XML and Semantic Web Technologies

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

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

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

SPARQL Specification

The SPARQL specification consists of the following parts:

- SPARQL Query Language for RDF (REC-2008/01/15)
- SPARQL Query Results XML Format (REC-2008/01/15)
- SPARQL Protocol for RDF (REC 2008/01/15)

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.

Executing SPARQL queries on Windows

- download ARQ, the Jena query processor, from http://jena.sourceforge.net/ARQ/ (current version is 2.7.0).
- add environment variable ARQROOT, pointing to the root path where ARQ is extracted to.
- add %ARQROOT%\bat to system path.

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⟩
   ⟨Var⟩ := ($ | ?) ⟨StringLiteral⟩
```

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.

II. Semantic Web / 3. SPARQL Query Language for 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

$\langle \textit{GraphPattern} \rangle$:= `{` $\langle \textit{PatternElement} \rangle$ (`.` $\langle \textit{PatternElement} \rangle$)* `}`

$\langle \textit{PatternElement} \rangle$:= $\langle \textit{Triples} \rangle$
| $\langle \textit{GraphPattern} \rangle$
| `optional`? $\langle \textit{GraphPattern} \rangle$
| `filter` $\langle \textit{Expression} \rangle$
| $\langle \textit{GraphPattern} \rangle$ `union` $\langle \textit{GraphPattern} \rangle$*
| `graph` ($\langle \textit{Var} \rangle$ | $\langle \textit{BlankNode} \rangle$ | $\langle \textit{URI} \rangle$) $\langle \textit{GraphPattern} \rangle$

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  |          |
--------------------------
```

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.

Constraints / `filter` (2)

Also "‘negative"’ queries are possible.

```
prefix : <http://www.cgnm.de/rdf/family#>

SELECT ?s ?age
WHERE  { ?s :age ?age.
         optional {?s :marriedTo ?x}.
         filter (!bound(?x))
}
```

Figure 13: Query for all non-marries persons.

```
-----------------
| s        | age |
=================
| :Gisa    | 1   |
| :Fred    | 2   |
| :Dennis  | 22  |
-----------------
```

Figure 14: 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 15: 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 16: Result.

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

Restricting returned values (1) / `filter`, `union`

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

Figure 17: Query for all persons, ignoring age is
.

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

Figure 18: 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 explicitly given by the `--namedGraph <URI>` commandline 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 19: 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 20: 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:family-smith.n3>
   { ?x :marriedTo ?y } }
}
```

Figure 21: 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 22: Result.

Specifying Data Sources / `from` clause

`from` may be used to explicitly point to a target resource URI.

```
prefix : <http://www.cgnm.de/rdf/family-miller#>
prefix r: <http://www.cgnm.de/rdf/relatives#>

SELECT *
FROM <file:family-miller.n3>
WHERE {
      ?s r:age ?age. FILTER(?age >30)
}
```

Figure 23: Sample Query for `from`.

```
---------------
| s      | age |
===============
| :Bert | 49  |
| :Anne | 45  |
---------------
```

Figure 24: Result.

Specifying Data Sources / `from named` clause

`from named` may be used to use arbitrarily many background graphs in the query.
URIs are resolved against the namespace declarations in the query.

```
prefix r: <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 ?age ?s
FROM NAMED <file:family-smith.n3>
WHERE {
      graph <file:family-smith.n3> {?s r:age ?age. FILTER(?age >30)}
}
```

Figure 25: Querying the ages of persons in the smith family.

```
---------------------
| age | s           |
=====================
| 52  | smith:Adam |
---------------------
```

Figure 26: Result.

Specifying Data Sources / `from named` clause (2)

`from named` may be used to use arbitrarily many background graphs in the query. URIs are resolved against the namespace declarations in the query.

```
prefix r: <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 ?g ?age ?s
FROM NAMED <file:family-smith.n3>
FROM NAMED <file:family-miller.n3>
WHERE {
      graph ?g  {?s r:age ?age. FILTER(?age >30)}
}
```

Figure 27: Querying the ages of persons in different background graphs.

```
------------------------------------------
| g                  | age | s            |
==========================================
| <family-smith.n3>  | 52  | smith:Adam   |
| <family-miller.n3> | 49  | miller:Bert  |
| <family-miller.n3> | 45  | miller:Anne  |
------------------------------------------
```

Figure 28: Result.

Querying implicit Data Sources

One Drawback of `from`/ `from named` is that background graphs need to be known at query writing time.

This holds only true in very rare situations.

It is (e.g. with ARQ) easily possible, to dynamically add (named) background graphs ("‘context"’s) using the command line.

Querying implicit Data Sources / Example

sparql –namedGraph family-miller.n3 –query from-implicit-named.sq

```
prefix r: <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 ?g ?age ?s
WHERE {
        graph ?g  {?s r:age ?age}.
        FILTER(?age >30)
}
```

Figure 29: Querying the ages of persons in unknown named background graphs.

