# Tutorial on Semantic Web Technologies

Ivan Herman, W3C

24 May, 2005, Amsterdam

# W3C® > What Will You Hear?

a) Introduction
b) Basic RDF
c) RDF Vocabulary Description Language (RDFS)
d) Some Predefined Classes (Collections, Containers)
e) RDF(S) in Practice
f) Ontologies (OWL)
g) RDF Data Access, a.k.a. Query (SPARQL)
h) Future Developments
i) Available Documents, Tools
j) Some Application Examples

# Introduction

# W3C® > Towards a Semantic Web

- The current Web represents information using
  - natural language (English, Hungarian, Chinese,…)
  - 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: 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 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
- 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"
  - what does it mean to be
    - "elegant"?
    - "*most* elegant"?
  - mathematicians ask these questions all the time…
- It is necessary to characterize the service
  - not only in terms of input and output parameters…
  - …but also in terms of its *semantics*

# > What Is Needed?

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

# What Is Needed (Technically)?

- To make metadata machine processable, we need:
  - unambiguous names for resources (URIs)
  - a common data model for expressing metadata (RDF)
    - and ways to access the metadata on the Web
  - common vocabularies (Ontologies)
- *The "Semantic Web" is a metadata based infrastructure for reasoning on the Web*
- It *extends* the current Web (and does not replace it)

# The Semantic Web is Not

- "Artificial Intelligence on the Web"
  - although it uses elements of logic…
  - … it is much more down-to-Earth (we will see later)
  - it is all about properly representing and characterizing metadata
  - of course: AI systems *may* use the metadata of the SW
    - but 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!

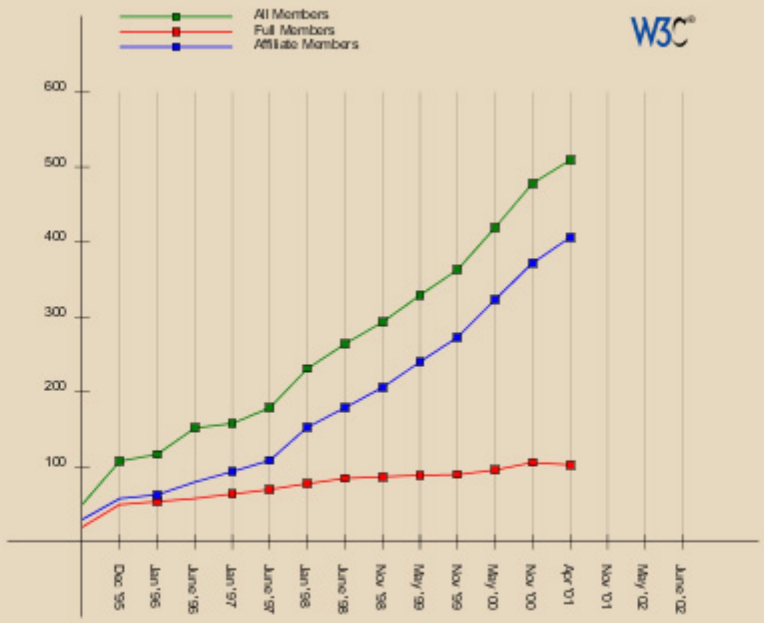
# W3C® > This Course Will

- 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 metadata formalism




W3C Membership evolution 1994-2001

- 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, …

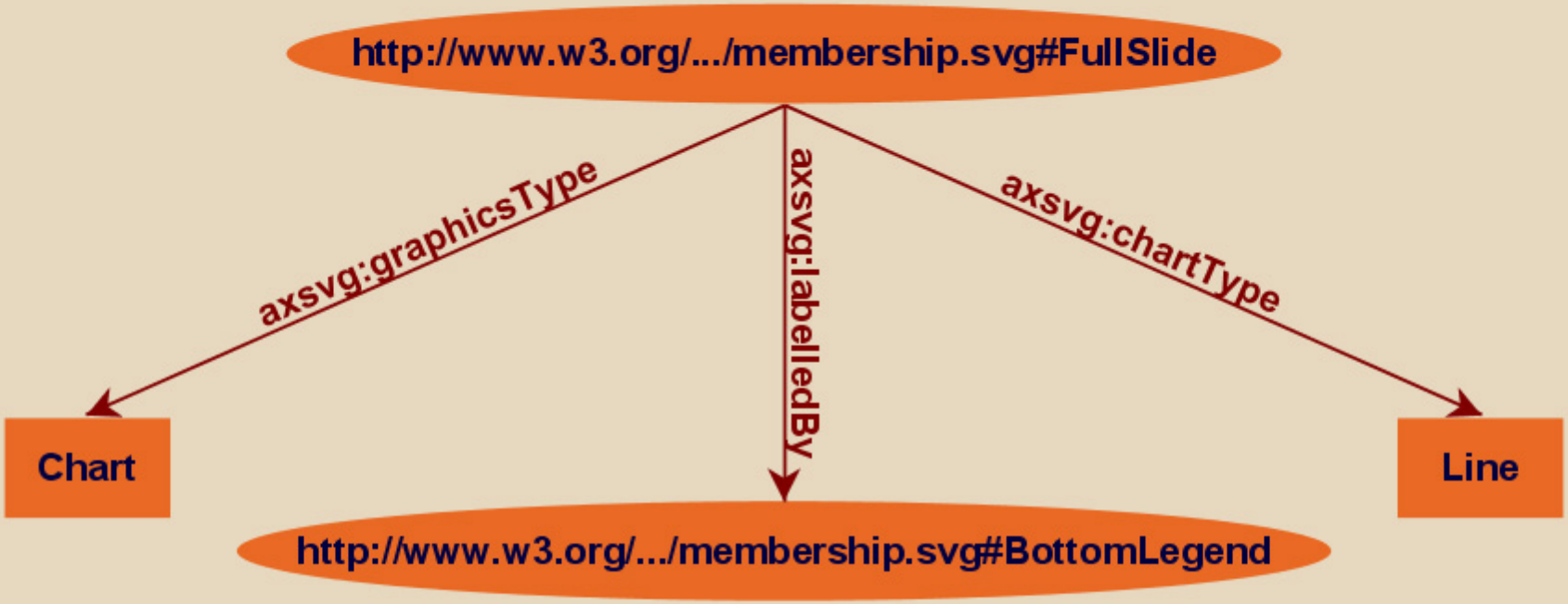
# W3C® > Resource Description Framework

- **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 (e.g., RDF/XML, n3, Turtle, RXR, …)
  - RDF/XML is the "official" W3C format

# RDF is a Graph

- An (s,p,o) triple can be viewed as a 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
  - the formal semantics of RDF is also described using graphs (see the RDF Semantics document)
- One should "think" in terms of graphs, and…
  …XML or n3 syntax are only the tools for practical usage!
  - the term "serialization" is often used for encoding
- RDF authoring tools usually work with graphs, too
  (XML or n3 is done "behind the scenes")

# A Simple RDF Example




```
<rdf:Description
      rdf:about="http://.../membership.svg#FullSlide">
   <axsvg:graphicsType>Chart</axsvg:graphicsType>
   <axsvg:labelledBy
       rdf:resource="http://.../membership.svg#BottomLegend"/>
   <axsvg:chartType>Line</axsvg:chartType>
</rdf:Description>
```

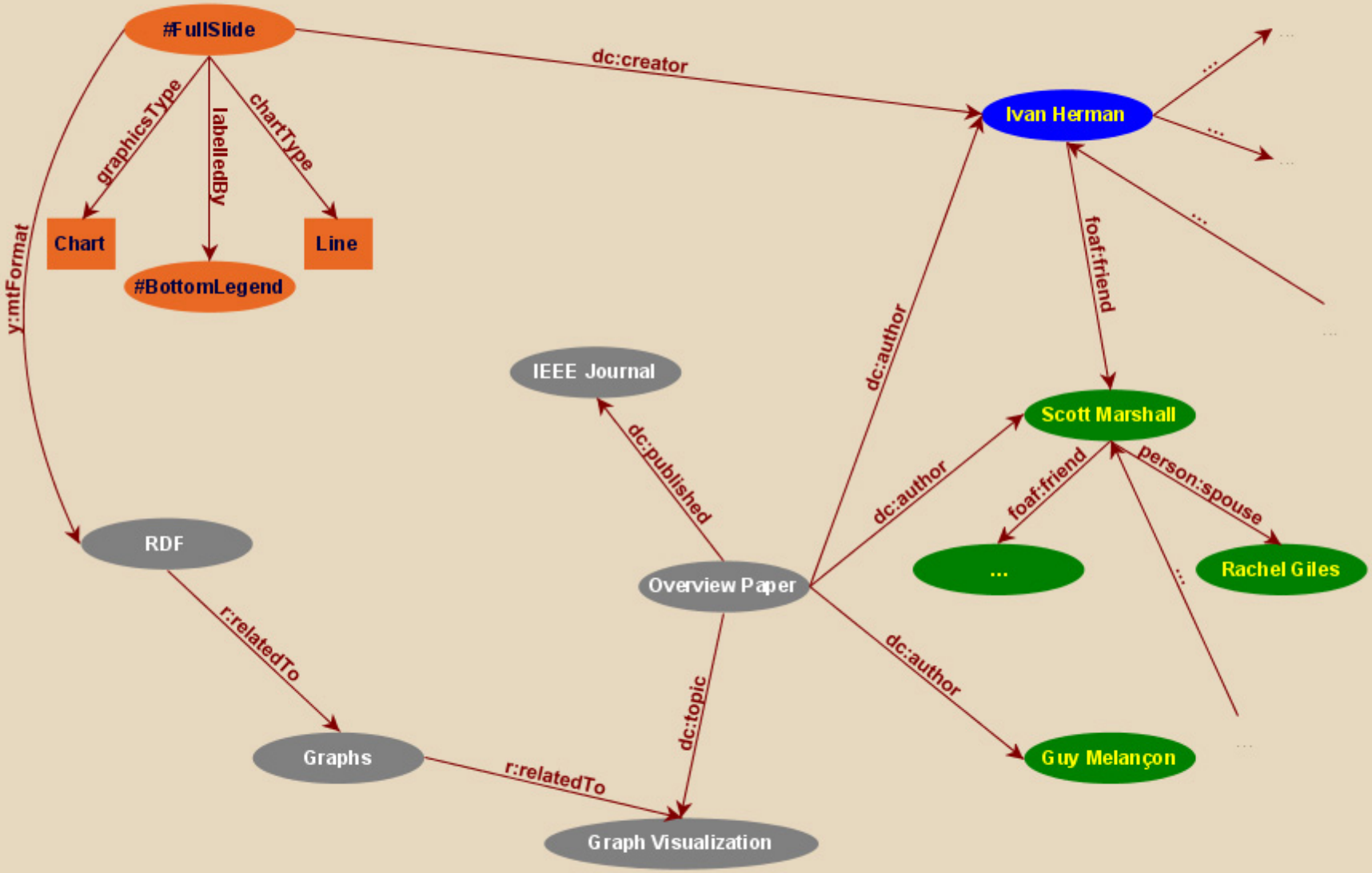
# URI-s Play a Fundamental Role

- One can *uniquely* identify all resources on the web
- Uniqueness is vital to make consistent statements
- *Anybody* can create metadata on *any* resource on the Web
  - e.g., the *same* SVG file could be annotated through other terms
- *URI-s ground RDF into the Web*
  - e.g., information can be retrieved using existing tools

