Tutorial on Semantic Web Technologies

Trento, Italy, on the 14th December, 2005

Tutorial on Semantic Web Technologies

Slides of the tutorial given in Trento, Italy, on the 14th of December, 2005, as part of the 2nd Italian Semantic Web Workshop on Semantic Web Applications and Perspectives (SWAP2005), held at the University of Trento on the 14-15-16 December, 2005.

These slides are in XHTML. A printable version in PDF is also available.

Introduction

Towards a Semantic Web

- The current Web represents information using
  - *natural language (English, Hungarian, Italian,…)*
  - *graphics, multimedia, page layout*
- Humans can process this easily
  - *can deduce facts from partial information*
  - *can create mental associations*
  - *are used to various sensory information*
    - (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!*

However…

- 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?
  - *…*

Example: Searching

- The best-known example…
  - *Google et al. are great, but there are too many false hits*
  - *adding (maybe application specific) descriptions to resources should improve this*

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 (i.e., on the Web!)
- (M. Dertouzos: The Unfinished Revolution)

Example: Data(base) Integration

- Databases are very different in structure, in content
- Lots of applications require managing *several* databases
  - *after company mergers*
  - *combination of administrative data for e-Government*
  - *biochemical, genetic, pharmaceutical research*
  - *etc.*
- Most of these data are now on the Web (though not necessarily public yet)
- The *semantics* of the data(bases) should be known (how this semantics is mapped on internal structures is immaterial)

Example: Digital Libraries

- It is a bit like the search example
- It means catalogs 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*
- But it is more: software agents should also be librarians!
  - *help you in finding the right publications*

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"*
  - *but 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?

- (Some) data should be available for machines for further processing
- Data should be possibly exchanged, merged, combined on a Web scale
- Sometimes, data may describe other data (like the library example, using metadata)…
- … but sometimes the data is to be exchanged by itself, like my calendar or my travel preferences
- Machines may also need to *reason* about that data

What Is Needed (Technically)?

- To make data machine processable, we need:
  - *unambiguous names for resources that may also bind data to real world objects: URI-s*
  - *a common data model to access, connect, describe the resources: RDF*
  - *access to that data: SPARQL*
  - *common vocabularies: RDFS, OWL, SKOS*
  - *reasoning logics: OWL, Rules*
- *The "Semantic Web" is an infrastructure for the interchanging and integrating data 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, interchanging, integrating data*
  - *of course: AI systems may use the data 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, Nokia, Vodaphone, …)*

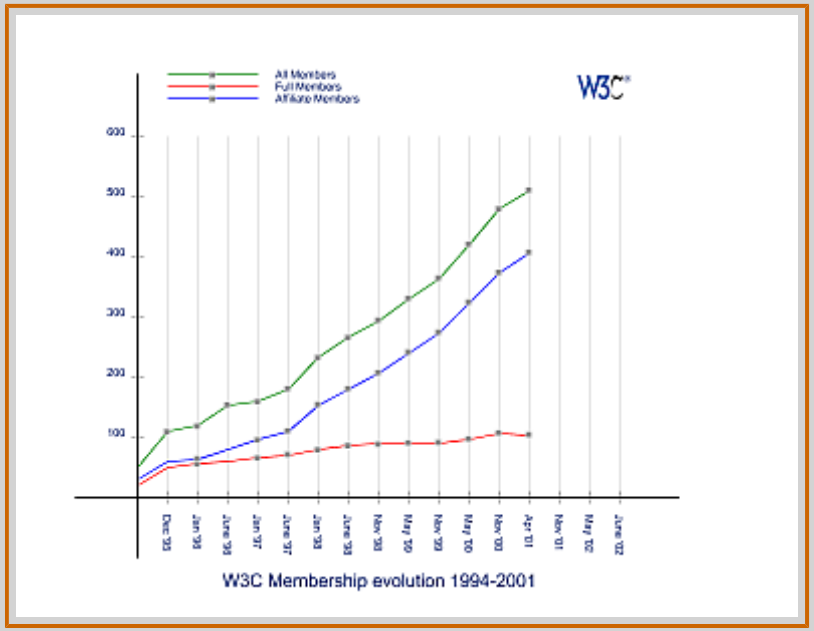
This Course Will

- 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

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 data formalism for that metadata*



Statements

- The data 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, …*

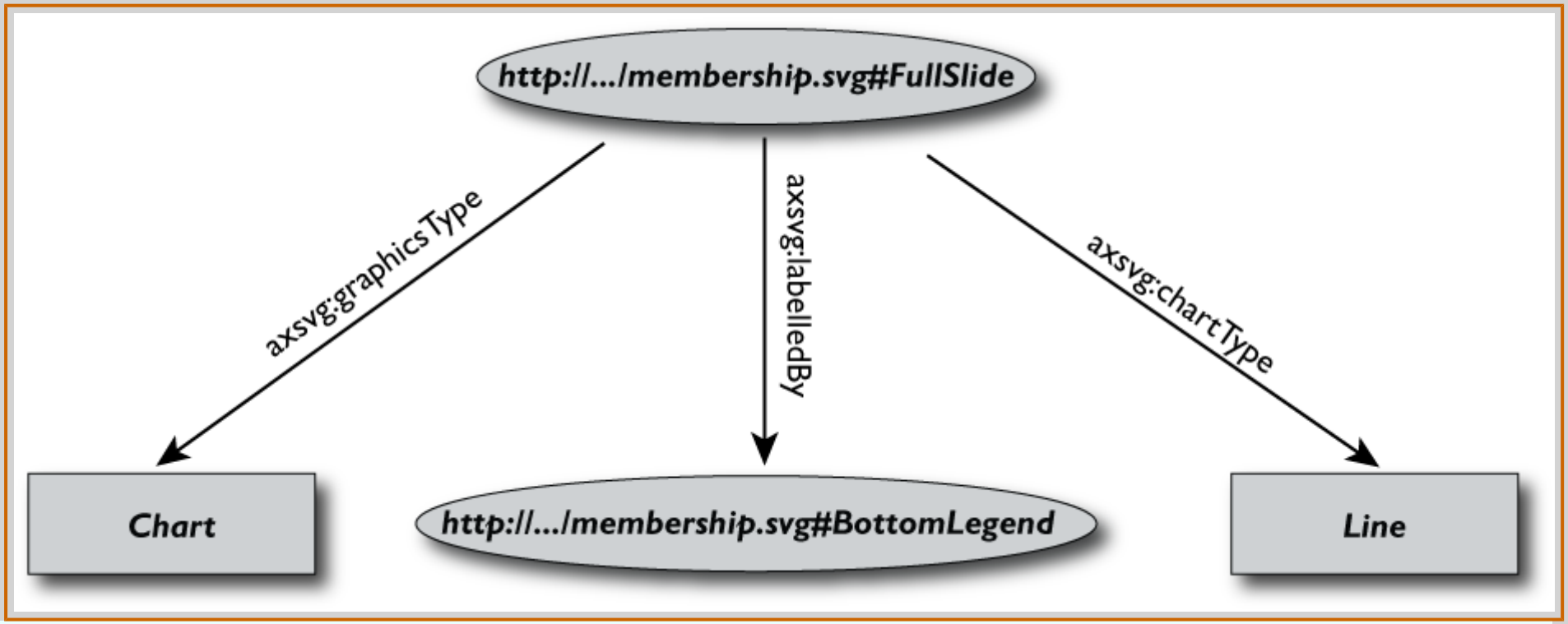
Resource Description Framework

- 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
    - you can also think about a property/value pair *attached to a resource*
- *RDF* is a general model for such statements
  - *… with machine readable formats (RDF/XML, Turtle, n3, RXR, …)*
  - *RDF/XML is the "official" W3C format*

RDF is a Graph

- An (s,p,o) triple can be viewed as a labeled edge in a graph
  - *i.e., a set of RDF statements is a directed, labeled graph*
    - both “objects” and “subjects” are the graph nodes
    - “properties” are the edges
- One should “think” in terms of graphs; XML or Turtle syntax are only the tools for practical usage!
- RDF authoring tools usually work with graphs, too (XML or Turtle 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://...#BottomLegend/>
    <axsvg:chartType>Line</axsvg:chartType>
</rdf:Description>
```

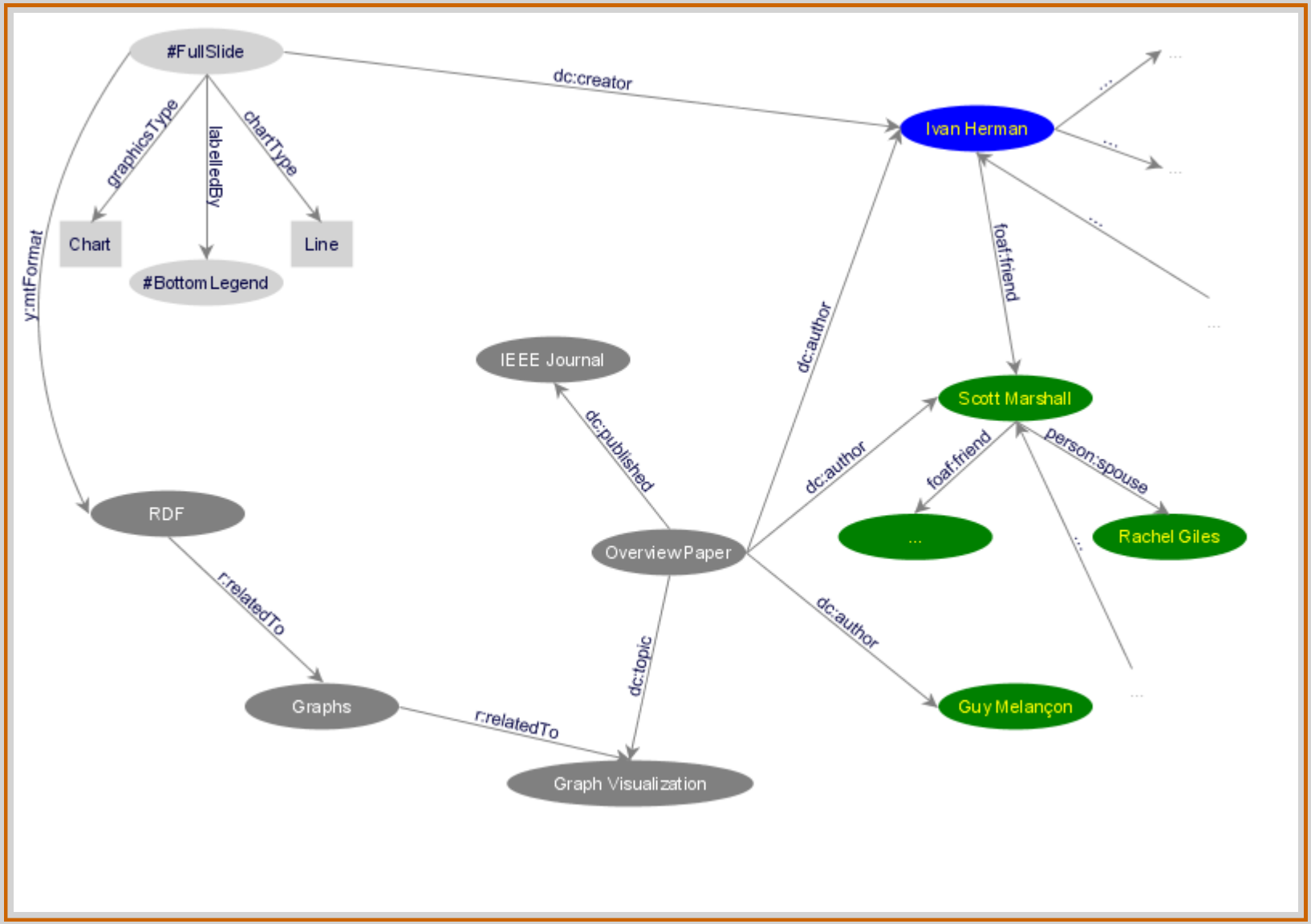
URI-s Play a Fundamental Role

- *Anybody* can create (meta)data on *any* resource on the Web
  - *e.g., the same SVG file could be annotated through other terms*
  - *semantics is added to existing Web resources via URI-s*
- Data exist on the Web, because it is accessible through standard Web means
- *URI-s ground RDF into the Web*
  - *information can be retrieved using existing tools*
  - *this makes the "Semantic Web", well… "Semantic Web"*

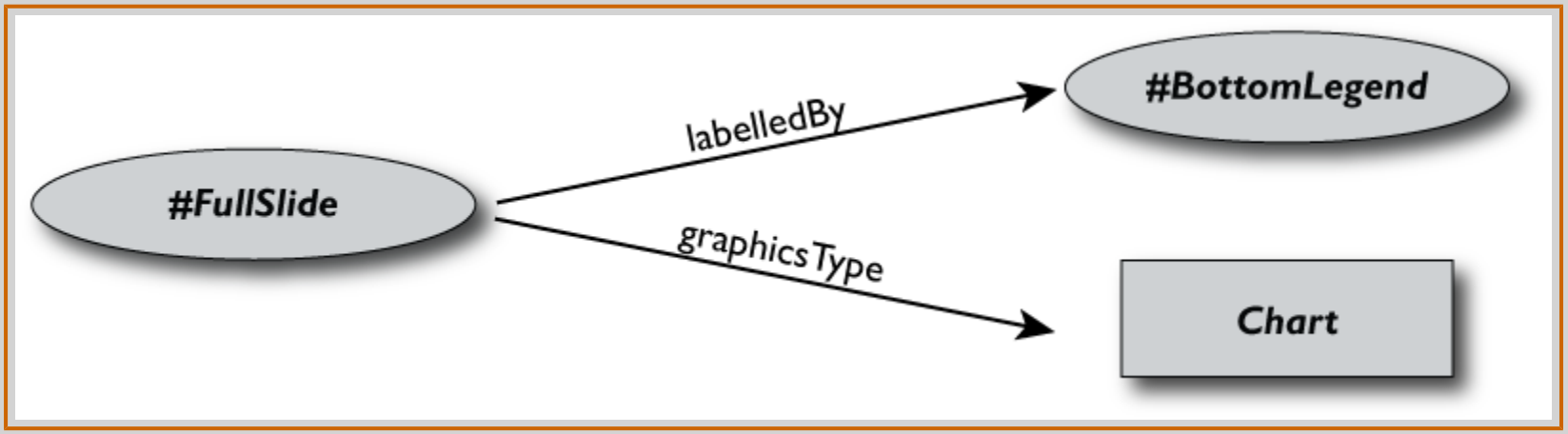
URI-s: Merging

- It becomes easy to *merge* data
  - *e.g., applications may merge the SVG annotations*
- Merge can be done because statements refer to the *same* URI-s
  - *nodes with identical URI-s are considered identical*
- Merging is a *very* powerful feature of RDF
  - *metadata may be defined by several (independent) parties…*
  - *…and combined by an application*
  - *one of the areas where RDF is much handier than pure XML*

What Merge Can Do...



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-ns#">
    <rdf:Description rdf:about="#FullSlide">
       «Element for labelledBy»
           <rdf:Description rdf:about="#BottomLegend"/>
       «/Element for labelledBy»
    </rdf:Description>
<rdf:RDF>
```

- Note the usage of *namespaces*!

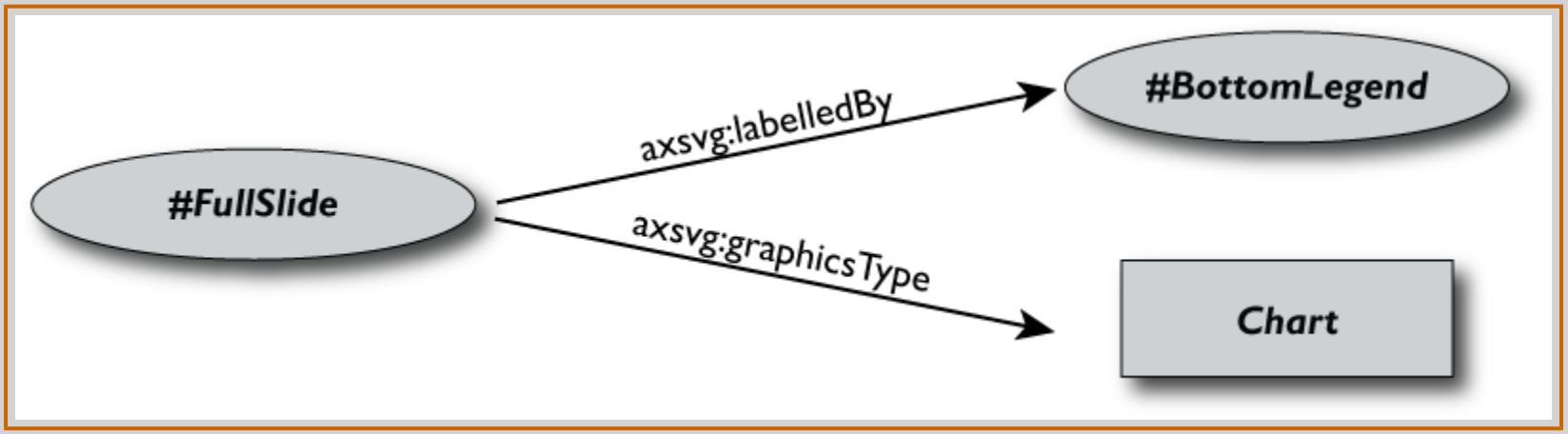
RDF/XML Principles (cont)

- 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-ns#">
    <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…)

