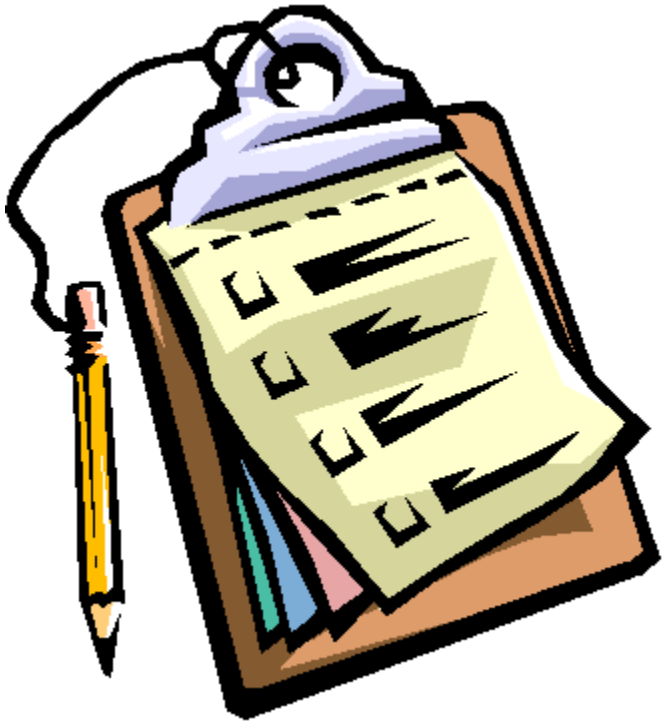
Internet Engineering

Semi-structured Data
XML, JSON

Sadegh Aliakbary

Agenda

- Introduction to XML and JSON
- Structure of an XML
  - DTD
  - XSD
- XML Parsing
  - DOM & SAX
- XML Query
  - XPath



XML

- XML: eXtensible Markup Language
- A markup language to describe data structure

```
<course>
  <title> InternetEngineering </title>
  <teacher> Aliakbary </teacher>
  <students>
     <student><fname>Ali</fname> <lname>Alavi</lname> </student>
     <student><fname>Taghi</fname><lname>Taghavi</lname> </student>
     …
  </students>
</course>
```

Why to Study XML: Benefits

- Simplify data sharing & transport
  - XML is text based and platform independent
- Extensive tools to process XML
  - To validate, to present, to search, …
- Extensible for different applications
  - A powerful tool to model/describe complex data
- A format for transferring data
  - e.g., in web application, data separation from HTML
  - E.g., table structure by HTML, table data by XML

XML Document Elements

- Markup
  - Elements
    - Tag + Content
  - Element Attributes
- Content
  - Parsed Character Data
  - Unparsed Character Data (CDATA)

XML Elements

- XML element structure
  - Tag + content

<tagname attribute="value">

Content

</tagname>

- *No predefined tag*
- If content is not CDATA, is parsed by parser
  - A value for this element
  - Child elements of this element

XML Elements' Attributes

- Tags (elements) are customized by attribute
  - *No predefined attributes*

```
<os install="factory">Windows</os>
<os install="user">Linux</os>
```

- Attribute vs. Tags (elements)
  - Attributes can be replaced by elements
    - Attribute cannot be repeated for an element
    - Attribute cannot have children

Basic XML Document Structure

```
<?xml version="1.0" encoding="UTF-16"?>
<root-tag>
<inner-tags>
  Data
</inner-tags>
<!-- Comment -->
</root-tag>
```

Example

```
<?xml version="1.1" encoding="UTF-8" ?>
<notebook>
   <name>ThinkPad</name>
   <model>T500</model>
   <spec>
     <hardware>
       <RAM>4GB</RAM>
     </hardware>
     <software>
       <OS>Linux, FC21 </OS>
     </software>
   </spec>
</notebook>
```

Example (CDATA)

