Introduction to the Semantic Web

Stavanger, Norway, 2007-04-23

Ivan Herman, W3C

Introduction

Towards a Semantic Web

- The current Web represents information using
  - *natural language (English, Hungarian, Norwegian,…)*
  - *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?
  - *…*

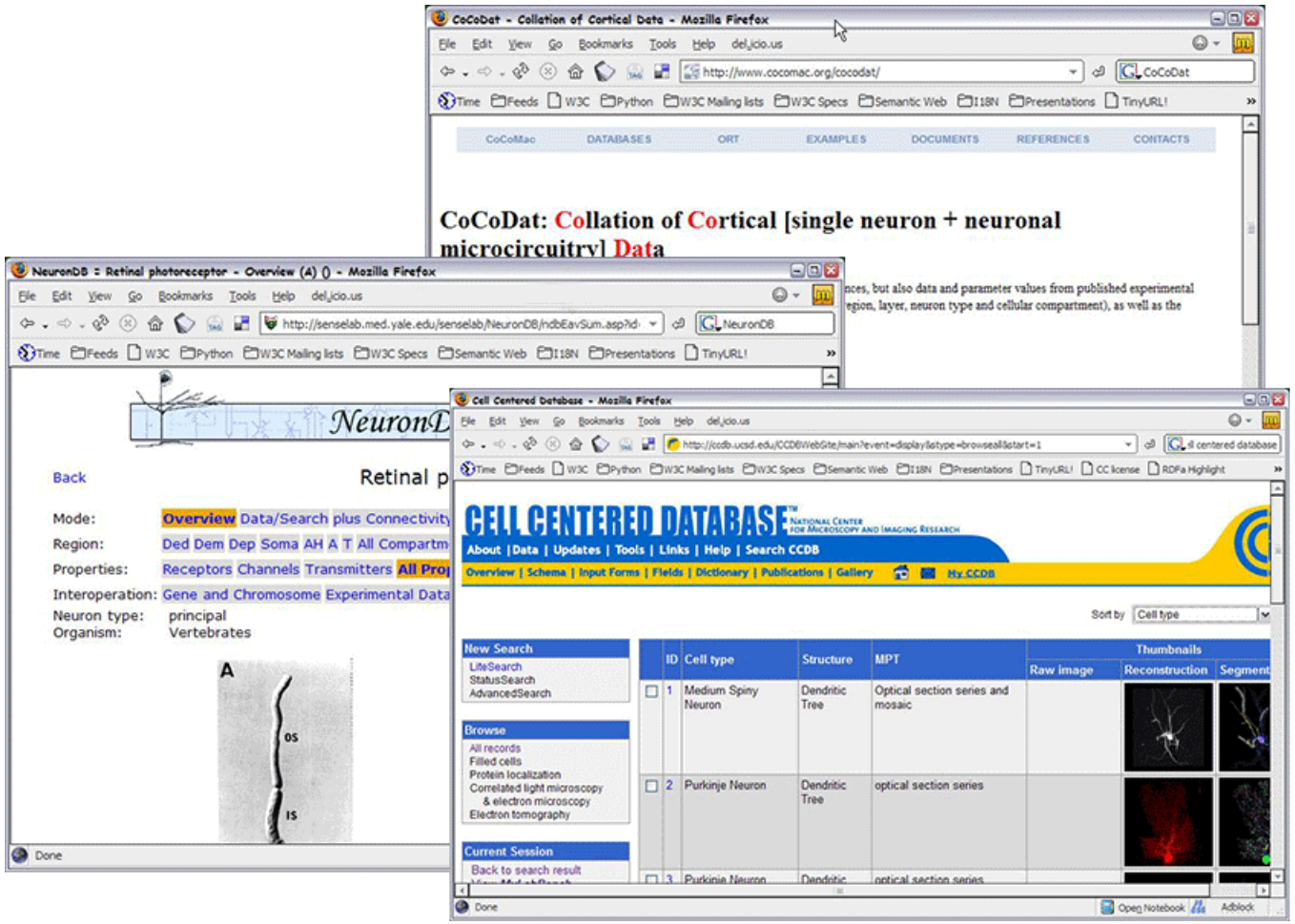
Example: Automatic Airline Reservation

- 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)

And the problem *is* real



Introductory Example

- 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*

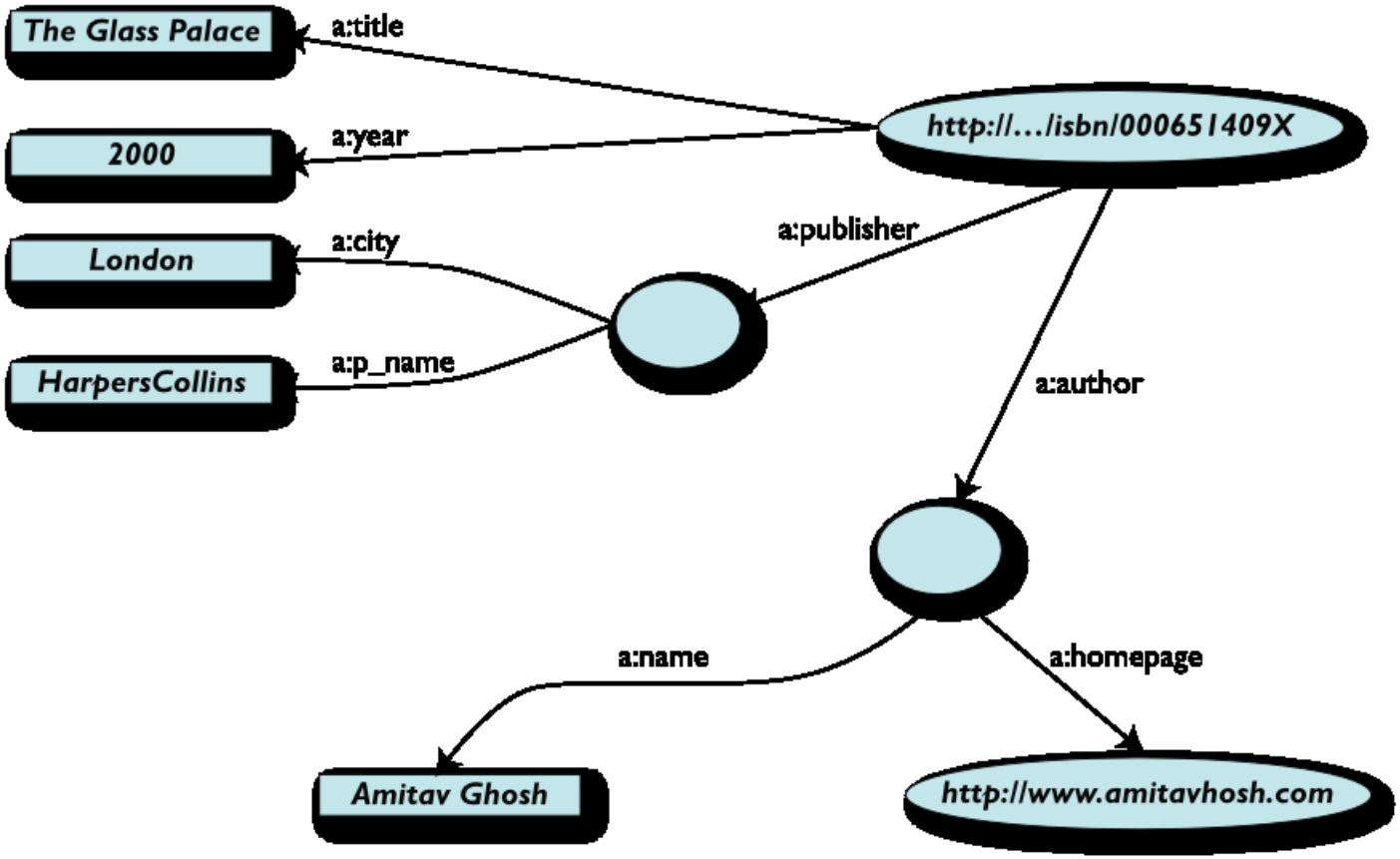
A *Simplifed* Bookstore Data (Dataset “A”)