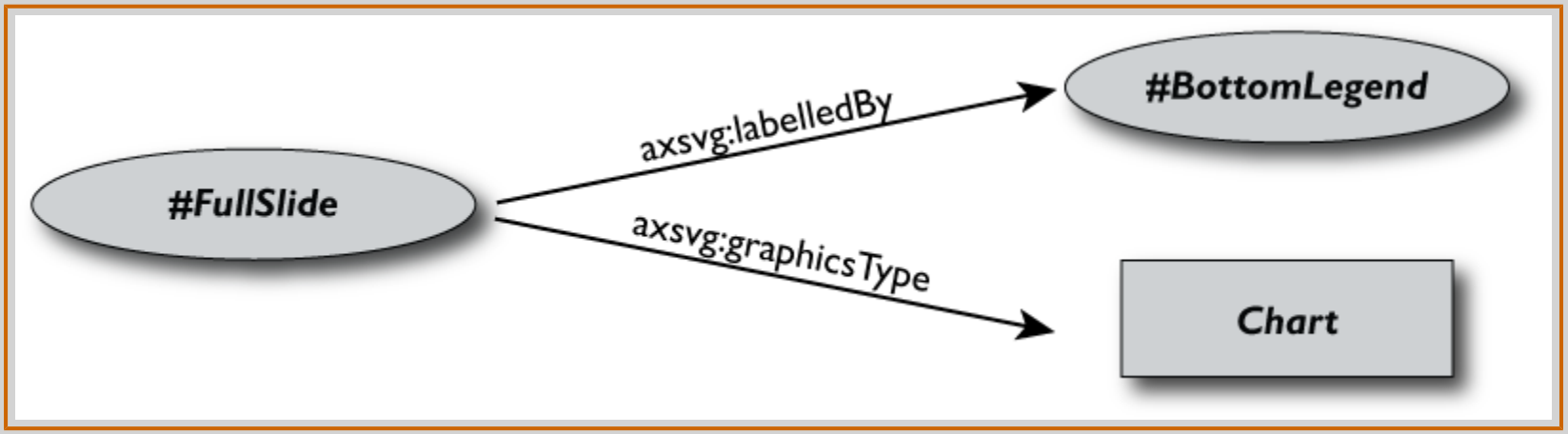
Several Properties on the Same Node



- 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

- (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>
```

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 PHP, Java, etc. (see later)

Python Example

In Python syntax:

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

Merge in Practice

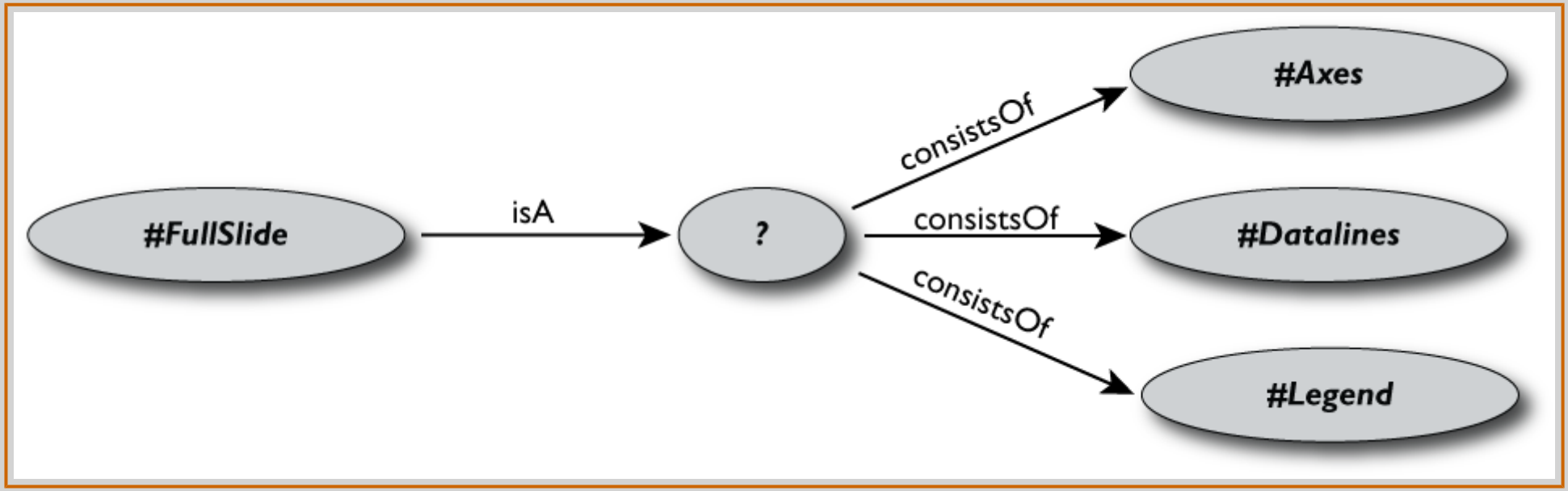
- Development environments merge graphs automatically
  - *e.g., in Python, the Triple Store can “load” several files*
  - *the load merges the new statements automatically*
- Merging the RDF/XML files into one is also possible
  - *but not really necessary, the tools will merge them for you*
  - *keeping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

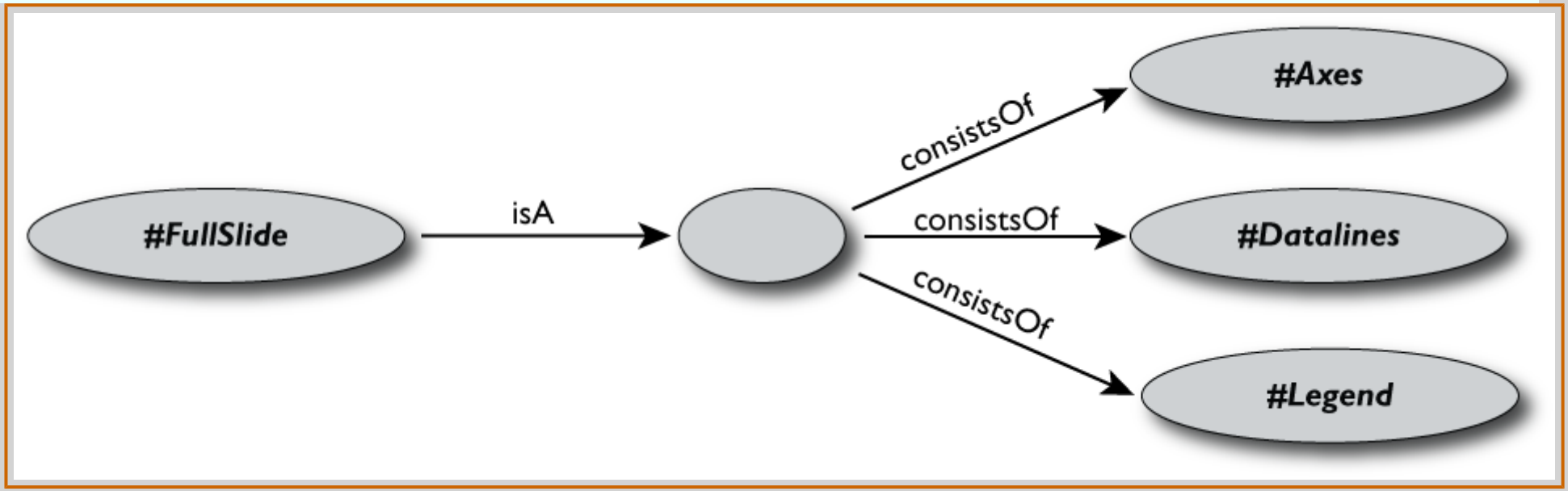
```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:isA>
       <rdf:Description rdf:nodeID="A234"/>
   </axsvg:isA>
</rdf:Description>
<rdf:Description rdf:nodeID="A234">
   <axsvg:consistsOf rdf:resource="#Axes"/>
   …
</rdf:Description>
```

- Almost like `rdf:ID`, but…
- …`A234` is *invisible* from outside the file
  - *it is mapped on an internal identifier*
  - *the same value can be repeated to denote the same resource*

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"/>
          …
      </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:isA>
</rdf:Description>
```

RDF Vocabulary Description Language

(a.k.a. RDFS)

Need for RDF Schemas

- Defining the data 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? Are there (logical) relationships among the terms?
- This is where RDF Schemas come in
  - *officially: "RDF Vocabulary Description Language"; the term "Schema" is retained for historical reasons…*

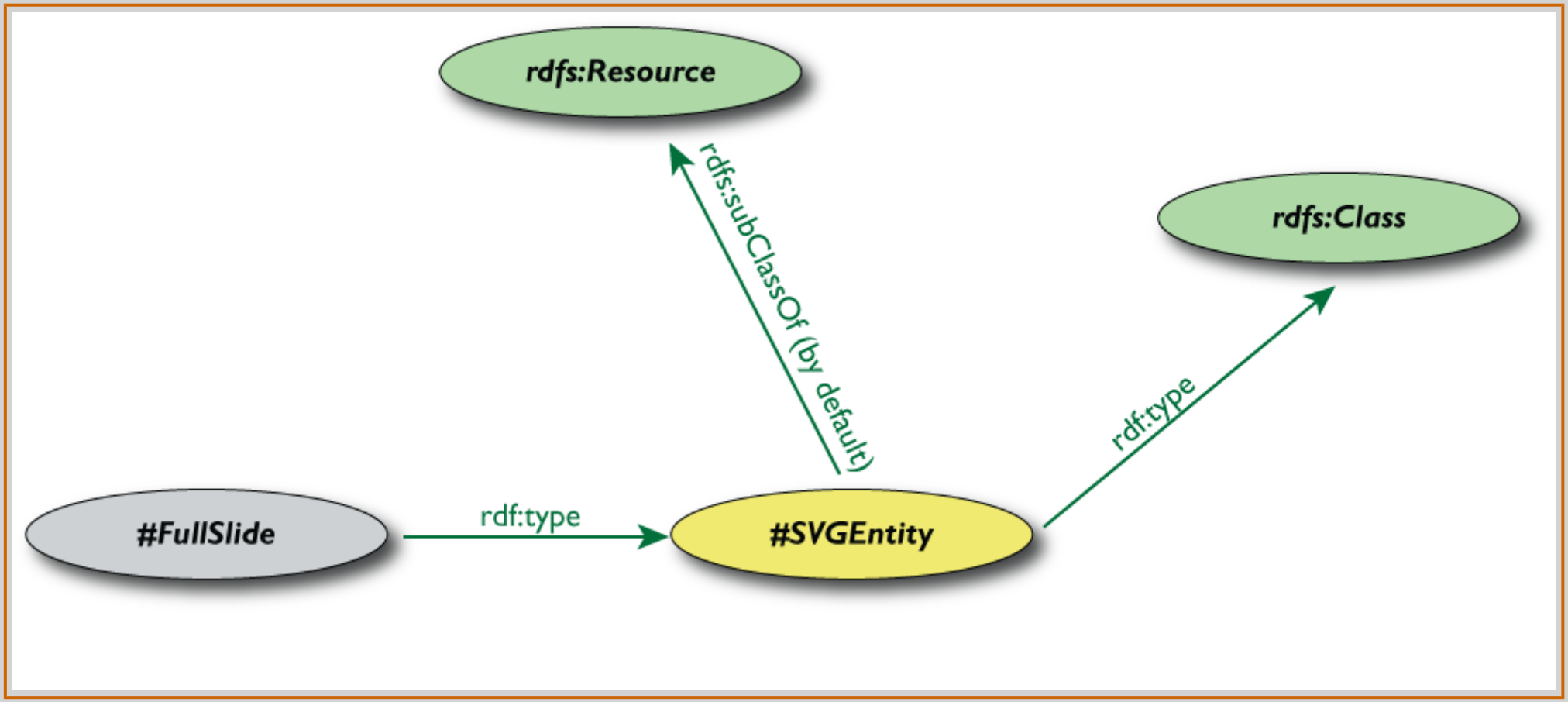
Classes, Resources, …

- Think of well known in traditional ontologies:
  - *use the term "mammal"*
  - *"every dolphin is a mammal"*
  - *"Flipper is a dolphin"*
  - *etc.*
- RDFS defines *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")*
    - "mammal", "dolphin", …

Classes, Resources, … (cont.)

- Relationships are defined among classes/resources:
  - *“typing”: an individual belongs to a specific class (“Flipper is a dolphin”)*
  - *“subclassing”: instance of one is also the instance of the other (“every dolphin is a mammal”)*
- *RDFS formalizes these notions in RDF*

Classes, Resources in RDF(S)



- RDFS defines `rdfs:Resource`, `rdfs:Class` as nodes; `rdf:type`, `rdfs:subClassOf` as properties

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#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 traditional knowledge representation this separation is often referred to as: “Terminological axioms”, “Terminological box”, “TBox”; and “Assertions”, “Assertion box”, “ABox”

An Aside: Typed Nodes in RDF/XML

- 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>
```

Schema Example in RDF/XML (alt.)

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

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

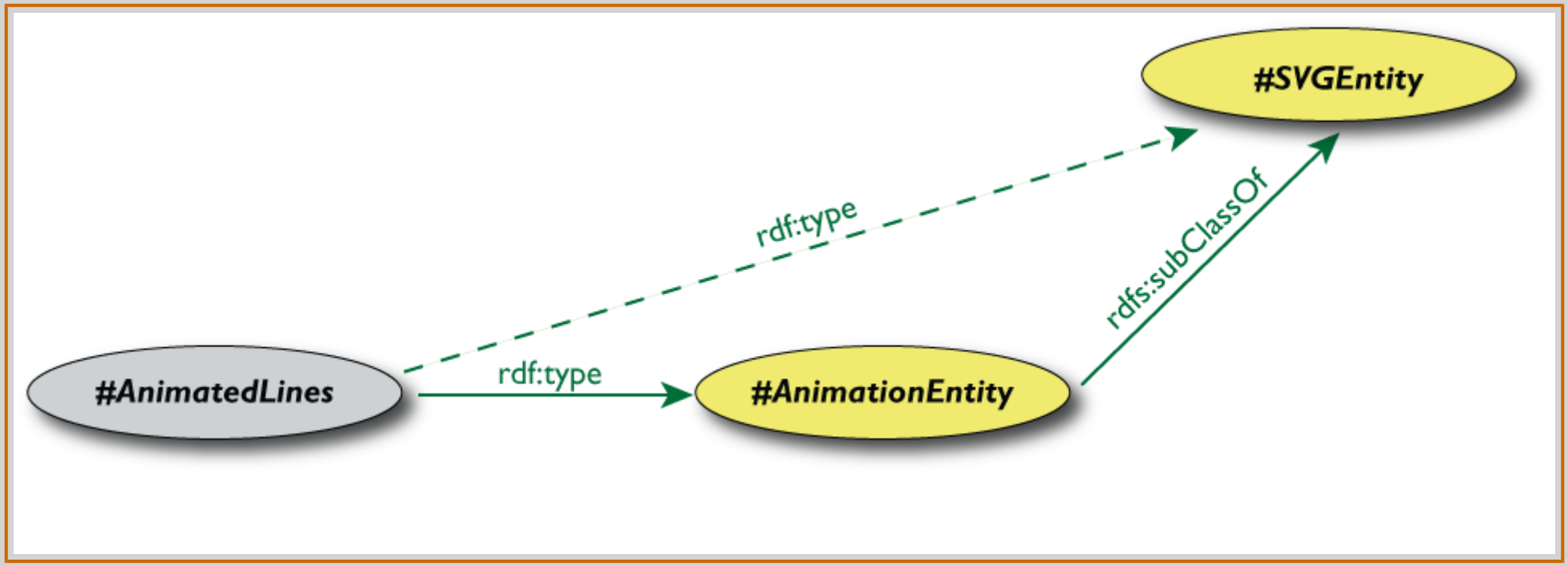
- In the rdf data on a specific graphics:

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

Typed Nodes (cont)

- A resource may belong to several classes
  - `rdf:type` *is just a property…*
  - *"Flipper is a mammal, but Flipper is also a TV star…"*
- i.e., it is *not* like a datatype in this sense!
- The type information may be very important for applications
  - *e.g., it may be used for a categorization of possible nodes*

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

Inference: Let Us Be Formal…

- The RDF Semantics document has a list of (44) *entailment rules*:
  - *"if such and such triplets are in the graph, add this and thistriplet"*
  - *do that recursively until the graph does not change*
  - *this can be done in polynomial time for a specific graph*
- The relevant rule for our example:

```
If:
  uuu rdfs:subClassOf xxx .
  vvv rdf:type uuu .
Then add:
  vvv rdf:type xxx .
```

- Whether those extra triplets are *physically added* to the graph, or *deduced* when needed is an implementation issue

Properties (Predicates)

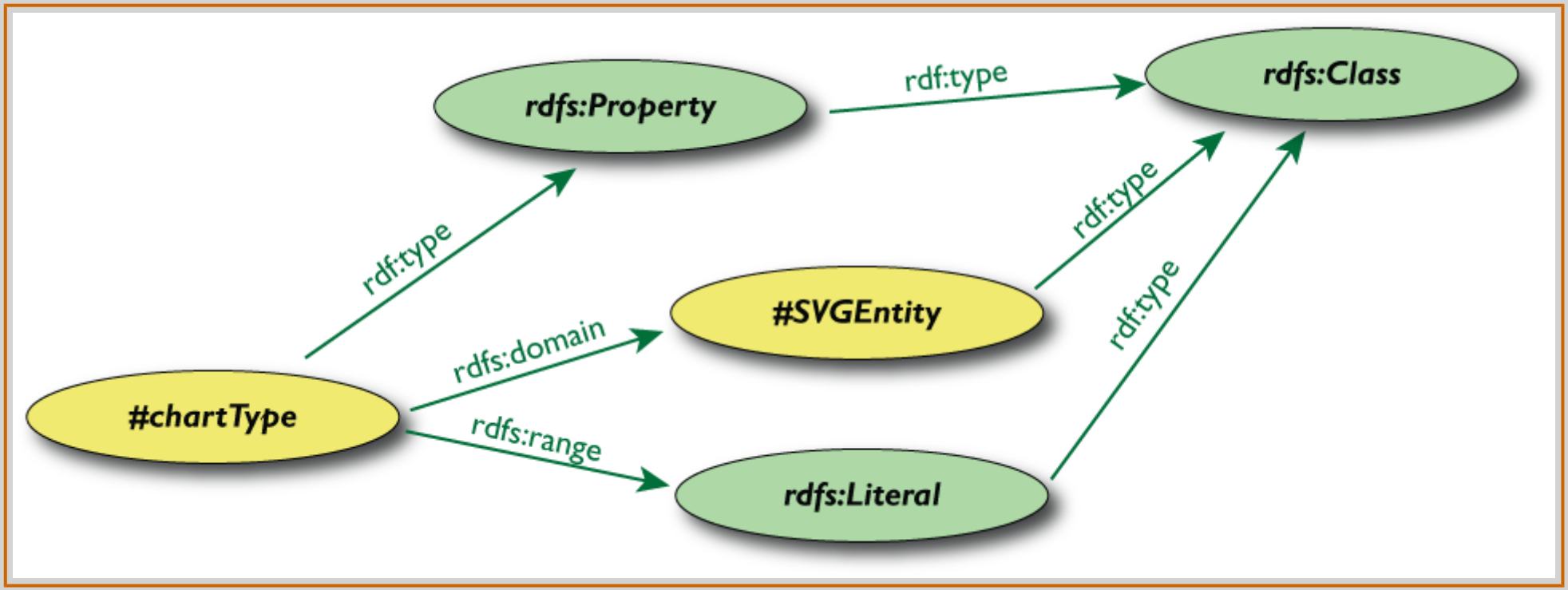
- Property is a special class (`rdf:Property`)
  - *i.e., properties are also resources*
- Properties are constrained by their range and domain
  - *i.e., what individuals can be on the "left" or on the "right"*
- There is also a possibility for a "sub-property"
  - *all resources bound by the "sub"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

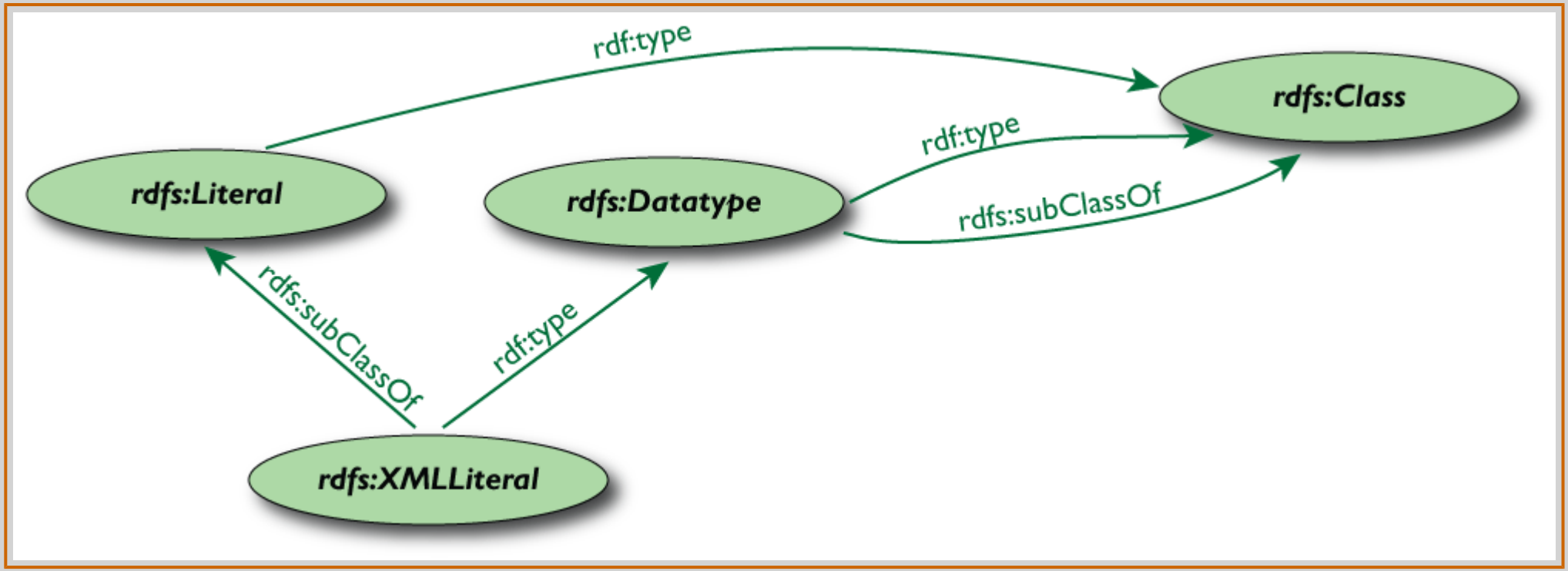
Property Specification in XML

Same example in XML/RDF:

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

Literals

- Literals may have a data type (floats, ints, etc) defined in XML Schemas, including full XML Fragments
- (Natural) language can also be specified (via `xml:lang`)



Literals in RDF/XML

- Typed literals:

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

Literals in RDF/XML (cont.)

- 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/>
```

