From Semistructured/XML Data to the Semantic Web

V. Christophides
Computer Science Department, University of Crete
and Institute for Computer Science - FORTH
Heraklion, Crete, Greece

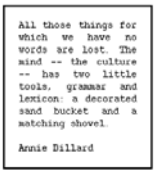
Outline

- A Brief History and Preliminary Notions
  - Documents
  - Semistructured Data
  - XML
- Sigmod'94 Test of Time Award and the XQuery Standard
  - Contributions
  - In Search of an XML Query Language
  - XQuery
- Myths and Really about the Semantic Web
  - Interoperability and Heterogeneity
  - Two Cultures on the Semantic Web
  - ICS-FORTH Contributions

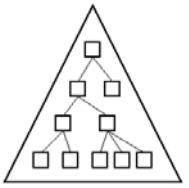
A Brief History and Preliminary Notions

What is a Document?

- Content: the components (words, images etc.) which make up a document
- Structure: the organization and inter-relationship of the components
- Presentation: how a document looks and what processes are applied to it



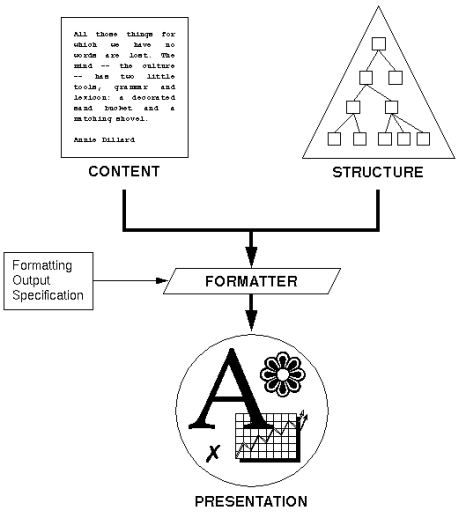
CONTENT


STRUCTURE


PRESENTATION

Separating these Things Means...

- The content can be re-used
  - for printing
  - for querying
  - for exchanging
- The structure can be formally validated
- The presentation can be customized for
  - different media
  - different audiences
- … in short, the information can be uncoupled from its processing



Documents vs Databases

Document world

- plenty of small documents
  - usually static
- implicit structure
  - section, paragraph, toc,
- tagging
  - human friendly
- content
  - form/layout, annotation
- paradigms
  - “Save as”, WYSIWYG
- metadata
  - author name, date, subject

Database world

- a few large databases
  - usually dynamic
- explicit structure
  - types
- records
  - machine friendly
- content
  - data, methods
- paradigms
  - Data Independence, Transaction Management, Query Languages
- metadata
  - schema description

Semistructured Data: A Personal Address Book

Records:

heterogeneous record collections

- name: V. Christophides
- phone: 391628
- phone: 393538

multiple attributes

attributes with different types in different objects

- name:
  - first name: Val
  - last name: Tannen
- phone: +1 (215) 898-2665

missing or additional attributes

- name: Abiteboul
- affiliation: INRIA

Semistructured Data vs Traditional Databases

- Data Schema is not what it used to be:
  - not given in advance (self-describing, schema-less),
  - descriptive, not prescriptive (ignored during querying),
  - partial (documents and data mixed together),
  - rapidly evolving (without notice),
  - may be large (compared to the size of the data)
- Data Types are not what they used to be:
  - elements and attributes are not strongly typed (irregular types)
    - missing or additional attributes
    - multiple attributes
  - elements in the same collection may have different types i.e., heterogeneous collections (contextual types)
    - attributes with different types in different elements
  - switch on/off data typing (indicative types)
    - accept elements and attributes not strictly conforming

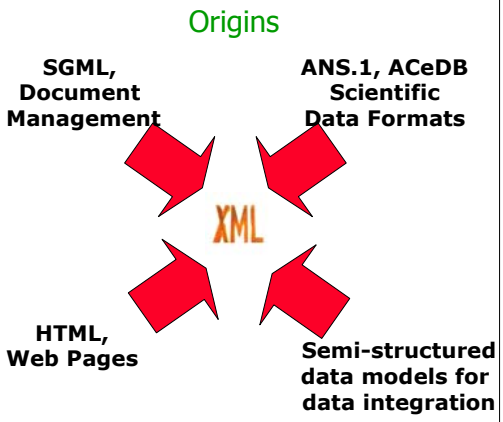
However Schemas are Useful for …

- Data readers
  - What info is in a given collection?
  - Thus, what queries might make sense?
- Data writers
  - What should I call this piece of info?
  - Is it okay to put this kind of data here?
- Efficient/effective data manipulation
  - Optimize query processing
  - Facilitate integration of multiple data sources
  - Improve storage
  - Construct indexes, statistics
  - Forbid certain types of updates

Towards a Convergence

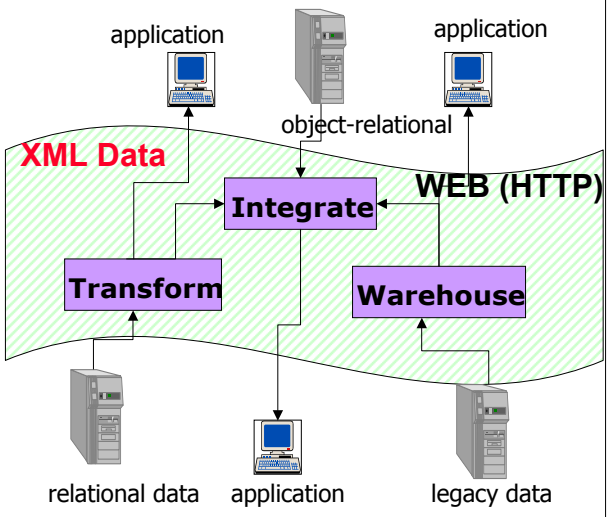
- Databases: relax rigid constraints imposed by schemas
  - Move to a flexible type system: semistructured data
- Documents: enrich formatting instructions with structuring/ semantic information
  - Add "types" to documents: XML

Semistructured Data $\tilde{=}$ XML



XML and Paradigm Shift on the Web

- New Web standard XML:
  - XML generated by applications
  - XML consumed by applications
- Data exchange:
  - across platforms
  - across organizations
- Web: from collection of documents to Web data published as documents

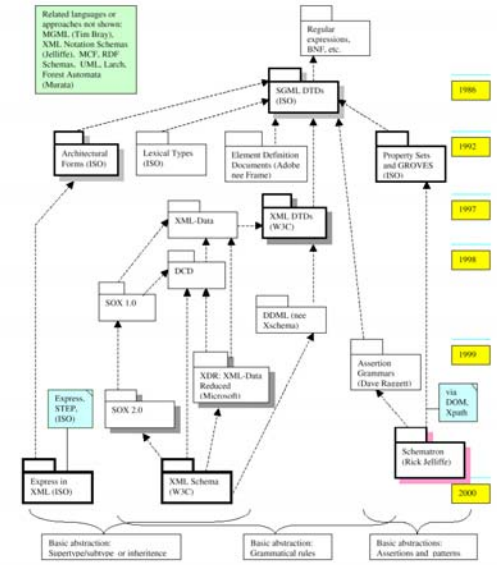


What is XML?

- Markup Meta-Language for domain or application specific structured information
  - Mathematical, chemical, musical, publishing, etc.
- Developed by the SGML Editorial Board formed under the auspices of the World Wide Web Consortium (W3C)
  - Founded in 1996 by Jon Bosac (Sun) and various Web/SGML vendors: Textuality, Netscape, Microsoft, INSO, HP, Highland, NCSA, ArbortText, GRIF, SoftQuand
- Subset of SGML optimized for use in the Inter/Intranet
  - SGML is proving difficult to implement for Web/Intranet applications
  - SGML has been hard to cost-justify to management

$$\text{HTML4.0} \in \text{XML} \subset \text{SGML}$$

The Big Picture

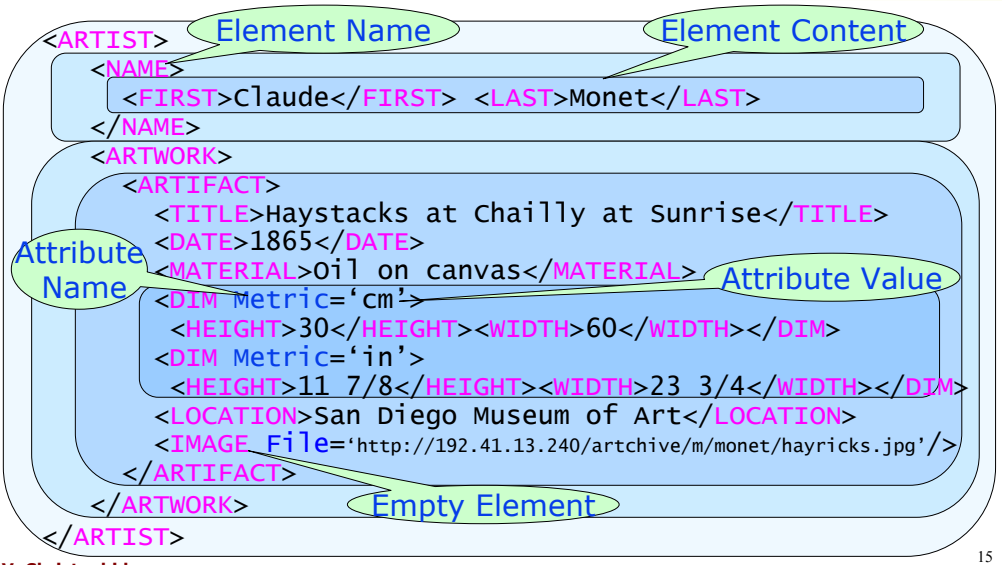


© Rick Jelliffe

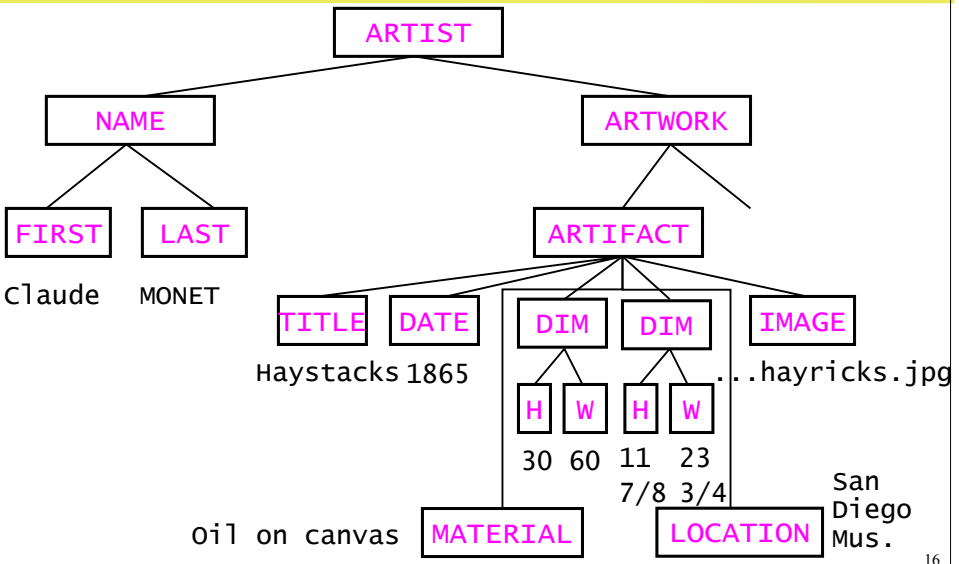
XML Core Markup Features

- Elements: Components of the tree logical structure defined by a DTD
  - identified in a document instance by descriptive markup, usually a start-tag and end-tag
- Attributes: Characteristics associated to the elements (other than their content and type)
  - may be applied to one specific instance of a given element
- Entities: Named fragments of information that can be stored separately from a document (or a DTD)
  - can be included in the document (or the DTD) one or more times by reference to their names

