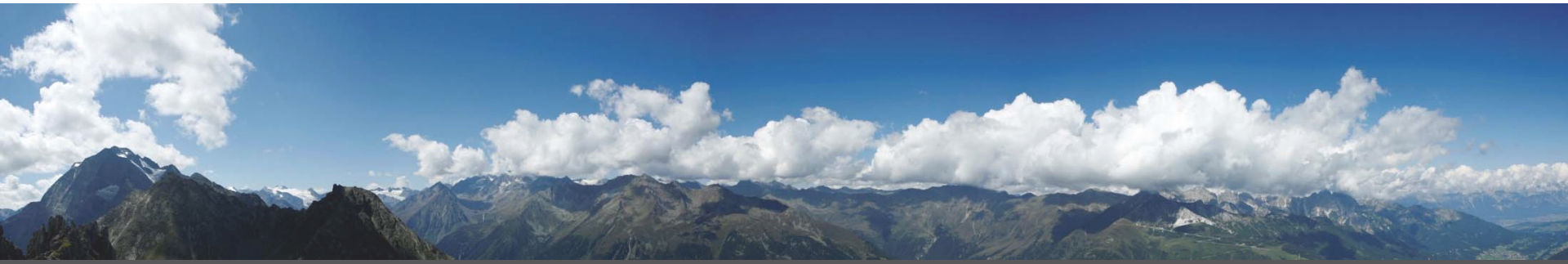
STI · INNSBRUCK

Semantic Web, RDF and SPARQL

1st Semantic Web Services Winter Retreat

Ioan Toma



Overview

STI · INNSBRUCK

- From Web to Semantic Web
- RDF
- SPARQL

Why do we need Semantic Web?

FROM WEB TO SEMANTIC WEB

Today Web

STI · INNSBRUCK

- 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

Today Web

- 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…

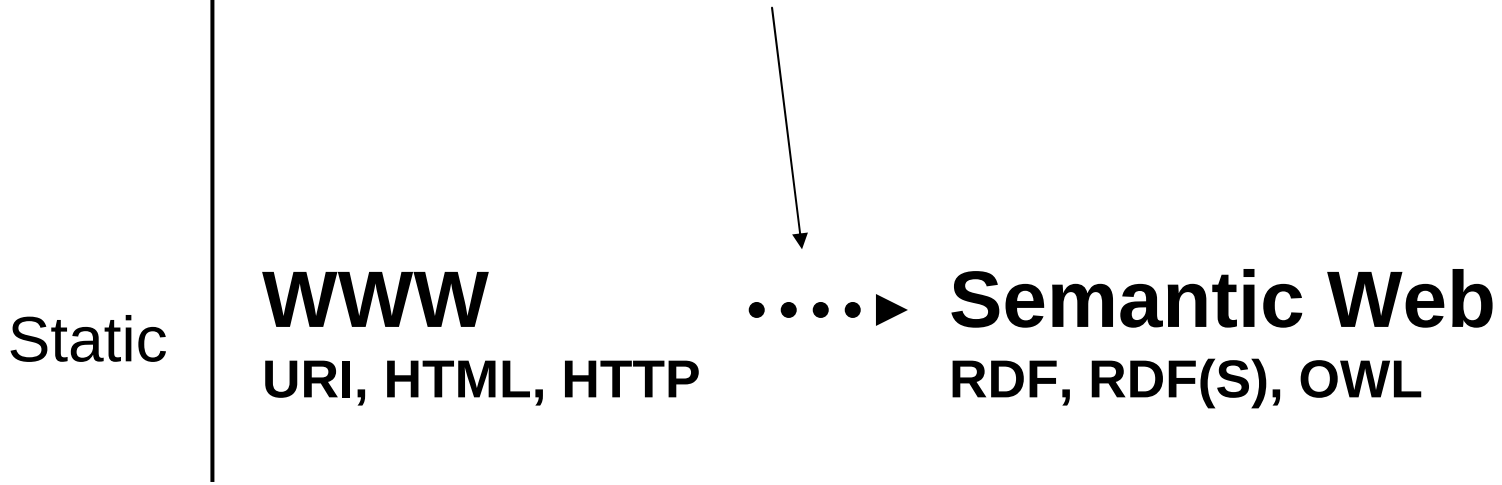
STI · INNSBRUCK

- 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?
  - …

The Vision (contd.)

Serious Problems in

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



What is the Semantic Web?

