Chapter 6
RDF/XML: XML Syntax of RDF Data

- 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 RDF/XML”
  - 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 Turtle 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>
  (namespace URL: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
- 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 blank nodes in nested Turtle data.
- there can be multiple descriptions of the same resource (as in Turtle).
- 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/meta#">
 <rdf:Description rdf:about="persons/john">
  <rdf:type rdf:resource="meta#Person"/>
  <name>John</name>
  <age>35</age>
  <hasChild>
   <rdf:Description rdf:about="persons/alice">
    <rdf:type rdf:resource="meta#Person"/>
    <name>Alice</name>
    <age>10</age>
   </rdf:Description>
  </hasCild>
  <hasChild rdf:resource="persons/bob"/>
 </rdf:Description>
 <rdf:Description rdf:about="persons/bob">
  <rdf:type rdf:resource="meta#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/meta#>
- element names are the property names

```
prefix : <foo://bla/meta#>
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/meta#">
 <Person rdf:about="john">
  <name>John</name>
  <age>35</age>
  <hasChild>
   <Person rdf:about="alice">
    <name>Alice</name>
    <age>10</age>
   </Person>
  </hasChild>
  <hasChild 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/meta#>
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/meta#">
 <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/meta#>
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

- does only work with #-namespaces, otherwise constructs foo://bla/persons/#john

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

[Filename: RDF/john-ids.rdf]

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

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

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

- URIs are then <foo://bla/flatpersons#john>and <foo://bla/meta#name>

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/meta#">
 <Person rdf:nodeID="john">
  <name>John</name>
  <age>35</age>
  <hasChild>
   <Person rdf:nodeID="alice">
    <name>Alice</name>
    <age>10</age>
   </Person>
  </hasChild>
  <hasChild 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/meta#”
- element names are the property names

```
# jena -q -qf john-local-rdf.sparql
prefix : <foo://bla/meta#>
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/meta#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 Turtle 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/meta#">
 <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).

MACHINE-READIBILITY

- RDF/XML is usually automatically generated,
- not intended to be read by humans,
- processes as XML serialization of RDF data
  - as a file: → RDF/XML parser → RDF graph
  - as an (XML!) stream: possible preprocessing and then mapping to a graph/DB
  - use generic tools for XML stream processing, like SAX/StAX or the XML Digester (see slides for XML lecture/XML lab course)
    - generic processing stream→RDF graph
    - preprocessing stream→ filtered/modified graph
      (but note that nodes can occur in arbitrary order)

    Practical Exercise: Write an XML/RDF Parser in SAX/StAX/Digester.

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-uri*$_2$"/>
  </owl:Class>

  used in RDF/XML 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-uri*$_2$"/>
  <rdfs:domain rdf:resource="*class-uri*$_1$"/>
  <rdfs:range rdf:resource="*class-uri*$_2$"/>
  </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>
(absolute URI or relative URI + base-URI)
reference by rdf:resource="*class-uri*"

<owl:Class rdf:ID="*classname*"> ... </owl:Class> (+ base-URI or extending the file's URL)
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 URLs refer to the same file

- use rdf:ID and rdf:resource="#..."

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

<http://foo/bla/metatest.rdf> (or with appropriate setting of xml:base)

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

defines URIs <http://foo/bla/metatest.rdf#Country> and <http://foo/bla/metatest.rdf#capital> etc. that can be used in another RDF document as (the same applies to the Turtle format)

```
<rdf:RDF xmlns:m="http://foo/bla/metatest.rdf#"
     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>
```


Example

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#">
 <owl:Class rdf:ID="Country"/>
 <owl:Class rdf:ID="City"/>
 <rdf:Property rdf:ID="capital">
   <rdfs:domain rdf:resource="#Country"/>
   <rdfs:range rdf:resource="#City"/>
 </rdf:Property>
</rdf:RDF>
```

[Filename: RDF/metatest.rdf]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:m="file:///home/may/teaching/SemWeb/RDF/metatest.rdf#"
   xml:base="bla:dontmatterwhat#">
 <m:Country rdf:ID="Germany">     <!-- bla:dontmatterwhat#Germany -->
   <m:capital rdf:nodeID="Berlin"/>   <!-- local blank node _:b0 -->
 </m:Country>
</rdf:RDF>
```

