

3. Towards a Semantic Web	26. RDF in Programming	51. Classes, Resources	76. Ontologies (continued)	101. OWL Lite	126. SW Application
4. Towards a Semantic Web	27. Python Example	52. Classes, Resources	77. W3C's Ontology	102. "Description Logic"	127. SW Application
5. However...	28. Use of RDF in Ontology	53. Schema Example	78. Classes in OWL	103. Ontology Example	128. SW Application
6. The Semantic Web	29. Merging	54. Schema Example	79. Need for Enumerations		129. SW Application
7. What Is Needed (The Semantic Web)	30. Merge Shown as a Graph	55. Inferred Properties	80. (OWL) Classes and Properties	105. Semantic Web Applications	130. SW Application
8. The Semantic Web	31. Merge in Practice	56. Properties (Prediction)	81. Same in RDF/XML	106. Semantic Web Applications	131. SW Application
9. This Course Will	32. Adding New Statements	57. Properties (continued)	82. Union of Classes	107. "Best Practices"	132. SW Application
	33. Blank Nodes	58. Property Specificity	83. Same in RDF/XML	108. RDF Data Access	133. Further Information
	34. Blank Nodes: Tutorial	59. Property Specificity	84. Property Restrictions	109. Data Access Examples	
	35. Blank Nodes: Let's Try	60. Literals	85. Property Restrictions	110. Rules	
11. Problem Example	36. Blank Nodes: Solution	61. Literals in RDF/XML	86. Cardinality Restrictions	111. RDF API-s	
12. Statements	37. Typed Nodes	62. Literals in RDF/XML	87. Same in RDF/XML	112. Trust	
13. Resource Description	38. Typed Nodes (continued)		88. Property Characteristics	113. A Number of Related Topics	
14. RDF is a Graph	39. Sequences	64. Small Practical Issues	89. Characterization		
15. A Simple RDF Example	40. Sequences (continued)	65. Binding RDF to a Language	90. Same in RDF/XML	115. Available Specifications	
16. URI-s Play a Fundamental Role	41. Sequences (continued)	66. RDF/XML with XML	91. OWL: Additional Features	116. Available Specifications	
17. RDF/XML Principles	42. Sequences (continued)	67. RDF Can Also Be Represented as XML	92. Term Equivalences	117. Available Specifications	
18. RDF/XML Principles	43. An Aside: Typed Literals	68. RDF/XML has its Own Syntax	93. Example: Connecting to a Database	118. Some Books	
19. RDF/XML Principles	44. Other Containers	69. Programming Practicalities	94. Another Use of Examples	119. Further Information	
20. Several Properties	45. Collections (Lists)	70. Programming Practicalities	95. Versioning, Annotations	120. Further Information	
21. Several properties	46. The Same in RDF/XML	71. Jena	96. OWL and Logic	121. Public Fora at W3C	
22. Adding a New property	47. Our Graphical Shorthand	72. Jena (continued)	97. Examples for Logic	122. Some Tools	
23. Adding a New property	48. Some Words of Motivation	73. Lots of Other tools	98. However: Ontology Languages	123. Some Tools (Continued)	
24. A Very Useful Simplification			99. OWL Full		
25. Simplification in OWL	50. Back to Typing: Final	75. Ontologies	100. OWL Descriptions	125. SW Applications	

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

- The current Web represents information using
 - natural language (English, Hungarian, Finnish,...)
 - 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!)

- 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!

- However: machines are ignorant!
 - partial information is unusable
 - difficult to make sense from, e.g., an image
 - drawing analogies automatically is difficult
 - difficult to combine information
 - is **<foo:creator>** same as **<bar:author>**?
 - how to combine different XML hierarchies?
 - ...
- But you know that better than I do...

- 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)

- “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 it 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

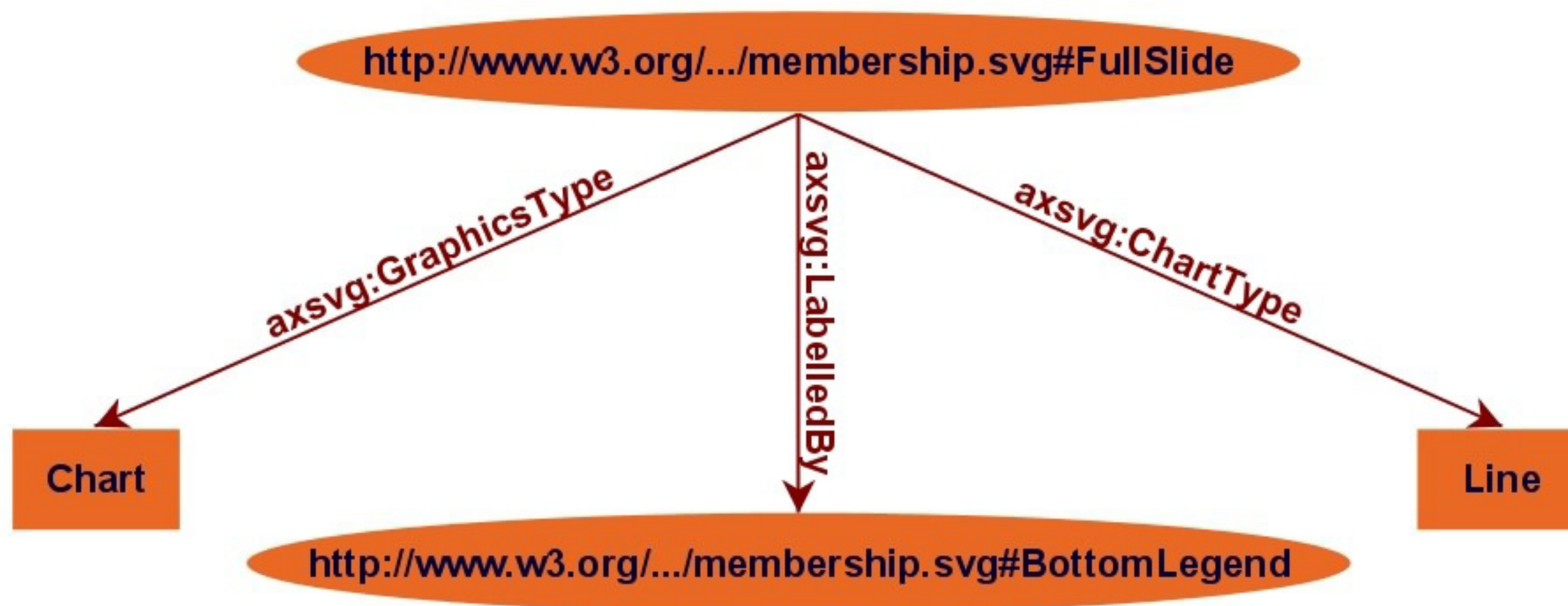
- Convey the meaning of a figure through text (important for accessibility)
 - add *metadata* to the image describing the content
 - let a tool produce some *simple output* using the metadata
 - use a standard metadata formalism



- 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, ...

- 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

- 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”)

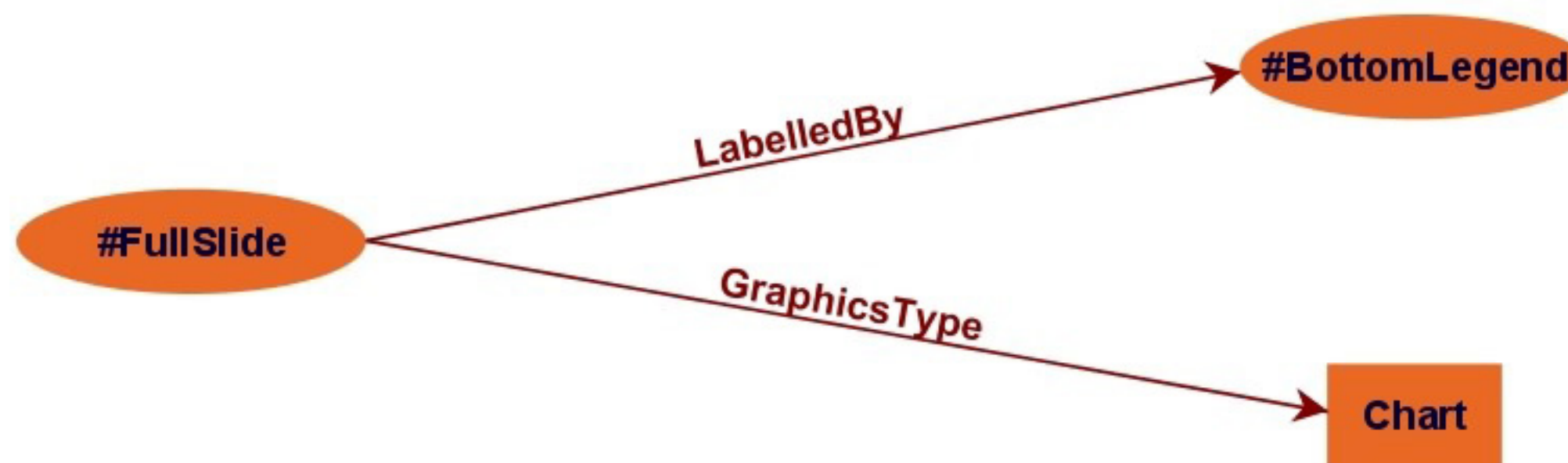


```
<rdf:Description
  rdf:about="http://.../membership.svg#FullSlide">
  <axsvg:GraphicsType>Chart</axsvg:GraphicsType>
  <axsvg:LabelledBy
    rdf:resource="http://.../membership.svg#BottomLegend"/>
  <axsvg:ChartType>Line</axsvg:ChartType>
</rdf:Description>
```

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



- 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»
```




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

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

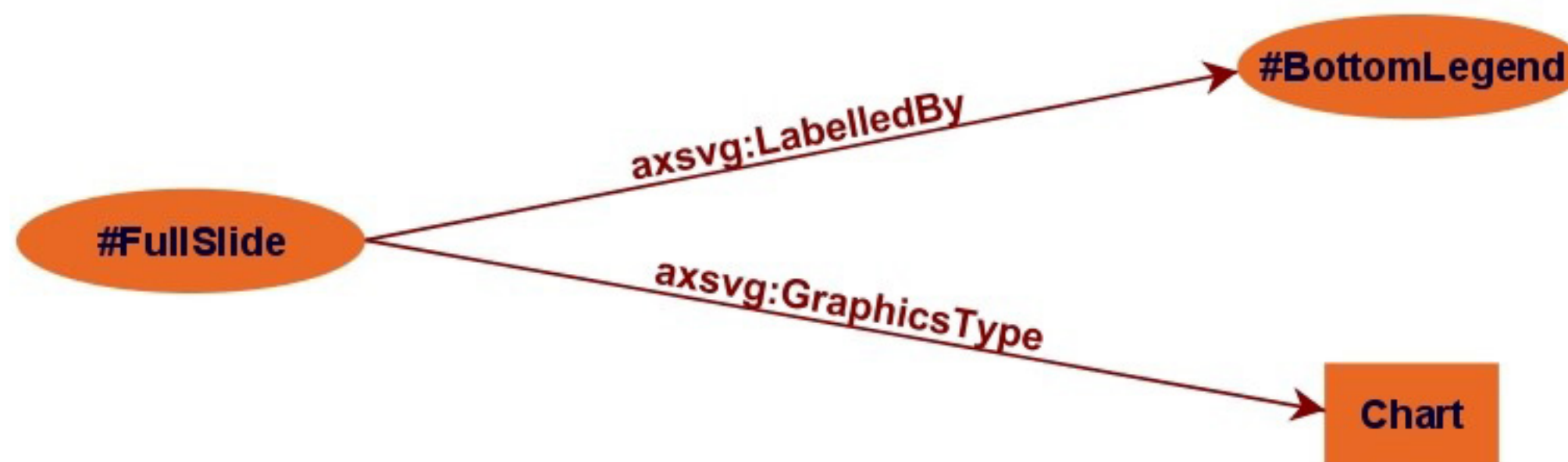
- Note the usage of *namespaces*!



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

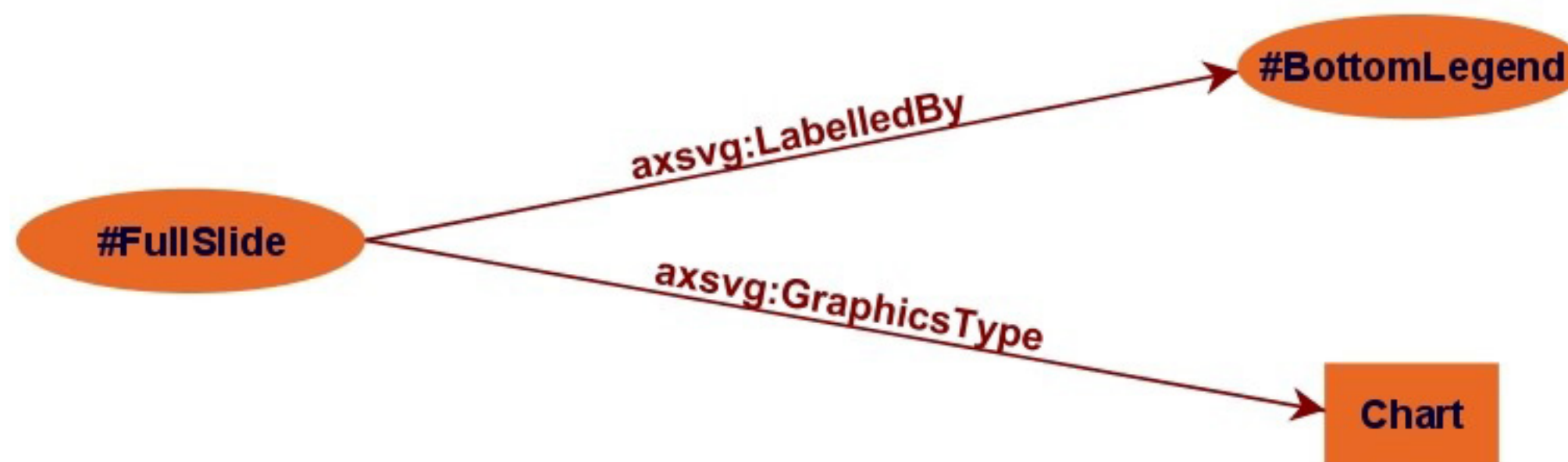
```
<rdf:RDF
  xmlns:axsvg="http://svg.example.org#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-1999#"
  <rdf:Description rdf:about="#FullSlide">
    <axsvg:LabelledBy>
      <rdf:Description rdf:about="#BottomLegend",
    </axsvg:LabelledBy>
  </rdf:Description>
</rdf:RDF>
```

(To save space, we will omit namespace declarations...)



- 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>
```