STI · INNSBRUCK

- *"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…

STI · INNSBRUCK

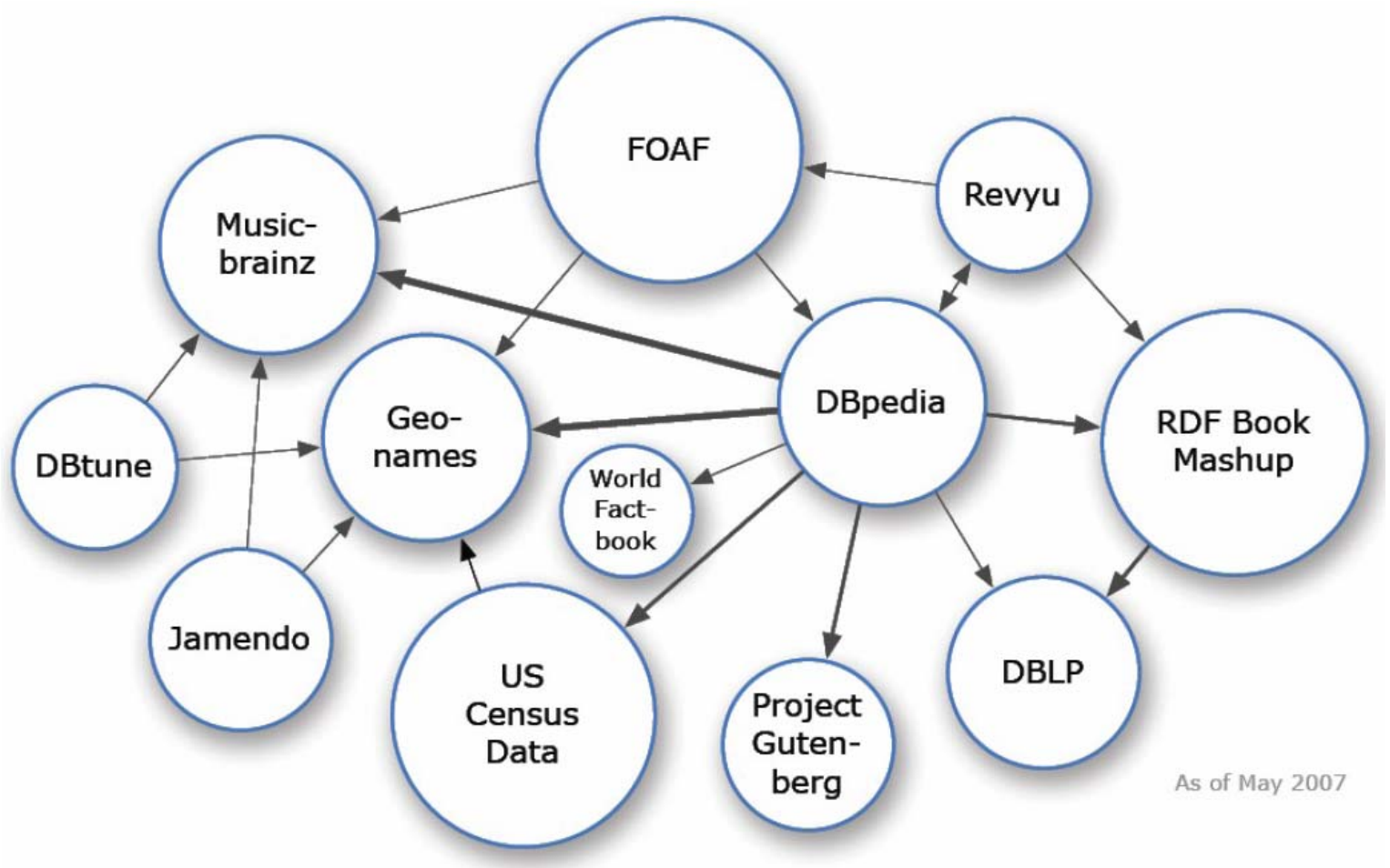
- Web Data Annotation
  - connecting (syntactic) Web objects, like text chunks, images, … to their semantic meaning (e.g., this image is about Innsbruck, SWSR is taking place in Seefeld)
- Data Linking on the Web (Web of Data)
  - global networking of knowledge through URI, RDF, and SPARQL (e.g., connecting 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

