XML (and JSON)

Olaf Hartig

• • •

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

About XML

- E**x**tensible **M**arkup **L**anguage
- Standardized format for data exchange

Olaf Hartig
Linköping University
olaf.hartig@liu.se

Patrick Lambrix
Linköping University

Text Document
(with data about contacts)

```
<contacts>
<person>
<name>Olaf Hartig</name>
<affil>Linköping University</affil>
<email>olaf.hartig@liu.se</email>
</person>
<person>
<name>Patrick Lambrix</name>
<affil>Linköping University</affil>
</person>
</contacts>
```

XML Document

XML Documents

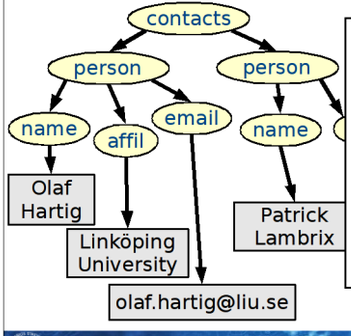
- Ordinary text files with special markup (labeled brackets) that indicates elements in the data
- Elements consist of:
  - Start tag
  - Content
  - End tag
- Content may be other elements (nesting!), or text, or a mixture of both
- One root element per XML document

```
<contacts>
<person>
<name>Olaf Hartig</name>
<affil>Linköping University</affil>
<email>olaf.hartig@liu.se</email>
</person>
<person>
<name>Patrick Lambrix</name>
<affil>Linköping University</affil>
</person>
</contacts>
```

XML Document

XML Tree

- Nested elements and their text content form an ordered tree




```
<contacts>
<person>
<name>Olaf Hartig</name>
<affil>Linköping University</affil>
<email>olaf.hartig@liu.se</email>
</person>
<person>
<name>Patrick Lambrix</name>
<affil>Linköping University</affil>
</person>
</contacts>
```

XML Document

Attributes and Empty Elements

- Elements may have attributes (name-value pairs)
  - Attributes added to start tag
  - Example:

```
...
<person id="3861">
  ...
</person>
...
```

- Elements may be empty (no content)
  - Empty elements may still have attributes
  - Special syntax:

```
...
<nothing/>
<almost_nothing  but="something"/>
...
```


Well-formed XML

- An XML document is *well-formed* if it meets a number of syntactic rules
  - Each end tag must match a start tag
  - Tags must be nested correctly
  - No element with attributes that have the same name
  - etc.

Document Type Definition (DTD)

- Defines the elements and attributes that can be used in an XML document
  - i.e., like a "grammar" for a class of documents
- DTD of an XML document can be extern or intern:

Extern: separate file, referenced in the header of the XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE contacts SYSTEM "contact_information.dtd">

<contacts> ... </contacts>
```

Intern: specified directly in the header (next slide)

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 (#PCDATA | i)*>
    <!ELEMENT section (title, content?, section*)>
]>
```

1 or more

0 or 1

0, 1, or more

optional

*p*arsed *c*haracter *data* (i.e., a "text" node)

alternative

recursion

XML Schema

- Defines the elements and attributes that can be used in an XML document (like a DTD)
- In contrast to DTDs, XML Schemas ...
  - ... are richer and more powerful
  - ... support data types
    (for attribute values and "text" content)
  - ... support namespaces
    (for element names and attribute names)
  - ... are extensible to future additions
    (extensions of element definitions)
  - ... are written in XML

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>
```


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 a schema for the XML data
    - Guards against errors
    - Helps with processing
  - Facilitate information exchange
    - People can agree on a common DTD or XML Schema to exchange data

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="muscleccell"/>
    </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 Involving Node-Sets

/minimodel/listOfSpecies[species/@compartment="cell"]

- There may be multiple species elements, so **species/@compartment** in general returns a so called *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*) returns the number of 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" (the size of the node-set containing the current node)

name() returns the tag name of 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 or 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
  - Aggregation, sorting, and several more…
- Result of evaluating an XQuery expression can be a new XML document
  - In contrast, result of evaluating an XPath expression is 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="muscleccell"/>
    </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)

About JSON

- **J**ava**S**cript **O**bject **N**otation
- Another standardized format for data exchange
- Text files that describe objects
  - *Object*: unordered container of key-value pairs
  - *Key*: any string
  - *Value*: number, string, boolean, null, another object, or an array
  - *Array*: ordered sequences of values

About JSON

```
{ "contacts": [
   {
      "name": "Olaf Hartig",
      "affil": "Linköping University",
      "email": "olaf.hartig@liu.se"
   },
   {
      "name": "Patrick Lambrix",
      "affil": "Linköping University"
   }
 ]
}
```

JSON Document

- *Object*: unordered container of key-value pairs
- *Key*: any string
- *Value*: number, string, boolean, null, another object, or an array
- *Array*: ordered sequences of values