Semantic Web

SPARQL

Gerd Gröner, Matthias Thimm

{groener,thimm}@uni-koblenz.de

Institute for Web Science and Technologies (WeST)
University of Koblenz-Landau

July 16, 2013

Agenda for this Week

- SPARQL (2 lectures)
- Ontology Engineering
- Pattern-based Ontology Design
- Semantic Search
- Semantic Web Applications, Closing

Outline

Outline

Outline

SPARQL Protocol

- The SPARQL protocol specifies how queries and query results are exchanged / sent between providing and requesting service.
- Queries and results are sent over the network.

Protocol is specified by two parts

1. SPARQL interface (abstract interface, independent of concrete implementation)
2. bindings of this interface
    - for the query (request)
        - HTTP binding
        - SOAP binding
    - for the result
        - XML result binding

- WSDL 2.0 (Web service description langauge) to describe the SPARQL protocol

SparqlQuery Interface

- SparqlQuery is the only interface of the SPARQL protocol.
- It contails one operation: 'query'

$\rightarrow$ the 'query' operation described by an *in-out message exchange pattern*

- this *in-out message exchange pattern* consists of two messages:
  - 'in'-message
  - 'out'-message

Interface and Operation — the query 'in' message

- Content of an 'in' message (of a SPARQL query operation) is an instance of an XML Schema complex type

Interface and Operation — the query ‘out’ message

- contents of the ‘out’ message of the query operation is an instance of an XML Schema complex type
- the ‘out’ message is composed of either:
  - a *SPARQL result document* (for SELECT and ASK queries), or
  - a serialized RDF graph (e.g., in the RDF/XML syntax) (for CONSTRUCT and DESCRIBE queries)

- Additionally, a specification of return message in case of
  - incorrect (malformed) queries, or
  - refused request

SPARQL Query Binding: HTTP Binding

Query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?who
WHERE { ?book dc:creator ?who }
```

conveyed to a SPARQL query service *http://www.example.com/sparql/*, which is illustrated by the following HTTP trace:

```
GET /sparql/?query=EncodedQuery HTTP/1.1
Host: www.example.com
User-agent: my-sparql-client/0.1
```

SPARQL Query Binding: SOAP Binding

SOAP binding:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
     xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <query-request xmlns="http://www.w3.org/2005/09/
            sparql-protocol-types/#">
     <query>SELECT ?x ?y WHERE {?x isRelatedTo ?y}</query>
   </query-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Result XML Binding

Remember: previously introduced example query:

```
SELECT  ?book  ?who
WHERE  { ?book dc:creator ?who }
```

Result:

```
HTTP/1.1 200 OK
 Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
 Connection: close
 Content-Type: application/sparql-results+xml

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

 <head>
   <variable name="book"/>
   <variable name="who"/>
 </head>
 <results distinct="false" ordered="false">
   <result>
      <binding name="book"><uri>http://www.example/book5</uri></bi
      <binding name="who"><bnode>r29392923r2922</bnode></binding>
   </result>
  ...
</sparql>
```

SPARQL Result XML Binding

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="name"/>
    <variable name="mbox"/>
  </head>

  <results>
    <result>
      <binding name="name"> ... </binding>
      <binding name="mbox"> ... </binding>
    </result>

    <result>
      <binding name="name"> ... </binding>
      <binding name="mbox"> ... </binding>
    </result>
     ...
  </results>
</sparql>
```

Binding Types inside Results

```
<result>
  <binding name="x">
    <bnode>r2</bnode>
  </binding>

  <binding name="hpage">
    <uri>http://work.example.org/bob/</uri>
  </binding>

  <binding name="name">
    <literal xml:lang="en">Bob</literal>
  </binding>

  <binding name="age">
    <literal
   datatype="http://www.w3.org/2001/XMLSchema#integer">30
    </literal>
  </binding>

  <binding name="mbox">
    <uri>mailto:bob@work.example.org</uri>
  </binding>
</result>
```

Outline

Networked Graphs

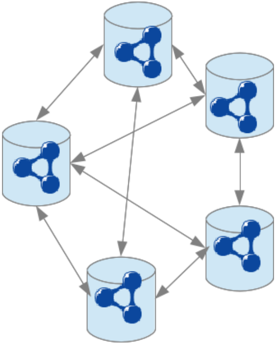
Simon Schenk and **Steffen Staab**
"Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web",
Proceedings of the 17th International World Wide Web Conference, WWW2008, Bejing, China, 2008.

