3 XML Data Management and Bioinformatics applications

using material from
- Alan Robinson, (recommended ! http://industry.ebi.ac.uk/~alan/XMLWorkshop/)
- Silverschatz and M. Sapossnek .

3.6 Bioinformatic DTDs (from Robinson, 2000)

- Some DTD's have been proposed publicly as XML formats for biological data

– GAME	Drosophila Genome Project/Celera
– BIOML	ProteoMetrics
– XML Schema	EMBL sequence records
– BSML	VisualGenomics
– CML	OMF
– DAS	CSHL
– BSA	OMG-LSR
– BLAST	Various
– INSDSeq	Sequences

Small part of Game DTD

```
<!ELEMENT game ( seq+, map_position, annotation*,
computational_analysis* ) >
<!ATTLIST game version NMTOKEN #REQUIRED >

<!ELEMENT seq ( name, description?, residues?, dbxref*, organism?
   ) >
<!ATTLIST seq version CDATA #REQUIRED >

<!ATTLIST seq type (AA | DNA | RNA) #IMPLIED >
<!ATTLIST seq length NMTOKEN #REQUIRED >
<!ATTLIST seq md5checksum CDATA #IMPLIED >
<!--  ** Flag for discussion - unique id?  Same as name?  -->
<!ATTLIST seq id IDREF #REQUIRED >

<!ELEMENT name ( #PCDATA ) >

<!ELEMENT description ( #PCDATA ) >

<!ELEMENT residues ( #PCDATA ) >
```

See local copy of DTD

Bioinformatic Sequence Markup Language

- BSML
  - "…aims to provide a single document interface to integrate all project information, complete with protocols for network data retrieval."
  - "… is an extensible language specification and container for bioinformatic data."
  - Two kinds of information:
    - The Definitions section encodes the bioinformatic data (sequences, sets, sequence features, analytical outputs, relationships, annotations)
    - The optional Display section encodes information for graphic representation of the bioinformatic data."

Cf. A Robinson

So what?

- Current DTD's biased towards sequence data
  - gene expression DTD GML21 exists, proprietary
  - BLAST output DTD
- Probably more XML DTD's will be produced as people "roll their own": BAD
- Without co-ordination - likely to be all subtly different and incompatible in syntax and semantics
- Needed: common namespace,
  xml schema for different applications
  (sequence description, expression, …)
- Next step: use XML DB instead of RDB ?!

3.7 Why Query Languages for XML?

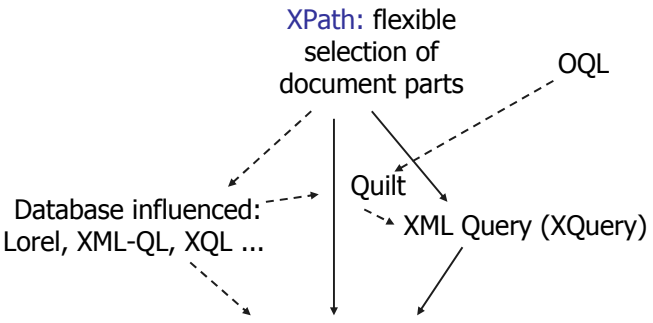
- Data Extraction and Filtering
  - Transformation with XSLT requires pattern sophisticated extraction language -> XPath
  - Extraction from large "documents"
    "Blast output which produced more than 3 hits
- Construction of XML documents
  "Output an XML document of all hits..."
- Data conversion
  - Restructuring of documents , e.g. for data exchange
- Data integration
  - Combine different documents in a new one

Why not SQL / OQL ?

- Main Argument: Different data models
  - 'Tree' not 'table' or 'Object'
- Optional parts in XML document

  "price < 100 if <price> element exists "
- Text retrieval
  - Not addressed in SQL / OQL
  - ... except user defined type "text"
- Construction: how to represent SQL query results as XML?
  - Only addressed in new SQL extension SQLX (or SQL XML)

Language development



XPath 2.0 ~ XQuery 1.0 (W3C Standard)

http://www.w3.org/XML/Query .html

Influenced by

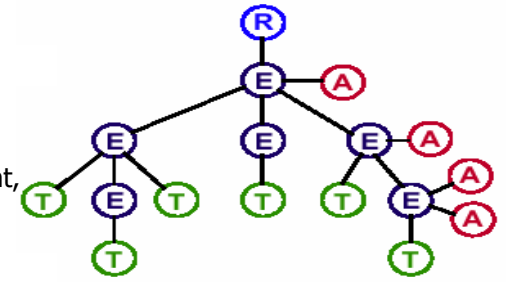
sublanguage / offspring

XPATH

- Language for navigating in XML-Documents
- W3C-recommendation 11/99:
  http://www.w3.org/TR/xpath
- Used for selecting parts of a document in a declarative way