XML Data Representation: The Document View



XML Data Representation: The Database View



XML vs. HTML Markup

HTML describes the presentation

```
<B>MONET, Claude<B><BR>
Haystacks at Chailly at Sunrise<BR>
1865<BR>
Oil on canvas<BR>
30 x 60 cm (11 7/8 x 23 3/4 in.)<BR>
San Diego Museum of Art <BR>
<P><IMG SRC="http://192.41.13.240
/artchive/m/monet/hayricks.jpg">
```

XML describes the information content

```
<ARTIST>
<NAME><FIRST>Claude</FIRST><LAST>Monet</LAST></NAME>
  <ARTWORK>
     <ARTIFACT>
        <TITLE>Haystacks at Chailly at Sunrise</TITLE>
        <DATE>1865</DATE>
        <MATERIAL>Oil on canvas</MATERIAL>
        <DIM Metric='cm'>
           <HEIGHT>30</HEIGHT><WIDTH>60</WIDTH></DIM>
        <DIM Metric='in'>
           <HEIGHT>11 7/8</HEIGHT><WIDTH>23  3/4</WIDTH></DIM>
        <LOCATION>San Diego Museum of Art</LOCATION>
        <IMAGE File='http://192.41.13.240/artchive/m/monet/hayricks.jpg'/>
     </ARTIFACT>
  </ARTWORK>
</ARTIST>
```


The Secrets of XML Popularity

- It looks like HTML...
  - Simple, familiar, easy to learn, human-readable
  - Universal and portable
  - Supported by the W3C: trusted and quickly adopted by the industry
- …but it's more than HTML!
  - flexible: you can represent any information
  - extensible: you can represent it the way you want!
- Increasing typing precision in XML specifications
  - Well-Formed: already better than plain text
  - Valid: structure conforms to a DTD or an XML Schema

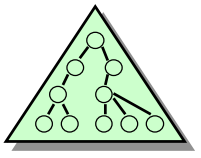


XML Document Type Definition (DTD)

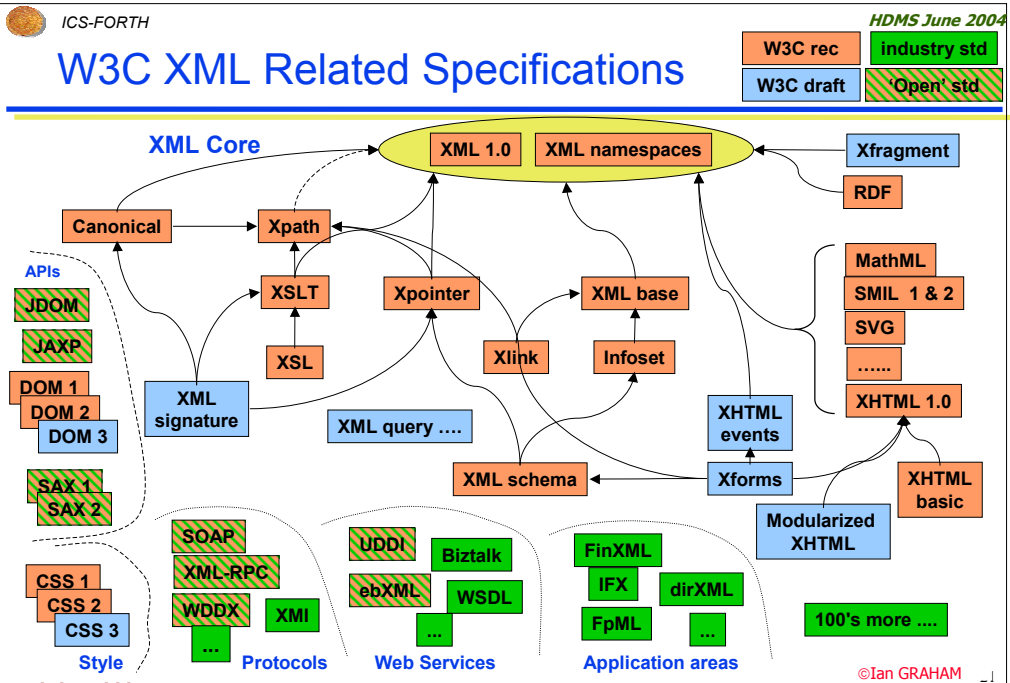
```
<!DOCTYPE artist [
<!ELEMENT artist (name, born, death, artwork, nationality?,
                  influences)>
<!ATTLIST artist oid ID #REQUIRED>
<!ELEMENT name      (first, last)>
<!ELEMENT first     (#PCDATA)>
<!ELEMENT last      (#PCDATA)> ...
<!ELEMENT artwork   (artifact+)>
<!ELEMENT artifact (title, date, material, dim*, location, image)>
<!ELEMENT title     (#PCDATA)> ...
<!ELEMENT dim       (height, width)>
<!ATTLIST dim metric (cm | in) 'cm'>
<!ELEMENT location (#PCDATA)>
<!ELEMENT image EMPTY>
<!ATTLIST image file ENTITY #REQUIRED>
<!ELEMENT influences (PCDATA | aref)*>
<!ELEMENT aref EMPTY>
<!ATTLIST aref oref IDREF #IMPLIED>
]>
```


We Want to Build XML-enabled Web Applications

- There is an urgent need for robust XML tools
- Designing XML tools is a data management problem:
  - XML 1.0 to describe semistructured data/documents = Syntax for trees
  - XML data models to describe the information content = Data model for trees
  - XML schemas to describe the structure of information = Data definition language for trees
  - XML languages to describe information processing = Data manipulation language for trees



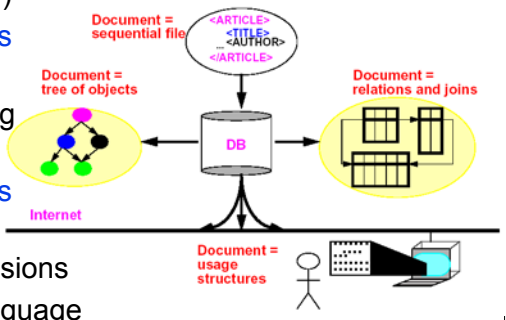
W3C XML Related Specifications



Sigmod'94 Test of Time Award and the XQuery Standard

From Structured Documents to Novel Query Facilities

- How we can represent SGML DTDs with ODMG schemas?
  - ordered tuples (sequence connector ",")
  - union types (choice connector "|")
- How we can store SGML documents in ODMG databases?
  - schema-aware (specific) mapping vs schema-oblivious (generic)
- How we can query SGML databases with OQL?
  - POQL: Generalized Path Expressions
  - Strongly Static Typed Query Language
- Follow-up publications
  - query optimization (centralized, distributed)



SIGMOD'94 Talk Slides

Example of an SGML DTD

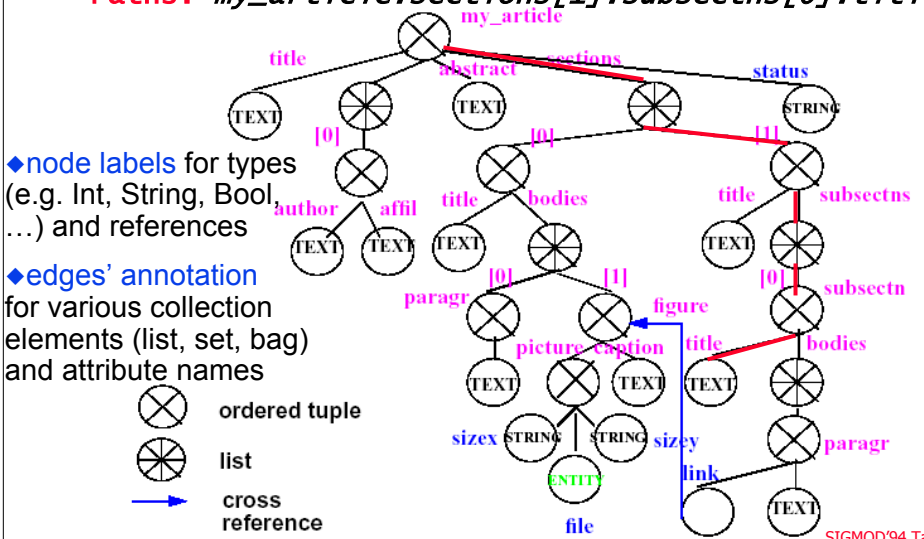
```
<!DOCTYPE article [
<!ELEMENT article (title, (author, affil)+, abstract, section+,
                   acknl, biblio)>
<!ATTLIST article status (final | draft) "draft">
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>.......
<!ELEMENT biblio (#PCDATA)>
<!ELEMENT section (title, body*, subsectn*)>
<!ELEMENT subsectn (title, body+)>
<!ELEMENT body (figure | paragr)>
<!ELEMENT figure (picture, caption?)>
<!ATTLIST figure id ID #REQUIRED>
<!ELEMENT picture EMPTY>
<!ATTLIST picture sizex NMTOKEN "16cm" sizey NMTOKEN #IMPLIED
                  file ENTITY #REQUIRED>
<!ELEMENT paragr (PCDATA | ref)*>
<!ELEMENT ref EMPTY>
<!ATTLIST ref link #IDREF #IMPLIED>
<!ENTITY fig1 SYSTEM "/u/christop/SGML/image1" NDATA bitmap>
<!NOTATION bitmap PUBLIC "-//local//NOTATION Bitmap Images//EN">]>
```