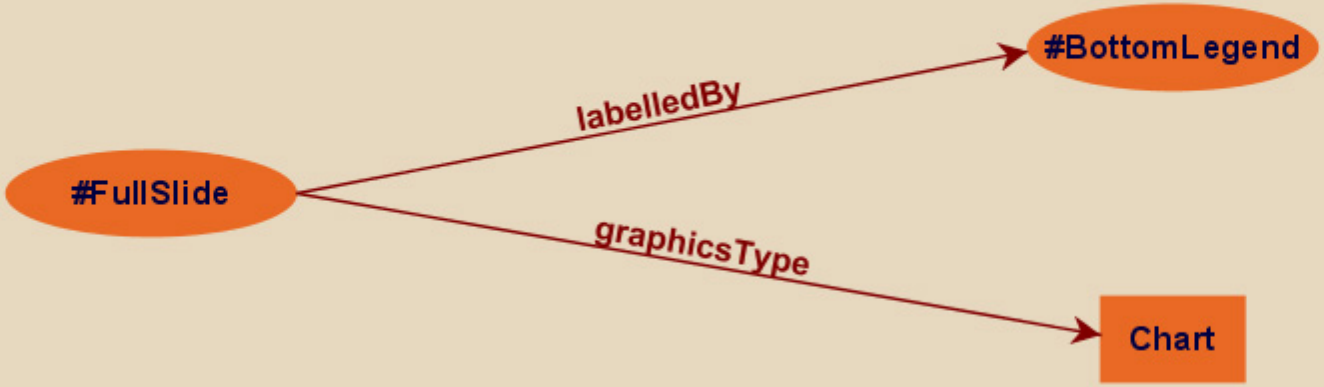
# URI-s: Merging

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

# Use of RDF in Our Example

The tool:

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

# Merge in Practice

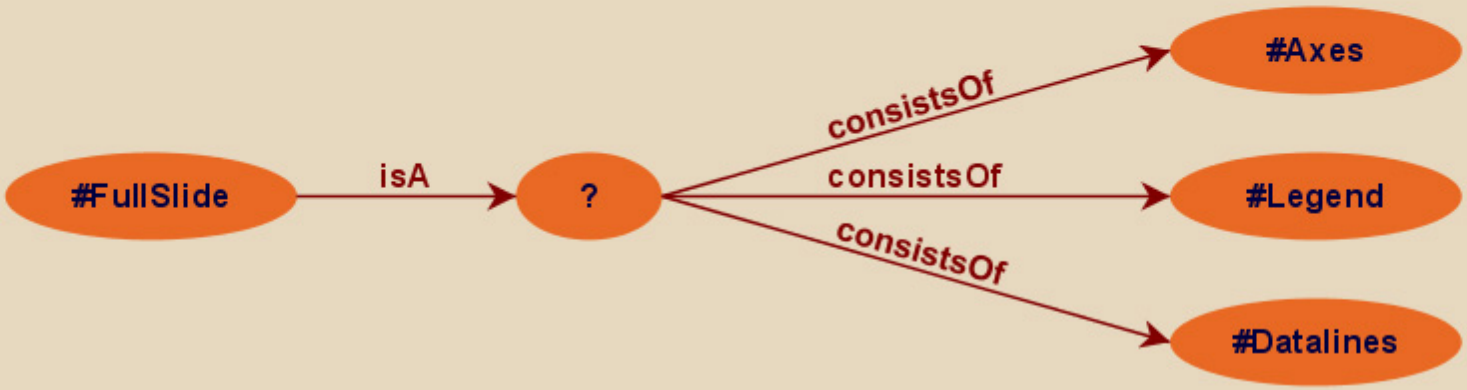
- Development environments merge graphs automatically
  - e.g., in Python, the Triple Store can “load” several files
  - the load merges the new statements automatically
- Merging the RDF/XML files into one is also possible
  - but not really necessary, the tools will merge them for you
  - keeping them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

# > Adding New Statements

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

# Blank Nodes

- Consider the following statement:
  - "the full slide is a «thing» that consists of axes, 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`**

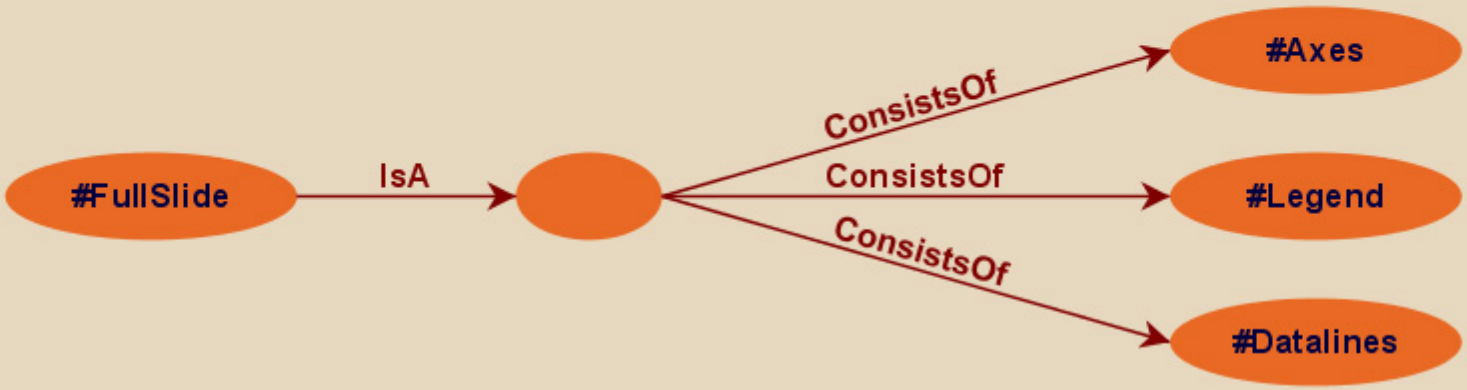
```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:isA>
      <rdf:Description rdf:about="#Thing"/>
   </axsvg:isA>
</rdf:Description>
<rdf:Description rdf:ID="Thing">
   <axsvg:consistsOf rdf:resource="#Axes"/>
   <axsvg:consistsOf rdf:resource="#Legend"/>
   <axsvg:consistsOf rdf:resource="#Datalines"/>
</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: Let the System Do It

- Let the system create a **nodeID** internally

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



# Blank Nodes: Some More Remarks

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

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

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

# RDF Vocabulary Description Language

(a.k.a. RDFS)

# > Need for RDF Schemas

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

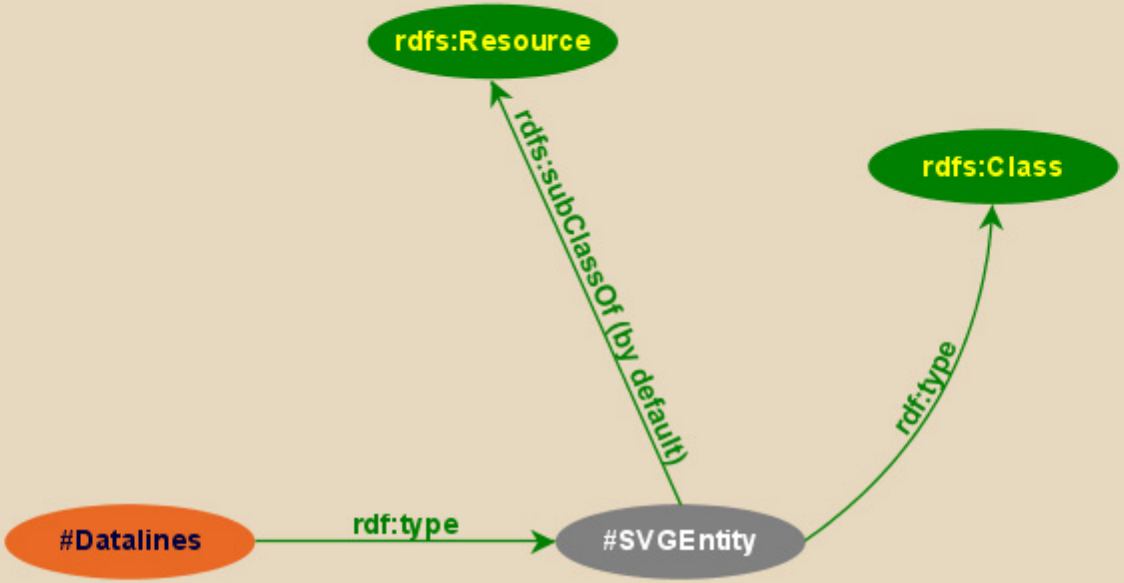
# > Classes, Resources, …

- Think of well known in traditional ontologies:
  - use the term “person”
  - “every Leiden Graduate is a person”
  - “Ivan Herman is a Leiden Graduate”
  - 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”)
    - “person”, “Leiden Graduate”, …

# > Classes, Resources, … (cont.)

- Relationships are defined among classes/resources:
  - "typing": an individual belongs to a specific class
    - "Ivan Herman is a Leiden Graduate"
  - "subclassing": instance of one is also the instance of the other
    - "every Leiden Graduate is a person"
- *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
- User should create RDF Schema file for the user types
  (note: RDFS is also RDF!)

# Schema Example in RDF/XML

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

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

# 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:

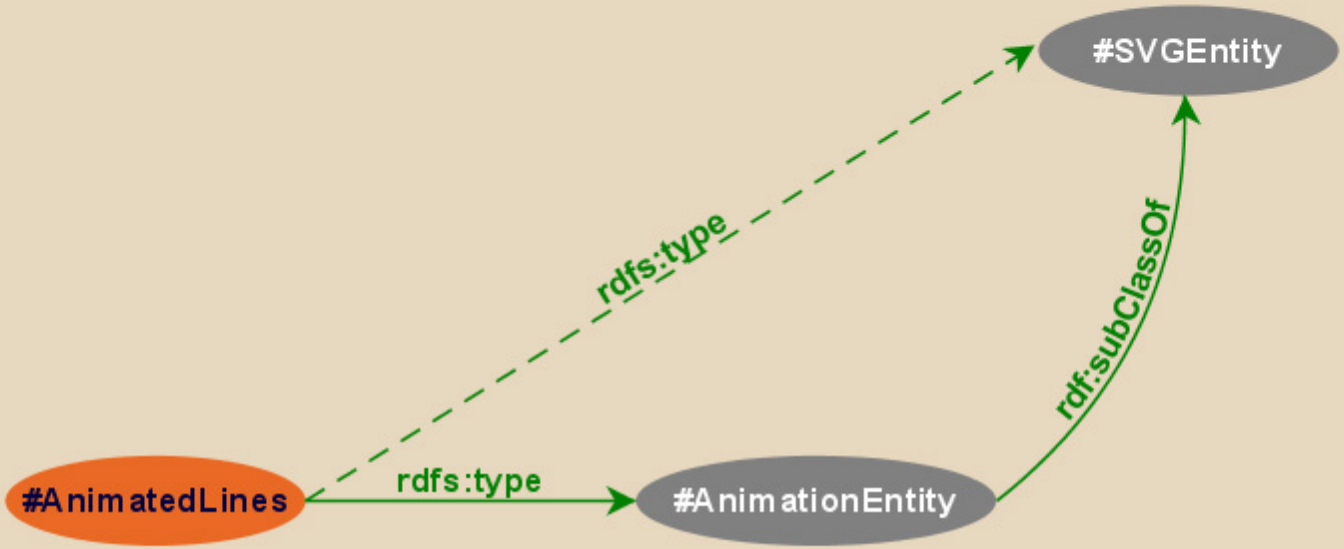
```
<rdf:RDF xmlns:axsvg="axsvg-schema.rdf#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <axsvg:SVGEntity rdf:about="#Datalines">
        ...
    </axsvg:SVGEntity>
```

# Typed Nodes (cont)

- A resource may belong to several classes
  - `rdf:type` is just a property…
  - "Ivan Herman is a Leiden Graduate, but he is also Hungarian…"

  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 *entailment rules*:
  - "if such and such triplets are in the graph, add this and this triplet"
  - do that recursively
  - 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 .
```