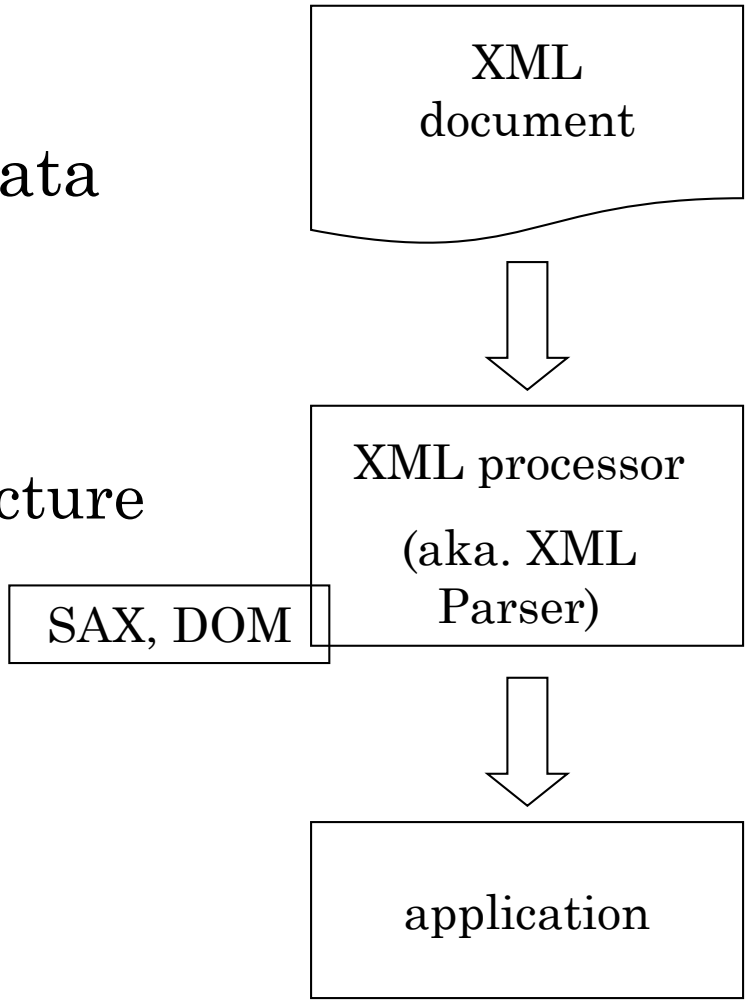
```
<operator>
  <comparison>
     <![CDATA[
          < <= == >= > !=
     ]]>
  </comparison>
</operator>
```

XML vs. HTML

- Tags
  - HTML: Predefined fixed tags
  - XML: No predefined (meta-language)
    - User defined tags & attributes
- Purpose
  - HTML: data + presentation
  - XML: structure + data
- Rules' strictness
  - HTML: loose (not XHTLM)
  - XML: strong/strict rule checking
- HTML is case-insensitive
  - But <Letter> is different from <letter> in XML

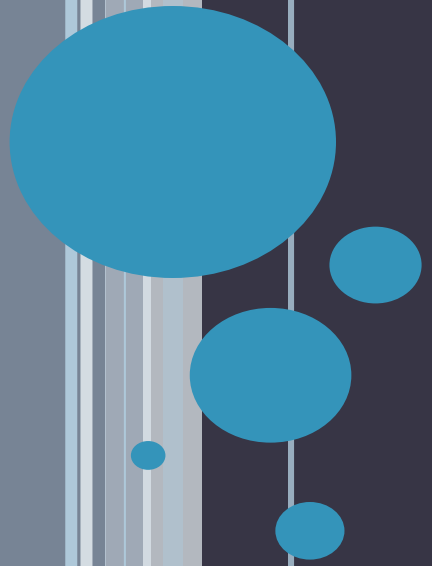
XML in General Application

- XML by itself does not do anything
- XML just describes the structure and the data
- Other applications **parse** XML and use it
  - Parsing: reading the XML file/string and getting its content according to the structure
- A similar approach is used for formats
- (e.g., zip, docx, and even user-defined format)
- so, what is the advantages of XML?!!!
  - *XML is standard*
  - *Available XML tools & technologies*



XML Technology Components

- Data structure (tree) representation
  - XML document (a text file or a string)
- Validation & Conformance
  - Document Type Definition (DTD) or XML Schema
- Element access & addressing
  - XPath, DOM, SAX
- Display and transformation
  - E.g., XSLT
- Programming, Database, Query, …



XML: Semi-structured Data

Types of Data

- Types of data:
  - Structured
  - Semistructured
  - Unstructured Data
- **Structured data**
  - Represented in a strict format
  - Example: information stored in relational databases

