

Semantic Web

Hands on: FOAF, and RDF Query Languages

N. Henze

IVS Semantic Web Group

22.11.2007

- 1 FOAF – Friend of a Friend
 - Social Networks
 - Basic ideas
 - Syntax
 - Writing your first FOAF document
 - The FOAF ontology

- 2 RDF query languages
 - Query RDF

- 3 TRIPLE

- 4 Relational query languages
 - SPARQL

- Social Networks - Networks of social interaction
- E.g.
 - in Academics: Co-authoring, advising, serving on communities
 - Movie staff: Directing and acting
 - between people by making phone calls or transmitting infections
 - between countries via trading relations
 - between papers through citation
 - between Web pages by hyperlinking them to other Web pages
 - ...

How to analyze social networks?

- one approach: analyze graph properties:
 - connectivity
 - distance
 - in-degree
 - out-degree
 - ...

Communities everywhere

Based on <http://www-106.ibm.com/developerworks/xml/library/x-foaf.html>

- Augment e-mail filtering by prioritizing mails from trusted colleagues
- Provide assistance to new entrants in a community
- Locate people with interests similar to yours

Based on <http://rdfweb.org/mt/foaflog/archives/2003/07/28/12.46.56/index.html>

- FOAF is the abbreviation for “Friend of a Friend”
- Vision: “FOAF is all about creating and using machine-readable homepages that describe people, the links between them and the things they create and do.”
<http://rdfweb.org/mt/foaflog/archives/2003/07/28/12.46.56/index.html>
- Schema: The FOAF Vocabulary Specification (Life Edition: May 19th, 2005: <http://xmlns.com/foaf/0.1/>)
- Basic ingredients: FOAF document format + Semantic Web idea:
 - people publish information in the FOAF document format
 - by following links between people’s descriptions, machines can identify the social network between all FOAF participants
 - based on this information, additional services become possible (e.g. expert search)

- **The unique name problem again:** When you start entering data about real-world things, like people, into a computer system, you come up against the difficulty of creating a unique name for each thing.
- Here: we are looking for a **decentralized naming strategy**
- What about using mail-addresses similar to the URI approach?
mailto:edd@xml.com lives in the UK
- ... but this is nonsense
- you should write
mailto:edd@xml.com is an e-mail address
- ... a person and that person's email address are not the same thing
- thus correctly you might write
The person with e-mail address mailto:edd@xml.com lives in the UK.

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

  <foaf:Person>
    <foaf:name>Edd Dumbill</foaf:name>
    <foaf:mbox rdf:resource="mailto:edd@xml.com" />
  </foaf:Person>

</rdf:RDF>
```

Example (describing a person)

```
<rdf:RDF
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <foaf:Person>
    <foaf:name>Dr. Stefan Decker</foaf:name>
    <foaf:homepage>
      http://www.isi.edu/~stefan
    </foaf:homepage>
    <foaf:title>Dr.</foaf:title>
    <foaf:depiction>
      http://www.deri.ie/images/members/stefan_decker.jpg
    </foaf:depiction>
    <foaf:family_name>Decker</foaf:family_name>
    <foaf:givenname>Stefan</foaf:givenname>
    <foaf:mbox rdf:resource="stefan.decker@deri.org" />
  </foaf:Person>
</rdf:RDF>
```

- The property **foaf:knows** can be used to link two people together, and it has the basic semantics of some kind of personal acquaintance.

Example (Establishing the acquaintance of two people)

```
<foaf:Person>
  <foaf:name>Edd Dumbill</foaf:name>
  <foaf:mbox rdf:resource="mailto:edd@xml.com" />
  ...
  <foaf:knows>
    <foaf:Person>
      <foaf:mbox
        rdf:resource="mailto:simon@xmlhack.com"/>
      <foaf:name>Simon St.Laurent</foaf:name>
    </foaf:Person>
  </foaf:knows>
</foaf:Person>
```

Property: foaf:knows

Based on <http://xmlns.com/foaf/0.1/>

- knows - A person known by this person (indicating some level of reciprocated interaction between the parties).
- The foaf:knows property relates a foaf:Person to another foaf:Person that he or she knows.
- Weak semantic specification (friend, acquaintance, co-worker?).
- No face-to-face knowledge of other person required
- Doesn't need to be complete, probably never will be complete (Web-like).
- To provide additional levels of representation beyond mere 'knows', FOAF applications can do several things.
 - specialize foaf:knows by adding subproperties.
 - use additional relations from other RDF schemas.

Based on *<http://xmlns.com/foaf/0.1/>*

- Better approach: encode mail addresses
- SHA1 (RFC3174) checksum is (with very high probability) still unique, and mail address unreadable

Example (Encoded mail address)

