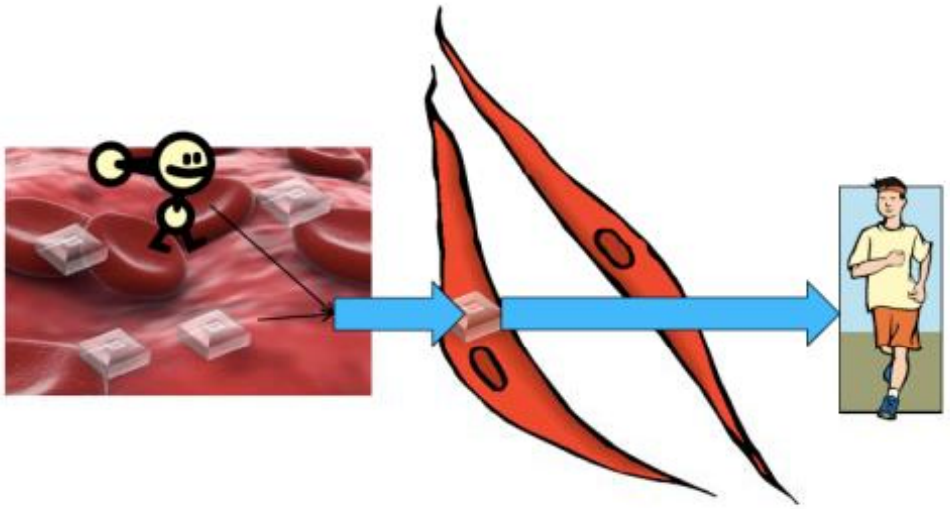
XML and RDF

• • •

Slides based on slides by Lena Strömbäck and Fang Wei-Kleiner

XML motivation example



Muscle cell

Describe the transformation process of sugar in the blood to the energy in muscle cells.

Relational modeling

Compartment	
Id	Name
Blood	Inblood
Cell	Musclecell

Reaction	
Id	Name
Tocell	Sugartocell
Move	Makemovement

Reactant	
Reaction	Species
ToCell	Sug1
ToCell	Ins
Move	Sug2

Species		
Id	Name	Compartment
Sug1	Sugar	Blood
Ins	Insulin	Blood
Sug2	Sugar	Cell
En	Energy	Cell

Product	
Reaction	Species
ToCell	Sug2
Move	En

Modeling problem

- Far from semi-structured proposal
  - Not suitable for describing tree structure
  - Too general or many tables
- Static – all attributes typed
- All data entries atomic – in principle

XML representation

- Ordered tree
  - Similar to semi-structured proposal
- Extensible
  - New kinds of data can be integrated
- Flexible
  - Easy to mix different kinds of data

```
<?xml version="1.0" encoding="UTF-8"?>
<minimodel name="sugartransport"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="minimodel.xsd">
   <listOfCompartments>
      <compartment id="blood" name="inblood" />
      <compartment id="cell" name="musclecell" />
   </listOfCompartments>
   <listOfSpecies>
      <species id="sug1" name="sugar" compartment="blood" />
      <species id="ins" name="insulin" compartment ="blood"/>
      <species id="sug2" name="sugar" compartment ="cell"/>
      <species id="en" name="energy" compartment ="cell"/>
   </listOfSpecies>
   <listOfReactions>
      <reaction id="tocell" name="sugartocell">
         <listOfReactants>
            <speciesReference species="sug1"/>
            <speciesReference species="ins"/>
         </listOfReactants>
         <listOfProducts>
            <speciesReference species="sug2"/>
         </listOfProducts>
      </reaction>
      <reaction id="move" name="makemovement">
         <listOfReactants>
            <speciesReference species="sug2"/>
         </listOfReactants>
         <listOfProducts>
            <speciesReference species="en"/>
         </listOfProducts>
      </reaction>
   </listOfReactions>
</minimodel>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<minimodel name="sugartransport"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="minimodel.xsd">
   <listOfCompartments>
      <compartment id="blood" name="inblood" />
      <compartment id="cell" name="musclecell" />
   </listOfCompartments>
   <listOfSpecies>
      <species id="sug1" name="sugar" compartment="blood" />
      <species id="ins" name="insulin" compartment ="blood"/>
      <species id="sug2" name="sugar" compartment ="cell"/>
      <species id="en" name="energy" compartment ="cell"/>
   </listOfSpecies>
   <listOfReactions>
      <reaction id="tocell" name="sugartocell">
         <listOfReactants>
            <speciesReference species="sug1"/>
            <speciesReference species="ins"/>
         </listOfReactants>
         <listOfProducts>
            <speciesReference species="sug2"/>
         </listOfProducts>
      </reaction>
      <reaction id="move" name="makemovement">
         <listOfReactants>
            <speciesReference species="sug2"/>
         </listOfReactants>
         <listOfProducts>
            <speciesReference species="en"/>
         </listOfProducts>
      </reaction>
   </listOfReactions>
</minimodel>
```

