Tutorial on Basic SW Technologies

3. Towards a Semant
4. Towards a Seman
5. Towards a Seman
6. However…
7. Example: Searchi
8. Example: Automat
9. Example: Semanti
10. What Is Needed?
11. What Is Needed (
12. The Semantic W
13. This Course Will
15. Problem Exampl
16. Statements
17. Resource Descri
18. RDF is a Graph
19. A Simple RDF E
20. URI-s Play a Fun
21. URI-s in this Tuto
22. RDF/XML Princip
23. RDF/XML Princip
24. RDF/XML Princip
25. Several Propertie
26. Several property
27. Adding a New pr
28. Adding a New pr
29. A Very Useful Si
30. Simplification in (
31. RDF in Programm
32. Python Example
33. Use of RDF in O
34. Merging
35. Merge Shown as
36. Merge in Practice
37. Adding New Stat
38. Blank Nodes
39. Blank Nodes: Tu
40. Blank Nodes: Bla
41. Blank Nodes: Let
42. Blank Nodes: So
43. Typed Nodes
44. Typed Nodes (co
45. An Aside: Use of
46. Sequences
47. Sequences (cont
48. Sequences (cont
49. Sequences (cont
50. An Aside: Typed
51. Other Containers
52. Collections (Lists
53. The Same in RDF
54. Our Graphical Sh
55. Some Words of W
57. Back to Typing: F
58. Classes, Resour
59. Classes, Resour
60. Schema Example
61. Schema Example
62. Inferred Propertie
63. Main RDFS Clas
64. Properties (Predi
65. Properties (cont.)
66. Property Specific
67. Property Specific
68. Main RDFS Prop
69. Collections/Conta
70. Literals
71. Literals in RDF/X
72. The Power of XM
74. Small Practical Is
75. RDF/XML has its
76. Binding RDF to a
77. RDF/XML with X
78. RDF Can Also be
79. Programming Pra
80. Programming Pra
81. Jena
82. Jena (cont)
83. Lots of Other tool
84. RDF or XML?
85. Where Does the
87. Ontologies
88. Ontologies (cont
89. W3C's Ontology
90. Classes in OWL
91. Need for Enumer
92. (OWL) Classes c
93. Same in RDF/XM
94. Union of Classes
95. Same in RDF/XM
96. Property Restricti
97. Property Restricti
98. Cardinality Restri
99. Same in RDF/XM
100. Property Charac
101. Characterizatio
102. Same in RDF/X
103. OWL: Additiona
104. Term Equivalen
105. Example: Conn
106. Another Use of
107. Versioning, Ann
108. However: Ontol
109. OWL Full
110. OWL Descriptio
111. OWL Lite
112. "Description Lo
113. DL Abstract Sy
114. Ontology Exam
116. "Semantic Web
117. "Best Practices"
118. RDF Data Acce
119. Data Access Ex
120. RDF API-s
121. Semantic Web S
123. Available Speci
124. Available Speci
125. Available Speci
126. Some Books
127. Further Informat
128. Further Informat
129. Public Fora at W
130. Some Tools
131. Some Tools (Co
133. SW Applications
134. SW Application
135. SW Application
136. SW Application
137. SW Application
138. SW Application
139. SW Application
140. SW Application
141. SW Application
142. Some Closing W
143. OWL: Does it S
144. Further Informat

PART I: Introduction
PART II: Basic RDF
PART III: RDF Vocabulary Description Language (RDFS)
PART IV: RDF(S) in Practice
PART V: Ontologies (OWL)
PART VI: Future Developments
PART VII: Available Documents, Tools
PART VIII: Some Application Examples

PART I: Introduction

Towards a Semantic Web

- The current Web represents information using
  - natural language (English, German, Hungarian,...)
  - 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
    - (well, sort of... people with disabilities may have serious problems on the Web with rich media!)

Towards a Semantic Web

- Tasks often require to *combine* data on the Web:
  - hotel and travel infos may come from different sites
  - searches in different digital libraries
  - etc.
- Again, humans combine these information easily
  - even if different terminologies are used!

Towards a Semantic Web

- *The future of the Web is a universal medium for the exchange of data*
- This means applications should have access to data, too:
  - interconnection of personal information management
  - personalized services (e.g., news)
  - integrating heterogenous Web databases
  - better search engines
  - adaptive presentations (e.g., for PDA-s, phones)
  - …

  "The bane of my existence is doing things that I know the computer could do for me"
  – Dan Connolly, The XML Revolution

However...

- However: machines are ignorant!
  - partial information is unusable
  - difficult to make sense from, eg, an image
  - drawing analogies automatically is difficult
  - difficult to combine information
    - is `<foo:creator>` same as `<bar:author>`?
    - how to combine different XML hierarchies?
  - ...

Example: Searching

- The best-known example...
  - Google et al. are great, but there are too many false hits
  - adding descriptions to resources should improve this
- Related area: Digital Libraries
  - It means catalogues on the Web
    - librarians have known how to do that for centuries
    - goal is to have this on the Web, World-wide
    - extend it to multimedia data, too

Example: Automatic Assistant

- Your own personal (digital) automatic assistant
  - knows about your preferences
  - builds up knowledge base using your past
  - can *combine* the local knowledge with remote services:
    - hotel reservations, airline preferences
    - dietary requirements
    - medical conditions
    - calendaring
    - etc
- It communicates with *remote* information (ie, on the Web!)
  (D. Dertouzos: The Unfinished Revolution)

Example: Semantics of Web Services

- Web services technology is great
- But if services are ubiquitous, searching issue comes up for example:
  - "find me the most elegant Schrödinger equation solver"
  - what does it mean to be
    - "elegant"?
    - "*most* elegant"?
  - mathematicians ask these questions all the time…
- It is necessary to characterize the service
  - not only in terms of input and output parameters…
  - …but also in terms of its *semantics*

What Is Needed?

- A resource should provide *information* about itself
  - also called “metadata”
  - metadata should be in a machine processable format
  - agents should be able to “reason” about (meta)data
  - metadata vocabularies should be defined

What Is Needed (Technically)?

- To make metadata machine processable, we need:
  - unambiguous names for resources (URIs)
  - a common data model for expressing metadata (RDF)
    - and ways to access the metadata on the Web
  - common vocabularies (Ontologies)

  The "Semantic Web" is a metadata based infrastructure for reasoning on the Web
- It *extends* the current Web (and does not replace it)

The Semantic Web is Not

- "Artificial Intelligence on the Web"
  - although it uses elements of logic…
  - … it is much more down-to-Earth (we will see later)
  - it is all about properly representing and characterizing metadata
  - of course: AI systems *may* use the metadata of the SW
    - but that is a layer *way above it*
- "A purely academic research topic"
  - SW is out of the university labs now
  - lots of applications exist already (see examples later)
  - big players of the industry use it (Sun, Adobe, HP, IBM,…)
  - of course, much is still be done!

This Course Will

- Present the basic model used in the Semantic Web (RDF)
- Show how to represent RDF in XML for the Web
- Introduce the usage of Ontologies on the top of RDF
- Give an idea on how SW applications can be programmed
- Give some examples of SW applications
- Hints for further study

PART II: Basic RDF

Problem Example for the Course

- Convey the meaning of a figure through text (important for accessibility)
  - add *metadata* to the image describing the content
  - let a tool produce some simple output using the metadata
  - use a standard metadata formalism

W3C Membership evolution 1994-2001

Statements

- The metadata is a set of *statements*
- In our example:
  - "the type of the full slide is a chart, and the chart type is «line»"
  - "the chart is labeled with an (SVG) text element"
  - "the legend is also a hyperlink"
  - "the target of the hyperlink is «URI»"
  - "the full slide consists of the legend, axes, and data lines"
  - "the data lines describe full and affiliate members, all members"
- The statements are about *resources*:
  - SVG elements, general URI-s, …

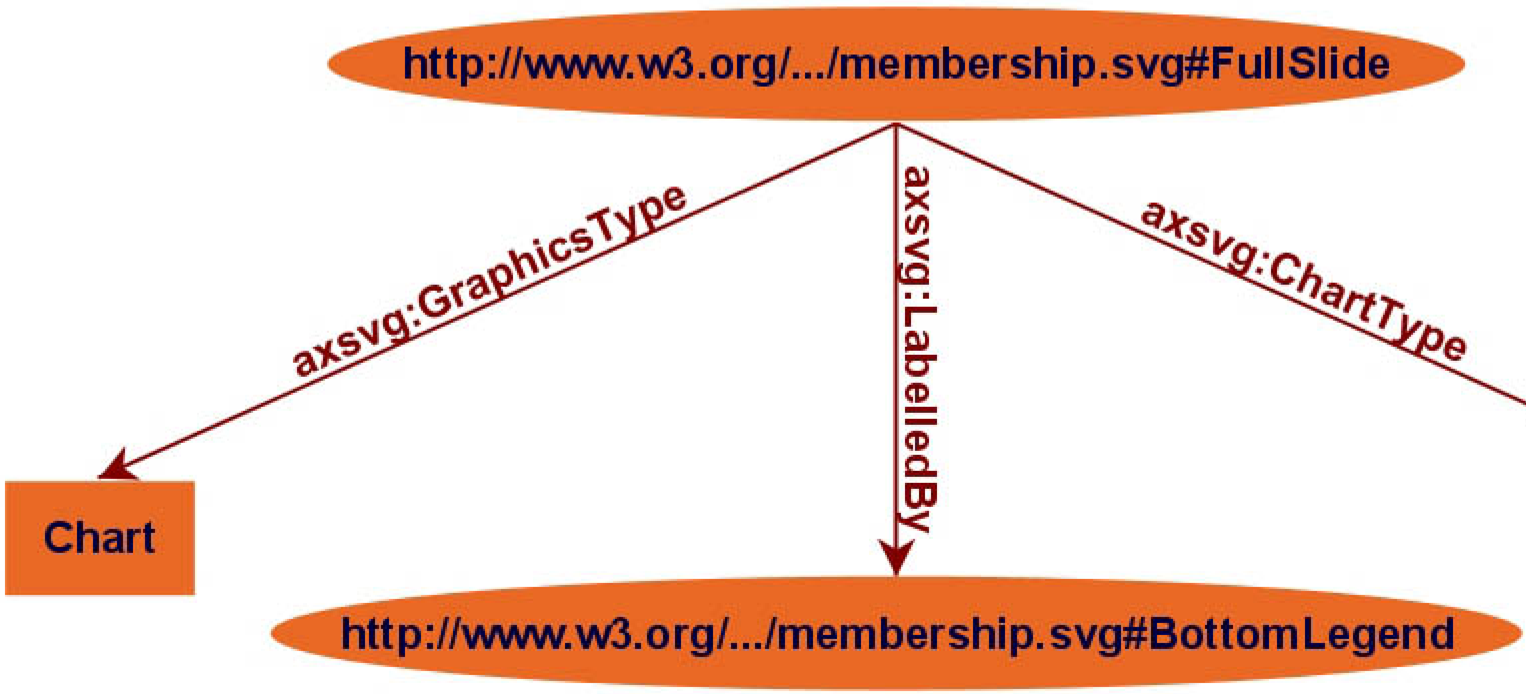
Resource Description Framework

- Statements can be modeled (mathematically) with:
  - *Resources:* an element, a URI, a literal, …
  - *Properties:* directed relations between two resources
  - *Statements:* "triples" of two resources bound by a property
    - usual terminology: (s,p,o) for subject, properties, object
- *RDF* is a general model for such statements
  - … with machine readable formats (e.g., RDF/XML, n3, Turtle, RXR)
  - RDF/XML is the "official" W3C format

RDF is a Graph

- An (s,p,o) triple can be viewed as a labelled edge in a graph
  - i.e., a set of RDF statements is a *directed, labelled graph*
    - both "objects" and "subjects" are the graph nodes
    - "properties" are the edges
  - the formal semantics of RDF is also described using graphs (see the RDF Semantics document)
- One should "think" in terms of graphs, and…
  …XML or n3 syntax are only the tools for practical usage!
  - the term "serialization" is often used for encoding
- RDF authoring tools usually work with graphs, too (XML or n3 is done "behind the scenes")

A Simple RDF Example




```
<rdf:Description
      rdf:about="http://.../membership.svg#FullSlide">
  <axsvg:GraphicsType>Chart</axsvg:GraphicsType>
  <axsvg:LabelledBy
      rdf:resource="http://.../membership.svg#BottomLegend"/>
  <axsvg:ChartType>Line</axsvg:ChartType>
</rdf:Description>
```

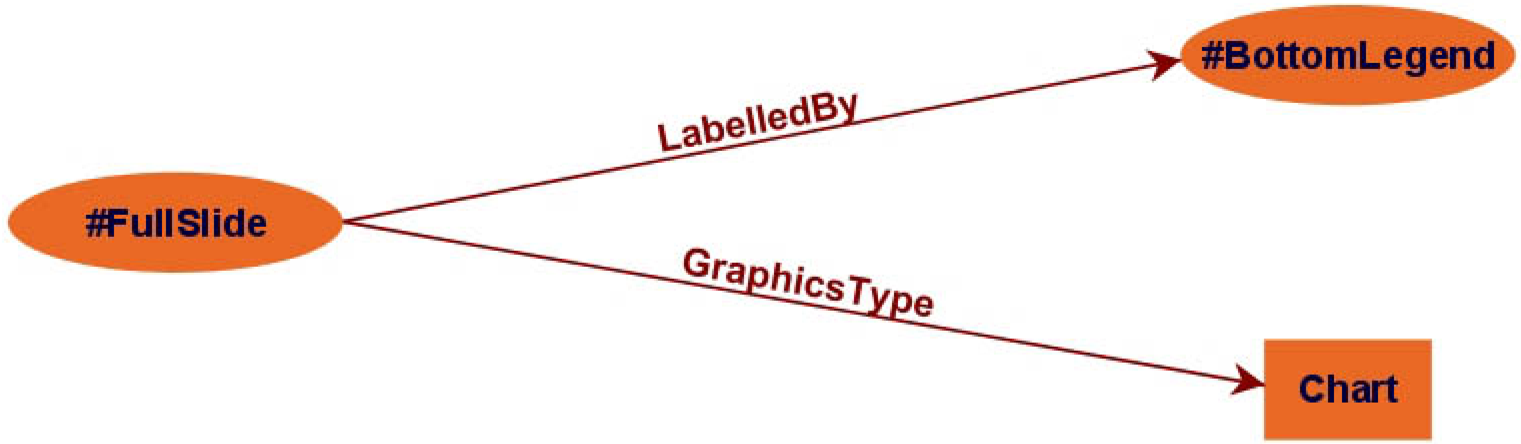
URI-s Play a Fundamental Role

- One can *uniquely* identify all resources on the web
- Uniqueness is vital to make consistent statements
- *Anybody* can create metadata on *any* resource on the Web
  - e.g., the *same* SVG file could be annotated through other terms
- It becomes easy to *merge* metadata
  - e.g., applications may merge the SVG annotations
  - this can be done because they refer to the *same* URI-s!
- *URI-s ground RDF into the Web*
  - e.g., information can be retrieved using existing tools

URI-s in this Tutorial

- In the examples, only the “fragment identifier” will be used
    - i.e., the part after the “#” character
- This is ok if the RDF metadata is:
    - encoded in XML (see later how)
    - is enclosed within an SVG or XHTML file, for example
        - just like linking with an **a** element in (X)HTML *within* a file

RDF/XML Principles



- Encode nodes and edges as XML elements or with literals:

```
«Element for #FullSlide»
    «Element for LabelledBy»
        «Element for #BottomLegend»
    «/Element for LabelledBy»
«/Element for #FullSlide»
«Element for #FullSlide»
    «Element for GraphicsType»
        Chart
    «/Element for GraphicsType»
«/Element for #FullSlide»
```

RDF/XML Principles (cont)

- Encode the resources (i.e., the nodes):

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-r
    <rdf:Description rdf:about="#FullSlide">
       «Element for GraphicsType»
          <rdf:Description rdf:about="#BottomLegend"/
       «/Element for GraphicsType»
    </rdf:Description>
<rdf:RDF>
```

- Note the usage of *namespaces*!

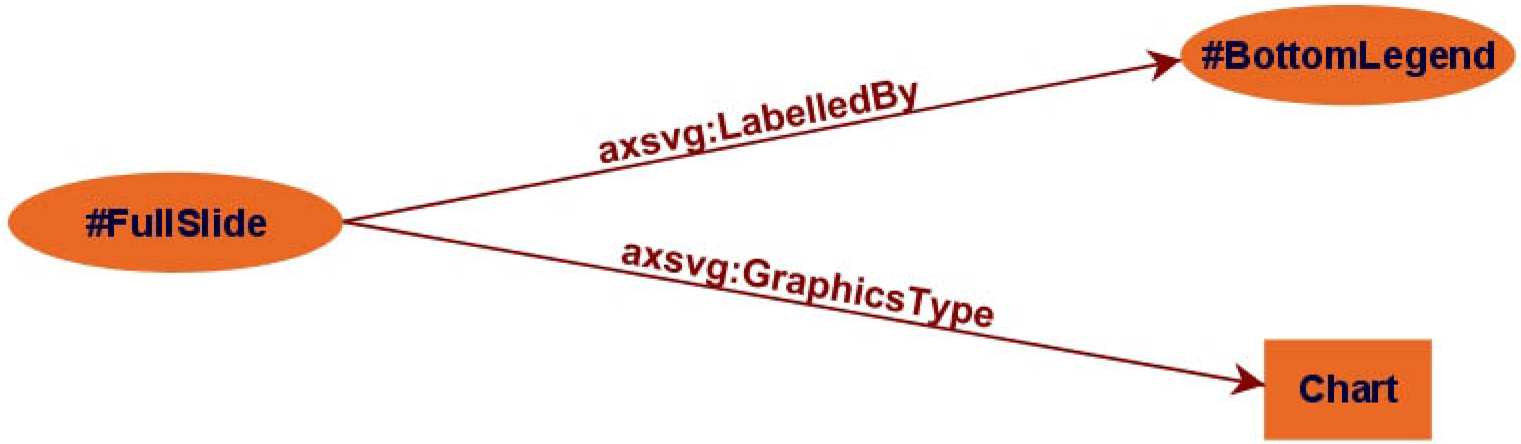
RDF/XML Principles (cont)

- Encode the property (i.e., edge) in its own namespace:

```
<rdf:RDF
  xmlns:axsvg="http://svg.example.org#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-r
    <rdf:Description rdf:about="#FullSlide">
       <axsvg:LabelledBy>
           <rdf:Description rdf:about="#BottomLegend"/
       </axsvg:LabelledBy>
    </rdf:Description>
<rdf:RDF>
```

- To save space, we will omit namespace declarations...

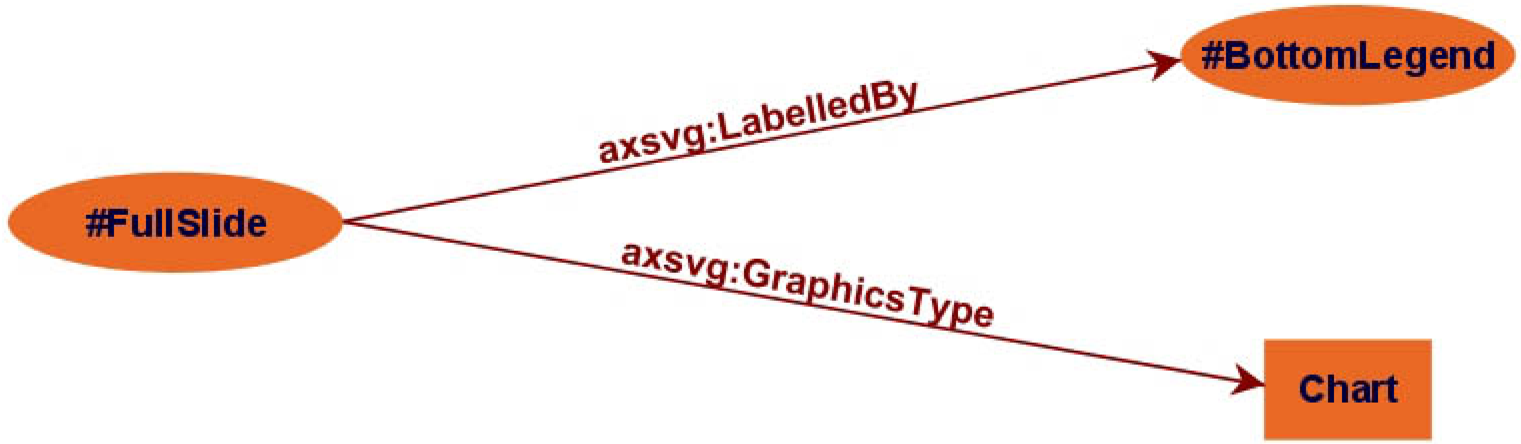
Several Properties on the Same Node



- The "canonical" solution:

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:LabelledBy>
       <rdf:Description rdf:about="#BottomLegend"/
   </axsvg:LabelledBy>
</rdf:Description>
<rdf:Description rdf:about="#FullSlide">
   <axsvg:GraphicsType>
       Chart
   </axsvg:GraphicsType>
</rdf:Description>
```

Several property on the same node



- The “simplified” version:

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:LabelledBy>
       <rdf:Description rdf:about="#BottomLegend"/
   </axsvg:LabelledBy>
   <axsvg:GraphicsType>
       Chart
   </axsvg:GraphicsType>
</rdf:Description>
```

- There are lots of other simplification rules, see later

Adding a New property

- The "canonical" solution:

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:LabelledBy>
       <rdf:Description rdf:about="#BottomLegend"/
   </axsvg:LabelledBy>
</rdf:Description>
<rdf:Description rdf:about="#BottomLegend">
   <axsvg:IsAnchor>True</axsvg:IsAnchor>
</rdf:Description>
```

Adding a New property

- The “alternative” solution:

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:LabelledBy>
      <rdf:Description rdf:about="#BottomLegend">
         <axsvg:IsAnchor>True</axsvg:IsAnchor>
      </rdf:Description/>
   </axsvg:LabelledBy>
</rdf:Description>
```

- Which version is used is a question of taste

A Very Useful Simplification

- The following structure:

```
<property>
    <rdf:Description rdf:about="URI"/>
</property>
```

appears very often. It can be replaced by:

```
<property rdf:resource="URI"/>
```

Simplification in Our Example

- Can be expressed by:

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:LabelledBy rdf:resource="#BottomLegend"/
</rdf:Description>
```

RDF in Programming Practice

- For example, using Python+RDFLib:
  - a “Triple Store” is created
  - the RDF file is parsed and results stored in the Triple Store
  - the Triple Store offers methods to retrieve:
    - triples
    - (property,object) pairs for a specific subject
    - (subject,property) pairs for specific object
    - etc.
  - the rest is conventional programming…
- Similar tools exist in XSLT, Java, etc. (see later)

Python Example

In Python syntax:

```
# import the libraries
from rdflib.TripleStore import TripleStore
from rdflib.URIRef import URIRef
# resource for a specific URI:
subject = URIRef("URI_of_Subject")
# create the triple store
triples = TripleStore()
# parse an RDF file and store it in the triple store
triples.load("membership.rdf")
# do something with (p,o) pairs
for (p,o) in triples.predicate_objects(subject) :
   do_something(p,o)
```

Use of RDF in Our Example

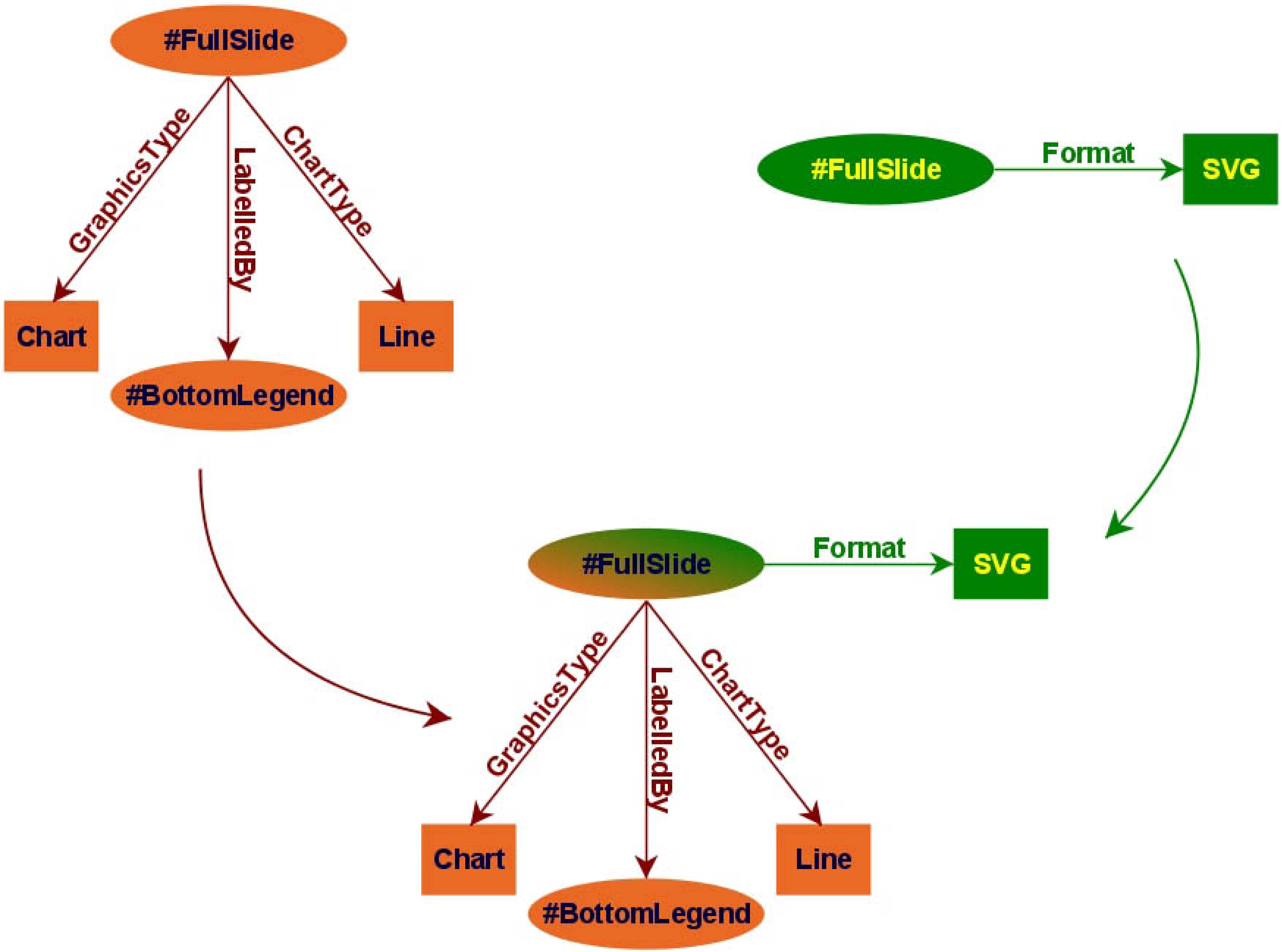
The tool:

1. Uses an RDF parser to extract metadata
2. Resolves the URI-s in RDF to access the SVG elements
3. Extracts information for the output
    - e.g., text element content, hyperlink data, descriptions
4. Combines this with a general text
5. Produces a (formatted) text for each RDF statement

Merging

- RDF statements are made on *any* URI-s
- There may be several graphs using identical URI-s
- An application *merges* these graphs (conceptually)
  - nodes with identical URI-s are considered identical
  - the rest is quite obvious
- Merging is a *very* powerful feature of RDF
  - metadata may be defined by several (independent) parties...
  - ...and combined by an application

Merge Shown as Graphs



Merge in Practice

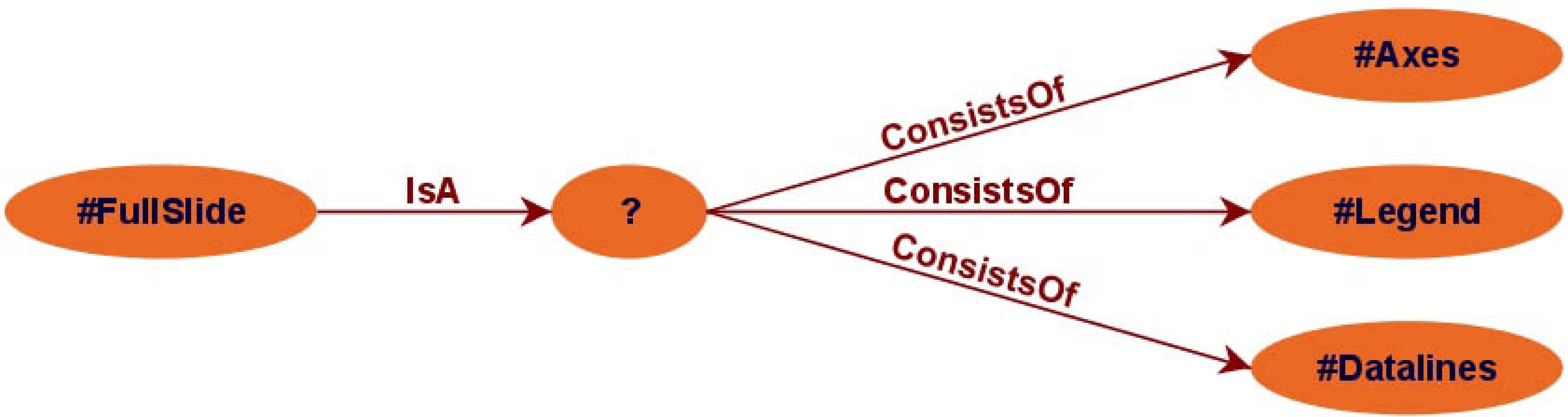
- Development environments merge graphs automatically
  - e.g., in Python, the Triple Store can “load” several files
  - the load merges the new statements automatically
- Merging the RDF/XML files into one is also possible
  - but not really necessary, the tools will merge them for you
  - keeping them separated may make maintenance easier
  - some of the files may be on a remote site anyway!

Adding New Statements

- Adding a new statement is also very simple
  - e.g., in Python+RDFLib: `store.add((s,p,o))`
- In fact, it can be seen as a special case of merging
- This is a *very* powerful feature, too
  - managing data in RDF makes it very flexible indeed...

Blank Nodes

- Consider the following statement:
  - "the full slide is a «thing» that consists of axes, the legend and the datalines"
- Until now, nodes were identified with a URI. But...
- ...what is the URI of «thing»?



Blank Nodes: Turn Them into Regulars

- In the XML serialization: give an id with **rdf:ID**

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:IsA>
       <rdf:Description rdf:about="#Thing"/>
   </axsvg:IsA>
</rdf:Description>
<rdf:Description rdf:ID="Thing">
   <axsvg:ConsistsOf rdf:resource="#Axes>
   <axsvg:ConsistsOf rdf:resource="#Legend>
   <axsvg:ConsistsOf rdf:resource="#Datalines>
</rdf:Description>
```

- Defines a fragment identifier within the RDF portion
- Identical to the **id** in HTML, SVG, ...
- Can be referred to with regular URI-s from the outside

Blank Nodes: Blank Node Identifiers

- Use an internal identifier

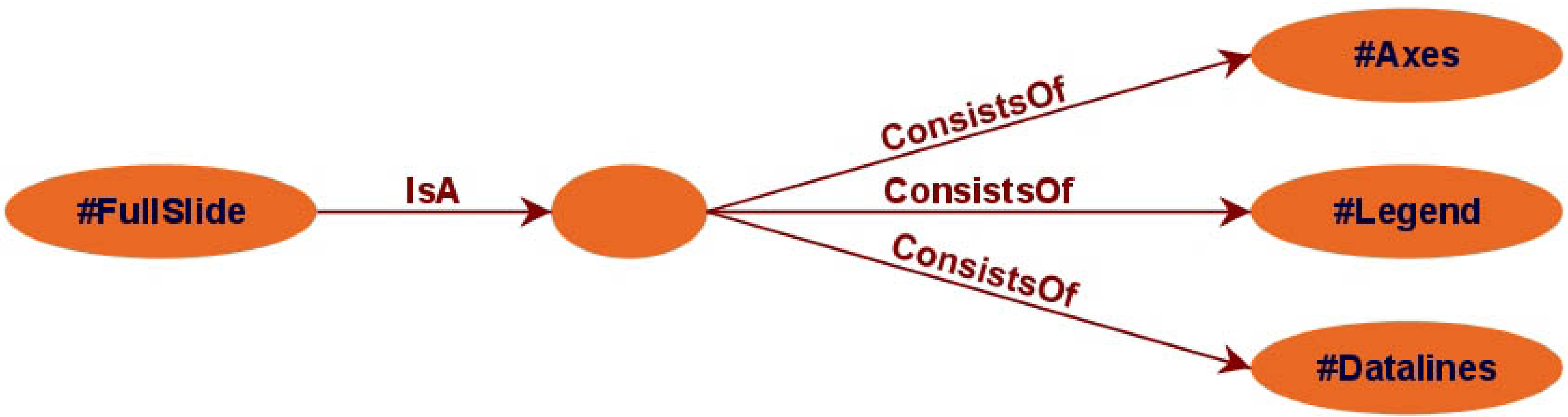
```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:IsA>
       <rdf:Description rdf:nodeID="Thing"/>
   </axsvg:IsA>
</rdf:Description>
<rdf:Description rdf:nodeID="Thing">
   <axsvg:ConsistsOf rdf:resource="#Axes>
   <axsvg:ConsistsOf rdf:resource="#Legend>
   <axsvg:ConsistsOf rdf:resource="#Datalines>
</rdf:Description>
```

- Almost like **rdf:ID**, but...
- ...**Thing** is *invisible* from outside the file!

Blank Nodes: Let the System Do It

- Let the system create a `nodeID` internally

```
<rdf:Description rdf:about="#FullSlide">
    <axsvg:IsA>
        <rdf:Description>
            <axsvg:ConsistsOf rdf:resource="#Axes>
            <axsvg:ConsistsOf rdf:resource="#Legenc
            <axsvg:ConsistsOf rdf:resource="#Datali
        </rdf:Description/>
    </axsvg:IsA>
</rdf:Description/>
```



Blank Nodes: Some More Remarks

- Blank nodes require attention when merging
  - blanks nodes in different graphs are *different*
  - the implementation must be be careful with its naming schemes
- The XML Serialization introduces a simplification

  (i.e., the blank **Description** may be omitted):

```
<rdf:Description rdf:about="#FullSlide">
    <axsvg:IsA rdf:parseType="resource">
        <axsvg:ConsistsOf rdf:resource="#Axes>
        <axsvg:ConsistsOf rdf:resource="#Legend>
        <axsvg:ConsistsOf rdf:resource="#Datalines>
    </axsvg:IsA>
</rdf:Description/>
```

Typed Nodes

- To emphasize that a node is of a specific *class*
  - i.e., it is part of a possible set of individuals
  - e.g., **#Datalines** node is an "SVG entity"
- There is a separate document on how to define classes
  - "RDF Vocabulary Description Language", a.k.a. "RDF Schemas"
  - see later in this tutorial
- We can use the special RDF property **rdf:type**:

```
<rdf:Description rdf:about="#Datalines">
  <rdf:type
    rdf:resource="http://.../axsvg-schema.rdf#SVGEnti
   ...
</rdf:Description/>
```

Typed Nodes (cont)

- A resource may belong to several classes

  (`rdf:type` is just a property…)
- The type information may be very important for applications
  - e.g., it may be used for a categorization of possible nodes
- The `rdf` namespace contains predefined classes
  - see later…

An Aside: Use of Entities

- Namespaces cannot be used *within* URI-s!
  - i.e., the full URI has to be spelled out
  - a frequent practice is to use XML entities
- So, instead of:

```
rdf:resource="http://.../axsvg-schema.rdf#SVGEntit
```

  one can use (with an entity defined in the header):

```
rdf:resource="&axsvg-schema;SVGEntity"/>
```

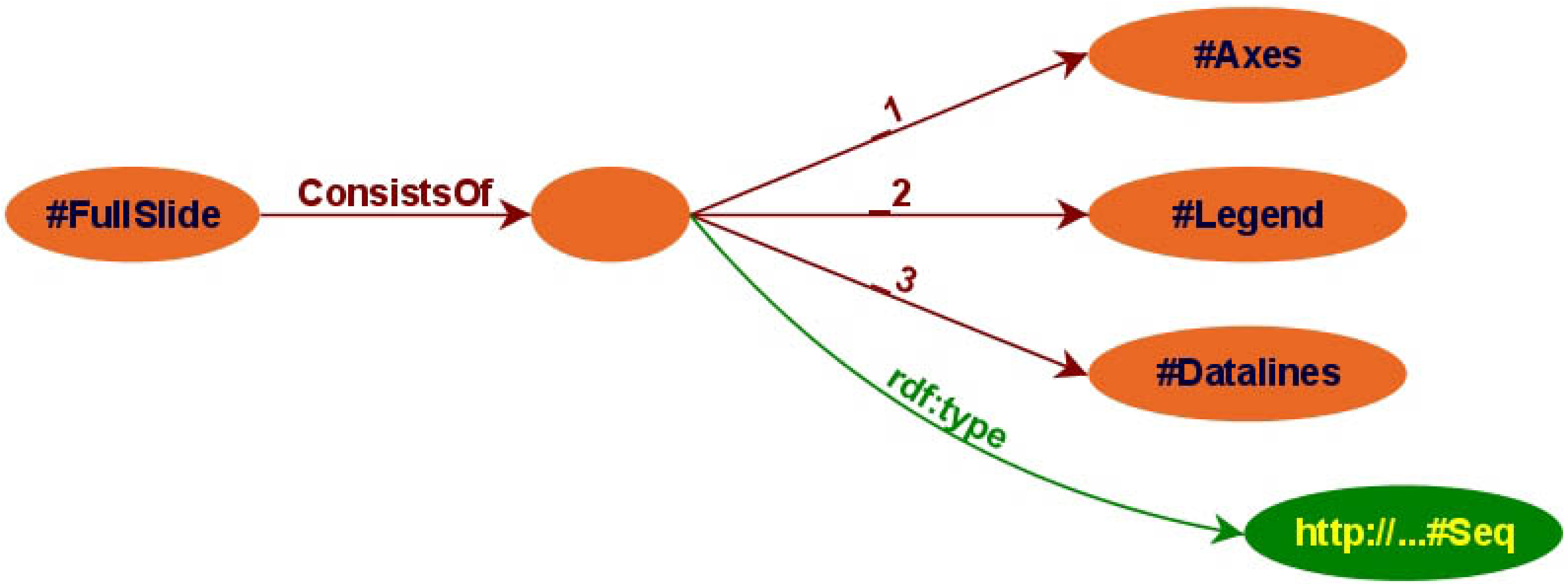
- A frequent "idiom" in RDF applications!

Sequences

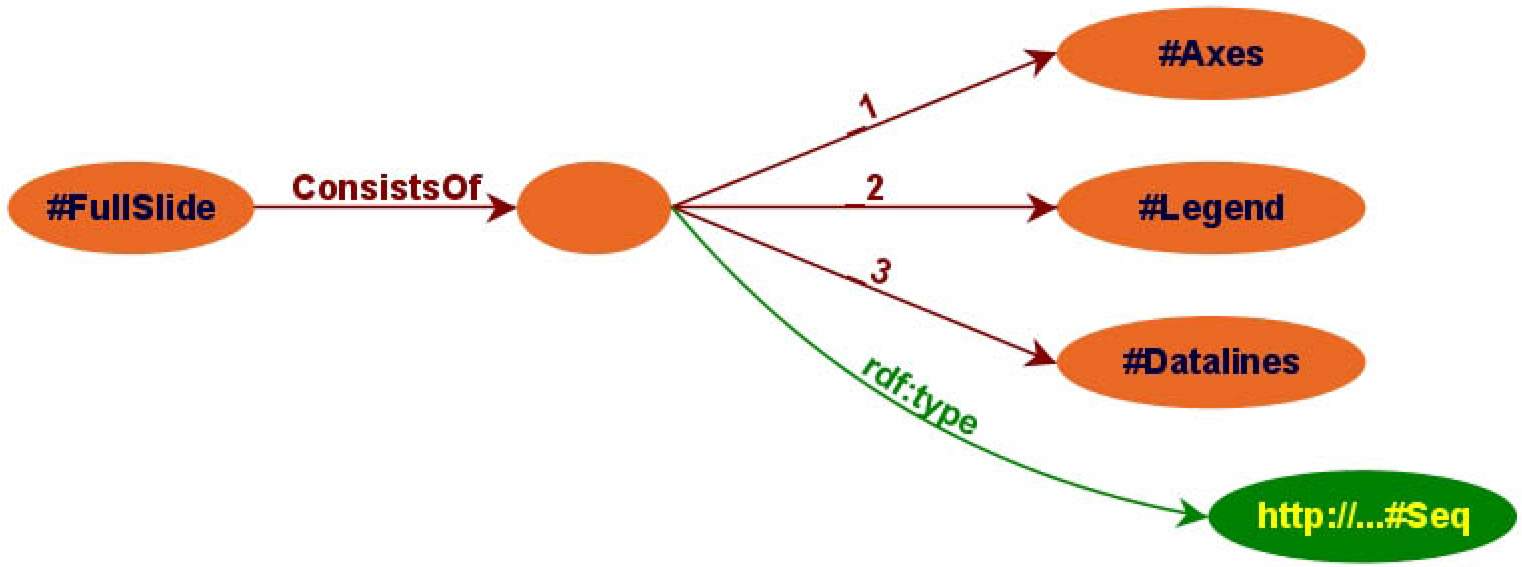
- We used the following statement:
  - “the full slide is a «thing» that consists of axes, the legend and the datalines”
- But we also want to express the constituents *in this order*
- Using blank nodes is not enough

Sequences (cont)

- One can use the predefined:
  - RDF class **Seq**
  - RDF properties **rdf:_1**, **rdf:_2**, ...
- The *agreed semantics* is of a sequential containment



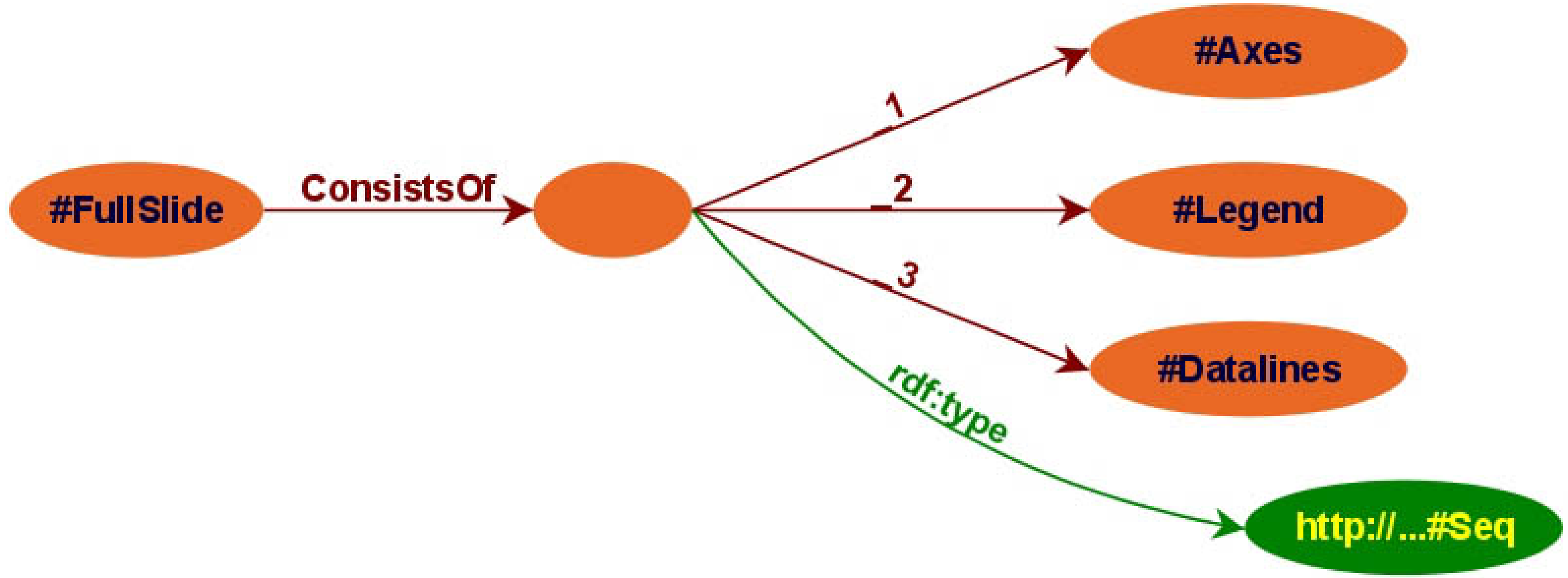
Sequences (cont)



- In RDF/XML:

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:ConsistsOf>
       <rdf:Description>
           <rdf:type rdf:resource="http:...rdf-syntax-ns
           <rdf:_1 rdf:resource="#Axes>
           ...
       </rdf:Description>
   </axsvg:ConsistsOf >
</rdf:Description/>
```

Sequences (cont)



- A simplified alternative (this is only syntax...):

```
<rdf:Description rdf:about="#FullSlide">
    <axsvg:ConsistsOf>
        <rdf:Seq>
            <rdf:li rdf:resource="#Axes>
            ...
        </rdf:Seq>
    </axsvg:ConsistsOf >
</rdf:Description/>
```

An Aside: Typed Nodes in RDF/XML

- A frequent simplification rule: instead of:

```
<rdf:Description rdf:about="http://...">
    <rdf:type rdf:resource="http://..../something#Class
    ...
</rdf:Description>
```

use:

```
<yourNameSpace:ClassName rdf:about="http://...">
    ...
</yourNameSpace:ClassName>
```

- Usage of **rdf:Seq** is based on this simplification rule

Other Containers

- **`rdf:Bag`**

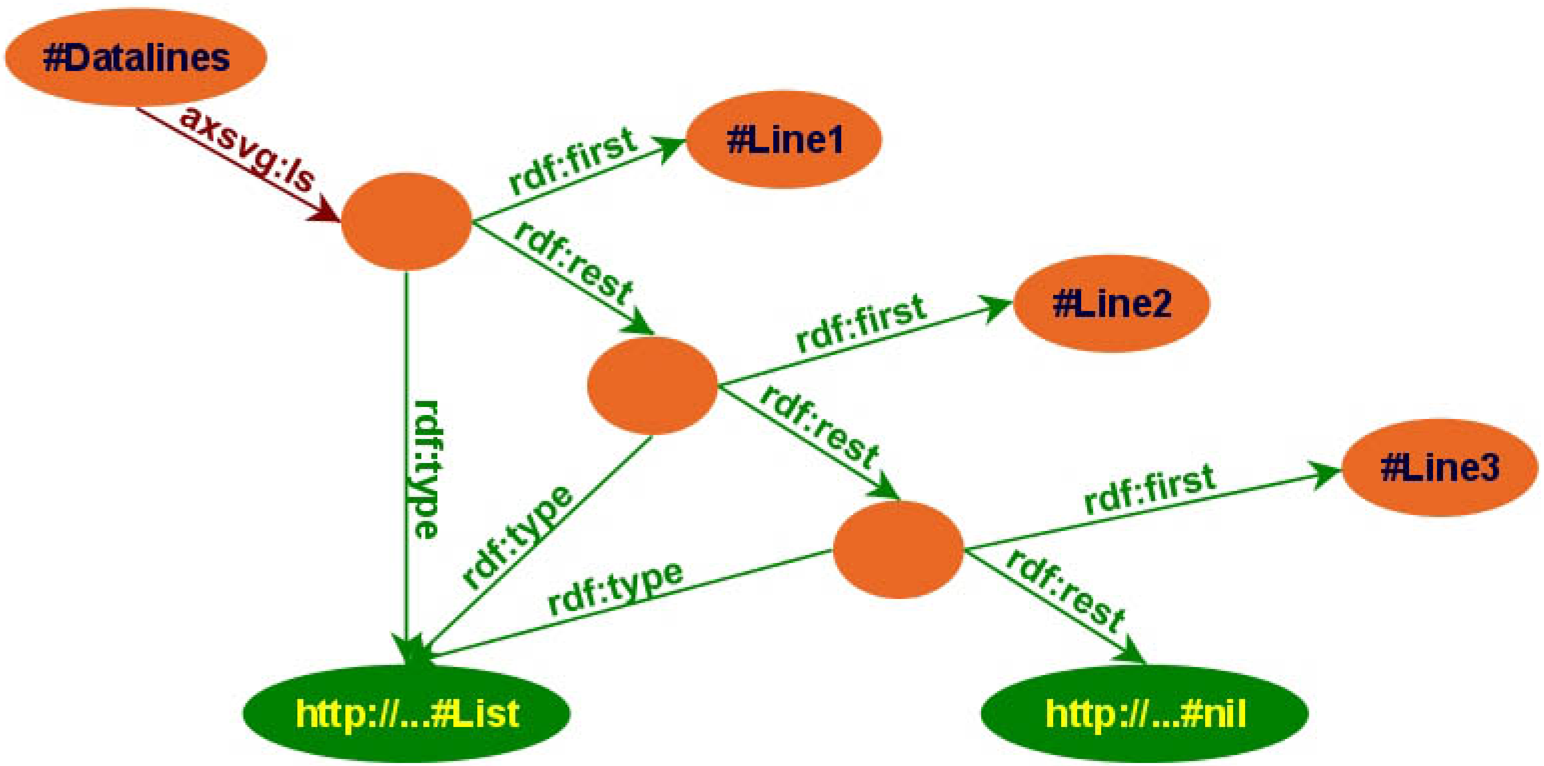
  a general bag, no particular semantics attached

- **`rdf:Alt`**

  attached semantics: *only one* of the constituents is “valid”

Collections (Lists)

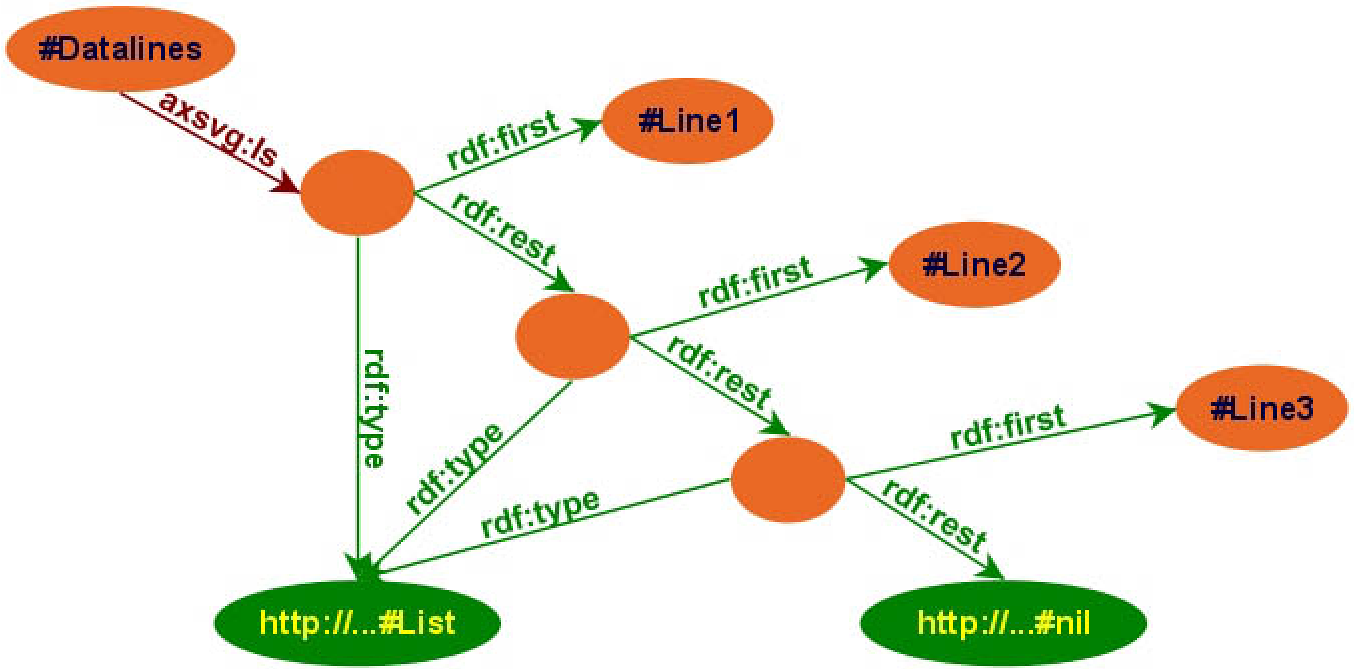
- RDF also includes *lists*
  - familiar structure for Lisp programmers...