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

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

ID	Publisher Name	City
id_qpr	Harper Collins	London

1st Step: Export Your Data as a Set of *Relations*



Some Notes on the Exporting the Data

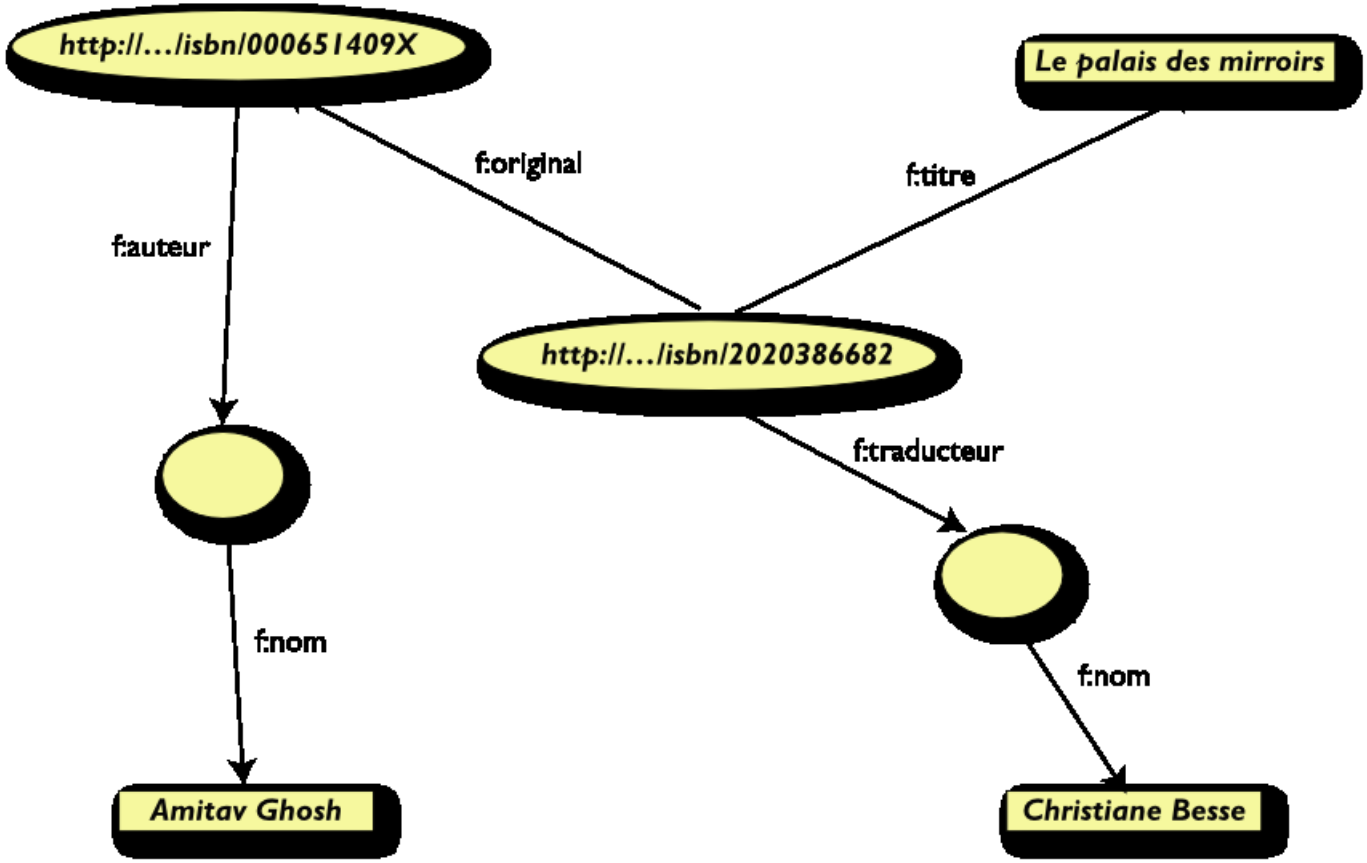
- Relations form a graph
  - *the nodes refer to the “real” data or contain some literal*
  - *how the graph is represented in machine is immaterial for now*
- Data export does *not* necessarily mean physical conversion of the data
  - *relations can be generated on-the-fly at query time*
    - via SQL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

Another Bookstore Data (dataset “F”)

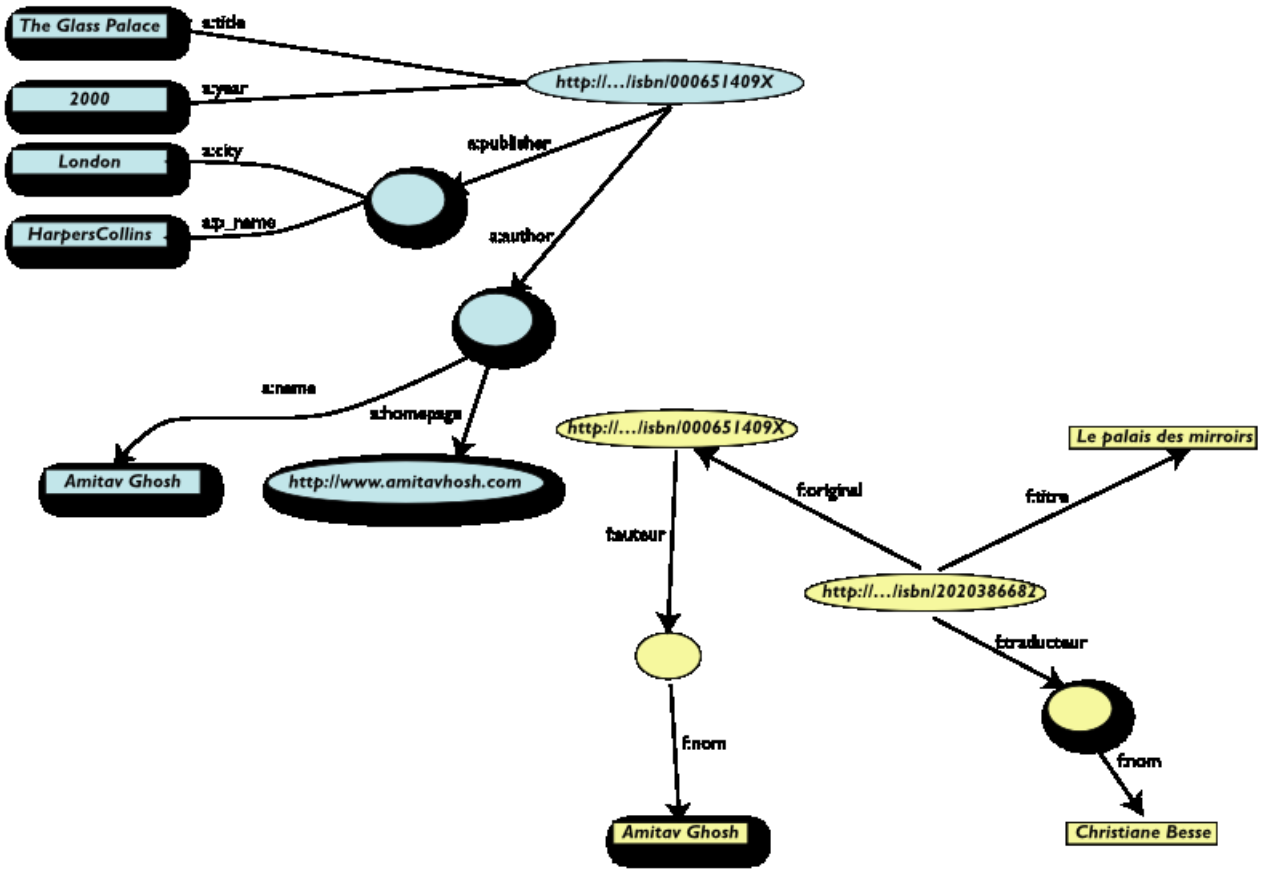
ID	Titre	Auteur	Traducteur	Original
ISBN 2020386682	Le Palais des miroirs	i_abc	i_qrs	ISBN 0-00-651409-X