Networked Graphs

- Basic Idea: define RDF graphs
  - *extensionally* by listing statements (RDF triples) or
  - *intensionally* using views
    - possible to have view within another view (recursion)

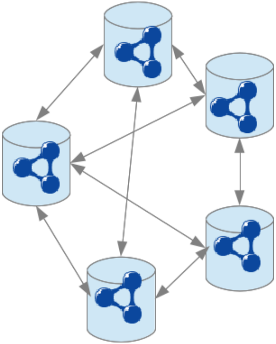
- Integrate with existing Semantic Web infrastructure
- Easy exchange of (networked) graphs
- Use existing data sources (RDF graphs)

Background: The Vision of the Semantic Web



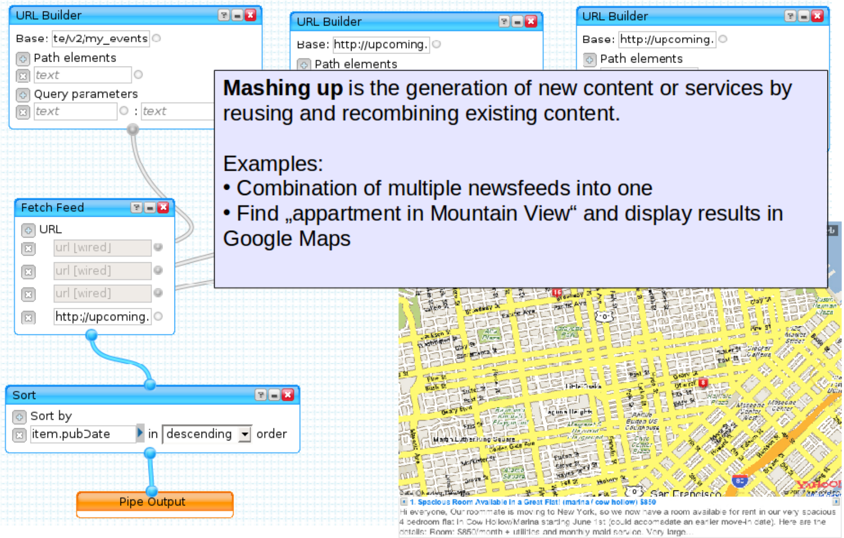
- A Web of Machine understandable data.
- Allowing for Reuse of data.
- A Distributed Infrastructure
- Managed by autonomous agents
- Fundamental technologies:
  - RDF, SPARQL
  - HTTP GET

Background: The Vision of the Semantic Web



- A Web of Machine understandable data.
- Allowing for Reuse of data.
- A Distributed Infrastructure
- Managed by autonomous agents
- Fundamental technologies:
  - RDF, SPARQL
  - HTTP GET

Motivation for Networked Graphs: Mashups



Mashing up is the generation of new content or services by reusing and recombining existing content.

Examples:

- Combination of multiple newsfeeds into one
- Find „appartment in Mountain View“ and display results in Google Maps

Mashups and Semantic Web

Mashups most popular model:

- Hack-and-Hope
- Disadvantages:
  - Screen-Scraping (reading, extracting text from documents (Web pages))
  - No agreement on a data model
- It may be different sometimes:
  - Google Web service
  - Amazon Web service

Semantic Web most popular model:

- Crawl-Integrate-and-Reason
- Disadvantages:
  - Data is **outdated**
  - Data is not declarative, but only has implicit semantics
  - Scalability problems
  - **Access rights:** not all is allowed to be copied / published
  - **Provenance** – data sources can be blurred

Mashups and Semantic Web

Mashups most popular model:

- Hack-and-Hope
- Disadvantages:
  - Screen-Scraping (reading, extracting text from documents (Web pages))
  - No agreement on a data model
- It may be different sometimes:
  - Google Web service
  - Amazon Web service

Semantic Web most popular model:

- Crawl-Integrate-and-Reason
- Disadvantages:
  - Data is **outdated**
  - Data is not declarative, but only has implicit semantics
  - Scalability problems
  - **Access rights:** not all is allowed to be copied / published
  - **Provenance** – data sources can be blurred