Unstructured Data

- **Unstructured data** or **unstructured** information
- Information that either does not have a pre-defined **data** model
- **Unstructured** information is typically text-heavy
- **Text**
- **`Example: HTML`**
  - An HTML web page is tagged
  - but HTML mark-up typically serves solely for rendering

HTML Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

- Note:
- HTML tags typically serves solely for rendering
- They are not about the “meaning” of data

Semistructured Data

- Has a certain structure
- Not all information collected will have identical structure
- Schema information mixed in with data values
- **Self-describing data**
- Such as?
  - **XML** and **JSON**

XML Example

- Note:
- Not all information collected in identical structure
- Schema information mixed in with data values
- **Self-describing data**
- Tags, elements, and attributes

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>ابوالقاسم فردوسی</author>
    <title>شاهنامه</title>
    <genre>ادبیات</genre>
    <price>۱۰۰</price>
    <publisher>امیرکبیر</publisher>
  </book>
  <book id="bk102">
    <author>رضا امیرخانی</author>
    <title>من او</title>
    <price>۴۵</price>
    <pages>234</pages>
  </book>
</catalog>
```

JSON Example

```
{
 "firstName": "John",
 "lastName": "Smith",
 "address": {
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
  "postalCode": "10021-3100"
  },
 "phoneNumbers": [
  {
   "type": "home",
   "number": "212 555-1234"
  },
  {
   "type": "office",
   "number": "646 555-4567"
  }
 ]}
```

JSON

- **J**ava**S**cript **O**bject **N**otation
  - Today, it is not restricted to javascript
- JSON is text-based
- Written with JavaScript object notation
  - It is easy to use Javascript for parsing a JSON
  - Example:
  - ***var myObj = { "name":"John", "age":31, "city":"New York" };***

Example

```
//Storing data:
myObj = { "name":"Ali", "age":31, "city":"Yazd" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

//Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

XML Hierarchical Data Model

- **Elements** and **attributes**
  - Main structuring concepts used to construct an XML document
- **Simple elements**
  - Contain data values
- **Complex elements**
  - Constructed from other elements hierarchically
- XML tag names
  - Describe the meaning of the data elements in the document

XML Hierarchical Data Model (cont'd.)

- XML attributes
  - Describe properties and characteristics of the elements (tags) within which they appear
- May **reference** another element in another part of the XML document
  - Common to use attribute values in one element as the references to another element

XML Query Languages

- **XPath**
  - a **query language** standards
  - Specify path expressions to identify certain nodes (elements) or attributes
  - that match specific patterns
- Other XML Query Languages
  - E.g., XQuery
    - Uses XPath expressions but has additional constructs

XPath Examples

- /company/department
  - returns all department nodes (elements) and their descendant subtrees.
- //employee [salary > 100]/employeeName
- /company/employee [salary > 100]/employeeName
- /company/project/projectWorker [hours >= 20.0]

XPath expressions

- Returns a sequence of items that satisfy a certain pattern as specified by the expression
- Either values (from leaf nodes) or elements or attributes
- **Qualifier conditions**
  - Further restrict nodes that satisfy pattern
- **Separators** used when specifying a path:
  - Single slash (/) and double slash (//)
- A single slash : the tag must appear as a direct child of the previous (parent) tag
- A double slash : the tag can appear as a descendant of the previous tag *at any level*

XPath: Basic Syntax

Expression	Description
/	Selects from the root node
//	Selects nodes from descendants of the current node
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes
|	Selects several paths (e.g., //title | //price)

XPath: Basic Syntax: Example

```
<bookstore>
  <book>
      <title lang="eng">عطر سنبل عطر کاج</title>
      <price>25</price>
  </book>
</bookstore>
```

Path Expression	Result
/bookstore	Selects the root element bookstore
/bookstore/book	Selects all book elements that are children of bookstore
/bookstore/book/title	Selects titles of all books
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

XPath: Advanced Syntax

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00

Indices start from one (not zero)

XML Documents and Databases

- A DBMS may store XML as text
- A DBMS may support XML data-type
  - And its operations
    - E.g, Xpath
    - Or even indexing on specific elements
    - E.g.,
    - SELECT College_Details.query('/STUDENTINFO') FROM College_Master WHERE College_ID = 1;

XML Parsing

- XML is **Well formed**
  - Every element should match pair of start and end tags
- XML Parsing
  - Interpreting the meaning of an XML
  - Approaches: DOM & SAX

