

XML, XPath, and XSLT

Jim Fawcett

CSE 681 – Software Modeling and Analysis

Fall 2016

Topics

- XML is an acronym for eXtensible Markup Language.
 - Its purpose is to describe structured data
- XPath is a language for navigating through an XML document.
 - It's used to select specific pieces of information from the document
- XSLT is a language for transforming XML into something else.
 - Often used to generate HTML or another XML document.

Introduction to XML

- XML is a tagged markup language designed to describe data: [LectureNote.xml](#)
- XML has only a couple of predefined tags
 - All the rest are defined by the document designer.
 - XML can be used to create languages
- XML is commonly used to:
 - Define data structures.
 - Define messages
 - Create web pages

Validation

- To be correct XML a set of markup needs only to be well formed, see [Well-Formed XML](#).
- To determine if an XML document belongs to some document type, XML uses either:
 - Document Type Definition (DTD)
 - XML SchemaXML that satisfies a Schema or DTD is said to be valid.
- DTDs and Schemas define allowable tags, attributes, and value types, and may also specify where these may occur in the document structure.
 - XML schemas are written in XML, DTDs are not.

XML Element

- Elements are building blocks for XML documents
- Element syntax:
 - Elements are composed of tags, attributes, and a body:
body:
`<tag * [attribName="value"] >body</tag>`
example:
`<book author="Prosise">Programming .Net</book>`
 - All parts of the element are unicode text
 - body may contain both plain text and markup, e.g. lower level elements.
 - Tags and attributes are case sensitive and user defined.

Element Naming Rules

- XML names are composed of unicode characters
 - Tag names must begin with a letter or underscore
 - Other tag name characters may contain characters, underscores, digits, hypens, and periods
 - Names may not contain spaces nor start with the string “xml” or any case variant of “xml”.
 - Attribute names follow the same rules as tag names, and are also required to be unique within the tag in which they are embedded.

Element Body Rules

- Element bodies may contain plain text or markup or both.
 - By plain text, we mean character strings with no markup.
 - Markup is text with embedded markup characters:
 - & < > ` and `
 - Elements may also contain CDATA sections, designed to support text including large sections of markup characters but not interpreted as markup:
 - `<![CDATA[...]]>`
 - These cannot be used to carry binary data.

Illegal Characters

- Certain characters are reserved for markup and are illegal in names and payload text:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

- We represent them in plain text with the escape sequences shown on the left, e.g.: < if we want a less than character in payload text.

XML Structure

- An XML document is defined by a standard opening processing instruction:
 - `<?xml version="1.0"?>`
 - Processing instructions and comments are the only XML tags that are not closed (see next page)

- The XML body starts with a single root element.
- An element is text of the form:

```
<someTag anAttribute="someValue">payload text</someTag>
```

where the payload may be one or more child elements or simply text or both.

- Comments take the form:

```
<!-- a comment -->
```

Well-Formed XML

- XML has a few rules:
 - There may be only a single root
 - All tags, except for processing instructions, must be closed:
 - `<myTag someAttrib="value">...</myTag>`
 - `<myTag someAttrib="value"/>`
 - Attribute values must be quoted
 - XML tags are case sensitive
 - All markup and payload is text with one exception:
 - An element may define a CDATA section
 - CDATA is not parsed, and so may contain anything except the CDATA terminator

CDATA

- A CDATA section has the syntax:

```
<![CDATA[ ... ]]>
```

- CDATA is not parsed except to look for the terminator “]]>” so it may contain anything.
 - It is not a good idea to try to store binary data in a CDATA section because the “]]>” sequence could appear as part of the binary data.

XML Documents

- An XML document is well-formed XML if it contains:
 - A prolog: `<?xml version="1.0"?>`
 - An optional link to an XSLT stylesheet
 - An optional reference to a DTD or schema, used for validation
 - Optional processing instructions
 - Optional comments
 - A body with a single root, which may contain any number of text sections, elements, and comments
 - An optional epilogue consisting of comments and processing instructions

Processing Instructions

- Processing instructions are used to capture information for XML parsers and proprietary applications.
 - Syntax: `<? PI-target *[attrib="value"]?>`
- The most common processing instructions are:
 - Document banner:
`<?xml version="1.0" encoding="utf-8"?>`
 - XSLT style-sheet reference:
`<?xml-stylesheet type="text/xsl" href="courses.xsl"?>`
- Other hypothetical instructions:
 - `<? robots index="no" follow="yes" ?>`
 - `<? word document="aDoc.doc" ?>`

Namespaces

- Namespaces are declared with special attributes and prefixes:
 - `<tag xmlns:prefix="uri">body</tag>`
 - The uri should be unique, so current style is to use a url, e.g., www.ecs.syr.edu.
 - These urls need not be bound to some real site.
 - Attributes do not inherit the namespace of their element, so you need to do this:
`<tag xmlns:a="uri" a:myAttrib="value">body</tag>`
- Namespaces are used to distinguish different elements that happen to have the same tag name, but are not intended to mean the same thing.
 - Perhaps, they have different structures