Basic document model:

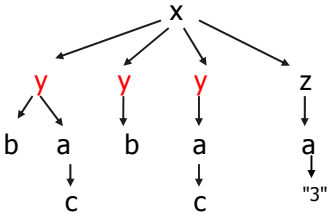
Tree with node types
root, element, attribute,
text, namespace, comment,
processing instruction



XPath expression: mapping from a node (the context node) to a set of nodes

XPath expressions

- Expression e specifies paths in an XML document
- Value of e : Set of nodes which match the path specified by the expression, boolean, number or string



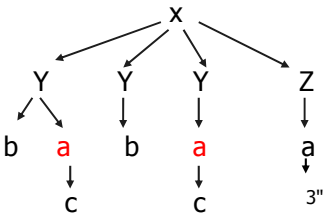
e = /x/y

- Context defines where to start (above root)
- Application (e.g. XSLT) defines context by position in tree
- Absolute location path* starts at root /, relative from context node

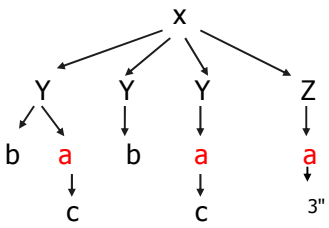
*The only type of expression we discuss here

XPath expressions

Examples



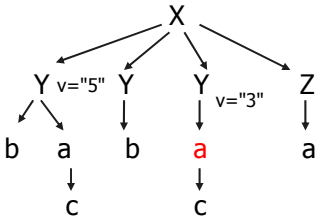
e = /x/y/a



e = //a

XPath: syntax (long)

- Basic syntax of a location step:
  axis::nodetest[predicate]
- Axis: direction from context node (e.g. child, following, parent, ancestor, decendant..)
- Nodetest: basically the <element> to select,
- Predicate: predicate on elements and attributes which filters nodes on the path



e = root::X/child::Y[@v ="3"]/child::a

XPath: abbreviated syntax

root:: / /x

Parent:: .. //c ../../@v

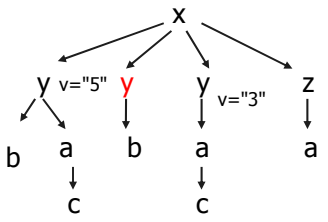
Self:: . ./a

Attribute:: @ attribute of self

// decendents of self, any depth //

* all elements /*/a

[n] n-th element /x/y[2]



built-in functions:

e.g.
count() = number of nodes in context node set;
true(), false(), ...

Examples

Find BBB Elements which have name attribute

```
<AAA>
      <BBB id = "b1"/>
      <BBB id = "b2"/>
      <BBB name = "bbb"/>
      <BBB/>
   </AAA>
```

//BBB[@name]

BBB elements without attributes

```
<AAA>
      <BBB id = "b1"/>
      <BBB id = "b2"/>
      <BBB name = "bbb"/>
      <BBB/>
   </AAA>
```

//BBB[not (@*)]

Try: http://www.zvon.org/xxl/XPathTutorial/General/examples.html

Examples

BBB elements and EEE elements which are children of root AAA

```
<AAA>
    <BBB/>
    <CCC/>
    <DDD>
        <CCC>
            <BBB/>
        </CCC>
    </DDD>
    <EEE/>
</AAA>
```

/AAA/EEE | // BBB

```
<AAA>
    <BBB>
        <DDD>
            <CCC>
                <DDD/>
                <EEE/>
            </CCC>
        </DDD>
    </BBB>
    <CCC>
        <DDD>
            <EEE>
                <DDD>
                    <FFF/>
                </DDD>
            </EEE>
        </DDD>
    </CCC>
</AAA>
```

All /AAA/BBB descendants

/AAA/BBB/descendant::*

XPath examples

```
<addressBook>
    <address>
        <firstName>John</firstName>
        <surname>Smith</surname>
        <email>smithj@world.org</email>
        <tel type = "work">234-123-222</tel>
    </address>
    <address>
        <firstName>Alice</firstName>
        <surname>Brown</surname>
        <email>Alice.Brown@europe.com</email>
        <tel type = "home">22-33-444</tel>
        <tel type = "work">11-43-222</tel>
    </address>
    <address>
        <firstName>George</firstName>
        <surname>White</surname>
        <email>gw@rock.com</email>
    </address>
</addressBook>
```

