



University of Dublin
Trinity College



4D2b – Navigating an XML Document

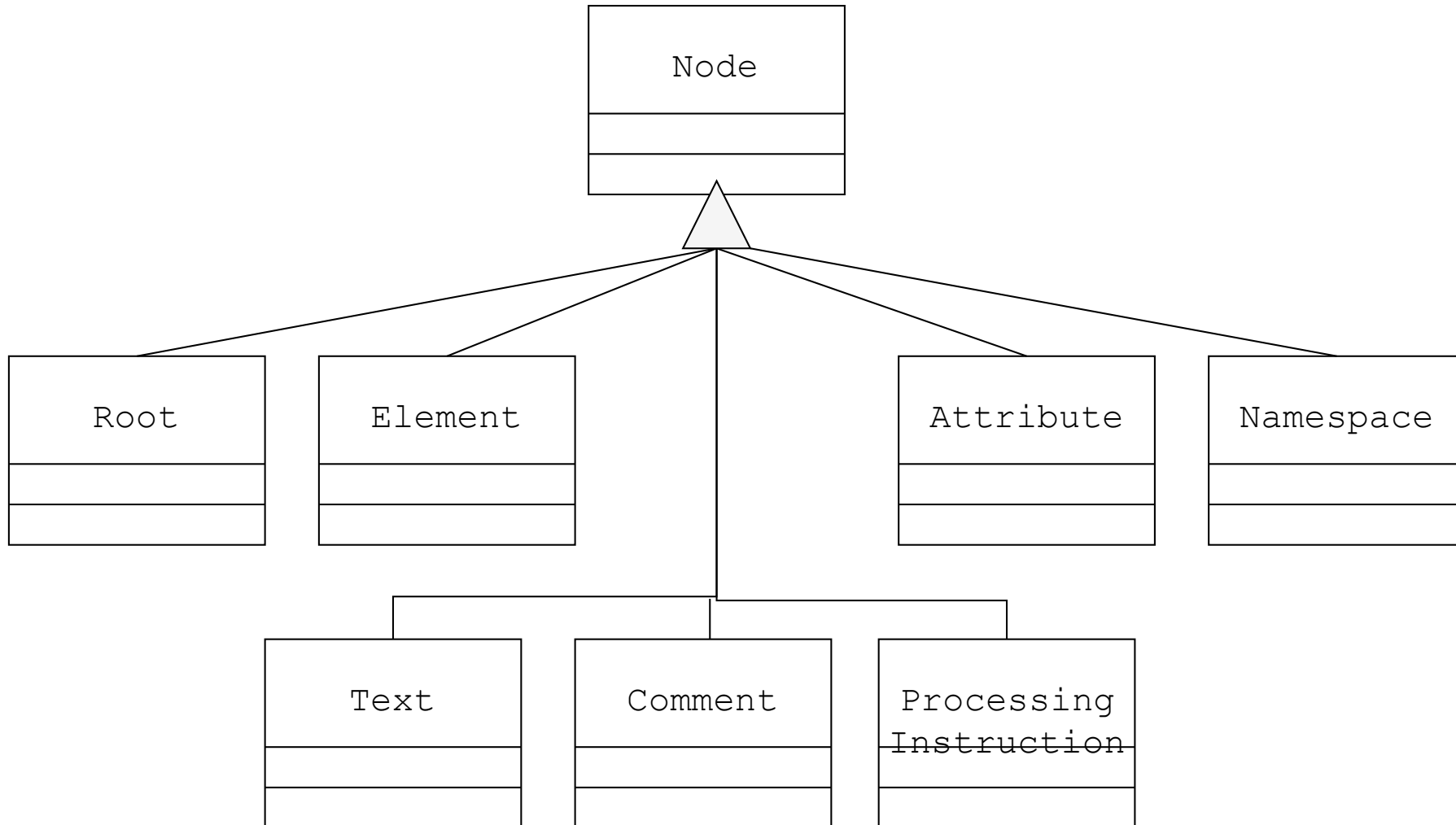
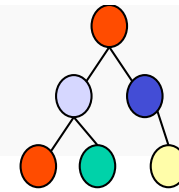
Owen.Conlan@scss.tcd.ie

What is XPath?