Mashups and Semantic Web

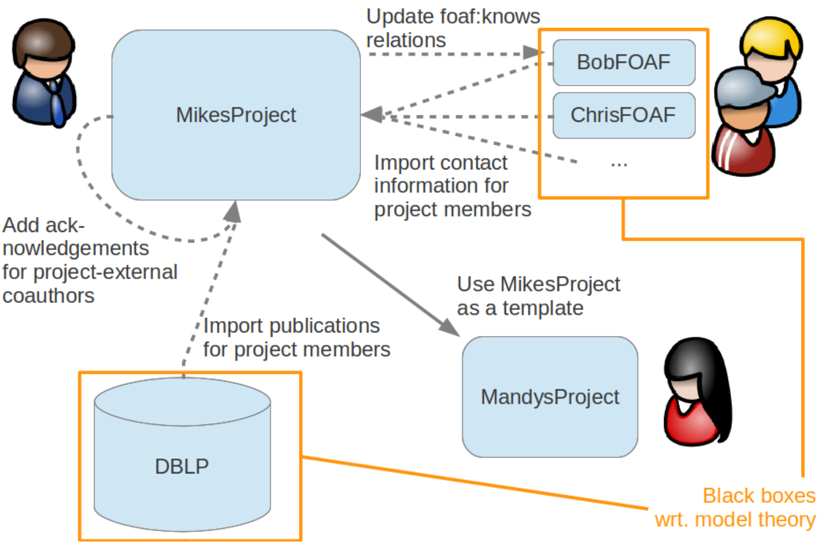
Mashups most popular model:

- Hack-and-Hope
- Disadvantages:
  - Screen-Scraping (reading, extracting text from documents (Web pages))
  - No agreement on a data model
- It may be different sometimes:
  - Google Web service
  - Amazon Web service

Semantic Web most popular model:

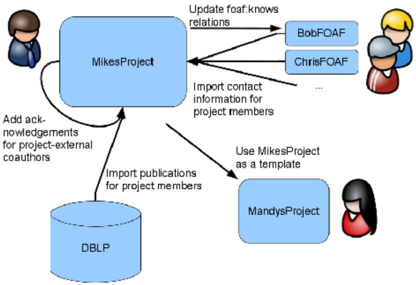
- Crawl-Integrate-and-Reason
- Disadvantages:
  - Data is **outdated**
  - Data is not declarative, but only has implicit semantics
  - Scalability problems
  - **Access rights:** not all is allowed to be copied / published
  - **Provenance** – data sources can be blurred

Networked Graphs — A Concrete Scenario



Objectives in this Scenario

- A Web of Machine understandable data.
- Allowing for Reuse of data.
- A Distributed Infrastructure
- Managed by autonomous agents
- Hardly any assumptions:
  - RDF, SPARQL
  - HTTP GET

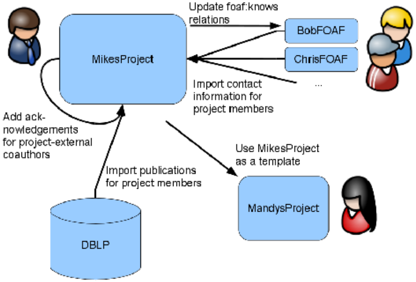


⇒ **Objectives:**

- Distributed Views
- Re-use existing languages/protocols (RDF extension)
- Default-negation
- Easy to learn, easy to exchange

Objectives in this Scenario

- A Web of Machine understandable data.
- Allowing for Reuse of data.
- A Distributed Infrastructure
- Managed by autonomous agents
- Hardly any assumptions:
  - RDF, SPARQL
  - HTTP GET



⇒ **Objectives:**

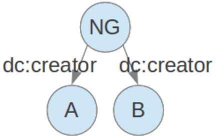
- Distributed Views
- Re-use existing languages/protocols (RDF extension)
- Default-negation
- Easy to learn, easy to exchange

Remember: Named Graphs

Name

Statements

```
:DBLP {
   :NGPaper dc:creator         :Alice.
   :NGPaper dc:creator         :Bob.
}
```




```
:bobFOAF {
   :bobFOAF foaf:primaryTopic   :Bob.
   :Bob     foaf:name           "Bob"^^xsd:String.
   :Bob     foaf:currentProject :K-Space.
   :K-Space foaf:fundedBy       :EU.
   :Bob     foaf:phone          "+49-261-287-2868".
}
```

Literal

URI

Used to group a set of triples.

Remember: SPARQL CONSTRUCT Queries

Build an RDF graph from existing triples (here from **named** graphs):

```
CONSTRUCT {
   ?member   foaf:currentProject  :SemWebProject.
   ?member   foaf:phone   ?phone }
FROM NAMED :bobFOAF
FROM NAMED :chrisFOAF
FROM NAMED  ...
WHERE {
    GRAPH ?foafFile {
          ?foafFile  foaf:primaryTopic ?member
            OPTIONAL {
                  ?member foaf:phone ?phone
                   FILTER (REGEX(?phone, ''^\+49''))
                 }
         }
}
```

Query: "Find the persons described in the FOAF files, their phone numbers, if available. German phone numbers only."

