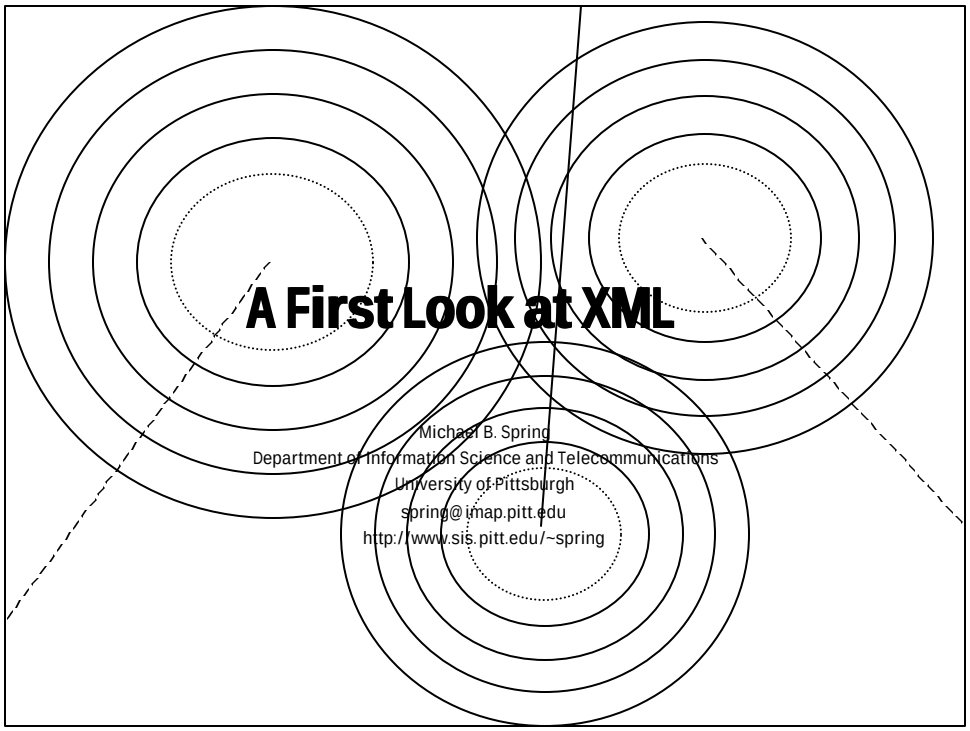

Overview

- Origins of XML
- XML Basics
- Related and Companion Standards for XML
  - XPath
  - XPointer
  - XML Linking Language (XLL)
  - XML Style Language (XSL)
  - XSL Transformation (XSLT)
  - Datatypes
  - Namespaces
- Suggested Readings
  - XML Standard -- www.w3c.org/XML
  - Goldfarb and Prescod, The XML Handbook Chapters 1-5 & 53-64

Background

- XML is simplified SGML written to be easier to implement in browsers
- Features defined for SGML that no longer make sense – given the evolution in computing are gone
- XML is extended SGML to handle a number of new tasks
- XML is the proud owner of all the companion standards never developed for SGML
- The companion standards for XML are still in a state of flux
- The XML tools available today are nothing compared to what will be available in a year or so.

A Note about SGML

- SGML is being pushed in the background
- SGML was (is still being) used in corporate settings
  - SGML editors and tools were built and used to manage large document projects
  - SGML folks saw XML as a simple a display language
  - SGML documents were to be converted to XML for display
- SGML was dependent on two companion standards:
  - The Document Style Semantics and Specification Language (DSSSL) for presentation
  - The HyTime Language was developed to provide new forms of linking (HyTime was originally for multimedia synchronization)
- The development of companion standards for XML has been explosive

XML Family of Standards

- Related Standards
  - Schemas provides a way to define DTD as a normal XML document – requiring only a single parser. Schemas also allow:
    - Data types to be incorporated
    - Namespaces to be incorporated
    - Other new features to be added
  - Namespaces provide a mechanism for multiple inheritance
  - Datatypes allow more formal typing of data
- Companion standards
  - XPath of XML Path Language describes documents
  - The XML Pointer Language allows paths as anchors
  - The XML Linking Language provides more linking capability
  - The XML Style Language provides more presentation capability
  - XSLT provides for the transformation of documents

