Dynamic Web Application Development using XML and Java

by David Parsons

Chapter 6

Transforming XML: XPath and XSLT

Learning Objectives

- To understand the syntax of XPath expressions
- To understand and be able to navigate the tree structure of XML documents
- To be able to construct XPath expressions that will extract nodes from an XML document
- To be able to write XSL Transformations that generate output documents in XML or XHTML
- To understand the use of different character encodings when generating XML documents
- To understand the difference between output driven and input driven transformations

XPath: Querying XML

- XPath (XML Path) provides a language for accessing parts of an XML document. It is used by both eXtensible Stylesheet Language Transformations (XSLT) and the XML Pointer Language (XPointer)
- The main role of XPath is to provide an expression syntax appropriate for selecting one or more nodes from an XML document.
  - To extend this role it also provides some facilities for manipulating strings, numbers and Booleans.
  - In the context of XSLT it is used for pattern matching, which is the aspect we will focus on here.

XPath And XML Trees

- To understand the way that the XPath data model works, we need to visualise an XML document as a tree of nodes
- There are seven types of node:
  - root nodes
  - element nodes
  - text nodes
  - attribute nodes
  - namespace nodes
  - processing instruction nodes
  - comment nodes
- The main nodes that we process in XPath expressions will be element and attribute nodes.

Example XML Document

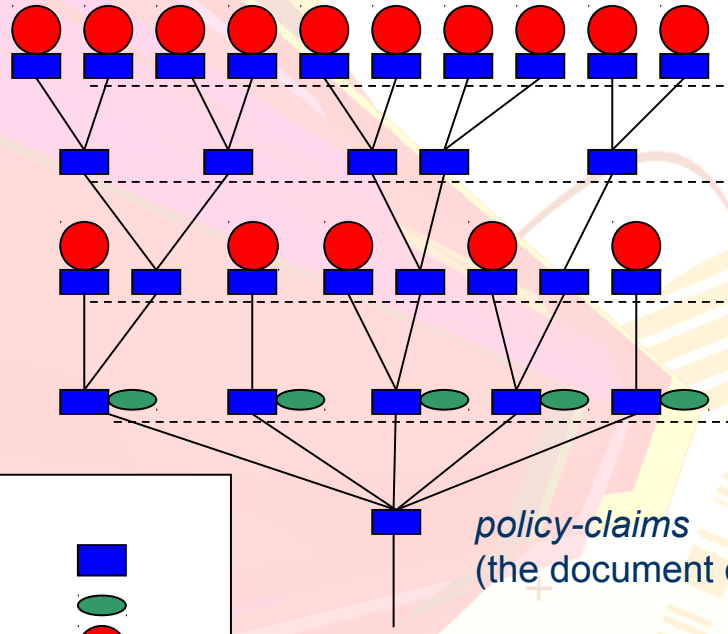
- Represents claims made against policies.
- The root ‘policy-claims’ element contains one or more ‘policy’ elements
- each policy has
  - a ‘type’ attribute
  - a ‘policy-holder’ element (a string)
  - optional ‘claims’ element
    - If present, the ‘claims’ element will contain one or more ‘claim’ elements, and each of these will contain a ‘year’ (Gregorian calendar ‘gYear’ type) and ‘details’ (a string):

- XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="policy-claims">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="policy" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="policy-holder" type="xs:string"/>
       <xs:element name="claims" minOccurs="0" maxOccurs="1">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="claim" minOccurs="1" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="year" type="xs:gYear"/>
             <xs:element name="details" type="xs:string"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="type" type="xs:string"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

policy-claims.xsd

XML Document As A Tree



Within a *claim* there are *year* and *details* elements containing text nodes

A *claims* element has one or more *claim* elements

Here, there are *policy-holder* elements with text nodes and there may be a *claims* element

policy elements with *policy-holder* branches, *type* attributes and optional *claims* elements

policy-claims
(the document element)

/ (the root)

KEY:
Element
Attribute
Text node

‘Family Tree’ Vocabulary

- XPath syntax refers to ‘parent’, ‘child’, ‘ancestor’ and ‘descendent’ nodes.
  - ‘policy-holder’ is a child node of ‘policy’.
  - ‘policy’ is the parent of ‘policy-holder’.
  - ‘policy’ is an ancestor of ‘claim’
  - ‘claim’ is a descendent of ‘policy’.