ID	Nom
i_abc	Amitav Ghosh
i_qrs	Christiane Besse

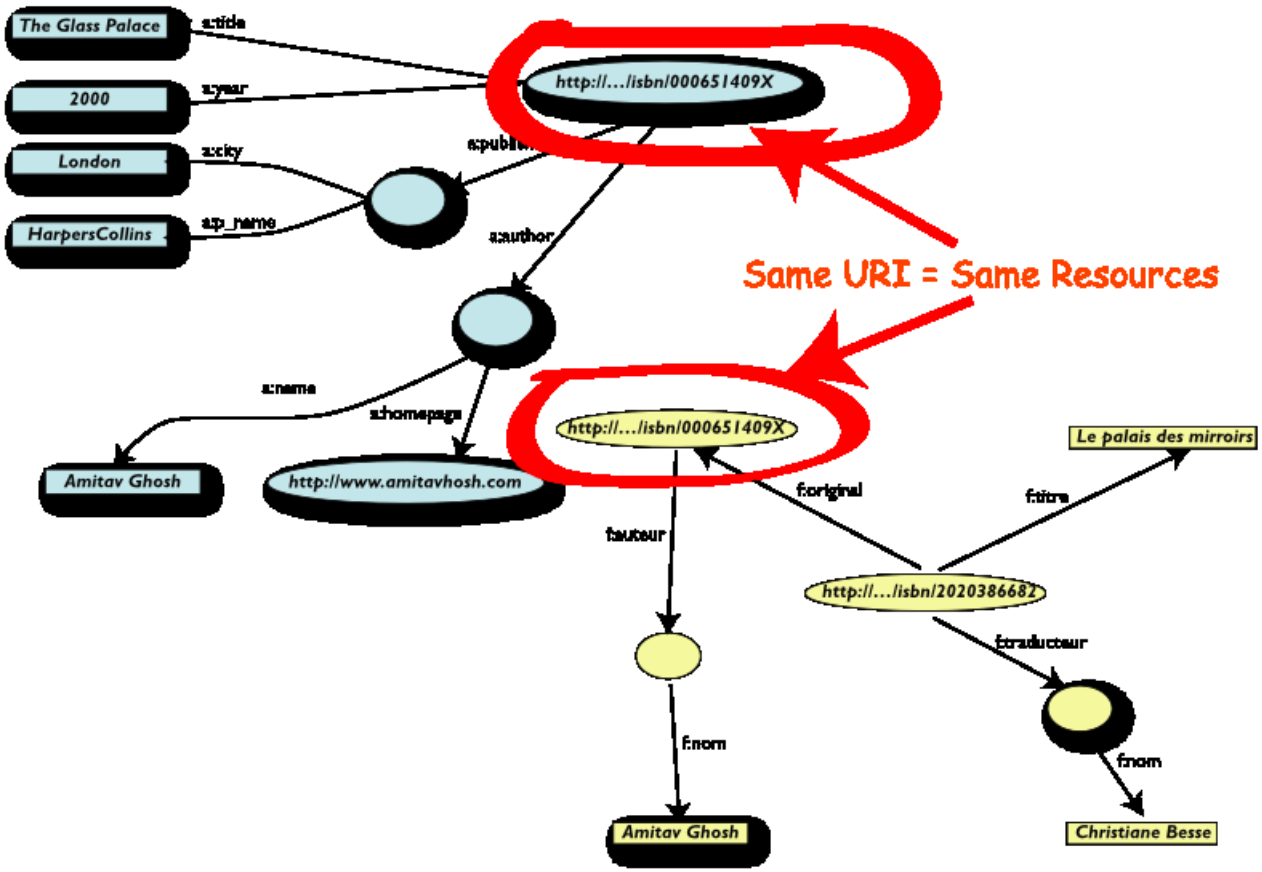
2nd Step: Export Your Second Set of Data



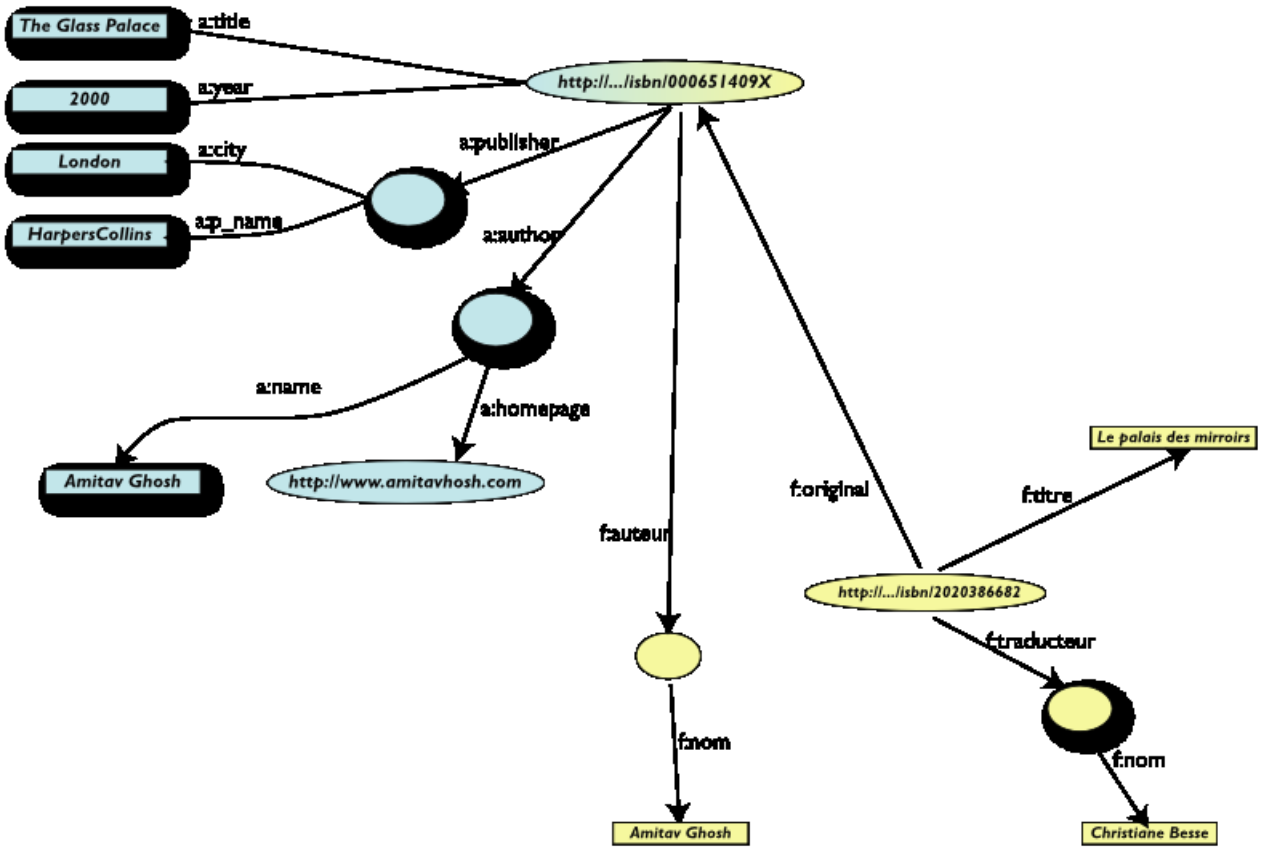
3rd Step: Start Merging Your Data



3rd Step: Start Merging Your Data (cont.)

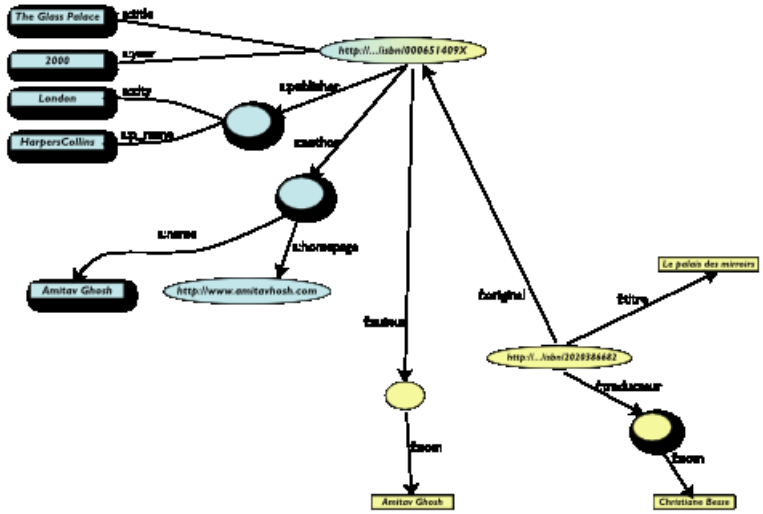


3rd Step: Merge Identical Resources



Start Making Queries…

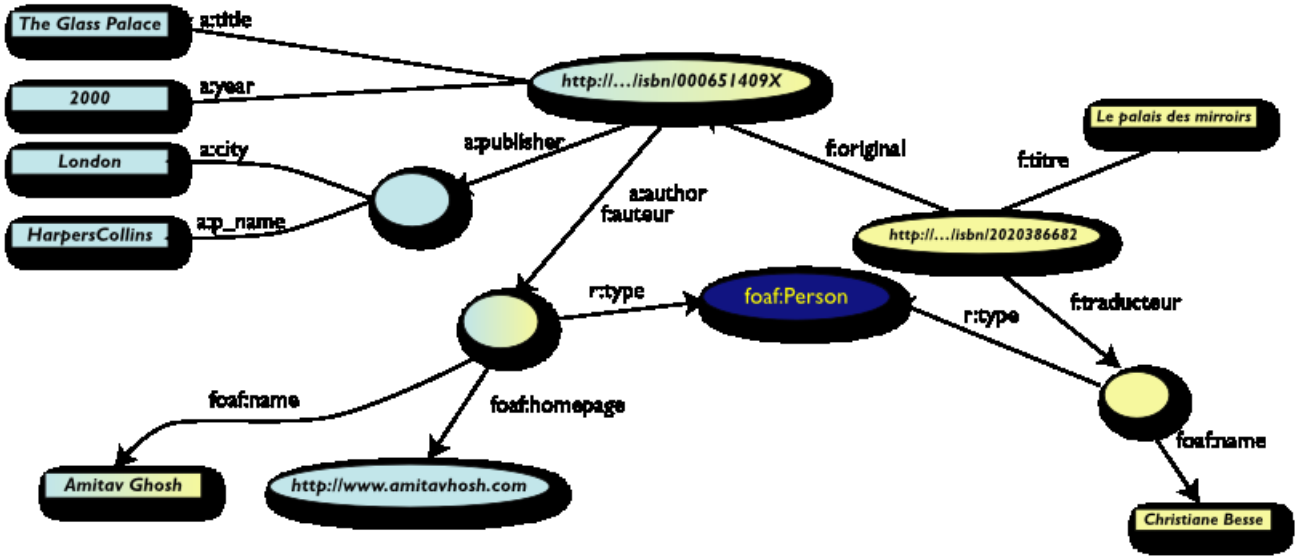
- User of data “F” can now ask queries like:
  - *«donnes-moi le titre de l’original»*
  - *(ie: “give me the title of the original”)*
- This information is not in the dataset “F”…
- …but can be automatically retrieved by merging with dataset “A”!



