Introduction to the Semantic Web

Ivan Herman, World Wide Web Consortium

WWW2006, Edinburgh, UK, 2006-05-24


PREMIER

Introduction to the Semantic Web

Slides of the tutorial given at the WWW2006 Conference, Edinburgh, Scotland, United Kingdom, on the 24th of May, 2006.

Introduction

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 information 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*
    - e.g., if you search in for “yacht racing”, the America’s Cup will *not* be found
  - *adding (maybe application specific) descriptions to resources should improve this*
- Search can also be very application–dependent (digital libraries, specialized knowledge bases, …)

Example: Automatic Airline Reservation

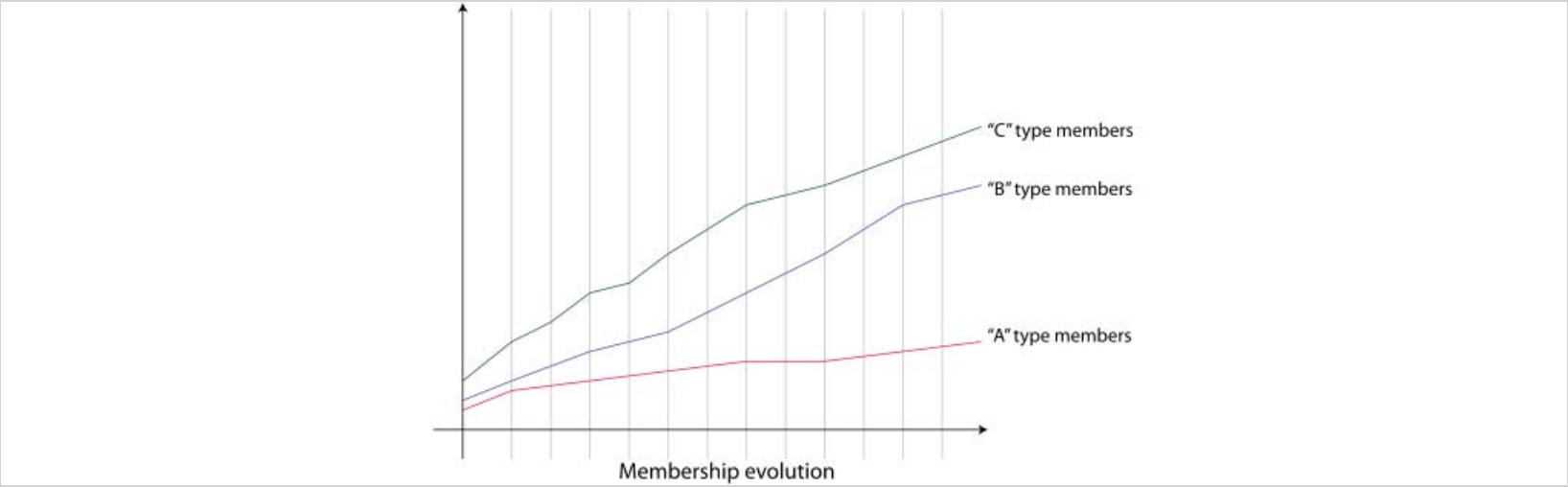
- Your automatic airline reservation
  - *knows about your preferences*
  - *builds up knowledge base using your past*
  - *can combine the local knowledge with remote services:*
    - airline preferences
    - dietary requirements
    - calendaring
    - etc
- It communicates with *remote* information (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: Image Annotation

- Task: convey the meaning of a figure through text (important for accessibility)
  - *add (meta)data to the image describing the content to let a tool produce some simple output using the metadata*



What Is Needed?

- (Some) data should be available for machines for further processing
- Data should be possibly combined, connected, merged 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*
  - *define common vocabularies: RDFS, OWL, SKOS*
  - *reasoning logics: OWL, Rules*
- *The "Semantic Web" is an extension of the current Web, providing an infrastructure for the integration of data on the Web*

Basic RDF

RDF Triples

- We said “connecting” data…
- But a simple connection is not enough… it should be named somehow
  - *a connection from “me” to my calendar is not the same as the connection from “me” to my CV (even if all of these are on the Web)*
  - *the first connection should somehow say “myCalendar”', the second “myCV”*
- Hence the RDF Triples: a *labelled connection between two resources*

RDF Triples (cont.)

- An RDF Triple (s,p,o) is such that:
  - *"s", "p" are URI-s, ie, resources on the Web; "o" is a URI or a literal*
  - *conceptually: "p" connects, or relates the "s" and "o"*
  - *note that we use URI-s for naming: i.e., we can use* **`http://www.example.org/myCalendar`**
  - *here is the complete triple:*

```
(http://www.ivan-herman.net, http://…/myCalendar, http://…/calendar)
```

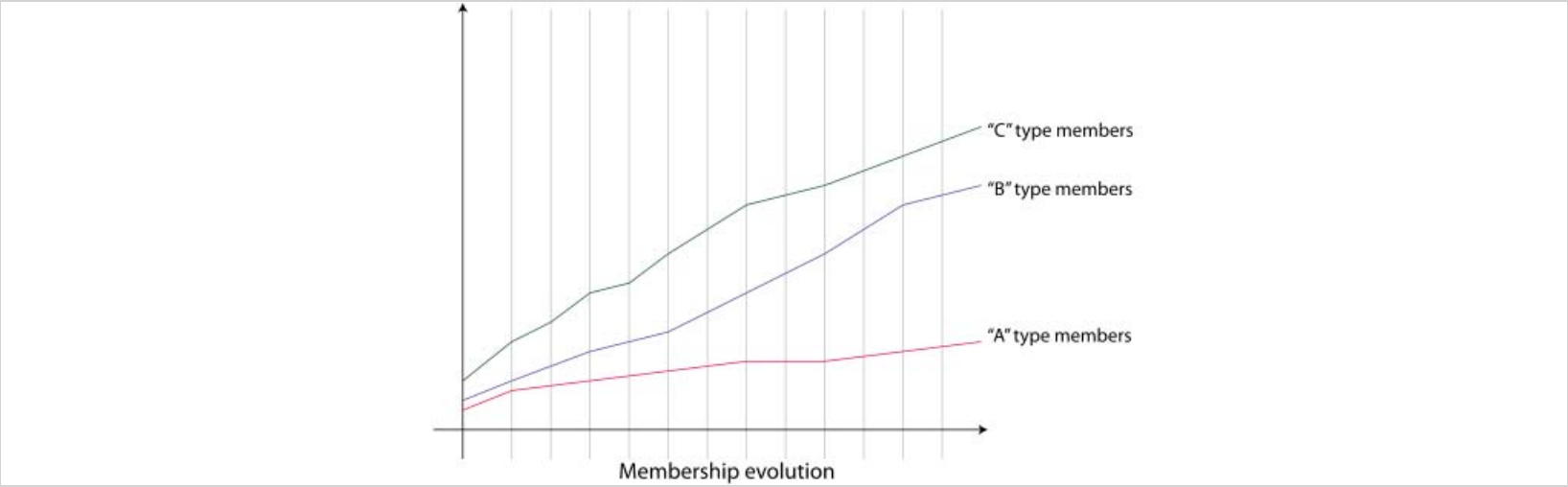
- *RDF* is a general model for such triples
  - *… with machine readable formats (RDF/XML, Turtle, n3, RXR, …)*

RDF Triples (cont.)

- RDF Triples are also referred to as *"triplets"*, or *"statement"*
- The s, p, o resources are also referred to as *"subject"*, *"predicate"*, *"object"*, or *"subject"*, *"property"*, *"object"*
- Resources can use *any* URI; i.e., it can denote an element *within* an XML file on the Web, not only a "full" resource, e.g.:
  - ***http://www.example.org/file.xml#xpointer(id('calendar'))***
  - ***http://www.example.org/file.html#calendar***

An Example for URI Usage

- If the figure is in SVG (i.e., XML) then all elements can be addressed by a URI!



Possible Statements Example:

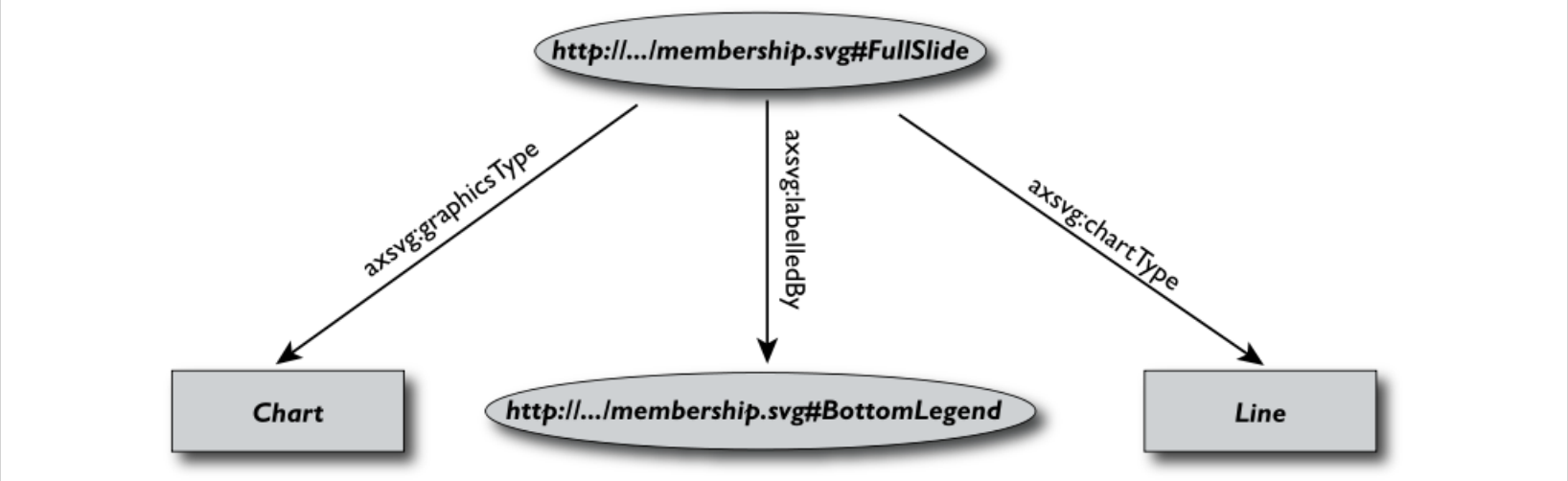
- In the annotation example:
  - *“the type of the full slide is a chart, and the chart type is «line»”*
  - *“the chart is labeled with an (SVG) text element”*
  - *“the legend is also a hyperlink”*
  - *“the target of the hyperlink is «URI»”*
  - *“the full slide consists of the legend, axes, and data lines”*
  - *“the data lines describe «A», «B», and «C» type members”*
- The second statement can be something like:

```
(URI For Slide, URI for Predicate, URI for SVG Text Element)
```

RDF is a Graph

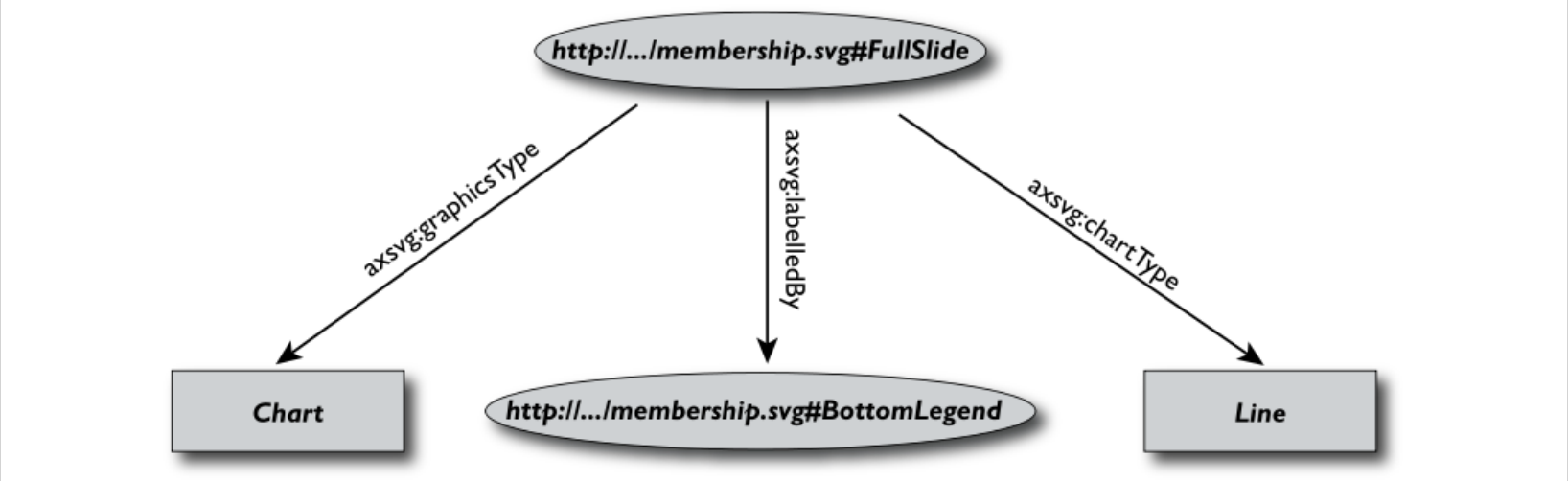
- An (s,p,o) triple can be viewed as a labeled edge in a graph
  - *i.e., a set of RDF statements is a directed, labeled graph*
    - both “objects” and “subjects” are the graph nodes
    - “properties” are the edges
- One should “think” in terms of graphs; XML or Turtle syntax are only the tools for practical usage!
- RDF authoring tools may work with graphs, too (XML or Turtle is done “behind the scenes”)

A Simple RDF Example (in RDF/XML)




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

A Simple RDF Example (in Turtle)




```
<http://.../membership.svg#FullSlide>
    axsvg:graphicsType "Chart";
    axsvg:labelledBy <http://...#BottomLegend>;
    axsvg:chartType "Line".
```

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*
  - *URI-s make it possible to link (via properties) data with one another*
- *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 in many applications*

What Merge Can Do…

See the “tabulator” example…

RDF in Programming Practice

- For example, using Java+Jena (HP's Bristol Lab):
  - *a "Model" object is created*
  - *the RDF file is parsed and results stored in the Model*
  - *the Model 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 Python, PHP, etc. (see later)

