Introduction to the Semantic Web

International Conference on Dublin Core and Metadata Applications, Singapore, 2007-08-31

Ivan Herman, W3C

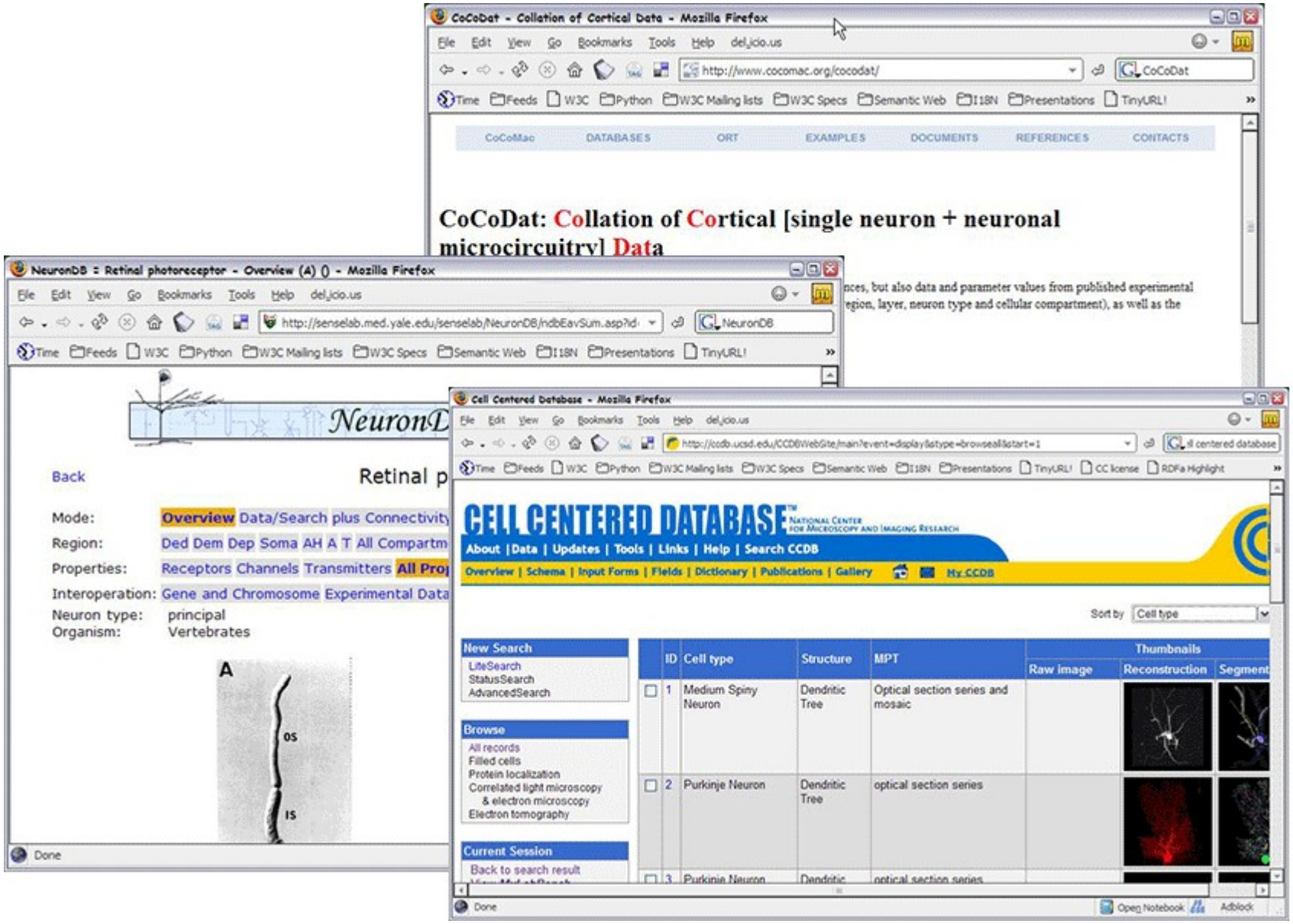
> Introduction

- This audience knows the value of machine readable data very well…
- But: on the Semantic Web the terminology does not separate the concept of metadata and data
  - data could be metadata the way we know it
  - but it could be, say, my calendar on line…
    - one's metadata may be somebody else's data…

> Example: data(base) integration

- Databases are very different in structure, in content
- Lots of applications require managing/merging several databases
  - after company mergers
  - combination of administrative data for e-Government
  - biochemical, genetic, pharmaceutical research
  - etc.
- Most of these data are accessible from the Web (though not necessarily public yet)
  - again, some of the information may be in metadata
  - but some is just the data itself

> And the problem *is* real…



> What is needed?

- (Some) data should be available for machines for further processing
- Data should be possibly combined, merged on a Web scale
- Machines may also need to *reason* about that data

> In what follows…

- We will use a simplistic example to introduce the main Semantic Web concepts
- We take, as an example area, data integration

> The rough structure of data integration

1. Map the various data onto an abstract data representation
   - make the data independent of its internal representation…
2. Merge the resulting representations
3. Start making queries on the whole!
   - queries that could not have been done on the individual data sets

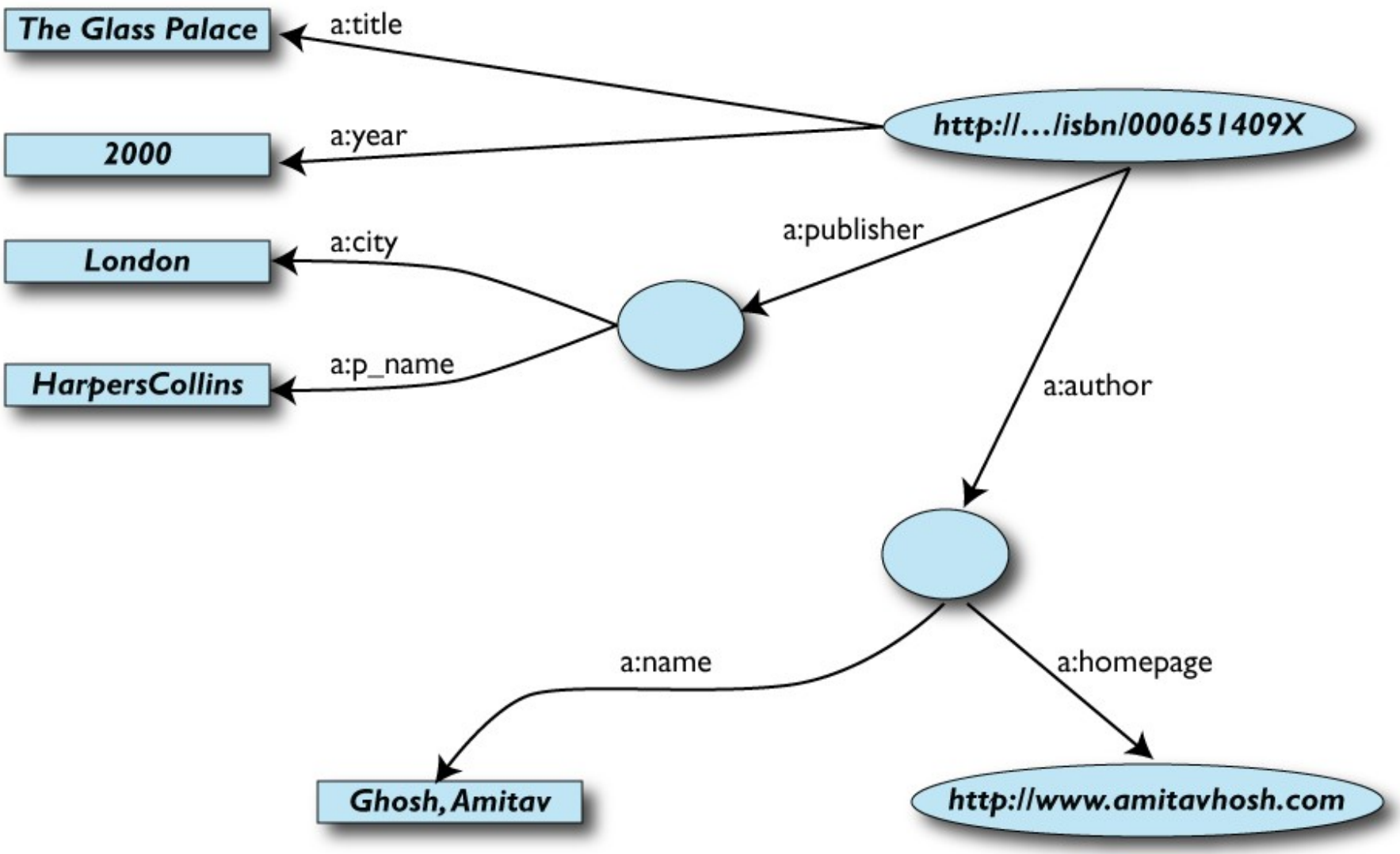
> A *simplified* bookstore data (dataset "A")

ID	Author	Title	Publisher	Year
ISBN 0-00-651409-X	id_xyz	The Glass Palace	id_qpr	2000

ID	Name	Home page
id_xyz	Ghosh, Amitav	http://www.amitavghosh.com/

ID	Publ. Name	City
id_qpr	Harper Collins	London

> 1st: export your data as a set of *relations*



> Some notes on the exporting the data

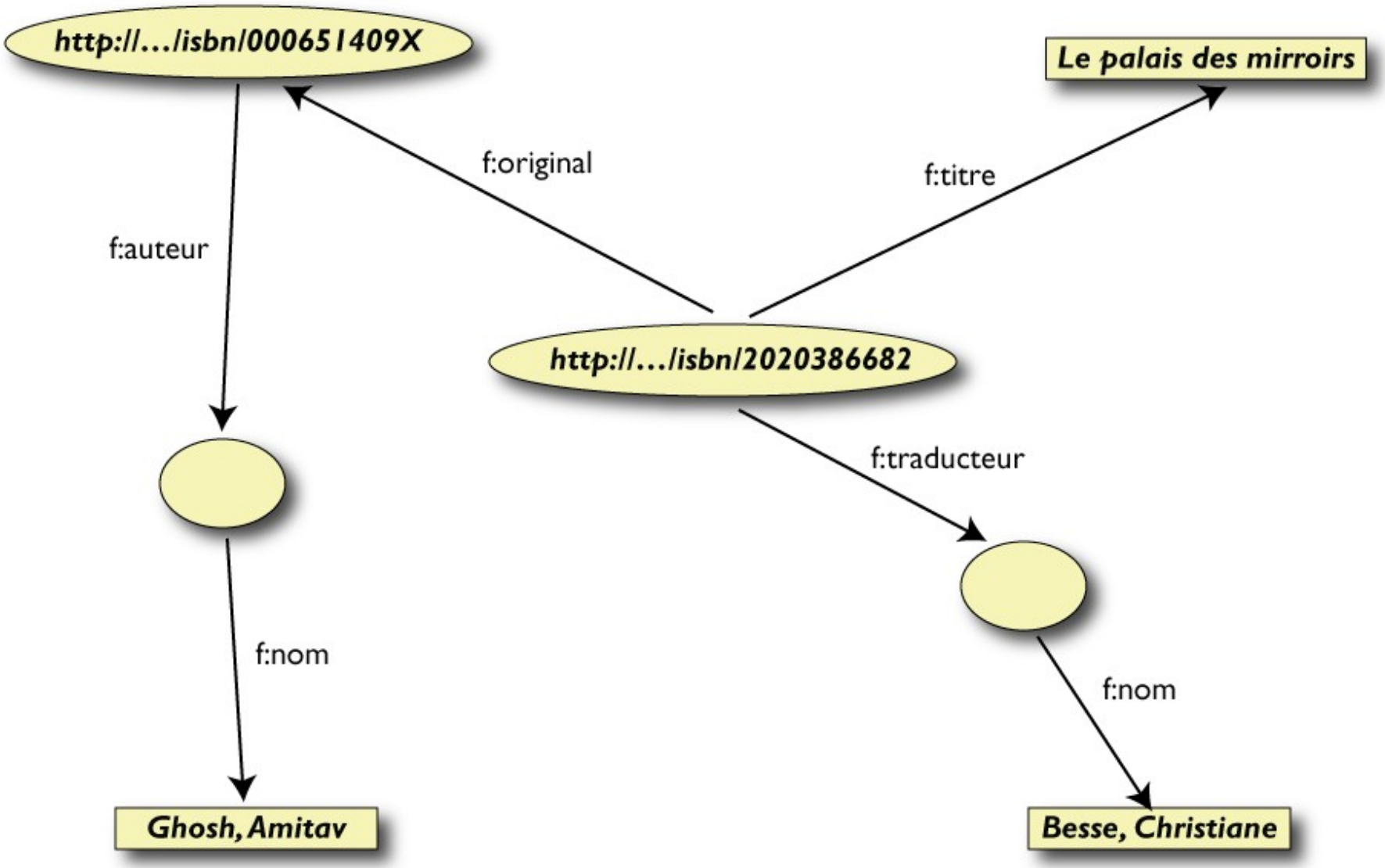
- Relations form a graph
  - the nodes refer to the “real” data or contain some literal
  - how the graph is represented in machine is immaterial for now
- Data export does *not* necessarily mean physical conversion of the data
  - relations can be generated on-the-fly at query time
    - via SQL “bridges”
    - scraping HTML pages
    - extracting data from Excel sheets
    - etc.
- One can export *part* of the data

> Another bookshop data (dataset “F”)

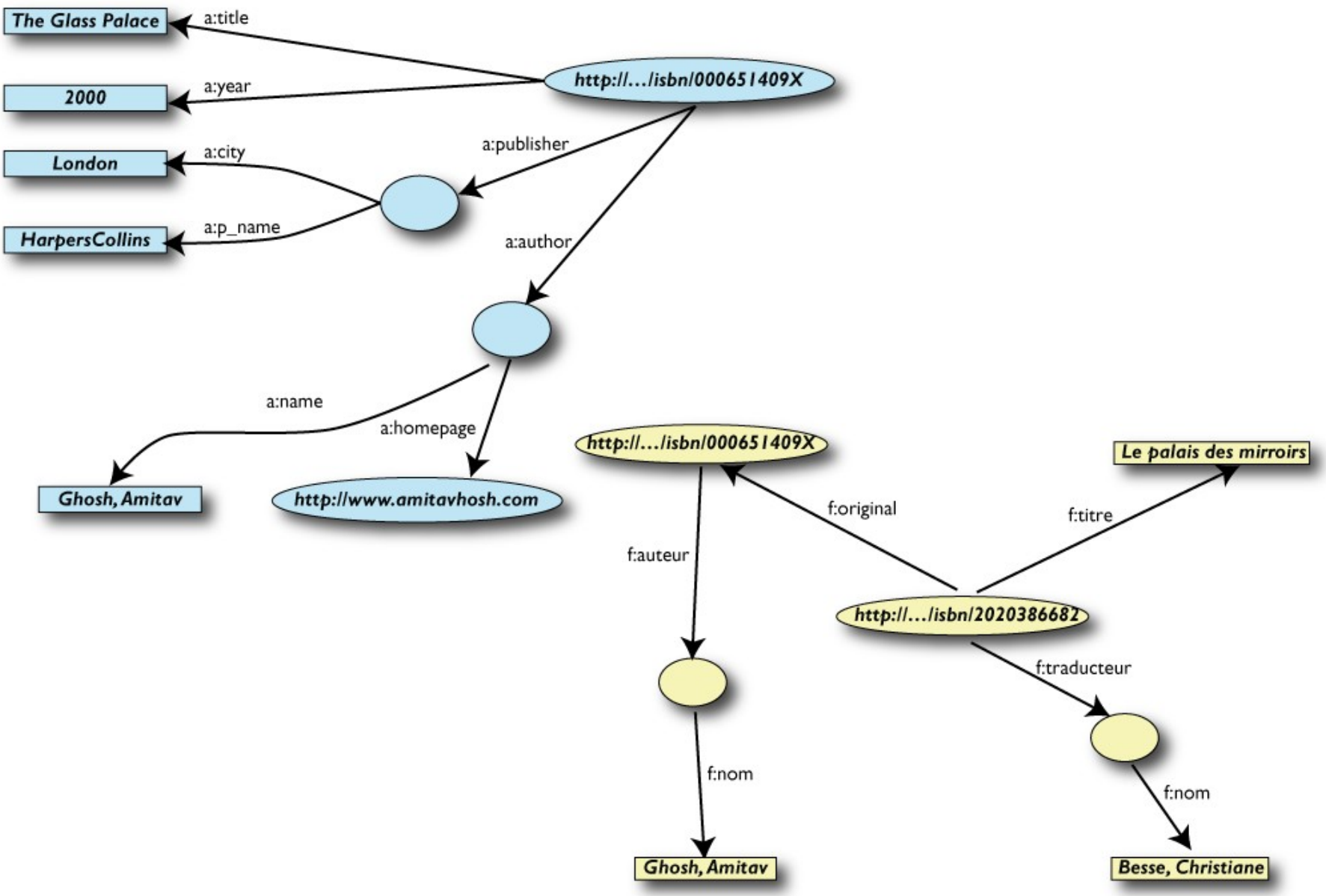
ID	Titre	Auteur	Traducteur	Original
ISBN 2020386682	Le Palais des miroirs	i_abc	i_qrs	ISBN 0-00-651409-X