- There are 44 of those…

# > 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

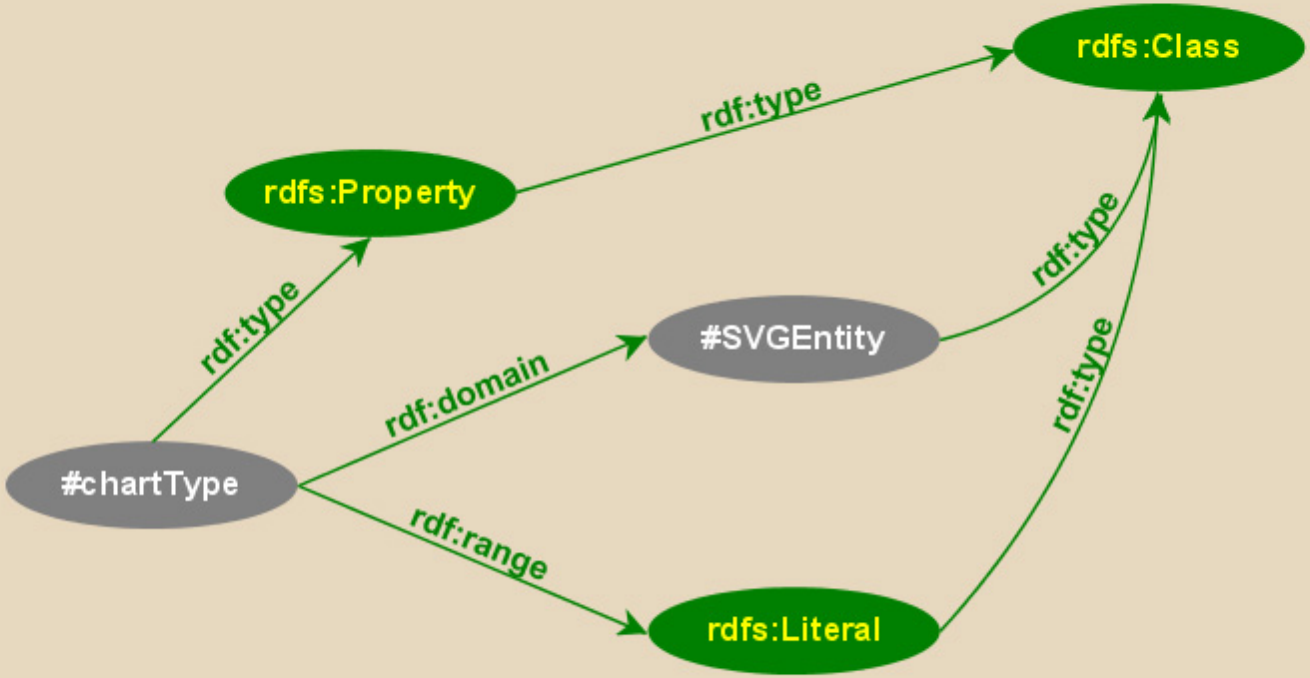
# Properties (cont.)

- 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:rang`**

# Property Specification Example



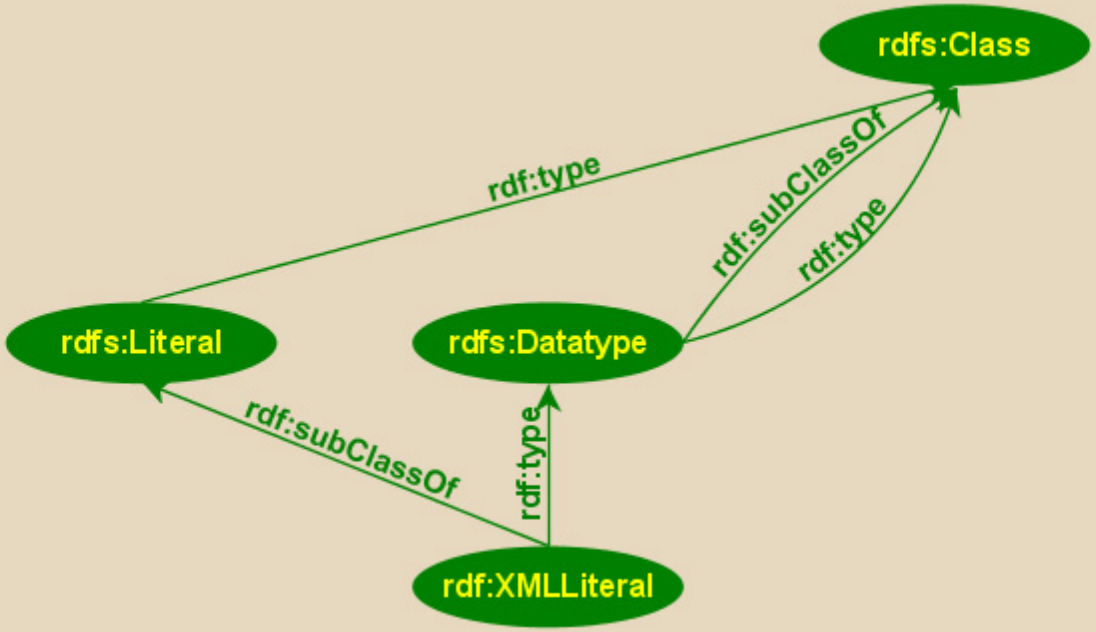

- Note that one cannot define *what* literals can be used
- This requires ontologies (see later)

# 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 may have a data type
  - floats, int, etc.
  - most of the types defined in XML Schemas
- (Natural) language can be specified (via `xml:lang`)
- Formally, data types are separate RDFS classes
- Full XML fragments may also be literals



# Literals in RDF/XML

- Typed literals:

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

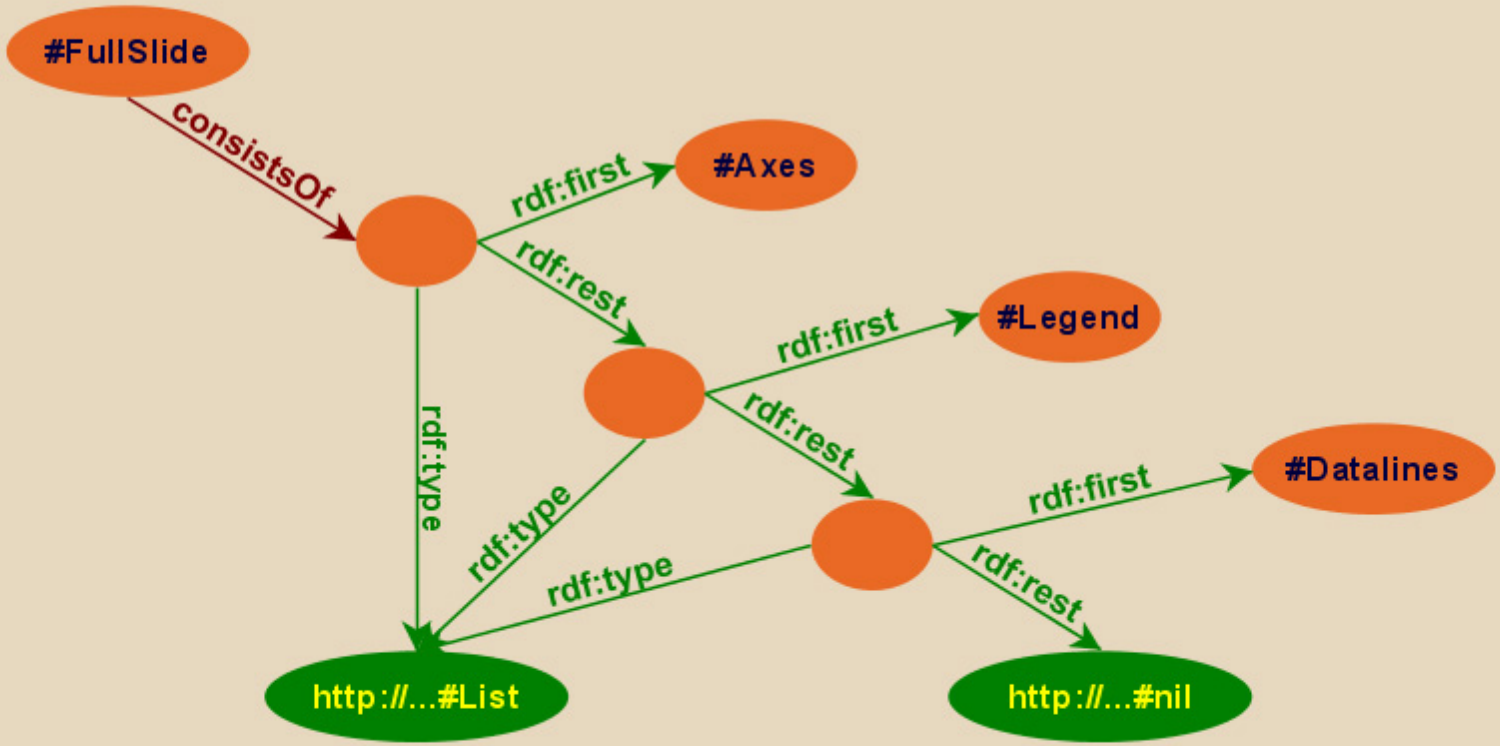
  …just classes with an agreed semantics
- These are:
  - collections (a.k.a. lists)
  - containers: sequence, bag, alternatives

# 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.)

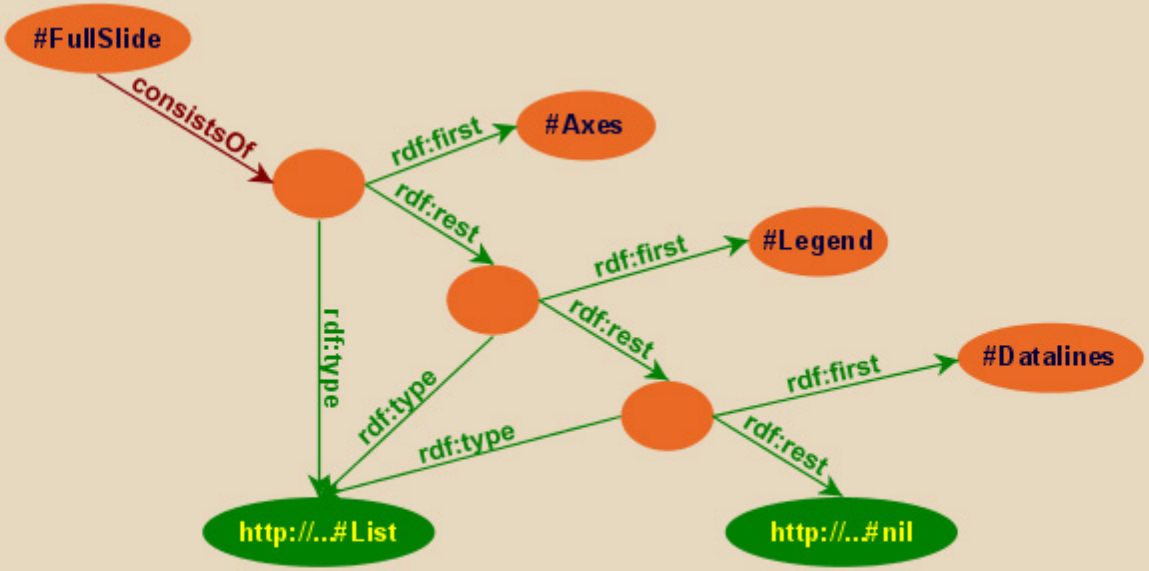
- Familiar structure for Lisp programmers…



# The Same in RDF/XML

List in terms of XML:

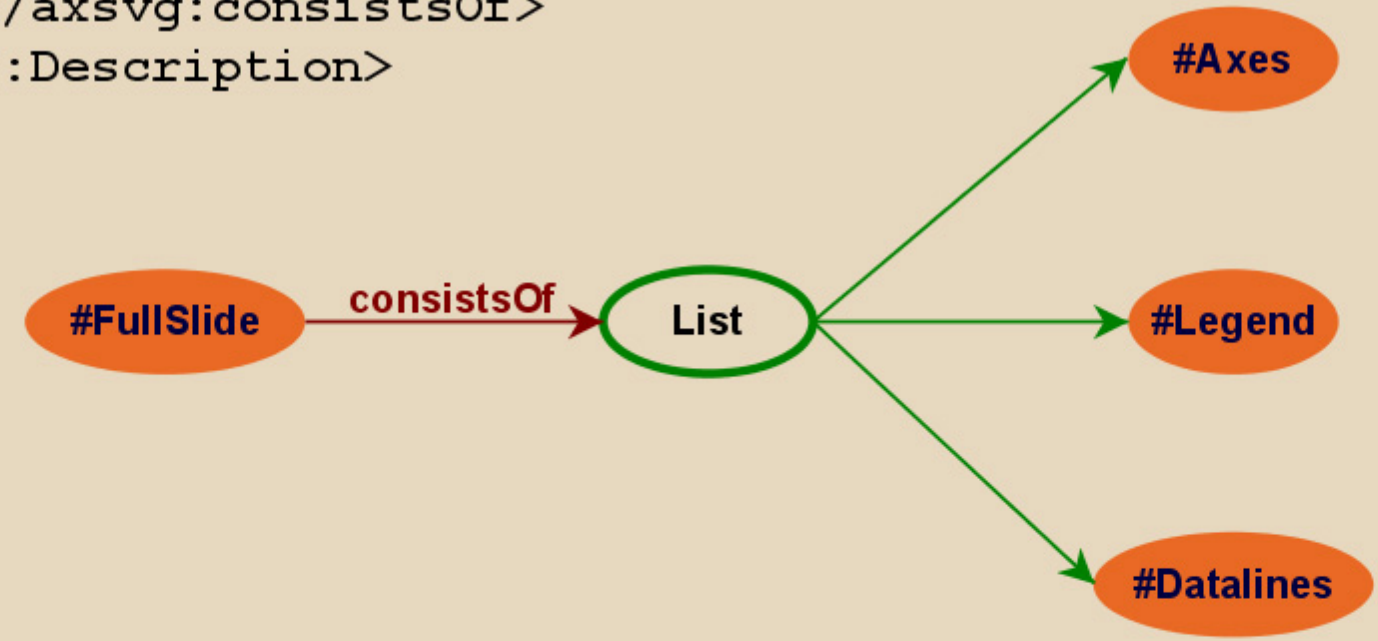
```
<rdf:Description rdf:about="#FullSlide">
    <axsvg:consistsOf rdf:parseType="Collection">
        <rdf:Description rdf:about="#Axes"/>
        <rdf:Description rdf:about="#Legend"/>
        <rdf:Description rdf:about="#Datalines"/>
    </axsvg:consistsOf>
</rdf:Description>
```



# > Our Graphical Shorthand

(To simplify the images…)

```
<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, Bags, Alt-s
- They all have the agreed semantics, with some syntactic help in RDF/XML

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

- A Sequence may be seen as an alternative to collections
  - but a collection is “closed” (via **`rdfs:nil`**)
  - whereas a named **`Seq`** node might be extended

# 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 (perfectly o.k.!)
    - using conventions instead of namespaces in metas
  - put RDF in a comment (e.g., Creative Commons)
- XHTML 2.0 will have a separate 'metadata' module
  - essentially, the current meta/link elements are extended
  - one can define "triplets" using this formalism
  - in fact, a new RDF serialization… (like RDF/XML and n3)

# RDF Can Also Be Generated

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

# > 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
    - 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

# W3C® > Programming Practice

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

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

# > Programming Practice (cont)

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

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

- It is very easy to start with this
- Does not have (yet) powerful schema processing
  - no “inferred” 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);
}
```

# > Jena (cont)

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

# > Lots of Other tools

- There are other tools:
  - RDFSuite: another Java environment (from ICS-FORTH)
  - RDFStore: RDF Framework for Perl
  - Redland: RDF Framework, with bindings to C, C++, C#, …
  - Raptor: RDF Parser library, with bindings to C, C++, C#, Python, …
  - RAP: RDF Framework for PHP
  - SWI-Prolog: RDF Framework for Prolog
  - Sesame: Java based storage and query for RDF and RDFS
  - Kowari, Gateway: triple based database systems
    - they may have Jena interfaces, too
  - etc.
- See, for example:
  - tool list at W3C
  - Free University of Berlin list

# 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
    - necessary for complex merging
  - *construct* classes, not just name them
  - restrict a property range *when used for a specific class*
  - etc.

# Ontologies (cont.)

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

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

- We need a *Web Ontologies Language* to define:
  - more on 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

# W3C > W3C's Ontology Language (OWL)

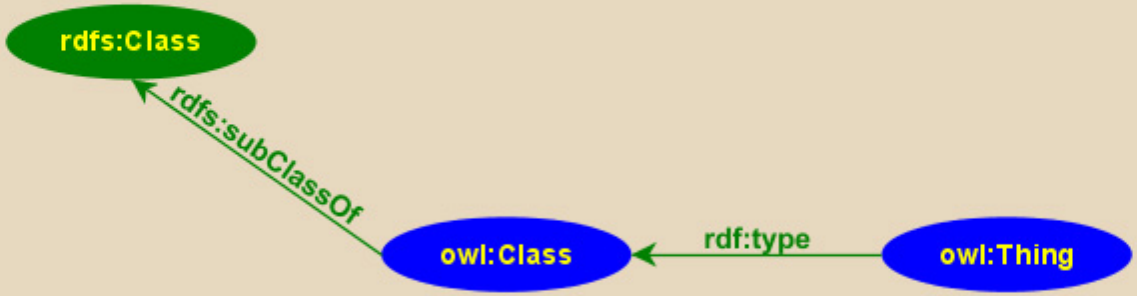
- A layer *on top* of RDFS with additional possibilities
- Outcome of various projects:
  1. a DARPA project: DAML
  2. a EU project: OIL
  3. an attempt to merge the two: DAML+OIL
  4. the latter was submitted to W3C
  5. lots of coordination with the core RDF work
  6. recommendation since early 2004

# Classes in OWL

- In RDFS, you can subclass existing classes…

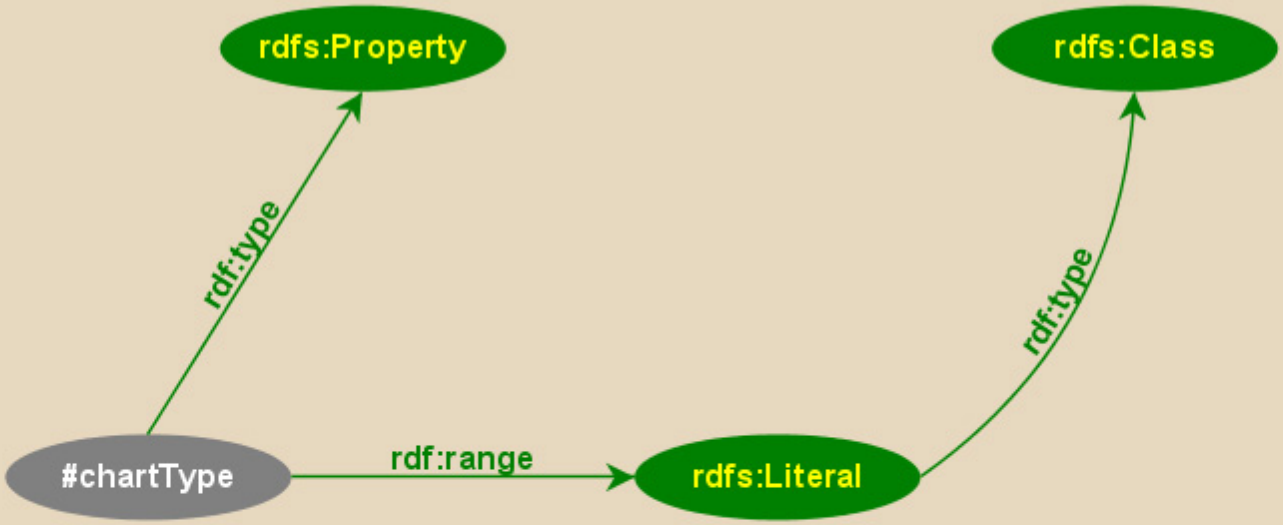
  … but, otherwise, that is all you can do
- In OWL, you can *construct* classes from existing ones:
  - enumerate its content
  - through intersection, union, complement
  - through property restrictions
- To do so, OWL introduces its own `Class`…

  … and `Thing` to differentiate the *individuals* from the *classes*



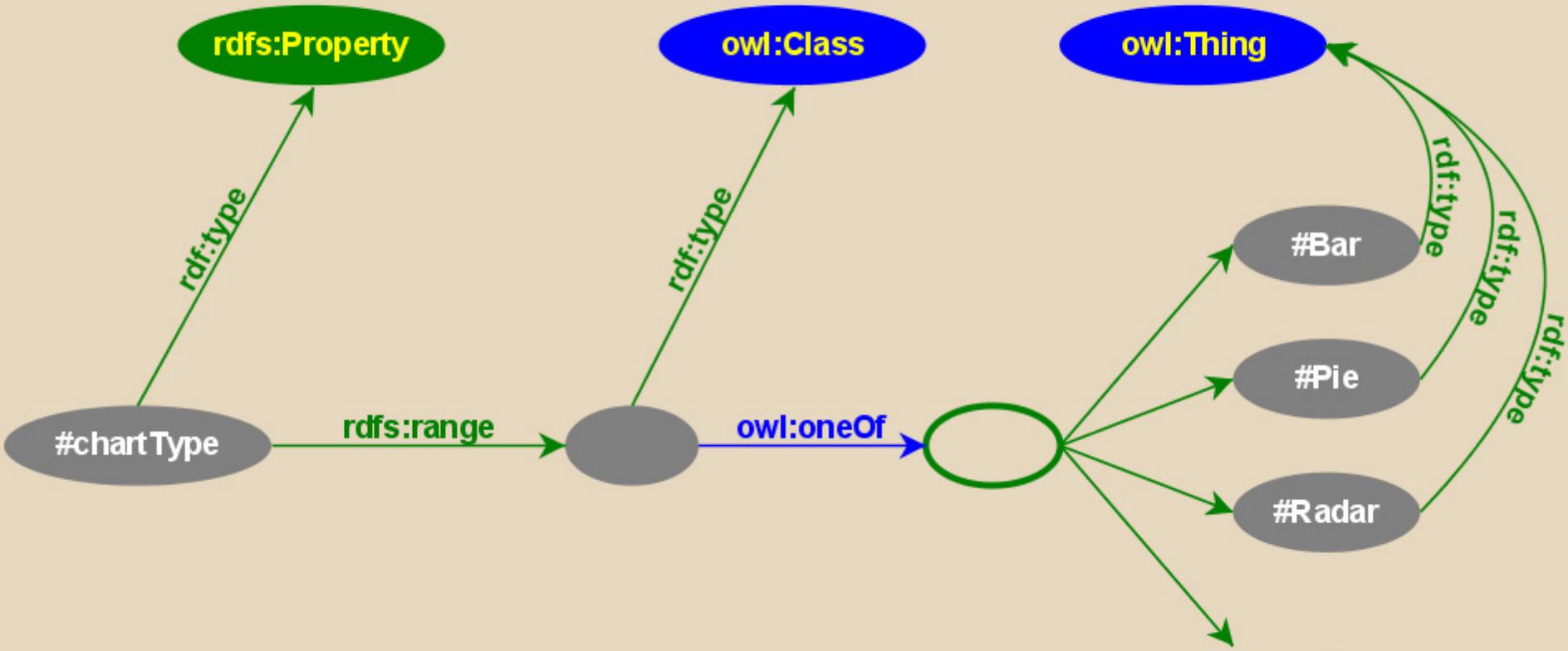
# Need for Enumeration

- Remember this issue?
  - one can use XML Schema types to define 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