Jena Example

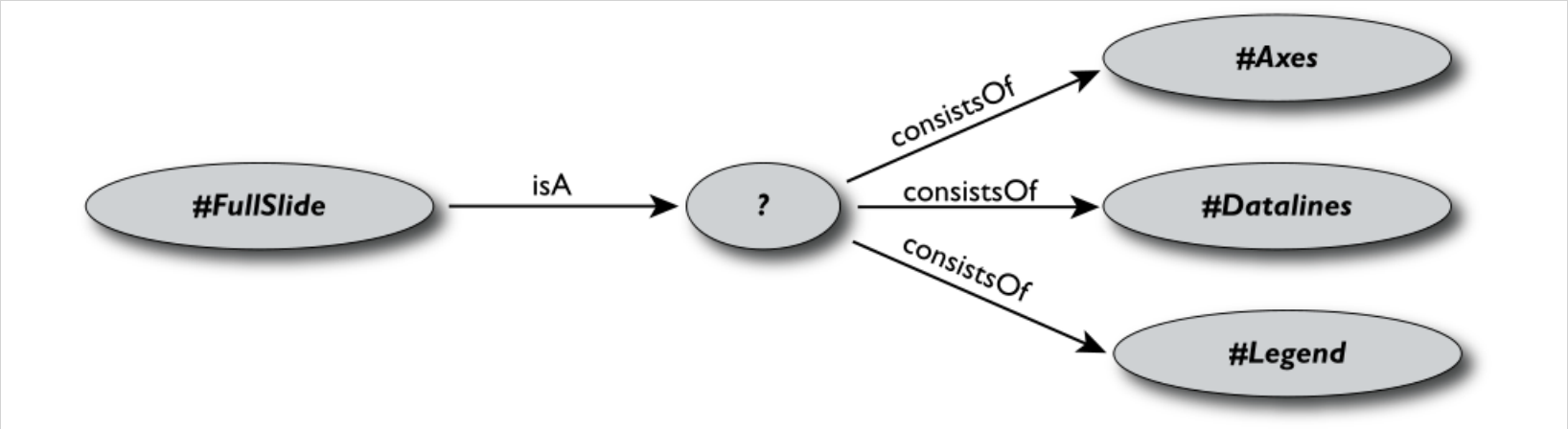
```
 // create a model
Model model=new ModelMem();
Resource subject=model.createResource("URI_of_Subject")
// 'in' refers to the input file
model.read(new InputStreamReader(in));
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
   st = iter.next();
   p = st.getProperty();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

Merge in Practice

- Environments merge graphs automatically
  - *e.g., in Jena, the Model can load several files*
  - *the load merges the new statements automatically*

“Internal” 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»?



One Solution: Define Extra URI-s

- Give an id with **rdf:ID** (essentially, defining a URI)

```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:isA rdf:resource="#Thing"/>
</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 file
- Identical to the **id** in HTML, SVG, … (i.e., it can be referred to with regular URI-s from the outside)
- Note: this is an RDF/XML feature only!

Blank Nodes

- Use an *internal* identifier

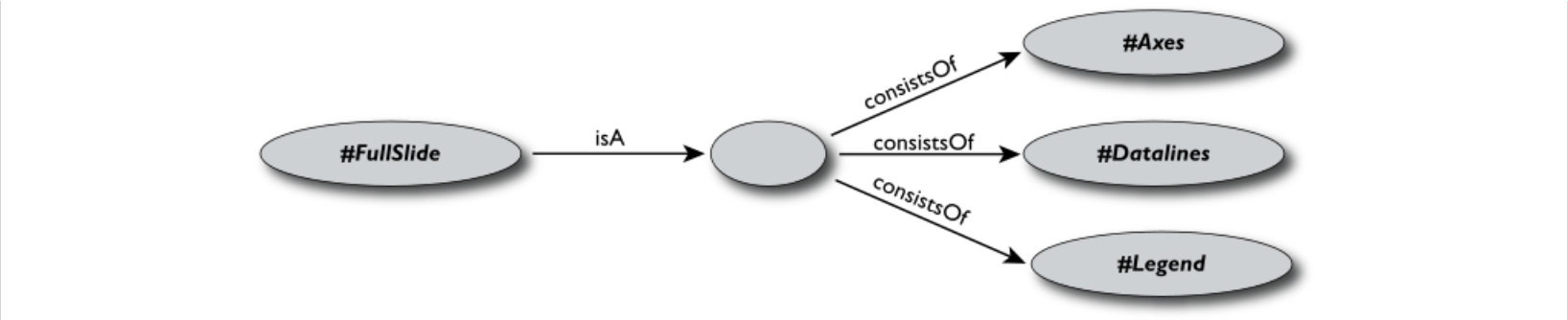
```
<rdf:Description rdf:about="#FullSlide">
   <axsvg:isA rdf:nodeID="A234"/>
</rdf:Description>
<rdf:Description rdf:nodeID="A234">
   <axsvg:consistsOf rdf:resource="#Axes"/>
</rdf:Description>
:FullSlide axsvg:isA _:A234.
_:A234 axsvg:consistsOf :Axes".
```

- **A234** is *invisible* from outside the file (*it is not a "real" URI!*)
  - *it is an internal identifier for a resource*

Blank Nodes: the System Can Also Do It

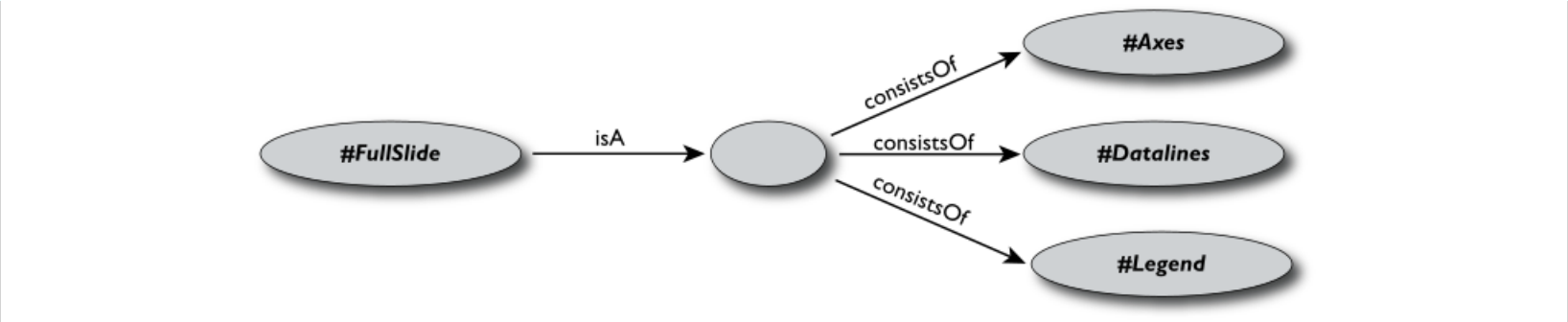
- Let the system create a **nodeID** internally (you do not really care about the name…)

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



Same in Turtle

```
:FullSlide axsvg:isA [
    axsvg:consistsOf :Axes;
    …
].
```



Blank Nodes: Some More Remarks

- Blank nodes require attention when merging
  - *blanks nodes with identical nodeID-s in different graphs are different*
  - *the implementation must be be careful with its naming schemes when merging*
- From a logic point of view, blank nodes represent an “existential” statement (“there is a resource such that…”)

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`, …
  - `myCV`, `myCalendar`, …
  - *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…*

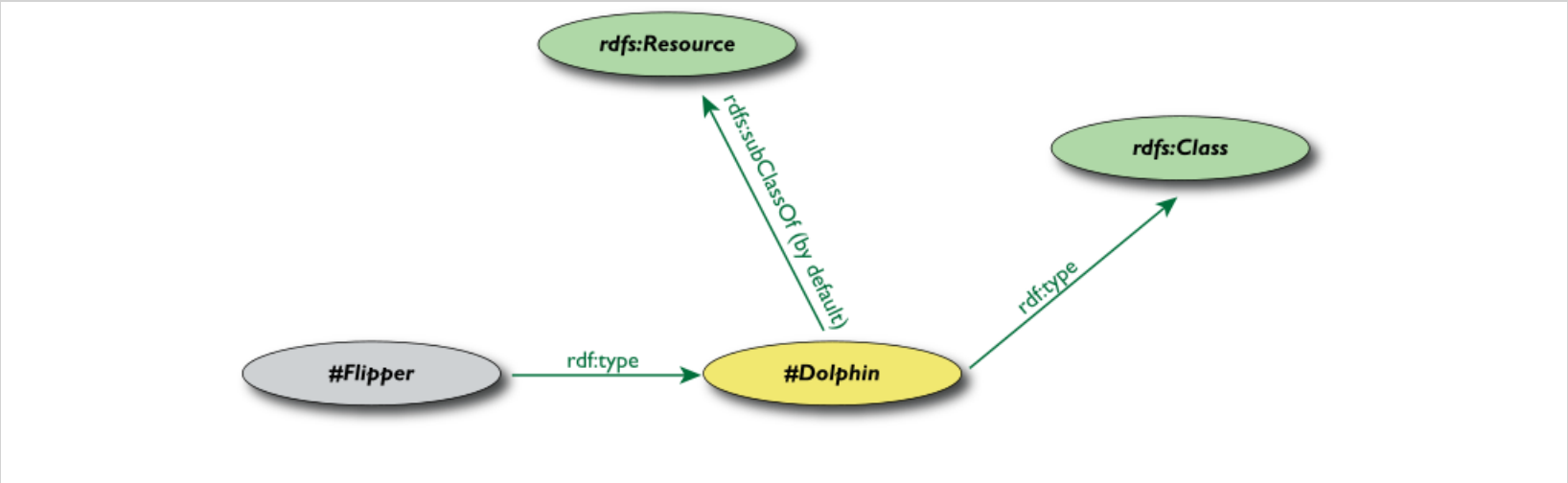
Classes, Resources, …

- 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
  - *(these are all special URI-s, we just use the namespace abbreviation)*

Schema Example in RDF/XML

- The schema (“application’s data types”):

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

- The RDF data on a specific animal (“using the type”):

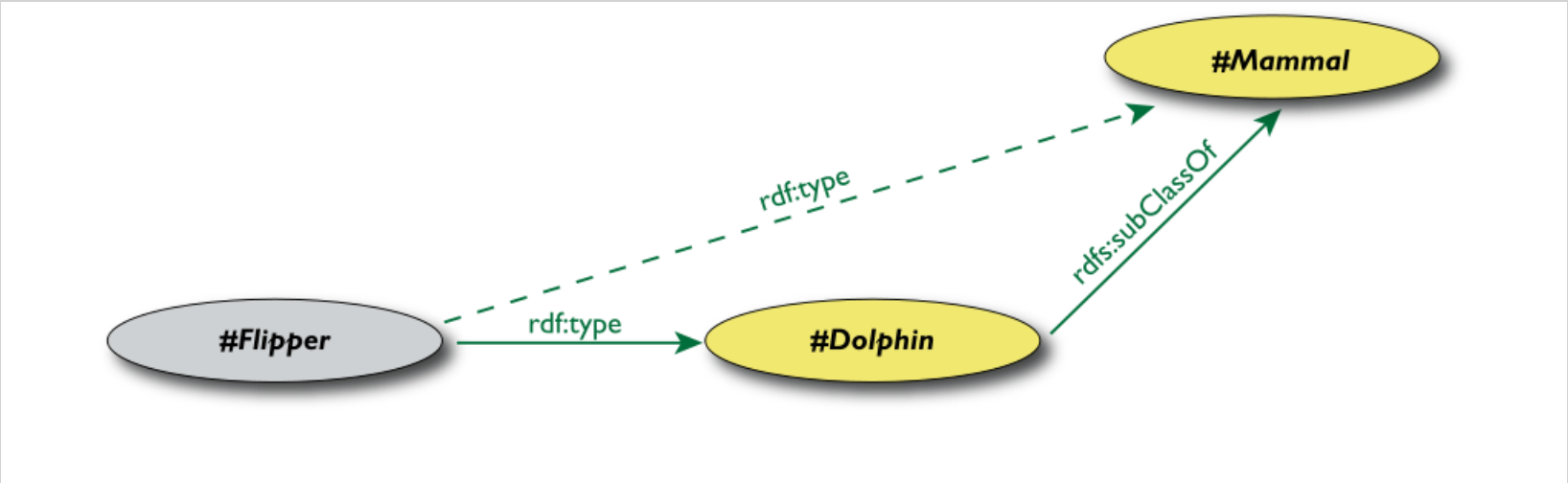
```
<rdf:Description rdf:about="#Flipper">
   <rdf:type rdf:resource="animal-schema.rdf#Dolphin"/>
</rdf:Description>
```

- In traditional knowledge representation this separation is often referred to as: “Terminological axioms” and “Assertions”

Further Remarks on Types

- 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!
- The type information may be very important for applications
  - *e.g., it may be used for a categorization of possible nodes*
  - *probably the most frequently used* `rdf` *predicate…*

Inferred Properties



- • *(#Flipper rdf:type #Mammal)*
- is *not* in the original RDF data…
- …but can be *inferred* from the RDFS rules
- Better RDF environments 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 this triplet"*
  - *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

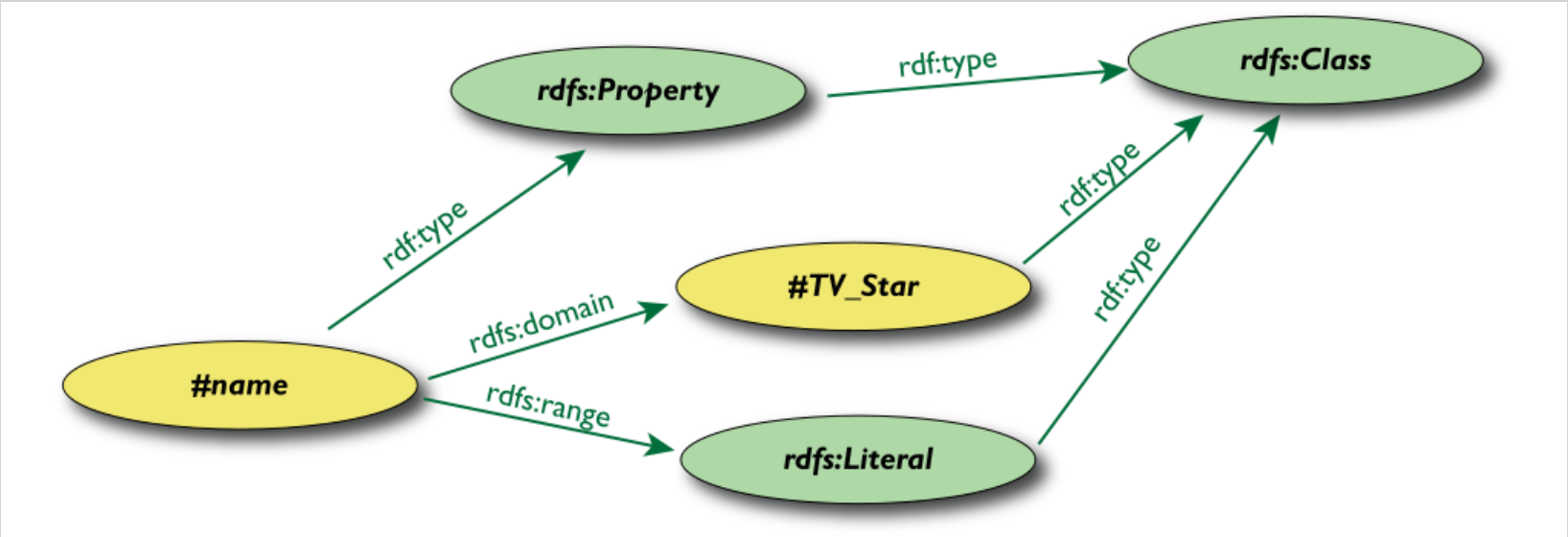
- Property is a special class (`rdf:Property`)
  - *properties are also resources identified by URI-s*
- Properties are constrained by their range and domain
  - *i.e., what individuals can serve as object and subject*
- There is also a possibility for a “sub-property”
  - *all resources bound by the “sub” are also bound by the other*

Properties (cont.)