ID	Nom
i_abc	Ghosh, Amitav
i_grs	Besse, Christiane

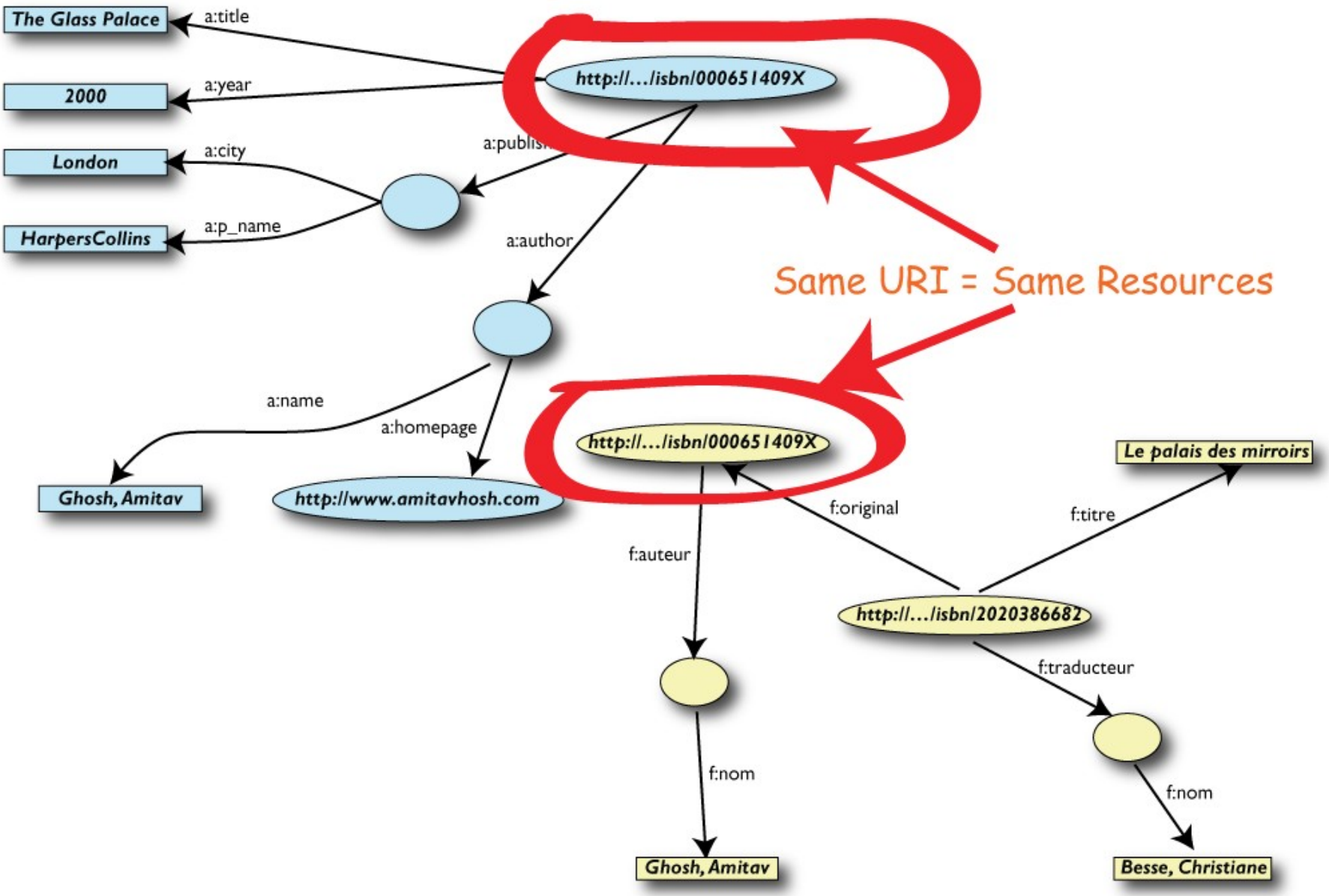
> 2nd: export your second set of data



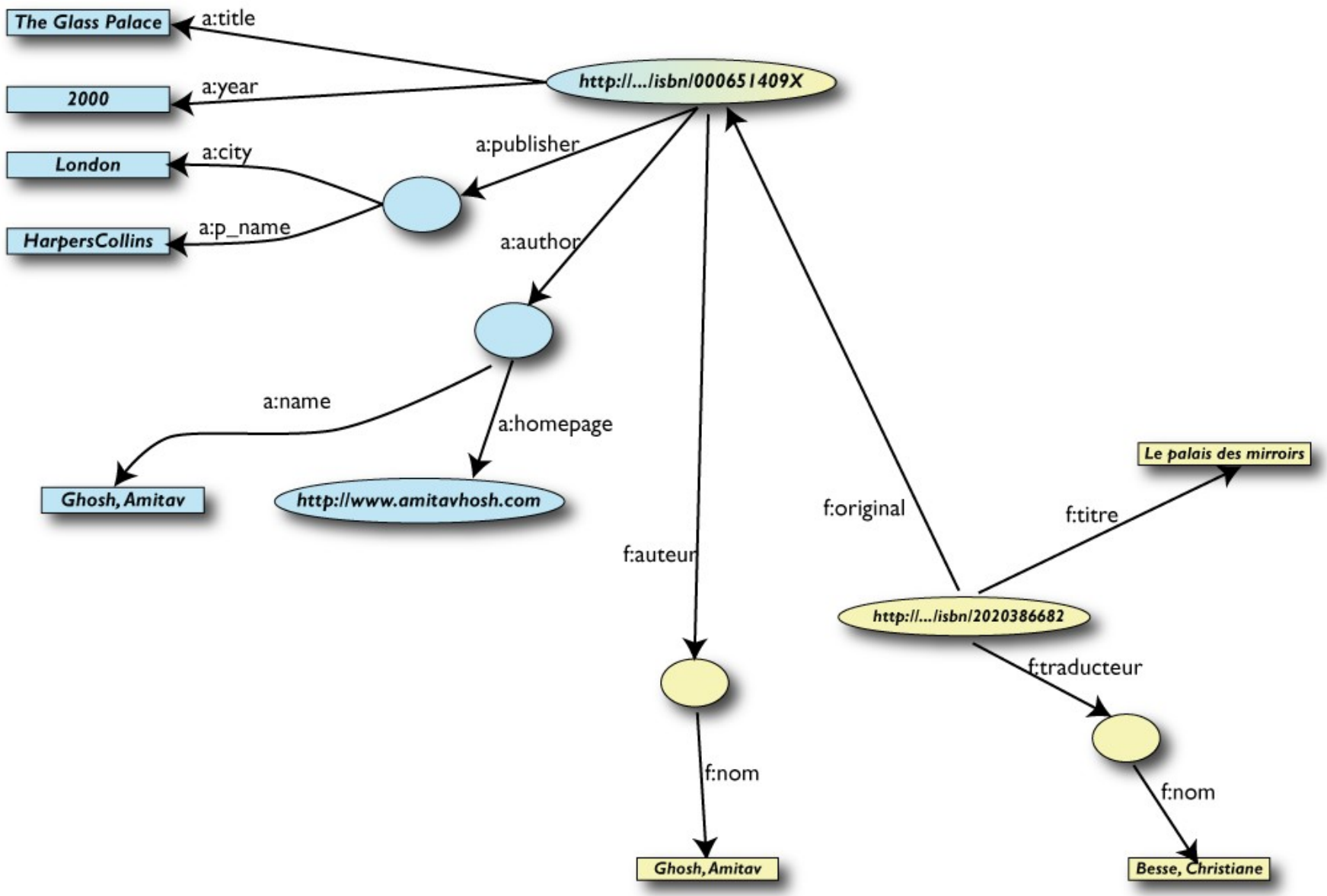
> 3rd: start merging your data



> 3rd: start merging your data (cont.)

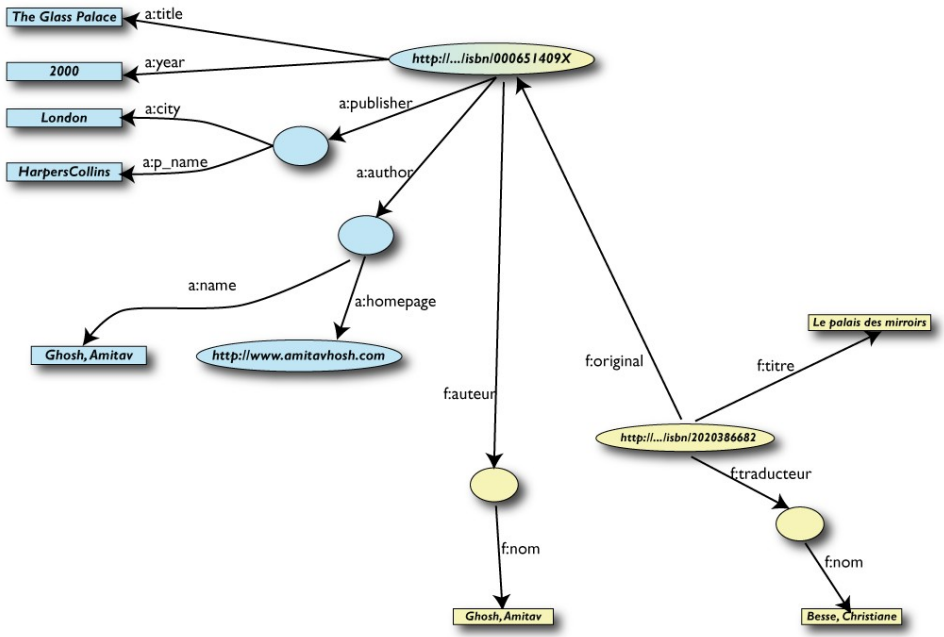


> 3rd: merge identical resources



> Start making queries…

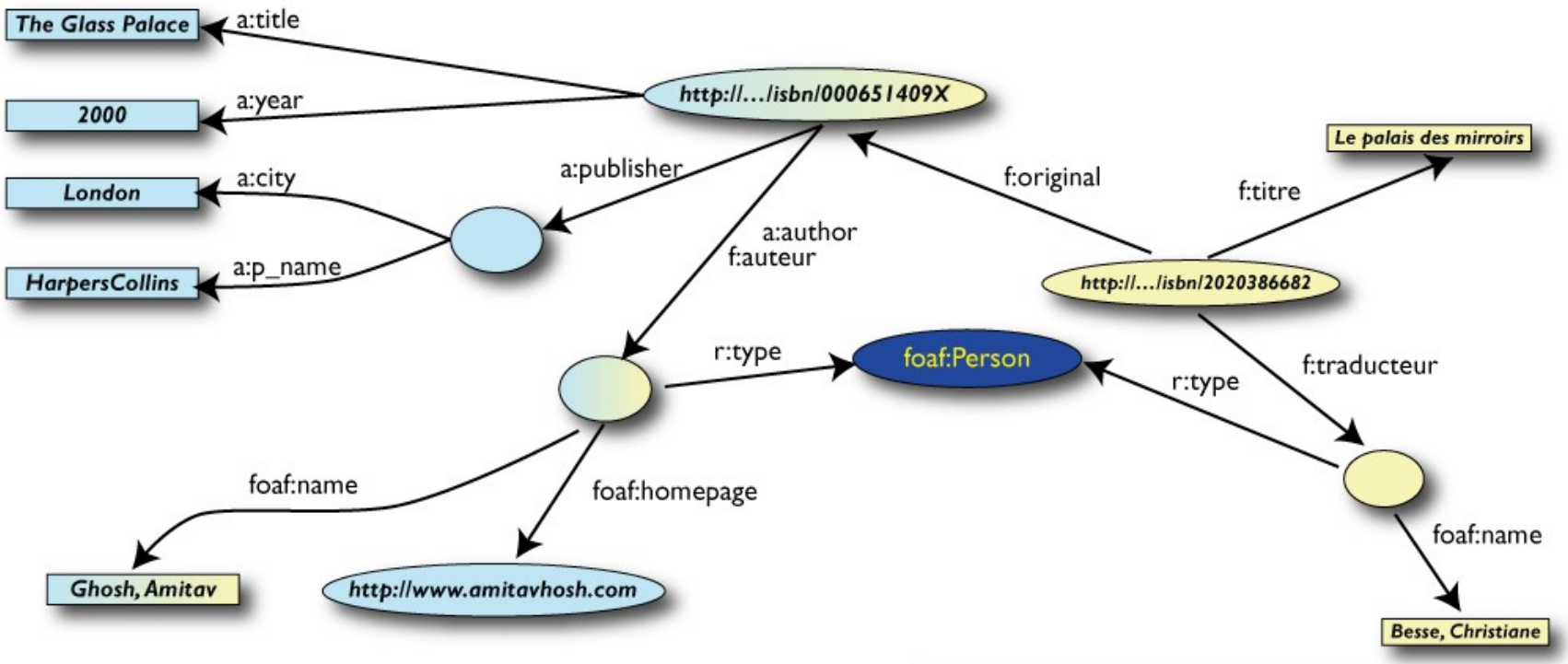
- User of data “F” can now ask queries like:
  - « donnes-moi le titre de l’original »
  - (ie: “give me the title of the original”)
- This information is not in the dataset “F”…
- …but can be retrieved by merging with dataset “A”!

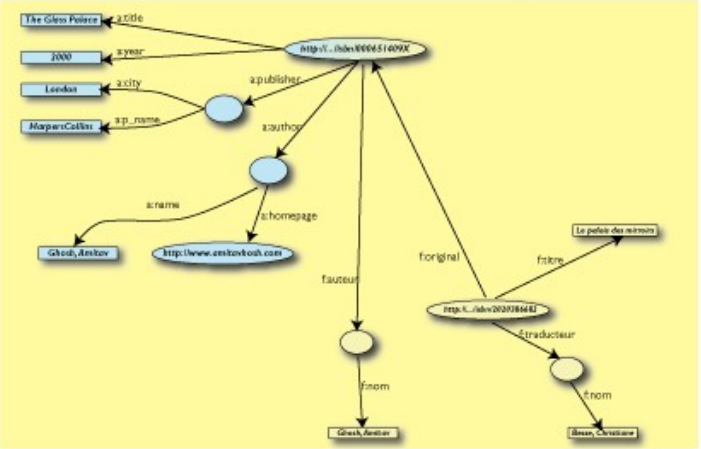


> However, more can be achieved…

- We “feel” that **`a:author`** and **`f:auteur`** should be the same
- But an automatic merge does not know that!
- Let us add some extra information to the merged data:
  - **`a:author`** same as **`f:auteur`**
  - both identify a “Person”
  - a term that a community may have already defined:
    - a “Person” is uniquely identified by his/her name and, say, homepage
    - it can be used as a “category” for certain type of resources

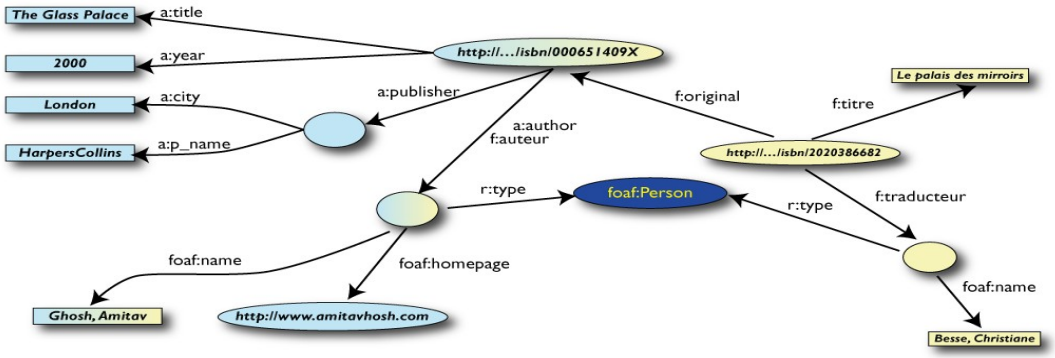
> 3rd revisited: use the extra knowledge





> Start making richer queries!

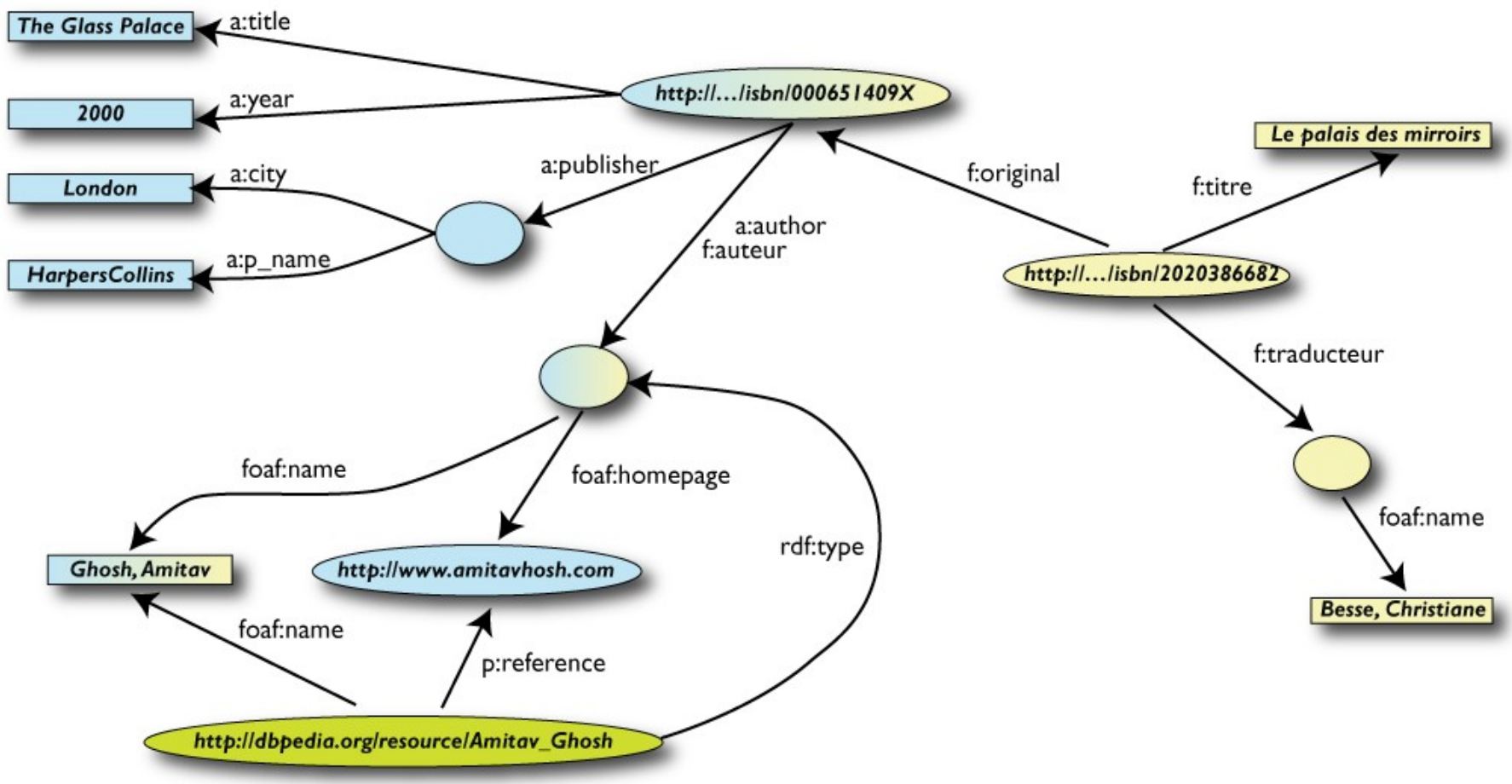
- User of dataset “F” can now query:
  - « donnes-moi la page d’accueil de l’auteur de l’original »
  - (ie, “give me the home page of the original’s author”)
- The information is not in datasets “F” or “A” …
- …but was made available by:
  - merging datasets “A” and datasets “F”
  - adding three simple extra statements as an extra “glue”



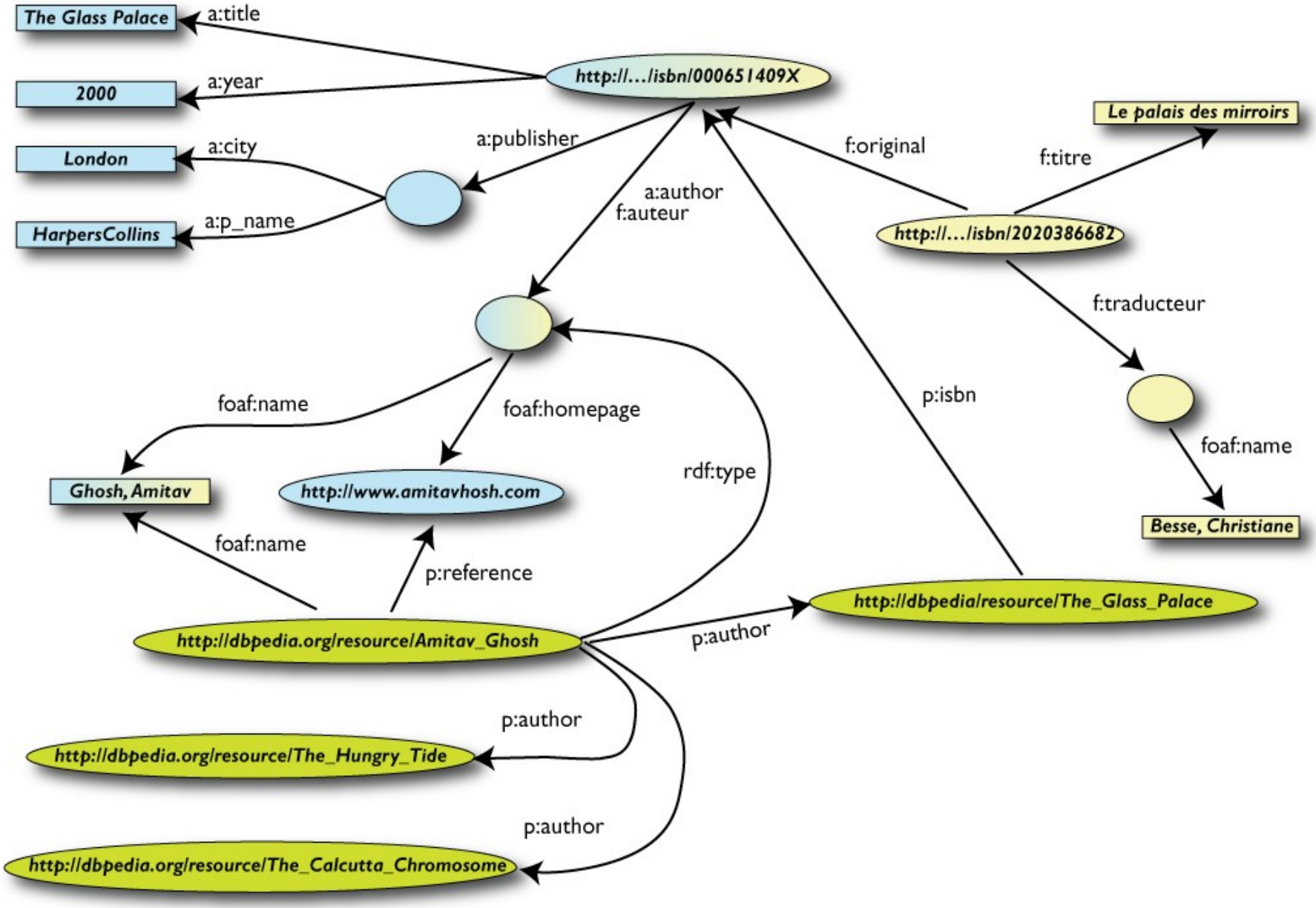
> Combine with different datasets

- Using, e.g., the “Person”, the dataset can be combined with other sources
- For example, data in Wikipedia can be extracted using dedicated tools