Representing SGML DTDs in O_2

```
class Article public type tuple(title: Title, a1: list
   (tuple(author:Author, affil:Affil)), abstract: Abstract, sections:
    list (Section), acknl: Acknl, biblio: Biblio, status:string)
class Title inherit Text
class Author inherit Text
class Affil inherit Text
class Abstract inherit Text
class Section public type tuple(title:Title, bodies:list (Body),
    subsectn:list(Subsectn))
class Subsectn public type tuple(title: Title, bodies: list (Body))
class Body public type union(figure: Figure, paragr: Paragr)
class Figure public type tuple(picture: Picture, caption: Caption,
    id: string)
class Picture public type tuple(sizex: string, sizey: string,
    file: Entity)
class Caption inherit Text
class Paragr public type list (tuple (text: Text, ref: Ref))
class Ref public type tuple(link: Figure)
name Articles: list (Article)
```


Representing SGML Documents in O_2

Paths: *my_article.sections[1].subsectns[0].title*



- ◆node labels for types (e.g. Int, String, Bool, …) and references
- ◆edges' annotation for various collection elements (list, set, bag) and attribute names

ordered tuple

list

cross reference

POQL Queries

- *Find subsections of articles having a figure with a caption containing the word "SGML"*

```
select ss
from Articles{a}.sections{s}.subsectns{ss},
     ss...figure(f)
where f.caption contains ("SGML")
```

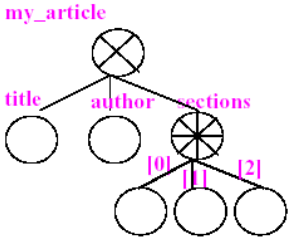
- *Find the paragraphs of articles just before figures*

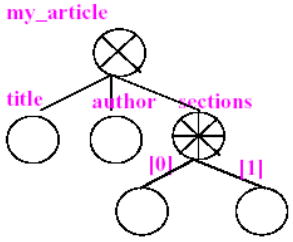
```
select p
from Articles{a}.sections{s},
     s...bodies[i].paragr(p),
     s...bodies[j].figure(f)
where i = j - 1
```


POQL Queries

- *Find the differences between two versions of my_article*

```
select @P                    select @P
from my_article@P      -     from my_article@P
```

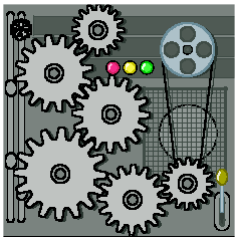





```
.title,.author,.sections,          .title,.author,.sections,
.sections[0],.sections[1], -       .sections[0],.sections[1]
.sections[2]
```

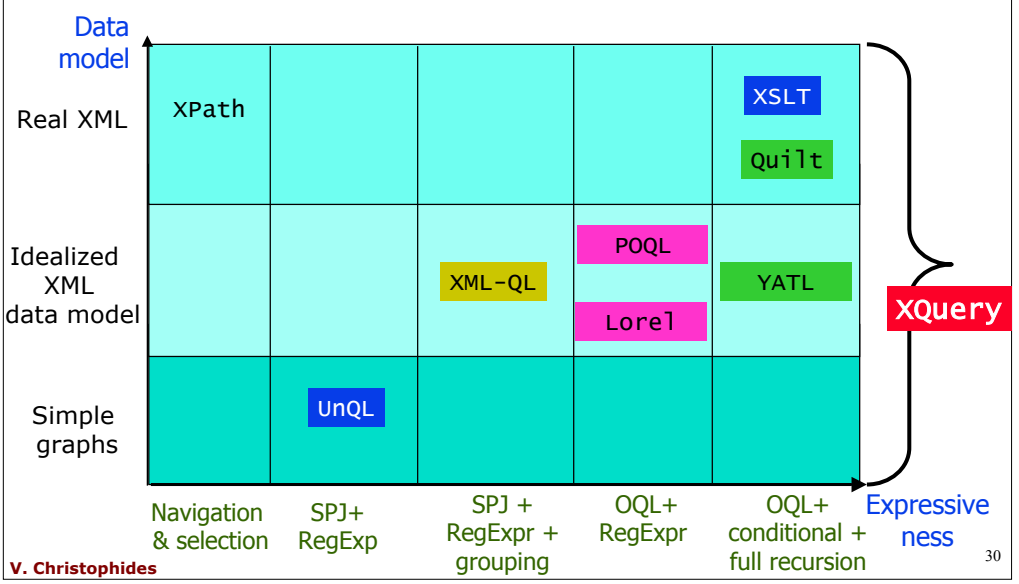
Q: { .sections[2] }

Some Relevant Query Languages



- Query languages for graph data
  - e.g. GOOD, GraphLog, Clean
- Query languages for the WEB
  - e.g. WebSQL, WebOQL
- Query languages for semi-structured data
  - e.g. MSL, UnQL, StruQL, YATL
- Research query (or programming) languages for XML
  - e.g. XML-QL, POQL, Lorel, XML-GL, Quilt, Xduce
- Industry query languages for XML
  - e.g. XQL
- Standard processing languages for XML (W3C standards)
  - e.g. XPath, XSLT

"Query" Language Paradigms



XML Query Language Requirements

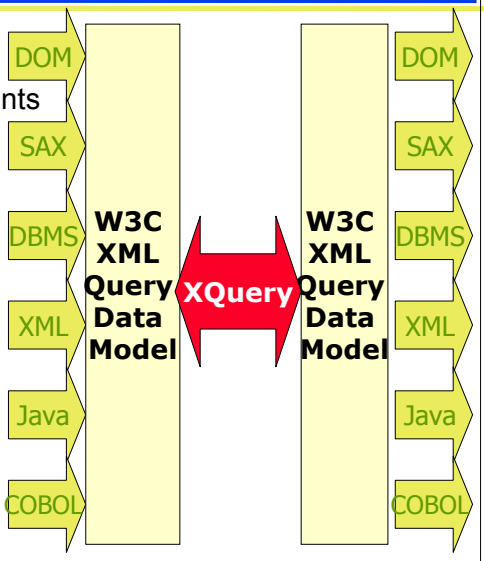
1. Select portions of a document
   - all
2. Copy portions of a document while preserving the hierarchy and the order of the nodes
   - UnQL (non-ordered graphs), YaTL, Quilt, XSLT
3. Combine (join) two documents
   - all except UnQL,XPath and XSLT (only intra-document joins)
4. Construct new documents
   - all except XPath
5. Navigate irregular or unknown document structures
   - full (vertical) regular expressions: POQL (GPE), UnQL, Lorel, XML-QL
   - simple (vertical) navigation: XPath, XSLT, Quilt
   - horizontal regular expressions: YaTL

XML Query Language Requirements

6. Formulate predicates on the tag and attribute names
   - all
7. Query and preserve the nodes global topological order
   - POQL, Quilt
8. Apply aggregation and sorting functions
   - all except XPath, UnQL, XML-QL
9. Apply existential and universal quantifiers
   - POQL, Lorel, Quilt, XSLT (not explicitelly!)
10. Apply full-text predicates and text operations
    - none satisfactorily

The XQuery W3C Standard

- W3C Specification (Working Draft 12 November 2003)
  - Satisfies the XML Query requirements
- Formal semantics based on XML Abstract Data Model
  - The input and output of an XQuery are instances of the XML Query Data Model
- XQuery is a functional language in which a query is represented as an expression
  - The result of the query is the result of the evaluation of the expression
  - Expressions are evaluated in a certain environment
  - XQuery expressions can be nested with full generality



XQuery Abstract Data Model

- Common for XPath 2.0 and XQuery 1.0
- A logical model based on the notion of an ordered tree and composed of
  - a set of logical entities
  - constructors and accessors for each entity
- Simplified Type System
  - Nodes

```
Node = DocNode | ElemNode | AttrNode | ValueNode
  | NSNode | PINode | CommentNode | InfoItemNode
```

  - XML Schema Primitive Types

```
string, boolean, ID, IDREF, decimal, QName, ...
```

  - Collections

sequence	set	bag	union
`[T]`	`{T}`	`{|T|}`	`T1 | T2`

  - References

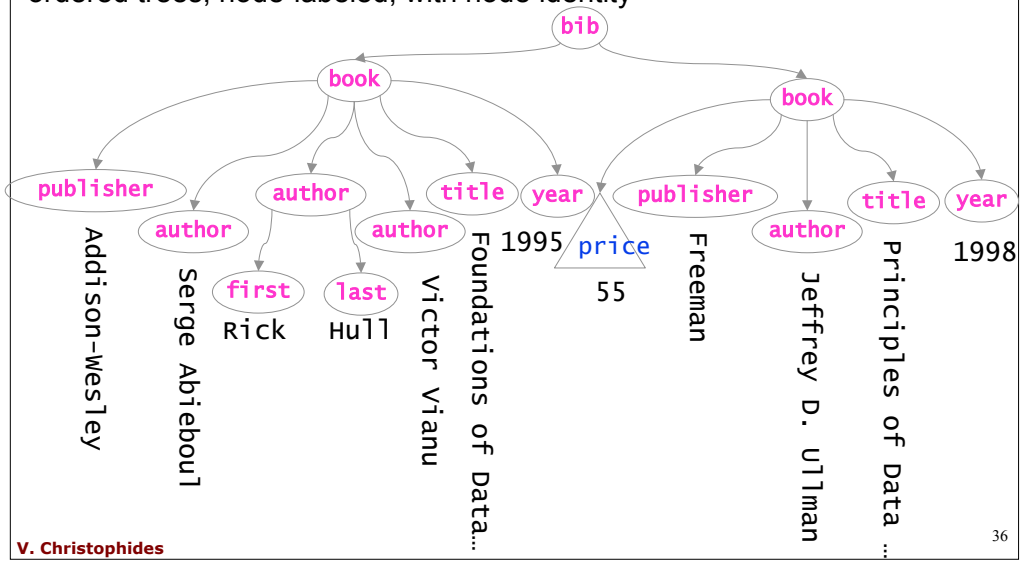
```
ref(T)
```

Bibliographic XML Data

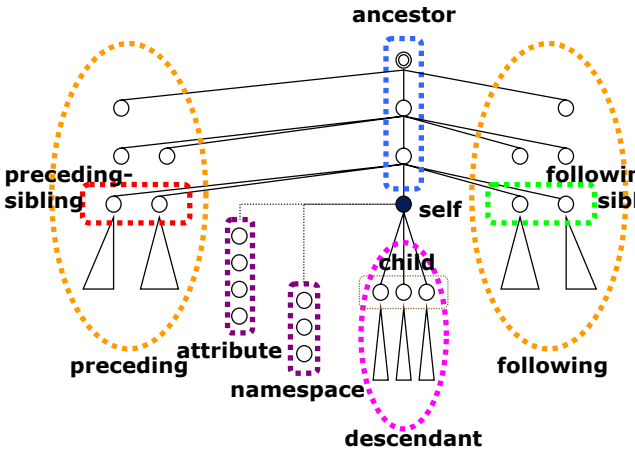
```
<bib>
  <book>
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
    <author><first>Rick</first><last>Hull</last></author>
    <author>Victor Vianu</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
 <book price = "55">
    <publisher>Freeman</publisher>
    <author>Jeffrey D. Ullman</author>
    <title>Principles of Databases</title>
    <year>1998</year>
  </book>
</bib>
```


Bibliographic XML Data Trees

ordered trees, node-labeled, with node identity



XPath Navigation Axes



Thirteen navigation axes:

- self (self::node() == .);
- child (omitted when abbreviated);
- parent (parent::node() ==..);
- attribute (abbreviated to @);
- namespace;
- descendant-or-self (descendant-or-self::node() ==//);
- descendant;
- ancestor-or-self;
- ancestor;
- preceding;
- preceding-sibling;
- following;
- following-sibling.

