Chapter 6
RDF/XML: RDF Data on the Web

- An XML representation of RDF data for providing RDF data on the Web

⇒ could be done straightforwardly as a “holds” relation mapped according to SQLX (see next slide).

- would be highly redundant and very different from an XML representation of the same data
- search for a more similar way: leads to “striped XML/RDF”
  - data feels like XML: can be queried by XPath/Query and transformed by XSLT
  - can be parsed into an RDF graph.
- usually: provide RDF/XML data to an agreed RDFS/OWL ontology.

A Straightforward XML Representation of RDF Data

Note: this is not RDF/XML, but just some possible representation.

- RDF data are triples,
- their components are either URIs or literals (of XML Schema datatypes),
- straightforward XML markup in SQLX style,
- since N3 has a term structure, it is easy to find an XML markup.

```
<my-n3:rdf-graph xmlns:my-n3="http://simple-silly-rdf-xml.de#">
 <my-n3:triple>
  <my-n3:subject type="uri">foo://bar/persons/john</my-n3:subject>
  <my-n3:predicate type="uri">foo://bar/meta#name</my-n3:predicate>
  <my-n3:object type="http://www.w3.org/2001/XMLSchema#string">John</my-n3:object>
 </my-n3 triple>
 <my-n3:triple> ... </my-n3 triple>
   :
</my-n3:rdf-graph>
```

- The problem is not to have *any* XML markup, but to have a useful one that covers the *semantics* of the RDF data model.

6.1 RDF/XML: RDF as an XML Application

- root element type: <rdf:RDF >
- not just “some markup”
- but covers the semantics of “resource description”

Markup

- “Striped RDF/XML” syntax as an abbreviated form (similar to the well-known XML structure)

RDF/XML Descriptions of Resources

<rdf:Description> elements collect a (partial) description of a *resource*:

- which resource is described: @rdf:about=“*uri*”
- subelements describe its properties (amongst them, its type as a special property),
  - element name: name of the property
    Note that this name is actually an URI.
    (this is where XML namespaces come into play)
  - value of the property:
    - element contents:
      text content or one or more nested <rdf:Description> elements
    - attribute @rdf:resource=“*uri*”: property points to another resource that has an RDF description of its own elsewhere
- can contain nested <rdf:Description> elements similar to the N3 structure.
- there can be multiple descriptions of the same resource (as in N3).
- later: different URI definition mechanisms

Example

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xml:base="foo://bla/"
         xmlns="foo://bla/names#">
 <rdf:Description rdf:about="persons/john">
  <rdf:type rdf:resource="names#Person"/>
  <name>John</name>
  <age>35</age>
  <child>
   <rdf:Description rdf:about="persons/alice">
    <rdf:type rdf:resource="names#Person"/>
    <name>Alice</name>
    <age>10</age>
   </rdf:Description>
  </child>
  <child rdf:resource="persons/bob"/>
 </rdf:Description>
 <rdf:Description rdf:about="persons/bob">
  <rdf:type rdf:resource="names#Person"/>
  <name>Bob</name>
  <age>8</age>
 </rdf:Description>
</rdf:RDF>
```

[Filename: RDF/john-rdfxml.rdf]

- xml:base determines the URI prefix, either flat (ending with a "#", or hierarchical, ending with a "/")
- in 2nd case: local parts can be hierarchical expressions
- default namespace set to <foo://bla/names#>
- element names are the property names

```
prefix : <foo://bla/names#>
select ?X ?Y ?A
from <file:john-rdfxml.rdf>
where {?X :hasChild ?Y . ?Y :age ?A}
```

[Filename: RDF/john-rdfxml.sparql]

ABBREVIATED FORM: STRIPED RDF/XML

- Full syntax:

  <rdf:Description rdf:about="*uri*">
  <rdf:type rdf:resource="*classname*">
  resource description
  </rdf:Description>

- Abbreviated syntax:

  <*classname* rdf:about="*uri*">
  resource description
  </*classname*>

- Striped RDF/XML: alternatingly *classname* – *propertyname* – *classname*
- domain terminology URIs = element names
- all attribute names are in the RDF namespace
- all object URIs are in attribute values
- all attribute values are object URIs
  (next: an even shorter form where this will not hold!)

Example: Striped

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xml:base="foo://bla/persons/"
   xmlns="foo://bla/names#">
 <Person rdf:about="john">
  <name>John</name>
  <age>35</age>
  <child>
   <Person rdf:about="alice">
    <name>Alice</name>
    <age>10</age>
   </Person>
  </child>
  <child rdf:resource="bob"/>
 </Person>
 <Person rdf:about="bob">
  <name>Bob</name>
  <age>8</age>
 </Person>
</rdf:RDF>
```

[Filename: RDF/john-striped.rdf]

- looks very much like well-known XML
- xml:base applies now only to objects' URIs
  e.g. <foo://bla/persons/alice>
- terminology URIs reside all in the namespaces
- same query as before:

```
# jena -q -qf john-striped.sparql
prefix : <foo://bla/names#>
select ?X ?Y
from <file:john-striped.rdf>
where {?X :hasChild ?Y}
```

[Filename: RDF/john-striped.sparql]

ABBREVIATED FORM: STRIPED RDF/XML WITH VALUE ATTRIBUTES

- Full syntax:

  <rdf:Description rdf:about="*uri*">
  <rdf:type rdf:resource="*classname*"
  <$property_1$>*value*</$property_1$>
  <$property_2$ rdf:resource="*uri*"/>
  </rdf:Description>

  where $property_1$ has a single, scalar value (string or number)

- Abbreviated syntax:

  <*classname* rdf:about="*uri*" prefix:$property_1$="*value*">
  <$property_2$ rdf:resource="*uri*"/>
  </*classname*>

- Striped RDF/XML: alternatingly *classname* – *propertyname* – *classname*
- domain terminology URIs = element and attribute names
  Note: attributes MUST be prefixed by an explicit namespace
- attribute values are object URIs or literal values.