Queries

q = /surname

result: -

q = // surname

result

q = // address [firstName="Alice"] /email

result

XPath examples

/address/tel[2]

```
<addressBook>
      <address ID=1 roomMate="3">
          <firstName >John</firstName>
          <surname>Smith</surname>
          <email>smithj@world.org</email>
          <tel type = "work">234-123-222</tel>
      </address>
      <address ID = 2 >
          <firstName>Alice</firstName>
          <surname>Brown</surname>
          <email>Alice.Brown@europe.com</email>
          <tel type = "home">22-33-444</tel>
          <tel type = "work">11-43-222</tel>
      </address>
      <address ID = 3  roomMate = "1">
          <firstName>George</firstName>
          <surname>White</surname>
          <email>gw@rock.com</email>
    </address>
  </addressBook>
```


XPath examples

//address[tel="22-33-444"]/tel/@type

```
<addressBook>
    <address>
        <firstName>John</firstName>
        <surname>Smith</surname>
        <email>smithj@world.org</email>
        <tel type = "work">234-123-222</tel>
    </address>
    <address>
        <firstName>Alice</firstName>
        <surname>Brown</surname>
        <email>Alice.Brown@europe.com</email>
        <tel type = "home">22-33-444</tel>
        <tel type = "work">11-43-222</tel>
    </address>
    <address>
        <firstName>George</firstName>
        <surname>White</surname>
        <email>gw@rock.com</email>
    </address>
</addressBook>
```

XPath examples

Find type ("home" or "work") for tel# 22-33-444

//tel[self::tel="22-33-444"]/@type

```
<addressBook>
    <address>
        <firstName>John</firstName>
        <surname>Smith</surname>
        <email>smithj@world.org</email>
        <tel type = "work">234-123-222</tel>
    </address>
    <address>
        <firstName>Alice</firstName>
        <surname>Brown</surname>
        <email>Alice.Brown@europe.com</email>
        <tel type = "home">22-33-444</tel>
        <tel type = "work">11-43-222</tel>
    </address>
    <address>
        ....
</addressBook>
```


XQuery = XPath 2.0

Basic structure of expressions

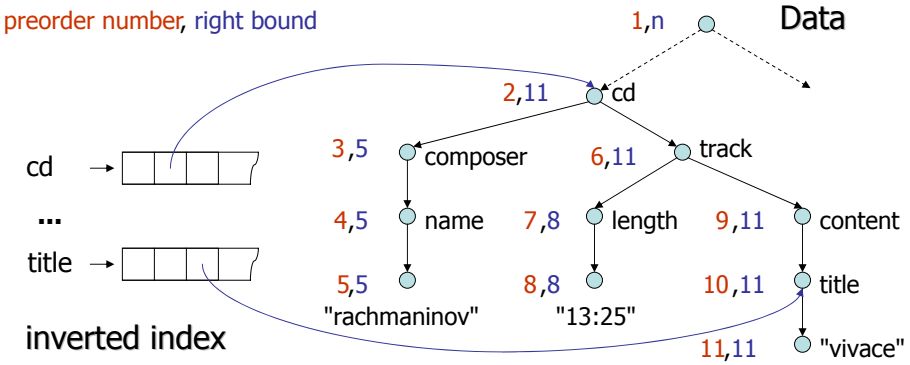
- Pattern clause
  - Matching of substructures (-> XPath)
  - Binding of variables (Syntax: $x)
- Filter clause
  - Predicate on variables , comparison with constants etc
- Construction clause
  - Construct result as an XML document

```
for  $a in  document("addressbook.xml") //address
let  $t := $a/tel
where  count($t) > 1
return  <manyPhones>
            {$a/firstName, $a/surname, <num> count($t) </num>}
        </manyPhones>
```

FLWR ("Flower")-expression

Implementation issues

Encoding trees goal: efficient tree manipulation e.g.: "Is x ancestor of y?"



$$ancestor(x, y) = pre(x) < pre(y) \wedge rb(x) \geq pre(y)$$

$$ancestor(cd, title) = 2 < 10 \wedge 11 \geq 10$$

Tools for managing XML documents ⇨ DBS ?!