- Addresses parts of an XML document
- W3C Recommendation (16 November 1999)
- Expression language
- Wildcards allowed
- Provides basic facilities for 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



Nodes in a Tree Model



Useful Properties of a Node

- Name
 - Except root, text and comment nodes
- String-value
 - E.g. text if text node, comment text if comment node, attribute value if attribute node
- 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



Path Descriptors

- Simple path descriptors are sequences of *location steps* separated by *slashes (/)*
- By default trying to match any child nodes from current location
- Sequence begins with *//*
 - Short hand trying to match any descendent nodes below current location



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



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



Wildcard *

- A asterix (*) 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



Attribute @

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



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



Predicate Filters []

- 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:

`/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...

```
<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



More Generally: Location Steps

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

Specifies direction to go in document tree

Tests whether nodes encountered should be selected for next step

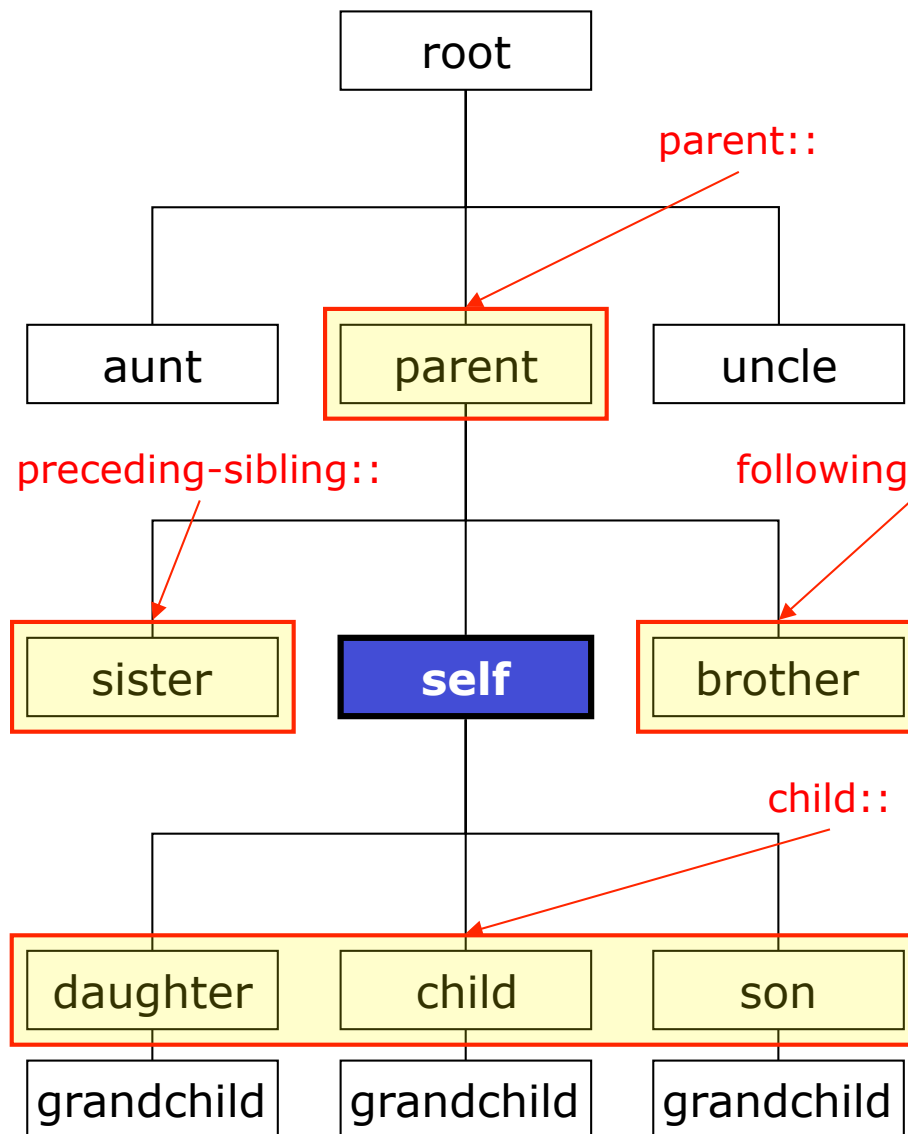
Filters nodes selected by the node test

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



Axes spec (1)

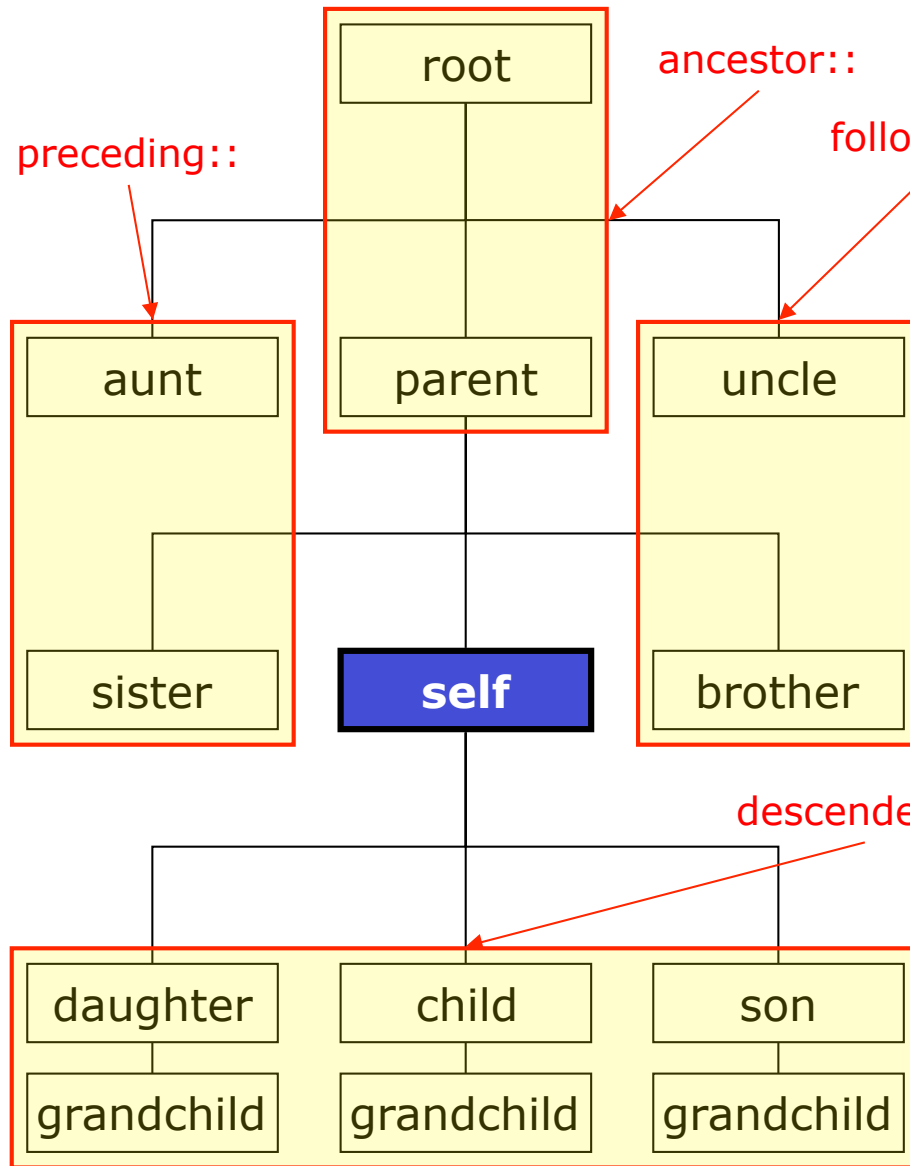
There are several directions/axes we can traverse from a node



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



Axes spec (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

- The default is to 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"
- Tests for checking element, attribute, and namespace name
- Tests for checking if the node is a text, comment, or processing instruction node
 - E.g. `text()`



Predicate Filters

- [] are used to hold predicates (conditions)
 - 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, ...
 - Multiple Tests
 - Keywords (and, or), consecutive predicates [][]



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>
    <paragraph>A few things [...]</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>
        <paragraph>In a very
          [...]</paragraph></item>
    </ol>
    <paragraph>XML documents are
      [...]</paragraph>
    <section>[...]</section>
    [...]
  </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 XQuerys



Tutorial

- <https://www.cs.tcd.ie/Owen.Conlan/php/xpath/xpathlab.html>
- Form contents -
 - books.xml
 - booksTable.xsl OR booksList.xsl