Example: Striped with Attributes

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xml:base="foo://bla/persons/"
   xmlns:p="foo://bla/names#">
 <p:Person rdf:about="john" p:name="John" p:age="35">
  <p:hasChild>
   <p:Person rdf:about="alice" p:name="Alice" p:age="10"/>
  </p:hasChild>
  <p:hasChild rdf:resource="bob"/>
 </p:Person>
 <p:Person rdf:about="bob" p:name="Bob" p:age="8"/>
</rdf:RDF>
```

[Filename: RDF/john-striped-attrs.rdf]

- looks even more like well-known XML

```
# jena -q -qf john-striped-attrs.sparql
prefix : <foo://bla/names#>
select ?X ?Y ?N
from <file:john-striped-attrs.rdf>
where {?X :hasChild ?Y . ?Y :name ?N}
```

[Filename: RDF/john-striped-attrs.sparql]

ABBREVIATIONS

- omit “blank” description nodes by
  <*property-name* rdf:parseType=“Resource”> ... </*property-name*>
- literal-valued properties can even be added to the surrounding property element.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:mon="http://www.semwebtech.org/mondial/10/meta#">
 <mon:City rdf:nodeID="hannover" mon:name="Hannover">
  <mon:population rdf:parseType="Resource">
   <mon:year>1995</mon:year> <mon:value>525763</mon:value>
  </mon:population>
  <mon:population mon:year="2002" mon:value="515001"/>
 </mon:City>
</rdf:RDF>
```

[Filename: RDF/parse-type.rdf]

- rdf:parseType is not a real RDF citizen:
  - it exists only in the RDF/XML serialization,
  - it is intended as a parsing instruction to the RDF/XML → RDF parser.

URI Representation/Construction Mechanisms

- describe a remote resource via its full global URI (as above)
  - attribute @rdf:about="*uri*" identifies a remote resource
- use a base URI by xml:base that sets the base URI for resolving relative RDF URI references (i.e., rdf:about, rdf:resource, rdf:ID and rdf:datatype), otherwise the base URI is that of the document.
  - set xml:base="*uri*" (e.g. in the root element)
  - @rdf:about="*relativepath*": the resource's global URI is then composed as *xmlbase relativepath* (note that *xmlbase* must end with "/" or "#")
  - @rdf:ID="*local-id*": the resource's global URI is then composed as *xmlbase#local-id*. *local-id* must be a simple qname (no path!)
  - then, use @rdf:resource="#*localpart*" in the object position for referencing it.
- only locally known IDs:
  - attribute @rdf:nodeID= "*name*": defines and describes a *local* resource that can be referenced only inside the same RDF instance by its ID
  - then, use @rdf:nodeID="*id*" in the object position of a property instead of @rdf:resource="*uri*"

Example: using global protocol://path#IDs

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xml:base="foo://bla#"
   xmlns="foo://bla/names#">
 <Person rdf:ID="john">
  <name>John</name>
  <age>35</age>
  <child>
   <Person rdf:ID="alice">
    <name>Alice</name>
    <age>10</age>
   </Person>
  </child>
  <child rdf:resource="#bob"/>
 </Person>
 <Person rdf:ID="bob">
  <name>Bob</name>
  <age>8</age>
 </Person>
</rdf:RDF>
```

[Filename: RDF/john-ids.rdf]

- xml:base determines the URI prefix
  IDs must then be qnames (e.g. "john/doe" not allowed)
- default namespace set to <foo://bla/names#>
- element names are the property names

```
# jena -q -qf john-ids-rdf.sparql
prefix : <foo://bla/names#>
select ?X ?Y
from <file:john-ids.rdf>
where {?X :hasChild ?Y}
```

[Filename: RDF/john-ids-rdf.sparql]

- URIs are then <foo://bla#john>and <foo://bla/names#name>;
  note: the "#" at the end of xml:base is optional.

Example: using local IDs

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns="foo://bla/names#">
 <Person rdf:nodeID="john">
  <name>John</name>
  <age>35</age>
  <child>
   <Person rdf:nodeID="alice">
    <name>Alice</name>
    <age>10</age>
   </Person>
  </child>
  <child rdf:nodeID="bob"/>
 </Person>
 <Person rdf:nodeID="bob">
  <name>Bob</name>
  <age>8</age>
 </Person>
</rdf:RDF>
```

[Filename: RDF/john-local.rdf]

- no xml:base
- all IDs must be qnames and are localized (e.g., _:b1)
- default namespace set to “foo://bla/names#”
- element names are the property names

```
# jena -q -qf john-local-rdf.sparql
prefix : <foo://bla/names#>
select ?X ?Y ?N
from <file:john-local.rdf>
where {?X :hasChild ?Y. ?Y :name ?N}
```

[Filename: RDF/john-local-rdf.sparql]

- a result of the query is e.g. ?X/_:b0, ?Y/_:b1, ?N/“Bob”
- these local resources cannot be referenced by other RDF instances.

