SPARQL
Query Language for RDF
Outline

• Motivation
• RDF
• The SPARQL Query Language
• Translating SPARQL to Xquery
• Demo: xQL-to-XQuery
Motivation

• RDF, RDF Schema, OWL are here to stay
  – meta-data is important
  – graph data is important
  – inference is important

• How to process this data?
  – SQL, XQuery etc. not a good match to process graphs
  – Let us develop yet another DSL
  – (and then think about whether big picture makes sense)
Here is where we are now!

Application 1

SPARQL

ARQ SPARQL processor

RDF Triples

Application 2

XQuery

XQuery Engine

XML Collections

Application 3

SQL

MySQL Server

Relational Database
use XQuery?

yes

no
Outline

• Motivation
• **RDF**
• The SPARQL Query Language
• Translating SPARQL to Xquery
• Demo: xQL-to-XQuery
Data Model

• RDF = Resource Description Framework
  – Central component: „resource“
  – Meta data for „resources“, machine-readable
  – Resources can be anything
  – Resources are identified by a URI

• Triplets: Statements about resources
  Subject Predicate Object
  – Subject (Resource): URI
  – Predicate (Property): URI
  – Object: URI or Literal

• Order is not important!
Triplets (graphical)

- Donald works for ETH

- Donald is 36 years old
Triplets (XML)

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="..."
         xmlns:eth="www.ethz.ch">
  <rdf:Description id = "Donald">
    <eth:workFor rdf:resource="#eth"/>
    <eth:age>36</eth:age>
  </rdf:Description>
</rdf:RDF>
```
Triplets (simple)

- Donald works for ETH
  
  eth:Donald eth:workFor eth:ETH

- Donald is 36 years old
  
  eth:Donald eth:age 36
Typed Literals

- Literals can be typed using XML Schema
- Donald is 36 years old; 36 is an integer
  
  `<eth:Donald eth:age "36" xsd:integer>`
- Serialization in XML
  
  ```xml
  ...<rdf:Description rdf:id="Donald">  
      <eth:age rdf:datatype="&xsd;integer">36</eth:age>  
  </eth:age>  
  </rdf:Description>  
  ```
Instances

• Predefined Property rdf:type
  – Semantics: is an instance of

• Example: Donald is a professor
  eth:Donald rdf:type staff:professor
  (Assumption: it is possible to be a professor.
   Talk about that later – in context of classes)
Containers

<rdf:Description rdf:ID="Donald">  
  <f:children>  
    <rdf:Seq>  
      <rdf:li rdf:resource="#Ferdi"/>  
      <rdf:li rdf:resource="#Elli"/>  
      <rdf:li rdf:resource="#Lilli"/>  
      <rdf:li rdf:resource="#Fina"/>  
    </rdf:Seq>  
  </f:children>  
</rdf:Description>
Outline

• Motivation
• RDF
• The SPARQL Query Language
• Translating SPARQL to Xquery
• Demo: xQL-to-XQuery
SPARQL: A Query Language for RDF

General Information
- Name is a recursive acronym
  - SPARQL = SPARQL Protocol and RDF Query Language.
- SPARQL is available as W3C Recommendation since 2008
- Queries may contain triple patterns, conjunctions, disjunctions, and optional patterns

Query Forms
- **SELECT**: return the value of variables which may be bound by a matching query pattern
- **ASK**: return true if a given query matches and false if not
- **CONSTRUCT**: return an RDF graph by substituting the values in given templates
- **DESCRIBE**: return an RDF graph which defines the matching resource
SPARQL: SELECT Queries

Example of an RDF tree: The Periodic System of Elements
SPARQL: SELECT Queries

RDF Tree

```
<table>
<thead>
<tr>
<th>cmp:FeO</th>
<th>cmp:has</th>
<th>pse:Fe</th>
<th>cmp:has</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pse:O</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>oxygen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iron</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

SPARQL Basic Query (1)
Get the ID of all elements which have the name “iron”

```
?element pse:name iron
```
SPARQL: SELECT Queries

Introductory Example

**SELECT:** Result returns variables and their bindings

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>

SELECT ?element
WHERE {
  ?element pse:name "iron".
}
```

"Get the ID of all elements which have the name “iron“"
SPARQL: Patterns

Basic Graph Pattern

- contains a set of triple patterns
- each triple consists of subject, predicate and object
- several triples in one basic graph pattern combined by conjunction

SPARQL code

```sparql
{  ?element chemistry:name ?name.
   ?element chemistry:group 18.
}
```

„all chemical elements which have a name and are in group 18
SPARQL: Patterns

Basic Graph Pattern: Example

```
<table>
<thead>
<tr>
<th>rdf:RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>period</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>hydrogen</td>
</tr>
</tbody>
</table>

ID [H]  ID [He]
```
SPARQL: Patterns

Basic Graph Pattern: Example

“all chemical elements which have a name and are in group 18"
SPARQL: Patterns

Constraints

- FILTER function can be added to a basic graph pattern
- restrict the result depending on Boolean conditions

SPARQL code