3.8 XML Data Management

- Approaches
  - Use Relational DBMS and map documents to relations
    - Generic mapping
    - DTD oriented mapping
    - Use SQL for queries, transform result set into XML document
  - Use XML (native or enhanced) data management system
    - Examples: Tamino
    - XLM enhancement of most RDBS
    - Support XPath and XQuery (some)
- Mapping documents to relations
  - Tree model of XML documents does not fit to table model of RDB

The mapping problem

- Variability of structures makes uniform mapping difficult

```
<text> .....This is a very long text.....
</text>
```

- "data centric"

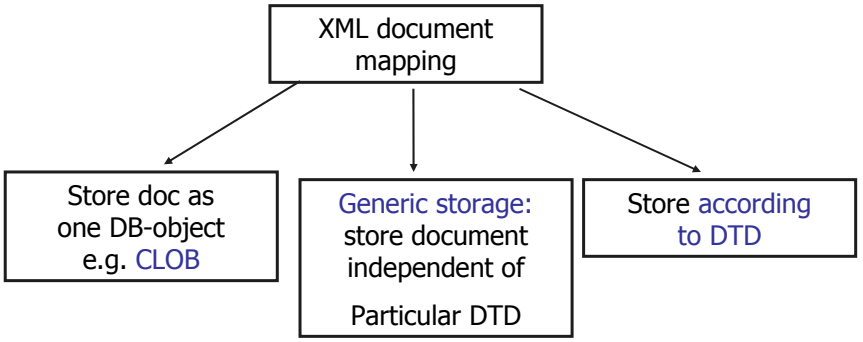
```
<orders>
  <order ID= 4330>
   <itemList>
      <item ID>
        <price>
         <quantity>
          <description> …A not so long txt
          </description>
      ...
  </orders>
```


Why mapping to a DB?

- Non functional characteristics of DBS
  - Fault tolerance
  - Concurrent access
  - Stability
- Searching and transforming data (to some extent) are at the heart of every DBS
- Performance

- ….depends on types of operations and data
  - Searching in large text strings?
  - Tree traverse ?
  - Joins?

Classification of mappings



References:
M. Klette, H. Meyer: Speicherung von XML-Dokumenten – eine Klassifikation, Datenbank-Spektrum 5/2003
D. Florescu, D. Kossmann: A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Inria TR, 1999
A. Chaudri et al (eds.): XML Data Management, Addison-Wesley 2003

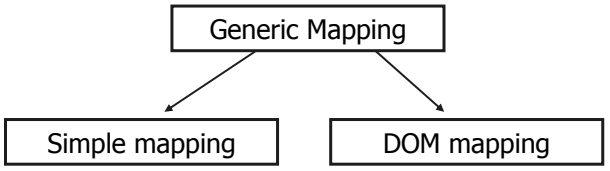
Example document

```
<Movie Title="American Beauty" RunningTime="121"
  imdbCode="0169547" Rating="R" >
  <Director>
            <First> Sam  </First>
            <Last> Mendes </Last>
  <Award From="Oscar" Category="Best Director">  </Award>
  </Director>
 <PlotSummary> Lester Burnham is in a mid mid-life crisis…
</PlotSummary>
<Cast>
    <Actor Role="Lester Burnham">
       <First> Kevin Kevin</First>
       <Last> Spacey Spacey</Last>
       <Award From="Oscar" Category="Best Actor" >   </Award>
      <Award From="BAFTA" Category="Best Actor">  </Award>
    </Actor>
   <Actress Role="Carolyn Burnham">
      <First> Annette Annette</First>
     <Last> Bening Bening</Last>
     <Award From="BAFTA" Category="Best Actress"> </Award>
    </Actress>
</Cast>
<Award From="Oscar" Category="Best Film">     </Award>
<Award From="BAFTA" Category="Best Film">     </Award>
</Movie>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Actor (First,Last,Award+)>
<!ATTLIST Actor Role CDATA #REQUIRED>
<!ELEMENT Actress (First,Last,Award)>
<!ATTLIST Actress Role CDATA #REQUIRED>
<!ELEMENT Award EMPTY>
  <!ATTLIST Award
  Category CDATA #REQUIRED
  From NMTOKEN #REQUIRED
  >
<!ELEMENT Cast (Actor,Actress)>
<!ELEMENT Director (First,Last,Award)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT Movie (Director,PlotSummary,Cast,Award+)>
  <!ATTLIST Movie
  Rating NMTOKEN #REQUIRED
  RunningTime NMTOKEN #REQUIRED
  Title CDATA #REQUIRED
  imdbCode NMTOKEN #REQUIRED
  >
<!ELEMENT PlotSummary (#PCDATA)>
```

DTD for Movie DB

Generic Mapping



Simple relational:

element- attribute- and edges relations
preserve order of elements

DOM oriented:

map class structure of DOM model
onto relations or classes (oo or OR DBS)

Simple

- Element table
  - An element row for each element in the document
    - A generated document id
    - A generated element id
    - Predecessor of element node
    - Order of childs
    - Value, if element has a value
- Attribute table
  - Row for each (attribute , value) pair in document
    - Element id of this attribute
    - Attribute name
    - Attribute value
    - Order (if needed)