Document Order

- As a consequence of having a tree-like structure, the nodes in an XML document appear in a document order clockwise from the root.
  - ‘policy-claims’ comes first
  - Then the first ‘policy’ node
  - Followed by a ‘claims’ node
  - Inside the ‘claims’ node is a ‘claim’, followed by ‘year’ and ‘details’ nodes, etc.

XPath Expressions and the Document Order

- XPath takes account not only of the tree structure of an XML document, but also of the document order.
- When several elements are returned by an XPath expression, they are returned in the same order as they are encountered in the document.
- Attributes however, do not have a document order, so if more than one attribute is returned the order is not fixed.

The Context – The Starting Point of an XPath Expression

- XPath is primarily a way of writing expressions that return an object that may be one of the following:
  - A set of nodes
  - A Boolean value
  - A floating-point number
  - A string of Unicode characters
- In order to evaluate an expression, the XPath query has to start at a particular node.
- The starting node used for the query is known as the *context*

Location Paths

- The most important part of XPath is the ability to express a *location path* to identify parts of an XML document.
  - Much of this syntax is based on the concepts of child, ancestor and descendent nodes.
  - child::* selects all elements that are children of the current context node.
  - Ancestors and descendent nodes are indicated by ancestor:: and descendent::
  - In addition, we can select attribute nodes by using the attribute:: prefix in an XPath expression.
    - attribute::type would select the 'type' attribute of the 'policy' node, if that was the current context node.

Location Paths

- Either relative to the current node or absolute (from the root node).
- The path from a parent node to a child node is indicated by the '/' character.
- A relative path begins with the name of a node.

```
child::policy/child::policy-holder
```

  - only makes sense if the current context is the 'policy-claims' node.
- An absolute path begins with the root node ('/')

```
/child::policy-claims/child::policy/child::policy-holder
```

- current context does not matter

Abbreviated syntax

- The most important abbreviation is that ‘child::’ can be left out of the location path. In effect, ‘child::’ is the default, so the location path

```
/policy-claims/policy/policy-holder
```

- Is an abbreviation of:

```
/child::policy-claims/child::policy/child::policy-holder
```

- There is also an abbreviation for attributes; attribute:: can be abbreviated to @
- Instead of referring to attribute::type in a location path we could use the abbreviated form @type

XPath Operators

Operator	Meaning
/	Child operator. Selects children of whatever is to the left of it. If there is nothing to the left, it starts at the root element. In XPath a 'child' is an immediate child (e.g. grandchildren are not children)
//	Stands for any number of intermediate elements, to express ancestor - descendant relationships
.	The current context (the current node)
..	The parent of the current node
*	Wildcard. Matches all elements
@	Distinguishes attributes from elements (attribute prefix)

Example – Policy Claims

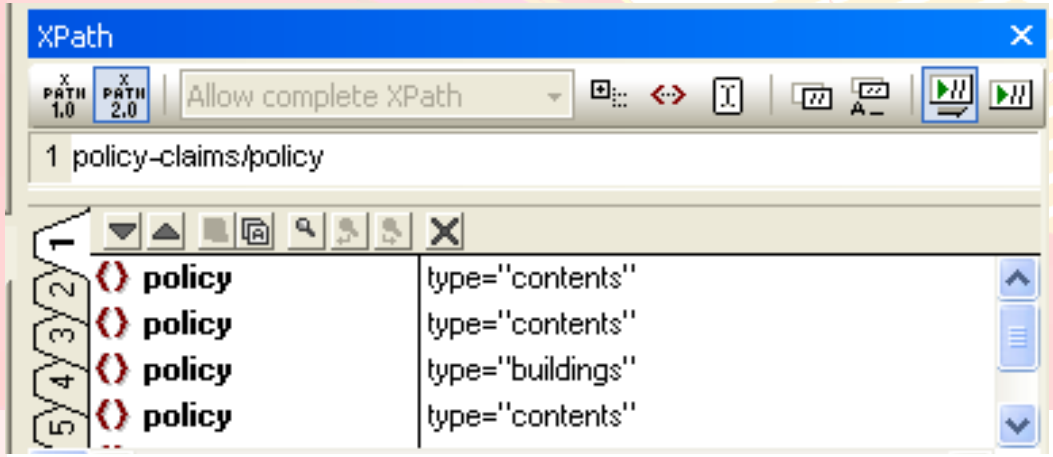
```
<?xml version="1.0"?>
<policy-claims xmlns:xsi=… >
 <policy type="contents">
  <policy-holder>A. Liu</policy-holder>
  <claims>
   <claim>
    <year>2002</year>
    <details>Stolen TV</details>
   </claim>
  </claims>
 </policy>
 <policy type="contents">
  <policy-holder>B. Singh</policy-holder>
 </policy> <policy type="buildings">
  <policy-holder>C. Jones</policy-holder>
   <claims>
    <claim>
     <year>2004</year>
      <details>Fire damage to Kitchen</details>
    </claim>
  </claims>
 </policy>
```

