XML Overview

- XML is a meta-language, a simplified form of SGML (Standard Generalized Markup Language)
- XML was initiated in large parts by Jon Bosak of Sun Microsystems, Inc., through a W3C working group
- References:
  3. “XML Schema Tutorial” by Roger L. Costello
     http://www.xfront.com/
XML Overview (cont.)

References:
4. XML namespaces http://www.w3.org/TR/1999/REC-xml-names-19990114/
5. XML specifications http://www.w3.org/TR/REC-xml/
6. XML Schema Part 0: Primer http://www.w3.org/TR/xmlschema-0/
9. WWW Consortium XML Site: http://www.w3.org/XML/

An XML compliant application generally needs three files to display XML content:

- The XML document
  - Contains the data tagged with meaningful XML elements

- A document type definition – DTD or Schema
  - Specifies the rules how elements and attributes are logically related

- A stylesheet
  - Dictates the formatting when the XML document is displayed. Examples: CSS - cascading style sheets, XSL - extensible stylesheet language
XML Terminology

- **Element, e.g.,**:
  ```xml
  <Body>
  This is text formatted according to the body element
  </Body>
  ```
- **An element consists always of two tags:**
  - An opening tag, e.g., `<Body>`
  - A closing tag, e.g., `</Body>`
- **An element can have attributes, e.g.,**:
  ```xml
  <Price currency="Euro">25.43</Price>
  ```
- **Attribute values must always be in quotes (unlike HTML)**

A Simple XML Document

- **Example: Book description**

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE BOOKCATALOG SYSTEM "http://tt.com/bookcatalog.dtd">
<!-- Here begins the XML data -->
<book>
  <title>The spy who came in from the cold</title>
  <author>John <lastname>Le Carre</lastname></author>
  <price currency="USD">5.59</price>
  <review><author>Ben</author>Perhaps one of the finest...</review>
  <review><author>Jerry</author>An intriguing tale of...</review>
  <bestseller authority="NY Times"/>
</book>
```
A Simple XML Document

- **Markup:** Text delimited by angle brackets (<...>)
- **Character Data:** the rest
- **Element names are not unique**
  - (e.g., two `<review>`)
- **Attribute names are unique within an element**
  - (e.g., one “currency” attribute in price)
- **Elements can be empty and hence presented concisely**
  - (e.g., `<bestseller></bestseller>` = `<bestseller/>`)
- **An XML document is well-formed if it satisfies simple syntactic constraints**

---

A Simple Document Type Definition

- **Example DTD**

```xml
<!ELEMENT book (title, author+, price, review*,
  bestseller?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author
  (#PCDATA|lastname|firstname| fullname)*)
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA "USD"
  source (list|regular|sale) list
  taxed CDATA #FIXED "yes">
<!ELEMENT bestseller EMPTY>
<!ATTLIST bestseller authority CDATA #REQUIRED>
```
The DTD Language (0)

- Example DTD

```xml
<!ELEMENT book (title, author+, price, review*, bestseller?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA|lastname|firstname|fullname)*>
<!ELEMENT price (#PCDATA)>
```

- Required child elements for book element:
  - title, author, price
- #PCDATA: parsed character data

The DTD Language (1)

- An XML compliant document is composed of elements:
  - Simple elements
    ```xml
    <!ELEMENT title ANY>
    ```
    - The element can contain valid tags and character data
    ```xml
    <!ELEMENT title (#PCDATA)>
    ```
    - The element cannot contain tags, only character data
  - Nested elements
    ```xml
    <!ELEMENT book (title)>
    <!ELEMENT title (#PCDATA)>
    ```
_nested and ordered elements:

```xml
<!ELEMENT books (title, author)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (#PCDATA)>  
```

- The order of the elements must be title, then author

_nested either-or elements:

```xml
<!ELEMENT books (title|author)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT authors (#PCDATA)>  
```

- There must be either a title or an author element, but not both.

---

**Grouping and recurrence:**

