Navigating an XML Document

Owen.Conlan@scss.tcd.ie
Athanasios.Staikopoulos@scss.tcd.ie
What is XPath?

• Language for addressing parts of an XML document
  – Used in XSLT, XQuery
• A dedicated and powerful expression language for forming queries based on the tree structure of an XML document.
• It is used to
  – Locate nodes in a tree
  – Extract information
  – Provide basic operations over data: e.g., manipulation of strings, numbers and booleans
• Compact, non XML syntax for use within URIs and XML attribute values
• Operates on the abstract, logical structure of the XML document
XPath versions

- W3C Recommendation
  - XPATH 1.0 - (1999)
  - XPATH 2.0 – (2010) – backwards compatible

- XPath 1.0
  - considers a single XML document as a tree of nodes
  - Nodes have identity
  - Set of nodes – unordered collection of nodes

- XPath 2.0
  - More complex, is a superset of XPath 1.0
  - More elaborate data model
  - More functions
  - It does not considers on a single document tree, but on arbitrary data sets
  - These can be arranged in sequences of items – ordered sets
Basic Concepts

• Node Types
  – XML documents are treated as trees of nodes
  – The topmost element of the tree is called the root (or document) node
  – XPath defines seven node types

• Context Node
  – Provides the starting point (current node) that is basis of path navigation and evaluation
  – Default is the root (document)

• Location Steps
  – Provides the directions
  – Sequences the nodes
  – The evaluation of each node provides the current context
  – Example: /node1/node2/node3
## Node Types

- **XPath defines 7 node types**

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root/Document Node</td>
<td>The root of the tree representing the entire document contents, represented by the &quot;/&quot;</td>
</tr>
<tr>
<td>Element</td>
<td>Element nodes are defined by pairs of start &lt;title&gt; and end tags &lt;/title&gt;</td>
</tr>
<tr>
<td>Text</td>
<td>A character sequence in an element, comment, processing instruction, or namespace</td>
</tr>
<tr>
<td>Attribute</td>
<td>The name and value of an attribute in an element</td>
</tr>
<tr>
<td>Comment</td>
<td>Comments in an XML source document, such as &lt;!-- model diagram --&gt;</td>
</tr>
<tr>
<td>Processing Instruction</td>
<td>An instruction in the source document, such as the &lt;?xml-stylesheet href=&quot;book.xsl&quot; type=&quot;text/xsl&quot;?&gt;</td>
</tr>
<tr>
<td>Namespace</td>
<td>A namespace declaration</td>
</tr>
</tbody>
</table>
<ASSESSMENTS>
  <STUDENT name = "Smith">
    <MARK theCourse = "4BA1"> 75
    </MARK>
    <MARK theCourse = "4BA5"> 99
    </MARK>
  </STUDENT> ...
  <COURSE name = "4BA1" takenBy = "Smith, Jones, ...">
    <MARK> 60 </MARK>
  </COURSE>
  ...
</ASSESSMENTS>

Describes mark for individual student
Describes average mark for course
Useful Properties of a Node

• Name (Except root, text and comment nodes)
  – Qualified by the namespace, such as <xm:term> -"xm" is the namespace, "term" is the local part. They can be accessed by using the functions name(), namespace-uri(), local-name().

• String-value
  – E.g. text if text node, comment text if comment node, attribute value if attribute node.
  – It can be accessed by the string() function.

• Child
  – List of child nodes

• Parent
  – Every node except root

• Has-attribute
  – List of attribute nodes associated with element node

• Has-namespace
  – List of namespace nodes associated with element node
Data Types

• XPath 1.0
  - A **number** : stored as a floating point
  - A **string** : a sequence of characters
  - A **boolean** : a true or false value
  - A **node set** : an unordered collection of unique nodes

• XPath 2.0
  - Data types in XPATH 1.0 are pretty primitive
  - Supports data types taken from XML Schema
  - XPath 2.0 defines five additional datatypes
    - anyAtomicType, untyped, untypedAtomic, dayTimeDuration, and yearMonthDuration.
XPath Evaluation Results

- The result of an XPath 1.0 expression is a node set
  - Node sets
    - duplicates are not allowed (unique)
    - No order is implied

- The result of an XPath 2.0 expression is a sequence
  - Sequences
    - Are ordered collections (list)
    - Zero, one or more items are allowed (or just nodes)
    - duplicates are allowed
    - The empty sequence is a valid sequence

- Items
  - an item is a reference to a node or an atomic value
  - Each item has a value (42) as well as a type (xs:integer)

- Atomic values (integers, string, booleans, etc.)
Location Paths

• A location path (or path expression) identifies a set of nodes within an XML document
• A location path consists of a series of steps
• Simple path descriptors are sequences of location steps separated by slashes (/)
  – E.g. bookstore/book
• By default trying to match any child nodes from current location

• Document Root:
  – A forward slash (/) at the start of a location path indicates that the starting position for the context node is the document (root) node
Absolute & Relative Paths

• A location path can be absolute or relative.