Some Predefined Classes (Collections, Containers)

Predefined Classes

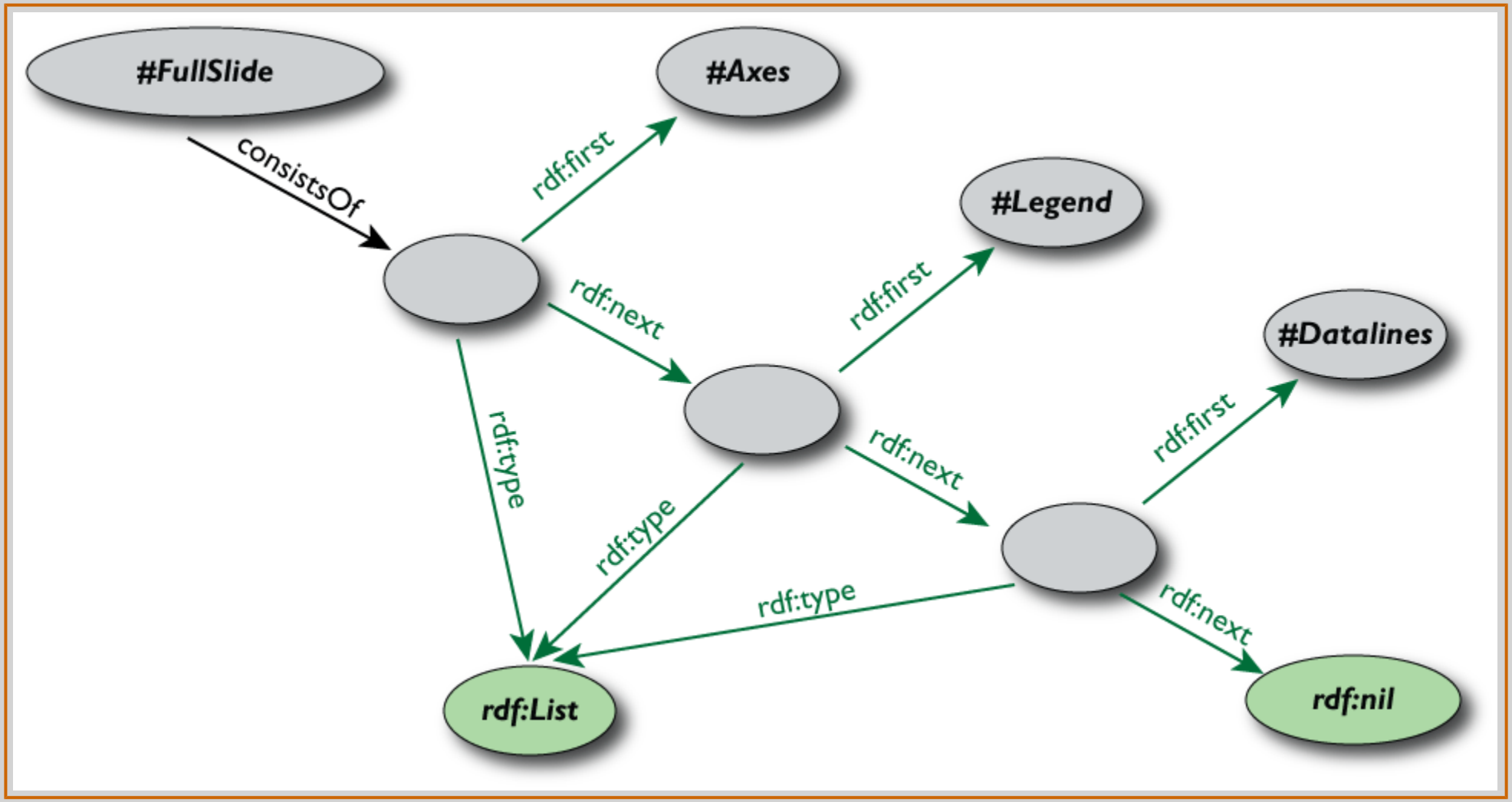
- RDF(S) has some predefined classes (and related properties)
- They are not new “concepts” in the RDF Model, just classes with an agreed semantics
- These are:
  - *collections (a.k.a. lists)*
  - *containers: sequence, bag, alternatives*

Collections (Lists)

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

Collections (Lists) (cont.)

- Familiar structure for Lisp programmers…

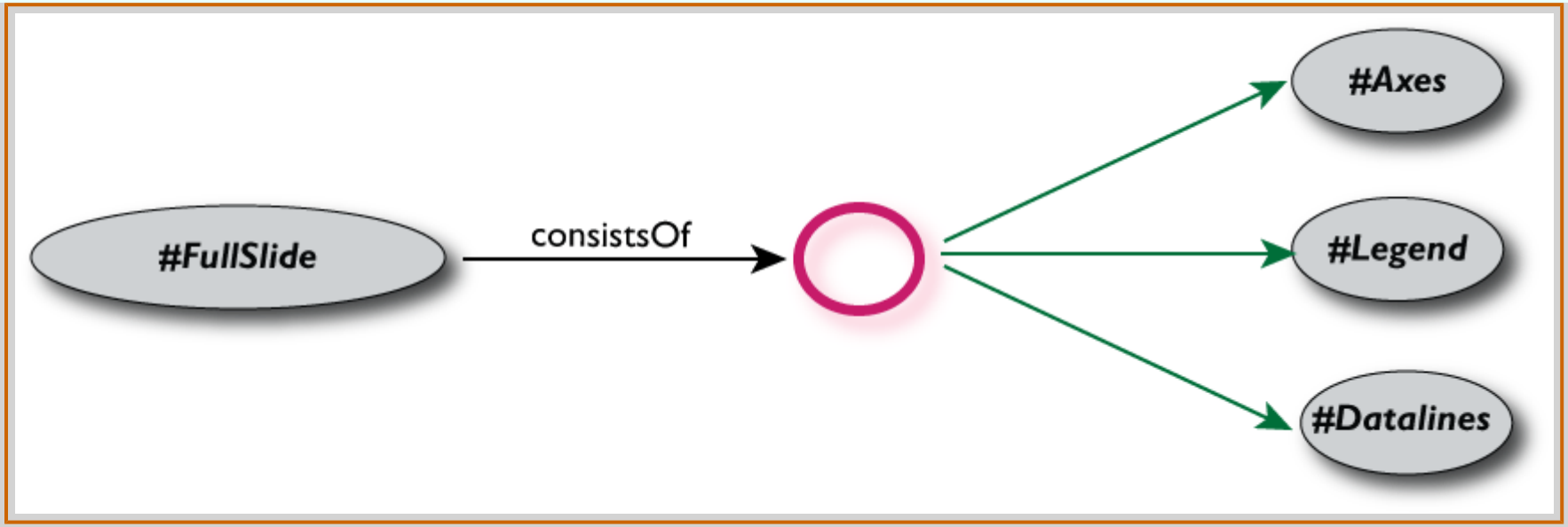


The Same in RDF/XML

```
<rdf:Description rdf:about="#FullSlide">
    <axsvg:consistsOf rdf:parseType="Collection">
        <rdf:Description rdf:about="#Axes"/>
        <rdf:Description rdf:about="#Legend"/>
        <rdf:Description rdf:about="#Datalines"/>
    </axsvg:consistsOf>
</rdf:Description>
```

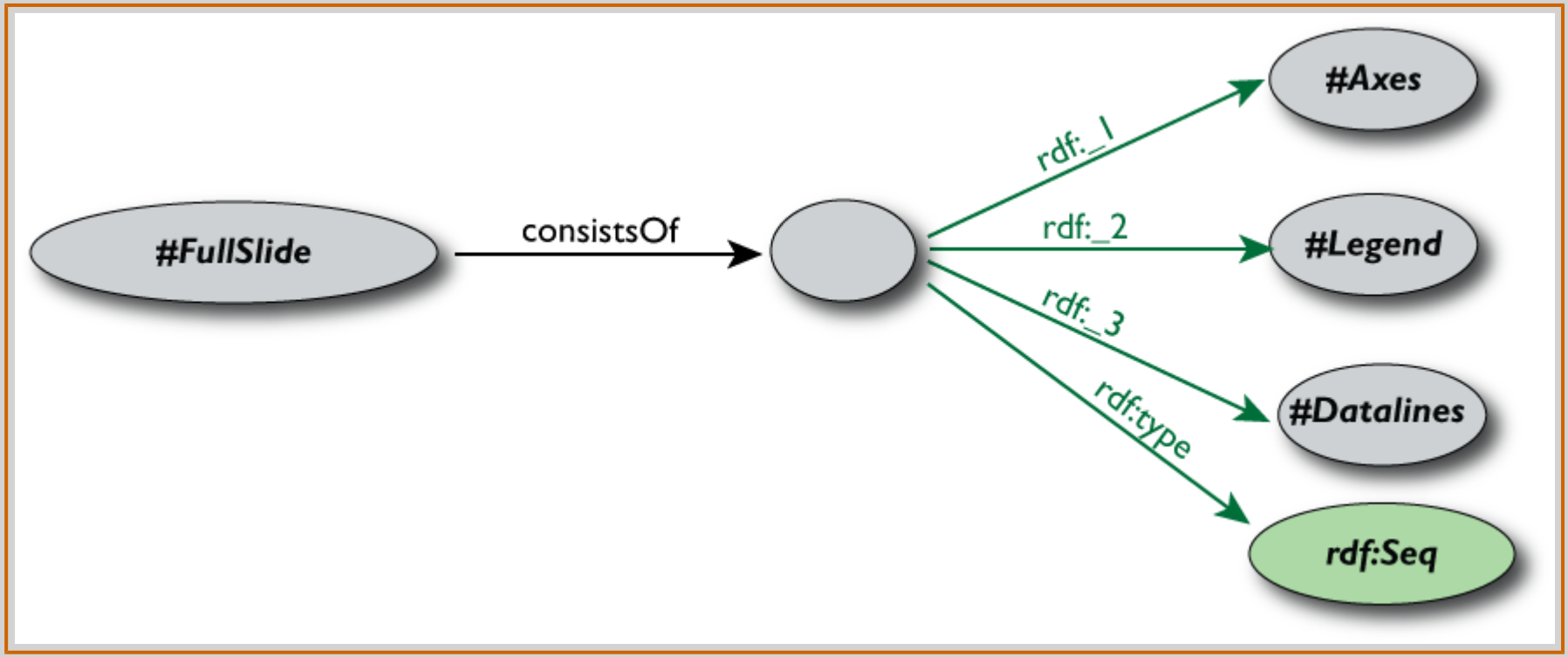
Our Graphical Shorthand

```
<rdf:Description rdf:about="#FullSlide">
    <axsvg:consistsOf rdf:parseType="Collection">
        <rdf:Description rdf:about="#Axes"/>
        <rdf:Description rdf:about="#Legend"/>
        <rdf:Description rdf:about="#Datalines"/>
    </axsvg:consistsOf>
</rdf:Description>
```



Sequences

- Use the predefined:
  - *RDF class* ***Seq***
  - *RDF properties* ***rdf:_1***, ***rdf:_2***, …
- The *agreed semantics* is of a sequential containment



Sequences (in RDF/XML)

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

Sequences (in simplified RDF/XML)

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

Other Containers

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

Some Words of Warning

- RDF/XML introduces a number of simplifications
  - *usage of* **`rdf:parseType`** *instead of* **`rdf:first`**, **`rdf:rest`**, *…*
- This can be deceptive when using, e.g., RDFLib:
  - **`rdf:Seq`** *does not appear directly; instead, a (possibly blank) node with a* **`rdf:type`** *property*
- *Never forget: only the graph is "real", the rest is convenience!*

RDF(S) in Practice

Small Practical Issues

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

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 (using conventions instead of namespaces in metas*
  - *put RDF in a comment (e.g., Creative Commons)*

RDF Can Also Be Generated

- Use intelligent “scrapers” or “wrappers” to extract a structure (hence RDF) from a Web page…
  - *using conventions in, e.g., class names*
  - *using the header conventions*
- … and then *generate* RDF automatically (e.g., via an XSLT script)

Formalizing the Scraper Approach: GRDDL

- GRDDL *formalizes* the scraper approach. For example:

```
<html xmlns="http://www.w3.org/1999/">
  <head profile="http://www.w3.org/2003/g/data-view">
    <title>Some Document</title>
    <link rel="transformation" href=
       href="http://…/dc-extract.xsl" />
    <meta name="DC.Subject" content="Some subject"/>
    ...
  </head>
  ...
</html>
```

- yields, by running the file through `dc-extract.xsl`

```
<rdf:Description rdf:about="">
  <dc:subject>Some subject</dc:subject>
</rdf:Description>
```

GRDDL (cont)

- The user has to provide `dc-extract.xsl`, with a definition of the corresponding meta-s, class id-s, etc
- A formalized approach towards the “microformats” world
- Still a W3C Team Submission, may get a formal status in 2006

Another Future Solution: XHTML2

- XHTML2 will have two metadata modules:
  - *extends the* ***link*** *and* ***meta*** *elements (e.g., meta elements may have children, thereby adding more complex data)*
  - *defines general attributes to add metadata to any elementss*

```
<span property="dc:date">March 23, 2004</span>
<span property="dc:title">Rollers hit casino for £1.3m</span>
  By <span property="dc:creator">Steve Bird</span> …
```

- may yield, by running the file through a processor

```
<rdf:Description rdf:about="">
  <dc:date>March 23, 2004</dc:date>
  <dc:title>Rollers hit casino for £1.3m</dc:title>
  <dc:creator>Steve Bird</dc:creator>
</rdf:Description>
```

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 (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*

Example for Another Serialization

- Turtle is another serialization of RDF (*not* based on XML)
  - *not a W3C recommendation*
  - *but used in some W3C documents, e.g., SPARQL (see later)*
- An example:

```
@prefix axsvg: <http://svg.example.org#>.
@prefix : <http://this.file.uri>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
:FullSlide
    axsvg:labelledBy :bottomLegend;
    axsvg:isA [
        axsvg:consistsOf :Axes, :Legend, :Datalines.
    ].
:bottomLegend axsvg:isAnchor "True"^^xsd:boolean.
```

RDF Data Access, a.k.a. Query (SPARQL)

Querying RDF Graphs/Depositories

- Remember the Python idiom:

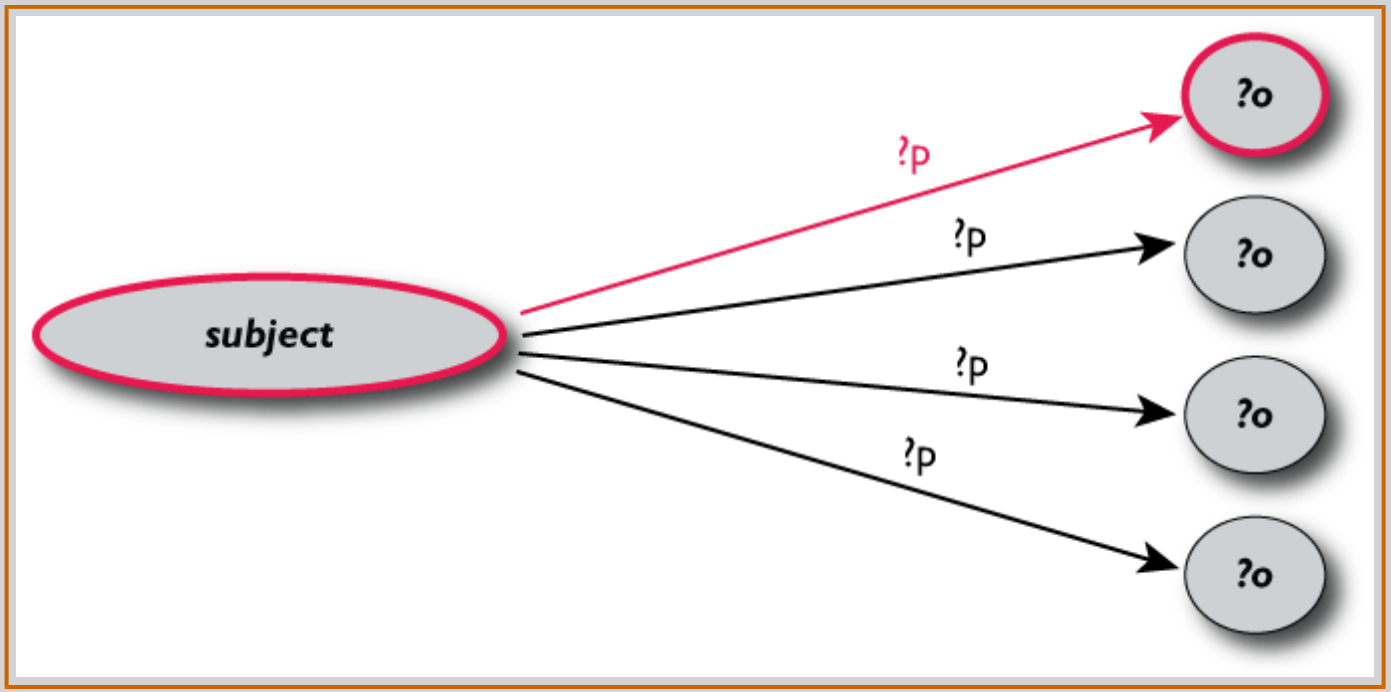
```
# do something with (p,o) pairs
for (p,o) in graph.predicate_objects(subject) :
        do_something(p,o)
```