STI · INNSBRUCK

- 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

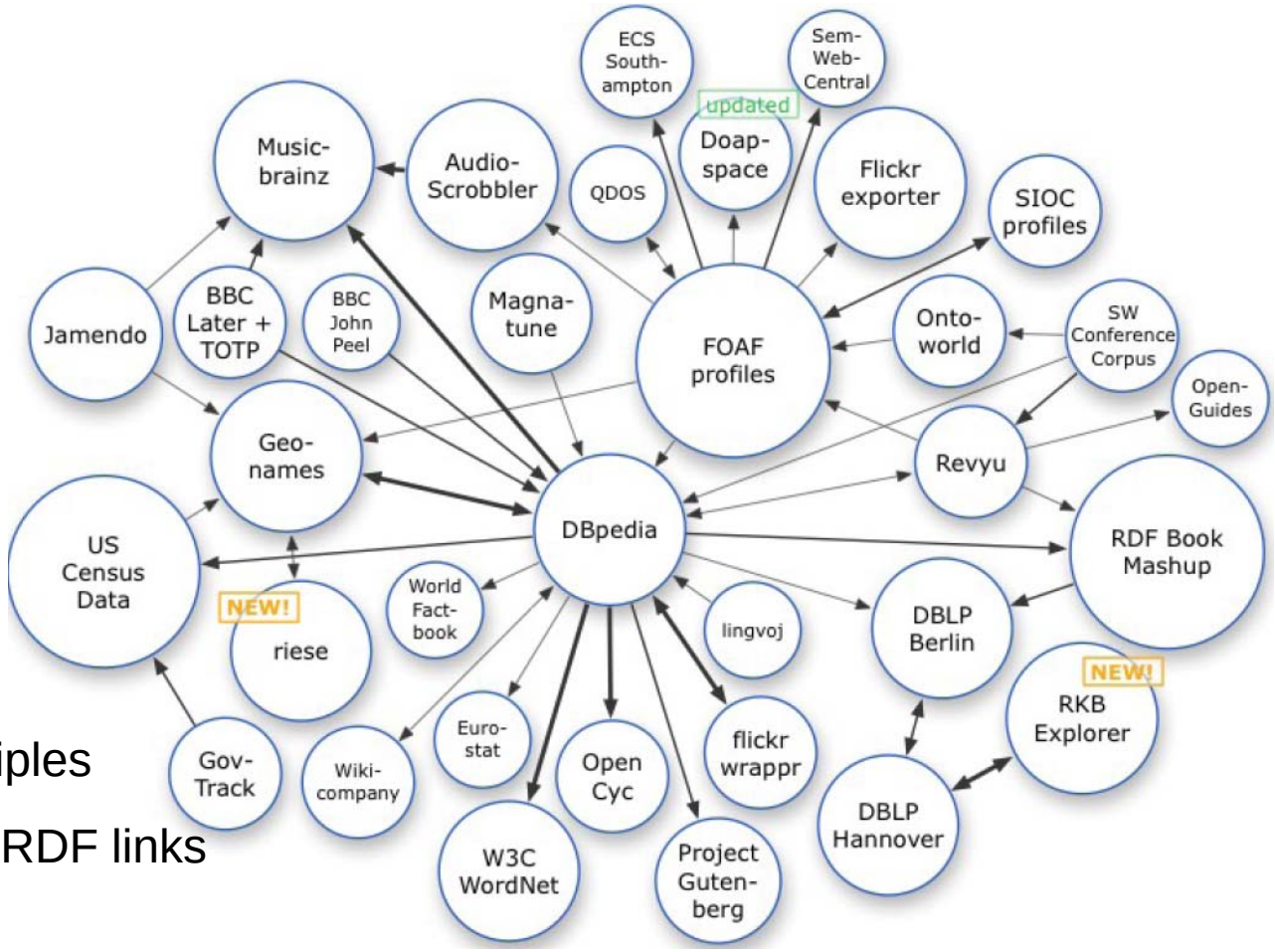


Over 500M RDF triples

Around 120K RDF links between data sources

Bubbles in April 2008

STI · INNSBRUCK



>2B RDF triples

Around 3M RDF links

Principles

STI · INNSBRUCK

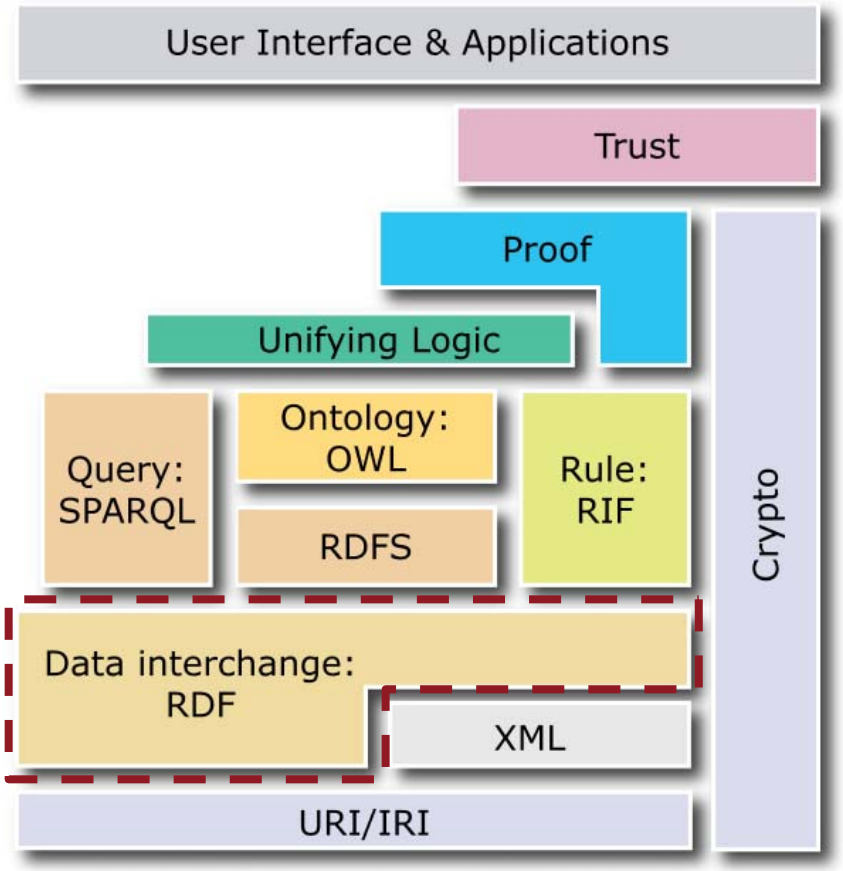
- Linked Data is simply about using the Web to create typed links between data from different sources.
- The principle of Linked data is to:
  - Use the RDF data model to publish structured data on the web
  - Use RDF links to interlink data from different data sources.
  - Use HTTP URIs to identify resource
    - To avoid other URI schemes (URNs or DOIs)