DocId	element	ID	predec	order	value
m002	movie	200	-	1	-
m002	director	201	200	1	
m002	first	202	201	1	Sam
m002	last	203	201	2	Mendes
m002	award	204	201	1	
m002	cast	205	200	1	
m002	actor	206	205	1	
m002	first	207	206	1	Kevin
m002	last	208	206	2	Spacey
…					

elemI D	DocId	attribute	val	order
200	m002	title	American Beauty	1
200	m002	Running time	124	2
204	m002	from	Oscar	1
…	…			

Simple generic mapping

- Advantage
  - Independent of document structure
  - Simple database structure
- Disadvantage
  - MANY joins to reconstruct the document
  - Attribute types
    - No problem if only "string" type represented
    - Cast to other types: inlining with one value column per type or value table for each type

	element	id	…		valString	valInt
m002	last	203	201	2	Mendes	null
m002	age	204	201	1	null	53

….

  - At least n recursive joins when traversing on an n-element path from root down the tree

DOM-oriented generic mapping

- Node model of DOM mapped to relations
  - Node table
  - Node : element | attribute
  - Value table

TreeStructTable

node_id	node_type	doc_id	parent	prev_sibling	next_sib-ling

NodeValueTab

node_id	tag	value

Alternative representation of order

- Coding of node order with parent-successor relation
  - preorder

    does not represent parent child relationship
  - (preorder# , bound) where
    bound = max {preorder#(y) | y successor of this}
  - y successor of x ⇔ pre(x) < pre (y) and bound(x) >= pre(y)

    very easy successor test

XML document as one DB object

- BLOB or CLOB

=> no reconstruction effort,
no query support
useful for document centric objects ?

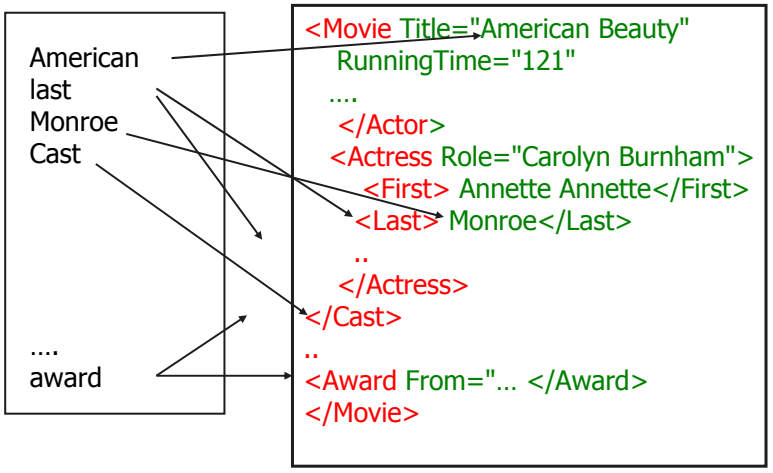
- (User) defined type "text"

=> no reconstruction effort,
indexing support queries like " find all occurences of 'actor' " (in a DB storing movies) ,
no separation of tags and data (!),

More sophisticated queries like:
"find occurences of 'Monroe' where 'actress' occurs in a window of n words before/after"
locates "actress" elements with (partial) value 'Monroe'

XML document as one DB object

- Indexed 'text ' attribute



XML document as one DB object

- Querying
  - only boolean queries on keywords
  - no difference between structure and content
  - only useful in document centric applications

- Search predicates (Oracle)

```
CREATE Table MovieTab AS (id INTEGER, txt CLOB)
SELECT id, txt, SCORE(1)
   From MovieTab
   WHERE CONTAINS (txt, 'Monroe',1)>0 ;
```