XML Parsing: DOM vs SAX

- **DOM** (Document Object Model)
  - Manipulate resulting tree representation corresponding to a well-formed XML document
- **SAX** (Simple API for XML)
  - Processing of XML documents on the fly
    - Notifies processing program through callbacks whenever a start or end tag is encountered
  - Makes it easier to process large documents
  - Allows for **streaming**

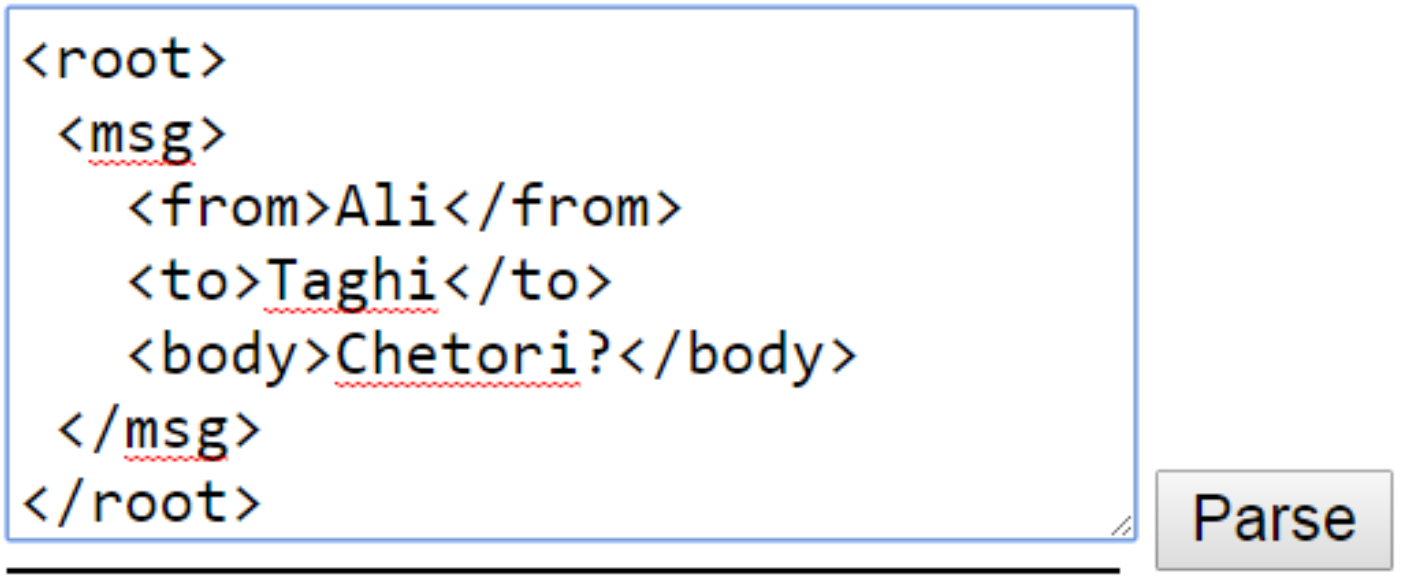
Example: XML DOM Parsing in JavaScript

- XML DOM is similar to HTML DOM
  - A tree of nodes (with different types: element, text, attr, …)
  - Nodes are accessed by `getElementsByTagName`
  - Nodes are objects (have method & fields)
  - DOM can be modified, e.g., create/remove nodes

XML DOM in JavaScript

- `DOMParser` can parse an input XML string
- Each node have
  - parentNode, childNodes, …
- Access to value of a node
  - In the DOM, everything is a node (with different types)
  - Element nodes do not have a content value
  - The content of an element is stored in a child node
    - To get content of a leaf element, the value of the first child node (text node) should got

Example: Message Parser




```
<body>
<textarea id="inputtext1" cols="50‟ rows="10">
</textarea>
<input type="button" onclick="parse()" value="Parse" />
  <br />
<div name="outputdiv"></div>
</body>
```

Example: Message Parser

```
function parse(){
  output = "";
  input = document.getElementById("inputtext1").value;
  parser = new DOMParser();
  xmlDoc = parser.parseFromString(input,"text/xml");
  messages = xmlDoc.getElementsByTagName("root")[0].children;
  for(i=0; i < messages.length; i++){
      msg = messages[i];
      fromNode = msg.getElementsByTagName("from")[0];
      fromText = fromNode.childNodes[0].nodeValue;
      toNode = msg.getElementsByTagName("to")[0];
      toText = toNode.childNodes[0].nodeValue;
      bodyNode = msg.getElementsByTagName("body")[0];
      bodyText = bodyNode.childNodes[0].nodeValue;
      output = output+fromText +" sent to " +toText + "<br />
               + bodyText ;
  }
  document.getElementsByName("outputdiv")[0].innerHTML =
  output;
}
```

