Semantic Web, RDF and SPARQL

1st Semantic Web Services Winter Retreat

Ioan Toma
Overview

• From Web to Semantic Web
• RDF
• SPARQL
Why do we need Semantic Web?

FROM WEB TO SEMANTIC WEB
• The current Web represents information using
  – natural language (English, German, Italian,…)
  – graphics, multimedia, page layout
• Humans can process this easily
  – can deduce facts from partial information
  – can create mental associations
  – are used to various sensory information
• Tasks often require to combine data on the Web
  – hotel and travel information may come from different sites
  – searches in different digital libraries
  – etc.
• Again, humans combine these information easily
  – even if different terminology's are used!
However...

• Machines are ignorant!
  – difficult to understand resources on the Web, e.g., a Web page
  – drawing analogies automatically is difficult
  – difficult to combine information automatically
    • is <foo:creator> same as <bar:author>?
    • how to combine different XML hierarchies?
  – …
Serious Problems in

- information finding,
- information extracting,
- information representing,
- information interpreting and
- and information maintaining.

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL
What is the Semantic Web?

- “An extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”
  - Sir Tim Berners-Lee et al., Scientific American, 2001: tinyurl.com/i59p
- “…allowing the Web to reach its full potential…” with far-reaching consequences
- “The next generation of the Web”
What is The Semantic Web?

- The next generation of the WWW
- Information has machine-processable and machine-understandable semantics
- Not a separate Web but an augmentation of the current one
- Ontologies as basic building block
The Semantic Web is about…

- **Web Data Annotation**
  - connecting (syntactic) Web objects, like text chunks, images, … to their semantic meaning (e.g., this image is about Innsbruck, SWSR is taking place in Seefeld)

- **Data Linking on the Web (Web of Data)**
  - global networking of knowledge through URI, RDF, and SPARQL (e.g., connecting my calendar with my rss feeds, my pictures, ...)

- **Data Integration over the Web**
  - Seamless integration of data based on different conceptual models (e.g., integrating data coming from my two favorite book sellers)
Linked Data

• A term coined by Tim Berners-Lee
• It describes HTTP-based Data Access by Reference for the Web
• Current web is changing from hyper**text** links (link documents) to hyper**data** links (linking data)
  – Data are small components of the resources
  – It drills deep to the details of the resources
• Linked data provides a powerful mechanism for meshing disparate and heterogeneous data
Bubbles in May 2007

Over 500M RDF triples
Around 120K RDF links between data sources
Bubbles in April 2008

>2B RDF triples
Around 3M RDF links
Principles

• Linked Data is simply about using the Web to create typed links between data from different sources.

• The principle of Linked data is to:
  – Use the RDF data model to publish structured data on the web
  – Use RDF links to interlink data from different data sources.
  – Use HTTP URIs to identify resource
    • To avoid other URI schemes (URNs or DOIs)
What LOD can bring?

- It will lift current document web up to a data web
- LOD browsers can let you navigate between different data sources by following RDF links.
- It can drill down to the lower granularity of the information
  - allowing you for more fine search on the web
  - meshing up different data through RDF links
  - making the built-on-top application easier
Semantic Web Stack

- User Interface & Applications
  - Trust
    - Proof
      - Unifying Logic
        - Query: SPARQL
          - Ontology: OWL
            - Rule: RIF
              - Crypto
                - Data interchange: RDF
                  - XML
                    - URI/IRI
THE RESOURCE DESCRIPTION FRAMEWORK
• shortly RDF
• RDF provides means to describe resources on the web
• RDF is designed to be read by computers and it is not designed to be displayed on the web.
• RDF is a W3C Recommendation
Principles of RDF 1

- **Resource (identified by URIs)**
  - A URI *identifies* a resource, but does not necessarily *point* to it
  - Correspond to nodes in a graph
  - E.g.:
    - http://www.w3.org/
    - http://example.org/#john
    - http://www.w3.org/1999/02/22-rdf-syntax-ns#Property

- **Properties (identified by URIs)**
  - Correspond to labels of edges in a graph
  - Binary relation between two resources
  - E.g.:
    - http://www.example.org/#hasName
    - http://www.w3.org/1999/02/22-rdf-syntax-ns#type

- **Literals**
  - Concrete data values
  - E.g.:
    - "John Smith", "1", "2006-03-07"
Principles of RDF 2

• Triple data model:
  <subject, predicate, object>
  – **Subject**: Resource or blank node
  – **Predicate**: Property
  – **Object**: Resource, literal or blank node

• Example:
  <ex:john, ex:father-of, ex:bill>

• Labeled, directed graphs
  – **Nodes**: resources, literals
  – **Edges**: properties
Resources

• A resource may be:
  – Web page (e.g. http://www.w3.org)
  – A person (e.g. http://www.fensel.com)
  – A book (e.g. urn:isbn:0-345-33971-1)
  – Anything denoted with a URI!