- 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



- (Note: the subject became also an object!)
- 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>
```




- 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" />
```



- Can be expressed by:

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


- 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 PHP, Java, etc. (see later)

In Python syntax:

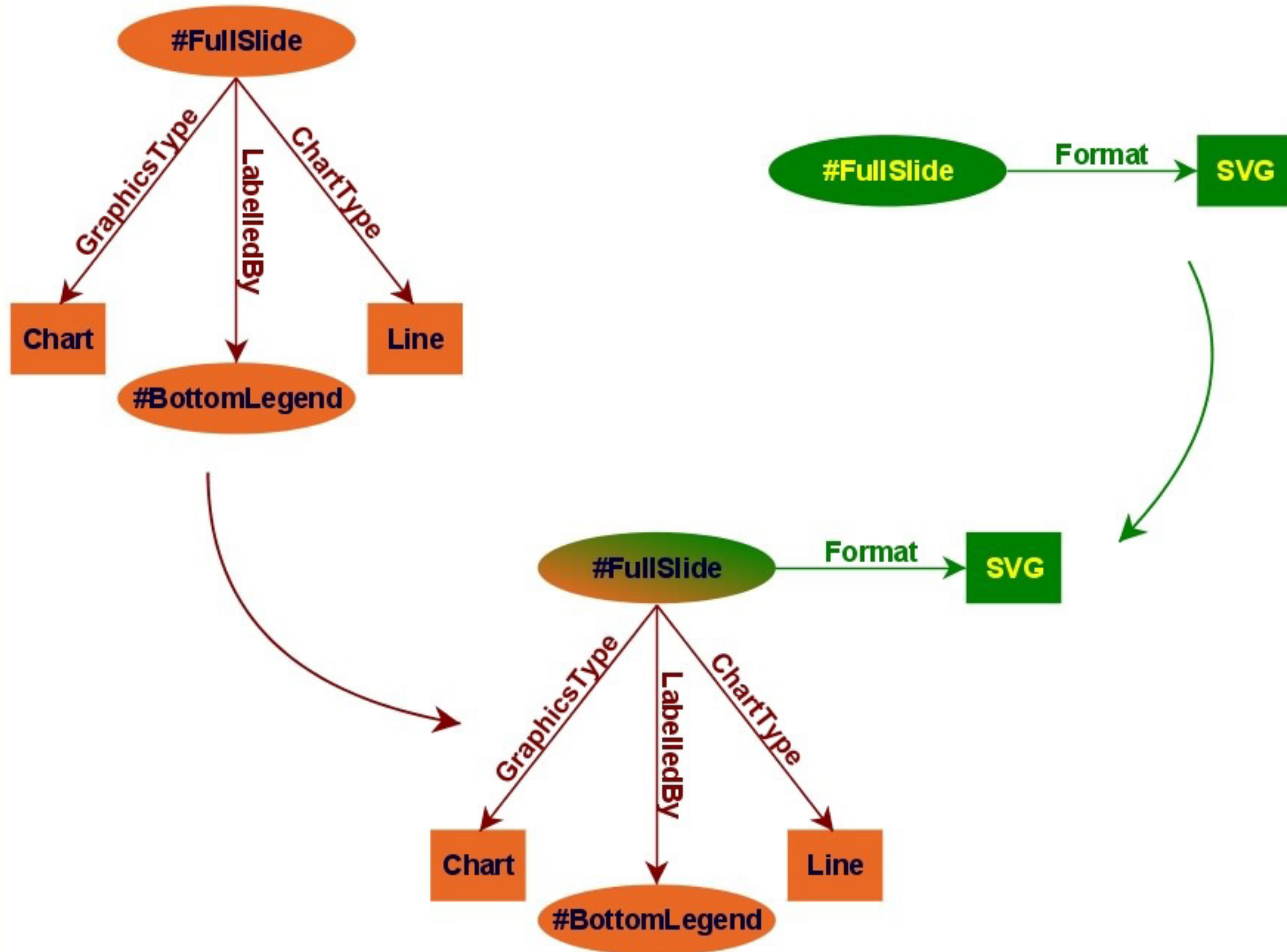
```
# import the libraries
from rdflib.TripleStore import TripleStore
from rdflib.URIRef import URIRef
# resource for a specific URI:
subject = URIRef("URI_of_Subject")
# create the triple store
triples = TripleStore()
# parse an RDF file and store it in the triple store
triples.load("membership.rdf")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


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

- 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



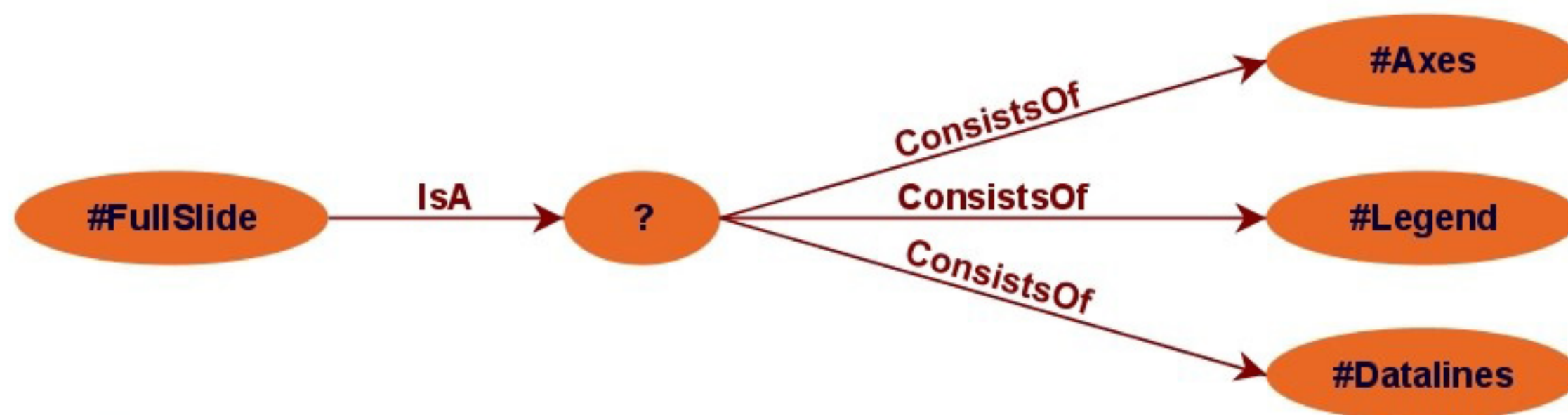
- **Development environments merge graphs automatically**
 - e.g., in Python, the Triple Store can “load” several files
 - the load merges the new statements automatically
- **Merging the RDF/XML files into one is also possible**
 - but not really necessary, the tools will merge them for you
 - keeping 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...

- 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»?



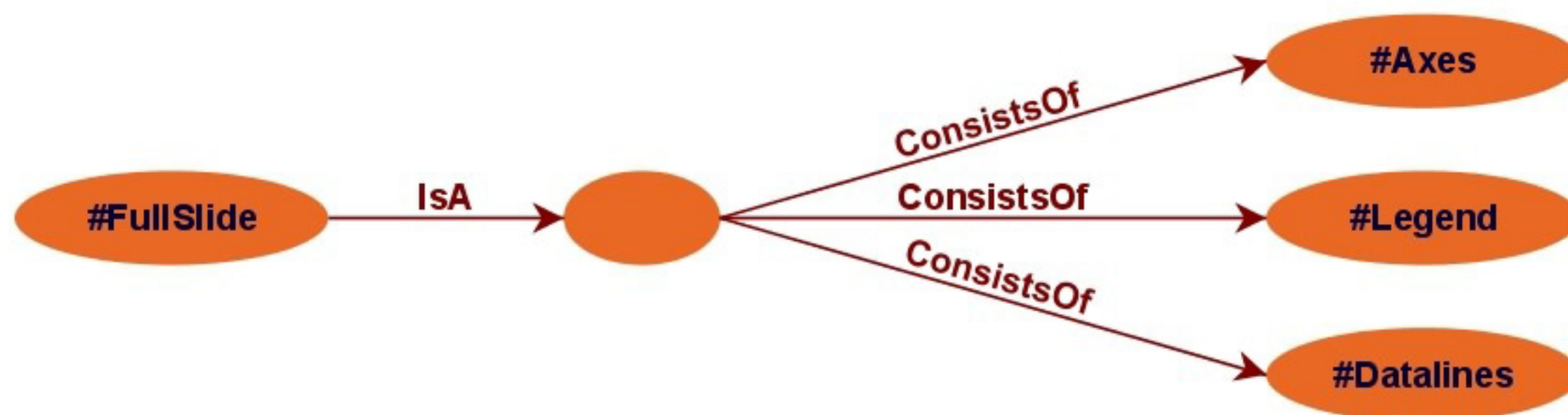
- In the XML serialization: give an id with **rdf:ID**

```
<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

- 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="#Legend" />  
      <axsvg:ConsistsOf rdf:resource="#Datalines" />  
    </rdf:Description/>  
  </axsvg:IsA>  
</rdf:Description/>
```



- Blank nodes require attention when merging
 - blanks nodes in different graphs are *different*
 - the implementation must 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/>
```