Basic XML Syntax

- An XML document consists of Unicode characters
- XML is case sensitive
  - 75% of XML errors are related to case sensitivity
- Tags, processing instructions, declarations, and entity references are all considered markup
- A name
  - begins with a letter or an underscore
  - include letters, digits,hyphens, underscores, colons and periods
  - names starting with xml are reserved
- A name token can start with a digit, hypen, or period
- Literal data is a single or double quoted string quotes of one type may be included in a literal string using the other type
- Whitespace is any combination of spaces, tabs, carriage returns, and line feeds

XML Document Syntax

- An XML document is made up of:
  - a prolog
    - <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    - <!DOCTYPE MYBOOK SYSTEM "mybook.dtd“
  - an instance
    - a nested set of elements beginning with the root element which has the same name as the dtd
- An XML elements begins and ends with tags
  - The start tag is <NAME>
  - The end tag is </NAME>
- An entity in an XML document is of the form &NAME;
- Comments are of the form <!-- message -->
- A processing instruction is included using <! > delimiters

Including markup as text

- XML has five predefined entities which may be used to avoid confusion
  - These allow reserved symbols to be included in output
    - < < > >
    - ' ‘ " “
    - & &
- CDATA sections may also be used to include data that would confuse a parser
  - <![CDATA[content]]>
  - obviously, the content cannot include]]

Developing a DTD

- An XML document requires a definition of the document
- This is accomplished by the DOCTYPE element – actually, a processing instruction
- The DOCTYPE may be defined locally:

```
<!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:

```
<!DOCTYPE note SYSTEM "note. dtd“>
```

- The two methods may be combines

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


The Content Model of and Element

- There are four basic types of content for an element
  - a grouping of subelements without text
  - a grouping of subelements and text
  - An empty element
  - an element that may have any content -- normally used only for development may be sequences or choices
- Elements in a content model may be
  - Required (default if no modifier)
  - Optional (?)
  - Repeatable (*, +)
- Elements in a content model may be
  - ordered (,)
  - ored (|)

Attributes

- An attribute definition defines the attributes of an element
- It takes the general form
  - <!ATTLIST gi name value/range default>
  - e.g. <!ATTLIST memo status ("draft"| "final") "final">
- Given this definition, the element could have an attribute value pair
  - <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. If the user supplied value is a name string, the "s can be eliminated -- the reserved words require the rni -- #.
- default values may include the following:
  - #REQUIRED -- must be supplied
  - #IMPLIED -- is optional and 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 -- a number
  - NAME -- a name string
  - NUMBERS
  - NAMES
  - NMTOKENS -- names that can begin with a number
  - NUTOKENS -- names that must begin with a number
  - ID -- must be a valid SGML name and they must be unique within the scope of an SGML document; ID attributes should be named consistently -- some would say they should be called id
  - IDREF -- need not be unique but they must match a value of an ID somewhere in teh same document.
- #, the rni, is not required in the value range because the instantiation can not be a user defined name.

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 uses the form &#ddd; where ddd are decimal digits that specify the unicode character
  - to reference the hex number use ෝ

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:stylesheet href="" type=""?>

XML Companion Standards

- XPath or XML Path Language describes documents
- Namespaces provide a mechanism for multiple inheritance
- Datatypes allow more formal typing of data
- XML Schema provides a recursive form of DTD specification
- The XML Pointer Language allows paths as anchors
- The XML Linking Language provides linking capability
- XSL -- the XML Style Language provides more presentation capability
- XSLT provides for the transformation of documents

HTML Linking

- 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

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 XSL 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

Xpath Expression

- an instance of an Xpath is called an expression
  - when the system processes the hierarchy, it builds a node set
  - we can then iterate through the node set as we see fit
  - a location path is the most important kind of expression
  - an expression can also include functions -- a function call expression
  - different operations are performed on expressions
    - predicates are used to constrain or qualify expressions
      - they are placed in [] within a location expression
    - a / is used as the mechanism for traversing the tree

Examples

- /mydoc/chap/section would select the sections in chaps in mydoc
- /mydoc/chap/para[7] would get the seventh para or all chaps
- /mydoc/chap[@stat="daft"] would select all chaps with stat attr public

Axes

- we can also move around a document using "axes“
  - the descendent axis allows us to step down a document
    - /mydoc//footnotes finds footnotes anywhere
  - the parent axis allows us to find the parent of a given node
    - /mydoc//fn/.. finds the parent of each fn
  - the attribute axis allows the attributes of an element to be explored
  - the namespace axis is used to gather namespace information (I.e. the actual xpath language includes namespace prefixes)