SAX Parsing Example in Java

```
public void startElement(String uri, String localName,
    String qName, Attributes attributes) throws SAXException {

    this.elementStack.push(qName);

    if("driver".equals(qName)){
        Driver driver = new Driver();
        this.objectStack.push(driver);
        this.drivers.add(driver);
    } else if("vehicle".equals(qName)){
        this.objectStack.push(new Vehicle());
    }
}
```

```
public void endElement(String uri, String localName,
    String qName) throws SAXException {

    this.elementStack.pop();
    if("vehicle".equals(qName)){
            Vehicle vehicle = (Vehicle) object;
            this.vehicles.put(vehicle.vehicleId, vehicle);
        }
    }
}
```

Namespaces

- In XML, element names are defined by developers
  - Results in a conflict
  - when trying to mix XML documents from different XML applications

XML file 1

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

XML file 2

```
<table>
      <name>Dinner Table</name>
      <width>80</width>
<length>120</length>
</table>
```

Namespaces

- Name conflicts in XML can easily be avoided by using a *qualified names* according to a prefix
  - Qualified name is the prefixed name
  - Prefix is the namespaces
- Step 1: Namespace declaration
  - Defines a label (prefix) for the namespace and associates it to the namespace identifier
    - URI/URL is used to be universally unique
- Step 2: Qualified name
  - namespace prefix: local name

Namespaces

```
<?xml version="1.0"?>
<sbu:course
   xmlns:sbu="http://sbu.ac.ir">
  <sbu:university>
      <sbu:name>
          Shahid Beheshti University
      </sbu:name>
  </sbu:university>
  <sbu:name>
       Internet Engineering
  </sbu:name>
</sbu:course>
```

Default Namespaces

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

<table xmlns="http://www.dinnertable.com">
  <name>Dinner Table</name>
  <width>80</width>
  <length>120</length>
</table>
</alltables>
```

Instead of xmlns:sbu

XSL

- XSL stands for eXtensible Stylesheet Language, and is a style sheet language for XML documents
- XSLT (XSL Transform)
  - Transforms XML into other formats, like HTML
- What is XSLT (XSL Transformations)?
  - XSLT is an XML file that transforms an XML document into another document: e.g., XML or XHTML
  - Uses XPath

XSLT Example: XML Data file

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="xslt-test.xslt"?>
<class>
  <course>
      <name>Internet Engineering</name>
      <semester>Spring 2012</semester>
  </course>
  <student>
      <name>Ali</name><family>Alizadeh</family>
      <grade>18.0</grade><number>123</number>
  </student>
  <student>
      <name>Babak</name><family>Babaki</family>
      <grade>7.0</grade><number>234</number>
  </student>
  <student>
      <name>Hassan</name><family>Hassani</family>
      <grade>19.0</grade><number>345</number>
  </student>
</class>
```