• If the location path starts with the root node (/) then you are using an absolute location path.
  For example,
  – /root/node1/node2
  – /html/body/h3

• If the location path begins with the name of a descendant, you're using a relative location path.
  For example,
  – node1/node2
  – //node1/node2 (anywhere in document)
The XPath data model treats an XML document as a tree of nodes, based on DOM.

Formally, a tree is a connected, acyclic, undirected graph.
Example: /ASSESSMENTS/
STUDENT/MARK

<ASSESSMENTS>
  <STUDENT name = “Smith”>
    <MARK theCourse = “4BA1”> 75 </MARK>
    <MARK theCourse = “4BA5”> 99 </MARK>
  </STUDENT>
  ...

  <COURSE name = “4BA1”, takenBy = “Smith, Jones, ... “>
  </COURSE>
  ...

</ASSESSMENTS>

Describes the set with these two MARK element nodes as well as any other MARK elements nodes for any other STUDENT.
Namespaces

• Any path expression can use a QNAME (prefix:local-name),
  For example, //foo:book
  – Selects all ‘book’ elements in the document that belong to the foo namespace
• Matching is based on the local name and the namespace name (and not the prefix)
• The prefix binding to a namespace (e.g., foo to http://foo.example.com) is not part of the path expression.
  – it is defined externally (application specific)
• A path expression without a prefix will only match elements without an associated namespace
Skipping Levels

• A double forward slash (//) matches any descendent nodes below the current location

• For example
  /section//cite
  – will match all ‘cite’ elements that are descendants of ‘section’

  //author
  – will match all ‘author’ elements in the document
Example: //MARK

<ASSESSMENTS>
  <STUDENT name = “Smith”>
    <MARK theCourse = “4BA1”> 75 </MARK>
    <MARK theCourse = “4BA5”> 99 </MARK>
  </STUDENT> ...

  <COURSE name = “4BA1”, takenBy = “Smith, Jones, ... “>
    <MARK> 60 </MARK>
  </COURSE> ...
</ASSESSMENTS>

Still returns nodes from the document with a node named “MARK” but this time not just those noted in student assessment statements e.g. a mark allocated to a course by an external examiner
Select Parent and Ancestors

• From the context node you can access your parent and ancestors
• ‘..’ matches the parent of the current context node
  ../section
• Navigate just like directories
• You can go back many levels
  ../../../body
Select Unknown Elements (*)

- XPath wildcard (*) put in place in a tag represents any one tag

- Example /*/*/MARK will return any MARK object appearing at the third level of nesting in the document
Example: /ASSESSMENTS/*

```
<ASSESSMENTS>
  <STUDENT name = “Smith”>
    <MARK theCourse = “4BA1”> 75 </MARK>
    <MARK theCourse = “4BA5”> 99 </MARK>
  </STUDENT> ...

  <COURSE name = “4BA1”, takenBy = “Smith, Jones, ...”>
    <MARK> 60 </MARK>
  </COURSE> ...
</ASSESSMENTS>
```

Return all nodes at first level of nesting in the document
Select Attribute @

- Attributes are referred to by putting ampersand (@) before the name
- Appear in the path as if nested within the tag

- For example
  /book/@lang
  - Select the ‘lang’ attribute of books
Example: /ASSESSMENTS/*/@name

<ASSESSMENTS>
  <STUDENT name = “Smith”>
    <MARK theCourse = “4BA1”> 75 </MARK>
    <MARK theCourse = “4BA5”> 99 </MARK>
  </STUDENT> ...

  <COURSE name = “4BA1”, takenBy = “Smith, Jones, … “>
    <MARK> 60 </MARK>
  </COURSE> ...

</ASSESSMENTS>

Select all “name” attributes appearing at first level of nesting
Select Several Paths (|)

- By using the union (|) operator in an XPath expression you can select several paths.

- For example
  
  
  - Selects all the ‘title’ elements AND the ‘price’ elements within the ‘book’ elements.
Predicates - Conditional Matching

- A tag in a path that is followed by a condition [...] will ensure that only nodes that satisfy the condition are included in the resultant set.