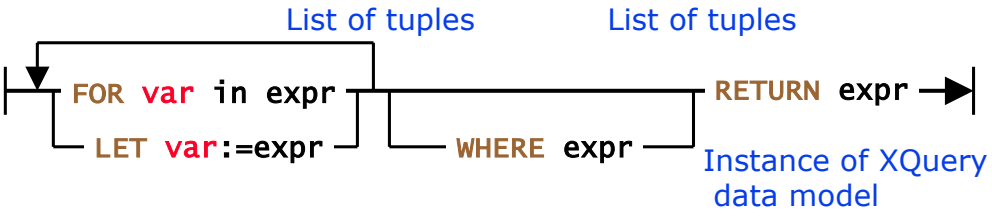
©Arnaud Sahuguet

XQuery Complex Expressions

- XPath expressions (for navigation)
  - reshuffled XPath 2.0
- FLWR expressions (for iteration):
  - FOR…LET…WHERE…RETURN…
- Sorting:
  - ORDERBY…ASCENDING/ DESCENDING
- Aggregate Functions:
  - COUNT AVG SUM MIN MAX
- Grouping:
  - implicit using nested queries
- (Left/Full Outer) Joins:
  - Implicit using nested queries
- Set operations:
  - UNION INTERSECT EXCEPT
- Conditional Expressions:
  - IF…THEN…ELSE…
- Quantifiers:
  - SOME/EVERY…SATISFIES
- Full-text predicates:
  - CONTAINS
- Type Switches:
  - TYPESWITCH…CASE…DEFAULT…
- Casting, Treating and Asserting:
  - CAST AS, TREAT, ASSERT
- Local Functions
  - Currently: arbitrary recursion

XQuery FLWR ("Flower") Expression

- A FLWR expression binds some expressions, applies a predicate, and constructs a new result



FOR and **LET** clauses generate a list of tuples of bound expressions, preserving document order

WHERE clause applies a predicate eliminating some of the tuples

RETURNS clause is executed for each surviving tuple, generating an ordered list of outputs

XQuery FLWR ("Flower") Expression: Example

- Query: *find all books titles published after 1995*

```
FOR $x IN document("bib.xml")/bib/book
WHERE $x/year >= 1995
RETURN $x/title
```

- Result:

```
<title>Foundations of Databases</title>
<title>Principles of Databases</title>
```

XQuery FOR vs. LET

- **FOR** Query: binds var to each element in the list expr

```
FOR $x IN document("bib.xml")//book
RETURN <result> $x </result>
```

  - Returns

```
<result> <book>...</book></result>
<result> <book>...</book></result>
```

- **LET** Query: binds var to the entire list expr

```
LET $x := document("bib.xml")//book
RETURN <result> $x </result>
```

  - Returns

```
<result> <book>...</book
         <book>...</book>
         ...
</result>
```

  - Useful for common subexpressions and for aggregations

Nesting/Composing XQuery Expressions

```
<bibliography>
  Expression
</bibliography>
```

- Every expression has a value and no side effects
- Expressions are fully composable
- Expressions propagate the error value
  - Exception: **and**, **or**, quantifiers have "early-out"

Nesting/Composing XQuery Expressions

```
<bibliography>
  {
    FOR $b IN Expression
    RETURN
     Expression
   }
</bibliography>
```


Nesting/Composing XQuery Expressions

```
<bibliography>
  {
    FOR $b IN Expression
    RETURN
     <book>
       {
           Expression ,
           Expression
       }
     </book> ORDERBY (Expression , Expression)
  }
</bibliography>
```

Nesting/Composing XQuery Expressions

```
<bibliography>
  {
     FOR $b IN document("bib.xml")//book
     RETURN
      <book>
        {
             $b/author,
             $b/title
        }
      </book>  ORDERBY (author, title)
  }
</bibliography>
```


XQuery & Types

- XML documents contains a wide range of information:
  - From …
    Loosely typed information (without a schema)
  - To …
    Rigidly structured data
- So, a language for querying XML must:
  - Avoid assumption about what is allowed
  - Allow data to be managed without frequent casting of values
  - Allow programmer to focus on the document and not on the whims of the type system

Static and Dynamic XQuery Typing

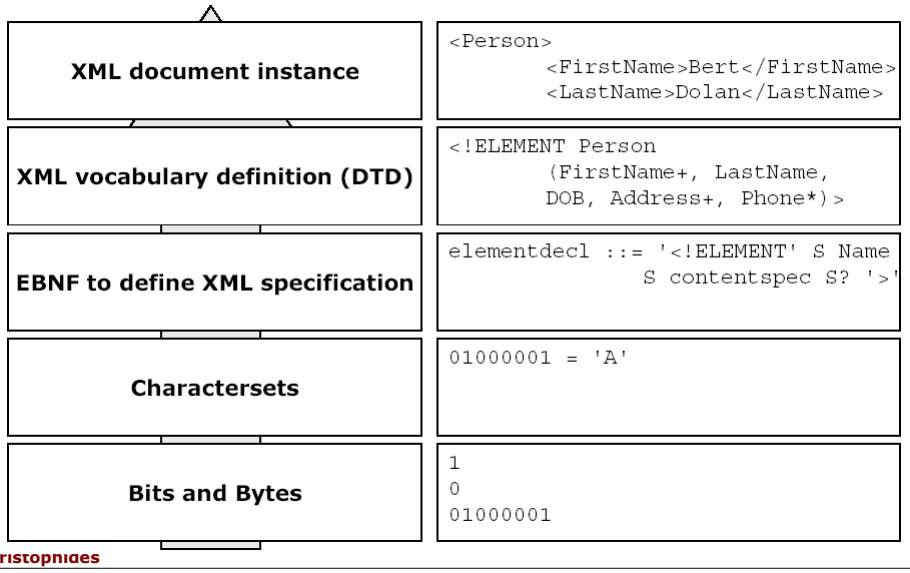
- Static Typing: to detect type errors before an XQuery is executed
  - Static type: is a compile-time property of an expression
  - Static type errors: our well-known type errors
- Dynamic Typing: specifies the relationship between input data, an XQuery expression, and output data
  - Dynamic type: the type of an operand that is determined at runtime
  - Dynamic error: occurs during evaluation of a query. Causes implicit invocation or an error function -- fn:error()
- A processor that implements static typing can detect some kinds of errors by comparing a query to the imported schemas
  - This means that no data is required to find these errors

Past and Ongoing Research on XML Data

- XML Data Semantics
  - Type Systems
  - Structural & Integrity Constraints
  - Incremental Validation
- XML Query Processing
  - XQuery Algebras
  - Tree Query Pattern Containment & Minimization
  - XPath Engines
  - Stream-based Query Processing
- XML Query Optimization
  - Storage Schemes
  - Labelling & Indexing Schemes
  - Structural Joins & Cost Models
  - Data Statistics & Compression
  - Benchmarks, Real & Synthetic Data
- XML Data Management
  - Updates, Evolution & Versioning
  - Access Control & Active Rules
  - Data Publishing & Relational Databases
  - Warehouses & View Maintenance
- XML Database Systems
  - Commercial DBMS
  - Native DBMS

Myths and Really about the Semantic Web: The ICS-FORTH Experience

XML Anatomy

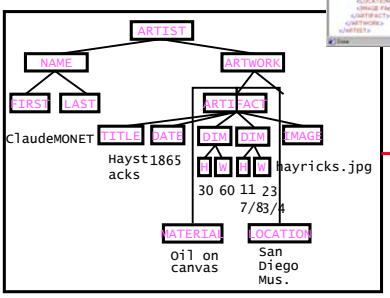


Is XML the Solution to Interoperability?

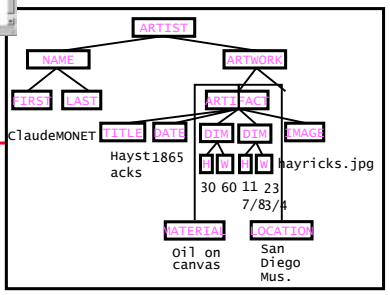
Document = medium for exchanging information



- Still need to agree on:
  - DTDs or Schemas
  - Meaning of tags
  - "Operations" on data
  - Meaning of operations

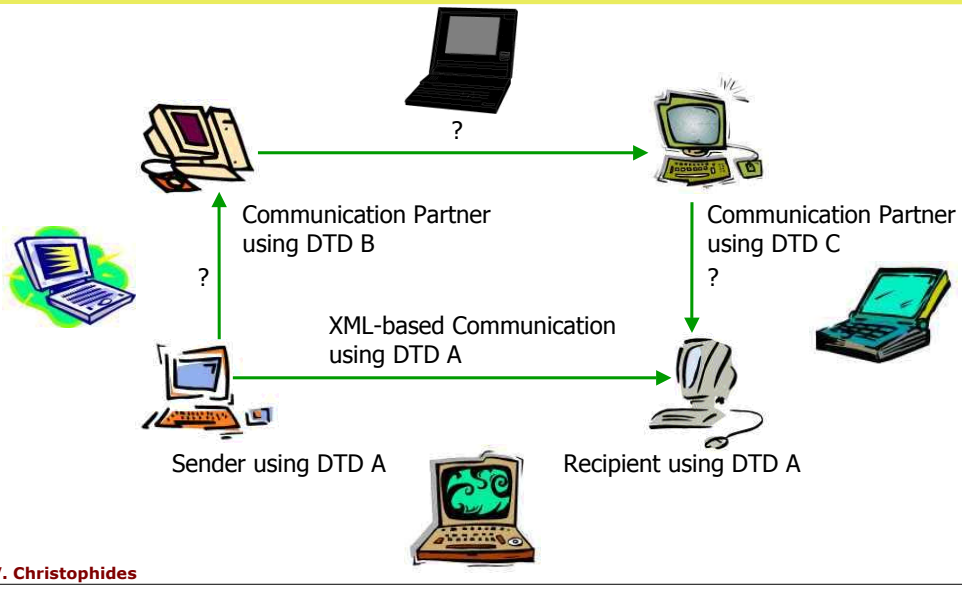


Communication

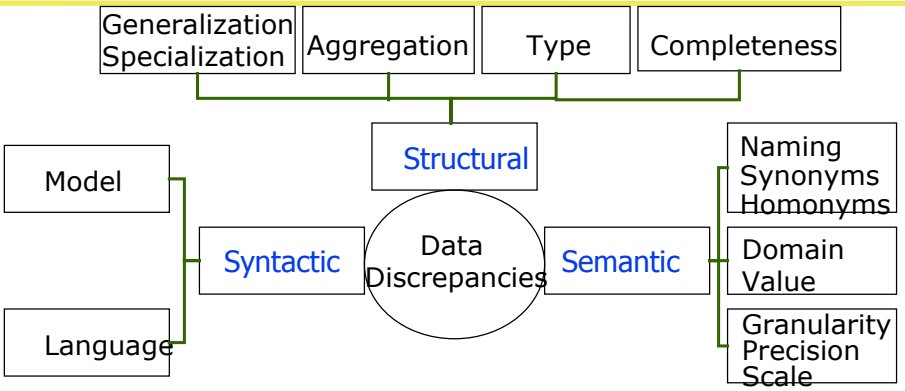


Application 1 Application 2

Large Scale Interoperation on the Web



Recall Data Heterogeneity



- XML is a Universal Format capturing data from different Models
  - Relational or Object DBMS
  - Document and File Repositories
- Semantic (and structural) heterogeneity occurs when there is a disagreement about the meaning, interpretation, or intended use of the same or related data