- Tag: reaction
- Start tag: <reaction>
- End tag: </reaction>
- Element: <reaction>…</reaction>
- Elements can be nested
- Attribute: <reaction id= “tocell”
- Namespaces allow external schemas and qualified names

  Xmlns:xsi=“…

Defining the XML model: DTD

- A Document Type Definition (DTD) defines the legal building blocks of an XML document.
- It defines the document structure with a list of legal elements and attributes.
- In the DTD all XML documents are one of:
  - Elements
  - Attributes
  - Entities
  - PCDATA – parsed character data
  - CDATA – character data

DTD example

```
<!DOCTYPE bibliography [
        <!ELEMENT bibliography (book+)>
        <!ELEMENT book (title, author*, publisher?, year?, section*)>
        <!ATTLIST book ISBN CDATA #REQUIRED>
        <!ATTLIST book price CDATA #IMPLIED>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT author (#PCDATA)>
        <!ELEMENT publisher (#PCDATA)>
        <!ELEMENT year (#PCDATA)>
        <!ELEMENT i (#PCDATA)>
        <!ELEMENT content (#PCDATAli)*>
        <!ELEMENT section (title, content?, section*)>
]>
```

+: 1+ *: 0+, ?: 0 or 1

#IMPLIED : optional

XML Schema

- The XML Schema defines the legal building blocks of an XML document.
- An XML Schema:
  - defines elements
  - defines attributes
  - defines which elements are child elements
  - defines the order of child elements
  - defines the number of child elements
  - defines data types for elements and attributes
  - defines default and fixed values for elements and attributes

DTD vs XML Schema

- Data

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

- DTD

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

DTD vs XML Schema

- XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

XML Schema vs. DTD

- XML Schemas are extensible to future additions
  - extend element definitions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

Why use DTD or XML Schema?

- Benefits of not using them
  - Unstructured data is easy to represent
  - Overhead of validation is avoided

- Benefits of using them
  - Serve as schema for the XML data
    - Guards against errors
    - Helps with processing
  - Facilitate information exchange
    - People can agree to use a common DTD or XML Schema to exchange data (e.g., XHTML)

RDF: Resource Description Framework

- Framework for describing resources on the web
- Designed to be read and understood by computers
- Not designed for being displayed to people
- Written in XML
- RDF is a W3C Recommendation

RDF example

```
<?xml version="1.0" encoding="UTF-8"?>

<species metaid="_506372" id="E1" name="MAPKKK activator"
compartment="compartment"
 initialConcentration="3e-05">
 <annotation>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
   xmlns:bqmodel="http://biomodels.net/model-qualifiers/">
   <rdf:Description rdf:about="#_506372">
    <bqbiol:isVersionOf>
     <rdf:Bag>
      <rdf:li rdf:resource="http://www.ebi.ac.uk/interpro/#IPR003577"/>
     </rdf:Bag>
    </bqbiol:isVersionOf>
   </rdf:Description>
  </rdf:RDF>
 </annotation>

</species>
```

RDF data model: triples

- A **Resource** is anything that can have a URI, such as our molecule "_506372 ”
- A **Property** is a Resource that has a name, such as “isVersionof"
- A **Property value** is the value of a Property, such as "IPR003577 "
- (note that a property value can be another resource)

Suitable for semi-structured data.