```xml
<!ELEMENT book (title, author+, price, review*, bestseller?)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (#PCDATA)>  
```

- 0 or 1 time
- 1 or more times
- 0 or more times

- declaration requires every book element to have a price sub-element
- the use of some element names (e.g., review, lastname) without a corresponding declaration is not an error; such elements are simply not constrained by this DTD
The DTD Language (4)

- Inside a DTD we can declare an entity which allows us to use an entity reference to substitute a series of characters, similar to macros.
  - Format:
    ```xml
    <!ENTITY name "replacement_characters">
    ```
  - Example for the © symbol:
    ```xml
    <!ENTITY copyright "&xA9;">  
    ```
  - Usage: entities must be prefixed with ‘&’ and followed by a semicolon (‘;’):
    ```xml
    <copyright>
    &copyright; 2000 MyCompany, Inc.
    </copyright>
    ```

The DTD Language (5)

- Parameter entity references appear only within a DTD and cannot be used in an XML document. They are prefixed with a %.
  - Format and usage:
    ```xml
    <!ENTITY % name "replacement_characters">
    ```
  - Example:
    ```xml
    <!ENTITY % pcdata "(#PCDATA)">  
    <!ENTITY authortitle %pcdata;>
    ```
The DTD Language (6)

- **External entities** allow us to include data from another XML document (think of an `#include<...>` statement in C):
  
  - Format and usage:
    
    ```
    <!ENTITY quotes SYSTEM "http://www.stocks.com/quotes.xml">
    
    Example:
    
    <document>
    <heading>Current stock quotes</heading>
    &quotes; <!-- data from quotes.xml -->
    </document>
    ```
  
  - **Works well for the inclusion of dynamic data.**

The DTD Language (7)

- **Attribute** declarations in the DTD. Attributes for various XML elements must be specified in the DTD.
  
  - Format and usage:
    
    ```
    <!ATTLIST target_element attr_name attr_type default>
    
    Examples:
    
    <!ATTLIST box length CDATA "0">  
    <!ATTLIST box width CDATA "0">  
    <!ATTLIST frame visible (true|false) "true">  
    
    <!ATTLIST person marital (single | married | divorced | widowed) #IMPLIED>
    ```
The DTD Language (8)

- Examples:

```xml
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA "USD"
  source (list|regular|sale) "list"
  taxed CDATA #FIXED "yes">
<!ELEMENT bestseller EMPTY>
<!ATTLIST bestseller authority CDATA #REQUIRED>
```

- Currency, of type character data, default USD
- Source, of one of the three enumerated types, default list
- Taxed, with the fixed value yes
- Fixed attribute type is a special case of default
  - It determines that the default value cannot be changed by an XML document conforming to the DTD
  - E.g., a book in our XML example must be taxed

The DTD Language (9)

- Default modifiers in DTD attributes:

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#REQUIRED</td>
<td>The attributes value must be specified with the element.</td>
</tr>
<tr>
<td>#IMPLIED</td>
<td>The attribute value can remain unspecified.</td>
</tr>
<tr>
<td>#FIXED</td>
<td>The attribute value is fixed and cannot be changed by the user.</td>
</tr>
</tbody>
</table>
The DTD Language (10)

**Datatypes in DTD attributes:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDATA</td>
<td>Character data</td>
</tr>
<tr>
<td>enumerated</td>
<td>A series of values of which only 1 can be chosen</td>
</tr>
<tr>
<td>ENTITY</td>
<td>An entity declared in the DTD</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>Multiple whitespace separated entities declared in the DTD</td>
</tr>
<tr>
<td>ID</td>
<td>A unique element identifier</td>
</tr>
<tr>
<td>IDREF</td>
<td>The value of a unique ID type attribute</td>
</tr>
<tr>
<td>IDREFS</td>
<td>Multiple whitespace separated IDREFs of elements</td>
</tr>
<tr>
<td>NM_TOKEN</td>
<td>An XML name token</td>
</tr>
<tr>
<td>NM_TOKENS</td>
<td>Multiple whitespace separated XML name tokens</td>
</tr>
<tr>
<td>NOTATION</td>
<td>A notation declared in the DTD</td>
</tr>
</tbody>
</table>

The DTD Language (11)

**Example: Sales Order Document**

“An order document is comprised of several sales orders. Each individual order has a number and it contains the customer information, the date when the order was received, and the items ordered. Each customer has a number, a name, street, city, state, and ZIP code. Each item has an item number, parts information and a quantity. The parts information contains a number, a description of the product and its unit price.

The numbers should be treated as attributes.”
The DTD Language (12)

- Example: Sales Order Document DTD