The Same in RDF/XML

List in terms of XML:

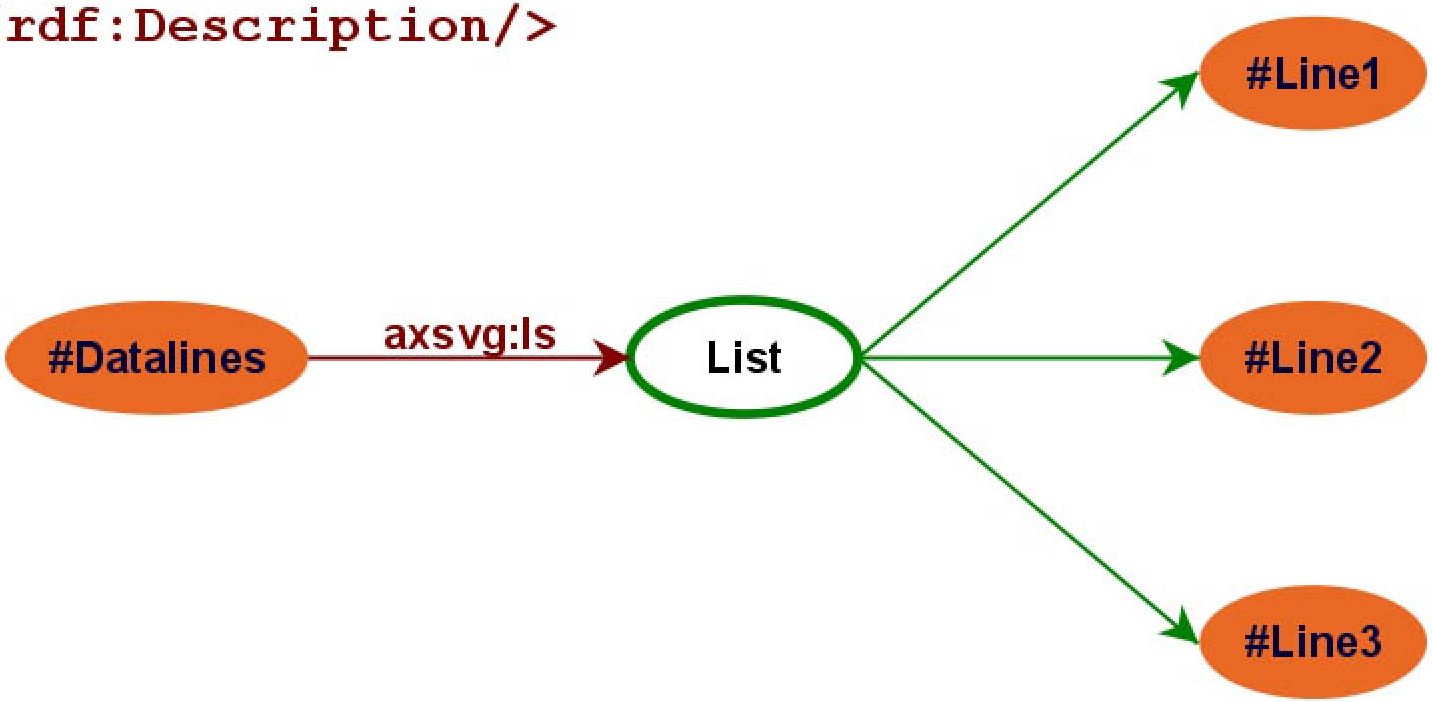
```
<rdf:Description rdf:about="#Datalines">
    <axsvg:Is rdf:parseType="Collection">
        <rdf:Description rdf:about="#Line1"/>
        <rdf:Description rdf:about="#Line2"/>
        <rdf:Description rdf:about="#Line3"/>
    </axsvg:Is >
</rdf:Description/>
```



Our Graphical Shorthand

(To simplify the images...)

```
<rdf:Description rdf:about="#Datalines">
    <axsvg:Is rdf:parseType="Collection">
        <rdf:Description rdf:about="#Line1"/>
        <rdf:Description rdf:about="#Line2"/>
        <rdf:Description rdf:about="#Line3"/>
    </axsvg:Is >
</rdf:Description/>
```



Some Words of Warning

- RDF/XML introduces a number of simplifications
  - usage of **`rdf:li`** instead of **`rdf:_1`**, **`rdf:_2`**, ...
  - usage of **`rdf:parseType`** instead of **`rdf:first`**, **`rdf:rest`**, ...
  - etc.
- This can be deceptive when using, e.g., RDFLib:
  - the triples in the Triple Store are the "real" ones!
    - i.e., **`rdf:_1`**, **`rdf:_2`** and *not* **`rdf:li`**
  - **`rdf:Seq`** does not appear directly
    - instead, a (possibly blank) node with a **`rdf:type`** property
  - etc.
- *Never forget: only the graph is "real", the rest is convenience!*

PART III: RDF Vocabulary Description Language
(a.k.a. RDFS)

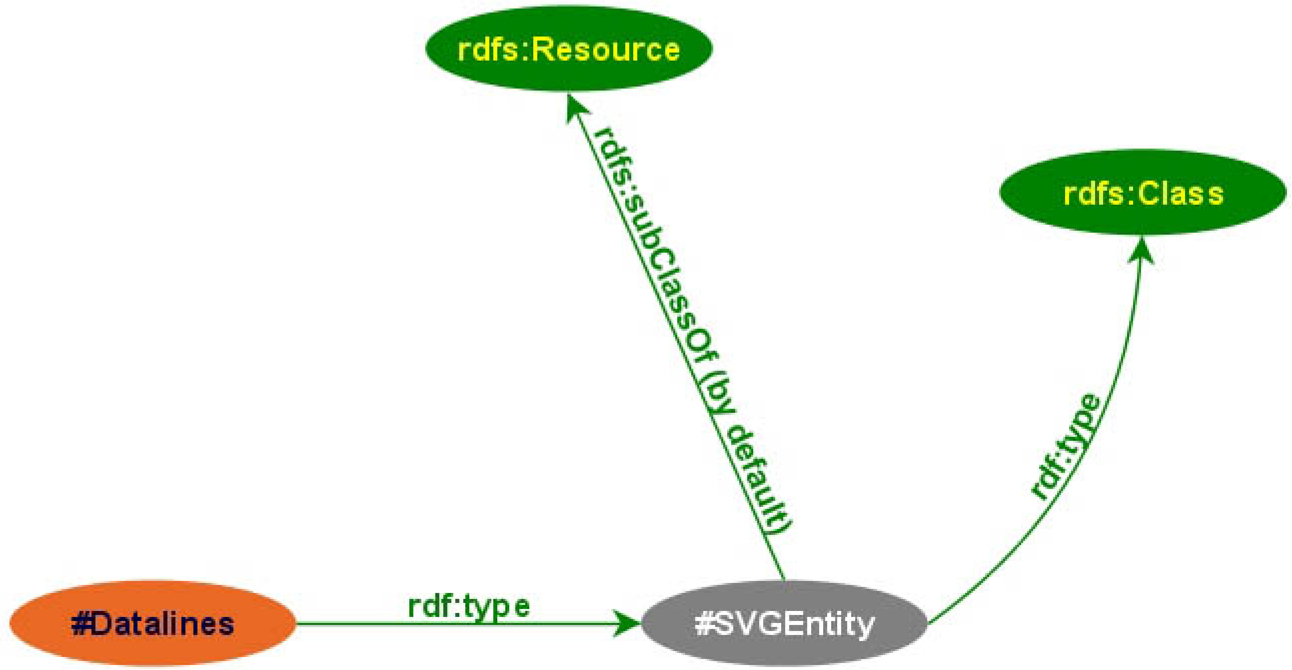
Back to Typing: RDF Schemas

- Adding metadata and using it from a program works...
- ... provided the program *knows* what terms to use!
- We used terms like:
  - **`Chart`**, **`LabelledBy`**, **`IsAnchor`**, ...
  - **`ChartType`**, **`GraphicsType`**, ...
  - etc
- Are they all known? Are they all correct?
- It is a bit like defining record types for a database
- This is where RDF Schemas come in
  - officially: “RDF Vocabulary Description Language”

Classes, Resources, …

- RDFS defines the terms of *resources* and *classes*:
  - everything in RDF is a "resource"
  - "classes" are also resources, but…
  - they are also a collection of possible resources (i.e., individuals)
- Relationships are defined among classes/resources:
  - "typing": an individual belongs to a specific class
  - "subclassing": instance of one is also the instance of the other
    - a bit like in object-based programming…
    - …but the same resource can have several types

Classes, Resources in RDF



- RDFS defines **`rdfs:Resource`**, **`rdfs:Class`** as nodes, …
  … **`rdf:type`**, **`rdfs:subClassOf`** as properties
- User should create RDF Schema file for the user types
- (Note: RDFS is also RDF!)

Schema Example in RDF/XML

- In `axsvg-schema.rdf` ("application's data types"):

```
<rdf:Description rdf:ID="SVGEntity">
  <rdf:type
   rdf:resource="http://www.w3.org/2000/01/rdf-schema#C
  />
</rdf:Description>
```

- In the rdf data on a specific graphics ("using the type"):

```
 <rdf:Description rdf:about="#Datalines">
    <rdf:type rdf:resource="axsvg-schema.rdf#SVGEntity'
 </rdf:Description>
```

Schema Example in RDF/XML (alt.)

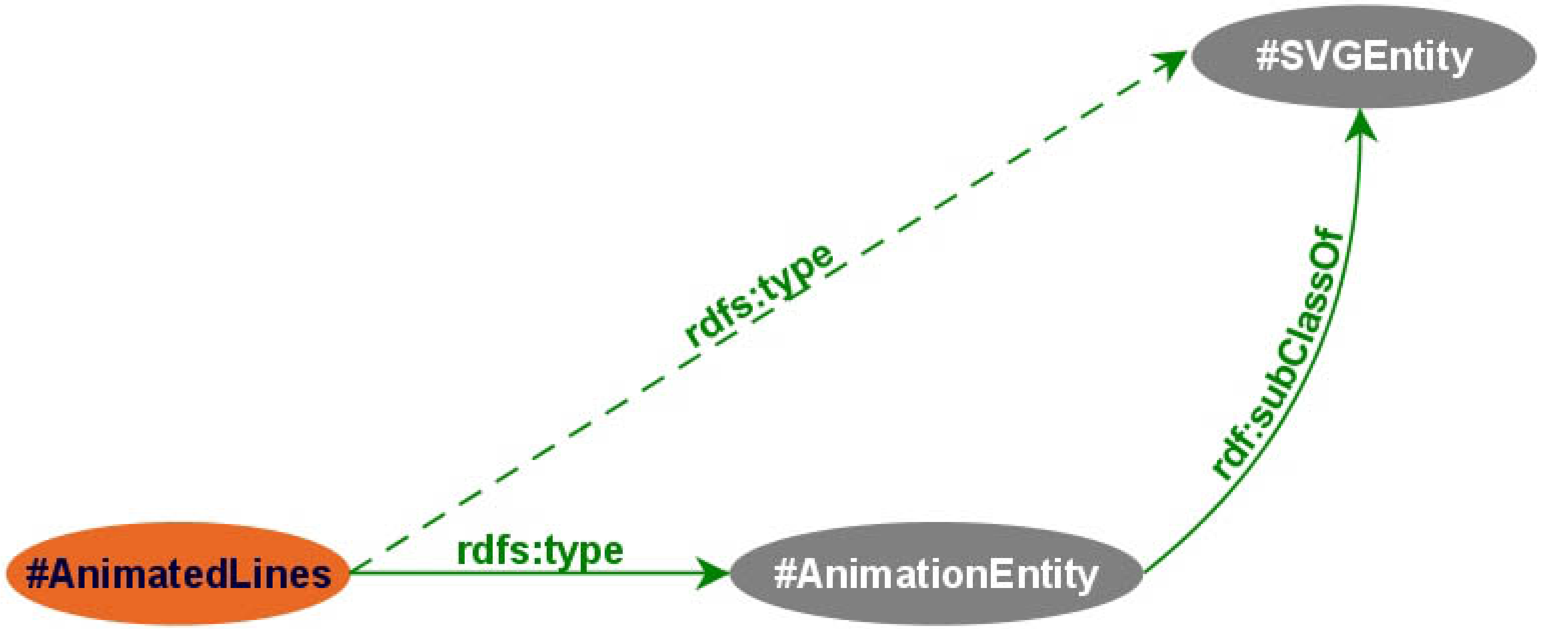
- In **axsvg-schema.rdf** (remember the simplification rule):

```
<rdfs:Class rdf:ID="SVGEntity">
   ...
</rdfs:Class>
```

- In the rdf data on a specific graphics:

```
<rdf:RDF xmlns:axsvg="axsvg-schema.rdf#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-r
    <axsvg:SVGEntity rdf:about="#Datalines">
       ...
    </axsvg:SVGEntity>
```

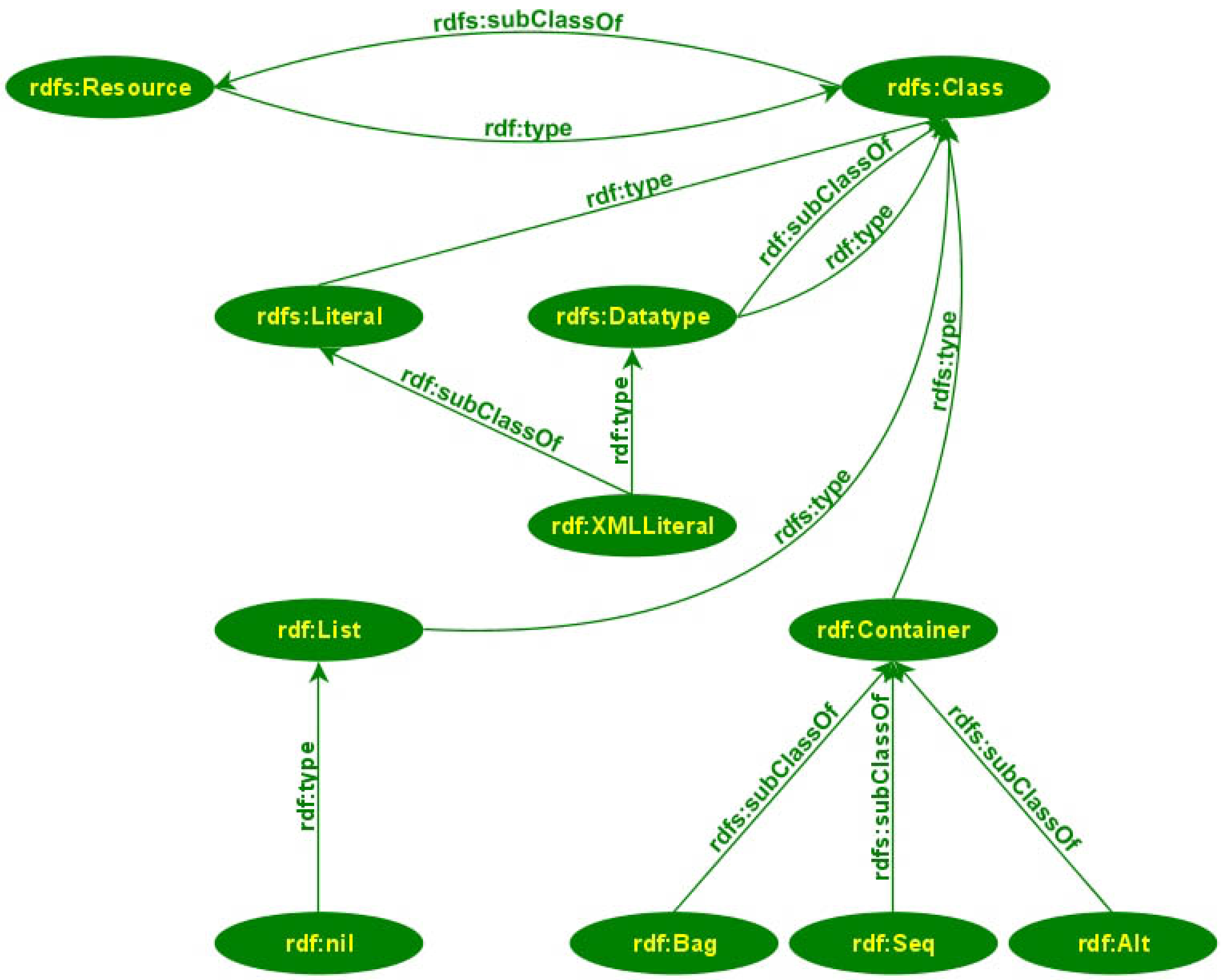
Inferred Properties



(#AnimatedLines rdf:type #SVGEntity)

- is *not* in the original RDF data…
- …but can be *inferred* from the RDFS rules
- Better RDF environments will return that triplet, too

Main RDFS Classes



Properties (Predicates)

- Property is a special class (`rdf:Property`)
  - i.e., properties are also resources
- Properties are constrained by their range and domain
  - i.e., what individuals can be on the "left" or on the "right"
- There is also a possibility for a "sub-property"
  - all resources bound by the "sub" are also bound by the other

Properties (cont.)