RDF Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

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

   <rdf:Description rdf:ID="protein">
      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
      <rdfs:subClassOf rdf:resource="#species"/>
   </rdf:Description>

</rdf:RDF>
```

Define relations between objects

Query languages for XML

- XPath
  - Path expressions with conditions
  - Building block of other standards (e.g. XQuery)
- XQuery
  - XPath + full-fledged SQL-like query language

```
- <minimodel name="sugartransport" xsi:noNamespaceSchemaLocation="minimodel.xsd">
  - <listOfCompartments>
      <compartment id="blood" name="inblood"/>
      <compartment id="cell" name="musclecell"/>
    </listOfCompartments>
  - <listOfSpecies>
      <species id="sug1" name="sugarinblood" compartment="blood"/>
      <species id="ins" name="insulin" compartment="blood"/>
      <species id="sug2" name="sugarincell" compartment="cell"/>
      <species id="en" name="energy" compartment="cell"/>
    </listOfSpecies>
  - <listOfReactions>
    - <reaction id="tocell" name="sugartocell">
      - <listOfReactants>
          <speciesReference species="sug1"/>
          <speciesReference species="ins"/>
        </listOfReactants>
      - <listOfProducts>
          <speciesReference species="sug2"/>
        </listOfProducts>
      </reaction>
    - <reaction id="move" name="makemovement">
      - <listOfReactants>
          <speciesReference species="sug2"/>
        </listOfReactants>
      - <listOfProducts>
          <speciesReference species="en"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
  </minimodel>
```

XPath

- XPath specifies path expressions that match XML data by navigating down (and occasionally up and across) the tree
- Example
  - Query: **/minimodel/listOfReactions/reaction**
    - Like a UNIX path
  - Result: all reaction elements reachable from root via the path

Basic Xpath constructs

/	separator between steps in a path
name	matches any child element with this tag name
*	matches any child element
@*name*	matches the attribute with this name
@*	matches any attribute
//	matches any descendent element or the current element itself
.	matches the current element
..	matches the parent element

Simple XPath examples

- All reactions

  /minimodel/listOfReactions/reaction
- All reaction names

  /minimodel/listOfReactions/reaction/@name
- All reaction elements, anywhere in the document

  //reaction
- All species id, anywhere in the document

  //species/@id
- Species of the minimodel

  /minimodel/*/species

Predicates in path expressions

[*condition*] matches the "current" element if *condition* evaluates to true on the current element

- Species in blood compartment

  /minimodel//species[@compartment="blood"]
- Reactions with listOfProducts as child element

  //reaction[listOfProducts]
- Name of species in blood compartment

  /minimodel//species[@compartment="blood"]/@name

Predicates

- Predicates can have and 's and or 's

- Species with id as 'en' and compartment as 'cell'

```
//species[@id="en" and @compartment="cell"]
```

- Species with id as 'en' or compartment as 'cell'