Example (with base URI and relative paths)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
   xml:base="http://www.semwebtech.org/mondial/10/">
 <mon:Country rdf:about="countries/D/" mon:name="Germany" mon:code="D">
  <mon:hasProvince>
   <mon:Province rdf:about="countries/D/provinces/Niedersachsen/" mon:name="Niedersachsen">
    <mon:hasCity>
     <mon:City rdf:about="countries/D/provinces/Niedersachsen/cities/Hannover/" mon:name="Hannover">
      <mon:population>
       <rdf:Description>
        <mon:year>1995</mon:year> <mon:value>525763</mon:value>
       </rdf:Description>
      </mon:population>
     </mon:City>
    </mon:hasCity>
    <mon:capital rdf:resource="countries/D/provinces/Niedersachsen/cities/Hannover/"/>
   </mon:Province>
  </mon:hasProvince>
 </mon:Country>
</rdf:RDF>
```

[Filename: RDF/a-bit-mondial.rdf]

- global URIs are e.g. <http://www.semwebtech.org/mondial/10/names#name>and <http://www.semwebtech.org/mondial/10/countries/D/provinces/Niedersachsen/cities/Hannover>
- rdf:Description used for a blank node (population) – this will even be shorter later

NAMES VS. URIS – XMLNS VS. XML:BASE

- element and attribute names are subject to namespace expansion,
- URIs in rdf:about, rdf:resource, rdf:ID and rdf:datatype are subject to expansion with xml:base.

- What if URIs from different areas are used?
  - inside a document, different (even hierarchically nested!) xml:base values can be used,
  - entities can be used inside URIs.

LOCAL XML:BASE VALUES

- here, it pays that with the XML level, there is an intermediate semantical level (in contrast to the pure N3 syntax)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
   xml:base="http://www.semwebtech.org/mondial/10/">
 <mon:Country xml:base="countries/D/" rdf:about="." mon:name="Germany" mon:code="D">
  <mon:has_city>
   <mon:City rdf:about="cities/Berlin" mon:name="Berlin"/>
  </mon:has_city>
 </mon:Country>
 <mon:Country xml:base="foo://bla/countries/F/" rdf:about="." mon:name="France" mon:code="F">
  <mon:has_city>
   <mon:City rdf:about="cities/Paris" mon:name="Paris"/>
  </mon:has_city>
 </mon:Country>
</rdf:RDF>
```

[Filename: RDF/url-expansion.rdf]

- relative xml:base expressions are appended: <http://www.semwebtech.org/mondial/10/countries/D/cities/Berlin>
- absolute xml:base expressions overwrite: <foo://bla/countries/F/cities/Paris>.

XML Entities in URIs

- if URIs from different bases are mingled in the document:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY mon "http://www.semwebtech.org/mondial/10/">
  <!ENTITY xyz "a:bc"> ] >
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:mon="this://is-actually-not-used"
         xmlns:f="foo://bla#"
         xml:base="foo://bla/">
 <f:Person rdf:about="persons/john" f:name="John" f:age="35">
  <!-- this is not expanded at all: -->
  <f:test rdf:resource="mon:countries/D/cities/Berlin"/>
  <!-- the right way is to use an entity: -->
  <f:lives-in rdf:resource="&mon;countries/D/cities/Berlin"/>
  <f:married-to rdf:resource="&xyz;#mary"/>
 </f:Person>
</rdf:RDF>
```

[Filename: RDF/url-entities.rdf]

```
# jena -q -qf url-entities.sparql
select ?X ?P ?Y
from <file:url-entities.rdf>
where {?X ?P ?Y}
```

[Filename: RDF/url-entities.sparql]

Specification of Datatypes in RDF/XML

- RDF uses XML Schema types
- yields typed literals such as “42”^^<http://www.w3.org/2001/XMLSchema#int>
- In RDF/XML, the type of a literal value is specified by an rdf:datatype attribute whose value is recommended to be one of the following: an XSD literal type URI or the URI of the datatype rdf:XMLLiteral.
  (but then, they cannot be abbreviated into attributes)

```
<mon:Country rdf:resource="http://www.semwebtech.org/mondial/10/countries/D">
  <mon:name
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Germany</mon:name>
  <mon:area
    rdf:datatype="http://www.w3.org/2001/XMLSchema#float">356910</mon:area>
</mon:Country>
```

[example next slide]

DATATYPES: EXAMPLE

note: `http://www.w3.org/2001/XMLSchema#` can be defined as an entity in the local DTD to the RDF/RDFS instance and is then used as rdf:datatype="&xsd;string"

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#"> ]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
   xml:base="http://www.semwebtech.org/mondial/10/">
 <mon:Country rdf:about="countries/D">
  <mon:name rdf:datatype="&xsd;string">Germany</mon:name>
  <mon:population rdf:datatype="&xsd;int">83536115</mon:population>
 </mon:Country>
</rdf:RDF>
```

[Filename: RDF/rdf-datatype.rdf]

- `jena -t -pellet -if rdf-datatype.rdf`
- Note: having linebreaks in the data yields unexpected results.

