

# XML Technologies

CSE 794R/ECE 694R

1

## XML: eXtensible Markup Language

- Markup language made up of nested tags
- Similar in structure to HTML but much more general
- HTML limitations:
  - fixed tags
  - mixes presentation and content
- XML focuses on structure/data
  - different applications need different structure

CSE 794R/ECE 694R

2

# Elements

- Surrounded by tags
- May contain other elements
- May contain content (text)
- May have associated attributes
- Hierarchical: form a TREE
  - i.e., ONE root element

CSE 794R/ECE 694R

3

# Tags

- Tag-syntax:
  - Usually come in pairs: open/close
    - `<contents>...</contents>`
  - Can have attributes
    - `<chapter title="Hello">`
  - Can include text
    - `<title version="3">Mission: Impossible</title>`
  - Short form of open/close for empty tag
    - ``

CSE 794R/ECE 694R

4

# Attributes

- Name/value pair: name="value"
- Names follow same rules as element names
- Values inside quotes
- Can NOT repeat
- Choice between attributes & elements:

## **attributes**

single-valued

metadata

flat

## **elements**

multi-valued

data

nesting structure

# Other Pieces

- Entity references
  - &xxx; to represent <, >, &, ", ' (lt, gt, amp, quot, apos)
  - can be user-defined too
- CDATA sections (unparsed data)
  - to include raw text (without translating <'s &'s etc)
  - <![CDATA[ raw text here ]]>
- Comments
  - <!-- -->
  - don't use -- inside!
  - not inside tags

# Processing Instructions

- XML provides the *processing instruction* as an alternative means of passing information to particular applications that may read the document.
- Processing instruction begins with `<?` and ends with `?>`.
- Immediately following the `<?` is an XML name called the target, possibly the name of the application for which this processing instruction is intended or possibly just an identifier for this particular processing instruction.
- The rest of the processing instruction contains text in a format appropriate for the applications for which the instruction is intended.

CSE 794R/ECE 694R

7

# Examples

```
<?robots index="yes" follow="no"?>

<?php
mysql_connect("database.unc.edu", "clerk", "password");
$result = mysql("HR", "SELECT LastName, FirstName FROM
Employees
ORDER BY LastName, FirstName");
$i = 0;
while ($i < mysql_numrows ($result)) {
    $fields = mysql_fetch_row($result);
    echo "<person>$fields[1] $fields[0] </person>\r\n";
    $i++;
}
mysql_close( );
?>
```

CSE 794R/ECE 694R

8

# Document Structure

- Two parts: prolog & body
  - prolog – declarations, e.g.,
    - `<?xml version="1.0" ?>`
    - `<?xml-stylesheet type="text/xsl" href="main.xsl"?>`
    - `<!DOCTYPE Main SYSTEM "dtd/Main.dtd">`
    - `<!DOCTYPE Main PUBLIC "http://www. etc ">`
  - body - elements

# Well-Formed XML

- XML Documents must follow number of rules, including
  - Elements may nest but may not overlap.
  - There must be exactly one root element.
  - Attribute values must be quoted.
  - An element may not have two attributes with the same name.
  - Comments and processing instructions may not appear inside tags.
  - No unescaped `<` or `&` signs may occur in the character data of an element or attribute.
- All XML documents **MUST** be well-formed
  - HTML need not be well-formed (well-formed HTML called XHTML)
- Well-formed is usually not enough...

## Valid XML

- DTD and XML Schema define the specific elements, nesting, attributes, etc. for a given XML application
- XML documents that conform to DTD or XML schema are *valid*
- Valid implies well-formed, but not vice-versa
- If DTD/XML Schema is unavailable, cannot validate (not invalid)

CSE 794R/ECE 694R

11

## DTD: Document Type Definition

- The DTD defines
  - how data is formatted
  - each allowed element in an XML document
  - the allowed attributes and possibly the acceptable attribute values for each element
  - the nesting and occurrences of each element, and any external entities.

CSE 794R/ECE 694R

12

# Element Declaration

- `<!ELEMENT elt_name (content_model)>`
- `content_model` can be:
  - #PCDATA ("Parsed Character DATA")  
e.g., `<name> yadda yadda </name>`
  - EMPTY e.g., `<img source="sfdsf"> </img>`
  - nested—three forms of nested element declaration:
    - sequence ( , )
    - choice ( | )
    - number suffix ( )? ( )\* ( )+can be nested and combined!
- `<!ELEMENT polygon (((x,y) | (r,t)), ((x,y) | (r,t)), ((x,y) | (r,t))+ )>`
  - three or more points
  - each point either cartesian or polar