```
--------------------------------------------
| g                   | age | s             |
============================================
| <family-miller.n3> | 49  | miller:Bert |
| <family-miller.n3> | 45  | miller:Anne |
--------------------------------------------
```

Figure 30: Result.

Querying implicit Data Sources / Example (2)

sparql –namedGraph family-miller.n3 –namedGraph family-smith.n3 –query from-implicit-named.sq

```
prefix r: <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 ?g ?age ?s
WHERE {
        graph ?g  {?s r:age ?age}.
        FILTER(?age >30)
}
```

Figure 31: Querying the ages of persons in unknown named background graphs.

```
------------------------------------------
| g                  | age | s            |
==========================================
| <family-smith.n3>  | 52  | smith:Adam   |
| <family-miller.n3> | 49  | miller:Bert  |
| <family-miller.n3> | 45  | miller:Anne  |
------------------------------------------
```

Figure 32: Result.

Querying implicit Data Sources / Example (3)

But: If triples are loaded as a background graph, they have to be queried from that location.

```
sparql –namedGraph family-miller.n3 –query from-implicit-named.sq
```

```
prefix r: <http://www.cgnm.de/rdf/relatives#>
SELECT ?age ?s
WHERE {
      ?s r:age ?age. FILTER(?age >30)
}
```

Figure 33: Querying the ages of persons in the default background graph.

```
-----------
| age | s |
===========
-----------
```

Figure 34: Result.

Sorting / `order` clause

$\langle$*OrderCondition*$\rangle$:= $\langle$*FunctionCall*$\rangle$ | $\langle$*Var*$\rangle$
| (`asc` | `desc`) `(` ($\langle$*FunctionCall*$\rangle$ | $\langle$*Var*$\rangle$) `)`

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

```

Figure 35: 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 36: Result.

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

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

Figure 37: Query for a subset of all persons sorted by descending age and ascending URI.

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

Figure 38: Result.

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

`ask` - Querying Statement existence

It is possible to ask whether statement(s) in a source graph exist.

```
prefix : <http://www.cgnm.de/rdf/family#>

ask {
        :Emil :fatherOf :Fred.
}
```

Figure 39: Is Emil the father of Fred?

```
Ask => Yes
```

Figure 40: Result.

`ask` - a more complex example

```
prefix : <http://www.cgnm.de/rdf/family#>

ask {
        ?x :fatherOf ?y.
        ?x :age ?age.
        FILTER (?age <= 18).
}
```

Figure 41: A more complex example.

```
Ask => No
```

Figure 42: Result.

Selecting Subgraphs of the Source

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

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

Figure 43: 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).

$\langle \textit{ConstructTemplate} \rangle$:= `{` $\langle \textit{Triples} \rangle$ (`.` $\langle \textit{Triples} \rangle$)* `.`? `}`

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 44: 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 45: Result.

Example (2)

```
prefix ismll: <http://www.ismll.de/rdf/meta#>
prefix r: <http://www.cgnm.de/rdf/relatives#>
prefix miller: <http://www.cgnm.de/rdf/family-miller#>
prefix smith: <http://www.cgnm.de/rdf/family-smith#>

construct {
  ?x r:age ?age .
  ?x ismll:source ?g.
}
where {
       graph ?g {?x r:age ?age.}
}

```

Figure 46: Adding source information.

```
@prefix r:      <http://www.cgnm.de/rdf/rel
@prefix miller:  <http://www.cgnm.de/rdf/fa
@prefix smith:   <http://www.cgnm.de/rdf/f
@prefix ismll:   <http://www.ismll.de/rdf/me

smith:Adam
      r:age         52 ;
      ismll:source  <file:///C:/Users/busche/w

miller:Anne
      r:age         45 ;
      ismll:source  <file:///C:/Users/busche/w

miller:Gisa
      r:age         1 ;
      ismll:source  <file:///C:/Users/busche/w
```

```

miller:Dennis
        r:age         22 ;
        ismll:source  <file:///C:/Users/busche/workspace35_2/XML2009_script/skript/exa

miller:Emil
        r:age         27 ;
        ismll:source  <file:///C:/Users/busche/workspace35_2/XML2009_script/skript/exa

miller:Fred
        r:age         2 ;
        ismll:source  <file:///C:/Users/busche/workspace35_2/XML2009_script/skript/exa

smith:Emil
        r:age         27 ;
        ismll:source  <file:///C:/Users/busche/w

miller:Clara
        r:age         24 ;
        ismll:source  <file:///C:/Users/busche/w

miller:Bert
        r:age         49 ;
        ismll:source  <file:///C:/Users/busche/w
```

Figure 47: Result.