- 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#SVGEntiti
    ...
</rdf:Description/>
```

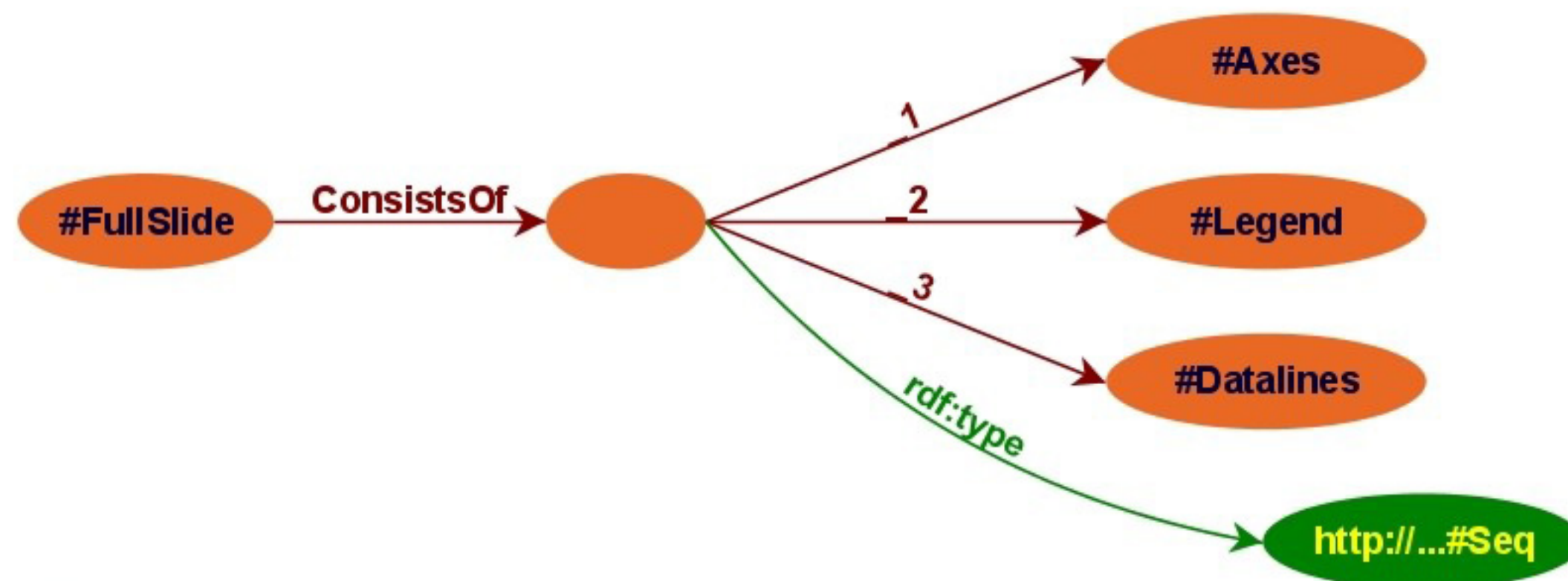

Typed Nodes (cont)

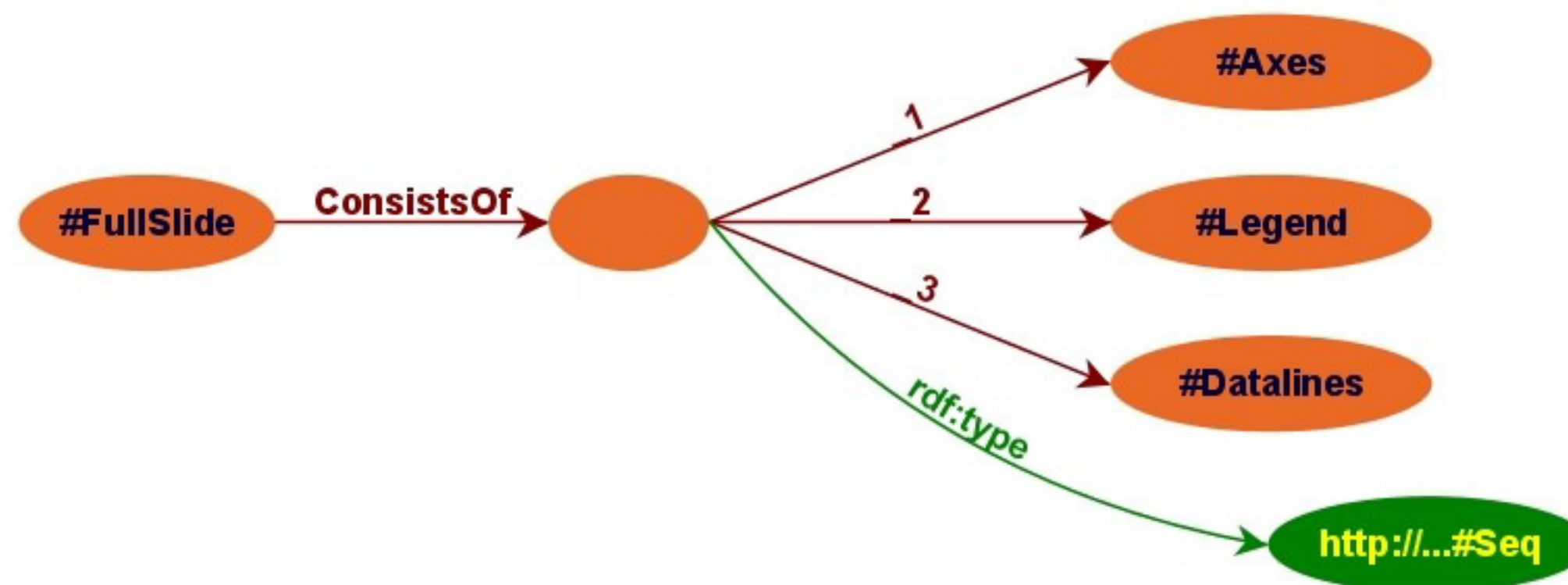


- A resource may belong to several classes
(**rdf:type** is just a property...)
- The type information may be very important for applications
 - e.g., it may be used for a categorization of possible nodes
- The **rdf** namespace contains predefined classes
 - see later...

- 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

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



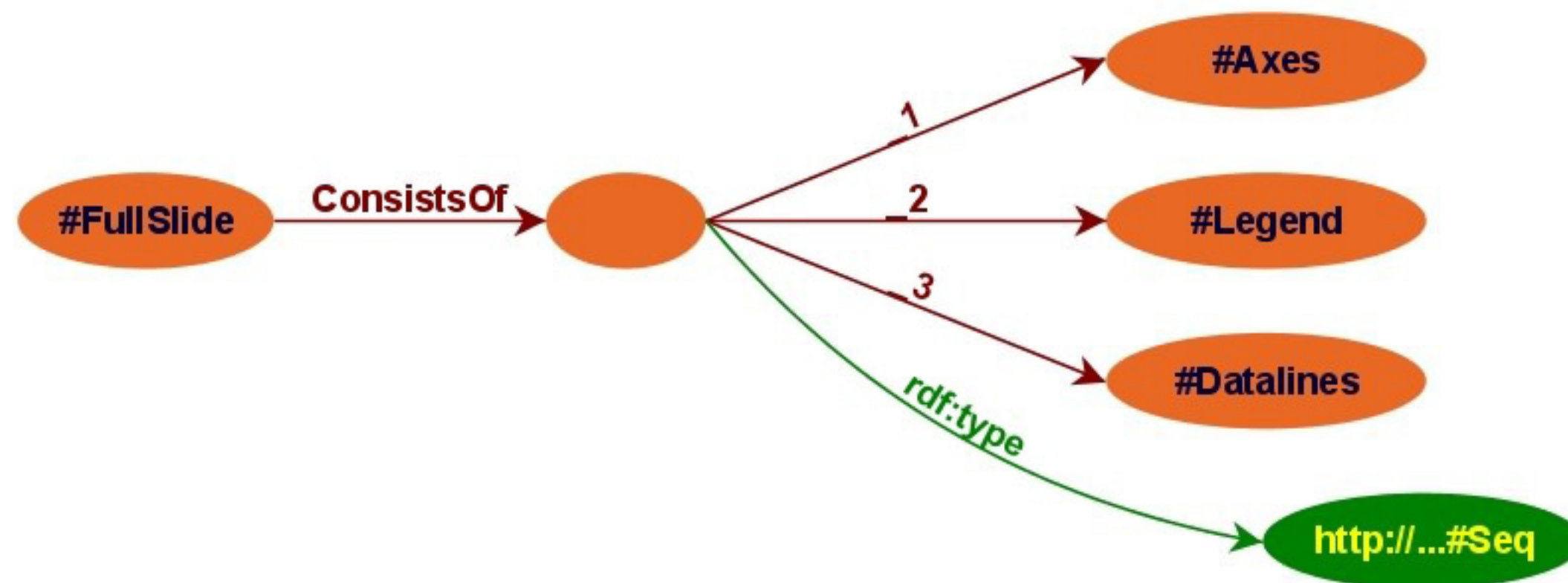


- In RDF/XML:

```

<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/>

```

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

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

- A frequent simplification rule: instead of:

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

use:

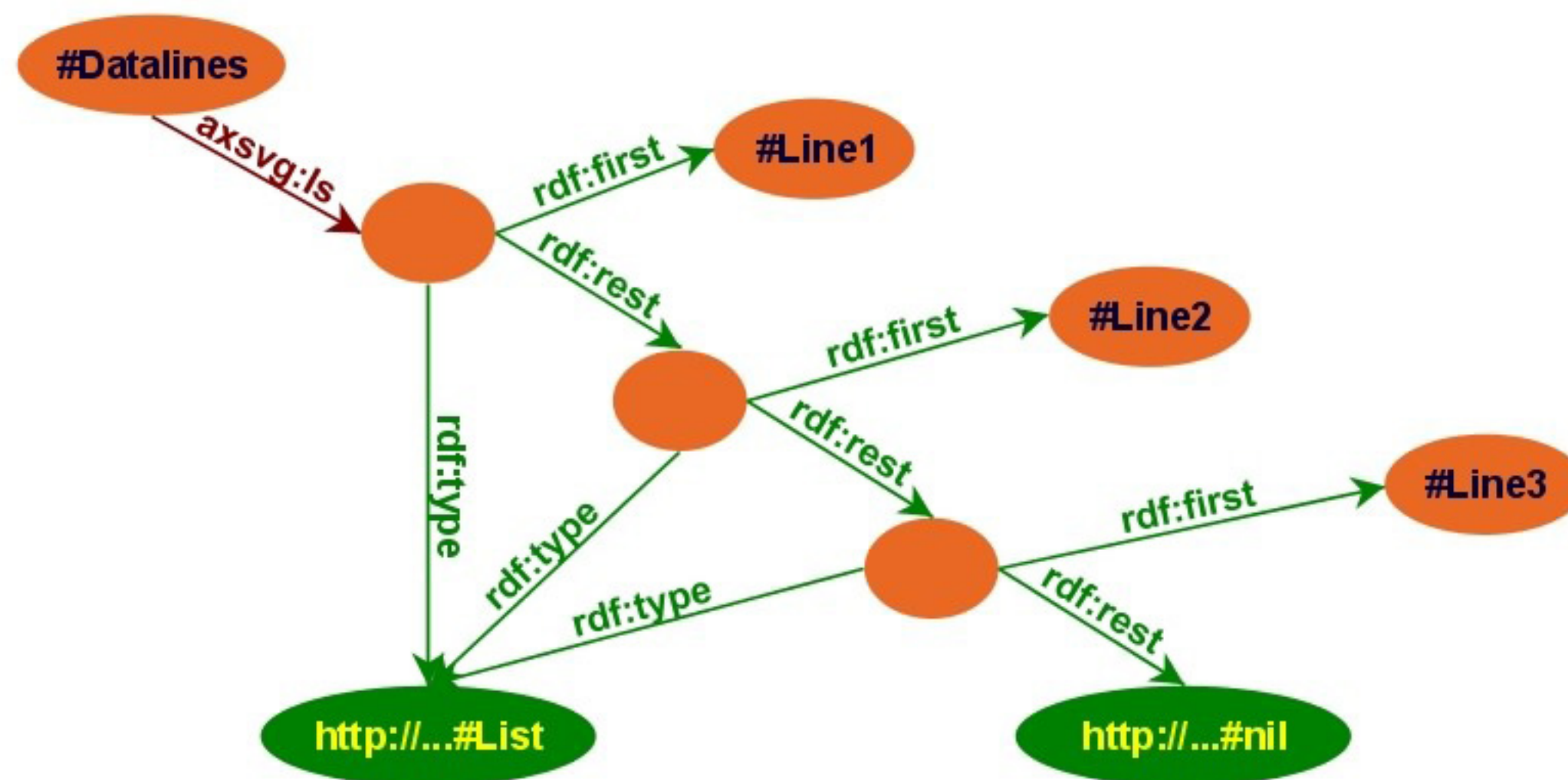
```
<yourNameSpace:ClassName rdf:about="http://...">  
  ...  
</yourNameSpace:ClassName>
```

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

- **rdf:Bag**
a general bag, no particular semantics attached
- **rdf:Alt**
attached semantics: *only one* of the constituents is “valid”

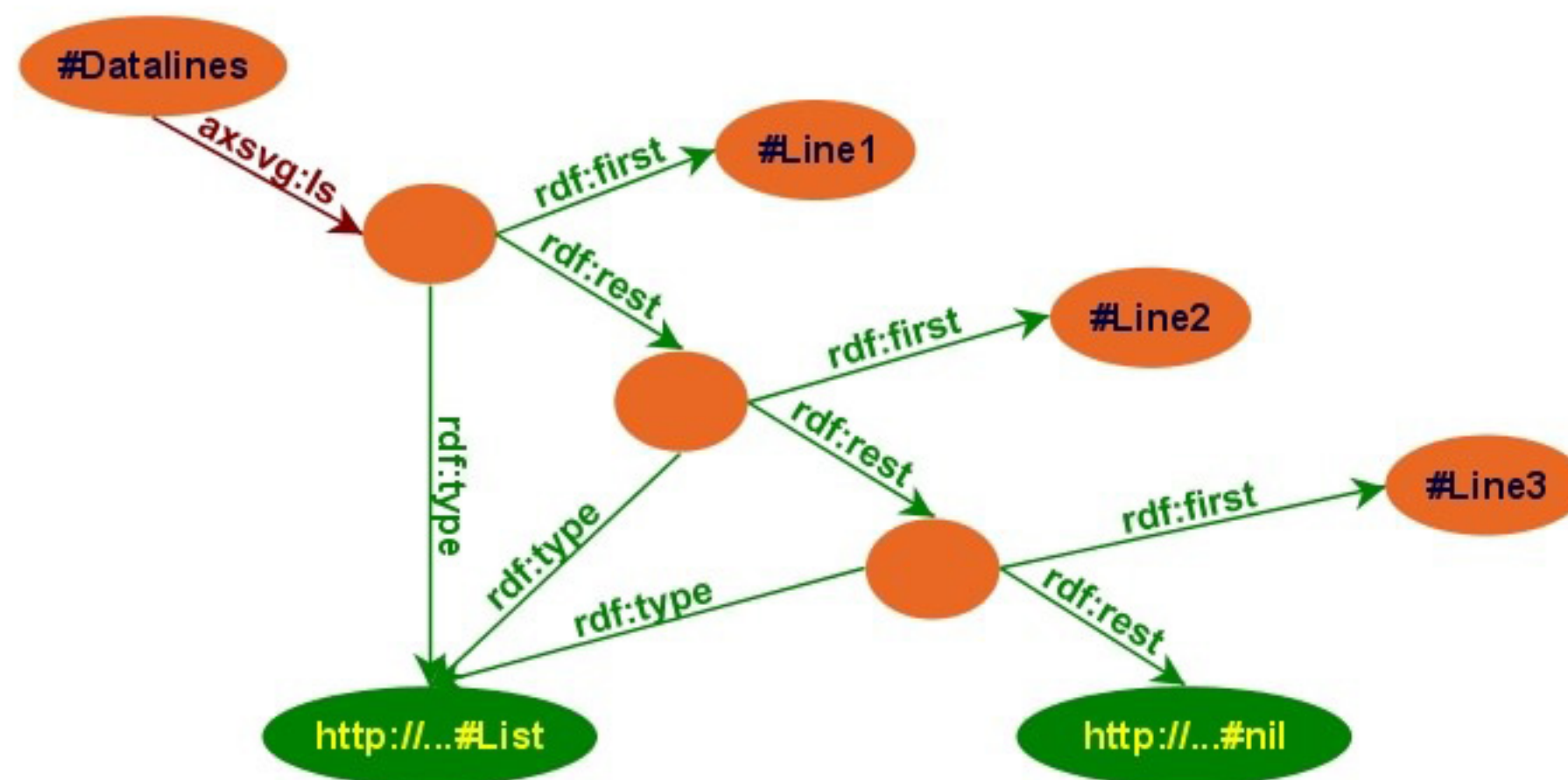
Collections (Lists)

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



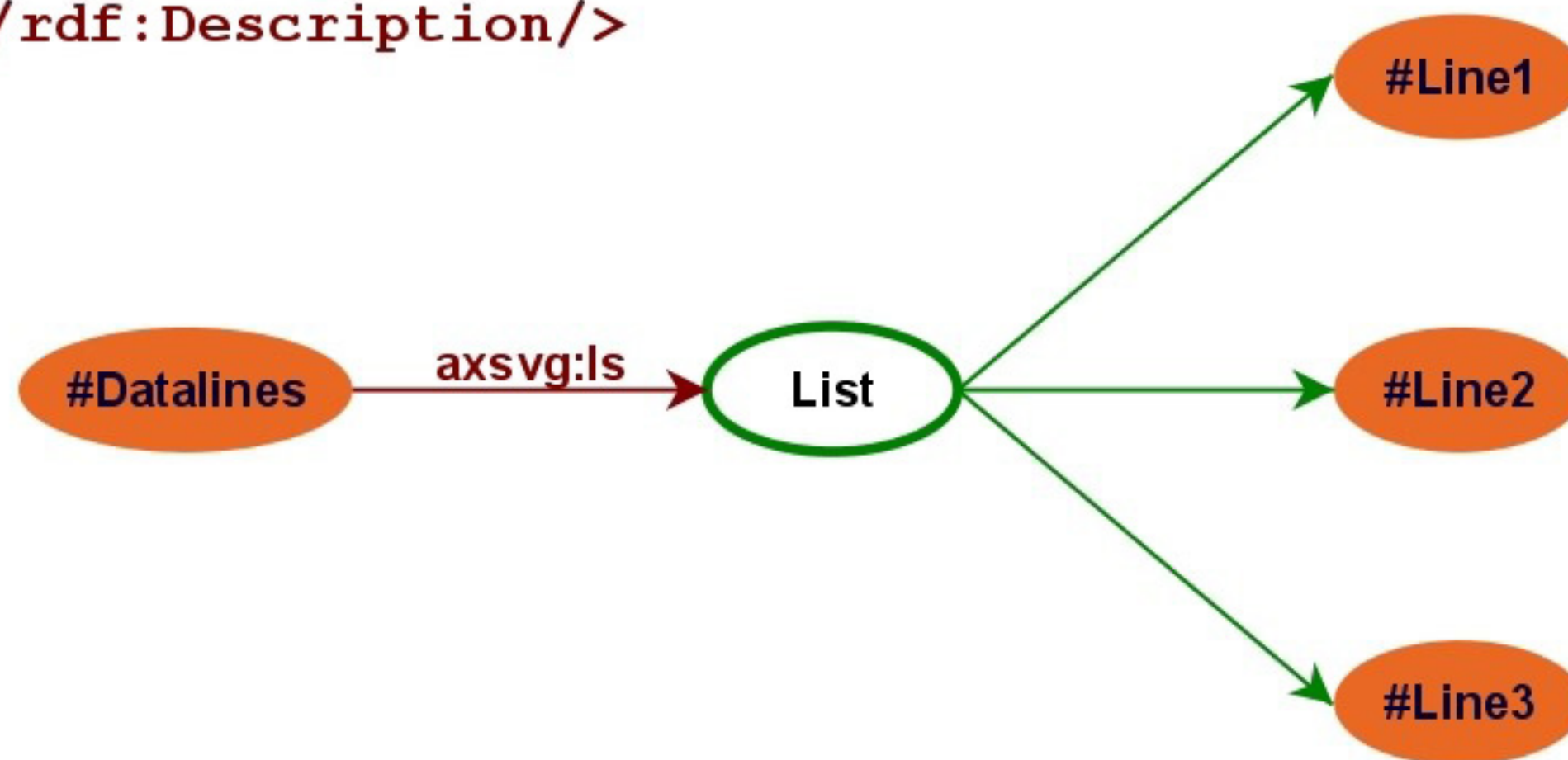
List in terms of XML:

```
<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/>
```



(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/>
```

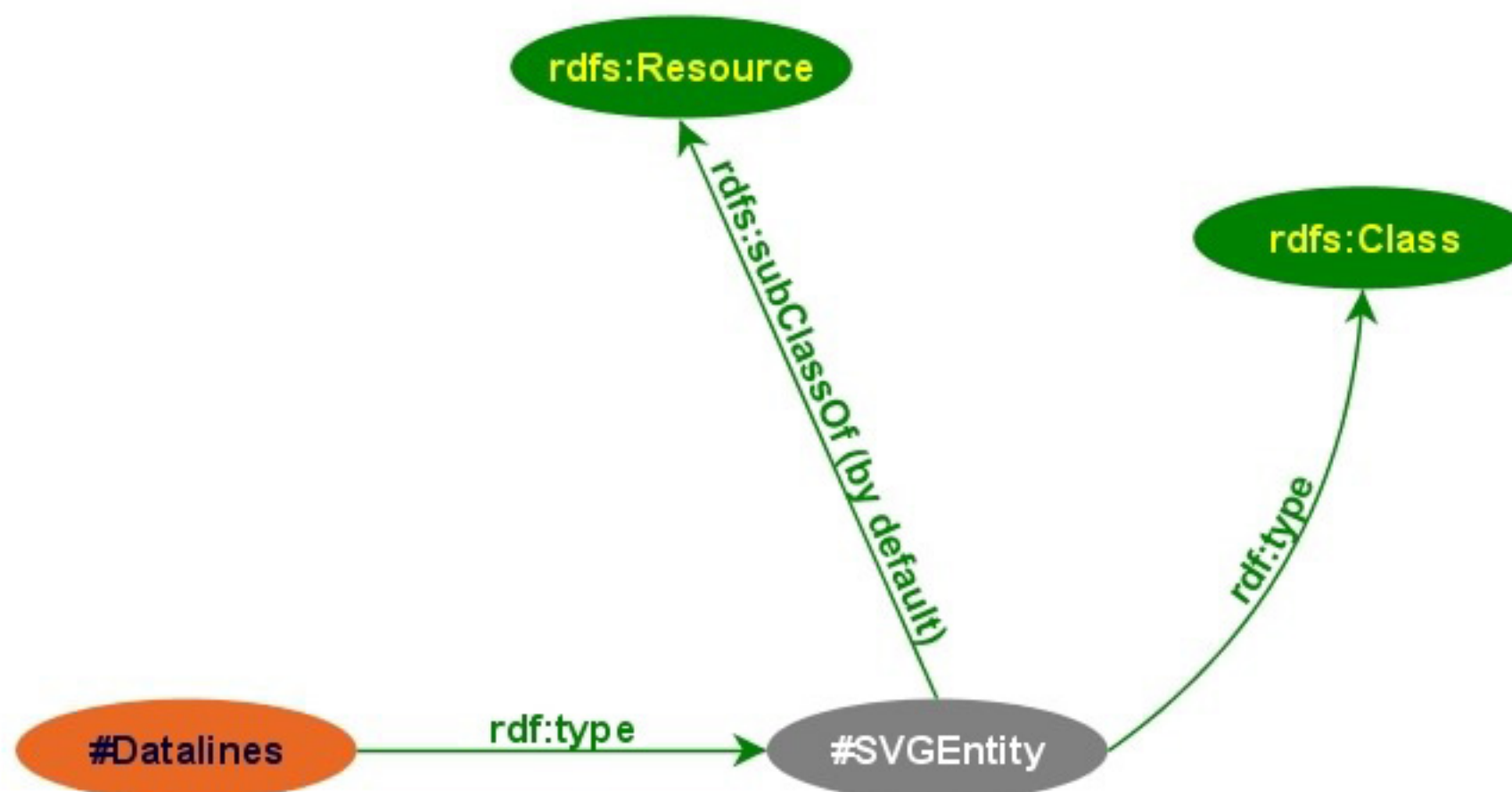


- RDF/XML introduces a number of simplifications
 - usage of **rdf:li** instead of **rdf:_1**, **rdf:_2**, ...
 - usage of **rdf:parseType** instead of **rdf:first**, **rdf:rest**, ...
 - etc.
- This can be deceptive when using, e.g., RDFLib:
 - the triples in the Triple Store are the “real” ones!
 - 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)

- 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”

- Think of well known in traditional ontologies:
 - use the term “mammal”
 - “every dolphin is a mammal”
 - “Flipper is a dolphin”
 - etc.
- 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) (e.g., “mammal”, “dolphin”)
- Relationships are defined among classes/resources:
 - “typing”: an individual belongs to a specific class (e.g., “Flipper is a dolphin”)
 - “subclassing”: instance of one is also the instance of the other (e.g., “every dolphin is a mammal”)



- 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!)

- 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#Class"/>  
</rdf:Description>
```

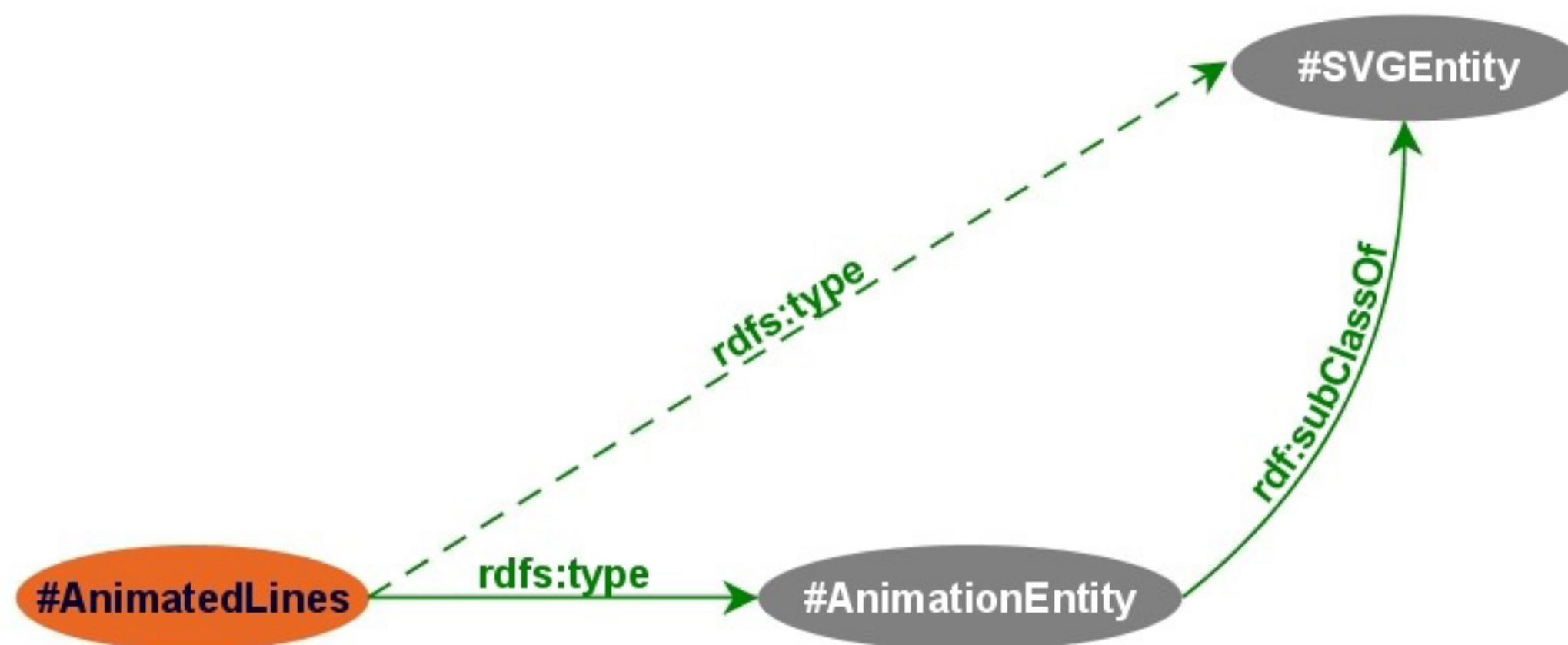
- In the rdf data on a specific graphics ("using the type"):

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


- 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>
```



(#AnimatedLines rdf:type #SVGEntity)