However, More Can Be Achieved…

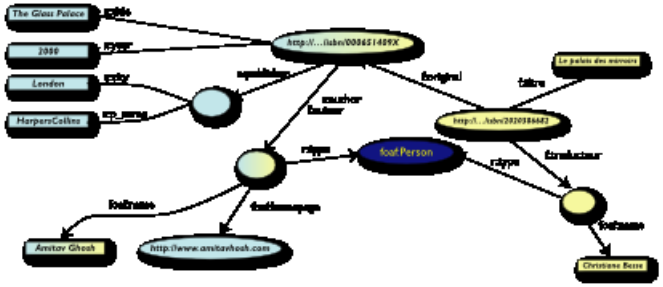
- We “feel” that `a:author` and `f:auteur` should be the same
- But an automatic merge doest not know that!
- Let us add some extra information to the merged data:
  - `a:author` *same as* `f:auteur`
  - *both identify a “Person”:*
    - a term that a community may have already defined:
      - a “Person”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

3rd Step 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 data is not in dataset “F”…
- …but was made available by:
  - *merging datasets “A” and “F”*
  - *adding three simple extra statements as an extra knowledge*
  - *using existing terminologies as part of that extra knoweledge*



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 simple (e.g., XSLT) tools
  - *there is an active development to add some simple semantic “tag” to wikipedia entries*
  - *we tacitly presuppose their existence in our example…*

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

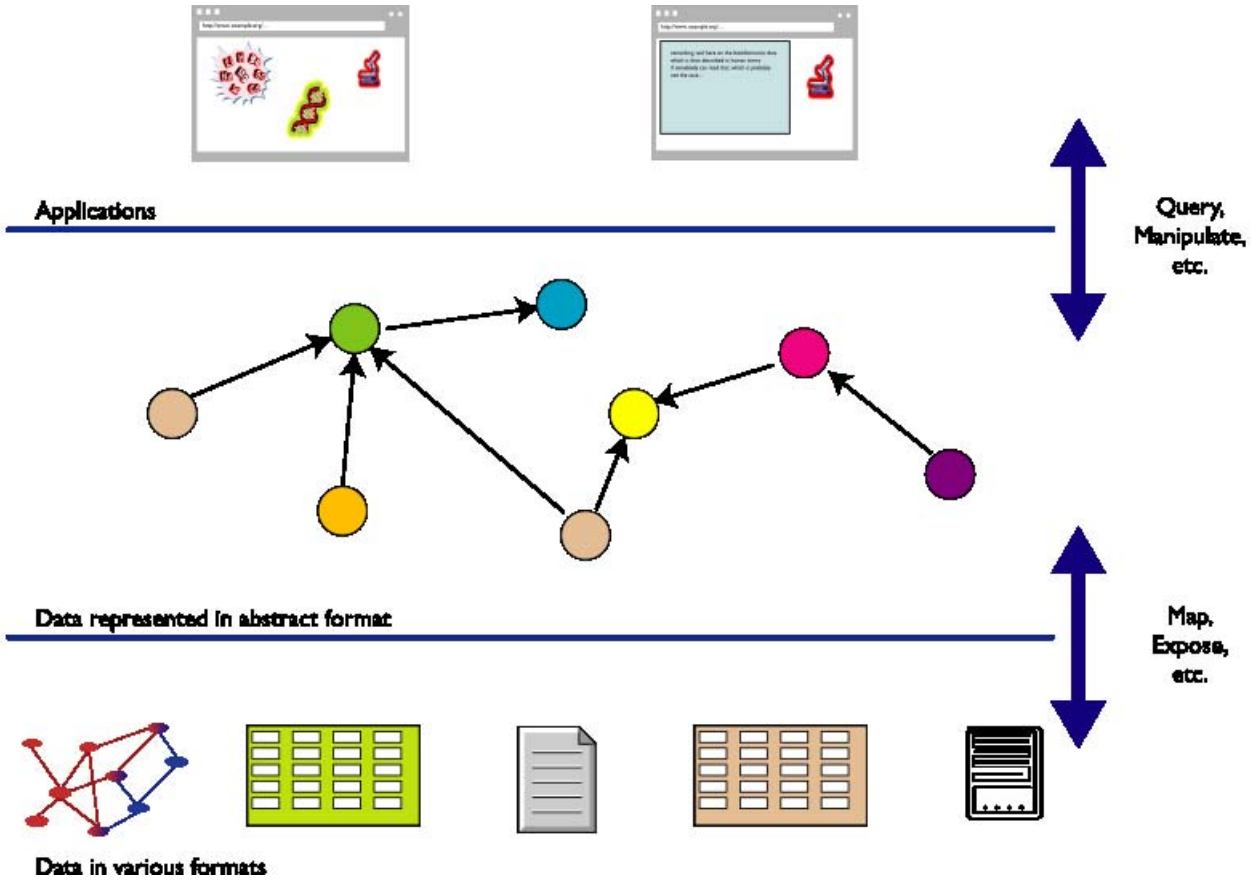
What Did We Do?

- 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 knowledge, 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

- The added extra knowledge could be much more complex to the merged datasets
  - *e.g., a full classification of various type of library data, types of books (literature or not, fiction, poetry, etc)*
  - *geographical information*
  - *information on inventories, prices*
  - *etc.*
- This is where *ontologies*, extra *rules*, etc, may come in
- Even more powerful queries can be asked as a result

What did we do? (cont)



The Abstraction Pays Off Because…

- … the graph representation is independent on the *exact* structures in, say, a relational database
- … a change in local database schemas, XHTML structures, etc, do *not* affect the whole, only the “export” step
  - *“schema independence”*
- … new data, new connections can be added seamlessly, regardless of the structure of other data sources

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 two tutorials…)

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 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/original`**
  - *here is the complete triple:*

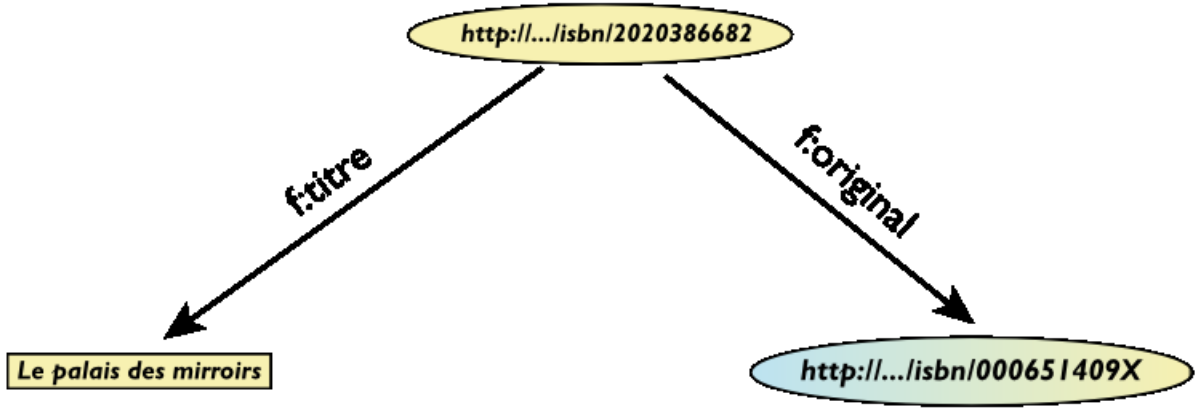
```
(<http://…isbn…6682>, <http://…/original>, <http://…isbn…409X>)
```

- *RDF* is a general model for such triples (with machine readable formats like RDF/XML, Turtle, n3, RXR, …)
- … *and that's it!* (simple, isn't it? ☺)

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; e.g., 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('home'))`
  - `http://www.example.org/file.html#home`
- RDF Triples form a *directed, labelled graph* (best way to think about them!)