• A URI is an identifier and not a location on the Web

• RDF allows making statements about resources:
  – http://www.w3.org has the format text/html
  – http://www.fensel.com has first name Dieter
  – urn:isbn:0-345-33971-1 has author Tolkien
Literals

- Plain literals
  - E.g. "any text"
  - Optional language tag, e.g. "Hello, how are you?"@en-GB

- Typed literals
  - E.g. "hello"^^xsd:string, "1"^^xsd:integer
  - Recommended datatypes:
    - XML Schema datatypes

- Only as object of a triple, e.g.:
  <http://example.org/#john>,
  <http://example.org/#hasName>,
  "John Smith"^^xsd:string
Datatypes

• One pre-defined datatype: **rdf:XMLLiteral**
  – Used for embedding XML in RDF

• Recommended datatypes are XML Schema datatypes, e.g.:
  – `xsd:string`
  – `xsd:integer`
  – `xsd:float`
  – `xsd:anyURI`
  – `xsd:boolean`
Blank Nodes

• Blank nodes are nodes without a URI
  – Unnamed resources
  – More complex constructs
• Representation of blank nodes is syntax-dependent
  – Blank node identifier

• For example:

  `<#john>, <#hasName>, _:johnsname`
  `<_:johnsname, <#firstName>, "John"^^xsd:string>`
  `<_:johnsname, <#lastName>, "Smith"^^xsd:string>`
Reification

- Reification: statements about statements

*Mary claims that John’s name is “John Smith”.*

\[
\langle \text{#myStatement}, \text{rdf:type}, \text{rdf:Statement} \rangle
\]
\[
\langle \text{#myStatement}, \text{rdf:subject}, \text{#john} \rangle
\]
\[
\langle \text{#myStatement}, \text{rdf:predicate}, \text{#hasName} \rangle
\]
\[
\langle \text{#myStatement}, \text{rdf:object}, \text{"John Smith"} \rangle
\]
• Reification: statements about statements

*Mary claims that John’s name is “John Smith”.*

\[
<\#myStatement>, \text{rdf:type}, \text{rdf:Statement}
\]
\[
<\#myStatement>, \text{rdf:subject}, <\#john>
\]
\[
<\#myStatement>, \text{rdf:predicate}, <\#hasName>
\]
\[
<\#myStatement>, \text{rdf:object}, \text{“John Smith”}
\]

\[
<\#john>, <\#hasName>, \text{“John Smith”}
\]
• Reification: statements about statements

*Mary claims that John’s name is “John Smith”.*

```xml
<#myStatement>, rdf:type, rdf:Statement
<#myStatement>, rdf:subject, <#john>
<#myStatement>, rdf:predicate, <#hasName>
<#myStatement>, rdf:object, "John Smith"

<#mary>, <#claims>, <#myStatement>
```
RDF Vocabulary

- RDF defines a number of resources and properties
- We have already seen: \texttt{rdf:XMLLiteral}, \texttt{rdf:type}, ...
- RDF vocabulary is defined in the namespace: \url{http://www.w3.org/1999/02/22-rdf-syntax-ns#}

- Classes:
  - \texttt{rdf:Property}, \texttt{rdf:Statement}, \texttt{rdf:XMLLiteral}
  - \texttt{rdf:Seq}, \texttt{rdf:Bag}, \texttt{rdf:Alt}, \texttt{rdf:List}
- Properties:
  - \texttt{rdf:type}, \texttt{rdf:subject}, \texttt{rdf:predicate}, \texttt{rdf:object},
  - \texttt{rdf:first}, \texttt{rdf:rest}, \texttt{rdf:}_n
  - \texttt{rdf:value}
- Resources:
  - \texttt{rdf:nil}
RDF Vocabulary

• Typing using \texttt{rdf:type}:
  \[
  \langle A, \texttt{rdf:type}, B \rangle
  \]
  “\textit{A belongs to class B}”

• All properties belong to class \texttt{rdf:Property}:
  \[
  \langle P, \texttt{rdf:type}, \texttt{rdf:Property} \rangle
  \]
  “\textit{P is a property}”

  \[
  \langle \texttt{rdf:type}, \texttt{rdf:type}, \texttt{rdf:Property} \rangle
  \]
  “\textit{\texttt{rdf:type} is a property}”
• Grouping property values:

“*The lecture is attended by John, Mary and Chris*”  

