CmpE 473
Internet Programming

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

XML

Based largely on

Service-Oriented Computing: Semantics, Processes, Agents
– Munindar P. Singh and Michael N. Huhns, Wiley, 2004

Examples from www.w3schools.com

Markup History

- None
- Ad hoc tags
- SGML (Standard Generalized Markup L): complex, few reliable tools
- HTML (HyperText ML): simple, unprincipled, mixes structure and display
- XML (eXtensible ML): simple, yet extensible subset of SGML to capture new vocabularies
  - Machine processible
  - Comprehensible to people: easier debugging

XML Basics

```
<?xml version="1.0“ encoding="ISO-8859-1"?>

<book>
    <title>My First XML</title>
    <prod id="33-657" media="paper"></prod>
    <chapter>Introduction to XML
     <para>What is HTML</para>
     <para>What is XML</para>
    </chapter>
    <chapter>XML Syntax
     <para>Elements must have a closing tag</para>
     <para>Elements must be properly nested</para>
    </chapter>
<!– Example comment -->
</book>
```

XML Declaration

Element content

Empty content

Mixed content

Simple content

Comment

Parsing

- An XML document maps to a parse tree.
  - Each tag has to end and end once
    - Contrast with HTML <p>
  - Nesting structure (one root)
    - Every other element under this root
  - Each attribute occurs at most once; quoted string
  - <note date="12/11/2002">
  - Tags are case sensitive
  - White space preserved
- Well-formed XML documents can be parsed

Viewing

- XML only has content
- Provide information on how to display the content
- For Cascading Style Sheets
  - <?xml-stylesheet type="text/css" href="cd_catalog.css"?>
- XSL (the eXtensible Stylesheet Language)
  - Written in XML
  - <?xml-stylesheet type="text/xsl" href="simple.xsl"?>

Name Conflicts

- Two XML documents with

```
<table>
 <tr>
     <td>Apples</td>
     <td>Bananas</td>
 </tr>
</table>
<table>
     <name>African Coffee Table</name>
     <width>80</width>
     <length>120</length>
</table>
```

- Combining will result in a conflict

Name Prefix

```
<h:table  <h:tr>
   <h:td>Apples</h:td>
   <h:td>Bananas</h:td>
   </h:tr>
</h:table>
```

- Add a different prefix for the other table (e.g., f)

Namespaces

```
<h:table xmlns:h="http://www.w3.org/TR/html4/"> <h:tr>
   <h:td>Apples</h:td>
   <h:td>Bananas</h:td>
   </h:tr>
</h:table>
```

- xmlns: to uniquely identify; not to locate
- URI: Uniform Resource Identifier
  - Identify resources of any kind; including resources that are not network-accessible
- URL: Uniform Resource Locator

Well-Formed XML

- A "Well Formed" XML document has correct XML syntax.
- Review:
  - Must have a root element
  - Must have a closing tag
  - Are case sensitive
  - Must be properly nested
  - Its attribute values are quoted
- Does not mean they are meaningful.
- Ex: Define a person with age 200.

Validating

- Applications have an explicit or implicit syntax for their particular XML-based tags
  - If explicit, may be expressed in Document Type Definitions (DTDs) and XML Schemas
    - Best referred to definitions elsewhere
    - XML Schemas, expressed in XML, are superior to DTDs
  - When docs are produced by external components, they should be validated

XML Schema

- A data definition language for XML: defines a notion of *schema validity*
  - Same syntax as regular XML documents
  - Local scoping of subelement names
  - Incorporates namespaces
  - Types
    - Primitive (built-in): string, integer, float, date, …
    - Primitive (built-in): ID (key), IDREF (foreign key)
    - simpleType constructors: list, union
    - Restrictions: intervals, lengths, enumerations, regex patterns
    - Flexible ordering of elements
  - Key and referential integrity constraints

XML Example

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

XML Schema Example

```
<?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“ default=“pinar”/>
            <xs:element name="from" type="xs:string"/>
             <xs:element name="heading" type="xs:string“ fixed=“test”/>
             <xs:element name="body" type="xs:string"/>
            <xs:element name=“signDate" type="xs:date"/>
    </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Simple Type (1)

- <xs:element name="xxx" type="yyy"/>
- Some built-in types
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
- <xs:element name="lastname" type="xs:string"/>
- <xs:element name="age" type="xs:integer"/>
- <xs:element name="dateborn" type="xs:date"/>

Simple Type (2)

- Default Value
  - Set when no value is assigned
  - <xs:element name="color" type="xs:string" default="red"/>
- Fixed Value
  - Default value
  - Cannot be overwritten
  - <xs:element name="color" type="xs:string" fixed="red"/>

Attributes

- Simple types cannot have attributes
- Attributes themselves are simple types
- XML Example:
  - <lastname lang="EN">Smith</lastname>
- XSD Definition:
  - <xs:attribute name="lang" type="xs:string"/>
- Optional by definition
- To enforce usage:
  - <xs:attribute name="lang" type="xs:string" use="required"/>

Simple Type Restrictions (Facets)

Restriction on Values

```
<xs:element name="age">
  <xs:simpleType>
      <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Restriction on a Set of Values

```
<xs:element name="car"
type="carType"/>
<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Simple Type Restrictions (Facets)