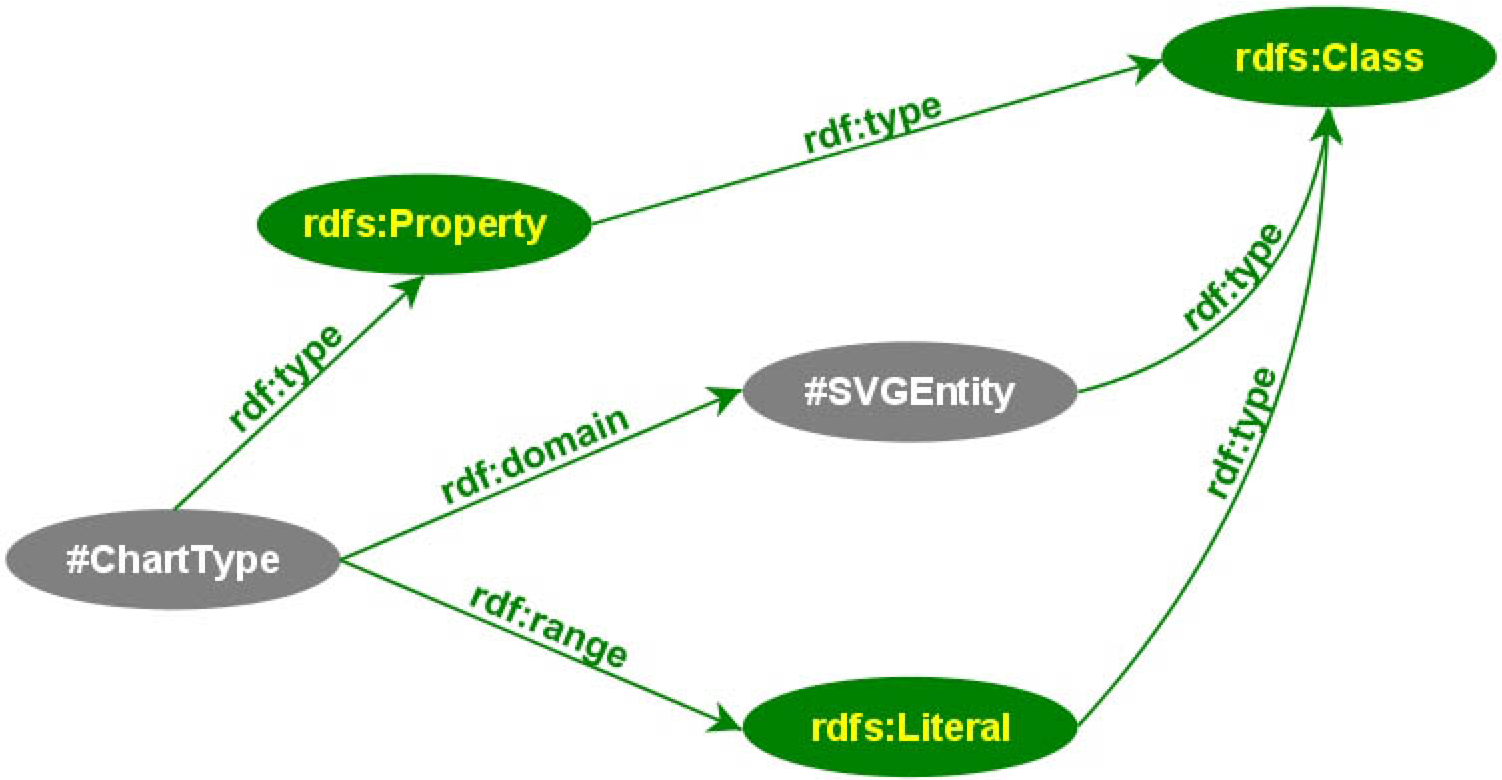
- Properties are also resources…
- So properties of properties can be expressed as… …RDF properties ☺
  - this twists your mind a bit, but you will get used to it
- For example:

  `(P rdfs:range C)` means:

  1. **`P`** is a property
  2. **`C`** is a class instance
  3. when using **`P`**, the "object" *must be* an individual in **`C`**

  - this is an RDF statement with subject **`P`**, object **`C`** and property **`rdfs:range`**

Property Specification Example



- Note that one cannot define *what* literals can be used
- This requires ontologies (see later)

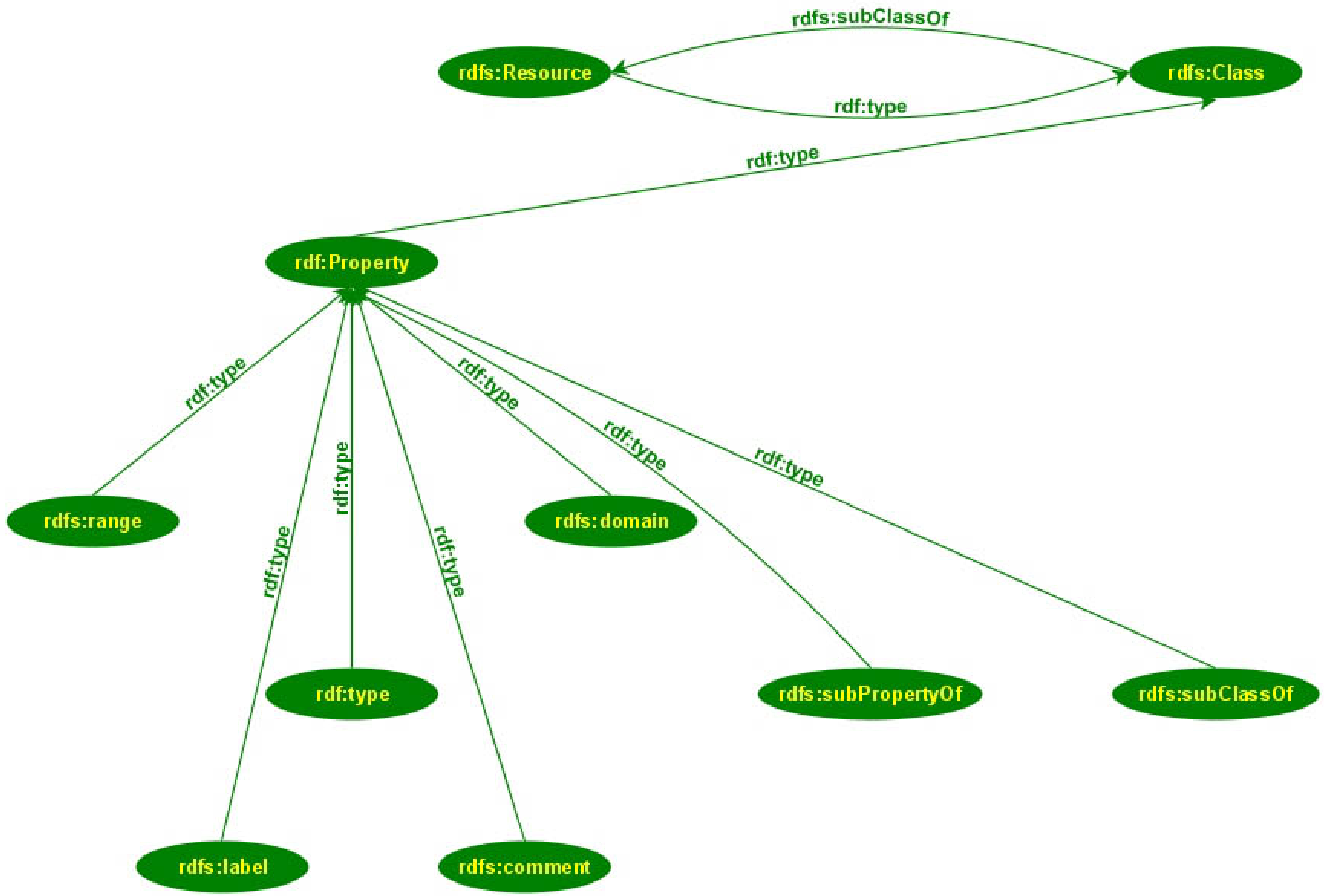
Property Specification in XML

Same example in XML/RDF:

```
<rdfs:Property rdf:ID="ChartType">
   <rdf:domain rdf:resource="#SVGEntity"/>
   <rdf:range rdf:resource="&rdfs;Literal"/>
</rdfs:Property>
```

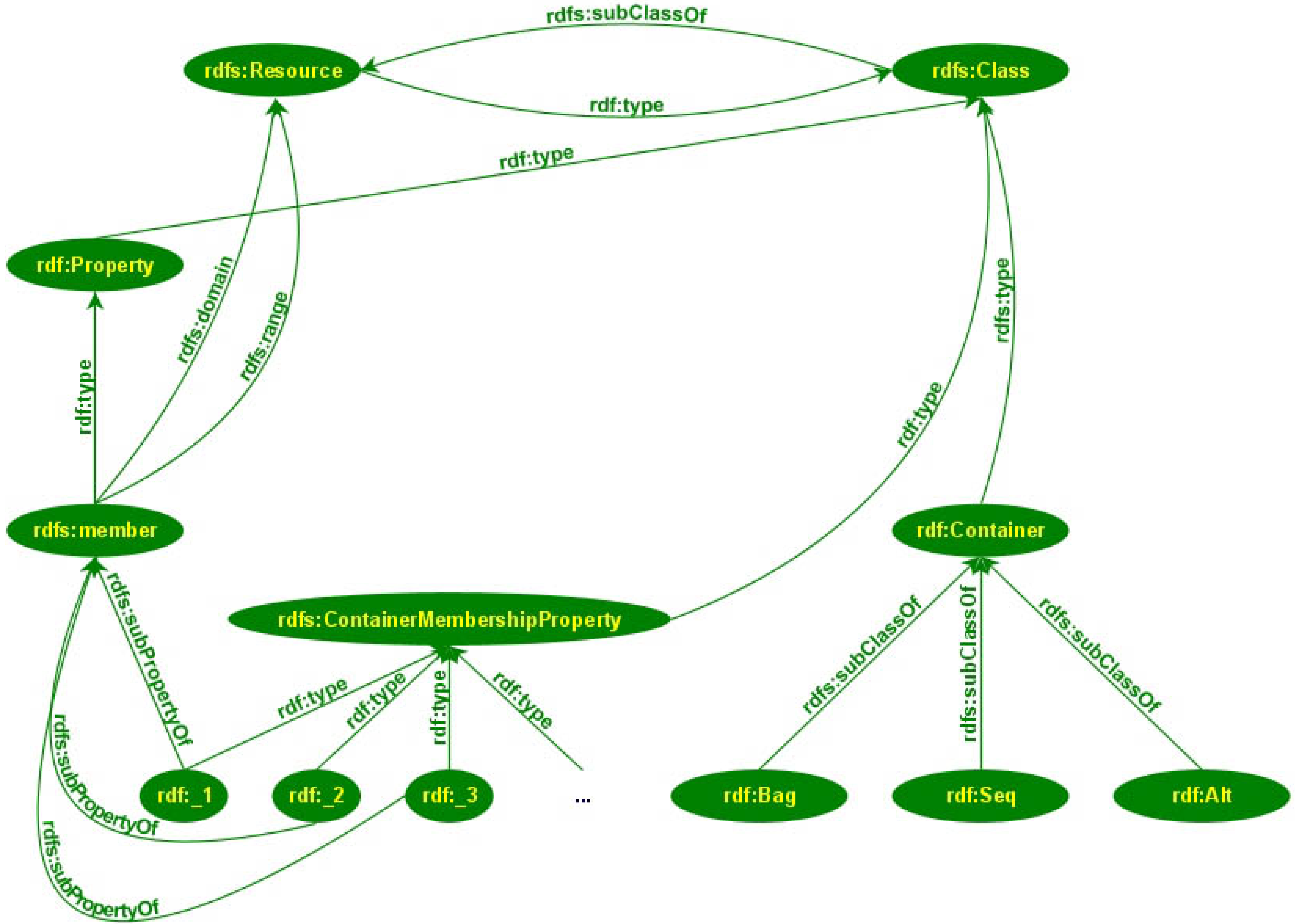
(note the usage of an entity)

Main RDFS Properties



Property hierarchy Property domains and ranges

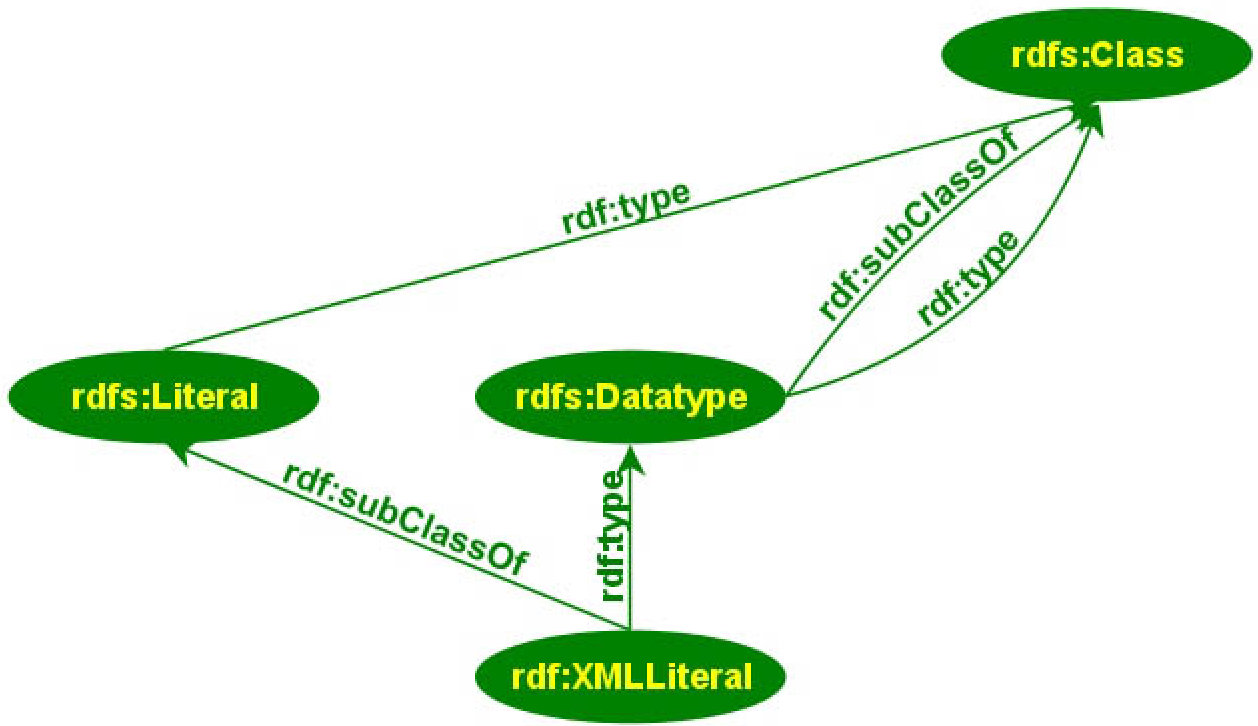
Collections/Containers in RDFS



Containers Collections

Literals

- Literals may have a data type
  - floats, int, etc
  - all types defined in XML Schemas
- Formally, data types are separate RDFS classes
- Full XML fragments may also be literals



Literals in RDF/XML

- Typed literals:

```
<rdf:Description rdf:about="#Datalines">
   <axsvg:IsAnchor
    rdf:datatype="http://www.w3.org/2001/XMLSchema#bool
        false
   </axsvg:IsAnchor>
</rdf:Description/>
```

- XML Literals:

```
<rdf:Description rdf:about="#Datalines">
   <axsvg:SVGContent rdf:parseType="Literal"
      xmlns:svg="http:....">
          <svg:line x1="..."/>
          <svg:path d="..."/>
          ...
   </axsvg:SVGContent>
</rdf:Description/>
```

The Power of XML Literal

- Using XML literals might be extremely powerful
- Makes it possible to “bind” RDF resources with XML vocabularies:

```
<rdf:Description rdf:about="#Path">
   <axsvg:algorithmUsed rdf:parseType="Literal"
        <math xmlns="...">
          <apply>
            <laplacian/>
            <ci>f</ci>
          </apply>
        </math>
   </axsvg:algorithmUsed>
</rdf:Description/>
```

PART IV: RDF(S) in Practice

Small Practical Issues

- RDF/XML files have a registered Mime type:

  `application/rdf+xml`

- Recommended extension: `.rdf`

RDF/XML has its Problems

- RDF/XML was developed in the “prehistory” of XML
  - e.g., even namespaces did not exist!
- Coordination was not perfect, leading to problems
  - the syntax cannot be checked with XML DTD-s
  - XML schemas are also a problem
  - encoding is verbose and complex
    - (eg, simplifications lead to confusions)

  But there is too much legacy code ☹
- Don’t be influenced (and set back...) by the XML format
  - the important point is the *model*, XML is just syntax
  - other “serialization” methods may come to the fore

Binding RDF to an XML Resource

- You can use the `rdf:about` as a URI for external resources
  - i.e., store the RDF as a separate file
- You may add RDF to XML directly (in its own namespace)
  - e.g., in SVG:

```
<svg ...>
  ...
  <metadata>
    <rdf:RDF xmlns:rdf="http://../rdf-syntax-ns#'
      ...
    </rdf:RDF>
  </metadata>
  ...
</svg>
```

RDF/XML with XHTML

- XHTML is still based on DTD-s (lack of entities in Schemas)
- RDF within XHTML's header does not validate…
- Currently, people use
  - `link/meta` in the header (perfectly o.k.!)
    - using conventions instead of namespaces in metas
  - put RDF in a comment (e.g., Creative Commons)
- XHTML 2.0 will have a separate 'metadata' module
  - essentially, the current meta/link elements are extended
  - one can define "triplets" using this formalism
  - in fact, a new RDF serialization… (like RDF/XML and n3)

RDF Can Also be Generated

- There might be conventions to use in XHTML…
  - eg, by using class names
- … and then *generate* RDF automatically
- There are tools and developments in this direction

Programming Practice

- We have already seen how to retrieve triples in RDFLib:

```
# import the libraries
from rdflib.TripleStore import TripleStore
from rdflib.URIRef import URIRef
# resource for a specific URI:
subject = URIRef("URI_of_Subject")
# create the triple store
triples = TripleStore()
# parse an RDF file and store it in the triple store
triples.load("membership.rdf")
# do something with (p,o) pairs
for (p,o) in triples.predicate_objects(subject) :
    do_something(p,o)
```

Programming Practice (cont)

- One can also edit triples, save it to an XML file, etc:

```
# add a triple to the triple store
triples.add((subject,pred,object))
# remove it
triples.remove_triples((subject,pred,object))
# save it in a file in RDF/XML
triples.save("filename.rdf")
```

- It is very easy to start with this
- Does not have (yet) powerful schema processing
  - no “inferred” property feature, for example
- You can get RDFLib at: `http://rdflib.net`

Jena

- RDF toolkit in Java from HP's Bristol lab
- The RDFLib features are all available:

```
// create a model (a.k.a. Triple Store in python)
Model model=new ModelMem();
Resource subject=model.createResource("URI_of_Subject'
// 'in' refers to the input file
model.read(new InputStreamReader(in));
StmtIterator iter=model.listStatements(subject,null,nu
while(iter.hasNext()) {
   st = iter.next();
   p = st.getProperty();
   o = st.getObject();
   do_something(p,o);
}
```

Jena (cont)

- But Jena is *much* more than RDFLib
  - it has a large number of classes/methods
    - listing, removing associated properties, objects
    - comparing full RDF graphs
    - manage typed literals
    - mapping **`Seq`**, **`Alt`**, etc. to Java constructs
    - etc.
  - *it has an "RDFS Reasoner"*
    - a new model is created with an associated RDFS file
    - all the "inferred" properties, types are accessible
    - errors are checked
  - and more…
- Of course, it is much bigger and more complicated…
- Is available at: **`http://jena.sourceforge.net/`**

Lots of Other tools

- There are other tools:
  - **RDFSuite**: another Java environment (from ICS-FORTH) also includes RDFS
  - **Sesame**: Java based storage and query for RDF and RDFS
  - **RDFStore**: a Perl API
  - **Redland**: RDF Framework for C
  - **RAP**: RDF Framework for PHP
  - **SWI-Prolog**: RDF Framework for Prolog
  - **Kowari** and **Tucana**: triple based database systems
    - they have Jena interfaces, too
  - etc.
- You can always start by:
  `http://www.w3.org/RDF/#developers`

RDF or XML?

