# XML and XSL

Mohammad Homayoon Fayez
Zealand Institute of Business and Technology

# XML

- XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed W3C in 1996.
- The design goals for XML:
  - XML shall be straightforwardly usable over the Internet.
  - XML shall support a wide variety of applications.
  - XML shall be compatible with SGML.
  - It shall be easy to write programs which process XML documents.
  - The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
  - XML documents should be human-legible and reasonably clear.
  - The XML design should be prepared quickly.
  - The design of XML shall be formal and concise.
  - XML documents shall be easy to create.
  - Terseness in XML markup is of minimal importance.

# XML (Extensible Markup Language)

- Used for the exchange of data on the web
- Describes a class of data objects called **XML Documents**
- A data object (textual object) is an XML document if it meets the well-formedness constraints described in W3C's XML specifications.(see e.g. version 1.1 http://www.w3.org/TR/xml11/ )
- It is considered well-formed if, for example
  - It contains one or more elements
  - There is exactly one root or document element
  - The elements delimited by start and end-tags nest properly
- It is valid if
  - It has an associated document type declaration and if the document complies with the constraints expressed in it
- Documents must begin with an XML declaration which specifies the version of XML being used `<?xml version="1.1"?>`

# XML Element

- An element has a type, identified by name
- An element may  have attributes
- Each attribute has a name and a value
- The boundaries of an element are either delimited by start-tag(<>) and end-tag(</>) or, for empty elements, by an empty-element tag.
- The text between the start-tag and end-tag is called the element's content
- An element with no content is said to be empty
  - Start- and end-tag example:

```
start-tag:   <student type="international">
                        //some content
end-tag:     </student>
```

  - Empty-element tag example:

```
<IMG src="http://www.mofa-easj.dk/img/logo.png"/>
<br></br>
<br/>
For more read the specification at: http://www.w3.org
```

# Styling XML

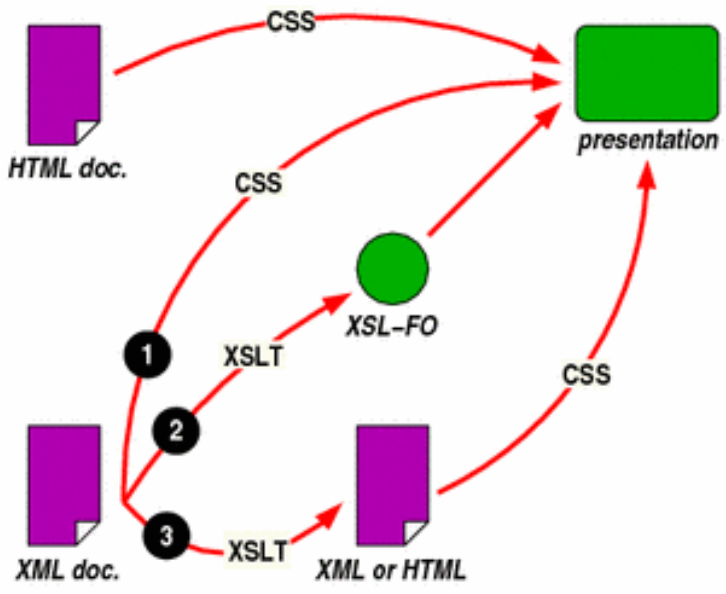
- You can present xml to the user but it is not such a good idea.
- You can make it more presentable, human-legible (one of the design goals. Remember?), by styling it
- Two choices
    - CSS : Simple but has limitations; Only styling
    - XSL : Styling and Transformation
    - Use css when you can
    - Use XSL when you must. E.g. when
        - something needs transformation, for example a list where some of its words have to be replaced by other words
        - empty elements to be replaced by some text

# XSL

- XSL is an XML application
- Consists of three parts
  - XSLT
    - the transformation engine
  - XPath
    - Models an XML document as a tree of nodes (elements, attributes, text, etc.)
    - Finds specific elements in the XML tree of nodes
  - XSL-FO
    - XSL Formatting Objects (a sub-language of XSL)
    - Describes a printable page with text
    - Uses all the CSS concepts and more, but in XML

