XPath and XSLT

Based on slides by Dan Suciu
University of Washington

Today's Lecture

- Some remarks about XML and DTSs
- One slide on XML Schema (much more next lecture)
- XPath
- XSLT

Notes about DTDs

```
<!ELEMENT Book (title, author*))
<!ELEMENT title #PCDATA>
<!ELEMENT author (name, address, age?)>
<!ATTLIST Book ID #REQUIRED>
<!ATTLIST Book pub IDREF #IMPLIED>
```

Notes:

- #PCDATA: Parsed character data. Entity references (such as &lt) will be replaced, no tags or child elements allowed
- Empty elements: EMPTY
  - <!ELEMENT image EMPTY>
  - <image src="bus.jpg> width="152" height="270"/>
- DTDs under construction: ANY
  - <!ELEMENT paragraph ANY>

Attributes Types

- Attribute types:
  - CDATA: Any string
    - In general: <![CDATA< …]]>
  - NMTOKEN, NMTOKENS: XML name (some syntactic restrictions)
    - <!ATTLIST book editions NMTOKENS #REQUIRED>
  - Enumeration
    - <!ATTLIST date weekday (Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday) #IMPLIED>

Attribute Types (Contd.)

- ID:
  - XML Name (can be used in IDREFs)
  - <!ATTLIST employee ssn ID #REQUIRED)
  - <employee ssn="_123_45_6789"/> (A number is not an XML name!)
- IDREF:
  - <!ATTLIST employee manager IDREF #REQUIRED)
- Similar IDREFS

Attribute Defaults

- #REQUIRED: mandatory
- #IMPLIED: optional
- #FIXED: Constant and immutable
- Values: default value is given as a string

Limitations of DTDs

- No namespaces
- No datatypes (basically, just strings)
- No integrity constraints (only IDREF and IDREFS), no typing of integrity constraints
- XML is ordered, but DTDs specify order
  - How can we make order immaterial in a DTD?
- No localization of elements (e.g., if name consists of first and last for customers, we cannot have a differently structured name anywhere else)

XML Schema: The One Slide

- Same syntax as XML
- Integrated with namespace
- Several built-in datatypes (string, integer, time, etc.)
- Construct complex types from simpler types
- Key constraints, referential integrity constraints
- Better mechanisms for order independence

- XML document that conforms to a schema is called *schema valid*, the document is an *instance* of the schema.

- MUCH more next lecture.

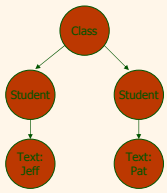
XPath

- http://www.w3.org/TR/xpath (11/99)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Link (XLink)
  - XML Pointer (XPointer)
  - XML Query
- Was originally part of XSL

XPath Data Model

- XPath views XML documents as trees with children and parents
- Special root node (not shown)
- Attributes are not considered children

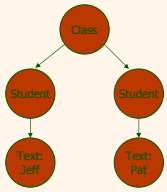
```
<Class>
  <Student>Jeff</Student>
  <Student>Pat</Student>
</Class>
```



XPath – Navigating Xml

- XPath provides operators to navigate the document tree

```
<Class>
  <Student>Jeff</Student>
  <Student>Pat</Student>
</Class>
```

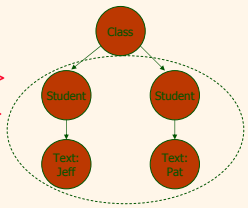


XPath – Navigating Xml

- Xml is similar to a file structure, but you can select more than one node:

  /Class/Student

```
<Class>
  <Student>Jeff</Student>
  <Student>Pat</Student>
</Class>
```



XPath – Navigating Xml

- Similar to Unix file system:
  - / -- root note
  - . -- current node
  - .. -- parent node
- An XPath expression looks just like a file path
  - Elements are accessed as /<element>/
  - Attributes are accessed as @attribute
  - Text is accessed with text()
- Everything that satisfies the path is selected
  - You can add constraints in brackets [ ] to further refine your selection