- **Q: For a specific application, should I use XML or RDF?**
- **A: It depends…**
  - XML's model is
    - a tree, i.e., a strong hierarchy
    - applications may rely on hierarchy position (e.g., `li` in HTML)
    - relatively simple syntax and structure
    - not easy to *combine* trees
  - RDF's model is
    - a *loose* collections of relations
    - applications may do “database”-like search
    - not easy to recover hierarchy
    - easy to combine relations in one big collection
    - great for the integration of heterogeneous information

Where Does the Metadata Come From?

- Q: Should we expect the author to type in all this metadata?
- A: Partially, but:
  - part of the metadata information is present in the tool…

    …but thrown away at output
    - e.g., a business chart can be generated by a tool…

      …it "knows" the structure, the classification, etc. of the chart

      …but, usually, this information is lost

      …storing it in metadata is easy!
  - "SW-aware" authoring tools will be of a great help

PART V: Ontologies (OWL)

Ontologies

- RDFS is useful, but does not solve all the issues
- Complex applications may want more possibilities:
  - can a program *reason* about some terms? E.g.:
    - "if «A» is left of «B» and «B» is left of «C», is «A» left of «C»?"
    - obviously true for humans, not obvious for a program …
    - … programs should be able to *deduce* such statements
  - if somebody else defines a set of terms: are they the same?
    - obvious issue in an international context
  - *construct* classes, not just name them
  - restrict a property range *when used for a specific class*
  - etc.

Ontologies (cont.)

- The Semantic Web needs a support of *ontologies*:

"defines the concepts and relationships used to describe and represent an area of knowledge"

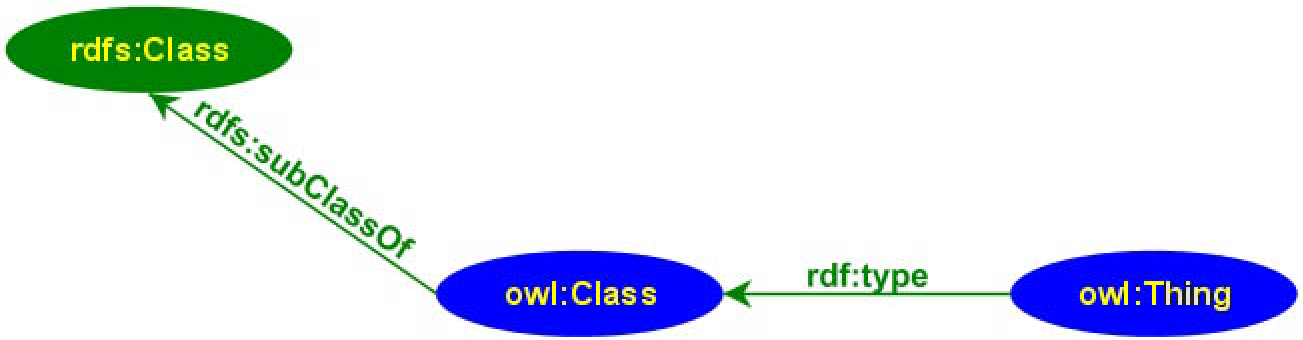
- We need a *Web Ontologies Language* to define:
  - the terminology used in a specific context
  - more constraints on properties
  - the logical characteristics of properties
  - the equivalence of terms across ontologies
  - etc

W3C's Ontology Language (OWL)

- A layer *on top* of RDFS with additional possibilities
- Outcome of various projects:
  1. a DARPA project: DAML
  2. a EU project: OIL
  3. an attempt to merge the two: DAML+OIL
  4. the latter was submitted to W3C
  5. lots of coordination with the core RDF work
  6. recommendation since early 2004

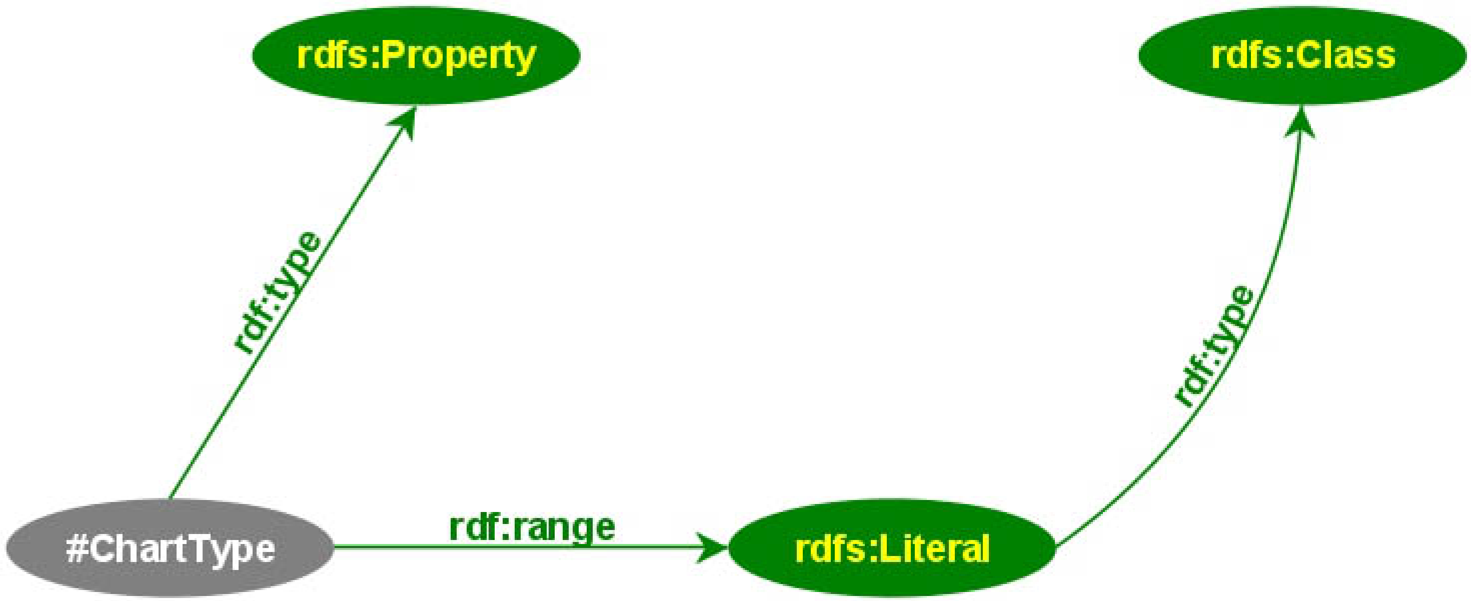
Classes in OWL

- In RDFS, you can subclass existing classes…
  … but, otherwise, that is all you can do
- In OWL, you can *construct* classes from existing ones:
  - enumerate its content
  - through intersection, union, complement
  - through property restrictions
- To do so, OWL introduces its own `Class`…
  … and `Thing` to differentiate the *individuals* from the *classes*



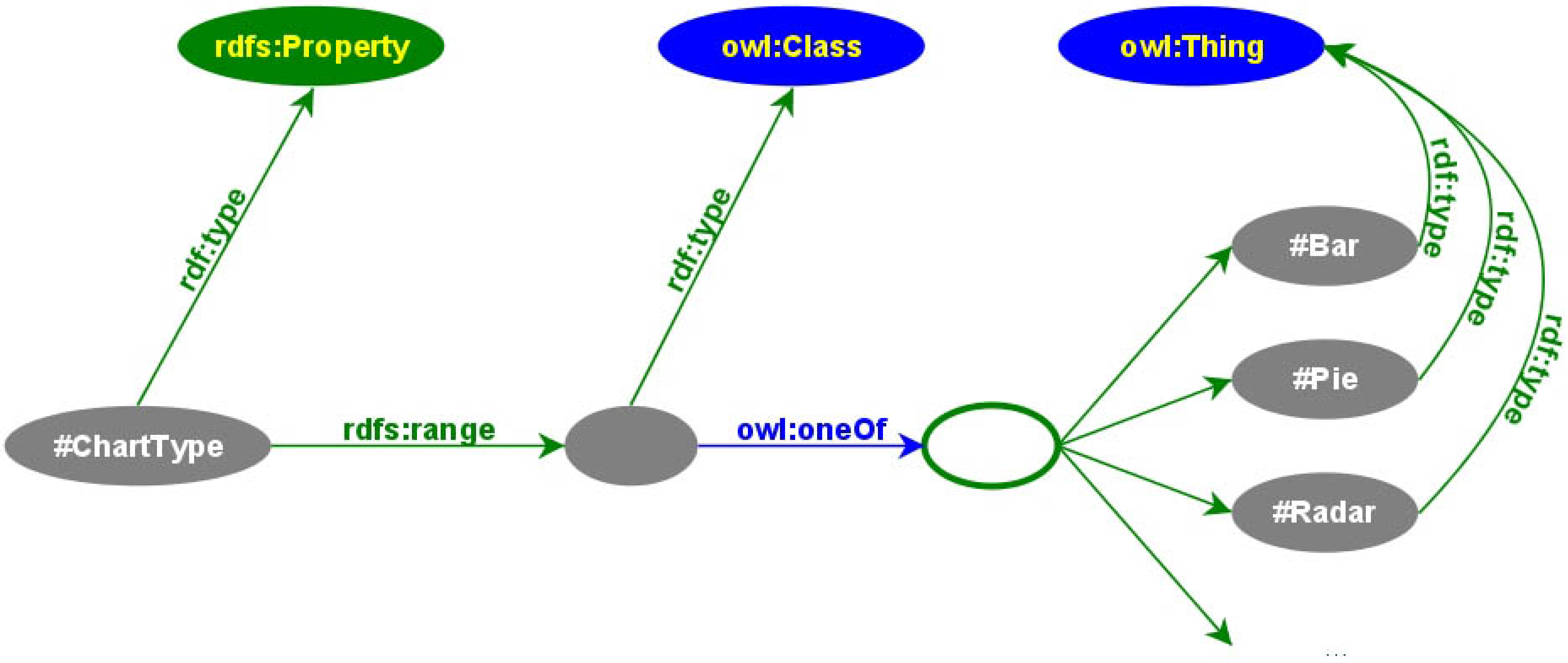
Need for Enumeration

- Remember this issue?
  - one can use XML Schema types to define an enumeration for **`ChartType`**, but…
  - …wouldn't it be better to do it *within* RDF?



(OWL) Classes can be Enumerated

- The OWL solution, where possible content is explicitly listed:



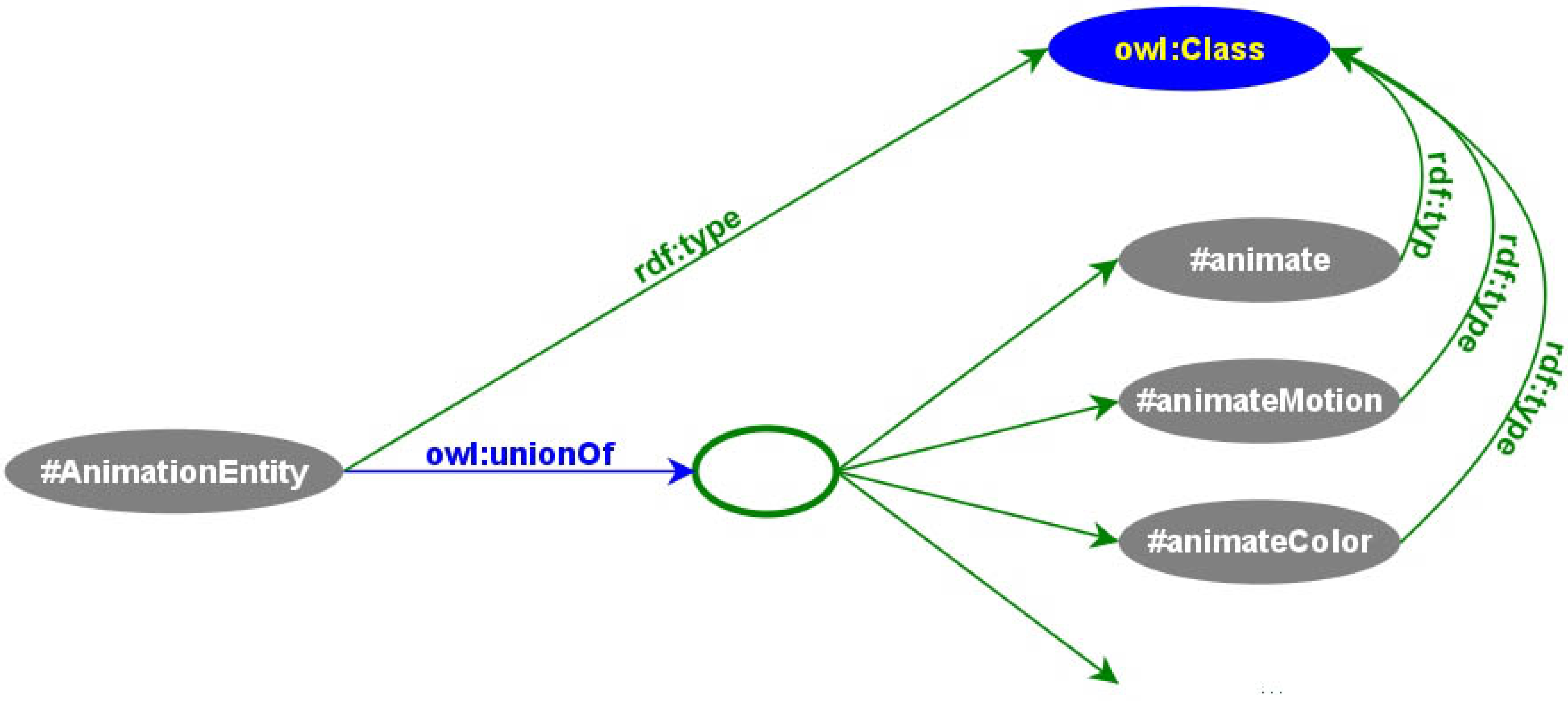
Same in RDF/XML

Enumeration in XML:

```
<rdf:Property rdf:ID="ChartType">
   <rdf:range>
       <owl:Class>
           <owl:oneOf rdf:parseType="Collection">
               <owl:Thing rdf:ID="Bar"/>
               <owl:Thing rdf:ID="Pie"/>
               <owl:Thing rdf:ID="Radar"/>
               …
           </owl:oneOf>
       </owl:Class>
   </rdf:range>
</rdf:Property>
```

Union of Classes

- Essentially, set-theoretical union:



Same in RDF/XML

Union in XML:

```
<owl:Class rdf:ID="AnimationEntity">
   <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#animate"/>
       <owl:Class rdf:about="#animateMotion"/>
       <owl:Class rdf:about="#animateColor"/>
       ...
   </owl:unionOf>
</owl:Class>
```

- Other possibilities: `complementOf`, `intersectionOf`

Property Restrictions

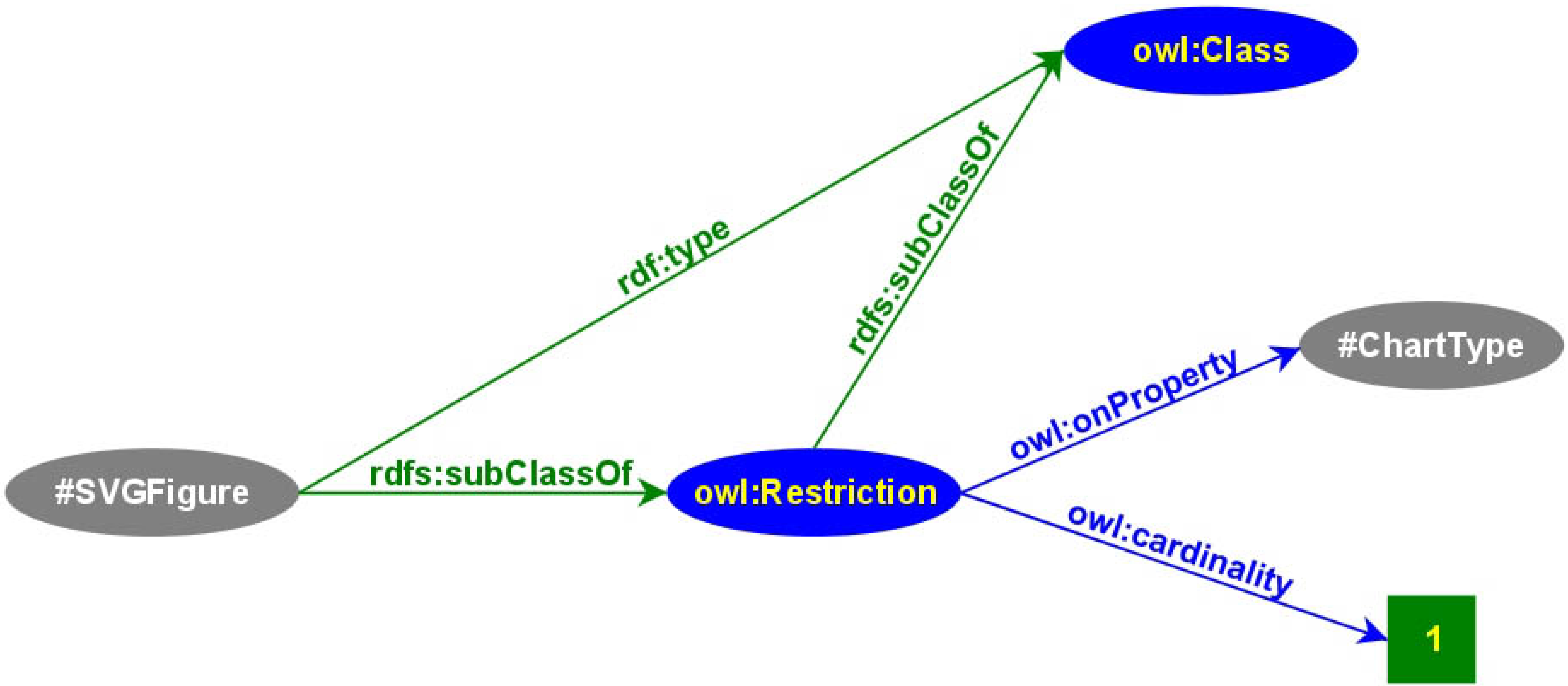
- (Sub)classes can be created by restricting the behaviour of a property *on that class*
- Restriction may be by:
  - value constraints (i.e., further restrictions on the range)
    - *all* values must be from a class
    - *at least one* value must be from a class
  - cardinality constraints
    (i.e., how many times the property can be used on an instance?)
    - minimum cardinality
    - maximum cardinality
    - exact cardinality

Property Restrictions (cont.)