`Describe` - returning all information about selected resources

Describes a (set of) resources in terms of subject, predicate, and object .

```
prefix : <http://www.cgnm.de/rdf/family#>

describe :Anne
```

Figure 48: Query describing Anne.

```
@prefix r:      <http://www.cgnm.de/rdf/relatives#> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema#> .
@prefix :       <http://www.cgnm.de/rdf/family#> .

:Anne
      :age         45 ;
      :marriedTo    :Bert ;
      :motherOf     :Clara ;
      :motherOf     :Dennis .
```

Figure 49: Result.

`Describe`, Example (2)

```
prefix : <http://www.cgnm.de/rdf/family#>

describe ?x ?y {
        ?x :age ?age.
        FILTER (?age >= 18).
        ?x :fatherOf ?y.
}
```

Figure 50: Query describing fathers.

```
@prefix xs:      <http://www.w3.org/2001/X
@prefix :       <http://www.cgnm.de/rdf/fan
:Emil
      :age         27 ;
      :fatherOf     :Fred ;
      :fatherOf     :Gisa ;
      :marriedTo    :Clara .
:Fred
      :age         2 .
:Clara
      :age         24 ;
      :marriedTo    :Emil ;
      :motherOf     :Fred ;
      :motherOf     :Gisa .
:Gisa
      :age         1 .
:Dennis
      :age         22 .
:Bert
      :age         49 ;
```

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

FOAF - Friend of a Friend

The FOAF Project was founded to describe people, and their links, through a (standardized) vocabulatory.

The FOAF Specification consists of its namespace document:

- FOAF Vocabulatory Specification 0.91 (SPEC-2007/11/02)

available at `http://xmlns.com/foaf/spec/`.

FOAF / Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person rdf:ID="me">
        <foaf:name>Lars Schmidt-Thieme</foaf:name>
        <foaf:title>Herr Prof. Dr. Dr.</foaf:title>
        <foaf:givenname>Lars</foaf:givenname>
        <foaf:family_name>Schmidt-Thieme</foaf:family_name>
        <foaf:mbox_sha1sum>3ff97691d04aa172889553373a3467666bf7c100</foa
    </foaf:Person>
</rdf:RDF>
```

Figure 52: Very basic information.

Combining FOAF with our example

```
@prefix miller: <http://www.cgnm.de/rdf/family-miller#> .
@prefix smith: <http://www.cgnm.de/rdf/family-smith#> .
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

smith:Emil foaf:name "Emil".
smith:Emil rdf:type foaf:Person.

miller:Clara foaf:name "Clara".
miller:Clara rdf:type foaf:Person.

smith:Emil foaf:knows miller:Clara.
```

Figure 53: Adding FOAF-Statements to our example.

Simple queries (1)

```
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#>
prefix foaf:      <http://xmlns.com/foaf/0.1/>
prefix rdf:               <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT *
where {
  ?x rdf:type foaf:Person.
}
```

Figure 54: Query for all people.

```
----------------
| x            |
================
| miller:Clara |
| smith:Emil   |
----------------
```

Figure 55: Result.

Simple queries (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#>
prefix foaf:        <http://xmlns.com/foaf/0.1/>
prefix rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT *
where {
  ?x rdf:type foaf:Person.
  graph <file:family-smith.n3>
       { ?x :marriedTo ?y}
}
```

Figure 56: Query for all people being married.

```
-----------------------------
| x          | y            |
=============================
| smith:Emil | miller:Clara |
-----------------------------
```

Figure 57: Result.

Simple queries (3)

```
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#>
prefix foaf:        <http://xmlns.com/foaf/0.1/>
prefix rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT * where {
 ?x rdf:type foaf:Person.
 graph <file:family-smith.n3>
        { ?x :marriedTo ?y}
 graph <file:family-miller.n3>
        { ?y :marriedTo ?z.
          ?y :age ?y_age.} }
```

Figure 58: Symmetric properties? Inferencing?

```
----------------------------------------------------
| x          | y             | z            | y_age |
====================================================
| smith:Emil | miller:Clara  | miller:Emil  | 24    |
----------------------------------------------------
```

Figure 59: Result.

Simple queries (3)

It is impossible with RDF(S) to define properties that define aspects like symmetry, identity, etc.

```
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#>
prefix foaf:     <http://xmlns.com/foaf/0.1/>
prefix rdf:              <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT * where {
  ?x rdf:type foaf:Person.
  ?z rdf:type foaf:Person.
  graph <file:family-smith.n3>  { ?x :marriedTo ?y}
  graph <file:family-miller.n3> { ?y :marriedTo ?z.
                                  ?y :age ?y_age.} }
```

Figure 60: No inferencing.

```
---------------------
| x | z | y | y_age |
=====================
---------------------
```

Figure 61: Result.

References