Example

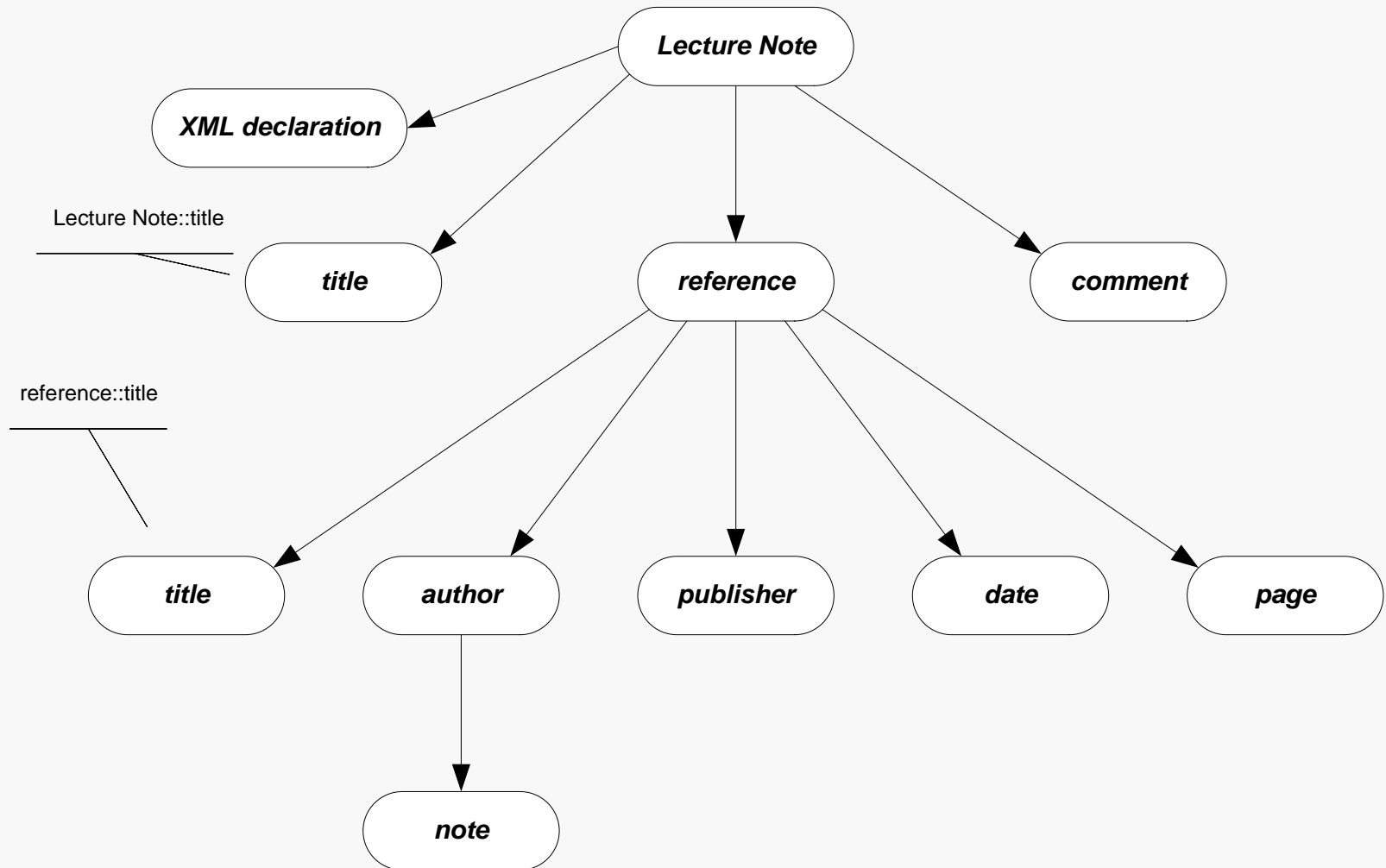
```
<?xml version="1.0"?>
<!-- XML test case -->
  <LectureNote course="cse681">
    <title>XML Example #1</title>
    <reference>
      <title>Programming Microsoft .Net</title>
      <author>
        Jeff Prosise
        <note company="Wintellect"></note>
      </author>
      <publisher>Microsoft Press</publisher>
      <date>2002</date>
      <page>608</page>
    </reference>
    <comment>Description of PCDATA</comment>
  </LectureNote>
```

Note: we can have both text and child nodes in the payload of an element.