```xml
<!-- DTD for example sales order document -->
<!ELEMENT Orders (SalesOrder+)>
<!ELEMENT SalesOrder (Customer,OrderDate,Item+)>
<!ELEMENT Customer
  (CustName,Street,City,State,ZIP)>  
<!ELEMENT OrderDate (#PCDATA)>     
<!ELEMENT Item (Part,Quantity)>     
<!ELEMENT Part (Description,Price)> 
<!ELEMENT CustName (#PCDATA)>      
<!ELEMENT Street (#PCDATA)>        
<!ELEMENT ... (#PCDATA)>           
<!ATTLIST SalesOrder SONumber CDATA #REQUIRED>    
<!ATTLIST Customer CustNumber CDATA #REQUIRED>     
<!ATTLIST Part PartNumber CDATA #REQUIRED>         
<!ATTLIST Item ItemNumber CDATA #REQUIRED>         
```

The DTD Language (13)

- Example: Sales Order XML Document

```xml
<Orders>
  <SalesOrder SONumber="12345">
    <Customer CustNumber="543">
      <CustName>ABC Industries</CustName>
      <Street>123 Main St.</Street>
      <City>Chicago</City>
      <State>IL</State> <ZIP>60609</ZIP>
    </Customer>
    <OrderDate>10222000</OrderDate>
    <Item ItemNumber="1">
      <Part PartNumber="234">
        <Description>Turkey wrench</Description>
        <Price>9.95</Price>
      </Part>
      <Quantity>10</Quantity>
    </Item>
  </SalesOrder>
</Orders>
```
The DTD Language (14)

- An XML document that satisfies the constraints of a DTD is said to be valid with respect to that DTD.
- *Document type declaration* (at the “prolog” of an XML document):
  ```xml
  <!DOCTYPE BOOKCATALOG SYSTEM "http://tt.com/bookcatalog.dtd">
  ```
- XML document claims validity with respect to the BOOKCATALOG DTD

The remainder is only for your reference…
Extensible Stylesheet Language (XSL)

- XSL is a language for transforming and formatting XML.
- Recently, the transformation and formatting parts of XSL were separated.
- Here, we focus on the XSL transformation language, called XSLT.
- An XSLT stylesheet is a collection of transformation rules that operate (non-destructively) on a source XML document (source tree) to produce a new XML document (result tree).
- Each rule consists of a pattern and a template:
  - Patterns matched against nodes of source tree
  - Templates instantiated to produce part of result tree

Example XSL

```xml
<xsl:stylesheet version= "1.0"
  xmlns:xsl="http://w3.org/XSL/Transform/1.0"
  xmlns="http://w3.org/1999/XSL/Transform"
  indent-result="yes">

  <xsl:template match="*">
    <!-- Your transformation rules here -->
  </xsl:template>

</xsl:stylesheet>
```

- Declare the XSL and XHTML namespaces used by the stylesheet.
- The XHTML namespace is made the default namespace.
Example XSL …

- Each template element describes one transformation rule
- The match attribute of a template element specifies the rule pattern while its content is the template used to produce the corresponding portion of the result tree

<!-- Rule 1 --> <xsl:template match="/">
    <html><head><title>Our New Catalog</title></head>
    <body>
        <xsl:apply-templates/>
    </body>
</html>
</xsl:template>

- The pattern "/" denotes the root of the source tree
- The template contains some standard XHTML header and trailer constructs
- The apply-templates element is a rule-processing instruction that denotes recursive processing of the contents of the matched element
- XSLT includes several other instructions which permit templates with constructs such as for-loops, conditional sections, and sorting

Example XSL …

<!-- Rule 2 --> <xsl:template match="book/title">
    <h1><xsl:apply-templates/></h1>
</xsl:template>

<!-- Rule 3 --> <xsl:template match="book/author">
    <b><xsl:apply-templates/></b>
</xsl:template>

- Pattern, "book/title" matches a title element if its parent is a book element
- The template calls for recursive processing of the contents, enclosed in XHTML literals for bold display (<b>...<b>)
- XSL processing includes implicit rules that match elements, attributes, and character data (text) not matched by any explicit rules; these rules simply copy data from source to result tree
- In our example, all character data (such as the text "The spy..." in the title) is copied to the result tree
XSL Example …

<!-- Rule 4 --> <xsl:template match="book/price">
  <xsl:apply-templates/>
  <xsl:apply-templates select="@*"/>
</xsl:template>

- An additional apply-template instruction to extract the currency attribute using the syntax @*

<!-- Rule 5 --> <xsl:template match="book/review[1]" priority="1.0">
  <xsl:apply-templates/>
</xsl:template>

- Matches only the first review element in each book element due to the "[1]" specification
- The template simply copies the contents to the result tree (using recursive processing with apply-templates combined with the default rules)

<!-- Rule 6 --> <xsl:template match="book/review" priority="0.5">
</xsl:template>
</xsl:stylesheet>

- Includes only the first review for each book:
- We ensure that the first review for each book is processed using Rule 5 instead of Rule 6 by assigning Rule 5 a higher priority
XML and Databases (1)

- “Is XML a database?”
- In a strict sense, no.
- In a more liberal sense, yes, but …
  - XML has:
    - Storage (the XML document)
    - A schema (DTD)
    - Query languages (XQL, XML-QL, …)
    - Programming interfaces (SAX, DOM)
  - XML lacks:
    - Efficient storage, indexes, security, transactions, multi-user access, triggers, queries across multiple documents

XML and Databases (2)

- Data versus Documents
  - There are two ways to use XML in a database environment:
    - Use XML as a data transport, i.e., to get data in and out of the database
      - Data is stored in a relational or object-oriented database
      - Middleware converts between the database and XML
    - Use a “native XML” database, i.e., store data in document form
      - Use a content management system
XML and Databases (3)

- **Data-centric documents**
  - Fairly regular structure
  - Fine-grained data
  - Little or no mixed content
  - Order of sibling elements often not significant

- **Document-centric documents**
  - Irregular structure
  - Larger-grained data
  - Lots of mixed content
  - Order of sibling elements is significant

XML and Databases (4)

- **Data-centric storage and retrieval systems**
  - Use a database
    - Add middleware to convert to/from XML
  - Use an XML server (specialized product for e-commerce)
  - Use an XML-enabled web server with a database backend

- **Document-centric storage and retrieval systems**
  - Content management system
  - Persistent DOM implementation
XML and Databases (5)

- Mapping document structure to database structure
  - Template-driven
    - No predefined mapping
    - Embedded commands process (retrieve) data
    - Currently only available from RDBMS to XML (unidirectional)