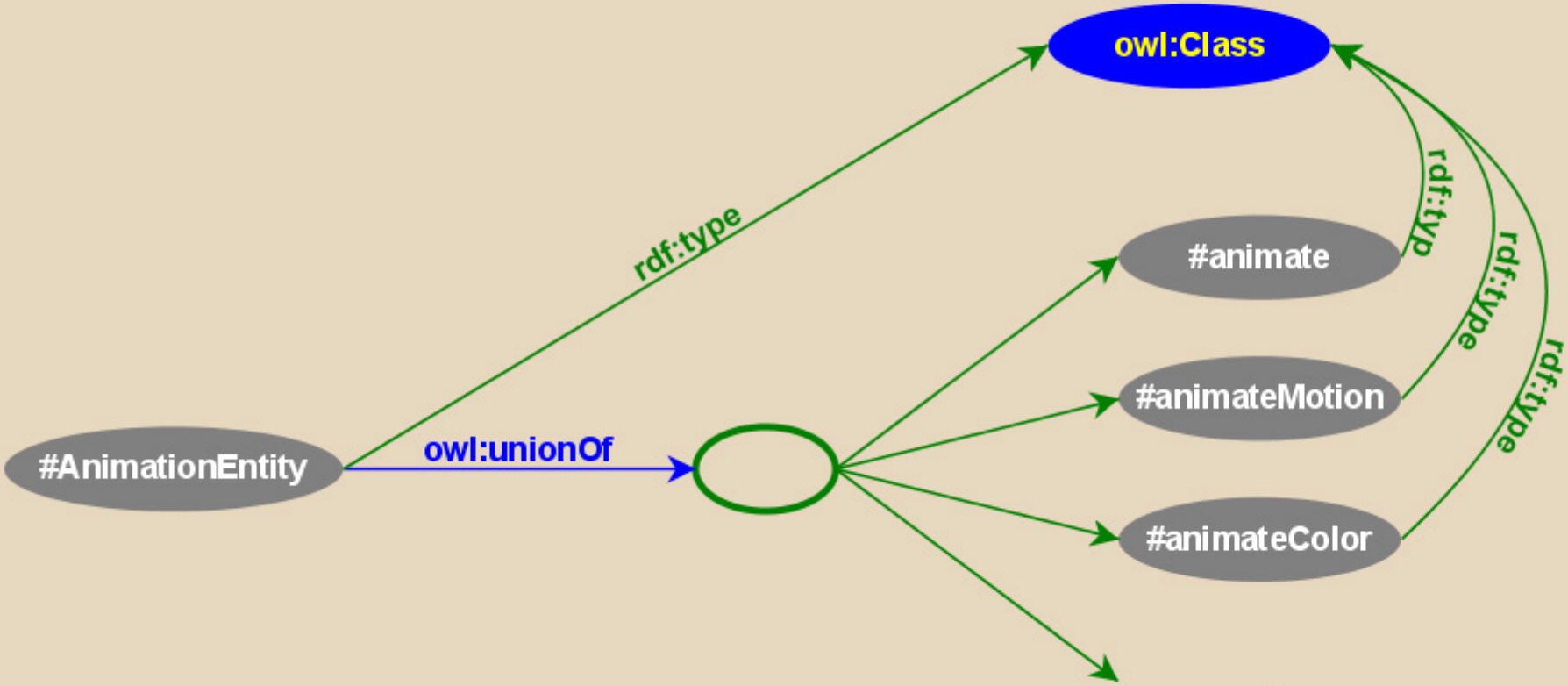
Enumeration in XML:

```
<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>
```

- the class consists of *exactly* of those individuals

# Union of Classes

- Essentially, set-theoretical union:



# Same in RDF/XML

Union in XML:

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

- Other possibilities: **`complementOf`**, **`intersectionOf`**

# Property Restrictions

- (Sub)classes created by restricting the property value *on that class*
- For example, “a Leiden Graduate is a person who has a PhD from Leiden University” means:
  - restrict the *value* of “has a PhD from” when applied to “person”…
  - …thereby define the class of “Leiden Graduate”

- Restriction may be by:
  - value constraints (i.e., further restrictions on the range)
    - *all* values must be from a class
    - *at least one* value must be from a class
  - cardinality constraints

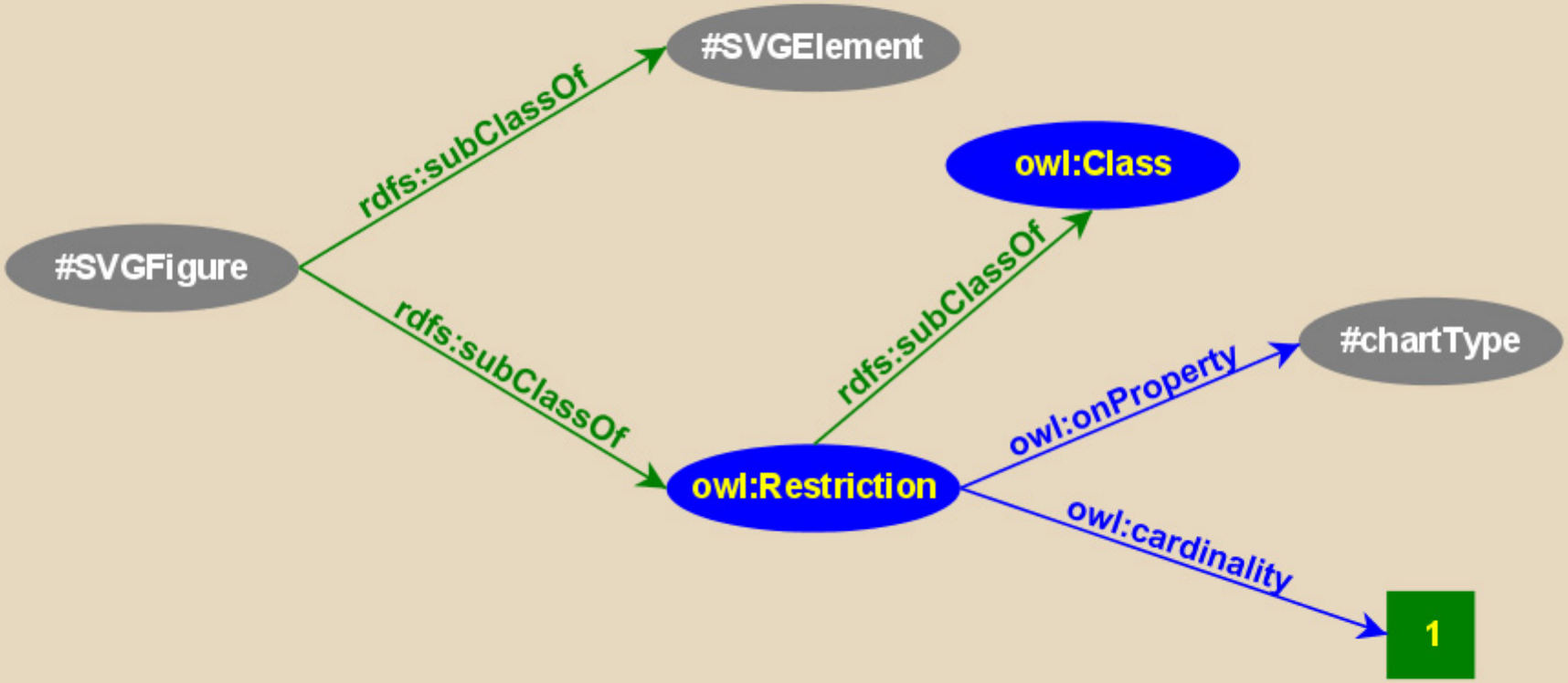
    (i.e., how many times the property can be used on an instance?)
    - minimum cardinality
    - maximum cardinality
    - exact cardinality

# Property Restrictions (cont.)

- Formally:
  - **`owl:Restriction`** defines a blank node with restrictions
    - refer to the property that is constrained
    - define the restriction itself
  - one can, e.g., subclass from this node

# Cardinality Restriction Example

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



# Same in RDF/XML

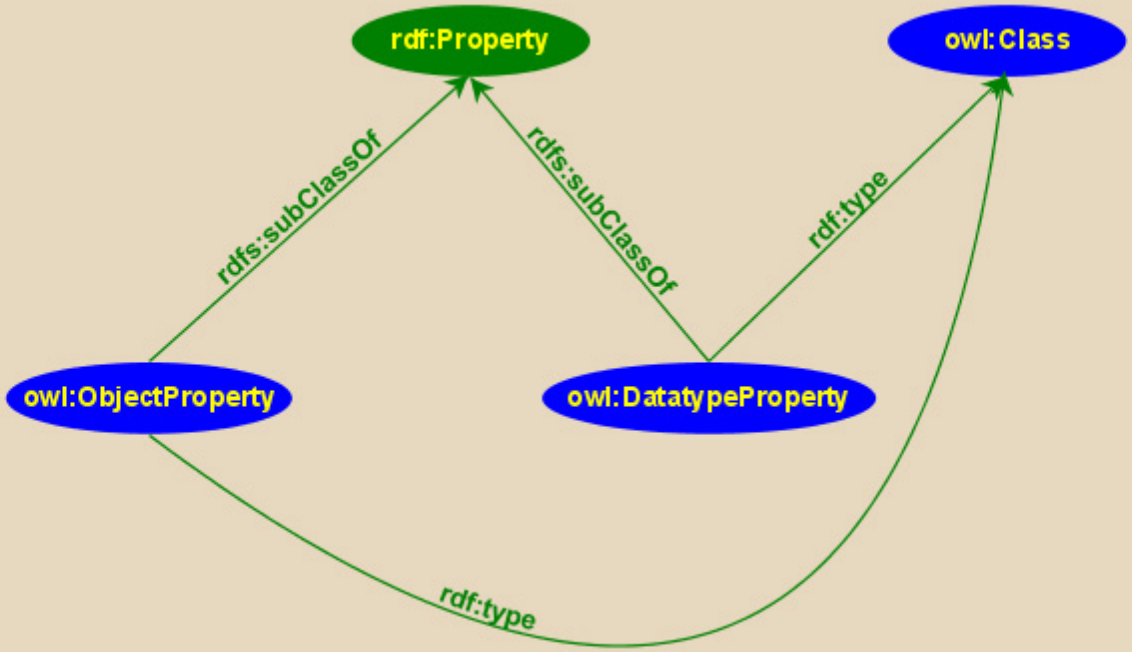
Cardinality constraint in XML:

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

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

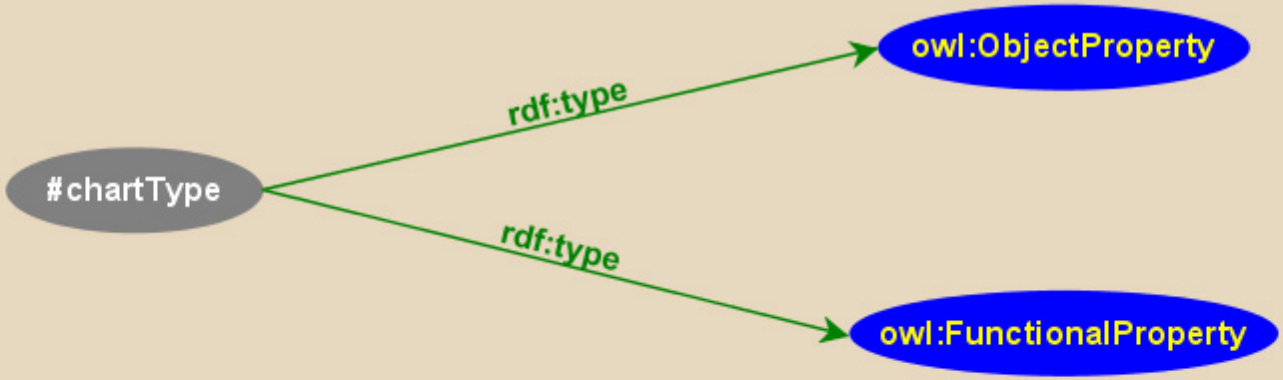
# Property Characterization

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



# Characterization Example

- An alternative for the cardinality=1 setting:



# Same in RDF/XML

Characterization in XML:

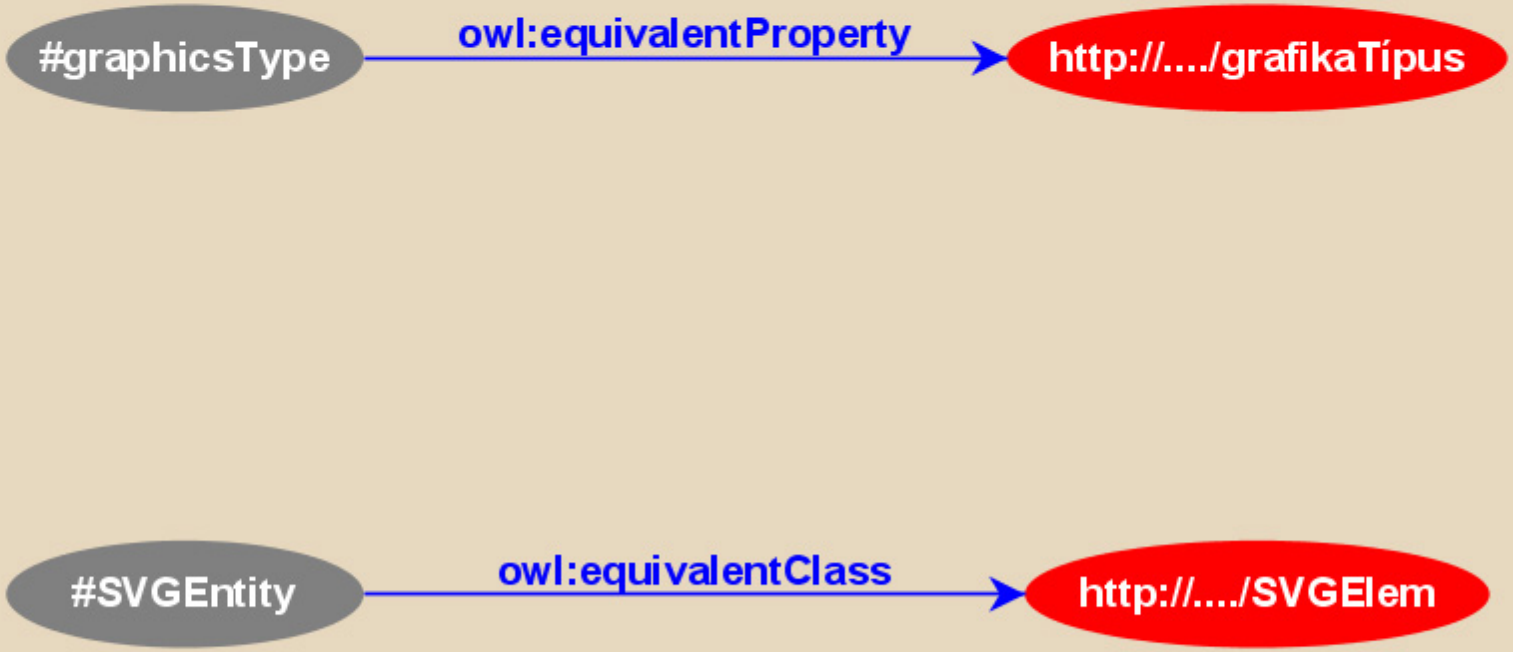
```
<owl:ObjectProperty rdf:ID="ChartType">
   <rdf:type rdf:resource="...../#FunctionalProperty/>
</owl:ObjectProperty>
```

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

- 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**

# Example: Connecting to Hungarian



- 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!
  - (the same is true for RDFS, by the way)
- Inference based on OWL is *within this framework only*
  - it seems modest, but has proved to be remarkably useful…

# Examples for Logic Formalism

- 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!*
  - i.e., SW ≠ AI!

# 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

- 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
- But: *an OWL Full ontology may be undecidable!*

# OWL Description Logic (DL)

- *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:DatatypePropery`** are *strictly separated*
  - i.e., a class *cannot* be an individual of another class
  - object properties' values must usually 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

# Note on OWL layers

- OWL Layers were defined to reflect compromises:
  - expressability vs. implementability
- Research may lead to new decidable subsets of OWL!
  - see, e.g., H.J. ter Horst's paper at ISWC2004

- 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

# DL Abstract Syntax

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

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

- 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 "ontology islands"
  - loosely connected ontologies bound by an application…
  - … connected via, e.g., a P2P architecture…
    - e.g., M.-C. Rousset's paper at ISWC2004

# Ontology Examples

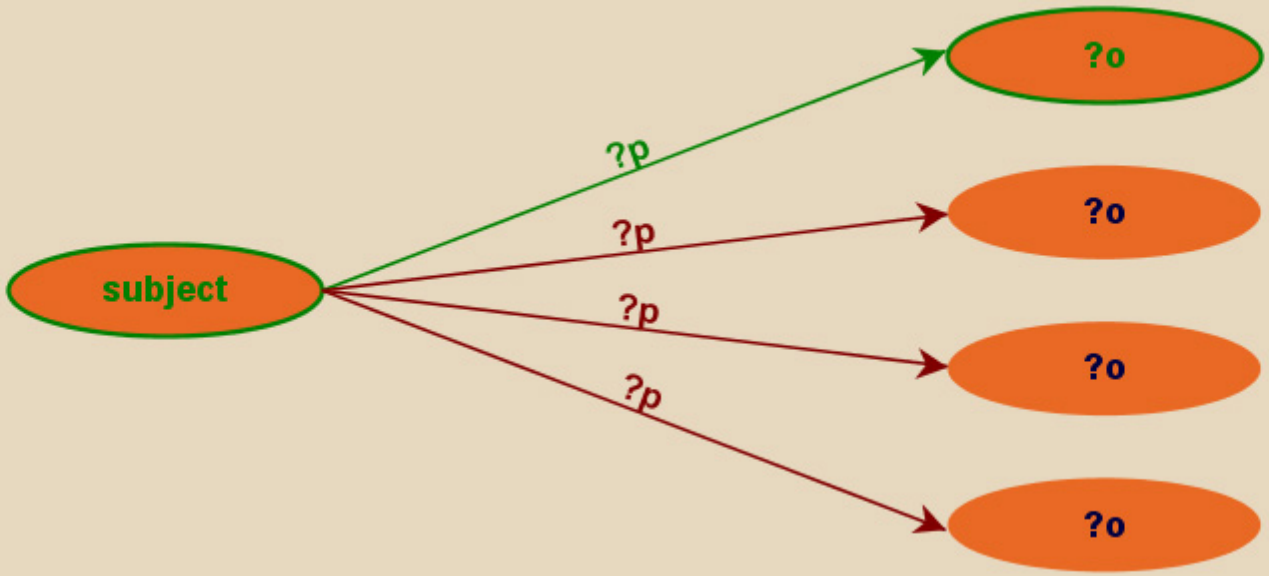
- A possible ontology for our graphics example
  - on the borderline of DL and Full
- International country list
  - example for an OWL Lite ontology

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

- Remember the Python idiom:

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

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



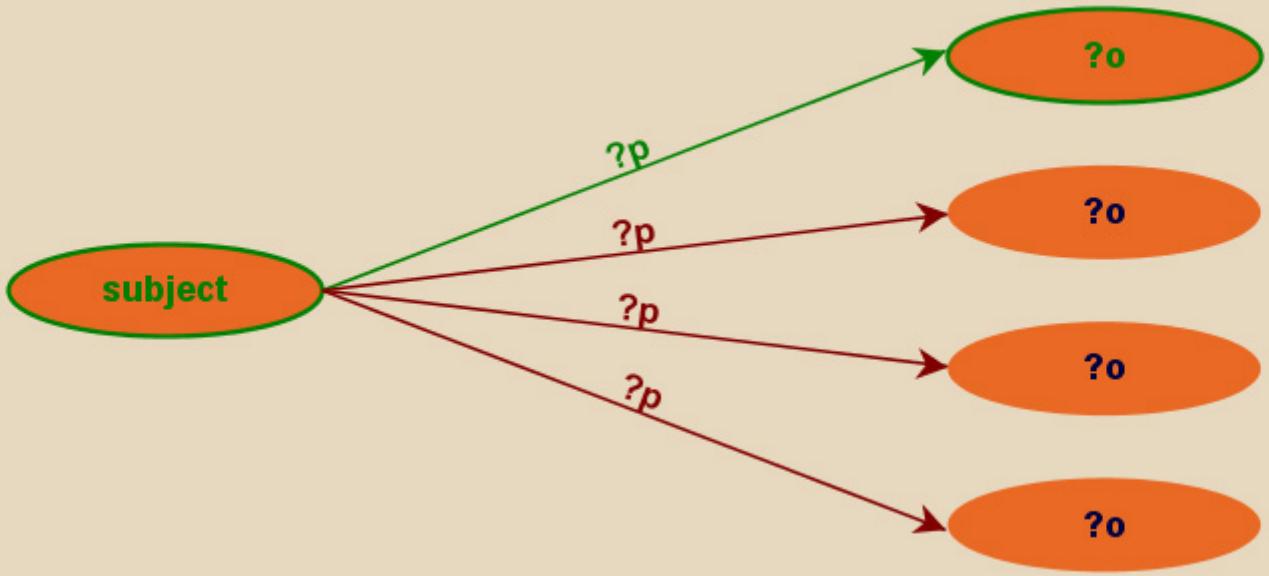
- In practice, more complex queries into the RDF data are necessary
- The fundamental idea: generalize the approach of *graph patterns*:
  - the pattern contains unbound symbols
  - by binding the symbols, subgraphs of the RDF graph may be matched
  - if there is such a match, the query returns the bound resources
- This is the goal of SPARQL (Query Language for RDF)
  - based on similar systems that already exist, e.g., in Jena
  - is programming language-independent query language
  - *still in a working draft phase*
    - Recommendation early 2006?