- Formally:
  - **owl:Restriction** defines a blank node with restrictions
    - refer to the property that is constrained
    - define the restriction itself
  - one can, e.g., subclass from this node, or...
  - ...use intersection of several property constraints, or...
  - ...declare the class to be *equal* to it

Cardinality Restriction Example

- "A full SVG figure must have *one* chart type":



Same in RDF/XML

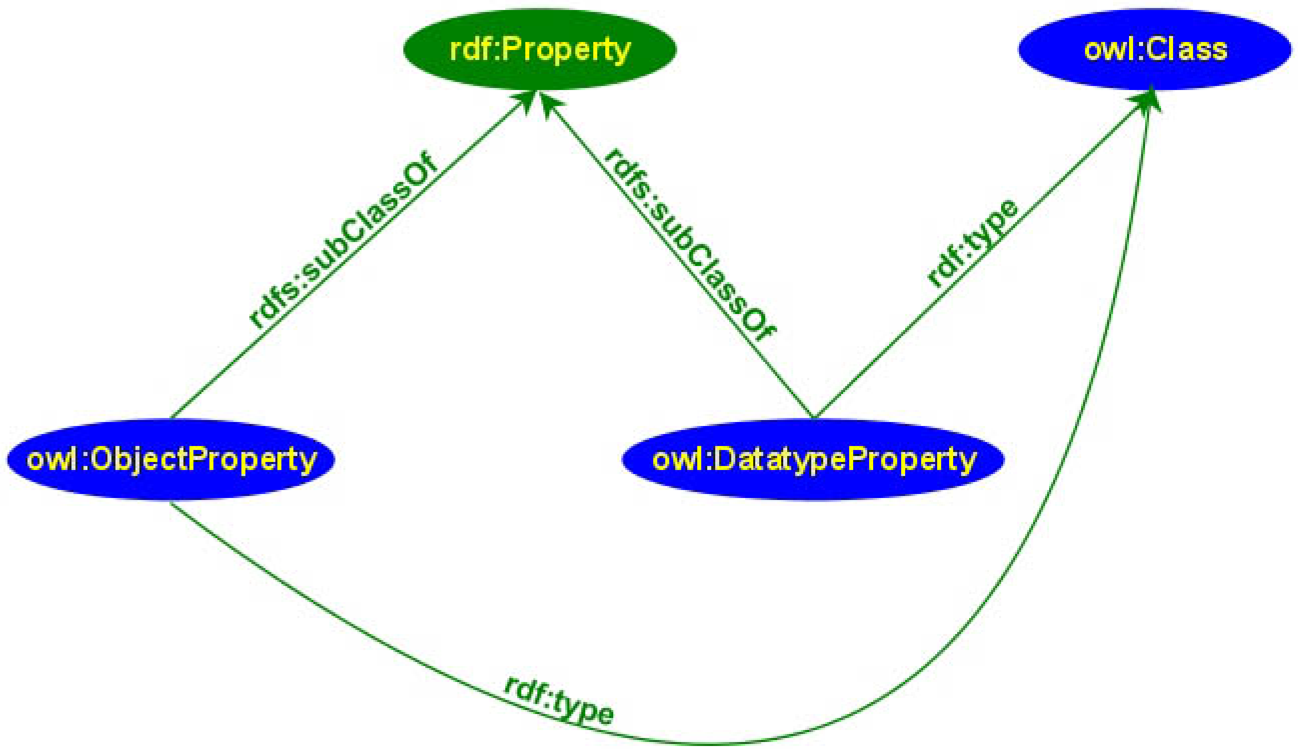
Cardinality constraint in XML:

```
<owl:Class rdf:ID="SVGFigure">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:about="#ChartType"/>
            <owl:cardinality
                rdf:dataype="...#nonNegativeInteger">
                1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- Note the usage of a typed literal
- `cardinality` could be replaced by:
  - `minCardinality`, `maxCardinality`
  - `someValuesFrom`, `allValuesFrom`

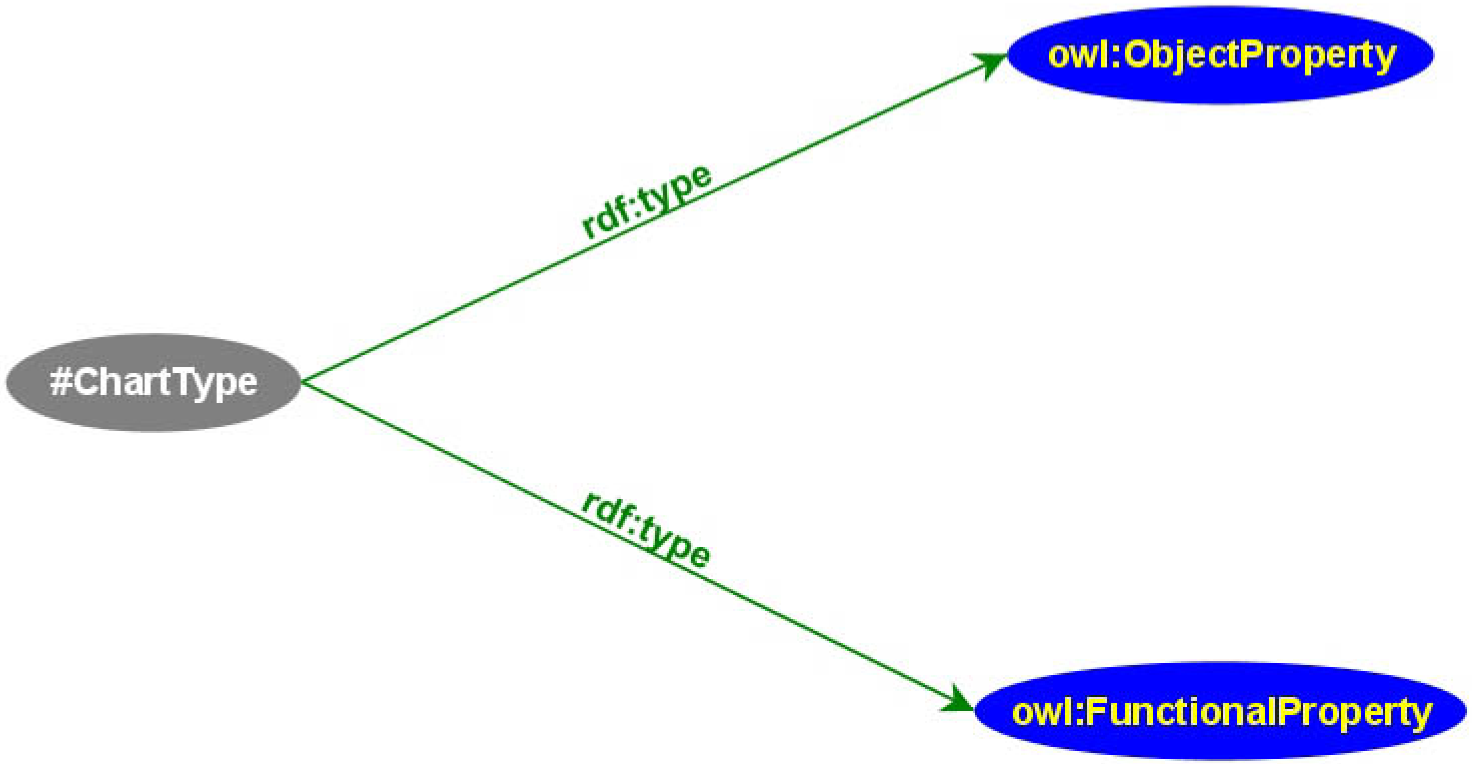
Property Characterization

- In RDFS, properties are constrained by domain and range
- In OWL, one can also characterize their *behaviour*
  - symmetric, transitive, functional, etc
- OWL separates data properties
  - "datatype property" means that its range are *typed* literals



Characterization Example

- An alternative for the cardinality=1 setting:
  (the difference is that this is valid everywhere, not restricted to a class only)



Same in RDF/XML

Characterization in XML:

```
<owl:ObjectProperty rdf:ID="ChartType">
   <rdf:type rdf:resource="...../#FunctionalProperty/>
</owl:ObjectProperty>
```

- Similar characterization possibilities:
  - `InverseFunctionalProperty`
  - `TransitiveProperty`, `SymmetricProperty`
- Range of `DatatypeProperty` can be restricted (using XML Schema)
- These features can be extremely useful for ontology based applications!

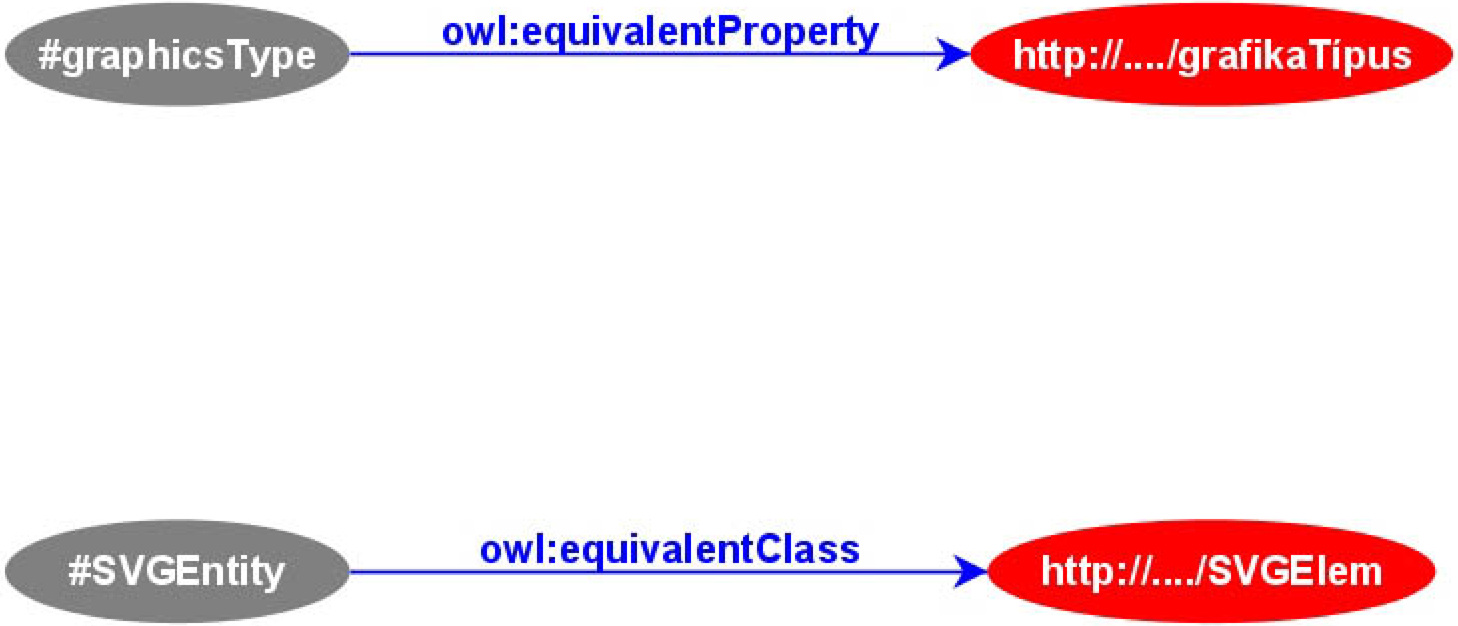
OWL: Additional Requirements

- Ontologies may be extremely a large:
  - their management requires special care
  - they may consist of several modules
  - come from different places and must be integrated
- Ontologies are *on the Web*. That means
  - applications may use several, different ontologies, or...
  - ... same ontologies but in different languages
  - equivalence of, and relations among terms become an issue

Term Equivalence/Relations

- For classes:
  - **owl:equivalentClass**: two classes have the same individuals
  - **owl:disjointWith**: no individuals in common
- For properties:
  - **owl:equivalentProperty**: equivalent in terms of classes
  - **owl:inverseOf**: inverse relationship
- For individuals:
  - **owl:sameAs**: two URI refer to the same individual (e.g., concept)
  - **owl:differentFrom**: negation of **owl:sameAs**

Example: Connecting to Hungarian



Another Use of Equivalence

- Equivalence can also be used for a *complete* specification of a class:

```
<owl:Class rdf:ID="SVGFigure_Chart">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:about="#ChartType"/>
            <owl:cardinality
                rdf:dataype="...#nonNegativeInteger">
                1
            </owl:cardinality>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

Versioning, Annotation

- Special class `owl:Ontology` with special properties:
  - `owl:imports`, `owl:versionInfo`, `owl:priorVersion`
  - `owl:backwardCompatibleWith`, `owl:incompatibleWith`
  - `rdfs:label`, `rdfs:comment` can also be used
- One instance of such class is expected in an ontology file
- Deprecation control:
  - `owl:DeprecatedClass`, `owl:DeprecatedProperty` types

However: Ontologies are Hard!

- A full ontology-based application is a very complex system
  - in fact, it is turning mathematical logic into a program
- Hard to implement, heavy to run...
- ... and not all applications may need it!
- Three layers of OWL are defined: Lite, DL, and Full
  - increasing level of complexity and expressiveness
    - “Full” is the whole thing
    - “DL (Description Logic)” restricts Full in some respects
    - “Lite” restricts DL even more

OWL Full

- No constraints on the various constructs
  - **owl:Class** is equivalent to **rdfs:Class**
  - **owl:Thing** is equivalent to **rdfs:Resource**
- This means that:
  - **Class** can also be an individual
    - it is possible to talk about class of classes, etc.
  - one can make statements on RDFS constructs
    - declare **rdf:type** to be functional…
  - etc.
- A real superset of RDFS

OWL Description Logic (DL)

- `owl:Class`, `owl:Thing`, `owl:ObjectProperty`, and `owl:DatatypePropery` are *strictly separated*
  - i.e., a class *cannot* be an individual of another class
- No mixture of `owl:Class` and `rdfs:Class` in definitions
  - essentially: use OWL concepts only!
- No statements on RDFS resources
- No characterization of datatype properties possible
- No cardinality constraint on transitive properties
- Some restrictions on annotations
- *Goal: maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure is realizable*

OWL Lite

- All of DL's restrictions, plus some more:
  - class construction can be done *only* through:
    - intersection
    - property constraints
- *Goal: provide a minimal useful subset, easily implemented*
  - simple class hierarchies can be built
  - property constraints and characterizations can be used

"Description Logic"

- The term refers to an area in knowledge representation
  - there are several variants of Description Logic
  - i.e.: OWL DL ≠ Description Logic…
  - but OWL DL is an embodiment of a Description Logic
- Traditional DL terms sometimes used (by experts…):
  - "named objects, concepts": definition of classes, individuals, …
  - "axioms": e.g., subclass or subproperty relationships, …
  - "facts": statements about indivudals (`owl:Thing`-s)

  but none of these are "standardized" in W3C…

DL Abstract Syntax

- There is also a non-XML based notation for OWL ("abstract syntax")
  - also used in the formal specification of OWL
  - it may become more widespread in future
  - currently only RDF/XML format is widely implemented
  - but AS → RDF/XML converters exist
  - e.g.:

```
Class(animate)
Class(animateMotion)
Class(animationEntity complete
  unionOf(animate animateMotion …)
)
```

Ontology Examples

- A possible ontology for our graphics example
  - on the borderline of DL and Full
- International country list
  - example for an OWL Lite ontology
- The big task is to *create* the ontologies
  - requires a good knowledge of the area to be described
  - some communities have similar expertise (eg, librarians)

PART VI: Future Developments

"Semantic Web Activity Phase 2"

- First phase (completed): core infrastructure
- Second phase: promotion and implementation needs
  - relevant working groups
  - outreach to user communities
    - life sciences
    - geospatial information systems
    - libraries and digital repositories
    - ...
  - intersection of SW with other technologies
    - Semantic Web Services
    - privacy policies
    - ...

