

Intelligent Systems on the World Wide Web

SeRQL

Lecture Slides
Steffen Staab
Institute for Applied Computer Science and Formal
Description Methods (AIFB)
Karlsruhe University

Acknowledgements to Sesame project,

N3 – Readable notation for RDF

Literals:

- "foo"
- "foo"@en
- "<foo/>"[^]<!http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral>

In the following, we will consider in particular a version necessary for understanding SeRQL

Slide 2

SeRQL – Sesame RDF Query Language

Some of SeRQL's most important features are:

- Graph transformation.
- RDF Schema support.
- XML Schema datatype support.
- Expressive path expression syntax.
- Optional path matching.

Path Expression



- {Person} <foo:worksFor> {Company} <rdf:type> {<foo:ITCompany>}
- Nodes: {node}
- Edges: {node} edge {node}

Multiple path expression:

- {Person} <foo:worksFor> {Company},
{Company} <rdf:type> {<foo:ITCompany>}

Node/Edge is either: Variable, URI, Literal; Node can be unnamed Var

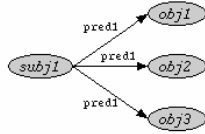
- {Person} <foo:worksFor> {} <rdf:type> {<foo:ITCompany>}
- {Painting} <art:painter_by> {} <art:name> {"Picasso"}
- {<comic:RoadRunner>} SomeRelation {<foo:WillyECoyote>}

Slide 3

Slide 4

Multi-Value Nodes

- {subj1} pred1 {obj1, obj2, obj3}
- ⇔
- {subj1} pred1 {obj1}, {subj1} pred1 {obj2}, {subj1} pred1 {obj3}



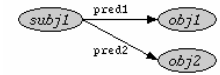
- Also for subject: {subj1, subj2, subj3}
- Both ends: {first} pred1 {middle1, middle2} pred2 {last}



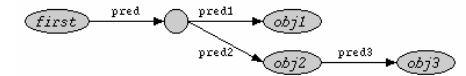
- Constraint: middle1 ≠ middle2

Branches

{subj1} pred1 {obj1}; pred2 {obj2}
 is equivalent to:
 {subj1} pred1 {obj1}, {subj1} pred2 {obj2}



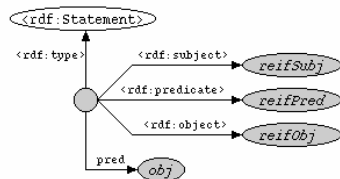
{first} pred {} pred1 {obj1};
 pred2 {obj2} pred3 {obj3}



Reification

{ {reifSubj} reifPred {reifObj} } pred {obj}

{_Statement} <rdf:type> {<rdf:Statement>},
 {_Statement} <rdf:subject> {reifSubj},
 {_Statement} <rdf:predicate> {reifPred},
 {_Statement} <rdf:object> {reifObj},
 {_Statement} pred {obj}



Optional Path Expressions

{Person} <person:name> {Name};
 <person:age> {Age}
 <person:email> {EmailAddress}

Matches only persons *with*
 Email addresses

{Person} <person:name> {Name};
 <person:age> {Age};
 [<person:email> {EmailAddress}]

Matches also persons without
 Email addresses, but if email
 exists this is bound to variable
 EmailAddress

Nested Optional Path Expressions

```
{Document} <foo:title> {Title};  
          [<foo:author> {Author} [<foo:name> {Name}];  
          [<foo:email> {Email}]]
```

Matching for Name of Author is only tried
if matching for Author succeeded, etc.

Slide 9

Select: Construct tuples from variable bindings

- SELECT C FROM {C} <rdf:type> {<rdfs:Class>}
- SELECT * FROM {S} <rdfs:label> {O}
- SELECT O, S FROM {S} <rdfs:label> {O}
- SELECT DISTINCT * FROM {Country1}
<foo:borders> {} <foo:borders> {Country2}

Slide 10

Construct

- CONSTRUCT {Parent} <foo:hasChild> {Child}
FROM {Child} <foo:hasParent> {Parent}
- CONSTRUCT {Artist} <rdf:type> {<art:Painter>};
 <art:hasPainted> {Painting}
FROM {Artist} <rdf:type> {<art:Artist>};
 <art:hasCreated> {Painting}
 <rdf:type> {<art:Painting>}
- CONSTRUCT *
FROM {SUB} <rdfs:subClassOf> {SUPER}
- CONSTRUCT DISTINCT {Artist} <rdf:type> {<art:Painter>}
FROM {Artist} <rdf:type> {<art:Artist>};
 <art:hasCreated> {} <rdf:type> {<art:Painting>}

Slide 11

Filter: Where Clause

Optional clause specifying boolean constraints on vars

Boolean constants

- SELECT * FROM {X} Y {Z} WHERE false

Value (in)equality

- Var = <!foo:bar> is true if the variable Var contains the URI
<!foo:bar>,
• Var1 != Var2 checks whether two variables are not equal.

Numerical comparisons

- XML Schema built-in numerical datatypes are supported
- SELECT Country
FROM {Country} <foo:population> {Population}
WHERE Population < "1000000"^^<xsd:positiveInteger>

Slide 12

Filter: Where Clause

Other constraints:

- isLiteral
- isResource
- Pattern matching – regular expressions
- and, or, not

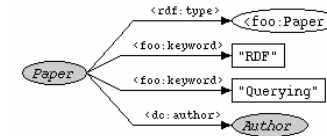
Slide 13

Example query 1

```

SELECT Author, Paper
FROM {Paper} <rdf:type> {<foo:Paper>};
      <foo:keyword> {"RDF", "Querying"};
      <dc:author> {Author}

USING NAMESPACE
  dc = <!http://purl.org/dc/elements/1.0/>,
  foo = <!http://www.foo.org/bar#>
  
```



Slide 14

Example query 2

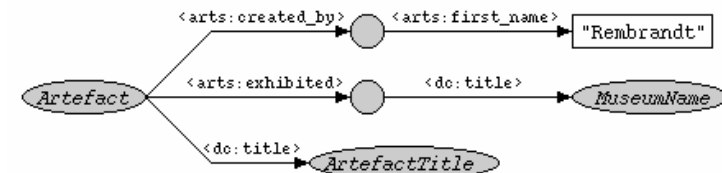
```

SELECT DISTINCT ArtefactTitle, MuseumName
FROM {Artefact} <arts:created_by> {} <arts:first_name>
  {"Rembrandt"},
  {Artefact} <arts:exhibited> {} <dc:title>
  {MuseumName},
  {Artefact} <dc:title> {ArtefactTitle}
WHERE isLiteral(ArtefactTitle)
      AND lang(ArtefactTitle) = "en"
      AND label(ArtefactTitle) LIKE "*night*"

USING NAMESPACE
  dc = <!http://purl.org/dc/elements/1.0/>,
  arts = <!http://www.arts.com/schema.rdf#>
  
```

Slide 15

Example query 2

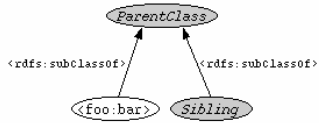


**Where clause
not easily
depictable!**

Slide 16

Example query 3

```
SELECT DISTINCT Sibling
FROM {Sibling, <!foo:bar>} <rdfs:subClassOf>
{ParentClass}
```



Add Ons

- Using Namespace
- Builtin Predicates
 - directSubclassOf
 - directSubpropertyOf
 - directType