Place holder for relevance

  - Score: relevance measure
  - Value of CONTAINS: 0..100

XML document as one DB object

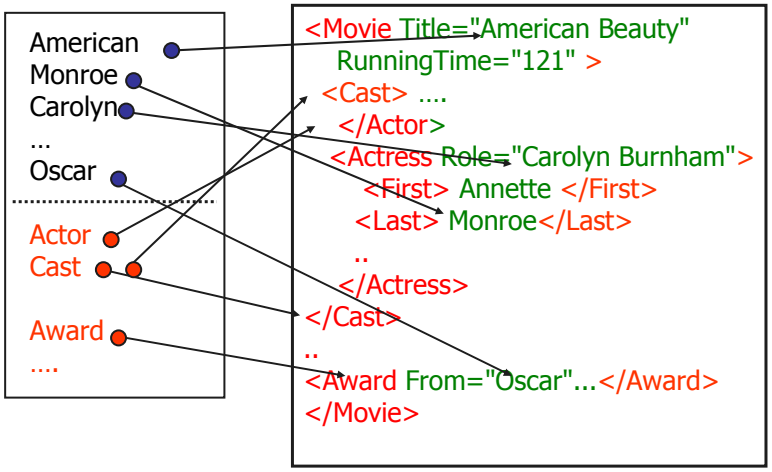
- Improvement "XML type"
  - Difference between markup and strings
  - Allows to search between
    <tag> and </tag>

```
CREATE Table MovieTab AS (id INTEGER, doc XMLType)
SELECT id, doc, SCORE(1)
   From MovieTab
   WHERE CONTAINS (txt, 'Monroe WITHIN Last',1)>0 ;

SELECT id
   FROM MovieTab
   WHERE CONTAINS (txt, 'Monroe INPATH(//Actress/Last'..)..;

…. CONTAINS (txt, 'HASPATH(//Actress/Last="Monroe" '..)..;
   // exact match required
```

- For performance reasons separation of structure index and
  data index

Sophisticated indexing

- Relational attribute of XML type



DTD-Oriented mapping

- Automatic approach
  - Map each element to a relation
    Object relational / object oriented: to a class
  - Attribute of element -> attribute of a relation / class
  - Tree structure by some ordering scheme
  - Easier to map into an object oriented / object relational data model
- User defined Mapping approach
  - Define a mapping between DTD and database schema
  - More flexible, more effort

See also R. Bourret: mapping DTDs to Databases

DTD-oriented approach

- Automatic:
  - Simple elements (not nested) -> attributes
  - Nested elements -> object types (classes)

```
DTD                              Classes
<!ELEMENT A (B, C)>              class A {
<!ELEMENT B (#PCDATA)>                  String b;
<!ATTLIST A                             C c;
F CDATA #REQUIRED>                      String f; }
 <!ELEMENT C (D, E)>             class C {
<!ELEMENT D (#PCDATA)>                  String d;
<!ELEMENT E (#PCDATA)>                  String e; }
```

  - Sibling order is lost
    Could be retained by artificial order attributes

DTD-oriented approach

- Mapping guidelines

	XML	Database Def
element	root	Relation
	simple	attribute
	sequence	attributes
	alternative |	attributes
	elem with ?	attribute, NULL
	elem +,*	SET, LIST
	complex	Class (Object type)
attribute	XML attr	Attribute of Rel
	#IMPLIED	^
	#REQUIRED	NOT NULL
	default	default

DTD-oriented approach

- Some issues
  - Data types
    DTD: manual adaption of schema necessary
    XML schema: adapt data types to database types (automatically)
  - Mixed content

    <desc> The values are
    <simple> like 'X'</simple> or
    <compound> like '(x,y)' </compound>
    </descr>
    - Map onto different attributes and define an order

    Tab (order, desc, simple, compound)
  - Order

DTD-oriented approach: order

```
<!ELEMENT A (#PCDATA | B | C)*>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

```
class A {
        String[] pcdata;
        int[] pcdataOrder;
        String[] b;
        int[] bOrder;
        String[] c;
        int[] cOrder; }
```

- Not supported by most systems
- + , * can also be mapped to SET or LIST – if supported by
  data model

DTD-oriented approach

- Advantages
  - Object oriented / Object relational allow fine-granular, "natural" mapping of object structure
  - DTD is needed (of course) – compared to generic mapping
  - SQL and XQuery / XPath as query languages as opposed to 'XMLtype' : basically operations on text
- Disadvantages
  - No document reconstruction
  - possible in principle, but expensive

User defined mapping

- Define your own mapping

  called Data access Definition in DB2