What LOD can bring?

STI · INNSBRUCK

- 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

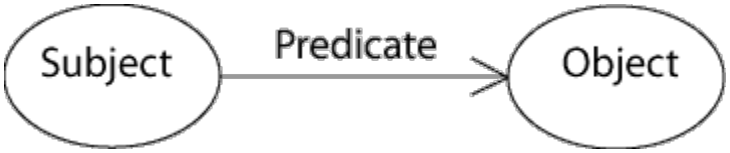
Semantic Web Stack



THE RESOURCE DESCRIPTION FRAMEWORK

Intro

- 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

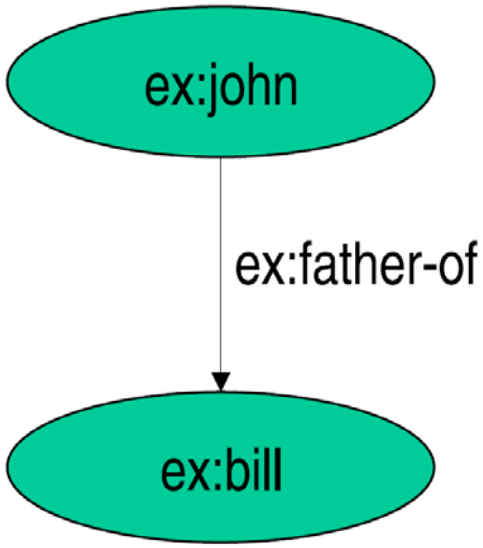
STI · INNSBRUCK

- 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

STI · INNSBRUCK

- 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

STI · INNSBRUCK

- 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!

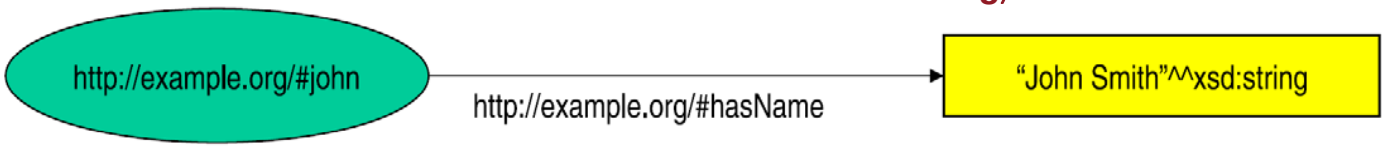
- A URI is an *identifier* and **not** a location on the Web

- 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

STI · INNSBRUCK

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

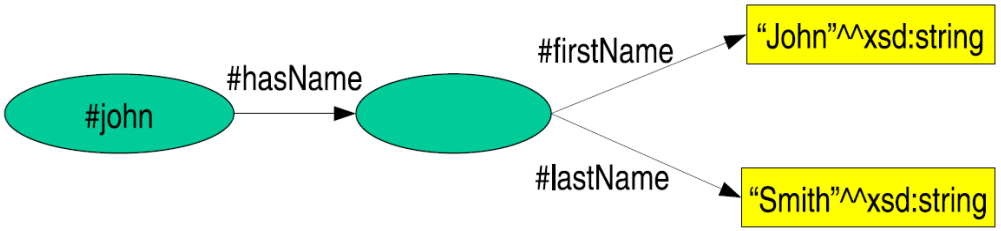
- Recommended datatypes are XML Schema datatypes, e.g.:
  - **`xsd:string`**
  - **`xsd:integer`**
  - **`xsd:float`**
  - **`xsd:anyURI`**
  - **`xsd:boolean`**

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".

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

Reification

- Reification: statements about statements

Mary claims that John's name is "John Smith".

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

⟨<#john>, <#hasName>, "John Smith"⟩

Reification

STI · INNSBRUCK

- Reification: statements about statements

Mary claims that John's name is "John Smith".