XMLLITERAL IN RDF/XML: EXAMPLE

- use rdf:parseType="Literal":

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xml:base="foo://bla/persons/"
   xmlns:p="foo://bla/names#">
 <p:Person rdf:about="john" p:name="John" p:age="35">
  <p:homepage rdf:parseType="Literal">
   <ht:html xmlns:ht="http://www.w3.org/1999/xhtml">
    <ht:body><ht:li>bla</ht:li></ht:body>
   </ht:html>
  </p:homepage>
  <p:hasChild rdf:resource="alice"/>
 </p:Person>
</rdf:RDF>
```

[Filename: RDF/rdf-xmlliteral.rdf]

```
prefix : <foo://bla/persons/>
select ?X ?P ?Y
from <file:rdf-xmlliteral.rdf>
where {:john ?P ?Y}
```

[Filename: RDF/rdf-xmlliteral.sparql]

- the resulting literal is
  "<ht:html . . . > . . . </ht:html>"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral>
- ... including the newlines (= XML text nodes) inside the XML fragment.

Using XMLLiterals: Exclusive Canonical XML

- Required by some software (e.g., in the “Jena” Semantic Web Framework)
- XML fragments/subtrees must be processable without their context – thus, namespaces must be present at appropriate levels in the tree.
- Details: `http://www.w3.org/TR/xml-exc-c14n/`
- can be obtained with `xmllint -exc-c14n x.xml > y.xml` (and analogously by other tools)

RDF/XML VS. “PURE” XML

- striped RDF/XML gives very much the look&feel of common XML documents:
  - nearly no “rdf:...” elements
  - no “rdf:...” elements that are relevant from the XML processing point of view
- can be processed with XPath/XQuery and XSLT as pure XML data
- can also be processed as RDF data in *combination* with RDFS/OWL metadata information (usually from a different source).

6.2 XML Syntax of RDFS/OWL

- RDFS/OWL descriptions are also <rdf:Description>s – descriptions of types/rdfs:/owl:Classes or rdf:Properties
- additionally include rdfs namespace declaration

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

same as above:

- Full syntax:

```
<rdf:Description rdf:about="class-uri">
  <rdf:type rdf:resource="owl:Class">
    resource description
</rdf:Description>
```

- Abbreviated syntax:

```
<owl:Class rdf:about="class-uri">
    resource description
</owl:Class>
```

- Full syntax:

```
<rdf:Description rdf:about="property-uri">
  <rdf:type rdf:resource="rdf:Property">
    resource description
</rdf:Description>
```

- Abbreviated syntax:

```
<rdf:Property rdf:about="property-uri">
    resource description
</rdf:Property>
```


RDF Schema Documents

- description of classes

```
<owl:Class rdf:about="uri">
  <rdfs:subClassOf rdf:resource="class-uri2"/>
</owl:Class>
```

used in XML/RDF data documents by <rdf:type resource="uri"/> or <uri>...</uri>, also used by <rdfs:subClassOf rdf:resource="uri"/> (and by rdfs:domain and rdfs:range).

- description of properties

```
<rdf:Property rdf:about="uri">
  <rdfs:subPropertyOf rdf:resource="property-uri2"/>
  <rdfs:domain rdf:resource="class-uri1"/>
  <rdfs:range rdf:resource="class-uri2"/>
</rdf:Property>
```

used by names of property elements and of property attributes in RDF/XML data documents, and for <rdfs:subPropertyOf rdf:resource="uri"/>.

- instead of @rdf:about="uri" the notations xml:base + local part or local-ids can be used.
- further subelements for class and property descriptions are provided by OWL.

DEFINING URIS OF RDFS CLASSES AND PROPERTIES

Classes and properties are "usual" resources, identified/defined by

<owl:Class rdf:about="*class-uri*"> ... </owl:Class>
reference by rdf:resource="*class-uri*"

<owl:Class rdf:ID="*classname*"> ... </owl:Class> (+ base-uri)
reference by rdf:resource="*#classname*" (local)
reference by rdf:resource="*base-uri#classname*" (from remote)

<owl:Class rdf:nodeID="*classname*"> ... </owl:Class>
reference by rdf:nodeID="*classname*" (only for local definitions)

(analogous for <rdf:Property>)

VERSION A: CLASSES AND PROPERTIES AS "REAL" RESOURCES IN THE RDFS/XML INSTANCE

Anything that is defined in an RDFS/OWL document - e.g., in

<http://www.semwebtech.org/mondial/10/meta#>

(or with appropriate setting of xml:base) as an

<owl:Class rdf:ID="Country"> <!-- subClassOf-defs etc.--> </owl:Class>
<rdf:Property rdf:ID="capital"> <!-- domain/range-defs etc.--> </rdf:Property>

defines URIs <http://www.semwebtech.org/mondial/10/meta#Country> and <http://www.semwebtech.org/mondial/10/meta#capital> etc. that can be used in another RDF document as (the same applies to the N3 format)