```
<ClassMap>
  <ElementType name = "Movies"/>
  <ToClassTable>
    <Table Name="MovieTable"/>
  </ToClassTable>
  <PropertyMap>
    <Attribute Name = "Title"/>
      <ToColumn>
        <Column Name = "Titel"/>
  ...
```

- Very flexible
- Makes sense if XML docs are mapped to an existing DB schema

3.9 Information Integration

- Goal
  - access to all kinds of data of an application domain independent of local, data format, query language
  - Online access to sources ("Portals") instead of locally copied data
  - "Semantic linking" of databases / database queries
- Architectures
  - Language and infrastructure for linking DBs
  - Integrated architecture using wrappers and "mediator"

Information integration: SRS

- Sequence Retrieval Service (SRS)
  - Link operators ">" and "<" allow to link databases

    Examples:

    SWISS-PROT > PDB

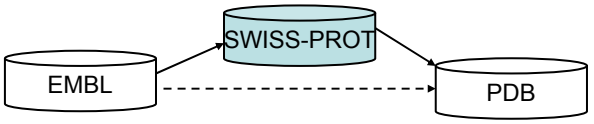
    finds all protein structures in PDB which are referenced by or linked to entries in SWISS-PROT

    [swissport-def:kinase] > PDB

    1. Kinase sequences in SWISS-PROT
    2. Link to PDB entries
       ⇨ Known tertiary structures of kinases

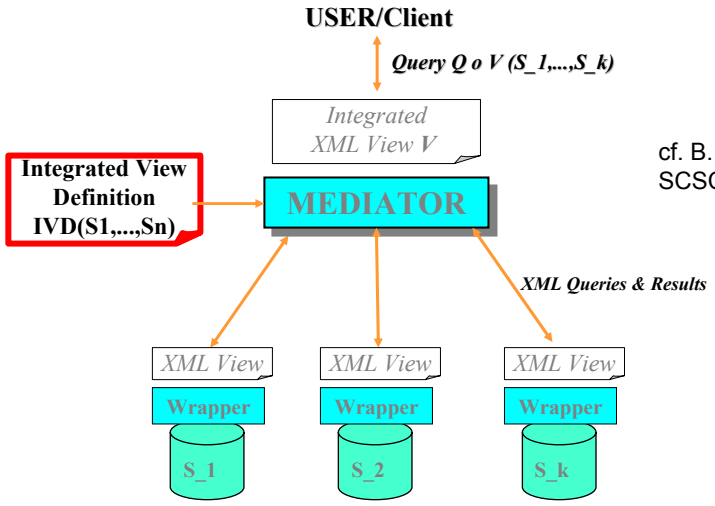
    Basis: database of metadata

  - Indirect Links



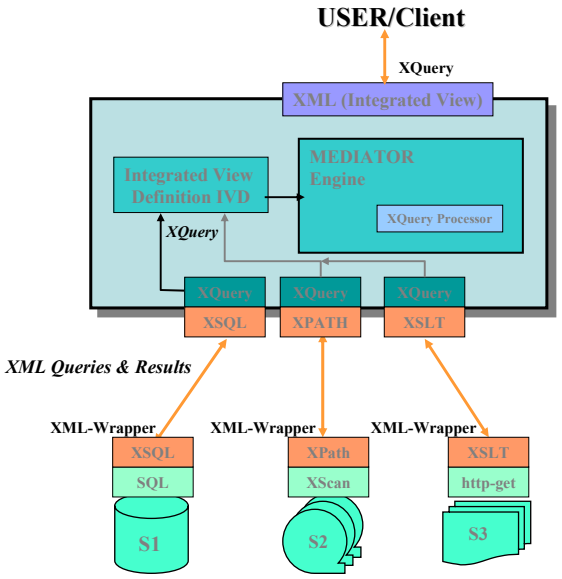
SRS finds "the best intermediate DB" to support a link

XML-Based Mediator Architecture



cf. B. Ludäscher, SCSC, UCSD

A Concrete XML-Based Mediator System



Syntactical view definition not sufficient.

Namespaces? Ontologies?

⇨ semantic integration

cf. B. Ludäscher, SCSC, UCDS

3.10 Ontologies

Def: **ontology** (from the Greek *ov* = *being* and *λόγος* = *word/speech*) is the most fundamental branch of metaphysics. It studies being or existence as well as the basic categories thereof—trying to find out what entities and what types of entities exist.

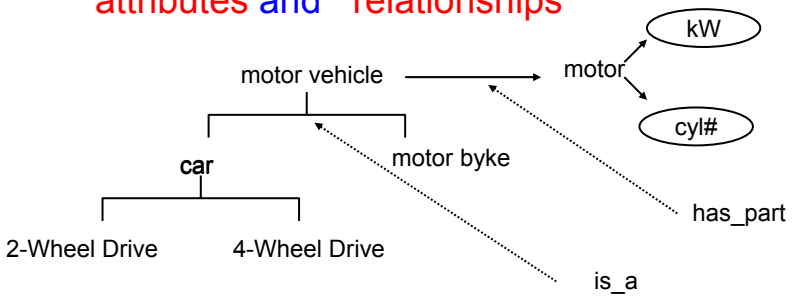
Def-2: In information science, **an ontology** is the product of an attempt to formulate an exhaustive and rigorous conceptual schema about a domain. An ontology is typically a hierarchical data structure containing all the relevant entities and their relationships and rules within that domain (e.g., a **domain ontology**).

cited from Wiki encyclopedia

Ontologies

Ontology:

A formal description of concepts using attributes and relationships



Directed Graph with node and edge semantics

Ontologies

Languages for ontologies

- purely descriptive
  example: RDF triples (source, predicate, destination)
  (person_url, is_author, doc_url)
- Logic language

example: CYC

- individuals	#$Berlin
- sets	#$cities
- predicates	#$is_elem #$berlin #cities
- functions	#hasPopulation

(#$relationAllExists #$biologicalMother #$ChordataPhylum #$FemaleAnimal)

for every instance of the collection #$ChordataPhylum,
there exists an instance of #$FemaleAnimal which is its mother (described by the predicate #$biologicalMother).

Ontologies in Bioinformatics

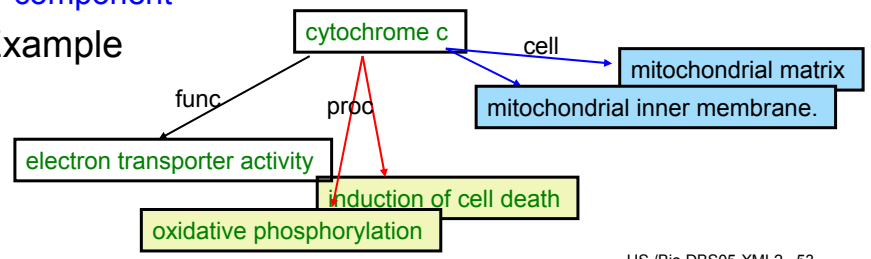
Goal:

consistent descriptions of gene products in different databases

Description of an entity by

molecular function, biological process and cellular component

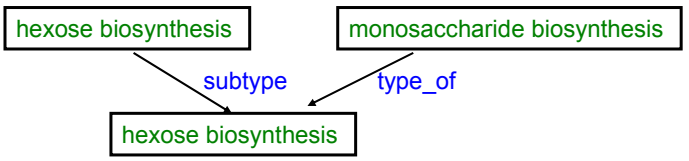
Example



expl from Gene Ontology,

Ontologies in Bioinformatics

Example: Generic relationships



Which relationships and other primitives for building an ontologie?

Define set of relations, attributes, processes, functions.....in XML

Onotologies in Bioinformatics

XML DTD for Gene Ontology exchange format