- Example

  /bookstore/book[price>35.00]

  - Selects the ‘books’ elements of a ‘bookstore’ where the ‘price’ element has a value greater than 35.00.
Example:

/ASSESSMENTS/STUDENT[MARK > 80]

<ASSESSMENTS>

-STUDENT name = “Smith”>
  -MARK theCourse = “4BA1”> 75 </MARK>
  -MARK theCourse = “4BA5”> 99 </MARK>
</STUDENT> ...

-COURSE name = “4BA1”, takenBy = “Smith, Jones, ...”>
  -MARK> 60 </MARK>
</COURSE> ...

</ASSESSMENTS>

This object is returned as it satisfies the condition
Example Attribute in the selection:

```
/ASSESSMENTS/STUDENT/MARK[@theCourse = '4BA1']
```

```
<ASSESSMENTS>
  <STUDENT name = "Smith">
    <MARK theCourse = "4BA1"> 75 </MARK>
    <MARK theCourse = "4BA5"> 99 </MARK>
  </STUDENT> ...

  <COURSE name = "4BA1", takenBy = "Smith, Jones, ...">
    <MARK> 60 </MARK>
  </COURSE> ...
</ASSESSMENTS>
```

This object is returned as well as any other student mark objects for 4BA1.
Over to you...

```
<database>
  <person age='34'>
    <name>
      <title> Mr </title>
      <firstname> John </firstname>
      <firstname> Paul </firstname>
      <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
  </person>

  <person>
    <name>
      <firstname> Mary </firstname>
      <surname> Donnelly </surname>
    </name>
  </person>
</database>
```

- /database
- //surname
- /*/person[@age]
- /*/person/@age
Over to you...

<database>
  <person age='34'>
    <name>
      <title> Mr </title>
      <firstname> John </firstname>
      <firstname> Paul </firstname>
      <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
  </person>
  <person>
    <name>
      <firstname> Mary </firstname>
      <surname> Donnelly </surname>
    </name>
  </person>
</database>

- /database
- //surname
- /*/person[@age]
- /*/person/@age
Over to you…

<database>
  <person age='34'>
    <name>
      <title> Mr </title>
      <firstname> John </firstname>
      <firstname> Paul </firstname>
      <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
  </person>
  <person>
    <name>
      <firstname> Mary </firstname>
      <surname> Donnelly </surname>
    </name>
  </person>
</database>

- /database
- //surname
  - /*/person[@age]
  - *//person/@age
Over to you...

<?xml version="1.0"?>
<database>
  <person age='34'>
    <name>
      <title> Mr </title>
      <firstname> John </firstname>
      <firstname> Paul </firstname>
      <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
  </person>
  <person>
    <name>
      <firstname> Mary </firstname>
      <surname> Donnelly </surname>
    </name>
  </person>
</database>

- /database
- //surname
- /*/person[@age]
- /*/person/@age
Over to you...

<database>

<person age='34'>
  <name>
    <title> Mr </title>
    <firstname> John </firstname>
    <firstname> Paul </firstname>
    <surname> Murphy </surname>
  </name>
  <hobby> Football </hobby>
  <hobby> Racing </hobby>
</person>

<person>
  <name>
    <firstname> Mary </firstname>
    <surname> Donnelly </surname>
  </name>
</person>

</database>

- /database
- //surname
- /*/person[@age]
- /*/person/@age
Anatomy of a Location Step

- A step in an XPath expression consists of three parts: an *axis*, a *node* test, and zero or more *predicate* tests.

```
Child::Student[name="paul"]
```

- Specifies direction to go in document tree
- Tests whether nodes encountered should be selected for next step
- Filters nodes selected by the node test
Axis

- An **axis** defines the nodes selected relative to the current node. In XPath there are 13 axes defined:

  - ancestor
  - ancestor-or-self
  - attribute
  - child
  - descendant
  - descendant-or-self
  - following
  - following-sibling
  - Namespace
  - parent
  - preceding
  - preceding-sibling
  - self
Axis ancestor::

- **ancestor**
  Selects all the nodes that are ancestors of the origin

- Syntax
  ancestor::node
Axis ancestor-or-self::

- **ancestor-or-self**
  Selects the same nodes as the ancestor axis, but starting with the origin node

- Syntax
  ancestor-or-self::node
Axis attribute::

- **attribute**
  If the origin node is an element, this axis selects all its attribute nodes. Otherwise, it selects nothing (an empty sequence). The order for attributes is arbitrary.