- Properties are also resources (named via URI–s)…
- So properties of properties can be expressed as… RDF properties
  - *this twists your mind a bit, but you can 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 within the RDF(S) framework *what* literals can be used

Property Specification Serialized

In XML/RDF:

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

In Turtle:

```
:name
  rdf:type   rdf:Property;
  rdf:domain :TV_Actor;
  rdf:range  rdfs:Literal.
```

Literals

- Literals may have a data type
  - *floats, integers, booleans, etc, defined in XML Schemas*
    - one can also define complex structures and restrictions via regular expressions, …
  - *full XML fragments*
- (Natural) language can also be specified (via `xml:lang`)

Literals Serialized

In RDF/XML

```
<rdf:Description rdf:about="#Flipper">
   <animal:is_TV_Star
      rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
         True
   </animal:is_TV_Star>
</rdf:Description/>
```

In Turtle

```
:Flipper
   animal:is_TV_Star
         "True"^^<http://www.w3.org/2001/XMLSchema#boolean>.
```

XML Literals in RDF/XML

- XML Literals
  - *makes it possible to "include" XML vocabularies into RDF:*

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

A Bit of RDFS Can Take You Far…

- Remember the power of “merge”?
- Sometimes, one or two extra RDFS statements provide the necessary glue:
  - ***foo:bar*** *is a subclass of* ***abc:efg***
  - ***qwt:xyz*** *is a subproperty of* ***klm:nop***
- by stating those (and using an RDFS aware environment) the merge becomes “complete”
- Of course, in some cases, more complex “glues” are necessary (see later…)

Some Predefined Classes (Collections, Containers)

Predefined Classes and Properties

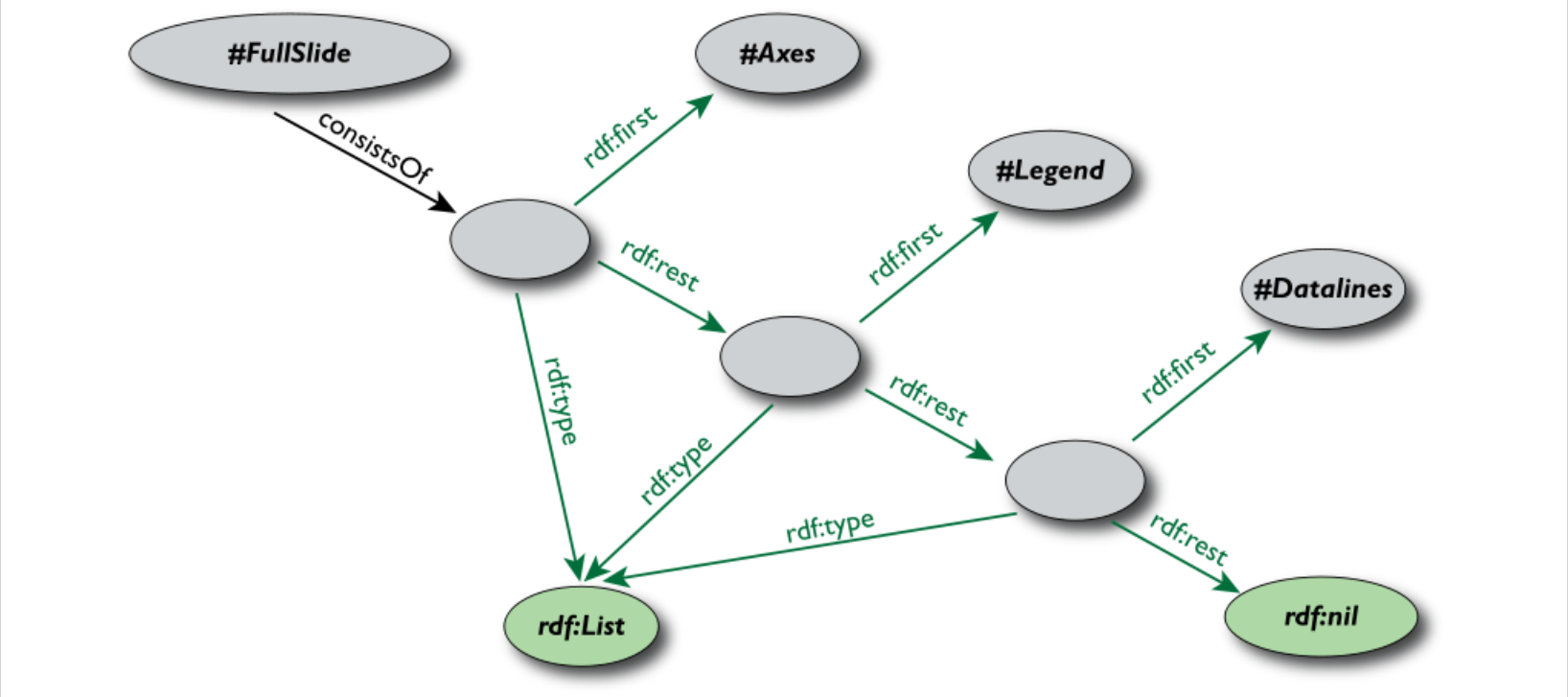
- RDF(S) has some predefined classes and properties
- They are not new “concepts” in the RDF Model, just resoruces with an agreed semantics
- Examples:
  - *collections (a.k.a. lists)*
  - *containers: sequence, bag, alternatives*
  - *reification*
  - **`rdfs:comment`**, **`rdf:seeAlso`**, **`rdf:value`**

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 and Turtle

```
<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>
:FullSlide axsvg:consistsOf (:Axes, :Legend, :Datalines).
```

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

- Using URI-s in RDF binds you automatically
- You may also 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
- 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 Extracted/Generated

- Use intelligent “scrapers” or “wrappers” to extract a structure (hence RDF) from a Web page…
  - *using conventions in, e.g., class names or header conventions like* `meta` *elements*
- … and then *generate* RDF automatically (e.g., via an XSLT script)
- Although they may not say it: this is what the “microformat” world is doing
  - *they may not extract RDF but use the data directly instead, but that depends on the application*
  - *other applications may extract it to yield RDF (e.g., RSS)*

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="http:…/dc-extract.xsl"/>
    <meta name="DC.Subject" content="Some subject"/>
    ...
  </head>
  ...
  <span class="date">2006-01-02</span>
  ...
</html>
```

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

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

```
    <dc:date>2006-01-02</dc:date>
  </rdf:Description>
```

GRDDL (cont)

- The user has to provide **`dc-extract.xsl`** and use its conventions (making use of the corresponding meta-s, class id-s, etc…)
- … but, by using the **`profile`** attribute, a client is instructed to find and run the transformation processor automatically
- A “bridge” to “microformats”
- Currently a W3C Team Submission, a Working Group has just been proposed, with a recommendation planned in the 1st Quarter of 2007

Another Future Solution: RDFa

- RDFa (formerly known as RDF/A) extends XHTML by:
  - *extending the* `link` *and* `meta` *elements (e.g., meta elements may have children, thereby adding more complex data; usable throughout the body, too)*
  - *defining general attributes to add metadata to any elements (a bit like the* `class` *in microformats, but via dedicated properties)*

RDFa (cont.)

- For example

```
<div about="http://uri.to.newsitem">
  <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>. See
  <a href="http://www.a.b.c/d.avi" rel="dcmtype:MovingImage">
  also video footage</a>…
</div>
```

- yields, by running the file through a processor:

```
<http://uri.to.newsitem>
  dc:date             "March 23, 2004";
  dc:title            "Rollers hit casino for £1.3m;
  dc:creator          "Steve Bird";
  dcmtype:MovingImage <http://www.a.b.c/d.avi>.
```

RDFa (cont.)

- Originally, RDFa was part of the XHTML2 development
- Plan is to develop it as an extra XHTML 1.X module
- It is a bit like the microformats approach but with more rigor
- It can easily be combined (i.e., used by) with GRDDL
- There is an RDFa document as well as a primer available for further reading

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

Querying RDF Graphs/Repositories

- Remember the Jena idiom:

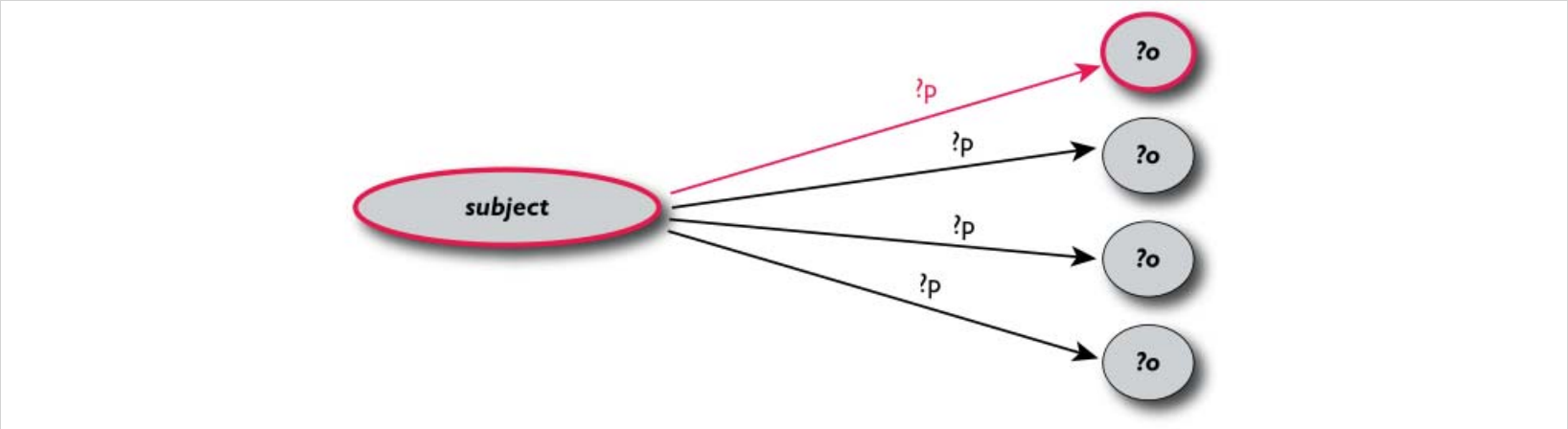
```
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty(); o = st.getObject();
    do_something(p,o);
```

- 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)

Analyze the Jena Example