- In practice, more complex queries into the RDF data are necessary
  - *something like: "give me the* `(a,b)` *pair of resources, for which there is an* `x` *such that* `(x parent a)` *and* `(b brother x)` *holds" (ie, return the uncles)*
  - *these rules may become quite complex*
- Queries become very important for *distributed* RDF data!
- This is the goal of SPARQL (Query Language for RDF)

Analyse the Python Example

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

- The **`(subject,p,o)`** is a *pattern* for what we are looking for
  - *with **p** and **o** as "unknowns"*



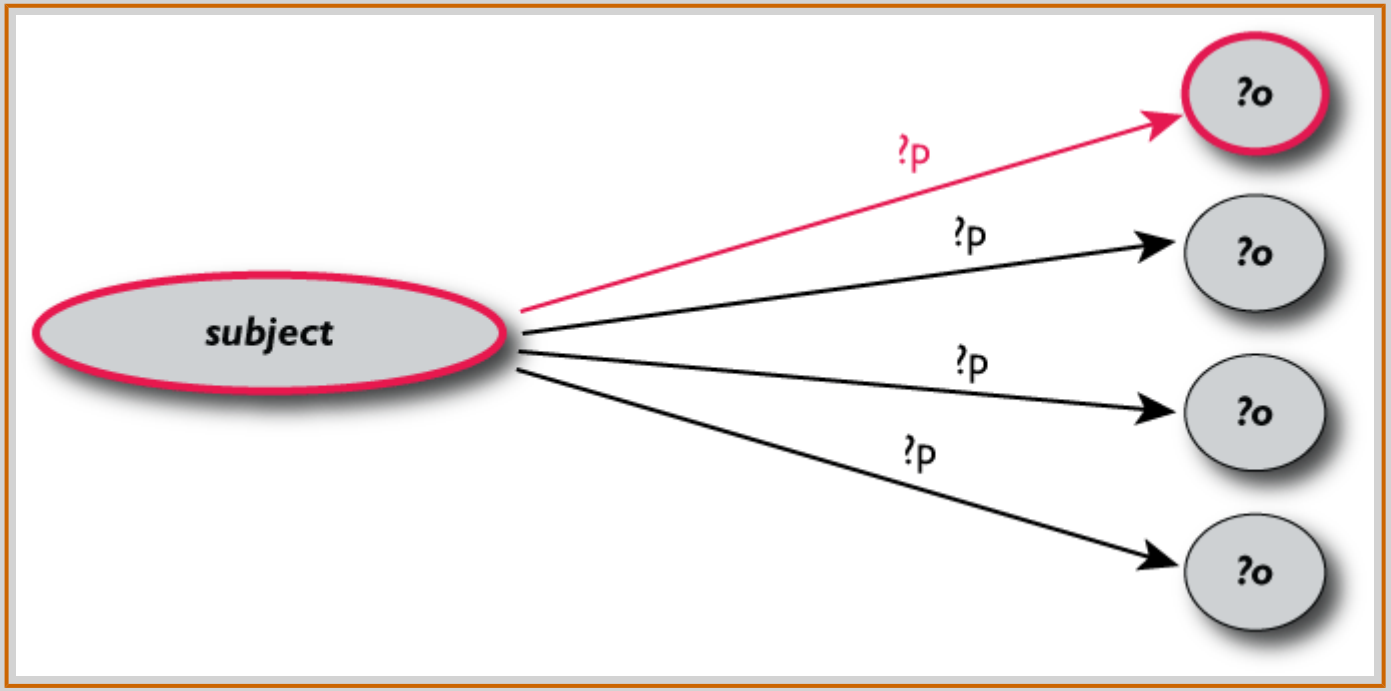
General: Graph Patterns

- The fundamental idea: generalize the approach of *graph patterns*:
  - *the pattern contains unbound symbols*
  - *by binding the symbols (if possible), subgraphs of the RDF graph are selected*
  - *if there is such a selection, the query returns the bound resources*
- SPARQL
  - *is based on similar systems that already existed in some environments*
  - *is a programming language-independent query language*
  - *is in a last call working draft phase (Recommendation in 2006?)*

Our Python Example in SPARQL

```
SELECT ?p ?o
WHERE {subject ?p ?o}
```

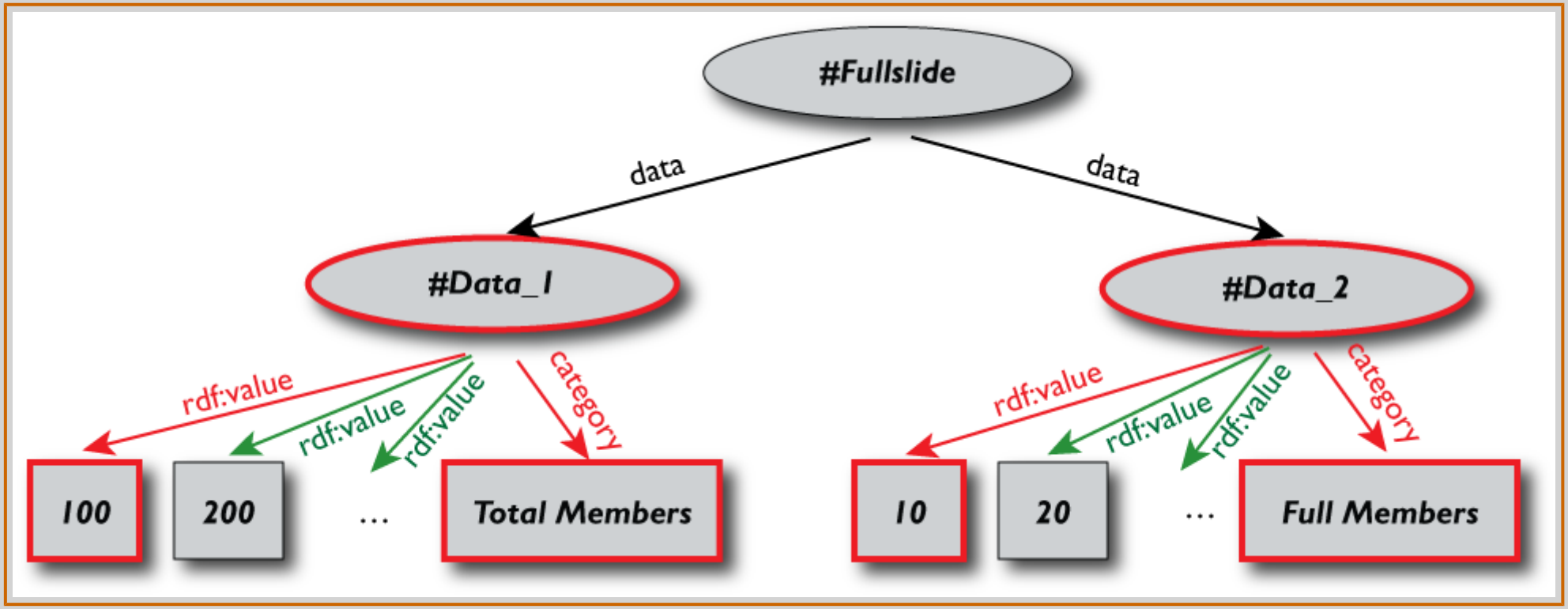
- The triplets in **WHERE** define the graph pattern, with **?p** and **?o** "unbound" symbols
- The query returns a list of matching **p,o** pairs



Simple SPARQL Example

```
SELECT ?cat ?val #note: not ?x!
WHERE { ?x rdf:value ?val. ?x category ?cat }
```

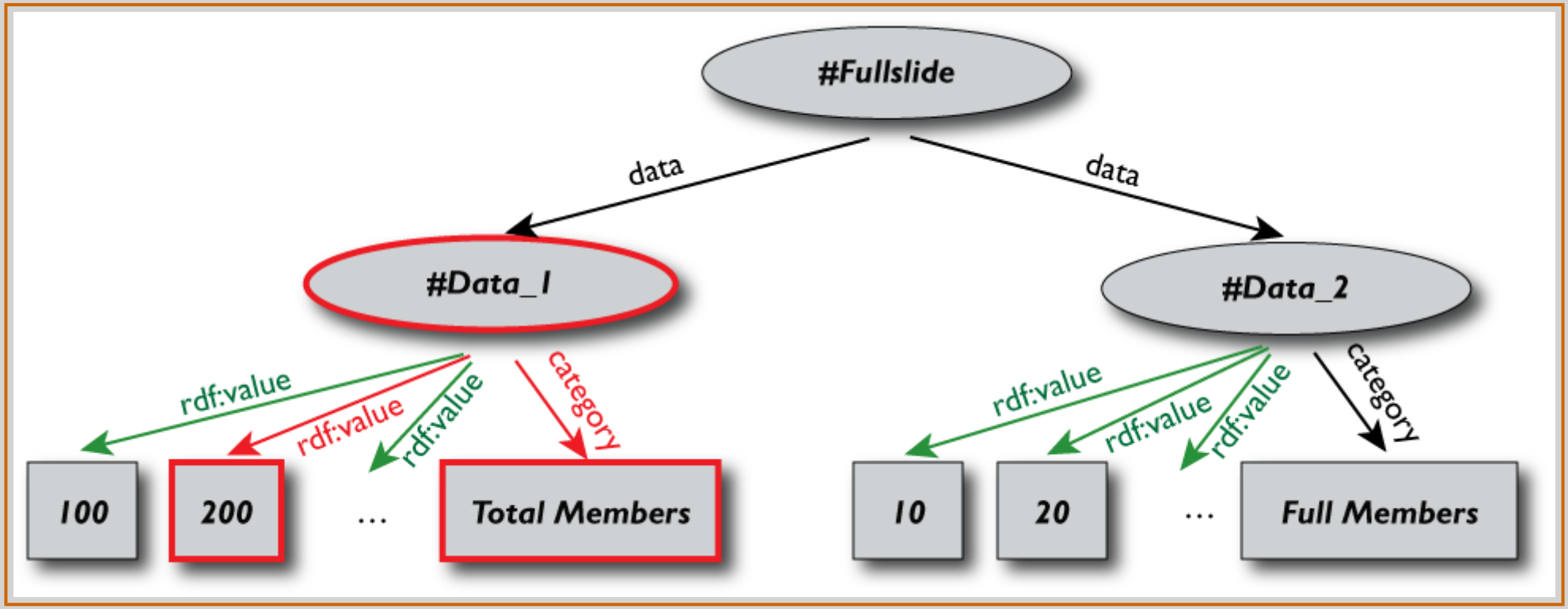
- Returns: `[["Total Members",100],["Total Members",200],…,["Full Members",10],…]`



Pattern Constraints

```
SELECT ?cat ?val
WHERE { ?x rdf:value ?val. ?x category ?cat. FILTER(?val>=200). }
```

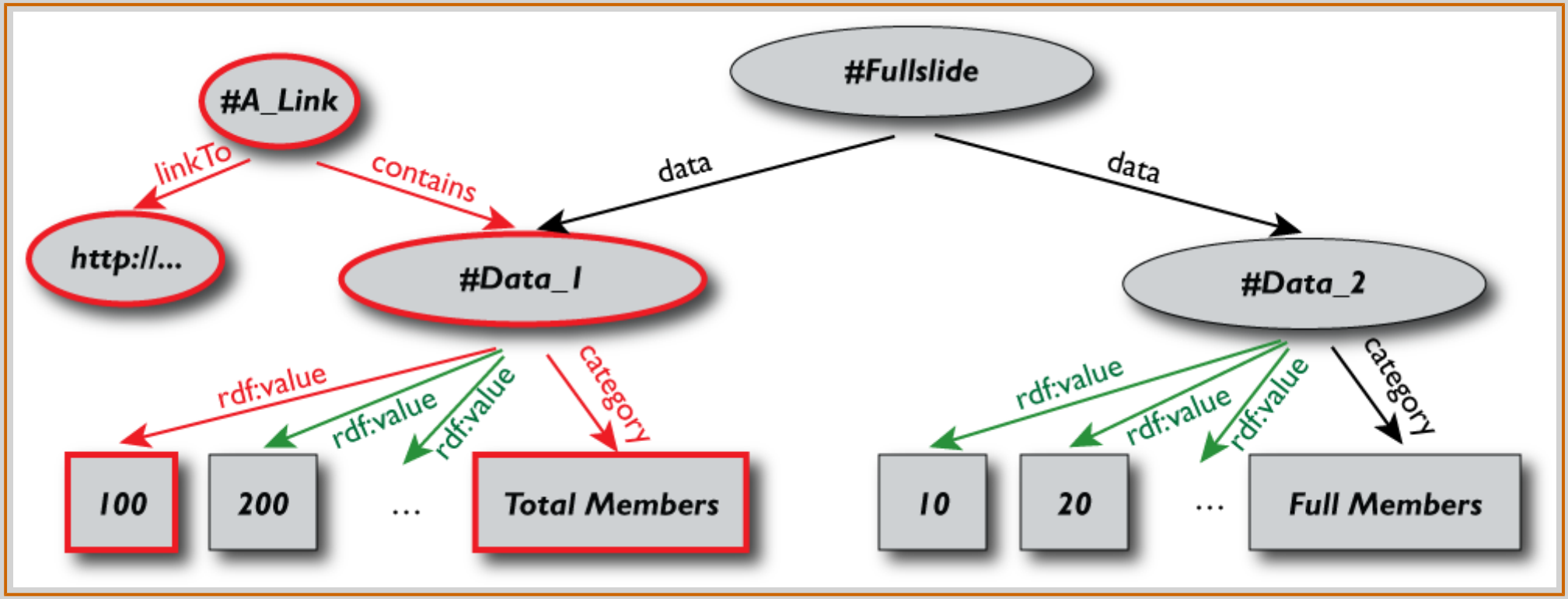
- Returns: `[["Total Members",200],…,]`
- SPARQL defines a base set of operators and functions



More Complex Example

```
SELECT ?cat ?val ?uri
WHERE { ?x rdf:value ?val. ?x category ?cat.
        ?al contains ?x.  ?al linkTo ?uri }
```

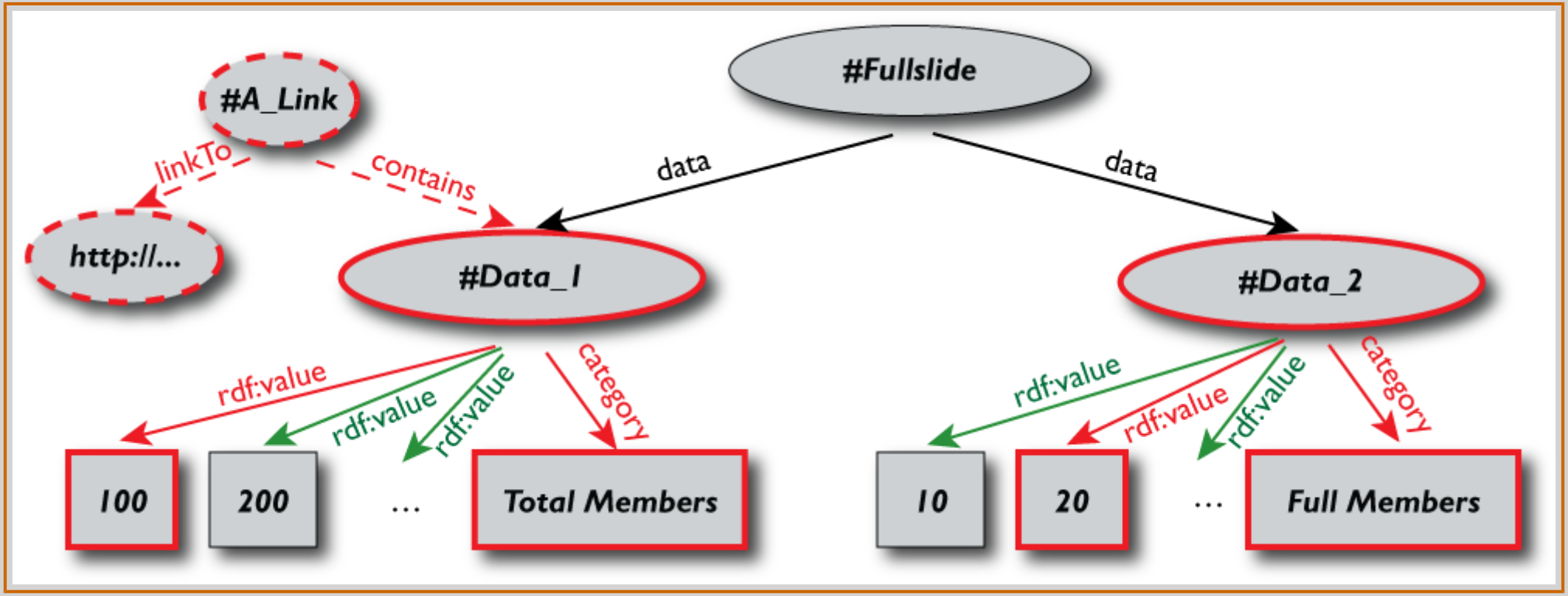
- Returns: `[["Total Members",100,Resource(http://...)],…,]`



Optional Pattern

```
SELECT ?cat ?val ?uri
WHERE    { ?x rdf:value ?val. ?x category ?cat.
           OPTIONAL ?al contains ?x. ?al linkTo ?uri }
```

- Returns: `["Total Members", 100, http://...],…,["Full Members",20, ],…,`



Other SPARQL Features

- Limit the number of returned results; remove duplicates, sort them, …
- Return the full *subgraph* (instead of a list of bound variables)
- Construct a graph combining a *separate* pattern and the query results
- Use datatypes and/or language tags when matching a pattern
- SPARQL is in last call Working Draft, i.e., the technical aspects are now fixed (modulo implementation problems)
  - *recommendation expected 3Q of 2006*
  - *there are a number of implementations already*

SPARQL Usage in Practice

- *Locally*, i.e., bound to a programming environments like RDFLib or Jena
- *Remotely*, e.g., over the network or into a database
  - *separate documents define the protocol and the result format*
    - SPARQL Protocol for RDF
    - SPARQL Results XML Format
  - *return is in XML: can be fed, e.g., into XSLT for direct display*
- An application pattern evolves: use (XHTML) forms to create a SPARQL Query to a database and display the result in XHTML
  - *there are a number of application experiments, demos, etc., already; see, e.g., the W3C's Talk database*

Remote Query Example

```
GET /qps?query-lang=http…&graph-id=http://my.example/3.rdf
    &query=SELECT+:…+WHERE:+…:HTTP/1.1
User-Agent: my-sparql-client/0.0
Host: my.example

200 OK HTTP/1.1
Server: my-sparql-server/0.0
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#>
  <head>
    <variable name="a/>
    ...
  </head>
  <results>
    <result ordered="false" distinct="false">
      <binding name="a"><uri>http:…</uri></binding>
      ...
    </result>
    <result> ... </result>
  </results>
</sparql>
```

Programming Practice

Programming Practice

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

```
# import the libraries
from rdflib.Graph import Graph
from rdflib.URIRef import URIRef
# resource for a specific URI:
subject = URIRef("URI_of_Subject")
# create the graph
graph = Graph()
# parse an RDF file and store it in the triple store
graph.parse("membership.rdf")
# do something with (p,o) pairs
for (p,o) in graph.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
graph.add((subject,pred,object))
# remove it
graph.remove_triples((subject,pred,object))
# save it in a file in RDF/XML
graph.serialize("filename.rdf")
```

Programming Practice (cont.)