```
<rdf:RDF xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
    xml:base="http://www.semwebtech.org/mondial/10/>
  <mon:Country rdf:about="countries/D" mon:name= "Germany">
    <mon:capital rdf:resource="countries/D/provinces/Berlin/cities/Berlin"/>
  </mon:Country>
</rdf:RDF>
```

VERSION B: "VIRTUAL" RESOURCES

Using rdf:about in a class definition specifies anything about a remore resource:

- straightforward by <owl:Class rdf:about="*class-uri*"> and <owl:Property rdf:about="*property-uri*">
- write the complete URI, or
- use appropriate xml:base or entities (XML/RDF), or prefixes (N3).

COMPARISON

- Version A: class/property resources are fragments of the RDFS instance:
  - \+ @rdf:resource can actually be dereferenced and yields the class/property definition
  - \- only practical if the RDFS is non-distributed
    (although remote RDFS instances can also describe this resource by using rdfs:about)

  $\Rightarrow$ centralized ontologies
- Version B: class/property resources are identified by a virtual URI
  - \+ arbitrary RDFS instances can contribute to the resource description
  - \- users/clients have to know where the resource descriptions can be found

  $\Rightarrow$ distributed ontologies (only a central/common schema for class/property URIs required)

USE CASES FOR CLASS URIS

- in XML/RDF or pure XML data documents by <rdf:type rdf:resource="*class-uri*"/>
  - expanded wrt. xml:base; but usually the xml:base of the data document is different from the base of the domain names (=namespace). Use an entity if needed.
- in XML/RDF or pure XML data documents by class elements:
  <*[namespace:]classname*> . . . </*[namespace:]classname*>
  - where *namespace+classname* yield the *class-uri*.
  - expanded wrt. default namespace xmlns= "..." if declared.
- references from RDFS/OWL XML documents by
  <rdfs:subClassOf rdf:resource="*class-uri*"/>
  (analogously for rdfs:domain and rdfs:range)
  - in such metadata documents, usually xml:base and namespace are the same.
- incremental RDFS descriptions of the same class in RDFS/OWL documents by
  <rdf:Description rdf:about="*class-uri*">. . . </rdf:Description>
  - expanded wrt. xml:base.
- and in N3 files (by full URI or with @prefix).

USE CASES FOR PROPERTY URIS

- in striped XML/RDF or pure XML data documents by property subelements or attributes:

  <*surrounding-element [namespace:]propertyname*="...">
  <*[namespace:]propertyname*> . . . </*[namespace:]propertyname*>
  :
  </*surrounding-element*>
  - where *namespace+elementname* yield the *property-uri*.
  - expanded wrt. default namespace xmlns= "..." if declared.
- references from RDFS/OWL XML documents by
  <rdfs:subPropertyOf rdf:resource="*property-uri*"/>
  - in such metadata documents, usually xml:base and namespace are the same.
- incremental RDFS descriptions of the same property in RDFS/OWL documents by
  <rdf:Description rdf:about="*class-uri*">. . . </rdf:Description>
  - expanded wrt. xml:base.
- and in N3 files (by full URI or with @prefix).

USE CASES FOR XML SCHEMA DATATYPES IN METADATA

- For literal properties, the domain of <rdf:Property> can refer to XML Schema types, e.g.

```
<rdf:Property rdf:ID="population">
  <rdfs:domain rdf:resource="#GeoThing"/>
    <!-- i.e., country, province, district, county -->
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</rdf:Property>
```


6.3 Example: World Wide RDF Web

- many information sources that describe resources
- higher level information management (e.g., portals): use some of these sources for accessing *integrated* information

Example (RDF source see next slide) – the example is not based on real data

- mondial: countries, cities
- `<http://www.semwebtech.org/mondial/10/meta>`: the geography ontology
- another resource: cities and their airports
- `<http://sw.iata.org/ontology>` (International Air Transport Assoc.): ontology about flight information
- `<bla://sw.iata.org/flights/>`*flight*: resource associated with a given flight (e.g. LH42).
- `<bla://sw.iata.org/airports/>`*abbrev*: resource associated with a given airport (e.g., FRA, CDG).
- there will probably be a Lufthansa RDF database that describes the flights in their terminology

Example (Cont'd) [Filename: RDF/flightbase.rdf]

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY mon "http://www.semwebtech.org/mondial/10/"> ]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
   xmlns:travel="http://www.semwebtech.org/domains/2006/travel#">
 <rdf:Description rdf:about="&mon;countries/D/provinces/Berlin/cities/Berlin">
   <travel:has_airport rdf:resource="bla://sw.iata.org/airports/BLN"/>
 </rdf:Description>
 <rdf:Description rdf:about="&mon;countries/F/provinces/IledeFrance/cities/Paris">
  <travel:has_airport rdf:resource="bla://sw.iata.org/airports/CDG"/>
 </rdf:Description>
 <rdf:Description rdf:about="bla://sw.iata.org/flights/LH42"
      xmlns:iata="http://sw.iata.org/ontology#">
  <rdf:type rdf:resource="http://sw.iata.org/ontology#Flight"/>
  <iata:from rdf:resource="bla://sw.iata.org/airports/BLN"/>
  <iata:to rdf:resource="bla://sw.iata.org/airports/CDG"/>
 </rdf:Description>
</rdf:RDF>
```


RDF vs. XML

Everything that can be expressed by XML can also be expressed by RDF

+ RDF can also be used to *describe* resources
  (pictures, movies, ..., programs, Web services, ...)
+ RDF can be represented as a graph, independent from the structure of the (distributed) RDF instances
+ RDF data can be distributed over different files that describe the same resources
+ RDF has a connection to global schema description mechanisms

o RDF/XML can be queried in the same way by XPath/XQuery ...

- but: which RDF and RDFS/OWL instances?
  - if local resources are used: relatively easy
  - if global resources are used: appropriate RDFs must be searched for.

6.4 Linked Open Data – World Wide RDF Web

... slides in this section are work in progress ...