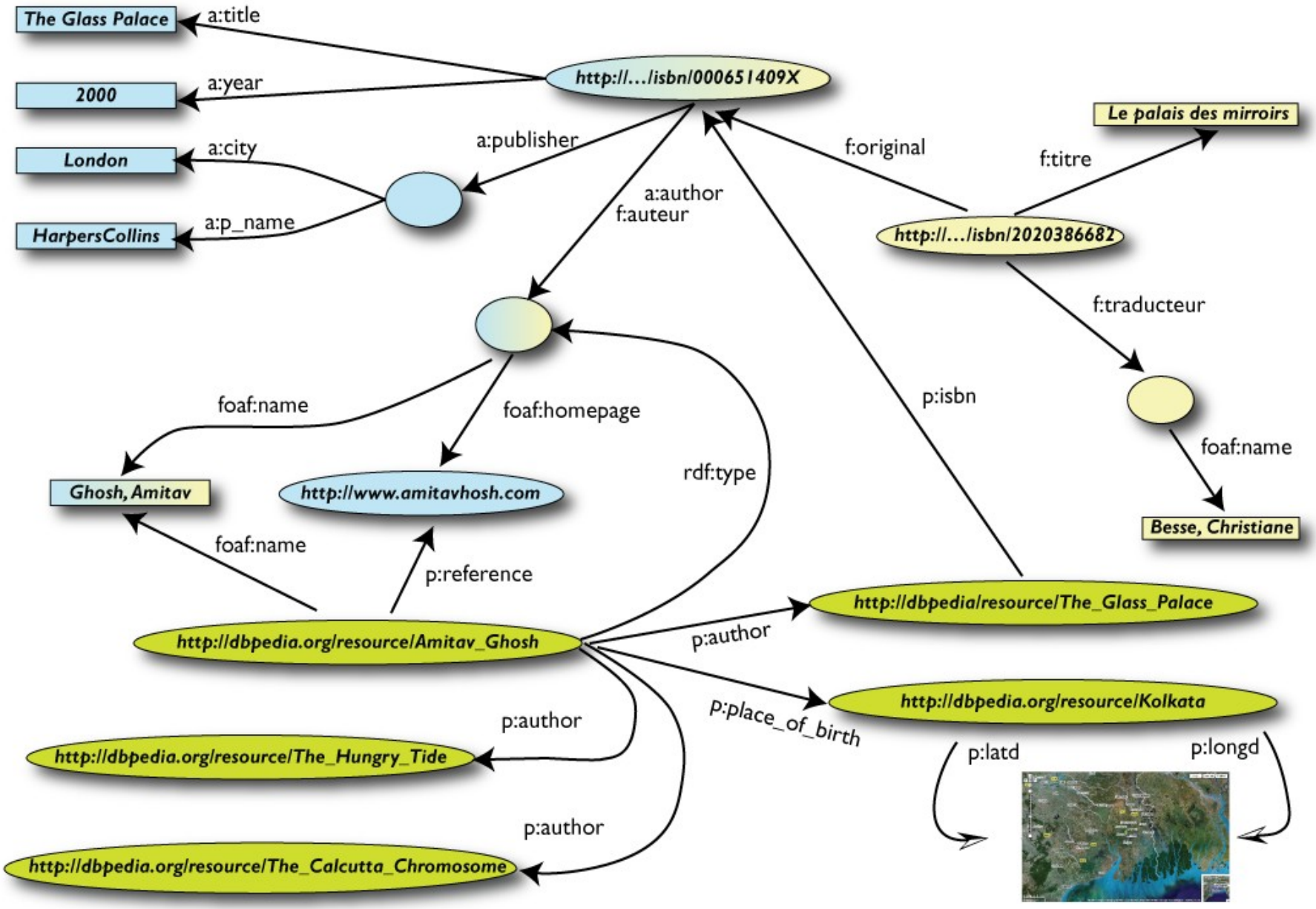
> Merge with Wikipedia data



> Merge with Wikipedia data



> Merge with Wikipedia data



> Is that surprising?

- Maybe but, in fact, no…
- What happened via automatic means is done all the time, every day by the users of the Web!
- The difference: a bit of extra rigor (e.g., naming the relationships) is necessary so that machines could do this, too

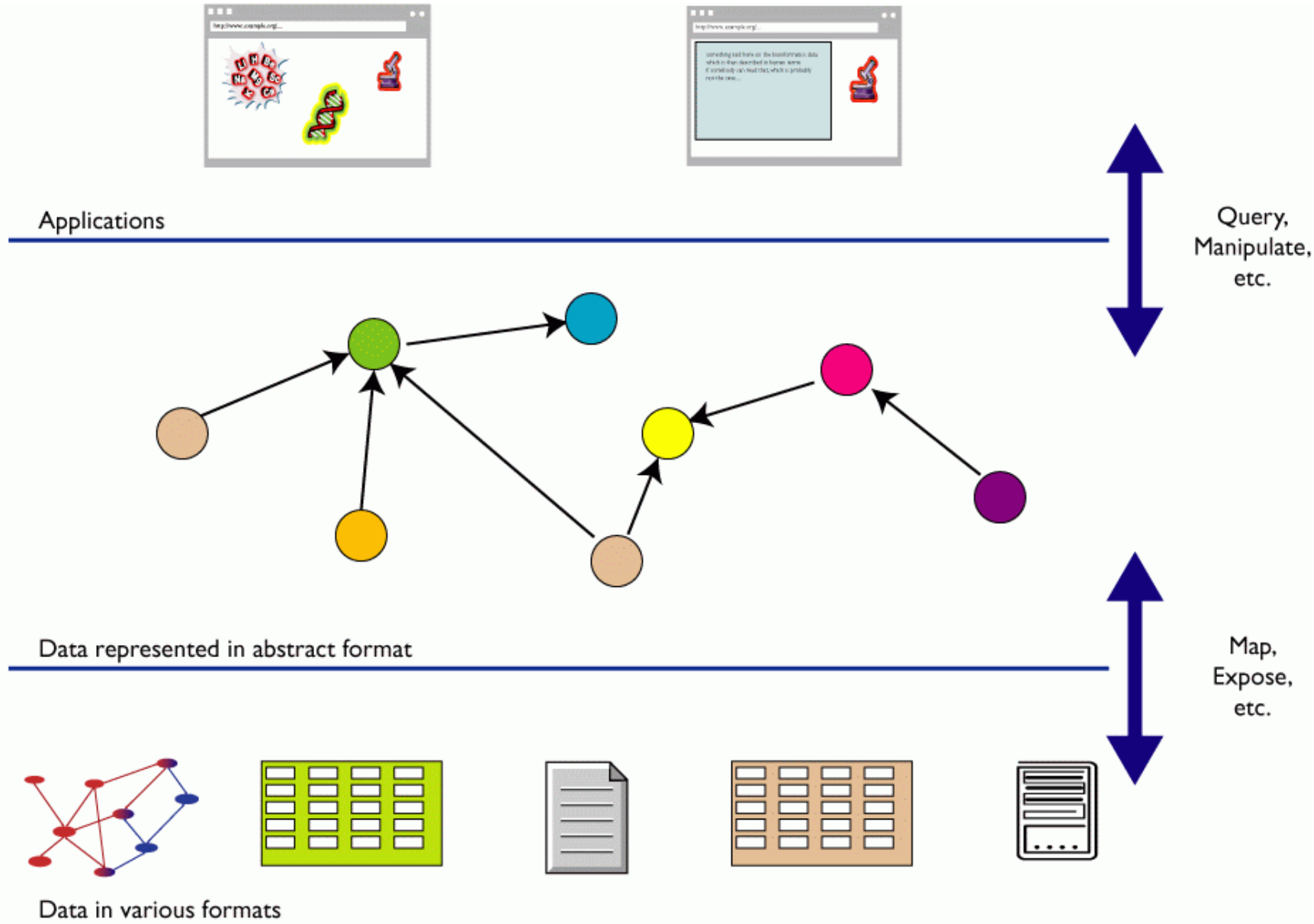
> What did we do?

- We combined different datasets
  - all may be of different origin somewhere on the web
  - all may have different formats (mysql, excel sheet, XHTML, etc)
  - all may have different names for relations (e.g., multilingual)
- We could combine the data because some URI-s were identical (the ISBN-s in this case)
- We could add some simple additional information (the “glue”), also using common terminologies that a community has produced
- As a result, new relations could be found and retrieved

> It could become even more powerful

- We could add extra knowledge to the merged datasets
  - e.g., a full library or bookshop data
  - more geographical information
  - etc.
- This is where various “vocabularies” (ontologies, thesauri, taxonomies) etc, may come in
- Even more powerful queries can be asked as a result

> What did we do? (cont)

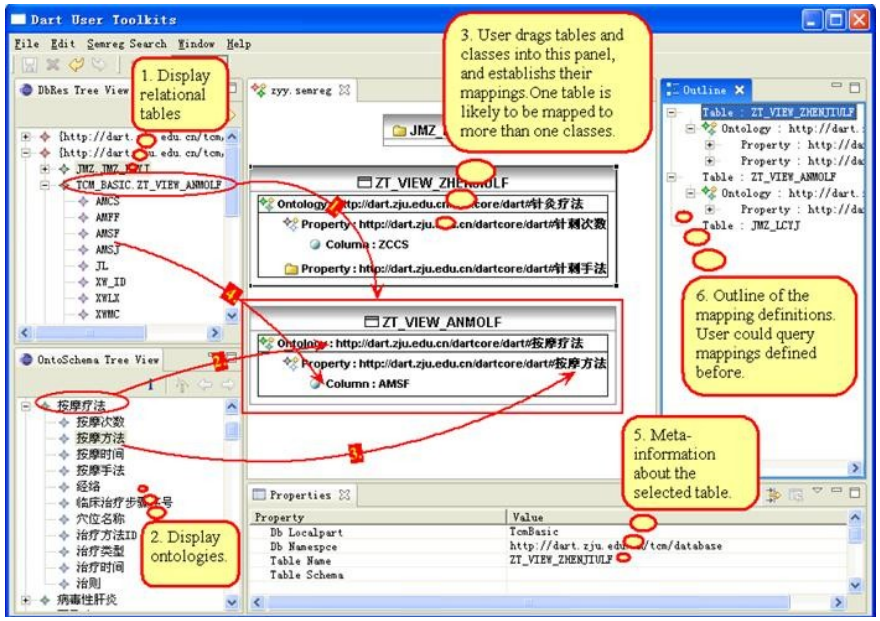


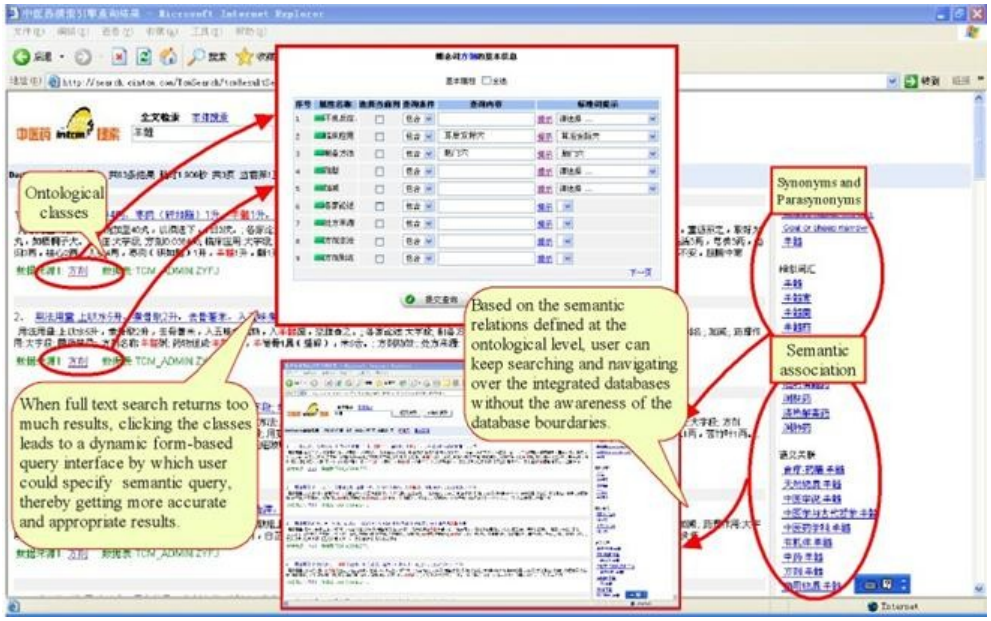
> So where is the Semantic Web?

- The Semantic Web provides technologies to make such integration possible!
- Hopefully you get a full picture at the end of the tutorial
- But let us see some real life examples first…

> Integrate knowledge for Chinese Medicine

- Integration of a large number of relational databases (on traditional Chinese medicine) using a Semantic Layer
  - around 80 databases, around 200,000 records each
- A visual tool to map databases to the semantic layer using a specialized ontology
- Form based query interface for end users

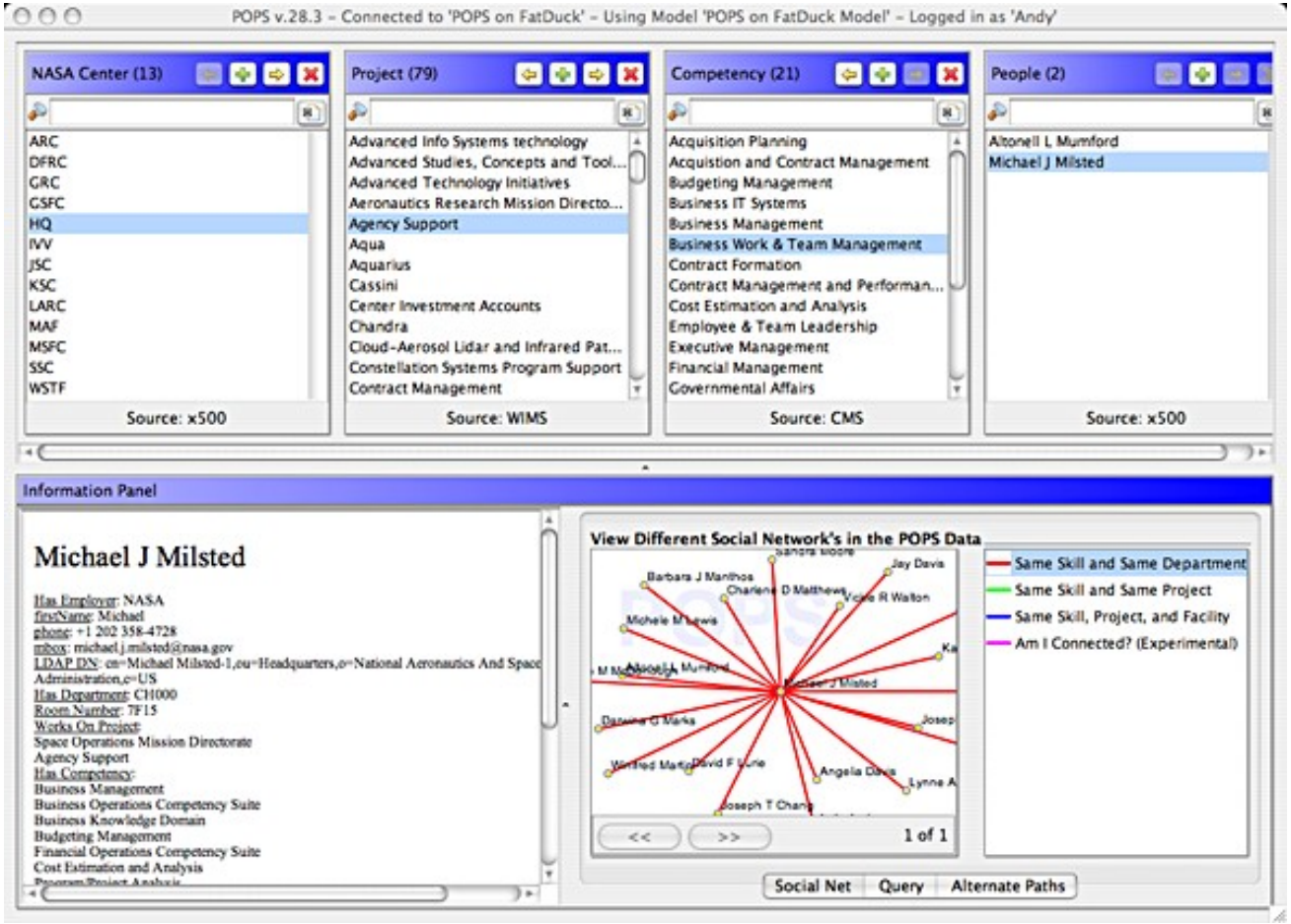




Courtesy of Huajun Chen, Zhejiang University, (SWEO Case Study)

> Find the right experts at NASA

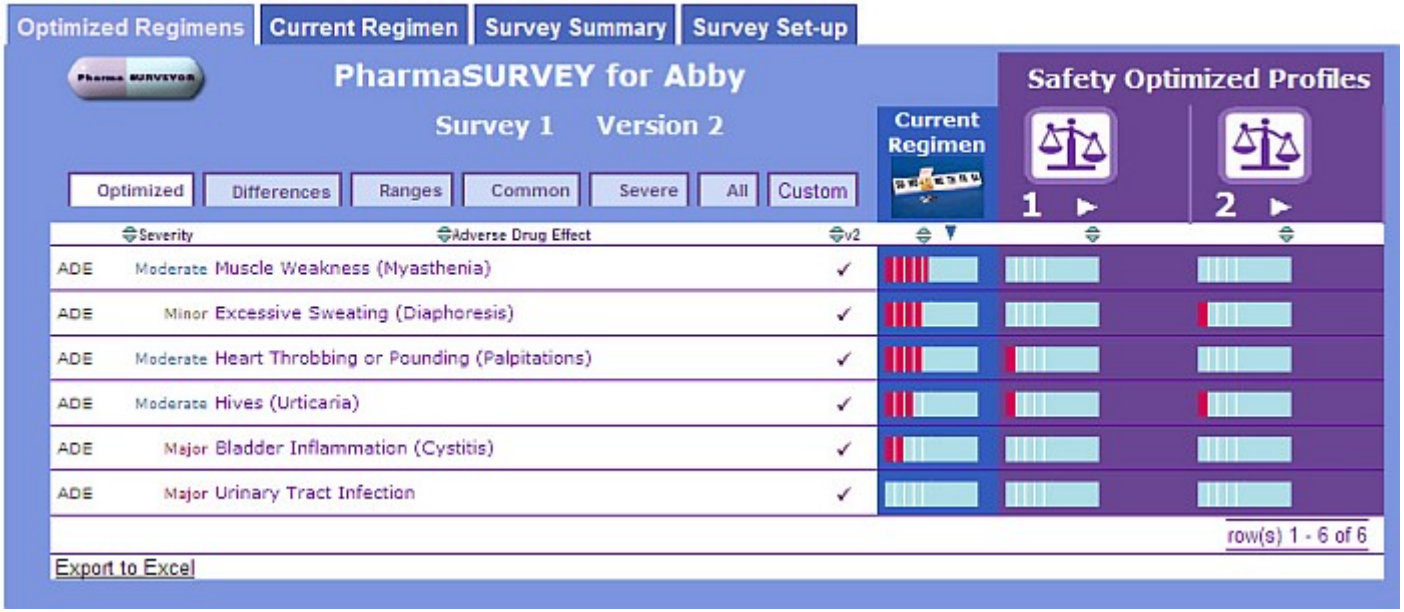
- Expertise locater for nearly 20,000 NASA civil servants using integration techniques over 6 or 7 geographically distributed databases, data sources, and web services…