- Starting with version 2.2.2, there is also a SPARQL API (in RDFLib:

```
from rdflib.sparql import SPARQLGraph, GraphPattern
p = GraphPattern([("?x",RDF.type,"?val"),("?x",category,"?cat")])
select = ("?cat","?val")
sparqlGraph = SPARQLGraph(graph)
results = sparqlGraph.query(select,p)
```

- (there is no parser yet, hence the usage of the API)
- Does not have (yet) schema processing (no “inferred” properties, for example)
- RDFLib is a typical example for the RDF Programming environments around; it is very easy to start with it!

Another example: Jena

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

```
 // create a model (a.k.a. Graph 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);
}
```

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
  - *an "RDFS Reasoner"*
  - *a full SPARQL implementation*
  - *a layer (Joseki) for remote access of triples (essentially, and triple database)*
  - *and more…*
- Probably the most widely used RDF environment in Java today

Lots of Other tools

- There are *lots* of other tools:
  - *RDF frameworks for specific languages: RDFStore (Perl), RAP (PHP), SWI-Prolog (Prolog),…*
  - *Redland: general RDF Framework, with bindings to C, C++, C#, Python, …, and with SPARQL facilities (Rasqal)*
  - *RDF storage systems: (Sesame, Kowari, Tucana, Gateway, @Semantics RDFStore, 3Store, Jena's Joseki, InferEd, Oracle Database 10g, …)*
    - some of these are based on an internal sql engine (3Store, Oracle), others are made bottom up as triple stores
    - most of them have or plan for SPARQL facilities
- See the tool list at W3C or the Free University of Berlin list

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»?"
    - programs should be able to *deduce* such statements
  - *if somebody else defines a set of terms: are they the same?*
  - *construct classes, not just name them*
  - *restrict a property range when used for a specific class*
  - *disjointness or equivalence of classes*
  - *etc.*

Ontologies (cont.)

- There is a need to support *ontologies* on the Semantic Web:

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

- We need a *Web Ontologies Language* to define:
  - *more on the terminology used in a specific context*
  - *more constraints on properties, logical characterisation of properties*
  - *etc.*
- Language should be a compromise between
  - *rich semantics for meaningful applications*
  - *feasibility, implementability*

W3C's Ontology Language (OWL)

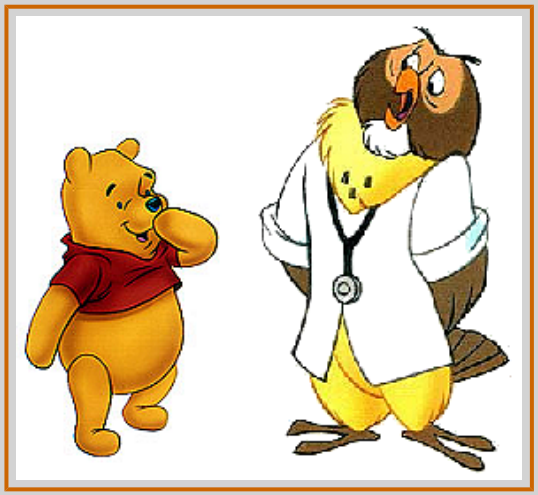
- A layer *on top* of RDFS with additional possibilities
- Outcome of various projects:
  1. *SHOE project: an early attempt to add semantics to HTML*
  2. *DAML-ONT (a DARPA project) and OIL (an EU project)*
  3. *an attempt to merge the two: DAML+OIL*
  4. *the latter was submitted to W3C*
  5. *lots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

It was a long road…

- Lots of requirements, influences on OWL
  - *research results on knowledge representation, model theory*
  - *the RDF/RDFS view of the world (triplets, syntactical issues, …), with OWL as a layer on top of RDF/RDFS*
  - *needs of the Web in general*
  - *balance between expressability and implementability*
- You can read about it in a paper of Horrocks, Patel-Schneider, and van Harmelen…
- The result: “OWL is now the most used KR language in the history of AI…” (Jim Hendler)

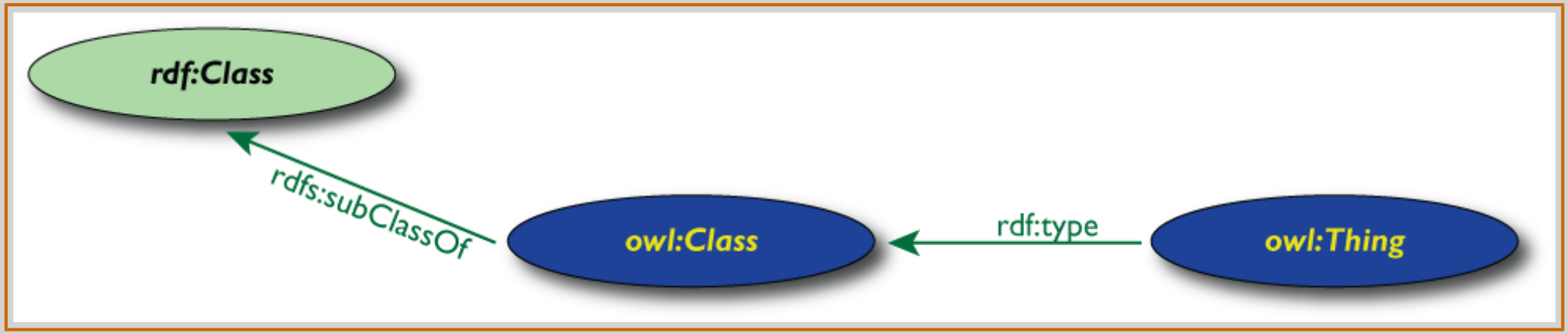
Why “OWL” and not “WOL”?

- Some urban legends…
  - *e.g., reference to Owl from Winie the Pooh, who misspelled his name as “WOL”*
- A reference to an AI project at MIT of the mid 70’s by Bill Martin, called “One World Language”…
  - *an early attempt for a KR language and associated ontology, intended to be a universal language for encoding meaning for computers*
- “Why not be inconsistent in at least one aspect of a language which is all about consistency” (Guus Schreiber)



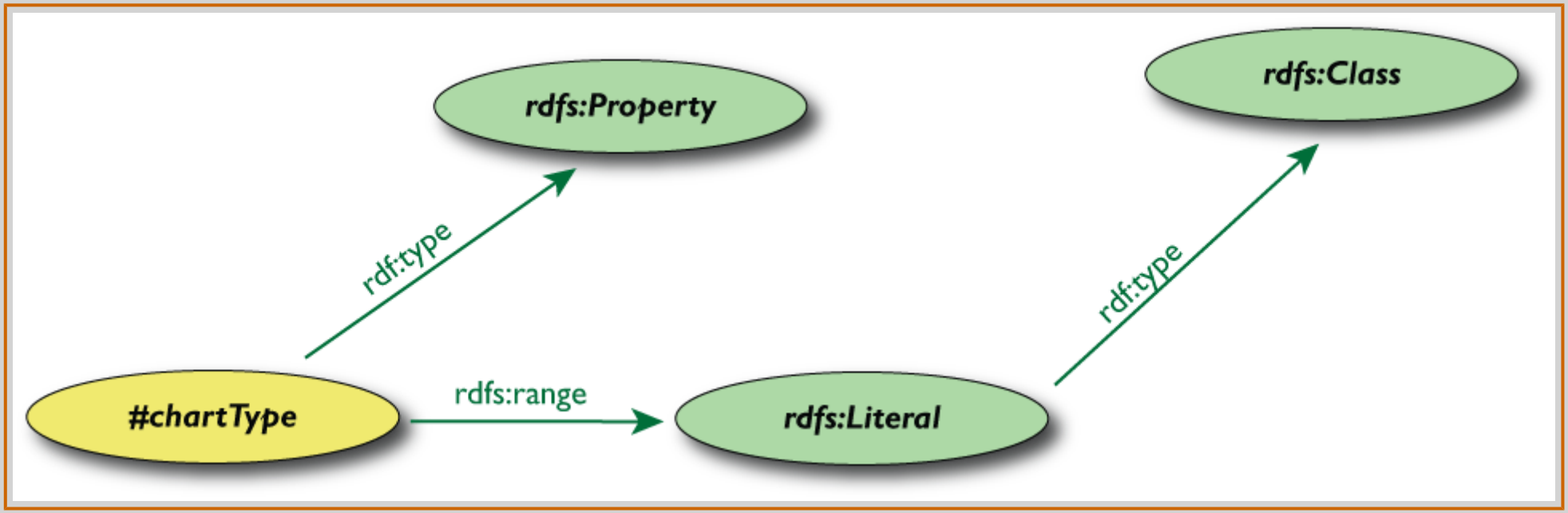
Classes in OWL

- In RDFS, you can subclass existing classes… that's all
- 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 *classes* from *individuals*



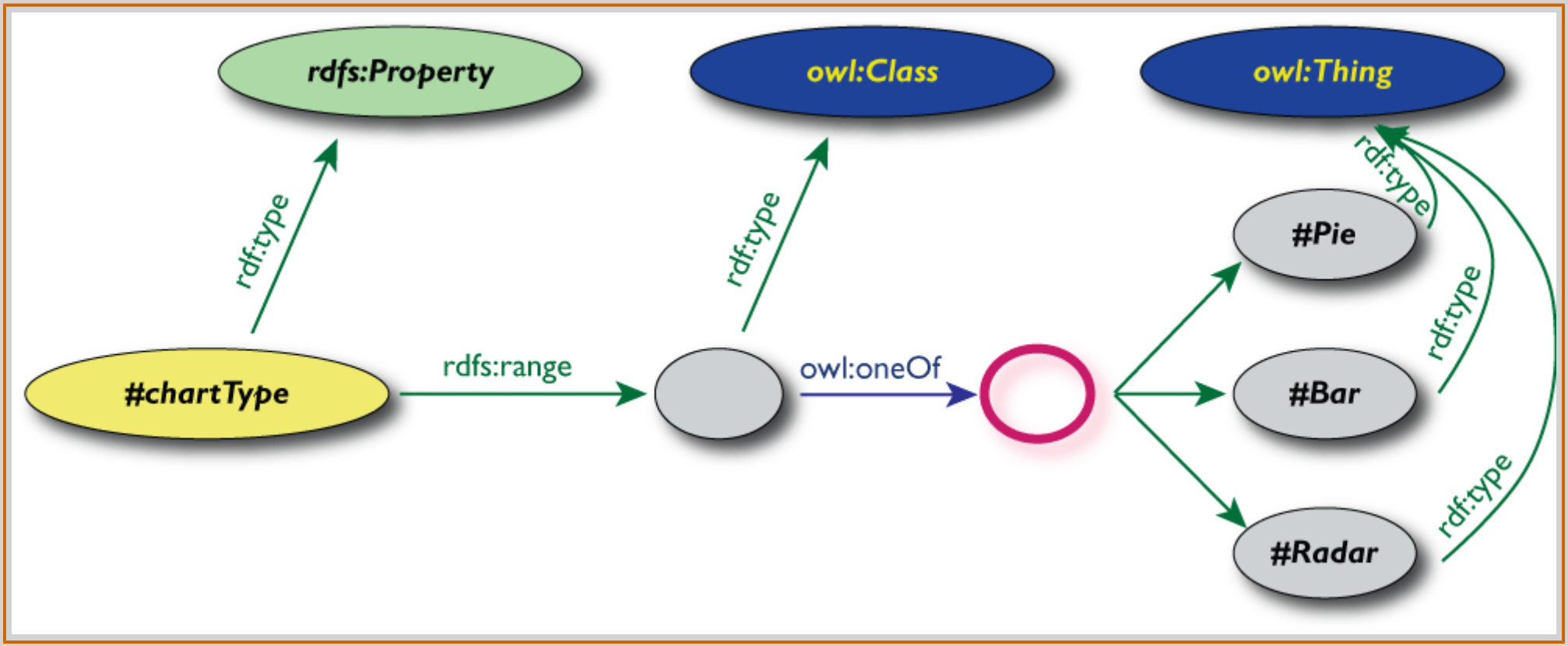
Need for Enumeration

- Remember this issue?
  - *one can use XML Schema types to define a* `ChartType` *enumeration…*
  - *…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:



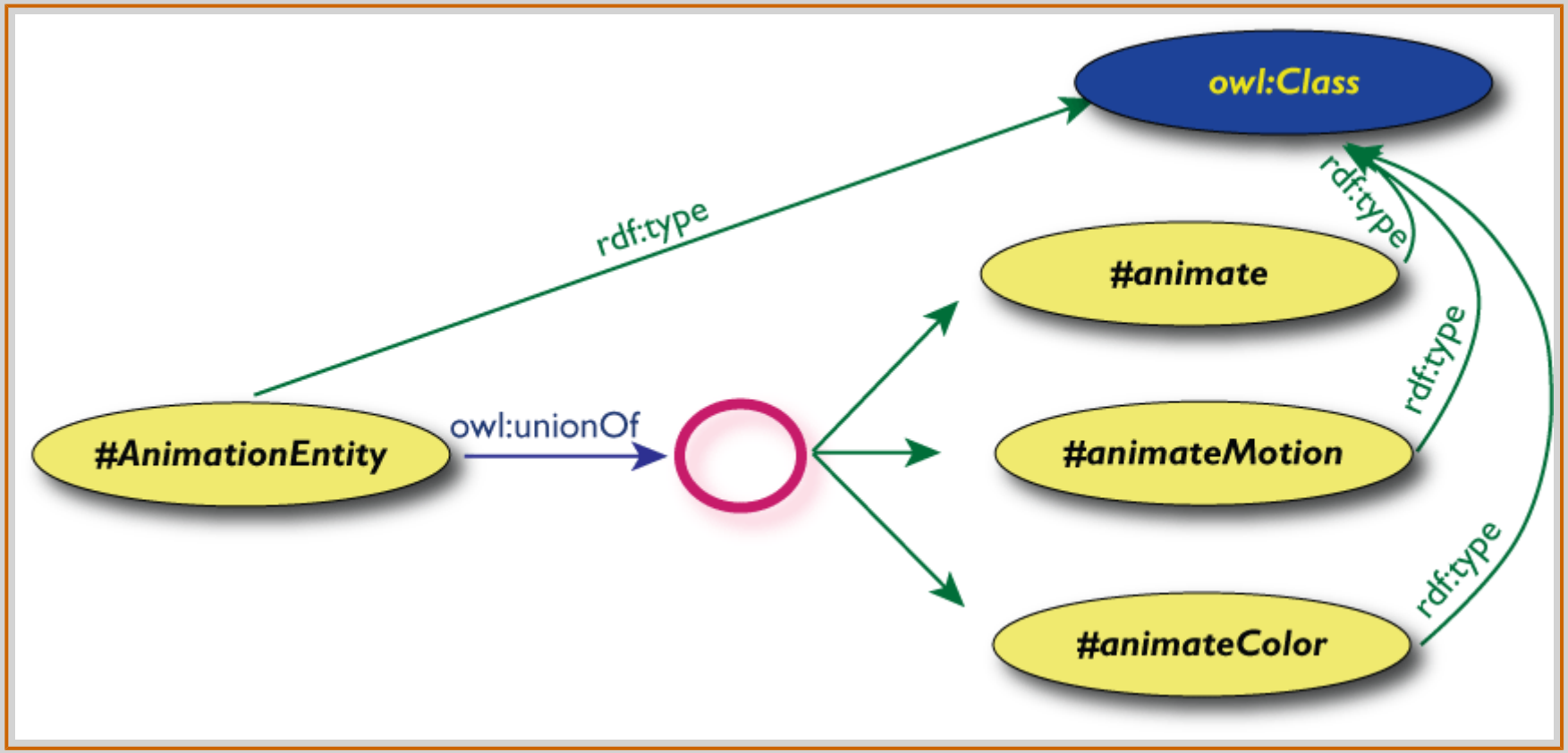
Same in RDF/XML

```
<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>
```

- The class consists of *exactly* of those individuals

Union of Classes

- Essentially, like a set-theoretical union:



Same in RDF/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

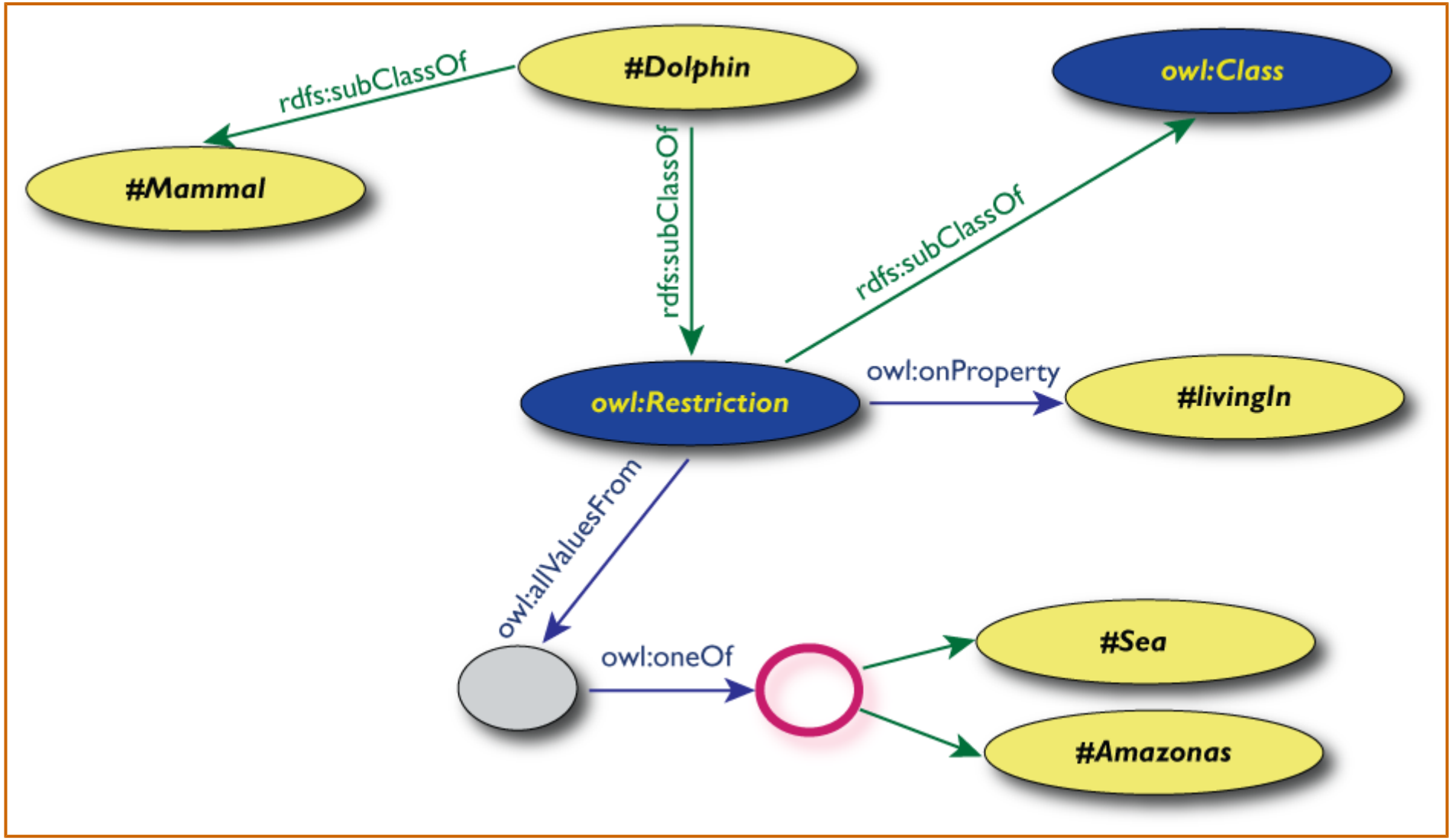
- (Sub)classes created by restricting the property value *on that class*
- For example, “a dolphin is a mammal living in water” means:
  - *restrict the value of “living in” when applied to “mammal”…*
  - *…thereby define the class of “dolphins”*