# XSL

1. Use CSS, if you don't want to transform the document
2. Use XSL-T (If the document have to be transformed)
   - Take the Original XML file
   - Transform it into the XSL-FO
   - Present the result on a screen or printer
3. OR
   - Generate a new XML or HTML document
   - Style it with CSS
   - Present the result




http://www.w3.org/Style/CSS-vs-XSL.en.html

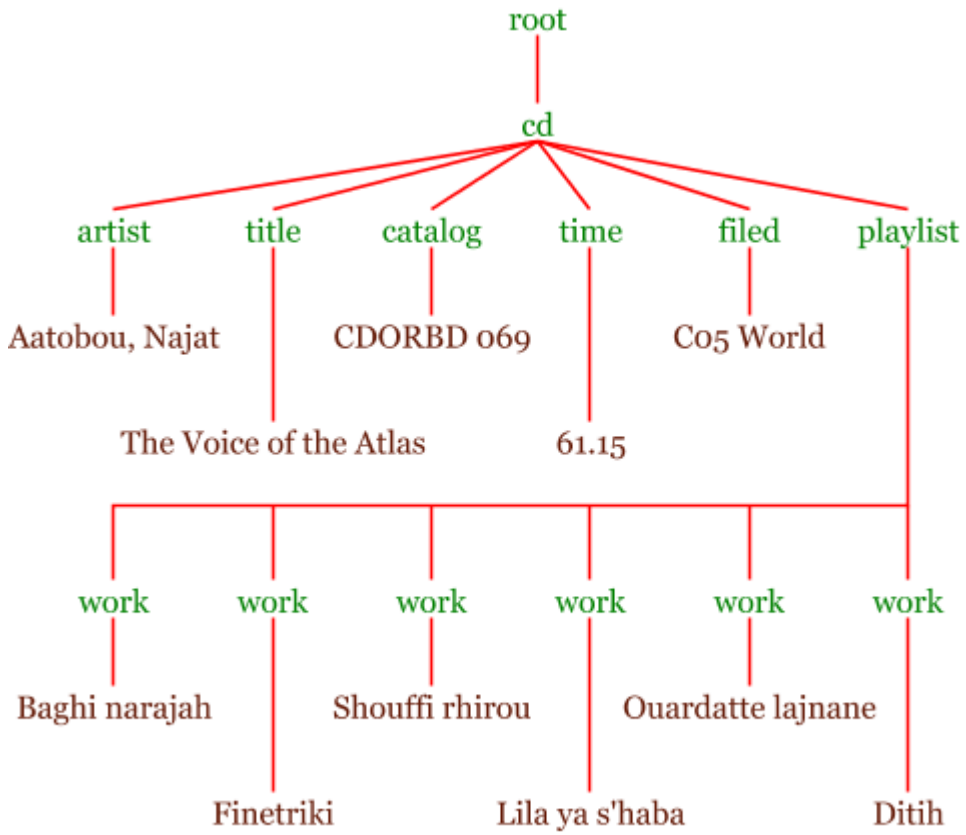
# XSLT + XPath

- A general transformation engine
- Content adaptation (different presentation of same XML document)
- One of the most widely used XML tools
- Many different implementation
  - Built in IE6, Microsoft Edge, Mozilla
  - Stand alone processors
    - Saxton, Xt, FOP, iXSLT etc.

