Introduction to Data Management
CSE 344

Lecture 12: XML and XPath

XML Outline

• What is XML?
• Syntax
• Semistructured data
• DTDs
• XPath

What is XML?

• Stands for eXtensible Markup Language
  1. Advanced, self-describing file format
  2. Based on a flexible, semi-structured data model
• Applications:
  – Data exchange
  – Storing data without a rigid schema: advertisements
  – Configuration files: e.g. Web.Config
  – Document markup: e.g. XHTML

We will study only XML as data

Some XML Benefits

• Simple human-readable format
• Machine readable format
• Software- and hardware- independent way of storing data
• Flexible structure (we will see today)

Data Exchange

• Relational data does not have a syntax
  – I can’t “give” you my relational database or parts of it
  – Need some file format: CSV (comma-separated-values), ASN.1
• XML
  – Is a more advanced file format
  – Also has its own data model: semistructured
• Main idea: applications exchange information in XML

XML vs Relational

• Relational data model
  – Rigid flat structure (tables)
  – Schema must be fixed in advance
  – Binary representation: good for performance, bad for exchange
  – Query language based on Relational Calculus
• Semistructured data model / XML
  – Flexible, nested structure (trees)
  – Does not require predefined schema (“self-describing”)
  – Text representation: good for exchange, bad for performance
  – Query language borrows from automata theory
From HTML to XML

HTML describes the presentation

The diagram shows a conversion from HTML to XML.

HTML

```html
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
Abiteboul, Hull, Vianu<br>
Addison Wesley, 1995
</p>
<p> <i> Data on the Web </i>
Abiteboul, Buneman, Suciu<br>
Morgan Kaufmann, 1999
</p>
```

XML Syntax

```xml
<bibliography>
  <book>
    <title> Foundations of Databases </title>
    <author> Abiteboul </author>
    <author> Hull </author>
    <author> Vianu </author>
    <publisher> Addison Wesley </publisher>
    <year> 1995 </year>
  </book>
  ...
</bibliography>
```

XML Terminology

- Tags: book, title, author, ...
- Elements: <book>, <title>, <author>, ...
- Elements are nested
- Empty element: <red></red> abbrv. <red/>
- An XML document: single root element

Well-formed XML document
- Has matching tags
- A short header
- And a root element

Well-Formed XML

```xml
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<book price = "55" currency = "USD">
  
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  ...
  <year> 1995 </year>
</book>
```

More XML: Attributes
**Attributes v.s. Elements**

```
<book price="55" currency="USD">
    <title>Foundations of DBs</title>
    <author>Abiteboul</author>
    ...  
    <year>1995</year>
</book>
```

Attributes are alternative ways to represent data.

**Comparison**

<table>
<thead>
<tr>
<th>Elements</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordered</td>
<td>Unordered</td>
</tr>
<tr>
<td>May be repeated</td>
<td>Must be unique</td>
</tr>
<tr>
<td>May be nested</td>
<td>Must be atomic</td>
</tr>
</tbody>
</table>

**XML Semantics: a Tree !**

```
<person id="o555">
    <name>Mary</name>
    <address>
        <street>Maple</street>
        <no>345</no>
        <city>Seattle</city>
    </address>
</person>
```

Order matters !!!

**XML Data**

- **XML is self-describing**
- Schema elements become part of the data
  - Relational schema: `person(name,phone)`
  - In XML `<person>`, `<name>`, `<phone>` are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

**Mapping Relational Data to XML Data**

The canonical mapping:

```
<person>
    <name>John</name>
    <phone>3634</phone>
    <order date="2002" product="Gizmo">
        <product>Gi</product>
    </order>
</person>
```

**Mapping Relational Data to XML Data**

Application specific mapping

<table>
<thead>
<tr>
<th>Person</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>3634</td>
</tr>
<tr>
<td>Sue</td>
<td>6343</td>
</tr>
<tr>
<td>Dick</td>
<td>6363</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>2002</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Sue</td>
<td>2004</td>
<td>Gadget</td>
</tr>
<tr>
<td>Dick</td>
<td>2002</td>
<td>Gadget</td>
</tr>
</tbody>
</table>
XML is Semi-structured Data

- Missing attributes:
  ```xml
  <person>
    <name>John</name>
    <phone>1234</phone>
  </person>
  <person>
    <name>Joe</name>
  </person>
  ```

- Could represent in a table with nulls

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1234</td>
</tr>
<tr>
<td>Joe</td>
<td>-</td>
</tr>
</tbody>
</table>

- Repeated attributes

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>2345</td>
</tr>
<tr>
<td></td>
<td>3456</td>
</tr>
</tbody>
</table>

- Impossible in tables:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>2345</td>
</tr>
<tr>
<td></td>
<td>3456</td>
</tr>
</tbody>
</table>

XML is Semi-structured Data

- Attributes with different types in different objects

```xml
<person>
  <name>John</name>
  <first>John</first>
  <last>Smith</last>
  <phone>1234</phone>
</person>
```

- Nested collections
- Heterogeneous collections:
  - `<db>` contains both `<book>`s and `<publisher>`s

Schema

- Structured name!