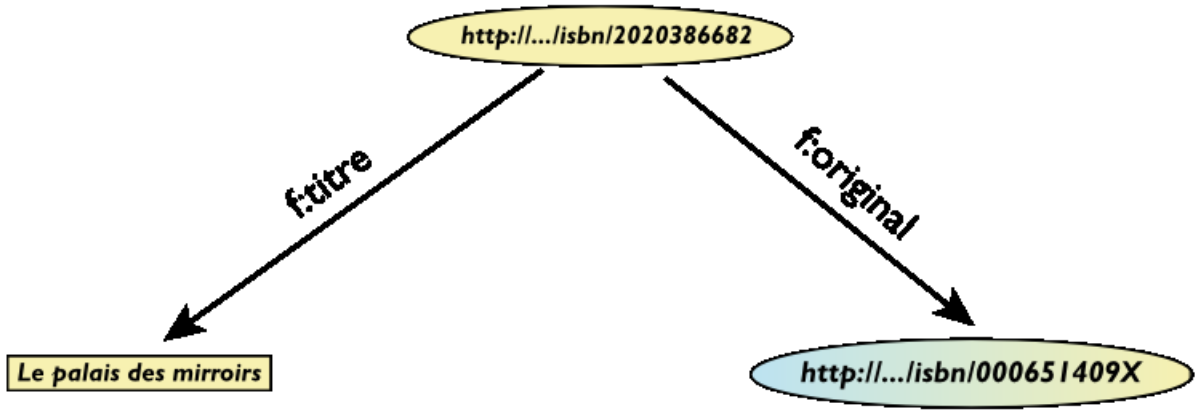
A Simple RDF Example (in RDF/XML)




```
<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)



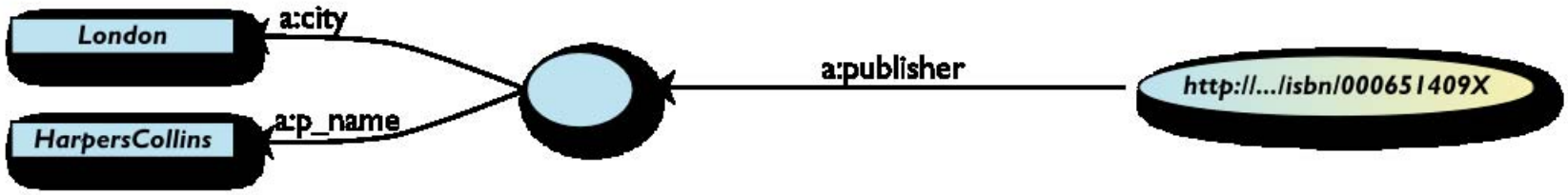

```
<http://…/isbn/2020386682>
    f:titre "Le palais des mirroirs"@fr;
    f:original <http://…/isbn/000651409X>.
```

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 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: Define Extra URI-s

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

```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:resource="#Thing"/>
</rdf:Description>
<rdf:Description rdf:ID="Thing">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</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, not part of the RDF model!
  - *Turtle has something similar, too*

Blank Nodes

- Use an *internal* identifier

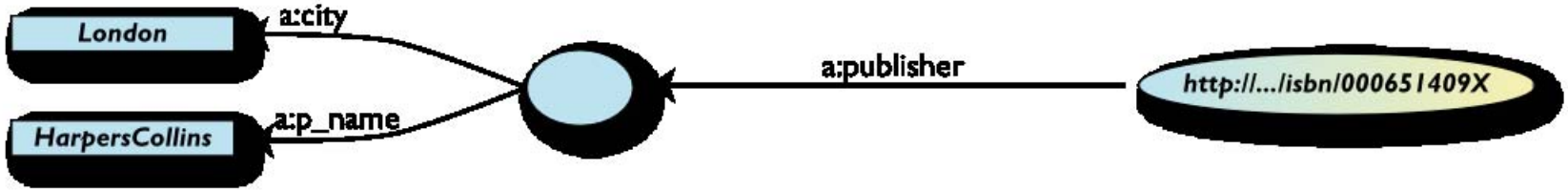
```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <a:publisher rdf:nodeID="A234"/>
</rdf:Description>
<rdf:Description rdf:nodeID="A234">
   <a:p_name>HarpersCollins</a:p_name>
   <a:city>HarpersCollins</a:city>
</rdf:Description>
<http://…/isbn/2020386682> a:publisher _:A234.
          _:A234 a:p_name "HarpersCollins".
```

- **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

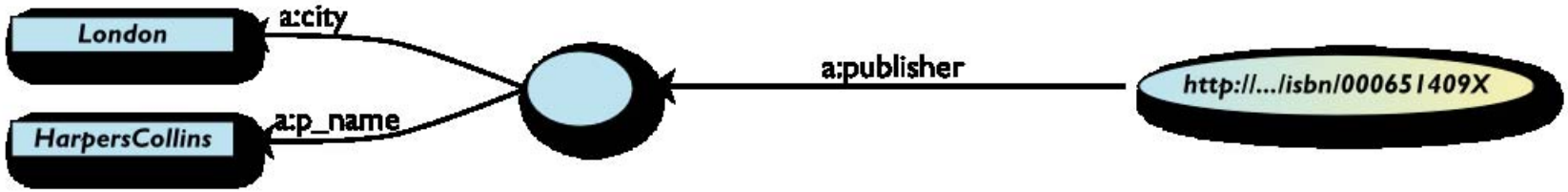
- Let the system create a **`nodeID`** internally (you do not really care about the name…)

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

RDFSchemas

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…*

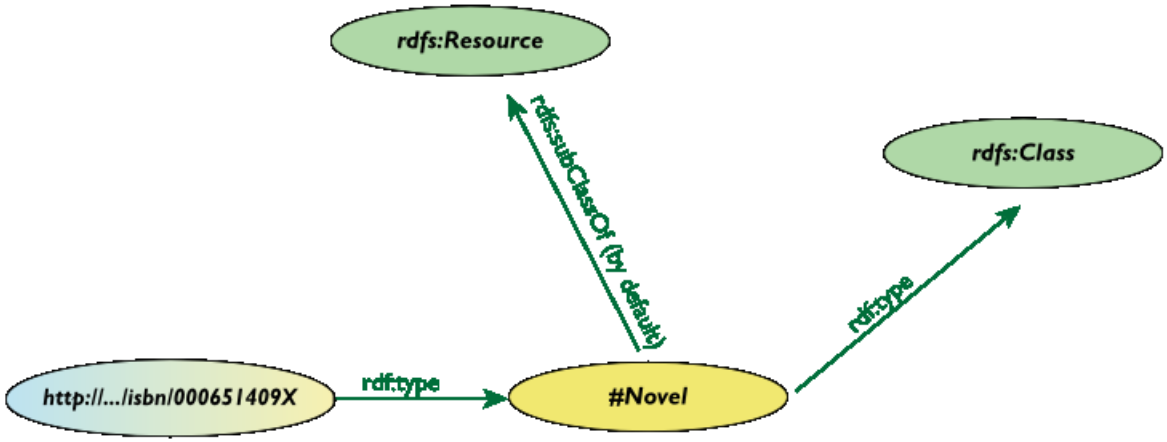
Classes, Resources, …

- Think of well known in traditional ontologies:
  - *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: “«isbn:000651409X» is a novel”
  - *“subclassing”: instance of one is also the instance 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”):

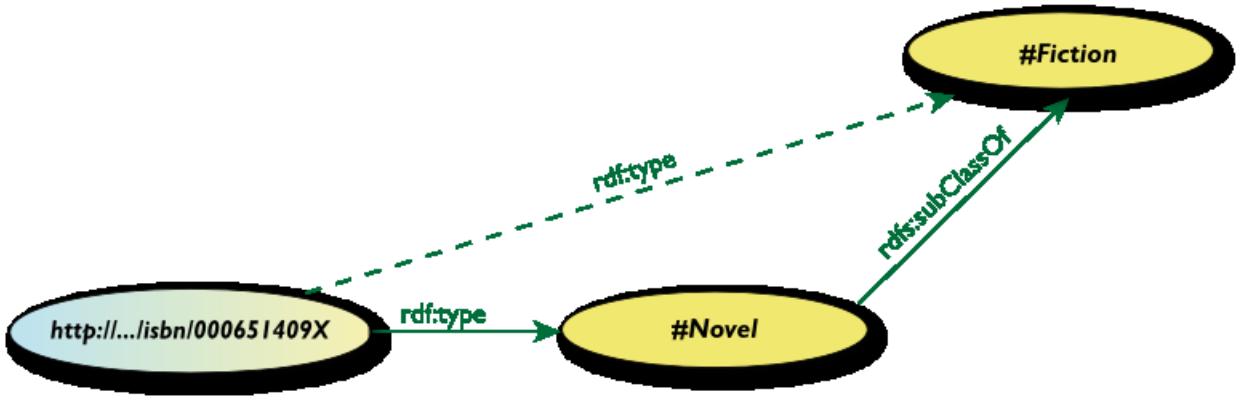
```
<rdf:Description rdf:about="http://…/isbn/000651409X">
   <rdf:type rdf:resource="http://…/bookSchema.rdf#Novel"/>
</rdf:Description>
```