Courtesy of Kendall Clark, Clark & Parsia, LLC

> Help in choosing the right drug regimen

- Help in finding the best drug regimen for a specific case
  - find the best trade-off for a patient
- Use an ontology for medical conditions, signs, symptoms
- Integrate data from various sources (patients, physicians, Pharma, researchers, etc)



Courtesy of Erick Von Schweber, PharmaSURVEYOR Inc., (SWEO Use Case)

> Basic RDF

> RDF triples

- Let us begin to formalize what we did!
  - we “connected” the data…
  - but a simple connection is not enough… it should be named somehow
  - hence the RDF Triples: *a labeled 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
    - "**s**", "**p**", and "**o**" stand for "subject", "predicate", and "object", respectively
    - 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/original**
  - here is the complete triple:

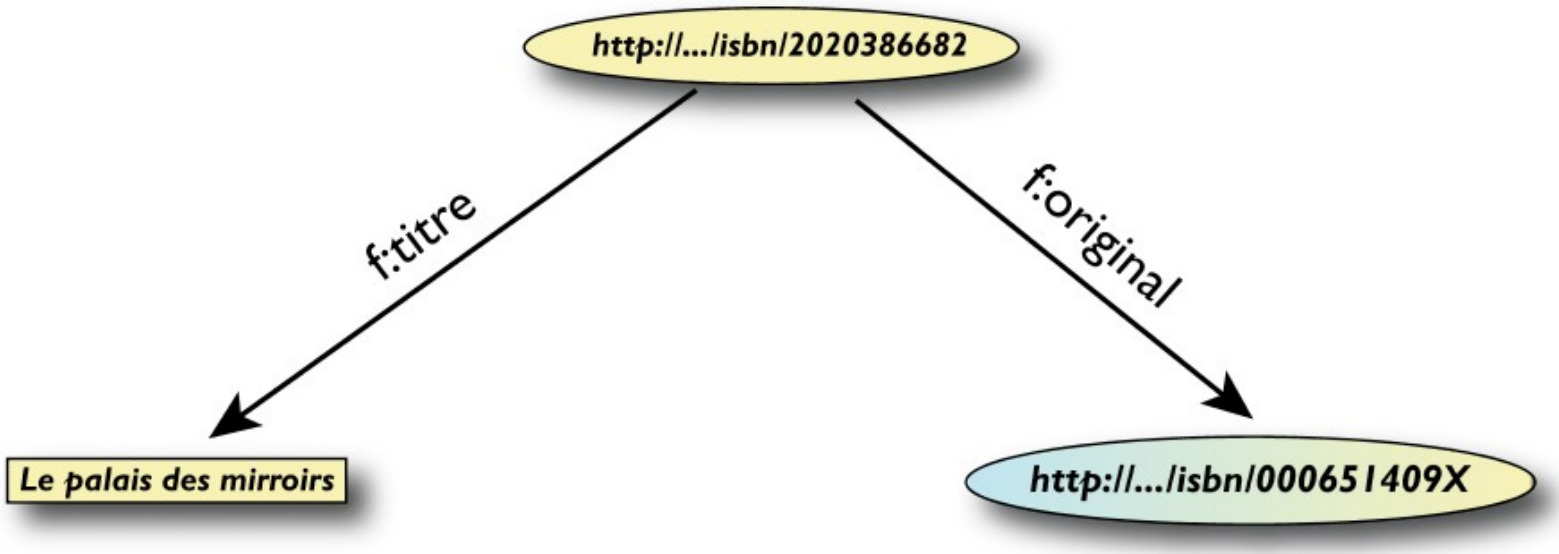
```
(<http://…isbn…6682>, <http://…/original>, <http://…isbn…409X>)
```

- *RDF* is a general model for such triples (with machine readable formats like RDF/XML, Turtle, n3, RXR, …)
- … *and that's it!*

> RDF triples (cont.)

- The "**p**" is also referred to as "property" in some cases
- Resources can use any URI; 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#element(home)**
  - **http://www.example.org/file.html#home**
- RDF triples form a directed, labeled graph (best way to think about them!)

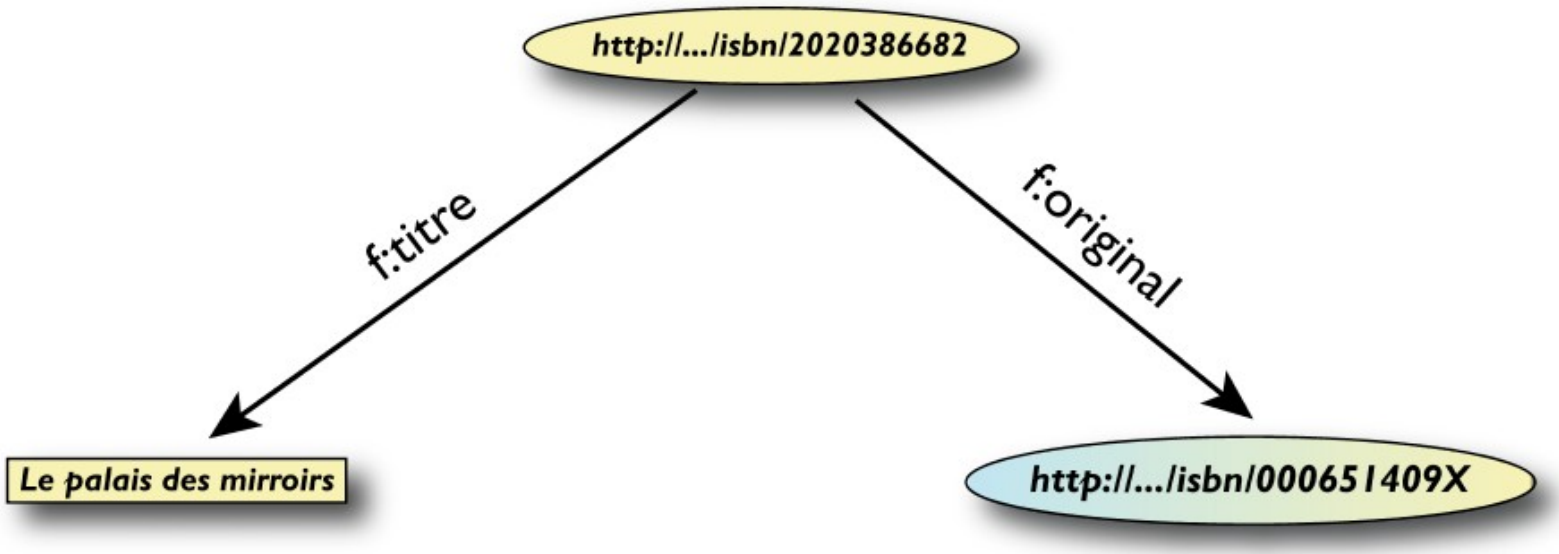
> A simple RDF example (in RDF/XML)




```
<rdf:Description rdf:about="http://…/isbn/2020386682">
    <f:titre xml:lang="fr">Le palais des mirroirs</f:titre>
    <f:original rdf:resource="http://…/isbn/000651409X"/>
</rdf:Description>
```

(Note: namespaces are used to simplify the URI-s)

> A simple RDF example (in Turtle)



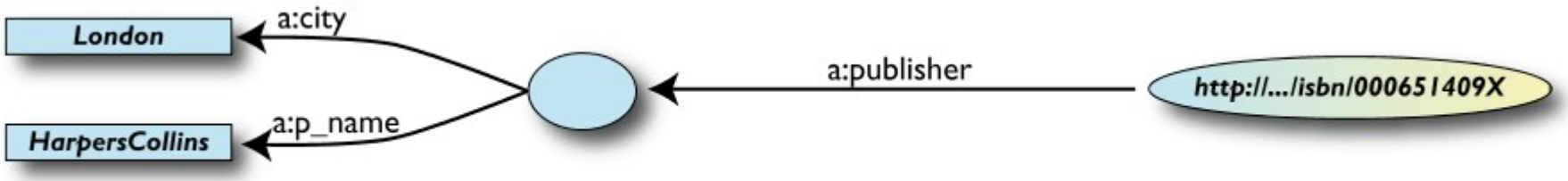

```
<http://…/isbn/2020386682>
    f:titre "Le palais des mirroirs"@fr;
    f:original <http://…/isbn/000651409X>.
```

> URI-s play a fundamental role

- *URI-s made the merge possible*
- Anybody can create (meta)data on any resource on the Web
  - e.g., the same XHTML 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 provide a syntax for naming and ground RDF into the Web*
  - information can be retrieved using existing tools
  - this makes the “Semantic Web”, well… “Semantic *Web*”

> “Internal” nodes

- Consider the following statement:
  - “the publisher is a «thing» that has a name and an address”
- Until now, nodes were identified with a URI. But…
- …what is the URI of «thing»?



> One solution: create an extra URI

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:resource="urn:uuid:f60ffb40-307d-…"/>
</rdf:Description>
<rdf:Description rdf:about="urn:uuid:f60ffb40-307d-…">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</rdf:Description>
```

- The resource will be “visible” on the Web as all other resources
  - care should be taken to define *unique* URI-s (hence the UUID in the example)
- Serializations may give syntactic help to define local URI-s (much like the id-s in HTML)

> Internal identifier (blank nodes)

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:nodeID="A234"/>
</rdf:Description>
<rdf:Description rdf:nodeID="A234">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</rdf:Description>
```

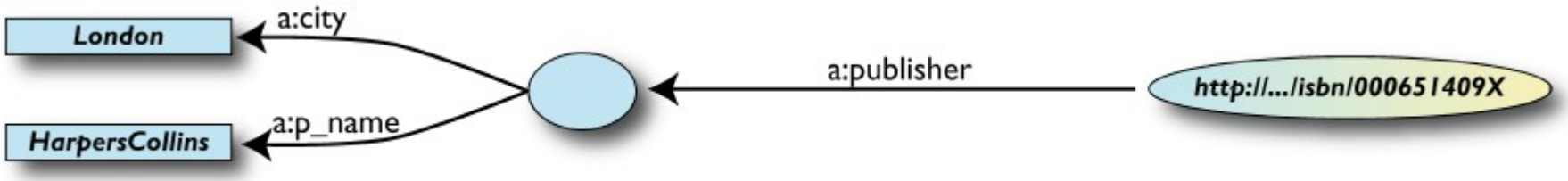
```
<http://…/isbn/2020386682> a:publisher _:A234.
_:A234 a:p_name "HarpersCollins".
```

- The exact syntax depends on the serialization format
- A234 is invisible from outside (it is not a “real” URI!); it is an internal identifier for a resource

> Blank nodes: the system can also do it

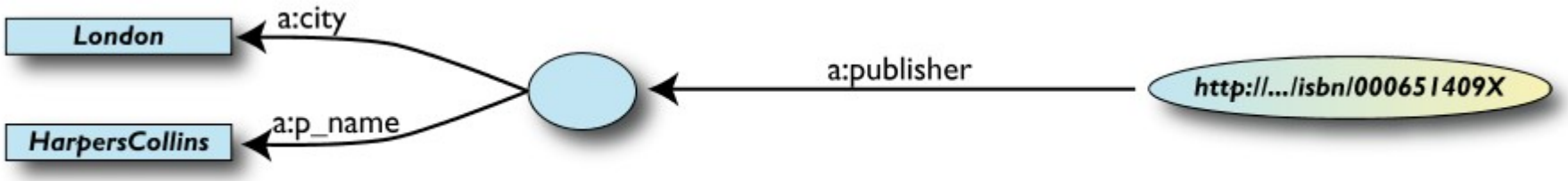
- Let the system create a “nodeID” internally (you do not really care about the name…)

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
  <a:publisher>
      <rdf:Description>
          <a:p_name>HarpersCollins</a:p_name>
          …
      </rdf:Description>
  </a:publisher>
</rdf:Description>
```



> Same in Turtle

```
<http://…/isbn/000651409X> a:publisher [
    a:p_name "HarpersCollins";
    …
].
```



> Blank nodes: some more remarks

- Blank nodes require attention when merging
  - blanks nodes with identical nodeID-s in *different* graphs are *different*
  - implementation must be be careful with its naming schemes when merging
- Many applications prefer not to use blank nodes and define new URI-s “on-the-fly”
  - eg, when triples are in a database
- You can think of blank nodes as representing an “existential” statement (“there is a resource such that…”)

> 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);
}
```

> RDF schemas

> Need for RDF schemas

- This is the simple form of our “extra knowledge”:
  - define the terms we can use
  - what restrictions apply
  - what extra relationships are there?
- This is where RDF Schemas come in
  - officially: “RDF Vocabulary Description Language”; the term “Schema” is retained for historical reasons…

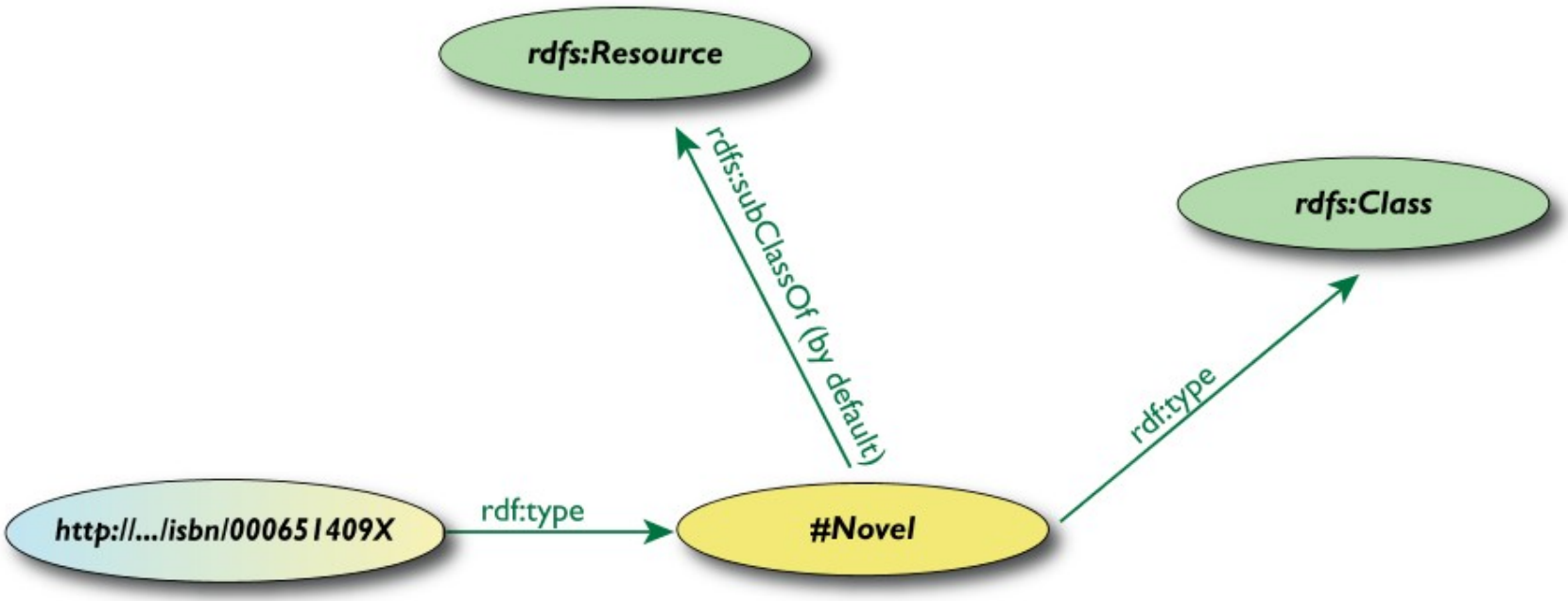
> Classes, resources, …

- Think of well known traditional ontologies or taxonomies:
  - use the term “novel”
  - “every novel is a fiction”
  - “«The Glass Palace» is a novel”
  - 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”)
    - “fiction”, “novel”, …

> Classes, resources, … (cont.)

- Relationships are defined among classes/resources:
  - "typing": an individual belongs to a specific class ("«The Glass Palace» is a novel")
    - to be more precise: "«http://.../**000651409X**» is a novel"
  - "subclassing": *all* instances of one are also the instances of the other ("every novel is a fiction")
- *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 part (“application’s data types”):

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

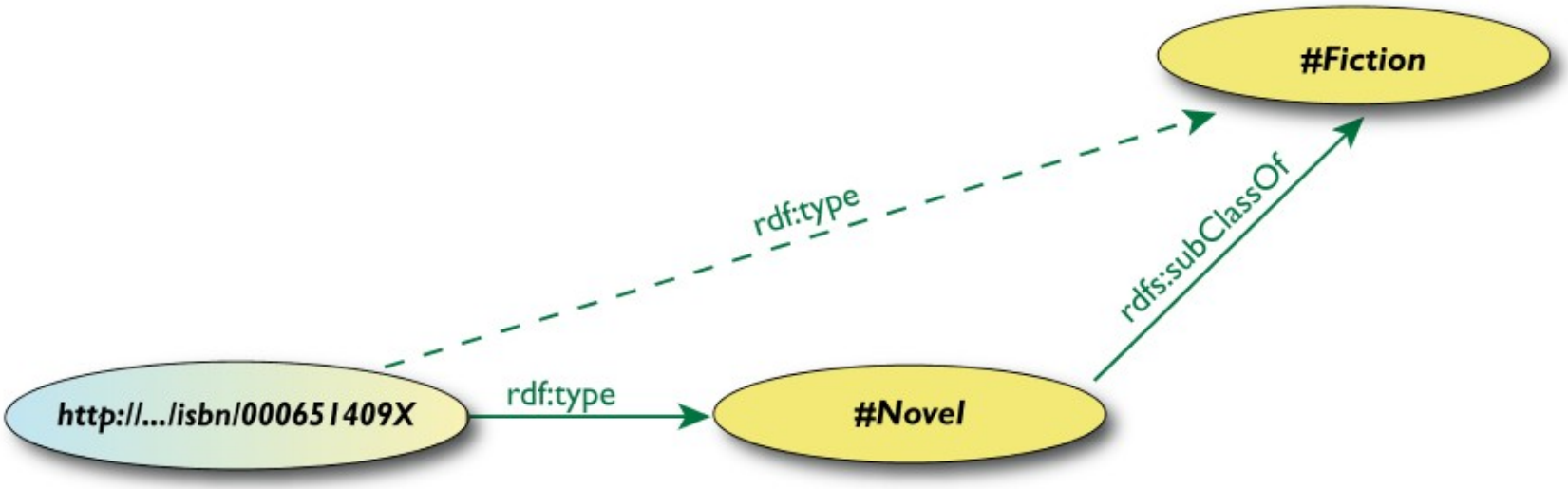
- The RDF data on a specific novel (“using the type”):

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <rdf:type rdf:resource="http://…/bookSchema.rdf#Novel"/>
</rdf:Description>
```

> Further remarks on types

- A resource may belong to several classes
  - rdf:type is just a property…
  - "«The Glass Palace» is a novel, but «The Glass Palace» is also an «inventory item »…"
- 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…
- (remember the "Person" in our example?)

> Inferred properties