- **Syntax**
  1. attribute:::lang
  2. @lang
Axis child::

- **child**
  Selects all the children of the origin node, in document order.

- **Syntax**
  1. child::node
  2. /node
Axis descendant::

• **descendant**
  Selects all the children of the origin node, and their children, and so on recursively. The resulting nodes are in document order.

• Syntax
  descendant::node
Axis descendant-or-self::

- **descendant-or-self**
  This is the same as the descendant axis, except that the first node selected is the origin node itself.

- Syntax
  1. Descendant-or-self::node
  2. //
Axis following::

• **following**
This selects all the nodes that appear after the origin node in document order, excluding the descendants of the origin node

• Syntax
following::node
Axis following-sibling::

- **following-sibling**
  This selects all the nodes that follow the origin node in document order, and that are children of the same parent node.

- **Syntax**
  Following-sibling::node
Axis namespace::

- **namespace**
  If the origin node is an element, this axis selects all the namespace nodes that are in scope for that element; otherwise, it is empty. The order of the namespace nodes is undefined.

- **Syntax**
  namespace::node
Axis parent::

- **parent**
  This axis selects a single node, the parent of the origin

Syntax
1. parent::node
2. ..
Axis proceeding::

• **proceeding**
  This selects all the nodes that appear before the origin node, excluding the ancestors of the origin node.

• **Syntax**
  proceeding::node
Axis proceeding-siblings::

• **proceeding-siblings**
  This selects all the nodes that precede the origin node, and that are children of the same parent node

• Syntax
  proceeding-siblings::node
Axis self::

- **self**
  
  This selects a single node, the origin node itself. This axis will never be empty.

Syntax

1. `self::node`
2. `.`
There are several directions/axes we can traverse from a node:

- **parent::**
- **preceding-sibling::**
- **following-sibling::**
- **child::**

```
<?xml version='1.0' ?>

<root>
  <aunt />  <parent>
    <sister />  <self>
      <son>
        <grandchild />  </son>
      </self>
    <child />  <daughter>
      <grandchild />  </daughter>
    </parent>
  <brother />
</root>
```
Axes example (2)

<?xml version='1.0' ?>
<root>
  <aunt />
  <parent>
    <sister />
    <self>
      <son>
        <grandchild />
      </son>
      <child />
      <daughter>
        <grandchild />
      </daughter>
    </self>
    <brother />
  </parent>
  <uncle></uncle>
</root>
Node Tests

• A node test defines the nodes to select.

• Test the node in the tree document

  – **By name of node**: test the node to see if it has an element name the same as that specified. E.g. child::Student would test if the child node has an element named “Student”

  – **By kind/type of node**: test the node if is a text, comment, or processing instruction node. E.g. text()

  – **By the schema defined type**
Node Tests: By Name

- selects nodes based on the node name

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Match all elements</td>
</tr>
<tr>
<td>@*</td>
<td>Select all the attributes</td>
</tr>
<tr>
<td>xm:*()</td>
<td>Matches all element nodes in the namespace with the &quot;xm&quot; prefix</td>
</tr>
<tr>
<td>*:term</td>
<td>Any name matching the local name &quot;term&quot;, regardless of namespace</td>
</tr>
</tbody>
</table>
Node Tests: by Type

• selects nodes based strictly upon their node type
• In XPath 1.0

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node()</td>
<td>True for a node of any type.</td>
</tr>
<tr>
<td>text()</td>
<td>True for a text node.</td>
</tr>
<tr>
<td>comment()</td>
<td>True for a comment node.</td>
</tr>
<tr>
<td>processing-instruction()</td>
<td>True for a processing instruction node.</td>
</tr>
</tbody>
</table>

• In XPath 2.0

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>element()</td>
<td>Matches any element node</td>
</tr>
<tr>
<td>attribute(“src”)</td>
<td>Matches any attribute named “src”</td>
</tr>
<tr>
<td>item()</td>
<td>Retrieves any item (node or atomic value)</td>
</tr>
<tr>
<td>element(“type”)</td>
<td>Matches any element node named “type”</td>
</tr>
</tbody>
</table>
## Node Tests: by Schema type

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>element(*, xs:date)</td>
<td>Any element of (simple) type xs:date</td>
</tr>
<tr>
<td>element (*, caption)</td>
<td>Matches any element node whose (schema) type is “caption” (or a type derived from “caption”) User defined type.</td>
</tr>
</tbody>
</table>
Predicates

- A predicate refers to the expressions (conditions) written in square brackets []. They restrict/filter the selected nodes in a node set.