# Our Example in SPARQL

- The Python example in SPARQL:

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

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



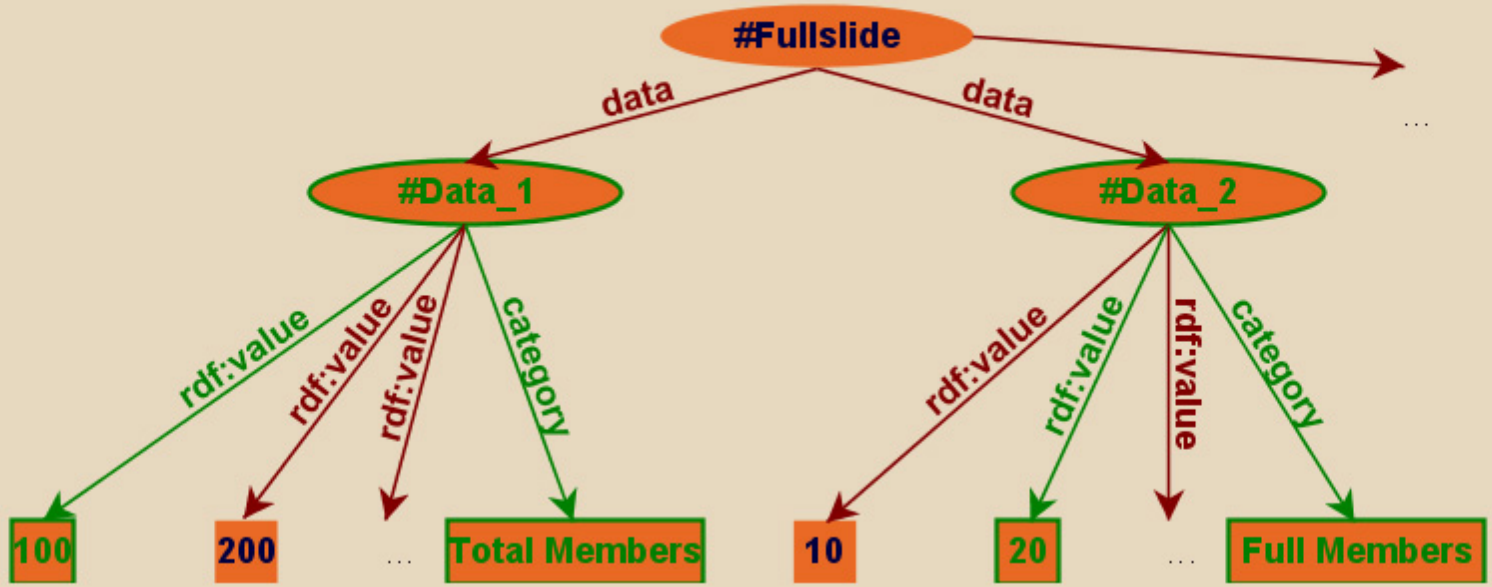
# Simple SPARQL Example

```
SELECT ?cat ?val
WHERE { ?x rdf:value ?val. ?x category ?cat }
```

Returns:

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

- Note the role of `?x`: it helps defining the pattern, but is *not* returned

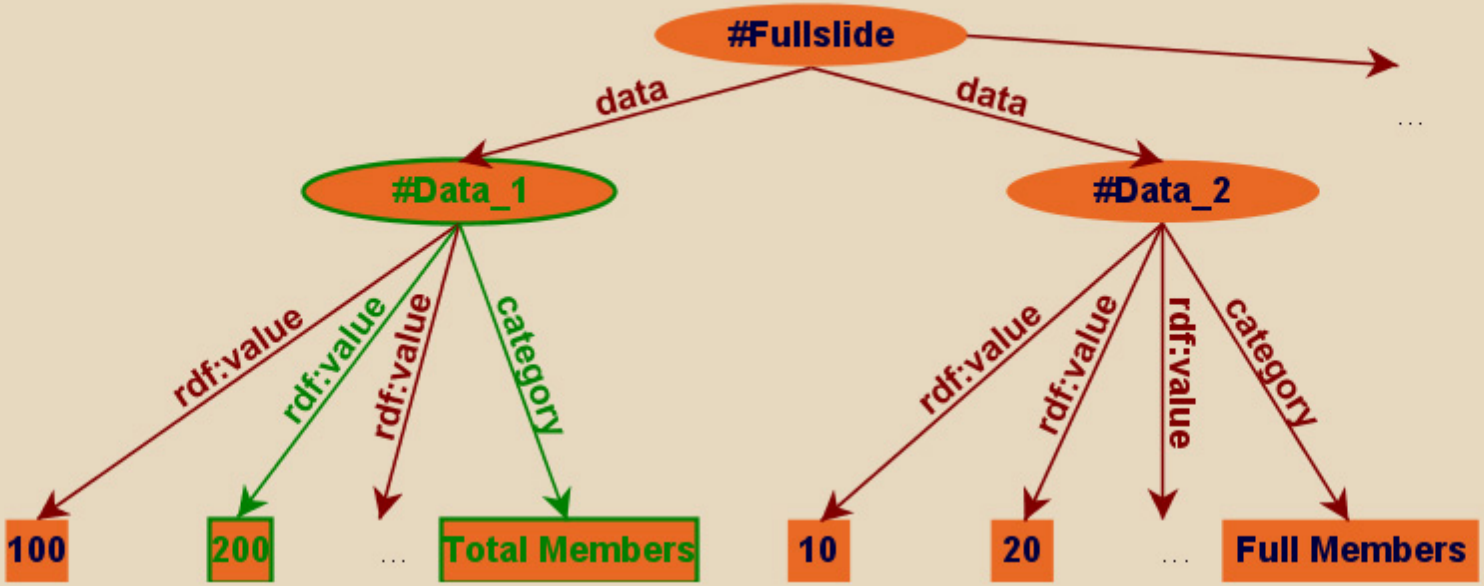


# Pattern Constraints

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

Returns: `[["Total Members",200],…,]`

- SPARQL defines a number of operators for **`FILTER`**
- Applications/implementations may plug in their own condition functions

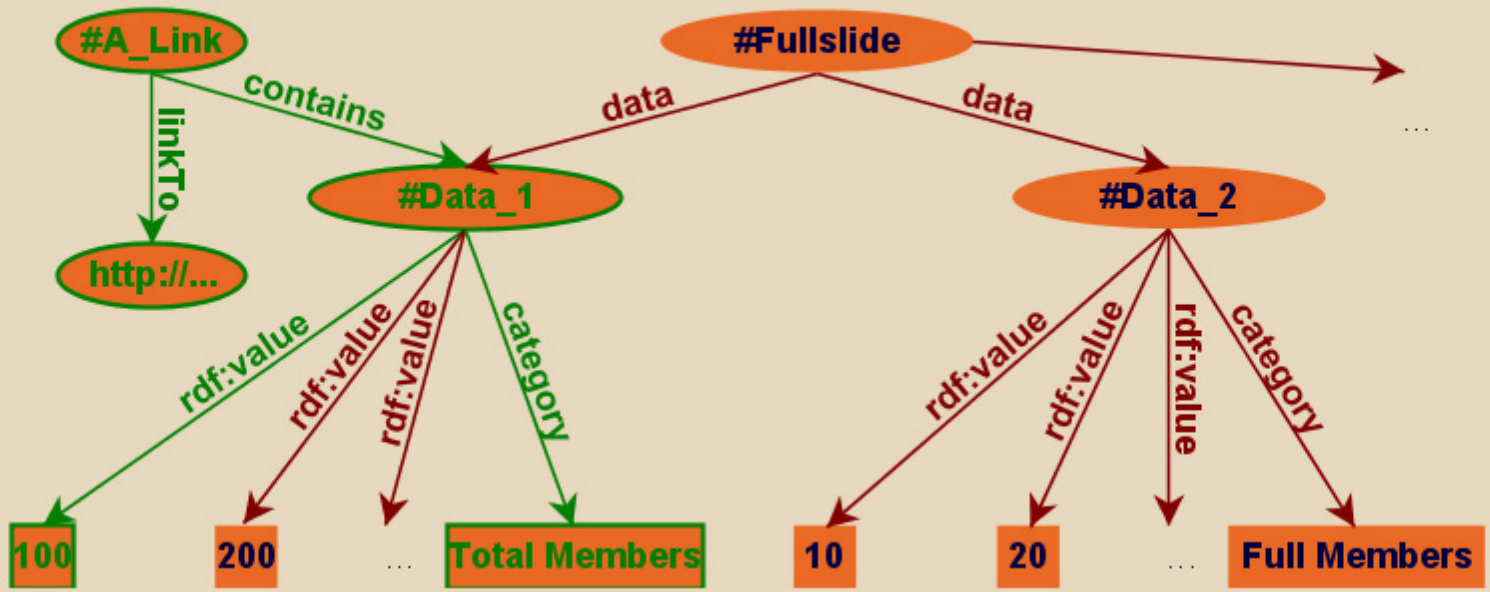


# > More Complex Example

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

Returns: `[["Total Members",100,Res],…,]`

(where `Res` is the resource for "`http://…`")



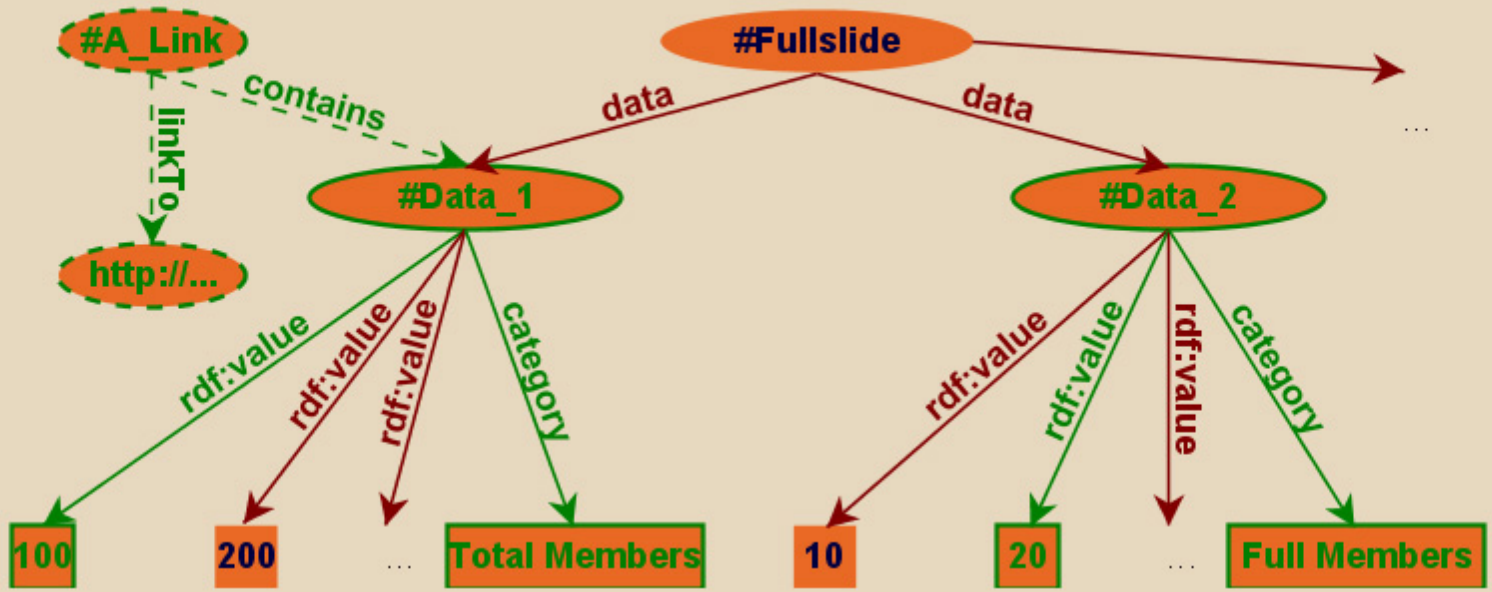
# > Optional Pattern

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

Returns: `["Total Members",100,Res],…,`

But also: `["Full Members",20, ],…,`

(note the empty return value!)



- Limit the number of returned results
- 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
- …
- Remember: SPARQL is still evolving!

- *Locally*, i.e., bound to a programming environment like RDFLib or Jena
  - details of binding to a programming language is language dependent
- *Remotely*, i.e., over the network
  - this usage is *very* important: there is growing number of RDF depositories…
  - 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
  - people may use RDF *only* through SPARQ
    - without knowing about RDF/XML, for example…
- There lots of SPARQL implementations already!