CSE 794R/ECE 694R

13

# Attribute Declaration

- `<!ATTLIST elt_name att_name att_type modifier>`
- `att_type` can be:
  - CDATA, character data
  - NMTOKEN, like XML Name, but can start with any of the valid chars (no whitespace)
  - ID, an XML Name (not NMTOKEN), so can't start with number!
    - `_unique_` within document (among ID attributes)
  - IDREF, some other element's ID
  - enumeration, each must be a valid XML Name ( | | | )
  - notice no "int" or "date" in list
  - could use NMTOKEN to prevent white space
- modifiers:
  - #IMPLIED--optional
  - #REQUIRED--must be present (no default provided)
  - #FIXED--constant (if not present, implicit; if present must be equal)
  - Literal string--default value if not included

CSE 794R/ECE 694R

14

# Entity Declarations

- XML has standard entity references
  - &lt; &amp; &gt; &quot; &apos;
- Can define your own in DTD
  - `<!ENTITY ent_name "ent_value">`
  - defines `&ent_name;` to be `ent_value` (in XML document)
  - `ent_value` can contain anything! (i.e., markup as well as text)  
e.g., `<!ENTITY pb "<b>Paolo Bucci</b>">`
  - can include other entities! (no cycles allowed, and must be balanced open/close, i.e., can't open tag in one entity and close in another)
  - can also point to external definition for entity  
e.g., `<!ENTITY pb_home SYSTEM "http://www.cse.ohio-state.edu/~bucci">`
  - now `&pb_home;` is replaced by document at that URI
  - tricky here (different parsers handle this differently)
  - safer to use JSP or server-side include

CSE 794R/ECE 694R

15

# Structuring

- Can include DTD in the document itself

```
<!DOCTYPE personnel [  
  <!ELEMENT person ... >  
>
```
- or refer to external document

```
<!DOCTYPE personnel SYSTEM "personal.dtd">  
<!DOCTYPE personnel PUBLIC "http://www. etc ">
```
- Can mix internal and external

CSE 794R/ECE 694R

16

# Namespaces

- Namespaces have two purposes in XML:
  - To distinguish between elements and attributes from different vocabularies with different meanings that happen to share the same name
  - To group all the related elements and attributes from a single XML application together so that software can easily recognize them

CSE 794R/ECE 694R

17

# Overview

- Namespaces are implemented by attaching a prefix to each element and attribute.
- Each prefix is mapped to a URI by an `xmlns:prefix` attribute.
- Default URIs can also be provided for elements that don't have a prefix.
- Default namespaces are declared by `xmlns` attributes.
- Elements and attributes that are attached to the same URI are in the same namespace.
- Elements from many XML applications are identified by standard URIs.

CSE 794R/ECE 694R

18

# NameSpaces

- Potential for name conflicts (tags, attributes)
- Solution: add a prefix to distinguish (scoping)  
`<namespace>: <name>`
- To identify namespace, use URI ("uniform resource identifier", superset of URL)
- Instead of `<class> ... </class>` have  
`<__ :class> ... </__ :class>`
- Qualified name (qname): `prefix:local name`
  - can't use ':' in prefix or in local name

CSE 794R/ECE 694R

19

# Binding

- Problem: URIs could look like  
`http://cse.ohio-state.edu/~bucci/794`
  - \*/ and ~ not legal in XML names!
- Solution:
  - use a short prefix which is bound to a URI!
  - to bind a prefix *prf*, use attribute `xmlns:prf`, e.g.,  

```
<prf:class
  xmlns:prf="http://cse.ohio-state.edu/~bucci/794">
  <prf:prof> ... </prf:prof>
  <prf:___> etc </prf:___>
</prf:class>
```
- `xmlns:prf` introduces prefix *prf*, valid for that subtree (INCLUDING root!)

CSE 794R/ECE 694R

20

# URIs

- Some common URIs:
  - xml—<http://www.w3.org/XML/1998/namespace>
  - xsl—<http://www.w3.org/1999/XSL/Transform>
  - xs—<http://www.w3.org/2001/XMLSchema>
- Namespace URIs do not necessarily point to any actual document or page

# Default Namespace

- Created with `<tag xmlns="...">`
  - applies to this and all descendent UNPREFIXED elements
  - does NOT apply to attributes, so you sometimes see

```
<tag xmlns="URI1" xmlns:foo="URI1">
```

*foo* is used to mark attributes

## Namespaces & DTDs

- Validity & namespaces are independent, i.e., validation treats a QName (pre:ln) just like any other
  - must match DTD declaration exactly
  - doesn't understand URI part
- Good practice:
  - put xmlns declaration IN THE DTD (means you're stuck with a particular prefix though), or
  - use parameter entity references (see "XML in a Nutshell")

CSE 794R/ECE 694R

23

## XSLT

- eXtensible Stylesheet Language Transformations
- An application for transforming one XML document into another
- This transformation is controlled by another document
  - the "stylesheet"
  - contains "templates"

CSE 794R/ECE 694R

24

# Stylesheet

- Itself an XML document! (begin with `<?xml version="1.0"?>`)
- Root element: `stylesheet`
- Its elements are in namespace `"http://www.w3.org/1999/XSL/Transform"`
- `<xsl:stylesheet xmlns:xsl="http://...Transform">`  
...  
`</xsl:stylesheet>`