  - Attribute Tests: @ indicates attribute
  - Boolean Tests (Functions): boolean, true, false, not, ...
  - Node Set Tests (Functions): count, id, position, last, ...
  - Number Tests (Functions): ceiling, floor, round, sum, ...
  - String Tests (Functions): concat, contains, string-length, substring, translate, ...

- There is no limit to the number of predicates in a step
  - Keywords (and, or), consecutive predicates [][]
Path Operators and Special Characters

- XPath expressions are constructed using the following operators and special characters

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Child operator, selects immediate children</td>
</tr>
<tr>
<td>//</td>
<td>Recursive descent, searches for the specified element at any depth</td>
</tr>
<tr>
<td>.</td>
<td>Indicates the current context (node)</td>
</tr>
<tr>
<td>..</td>
<td>The parent of the current context node</td>
</tr>
<tr>
<td>*</td>
<td>Wildcard, selects all elements regardless of the element name</td>
</tr>
<tr>
<td>@</td>
<td>Attribute, prefix for an attribute name</td>
</tr>
<tr>
<td>@*</td>
<td>Attribute wildcard, selects all attributes regardless of name</td>
</tr>
<tr>
<td>:</td>
<td>Namespace separator</td>
</tr>
<tr>
<td>()</td>
<td>Groups operations to explicitly establish precedence</td>
</tr>
<tr>
<td>[]</td>
<td>Applies a filter pattern</td>
</tr>
</tbody>
</table>
Operators

• An XPath 1.0 expression returns either a node-set, a string, a Boolean, or a number.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;+&quot;, &quot;-&quot;, &quot;+&quot;, &quot;div&quot; (divide), &quot;mod&quot;</td>
<td>Arithmetic operators</td>
</tr>
<tr>
<td>&quot;and&quot;, &quot;or&quot;, &quot;not() &quot;</td>
<td>Boolean operators</td>
</tr>
<tr>
<td>&quot;=&quot;&quot;, &quot;!=&quot;, &quot;,&lt;&quot;, &quot;&gt;&quot;, &quot;,&lt;=&quot;&quot;, &quot;,&gt;=&quot;&quot;</td>
<td>Comparison operators</td>
</tr>
</tbody>
</table>
XPath Functions

- Functions to manipulate strings:
  - concat(), substring(), contains(), substring-before(), substring-after(), translate(), normalize-space(), string-length()

- Functions to manipulate numbers:
  - sum(), round(), floor(), ceiling()

- Functions to get properties of nodes:
  - name(), local-name(), namespace-uri()

- Functions to get information about the processing context:
  - position(), last()

- Type conversion functions:
  - string(), number(), boolean()
XPath examples

<doc type="book" isbn="1-56592-796-9">
  <title>A Guide to XML</title>
  <author>Norman Walsh</author>
  <chapter>[...]</chapter>
  <chapter>
    <title>What Do XML Documents Look Like?</title>
    <paragraph>If you are [...]</paragraph>
    <ol>
      <item><paragraph>The document begins [...]</paragraph></item>
      <item><paragraph type="warning">There's no document [...]</paragraph></item>
      <item><paragraph>Empty elements have [...]</paragraph></item>
      <paragraph>In a very [...]</paragraph>
    </ol>
    <paragraph>XML documents are [...]</paragraph>
  </chapter>
</doc>

//paragraph

<paragraph>If you are [...]</paragraph>
<paragraph>A few things [...]</paragraph>
<paragraph>The document begins [...]</paragraph>
<paragraph type="warning">There's no document [...]</paragraph>
<paragraph>Empty elements have [...]</paragraph>
<paragraph>In a very [...]</paragraph>
<paragraph>XML documents are [...]</paragraph>

//ol//paragraph[@type='warning']

<paragraph type="warning">
  There's no document [...]
</paragraph>

doc/chapter[2]/ol/item[position()=last()]

<item><paragraph>Empty elements have [...]</paragraph>
  <paragraph>In a very [...]</paragraph></item>
Summary

• Selects (a set of) ELEMENTs within an XML document based on
  – Conditions
  – Hierarchy

• Usage
  – Retrieving info from a single XML document
  – Applying XSL style sheet rules
  – Making XQueries

© Declan O’Sullivan
Tutorial & Exercise


- Other XPath tools
  - XPath Checker – as Firefox addon
  - PathEnq – as Chrome plugin

- Form contents -
  - books.xml, books2.xml
  - booksTable.xsl OR booksList.xsl