```
<xs:element name="letter">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>

<xs:element name="gender">
     <xs:simpleType>
    <xs:restriction base="xs:string">
    <xs:pattern value="male|female"/>
    </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Restriction by Pattern

Simple Type Restrictions (Facets)

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9]{8}"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="letter">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-z])*"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Restriction by Pattern

complexType (1)

- Empty elements

```
<product pid="1345"/>
```

- Contain only other elements

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

- Contain only text

```
<food type="dessert">Ice cream</food>
```

- Contain both other elements and text

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

complexType (2)

Specifies types of elements with structure:

- Must use a *compositor* if >1 subelements
- Subelements with types
- Min and max occurrences (default 1) of subelements

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
    maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Compositors (1)

- *Sequence:* ordered
  - Can occur within other compositors
  - Allows varying min and max occurrence

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Compositors (2)

- *All:* unordered
  - Must occur directly below root element
  - Max occurrence of each element is 1

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Compositors (3)

- *Choice:* exclusive or
  - Can occur within other compositors

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Extensions

- <any> Element --- Anything can be put in place of it

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema: Key Namespaces

- http://www.w3.org/2001/XMLSchema
  - Conventional prefix: xsd
  - Terms for defining schemas: schema, element, attribute, …
  - The tag schema has an attribute targetNamespace
- http://www.w3.org/2001/XMLSchema-instance
  - Conventional prefix: xsi
  - Terms for use in instances: schemaLocation, null
- targetNamespace: user-defined

Document Object Model (DOM)

- Basis for parsing XML, which provides a node-labeled tree in its API
  - Conceptually simple: traverse by requesting tag, its attribute values, and its children
  - Processing program reflects document structure
  - Can edit documents
  - Inefficient for large documents: parses them first entirely to build the tree even if a tiny part is needed

DOM Example [Simeoni 2003]

```
Element s = d.getDocumentElement();
NodeList l =
   s.getElementsByTagName(“member”);
Element m = (Element) l.item(0);
int code = m.getAttribute(“code”);
NodeList kids = m.getChildNodes();
Node kid = kids.item(0);
String tagName = ((Element)kid).getTagName();
…
```

Parser

- Non-validating parser
  - Check for well-formdness
  - Doesn't check for validity
  - Doesn't need the schema
- Validating parser
  - Check for well-formness
  - Check for validity
- JAXP
  - Configure as validating or not
  - Validates a given XML document

APIs for XML

- Simple API for XML (SAX)
  - Sequential access parser API
  - Parser generates a sequence of events:
    - startElement, endElement, …
  - Programmer implements these as callbacks
    - More control for the programmer
  - Processing program does not reflect document structure
- Streaming API for XML (StAX)
  - Aims to be in the middle of two extremes
  - A cursor pulls information as needed
- Read JAXP: http://jaxp.java.net/

SAX Example [Saxproject.org]

```
class MySAXApp extends DefaultHandler {
public void startElement (String uri, String name, String
    qName, Attributes atts) {
    if ("".equals (uri))
      System.out.println("Start element: " + qName);
    else System.out.println("Start element: {" + uri + "}" +
    name);
}
public void endElement (String uri, String name, String
    qName) {
    if ("".equals (uri))
      System.out.println("End element: " + qName);
    else System.out.println("End element: {" + uri + "}" +
    name);
}
```

Uses of XML

- Exchanging information across software components
- Storing information in nonproprietary format
- XML documents represent structured descriptions:
  - Products, services, catalogs
  - Contracts
  - Queries, requests, invocations (as in SOAP)
- Data-centric versus document-centric (irregular, heterogeneous data, depend on entire doc for app-specific meaning) views

Directions

- Limitations of XML
  - Doesn't represent meaning
  - Enables multiple representations for the same information; transform if models known
- Trends: sophisticated approaches for
  - Querying and manipulating XML, e.g., XSLT
  - Binding to PLs and DBs
  - Semantics, e.g., RDF, OWL, …

Tools

- Many open and commercial tools
  - XMLSPY: XML and XML Schema edit & validate; XSL edit & transform
  - http://projects.apache.org/indexes/category.html#xml: Tools for manipulating and using XML
  - Eclipse: XML plug-ins
  - Example parsers: jdom, woodstox, dom4j, ...

JSON (1)

- JavaScript Object Notation
- Data-interchange format (as XML)
- Light-weight compared to XML
- Can natively be processed by Javascript functions and used as Javascript objects
- Advantage for Ajax

JSON (2)

- Data
  - Name: Value pairs
  - Values can be number, string, Boolean, array, object, null
- Object
  - {name:value, name:value}
- Array
  - Contains multiple objects
  - "employees":[

```
{"firstName":"John", "lastName":"Doe"},
{"firstName":"Anna", "lastName":"Smith"},
{"firstName":"Peter","lastName":"Jones"}
]
```

JSON (3)

```
<employees>
   <employee>
      <firstName>John</firstName> <lastName>Doe</lastName>
   </employee>
   <employee>
      <firstName>Anna</firstName> <lastName>Smith</lastName>
   </employee>
   <employee>
      <firstName>Peter</firstName> <lastName>Jones</lastName>
   </employee>
</employees>
```