

Semantic Web

RDF Query Languages

F. Abel, N. Henze, and D. Krause

IVS Semantic Web Group

20.11.2008

1 RDF query languages

- Query RDF

2 TRIPLE

3 Relational query languages

- SPARQL

Why do we need an RDF query language?

We have XML query languages. RDF is XML. Why don't we use XML query languages to query RDF?

- RDF aware: query for RDF triples
 - Subject, Predicate, Object
- No knowledge about the storage structure required
- Ontology support
 - Reasoning
 - support for class hierarchies
- Entailment

Definition (Entailment)

The set A entails the set B if and only if, in every model in which all sentences in A are true, all sentences in B are also true.

Notation: $A \models B$ - a set A entails a set B .

Entailment in RDF queries:

$KB \models Q$

KB: RDF knowledge base

Q: query

A query is true if it is entailed in the RDF source graph.

- Reactive rule query languages
 - e.g. *Algae*, possible actions in *Algae*: *ask* (query), *assert* (insert), or *fwrule* (combination of ask and assert, performs like a database *trigger*)
 - examples at <http://www.w3.org/2004/05/06-Algae/>
- Navigational access query languages
 - e.g. *Versa*, main technique: *traversal*
 - Forward traversal (returns objects):
subject selection - predicate selection -> object selection
 - Backward traversal (returns subjects):
subject selection <- predicate selection - object selection
 - examples at
<http://uche.ogbuji.net/tech/rdf/versa/versa-by-example>
- Pattern-based query languages
 - TRIPLE, Xcerpt (we look in detail at TRIPLE)
- Relational query languages
 - SPARQL, RDQL, RQL, SeRQL (we look in detail at SPARQL)

- Rule & querying language for RDF-annotated Resources
- Transformations: Triple \longrightarrow RDF, RDF \longrightarrow Triple
- Connectives, quantifiers for building logical formulae
- An RDF statement is written in Triple as
subject[predicate -> object]
- Several statements can be abbreviated as molecules
subject [predicate1 -> object1; predicate2 ->
object2; ...]
- Statements can be binded to explicit models:
subject [predicate ->object] @ model
- Triple homepage: triple.semanticweb.org

M. Sintek, S. Decker: TRIPLE - A query, inference, and transformation language for the semantic web. 1st International Semantic Web Conference, 2002

Example (Books)

```
@my:documents {
  my:book1 [
    dc:author->ms[name->"Matt Ruff"];
    dc:title->"G.A.S. Die Trilogie der Stadtwerke";
    dc:subject->Roman;
    dc:subject->Smart_Search;
    dc:subject->New_York].
}

@my:search{
FORALL O, P, V O[P->V] <- O[P->V]@my:documents.
FORALL S, D search_for_subjects(S,D) <- D[dc:subject -> S].
FORALL S, T search_for_titles_and_subjects(S,T) <-
  EXISTS D (D[dc:subject -> S] AND D[dc:title -> T]).
}

//query
FORALL A, B <- search_for_subjects(A,B)@my:search.
FORALL A, B <- search_for_titles_and_subjects(A,B)@my:search.
```

Result:

A = 'New_York', B = my:book1

A = 'Smart_Search', B = my:book1

A = 'Roman', B = my:book1

A = 'New_York', B = 'G.A.S. Die Trilogie der Stadtwerke'

A = 'Smart_Search', B = 'G.A.S. Die Trilogie der Stadtwerke'

A = 'Roman', B = 'G.A.S. Die Trilogie der Stadtwerke'

Example (RDF-Schema in TRIPLE.)

```
// namespace declarations
// rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#".
// rdfs := "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#".

// definition of RDF Schema semantics
FORALL Mdl @rdfschema(Mdl) {
  FORALL O,P,V  O[P->V] <- O[P->V]@Mdl.
  FORALL O,P,V  O[P->V] <-
    EXISTS S (S[rdfs:subPropertyOf->P] AND O[S->V]).
  FORALL O,P,V  O[rdfs:subClassOf->V] <-
    EXISTS W (O[rdfs:subClassOf->W] AND W[rdfs:subClassOf->V]).
  FORALL O,P,V  O[rdfs:subPropertyOf->V] <-
    EXISTS W (O[rdfs:subPropertyOf->W] AND W[rdfs:subPropertyOf->V]).
  FORALL O,T  O[rdf:type->T] <-
    EXISTS S (S[rdfs:subClassOf->T] AND O[rdf:type->S]).
}
```