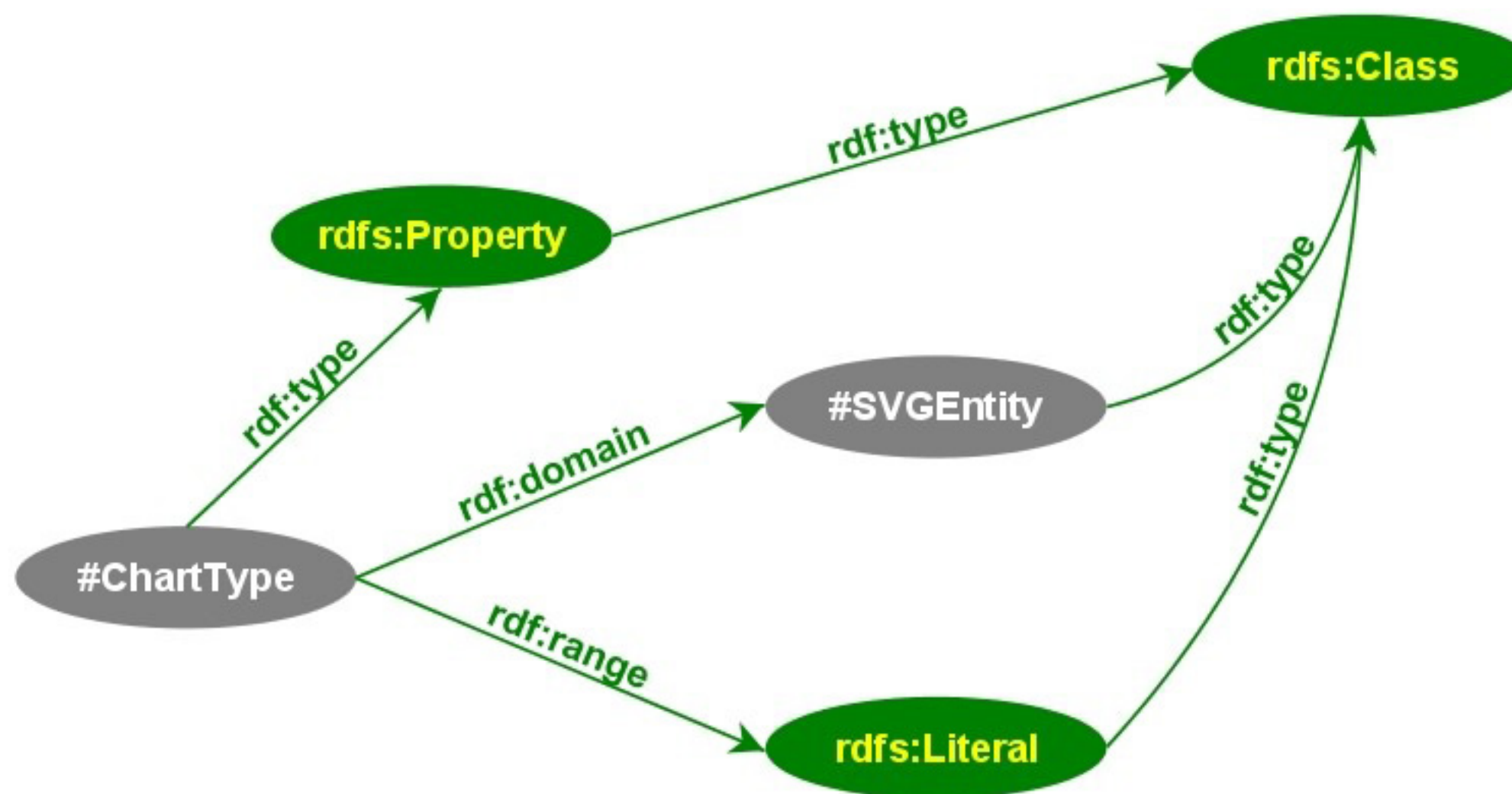
- is *not* in the original RDF data...
- ...but can be *inferred* from the RDFS rules
- Better RDF environments will return that triplet, too

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 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**

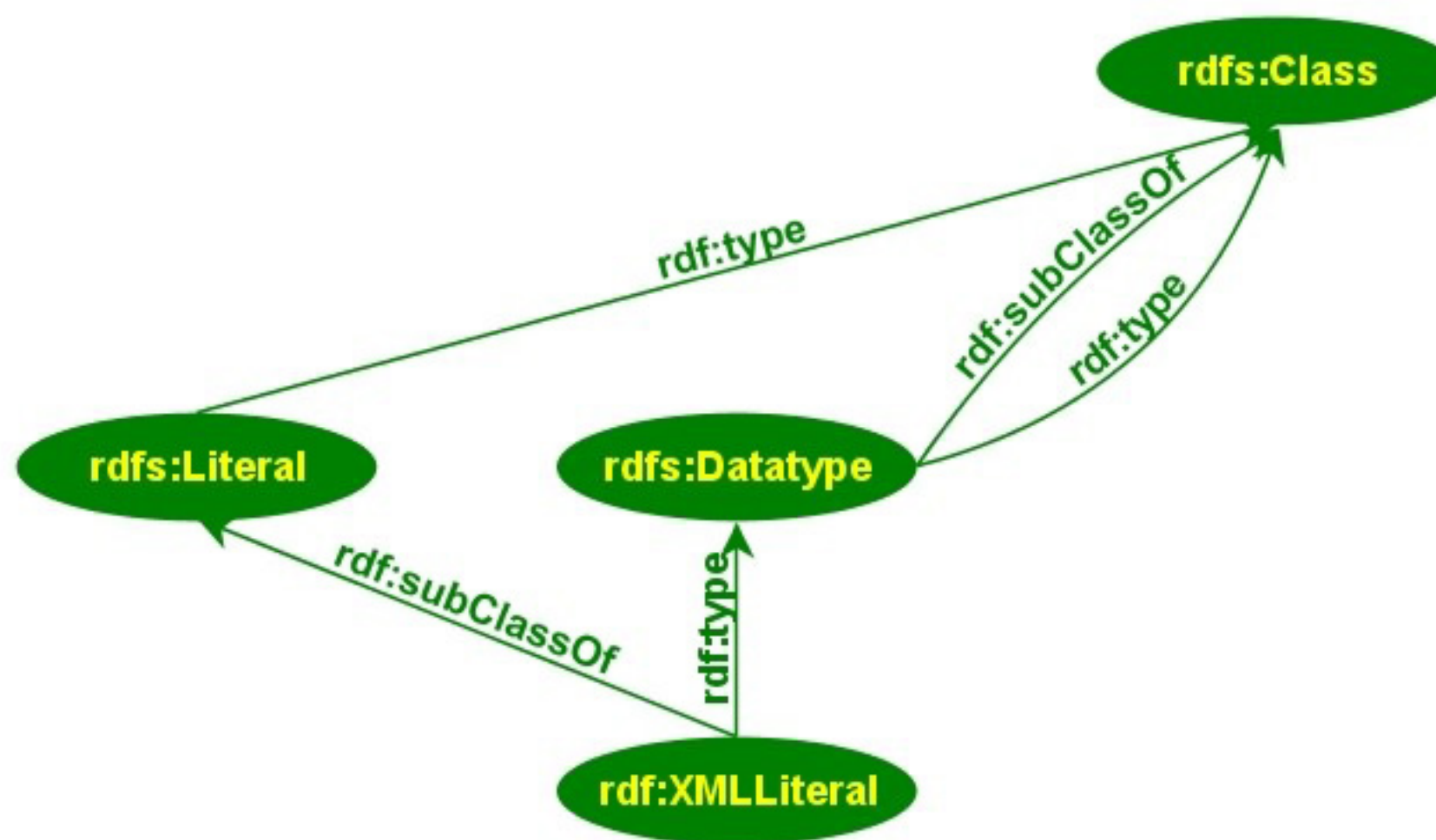


- Note that one cannot define *what* literals can be used
- This requires ontologies (see later)

Same example in XML/RDF:

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


- Literals may have a data type
 - floats, int, etc.
 - all types defined in XML Schemas
- (Natural) language can be specified
- Formally, data types are separate RDFS classes
- Full XML fragments may also be literals



- Typed literals:

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


- XML Literals:
 - 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

- RDF/XML files have a registered Mime type:
`application/rdf+xml`
- Recommended extension: `.rdf`

- 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>
```


- 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)

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

- 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
 - (e.g., 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

- 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)
```


- 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” properties, for example
- You can get RDFLib at: <http://rdflib.net>