```
<foaf:Person>
  <foaf:name>Dan Brickley</foaf:name>
  <foaf:mbox_sha1sum>
    241021fb0e6289f92815fc210f9e9137262c252e
  </foaf:mbox_sha1sum>
  <foaf:homepage
    rdf:resource="http://rdfweb.org/people/danbri/" />
  <foaf:img
    rdf:resource=".../danbri/mugshot/danbri-small.jpeg" />
</foaf:Person>
```

- Many FOAF tools use foaf:mbox_sha1sum in preference to exposing mailbox information. This is usually for privacy and SPAM-avoidance reasons. Other relevant techniques include the use of PGP encryption and the use of FOAF-based whitelists for mail filtering.

- Problem: how to find your friend's FOAF file
- Solution: explicit link to RDF file(s) containing referred FOAF descriptions

- 1 Alter the `rdf:RDF` element to add the RDF Schema namespace (add `rdfs`), as follows:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

- 2 You can then add links to other FOAF descriptions by adding one `rdfs:seeAlso` element for each additional file, as follows:

```
<rdfs:seeAlso
  rdf:resource="http://www.example.com/friends.xrdf"/>
<rdfs:seeAlso
  rdf:resource="http://www.ldodds.com/webwho.xrdf"/>
```

Autodiscovery: associating FOAF information with your web pages

Based on <http://rdfweb.org/topic/Autodiscovery>

- Autodiscovery is a means for automatically finding machine-processable resources associated with a particular web page. The discovery can be automated because the resource is linked to in a well-defined way (using the HTML link tag).
- ```
<html>
<head>
<link rel="meta" type="application/rdf+xml"
 title="FOAF" href="foaf.rdf" />
</head>
...
</html>
```
- This seems a bit weird. Would a FOAF file be an "metainformation" for the document? If it is in a person's profile page, it would. However, this autodiscovery isn't used only in profile pages...

- Therefore, we need to indicate the relationship between FOAF files and documents. Here's one solution:

```
<link rel="meta FOAF.MAKER" type="application/rdf+xml"
 title="FOAF" href="foaf.rdf"/>
```

- but then: machines won't know the linktype "FOAF.MAKER", so we have to also include the path to the foaf schema document. The complete "semantic autodiscovery" will be like this. Thus:

```
<link rel="schema.FOAF" href="http://xmlns.com/foaf/0.1/" />
<link rel="meta FOAF.MAKER" type="application/rdf+xml"
 title="FOAF" href="foaf.rdf"/>
```

## FOAF Basics

- [Agent](#)
- [Person](#)
- [name](#)
- [nick](#)
- [title](#)
- [homepage](#)
- [mbox](#)
- [mbox\\_sha1sum](#)
- [img](#)
- [depiction](#) (depicts)
- [surname](#)
- [family\\_name](#)
- [givenname](#)
- [firstName](#)

## Personal Info

- [weblog](#)
- [knows](#)
- [interest](#)
- [currentProject](#)
- [pastProject](#)
- [plan](#)
- [based\\_near](#)
- [workplaceHomepage](#)
- [workInfoHomepage](#)
- [schoolHomepage](#)
- [topic\\_interest](#)
- [publications](#)
- [geekcode](#)
- [myersBriggs](#)
- [dnaChecksum](#)

## Online Accounts / IM

- [OnlineAccount](#)
- [OnlineChatAccount](#)
- [OnlineEcommerceAccount](#)
- [OnlineGamingAccount](#)
- [holdsAccount](#)
- [accountServiceHomepage](#)
- [accountName](#)
- [icqChatID](#)
- [msnChatID](#)
- [aimChatID](#)
- [jabberID](#)
- [yahooChatID](#)

## Projects and Groups

- [Project](#)
- [Organization](#)
- [Group](#)
- [member](#)
- [membershipClass](#)
- [fundedBy](#)
- [theme](#)

## Documents and Images

- [Document](#)
- [Image](#)
- [PersonalProfileDocument](#)
- [topic](#) (page)
- [primaryTopic](#)
- [tipjar](#)
- [sha1](#)
- [made](#) (maker)
- [thumbnail](#)
- [logo](#)

## Example