```
(<http://…/isbn/000651409X> rdf:type #Fiction)
```

- is not in the original RDF data…
- …but can be inferred from the RDFS rules
- better (“RDFS aware”) RDF environments return that triples, too

> Inference: let us be formal…

- The RDF Semantics document has a list of (44) *entailment rules*:
  - "if such and such triples are in the graph, add this and this triple"
  - do that recursively until the graph does not change
- 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

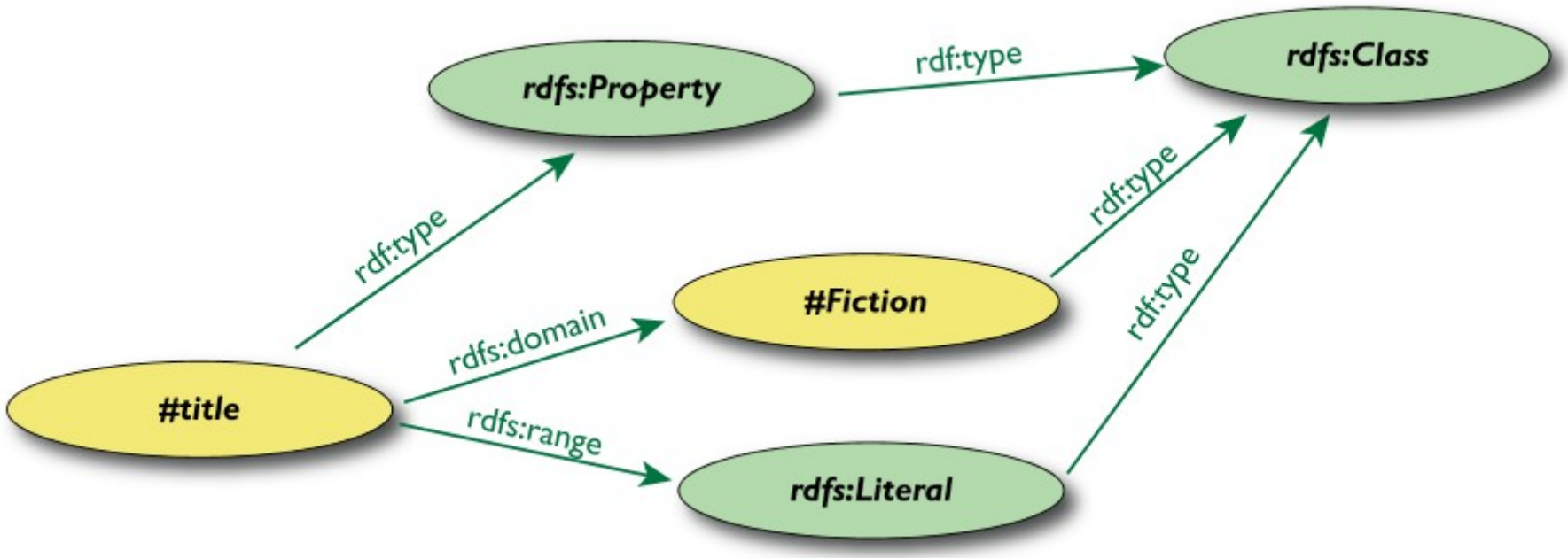
> Properties

- Property is a special class (**`rdf:Property`**)
  - properties are also resources identified by URI-s
- Properties' range and domain can be specified
  - i.e., what type of resources can serve as object and subject
  - an important purpose: to license inferences
    - I can *infer* that, say, the object is of a specific type
- 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:
  - **`P`** is a property
  - **`C`** is a class instance
  - 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



> Property specification serialized

- In XML/RDF:

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

- In Turtle:

```
:title
  rdf:type     rdf:Property;
  rdfs:domain  :Fiction;
  rdfs: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

> A bit of RDFS can take you far…

- Remember the power of merge?
- We could have used, in our example:
  - **`f:auteur`** is a subproperty of **`a:author`** and vice versa (although we will see other ways to do that…)
- Of course, in some cases, more complex knowledge is necessary (see later…)

> Simple Knowledge Organization System

> Simple Knowledge Organization System

- Goal: 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
- This is where *SKOS* comes in: define classes, properties, where those structures can be added

> Example: entries in a glossary

Assertion

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

Class

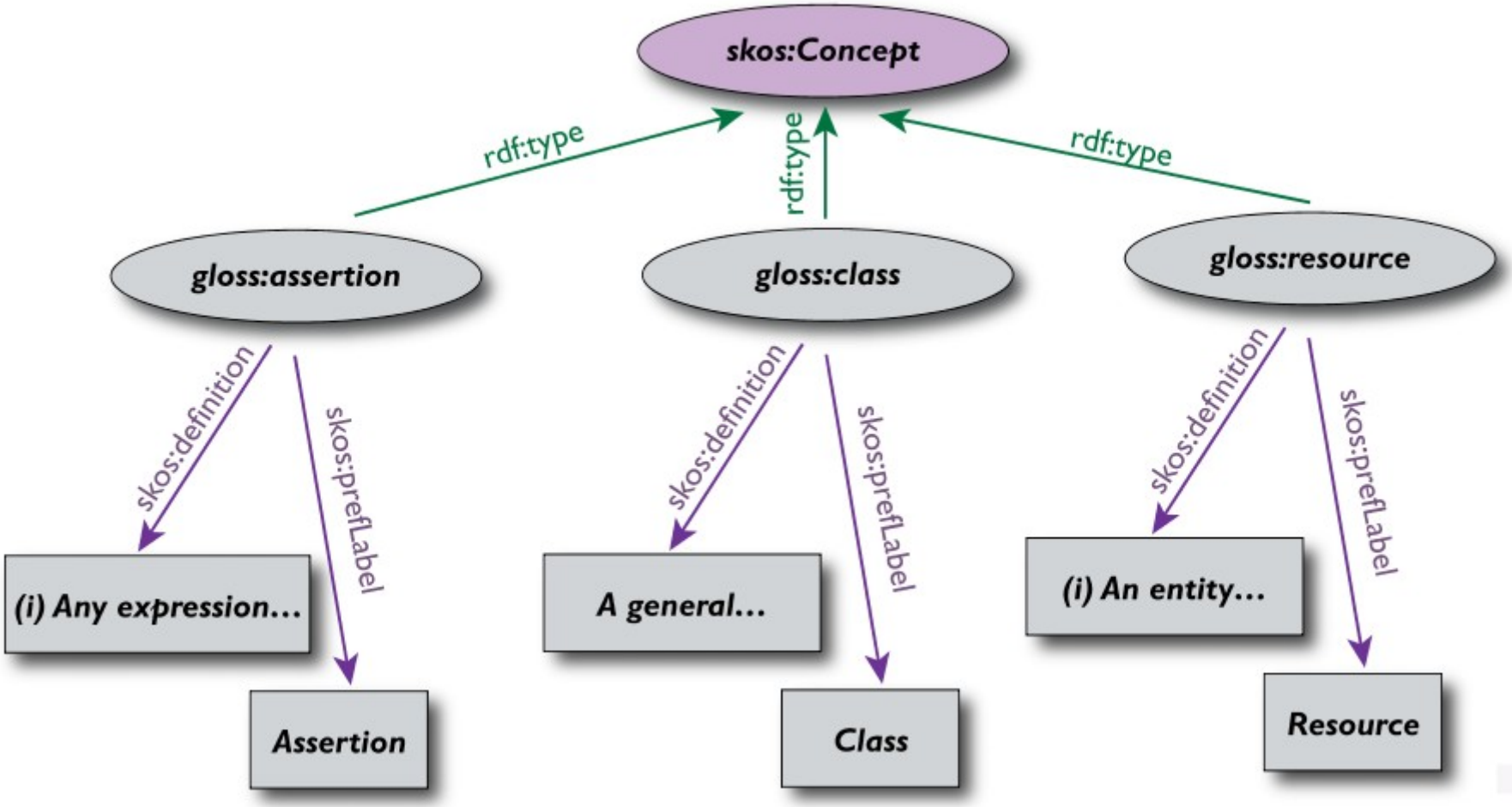
A general concept, category or classification. Something used primarily to classify or categorize other things.

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 in SKOS



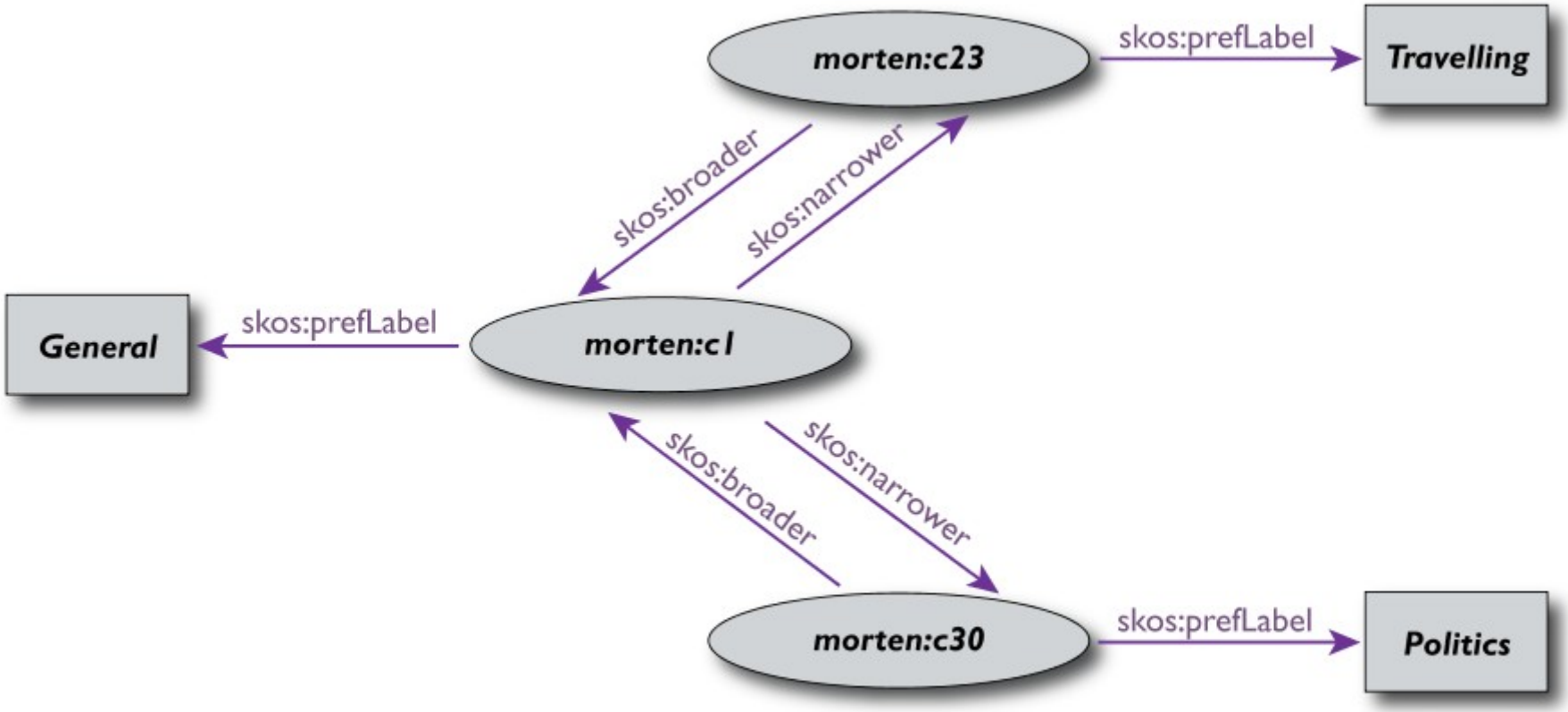
> Example: taxonomy

General
- Traveling
- Politics

SemWeb
- RDF
  - SKOS

(from MortenF's blog categories; note that the categorization is arbitrary!)

> Example: taxonomy in SKOS

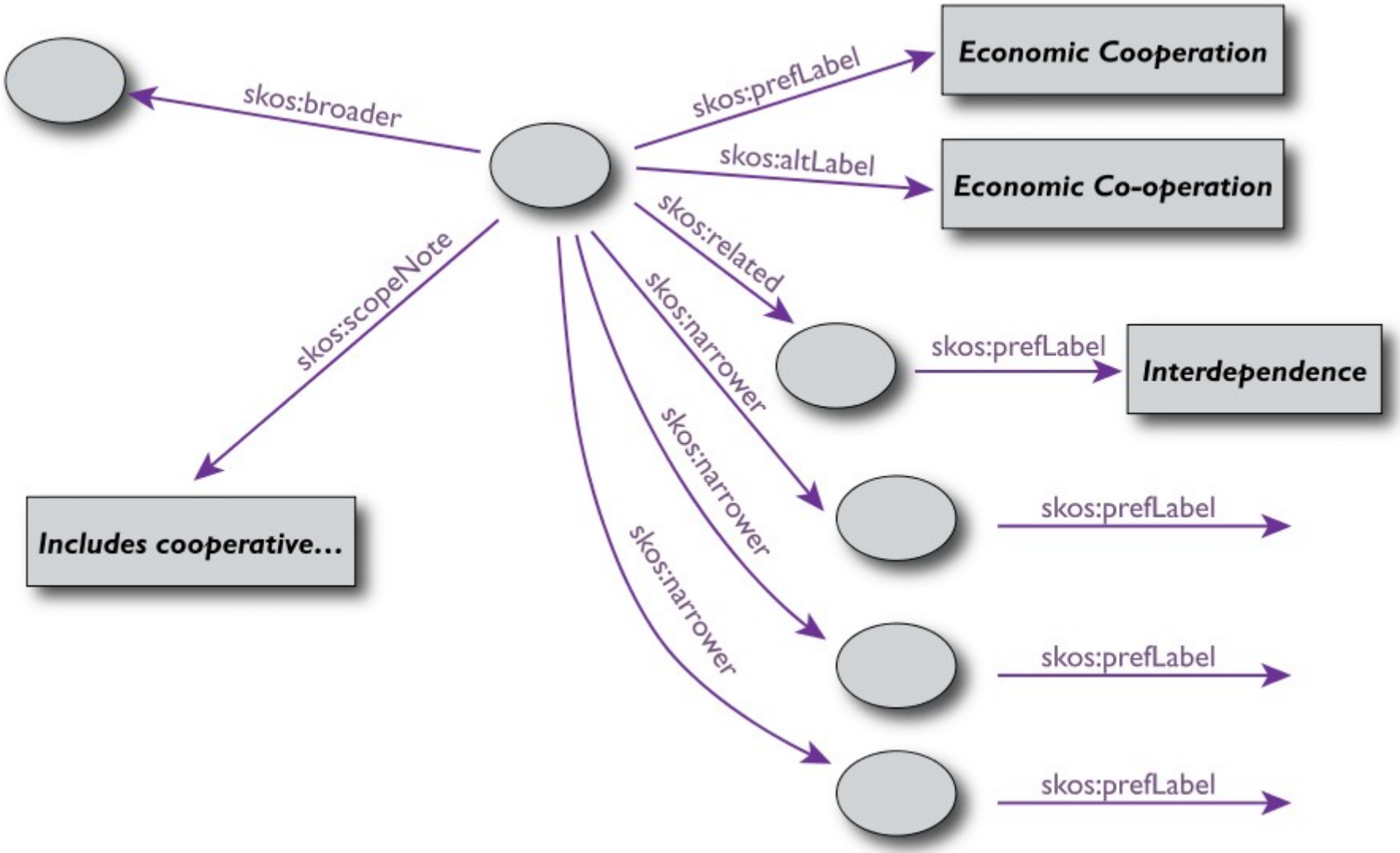


> Example: thesaurus

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 the UK Archival Thesaurus)

> Example: thesaurus in SKOS



> SKOS Core overview

- Classes and Predicates:
  - Basic description (**`Concept`**, **`ConceptScheme`**, …)
  - Labeling (**`prefLabel`**, **`altLabel`**, …)
  - Documentation (**`definition`**, **`scopeNote`**, **`changeNote`**, …)
  - Semantic relations (**`broader`**, **`narrower`**, **`related`**)
  - Subject indexing (**`subject`**, **`isSubjectOf`**, …)
  - Grouping (**`Collection`**, **`OrderedCollection`**, …)
  - Subject Indicator (**`subjectIndicator`**)
- Some simple inference rules (a bit like the RDFS inference rules) to define some extra semantics
- A bit of warning: SKOS is still evolving!

> RDF data access, a.k.a. query (SPARQL)

> Querying RDF graphs

- Remember the Jena idiom:

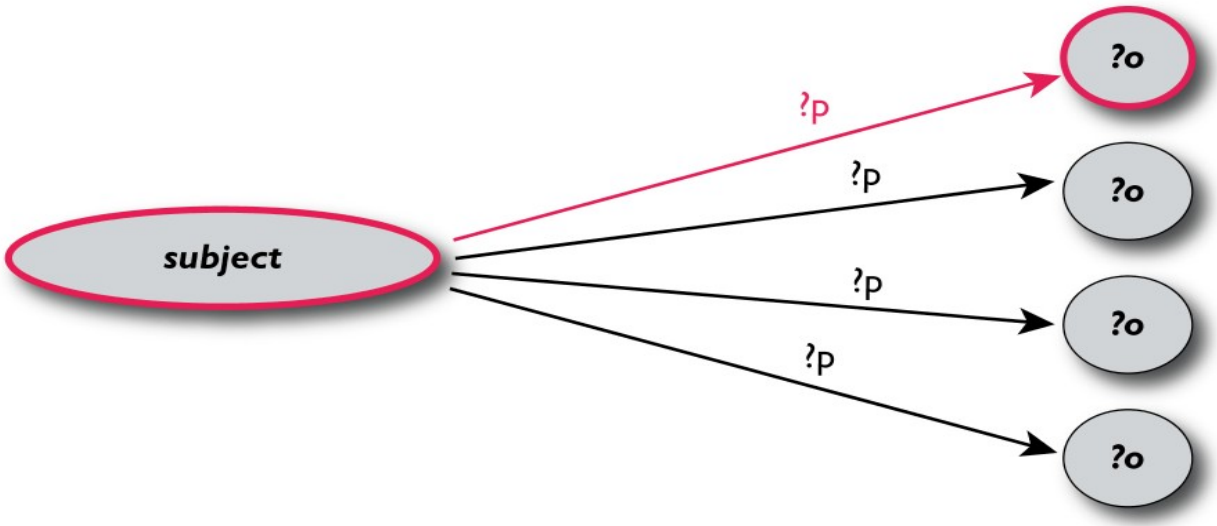
```
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty(); o = st.getObject();
    do_something(p,o);
```

- 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")