Property Restrictions in OWL

- 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 Restriction Example

- “A dolphin is a mammal living in the sea or in the Amazonas”:


rdfs:subClassOf
#Dolphin
owl:Class
#Mammal
rdfs:subClassOf
rdfs:subClassOf
owl:Restriction
owl:onProperty
#livingIn
owl:allValuesFrom
owl:oneOf
#Sea
#Amazonas

Restrictions 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

Same in RDF/XML

```
<owl:Class rdf:ID="Delphin">
    <rdfs:subClassOf rdf:resorce="#Mammal"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#livingIn"/>
        <owl:allValuesFrom rdf:resource="#UnionOfSeaAnAmazonas">
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

`allValuesFrom` could be replaced by:

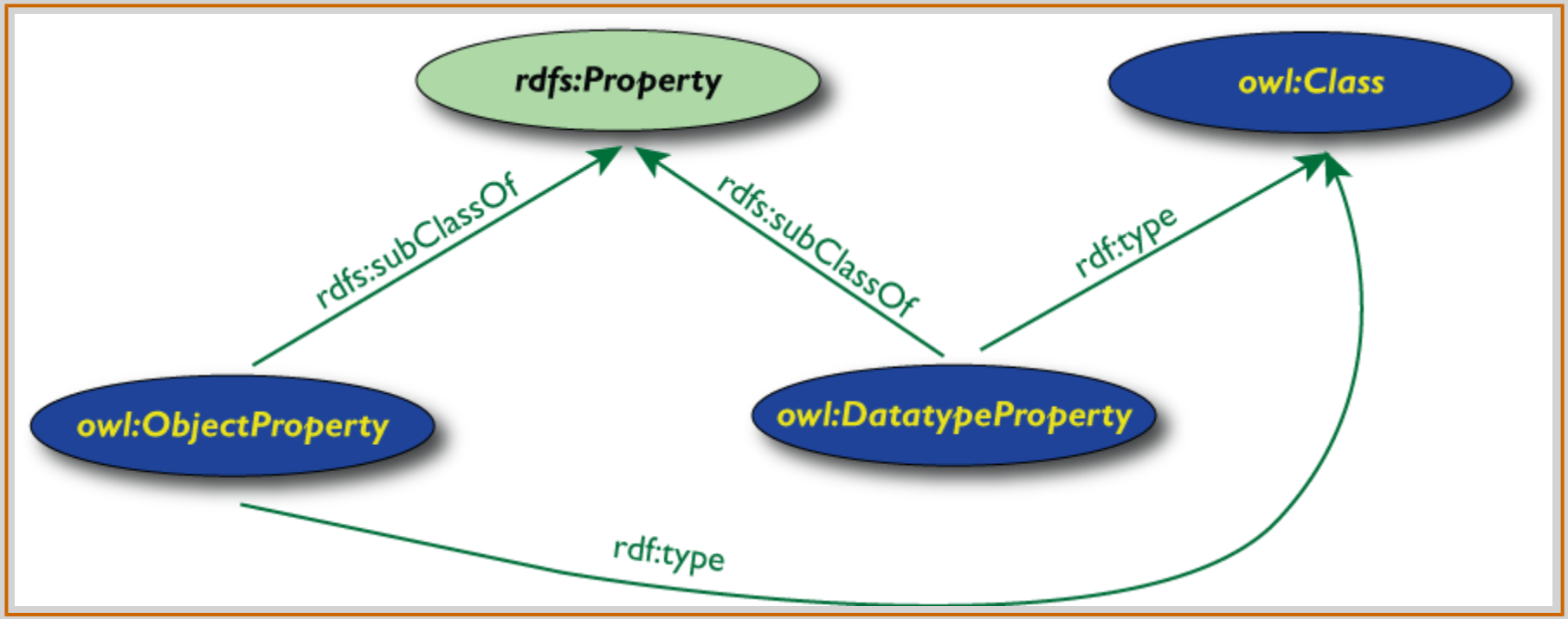
- `someValuesFrom`
- `cardinality`, `minCardinality`, `maxCardinality`

Cardinality Constraint in RDF/XML

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

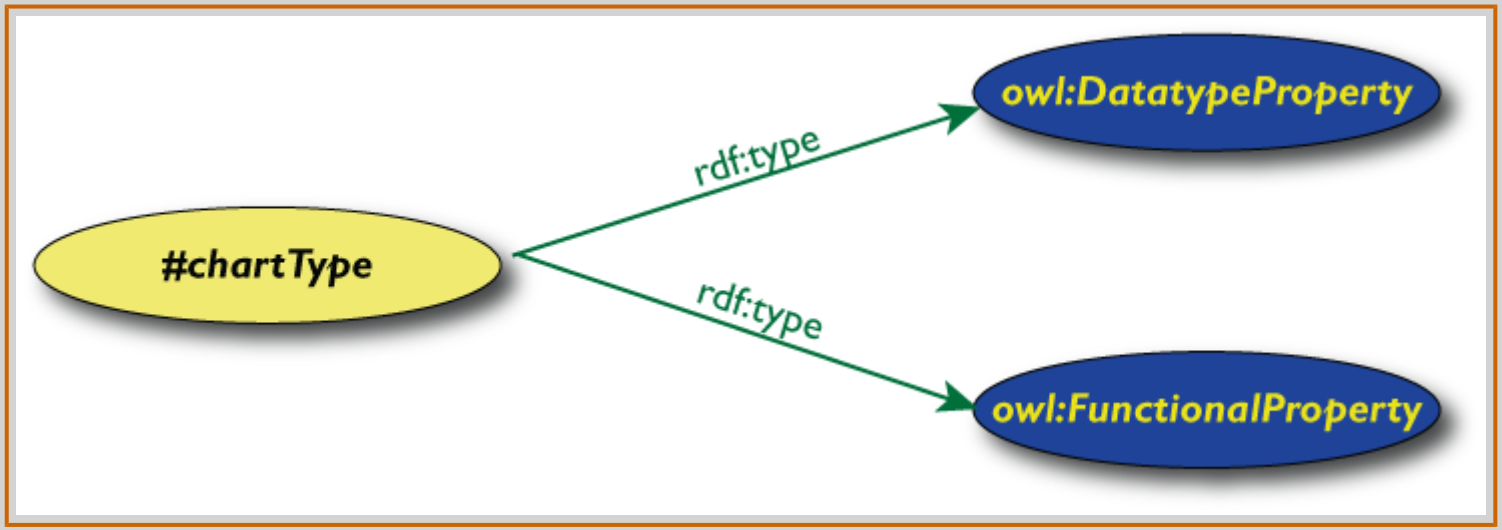
Property Characterization

- In OWL, one can characterize the *behavior* of properties (symmetric, transitive, …)
- OWL also separates data properties
  - *"datatype property" means that its range are typed literals*



Characterization Example

- “There should be only one chart type”



Same in RDF/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!

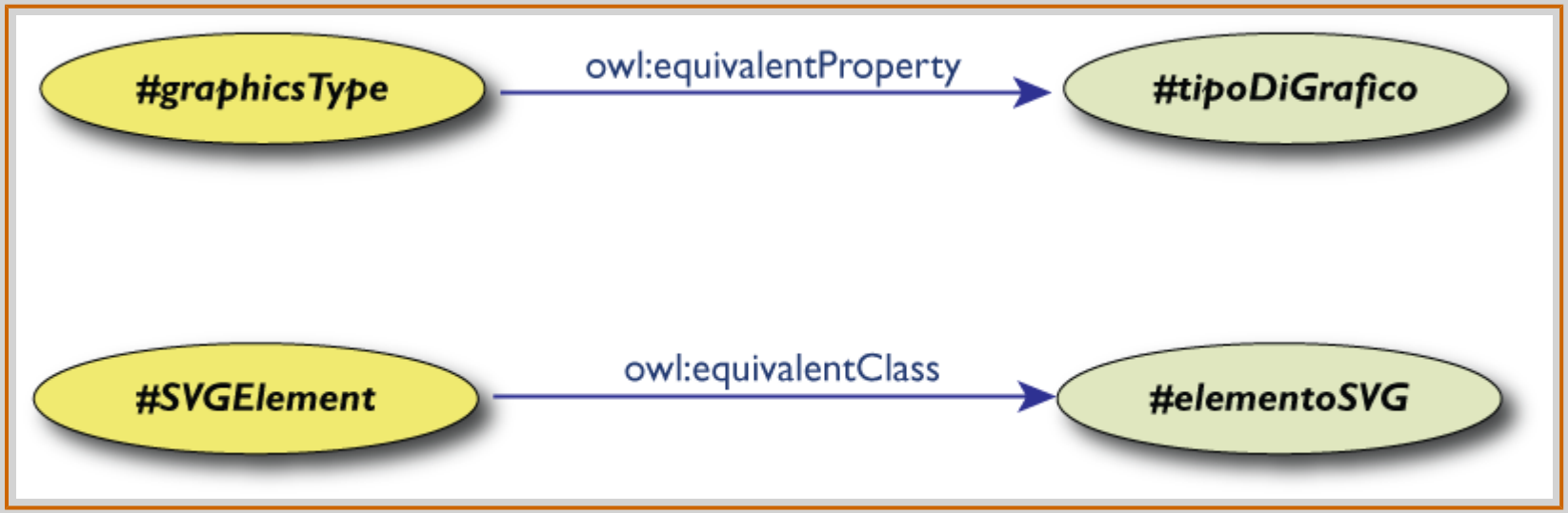
OWL: Additional Requirements

- 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 Italian



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*

OWL and Logic

- 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*
  - *OWL can be mapped to FOL to describe "traditional" ontology concepts … but it is not a general logic system pe se!*
- Inference based on OWL is *within this framework 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
  - *decreasing 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 (e.g., declare* `rdf:type` *to be functional…)*
  - *etc.*
- A real superset of RDFS
- But: *an OWL Full ontology may be undecidable!*

Example for a Possible Problem (in OWL Full)