CSE 794R/ECE 694R

25

# XSLT Template Rules

- To control what output is created from what input, you add template rules to the XSLT stylesheet.
- Each template rule is represented by an `xsl:template` element. This element has a `match` attribute that contains a pattern identifying the input it matches; it also contains a template that is instantiated and output when the pattern is matched.
- The terminology is a little tricky here: the `xsl:template` element is a template rule that contains a template. An `xsl:template` element is not itself the template.
- For example:  
`<xsl:template match="person">A Person</xsl:template>`

CSE 794R/ECE 694R

26

## Some Templates

- `<xsl:template match="___">stuff</xsl:template>`
  - every instance of \_\_\_ element replaced with stuff
  - stuff must be well-formed xml
- `<xsl:value-of select="___"/>`
  - replaced by contents of \_\_\_ (incl whitespace)
  - can refer to elements and attributes  
`<xsl:value-of select="@___"/>`
- Often see these two nested

CSE 794R/ECE 694R

27

## Traversal Control

- Default traversal: top to bottom (parent before children)
- To change order:  
`<xsl:apply-templates select="___"/>`
- The template that matches \_\_\_ is expanded
- The order of the template rules in the stylesheet doesn't matter. It's only the order of the elements in the input document that matters.

CSE 794R/ECE 694R

28

## Conditional Transformation

- Can have
  - `<xsl:if test="boolean expression">`
    - if only, no else
  - `<xsl:choose>`
    - case/switch, multiple selection
    - `<xsl:when test = "boolean expression">`
    - `<xsl:otherwise>`
  - `<xsl:for-each select = "node-set-expression">`

## Final Thoughts

- Default Templates: strip markup and recurse down
- XSLT is Turing-complete!!

# XPath

- Way to specify a set of nodes
  - e.g., root, element, attribute, text, comment, PI, namespace
- Expression gives a set of nodes, e.g.,
  - / (root)
  - name (all elements called "name")
  - @born (all attributes called "born")
  - \* @\* (wildcards: all elements, all attributes)
  - name/first\_name
  - ../name/@born (element/attribute)
  - name/last\_name= 'Turing'
  - not(../name/@died='1954')

CSE 794R/ECE 694R

31

# SAX: Simple API for XML

- API for accessing and working with XML document
  - obtain a parser (e.g., Xerces from apache.org)
  - create an "XMLReader" with a factory
- Key idea: call-backs
  - implement a ContentHandler
  - as parser reads document, calls appropriate method in your ContentHandler
- org.xml.sax

CSE 794R/ECE 694R

32

# ContentHandler

```
public interface ContentHandler {  
    public void startDocument() throws SAXException;  
    public void endDocument() throws SAXException;  
  
    public void startElement(String nsURI, String lName,  
        String qName, Attributes atts) throws SAXException;  
    public void endElement(String nsURI, String lName,  
        String qName) throws SAXException;  
  
    public void characters(char ch[], int start, int length)  
        throws SAXException;  
  
    ...  
}
```

CSE 794R/ECE 694R

33

# Observations

- SAX parser goes through document once
- It builds the tree, but then throws it away!
- It is up to the programmer to store what is needed

CSE 794R/ECE 694R

34

## DOM: Document Object Model

- The DOM defines an API for accessing and manipulating XML documents as tree structures.
- The DOM is defined by a set of W3C Recommendations that describe a programming language-neutral object model used to store hierarchical documents in memory.
- Several versions called *levels*: DOM Level 1, DOM Level 2, DOM Level—progressively more features

CSE 794R/ECE 694R

35

## DOM cont.

- Build the entire tree FIRST
  - must wait for parsing to complete before you can start working with it
  - store in memory
  - return tree to programmer
- `org.w3c.dom`
- `javax.xml.parsers`

CSE 794R/ECE 694R

36

# Parsing

- DocumentBuilderFactory dbf =  
DocumentBuilderFactory.newInstance();
- DocumentBuilder db =  
dbf.newDocumentBuilder();
- Document d = db.parse(new File(fileName));
- Element e = d.getDocumentElement();
- Node n = (Node) e;

CSE 794R/ECE 694R

37

# Node

- Basic type of tree element: interface Node
  - has a "type" field, one of: DOCUMENT\_NODE, ELEMENT\_NODE, ATTRIBUTE\_NODE, TEXT\_NODE, etc...
  - has a name (string), value (string), children (NodeList)
  - can get/set NodeValue(), get/remove/insert/append children nodes
- Different types have inherited interfaces
  - interface Element (directly supports working with contained attributes)
  - interface Attr
  - interface Text (elements with text data have a child node in tree of type TEXT\_NODE)
  - interface Document

CSE 794R/ECE 694R

38

## SAX vs. DOM

- Event-based parsers (SAX)
  - Low memory usage
  - Application can process data during parsing
  - Harder to access data scattered through doc
- Tree-based parsers (DOM)
  - High memory usage
  - Application must wait for parse completion
  - Easier to access data