Now: Networked Graphs

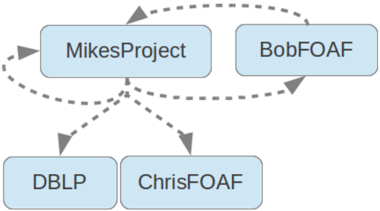
Basic Idea: Define RDF graphs

- extensionally by listing statements or
- intensionally using **views** based on SPARQL

```
:MikesProject {
   :SemWebProject rdf:type foaf:Project.
   ...
   :MikesProject ng:definedBy
     „CONSTRUCT {?member foaf:currentProject ?project.
                 ?member foaf:phone ?phone }
     FROM NAMED :bobFOAF  FROM NAMED :chrisFOAF  FROM NAMED...
     WHERE {
        GRAPH ?foafFile {
        ?foafFile foaf
           OPTIONAL {
              ?member
                FILTER
   :MikesProject ng:def
   ...
}
```

- Upwards compatible with RDF
- Self contained, hence easy exchange
- Use *existing data* through SPARQL
- No need to learn a new language

Independence Set




```
IS(BobFOAF) = { BobFOAF
          MikesProject,
          DBLP,
ChrisFOAF
}
```

Semantics of the independence set:

Iteratively evaluate all views until a fixpoint is reached. Possible problems:

- Termination of recursion
- Non-monotonic negation

Well-founded Semantics

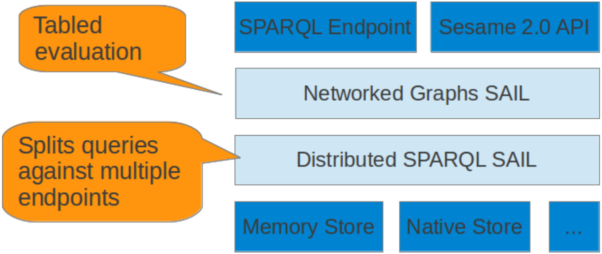
- Models of the Well Founded Semantics
  - closed-world (Here: rules only, graphs may be open worlds.)
  - pessimistic (negate as much as possible)
  - Interpretation of each fact is three-valued (true, false, unknown)
- Usage with RDF In RDF only true statements; false, unknown treated equally. In Networked Graphs false and unknown are used for interim results.

RDF Mashups

- with declarative, dynamic semantics, Mashups can be very powerful!
  - re-use and combine data from several data sources on the Web

Networked Graphs — Architecture

- Based on Sesame 2 (`http://www.openrdf.org/`)
- Open source (`http://west.uni-koblenz.de/Research/systems/NetworkedGraphs`)



Networked Graphs — Conclusion

- Networked Graphs allows for easy
  - formulation,
  - exchange and
  - distributed evaluation

  of rules and views for the Semantic Web using default negation

Networked Graphs integrate well with existing mechanisms and can be evaluated efficiently.

Networked Graphs are eases the re-use of existing data in various applications.

Outline

SPARQL 1.1

... some extensions in SPARQL 1.1:

- Projected expressions
- Aggregationen
- Subqueries
- Negation
- Property path
- Service descriptions
- Update language
- Update protocol
- HTTP RDF update (RESTful)
- Basic federated query

Projected expressions

- SELECT queries are no longer restricted to variables:

```
SELECT (?price * ?amount AS ?totalPrice)
WHERE { ... }
```

Aggregation

- Like in SQL: COUNT, SUM, MIN, MAX, GROUP BY

```
SELECT (MIN(?price) AS ?minPrice) ...
WHERE { ... }
GROUP BY ?article
```

Subqueries

- nested queries
  - multiple queries can be comingled in one query

```
SELECT ?article ?author
WHERE
  ?article ex:author ?author.
  { SELECT ?article
     WHERE { ... ?article ... }
     ORDER BY ...
     LIMIT ...
  }
```

Result of a query is nested in another query.

Negation

- The trick with OPTIONAL and BOUND is no longer needed.

```
SELECT ...
WHERE { ?person a foaf:Person .
        MINUS { ?person foaf:mbox ?email}
}
```

Both graph patterns are evaluated and then the difference is returned.