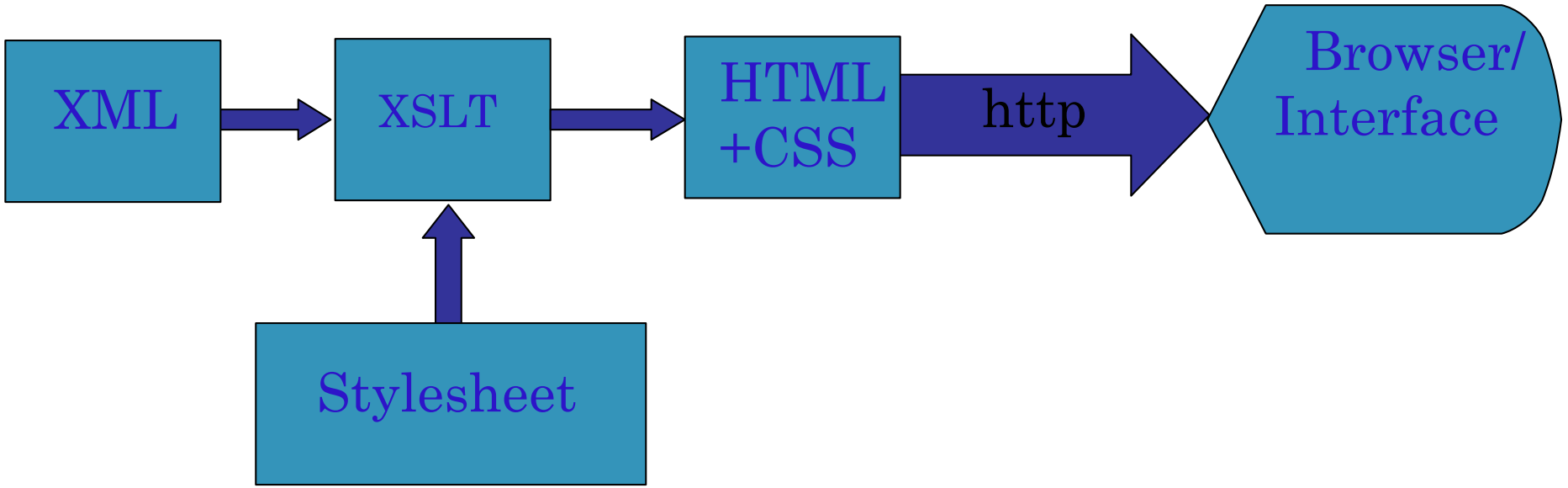
XSLT Example: XSLT file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html> <body>
      <h2>Course: <xsl:value-of select="/class/course/name"/></h2>
      <h3>Semester: <xsl:value-of select="//semester"/> </h3>
  <h3>Students:
  <xsl:for-each select="//family">
  "<xsl:value-of select="."/>"
  </xsl:for-each></h3>