XPath – Navigating Xml

```
<class name='CS 330'>
   <location building='Hollister' room='110'/>
   <professor>Johannes Gehrke</professor>
   <ta>Scott Selikoff </ta>
   <student_list>
      <student id='999-991'>John Smith</student>
      <student id='999-992'>Jane Doe</student>
   </student_list>
</class>
```

Starting Element Attribute Constraint

//class[@name='CS 330']/student_list/student/@id

Element Path Selection

Selection Result: The attribute nodes containing 999-991 and 999-992

XPath - Context

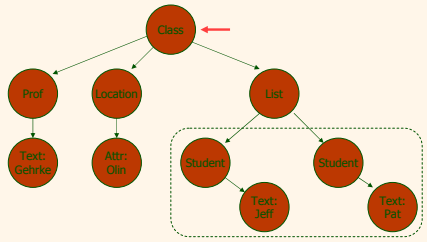
- *Context* – your current focus in an Xml document
- Use:

  //<root>/…

  When you want to start from the beginning of the Xml document

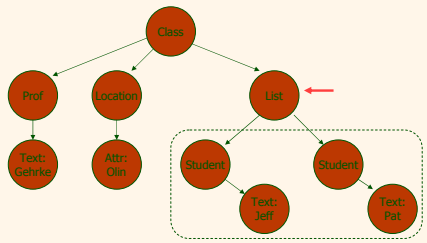
XPath - Context

XPath: List/Student



XPath - Context

XPath: Student



XPath – Examples

```
<Basket>
   <Cherry flavor='sweet'/>
   <Cherry flavor='bitter'/>
   <Cherry/>
   <Apple color='red'/>
   <Apple color='red'/>
   <Apple color='green'/>
   …
</Basket>
```

Select all of the red apples:

```
//Basket/Apple[@color='red']
```

XPath – Examples

```
<Basket>
   <Cherry flavor='sweet'/>
   <Cherry flavor='bitter'/>
   <Cherry/>
   <Apple color='red'/>
   <Apple color='red'/>
   <Apple color='green'/>
   …
</Basket>
```

Select the cherries that have some flavor:

```
//Basket/Cherry[@flavor]
```


XPath – Examples

```
<orchard>
  <tree>
    <apple color='red'/>
    <apple color='red'/>
  </tree>
  <basket>
    <apple color='green'/>
    <orange/>
  </basket>
</orchard>
```

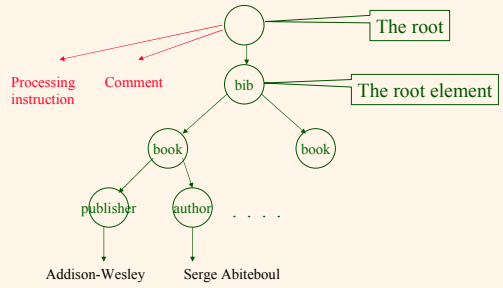
Select all the apples in the orchard:

```
//orchard/descendant()/apple
```


Example for XPath Queries

```
<bib>
   <book>  <publisher> Addison-Wesley </publisher>
           <author> Serge Abiteboul </author>
           <author> <first-name> Rick </first-name>
                    <last-name> Hull </last-name>
           </author>
           <author> Victor Vianu </author>
           <title> Foundations of Databases </title>
           <year> 1995 </year>
   </book>
   <book price="55">
           <publisher> Freeman </publisher>
           <author> Jeffrey D. Ullman </author>
           <title> Principles of Database and Knowledge Base Systems </title>
           <year> 1998 </year>
   </book>
</bib>
```

Example: Data Model



Example: Simple Expressions

/bib/book/year

Result: <year> 1995 </year>
<year> 1998 </year>

/bib/paper/year

Result: empty (there were no papers)

Example: Restricted Kleene Closure

//author

Result:<author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>

/bib//first-name
Result: <first-name> Rick </first-name>

Example: Functions

/bib/book/author/text()