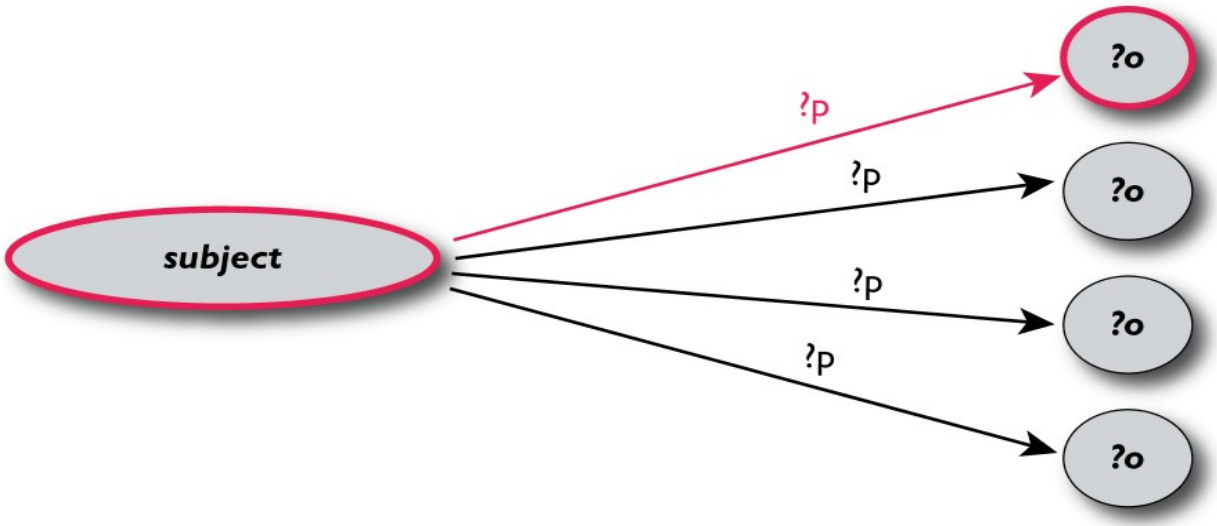
> General: graph patterns

- 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

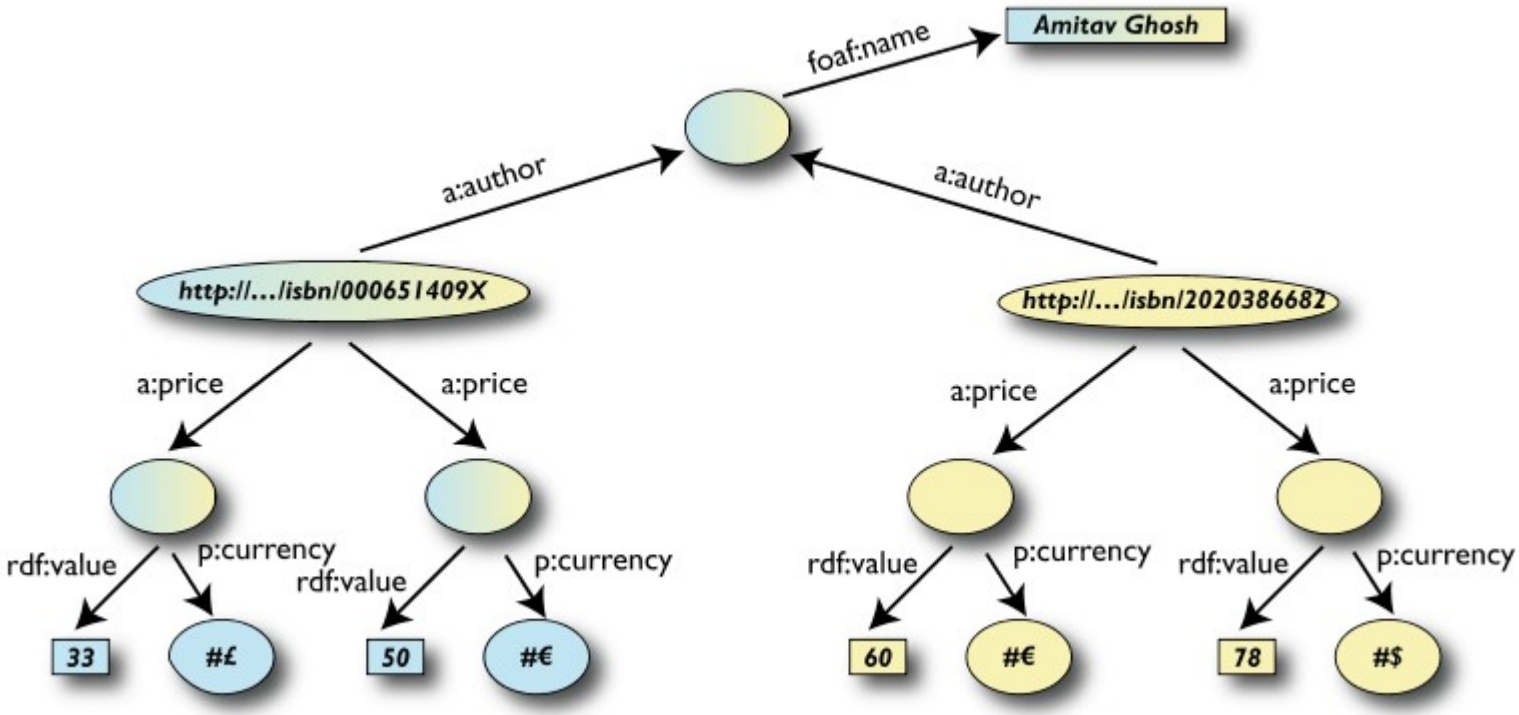
```
SELECT ?p ?o
WHERE {subject ?p ?o}
```

- The triples 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

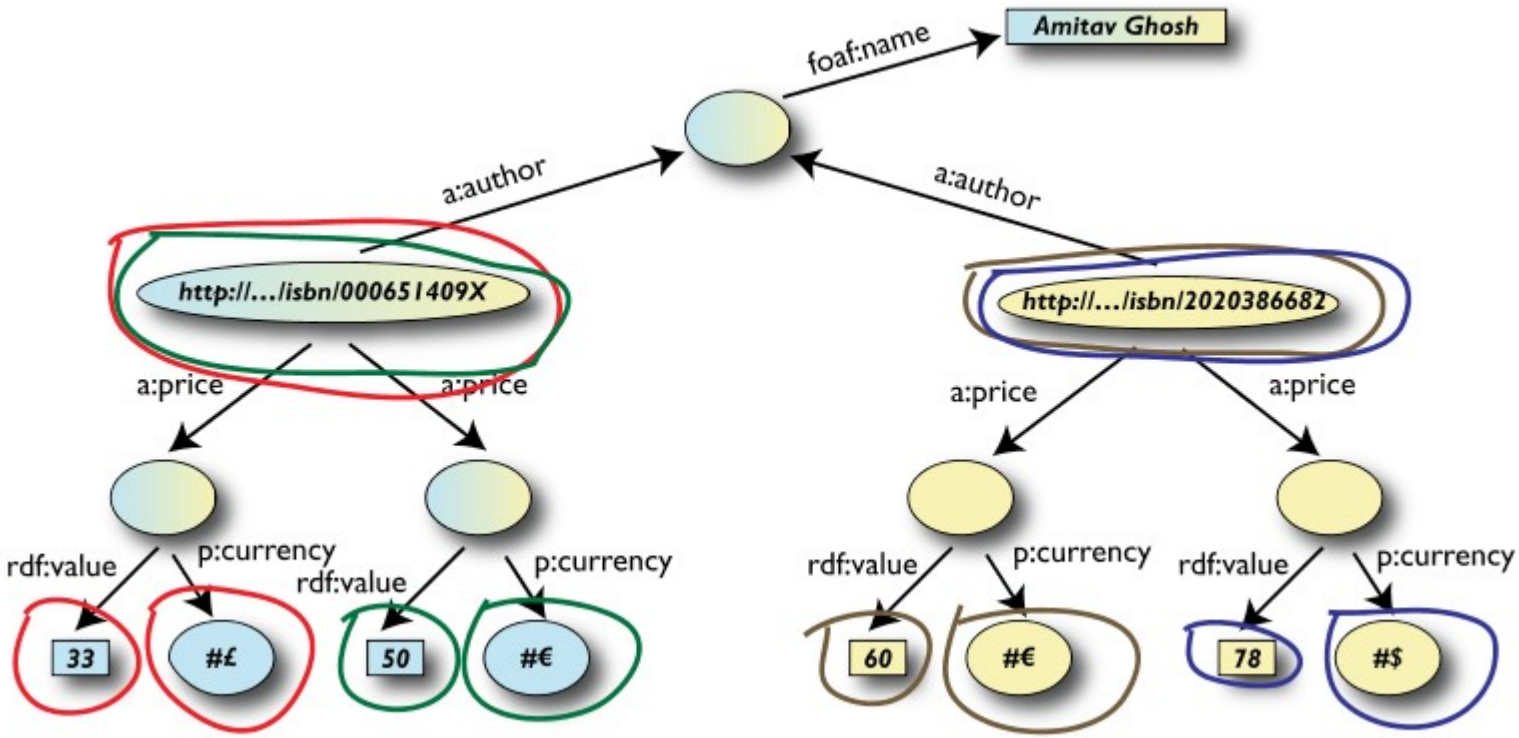
```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```



> Simple SPARQL example

```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```

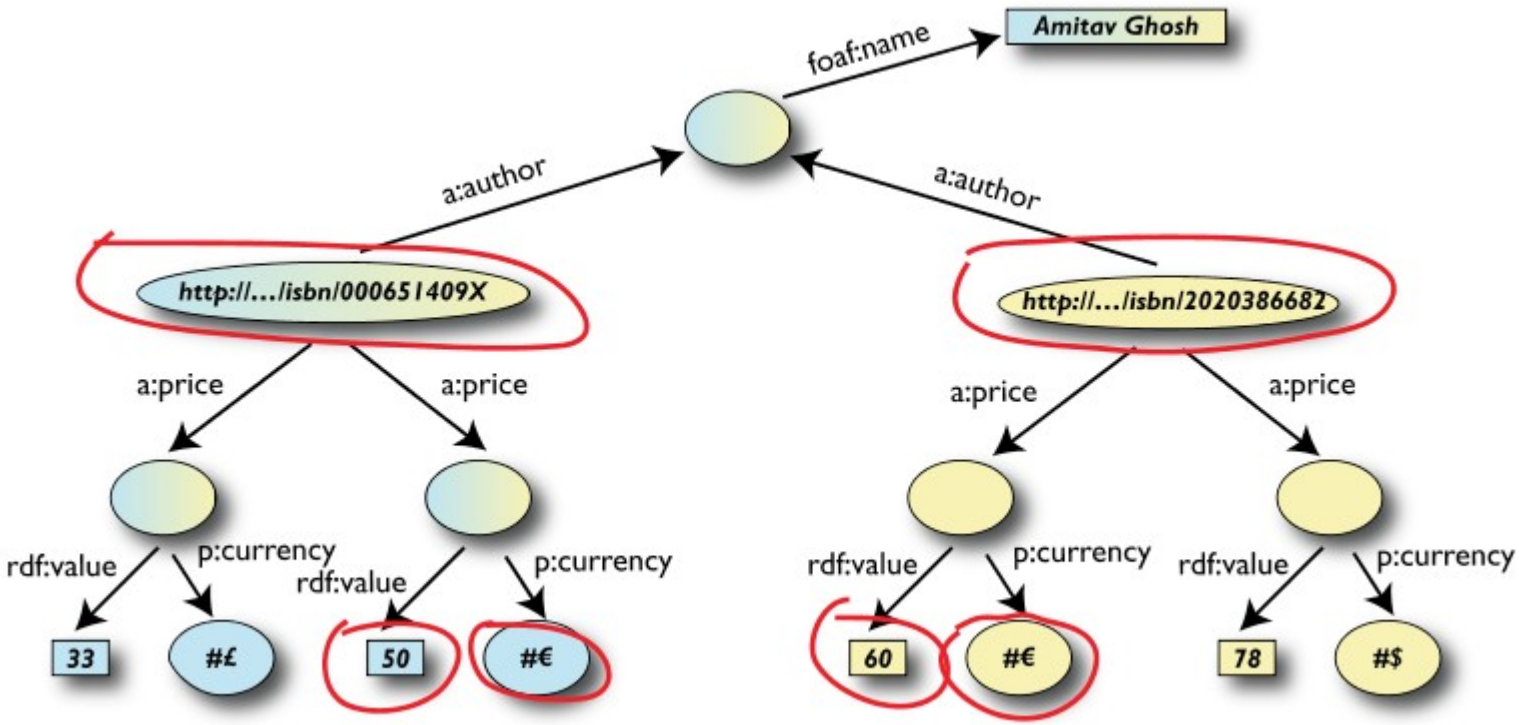
- Returns:
  [[<..49X>,33,£], [<..49X>,50,€], [<..6682>,60,€], [<..6682>,78,$]]



> Pattern constraints

```
SELECT ?isbn ?price ?currency # note: not ?x!
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.
        FILTER(?currency == € }
```

- Returns: [[<..409X>,50,€], [<..6682>,60,€]]



> Other SPARQL features

- Limit the number of returned results; remove duplicates, sort them, …
- Optional subpatterns (match if possible, return empty bindings otherwise)
- Specify several data sources (via URI-s) within the query (essentially, a merge!)
- *Construct* a graph combining a separate pattern and the query results, or simply *ask* whether a pattern matches
- Use datatypes and/or language tags when matching a pattern

> SPARQL usage in practice

- Locally, i.e., bound to a programming environments like Jena
  - less and less typical…
- Remotely, e.g., over the network
  - separate documents define the protocol and the result format
    - SPARQL Protocol for RDF with HTTP and SOAP bindings
    - SPARQL results in XML or JSON formats
  - big datasets often offer “SPARQL endpoints” for this protocol
    - In some cases one can query the local data only

> Get to RDF(S) data

> RDF can 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 meta elements
- … and then generate RDF automatically (e.g., via an XSLT script)
- This is similar to what “microformats” do (without referring to RDF, though)
  - they may not extract RDF but use the data directly instead in Web 2.0 applications, but the application is not all that different
  - other applications may extract it to yield RDF (e.g., RSS1.0)

> 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 meta-s, class id-s, etc…)
- …but, by using the **`profile`** attribute, a client is instructed to find and run the transformation processor automatically
- There is a mechanism for XML in general
  - a transformation can also be defined on an XML schema level
- A “bridge” to “microformats”
- Recommendation planned in September 2007

> Another upcoming solution: RDFa

- 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 foaf:name">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 an RDFa processor:

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

> RDFa (cont.)

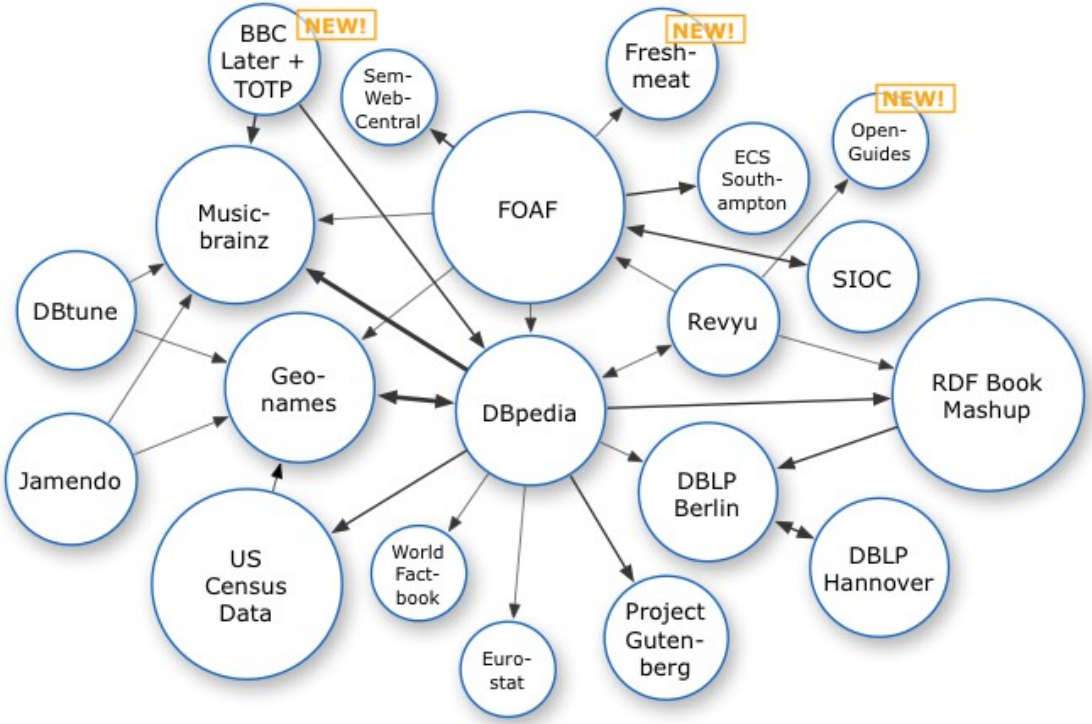
- RDFa extends (X)HTML a bit by:
  - defining general attributes to add metadata to any elements (a bit like the class in microformats, but via dedicated properties)
  - provides an almost complete “serialization” of RDF in XHTML
- It is a bit like the microformats approach but with more rigor and fully generic
  - makes it easy to mix different vocabularies, which is not that easy with microformats

> Bridge to relational databases

- Most of the data are stored in relational databases
  - “RDFying” them may be an impossible task
- “Bridges” are being defined:
  - a layer between RDF and the database
  - RDB tables are “mapped” to RDF graphs, possibly on the fly
  - in some cases the mapping is generic (columns represent properties, cells are, e.g., literals or references to other tables via blank nodes)…
  - … in other cases separate mapping files define the details
- SPARQL is becoming the tool of choice to query that data
  - ie, “SPARQL endpoints” are defined to query it

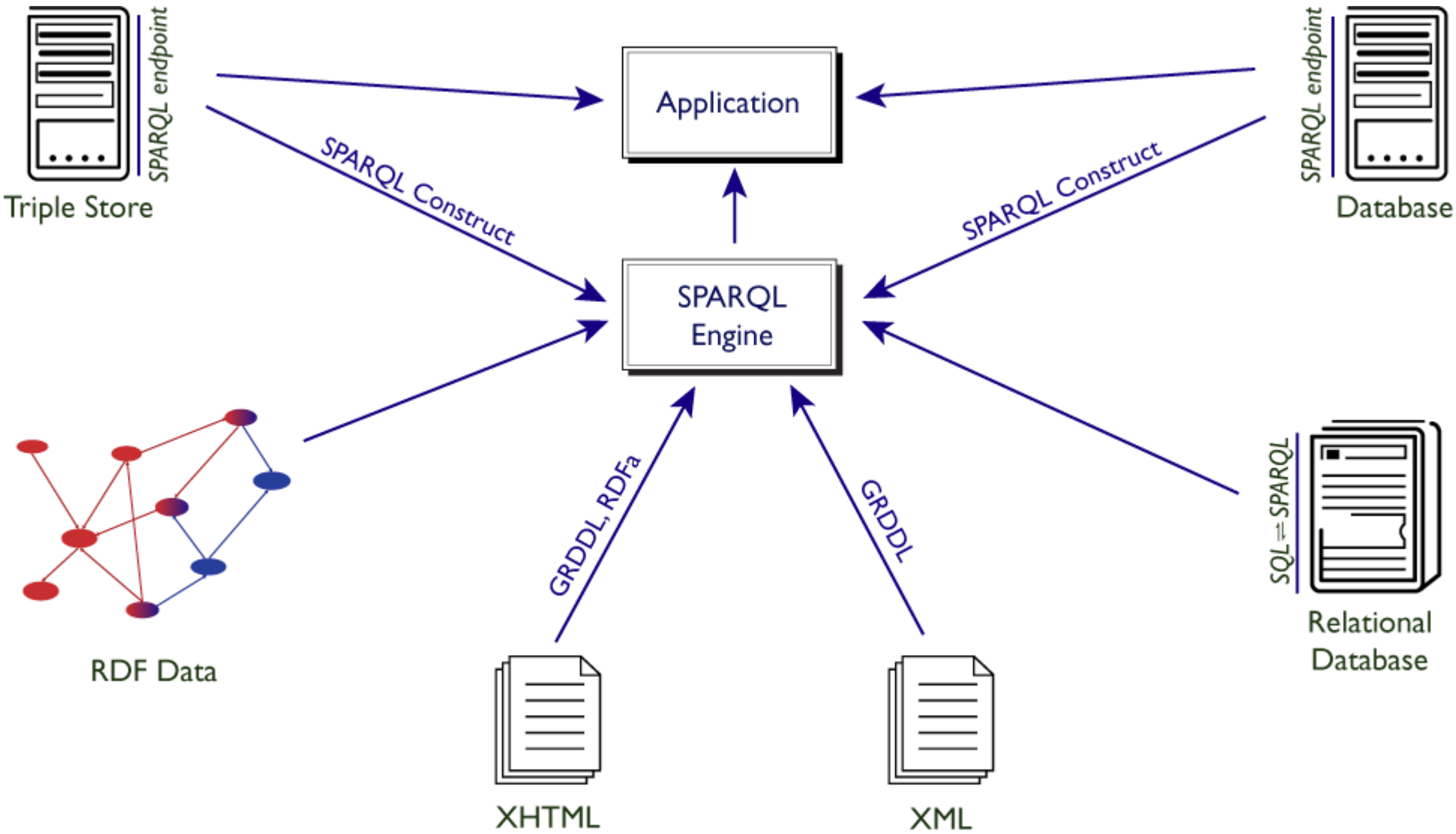
> Eg: Linking Open Data Community Project

- "Expose" open datasets in RDF
- *Set RDF links among the data items* for different datasets
- Set up SPARQL endpoints to query the data
- Over 2 billion triples served so far (August 2007)