  <table border="1">
    <tr> <th> Student # </th> <th>Name</th> <th>Family</th>
  <th>Grade</th></tr>
```

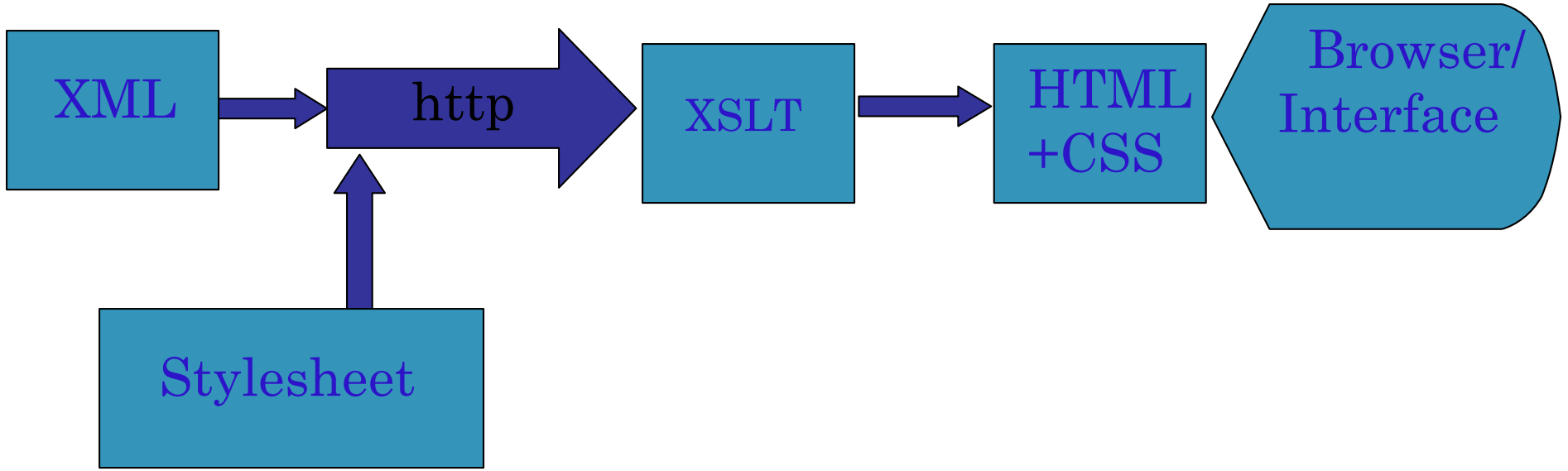
Where XSLT? Server Side

Server transforms XML to HTML/CSS;
Ship to client browser for display



Where XSLT? Client-Side

Server sends XML & Stylesheet to client
Client transforms XML to HTML & CSS



Structure of Data in XML

- An XML may follow a structure
  - Start and end tag pairs must follow that structure
- The structure is specified separately
- As XML **DTD (Document Type Definition)** or **XML schema (XSD)**
  - XML Schema utilize an XML-based syntax
  - XSD is more powerful

XML Validation

- XML is used to describe a [semi]structured data
  - The description must be *correct*
    - A valid XML file
- Correctness
  - Syntax
    - Syntax error → parser fails to parse the file
    - Syntax rules: e.g., all XML tags must be closed
  - Semantic (structure)
    - Application specific rules, e.g. student must have ID
    - Error → Application failure

XML Syntax Rules (Well-Formed)

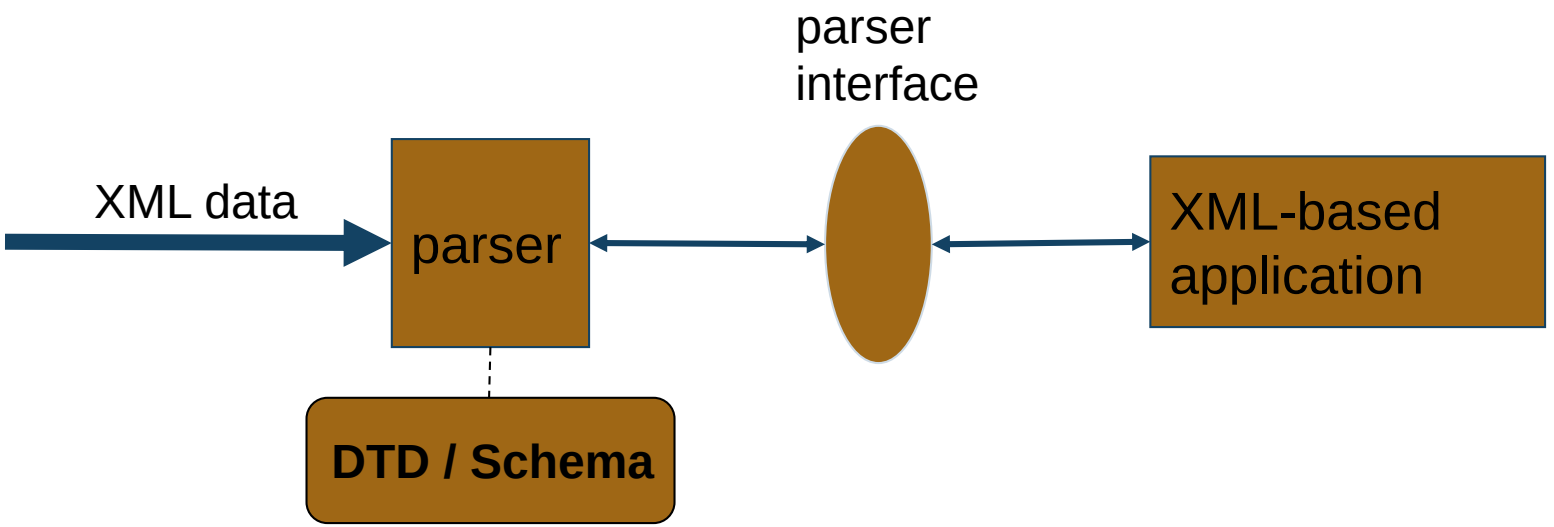
- Start-tag and End-tag, or self-closing tag
- Tags can't overlap
- Each XML document has **exactly one single** root element

```
<parent>
  <child>content</child>
  <child attribute="att"/>
</parent>
```

- **&**, **<**, **>** are represented by **&** **<** **>**

XML Validation (cont'd)

- Document Type Definition (DTD) or XML Schema
  - A language to define document type
    - The rules of the structure of XML
  - *Internal* or *External*



DTD (Document Type Definition)

- DTD is a set of structural rules called declarations
- DTD specifies:
  - A set of elements and attributes that can be in XML
  - Where these elements and attributes may appear
- **<!ELEMENT>**: to define tags
  - For leaf nodes: Character pattern
  - For internal nodes: List of children
- **<!ATTLIST>** : to define tag attributes
  - Includes: name of the element, the attribute's name, its type, and a default option

Example: External DTD

sample.dtd