Document Type Definitions (DTD)

- Part of the original XML specification
- An XML document may have a DTD
- XML document:
  - **Well-formed**: if tags are correctly closed
  - **Valid**: if it has a DTD and conforms to it
- Validation is useful in data exchange
- Use `http://validator.w3.org/check` to validate

DTD

- **Goals:**
  - Define what tags and attributes are allowed
  - Define how they are nested
  - Define how they are ordered

- Superseded by XML Schema (Book Sec. 11.4)
  - Very complex: DTDs still used widely
Very Simple DTD

```xml
<!DOCTYPE company [ 
  <!ELEMENT company ((person|product)*)> 
  <!ELEMENT person (ssn, name, office, phone?)> 
  <!ELEMENT phone (#PCDATA)> 
  <!ELEMENT office (#PCDATA)> 
  <!ELEMENT product (pid, name, description?)> 
  <!ELEMENT pid (#PCDATA)> 
  <!ELEMENT description (#PCDATA)> ]>
```

Example of valid XML document:

```
<company>
  <person>
    <ssn>123456789</ssn>
    <name>John</name>
    <office>B432</office>
    <phone>1234</phone>
  </person>
  <person>
    <ssn>987654321</ssn>
    <name>Jim</name>
    <office>B123</office>
  </person>
  <product>...
```

DTD: The Content Model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

DTD: Regular Expressions

- Sequence
- Optional
- Kleene star
- Alternation

DTD: Attributes

From "sample-xml-with-dtd.xml"

```
<!DOCTYPE bib [ 
  <!ELEMENT bib (book)*> 
  <!ELEMENT book (title, (author+ | editor+), publisher?, price)> 
  <!ATTLIST book year CDATA #REQUIRED > 
  ... ]>
```

DTD: Text

- #PCDATA ("Parsed Character Data") = the text inside elements
- CDATA ("Character Data") = the text inside attributes
- There is no #CDATA and no PCDATA
Cross-references

- An attribute is called an ID attribute if it is unique
- An attribute is called IDREF if it references an ID
- The DTD defines which attribute(s) are ID/IDREFs

From "mondial.dtd" (HW4)
```xml
<!ATTLIST country
car_code ID #IMPLIED
area CDATA #IMPLIED
capital IDREF #IMPLIED
memberships IDREFS #IMPLIED>
```

Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML
- XSLT = recursive traversal
  - will not discuss in class

Sample Data for Queries

```
<book>
  <publisher>Addison-Wesley</publisher>
  <author>Serge Abiteboul</author>
  <author><first-name>Rick</first-name><last-name>Ullman</last-name></author>
  <title>Foundations of Databases</title>
  <year>1995</year>
</book>
```

```
<book price="55">
  <publisher>Freeman</publisher>
  <author>Jeffrey D. Ullman</author>
  <title>Principles of Database and Knowledge Base Systems</title>
  <year>1998</year>
</book>
```

Data Model for XPath

XPath returns a sequence of items. An item is either:

- A value of primitive type, or
- A node (doc, element, or attribute)

```
<bib>
  <book>
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
  </book>
</bib>
```

XPath: Simple Expressions

```
/bib/book/year
```

Result: `<year>1995</year>`

```
/bib/book/year
```

Result: `<year>1998</year>`

```
/bib/paper/year
```

Result: empty  (there were no papers)
XPath: Restricted Kleene Closure

Result:

```xml
<author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>
```

Result: 

```xml
<first-name> Rick </first-name>
```

XPath: Attribute Nodes

Result: “55”

@price means that price has to be an attribute

XPath: Wildcard

Result: 

```xml
<author/> Rick <first-name>
<last-name> Hull </last-name>
```

* Matches any element
@* Matches any attribute

XPath: Text Nodes

Result: 

```xml
Surname
...
Surname
```

Functions in XPath:

- `text()` matches the text value
- `node()` matches any node (= * or @* or `text()`)
- `name()` returns the name of the current tag

XPath: Predicates

Result: 

```xml
<author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
```

XPath: More Predicates

Result: 

```xml
<last-name> ... </last-name>
<last-name> ... </last-name>
```

How do we read this?
First remove all qualifiers (predicates):

```xml
/bib/book/author/last-name
```

Then add them one by one:

```xml
/bib/book/author/last-name[address[./zip[city]]]last-name
```
XPath: More Predicates

/bib/book[@price < 60]
/bib/book[author/@age < 25]
/bib/book[author/text()]

XPath: Position Predicates

/bib/book[2]
/bib/book[last()]
/bib/book[@year = 1998][2]
/bib/book[2][@year = 1998]

The 2nd book
The last book
The 2nd of all books in 1998
2nd book IF it is in 1998

XPath: More Axes

. means current node /bib/book[.//review]
/bib/book[.//review] Same as /bib/book[review]
/bib/author/. /first-name Same as /bib/author/first-name

XPath: More Axes

.. means parent node /bib/author/.. /author/zip
/bib/book[.//review/..//comments]
/bib/book[.//review/..//comments]
/bib/book[.//*[comments][review]]
/bib/book[.//*[comments][review]]
Hint: don't use ..

XPath: Summary

bib matches a bib element
* matches any element
/ matches the root element
/bib matches a bib element under root
bib/paper matches a paper in bib
bib/paper matches a paper in bib, at any depth
/paper matches a paper at any depth
paper/book matches a paper or a book
@price matches a price attribute
bib/book/@price matches price attribute in book, in bib
bib/book[@price="55"] or @price="55"]author/last-name matches...
bib/book[@price="55"] +/author/last-name matches...