Result: Serge Abiteboul
Jeffrey D. Ullman

Rick Hull doesn't appear because he has firstname, lastname

Functions in XPath:

- text() = matches the text value
- node() = matches any node (= * or @* or text())
- name() = returns the name of the current tag

Example: Wildcard

//author/*

Result: <first name> Rick </first name>
<last name> Hull </last name>

* Matches any element

Example: Attribute Nodes

/bib/book/@price

Result: "55"

@price means that price is has to be an attribute

Example: Qualifiers

/bib/book/author[firstname]

Result: <author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>

Example: More Qualifiers

/bib/book/author[firstname][address[//zip][city]]/lastname

Result: <lastname> … </lastname>
<lastname> … </lastname>

Xpath: More Qualifiers

/bib/book[@price < "60"]

/bib/book[author/@age < "25"]

/bib/book[author/text()]

Xpath: Summary

bib	matches a bib element
*	matches any element
/	matches the root element
/bib	matches a bib element under root
bib/paper	matches a paper in bib
bib//paper	matches a paper in bib, at any depth
//paper	matches a paper at any depth
paper|book	matches a paper or a book
@price	matches a price attribute
bib/book/@price	matches price attribute in book, in bib
bib/book/[@price<"55"]/author/lastname	matches…

XPath – In-Class Exercise

```
<class name='CS 330'>
   <location building='Hollister' room='110'/>
   <professor>Johannes Gehrke</professor>
   <ta>Scott Selikoff </ta>
    <student id='999-991'><first>John</first><last> Smith</last></student>
    <student id='999-992'><first>Jane</first></student>
</class>
```

/class/student[2]
/class/student/text()
/class/student

Overview

- Querying XML: XPath
- Transforming XML: XSLT

Xslt – Transforming Xml

Amazon.com order form:

```
<single_book_order>
  <title>Databases</title>
  <qty>1</qty>
</single_book_order>
```

Supplier's order form:

```
<form7957>
  <purchase item='book' property='title' value='Databases' quantity='1'/>
</form7957>
```


Xslt - Extensible Style Language for Transformation

- Xslt is a language for transforming or converting one Xml format into another Xml format.
- Benefits:
  - No need to parse or interpret many different Xml formats – they can all be transformed to a single format to facilitate interpretation
  - Language looks like XML! (remember, XML defines languages!)

Xslt – A First Look

```
<single_book_order>
  <title>Databases</title>
  <qty>1</qty>
</single_book_order>
        ↓
<form7957>
  <purchase item='book' property='title' value='Databases' quantity='1'/>
</form7957>
```

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'>
  <xsl:template match='single_book_order'>
    <form7957><purchase item='book' property='title' value='{title}'
                        quantity='{qty}'/></form7957>
  </xsl:template>
</xsl:stylesheet>
```

Xslt – Header

- Xslt stylesheets MUST include this body:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'>
   …
</xsl:stylesheet>
```


Xslt – Templates

- Xslt stylesheets are a collection of templates
  - Templates are like functions
  - The body of a template is the output of a transformation

Xslt - Templates

- You define a template with the <xsl:template match=''> instruction
- You call a template with the <xsl:apply-templates select=''> instruction

1. All elements or attributes that satisfy the the select attribute expression are selected.
2. For each element or attribute that is selected:
   i. A matching template is found in the stylesheet.
   ii. The body of the template is executed.