Courtesy of Chris Bizer and Richard Cyganiak, Free University of Berlin

> SPARQL as a unifying point



> Ontologies (OWL)

> Ontologies

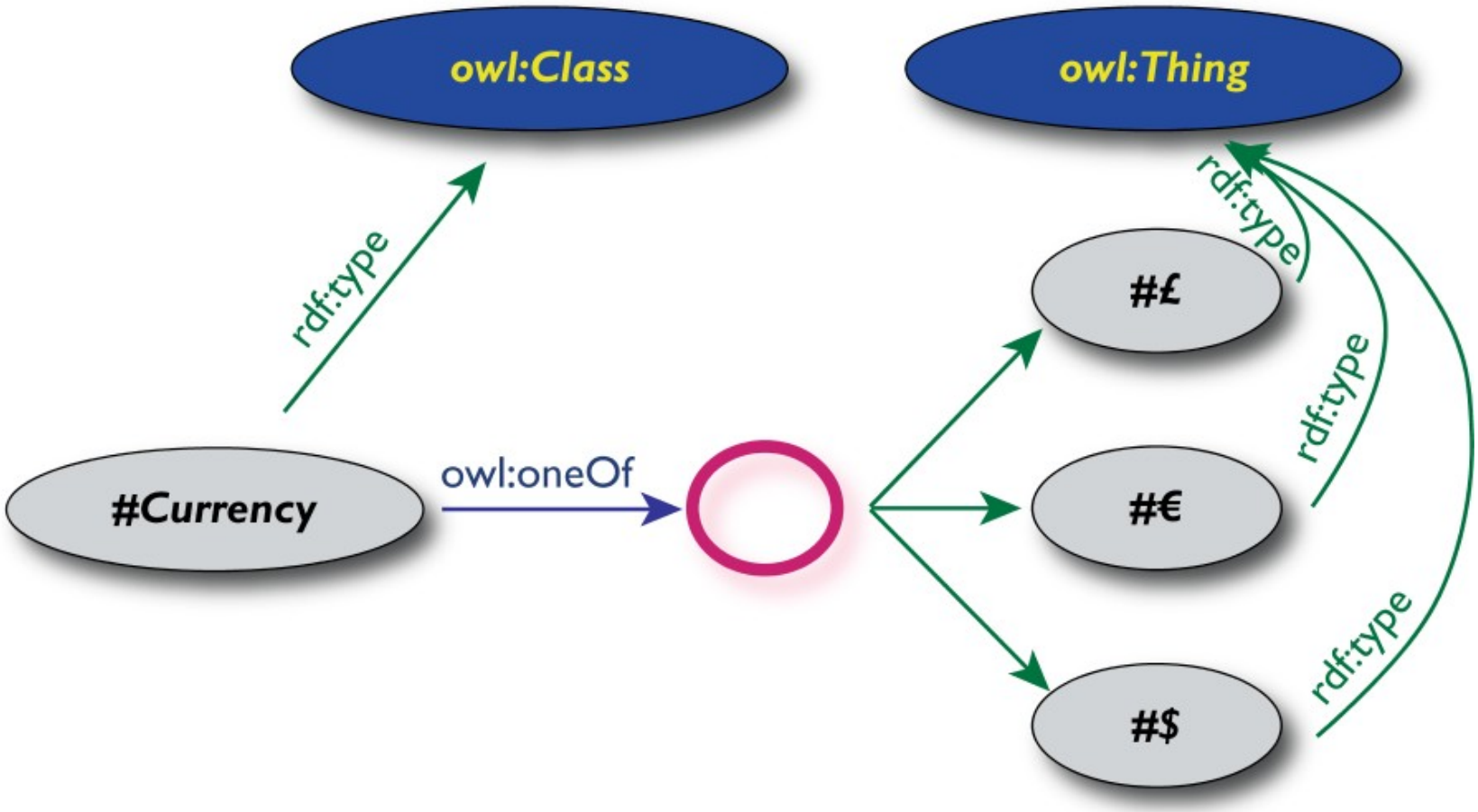
- RDFS is useful, but does not solve all possible requirements
- Complex applications may want more possibilities:
  - similarity and/or differences of terms (properties or classes)
  - construct classes, not just name them
  - can a program reason about some terms? E.g.:
    - "if «Person» resources «A» and «B» have the same «**`foaf:email`**» property, then «A» and «B» are identical"
  - etc.
- This lead to the development of OWL (Web Ontology Language)

> 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

> OWL classes can be "enumerated"

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



> Same serialized

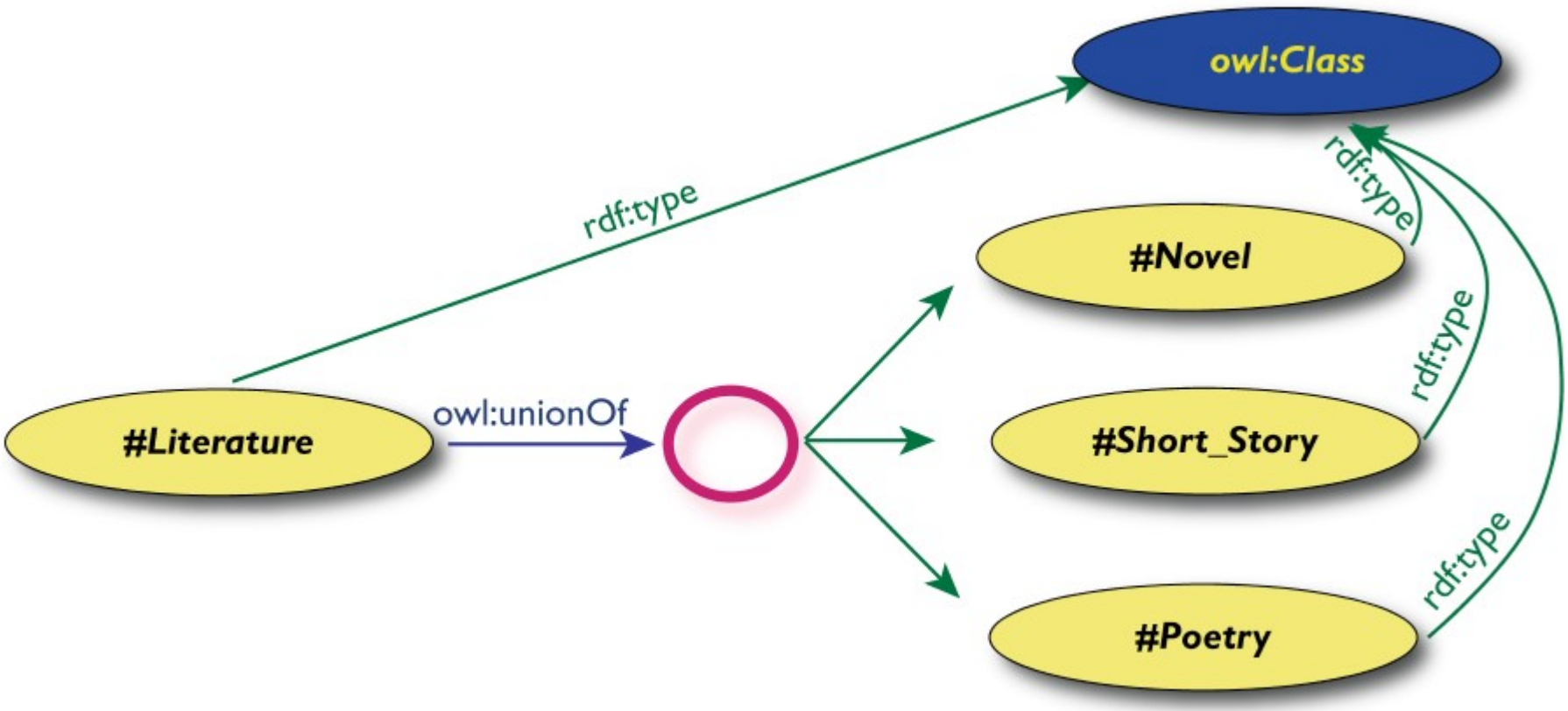
```
<owl:Class  rdf:ID="Currency">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:ID="£"/>
    <owl:Thing rdf:ID="€"/>
    <owl:Thing rdf:ID="$"/>
    …
  </owl:oneOf>
</owl:Class>
```

```
:£ rdf:type owl:Thing.
:€ rdf:type owl:Thing.
:$ rdf:type owl:Thing.
:Currency
    rdf:type owl:Class;
    owl:oneOf (:€ :£ :$).
```

- The class consists of *exactly* of those individuals

> Union of classes

- Essentially, like a set-theoretical union:



> Same serialized

```
:Novel            rdf:type owl:Class.
:Short_Story      rdf:type owl:Class.
:Poetry           rdf:type owl:Class.
:Literature rdf:type owlClass;
   owl:unionOf (:Novel :Short_Story :Poetry).
```

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

> Property restrictions

- (Sub)classes created by restricting the property values on that class
- For example, "a *listed* price is a price which is either in €, £, or $" means:
  - the value of "**`p:currency`**" when applied to the price resource must take one of those values…
  - …thereby define the class of "listed price"

> Property restrictions in OWL

- Restriction may be by:
  - value constraints (ie, further restrictions on the range)
    - *all* values must be from a class (like the price example)
    - *some* value must be from a class
  - cardinality constraints (ie, how many times the property is used on an instance?)
    - minimum cardinality
    - maximum cardinality
    - exact cardinality

> Somewhat more formally

```
<owl:Class rdf:ID="Listed_Price">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://…#currency"/>
        <owl:allValuesFrom rdf:resource="#Currency">
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