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

⟨<#mary>, <#claims>, <#myStatement>⟩
```

RDF Vocabulary

STI · INNSBRUCK

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

- Classes:
  - **`rdf:Property`**, **`rdf:Statement`**, **`rdf:XMLLiteral`**
  - **`rdf:Seq`**, **`rdf:Bag`**, **`rdf:Alt`**, **`rdf:List`**
- Properties:
  - **`rdf:type`**, **`rdf:subject`**, **`rdf:predicate`**, **`rdf:object`**,
  - **`rdf:first`**, **`rdf:rest`**, **`rdf:_n`**
  - **`rdf:value`**
- Resources:
  - **`rdf:nil`**

RDF Vocabulary

STI · INNSBRUCK

- Typing using **`rdf:type`**:

  `<A, rdf:type, B>`

  "A belongs to class B"

- All properties belong to class **`rdf:Property`**:

  `<P, rdf:type, rdf:Property>`

  "P is a property"

  `<rdf:type, rdf:type, rdf:Property>`

  "rdf:type is a property"

RDF Containers

STI · INNSBRUCK

- Grouping property values:

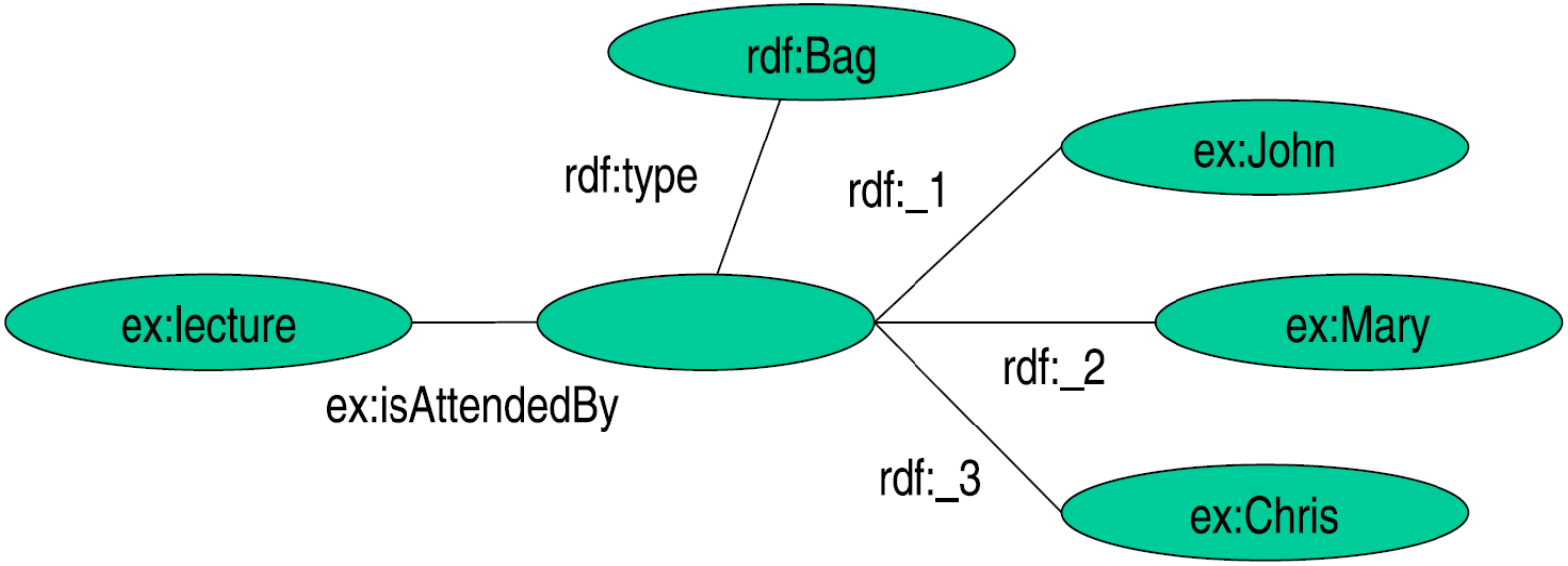
"The lecture is attended by John, Mary and Chris" **Bag**

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

"The source code for the application may be found at ftp1.example.org, ftp2.example.org, ftp3.example.org" **Alt**

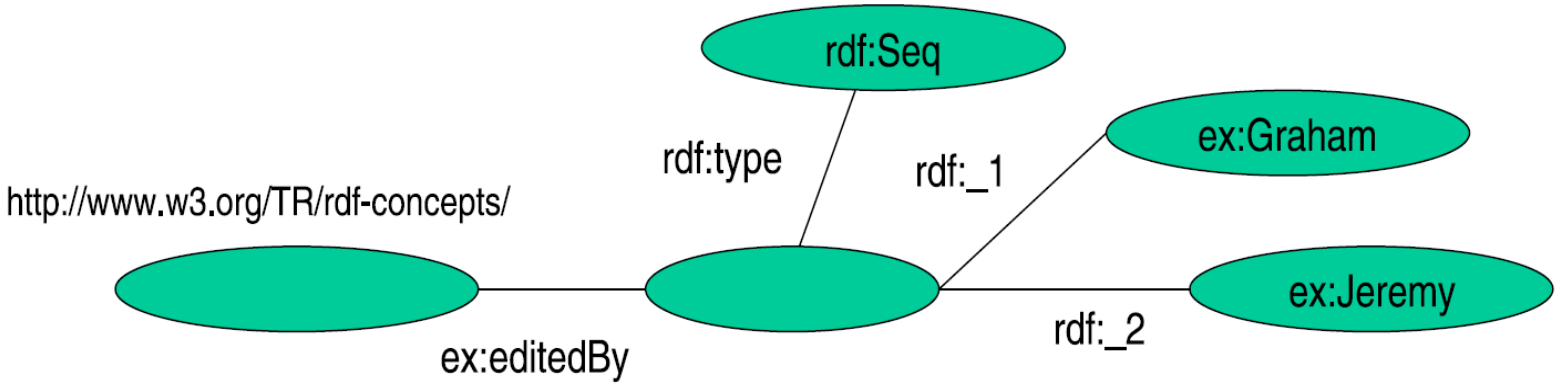