“*[RDF-Concepts] is edited by Graham and Jeremy (in that order)*”

“*The source code for the application may be found at ftp1.example.org, ftp2.example.org, ftp3.example.org*”
"The lecture is attended by John, Mary and Chris"
“[RDF-Concepts] is edited by Graham and Jeremy \textit{(in that order)}”
“The source code for the application may be found at ftp1.example.org, ftp2.example.org, ftp3.example.org”
RDF Containers 2

- Three types of containers:
  - `rdf:Bag` - unordered set of items
  - `rdf:Seq` - ordered set of items
  - `rdf:Alt` - set of alternatives
- Every container has a triple declaring the `rdf:type`

- Items in the container are denoted with
  - `rdf:_1`, `rdf:_2`, ..., `rdf:_n`
RDF Containers 2

• Three types of containers:
  – rdf:Bag - unordered set of items
  – rdf:Seq - ordered set of items
  – rdf:Alt - set of alternatives

• Every container has a triple declaring the rdf:type

• Items in the container are denoted with
  – rdf:_1, rdf:_2, . . . , rdf:_n

• Limitations:
  – Semantics of the container is up to the application
  – What about closed sets?
    • How do we know whether Graham and Jeremy are the only editors of [RDF-Concepts]?
“[RDF-Concepts] is edited by Graham and Jeremy (in that order) and nobody else”
Serializing RDF for the Web
  - XML as standardized interchange format:
    - Namespaces (e.g. rdf:type, xsd:integer, ex:john)
    - Encoding (e.g. UTF8, iso-8859-1)
    - XML Schema (e.g. datatypes)

Reuse of existing XML tools:
  - Syntax checking (i.e. schema validation)
  - Transformation (via XSLT)
    - Different RDF representation
    - Layout (XHTML)
    - Different XML-based formats

Parsing and in-memory representation/manipulation (DOM/SAX)

...
<#john, #hasName, "John">
<#john, #marriedTo, #mary>

<!ENTITY ex "http://example.org/#">

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex="http://example.org#">
   <rdf:Description rdf:about="http://example.org/#john">
     <ex:hasName>John</ex:hasName>
     <ex:marriedTo rdf:resource="ex:mary"/>
   </rdf:Description>
</rdf:RDF>
<!ENTITY ex "http://example.org/"/>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="http://example.org#">
    <rdf:Description rdf:about="&ex;lecture">
        <ex:isAttendedBy>
            <rdf:Bag>
                <rdf:li rdf:resource="&ex;John"/>
                <rdf:li rdf:resource="&ex;Mary"/>
                <rdf:li rdf:resource="&ex;Chris"/>
            </rdf:Bag>
        </ex:isAttendedBy>
    </rdf:Description>
</rdf:RDF>
Semantic Web Stack

User Interface & Applications

Trust

Proof

Unifying Logic

Query: SPARQL

Ontology: OWL

Rule: RIF

RDFS

Data interchange: RDF

XML

URI/IRI

Crypto
How to query RDF data

SPARQL
Querying RDF

• SPARQL
  – RDF Query language
  – Uses SQL-like syntax

• Example:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?title
FROM <http://example.org/library>
```
SPARQL Queries

PREFIX dc: http://purl.org/dc/elements/1.1
SELECT ?title
FROM <http://example.org/library>

- PREFIX
  - Prefix mechanism for abbreviating URIs
- SELECT
  - Identifies the variables to be returned in the query answer
    - SELECT DISTINCT
    - SELECT REDUCED
- FROM
  - Name of the graph to be queried
    - FROM NAMED
- WHERE
  - Query pattern as a list of triple patterns
- LIMIT
- OFFSET
- ORDER BY
Example RDF Graph

```xml
<http://example.org/#john> <http://.../vcard-rdf/3.0#FN> "John Smith"

<http://example.org/#john> <http://.../vcard-rdf/3.0#N> _:X1
   _:X1 <http://.../vcard-rdf/3.0#Given> "John"
   _:X1 <http://.../vcard-rdf/3.0#Family> "Smith"

<http://example.org/#john> <http://example.org/#hasAge> "32"

<http://example.org/#john> <http://example.org/#marriedTo>
   <#mary>

<http://example.org/#mary> <http://.../vcard-rdf/3.0#FN> "Mary Smith"

<http://example.org/#mary> <http://.../vcard-rdf/3.0#N> _:X2
   _:X2 <http://.../vcard-rdf/3.0#Given> "Mary"
   _:X2 <http://.../vcard-rdf/3.0#Family> "Smith"