```
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty(); o = st.getObject();
    do_something(p,o);
```

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



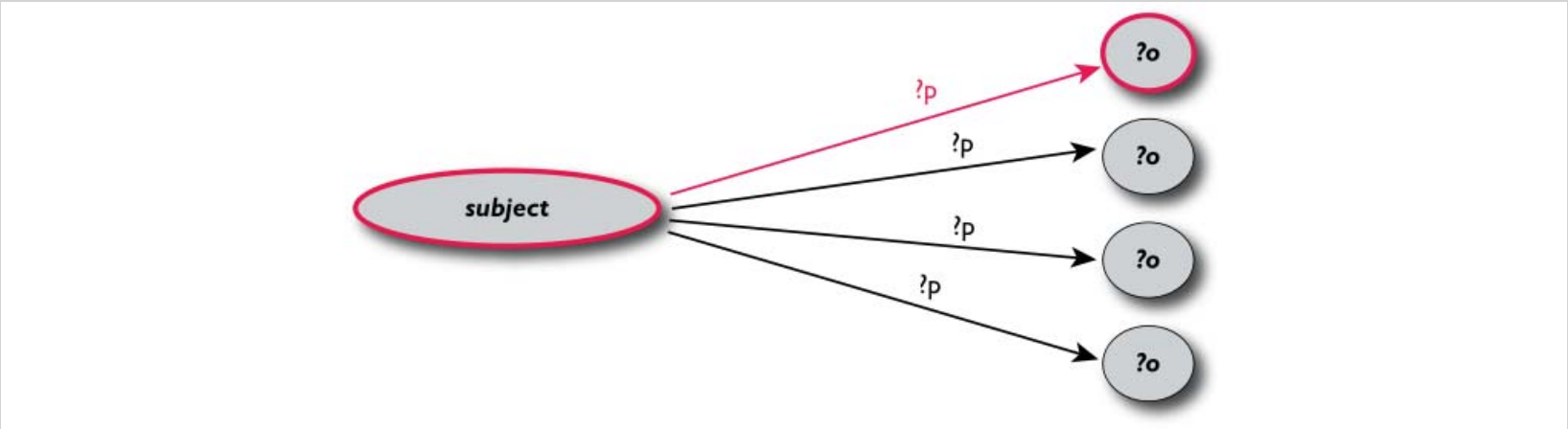
General: Graph Patterns

- The fundamental idea: generalize the approach to *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*

Our Jena Example in SPARQL

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

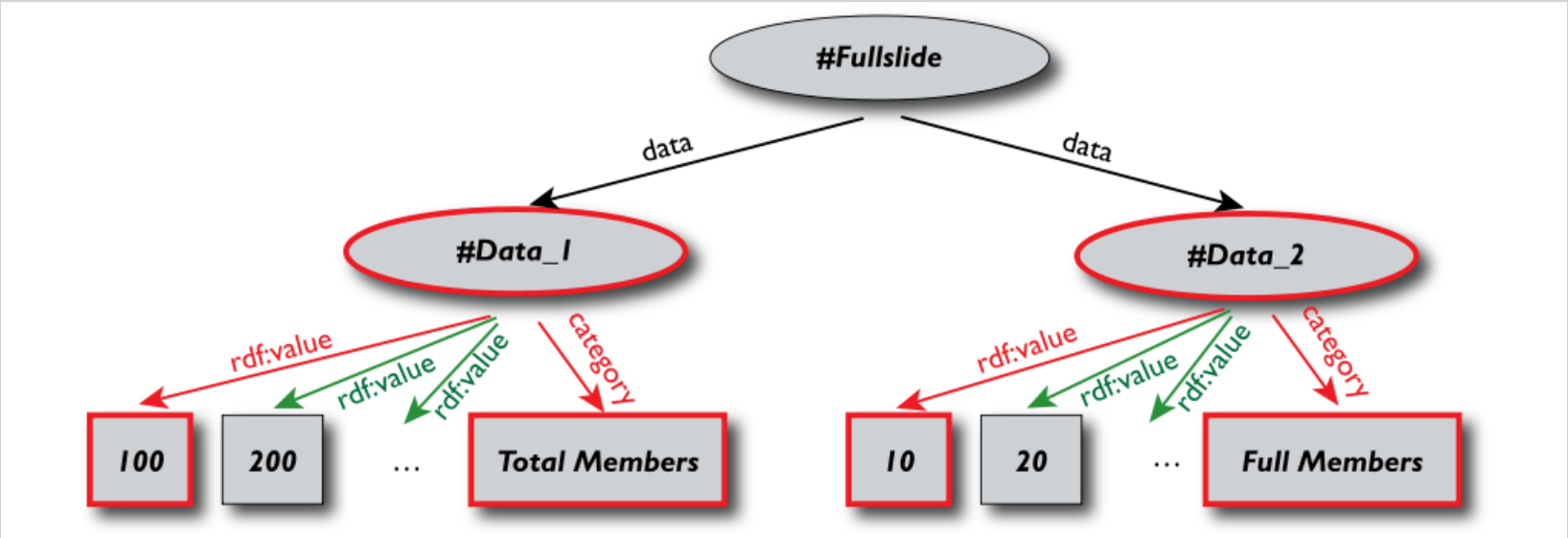
- The triplets in **WHERE** define the graph pattern, with **?p** and **?o** “unbound” symbols
- The query returns a list of matching **p,o** pairs



Simple SPARQL Example

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

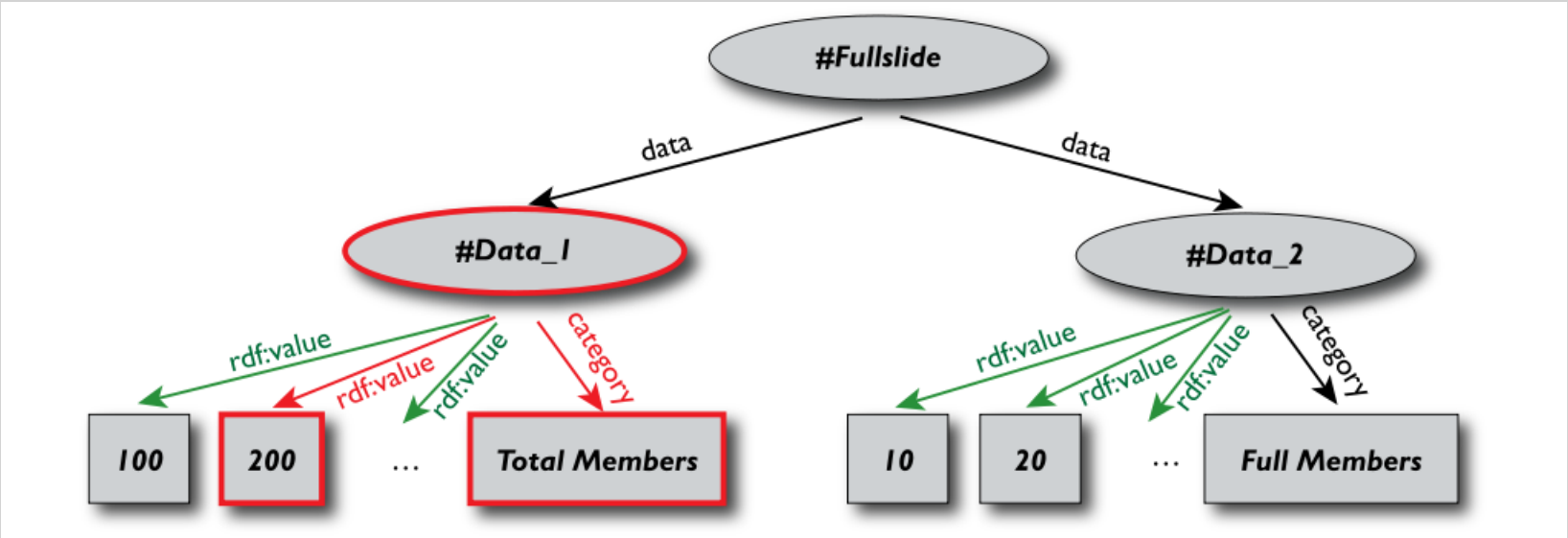
- Returns: `[["Total Members",100],["Total Members",200],…,["Full Members",10],…]`



Pattern Constraints

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

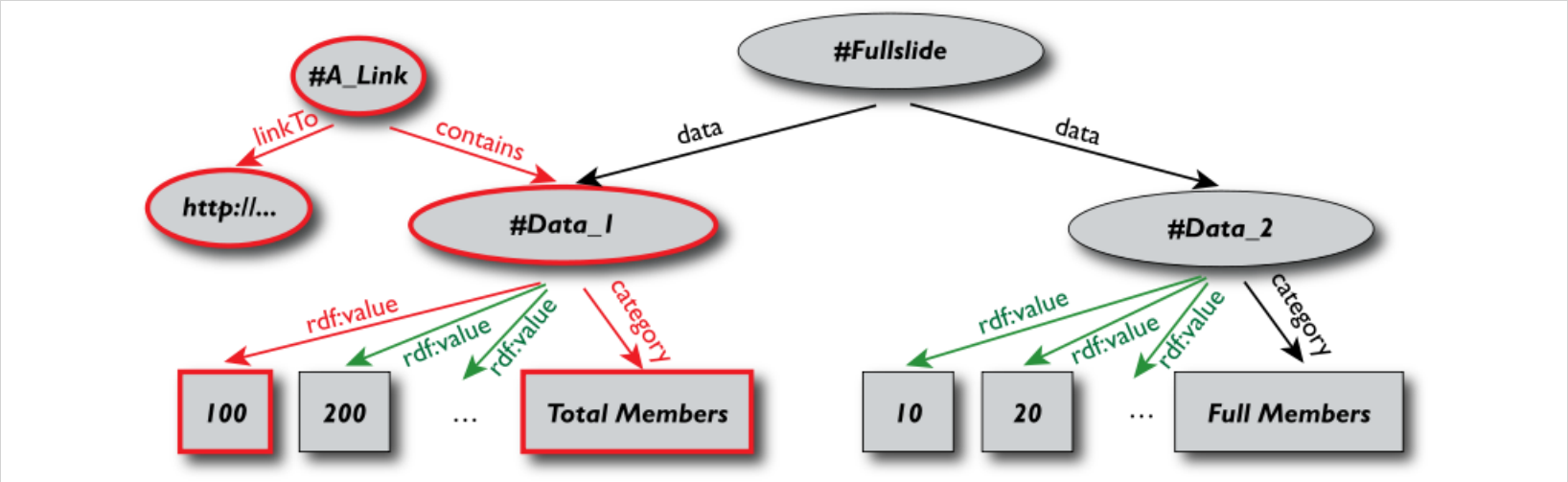
- Returns: `[["Total Members",200],…,]`
- SPARQL defines a base set of operators and functions



More Complex Example

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

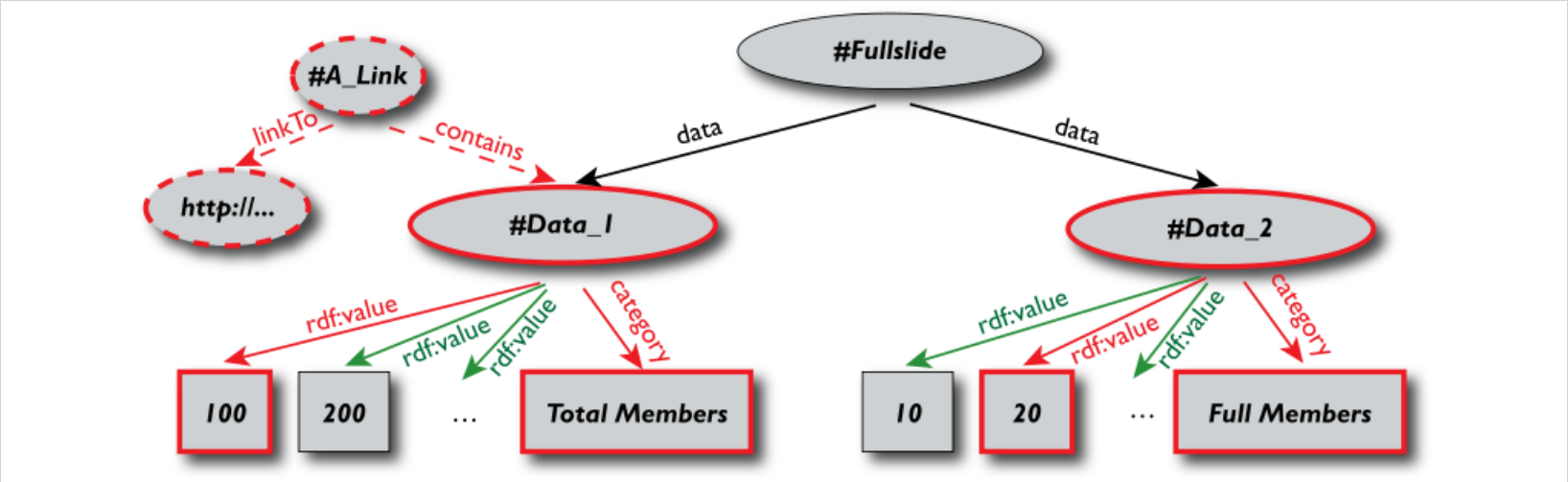
- Returns: `[["Total Members",100,Resource(http://...)],…,]`



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,Resource(http://...)], …, ["Full Members",20, ],…,`



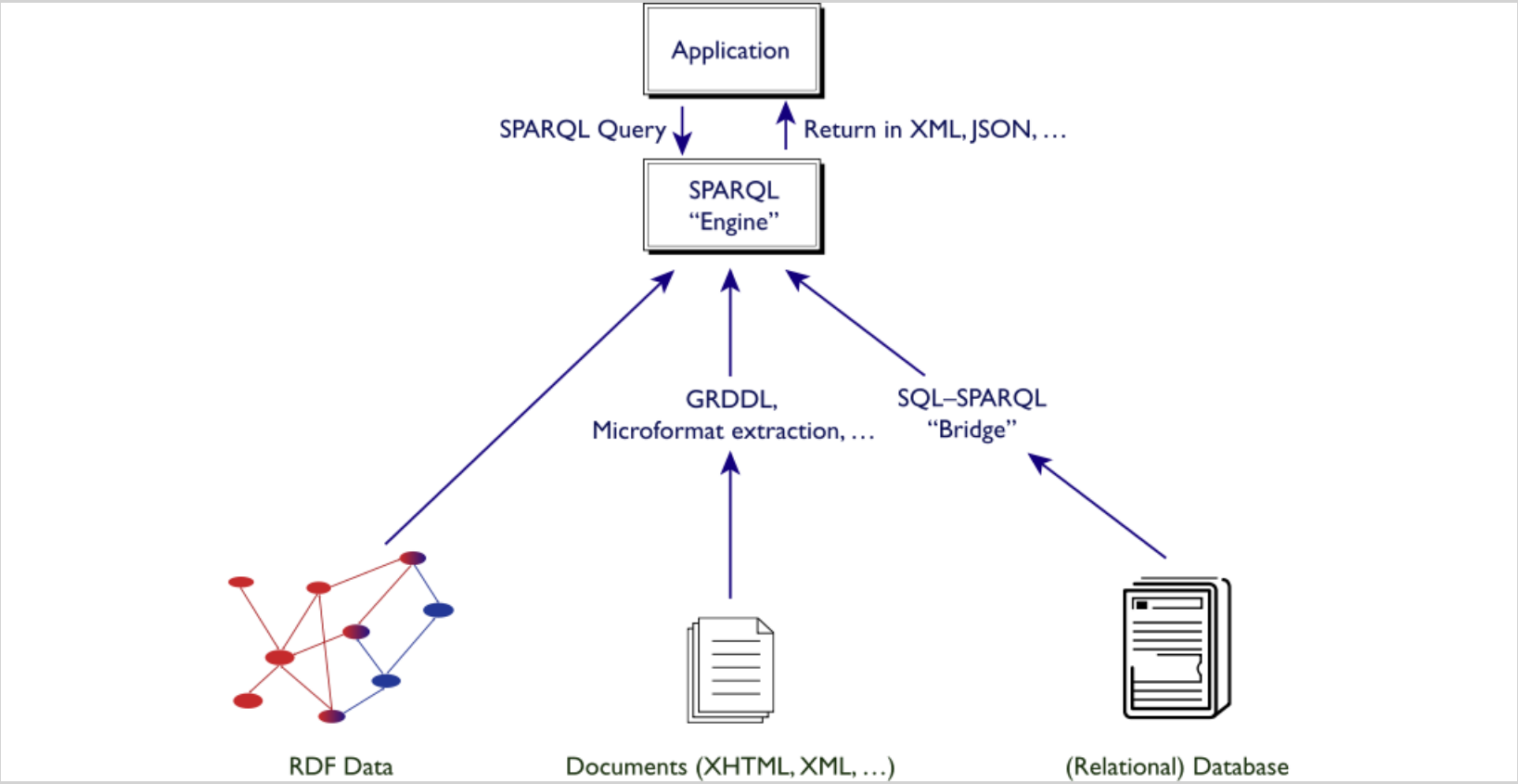
Other SPARQL Features

- Limit the number of returned results; remove duplicates, sort them,…
- Specify several data sources (via URI-s) within the query (essentially, a merge!)
- Construct a graph combining a *separate* pattern and the query results
- Use datatypes and/or language tags when matching a pattern
- SPARQL is a “Candidate Recommendation”, i.e., the technical aspects are now finalized (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 Jena
- *Remotely*, e.g., over the network or into a database
  - *separate documents define the protocol and the result format*
    - SPARQL Protocol for RDF with HTTP and SOAP bindings
    - SPARQL Results XML Format
    - there is also a JSON binding (soon a W3C note…)
- There are already a number of applications, demos, etc.,

SPARQL Usage in Practice



Programming Practice

We have seen Jena

