A First Look at XML
Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring

September 4, 2001

Introduction to XML

Overview

• XML Basics
• Documents and Document Type Definitions
• Related and Companion Standards for XML
  • Schemas, namespaces, and datatypes
  • XPath
  • XML Linking Language (XLL)
  • XML Style Language (XSL)
  • XSL Transformation (XSLT)

September 4, 2001

Introduction to XML

Background

• XML is both simplified and extended SGML
  • Complex features defined for SGML that no are longer
    needed were eliminated
  • New capabilities, particularly related to datatypes were added
• XML has an extensive set of companion standards
  • The XML companion standards are in a state of flux
• The XML tools and standards are still in transition

Suggested Readings

• XML Standard – www.w3c.org/XML
• Goldfarb and Prescod, The XML Handbook
• Harold, The XML Bible

September 4, 2001

Introduction to XML
The Basic Idea

• XML and SGML are languages that define the rules for describing classes of documents.
  • Under SGML these definitions were called Document Type Definitions or DTDs.
  • Under XML, DTDs were used initially, but have now given way to Schemas.
• There are syntactic rules for writing documents generally and additional rules imposed by the DTD or Schema.
  • A document that meets the syntactic rules defined for writing XML documents is said to be “well formed”.
  • A document that complies with the constraints set out in a DTD or Schema is said to be valid.

XML Family of Standards

• Schemas provide a number of new capabilities:
  • Document schemas allow DTDs to be defined as XML documents—requiring only a single parser.
  • Namespaces allow for modular document definition, multiple inheritance, and collision avoidance.
  • Data types can be defined to allow for attribute value and content value controls.
• Companion standards:
  • XPath or the XML Path Language allows navigation of the document tree.
  • The XPointer allows tree components as targets.
  • The XML Linking Language defines linking capability.
  • The XML Style Language defines presentation capability.
  • XSLT provides for the transformation of documents.
Basic XML Syntax

- An XML document is made up of:
  - a prolog:
    `<!DOCTYPE MYBOOK SYSTEM "mybook.dtd">` (DTD form)
  - an instance
    » a nested set of elements beginning with the root element which has the same name as the DTD (Using Schema, the root element is associated with the schema name space)
  - An element begins and ends with tags
    » The start tag is of the form `<NAME>`
    » The end tag is of the form `</NAME>`
  - Tag names:
    » Are case sensitive (75% of XML errors are related to case sensitivity)
    - begins with a letter or underscore and may include letters, digits, hyphens, underscores, colons and periods

Additional XML Syntax

- The characters in an XML document are Unicode
- Whitespace is any combination of spaces, tabs, carriage returns, and line feeds, and is generally ignored
- Tags, processing instructions, comments and entity references are all considered markup
  » Entity references are of the form `&NAME;`
  » Comments are of the form `<!-- message -->`
  » A processing instruction is included using `<? ?>` delimiters
- In a DTD or Schema, additional forms appear, called declarations. These include:
  » Element declarations or content models
  » Attribute declarations
  » Entity definitions

“Escaping” Markup in Content

- XML has five predefined entities which may be used to allow markup symbols to be included in the content:
  - `&lt;` ::= `<`
  - `&gt;` ::= `>`
  - `&apos;` ::= `
  - `&quot;` ::= ``
  - `&amp;` ::= `&`
- CDATA sections may also be used to include data that would confuse a parser
  » `<![CDATA[ content ]]>`
  » obviously, the content cannot include `]]>`
Developing a DTD

- A formal XML document complies with a document type definition, or a schema.
- This is accomplished by the DOCTYPE element, which may be defined locally:

```xml
<!DOCTYPE note [
<!ELEMENT note (to, from, date, message)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT message (#PCDATA)> ]>
```
- The DOCTYPE may also be referenced:

```xml
<!DOCTYPE note SYSTEM "note.dtd">
```
- The two methods may be combined:

```xml
<!DOCTYPE note SYSTEM "note.dtd" [
<!ENTITY DOLLAR "$"> ]>
```

The Content Model of an Element

- The content of an element may be:
  - a set of subelements
  - text
  - a set of subelements and text
  - an empty element
  - an element with "ANY" content
    - normally used only for development
- Elements in a content model may be:
  - Required (default if no modifier)
  - Optional (?)
  - Repeatable (*, +)
- Elements in a content model may be:
  - ordered (,)
  - unordered (|)
  - (Schemas only) – an unordered set