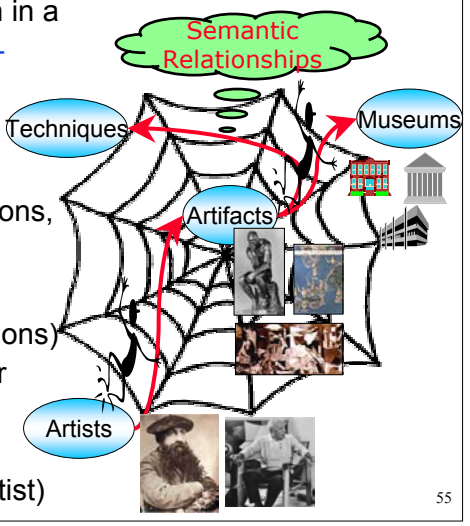
Interoperability is still an Open Issue !

- Semantic discrepancies :
  - Synonymy & Polysemy & Taxonomy
    - <ARTIFACT> vs. <ARTEFACT>
    - is <ARTWORK> paintings or songs ?
    - how < … Style='Impressionism'> is related to < … Style='Pointillism'> ?
- Structural discrepancies :
  - Aggregation
    - <NAME><FIRST>Claude</FIRST><LAST>Monet</LAST></NAME> vs <NAME>Claude Monet</NAME>
  - Type
    - <ARTIFACT Kind='Painting'> ... </ARTIFACT> vs <PAINTING>Claude Monet</PAINTING>
- Syntactic discrepancies :
  - <ARTIST Name='Claude Monet'> ... </ARTIST> vs <ARTIST> <NAME>Claude Monet</NAME> ... </ARTIST>

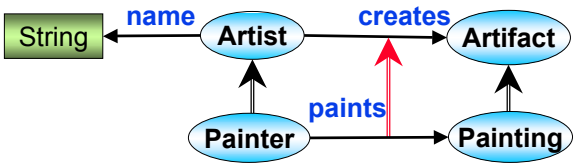
More than Web Data: Semantics on the Web
More than Web Applications: Web Services

The Semantic Web Vision: A Web of Meaning

- The "Next Generation Web" aims to provide infrastructure for expressing information in a precise, human-readable, and machine-interpretable form
- Enable both syntactic and semantic/ structural interoperability among independently-developed Web applications, allowing them to efficiently perform sophisticated tasks for humans
- Enable Web resources (data & applications) to be accessible by their meaning rather than by keywords and syntactic forms
  - Conceptual Navigation & Querying
  - Inference Services (Picasso is an Artist)



A First Step Towards the SW: RDF and RDFS

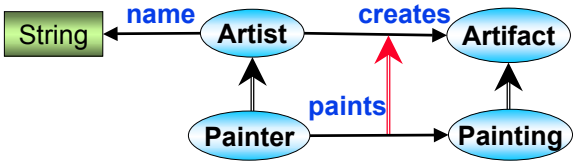



```
<Artist rdf:about="picasso132">
  <name>Pablo Picasso</name>
  <creates>
    <Artifact rdf:about=
    http://www.artchive.com/woman.jpg/>
  </creates>
</Artifact>
```

```
<Artist rdf:about="picasso132"
        name ="Pablo Picasso">
  <creates Artifact =
    http://www.artchive.com/woman.jpg/>
</Artifact>
```

```
<Painter rdf:about="picasso132">
  <name>Pablo Picasso </name>
  <paints>
    <Painting rdf:about=
    "http://www.artchive.com/woman.jpg"/>
  </paints>
  <paints>
    <Painting rdf:about="http://
     museoreinasofia.mcu.es/guernica.gif">
    </Painting>
  </paints>
</Painter>
```

A First Step Towards the SW: RDF and RDFS




```
<rdfs:Class rdf:ID="Artist"/>

<rdfs:Class rdf:ID="Artifact"/>

<rdfs:Class rdf:ID="Painter">
 <rdfs:subClassOf rdf:resource="#Artist"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Painting">
 <rdfs:subClassOf rdf:resource="#Artifact"/>
</rdfs:Class>

<rdf:Property rdf:ID="name">
 <rdfs:domain rdf:resource="#Artist"/>
 <rdfs:range rdf:resource="http://www.w3.org/
                  rdf-datatypes.xsd#String"/>
</rdf:Property>
```

```
<rdf:Property rdf:ID="creates">
 <rdfs:domain rdf:resource="#Artist"/>
 <rdfs:range rdf:resource="#Artifact"/>
</rdf:Property>

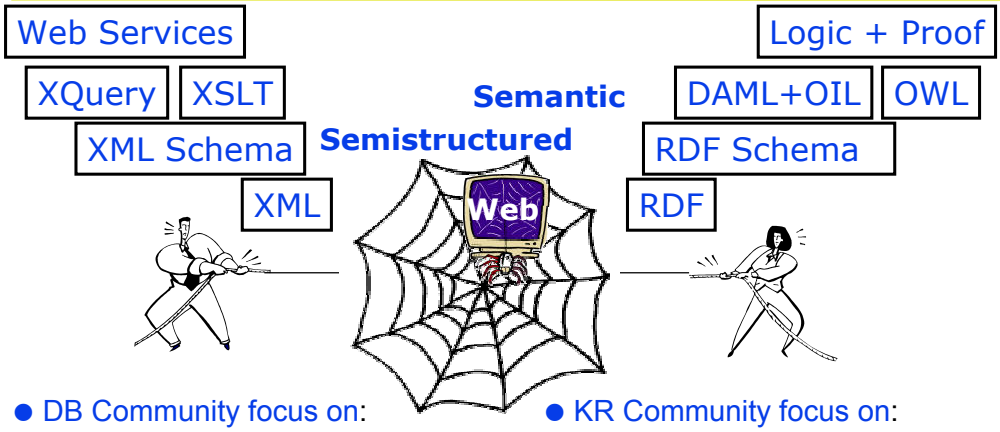
<rdf:Property rdf:ID="paints">
 <rdfs:domain rdf:resource="#Painter"/>
 <rdfs:range rdf:resource="#Painting"/>
 <rdfs:subPropertyOf
                  rdf:resource="#creates"/>
</rdf:Property>

<rdf:Property rdf:ID="created">
 <rdfs:domain rdf:resource="#Painting"/>
 <rdfs:rangerdf:resource="http://www.w3.org/
                  rdf-datatypes.xsd#Date"/>
</rdf:Property>
```


Is RDF/S the Solution to Interoperability?

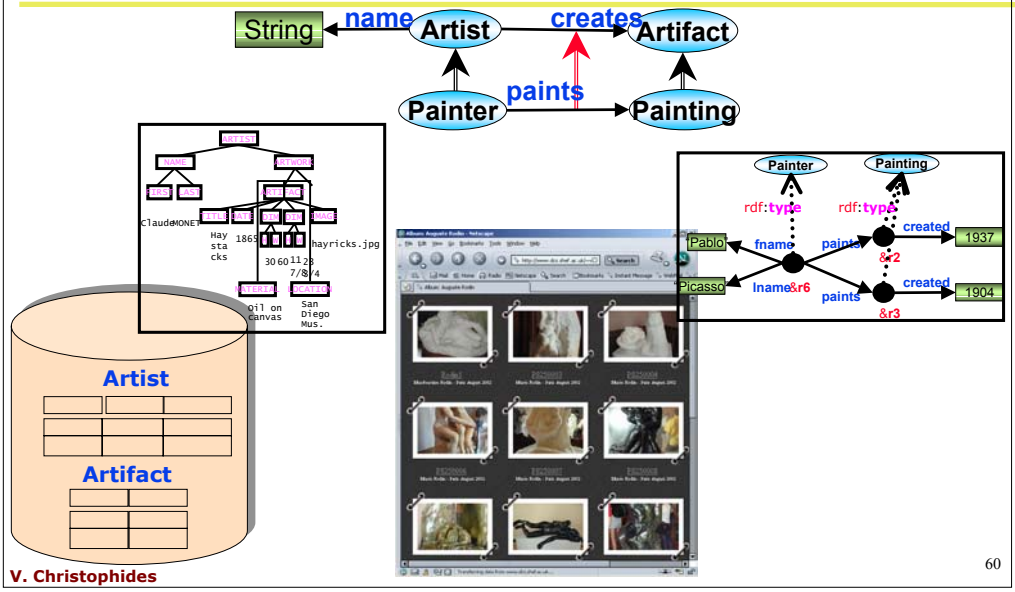
- RDF/S abstracts from the syntactic discrepancies of XML data (elements vs attributes)
  - but it introduce new ones, related to its own model & syntax (classes vs properties, unique identifiers of resources)
  - ➔we can't read arbitrary XML data and interpret them as RDF!
- RDF/S provides core primitives for modeling the semantics of data in a domain of discourse (extended ER models)
  - however application data reside in autonomous sources, structured according to different schemas
  - ➔we can't expect that all existing data will be published on the SW as RDF/S data committing to one commonly agreed ontology (schema)!
- We still need expressive languages for mapping ontologies as well as translate accordingly the data from one application to another
  - finding semantic mappings is now the bottleneck!
  - ➔largely done by hand, labor intensive & error prone !