- Web sources (Web services or “traditional” Web Servers provide HTML (for browsers) or RDF (for data processing) data
- simple, with resources of the form `filename.html`, `filename.rdf`, or `filename.n3` etc.
- RDF data with virtual URIs (and also URLs) in the data only
- more intricate: URLs in the data and as *real, accessible* URLs
  cf. Linked Data principles: `https://www.w3.org/DesignIssues/LinkedData.html` or `http://www.lod-cloud.net/#about`
  - There must be resolvable http:// (or https://) URIs.
  - They must resolve, with or without content negotiation, to RDF data in one of the popular RDF formats (RDFa, RDF/XML, Turtle, N-Triples).
  - Access of the entire dataset must be possible via RDF crawling, via an RDF dump, or via a SPARQL endpoint.
  - overview of existing LOD data sources: `www.lod-cloud.net`

LOD Example: DBpedia

- data extracted from wikipedia
- data correctness: rather bad
- `http://dbpedia.org/page/France` Web page describing the RDF data about France;
- `http://dbpedia.org/data/France` delivers RDF/XML data about France (even to the Browser); this can easily be parsed and queried by tools.
- The URIs in dbpedia are actually of the form `http://dbpedia.org/resource/France` which redirects browsers to the HTML, and RDF tools to the XML/RDF data URL.

```
select ?X ?P ?Y
from <http://dbpedia.org/resource/France>
where {?X ?P ?Y}
```
[Filename: RDF/dbpedia-france.sparql]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X ?Y
from <http://dbpedia.org/resource/France>
where {?X owl:sameAs ?Y}
```
[Filename: RDF/dbpedia-france-same-as.sparql]

- ... yields France owl:sameAs `http://sws.geonames.org/3017382/`.

LOD Example: geonames

- `http://sws.geonames.org/3017382/` in the Browser: annotated google map with France;
- `http://sws.geonames.org/3017382/about.rdf` delivers RDF/XML data about France (For RDF, about.rdf is the equivalent to HTML's index.html)
- URL access to plain `http://sws.geonames.org/3017382/` with a *well-configured (see below)* RDF tool yields the triples:

```
select ?X ?P ?Y
from <http://sws.geonames.org/3017382/>
where {?X ?P ?Y}
```
[Filename: RDF/geonames-france.sparql]

- "rapper" tool: `rapper http://sws.geonames.org/3017382/`
- `http://sws.geonames.org/3017382/` is the resource describing France; while
- the triple <http://sws.geonames.org/3017382/about.rdf> <http://purl.org/dc/terms/modified> "2018-02-06"^^<http://www.w3.org/2001/XMLSchema#date> describes the *document* that describes France.

LOD Example: insee.fr

- many LOD sources are provided by e-government institutions.
- e.g., insee.fr:
  - Browser: HTML table that shows RDF triples,
  - RDF: request triples
- use a generic query+shell invocation for viewing LOD sources:

```
# call: jena -q -if http://id.insee.fr/geo/commune/05046 -qf alltriples.sparql
select ?X ?P ?Y
where {?X ?P ?Y}
```
[Filename: RDF/alltriples.sparql]

LOD: Technical Issues

Some URLs are both “HTML URLs” and “LOD RDF URLs”:

- For HTML consumers (browsers), HTML should be delivered;
- For RDF consumers, RDF/XML, RDF/A (HTML with RDF annotations), N3 text etc should be delivered.

⇒ HTTP Request contains appropriate information in the "accept" header:

  - Browsers e.g.: text/html,application/xhtml+xml,
  - XML consumers in general: application/xml,
  - RDF comsumers: application/rdf+xml, text/rdf, ... and some more.

- Technical observation
  - when running the jena-based tool from command line, it obtains triples;
  - when running it from the Web Service, it obtained HTML (and threw an error)

  ⇒ don’t access LOD data via simple GET *url*, but explicitly opening of HTTP connections with explicit
  ACCEPT: {text/turtle|application/rdf+xml} header.

LOD: SPARQL Endpoints

Optionally, LOD sources provide also a SPARQL endpoint.

Recall that SPARQL is also a communication *protocol* (for exchanging RDF data):

- W3C Recommendation at `http://www.w3.org/TR/sparql11-protocol/`
- usually at the URL `http://.../sparql`; GET, POST via HTTP
- optionally by a Web Form, e.g. `http://dbpedia.org/sparql`
- GET via Browser shows formatted HTML table:
  `http://dbpedia.org/sparql?query=select%20%20distinct%20?C%20where%20{[]%20a%20?C}%20LIMIT%20100`
  `http://dbpedia.org/sparql?query=select+?X+where+{?X+a+<http://dbpedia.org/ontology/Country>}`
- as Web Service:
  - result in SPARQL Query Results XML Format (set of tuples of bindings in XML) (described at `https://www.w3.org/TR/rdf-sparql-XMLres/`)
  - example (via Browser): `http://rdf.insee.fr/sparql`, set response format to “RDF/XML” (actually, it is not RDF/XML, but SPARQL Query Results XML Format).

Aside: Technical details of Querying a SPARQL Endpoint by HTTP+SPARQL Protocol

- See `https://www.w3.org/TR/sparql11-protocol/#query-operation`:
- HTTP GET,
- HTTP POST with URL-encoded parameters in message body,
- HTTP POST with unencoded SPARQL query in message body;
- query,
- optionally default graph and named graphs may be specified;
- accept media types for answer:
  SELECT query: application/sparql-results+xml
  CONSTRUCT query: application/rdf+xml or text/turtle
- Not every SPARQL server supports each of these access methods;
- wget and curl as quick-and-dirty UNIX tools for sending HTTP requests;
- Java sample code.

Samples: HTTP GET with parameters

- GET URLs contain the parameters in url-encoded form:

```
## wget -O -   outputs to stdout.
## optionally add    --header='Accept: application/sparql-results+xml'

wget -O - \
  http://dbpedia.org/sparql?query=select+distinct+?X+where+{?Y+a+?X}
wget -O - \
  http://id.insee.fr/sparql?query=select+distinct+?X+where+{?Y+a+?X}
                  ## wget: ok; GET URL via firefox and via java don't work