<http://example.org/#mary> <http://example.org/#hasAge> "29"
```
Query forms:

- **SELECT**
  - returns all, or a subset of the variables bound in a query pattern match
  - formats: XML or RDF/XML

- **CONSTRUCT**
  - returns an RDF graph constructed by substituting variables in a set of triple templates

- **DESCRIBE**
  - returns an RDF graph that describes the resources found.

- **ASK**
  - returns whether a query pattern matches or not.
Graph Patterns

Basic Graph Pattern – set of Triple Patterns

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns .- additional patterns may extend the solution

Alternative Graph Pattern – two or more possible patterns are tried

Patterns on Named Graphs - patterns are matched against named graphs
SPARQL Queries: All Full Names

“Return the full names of all people in the graph”

PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?fullName
WHERE {?x vCard:FN ?fullName}

result:

fullName
"John Smith"
"Mary Smith"
“Return the relation between John and Mary”

PREFIX ex: <http://example.org/#>
SELECT ?p
WHERE {ex:john ?p ex:mary}

result:

p

<http://example.org/#marriedTo>
Graph Patterns

**Basic Graph Pattern** – set of *Triple Patterns*

**Group Pattern** - a set of graph patterns must all match

**Value Constraints** - restrict RDF terms in a solution

**Optional Graph Patterns** - additional patterns may extend the solution

**Alternative Graph Pattern** – two or more possible patterns are tried

**Patterns on Named Graphs** - patterns are matched against named graphs
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ex: <http://example.org/#!>

SELECT ?y
WHERE {?x vCard:FN "John Smith".
    ?x ex:marriedTo ?y}
"Return the spouse of a person by the name of John Smith"

PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ex: <http://example.org/#!>
SELECT ?y
WHERE {?x vCard:FN "John Smith".
  ?x ex:marriedTo ?y}

result:

Y

@prefix ex: <http://example.org/#!> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [ vcard:Given "John" ; vcard:Family "Smith" ] ;
ex:hasAge 32 ;
ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [ vcard:Given "Mary" ; vcard:Family "Smith" ] ;
ex:hasAge 29 .
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
    ?name vCard:Given ?firstName}
“Return the first name of all people in the KB”

PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
    ?name vCard:Given ?firstName}

result:

name firstName
===============
_:a "John"
_:b "Mary"
"Rewrite the naming information in original graph by using the foaf:name"

PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x vCard:FN ?name }

result:

#john foaf:name "John Smith"
#marry foaf:name "Marry Smith"
“Rewrite the naming information in original graph by using the foaf:name”

PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT { ?x foaf:name ?name } WHERE { ?x vCard:FN ?name }

result:

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith";

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:mary
  vcard:FN "Marry Smith";

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:ex="http://example.org">
  <rdf:Description rdf:about=ex:john>
    <foaf:name>John Smith</foaf:name>
  </rdf:Description>
  <rdf:Description rdf:about=ex:mary>
    <foaf:name>Marry Smith</foaf:name>
  </rdf:Description>
</rdf:RDF>
"Are there any married persons in the KB?"

PREFIX ex: <http://example.org/#>
ASK { ?person ex:marriedTo ?spouse }

result:

yes

=================

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [ vcard:Given "John" ; vcard:Family "Smith" ] ;
ex:hasAge 32 ;
ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [ vcard:Given "Mary" ; vcard:Family "Smith" ] ;
ex:hasAge 29 .

www.sti-innsbruck.at
Graph Patterns

**Basic Graph Pattern** – set of *Triple Patterns*

**Group Pattern** - a set of graph patterns must all match

**Value Constraints** - restrict RDF terms in a solution

**Optional Graph Patterns** - additional patterns may extend the solution

**Alternative Graph Pattern** – two or more possible patterns are tried

**Patterns on Named Graphs** - patterns are matched against named graphs
“Return all people over 30 in the KB”

PREFIX ex: <http://example.org/#>
SELECT ?x
WHERE {?x hasAge ?age .}
FILTER(?age > 30)

result:

x

<http://example.org/#john>
Graph Patterns

**Basic Graph Pattern** – set of *Triple Patterns*

**Group Pattern** - a set of graph patterns must all match

**Value Constraints** - restrict RDF terms in a solution

**Optional Graph Patterns** - additional patterns may extend the solution

**Alternative Graph Pattern** – two or more possible patterns are tried

**Patterns on Named Graphs** - patterns are matched against named graphs
SPARQL Queries: Optional Patterns