```xml
<?xml version="1.0"?>
<FlightInfo>
  <Intro>The following flights have available seats:</Intro>
  <SelectStmt>SELECT Airline, FltNumber, Depart, Arrive FROM Flights</SelectStmt>
  <Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>
```

XML and Databases (6)

- Template-driven - Example result:

```xml
<?xml version="1.0"?>
<FlightInfo>
  <Intro>The following flights have available seats:</Intro>
  <Flights>
    <Row>
      <Airline>ACME</Airline>
      <FltNumber>123</FltNumber>
      <Depart>Dec 12, 2000, 13:43</Depart>
      <Arrive>Dec 13, 2000, 01:21</Arrive>
    </Row>
  </Flights>
  <Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>
```
Mapping document structure to database structure

- Model-driven
  - A data model is imposed on the structure of the XML document
  - This model is mapped to the structures in the database
  - There are two common models:
    - Model the XML document as a single table or a set of tables (table-based mapping; bi-directional)
    - Model the XML document as a tree of data-specific objects (object-relational mapping)

Single table or set of tables:

```xml
<?xml version="1.0"?>
<database>
  <table>
    <row>
      <column1>...</column1>
      <column2>...</column2>
      ...
    </row>
    ...
  </table>
</database>
```

Tree organization:

```
Orders
  |
  SalesOrder
  /
  \   
Customer Item Item
  |
  Part Part
```
Generating DTDs from a database schema and vice versa

- Many times the DTD does not change often for an application and does not need to be automatically generated.
- Some simple conversions are possible
  - Example: DTD from relational schema:
    1. For each table, create an ELEMENT.
    2. For each column in a table, create an attribute or a PCDATA-only child ELEMENT.
    3. For each primary key/foreign key relationship in which a column of the table contributes the primary key, create a child ELEMENT.

Document-centric storage and retrieval systems

- Content management system
  - Allows the storage of discrete content fragments, such as examples, procedures, chapters, as well as metadata such as author names, revision dates, etc.
  - Many content management systems are built on top of relational or object-oriented database systems.
  - Examples:
    - BladeRunner (Interleaf), SigmaLink (STEP), Parlance Content Manager (XyEnterprise), Target 2000 (Progressive Information Technology)
- Persistent DOM implementation
XML Resources


- W3C: World Wide Web Consortium
  http://www.w3.org/XML
  http://www.w3.org/TR/REC-xml
  http://www.w3.org/TR/xsl
  http://www.w3.org/TR/xslt

  O’Reilly Network affiliate site
  http://www.xml.com

- Ron Bourret's web site
  http://www.rpbourret.com/xml