```
 // create a model
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; it has
  - *a large number of classes/methods*
    - adding triplets to a graph, serialize it
    - comparing full RDF graphs
    - manage typed literals
    - etc.
  - *an "RDFS Reasoner"*
  - *a full SPARQL implementation*
  - *a layer (Joseki) to create a 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, includes a SPARQL engine), SWI-Prolog (Prolog), RDFLib for Python…, …*
  - *Redland: general RDF Framework, with bindings to C, C++, C#, Python, …, and with a SPARQL engine (Rasqal)*
  - *RDF storage systems: (Sesame, Kowari, Tucana, Gateway, @Semantics RDFStore, Virtuoso, 3Store, Jena's Joseki, InferEd, Oracle Database 10g, Allegro…)*
    - 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

SPARQL as the *only* interface to RDF data?

- **http://xmlarmyknife.org/api/rdf/sparql/query?query-uri=http://www.w3.org/2006/05/armyKnife.rq**

  with the query:

```
SELECT ?translator ?translationTitle ?originalTitle ?originalDate
FROM <http://…/TR_and_Translations.rdf>
WHERE {
   ?trans rdf:type trans:Translation;
          trans:translationFrom ?orig;
          trans:translator      [ contact:fullName ?translator ];
          dc:language           "fr";
          dc:title              ?translationTitle.
   ?orig  rdf:type rec:REC;
          dc:date               ?originalDate;
          dc:title              ?originalTitle.
}
ORDER BY ?translator ?originalDate
```

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"

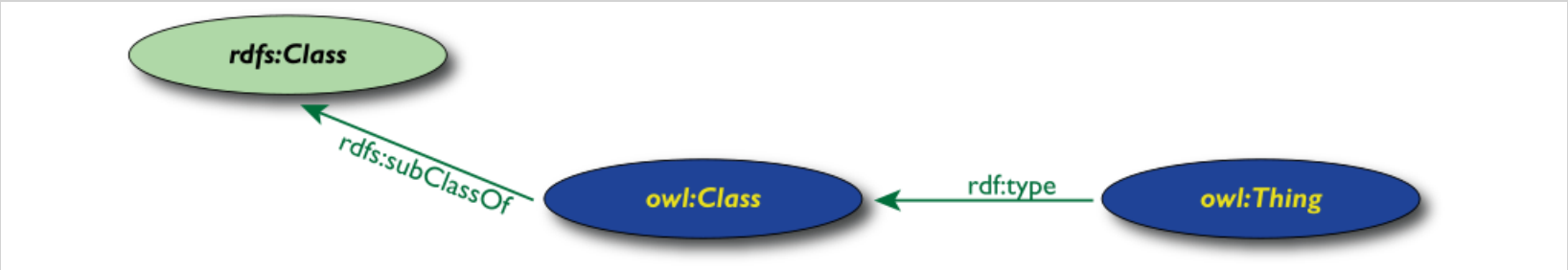
- We need a *Web Ontologies Language* to define:
  - *more on the terminology used in a specific context*
  - *more constraints on properties, logical characterization of properties*
  - *etc.*
- Language should be a compromise between
  - *rich semantics for meaningful applications*
  - *feasibility,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

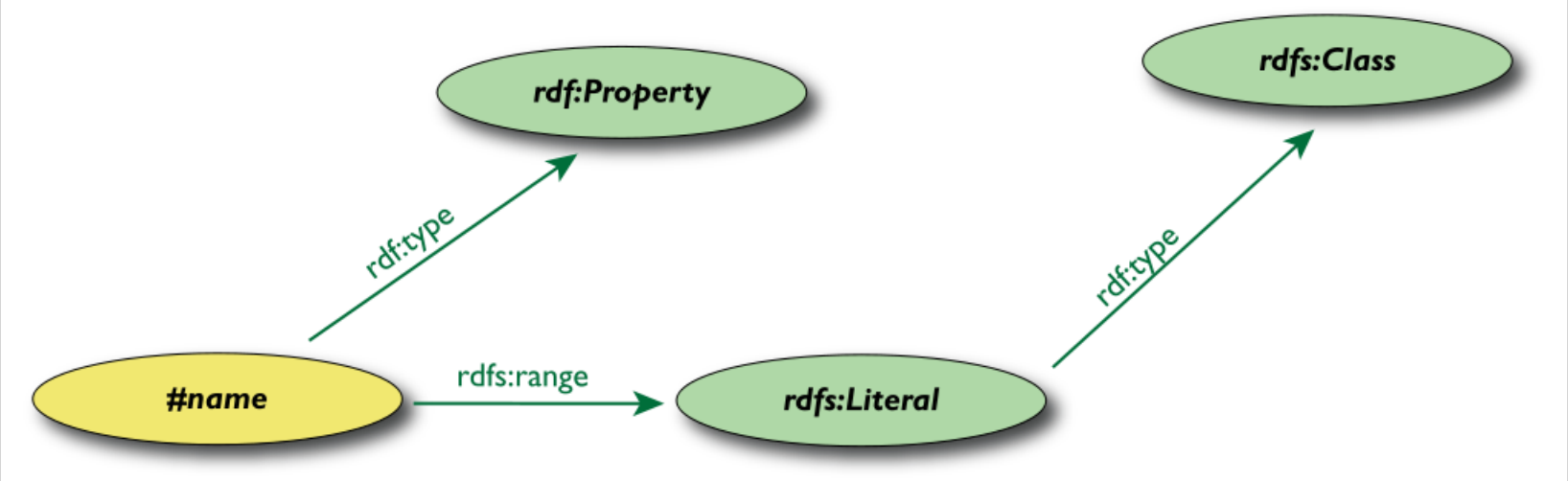
Classes in OWL

- 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*



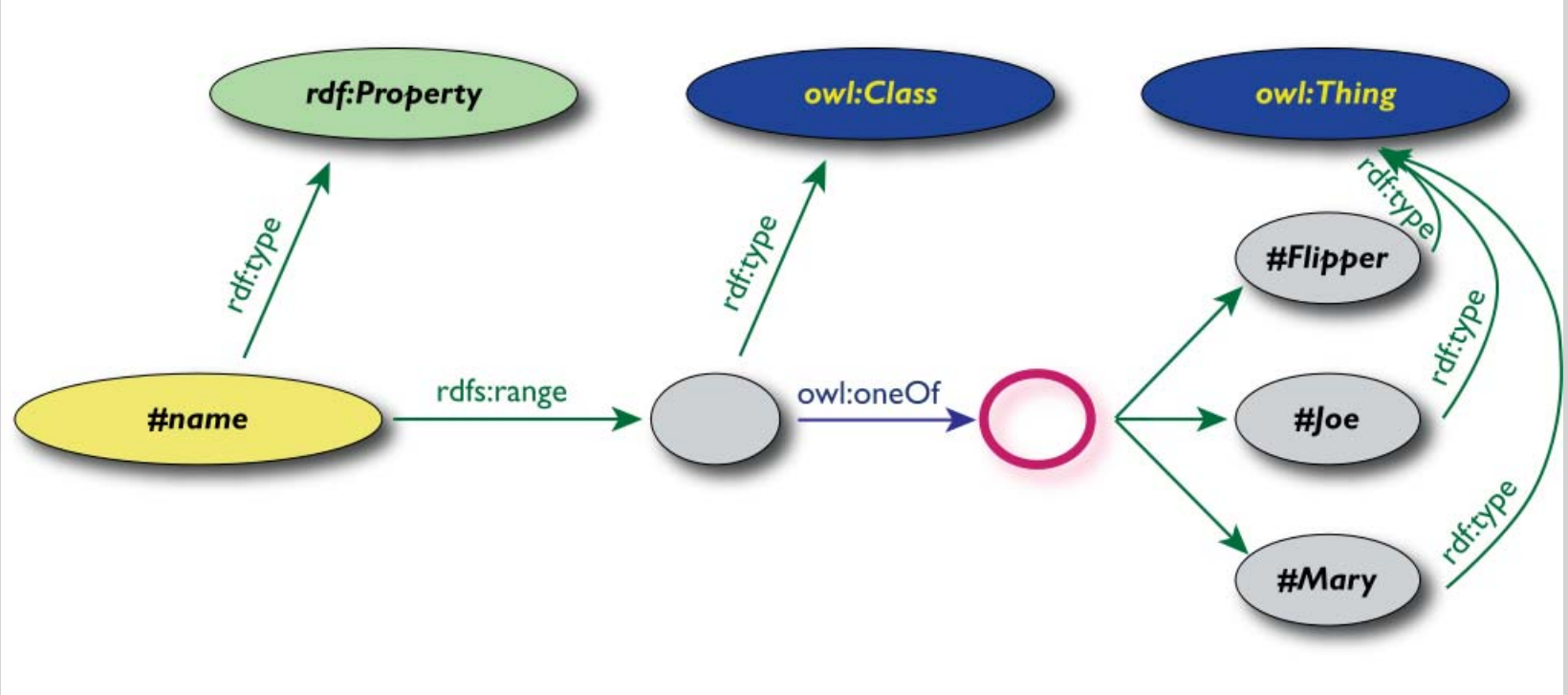
Need for Enumeration

- Remember this issue?
  - *one can use XML Schema types to define a* **`name`** *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 Serialized

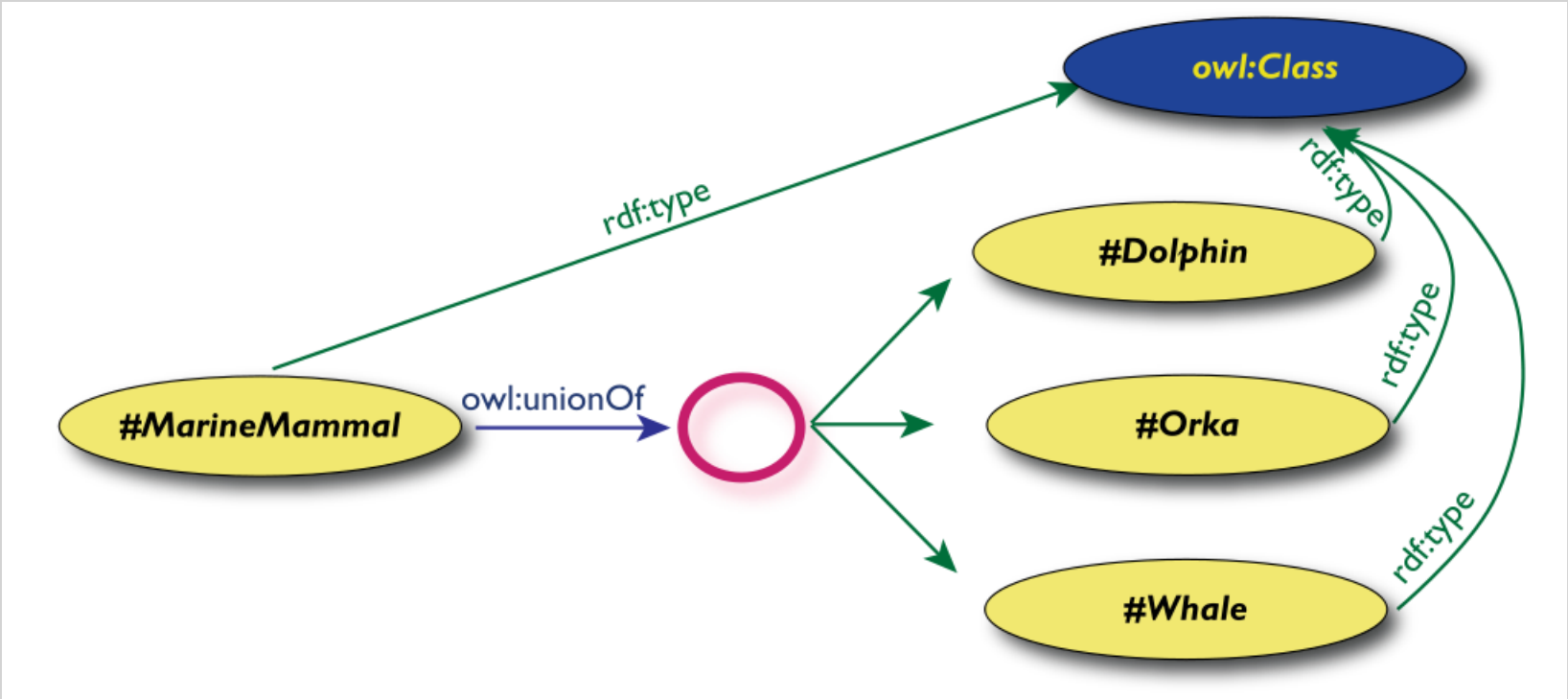
```
<rdf:Property rdf:ID="name">
   <rdf:range>
       <owl:Class>
           <owl:oneOf rdf:parseType="Collection">
               <owl:Thing rdf:ID="Flipper"/>
               <owl:Thing rdf:ID="Joe"/>
               <owl:Thing rdf:ID="Mary"/>
               …
           </owl:oneOf>
       </owl:Class>
   </rdf:range>
</rdf:Property>
:Flipper rdf:type owl:Thing.
:Joe     rdf:type owl:Thing.
:Mary    rdf:type owl:Thing.
:name rdf:type rdf:Property;
   rdf:range [
       rdf:type owl:Class;
       owl:oneOf (:Flipper, :Joe, :Mary).
```

```
].
```