```
:Listed_Price rdf:type owl:Class;
    rdfs:subClassOf [
      rdf:type           owl:Restriction;
      owl:onProperty     <http://…#currency>;
      owl:allValuesFrom  :Currency.
   ].
```

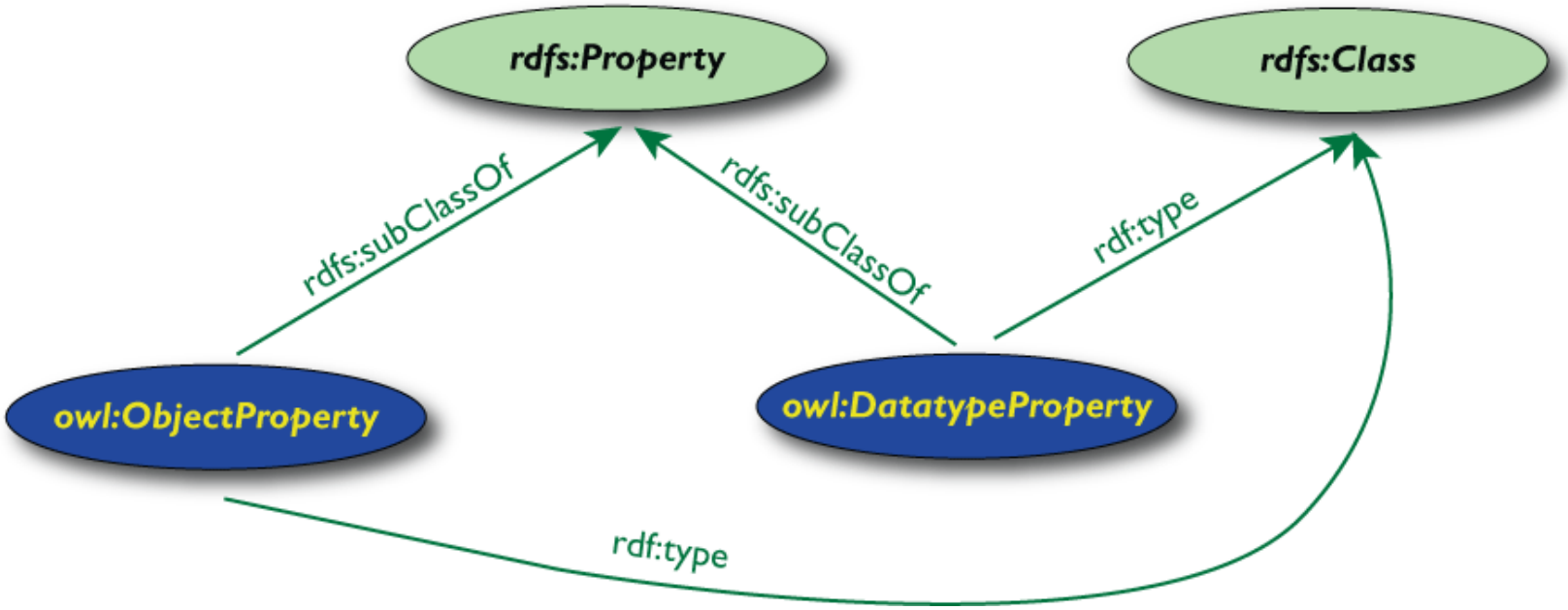
- "**allValuesFrom**" could be replaced by "**someValuesFrom**", "**cardinality**", "**minCardinality**", or "**maxCardinality**"

> A word of warning…

- Cardinality restrictions are *not* used as syntactic restrictions to "reject" RDF data
  - eg, because not enough properties are set
- It means: "the remaining relations are out there somewhere" even if not all are known…

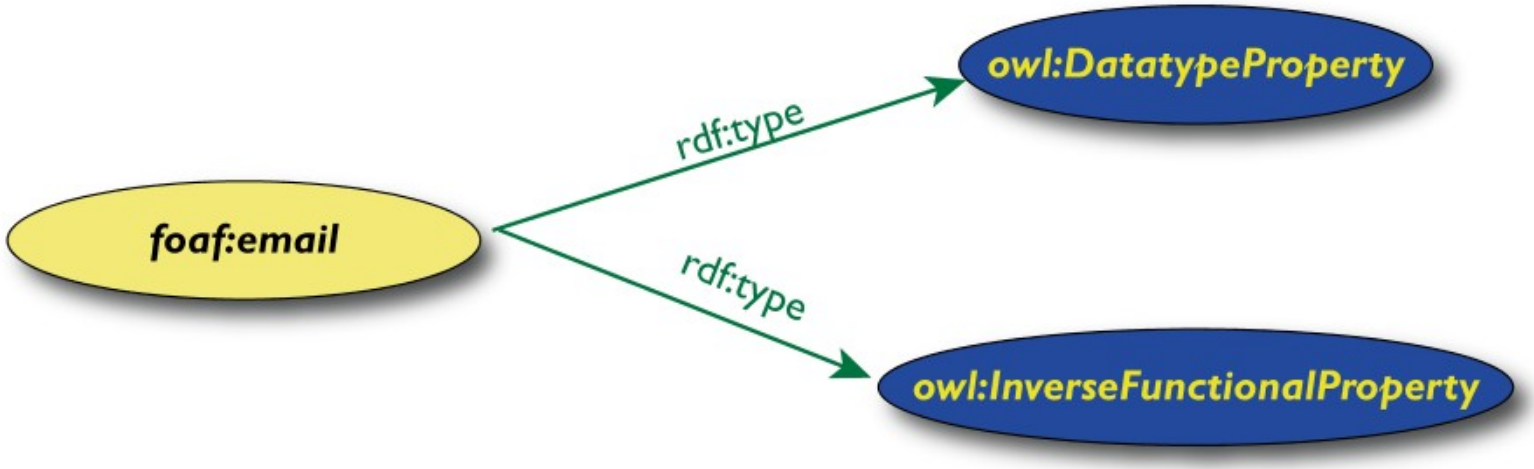
> Property characterization

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



> Characterization example

- "**`foaf:email`**" is inverse functional (i.e., two different subjects cannot have identical objects)



- Could be "**`FunctionalProperty`**", "**`TransitiveProperty`**", "**`SymmetricProperty`**"

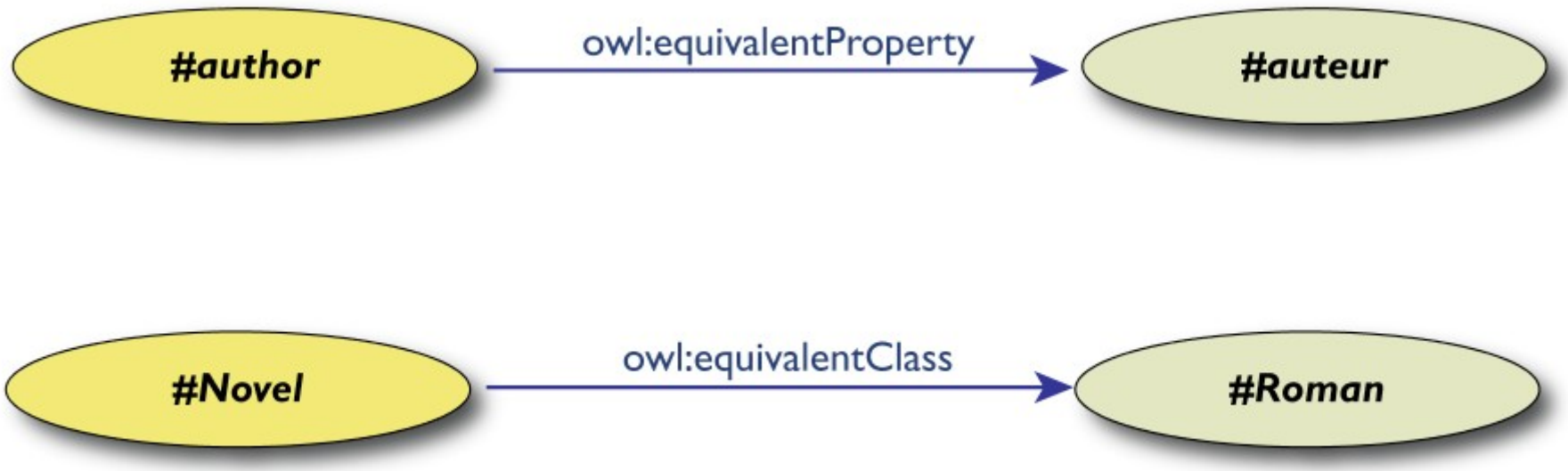
> OWL: additional requirements

- 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

- For classes:
  - **`owl:equivalentClass`**: two classes have the same individuals
  - **`owl:disjointWith`**: no individuals in common
- For properties:
  - **`owl:equivalentProperty`**
    - remember the **`a:author`** vs. **`f:auteur`**?
  - **`owl:inverseOf`**: inverse relationship
- For individuals:
  - **`owl:sameAs`**: two URIs refer to the same individual (e.g., concept)
  - **`owl:differentFrom`**: negation of **`owl:sameAs`**

> Example: connecting to French



> 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
- A real superset of RDFS
- But: *an OWL Full ontology may be undecidable!*

> OWL Description Logic (DL)

Maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure is realizable (well, in current 2004…)

- Classes and individuals are strictly separated: a class cannot be an individual of another class
- No characterization of datatype properties possible
- …

> OWL Lite

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
  - …

> 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
- Future OWL versions may define further subsets that are simpler, easier to implement and still have enough functionality
  - referred to as “tractable fragments”

> Ontology development

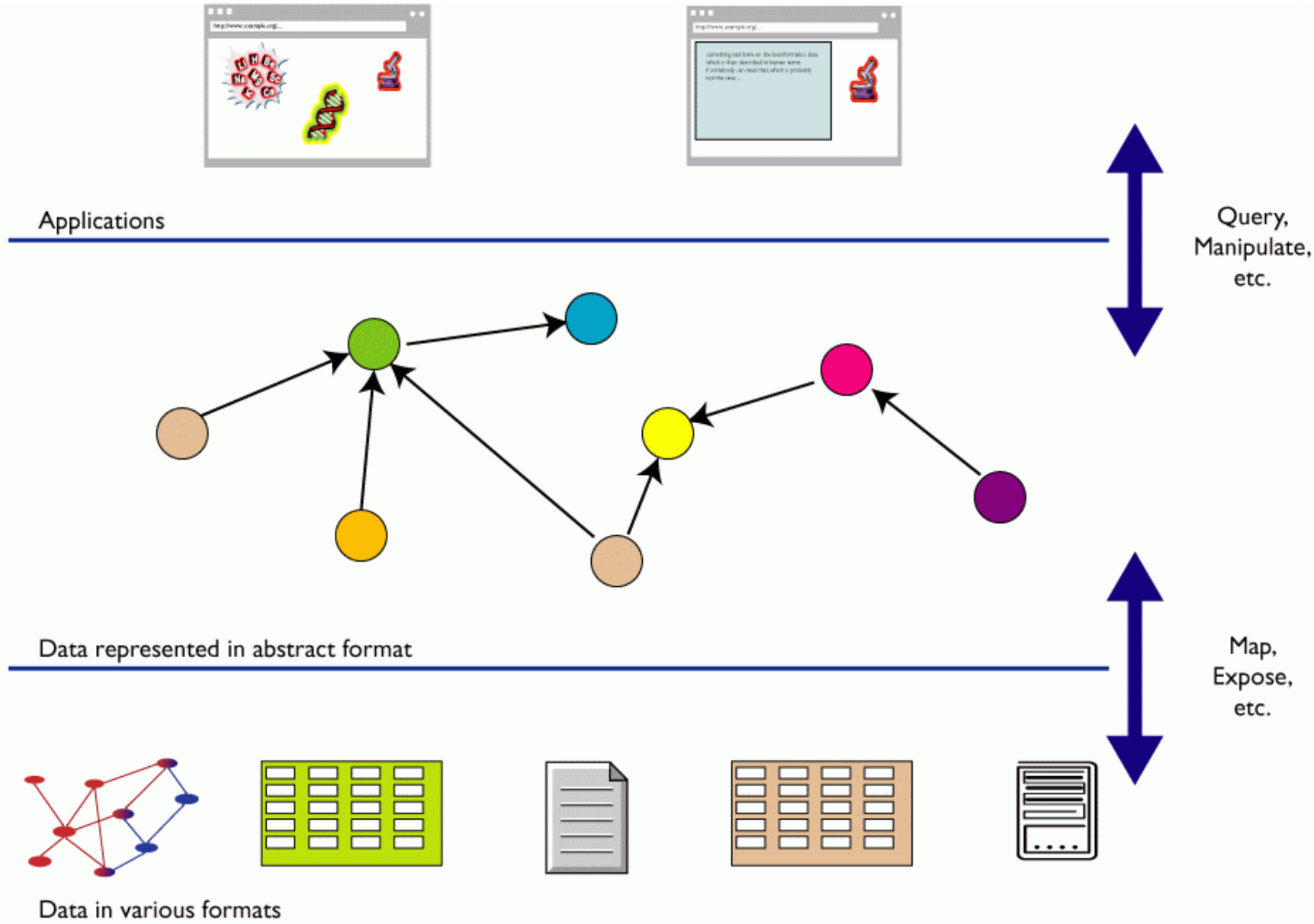
- The hard work is to *create* the ontologies
  - requires a good knowledge of the area to be described
  - some communities have good expertise already
  - *OWL is just a tool to formalize ontologies*
- Large scale ontologies are often developed in a community process
- Ontologies should be *shared* and *reused*
  - can be via the simple namespace mechanisms…
  - …or via explicit inclusions
- Applications can also be developed with very small ontologies, though

> Ontologies examples

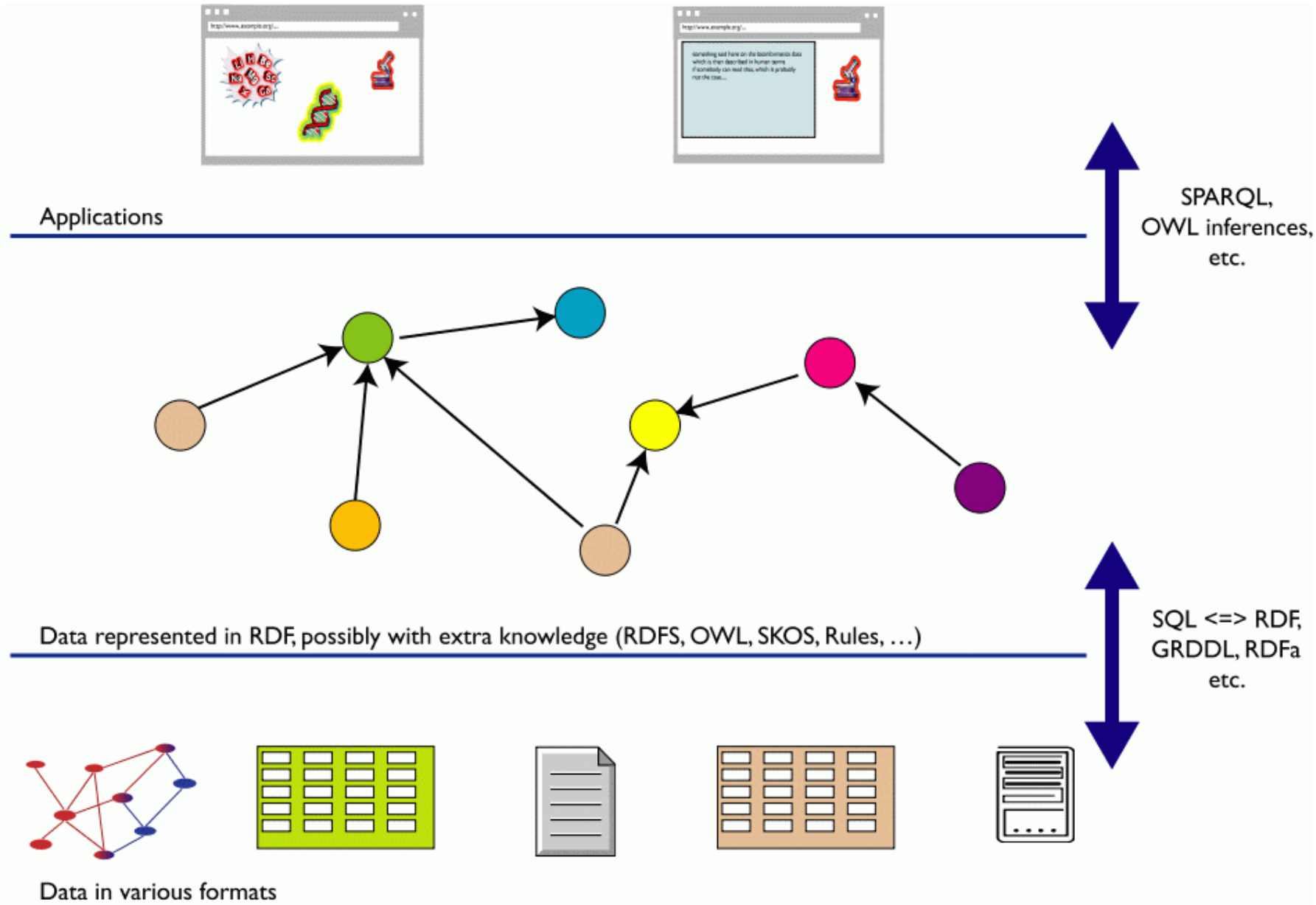
- International Country List
  - example for an OWL Lite ontology
- Large ontologies are being developed (converted from other formats or defined in OWL)
  - eClassOwl: eBusiness ontology for products and services, 75,000 classes and 5,500 properties
  - National Cancer Institute's ontology: about 58,000 classes
  - Open Biomedical Ontologies Foundry: a collection of ontologies, including the Gene Ontology to describe gene and gene product attributes in any organism or protein sequence and annotation terminology and data (UniProt)
  - BioPAX: for biological pathway data

> What have we achieved? What is available?

> Remember the integration example?



> Same with what we learned



> Lots of tools

- *Lots* of tools are available. Are listed on W3C's wiki:
  - RDF programming environment for 14+ languages, including C, C++, Python, Java, Javascript, Ruby, PHP,… (no Cobol or Ada yet…)
  - 13+ Triple Stores, ie, database systems to store datasets
  - SPARQL "endpoints"
  - converters to and from RDF
  - validators for RDF, OWL, …
  - etc
- Some of the tools are Open Source, some are not; some are very mature, some are not: *it is the usual picture of software tools*, nothing special any more!
- *Anybody can start developing RDF-based applications today*

> "Core" vocabularies

- There are also a number "core vocabularies" (not necessarily OWL based)
  - FOAF: about people and their organizations
  - DOAP: on the descriptions of software projects
  - Music Ontology: on the description of CDs, music tracks, …
  - SIOC: Semantically-Interlinked Online Communities
  - vCard in RDF
  - DCMI's vocabularies: guess this one… ☺
  - …
- Hopefully LOM, DC/RDA, etc, will enrich this list soon!
- One should never forget: ontologies/vocabularies must be shared and reused!

> 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"
- GRDDL Primer has just been published, RDFa Primer in preparation
- The W3C Semantic Web Activity Homepage has links to all the specifications

> Some books

- J. Davies, D. Fensel, F. van Harmelen: Towards the Semantic Web (2002)
- S. Powers: Practical RDF (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)
- …

See the separate Wiki page collecting book references

> Further information

- Dave Beckett's Resources at Bristol University
  - *huge* list of documents, publications, tools, …
- Planet RDF aggregates a number of SW blogs

> SWBP Working Group documents

- Documents for ontology engineering
  - "Best Practice Recipes for Publishing RDF Vocabularies"
  - "Defining N-ary relations"
  - "Representing Classes as Property Values";
  - "XML Schema Datatypes in RDF and OWL"
  - etc
- See the Group's homepage for further links
- Work is continuing in the SW Deployment Group with new documents

> Deployment, more application examples

> The “corporate” landscape is moving

- Major companies offer (or will offer) Semantic Web tools or systems using Semantic Web: Adobe, Oracle, IBM, HP, Software AG, GE, Northrop Gruman, Altova, …
- Some of the names of active participants in W3C SW related groups: ILOG, HP, Agfa, SRI International, Fair Isaac Corp., Oracle, Boeing, IBM, Chevron, Siemens, Nokia, Merck, Pfizer, AstraZeneca, Sun, Eli Lilly, …

> May start with small communities

- The needs of a deployment application area:
  - have serious problem or opportunity
  - have the intellectual interest to pick up new things
  - have motivation to fix the problem
  - its data connects to other application areas
  - have an influence as a showcase for others
- The high energy physics community played this role for the Web in the 90's

> Some deployment communities

- The technology is picked up by specialized communities
  - just like the high energy physics community did for the original Web…
- Some examples: defense, eGovernment, energy sector, financial services, health care, oil and gas industry, life sciences … *digital libraries*
- Health care and life science sector is now very active
  - also at W3C, in the form of an Interest Group

> Data integration

- Very important for large application areas (life sciences, energy sector, eGovernment, financial institutions), as well as everyday applications (eg, reconciliation of calendar data)
- Developments are under way at various companies, institutions
  - not always easy to find out the details…
- Data integration comes to the fore as one of *the* SW application areas
- We have already seen some examples; some more are:
  - Pfizer, Eli Lilly, MITRE Corp., Elsevier, …
  - EU R&D Projects like Sculpteur and Artiste
  - UN FAO's MeteoBroker, …
  - Semantic Digital Library projects (JeromeDL, Simile, Fedora,…)

> Web sites, portals, local site search

- Portal's internal organization makes use of semantic data, ontologies
  - integration with external and internal data
    - there is a clear overlap here with data integration applications!
  - better queries, often based on controlled vocabularies or ontologies…
- These are very close to the metadata based applications… but the underlying vocabularies (ontologies) may be much more complex

> Semantic portal for art collections

Search results - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://e-culture.multimedian.nl/demo/keyword?key=monet&schema=http%3A%2F%2Fwww.multimedian.nl%2Fpr

World Clock Feeds W3C Uniview Bibsonomy Semantic Web W3C Mailing lists Presentations Bookmarklets Python SWEO Survey

The MultimediaN E-culture

Search

monet Search

Basic Search | Advanced Search | /facet | Relation Search | My Collection (en nl id ru uk zh)

Search results on monet organized by category (37 results).

Works with matching title (2)

Claude Monet Painting ...
Sargent, John Singer

Monet painting in his ...
Renoir, Pierre Auguste

Works created by artists with matching name (35)

"The Red Kerchief: Po ...
Monet, Claude

"The Stroll, Camille ...
Monet, Claude

La Corniche near Monaco
Monet, Claude

The Thames at Westminster
Monet, Claude

Water Lilies
Monet, Claude

1830 1835 1840 1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910 1915 1920 1925 1930

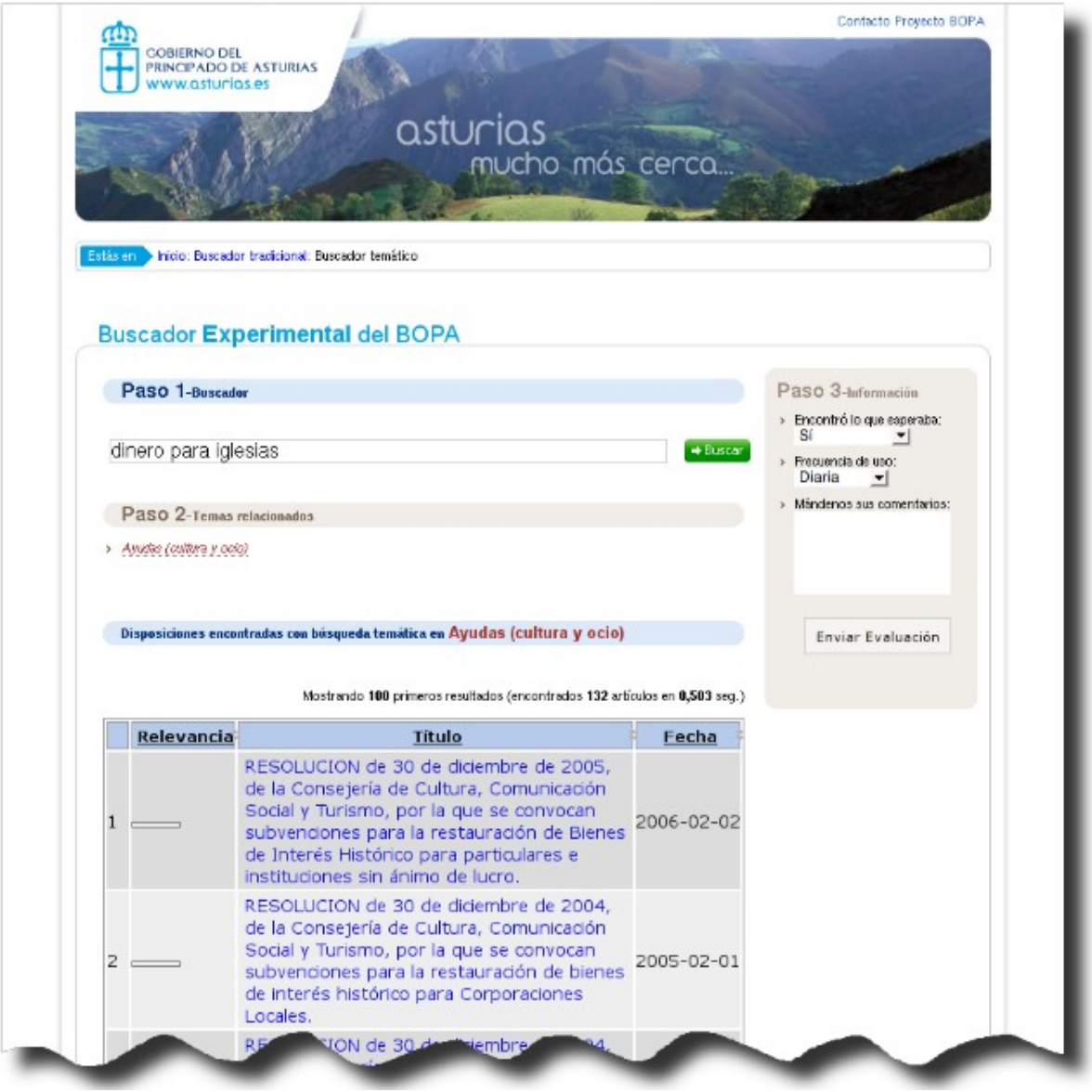
Monet, Claude
Renoir, Pierre Auguste
Impressionist
Sargent, John Singer

Done Adblock

Courtesy of Jacco van Ossenbruggen, CWI, and Guus Schreiber, VU Amsterdam

> Portal to Principality of Asturias' documents

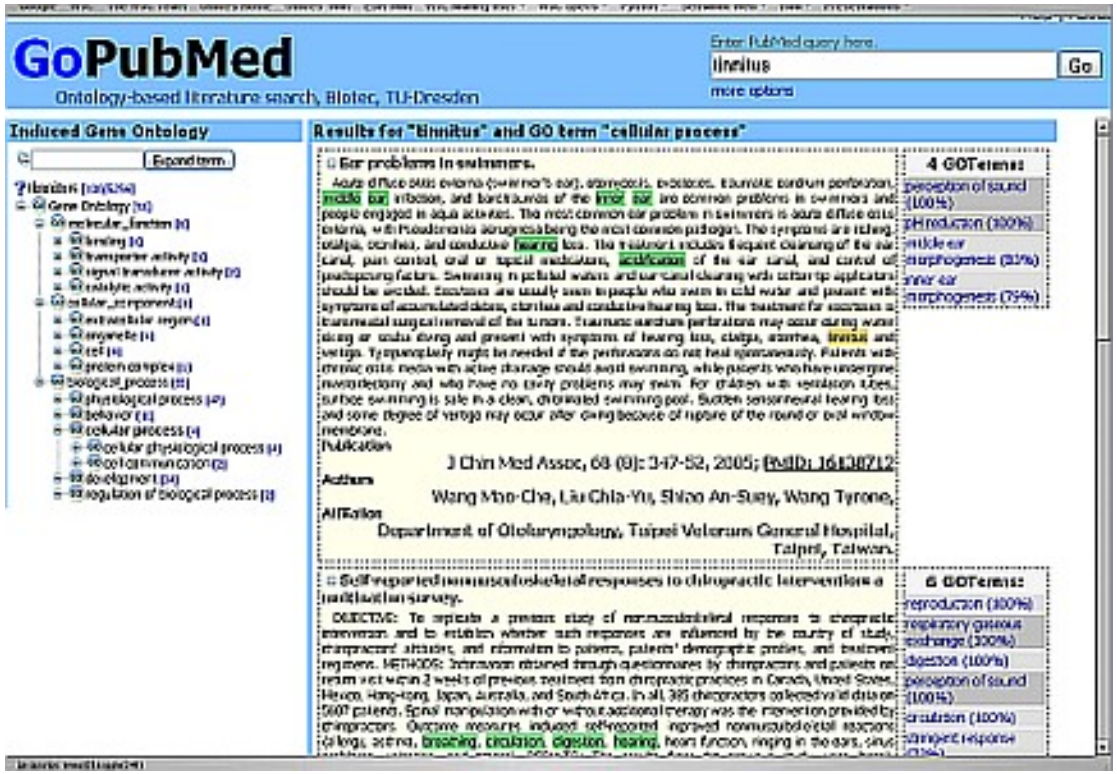
- Search through governmental documents
- A "bridge" is created between the users and the juridical jargon using SW vocabularies and tools



Courtesy of Diego Berrueta and Luis Polo, CTIC, U. of Oviedo, and the Principality of Asturias, (SWEO Case Study)

> 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



> Other examples…

- Vodafone's Live Mobile Portal
- Sun's White Paper and System Handbook collections
- Nokia's S60 support portal
- Yahoo!'s food and finance portals
- Oracle's virtual pressroom
- Opera's community site
- Dow Jones' Synaptica

> Other application areas come to the fore

- Content management
- Business intelligence
- Collaborative user interfaces
- Sensor-based services
- Linking virtual communities
- Grid infrastructure
- Multimedia data management
- Etc

> Thank you for your attention!

- My email: ivan@w3.org
- These slides are publicly available on:

http://www.w3.org/2007/Talks/0831-Singapore-IH/

- You can also go to my general presentations' site:

http://www.w3.org/People/Ivan/CorePresentations/

- where other slide sets are available (extended version of the tutorial, semantics of RDF and OWL, further application examples, etc)