“Return all people and (optionally) their spouse”

PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE {?person ex:hasAge ?age .
OPTIONAL { ?person ex:marriedTo ?spouse } }

result:

?person ?spouse
=============================<http://example.org/#mary><http://example.org/#john> <http://example.org/#mary>
60

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [ vcard:Given "John" ; vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [ vcard:Given "Mary" ; vcard:Family "Smith" ] ;
  ex:hasAge 29 .

<http://example.org/#john> <http://example.org/#mary>
SPARQL Queries: Optional Patterns

PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE {?person ex:hasAge ?age .
OPTIONAL { ?person ex:marriedTo ?spouse.
?spouse vCard:FN ?name} 
FILTER (bound(?name))}

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
ex:hasAge 32 ;
ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
ex:hasAge 29 .
“Return all people and their spouse if the spouse has a name”

PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE { ?person ex:hasAge ?age .
OPTIONAL { ?person ex:marriedTo ?spouse.
?spouse vCard:FN ?name}
FILTER (bound(?name))}

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [ vcard:Given "John" ; vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [ vcard:Given "Mary" ; vcard:Family "Smith" ] ;
  ex:hasAge 29 .
"Return all people and their spouse if the spouse has a name"

PREFIX ex: <http://example.org/#>  
SELECT ?person, ?spouse  
WHERE {?person ex:hasAge ?age .  
OPTIONAL { ?person ex:marriedTo ?spouse.  
   ?spouse vCard:FN ?name}  
FILTER (bound(?name))}  

result:

?person ?spouse  
=============================
<http://example.org/#john> <http://example.org/#mary>

@prefix ex: <http://example.org/> .  
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
ex:john  
vcard:FN "John Smith" ;  
vcard:N [  
vcard:Given "John" ;  
vcard:Family "Smith" ] ;  
ex:hasAge 32 ;  
ex:marriedTo :mary .  
ex:mary  
vcard:FN "Mary Smith" ;  
vcard:N [  
vcard:Given "Mary" ;  
vcard:Family "Smith" ] ;  
ex:hasAge 29 .
Graph Patterns

**Basic Graph Pattern** – set of *Triple Patterns*

**Group Pattern** - a set of graph patterns must all match

**Value Constraints** - restrict RDF terms in a solution

**Optional Graph Patterns** - additional patterns may extend the solution

**Alternative Graph Pattern** – two or more possible patterns are tried

**Patterns on Named Graphs** - patterns are matched against named graphs
“Return all people and their spouses”
PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE { {?person ex:marriedTo ?spouse} UNION
{?spouse ex:marriedTo ?person } }
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
vcard:FN "John Smith" ;
vcard:N [ vcard:Given "John" ;
vcard:Family "Smith" ] ;
ex:hasAge 32 ;
ex:marriedTo :mary .
ex:mary
vcard:FN "Mary Smith" ;
vcard:N [ vcard:Given "Mary" ;
vcard:Family "Smith" ] ;
ex:hasAge 29 .
“Return all people and their spouses”

PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE { {?person ex:marriedTo ?spouse} UNION
   {?spouse ex:marriedTo ?person} }

@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
vcard:FN "John Smith" ;
vcard:N [ vcard:Given "John" ; vcard:Family "Smith" ] ;
ex:hasAge 32 ;
ex:marriedTo :mary .
ex:mary
vcard:FN "Mary Smith" ;
vcard:N [ vcard:Given "Mary" ; vcard:Family "Smith" ] ;
ex:hasAge 29 .

result:
?person ?spouse
================================
<http://example.org/#john> <http://example.org/#mary>
<http://example.org/#mary> <http://example.org/#john>
Graph Patterns

Basic Graph Pattern – set of *Triple Patterns*

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns - additional patterns may extend the solution

Alternative Graph Pattern – two or more possible patterns are tried

Patterns on Named Graphs - patterns are matched against named graphs
Querying the Dataset

# Graph: http://example.org/foaf/aliceFoaf
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .
_:a foaf:knows _:b .
_:b rdfs:seeAlso <http://example.org/foaf/bobFoaf> .
_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
_:b foaf:age 32 .

# Graph: http://example.org/foaf/bobFoaf
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
_:1 foaf:mbox <mailto:bob@work.example> .
_:1 rdfs:seeAlso <http://example.org/foaf/bobFoaf> .
_:1 foaf:age 35 .
Querying the Dataset - Accessing Graph Labels

PREFIX foaf: <http://xmlns.com/foaf/0.1#>
SELECT ?src, ?bobAge
WHERE {
  GRAPH ?src
    {?
x foaf:mbox <mailto:bob@work.example>.
    ?x foaf:age ?bobAge}
}

result:
?src ?bobAge
=============================<http://example.org/foaf/aliceFoaf> 32
<http://example.org/foaf/bobFoaf> 35
Questions?

ioan.toma@sti2.at