```
<!ELEMENT term (id|name|namespace|def?|is_a*|alt_id*|subset*|comment?|is_anonymous?|
is_obsolete?|Is_root?|xref_analog*|xref_unknown*|synonym*|relationship*|intersection_of*
|union_of*| lexical_category?)+>

<!-- TERM ELEMENTS -->
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT namespace (#PCDATA)>
  <!ELEMENT def (defstr|dbxref*)+>
<!-- DBXREF ELEMENTS -->
  <!ELEMENT xref_analog (acc|dbname|name)+>
  <!ELEMENT xref_unknown (acc|dbname|name)+>
  <!ELEMENT is_a (#PCDATA)>
  <!ELEMENT relationship (type|to)+>
    <!ELEMENT type (#PCDATA)>
    <!ELEMENT to (#PCDATA)>
  <!ELEMENT alt_id (#PCDATA)>
  <!ELEMENT subset (#PCDATA)>
  <!ELEMENT comment (#PCDATA)>    .....
```



```
<term>
   <id>GO:0001868</id>
   <name>regulation of complement activation, lectin pathway</name>
   <namespace>biological_process</namespace>
   <def>
    <defstr>Any process that modulates the frequency, rate or extent of the
    lectin pathway of complement activation.</defstr>
    <dbxref>
      <acc>0781735149</acc>
      <dbname>ISBN</dbname>
      <name>Fundamental Immunology</name>
    </dbxref>
    <dbxref>
      <acc>add</acc>
      <dbname>MGI</dbname>
      <name></name>
    </dbxref>
   </def>
   <is_a>GO:0030449</is_a>
   <relationship>
     <type>part_of</type>
     <to>GO:0001867</to>
   </relationship>
</term>
```

A term definition from the Gene Ontology

... used by human readers ("what is the definition of...") and by programs:

- interoperability
- integration
- query processing

Summary

- Strength and weaknesses of XML
  - flexible data structuring meta language for non-regular data
  - standardized
  - human and machine readable
  - Data management of XML documents advanced
  - XML took off already
  - ideal for syntactic integration and of course: data exchange
- Weaknesses
  - no inheritance
  - not types -> solved by XML schema
  - concept of "relationship" not well developed (IDREF)
  - Only "exact" query(XPath), structural similarity?