Xslt – Template Matching

Stylesheet

```
<xsl:template match='basket'>
  <new_basket>
  <xsl:apply-templates select='apple'/>
  <xsl:apply-templates select='box'/>
  </new_basket>
</xsl:template>

<xsl:template match='apple'>
  <apple/>
</xsl:template>

<xsl:template match='box'>
  <box/>
  <xsl:apply-templates/>
<xsl:template>
```

Xml

```
<basket>
  <apple color='red'/>
  <apple color='green/>
  <apple color='green/>
  <box>
    <orange taste='good'/>
    <peach/>
    <apple color='red'/>
  </box>
</basket>
```

Transformed Xml:

```
<new_basket>
  <apple/> <apple/> <apple/>
  <box/><apple/>
</new_basket>
```


Xslt – choose Instruction

- <xsl:choose> instruction is similar to a C++ or Java switch statement
- <xsl:when test=''> instruction is similar to the case statement
- <xsl:otherwise> instruction is similar to the default statement

Xslt – choose Example

Original Xml:

```
<customer>
  <order id='5'>
    <item><title>Database Management Systems</title></item>
  </order>
</customer>
```

Xslt Stylesheet:

```
<xsl:template match='customer'>                    ← FUNCTION
  <xsl:choose>                                     ← SWITCH
    <xsl:when test='order/@id'>                    ← CASE
      <single_book_order>
        <title><xsl:value-of select='order/item/title'/></title>
      </single_book_order>
    </xsl:when>
    <xsl:otherwise><single_book_order><fail/>      ← DEFAULT
        </single_book_order></xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Output Xml:

```
<single_book_order><title>Database Management Systems</title></single_book_order>
```

Xslt – choose Example 2

Original Xml:
```
<customer>
  <order>
    <item><title>Database Management Systems</title></item>
  </order>
</customer>
```

Xslt Stylesheet:
```
<xsl:template match='customer'>                          ← FUNCTION
  <xsl:choose>                                           ← SWITCH
    <xsl:when test='order/@id'>                          ← CASE
      <single_book_order>
        <title><xsl:value-of select='order/item/title'/></title>
      </single_book_order>
    </xsl:when>
    <xsl:otherwise><single_book_order><fail/>            ← DEFAULT
        </single_book_order></xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Output Xml:
```
<single_book_order><fail/></single_book_order>
```


Xslt – for-each Instruction

- <xsl:for-each select='item'> instruction is similar to a foreach iterator or a for loop
- The select attribute selects a set of elements from an Xml document

Xslt – if Instruction

- <xsl:if test=''> instruction is similar to an if statement in Java or C++
- The test attribute is the if condition:
  - True
    - statement is true
    - test returns an element or attribute.
  - False
    - statement is false
    - test returns nothing
- There is no 'else', so use the <xsl:choose> operator in this situation.

Xslt – for-each and if Example

Original Xml:
```
<basket>
  <apple color='red' condition='yummy'/>
  <apple color='green' condition='wormy/>
  <apple color='red' condition='crisp'/>
</basket>
```

Xslt Stylesheet:
```
<xsl:template match='basket'>                          ← FUNCTION
  <condition_report>
  <xsl:for-each select='apple'>                        ← FOR LOOP
    <xsl:if test="contains(@color, 'red')">            ← IF
      <condition><xsl:value-of select='@condition'/></condition>
    </xsl:if>
  </xsl:for-each>
  </condition_report>
</xsl:template>
```

Output Xml:
```
<condition_report>
  <condition>yummy</condition>
  <condition>crisp</condition>
</condition_report>
```


XSLT: Examples

- Demonstration of transform1.xsl to transform5.xsl

XSLT: Default Template Rules

Element and root nodes:

```
<xsl:template match="*|/">
    <xsl:apply-templates/>
<xsl:template>
```

Text and attribute nodes:

```
<xsl:template match="text()|@*">
    <xsl:value-of-select="."/>
</xsl:template>
```

Note: We need to specify an XPath expression to select attributes.

XSLT: Modes

- We might use the same input in different contexts
  - Example: Formatting section titles in TOC versus chapter
- Idea: use different modes
  - Attribute in xsl instructions
  - Show example transform6.xsl

Summary

- Shortcomings of DTDs
- XPath: Language for navigating XML documents
  - No joins!
- XSLT: Transforming XML documents
  - Turing-complete
- Next lecture:
  - XML Schema
  - Xlinks, Xpointers
  - XQuery