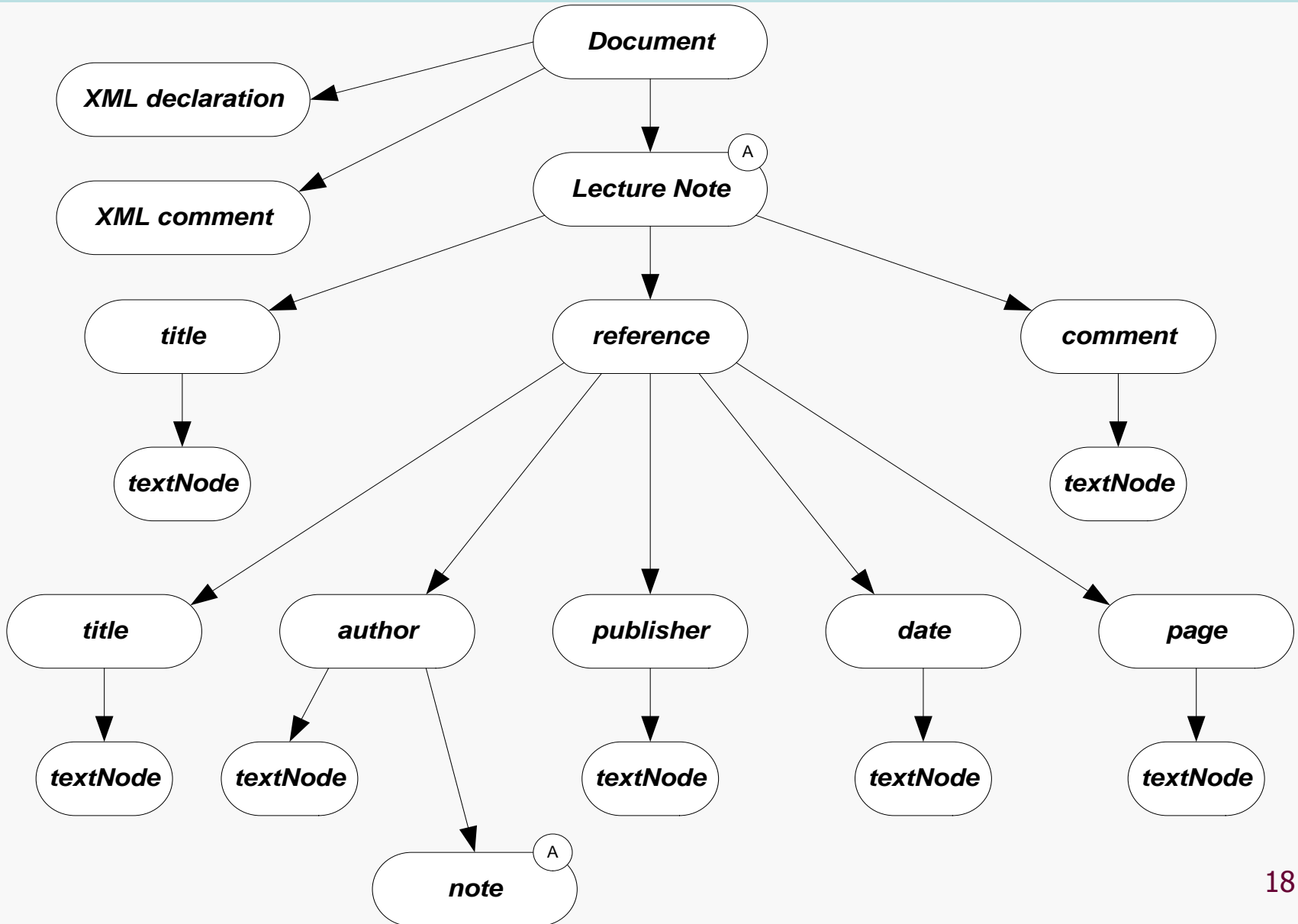
[LectureNote.xml](#)

[Webmonkey | Reference: Special Characters](#)

XML Node Structure



XML Parse Tree



XML Presentation

- There are several ways XML data can be presented to a user:
 - XML data island in an HTML page, interpreted by script
 - XML file interpreted by script in an HTML page
 - XML island or file bound to an HTML table
 - XML file bound to a GridView control
 - XML styled with an XSL style sheet
 - Essentially, the XSL sheet creates viewable HTML
 - Read, interpreted, and modified by an application
 - The .Net System.XML library provides very effective support for this.