```
<owl:Class rdf:ID="A">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource=".../22-rdf-syntax-ns#type"/>
      <owl:allValueFrom rdf:about="#B"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
<owl:class rdf:ID="B">
  <owl:complementOf rdf:parseType="Collection">
    <owl:Class rdf:about="#A"/>
  </owl:complementOf>
</owl:class>
<owl:Thing rdf:ID="C">
  <rdf:type rdf:resource="#A"/>
</owl:Thing>
```

- if **C** is of type **A** then it *must* be of the complement type, i.e., *not* of **A**…

OWL Description Logic (DL)

> *Goal: maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure is realizable*

- **`Class`**, **`Thing`**, **`ObjectProperty`**, **`DatatypePropery`** are *strictly separated* : a class *cannot* be an individual of another class
  - *object properties' values must usually be an* ***`owl:Thing`*** *(except, e.g., for* ***`rdf:type`****)*
- No mixture of **`owl:Class`** and **`rdfs:Class`** in definitions (essentially: use OWL concepts only!)
- No statements on RDFS resources (e.g., **`rdf:type`**)
- No characterization of datatype properties possible
- …

OWL Lite

- *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 or property constraints*
  - *cardinality restriction with 0 and 1 only*
  - *…*

Note on OWL layers

- OWL Layers were defined to reflect compromises:
  - *expressability vs. implementability*
- Some application just need to express and interchange terms (with possible scruffiness): OWL Full is fine
  - *they may build application specific reasoning instead of using a general one*
- Some applications need rigor; then OWL DL/Lite might be the good choice
- Research may lead to new decidable subsets of OWL!
  - *see, e.g., H.J. ter Horst's paper at ISWC2004 or in the Journal of Web Semantics (October 2005)*

“Description Logic”

- The term refers to an area in knowledge representation
  - *a special type of “structured” First Order Logic (logic with safety guards…)*
  - *formalism based on “concepts” (i.e., classes), “roles” (i.e., properties), and “individuals”*
  - *based on model theoretic semantics (like RDF, RDFS, and OWL!)*
- There are several variants of Description Logic
  - *i.e., OWL DL and Lite are embodiments of distinct Description Logics*
    - for connaisseurs: OWL DL ≈ $\mathcal{SHOIN}$ (**D**), OWL Lite ≈ $\mathcal{SHIF}$ (**D**)
  - *some major differences: usage of URI-s, reference to XML Schema datatypes, version control, built-in annotation…*

“Description Logic” (cont.)

- Traditional DL has its own terminology:
  - *named objects or concepts ⇔ definition of classes, relationships among classes, …*
  - *roles ⇔ properties*
  - *(termilogical) axioms ⇔ subclass and subproperty relationships, …*
  - *facts or assertions ⇔ statements on individuals (`owl:Thing`-s)*
- There is also a compact mathematical notation for axioms, assertions, etc:
  - $AnimationEntity \equiv Animate \sqcup AnimateMotion$
  - $Delphin \sqsubseteq Mammal \sqcap \forall living.Water$
- You may see these in papers, books…

OWL-DL “Abstract Syntax”

- There is also a non-XML based notation for OWL DL (and OWL Lite) defined by W3C (also used in the formal specification of OWL DL)
  - *currently only RDF/XML format is widely implemented, but AS → RDF/XML converters exist (i.e., it may become more widespread in future)*
  - *it makes writing ontologies with DL restrictions easier…*

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

Ontology Developement

- 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)*
  - *OWL is just a tool to formalize ontologies*
- Large scale ontologies are often developed in a community process
- Ontologies should be *shared* and *reused*
  - *can be via the simple namespace mechanisms…*
  - *…or via explicit inclusions*
- Applications can also be developed with very small ontologies, though! ("a small ontology can take you far…")

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*

Simple Knowledge Organisation System (SKOS)

Simple Knowledge Organisation System

- Goal: porting ("Webifying") thesauri: representing and sharing classifications, glossaries, thesauri, etc, as developed in the "Print World". For example:
  - *Dewey Decimal Classification, Art and Architecture Thesaurus, ACM classification of keywords and terms…*
  - *DMOZ categories (a.k.a. Open Directory Project)*
- The system must be simple to allow for a quick port of traditional data (done by "traditional" people…)
- *This is where SKOS comes in*

Example: Entries in a Glossary (1)

“Assertion”

“(i) Any expression which is claimed to be true. (ii) The act of claiming something to be true.”

“Class”

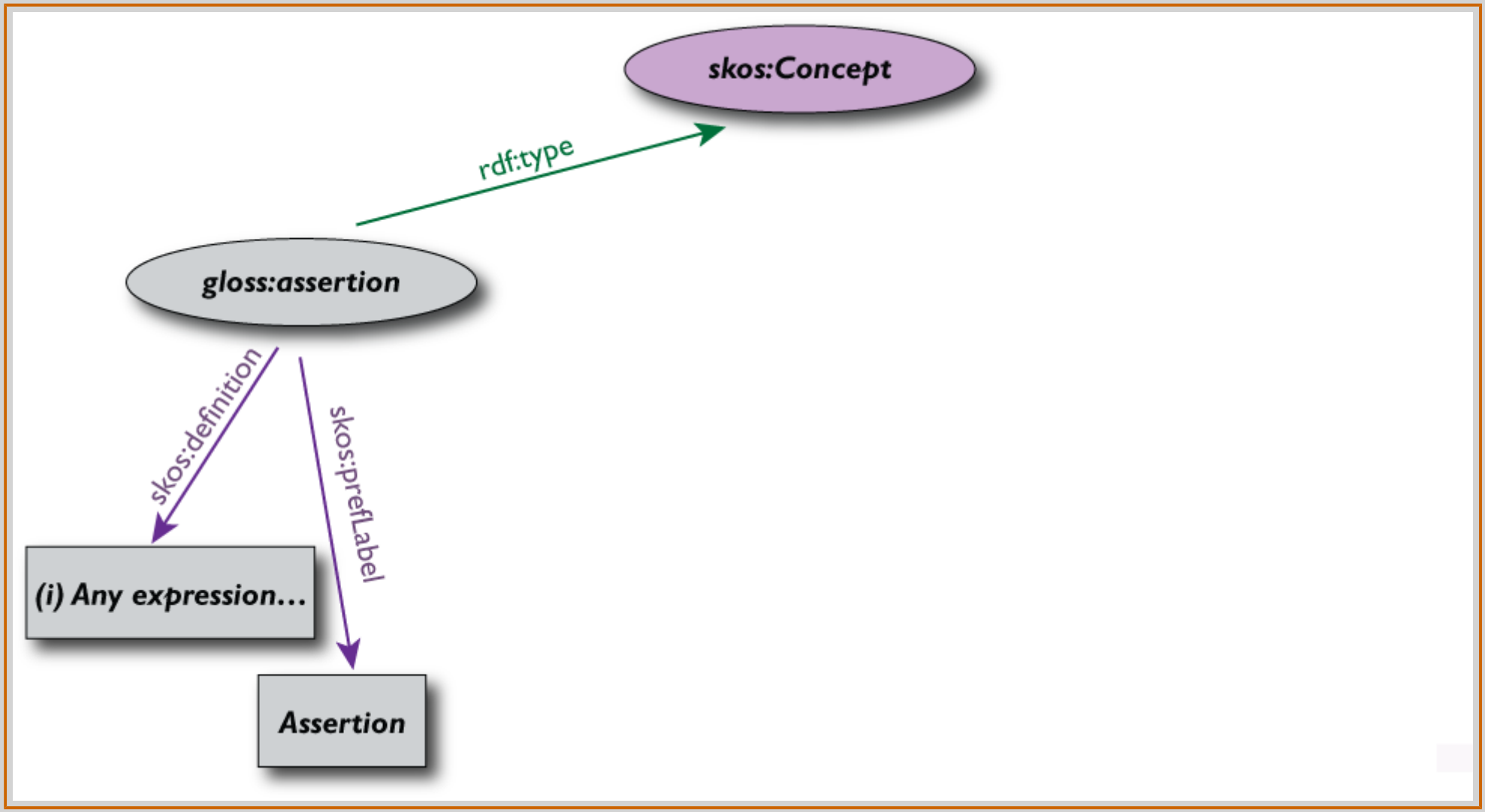
“A general concept, category or classification. Something used primarily to classify or categorize other things.”

“Resource”

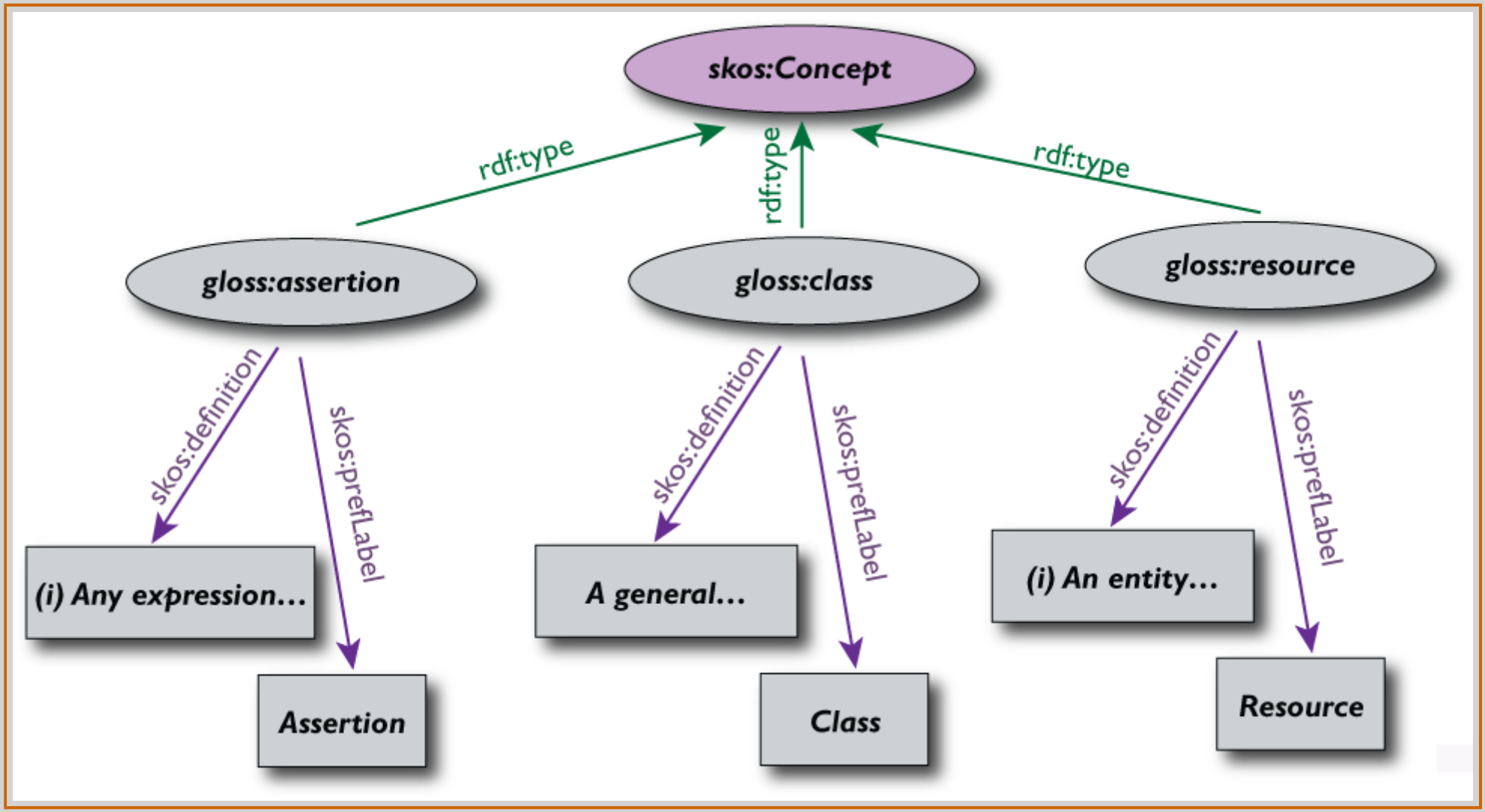
“(i) An entity; anything in the universe. (ii) As a class name: the class of everything; the most inclusive category possible.”

(from the RDF Semantics Glossary)

Example: Entries in a Glossary (2)



Example: Entries in a Glossary (3)



Example: Taxonomy (1)

Illustrates “broader” and “narrower”

General

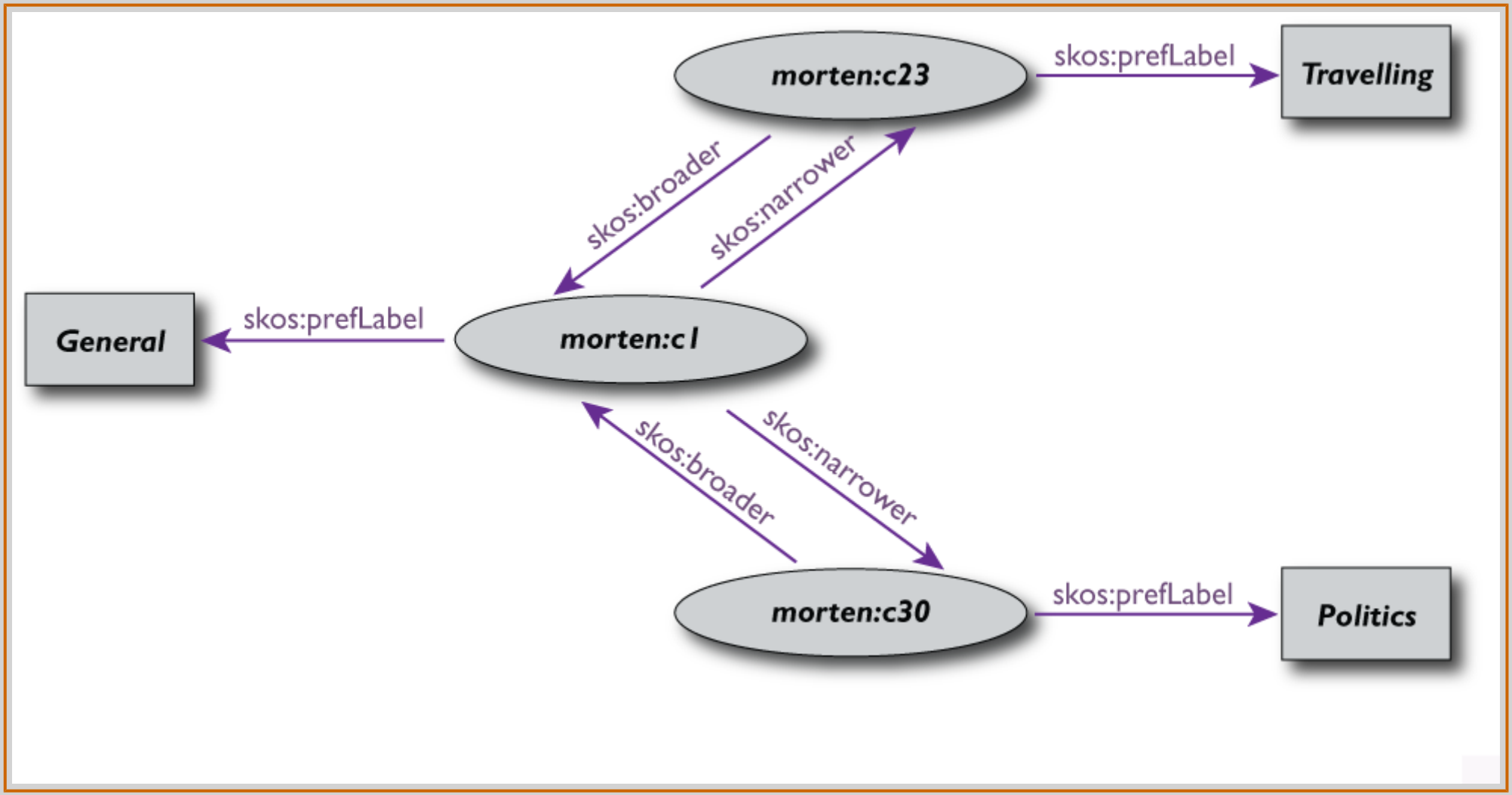
- Travelling
- Politics

SemWeb

- RDF
  - *OWL*

(From MortenF’s weblog categories. Note that the categorization is arbitrary!)

Example: Taxonomy (2)



Example: Thesaurus (1)

Term
Economic cooperation

Used For
Economic co-operation

Broader terms
Economic policy

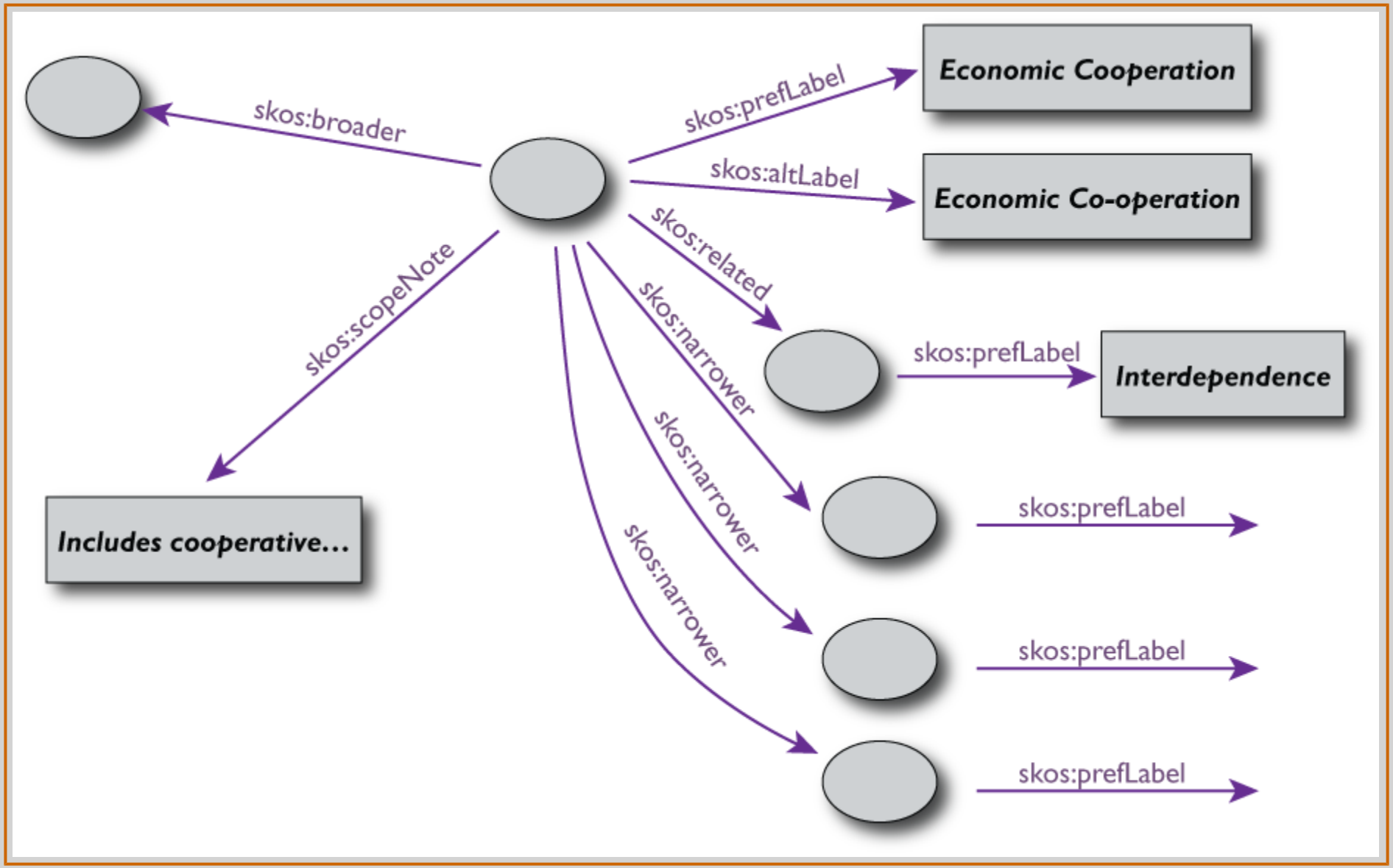
Narrower terms
Economic integration, European economic cooperation, …

Related terms
Interdependence

Scope Note
Includes cooperative measures in banking, trade, …

(from UK Archival Thesaurus)

Example: Thesaurus (2)



SKOS Core Overview

- Classes and Predicates:
  - *Basic description (**`Concept`**, **`ConceptScheme`**, **…**)*
  - *Labelling (**`prefLabel`**, **`altLabel`**, **`prefSymbol`**, **`altSymbol`** …)*
  - *Documentation (**`definition`**, **`scopeNote`**, **`changeNote`**, **…**)*
  - *Semantic relations (**`broader`**, **`narrower`**, **`related`**)*
  - *Subject indexing (**`subject`**, **`isSubjectOf`**, **…**)*
  - *Grouping (**`Collection`**, **`OrderedCollection`**, **…**)*
  - *Subject Indicator (**`subjectIndicator`**)*
- Some inference rules (a bit like the RDFS inference rules) to define some semantics

Why Having SKOS *and* OWL?

- OWL's precision not always necessary or even appropriate
  - *"OWL a sledge hammer / SKOS a nutcracker", or "OWL a Harley / SKOS a bike"*
  - *complement each other, can be used in combination to optimize cost/benefit*
- Role of SKOS is
  - *to bring the worlds of library classification and Web technology together*
  - *to be simple and undemanding enough in terms of cost and required expertise*

SKOS Documents

- SKOS documents should be finalized in 2006:
  - *"Quick Guide to Publishing a Thesaurus on the Semantic Web"* and *"SKOS Core Guide"*
  - *"SKOS Core Vocabulary Specification"*
  - *"SKOS Mapping Vocabulary Specification"*
- It is not yet decided whether SKOS remains a "W3C Note" or a Recommendation

“Core” Vocabularies

- A number of public “core” vocabularies evolve to be used by applications, e.g.:
  - *SKOS Core: about knowledge systems*
  - *Dublin Core: about information resources, digital libraries, with extensions for rights, permissions, digital right management*
  - *FOAF: about people and their organizations*
  - *DOAP: on the descriptions of (mainly open source) software projects*
  - *MusicBrainz: on the description of CDs, music tracks, …*
  - *…*
- They share the underlying RDF model (provides mechanisms for extensibillity, sharing, …)

What is Coming?

Semantic Web Activity “Phase 2”

- First phase (completed): core infrastructure
- Second phase: promotion and implementation needs
  - *outreach to user communities (life sciences, geospatial information systems, libraries and digital repositories, …)*
    - a separate Interest Group on Health Care and Life Sciences (HCLS) Interest Group has just started!
  - *intersection of SW with other technologies (Semantic Web Services, privacy, …)*
  - *further technical development (e.g., SPARQL, SKOS, Rules)*
- Separate Working Group on “Deployment and Best Practices”

Rules

- OWL can be used for simple inferences
- Applications may want to express domain-specific knowledge, like “Horn clauses”:
  - *(prem-1 ∧ prem-2 ∧ …) ⇒ concl*
  - *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»”*
- There is also a large corpus of rule based systems and languages, though not necessarily bound to the Web (yet)
- Several attempts already to combine Semantic Web with Rules (Metalog, RuleML, SWRL, WRL, cwm, …)
  - *note: cwm, for example, defines Horn predicates in terms of graph patterns, a connection to SPARQL…*

W3C's Rules Workshop

- W3C held a Workshop in April 2005; lots of issues and user cases were identified
- Some interesting scenarios have already been identified:
  - *exchange mail filtering rules among clients*
  - *rules to search into data (databases, RDF stores)*
  - *rules to analyse medical, banking, or other data data (merging rules of different origins)*
- interest from financial services, business rules, life science community,…

Rules Interchange Format Working Group

- The W3C Working Group started at the beginning of November 2005
- Work is planned in two “phases”:
  1. *construct an extensible format for rule interchange*