Two Cultures on the Future Web: DB vs KR

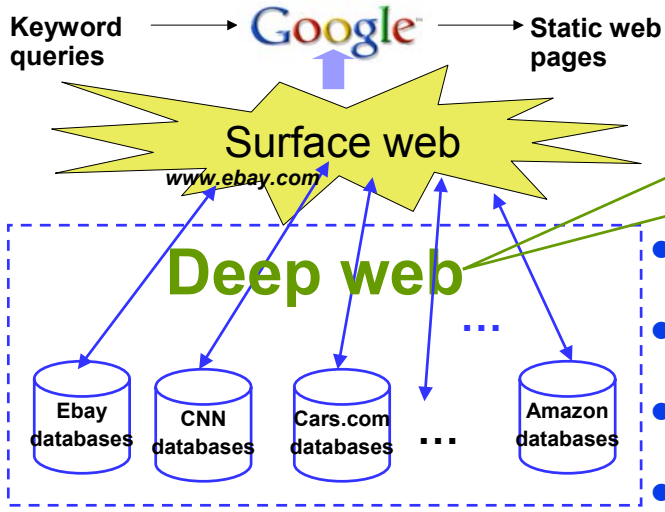


- DB Community focus on:
  - XML Data Semantics (Typing, Constraints)
  - XML Data Manipulation Languages (Querying, Views, Programming)
- KR Community focus on:
  - Ontology Languages (Frame / Description Logics)
  - Reasoners and Theorem Provers

Similar Motivations but different Application Contexts!



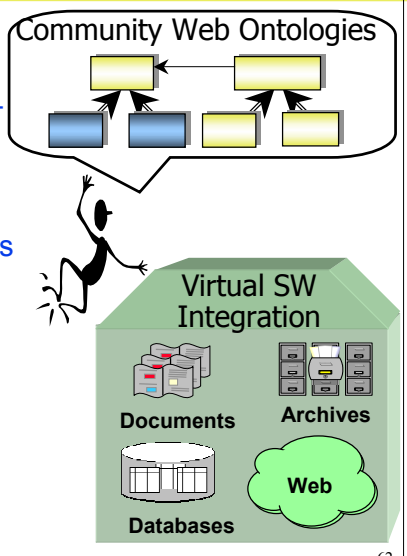
Visible (Surface) vs Invisible (Deep) Web



- Variety of Data formats & search mechanisms
- Accessible from specific HTML pages
- Higher Quality Information
- Not indexed by Google or other major search engines

Our Vision: Combine DB and KR Approaches

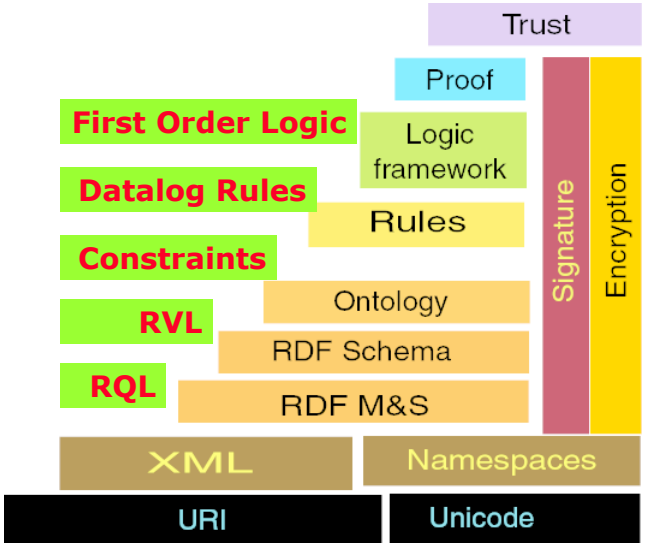
- Provide a useful, comprehensive, and high-level access to community resources
  - Ontologies as shared, formal conceptualizations of particular domains
- Build scalable technologies for managing semantically rich data and metadata
  - Declarative Querying/Viewing Languages
  - Efficient Storage for Voluminous Descriptive Information
- Support an expressive SW Integration Middleware
  - Establish Mapping/Translation Rules
  - Reformulate Conceptual Queries
  - Exploit data semantics for Query Optimization and Consistency Checking



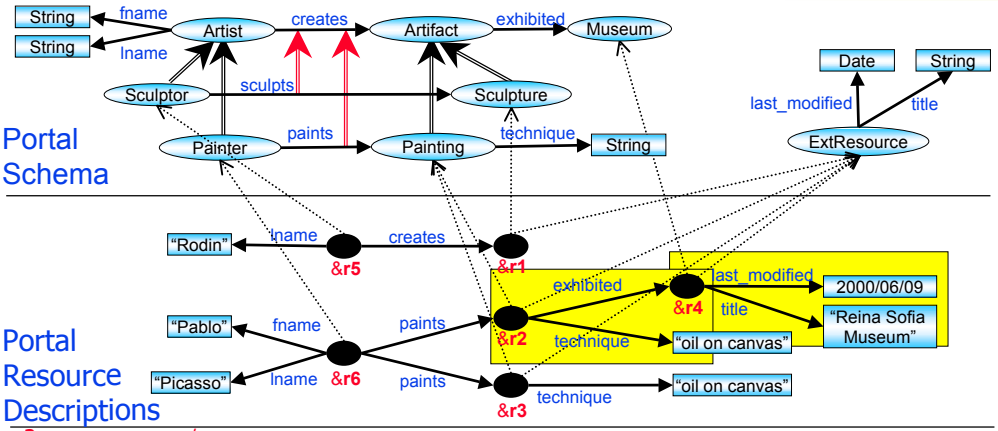
W3C Semantic Web Activity

- Semantic Web Activity (http://www.w3.org/2001/sw/)
  - "Established to serve a leadership role, in both the design of enabling specifications and the open, collaborative development of technologies that support the automation, integration and reuse of data across various applications"
  - Successor to the W3C Metadata Activity
- RDF Core Working Group (http://www.w3.org/2001/sw/RDFCore/)
  - Responsible for the Resource Description Framework RDF (http://www.w3.org/RDF/)
- Web Ontology Working Group (http://www.w3.org/2001/sw/WebOnt/)
  - Charter: Build upon the RDF Core work a language for defining structured web based ontologies which will provide richer integration and interoperability of data among descriptive communities
  - Developing Ontology Web Language OWL (http://www.w3.org/2004/OWL/)
    - Based on DAML+OIL, developed in DARPA's Agent Markup Language program

SW Layer Cake and ICS-FORTH Vision

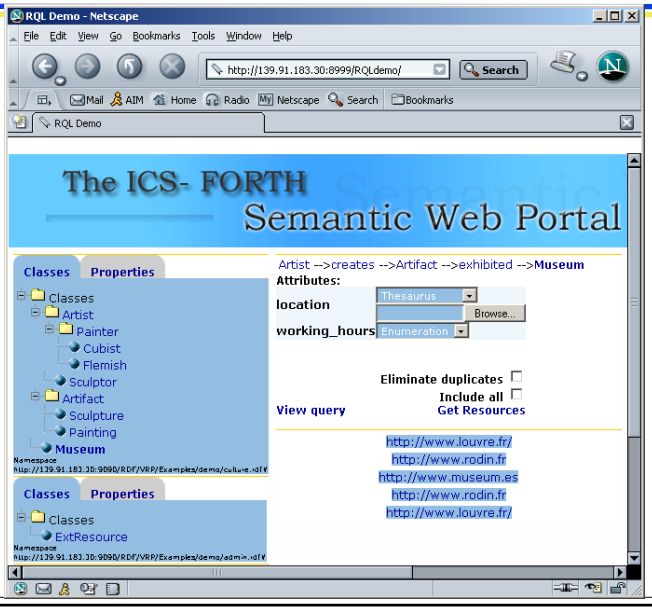


A Cultural Community Web Portal in RDF



- node labels for literal types and class names
- edges' annotation for property names

Semantic Web Portal Interface



Advantages of RDF/S vs. Well-Known Formalisms

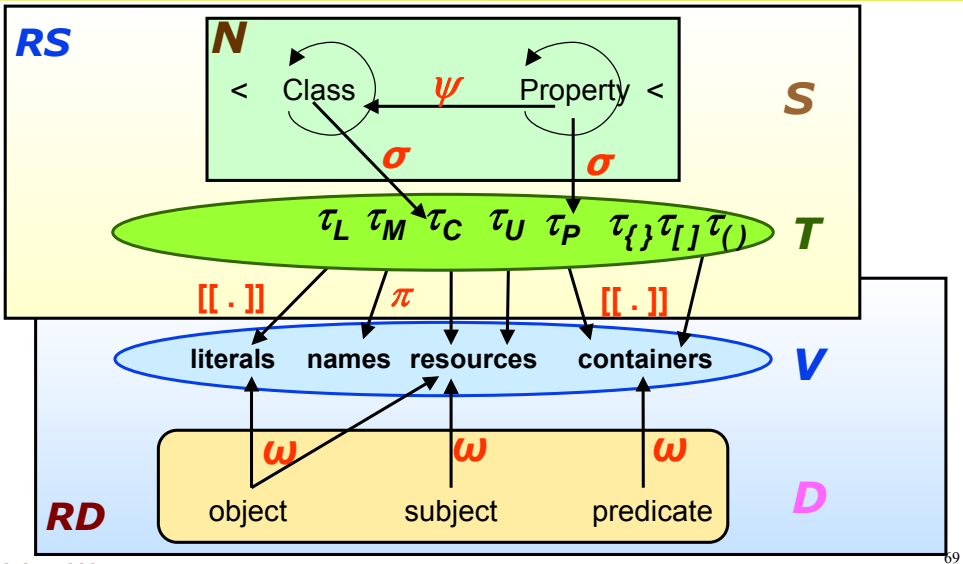


- Relational or Object Database Models (ODMG, SQL)
  - Instances may be associated with different properties
  - Heterogeneous Collections
- Semistructured or XML Data Models (OEM, UnQL, YAT, XML Schema)
  - Labels on both nodes or edges
  - Both class and property subsumption
- Knowledge Representation Languages (Telos, DL, F-Logic)
  - Supports complex values (bags, sequences)

A Formal Data Model for RDF/S

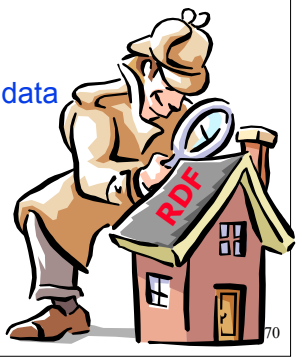
- An RDF schema is a tuple: S = (RS, σ)
  - RS = (V_S, E_S, H, ψ, λ, N, <) is a valid RDF Schema
  - σ is a type function: N $\rightarrow$ **T**
- An RDF description base, instance of a schema S, is a tuple: D = (RD,ω)
  - RD=(RS, V_D, E_D, ψ, λ) is a set of valid resource descriptions
  - ω is a valuation function: $V_D \cup E_D \rightarrow$ **V** such that:
    - $\forall$ n $\in$ V_D, ω (n) $\in$ [[σ (λ (n))]]
    - $\forall$ p $\in$ E_D from node n to n', [ω(n), ω(n')] $\in$ [[p]]

