^xs:integer.
:Gisa :age "1"^^xs:integer.
```

Figure 6: A fresh example.

Kinds of Graph Patterns

```
⟨GraphPattern⟩ := { ⟨PatternElement⟩ ( . ⟨PatternElement⟩ )* }

⟨PatternElement⟩ := ⟨Triples⟩
                  | ⟨GraphPattern⟩
                  | optional? ⟨GraphPattern⟩
                  | filter ⟨Expression⟩
                  | ⟨GraphPattern⟩ union ⟨GraphPattern⟩*
                  | graph ( ⟨Var⟩ | ⟨BlankNode⟩ | ⟨URI⟩ ) ⟨GraphPattern⟩
```

Optional Graph Patterns

```
prefix : <http://www.cgnm.de/rdf/family#>
select *
where { ?x :marriedTo ?y }
```

Figure 7: Query for married persons .

```
| x      | y      |
===================
| :Emil  | :Clara |
| :Anne  | :Bert  |
| :Bert  | :Anne  |
| :Clara | :Emil  |
-------------------
```

Figure 8: Result.

```
prefix : <http://www.cgnm.de/rdf/family#>
select *
where { ?x :age ?z
        optional { ?x :marriedTo ?y } }
```

Figure 9: Query for all persons and their spouse (if any).

```
-------------------------
| x       | z  | y      |
=========================
| :Fred   | 2  |        |
| :Emil   | 27 | :Clara |
| :Bert   | 49 | :Anne  |
| :Anne   | 45 | :Bert  |
| :Clara  | 24 | :Emil  |
| :Dennis | 22 |        |
| :Gisa   | 1  |        |
-------------------------
```

Figure 10: Result

Constraints / filter

```
prefix : <http://www.cgnm.de/rdf/family#>
select *
where { ?x :age ?z
      filter ?z >= 18 }
```

Figure 11: Query for all adult persons .

```
----------------
| x         | z  |
================
| :Emil     | 27 |
| :Bert     | 49 |
| :Anne     | 45 |
| :Clara    | 24 |
| :Dennis   | 22 |
----------------
```

Figure 12: Result.

Unions of Graph Patterns ("group patterns") / union

```
prefix : <http://www.cgnm.de/rdf/family#>
select *
where { { ?x :marriedTo ?y }
    union { ?x :age ?z
          filter ?z < 18 } }
```

Figure 13: Query for all married persons and all non-adults.

```
--------------------------
| x        | y        | z      |
==========================
| :Emil    | :Clara   |        |
| :Anne    | :Bert    |        |
| :Bert    | :Anne    |        |
| :Clara   | :Emil    |        |
| :Fred    |          | 2      |
| :Gisa    |          | 1      |
--------------------------
```

Figure 14: Result.

Tuples matching several operand graph patterns of a union graph pattern are contained several times in the result.

Querying Several Sources / `graph`

SPARQL can deal with several sources at once:

- there is one unnamed default source (**background graph**) and
- there are arbitrary many named further sources (**named graphs**) (where names are specified as URIs)

In the `from` clause (and in implementations), the URI names of a source often are used to locate and retrieve the resource.

E.g., in ARQ named sources can be specified by the `-named <URI>` command-line option, where the file or http URI is used to retrieve the source.

Querying Several Sources / Example (1/2)

```
@prefix xs:   <http://www.w3.org/2001/XMLSchema#> .
@prefix : <http://www.cgnm.de/rdf/family-miller#> .
@prefix r: <http://www.cgnm.de/rdf/relatives#> .
:Anne r:age "45"^^xs:integer; r:marriedTo :Bert ; r:motherOf :Clara, :Dennis .
:Bert r:age "49"^^xs:integer; r:marriedTo :Anne ; r:fatherOf :Clara, :Dennis .
:Clara r:age "24"^^xs:integer; r:marriedTo :Emil; r:motherOf :Fred, :Gisa .
:Dennis r:age "22"^^xs:integer.
:Emil r:age "27"^^xs:integer; r:marriedTo :Clara; r:fatherOf :Fred, :Gisa .
:Fred r:age "2"^^xs:integer.
:Gisa r:age "1"^^xs:integer.
```

Figure 15: Data about the Miller family.