## curl: {a,b,c} has a special semantics for curl, so encode it or turn {..} off:
curl http://dbpedia.org/sparql?query=select+distinct+?X+where+%7B?Y+a+?X%7D
curl -g http://dbpedia.org/sparql?query=ask{}
curl http://id.insee.fr/sparql?query=select+distinct+?X+where+%7B?Y+a+?X%7D
curl -g http://id.insee.fr/sparql?query=ask{}
```

- results are in SPARQL Query Results XML Format.

Result: SPARQL Query Results XML Format

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
  <variable name="Concept"/>
 </head>
 <results distinct="false" ordered="true">
  <result>
   <binding name="Concept"><uri>http://www.w3.org/2002/07/owl#Thing</uri></binding>
  </result>
  <result>
   <binding name="Concept"><uri>http://www.w3.org/2002/07/owl#Class</uri></binding>
  </result>
  <result>
   <binding name="Concept"><uri>http://dbpedia.org/ontology/Person</uri></binding>
  </result>
  <result>
   <binding name="Concept"><uri>http://dbpedia.org/ontology/Country</uri></binding>
  </result>
   :
  and many more
   :
</sparql>
```


Samples: HTTP POST with URL-encoded parameters in message body

- POST: send parameters url-encoded in body

```
## wget: --method=POST must not be specified, if --post-data is used
## not necessary: --header='Content-Type: application/x-www-form-urlencoded' \

wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     http://dbpedia.org/sparql

wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     http://id.insee.fr/sparql

###curl -d: => POST;   -X POST not necessary
## curl optional: -H 'Content-Type: application/x-www-form-urlencoded'
curl -d 'query=select+distinct+?X+where+{?Y+a+?X}' \
     http://id.insee.fr/sparql
## equivalent:
curl --data-urlencode 'query=select distinct ?X where {?Y a ?X}' \
     http://id.insee.fr/sparql
```

Samples: HTTP POST with URL-encoded parameters in message body (cont'd)

```
### lotico.com delivers result as [application/sparql-results+json]:

wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     http://www.lotico.com:3030/lotico/sparql

### request results as xml:
wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     --header='Accept: application/sparql-results+xml' \
     http://www.lotico.com:3030/lotico/sparql

curl --data-urlencode 'query=select distinct ?X where {?Y a ?X}' \
     http://www.lotico.com:3030/lotico/sparql

curl --data-urlencode 'query=select distinct ?X where {?Y a ?X}' \
     --header 'Accept: application/sparql-results+xml' \
     http://www.lotico.com:3030/lotico/sparql
```


Samples: HTTP POST with unencoded SPARQL query in message body

- HTTP POST with unencoded SPARQL query in message body;
  (some services do not support this)
- parameters (default graph etc.) still as parameters

```
### not supported by http://dbpedia.org/sparql and http://id.insee.fr/sparql

wget -O - --method=POST \
     --header='Content-Type: application/sparql-query' \
     --body-data='select distinct ?X where {?Y a ?X}' \
     http://www.lotico.com:3030/lotico/sparql

curl -d 'select distinct ?X where {?Y a ?X}' \
   -H 'Content-Type: application/sparql-query' http://www.lotico.com:3030/lotico/sp
```

Querying a SPARQL Endpoint by Java (SPARQL Protocol) – GET

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
public class SparqlGET {
  public static void main(String[] args) { try {
    BufferedReader br = null;
    URL inputURL = new URL("http://dbpedia.org/sparql?query=" +
          "select+distinct+?X+where+{?X+a+<http://dbpedia.org/ontology/Country>}");
    HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("Accept", "application/sparql-results+xml");
    con.connect();
    String s = "";   StringBuffer res= new StringBuffer();
    br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
    br.close();
    System.out.println(res);  // or fill XML from Reader
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: RDF/SparqlGET.java]

Querying a SPARQL Endpoint by Java (SPARQL Protocol) – encoded POST

```
import java.io.BufferedReader; import java.net.HttpURLConnection; import java.net.URL;
import java.io.InputStreamReader;  import java.io.OutputStreamWriter;
public class SparqlEncPOST {
  public static void main(String[] args) { try {  BufferedReader br = null;
    URL inputURL = new URL("http://dbpedia.org/sparql");
    String q="query=construct+{+?X+a+<foo:country>}" +
             "+where+{?X+a+<http://dbpedia.org/ontology/Country>}";
    HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
    con.setRequestMethod("POST");
    con.setDoOutput(true);
    con.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    con.setRequestProperty("Accept", "text/turtle");
    con.connect();
      OutputStreamWriter wr = new OutputStreamWriter(con.getOutputStream());
      wr.write(q); wr.flush(); wr.close();
    String s = "";   StringBuffer res= new StringBuffer();
    br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
    br.close();  System.out.println(res);  // or fill XML from Reader
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: RDF/SparqlEncPOST.java]

Querying a SPARQL Endpoint by Java (SPARQL Protocol) Direct POST

```
import java.io.BufferedReader; import java.net.HttpURLConnection; import java.net.URL;
import java.io.InputStreamReader;  import java.io.OutputStreamWriter;
public class SparqlDirectPOST {
  public static void main(String[] args) { try {
    BufferedReader br = null;
    URL inputURL = new URL("http://www.lotico.com:3030/lotico/sparql");
    String q="select distinct ?C where {?X a ?C}";
    HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
    con.setRequestMethod("POST");
    con.setDoOutput(true);
    con.setRequestProperty("Content-Type", "application/sparql-query");
    con.setRequestProperty("Accept", "application/sparql-results+xml");
      OutputStreamWriter wr = new OutputStreamWriter(con.getOutputStream());
      wr.write(q); wr.flush(); wr.close();
    con.connect();
    String s = "";   StringBuffer res= new StringBuffer();
    br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
    br.close();   System.out.println(res);  // or fill XML from Reader
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: RDF/SparqlDirectPOST.java]