Introduction to XPath

- XPath provides a navigation facility within XML documents
 - XPath is used to extract specific information from XML documents:
 - In XSL style sheets
 - » `<xsl:template match=xpath expression>`
 - » `<xsl:for-each select=xpath expression>`
 - » `<xsl:value-of select=xpath expression>`
 - » `<xsl:apply-templates select=xpath expression>`
 - In C# programs that use the XML DOM
 - » `XmlNode.SelectSingleNode(xpath expression)`
 - » `XmlNode.SelectNodes(xpath expression)`
 - In Javascript code

XPath Components

- XPath syntax contains the following components:
 - Steps
 - A directory like syntax for defining elements and attributes at some specified level
 - /customers/customer/lastName
 - /customers/customer[@status = current]
 - Descent Steps
 - Steps that may occur at any level in an XML structure
 - //lastName
 - Filters
 - Elements or attributes that must be present to result in a match
 - /customers/customer[country]
 - Predicates
 - Condition that must be met to result in a match
 - /customers/customer[country="United States of America"]

XPath Node Set Functions

- XPath provides a number of functions that operate on sets of nodes:
 - `count()`
 - the number of nodes in a set
 - `/customers/customer[count(order) = 1]`, e.g., customers with only one order
 - `position()`
 - `position` returns the position of an XML node in a set of nodes:
 - `/customers/customer[position() = 1]`, e.g., first customer
 - `last()`
 - Returns the ordinal of the last node in a set
 - `/customers/customer/order[position() = last()]`, e.g., last order of each customer

XPath String Functions

- XPath has three commonly used string functions:
 - contains()
 - Returns true if string in first argument contains the second
 - `//customer[contains(jobTitle,"chief")]`
 - string-length()
 - Returns integer number of characters in string
 - `//customer[string-length(lastName) > 3]`
 - substring()
 - `substring(str,start,length)` returns substring of str starting at character start with number of characters equal to length
 - `//customer[substring(city,0,3) = "Los"]`

Other XPath Functions

- XPath number functions:
 - sum()
 - sum(products/product/price)
- Boolean functions
 - false()
 - true()
 - not()
 - //customer[not(count(orders) = 0)]

XPath Expressions

- XPath supports numerical, boolean, and comparison expressions:
 - create complex predicates
 - `//customer[count(orders) > 0 and State = "California"]`
- XPath unions
 - return the union of two node sets
 - `//books | //articles`

XPath Axes

- XPath axis specifies the direction of node selection from the context node:
 - child
 - Child nodes of the context node
 - parent
 - Parent node of the context node
 - ancestor
 - All ancestors of the context node
 - descendant
 - All descendants of the context node
 - attribute
 - Attributes of the context node

Axes Examples

- /customer/lastName
 - /child::customer/child::lastName
- //firstName
 - descendant::firstName
- //drive/@letter
 - //drive/attribute::letter
- //file/../@name
 - //file/parent::folder/@name
- //folder[parent::folder and not(child::file)]
 - Subdirectories with no files

Introduction to XSLT

- XSLT is an acronym for eXtensible Stylesheet Language – Transform.
- Designed to transform an input XML parse tree into a parse tree for the output – often XML or HTML.
- The transformations are defined as templates in a stylesheet, with extension xsl.
- .Net provides several classes to support this operation.

XSLT Template Processing

- `<xsl:template match=XPath expression>`
 `// processing defined for the`
 `// matching node set`
`</xsl:template>`
- Processing consists of:
 - Literals that are sent directly to the output
 - Templates with their results sent to the output
- An XSLT stylesheet can have an arbitrary number of templates.
- Templates are processed at two points in time:
 - When the transformation is first invoked.
 - Whenever `<xsl:apply-templates />` is encountered during processing.

apply-templates

- `<xsl:apply-templates />`
- The current selection is matched against all templates in the stylesheet.
- Each match executes the matching template's processing.
- The results are sent to the output.

for-each

- `<xsl:for-each select=XPath expression>`
 // processing for selections
`</xsl:for-each>`
- Each element of the matching node set is processed according to the body of the template.
- Results are sent to the output.

value-of Template Instruction

- `<xsl:value-of select=XPath expression />`
- Returns the value of the selected node
- The selection is from the context defined by the template selection (see previous slide).

Example

- The links, below, refer to an example of XSLT processing, executed on a web server, to render a webpage based on contents of an XML file:
 - www.ecs.syr.edu/faculty/fawcett/handouts/cse686/code/XSLTdemo/XSLTdemo.aspx
 - www.ecs.syr.edu/faculty/fawcett/handouts/cse686/code/XSLTdemo/XSLTFile.xsl
 - www.ecs.syr.edu/faculty/fawcett/handouts/cse686/code/XSLTdemo/XMLFile_NoStyleLink.xml
- Other references for XSLT
 - www.w3schools.com/xsl/xsl_languages.asp
 - <http://www.zvon.org/xxl/XSLTutorial/Books/Book1/>
 - [http://directory.google.com/Top/Computers/Data_Formats/Markup_Languages/XML/Style_Sheets/XSL/FAQs, Help, and Tutorials/](http://directory.google.com/Top/Computers/Data_Formats/Markup_Languages/XML/Style_Sheets/XSL/FAQs,_Help,_and_Tutorials/)

End of Presentation