A Formal Data Model for RDF/S

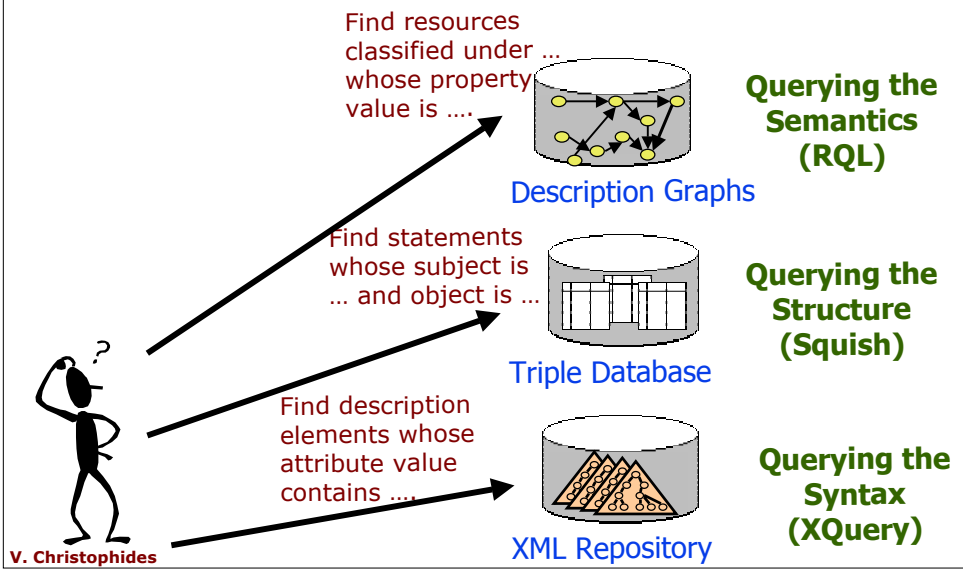


The RDF Query Language: RQL

- Declarative query language for RDF description bases
  - relies on a typed data model (literal & container types + union types)
  - follows a functional approach (basic queries and filters)
  - adapts the functionality of semistructured or XML query languages to RDF, but also:
    - treats properties as first-class citizens
    - exploits taxonomies of node and edge labels
    - allows querying of schemas as semistructured data



The RQL Approach



Discover the Schema of RDF Descriptions

- *Find the description of resources whose URI match "www.museum.es"*

```
select $C,  (select @P, Y
             from {Z ; $Z} @P {Y}
             where X = Z and $C = $Z)
from $C {X}
where X like "*http://www.museum.es*"
```

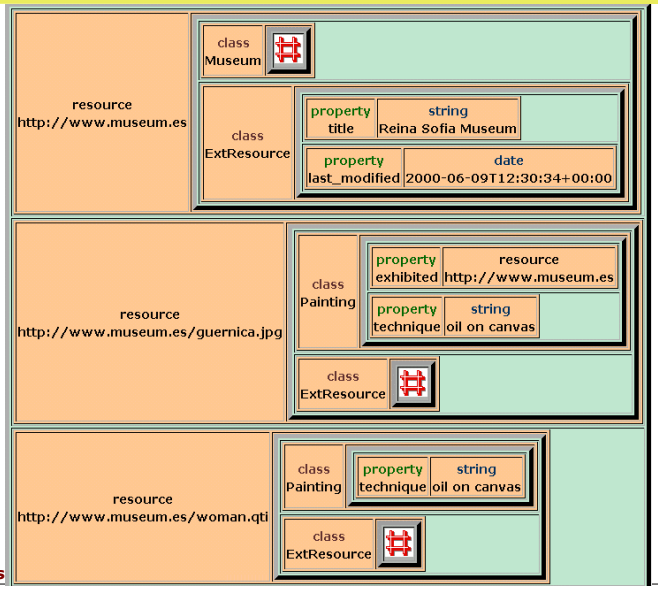
property variable

resource variables

class variables

patterns

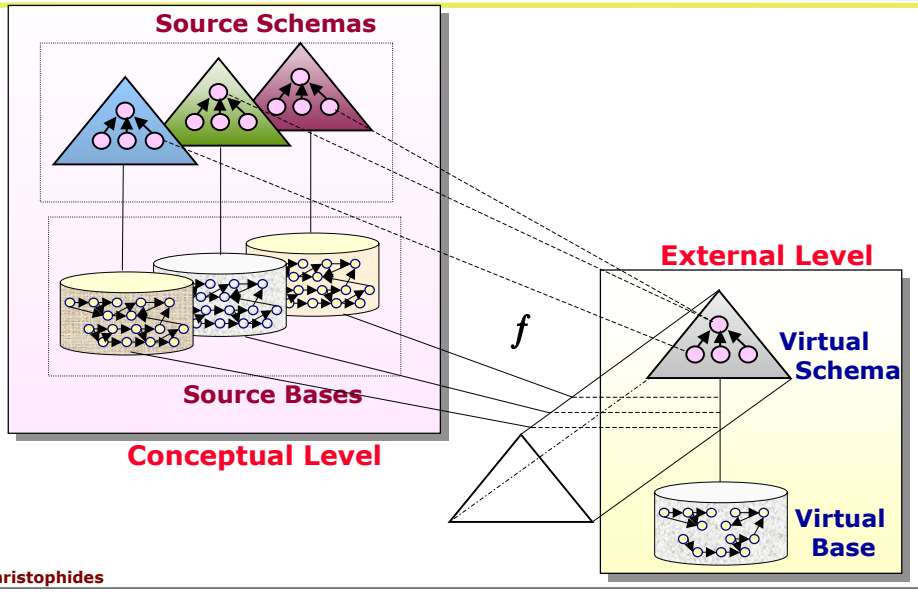
RQL Query Result



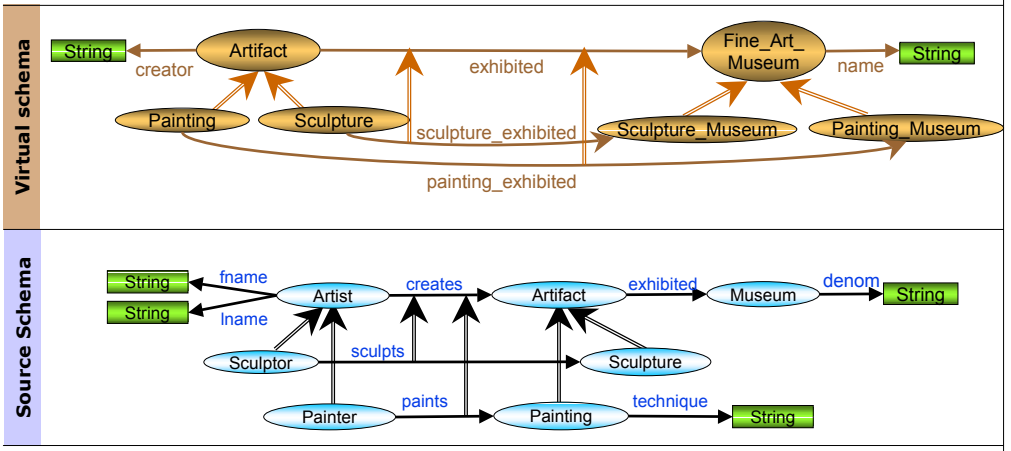
The RDF View Language: RVL

- Declarative view definition language for virtual RDF description bases and schemas
  - relies on the RQL typed data model
  - follows also a functional approach (object construction operators)
  - ensures logical data independence
    - view specifications are independent from those of the source schemas and bases,
    - the semantics of existing virtual schemas is not be altered by the definition of new ones
  - supports object-preserving and object-generating views
  - provides heavy data restructuring facilities
  - allows users to query and create views using both source and virtual schemas

The RVL Approach



An RVL virtual RDF/S schema and base



An RVL virtual RDF/S schema and base

- CREATE NAMESPACE myview=&http://www.ics.forth.gr/mycult.rdf#
- VIEW Class("Fine_Art_Museum"), Class("Painting_Museum"),
  Class("Sculpture_Museum"), Class("Artifact"),
  Class("Painting"), Class("Sculpture")
- VIEW Property("name", Fine_Art_Museum, xsd:string),
  Property("title", Artifact, xsd:string),
  Property("creator", Artifact, xsd:string),
  Property("exhibited", Artifact, Fine_Art_Museum),
  Property("sculpture_exhibited",Sculpture, Sculpture_Museum),
  Property("painting_exhibited", Painting, Painting_Museum)
- VIEW Fine_Art_Museum<Sculpture_Museum>,
  Fine_Art_Museum<Painting_Museum>,
  Artifact<Painting>, Artifact<Sculpture>
  exhibited<sculpture_exhibited>,
  exhibited<painting_exhibited>

An RVL virtual RDF/S schema and base

- VIEW Painting(X), painting_exhibited(X,Y), Painting_Museum(Y),
  name(Y,W), title(X,K), creator(X,Z)
  FROM {Z}n1:creates{X; n1:Painting}.n1:exhibited{Y}.n1:denom{W},
  {X}n1:title{K}
  USING NAMESPACE n1=&http://www.culture.mus/cult.rdf#

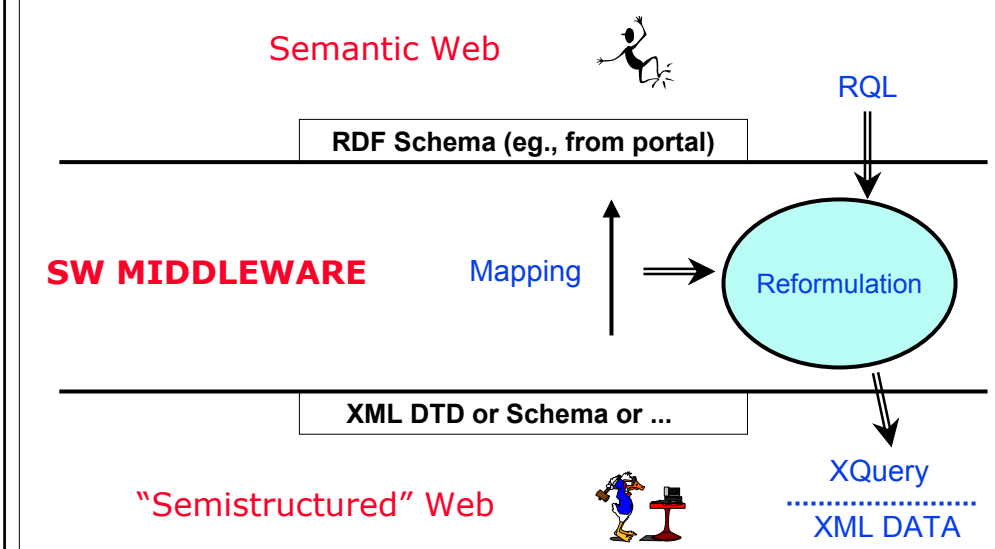
- VIEW Sculpture(X), sculpture_exhibited(X,Y), Sculpture_Museum(Y),
  name(Y,W), title(X,K), creator(X,Z)
  FROM {Z}n1:creates{X; n1:Sculpture}.n1:exhibited{Y}.n1:denom{W},
  {X}n1:title{K}
  USING NAMESPACE n1=&http://www.culture.mus/cult.rdf#

