

# CmpE 473 Internet Programming

Pinar Yolum  
[pinar.yolum@boun.edu.tr](mailto:pinar.yolum@boun.edu.tr)

Department of  
Computer Engineering  
Boğaziçi University

# Chapter 3 XML

Based largely on  
*Service-Oriented Computing: Semantics, Processes, Agents*  
– Munindar P. Singh and Michael N. Huhns, Wiley, 2004  
Examples from [www.w3schools.com](http://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>
```

The diagram illustrates the classification of XML content types. It shows a vertical stack of boxes: XML Declaration, Element content, Empty content, Mixed content, Simple content, and Comment. Lines connect these boxes to the corresponding parts of the XML code on the left. For example, 'XML Declaration' points to the opening tag, 'Element content' points to the <title> and <prod> tags, 'Empty content' points to the </prod> closing tag, 'Mixed content' points to the <chapter> tag and its children, 'Simple content' points to the <para> tags, and 'Comment' points to the <!-- Example comment --> line.

## 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

Fall 2007— Pinar Yolum

5

## 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"?>

Fall 2007— Pinar Yolum

6

## 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

Fall 2007— Pinar Yolum

7

## 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)

Fall 2007— Pinar Yolum

8

## 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

Fall 2007— Pinar Yolum

9

## Validating

- Applications have an explicit or implicit syntax for their particular XML-based tags
  - If explicit, may be expressed in 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

Fall 2007— Pinar Yolum

10

## 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

Fall 2007— Pinar Yolum

11

## 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>
```

Fall 2007— Pinar Yolum

12

## 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>
```

Fall 2007— Pinar Yolum

## 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"/>`

Fall 2007— Pinar Yolum

14

## 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"/>`

Fall 2007— Pinar Yolum

15

## 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: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>
```

Restriction by Pattern

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

Fall 2007— Pinar Yolum

18

## 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

Fall 2007— Pinar Yolum

19

## 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>`

Fall 2007— Pinar Yolum

--

## 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>
```

Fall 2007— Pinar Yolum

21

## 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>
```

Fall 2007— Pinar Yolum

22

## 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>
```

Fall 2007— Pinar Yolum

23

## 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>
```

Fall 2007— Pinar Yolum

24

## Extensions

- <any> Element
- ```
<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>
```

Fall 2007— Pinar Yolum

25

## 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

Fall 2007— Pinar Yolum

26

## 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

Fall 2007— Pinar Yolum

27

## 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();
...
```

Fall 2007— Pinar Yolum

28

## Parser

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

Fall 2007— Pinar Yolum

29

## Simple API for XML (SAX)

- 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
- Read
  - <http://www.saxproject.org/quickstart.html>
  - <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/>
  - JAXP: <http://java.sun.com/webservices/jaxp/index.jsp>

Fall 2007— Pinar Yolum

30

## 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);
}
```

Fall 2007— Pinar Yolum

31

## 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

Fall 2007— Pinar Yolum

32

## 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
  - Xml.apache.org: Tools for integration with Java
  - Eclipse: XML plug-ins