# Extract from a CD collection

```
<cd>
    <artist>Aatabou, Najat</artist>
    <title>The Voice of the
    Atlas</title>
    <label/>
    <catalog>CDORBD 069</catalog>
    <time>61.15</time>
    <filed>C05 World</filed>
    <playlist>
        <work>Baghi narajah</work>
        <work>Finetriki</work>
        <work>Shouffi rhirou</work>
        <work>Lila ya s'haba</work>
        <work>Ouardatte lajnane</work>
        <work>Ditih</work>
    </playlist>
</cd>
```

XML content as a tree




source: http://www.w3.org/Consortium/Offices/Presentations/XSLT_XPATH/#[8]

## XSLT Structure

- XSLT transforms an XML source tree into an XML Result tree, using templates
- A simple template example

```
<xsl:template match="/students">
    <html>
        …
    </html>
    </xsl:template>
```

- This template processes the student element and generates an html document for the student

## XSLT Transformation

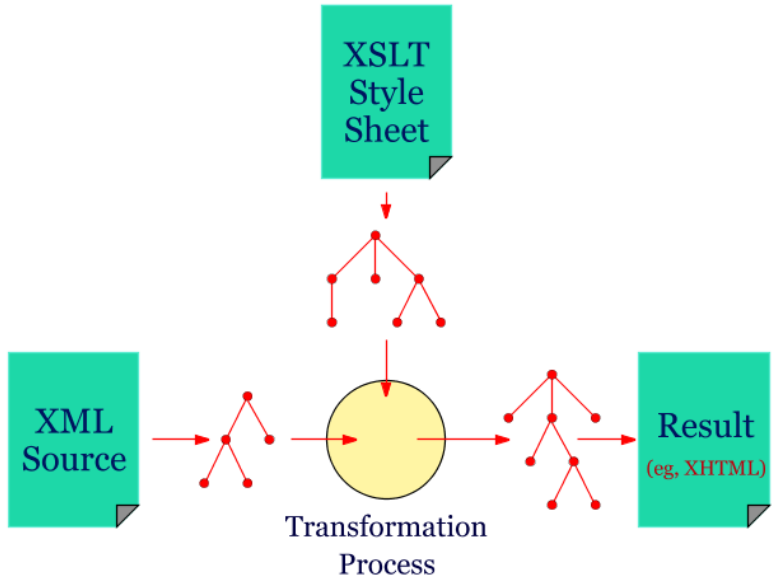



Image source: http://www.w3.org/Consortium/Offices/Presentations/XSLT_XPATH/#(9)

# Template Body

The template body contains XSLT instructions

- Processes all student nodes
- choose (conditional testing )Choose country nodes
- test tests the content of the element ‘country’
- when (IF) country node text matches ‘Danmark’
  - Generate a <tr> (table row) and a <td> (table cell) containing the name and country and
  - Change the background color of the entire row to #ff00ff
- otherwise (Else)
  - Generate name and type of the student
  - Set the background color to #ffffff (white)

Extract from a XSLT template

```
<xsl:for-each select="student">
<xsl:sort select="name"/>
<xsl:choose>
    <xsl:when test="country='Denmark'">
       <tr bgcolor="#ff00ff">
           <td><xsl:value-of select="name"/></td>
           <td><xsl:value-of select="country"/></td>
       </tr>
    </xsl:when>
    <xsl:otherwise>
       <tr bgcolor="#ffffff">
          <td><xsl:value-of select="name"/></td>
          <td><xsl:value-of select="@type"/></td>
       </tr>
     </xsl:otherwise>
</xsl:choose>
</xsl:for-each>
```

# JSON

- Javascript Object Notation
- Light weight data interchange format
- human-legible and clear
- Easy for machine to parse and generate
- supported by all modern programming languages.
- Built on two structures
  - A collection of name/value pairs
  - An ordered list of values

# JSON

- **A collection of Name-value pairs:**
  - is realized as an object
  - begins with { and ends with }
  - Each name is followed by (:)
  - each name-value pair is separated by (,)

```
{"firstName": "John", "lastName": "Smith", "age": 25}
```

- **An Ordered list of values**
  - is realized as an array.

```
{"phoneNumber":
    [{
      "type": "home",
      "number": "21255512"
    },
    {
       "type": "fax",
       "number": "64655545"
    }]
 }
```