Attributes

- An attribute definition defines the attributes of an element. It takes the general form:

```xml
<!ATTLIST   element  name value/range default>
```
- The element memo would appear as follows:

```xml
<memo status = "draft">
```
- The value range must either be a group or a reserved word.
- The default must be either a reserved name or a user supplied value. Default values may include the following:
  - #REQUIRED -- must be supplied
  - #IMPLIED -- optional (will be supplied by the system if absent)
  - #CURRENT -- is the most recent value.
Reserved Words for Values

- The reserved words can be:
  - CDATA -- character data
  - NUMBER (NUMBERS) -- a number or numbers
  - NAME (NAMES) -- one or multiple name strings
  - NMTOKENS -- names that can begin with a number
  - NMTOKENS -- names that must begin with a number
  - ID -- a valid SGML name unique within the scope of a document
  - IDREF -- must match an ID in the document

Entities

- Entities may be of any number of forms -- consistent with SGML
- The keywords INCLUDE and IGNORE may be used to allow conditional sections
  - `<!INCLUDE [ stuff to include ]>`
  - `<! [IGNORE [ stuff to ignore ]>`
- An entity may be defined that makes this more flexible
  - `<ENTITY % notes "IGNORE">`
  - `<! [notes] [ stuff to include ]>`
- Character references use the form `&#ddd;` where `ddd` are decimal digits that specify the Unicode character
  - To reference the hex number use `&#xddd;`

Processing Instructions

- A processing instruction begins with `<?` and ends with `?>`
  - The `<?` is followed by a target processor name
  - After the target processor name is any processing instructions followed by `?>`
- The XML processing instruction for stylesheets looks as follows
  ```xml
  <?xml:stylesheet href="/url" type="text/xsl"?>
  ```
XML Companion Standards

- Schemas provide new capabilities beyond DTDs
- Namespaces provide a mechanism for multiple inheritance
- The content model (aka DTD) is a normal XML document
- Datatypes allow more formal typing of data
- XPath or XML Path Language describes documents
- The XML Pointer Language allows paths as anchors
- The XML Linking Language provides extended linking capability
- XSL -- the XML Style Language provides presentation control
- XSLT provides for the transformation of documents

Schemas

- XML Schemas replace DTDs
- Schemas are recursively defined – i.e., a Schema is an XML document that is defined by a Schema that is an XML document...
- Schemas recognize and allow for datatypes both for attribute values and element content
- Schemas recognize “namespaces”, a relaxation of the one DTD/document constraint
- Schemas allow additional new features to be defined at a later date
- The Schema debate has been clouded by the simultaneous development of datatypes (Schema part 2) and RDF – Schema of a different color

Namespaces

- Namespaces provide
  - namespaces are being widely used, but they are controversial
  - a way to expand the scope of XML elements
  - a mechanism for multiple inheritance
- Namespaces allow two different elements with the same name
  - the basic idea is to provide a qualifier for the element
  - thus sis.pitt.edu:para could be distinguished from gsia.cmu.edu:para
  - domain names could work but it would require a registered name for a namespace
Namespace rules

- The attribute `xmlns` defines the default namespace for an element
  ```html
  <html xmlns="http://www.w3.org/TR/WD-HTML40">
  ```
- The attribute name `xmlns` is reserved
- This attribute defines the default namespace
- Subelements of this element are defined within the namespace
- A subelement with a different default namespace attribute is allowed – all subelements must be defined within that namespace
- Attribute names are also defined in the namespace
- The advantage of this form of namespace declaration is that it allows element names to be used without a prefix.

Qualified Namespaces

- An attribute name composed of `xmlns` followed by a name is called a qualified namespace
  ```html
  <mbs:email xmlns:mbs="http://….">
  ```
- The namespace is called `mbs`
- Subelements of this namespace must be prefixed by `mbs`:
- This allows multiple namespaces to be used simultaneously. Thus:
  ```html
  <html:html xmlns:spring="href1" xmlns:html="href2" xmlns:math="href3">
  ```
- Allows the children of `html` to be:
  ```html
  <html:xxx>
  <math:yyy>
  <spring:vv>
  ```