# > 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"?>
<results xmlns="http://www.w3.org/2001/sw/DataAccess/rf1/result
  <result>
    <a uri="http://work.example.org/alice/"/>
    <b uri="http://work.example.org/#name"/>
  </result>
  <result>
    <a uri="http://work.example.org/bob/"/>
    <b uri="http://work.example.org/#name"/>
  </result>
</results>
```

# Current and Future Developments

# W3C® > 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
    - …
  - intersection of SW with other technologies
    - Semantic Web Services
    - privacy policies
    - …
  - further technical development (e.g., SPARQL)
- There is a separate Working Group on “Deployment and Best Practices”

- OWL can be used for simple inferences
- Applications may want to express domain-specific knowledge, e.g.:
  - (prem-1 ∧ prem-2 ∧ …) ⇒ (concl-1 ∧ concl-2 ∧ …)
  - 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 (Horn clauses):

    ∀x,z: ((∃y: (y parent x) ∧ (y brother z)) ⇒ (z uncle x))
- Lots of research is happening to extend RDF/OWL

  (Metalog, RuleML, SWRL, cwm, …)
- W3C had a workshop in April 2005
  - the W3C way to explore possible standardization …
  - results are being worked on as we speak…

- 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?
  - etc.
- Some of the basic building blocks are available:
  - e.g., XML Signature/Encryption
- Much is missing, e.g.:
  - how to "express" trust? (E.g., trust in context.)
  - how to "name" a full graph
  - a "canonical" form of triplets (in RDF/XML or other)
    - necessary for unambiguous signatures
  - exhaustive tests for inference engines
  - protocols to check, for example, a signature
- It is on the "future" stack of W3C …

# > A Number of Other Issues…

- Lot of R&D is going on:
  - improve the inference algorithms and implementations
  - improve scalability, reasoning with OWL Full
  - temporal & spatial reasoning, fuzzy logic
  - better modularization (import or refer to *part of* ontologies)
  - procedural attachments
  - open world and non-unique-name assumptions; in OWL:
    - if something cannot be proved, it might still be true
    - two individuals with different names might be identical

    it is o.k. on the Web, but might be a problem for applications
  - …
- This mostly happens outside of W3C, though
  - W3C is not a research entity…

# Available Documents, Tools

# W3C® > Available Specifications: Primers

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

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

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

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

# W3C® > Available Specifications: RDF

**RDF: Concepts and Abstract Syntax**

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

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

# W3C® > Available Specifications: Ontology

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

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

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

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

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

# W3C® > Some Books

- M. Dertouzos: The Unfinished Revolution (1995)
  - an early “vision” book (not only on the Semantic Web)
- T. Berners-Lee: Weaving the Web (1999)
  - another “vision” book
- 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)
- A. Gómez-Pérez, M. Fernández-López, O. Corcho: Ontological Engineering (2004)
- …

# W3C® > Further Information

- Bristol University
  - Dave Beckett's Resources
  - 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,…
- WSIndex
  - Web Services & Semantic Web Index

# W3C® > SWBP Working Group Documents

- A separate Working Group at W3C
- SWBP&D Working Group's Documents, e.g.,
  - "Defining N-ary relations"
  - "Representing Classes As Property Values"
  - Semantic Web Tutorials
  - "XML Schema Datatypes in RDF and OWL"
  - "RDF/A" (RDF in XHTML2)
  - "Ontology Driven Architectures in Software Engineering"
  - "Managing a Vocabulary for the Semantic Web"
  - "XML Schema Datatypes in RDF and OWL"
  - …

# > Further Information (cont)

- Full, interactive view of the RDFS and OWL definitions
  - requires an SVG client
- References on Description logic:
  - online courses
  - a general introduction
- "Ontology Development 101"
- OWL Reasoning Examples
- *Lots* of papers at WWW2004, and WWW2005

**Semantic Web Interest Group**

a forum for discussions on applications

URI: http://lists.w3.org/Archives/Public/semantic-web/

**RDF Logic**

public (archived) mailing list for technical discussions

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

# W3C® > Some Tools

**(Graphical) Editors**

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

**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:
  - WonderWeb
  - Pellet

**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

# Some Application Examples

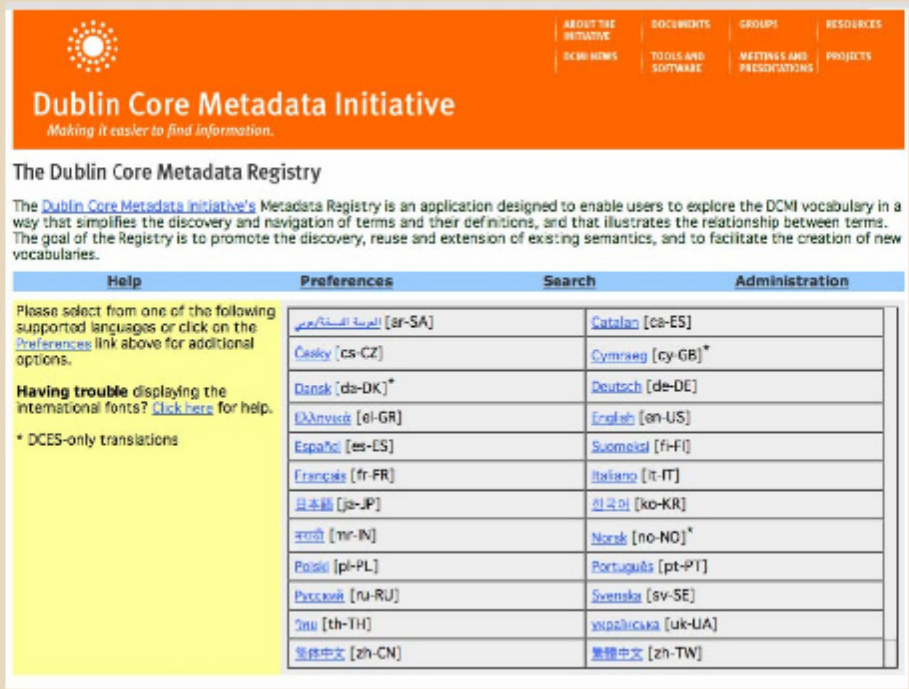
- Large number of applications emerge
  - first applications were RDF only
  - but recent ones use ontologies, too
    - huge number of ontologies exist already, with proprietary formats
    - converting them to RDF/OWL will be a major task

      (but there are converters)
    - but it will be worth it!
- See, for example, on WSIndex…
  - portal on "Web Services and Semantic Web Resources"

    with a separate page for SW applications
- … or the SW Technology Conference
  - *not* a scientific conference, but commercial people making money!

## Dublin Core

- vocabularies for distributed Digital Libraries
- one of the first metadata vocabularies in RDF
- extensions exist, e.g., PRISM that includes digital right tracking



# W3C® > SW Application Examples (cont)

## Data integration

- achieve semantic integration of corporate resources or different databases
- RDF/RDFS/OWL based vocabularies as an “interlingua” among system components
- early experimentation at Boeing (see, e.g., a WWW11 paper)
- similar approaches: Sculpteur project, MITRE Corp., MuseoSuomi, …
- there are companies specializing in the area

## Oracle's Network Data Model

- an RDF data model to store RDF statements
- Java Ntriple2NDM converter for loading existing RDF data
- an **`RDF_MATCH`** function which can be used in SQL to find graph patterns (similar to SPARQL)
- will be release as part of Oracle Database 10.2 later this year

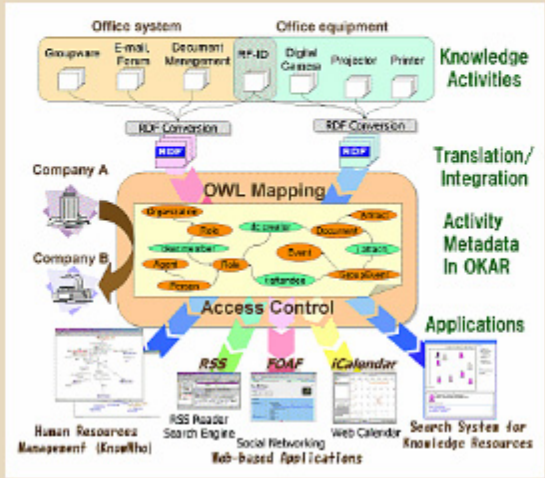
# > SW Application Examples (cont)

## Sun's SwordFish

- Sun provides assisted support for its products, handbooks, etc
- public queries go through an internal RDF engine for, eg:
  - White Papers collection
  - System Handbooks collection

## Fujitsu's and Ricoh's OKAR

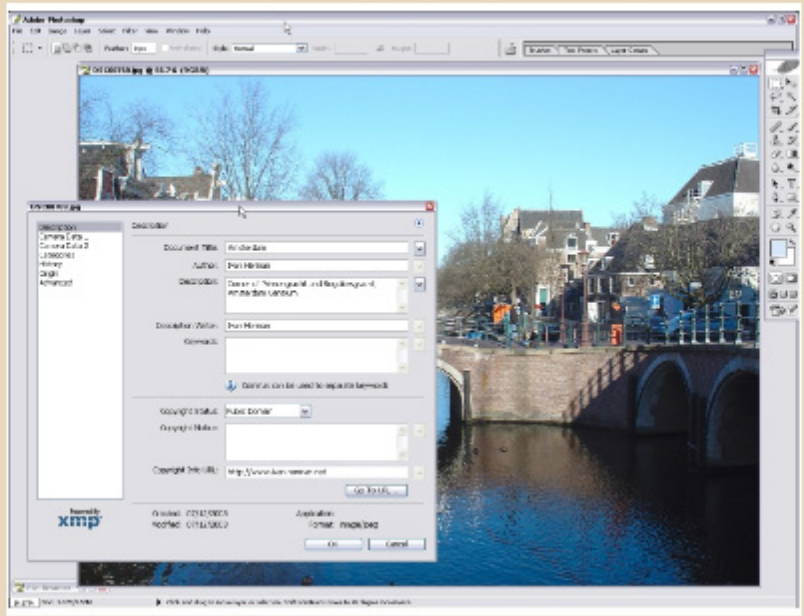
- management of office information, projects, personal skills, calendars, …
- e.g., "find me a person with a specific skill"



# W3C® > SW Application Examples (cont)

## XMP

- Adobe's tool to add RDF-based metadata to *all* their file formats
  - used for more effective organization
  - supported in Adobe Creative Suite (over 700K desktops!)
  - support from 30+ major asset management vendors
- the tool is available for all!



# W3C® > SW Application Examples (cont)

## IBM – Life Sciences and Semantic Web

- IBM Internet Technology Group
  - focusing on general infrastructure for Semantic Web applications
- develop user-centered tools
  - power of Semantic Web technologies, but hide the underlying complexity
- 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

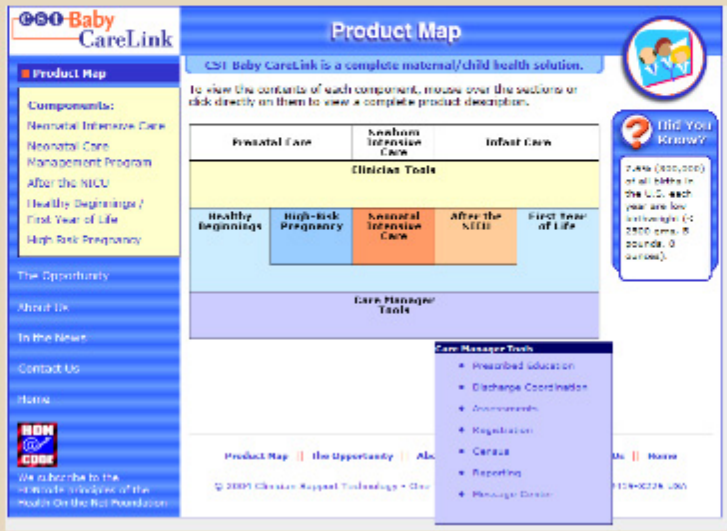
## Open Medical Ontologies

- an umbrella web site for ontologies in the biological and medical domains
- e.g., Gene Ontology (also available in OWL)

# W3C > SW Application Examples (cont)

## 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
  - 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!*



**These slides are at:**
http://www.w3.org/2005/Talks/0524-Amsterdam-IH/

**Semantic Web homepage**
http://www.w3.org/2001/sw/

**More information about W3C:**
http://www.w3.org/

**Mail me:**
ivan@w3.org