```
<!ELEMENT note (to+,from, heading*,main)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT main (#PCDATA)>
```

from and ***main*** must occur once, and only once inside the "note" element

external DTDs are identified by the keyword SYSTEM

external-dtd.xml

```
<?xml version="1.0" ?>
<!DOCTYPE note SYSTEM "sample.dtd" >
<note>
        <to>Ali</to>
        <to>Taghi</to>
        <from>Naghi</from>
        <main>This is message</main>
</note>
```

ELEMENT Declaration

- General form of internal nodes
  - `<!ELEMENT` element_name `(`list of children`)>`
  - To control the number of times a child may appear
    - `+` : One or more
    - `*` : Zero or more
    - `?` : Zero or one

ELEMENT Declaration

- General form of leaf nodes
  - **<!ELEMENT** element_name **(#**type**)>**
  - Where, types
    - **PCDATA**: Most commonly used, the content will be parsed,
      - i.e. **< > &** is not allowed
    - **ANY**: Any character can be used (i.e., CDATA)
    - **EMPTY**: No content

ATTLIST Declaration

- **<!ATTLIST**

  element_name attribute_name attribute_type attr_value**>**

- *element_name*: The name of the corresponding element
- *attribute_name*: The name of attribute
- *attribute_type*: Commonly:

CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id

- *attr_value*:
  - A value: The default value of the attribute
  - **#REQUIRED**: The attribute is mandatory
  - **#IMPLIED**: The attribute is optional

Examples

<!ATTLIST element attr_name attr_type attr_value>

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type="check" />
```

```
<!ATTLIST TV id ID #REQUIRED>
<!ATTLIST TV name CDATA #REQUIRED>
<!ATTLIST PROGRAM VTR CDATA #IMPLIED>
```

XML Schema

- XML Schema describes the structure of an XML file
  - Also referred to as XML Schema Definition (XSD)
- Similar to OOP
  - Schema is a class & XML files are instances
  - Schema specifies
    - Elements and attributes, where and how often
    - Data type of every element and attribute

XSD vs DTD

XML Schemas benefits (DTD disadvantages):

- Created using basic XML syntax
  - DTD has its own syntax
- Supports built-in and user-defined data types
  - DTD does not fully support data type

Schema (cont'd)

- XML schema is itself an XML-based language
  - Has its own predefined tags & namespace

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

- Two categories of data types
  - *Simple*: Cannot have nested elements or attribute (i.e., itself is a leaf or attribute)
    - Primitive: `string`, `Boolean`, `integer`, `float`, ...
    - Derived: `byte`, `long`, `unsignedInt`, ...
    - User defined: restriction of base types
  - *Complex*: Can have attribute or/and nested elements

XML Schema Example: note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <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="date" type="xs:date"/>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

```
<note>
        <to>Ali</to>
        <to>Taghi</to>
        <from>Naghi</from>
        <main>This is message</main>
</note>
```

XML Schema (cont'd)

- Simple element declaration

```
<xs:element name="a name" type="a type" />
<xs:attribute name="a name" type="a type" />
```

- Complex element declaration

```
<xs:element name="a name">
  <xs:complexType>
    <xs:sequence> or <xs:all> or <xs:choice>
       <xs:element name
            minOccurs="..." maxOccurs="..."/>
    </xs:sequence> or </xs:all> or </xs:choice>
  </xs:complexType>
</xs:element>
```

`maxOccurs="unbounded"` : an unlimited number of times

XML Schema Example: note.xml

```
<?xml version="1.0"?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="note.xsd">

  <to>Ali</to>
  <from>Reza</from>
  <date>1391/1/1 </date>
</note>
```

XML Tools

- XML libraries
- Java XML libraries
  - For parsing (DOM & SAX)
  - For validating (DTD & XSD)
  - Utility technologies
    - E.g., XMLBeans, …

Exercise

- UNIVERSITY example
  - Instructors
  - Students
  - Classes
- Write an XML example
  - Write an XPath query for retrieving instructor with name="Aliakbary"
- Write an appropriate XSD for describing this XML

References

- W3Schools Online Web Tutorials
  http://www.w3schools.com/
- TutorialsPoint
  www.**tutorialspoint**.com/
- Internet Engineering course, Amirkabir University of Technology, Dr. Bahador Bakhshi
  http://ceit.aut.ac.ir/~bakhshis/

The End