RDF Containers: Bag

"The lecture is attended by John, Mary and Chris"



RDF Containers: Seq

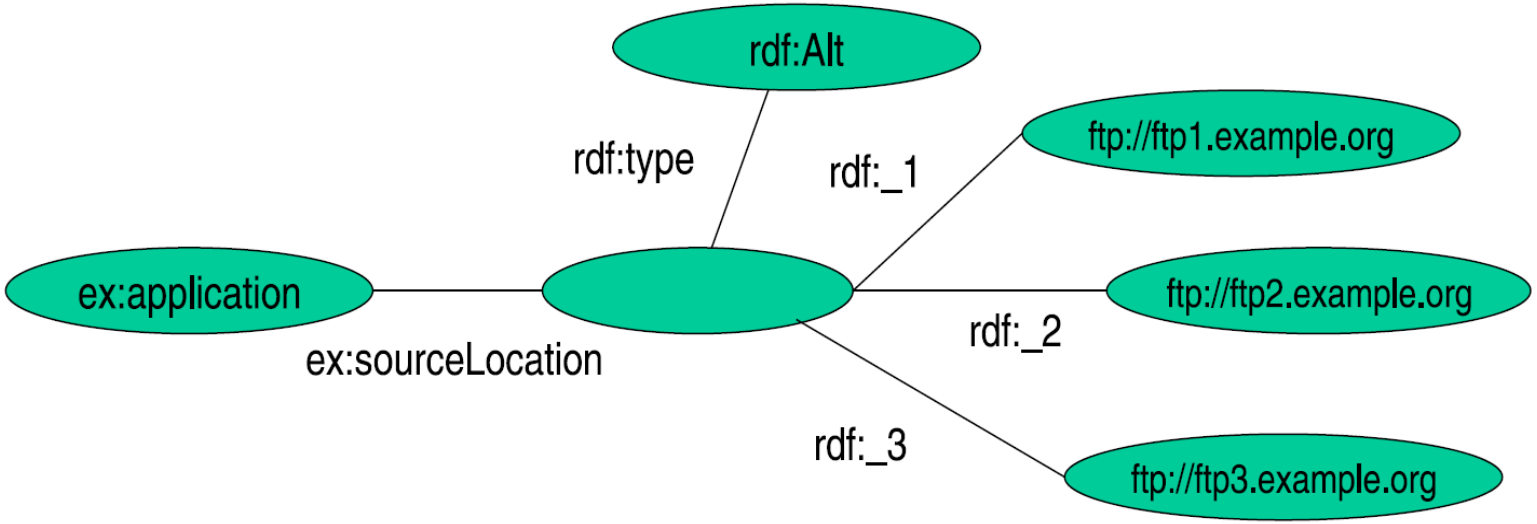
"[RDF-Concepts] is edited by Graham and Jeremy (in that order)"



RDF Containers: Alt

STI · INNSBRUCK

"The source code for the application may be found at ***ftp1.example.org***, ***ftp2.example.org***, ***ftp3.example.org***"



RDF Containers 2

STI · INNSBRUCK

- 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

STI · INNSBRUCK

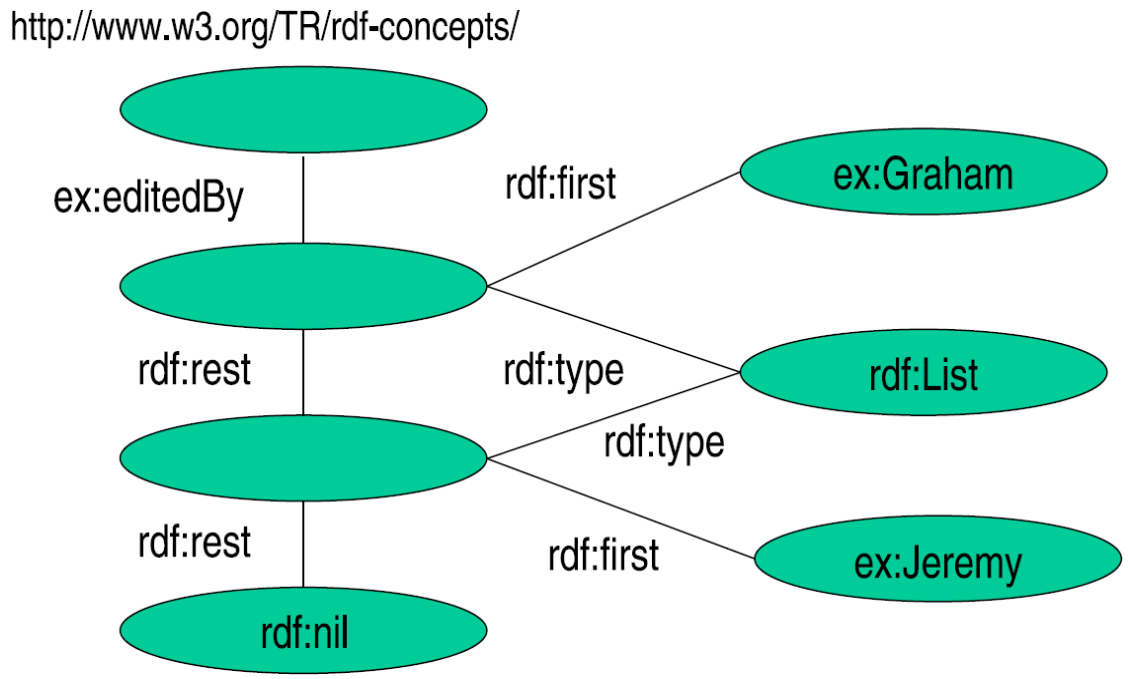
- Three types of containers:
  - **`rdf:Bag`** - unordered set of items
  - **`rdf:Seq`** - ordered set of items
  - **`rdf:Alt`** - set of alternatives
- Every container has a triple declaring the **`rdf:type`**

- 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 Collections

"[RDF-Concepts] is edited by Graham and Jeremy (in that order) and nobody else"



RDF/XML 1

STI · INNSBRUCK

- 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)
- . . .

RDF/XML 2

STI · INNSBRUCK

<#john, #hasName, “John”>

<#john, #marriedTo, #mary>

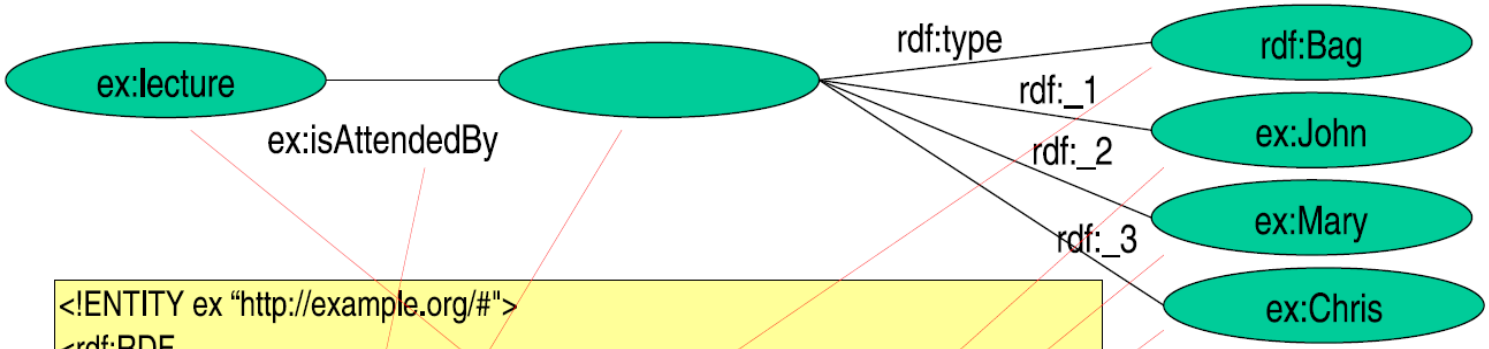
```
<!ENTITY ex “http://example.org/#">