```
{ 
  ?element chemistry:name ?name. 
  FILTER regex(?name, "ium$")
}
```

„retrieve elements whose name end in ‘ium’ “
SPARQL: Patterns

Constraints: Example

„retrieve elements whose name end in ‘ium’“
SPARQL: Patterns

**Optional Pattern**

- Goal: supplement the solution with additional information
- Bind variables within OPTIONAL clause to one or many solutions
- Variable is unbound (=empty) if OPTIONAL clause does not match

**SPARQL code**

```sparql
{  
  pattern  
  OPTIONAL { pattern }  
  OPTIONAL { pattern }  
  ...  
}
```
SPARQL: Patterns

Optional Pattern: Example

```
{
  ?element chemistry:name ?name.
  OPTIONAL {
  }
}
```

„elements which have a name and optionally a color“
SPARQL: Patterns

**Alternative Pattern**

- Combination of all solutions
- Total pattern matches if one or several pattern matches
- If more than one alternative found, return all solutions

SPARQL code

```sparql
{ pattern } UNION { pattern } UNION { pattern } ...
```
SPARQL: Patterns

Alternative Pattern: Example

SPARQL code

```
{
  ?element chemistry:group 16.
}
UNION {
}
```

„elements which have a optionally a color or are in group 16“
SPARQL: Patterns

**Group Graph Pattern**

- In a Group Graph Pattern all patterns must match

**SPARQL code**

```sparql
{ pattern }
{ pattern }
{ pattern }
{ pattern }
...
```
SPARQL: Patterns

**Named Graph Pattern**

- Adding additional RDF documents
- Name of the graph may again a variable
- Each query must define a default graph which is active when no named graph is in scope

**SPARQL code**

```sparql
GRAPH ?src
{
  ?compound comp:name ?compoundName.
}
```

„retrieves elements whose name end in ‘ium’“
SPARQL: ASK Queries

Back to Introductory Example

- **ASK**: Test if a query pattern has a solution

SPARQL code:

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>
ASK {
    ?element pse:name "iron".
}
```

"Is there an element which have the name "iron"?"
SPARQL: CONSTRUCT Queries

Returning an RDF Graph

- **CONSTRUCT**: Graph specified by a graph template

SPARQL code

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT {
    <http://example.org/person#Alice> vcard:FN ?name
}
WHERE {
    ?x foaf:name ?name
}
```
SPARQL: DESCRIBE Queries

Data about Resources

- **DESCRIBE**: Returning an RDF graph with data about a resource

**SPARQL code**

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE {
  ?x foaf:name "Alice"
}
```
Outline

• Motivation
• RDF
• The SPARQL Query Language
• Translating SPARQL to XQuery
• Demo: xQL-to-XQuery
Motivation

Solution: We need a “.NET for database systems”
Translating SPARQL to XQuery

Example 1: SPARQL Basic Pattern
Get the ID and color of all elements which have the name “iron”

SPARQL code

```sparql
PREFIX pse: <http://www.daml.org/2003/01/pse#>

SELECT ?element ?color
WHERE {
  ?element pse:name "iron".
}  ORDER BY ?color
```
Translating SPARQL to XQuery

Example 1: SPARQL Basic Pattern

XQuery output

```xquery
let $doc := fn:collection("chemistry")
for $element in $doc/element
for $color in $element/color
where $element/name = "iron"
return
<result>
  <element>{$element/@ID}</element>
  <color>{$element/color}</color>
</result>
```

Translating SPARQL to XQuery

Example 1: SPARQL Basic Pattern

XQuery output

```xquery
let $doc := fn:collection("chemistry")
for $element in $doc/element
for $color in $element/color
where $element/name = "iron"
return
<result>
  <element>{$element/@ID}</element>
  <color>{$element/color}</color>
</result>
```
Translating SPARQL to XQuery

Example 2: SPARQL Graph Pattern
Get the ID of all compounds for the element with the name “iron”

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>
PREFIX cmp: <http://www.daml.org/2003/01/compounds#>

SELECT ?compound
WHERE {
  ?element pse:name "iron" .
  ?compound cmp:has ?element .
}
```
Translating SPARQL to XQuery

Example 2: SPARQL Graph Pattern

XQuery output

```
let $doc := fn:collection("chemistry")
for $elem in $doc/element
for $comp in $doc/compound
where $comp/has/@resource = $elem/@ID 
  and $elem/name = "iron"
return
  <result>
    <compound>{$comp/@ID}</compound>
  </result>
```
Translation Process

- **xQL Lexer**
  - generate tokens
  - token list

- **xQL Parser**
  - grammar check
  - create object tree
  - object tree

- **Error checking**
  - variable binding
  - schema validation
  - object tree

- **Optimizer**
  - input optimizer
  - object tree

- **RDF/CSV Lexer**
  - generate tokens
  - token list

- **RDF/CSV Parser**
  - grammar check
  - create object tree
  - object tree

- **Error checking**
  - variable binding
  - schema validation
  - object tree

- **XQuery Renderer**
  - query languages
  - import data
  - object tree

- **Output Optimizer**
  - output optimizer

- **Generic generation of XQuery code**
SPARQL: A Query Language for RDF

**SPARQL Basic Pattern**

Abstract translation rule for a SPARQL Basic Pattern to XQuery