- 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…*
  - *“«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 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

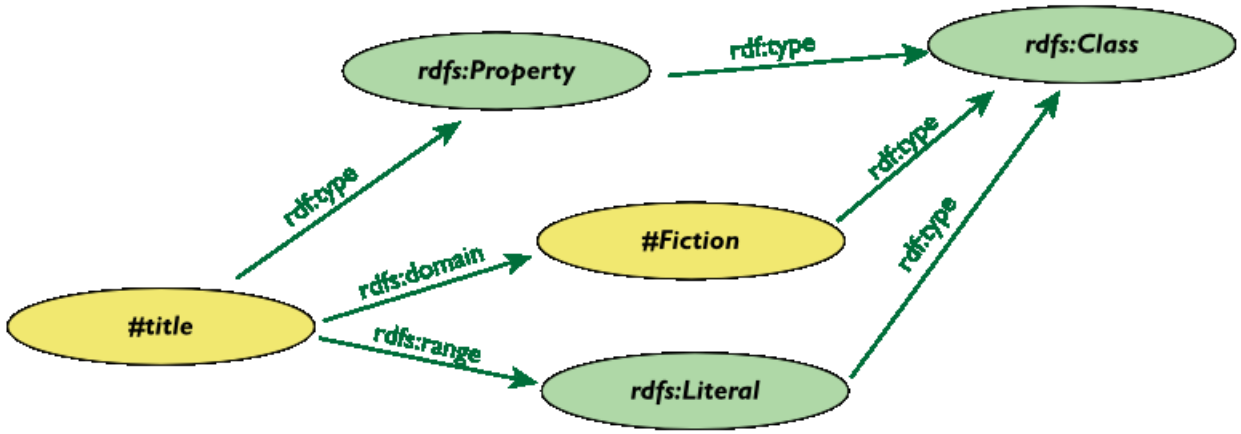
- Property is a special class (`rdf:Property`)
  - *properties are also resources identified by URI-s*
- Properties are constrained by their range and domain
  - *i.e., what individuals can serve as object and subject*
- There is also a possibility for a “sub-property”
  - *all resources bound by the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



Property Specification Serialized

In XML/RDF:

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

In Turtle:

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

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?
- 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 are necessary (see later…)

Predefined Classes and Properties

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

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

Querying RDF Graphs/Repositories

- Remember the Jena idiom:

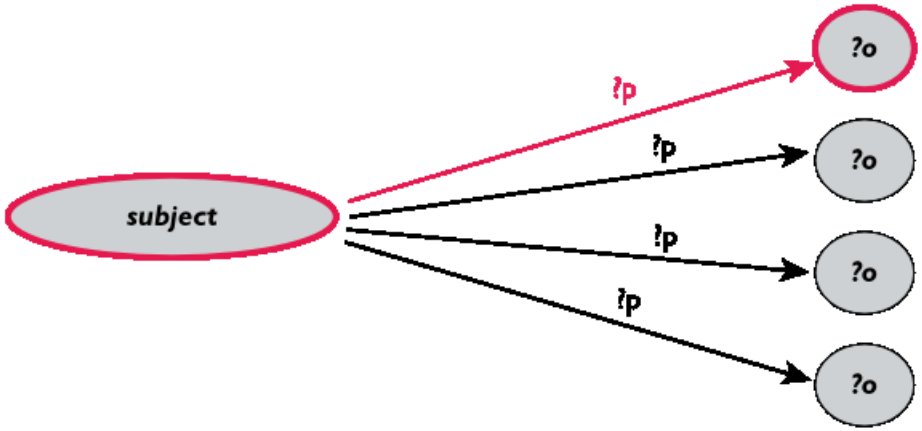
```
StmtIterator iter=model.listStatements(subject,null,null);
while(iter.hasNext()) {
    st = iter.next();
    p = st.getProperty(); o = st.getObject();
    do_something(p,o);
```

- 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")



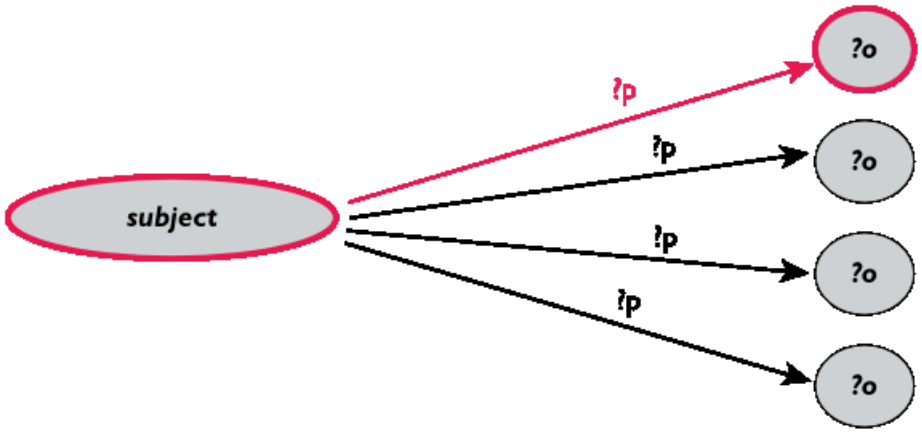
General: Graph Patterns

- 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}
```

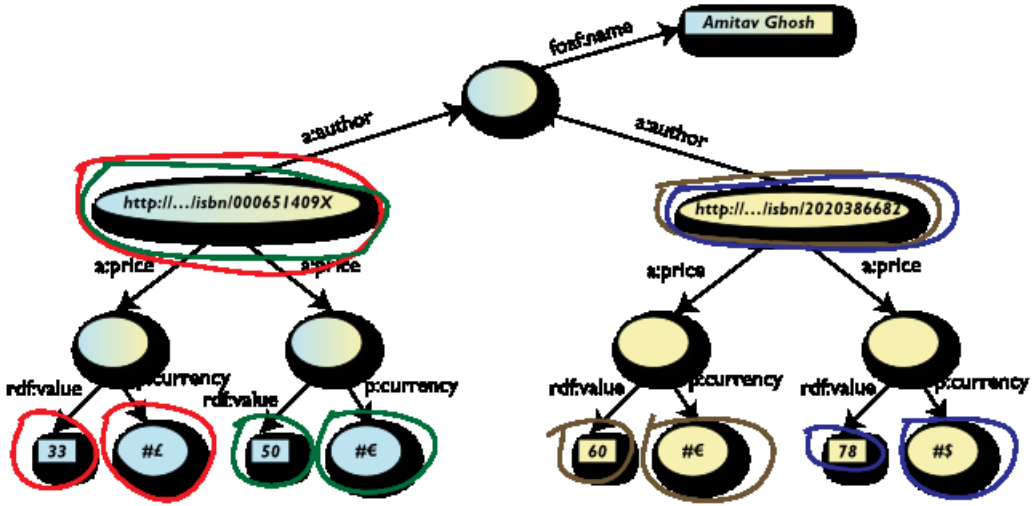
- The triplets in **`WHERE`** define the graph pattern, with **`?p`** and **`?o`** “unbound” symbols
- The query returns a list of matching **`p,o`** pairs



Simple SPARQL Example

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

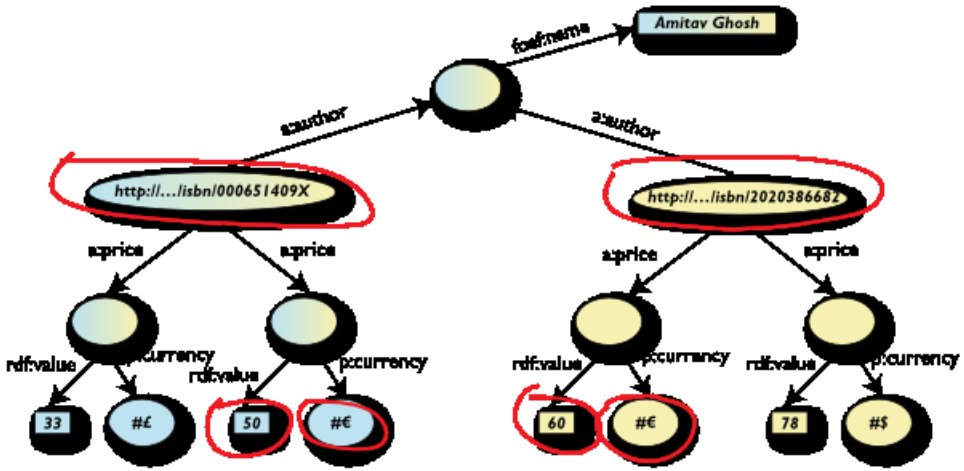
- Returns: `[[<..49X>,33,£], [<..49X>,50,€], [<..6682>,60,€], [<..6682>,78,$]]`



Pattern Constraints