  2. *define more complex extensions*

RIF Phase 1 Goals

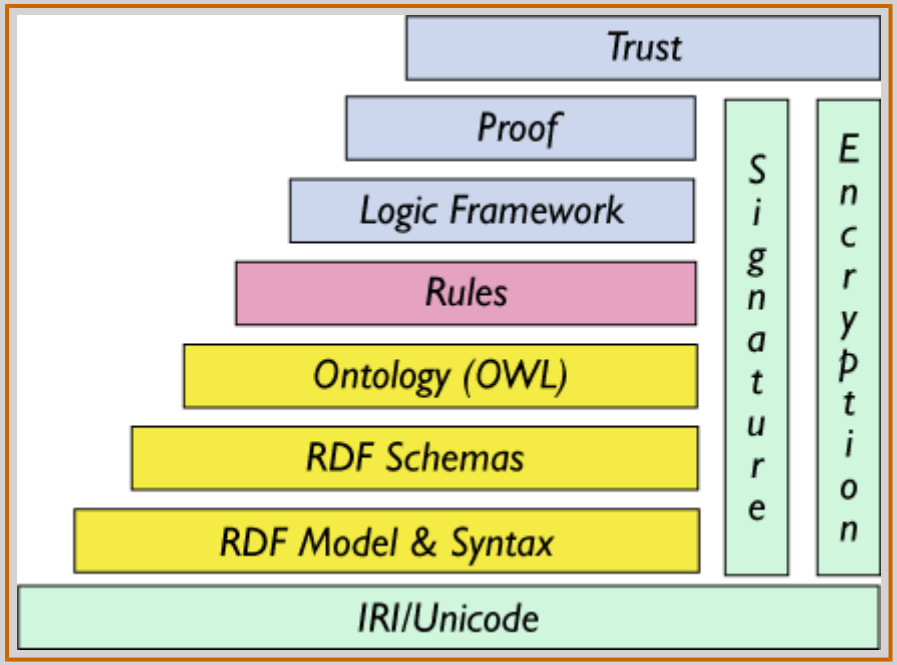
- An *interchange format* to exchange rules among rule engines and systems
  - *probably based on "full Horn Logic" with some simple datatypes (int, boolean, strings, …)*
  - *make it relatively simple, leave the more complex issues to Phase 2*
  - *make a new type of data accessible for the Web…*
- An *extensible format* to allow more complex alternatives to be defined
  - *e.g., fuzzy and/or temporal logic*
- Recommendation in May 2007

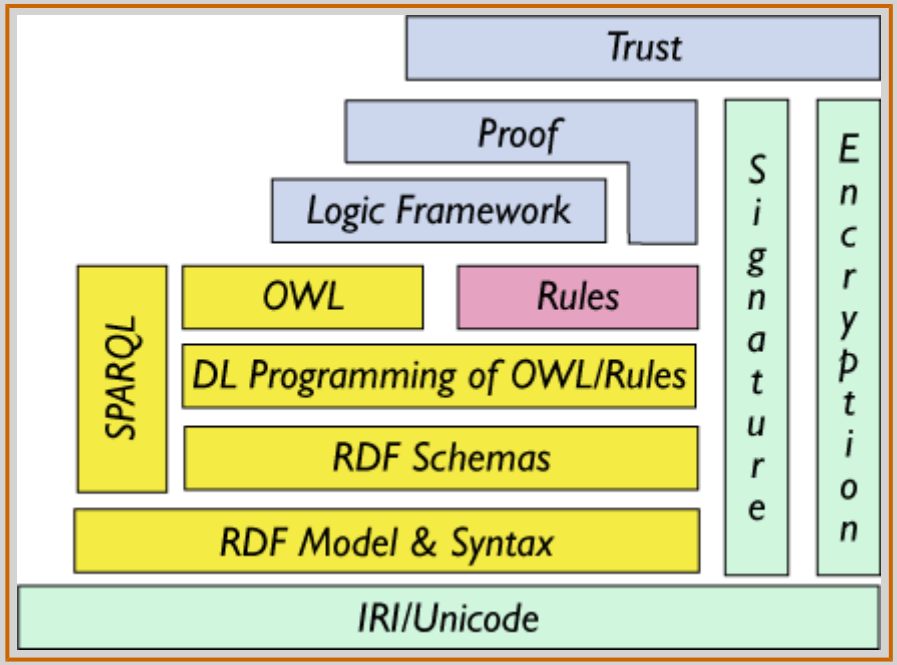
RIF Phase 2 Goals

- Define more complex extensions
  - *towards First Order Logic (FOL), Logic Programming systems…*
  - *syntactic extensions to Horn logic like Lloyd-Topor (i.e., making the rule bodies and head richer), HiLog (i.e., some form of higher order syntax)*
- First recommendation(s) in May 2008

Lots of Theoretical Questions to Solve

- Open vs. Closed Worlds, monotonicity vs. non-monotonicity
- How to use various logic systems (Description Logic, F-Logic, Horn,…) in a coherent framework
- Relationship to RDFS and OWL: “One Tower” vs. “Two Towers” model:





Beyond Rules: Trust

- Can I trust a (meta)data on the Web?
  - *is the author the one who claims he/she is, can I check his/her credentials?*
  - *can I trust the inference engine?*
  - *etc.*
- There are issues to solve, e.g.,
  - *how to "name" a full graph*
  - *protocols and policies to encode/sign full or partial graphs (blank nodes may be a problem to achieve uniqueness)*
  - *how to "express" trust? (e.g., trust in context)*
- It is on the "future" stack of W3C and the SW Community …

A Number of Other Issues…

- Lot of R&D is going on:
  - *improve the inference algorithms and implementations, scalability, reasoning with OWL Full*
  - *temporal & spatial reasoning, fuzzy logic*
  - *better modularization (import or refer to part of ontologies)*
  - *procedural attachments (e.g., in rules)*
  - *ontology management on the Web*
  - *…*
- This mostly happens outside of W3C, though
  - *W3C is not a research entity…*

Available Documents, Tools

Available Specifications: Primers, Guides

- The “RDF Primer” and the “OWL Guide” give a formal introduction to RDF(S) and OWL
- SKOS has its separate “SKOS Core Guide”
- The “RDF Test Cases” and the “OWL Test Cases” can be useful resources, too

Available Specifications (cont)

- “RDF: Concept and Abstract Syntax”, “RDF Vocabulary Description Language (RDF Schema)”, and “RDF Semantics” together define the detailed concepts and semantics of RDF+RDFS; “RDF/XML Serialization” defines RDF/XML
  - *Note: there is a previous Recommendation of 1999 that is superseded by these*
- SPARQL is defined by the “SPARQL Query Language for RDF”, “SPARQL Protocol for RDF", and the “SPARQL Query Results XML Format” documents
- SKOS is formally defined by “SKOS Core Vocabulary Specification”

Available Specifications (cont)

- "OWL Overview" gives a simple listing of the OWL properties, "OWL Reference" contains a more detailed (though informal) listing of features
  - *see, e.g., the Overview document to find what is and what is not allowed in OWL Lite or OWL DL*
- "OWL Semantics and Abstract Syntax" is the normative definition

Some Books

- 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)
- F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider: The Description Logic Handbook (2003)
- G. Antoniu, F. van Harmelen: Semantic Web Primer (2004)
- A. Gómez-Pérez, M. Fernández-López, O. Corcho: Ontological Engineering (2004)
- …

Further Information

- Dave Beckett's Resources at Bristol University
  - *huge list of documents, publications, tools, …*
- Semantic Web Community Portals, e.g.:
  - *Semanticweb.org*
  - *"Business model IG" (part of semanticweb.org)*
  - *list documents, software, host project pages, etc,…*
- The Semantic Web Activity page at W3C lists a number of commercial tools

SWBP Working Group Documents

- Documents for ontology engineering
  - *“Managing a Vocabulary for the Semantic Web”*
  - *“Defining N-ary relations”*
  - *“Representing Classes As Property Values”*
  - *“Representing "value partitions" and "value sets"”*
  - *“XML Schema Datatypes in RDF and OWL”*
- Semantic Web Tutorials (list of references)
- Metadata modules in XHTML2 (jointly with the HTML WG)
- Links with OMG’s “Ontology Definition Metamodel”
- “Ontology Driven Architectures in Software Engineering”

Further Information (cont)

- Description Logic links:
  - *online course by Enrico Franconi,*
  - *teaching material and links by Ian Horrocks*
- "Ontology Development 101"
- OWL Reasoning Examples
- *Lots* of papers at WWW2003, WWW2004, and WWW2005, ISWC200X conference proceedings (unfortunately, not on-line…)

Public Fora at W3C

Semantic Web Interest Group

a forum for discussions on applications

RDF Logic

public (archived) mailing list for technical discussions

Some Tools

(Graphical) Editors

- For RDF: IsaViz (Xerox Research/W3C), RDFAuthor, Longwell (MIT)
- For OWL: Protege 2000 (Stanford Univ.) SWOOP (Univ. of Maryland)Orient (IBM Alphawork)
- Altova's SemanticWorks
- …

Further info on RDF/OWL tools at:

SemWebCentral (see also previous links…)

Programming environments

We have already seen some;
but Jena 2 and SWI-Prolog do OWL reasoning, too!

Some Tools (Cont.)

Validators

For RDF: W3C RDF Validator; For OWL-DL: WonderWeb, Pellet (can also be downloaded as a reasoner tool)

Reasoners that can be built into an application

Pellet, KAON2

Ontology converter (to OWL)

at the Mindswap project

Relational Database to RDF/OWL converter

D2R Map

Schema/Ontology/RDF Data registries

e.g., SchemaWeb, SemWeb Central, Ontaria, rdfdata.org,…

Metadata Search Engine

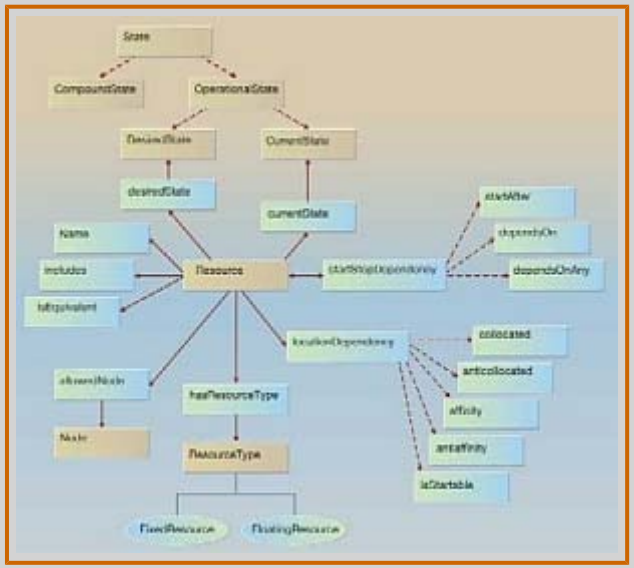
Swoogle

Oracle's Spatial RDF Data Model

- An RDF data model to store RDF statements (available in Oracle Database 10*g*)
- An **`SDO_RDF_MATCH`** table function (usable from SQL) to query triplets
  - *has the capabilities of SPARQL on an "API level" already*
  - *it also has some Horn logic inference capabilities*
- Java Ntriple2NDM converter for loading existing RDF data
- See the Oracle Semantic Technology Center for more detals…

IBM – Life Sciences and Semantic Web

- IBM Internet Technology Group
  - *focusing on general infrastructure for Semantic Web applications*
- Integrated tool kit (storage, query, editing, annotation, visualization)
- Common representation (RDF), unique ID-s (LSID), collaboration, …
- Focus on Life Sciences (for now)
  - *but a potential for transforming the scientific research process*



Some Application Examples

SW Applications

- Large number of applications emerge:
  - *first applications were RDF only…*
  - *…but recent ones use ontologies, too*
    - huge number of ontologies exist already, with proprietary formats
    - converting them to RDF/OWL is a significant task (but there are converters)
- Most applications are still “centralized”, not many decentralized applications yet
- For further examples, see, for example, the SW Technology Conference
  - *not a scientific conference, but commercial people making real money!*

Data integration

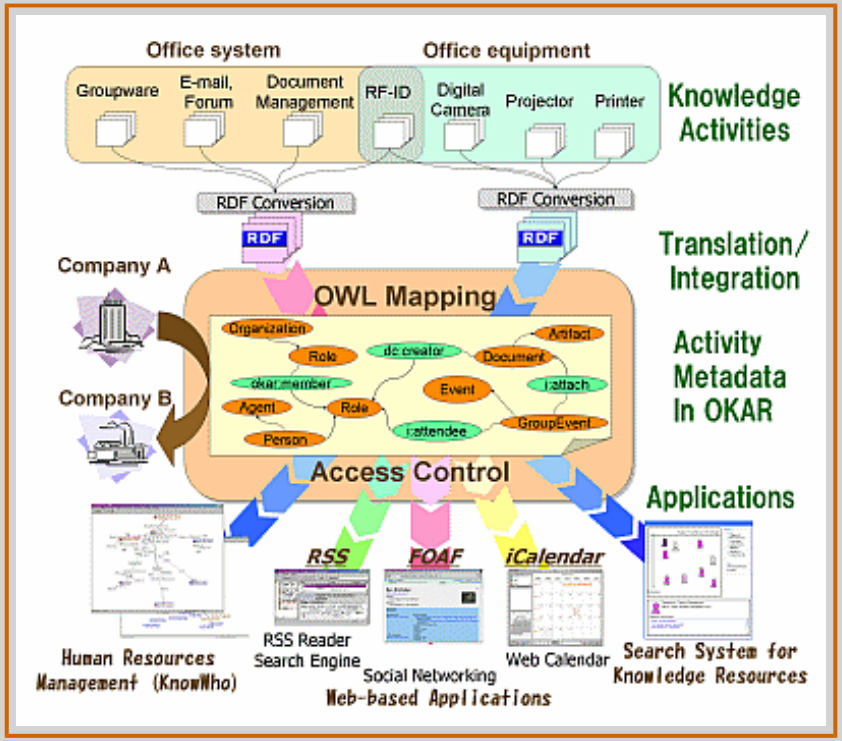
- Semantic integration of corporate resources or different databases
- RDF/RDFS/OWL based vocabularies as an “interlingua” among system components (early experimentation at Boeing, see, e.g., a WWW11 paper)
- Similar approaches: Sculpteur project, MITRE Corp., MuseoSuomi, …

Portals

- Vodaphone's Live Mobile Portal
  - *search application (e.g. ringtone, game, picture) using RDF*
    - page views per download decreased 50%
    - ringtone up 20% in 2 months
- Sun’s SwordFish: public queries for support, handbooks, etc, go through an internal RDF engine for White Paper Collections and System Handbook collections
- Nokia has a somewhat similar support portal

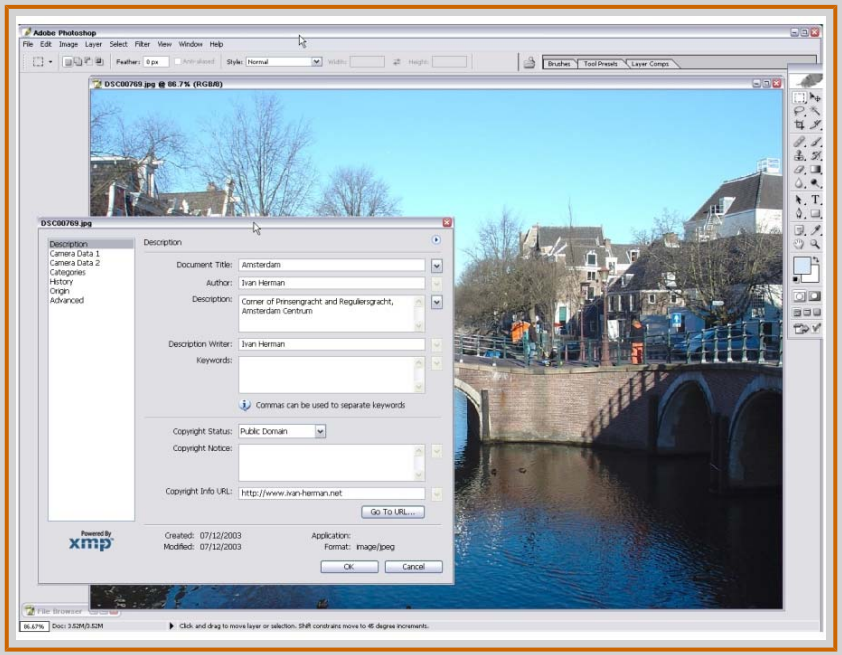
OKAR Fujitsu's and Ricoh's OKAR

- Management of office information, projects, personal skills, calendars, …
  - *e.g., "find me a person with a specific skill"*
- Still an R&D project



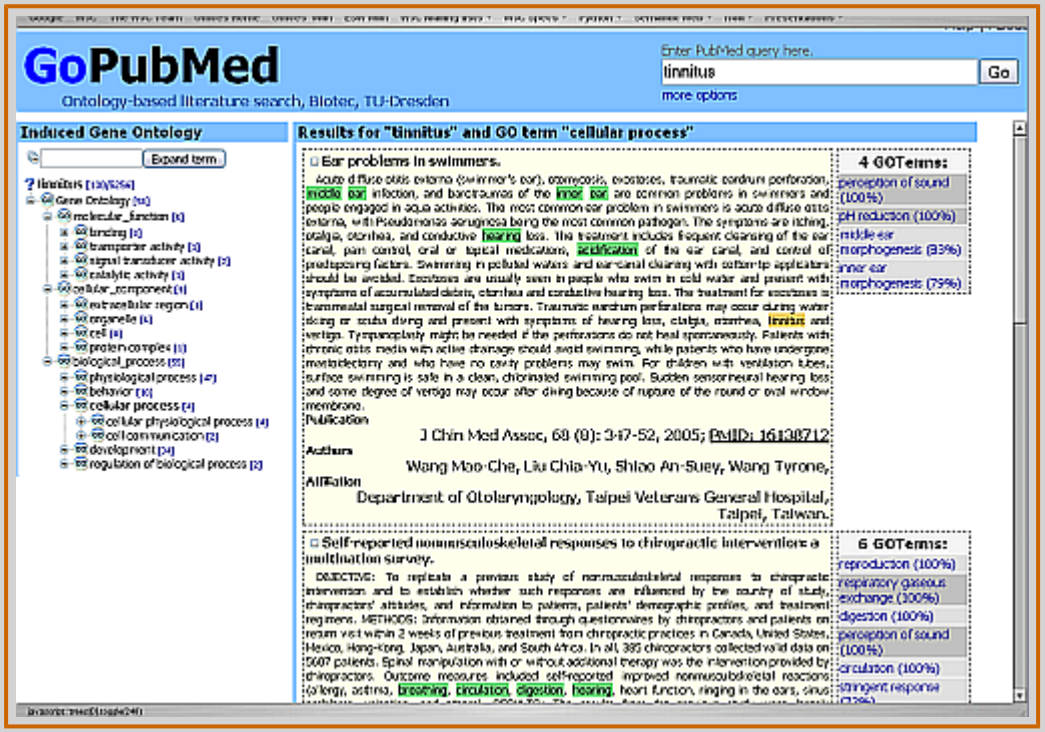
Adobe's XMP

- Adobe’s tool to add RDF-based metadata to *most* of their file formats
  - *used for more effective organization*
  - *supported in Adobe Creative Suite*
  - *support from 30+ major asset management vendors*
- The tool is available for all!



Improved Search via Ontology: GoPubMed

- Improved search on top of pubmed.org
- Search results are ranked using the specialized ontologies
- Extra search terms are generated and terms are highlighted
- Importance of *domain specific ontologies* for search improvement

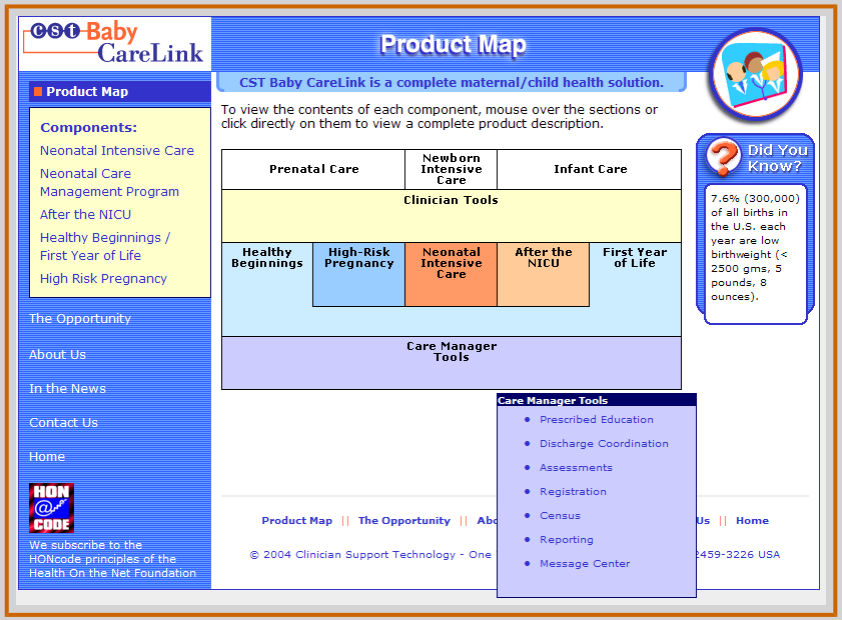


Creative Commons

- 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

- Center of information for the treatment of premature babies
- Provides an OWL service *as a Web Service*
  - *combines disparate vocabularies like medical, insurance, etc*
  - *users can ask complex questions and add new entries to ontologies*
- *Example for the synergy of Web Services and the Semantic Web!*



Further Information

These slides are at:

http://www.w3.org/2005/Talks/1214-Trento-IH/

Semantic Web homepage

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

More information about W3C:

http://www.w3.org/ or http://www.w3c.it/

Mail me:

ivan@w3.org