Datatypes Under DTDs

- SGML and XML under DTDs, put content outside the scope of the standard.
  - Element content was only checked related to syntax – i.e. `PCDATA`, `CDATA`, `NDATA`.
  - Attribute values allowed some “datatypes” checking, but it was minimal as well
    - `ID` `IDREF`
    - `NMTOKEN` `NOTATION`
- For e-commerce applications, XML required stronger data type capabilities
Schema Datatypes

• The datatype work under schemas adds more primitive types, which can be applied not only to attribute values but to element content
  • string
  • Boolean
  • number
  • DateTime
  • binary
  • uri
• Additional system datatypes, called generated are defined based on the primitive types.
  • integer
  • decimal
  • real
  • date
  • time
  • timePeriod

Defining and Using a datatype

• The user may also define datatypes. For example:
  
  ```
  <xsd:simpleType name="pubyear">
    <xsd:restriction base="xsd:gYear">
      <xsd:minInclusive value="1000"/>
      <xsd:maxInclusive value="3000"/>
    </xsd:restriction>
  </xsd:simpleType>
  ```

• An attribute declaration could now be defines:

  ```
  <attribute name = "Date"  type = "pubyear"/>
  ```

• An element declaration could be defined:

  ```
  <xsd:element name= "publishdate" type = "pubyear"/>
  ```

XPath

• XPath is a kind of SQL for XML
• XPath views a document as a tree of nodes
• The XML Path language identifies parts of an the XML document tree
  • XPath is used by XPointer to build a web address
  • XPath is used by XSLT to transform a document
• The topmost part of the tree is the root
  • it is not the same as the root element
  • a node contains all the comments, elements, text, attributes and PI
Examples of XPaths

- An XPath identifies a set of nodes in a document tree:
  - `/mydoc/chap/section`
    - would select the section elements of the chap elements in mydoc
  - `/mydoc/chap/para[7]`
    - would get the seventh para elements of all chap elements
  - `/mydoc/chap[@stat="draft"]`
    - would select all chaps whose stat attribute has a value of "draft"

XPath Expression

- An instance of an XPath is called an expression
- Applications process the expression returning a node set which the application then processes
- An expression can be absolute or relative
- An expression can traverse the tree in a number of different directions—called axes
- An expression can include a variety of tests in the form of functions and predicates

Axes and Tests

- Axes specify how the tree is traversed
- There are actually 13 different axes. For example:
  - The child axis, the default, looks down the tree one step
  - The parent axis looks up the tree one step
  - The attribute axis allows the attributes of a node to be explored
- Functions and predicates can be used to filter nodes
  - `text()` and `comment()` are examples of functions that would allow only one kind of node
  - `para[7]`, `para[last()]`, `para[footnote]` are examples of predicates that select the seventh, last, and paragraphs with footnotes respectively
**HTML Anchors**

- HTML linking is currently based on URL, URI, and URN's.
- It is important to know what is what.
  - URL's and URN's are URI.
  - URI is an abstract concept.
  - URN's will probably manifest themselves in new forms.
  - What we commonly put in an href is a URL.
  - A fragment identifier is an addition to the URL.
- It is based on fixed semantics – `<A NAME="lit">`.
- XML linking is much more robust than HTML linking.
- XML linking will require/allow radically new kinds of applications.

**Paths and Pointers**

- Path information allows a link to be made to a specific location within a document using Xpointer.
- Xpointer extends the capabilities of URI, URL, URN, and fragment identifier.
- In some ways, Xpointer is a shell for Xpath.
- Consider the following URL:
  ```
  http://www/c/g/xyz.xml#xptr(/mydoc/chap[3])
  ```

**XLINK**

- xlink is an example of one namespace.
  ```
  <mbs:link
      xmlns:xlink="http://www.w3.org/XML/Xlink/0.9"
      xlink:type="simple">
  ```
- XML does not know about namespaces, therefore care has to be taken in using them.
- The namespace abbreviations must be hardcoded in the dtd.
The XML Linking Language (XLL)

- XLL provides more linking capability
- simple linking, like that in html would look as follows
  - `<citation xlink:type="simple" xlink:href=URL>text</citation>`
- use of the xlink attributes requires the xlink namespace
  - `<rootname xmlns:xlink="http://www.w3.org/XML/Xlink/0.9">`