```
SELECT ?isbn ?price ?currency
WHERE { ?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.
        FILTER(?currency == €) }
```

- Returns: `[ [<..49X>,50,€], [<..6682>,60,€]]`
- SPARQL defines a base set of operators and functions



Other SPARQL Features

- Optional patterns; if they match variables, fine, if not, do not care
- 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 quite mature already
  - *recommendation expected 3rd Q of 2007*
  - *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 in XML or JSON formats
- There are already a number of applications, demos, etc.,

Get to RDF(S) Data

Simplest: Write your own RDF Data…

- The simplest aproach: write your own RDF data in your preferred syntax…
- You may add RDF to XML directly (in its own namespace); e.g., in SVG:

```
<svg ...>
  ...
  <metadata>
    <rdf:RDF xmlns:rdf="http://../rdf-syntax-ns#">
      ...
    </rdf:RDF>
  </metadata>
    ...
</svg>
```

- However: *this does not scale!*

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* `meta` *elements*
- … and then *generate* RDF automatically (e.g., via an XSLT script)
- This is similar to what “microformat” do (without referring to RDF, though)
  - *they may not extract RDF but use the data directly instead in Web2.0 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 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
- There is a mechanism for XML in general
  - *a transformation can also be defined on an XML schema level*
- A “bridge” to “microformats”
- Currently a Working Group, with a recommendation planned in the 2nd Quarter of 2007

Another Upcoming Solution: RDFa

- 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*
  - *the same mechanism can also be used for general XML content*

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.)

- 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*
- It can easily be combined with GRDDL

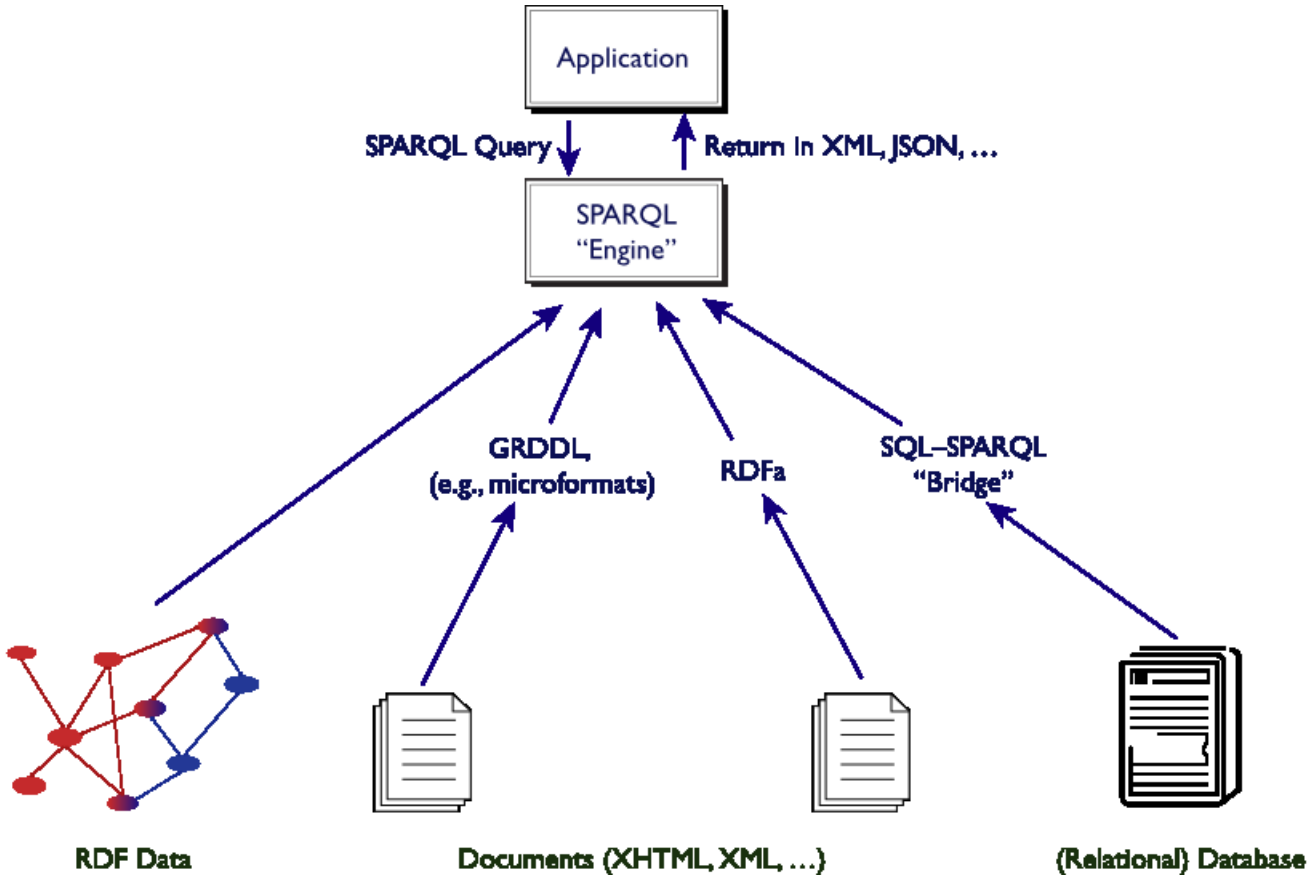
RDFa and GRDDL

- Both solutions aim at “binding” existing structural data with RDF
  - *GRDDL “brings” structured data to RDF*
  - *RDFa “brings” RDF to structured data (HTML)*
- The *same* URI may be interpreted as
  - *a web page to be displayed by a browser*
  - *as RDF data to be integrated*
- (compare to a credit card: a human can read its number and owner; a card reader can access to its data)

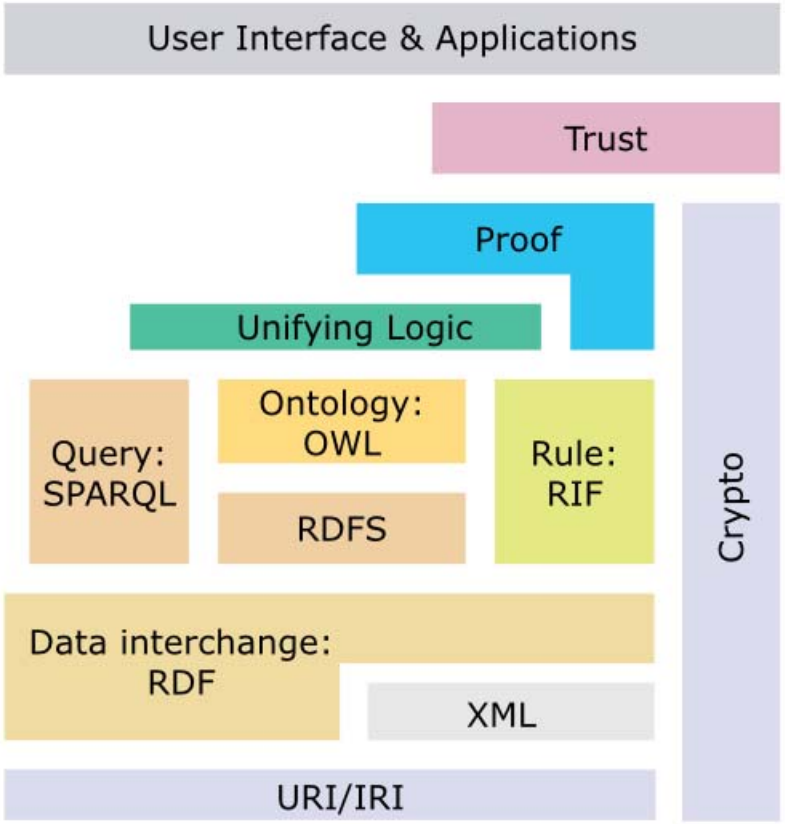
Bridge to Relational Databases

- Most of the data are stored in relational databases
- “RDFying” them is an impossible task
- “Bridges” are being defined:
  - *a layer between RDF and the database*
  - *RDB tables are “mapped” to RDF graphs 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*
- This is a *very* important source of RDF data

SPARQL As a Unifying Force