[Filename: RDF/metatest2.rdf]

Example (cont'd)

```
select ?X ?P ?Y
from <file:metatest.rdf>
from <file:metatest2.rdf>
where {?X ?P ?Y}
```

[Filename: RDF/metatest.sparql]

- call `jena -q -qf metatest.sparql`
- illustrates the constructed URLs, like
  `<file:///home/may/teaching/SemWeb/RDF/metatest.rdf#capital>`
  (using xml:base or an http: URL makes more sense in reality)
- call `jena -pellet -q -qf metatest.sparql`
- lists lots of "derived" OWL axioms, but also derives that
  `_:b0 a <file:///home/may/teaching/SemWeb/RDF/metatest.rdf#City>`

Version B: Virtual (URI) or remote (URL) Resources

Using rdf:about in a class definition specifies anything about a remote resource URL/URI:

- 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 (RDF/XML), or base and prefixes (Turtle).

Comparison

- Version A: class/property resources are represented by XML fragments of the RDFS+OWL instance:
  - when directly extending the file's URL:
    @rdf:resource similar to an HTML anchor inside the XML document, contains the class/property definition
    (This matches the original HTML+XML flavor of the late 1990s, but such HTML-style derefencing is not implemented by default, Linked Open Data turned out to work in a different way based on Web Services)
  - only practical if the RDFS+OWL is non-distributed
    (although remote RDFS+OWL instances can also add to this resource's description by using rdfs:about)

  ⇒ 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

  ⇒ distributed ontologies (only a common syntax pattern for class/property/instance URIs is required)

USE CASES FOR CLASS URIS IN RDF/XML DOCUMENTS

- in striped RDF/XML instance data documents by class elements:
  <*[nsprefix:]classname*> . . . </*[nsprefix:]classname*>
  - non-prefixed: *default-namespace+classname* yields the *class-uri*.
  - prefixed: *nsprefix-namespace+classname* yields the *class-uri*.
- in striped RDF/XML instance data documents by <rdf:type rdf:resource="*class-uri*"/>
  - expanded wrt. xml:base; but usually the xml:base of the instance data document is different from the base of the domain names (=namespace). Use an entity if needed.
- references from RDFS+OWL documents by
  <rdfs:subClassOf rdf:resource="*class-uri*"/>
  (analogously for rdfs:domain and rdfs:range and OWL properties)
  - expanded wrt. xml:base.
- incremental RDFS/OWL descriptions of the same class in RDFS/OWL documents by
  <rdf:Description rdf:about="*class-uri*">. . . </rdf:Description>
  - expanded wrt. xml:base.
- (note that in Turtle, the user can decide to expand wrt. <relURL>+@base or prefix:qname)

USE CASES FOR PROPERTY URIS IN RDF/XML DOCUMENTS

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

  <*surrounding-element [nsprefix:]propertyname*="...">
  <*[nsprefix:]propertyname*> . . . </*[nsprefix:]propertyname*>
  :
  </*surrounding-element*>

  – non-prefixed: *default-namespace+classname* yields the *class-uri*.
  – prefixed: *nsprefix-namespace+elementname* yields the *property-uri*.

- references from RDFS+OWL RDF/XML documents by
  <rdfs:subPropertyOf rdf:resource="*property-uri*"/>
  – expanded wrt. xml:base.

- incremental RDFS+OWL descriptions of the same property in RDF/XML documents by
  <rdf:Description rdf:about="*class-uri*">. . . </rdf:Description>
  – expanded wrt. xml:base.

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="indepDate">
  <rdfs:domain rdf:resource="#Country"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
</rdf:Property>
```

6.3 Example Sketch: 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.