Example (RDFS cars example)

```
@cars {  
  // xyz := "http://www.w3.org/2000/03/example/vehicles#".  
  xyz:MotorVehicle[rdfs:subClassOf -> rdfs:Resource].  
  xyz:PassengerVehicle[rdfs:subClassOf -> xyz:MotorVehicle].  
  xyz:Truck[rdfs:subClassOf -> xyz:MotorVehicle].  
  xyz:Van[rdfs:subClassOf -> xyz:MotorVehicle].  
  xyz:MiniVan[  
    rdfs:subClassOf -> xyz:Van;  
    rdfs:subClassOf -> xyz:PassengerVehicle].  
}  
  
// query  
FORALL X,Y <- X[rdfs:subClassOf->Y]@rdfschema(cars).
```

Result:

```
X = xyz:'MotorVehicle', Y = rdfs:'Resource'  
X = xyz:'PassengerVehicle', Y = rdfs:'Resource'  
X = xyz:'PassengerVehicle', Y = xyz:'MotorVehicle'  
X = xyz:'Truck', Y = rdfs:'Resource'  
X = xyz:'Truck', Y = xyz:'MotorVehicle'  
X = xyz:'Van', Y = rdfs:'Resource'  
X = xyz:'Van', Y = xyz:'MotorVehicle'  
X = xyz:'MiniVan', Y = rdfs:'Resource'  
X = xyz:'MiniVan', Y = xyz:'MotorVehicle'  
X = xyz:'MiniVan', Y = xyz:'Van'  
X = xyz:'MiniVan', Y = xyz:'PassengerVehicle'
```

- derived from database query language SQL
- similar syntax (select – from – where construct)
- most successful at the moment
- knowledge about SQL simplifies understanding of relational RDF query languages

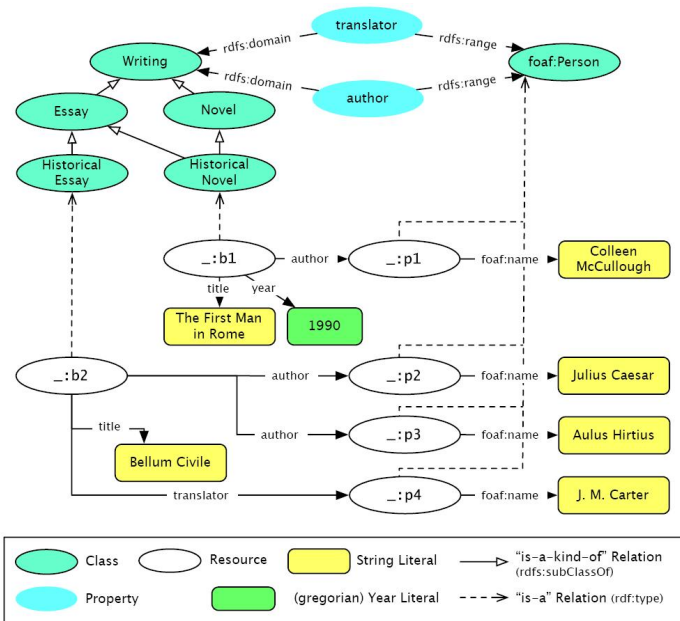
<http://www.w3.org/TR/rdf-sparql-query/>

- April 2006 W3C Candidate Recommendation
- October 2006 W3C Working Draft (step back!)
- June 2007 Candidate Recommendation
- W3C Proposed Recommendation 12 November 2007

Consists out of three specifications

- query language specification
- query results XML format
- data access protocol

Example ontology (<http://example.org/books#>)



RDF data (our Knowledgebase) is stored in TURTLE

- URI abbreviation
- each RDF triple has to be written explicitly

Example (TURTLE)

```
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
...

:Writing    rdfs:type    rdfs:Class.
:translator rdfs:domain  :Writing.
:translator rdfs:range   foaf:Person.
:b1         :year       "1900"^^xsd:NonNegativeInteger.
...
```

Abbreviation possible:

```
:translator rdfs:domain :Writing ;
            rdfs:range  foaf:Person .
```

All queries are made out of two main blocks:

- SELECT orders/structures the output
- WHERE generates the output

Example (A simple query)

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?s foaf:name ?name }
```

Example (Match RDF literals)

```
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
SELECT ?x
WHERE { ?x year 1990.
       ?x title "The first man in Rome"^^xsd:String }
```

FILTER is used to apply value constraints.

Example (Restrict the value of Strings)

```
SELECT ?x
WHERE { ?x foaf:name ?name .
        FILTER regex (?name, "Julius") }
```

Example (Restrict the value of numbers)