XLINK Attributes

- the definition of Xlink allows a variety of different link types to be developed.
- many of these are defined by the show attribute of xlink; xlink:show may be set to the following values
  - "replace" does what we see on the WWW
  - "new" causes a new window to be opened
  - "parsed" causes the href to be parsed and included
- another attribute of xlink is actuate which can take the following values
  - "user" indicates that traversal is based on user action
  - "auto" specifies that traversal should be automatic

Extended Links

- extended links include links that make use of the locator element
  - `<mylink xlink:type="extended">`
    - `<locator xlink:type="locator" xlink:href= "url" xlink:role="type of link">`
    - `<locator xlink:type="locator" xlink:href= "url" xlink:role="type of link">`
  - `</mylink>`
- link groups allow sets of documents to be linked together
  - behavior and processing of these is undefined
    - `<link group>`
      - `<link document:href="url"/>`
      - `<link document:href="url"/>`
      - `<link document:href="url"/>`
  - `</link group>`
The XML Style Language

- XSL provides more presentation capability
- XSL is a rendition language
  - it provides an alternative to the CSS as a style sheet
  - it makes use of the xmlns:xsl and xmlns:fo
    - fo stands for formatting object
    - this looks very similar to the layout hierarchy specified for interpress and ODA
    - the layout root is fo:root
      - the root has one or more fo:page-sequences
      - within the page sequences are flow elements fo:flow

Flow Elements

- the fo:flow elements contain other blocks
- fo:block
- fo:inline-graphic
- fo:display-graphic
- fo:display-rule (a ruling line)
- fo:display-sequence a set of attributes for a set of blocks
- fo:table
- fo:list-block
- fo:list-item-body
- fo:list-item-label and fo:list-item
- fo:simple-link
- fo:page-number

XSLT

- XSLT provides for the transformation of documents
- We can select, match, choose, filter, get the value of, etc.
- XSLT has two main functions
  - an intermediate language for making html documents from XML
  - an ultimate processor for taking XML documents to multiple forms
XSLT Processors

- Keep in mind that the XSL processor in ie5 is not fully conforming
- XSLT makes use of XSL style sheets
  - formally, the style sheet would begin:
    - `<xsl:stylesheet xmlns:xsl=http://www.w3.org/Transform/1.0`
    - `<xsl:stylesheet xmlns:html=http://www.w3.org/TR/REC-html40`
    - `<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>`
  - `<xsl:stylesheet version="1.0"`
  - `<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>`
  - `<xsl:stylesheet xmlns:html=http://www.w3.org/TR/REC-html40>`
  - `<xsl:stylesheet xmlns:xsl=http://www.w3.org/Transform/1.0>`

- for ie 5 use
  - `<xsl:stylesheet version="1.0"`
  - `<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>`
  - `<xsl:stylesheet xmlns:xsl=http://www.w3.org/Transform/1.0>`

- The rules are template rules

Style Sheets and Templates

- A style sheet is made up of a set of templates
- The template contain instructions on how to process the element they match
  - `<xsl:template match="book">`
  - `<xsl:template match="section">`
  - `<xsl:template match="para">`
  - `<xsl:template match="title">`
  - `<xsl:template match="relative Xpath">`
  - `<xsl:template match="relative Xpath">`
  - `<xsl:template match="relative Xpath">`

- A template does three things:
  - It outputs the literal content of the template element
  - It selects content from the input document and outputs it
  - It applies additional templates
  - `<xsl:value-of select="relative Xpath">`
  - `<xsl:value-of select="relative Xpath">`
  - `<xsl:value-of select="relative Xpath">`

- Apply-templates provides for recursive handling of the templates

An Example

- There are a whole series of structures and tests available as well. For example:
  - `<xsl:if select="relative Xpath">`
  - `<xsl:if select="relative Xpath">`
  - `<xsl:if select="relative Xpath">`
- Within a template, the order of subelements can be changed, in this case processing the paragraphs of a section before the title.
  - `<xsl:template match="section">`
  - `<xsl:template match="section">`
  - `<xsl:template match="section">`
  - `<xsl:template match="section">`
XML Tools

- There are lots of different XML tools out there.
- Two new commercial offerings that now incorporate schema and datatypes are:
  - XMLSpy
  - TurboXML
- To play with validation and wellformedness as well as XSLT:
  - IE5
  - xt by James Clark
- To code XML programmatically:
  - jaxp
  - xerces and xalan