```
//species[@id='en' or @compartment='cell']
```

Predicates involving nodesets

- **/minimodel/listOfSpecies [species/@compartment="cell"]**
- There may be multiple species element, so **species/@compartment** in general returns a node-set (in XPath terminology)
  - The predicate evaluates to true as long as it evaluates true for at least one node in the node-set, i.e., at least one species has a compartment with value "**cell**"
- Tricky query

/minimodel/listOfSpecies [species/@compartment='cell' and species/@compartment !='cell']

XPath functions

- contains(*x*, *y*) true if string ***x*** contains string ***y***
- count(**node-set**) counts the number nodes in **node-set**
- position() returns the context position (roughly, the position of the current node in the node-set containing it)
- last() returns the "context size" (roughly, the size of the node-set containing the current node)
- name() returns the tag name of the current element

More examples

- All elements whose tag names contain "species"

  //*[contains(name(), "species")]
- Name of the first species

  /minimodel/listOfSpecies/species[position()=1]/@name

  A shorthand: **/minimodel/listOfSpecies/species[1]/@name**
- Name of the last species

  /minimodel/listOfSpecies/species[position()=last()]/@name
- Lists with fewer than 10 species

  /minimodel/listOfSpecies [count(species)<10]
- All elements whose parent's tag name is not "reaction"

  //*[name()!='reaction']/*

General XPath location steps

- Technically, each XPath query consists of a series of location steps separated by /
- Each location step consists of
  - An axis: one of **self, attribute, parent, child, ancestor, ancestor-or-self, descendant, descendant-or-self, following, following-sibling, preceding, preceding-sibling,** and **namespace**
  - A node-test: either a name test (e.g., **reaction**, *) or a type test (e.g., **text(), node(), comment()**), separated from the axis by ::
  - Zero of more predicates (or conditions) enclosed in square brackets

Example

- Verbose (axis, node test, predicate):

/child::minimodel/descendant-or-self::node()/child::species[attribute::id="sug1"]

- Abbreviated:

/minimodel//species[@id='sug1']

- **child** is the default axis
- // stands for **/descendant-or-self::node()/**

Example

- Which of the following queries correctly find the third **speciesReferences** in the entire input document?
- **//speciesReference [position()=3]**
  - Finds all third **speciesReferences** (for each of its parent element)
- **/descendant-or-self::node()[name()="speciesReference" and position()=3]**
  - Returns the third element in the document if it is a **speciesReferences**
- **/descendant-or-self::node()[name()="speciesReference"][position()=3]**
  - After the first condition is passed, the evaluation context changes
    - Context size: # of nodes that passed the first condition
    - Context position: position of the context node within the list of nodes

XQuery

- XPath + full-fledged SQL-like query language
- XQuery expressions can be
  - XPath expressions
  - FLWOR expressions
  - Quantified expressions
  - Aggregation, sorting, and more…
- An XQuery expression in general can return a new result XML document
  - Compare with an XPath expression, which always returns a sequence of nodes from the input document or atomic values (boolean, number, string, etc.)

FLWOR expression

```
<result>{
  for $s in doc("minimodel.xml")/minimodel//species
  let $n := $s/@name
  where $s/@compartment = 'cell'
  return
    <cellspecies>
      { $s/@id }
      { $n }
    </cellspecies>
}</result>
```

for: loop
let: assignment
where: filter condition
order by
return: result construction

FLWOR expression

```
for $x in doc('minimodel.xml')//species
    where data($x/@compartment)="blood"
    return $x
```

data function accepts a sequence of items
and returns their typed values

```
- <minimodel name="sugartransport" xsi:noNamespaceSchemaLocation="minimodel.xsd">
  - <listOfCompartments>
      <compartment id="blood" name="inblood"/>
      <compartment id="cell" name="musclecell"/>
    </listOfCompartments>
  - <listOfSpecies>
      <species id="sug1" name="sugarinblood" compartment="blood"/>
      <species id="ins" name="insulin" compartment="blood"/>
      <species id="sug2" name="sugarincell" compartment="cell"/>
      <species id="en" name="energy" compartment="cell"/>
    </listOfSpecies>
  - <listOfReactions>
    - <reaction id="tocell" name="sugartocell">
      - <listOfReactants>
          <speciesReference species="sug1"/>
          <speciesReference species="ins"/>
        </listOfReactants>
      - <listOfProducts>
          <speciesReference species="sug2"/>
        </listOfProducts>
      </reaction>
    - <reaction id="move" name="makemovement">
      - <listOfReactants>
          <speciesReference species="sug2"/>
        </listOfReactants>
      - <listOfProducts>
          <speciesReference species="en"/>
        </listOfProducts>
      </reaction>
    </listOfReactions>
  </minimodel>
```

Join in XQuery

```
for $x in doc('minimodel.xml')//species,
    $y in doc('minimodel.xml')//compartment
where data($x/@compartment)=data($y/@id)
return
    <pair> {data($x/@name)} is inside {data($y/@name)} </pair>

for $x in doc('minimodel.xml')//species,
    $y in doc('minimodel.xml')//compartment
          [@id=data($x/@compartment)]
return
    <pair> {data($x/@name)} is inside {data($y/@name)} </pair>
```

XQuery functions

```
declare function local:fak($n)
    {if ($n=1)
       then 1
       else ($n * local:fak($n - 1))
    };
<res>{local:fak(3)}</res>
```

Note: factorial（recursion）