- The class consists of *exactly* of those individuals

Union of Classes

- Essentially, like a set-theoretical union:



Same Serialized

```
<owl:Class rdf:ID="MarineMammal">
   <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#Dolphin"/>
       <owl:Class rdf:about="#Orca"/>
       <owl:Class rdf:about="#Whale"/>
       …
   </owl:unionOf>
</owl:Class>
:Dolphin  rdf:type owl:Class.
:Orca     rdf:type owl:Class.
:Whale    rdf:type owl:Class.
:MarineMammal rdf:type owlClass;
   owl:unionOf (:Dolphin, :Orca, :Whale).
```

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

Property Restrictions

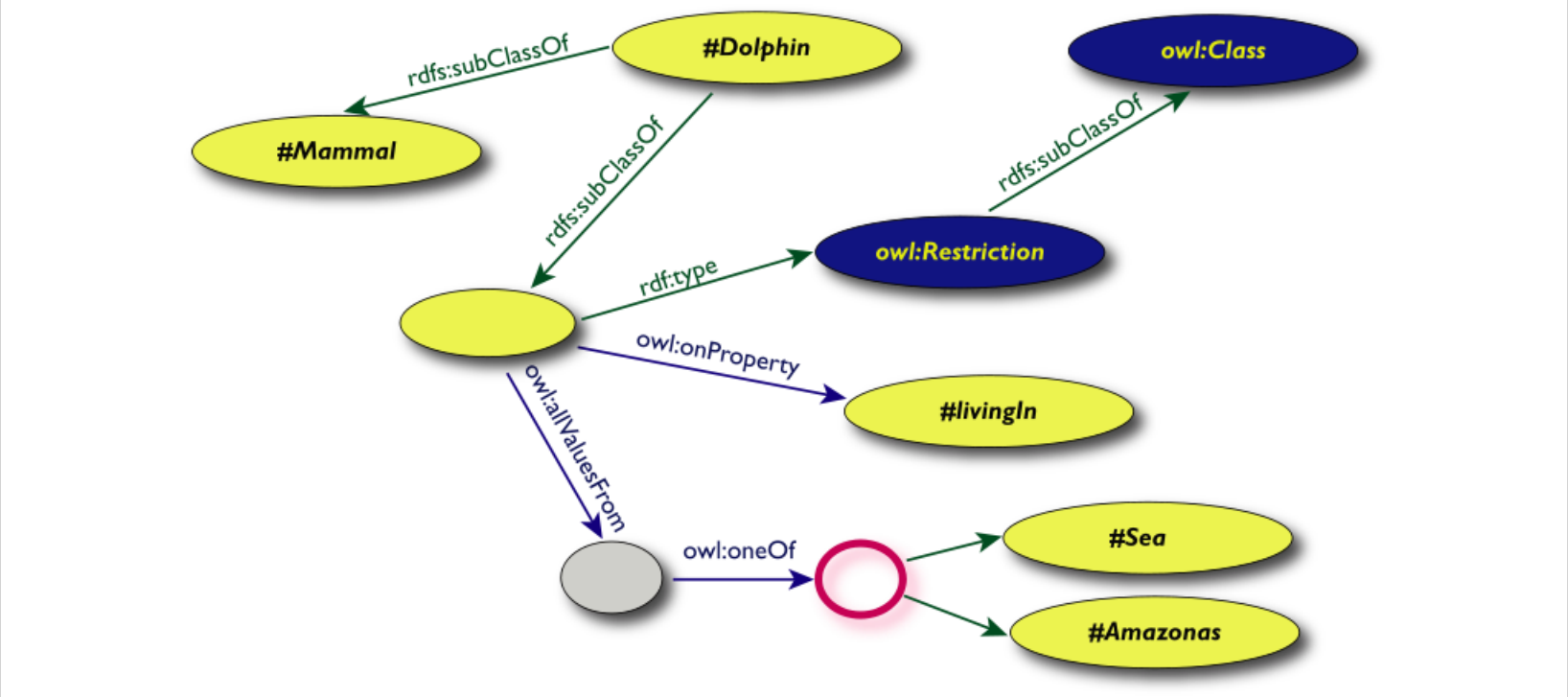
- (Sub)classes created by restricting the property value *on that class*
- For example, “a dolphin is a mammal living in sea or in the Amazonas” means:
  - *restrict the value of “living in” when applied to “mammal” to a specific set…*
  - *…thereby define the class of “dolphins”*

Property Restrictions in OWL

- Restriction may be by:
  - *value constraints (i.e., further restrictions on the range)*
    - *all* values must be from a class (like the dolphin example)
    - *some* values 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”:



Restrictions Formally

- Define a blank node of type `owl:Restriction` (which is a `owl:Class`) with a:
  - *a reference to the property that is constrained*
  - *a definition of the restriction itself*
- One can, e.g., subclass from this node

Same Serialized

```
<owl:Class rdf:ID="Dolphin">
    <rdfs:subClassOf rdf:resource="#Mammal"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#livingIn"/>
        <owl:allValuesFrom rdf:resource="#UnionOfSeaAndAmazonas">
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
:Dolphin rdf:type owl:Class;
    rdfs:subClassOf :Mammal;
    rdfs:subClassOf [
      rdf:type          owl:Restriction;
      owl:onProperty    :livingIn;
      owl:allValuesFrom :UnionOfSeaAndAmazonas.
   ]
.
```

allValuesFrom could be replaced by **someValuesFrom**, **cardinality**,

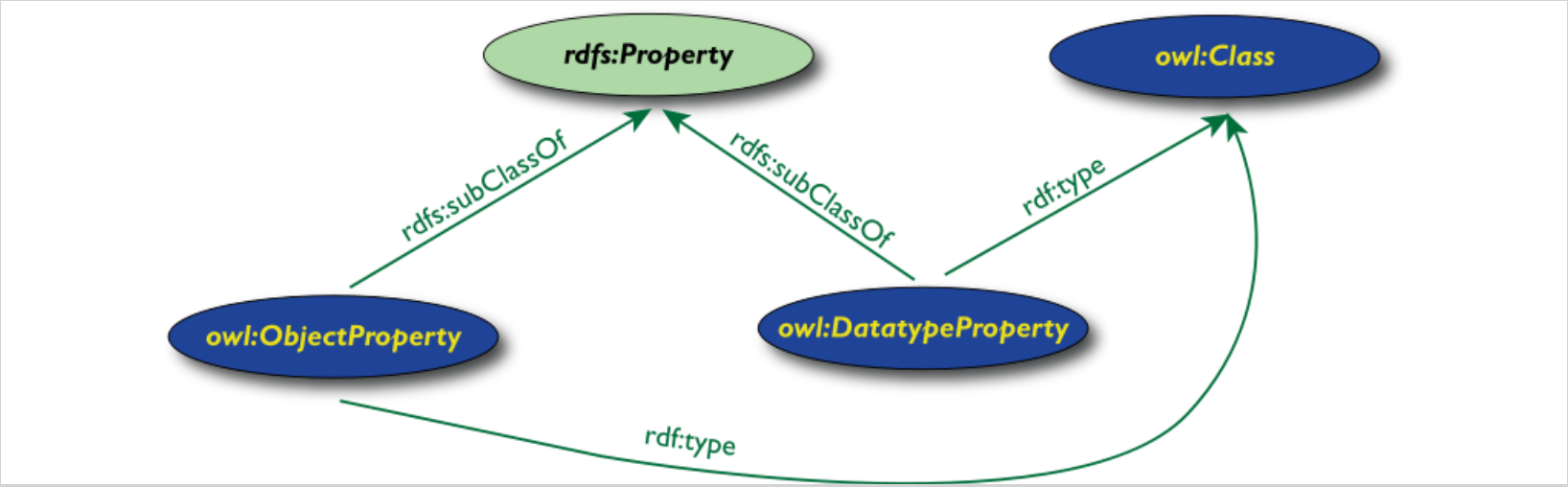
`minCardinality`, or `maxCardinality`

Cardinality Constraint Example

```
<owl:Class rdf:ID="Beluga">
    . . .
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#typeOfDorsalFins"/>
        <owl:cardinality rdf:datatype=".../nonNegativeInteger">
          0
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    . . .
</owl:Class>
:Beluga rdf:type owl:Class
    . . .
    rdfs:subClassOf [
      rdf:type        owl:Restriction;
      owl:onProperty  :typeOfDorsalFins;
      owl:cardinality "0"^^<.../nonNegativeInteger>.
    ];
```

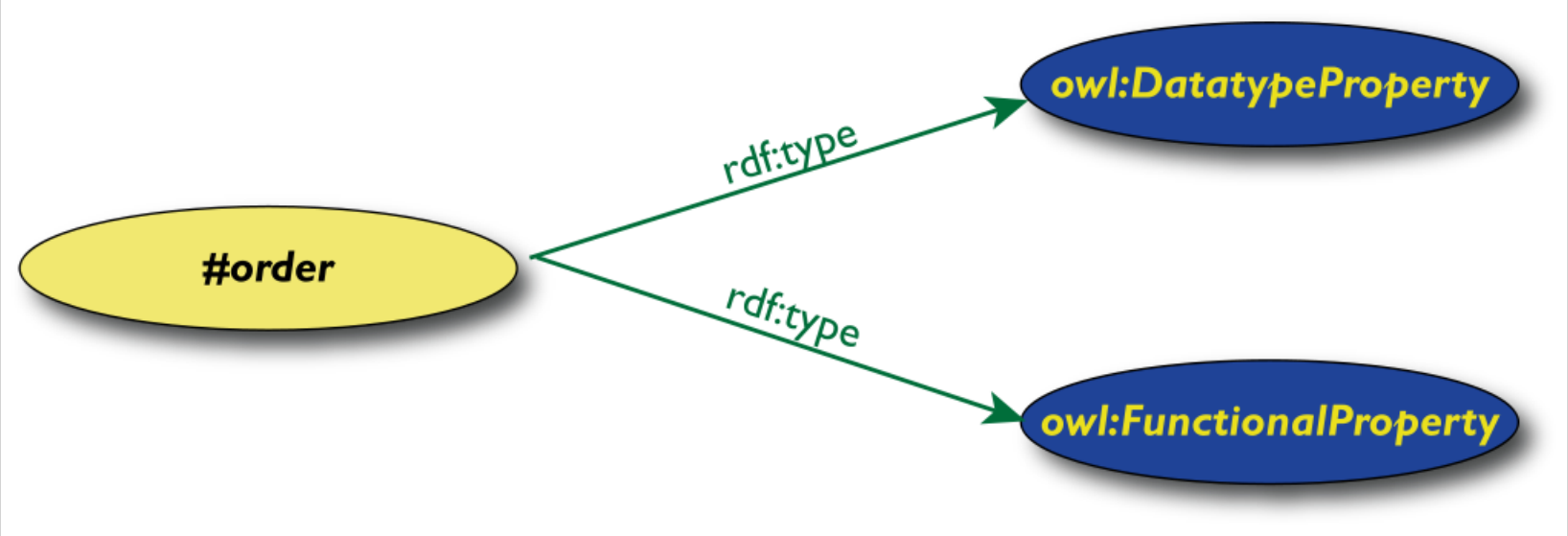
Property Characterization

- 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 order for each animal class” (in scientific classification)



Same Serialized

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

- Similar characterization possibilities:
  - ***InverseFunctionalProperty***
  - ***TransitiveProperty***, ***SymmetricProperty***
- These features can be *extremely* important for ontology based applications!

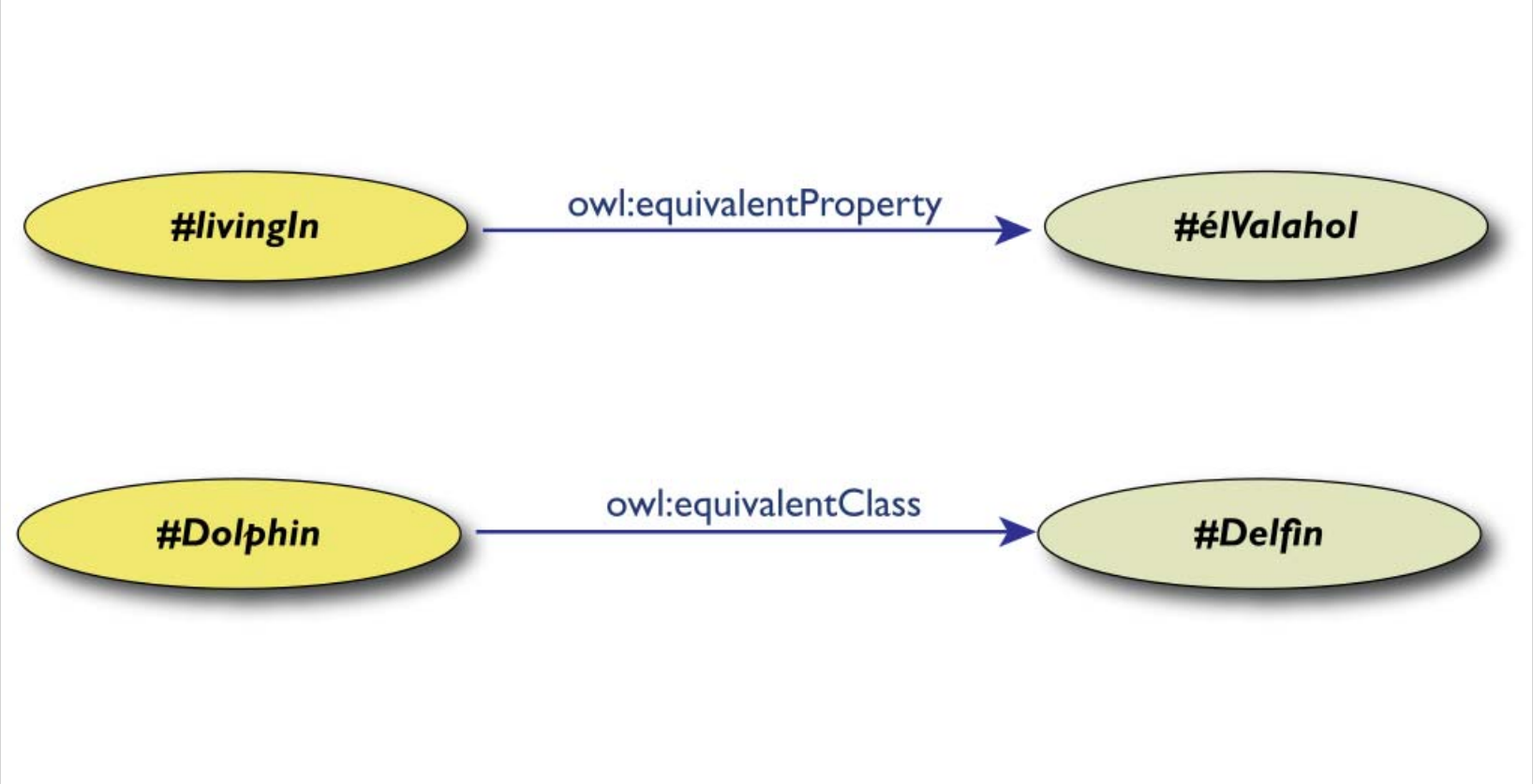
OWL: Additional Requirements

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

Term Equivalence/Relations

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

Example: Connecting to Hungarian



Versioning, Annotation

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

However: Ontologies are Hard!