```
@prefix xs:   <http://www.w3.org/2001/XMLSchema#> .
@prefix miller: <http://www.cgnm.de/rdf/family-miller#> .
@prefix r: <http://www.cgnm.de/rdf/relatives#> .
@prefix : <http://www.cgnm.de/rdf/family-smith#> .
:Adam r:age "52"^^xs:integer; r:marriedTo :Britta ; r:motherOf :Emil .
:Emil r:age "27"^^xs:integer; r:marriedTo miller:Clara; r:fatherOf miller:Fred, miller:G
```

Figure 16: Data about the Smith family.

Querying Several Sources / Example (2/2)

```
prefix : <http://www.cgnm.de/rdf/relatives#>
prefix miller: <http://www.cgnm.de/rdf/family-miller#>
prefix smith: <http://www.cgnm.de/rdf/family-smith#>
select *
where {
 { ?x :marriedTo ?y }
 union
 { graph <file:///home/lars/lehre/2005-ss-xml/skript/examples/sparql/family-smith.n3>
   { ?x :marriedTo ?y } }}
```

Figure 17: Query for all married people in these two families.

```
| x              | y              |
===================================
| miller:Clara   | miller:Emil    |
| miller:Anne    | miller:Bert    |
| miller:Bert    | miller:Anne    |
| miller:Emil    | miller:Clara   |
| smith:Emil     | miller:Clara   |
| smith:Adam     | smith:Britta   |
-----------------------------------
```

Figure 18: Result.

Sorting / `order` clause

⟨*OrderCondition*⟩ := ⟨*FunctionCall*⟩ | ⟨*Var*⟩
| (`asc` | `desc`) `[` (⟨*FunctionCall*⟩ | ⟨*Var*⟩) `]`

```
prefix : <http://www.cgnm.de/rdf/family#>
select *
where { ?x :age ?z }
order by desc [ ?z ] ?x
```

Figure 19: Query for all persons sorted by descending age and ascending URI.

```
----------------
| x        | z  |
================
| :Bert    | 49 |
| :Anne    | 45 |
| :Emil    | 27 |
| :Clara   | 24 |
| :Dennis  | 22 |
| :Fred    | 2  |
| :Gisa    | 1  |
----------------
```

Figure 20: Result.

Simple Cursor Functionalities / `limit` and `offset` clause

```
prefix : <http://www.cgnm.de/rdf/family#>
select *
where { ?x :age ?z }
order by desc [ ?z ] ?x
limit 2
offset 3
```

Figure 21: Query for a subset of all persons sorted by descending age and ascending URI.

```
----------------
| x       | z  |
================
| :Clara  | 24 |
| :Dennis | 22 |
----------------
```

Figure 22: Result.

III. Semantic Web / 3. SPARQL Query Language for RDF

1. Basic SPARQL queries

2. More on Queries Returning Tuples

3. Queries Returning RDF

Selecting Subgraphs of the Source

It is possible to copy a whole source on basis of its triples.

```
select * where { ?s ?p ?o }
```

Figure 23: Query for all triples in the source.

In the same manner, subsets of triples meeting some conditions can be selected, resulting in a subgraph of the source.

Creating New Triples

New triples can be created by the `construct` query,
a graph template that contains only triples and variables
(but no `optional`, `graph` etc. statements).

⟨*ConstructTemplate*⟩ := `{` ⟨*Triples*⟩ (`.` ⟨*Triples*⟩)* `.`? `}`

The template is instantiated once for each result tuple,
whereat variables are substituted by the values of result tuples.

Example

```
prefix : <http://www.cgnm.de/rdf/family#>
prefix r: <http://www.cgnm.de/rdf/relatives#>
construct {
  ?x r:marriedTo ?spouse .
  ?x r:isMarried true  }
where { ?x :marriedTo ?spouse }

```

Figure 24: Query that recodes the marriedOf property.

```
@prefix r:  <http://www.cgnm.de/rdf/relativ
@prefix xs: <http://www.w3.org/2001/XML
@prefix :   <http://www.cgnm.de/rdf/family

:Anne  r:isMarried   "true"^^xs:boolean ;
       r:marriedTo   :Bert .
:Emil  r:isMarried   "true"^^xs:boolean ;
       r:marriedTo   :Clara .
:Clara r:isMarried   "true"^^xs:boolean ;
       r:marriedTo   :Emil .
:Bert  r:isMarried   "true"^^xs:boolean ;
       r:marriedTo   :Anne .
```

Figure 25: Result.