- 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,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty();
    o = st.getObject();
    do_something(p,o);
}
```


- 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/>

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

PART V: Ontologies (OWL)

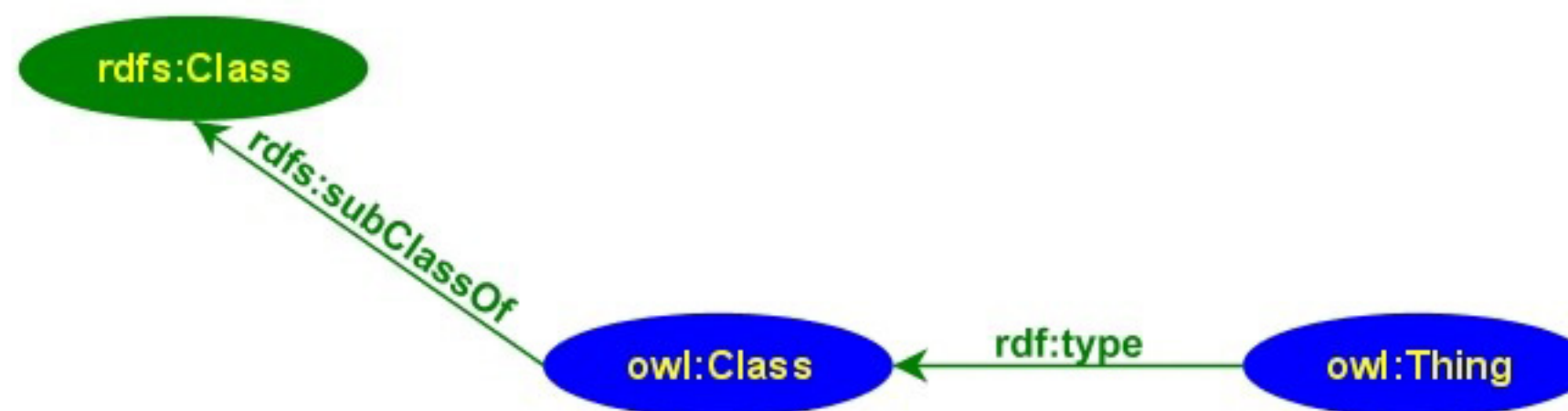
- RDFS is useful, but does not solve all the issues
- Complex applications may want more possibilities:
 - can a program *reason* about some terms? E.g.:
 - "if «A» is left of «B» and «B» is left of «C», is «A» left of «C»?"
 - obviously true for humans, not obvious for a program ...
 - ... programs 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.

- 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.
- Language should be a compromise between
 - rich semantics for meaningful applications
 - feasibility, implementability

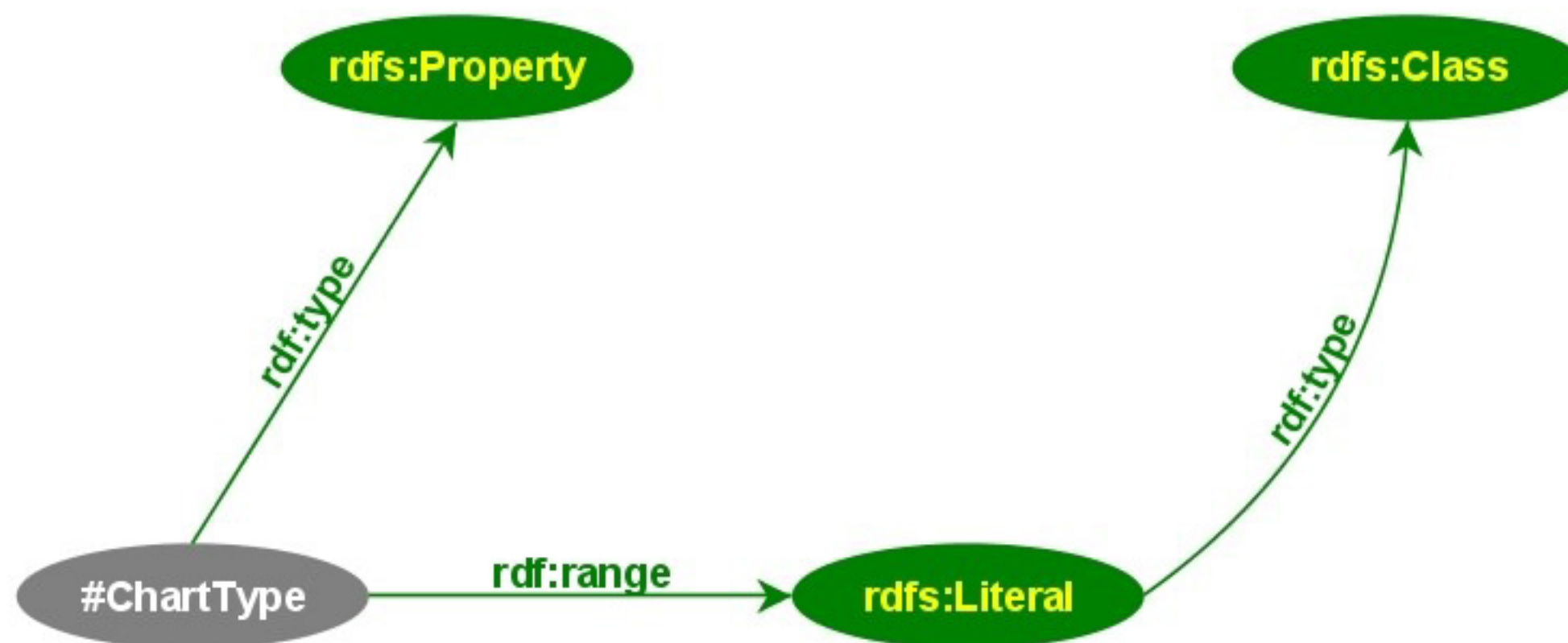
- A layer *on top* of RDFS with additional possibilities
- Outcome of various projects:
 1. a DARPA project: DAML
 2. a EU project: OIL
 3. an attempt to merge the two: DAML+OIL
 4. the latter was submitted to W3C
 5. lots of coordination with the core RDF work
 6. recommendation since early 2004



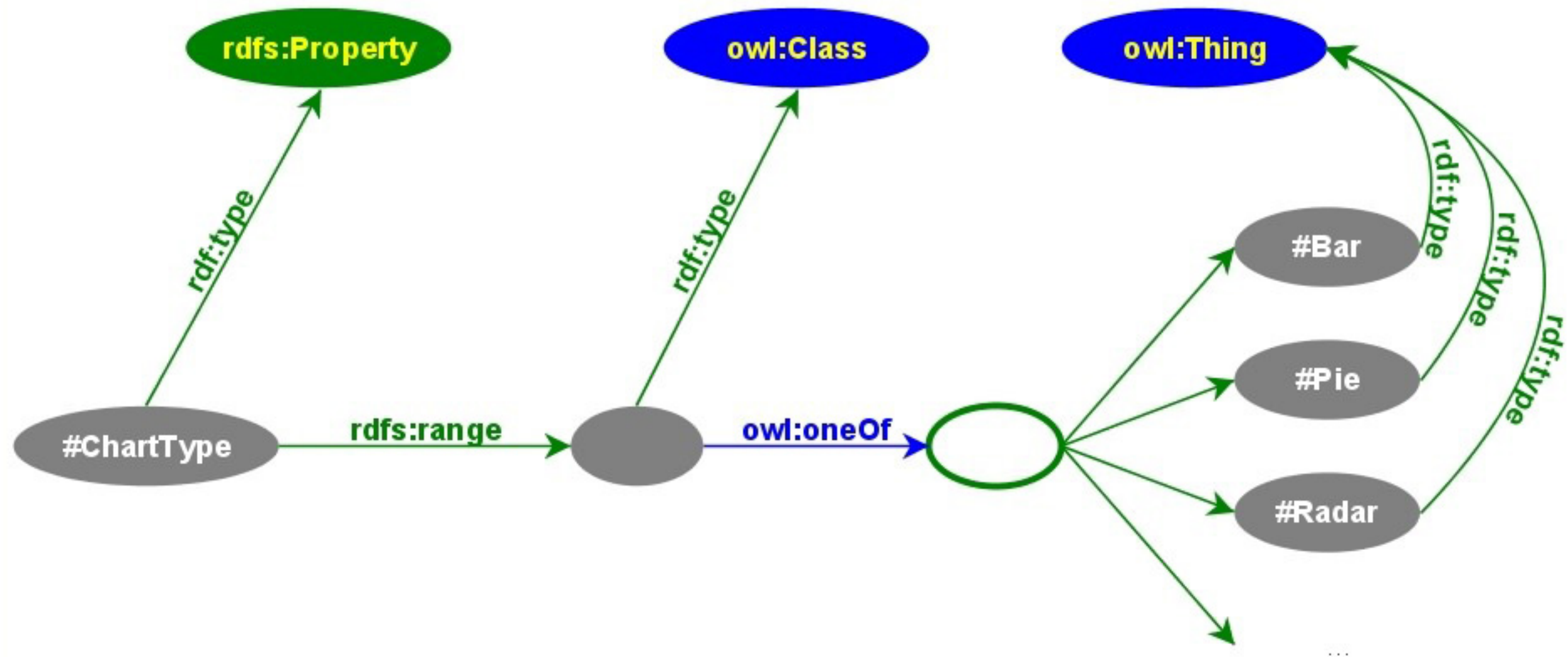
- In RDFS, you can subclass existing classes...
... but, otherwise, that is all you can do
- In OWL, you can *construct* classes from existing ones:
 - enumerate its content
 - through intersection, union, complement
 - through property restrictions
- To do so, OWL introduces its own **Class**...
...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



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



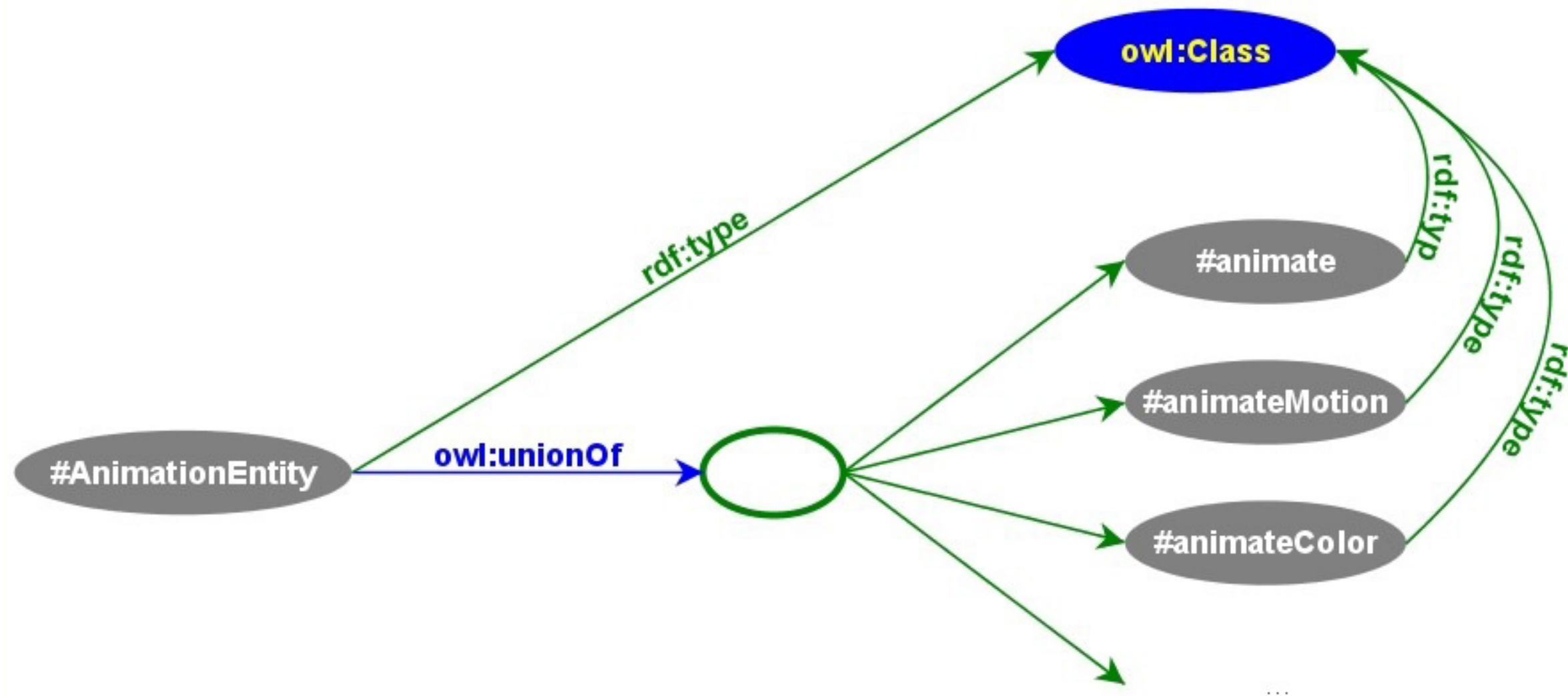
- The OWL solution, where possible content is explicitly listed:



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>
```


- Essentially, set-theoretical union:



Union in XML:

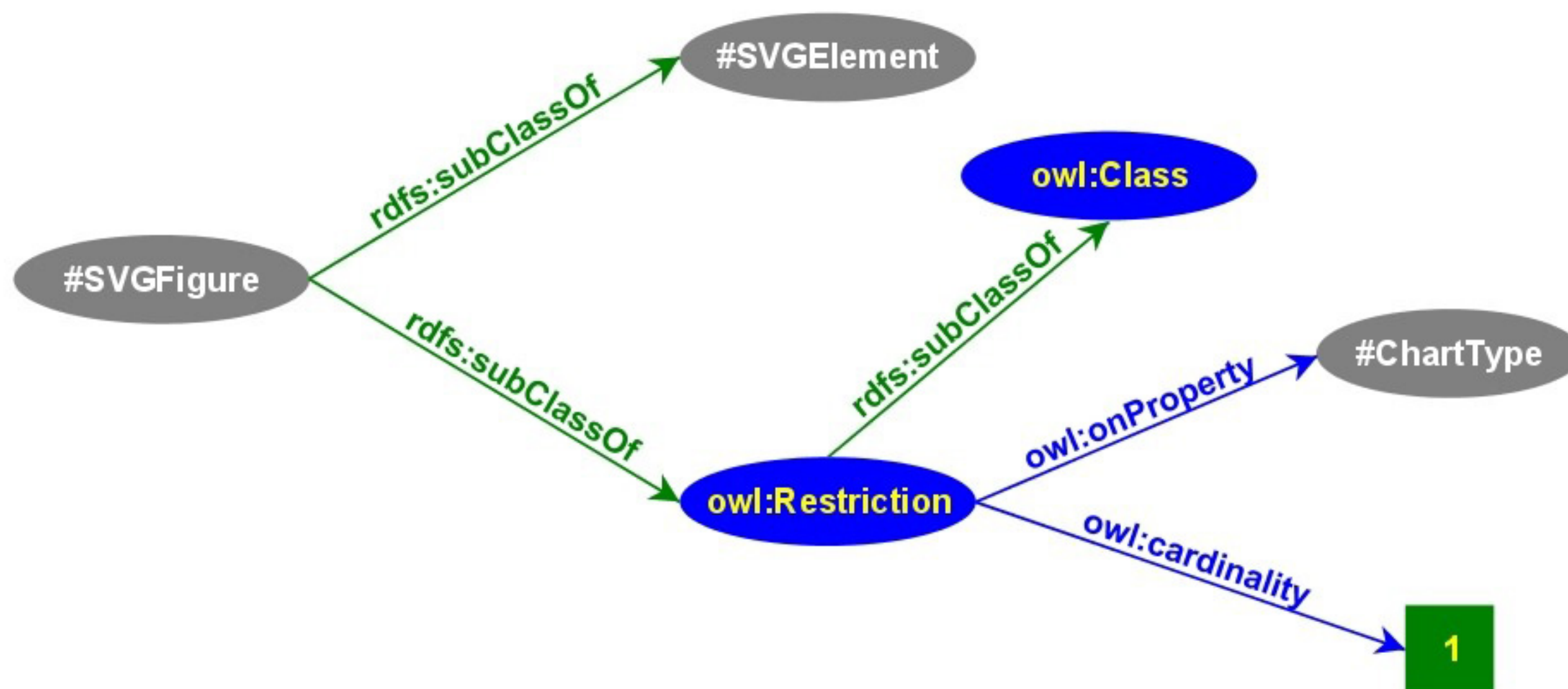
```
<owl:Class rdf:ID="AnimationEntity">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#animate" />
    <owl:Class rdf:about="#animateMotion" />
    <owl:Class rdf:about="#animateColor" />
    ...
  </owl:unionOf>
</owl:Class>
```

- Other possibilities: `complementOf`, `intersectionOf`

- (Sub)classes can be created by restricting the behavior of a property *on that class*
 - “a dolphin is a mammal living in water”
 - we restrict the value of “living in”
- 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

- 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

- “An SVG figure is an SVG element that have a *single* chart type”:

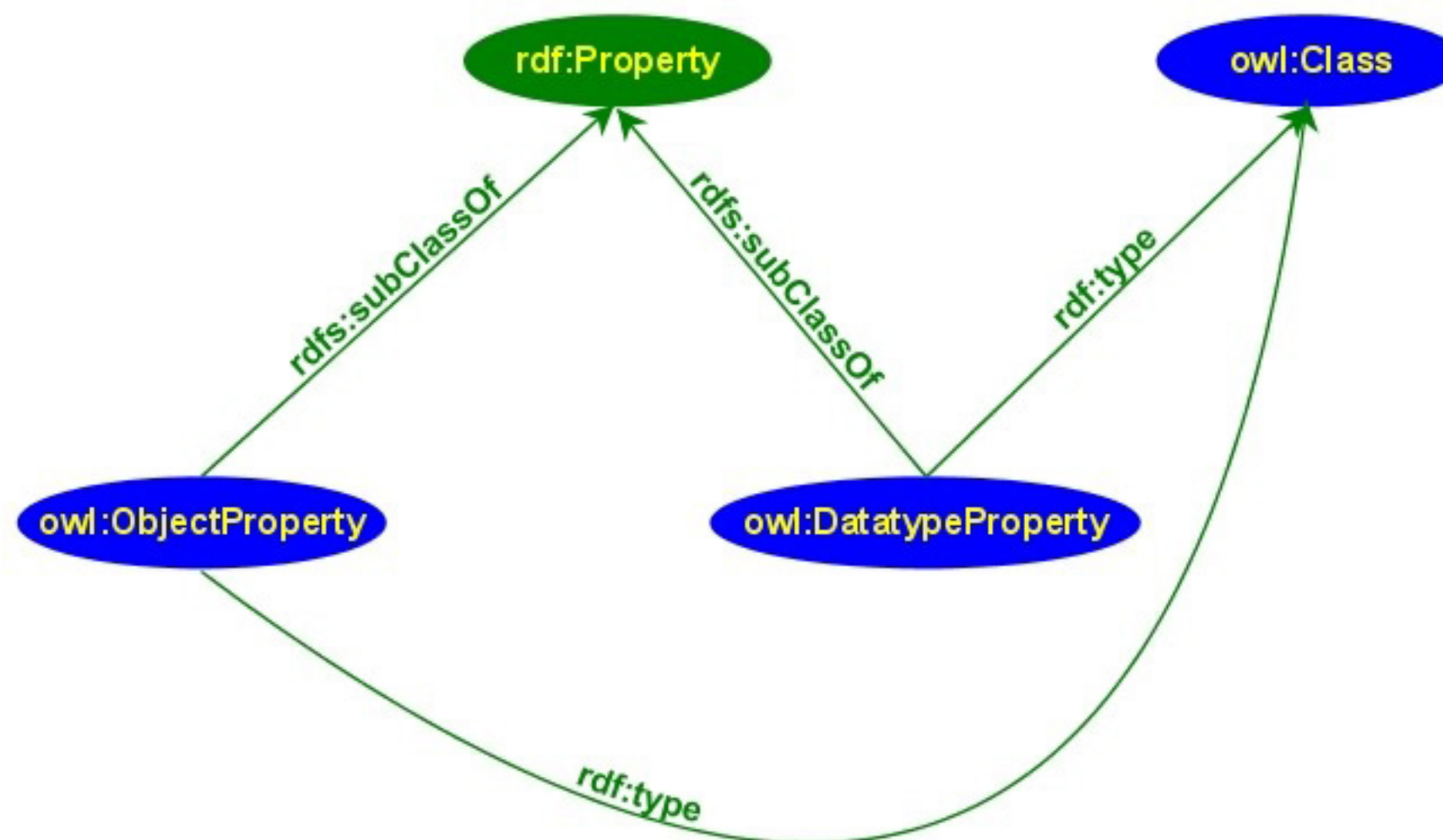


Cardinality constraint in XML:

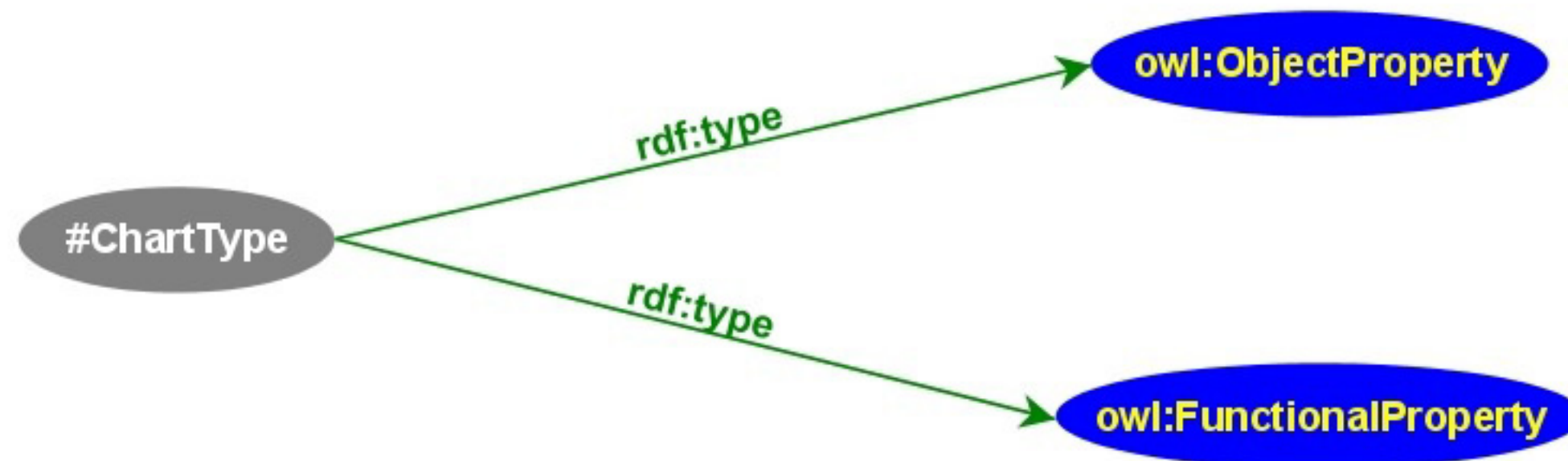
```
<owl:Class rdf:ID="SVGFigure">
  <rdfs:subClassOf rdf:about="#SVGElement"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:about="#ChartType"/>
      <owl:cardinality
        rdf:datatype="...#nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>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

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

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



- An alternative for the cardinality=1 setting:



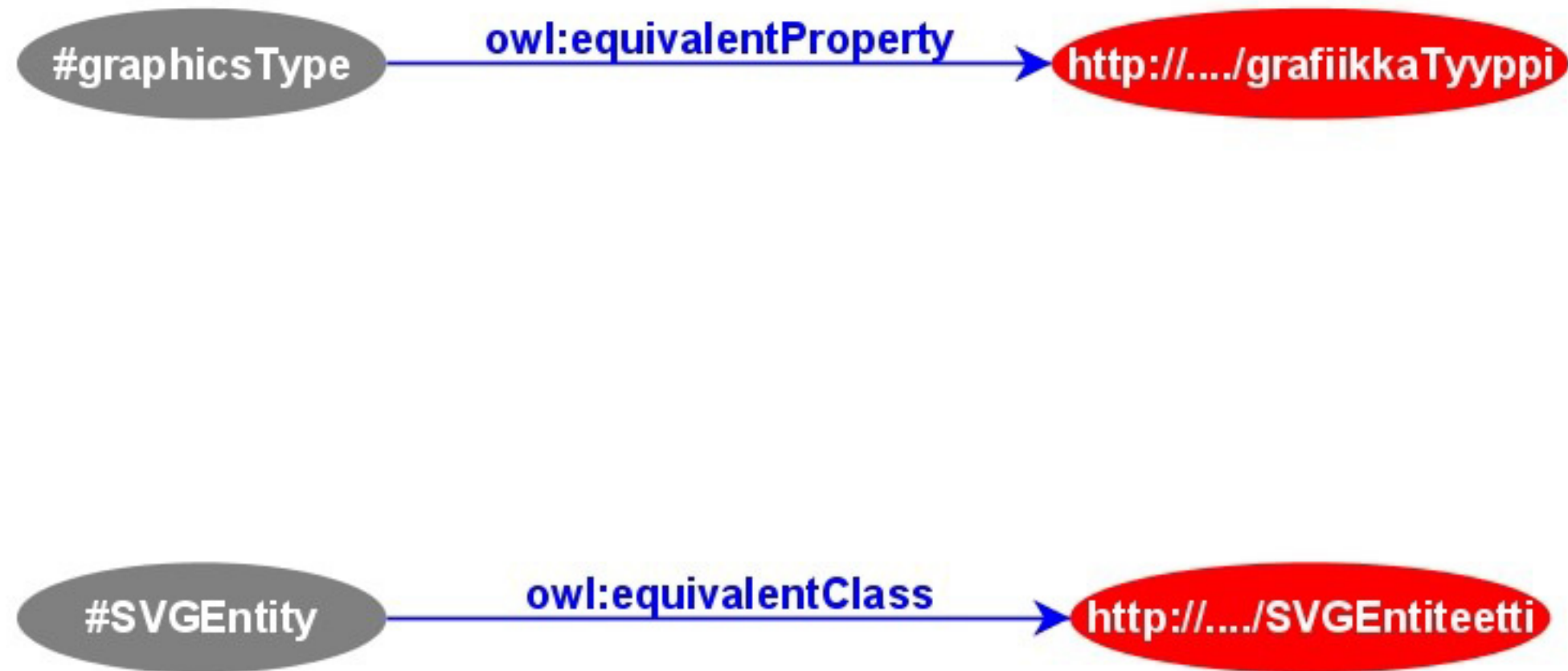
Characterization in XML:

```
<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!

- 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

- 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**



- 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:datatype="...#nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

- 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

- OWL expresses a *small subset* of First Order Logic
 - it has a “structure” (class hierarchies, properties, datatypes...), and “axioms” can be stated within that structure only
 - i.e., OWL uses FOL to describe “traditional” ontology concepts...
...but it is *not* a general logic system per se!
- Inference based on OWL is *within this framework only*
 - it seems modest, but has proven to be remarkably useful...
 - people in knowledge representation know that!

- The transitivity of **leftOf** is:

$$\forall x, y, z: (x \text{ leftOf } y \wedge (y \text{ leftOf } z)) \Rightarrow (x \text{ leftOf } z))$$

- Cardinality restriction:

$$\forall x: ((x \in X) \wedge (X \subset \text{dom}(\text{prop}))) \Rightarrow (\exists! y: x \text{ prop } y)$$

- Union, intersection, etc., can be trivially formalized, too
- etc.
- But, again: this is a *restricted form of FOL only!*

However: Ontologies are Hard!



- A full ontology-based application is a very complex system
- Hard to implement, may be 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

- 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

- *Goal: maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure is realizable*
- `owl:Class`, `owl:Thing`, `owl:ObjectProperty`, and `owl:DatatypeProperty` are *strictly separated*
 - i.e., a class *cannot* be an individual of another class
 - object properties' values must be an **`owl:Thing`**
 - except for `rdf:type`, `rdfs:subClassOf`, ...
- 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: provide a minimal useful subset, easily implemented*
 - simple class hierarchies can be built
 - property constraints and characterizations can be used
- **All of DL's restrictions, plus some more:**
 - class construction can be done *only* through:
 - intersection
 - property constraints

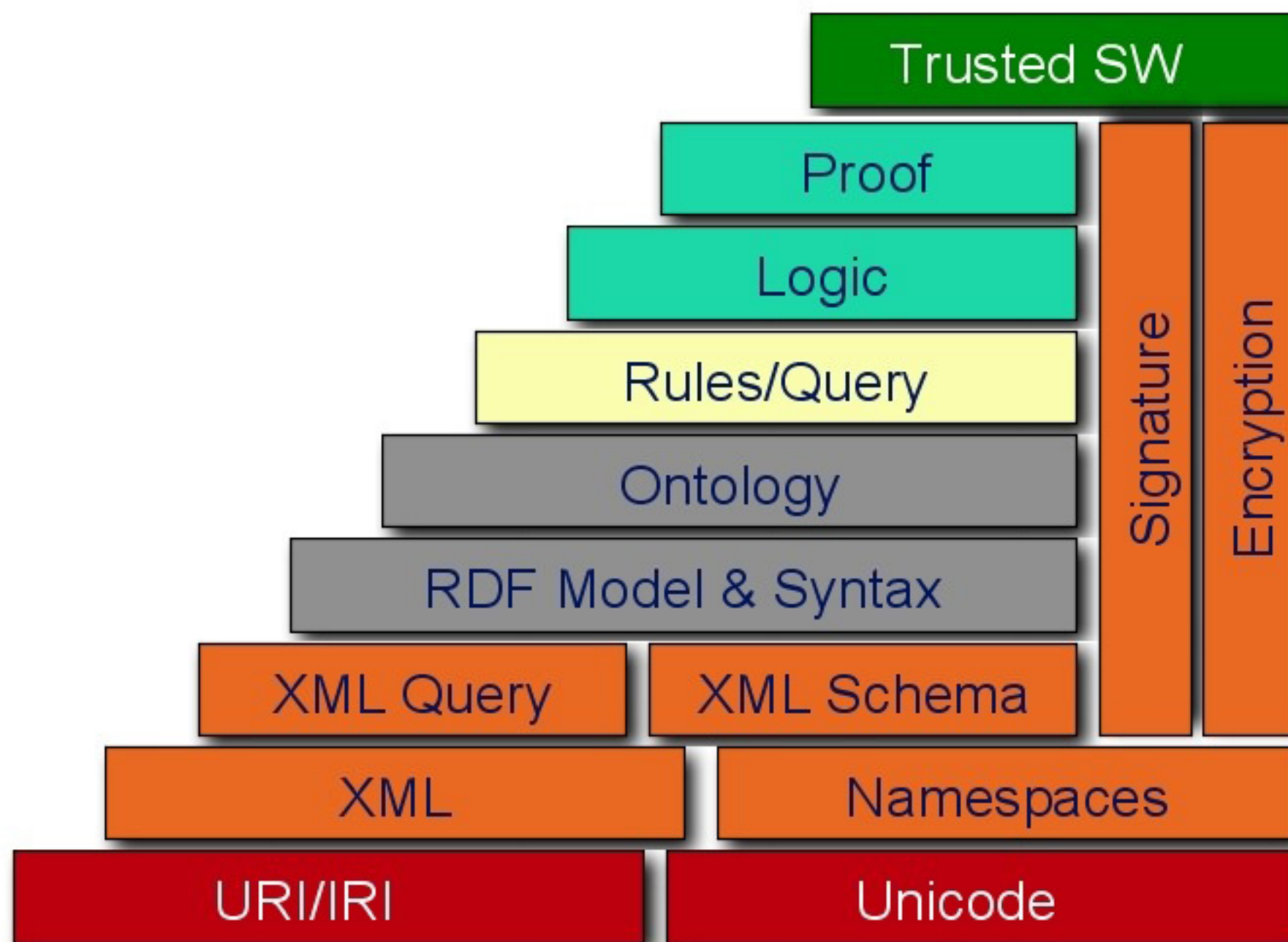
- The term refers to an area in knowledge representation
 - a special type of “structured” First Order Logic
 - there are several variants of Description Logic
 - i.e., OWL DL is an embodiment of *a* Description Logic
- Traditional DL terms sometimes used (by experts...):
 - “named objects, concepts”: definition of classes, individuals, ...
 - “axioms”: e.g., subclass or subproperty relationships, ...
 - “facts”: statements about individuals (**owl:Thing**-s)

none of these are “standardized” in W3C...

but you may see them in papers, references

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

PART VI: Future Developments



- 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 started work in this area recently
 - some initial drafts are already available

- In Python, for example, one uses:

```
# 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

- One may want something like:

```
SELECT (a,b)
```

```
WHERE [?x 'parent' a] and [b 'brother' ?x]
```

(i.e., 'b is the uncle of a')

- W3C started a standardization work in this area recently
 - precise relationships to XML Query has to be defined
 - concentrates also on *protocols* to extract subgraphs
 - e.g., using SOAP
- Such facilities already implemented in Jena, RAP,...

- OWL can be used for simple inferences
- Applications may require more, e.g., Horn clauses:
 - $(\text{ant-1} \wedge \text{ant-2} \wedge \dots) \Rightarrow (\text{cons-1} \wedge \text{cons-2} \wedge \dots)$
 - e.g.:
 - for *any* «X», «Y» and «Z»:
“if «Y» is a parent of «X», and «Z» is a brother of «Y»
then «Z» is the uncle of «X»”
 - using a logic formalism:
$$\forall x,z: ((\exists y: (y \text{ parent } x) \wedge (y \text{ brother } z)) \Rightarrow (z \text{ uncle } x))$$
- Lots of research is happening to extend RDF/OWL
(RuleML, SWRL, cwm, ...)
- W3C *may* initiate a standardization work in this area, too
 - question is whether results are “ripe” for standardization
 - and whether the necessary manpower is available

- 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)

- **Can I trust a metadata on the Web?**
 - is the author the one who claims he/she is?
 - can I check the credentials?
 - can I trust the inference engine?
 - what about IPR of the metadata?
 - etc.
- **Some of the basic building blocks are available:**
 - XML Signature/Encryption
 - XML based Key Management is in preparation
- **Much is missing, e.g.:**
 - a “canonical” form of RDF/XML
 - necessary for unambiguous signatures
 - exhaustive tests for inference engines
 - protocols to check, for example, a signature
- **It is on the “future” stack of W3C...**

- Knowledge representation is an active R&D area:
 - temporal & spatial reasoning
 - fuzzy logic
 - improve the inference algorithms and implementations
 - improve scalability
 - reasoning with OWL Full
 - ...
- They usually happen outside of W3C, though
 - W3C is not a research entity...

PART VII: Available Documents, Tools

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/>

RDF: Concepts and Abstract Syntax

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

Note: there is a previous Recommendation of 1999 that is superseded 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!

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/>

- 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
- J. Davies, D. Fensel, F. van Harmelen: Towards the Semantic Web (2002)
- S. Powers: Practical RDF (2003)
- D. Fensel, J. Hendler: Spinning the Semantic Web (2003)
- G. Antoniu, F. van Harmelen: Semantic Web Primer (2004)
- ...

- **Bristol University**
 - <http://www.ilrt.bristol.ac.uk/discovery/rdf/resources/>
 - huge list of documents, publications
- **Semantic Web Community Portal**
 - <http://www.semanticweb.org/>
 - “Business model IG” (part of the portal)
 - huge set of links to documents, software, ...
- **SemWeb Central**
 - <http://semwebcentral.org>
 - Open Source development archive
- **W3C team public presentations:**
 - <http://www.w3.org/2001/sw/EO/talks>
- **W3C’s Semantic Web home page:**
 - <http://www.w3.org/2001/sw/>

- Full, interactive view of the **RDFS** and **OWL** definitions
 - requires an SVG client
- 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**

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/>

(Graphical) Editors

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

Further info on RDF/OWL tools at:

<http://www.w3.org/2001/sw/WebOnt/impls>, or

<http://semwebcentral.org>

Programming environments

We have already seen some

but Jena 2 and SWI-Prolog do OWL reasoning, too!

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/Ontology registries


e.g., SchemaWeb, SemWeb Central, ...

PART VII: Some Application Examples

- 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!

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



[ABOUT THE INITIATIVE](#) | [DOCUMENTS](#) | [GROUPS](#) | [RESOURCES](#)
[DCMI NEWS](#) | [TOOLS AND SOFTWARE](#) | [MEETINGS AND PRESENTATIONS](#) | [PROJECTS](#)

Dublin Core Metadata Initiative

Making it easier to find information.

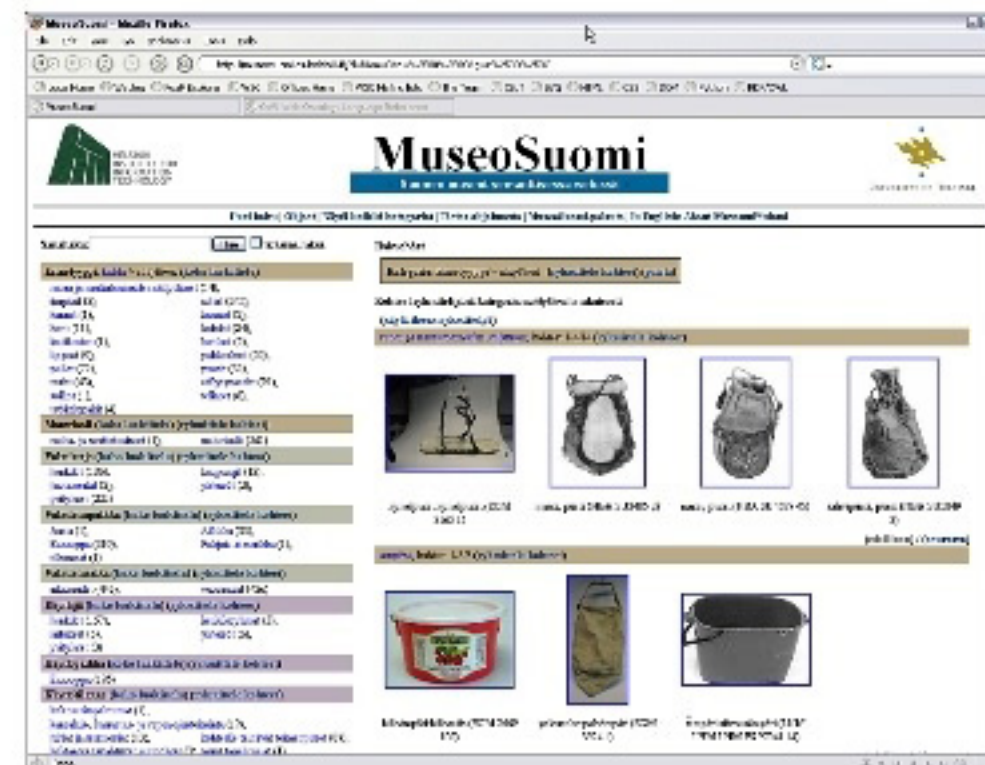
The Dublin Core Metadata Registry

The [Dublin Core Metadata Initiative's](#) Metadata Registry is an application designed to enable users to explore the DCMI vocabulary in a way that simplifies the discovery and navigation of terms and their definitions, and that illustrates the relationship between terms. The goal of the Registry is to promote the discovery, reuse and extension of existing semantics, and to facilitate the creation of new vocabularies.

Help	Preferences	Search	Administration
Please select from one of the following supported languages or click on the Preferences link above for additional options. Having trouble displaying the international fonts? Click here for help. * DCES-only translations	العربية/النسخة/العربي [ar-SA]	Catalan [ca-ES]	
	Česky [cs-CZ]	Cymraeg [cy-GB]*	
	Dansk [da-DK]*	Deutsch [de-DE]	
	Ελληνικά [el-GR]	English [en-US]	
	Español [es-ES]	Suomeksi [fi-FI]	
	Français [fr-FR]	Italiano [it-IT]	
	日本語 [ja-JP]	한국어 [ko-KR]	
	मराठी [mr-IN]	Norsk [no-NO]*	
	Polski [pl-PL]	Português [pt-PT]	
	Русский [ru-RU]	Svenska [sv-SE]	
	ไทย [th-TH]	українська [uk-UA]	
	繁体中文 [zh-CN]	繁體中文 [zh-TW]	

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



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/)

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

The top-left screenshot shows the W3C home page with an RSS reader sidebar. The top-right screenshot shows a W3C news article about the XML Binary Characterization Working Group. The bottom-right screenshot shows a Meerkat RSS feed for W3C news.

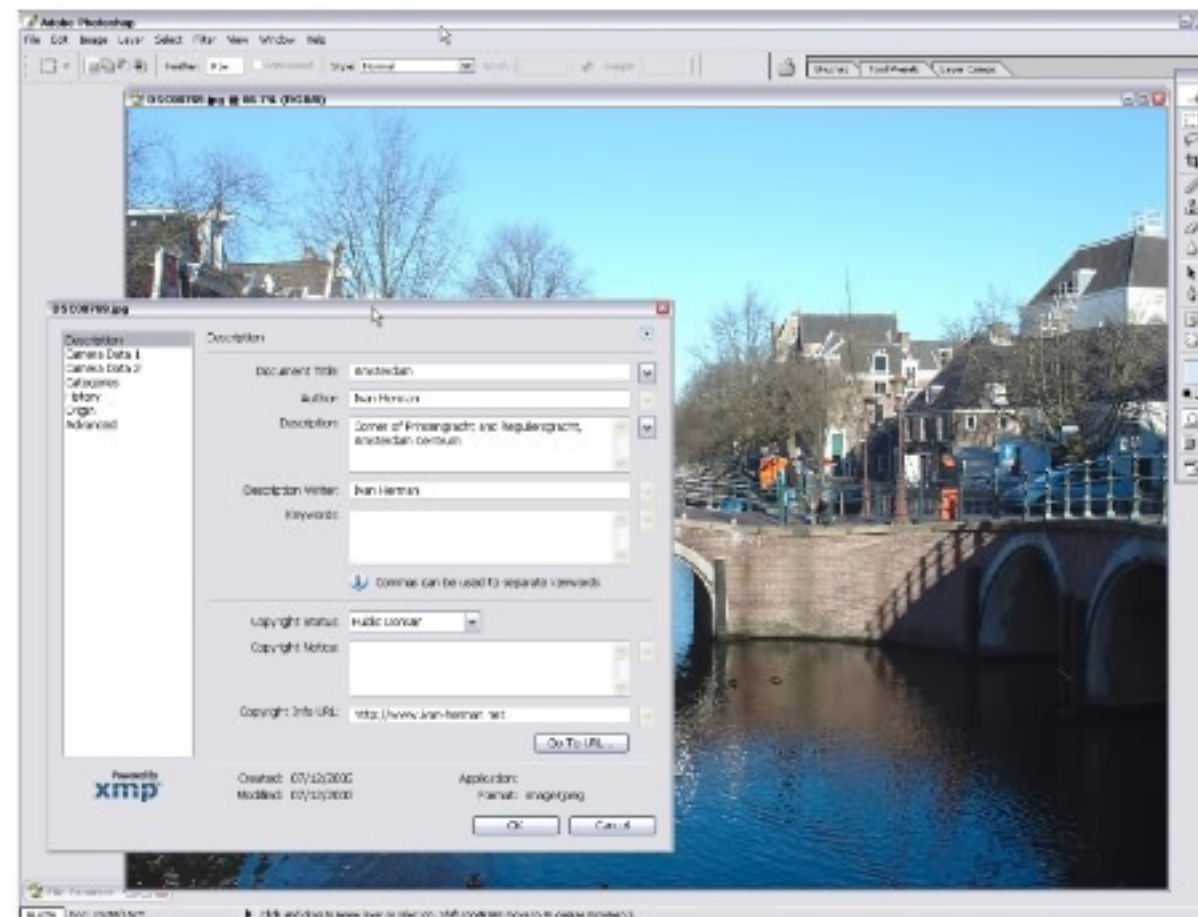
W3C Home Page (Left): The W3C logo and tagline "Leading the Web to Its Full Potential..." are visible. Below the logo, there are links to "Activities", "Technical Reports", "Site Index", "New Visitors", "About W3C", and "Join W3C". A sidebar on the left contains a list of W3C activities and technologies, including Accessibility, Arrays, Annotea, COOP, CSS, CSS Validator, Device Independence, DDM, HTML, HTML Tidy, HTML Validator, HTTP, iWML, Internationalization, Jigsaw, Lowwww, MATHML, Multimodal Interaction, OWL, Patent Policy, PICS, and PEARL.

W3C News Article (Top Right): The article is titled "W3C Launches XML Binary Characterization Working Group". It mentions that W3C is pleased to announce the creation of the XML Binary Characterization Working Group in the XML Activity. The group will analyze and develop use cases and measurements for alternate encodings of XML. Its goal is to determine if serialized binary XML transmission and formats are feasible. Participation is open to W3C Members. Read about the XML Activity (News archive).

Meerkat RSS Feed (Bottom Right): The feed is titled "MEERKAT: AN OPEN WIRE SERVICE". It shows a list of news items. The first item is "W3C Link Checker Released" with a date of 2004-04-01. The second item is "W3C Talks in April" with a date of 2004-03-29. The feed includes a search bar and a table of news items with columns for Action, Story, Source, Category, and Date.

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>



Mozilla

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

Brandsoft

- enterprise 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...

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!*

CST Baby CareLink

Product Map

CST Baby CareLink is a complete maternal/child health solution.

To view the contents of each component, mouse over the sections or click directly on them to view a complete product description.

Prenatal Care	Neonatal Intensive Care	Infant Care
Clinician Tools		
Healthy Beginnings	High-Risk Pregnancy	Neonatal Intensive Care
After the NICU		
First Year of Life		
Care Manager Tools		
<ul style="list-style-type: none"> Prescribed Education Discharge Coordination Assessments Registration Census Reporting Message Center 		

Components:

- Neonatal Intensive Care
- Neonatal Care Management Program
- After the NICU
- Healthy Beginnings / First Year of Life
- High Risk Pregnancy

Did You Know?

7.0% (300,000) of all births in the U.S. each year are low birthweight (< 2500 gms, 5 pounds, 8 ounces).

© 2004 Clinician Support Technology - One

These slides are at:

<http://www.w3.org/2004/Talks/0209-Helsinki-IH/>

Semantic Web homepage

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

More information about W3C:

<http://www.w3.org//Consortium/>

Finnish Office of W3C

<http://www.w3c.tut.fi/>

Mail me:

ivan@w3.org

Questions?