<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>
```

Head

Body

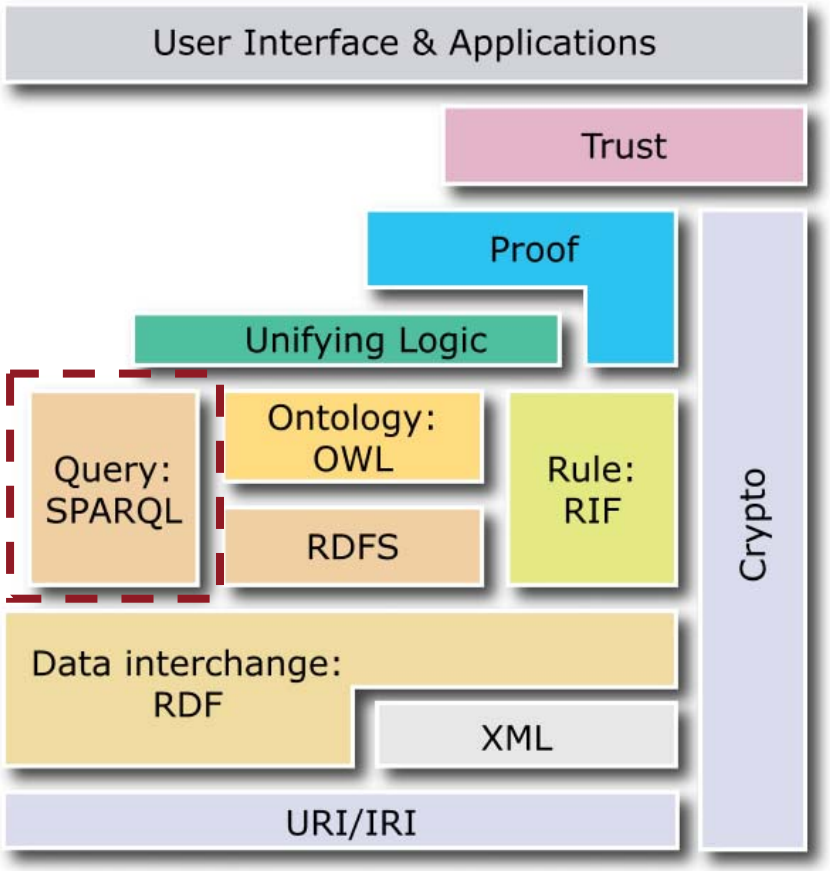
Foot

RDF/XML 3