```
<foaf:Image
 rdf:about=".../mugshot/danbri-small.jpeg"/>
```

## Example

```
<foaf:Person rdf:nodeID="person1">
 <foaf:depiction
 rdf:resource=".../mugshot/danbri-small.jpeg"/>
</foaf:Person>
```

## Example

```
<foaf:Person>
 <foaf:name>Dan Brickley</foaf:name>
 <foaf:depiction>
 <foaf:Image/>
 </foaf:depiction>
</foaf:Person>
```

Examples from <http://rdfweb.org/topic/FoafExamples>

## Example

```
<foaf:Person>
 <foaf:name>Dan Brickley</foaf:name>
 <foaf:depiction>
 <foaf:Image>
 <foaf:sha1>7eff...</foaf:sha1>
 </foaf:Image>
 </foaf:depiction>
</foaf:Person>
```

## Example

```
<foaf:Person>
 <foaf:name>Dan Brickley</foaf:name>
 <foaf:depiction>
 <foaf:Image
 rdf:about="http://.../danbri/mugshot/danbri-small.jpeg">
 <foaf:sha1>7eff...</foaf:sha1>
 </foaf:Image>
 </foaf:depiction>
</foaf:Person>
```

## Example (Dr. Diederich's FOAF file)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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/"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
 xmlns:srw="http://purl.org/net/inkel/rdf/schemas/lang/1.1#">

 <foaf:PersonalProfileDocument rdf:about="">
 <foaf:maker rdf:nodeID="jd"/>
 <foaf:primaryTopic rdf:nodeID="jd"/>
 </foaf:PersonalProfileDocument>

 <foaf:Document rdf:about="">
 <dc:title>FOAF description of Joerg Diederich (mainly professional)
 </dc:title>
 <dc:description>
 Metadata about Joerg Diederich in Friend-Of-A-Friend (FOAF) format
 </dc:description>
 <foaf:topic rdf:nodeID="jd"/><foaf:maker rdf:nodeID="jd"/>
 </foaf:Document>
```

## Example (Dr. Diederich's FOAF file)

```
<foaf:Person rdf:nodeID="jd">
 <srw:masters>en</srw:masters>
 <srw:masters>de</srw:masters>
 <srw:reads>es</srw:reads>
 <srw:reads>sv</srw:reads>
 <foaf:name>Joerg Diederich</foaf:name>
 <foaf:title>Dr.-Ing.</foaf:title>
 <foaf:givenname>Joerg</foaf:givenname>
 <foaf:family_name>Diederich</foaf:family_name>
 <foaf:gender>male</foaf:gender>
 <foaf:mbox_sha1sum>0ff4e..</foaf:mbox_sha1sum>
 <foaf:img
 rdf:resource="http://l3s.de/images/mitarbeiter/joerg_diederich.jpg"/>
 <foaf:depiction
 rdf:resource="http://l3s.de/images/mitarbeiter/joerg_diederich.jpg"/>
 <foaf:schoolHomepage
 rdf:resource="http://www.gymnasium-gross-ilsede.de/">
 <foaf:pastProject rdf:resource="http://www.comcar.de/">
 <foaf:currentProject
 rdf:resource="http://knowledgeweb.semanticweb.org/">
 <foaf:currentProject rdf:resource="http://reverse.net/">
```

## Example (Dr. Diederich's FOAF file)

```
<foaf:publications>
 <foaf:Document>
 <foaf:page rdf:resource="../../../~diederich/publications.php"/>
 </foaf:Document>
</foaf:publications>

<foaf:workInfoHomepage
 rdf:resource="http://www.l3s.de/~diederich/research.php"/>
<foaf:based_near>
 <geo:Point>
 <geo:lat>52.3211</geo:lat>
 <geo:long>9.818</geo:long>
 </geo:Point>
</foaf:based_near>
<foaf:phone rdf:resource="tel:+49-511-762-9749"/>
<foaf:member>
 <foaf:Organization>
 <foaf:homepage rdf:resource="http://www.l3s.de/" />
 <dc:title>L3S Research Center</dc:title>
 </foaf:Organization>
</foaf:member>
<foaf:homepage rdf:resource="http://www.l3s.de/~diederich" />
```

## Example (Dr. Diederich's FOAF file)

```
<foaf:knows>
 <foaf:Person>
 <foaf:name>Rudi Studer</foaf:name>
 <foaf:mbox_sha1sum>be8...</foaf:mbox_sha1sum>
 <rdfs:seeAlso
 rdf:resource="...Personen/viewPersonFOAF/foaf_57.rdf" />
 </foaf:Person>
</foaf:knows>
...
<foaf:knows>
 <foaf:Person>
 <foaf:name>Holger Wache</foaf:name>
 <foaf:mbox_sha1sum>289...</foaf:mbox_sha1sum>
 <rdfs:seeAlso
 rdf:resource="http://www.cs.vu.nl/~holger/Holger%20Wache.rdf"/>
 </foaf:Person>
</foaf:knows>
...
</foaf:Person>
</rdf:RDF>
```