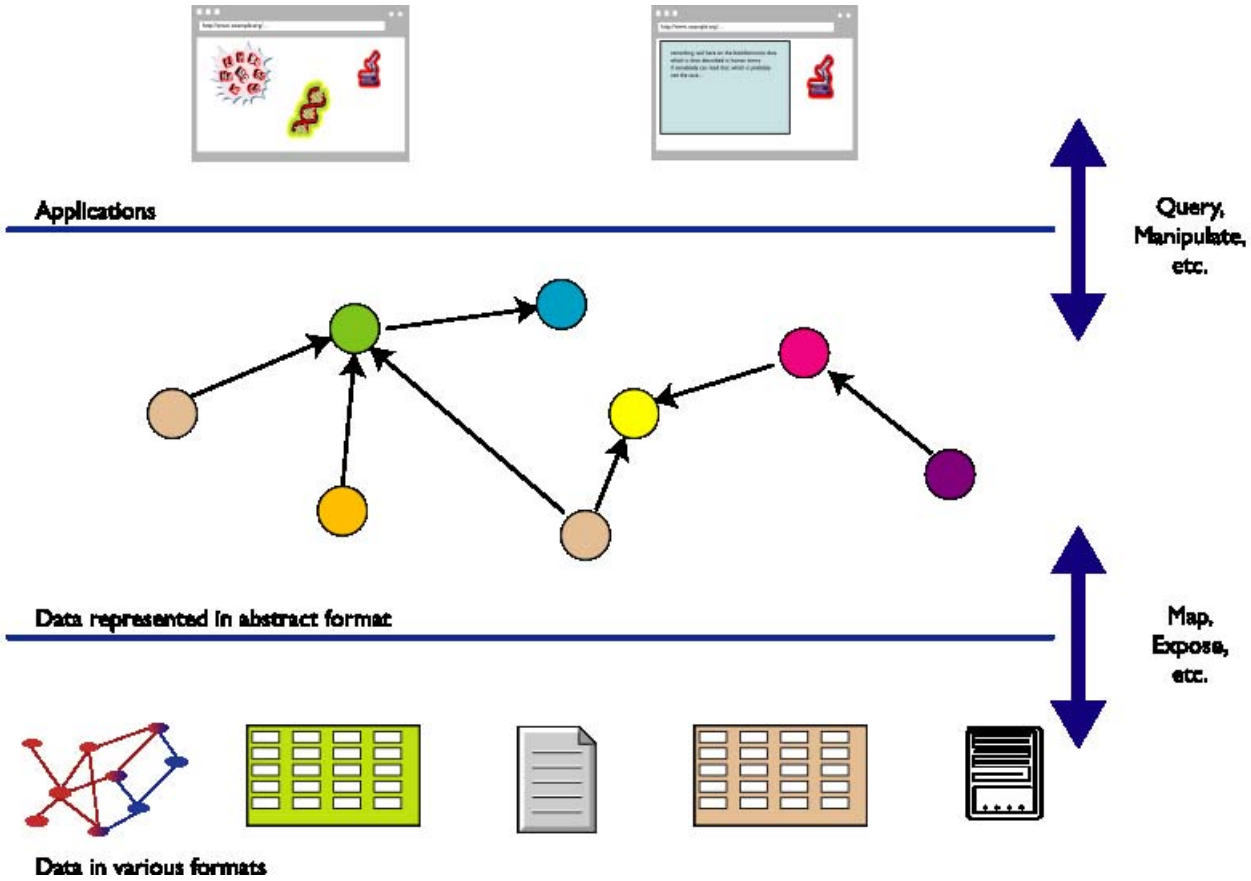
The “layercake”: where are we?



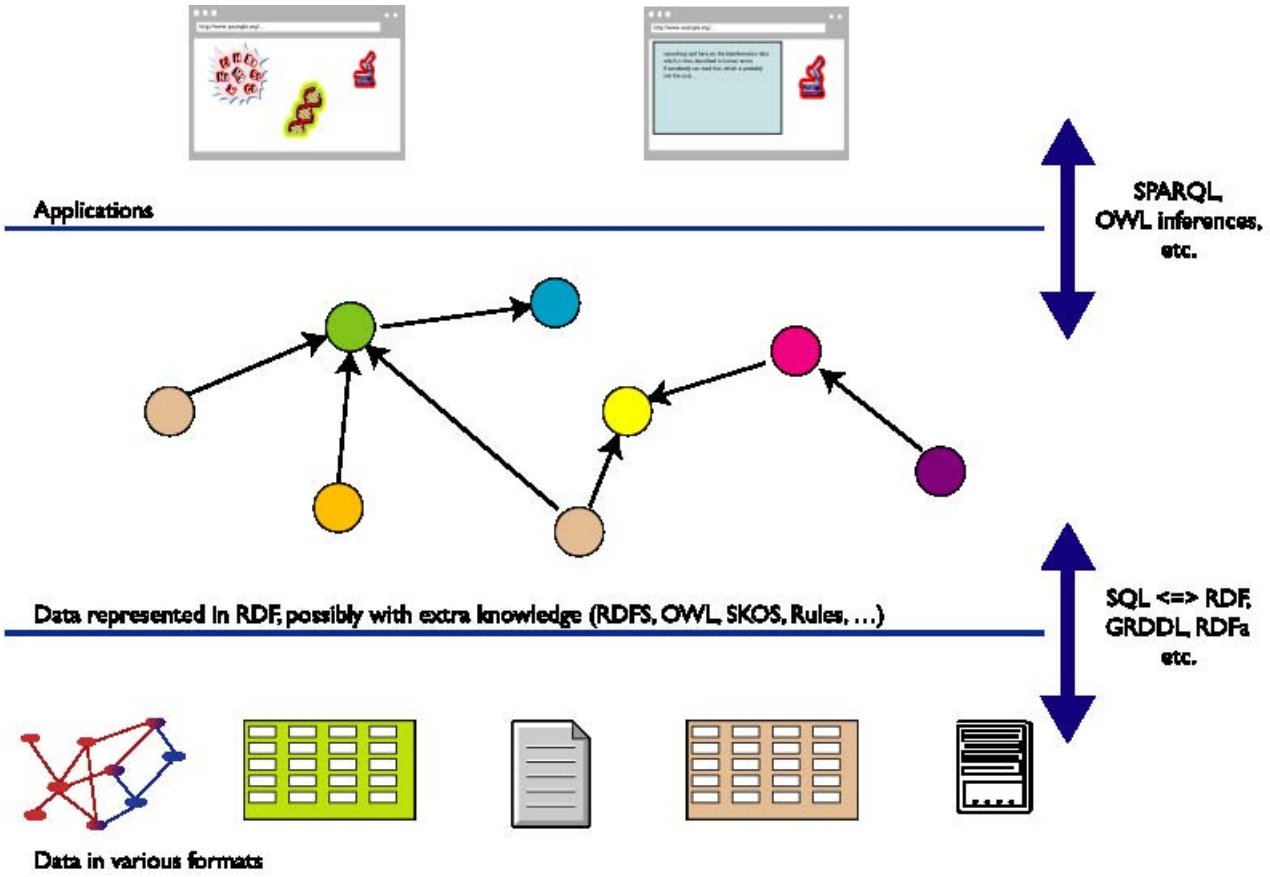
We have not talked about…

- **Ontologies (OWL)** stay around for Ian's tutorial!
- **Rules (RIF)** loosely: logic programming combined with the Semantic Web
  - *combine rules with RDF data*
  - *interchange rules among rule systems*
- the first draft has just been published…

Remember the integration example?



Same With What We Learnt



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
- Better modularization (import or refer to *part of* ontologies)
- Extensions of RDF and/or OWL (based on experience and theoretical advances)
- Temporal, spatial, fuzzy, probablistic, etc, reasoning
- …

SW in Practice

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 (possibly huge!) datasets*
  - *specialized editors, validators, …*
  - *SPARQL "endpoints" (ie, you can experiment with RDF data without any installations)*
  - *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

- A number of public “core” vocabularies evolve to be used by applications, e.g.:
  - *Dublin Core: about information resources, digital libraries, with extensions for rights, permissions, digital right management*
  - *FOAF: about people and their organizations*
  - *DOAP: on the descriptions of software projects*
  - *Music Ontology: on the description of CDs, music tracks, …*
  - *SIOC: Semantically-Interlinked Online Communities*
  - *vCard in RDF*
  - *SKOS to describe taxonomies, simple vocabularies, thesauri*
  - *…*
- They share the underlying RDF model (provides mechanisms for extensibility, sharing, …)

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 books

Further Information

- Dave Beckett's Resources at Bristol University
  - *huge list of documents, publications, tools, …*
- Semantic Web Community Portals, e.g.:
  - *Semanticweb.org*
  - *SW Tutorials on XML.com*
  - *Planet RDF: a blog aggregator on SW topics*
- The Semantic Web Activity has a number of further links

Public Fora and Resources at W3C

Semantic Web Interest Group

a forum developers with archived (and public) mailing list, and a constant IRC presence on freenode.net#swig; anybody can sign up on the list

Semantic Web Education and Outreach Interest Group

public archives of the Interest Group (although only members can sign up on the list directly)

Semantic Web Deployment Working Group

public archives of the Working Group (although only members can sign up on the list directly)

And the applications?

Come to my presentation tomorrow…

Thank you for your attention!

These slides are publicly available on:

```
http://www.w3.org/2007/Talks/0423-Stavanger-IH/
```

in XHTML and PDF formats; the XHTML version has active links that you can follow