"Best Practices" Work

- "Semantic Web Best Practices and Deployment"
  - recommendations for practical deployment
  - engineering guidelines
  - ontology/vocabulary development practices
  - educational material
  - effective demonstrations
  - information on applications
  - etc
- Goal is to increase awareness on SW
- W3C has just started work in this area

RDF Data Access (a.k.a. Queries)

- We used, in Python, the query:

```
# do something with (p,o) pairs
for (p,o) in triples.predicate_objects(subject) :
   do_something(p,o)
```

  "predicate_objects" returns a *subgraph*
- Applications may want more
  - i.e., return complex subgraph with parts missing
- Very important for large and *distributed* RDF depositories
- There are more than 20 RDF Query languages

Data Access Example

- One may want something like:

```
SELECT (a,b)
  WHERE [?x 'parent' a] and [b 'brother' ?x]
```

(ie, 'b is the uncle of a')

- W3C has just started a standardization work in this area
  - precise relationships to XML Query has to be defined
  - concentrates also on protocols to extract subgraphs
    - e.g., using SOAP

RDF API-s

- We have seen Jena and RDFLib
- There are lots of other programming environments
  - Redland, RDFStore, RAP, etc.
- Each use their own "view" on binding RDF to programming concepts
- A standardization would enhance interoperability
  - similar to the DOM Specification for XML:
    - common vocabulary is developed in terms of OMG's IDL
    - there are IDL "bindings" to C, C++, Python, etc.
- W3C may initiate a standardization work in this area, or ...
- ... leave it to others to standardize in practice
  - (it is not clear whether this is the task of W3C)

Semantic Web Services Interest Group

- Forum on the integration of Web Services and the Semantic Web, e.g.
  - a more semantic oriented description of interfaces
  - constraint descriptions of choreographies
  - etc.
- Currently a forum, may lead to more specific standards (or influence work in other groups)
- Work has started only recently

PART VII: Available Documents, Tools

Available Specifications: Primers

RDF Primer
URI: http://www.w3.org/TR/rdf-primer

OWL Guide
URI: http://www.w3.org/TR/owl-guide/

RDF Test Cases
URI: http://www.w3.org/TR/rdf-testcases/

OWL Test Cases
URI: http://www.w3.org/TR/owl-test/

Available Specifications: RDF

RDF: Concepts and Abstract Syntax

URI: http://www.w3.org/TR/rdf-concepts/

Note: there is a previous Recommendation of 1999 that is superceeded by these

RDF Semantics

URI: http://www.w3.org/TR/rdf-mt/

Precise, graph based definition of the semantics

This is primarily for implementers

RDF/XML Serialization

URI: http://www.w3.org/TR/rdf-syntax-grammar/

N3 Serialization Primer

URI: http://www.w3.org/2000/10/swap/Primer

Note: this is not part of the W3C Recommendation track!

Available Specifications: Ontology

RDF Vocabulary Description Language (RDF Schema)
URI: http://www.w3.org/TR/rdf-schema/

OWL Overview
URI: http://www.w3c.org/TR/owl-features/

OWL Reference
URI: http://www.w3c.org/TR/owl-ref/

OWL Semantics and Abstract Syntax
URI: http://www.w3c.org/TR/owl-semantics/

OWL Use Cases and Requirements
URI: http://www.w3.org/TR/webont-req/

Some Books

- M. Dertouzos: The Unfinished Revolution (1995)
  - an early "vision" book (not only on the Semantic Web)
- T. Berners-Lee: Weaving the Web (1999)
  - another "vision" book
- S. Powers: Practical RDF, (2003)
- D. Fensel, J. Hendler: Spinning the Semantic Web (2003)
- G. Antoniu, F. van Harmelen: Semantic Web Primer (2004)
- ...

Further Information

- Full, interactive view of the RDFS and OWL definitions
  - requires an SVG client
- Bristol University has a huge list of documents, publications:
  - URI: http://www.ilrt.bristol.ac.uk/discovery/rdf/resources/
- DAML Ontology Library:
  - some of them are still in DAML, but being converted to OWL
  - URI: http://www.daml.org/ontologies/
- The SWAD-Europe project reports:
  - lots of information on RDF integration, for example
  - URI: http://www.w3.org/2001/sw/Europe/reports/intro.html
- W3C's Semantic Web home page is also a good start:
  - URI: http://www.w3.org/2001/sw/

Further Information (cont)

- References on Description logic:
  - Online courses: http://dl.kr.org/courses.html
  - A general introduction: http://www.inf.unibz.it/~franconi/dl/course/dlhb/dlhb-01.pdf
- Ontology Development 101
  - URI: http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
- OWL Reasoning Examples:
  - URI: http://owl.man.ac.uk/2003/why/latest/
- *Lots* of papers at WWW2003 and WWW2004

Public Fora at W3C

Semantic Web Interest Group

a forum for discussions on applications

URI: http://www.w3.org/RDF/Interest

RDF Logic

public (archived) mailing list for technical discussions

URI: http://lists.w3.org/Archives/Public/www-rdf-logic/

Some Tools

(Graphical) Editors

- IsaViz (Xerox Research/W3C)
- RDFAuthor (Univ. of Bristol)
- Longwell (MIT)
- Protege 2000 (Stanford Univ.)
- SWOOP (Univ. of Maryland)
- ...

Further info on OWL tools at:
http://www.w3.org/2001/sw/WebOnt/impls

Programming environments

We have already seen some
But: Jena 2 does OWL reasoning already!
SWI-Prolog is an RDF/OWL Framework in Prolog

Some Tools (Cont.)

Validators

- For RDF:
  - http://www.w3.org/RDF/Validator/
- For OWL:
  - http://owl.bbn.com/validator/
  - http://phoebus.cs.man.ac.uk:9999/OWL/Validator
  - http://www.mindswap.org/2003/pellet/demo.shtml

Ontology converter (to OWL)

at http://www.mindswap.org/2002/owl.html

Schema registries

e.g., EU Cores project (and its possible followers)

PART VII: Some Application Examples

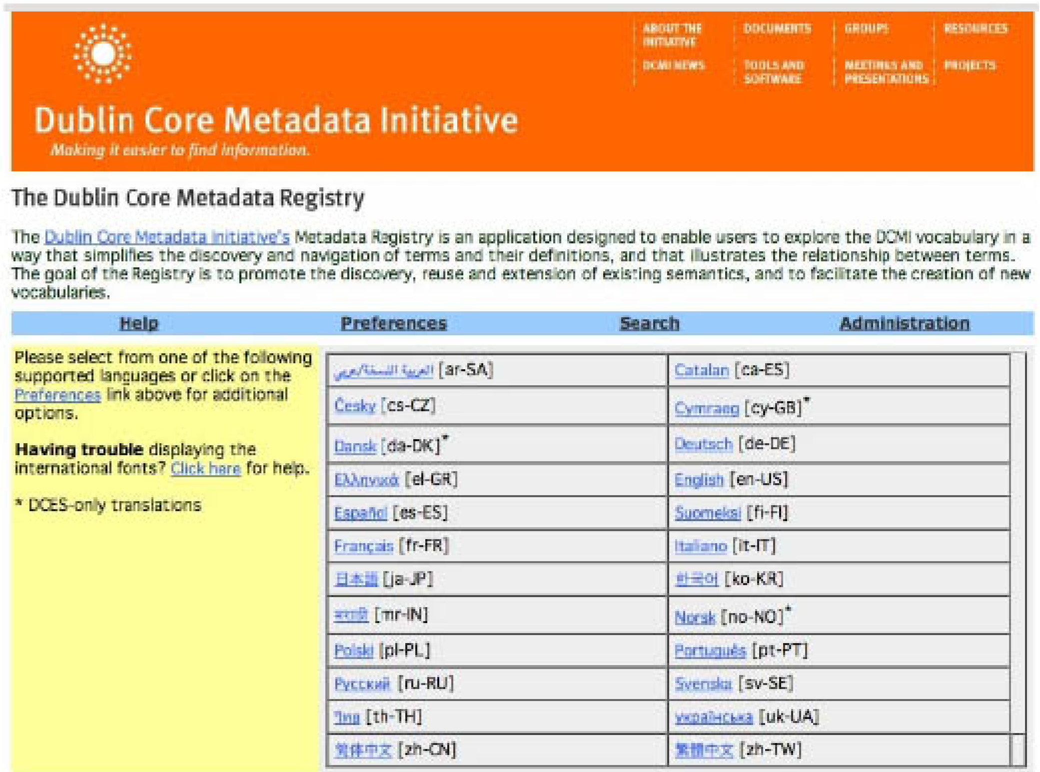
SW Applications

- Large number of applications emerge
  - some applications use RDF only
  - others use ontologies, too
    - huge number of ontologies exist, using proprietary formats
    - converting them to RDF/OWL will be a major task (but there are converters)
    - but it will be worth it!
- SWAD-Europe survey:
  - URI: http://www.w3.org/2003/11/SWApplSurvey
  - lists more than 50 applications in 12 categories…
  - and is already more than a years old!

SW Application Examples

Dublin Core

- vocabularies for distributed Digital Libraries
- one of the first metadata vocabularies in RDF
- URI: http://www.dublincore.org
- extensions exist, eg, PRISM that includes digital right tracking



SW Application Examples (cont)

Data integration

- achieve semantic integration of corporate resources or different databases
- RDF/RDFS/OWL based vocabularies as an "interlingua" among system components
- Boeing example: http://www.cs.rutgers.edu/~shklar/www11/final_submissions/paper3.pdf
- similar approaches: Artiste project, MITRE Corp., MuseoSuomi, …
- there are companies specializing in the area

SW Application Examples (cont)

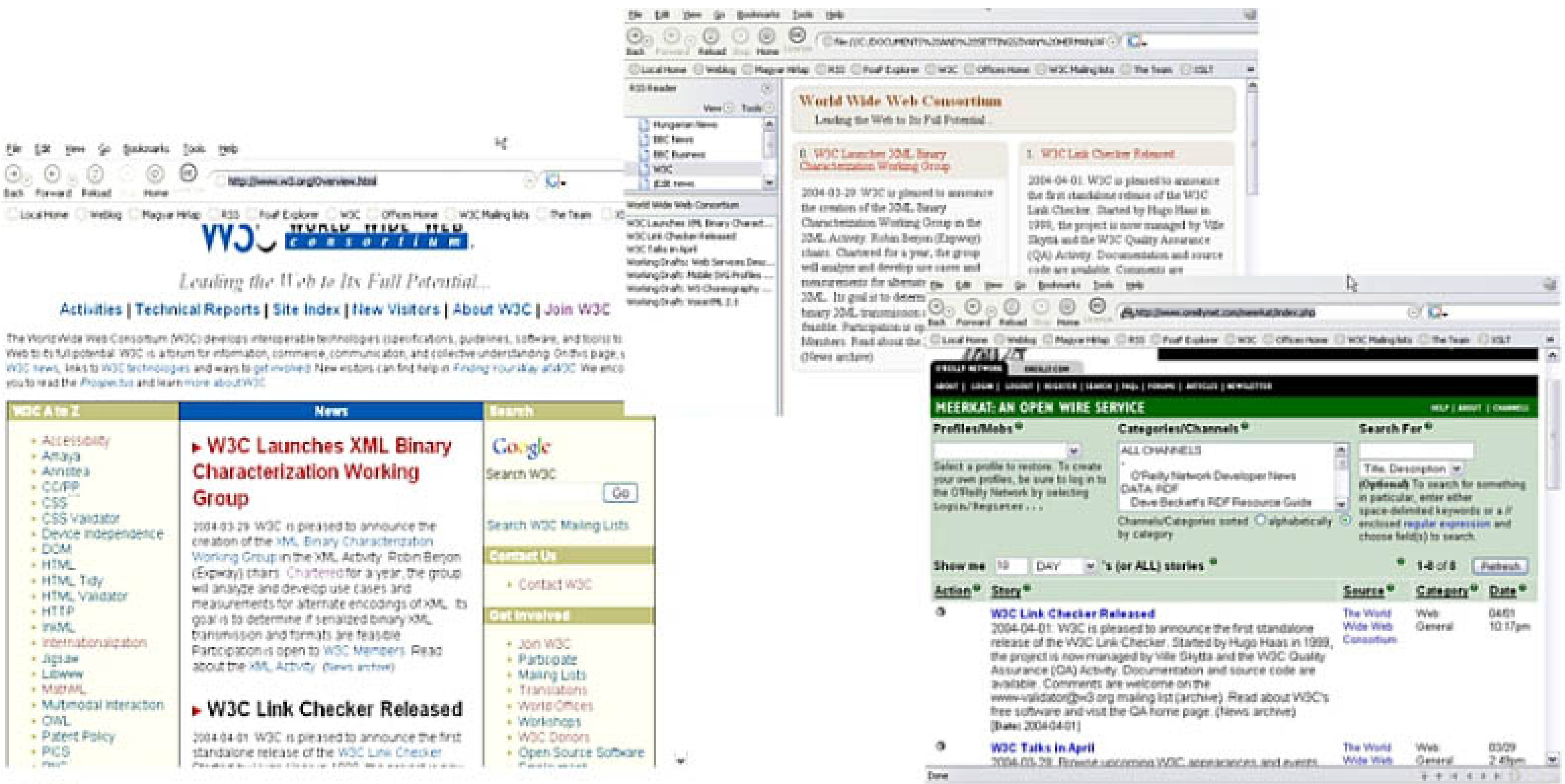
Sun's SwordFish

- Sun provides assisted support for its products, handbooks, etc
- Public queries go through an internal RDF engine for, eg:
  - Sun's White Papers collection (http://www.sun.com/servers/wp.html/)
  - Sun's System Handbooks collection (http://sunsolve.sun.com/handbook_pub/)

SW Application Examples (cont)

Web Content Syndication (RSS)

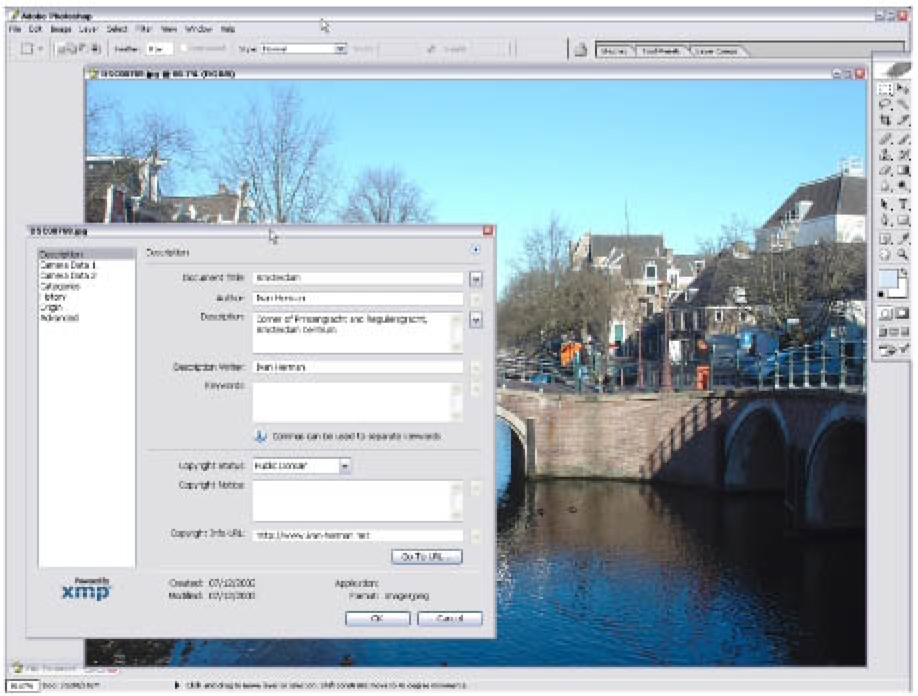
- can be used to specify the *important* content of a page
- there is a Yahoo discussion group and (non-W3C) working group
- URI: http://purl.org/rss/
- widely used in the weblog world!
- example: W3C home page syndicated



SW Application Examples (cont)

XMP

- Adobe's tool to add RDF-based metadata to *all* their file formats
  - eg, Photoshop in Creative Suite
  - millions of people use RDF without knowing it…
- the tool is available for all!
- URI: http://www.adobe.com/products/xmp/main.html
- See, eg, AI → SVG example



SW Application Examples (cont)

Web Services Descriptions

- mapping of WSDL1.2 to RDF
- Web Choreography development in terms of RDF/OWL
  - initiatives already exist, e.g., OWL-S or WSMO
- may be done by various W3C groups

Gene Ontology Consortium

- controlled vocabularies to describe aspects of gene products
- URI: http://www.geneontology.org

OntoWeb

- ontology-based information exchange for knowledge management and electronic commerce
- URI: http://ontoweb.aifb.uni-karlsruhe.de/

SW Application Examples (cont)

Mozilla

- internal data are stored in RDF (eg, bookmarks, conf. files)

Brandsoft

- entreprise Web Management
- all business models are stored in RDF
- easy to set up internal rules

Creative Commons

- an environment to express rights of digital content on the Web
  - legal constraints referred to in RDF, added to pages
- there are specialized browsers, browser plugins
- more than 1,000,000 users worldwide(!)
  - without knowing that they use RDF...

SW Application Examples (cont)

Baby CareLink

- centre of information for the treatment of premature babies
- provides an OWL service *as a Web Service*
  - combines disparate vocabularies like medical, insurance, etc
  - remember: ontology is hard!
  - users can add new entries to ontologies
  - complex questions can be asked through the service
- *perfect example for the synergy of Web Services and the Semantic Web!*

Some Closing Words on OWL

- Ontologies/OWL helps in finding *new* relationships
  - e.g.: Life Sciences:
    - most of the drug experiments are unsuccessful
    - but the information from *each* experiment may be valuable
    - by "binding" this information new insights can be gained

    (currently, life sciences are *very* excited by the prospects of the Semantic Web!)
- RDF/OWL offers flexibility
  - e.g., if the schema of a database has to be changed often
    - this can occur in a dynamic environment
  - redoing schemas may be costly and complicated...
  - whereas adding a new RDFS or OWL rule is a breeze!

OWL: Does it Scale?

- Q: with *huge* ontologies on the Web, does this scale?
- A: yes and no…
  - obviously, reasoning over *huge* ontologies may be a problem
    - and combination of ontologies may lead to this
  - but "a little semantics can take you far" (Jim Hendler)

    i.e., small OWL ontologies may be very powerful by themselves!
  - also: applications may use "islands" of ontologies, and loosely bind them
    - remember **`owl:sameAs`**?
  - however, further work is still needed here
  - note: *there are already applications with large ontologies!*

Further Information

These slides are at:

http://www.w3.org/2004/Talks/0608-StAugustin-IH/

Semantic Web homepage

http://www.w3.org/2001/sw/

More information about W3C in Germany and Austria:

http://www.w3c.de or http://www.w3c.at

Mail me:

ivan@w3.org