Semantic Web Integration Middleware (SWIM)

- The bulk of existing data is not yet in RDF/S (or any other form suitable for the SW)
  - Data physically stored in relational DBs and/or published as virtual XML
- SW applications require viewing data as virtual RDF
  - valid instances of domain or application-specific RDF/S schemas
- Need the ability to manipulate data with high-level query or view languages (RQL, RVL)
- How to do it?
  - republish XML as RDF
  - publish relational data as RDF
  - do both



Republish XML as RDF

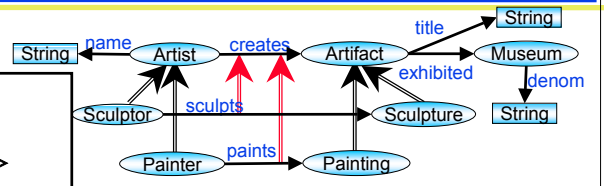


Semantic Web Middleware

- Practical concerns:
  - XML publishing systems often provide an XML query interface
    - SW middleware can function as an alternative to the XML publishing systems;
    - SW middleware provides direct access to underlying DBMSs
  - SW middleware may also be required to integrate DBMS data with data in native XML storage
- SW middleware tasks:
  - Specify mappings: XML→ RDF, RDB → RDF
  - Verify conformance to the semantics of employed schemas
  - Reformulate queries (i.e., compose RQL queries with mappings to produce XML or RDB queries)
  - Provide further abstractions of RDF data/schemas (RVL views)
  - Compose queries with views

Motivating Example

```
<ArtDB>
<Sculptor name="Rodin">
   <sculpts>
      <Sculpture title="thinker"/>
   </sculpts>
</Sculptor>
<Painter name="Rodin">
   <paints>
      <Painting title="crucifixion"/>
   </paints>
</Painter>
<Painter name="Picasso">
   <paints>
      <Painting title="guernica">
         <exhibited>
           Reina Sofia
         </exhibited>
      </Painting>
   </paints>
</Painter>
</ArtDB>
```



Artifacts			
title(key)	Artist	exhibited	kind
guernica	Picasso	ReinaSofia	Painting
crucifixion	Rodin	NULL	Painting
thinker	Rodin	NULL	Sculpture

Introducing a SW Middleware Server

- By designing (or importing) a (virtual) RDF/S cultural schema, we can answer queries using RQL
  - *"List the names of all artists that have created artifacts exhibited at the Reina Sofia Museum"*

```
SELECT   Z
FROM     {X} creates.exhibited.denom {V}, {X} name {Z}
WHERE    V = "Reina Sofia Museum"
```

- Actual data can only be queried using an XML language (e.g., XQuery) or SQL
- The RQL query needs to be reformulated into an XML query
- Reformulation cannot be ad hoc; needs to be driven by a formal description of the relationship between XML and RDF data
- Need a formal basis for expressing such mappings

Mappings: Background

- From relational database theory
  - query containment, query + view composition, query rewriting using views are solvable for a fairly large class of queries in the presence of certain classes of constraints
    - embedded implicational dependencies
- A robust formalism to rely on: conjunctive queries and views (non-recursive Datalog)
- A formal data model for RDF/S
  - Validity constraints
- High-level query and view languages for RDF/S adhering to the formal model

RQL Translation

```
SELECT   Z
FROM     {X} creates.exhibited.denom {V}, {X} name {Z}
WHERE    V = "Reina Sofia Museum"
```

"Paths" provide shorthand notation for sequences of patterns:

```
SELECT   Z
FROM     {X} creates {Y}, {Y} exhibited {U}, {U} denom {V}, {X} name {Z}
WHERE    V = "Reina Sofia Museum"
```

In the internal model:

```
ans(Z) :-- P_SUB(P1, name), P_EXT(X, P1, Z),
           P_SUB(P2, creates), P_EXT(X, P2, Y),
           P_SUB(P3, exhibited), P_EXT(Y, P3, U),
           P_SUB(P4, denom), P_EXT(U, P4, "Reina Sofia Museum")
```

A conjunctive query!

All Together: An XPath/Datalog Program

```
ans(Z)  :--  P_SUB(P1, name), P_EXT(X, P1, Z),
             P_SUB(P2, creates), P_EXT(X, P2, Y), …
```

from query

```
…
P_SUB(paints, creates)   :--
P_SUB(sculpts, creates)  :--
…
```

from schema

```
P_EXT(X, paints, Y)  :--  //Painter (X),  .//Painting (X, Y)
…
P_EXT(X, name, X)  :--  //Sculptor (X),  ./@name(X, Y)
P_EXT(X, name, Y)  :--  //Painter (X),  ./@name(X, Y)
…
```

from mapping

A reformulation, of sorts, but unacceptably inefficient!

Improving the Reformulation (1)

After "partial evaluation" using the schema facts:

ans(Z) :-- P_EXT(X, name, Z), P_EXT(X, paints, Y), …

ans(z) :-- P_EXT(X, name, Z), P_EXT(X, sculpts, Y), …

…

P_EXT(X, paints, Y) :-- //Painter (X), .//Painting (X, Y)

P_EXT(X, sculpts, Y) :-- //Sculptor (X), .//Sculpture (X, Y)

…

P_EXT(X, name, Y) :-- //Sculptor (X), ./@name(X, Y)

P_EXT(X, name, Y) :-- //Painter (X), ./@name(X, Y)

…

Improving the Reformulation (2)

After eliminating the intermediate predicates:

ans(Z) :-- //Painter (X), ./@name(X, Z) ,
//Painter (X), .//Painting (X, Y), …

ans(z) :-- //Sculptor (X), ./@name(X, Z),
//Painter (X), .//Painting (X, Y), …

unsatisfiable!

…

ans(z) :-- //Painter (X), ./@name (X, Z) ,
//Sculptor (X), .//Sculpture (X, Y), …

unsatisfiable!

ans(z) :-- //Sculptor (X), ./@name(X, Z),
//Sculptor (X), .//Sculpture (X, Y), …

Requires some reasoning about XPath that can be done with FO tools

…

Reformulation, Finally (1)

```
ans(Z)  :-  //Painter (X), .//Painting (X, Y),
            ./exhibited/text() (Y,"Reina Sofia Museum"),
            ./@name (X, Z)

ans(Z)  :-  //Sculptor(X), .//Sculpture (X, Y),
            ./exhibited/text() (y,"Reina Sofia Museum"),
            ./@name (x, z)
```

- XPath

```
doc("")//Painter[Painting/exhibited/text()="Reina Sofia
  Museum"]/@name

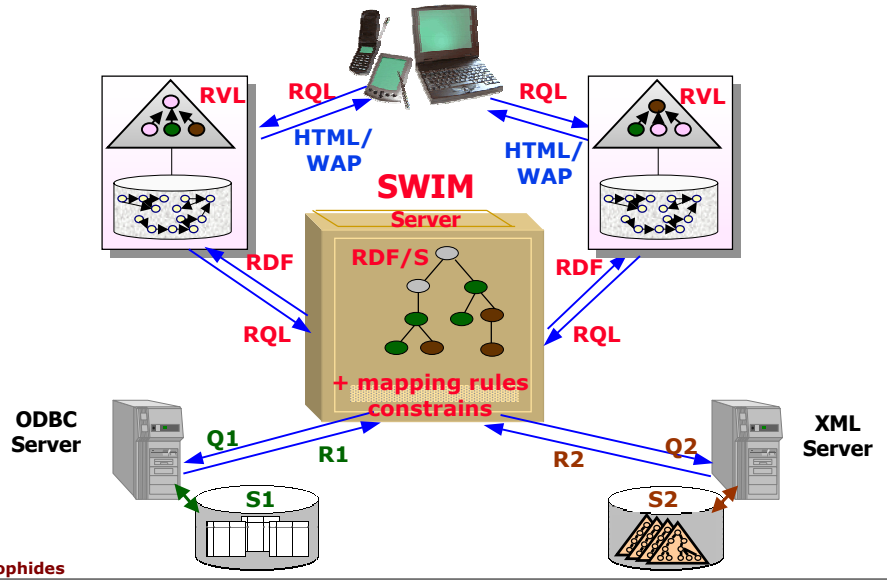
doc("")//Sculptor[Sculpture/exhibited/text()="Reina
  Sofia Museum"]/@name
```


Reformulation, Finally (2)

- XQuery

```
<result>
{
 for $x in document("")//Painter
 where $x/Painting/exhibited/text()="Reina Sofia Museum"
 return $x/@name
}
</result>
<result>
{
 for $x in document("")//Sculptor
 where $x/Sculpture/exhibited/text()="Reina Sofia Museum"
 return $x/@name
}
</result>
```

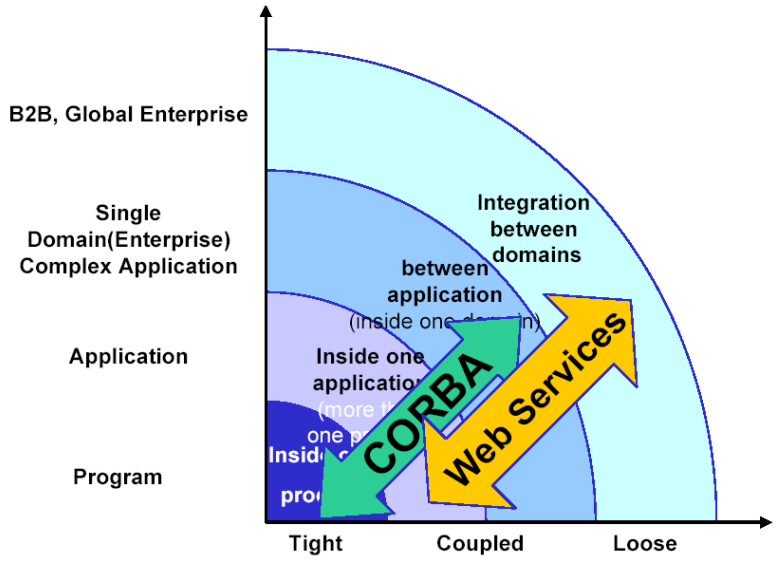
Let's go SWIM-ming



SWIM Flexibility

- Same framework can be used for publishing relational data directly as RDF
- Same framework can be used for composing RQL with RVL views
- Same framework can be used for heterogeneous integration (mediation)
- Minimization (eliminating redundancies) is essential
- Many desirable minimizations only hold under constraints
- For minimization under constraints, use the Chase&Backchase algorithm

Middleware Evolution & Interoperability





Thanks!
Questions?