```
SELECT ?y
WHERE { ?x year ?year.
        FILTER (?year >1990).
        ?x title ?y }
```

- most functions and operators are taken from XQuery and XPATH
- SPARQL uses XSD Datatypes
- operators:
 - unary: !, +, -, bound(), isLiteral(), lang(), datatype() ...
 - binary: ||, &&, =, !=, i, j, i=, *, /, +, -
 - trinary: regex(string, pattern, flags)

Example

RDF Graph:

```
_:a foaf:name "Alice".
_:a eg:shoeSize "9.5"^^xsd:float .
_:b foaf:name "Bob".
_:b eg:shoeSize "42"^^xsd:integer .
```

Query:

```
SELECT ?name ?shoeSize
WHERE { ?x foaf:name ?name ; eg:shoeSize ?shoeSize .
        FILTER ( datatype(?shoeSize) = xsd:integer ) }
```

OPTIONAL specifies optional parts of the RDF graph

Example (Optional)

```
SELECT ?name ?translator
WHERE {?name title ?title .
      OPTIONAL { ?name translator ?translator }
}
```

Example (Filter optional variables)

```
SELECT ?name ?translator
WHERE {?name title ?title .
      OPTIONAL { ?name translator ?translator .
                 ?translator foaf:name ?tname .
                 FILTER regex (?tname, "Carter")}
}
```

Joining Patterns with UNION

Example (Join names)

```
SELECT ?name
WHERE { {?x title ?name}
        UNION
        {?x foaf:name ?name}
}
```

Do we need a term for Intersect?

SPARQL - Specifying RDF Datasets

How to combine different RDF graphs as source?

- FROM specifies standard RDF graph
 - more than one FROM clause possible → merger RDF graphs
- FROM NAMED adds additional RDF graphs

```
# Graph: http://example.org/bob
_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .

# Graph: http://example.org/alice
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .
```

Example (A simple query with a RDF data source)

```
SELECT ?name
FROM <http://example.org/bob>
WHERE { ?x foaf:name ?name }
```

GRAPH - Use graph source in queries

Example (GRAPH)

```
SELECT ?who ?g ?mbox
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
WHERE
{
  GRAPH ?g { ?who foaf:mbox ?mbox }
}
```

Example (Restrict RDF data source)

```
SELECT ?who ?g ?mbox
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
WHERE
{
  GRAPH <http://example.org/bob> { ?who foaf:mbox ?mbox }
}
```

1 ! - not operator

Example

```
SELECT ?name
WHERE { ?x foaf:name ?name.
        FILTER (!(?name = "Julius Caesar")) }
```

2 Negation as Failure

Example

```
SELECT ?name
WHERE { ?x title ?name.
        OPTIONAL { ?x translator ?y }
        FILTER (!bound(?y)) }
```

- SELECT
 - Returns all or a subset of the bound variables
- CONSTRUCT
 - Constructs an RDF graph by using triple templates
- DESCRIBE
 - Returns an RDF graph that describes the located resources (identified resources and directly referenced resources)

Example

```
DESCRIBE <http://example.org/>
```

```
DESCRIBE ?x
```

```
WHERE { ?x foaf:mbox <mailto:alice@org> }
```

- ASK
 - Answer yes or no whether a query pattern matches or not

- ORDER BY

Example

```
...  
ORDER BY DESC(?name)
```

- Projection
- DISTINCT

Example

```
SELECT DISTINCT ?name WHERE { ?x foaf:name ?name }
```

- OFFSET
- LIMIT

Example

```
SELECT ?name  
WHERE { ?x foaf:name ?name }  
LIMIT 5  
OFFSET 10
```

SPARQL - Constructing an output graph

```
_:a    foaf:givenname    "Alice" .  
_:a    foaf:family_name "Hacker" .  
_:b    foaf:firstname   "Bob" .  
_:b    foaf:surname     "Hacker" .
```

Example (Create vCard from foaf)

```
CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name}  
WHERE     { ?x foaf:firstname ?fname }
```

Example (Blank node)

```
CONSTRUCT { ?x vcard:N _:v .  
            _:v vcard:givenName ?gname .  
            _:v vcard:familyName ?fname }  
WHERE  
{  
  { ?x foaf:firstname ?gname } UNION { ?x foaf:givenname ?gname } .  
  { ?x foaf:surname ?fname } UNION { ?x foaf:family_name ?fname } .  
}
```

- No composing or nesting of queries
- Neither aggregation nor grouping expressible
- Neither recursion nor arbitrary-length traversal operators
 - no transitive-closure type inference
 - no extraction of subgraphs of unknown extent