```
<!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>
```

Semantic Web Stack



How to query RDF data

SPARQL

Querying RDF

STI · INNSBRUCK

- 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>
WHERE { <http://example.org/book/book1> dc:title ?title }
```

SPARQL Queries

STI · INNSBRUCK

```
PREFIX dc: http://purl.org/dc/elements/1.1
SELECT ?title
FROM <http://example.org/library>
WHERE { http://example.org/book/book1 dc:title ?title }
```

- 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

STI · INNSBRUCK

```
<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

STI · INNSBRUCK

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

STI · INNSBRUCK

"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"
```

```
@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 .
```

SPARQL Queries: Properties

"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>
```

```
@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

STI · INNSBRUCK

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: Complex Patterns

STI · INNSBRUCK

```
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}
```

```
@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 .
```

SPARQL Queries: Complex Patterns

STI · INNSBRUCK

"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
==================
<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 .
```

SPARQL Queries: Blank Nodes

STI · INNSBRUCK

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

```
@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 .
```

SPARQL Queries: Blank Nodes

STI · INNSBRUCK

"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"
```

```
@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 .
```

SPARQL Queries: Building RDF Graph

STI · INNSBRUCK

"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"
```

```
@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 .
```

SPARQL Queries: Building RDF Graph

STI · INNSBRUCK

"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" ;
```

```
<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:marry>
       <foaf:name>Marry Smith</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

SPARQL Queries:
Testing if the Solution Exists

STI · INNSBRUCK

"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 .
```

Graph Patterns

STI · INNSBRUCK

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: Constraints (Filters)

STI · INNSBRUCK

"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>
```

```
@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

STI · INNSBRUCK

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

STI · INNSBRUCK

"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 } }
```

```
@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/#mary>
<http://example.org/#john> <http://example.org/#mary>
```

SPARQL Queries: Optional Patterns

STI · INNSBRUCK

```
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 .
```

SPARQL Queries: Optional Patterns

STI · INNSBRUCK

"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 .
```

SPARQL Queries: Optional Patterns

STI · INNSBRUCK

"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 .
```

result:

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

Graph Patterns

STI · INNSBRUCK

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: Matching Alternatives

STI · INNSBRUCK

“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 .
```

SPARQL Queries: Matching Alternatives

STI · INNSBRUCK

"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

STI · INNSBRUCK

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

STI · INNSBRUCK

```
# 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> .
<http://example.org/foaf/bobFoaf> rdf:type foaf:PersonalProfileDocument .
_: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 .
<http://example.org/foaf/bobFoaf> rdf:type foaf:PersonalProfileDocument .
```

Querying the Dataset - Accessing Graph Labels

STI · INNSBRUCK

```
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}
      }
```

 # 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> .

<http://example.org/foaf/bobFoaf> rdf:type foaf:PersonalProfileDocument .

_: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 .

<http://example.org/foaf/bobFoaf> rdf:type foaf:PersonalProfileDocument .

result:

```
?src ?bobAge
============================
<http://example.org/foaf/aliceFoaf> 32

<http://example.org/foaf/bobFoaf> 35
```

Questions?

ioan.toma@sti2.at