- See Tutorial at  
<http://www.bulat.f0.ru/tutorials/FOAFtut/eng/>
- OR use FOAF-a-Matic  
<http://www.ldodds.com/foaf/foaf-a-matic>
- OR let inspire yourself by the examples at  
<http://rdfweb.org/topic/FoafExamples>
- SHA Codes <http://downlode.org/perl/sha1sum.pl>
- Check you FOAF file with the RDF Validator  
<http://www.w3.org/RDF/Validator/>.
- nice visual feedback can be obtained from the FOAF Explorer  
<http://xml.mfd-consult.dk/foaf/explorer/>
- *seeAlso: exercise to this lecture!*

## foaf:Person

```
<rdfs:Class
 rdf:about="http://xmlns.com/foaf/0.1/Person"
 rdfs:label="Person"
 rdfs:comment="A person."
 vs:term_status="stable">

<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
<rdfs:subClassOf>
 <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Person"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
 <owl:Class rdf:about="http://xmlns.com/foaf/0.1/Agent"/>
</rdfs:subClassOf>

...
<rdfs:isDefinedBy rdf:resource="http://xmlns.com/foaf/0.1/" />
<owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
<owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
<owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Project"/>

</rdfs:Class>
```

## foaf:Agent

```
<rdfs:Class rdf:about="http://xmlns.com/foaf/0.1/Agent"
 vs:term_status="unstable"
 rdfs:label="Agent"
 rdfs:comment="An agent (eg. person, group,...).">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 <rdfs:subClassOf>
 <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Agent-3"/>
 </rdfs:subClassOf>
 <owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
</rdfs:Class>
```

## foaf:firstName

```
<rdf:Property rdf:about="http://xmlns.com/foaf/0.1/firstName"
 vs:term_status="testing"
 rdfs:label="firstName"
 rdfs:comment="The first name of a person.">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
 <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:isDefinedBy rdf:resource="http://xmlns.com/foaf/0.1/">
</rdf:Property>
```

## foaf:gender

```
<rdf:Property rdf:about="http://xmlns.com/foaf/0.1/gender"
 vs:term_status="testing"
 rdfs:label="gender"
 rdfs:comment="The gender of this Agent (typically but not
 necessarily 'male' or 'female').">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
 <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:isDefinedBy rdf:resource="http://xmlns.com/foaf/0.1/" />
 <!-- whatever one's gender is, and we are liberal in leaving room for more
 options than 'male' and 'female', we model this so that an agent has only
 one gender. -->
</rdf:Property>
```

# 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](http://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

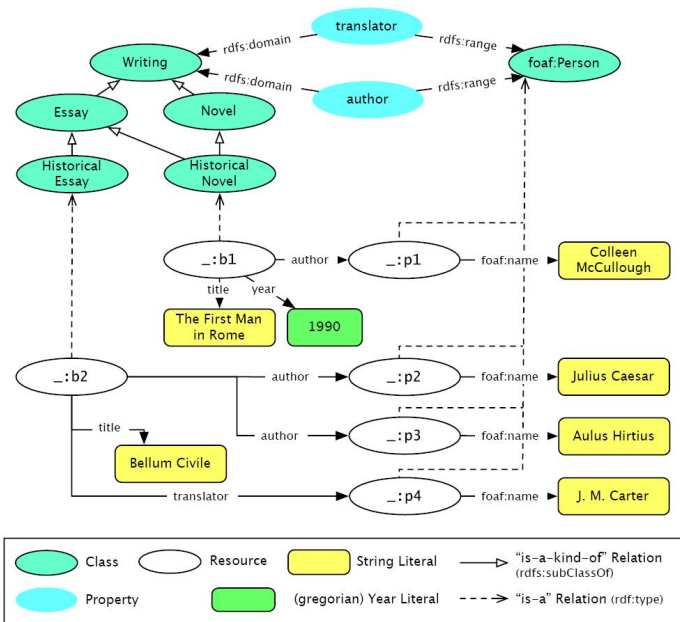
<http://www.w3.org/TR/rdf-sparql-query/>

- 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