Negation (2)

- Alternative construct NOT EXISTS

```
SELECT ...
WHERE { ?person a foaf:Person .
   { FILTER NOT EXISTS {
     ?person foaf:mbox ?email
     }
}
```

Both graph patterns are evaluated and then the difference is returned.

Property path

- Excerpt of constructors (let p be an IRI for a predicate, e.g., foaf:knows):
  - ^p inverse path
  - p* multiple times (including 0)
  - p+ multiple times at least 1
  - p? zero or one times
  - p1/p2 sequence
  - $p1|p2$ alternative

Property path (2)

- “regular expression”

```
SELECT  ...
WHERE {
    ?person  foaf:knows+  ?network .
  }
```

Property path (3)

```
SELECT    ...
WHERE {
   ?book  dc:title|rdfs:label  ?displayString  .
 }
```

- "regular expression"
- e.g., alternative

Property path (3)

```
SELECT    ...
WHERE {
   ?book dc:title|rdfs:label ?displayString .
 }
```

- "regular expression"
- e.g., alternative

Property path (3)

- “regular expression”
- e.g., Sequence:
  - “Find the names of people 2 foaf:knows links away.”

```
SELECT ...
WHERE {
    ?x foaf:mbox <mailto:alice@example> .
    ?x foaf:knows/foaf:knows/foaf:name ?name .
}
```

→ without property path?

Property path (3)

- "regular expression"
- e.g., Sequence:
  - "Find the names of people 2 foaf:knows links away."

```
SELECT    ...
WHERE {
    ?x foaf:mbox <mailto:alice@example> .
    ?x foaf:knows/foaf:knows/foaf:name ?name .
 }
```

→ without property path?

Property path (4)

→ equivalent query (without property path):

```
SELECT   ...
WHERE {
    ?x   foaf:mbox <mailto:alice@example> .
    ?x   foaf:knows ?a1 .
    ?a1  foaf:knows ?a2 .
    ?a2  foaf:name ?name .
 }
```

HAVING (over GROUPed sets)

HAVING operates over grouped solution sets, in the same way that FILTER operates over un-grouped sets

```
PREFIX : <http://data.example/>

SELECT (AVG(?size) AS ?asize)
WHERE {
  ?x :size ?size
}
GROUP BY ?x
HAVING(AVG(?size) > 10)
```

→ This will return average sizes, grouped by the subject, but only where the mean size is greater than 10.

HAVING (over GROUPed sets)

HAVING operates over grouped solution sets, in the same way that FILTER operates over un-grouped sets

```
PREFIX : <http://data.example/>

SELECT (AVG(?size) AS ?asize)
WHERE {
  ?x :size ?size
}
GROUP BY ?x
HAVING(AVG(?size) > 10)
```

→ This will return average sizes, grouped by the subject, but only where the mean size is greater than 10.

Summary

- SPARQL
  - Standard query language for RDF
    - SELECT, CONSTRUCT, ASK, DESCRIBE
    - Extensive filters, optional and alternative patterns
  - Protocol for queries and results
  - Based on triples model (subject-predicate-object)
    - No logic inference in language, only in underlying knowledge base
  - Named graphs
    - Separate or specialized knowledge

Learning goals

- Understanding basic and advanced queries using SPARQL.
- Gain knowledge about how to query reified data.

SPARQL endpoints

Name	URL	What's in there?
DBPedia	http://dbpedia.org/sparql	extensive RDF data from Wikipedia
SPARQLer	tttp://sparql.org/sparql.html	General-purpose query endpoint for Web-accessible data
DBLP	http://www4.wiwiss.fu-berlin.de/dblp/snorql/	Bibliographic data from computer science journals and conferences
LinkedMDB	http://data.linkedmdb.org/sparql	Films, actors, directors, writers, producers, etc.
World factbook	http://www4.wiwiss.fu-berlin.de/factbook/snorql/	Country statistics from the CIA World Factbook
bio2rdf	http://bio2rdf.org/sparql	Bioinformatics data from around 40 public databases

Exercise

- querying DBpedia endpoint
  - Find 10 cities (`http://dbpedia.org/ontology/City` that are located in (property `http://dbpedia.org/ontology/country`) Germany (`http://dbpedia.org/resource/Germany`)
  - Find 10 "Populated Places" (URI: `http://dbpedia.org/ontology/PopulatedPlace`) with their names (rdfs:label) in which the names are described in German (i.e., language tag "de").
  - Find 10 people (`http://dbpedia.org/ontology/Person`) with place of birth (`http://dbpedia.org/ontology/birthPlace`) Berlin (`http://dbpedia.org/resource/Berlin`)