```xml
∀ subjName (subjVars(patternSPA))
  for $subjName in xqllib:getSubj()
∀ predName (predVars(patternSPA))
  for $predName in xqllib:getPred($subjName)
∀ objName (objVars(patternSPA))
  for $objName in xqllib:getObj($predName)
  ( where
    ∀ constant (constants(patternSPA, subjName, predName, objName))
      $subjName = constant | $predName = constant | $objName = constant
    ∀ filterCondition filters(filterXqu)
      (and)? filterCondition
  )?
return
  <result>
    varName (vars(patternSPA))
    <varName>{data($varName)}</varName>
  </result>
```
Optimizer

What can we optimize?

• do not want to do the work twice
• only optimize what the XQuery engine does not
• optimization dependent on the XQuery engine (here: Zorba)
• exploit knowledge about the data

Example query

```xml
PREFIX : <http://www.daml.org/2003/01/pse#>

SELECT ?color
WHERE
{
  ?element :name "aluminium".
  ?element :compounds ?compounds.
}
```
Optimizer

Example

Translation without optimization

```java
for $subj in $doc_pse
for $obj in $subj/compound
where $subj/name = "aluminium"
return <result>
  <elem>{data($subj/rdf:ID)}</elem>
  <compound>{data($obj)}</compound>
</result>
```

Translation with „predicate pushdown“

```java
for $subj in $doc_pse[name = "aluminium"]
for $obj in $subj/compound
return <result>
  <elem>{data($subj/rdf:ID)}</elem>
  <compound>{data($obj)}</compound>
</result>
```
Evaluating the translation

Comparison to natively executed code

XQuery execution time

xQL to XQuery

XQuery engine

xQl server

? = compare

xQl execution time

config file

xQl query
Measurements: SQL to XQuery

Performance of a simple SQL query (BERLIN Query 6)

**SQL input**

```sql
SELECT nr, label
FROM product
WHERE label LIKE '%xyz%';
```

**XQuery output**

```xquery
let $doc := fn:collection("testDb")
for $product in $doc[name(.) = "product"]
where fn:matches($product/label, "xyz", "i")
return
  <result>
    <nr>{data($product/nr)}</nr>
    <label>{data($product/label)}</label>
  </result>
```
Measurements: SQL to XQuery

Performance of a simple SQL query

![Graph showing the comparison between XQuery and MySQL execution times for varying scaling factors. The graph displays bars for each scaling factor, with XQuery execution times represented by purple bars and MySQL execution times by yellow bars. The x-axis represents the scaling factor, ranging from 10 to 100, and the y-axis represents execution time in seconds, ranging from 0.0001 to 1.0. The graph illustrates how XQuery consistently outperforms MySQL across different scaling factors.]
Measurements: SPARQL to XQuery

Performance of a simple SPARQL query

**SPARQL input**

```sparql
PREFIX : <http://www.daml.org/2003/01/pse#>  

SELECT ?name ?color  
WHERE  
{  
?element :name ?name.  
OPTIONAL  
{  
}  
}  
```
Measurements: SPARQL to XQuery

Performance of a simple SPARQL query

XQuery output

declare namespace default = "http://www.daml.org/2003/01/periodictable/PeriodicTable";
let $doc_pse := fn:collection("pse")
let $GRAPH_0 :=
  for $node_element in $doc_pse[@rdf:ID]
  let $value_element := xqllib:getSubj("pse",$node_element)
  for $value_name in xqllib:getData("pse",$node_element/name)
  where fn:exists($node_element) and fn:exists($value_name)
  return
  <result>
  <var name="element">{$value_element}</var>
  <var name="name">{$value_name}</var>
  </result>
let $GRAPH_1 :=
  for $node_element in $doc_pse[@rdf:ID]
  let $value_element := xqllib:getSubj("pse",$node_element)
  for $value_color in xqllib:getData("pse",$node_element/color)
  where fn:exists($value_color) and fn:exists($node_element)
  return
  <result>
  <var name="color">{$value_color}</var>
  <var name="element">{$value_element}</var>
  </result>
let $GRAPH_2 := xqllib:optional($GRAPH_0,$GRAPH_1,("element"))
return
xqllib:formatSparqlXml($GRAPH_2,("name","color"))
Measurements: SPARQL to XQuery

Performance of a simple SPARQL query
Outline

• Motivation
• RDF
• The SPARQL Query Language
• Translating SPARQL to Xquery
• Demo: xQL-to-XQuery
Demo

Project URL: http://www.xql2xquery.org
Username: ethz
Password: xquery
Conclusion

All languages have their right to exist:

• **SQL**
  – Time-proven language for queries and updates of relational databases

• **SPARQL**
  – Query language for RDF trees

• **XQuery**
  – Turing-complete query and functional programming language
  – Designed to query collections of XML data