- A full ontology-based application is a very complex system
- 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)

```
:A rdf:type owl:Class;
   owl:equivalenClass [
      rdf:type          owl:Restriction;
      owl:onProperty    rdf:type;
      owl:allValuesFrom :B.
   ].
:B rdf:type owl:Class;
   owl:complementOf :A.
```

- Is the following true?

```
:c rdf:type :A.
```

- if **c** is of type **A** then it *must* be in **B**, but then it is in *the complement of* **A**, ie, it is *not* of type **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
- No characterization of datatype properties possible
- …

OWL Lite

Goal: provide a minimal useful subset, easily implemented

- 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*
  - *...*
- Simple class hierarchies can be built
- Property constraints and characterizations can be used

Note on OWL layers

- OWL Layers were defined to reflect compromises:
  - *expressibility 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)*

Ontology Development

- 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*
- There are also some large ontologies in the public:
  - *eClassOwl: eBusiness ontology for products and services, 75,000 classes and 5,500 properties*
  - *the Gene Ontology: to describe gene and gene product attributes in any organism*
  - *UniProt: protein sequence and annotation data, hundreds of millions of triples(!)*

Simple Knowledge Organization System (SKOS)

Simple Knowledge Organization 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”

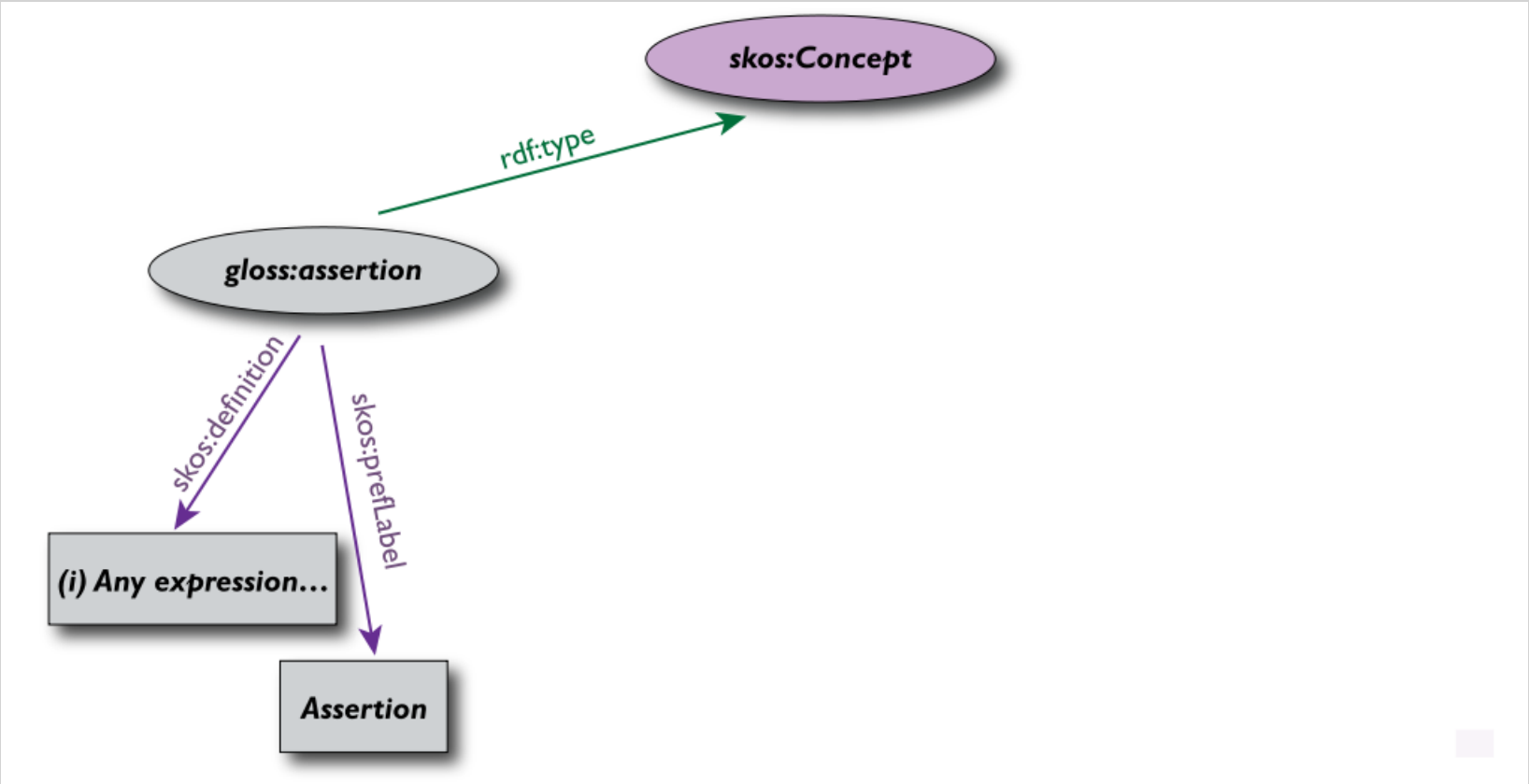
> “A general concept, category or classification. Something used primarily to classify or categorize other things.”

“Resource”

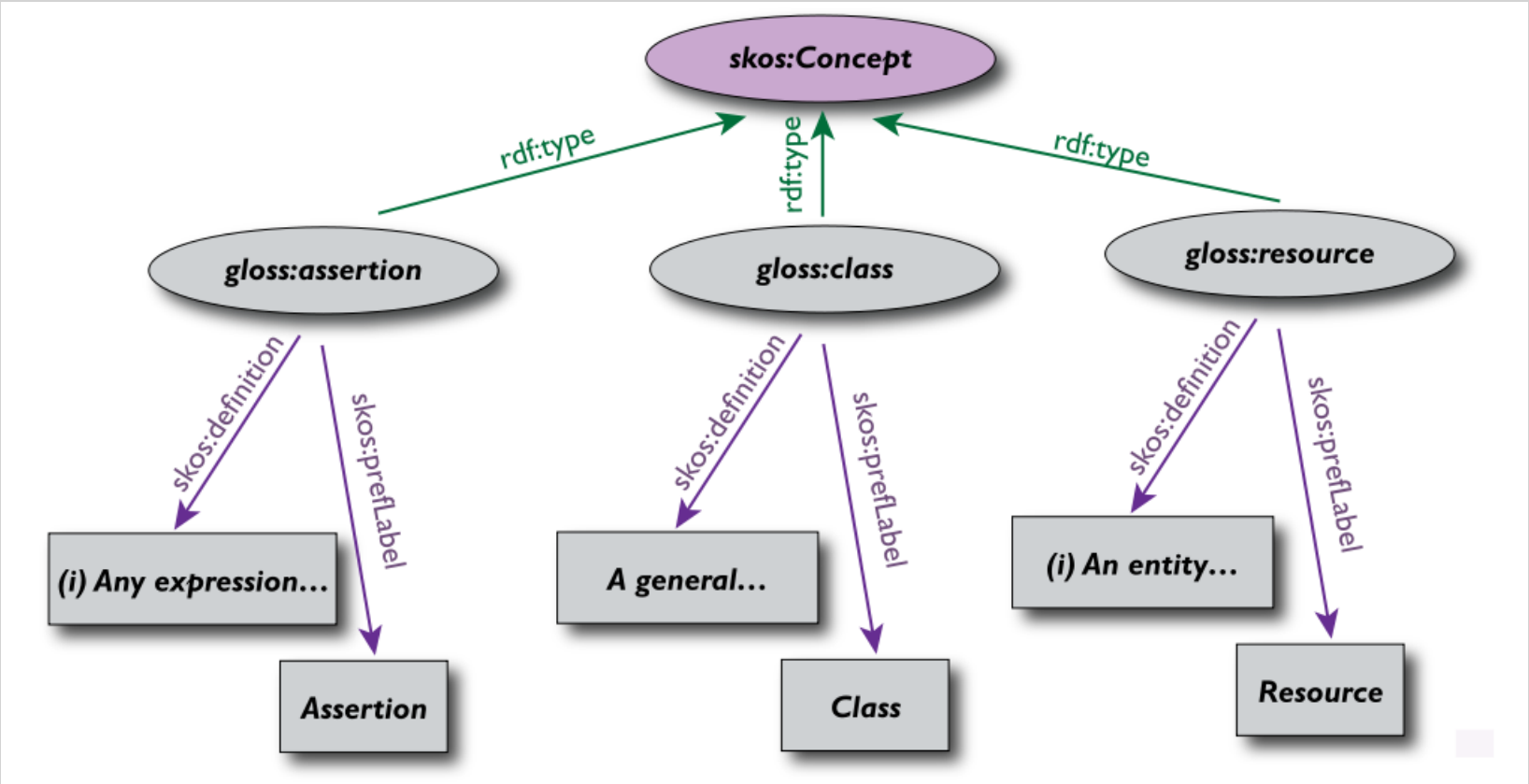
> “(i) An entity; anything in the universe. (ii) As a class name: the class of everything; the most inclusive category possible.”

(from the RDF Semantics Glossary)

Example: Entries in a Glossary (2)



Example: Entries in a Glossary (3)



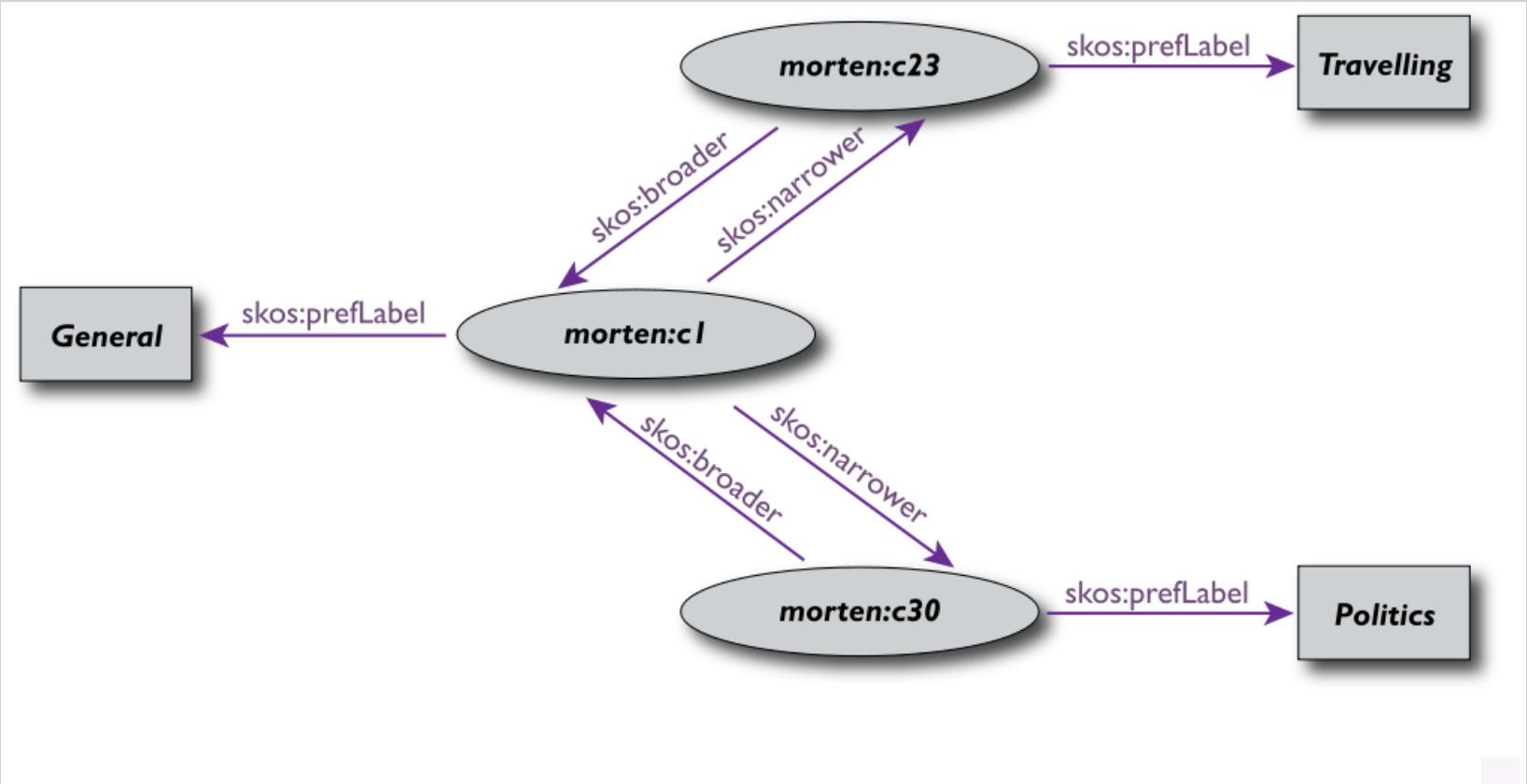
Example: Taxonomy (1)