LOD: Classes and Properties as Resources

The RDFS and OWL metadata about classes and properties is also accessible:

- Access to `http://dbpedia.org/ontology/Country`
  (or rapper http://dbpedia.org/ontology/Country
- Access to `http://dbpedia.org/ontology/name`

6.5 Further RDF Vocabulary: Reification

Take statements (=triples) as resources and make statements about them:

- `rdf:Statement` which has properties `rdf:subject`, `rdf:predicate`, `rdf:object`, that yield a resource.
- XML: give an ID to the statement.
  "The statement "Germany had 83536115 inhabitants" was valid in year 1997":

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
     xml:base="http://www.semwebtech.org/mondial/10/">
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#subject"/>
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#object"/>
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate"/>
  <mon:Country rdf:about="countries/D">
    <mon:name>Germany</mon:name>
    <mon:population rdf:ID="de-pop">83536115</mon:population>
  </mon:Country>
  <rdf:Description rdf:about="#de-pop">
    <mon:year>1997</mon:year>
  </rdf:Description>
</rdf:RDF>
```

[Filename: RDF/reification.rdf]

REIFICATION: EXAMPLE

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
     xml:base="http://www.semwebtech.org/mondial/10/">
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#subject"/>
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#object"/>
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate"/>
  <mon:Country rdf:about="countries/D">
    <mon:name>Germany</mon:name>
    <mon:population rdf:ID="de-pop">83536115</mon:p
  </mon:Country>
  <rdf:Description rdf:about="#de-pop">
    <mon:year>1997</mon:year>
  </rdf:Description>
</rdf:RDF>
```

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?X ?P ?V
from <file:reification.rdf>
where {?X a rdf:Statement; ?P ?V}
```

[Filename: RDF/reification.sparql]

Triples (added automatically by the RDF semantics)
(use `jena -t -if reification.rdf` or see RDF validator):
(<http://.../mondial/10/#de-pop> rdf:type rdf:Statement)
(<http://.../mondial/10/#de-pop> rdf:subject <http://.../mondial/10/countries/D>)
(<http://.../mondial/10/#de-pop> rdf:predicate <http://.../mondial/10/meta#population>)
(<http://.../mondial/10/#de-pop> rdf:object "83536115")
(<http://.../mondial/10/#de-pop> <http://.../mondial/10/meta#year> "1997")

... the above annotates a statement that is assumed to hold.

Reification in the Semantic Web

Annotating Statements:

- with probabilities, trust, “who says ...”, even negation!
- annotations can be in different files than the annotated statements (cf. out-of-line XLink arcs),
- can be used for reasoning.

Note:

- information about a *predicate* (which describes the predicate “name” (e.g., the source where all this data is taken from, transitivity or symmetry) or the set of all instances (cardinalities), or each its (range, domain))

is different

- from describing/annotating a *statement* (i.e. one instance of a predicate).

Reification and Annotation

- Statements that do *not* hold can also by annotated:

```
<!DOCTYPE rdf:RDF [ <!ENTITY mon "http://www.semwebtech.org/mondial/10/meta#"> ] >
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
     xml:base="http://www.semwebtech.org/mondial/10/">
  <owl:AnnotationProperty rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#subject"/>
  <owl:AnnotationProperty rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#object"/>
  <owl:AnnotationProperty rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate"/>
  <rdf:Statement rdf:ID="cap-d-bonn">  <!-- ********************* BONN ***************** -->
    <rdf:subject rdf:resource="countries/D/"/>
    <rdf:predicate rdf:resource="&mon;capital"/>
    <rdf:object rdf:resource="countries/D/provinces/NordrheinWestfalen/cities/Bonn"/>
    <mon:from>1949</mon:from>
    <mon:until>1990</mon:until>
  </rdf:Statement>
  <rdf:Description rdf:about="countries/D/"> <!-- *************** BERLIN *************** -->
   <mon:capital rdf:ID="cap-d-berlin"
     rdf:resource="countries/D/provinces/Berlin/cities/Berlin"/>
  </rdf:Description>
  <rdf:Description rdf:about="#cap-d-berlin">
    <mon:from>1990</mon:from>
  </rdf:Description>
</rdf:RDF>
```

[Filename: RDF/reification-2.rdf]

Reification and Annotation (Cont'd)

- queries against the above information

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?S ?X ?Y ?F ?U
from <file:reification-2.rdf>
where {{?X mon:capital ?Y} UNION
       {?S rdf:subject ?X ; rdf:predicate mon:capital; rdf:object ?Y
        OPTIONAL {?S mon:from ?F} . OPTIONAL {?S mon:until ?U}}}
```

[Filename: RDF/reification-2.sparql]

- (?X capital ?Y) contains only (Germany, Berlin), the *triple* that actually holds.
- The statement (Germany, capital, Bonn) is not a triple, but there is only annotation about it.

Annotated Knowledge Representation

- A knowledge base in a setting where annotations are important, thus would not contain any explicit triples but only described statements
- Applications must then interpret the semantics of the annotations:
  - heuristics to generate those triples that are assume to hold actually:
  - annotation of probabilities, opinion, provenance, trust, ...
  - if an RDF source contains a statement that is somewhere else annotated that it held only in earlier times, discard it,
  - if some statement does not hold, but is e.g. annotated as believed by a trusted person, consider it to be true.