```
/child::mydoc/child::sec[attribute::stat="draft"]
```

Node Tests

- there are a variety of node tests that we can use:
  - node() -- is it a node
  - text() -- is it a text node
    - /mydoc//text() == all of the text nodes in the document
  - comment() -- a comment node
    - /mydoc//comment() == all comment nodes
  - @name = an attribute node whose name is name
    - /mydoc//@status == all of the attribute nodes named status
  - . is used to refer to the current node
  - .. is the parent

Predicates in Xpath

- the simplest predicates test context
  - //chap/para[7] selects the seventh para
  - //chap/para[last()] selects the last para
  - //para[footnote] selects para's that have footnotes
  - //para[footnote][@important] selects para's that have footnotes and important attributes
- predicates can also test content
  - //chap[title = 'Introduction'] selects a chapter whose title value is Introduction
  - //chap/para[@type='ordered']

Predicates Continued

- predicates can test for ID which is unique in XML
  - id('mbs001') selects the element whose id is mbs001
  - id('mbs001')/para[position()=7] selects the seventh para of the element if id is ...
    - note the explicit form of the positional predicate

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])

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

- attribute names that begin xmlns define a name space -- xmlns is thus reserved
  - the name following xmlsn: is the abbreviation for the namespace

    <mbs:email xmlns:mbs="http://…."> defines a namespace
  - namespaces are scoped -- they apply to all children of the element thus,

    <html:html
    xmlns:spring="href1"
    xmlns:html="href2"
    xmlns:math="href3">
  - allows the children of html to be:

    <html:xxx>
    <math:yyy>
    <spring:vv>
  - there is a default namespace that is declared as follows:

    <html xmlns="http://www.w3.org/TR/WD-HTML40">
    - allows the children to be used without a prefix
    - you can also subdeclare a default namespace within a default namespace
    - attribute names, like element names can come form a namespace.

XLINK

- xlink is an example of one namespace
  - <mbslink 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

Datatypes

- XML Datatypes are need for e-commerce
  - Under XML elements and attributes can not be controlled
- While there are some built in datatypes for XML for attributes, they are not enough
  - ID IDREF
  - NMTOKEN NOTATION
- The datatypes work under schemas adds some more primitive types
  - string Boolean
  - number DateTime
  - binary uri

DataTypes 2

- it also allows addition generated data types.
- some of the datatypes already specified include:
  - integer decimal
  - real date
  - time timePeriod
- in addition, the user may generate there own complex datatypes
- here is a restricted datatype

```
<datatype name = "pubyear">
<basetype name = "integer">
<minInclusive>1000</minInclusive>
<maxInclusive>3000</maxInclusive>
</datatype>
```


Using a datatype

- with this done, we could include an attribute declarartion

```
<attrDecl name = "py" required="false">
<datatypeRef name = "pubyear"/>
</attrDecl>
```

- now finally, we could do the following

```
<!DOCTYPE book [
<!ELEMENT book ()>
<!NOTATION pubyear SYSTEM
  "definitions.xml\pubyear">
<!ATTLIST book  py CDATA #IMPLIED -- dtype:
  pubyear>
```

Schemas

- XML Schemas provide an extension to DTDs that makes them consistent in terms of notation with documents
- allows data types to be incorporated
- allows namespaces to be incorporated
- allows new features to be added

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

```
<xlink:group>
    <xlink:document href="url"/>
    <xlink:document href="url"/>
    <xlink:document href="url"/>
</xlink: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

Using XSLT

- 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
xmlns:html=http://www.w3.org/TR/REC-html40
result-ns="html">
…. rules…
</xsl:stylesheet>
```

  - for ie 5 use

```
<xsl:stylesheet version="1.0“
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

  - The rules are template rules

Examples

- A simple template follows the instructions in a template

```
<xsl:template match="book">
    … information on how to format book
</xsl:template>
```

- the literal information is text to be output the functional information is of two broad types

```
<xsl:value-of select="relative Xpath"/>
<xsl:apply-templates select="relative Xpath"/>
```

- the apply templates works recursively
- the apply-templates can also use a test attribute

```
<xsl:apply-templates test="relative Xpath"/>
```

- There are also conditional processing rules

```
<xsl:if select="relative Xpath">
do something
</xsl:if>
```

- there is also a choose tag in xsl

Ordering

- within a template, we can order the handling of subelements

```
<xsl:template match="section">
<xsl:value-of select="title"/>
<xsl:apply-templates select="para"/>
</xsl:template>
```

- the nodes selected or matched can be ored "|" allowing allowing sharing