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

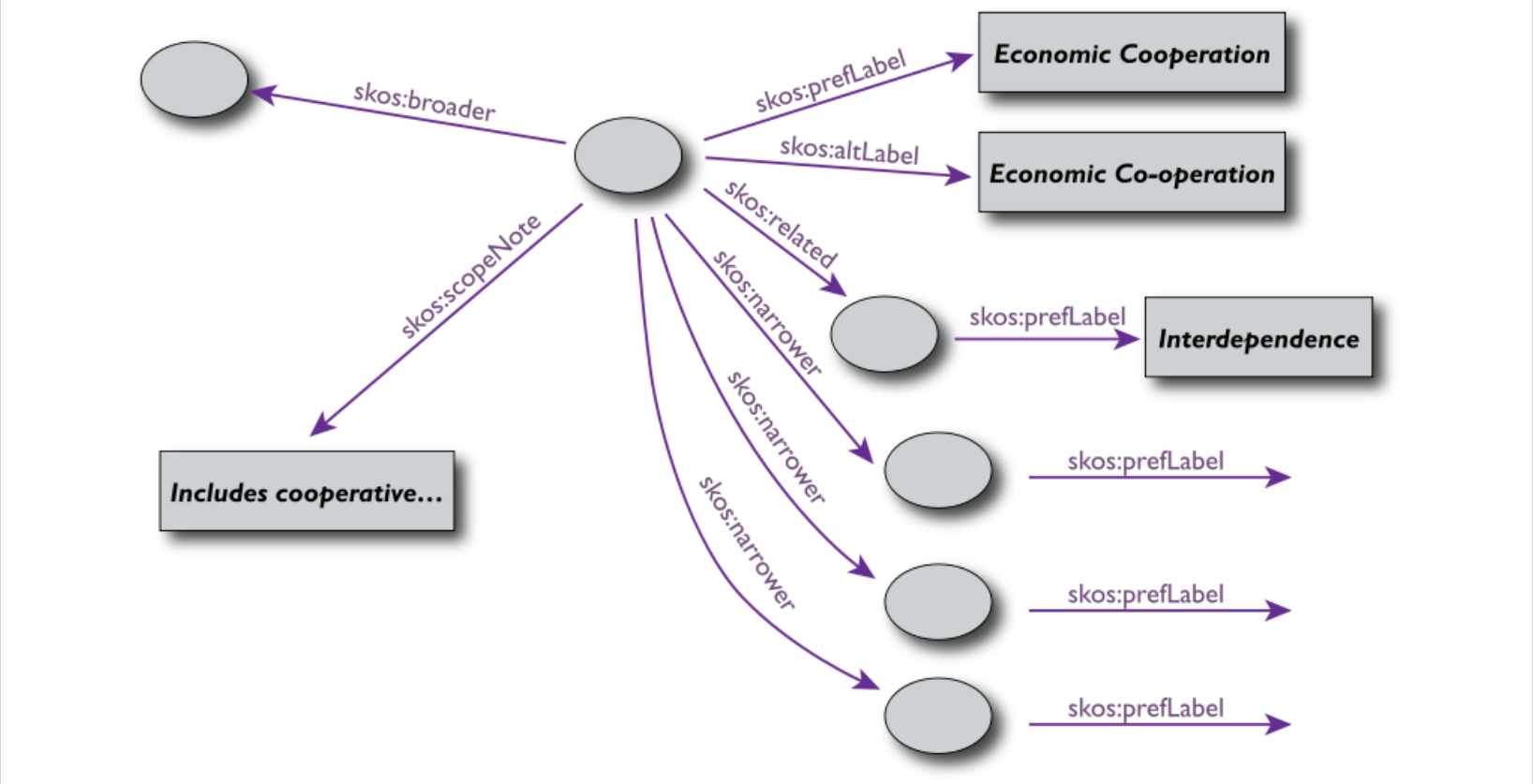
Interdependence

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*
- A typical example: the Glossary of project of W3C stores all terms in SKOS (and extracted from W3C documents)

SKOS Documents

- SKOS documents may be finalized in early 2007:
  - *"Quick Guide to Publishing a Thesaurus on the Semantic Web"* and *"SKOS Core Guide"*
  - *"SKOS Core Vocabulary Specification"*
  - *"SKOS Mapping Vocabulary Specification"*
- SKOS is currently a "W3C Note", will be put into a Recommendation track this year

“Core” Vocabularies

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

What is Coming?

Semantic Web Activity Phases

- First phase (practically completed): core infrastructure (RDFS, OWL, SPARQL)
- Current activities and plans at W3C:
  - *promotion and applications needs, outreach to user communities*
    - e.g., tutorials, best practice notes, business cases
    - a separate Interest Group on Health Care and Life Sciences (HCLS) Interest Group has started end of 2005
  - *Intersection of SW with other technologies (Semantic Web Services, privacy, …)*
  - *Further technical development (Rule Interchange Formats, GRDDL, SKOS, RDFa)*

Rules

- OWL can be used for simple inferences
- Applications may want to express domain-specific knowledge, like “Horn clauses”:
  - $(P_1 \wedge P_2 \wedge \ldots) \rightarrow C$
  - *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, …)

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*
- Great interest from financial services, business rules, life science community…

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 planned in May 2007

RIF Use Cases and Requirements

- The first draft has just been published
- Contains a number of use cases, e.g.:
  - *negotiating eBusiness contracts across rule platforms: supply vendor-neutral representation of your business rules so that others may find you*
  - *describing privacy requirements and policies, and let client "merge" those (e.g., when paying with a credit card)*
  - *medical decision support, combining rules on diagnoses, drug prescription conditions, etc,*
  - *extending OWL with rule-based statements (e.g., the uncle example)*

RIF Phase 2 Goals

- Define more complex extensions
  - *towards First Order Logic (FOL), Logic Programming systems…*
  - *syntactic extensions to Horn logic like Lloyd-Topor*
  - *actions, i.e., running procedural codes as part of rules*
- First recommendation(s) planned 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, Business Rules,…) in a coherent framework
- Relationships to RDFS, OWL
  - *semantical, model theoretical, syntactical issues*
  - *“One Tower” vs. “Two Towers” models*

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 …

Other Issues…

- Improve the inference algorithms and implementations, scalability, reasoning with OWL Full
- Better modularization (import or refer to *part of* ontologies)
- Ontology management on the Web
- Extensions of RDF and/or OWL (based on experience and theoretical advances)
- Temporal & spatial reasoning
- Probabilistic reasoning and/or fuzzy logic
- …

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)

- The RDF specification itself is spread over several documents (“RDF: Concept and Abstract Syntax”, “RDF Vocabulary Description Language (RDF Schema)”, “RDF Semantics”, and “RDF/XML Serialization”)
  - *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
  - *use 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 of the semantics

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
- Semantic Web Tutorials (list of references)
- Survey of RDF/Topic Map Maps Interoperability
- “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, WWW2005, and WWW2006; see also the 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, Cerebra's Construct

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

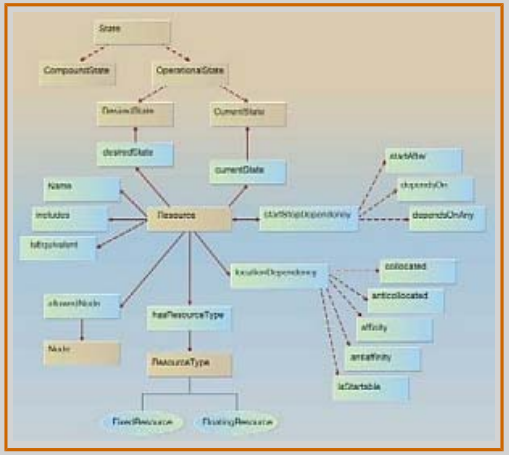
Swoogle

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 details…
- Oracle seems to aim for an role in this space…

IBM – Life Sciences and Semantic Web

- IBM Internet Technology Group
  - *focusing on general infrastructure for Semantic Web applications*
- Integrated toolkit (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*



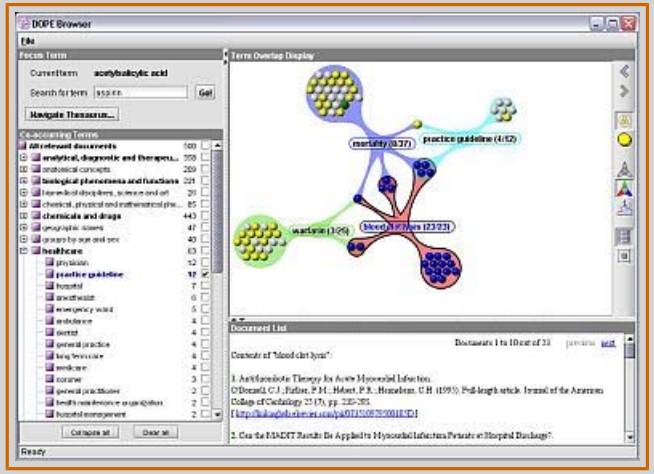
Some Application Examples

SW Applications

- Large number of applications emerge
- Most applications are still “centralized”, not many decentralized applications yet
- Huge datasets are accumulating. E.g.,:
  - *RDF version of Wikipedia: more than 47 million triplets, based also on SKOS, soon with a SPARQL interface*
  - *tracking the US Congress: data stored in RDF (around 25 million triplets) with a SPARQL interface*
- For further examples, see, for example, the Semantic Technology Conference series
  - *not a scientific conference, but commercial people making real money!*
  - *speakers in 2006: from IBM, Cisco, BellSouth, GE, Walt Disney, Nokia, Oracle, …*

Data integration

- Semantic integration of different data sources
- RDF/RDFS (possibly with OWL and/or SKOS) based vocabularies as an "interlingua" among system components
- Many different projects and R&D on this: Boeing, MITRE Corp., Elsevier, EU Projects like Sculpteur and Artiste, national projects like MuseoSuomi, …

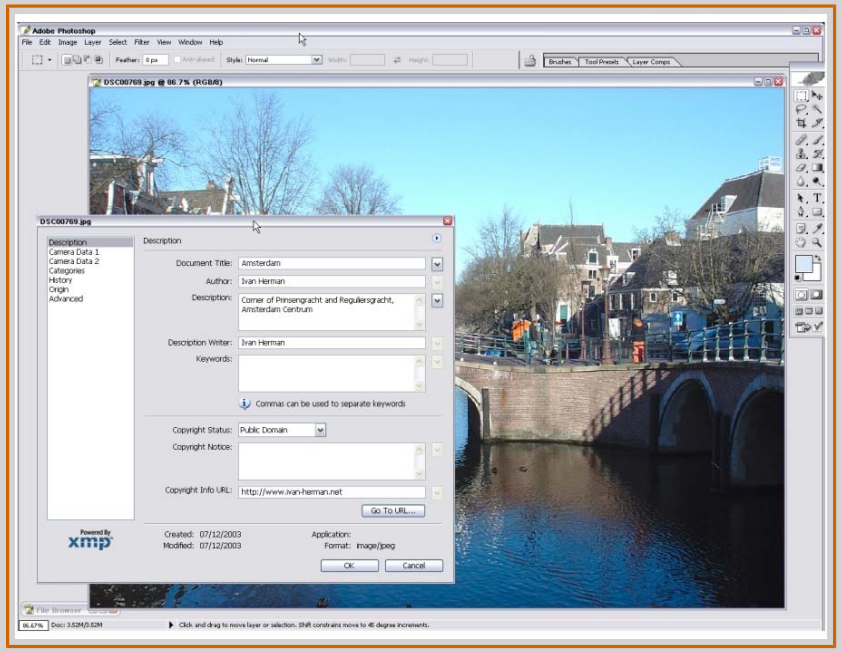


Portals

- Vodafone'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

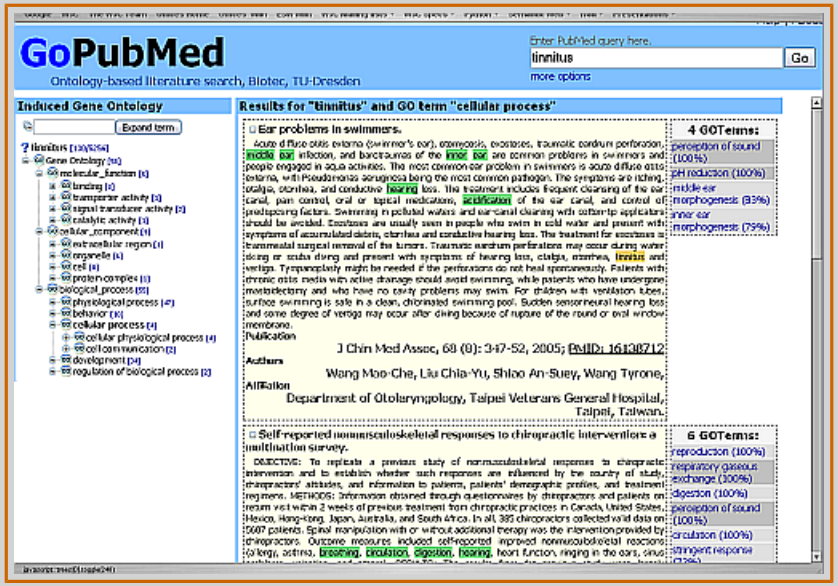
Adobe's XMP

- Adobe’s tool to add RDF-based metadata to *most* of their file formats
  - *supported in Adobe Creative Suite*
  - *support from 30+ major asset management vendors, with separate XMP conferences*
- 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



Further Information

These slides are at:

http://www.w3.org/2006/Talks/0524-Edinburgh-IH/

http://www.w3.org/2006/Talks/0524-Edinburgh-IH/Overview.pdf

Semantic Web homepage

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

More information about W3C:

http://www.w3.org/

Mail me:

ivan@w3.org