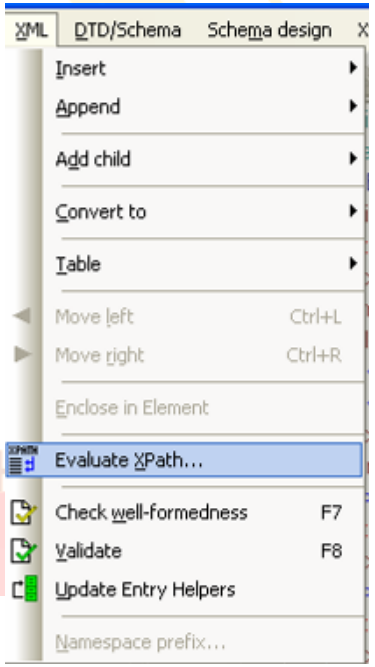
```
<policy type="contents">
 <policy-holder>D. Umaga
 </policy-holder>
  <claims>
   <claim>
    <year>1998</year>
    <details>Stolen bike</details>
   </claim>
   <claim>
    <year>2005</year>
    <details>Dropped Ming Vase
    </details>
   </claim>
  </claims>
 </policy>
 <policy type="buildings">
  <policy-holder>E. Tolstoy
  </policy-holder>
 </policy>
</policy-claims>
```

policy-claims.xml

Testing XPath In XML Spy

- XML Spy has an interactive XPath tool
  - Choose the XPath tab on the output window
  - Type your XPath queries into the text box and the result will be evaluated dynamically





Accessing Child Nodes

- One approach is to use a series of 'child' operators to specify the full path through the document, e.g.

```
/policy-claims/policy/policy-holder
```

- The resulting nodes would therefore be the five policy holders.

Accessing Descendents

- // selects an element without specifying the full path.
- Using the wildcard character (*) matches all the sub-elements beneath the selected nodes.
- This expression, which uses both the '//' operator and the wildcard, will return all the 'year' and 'details' elements'

```
//claim/*
```

- These would be the resulting nodes (note they are returned in the document order)

```
Year - 2002
Details - Stolen TV
Year - 2004
Details - Fire damage to Kitchen
Year - 1998
Details - Stolen bike
Year - 2005
Details - Dropped Ming Vase
```

Filtering

- Searching for specific elements, attributes or values.
- XPath filter patterns use square brackets and evaluate to a Boolean value

```
/policy-claims/policy[claims]
```

- Matches only policy elements that contain at least one ‘claims’ element child, in the case of our example document returning the first three ‘policy’ nodes.
- We can also query the data, e.g.

```
//claim[year = 2002]
```

- This would select only one of the ‘claim’ nodes in our document.
- There is also the usual set of relational operators (>, <, >=, <=) and the ‘!=’ symbol for ‘not equals’, that work with numeric data.

Attributes In XPath Queries

- Attributes and elements are treated in a similar way
  - Only difference is the use of the '@' symbol.
- The following expression will return the attribute 'type' nodes that have the value 'contents'

```
/policy-claims/policy[@type = "contents"]
```

- If 'type' had been an element, then the expression would be identical except for not including the '@'.
- Here, we use the query to select buildings policies that have claims made against them

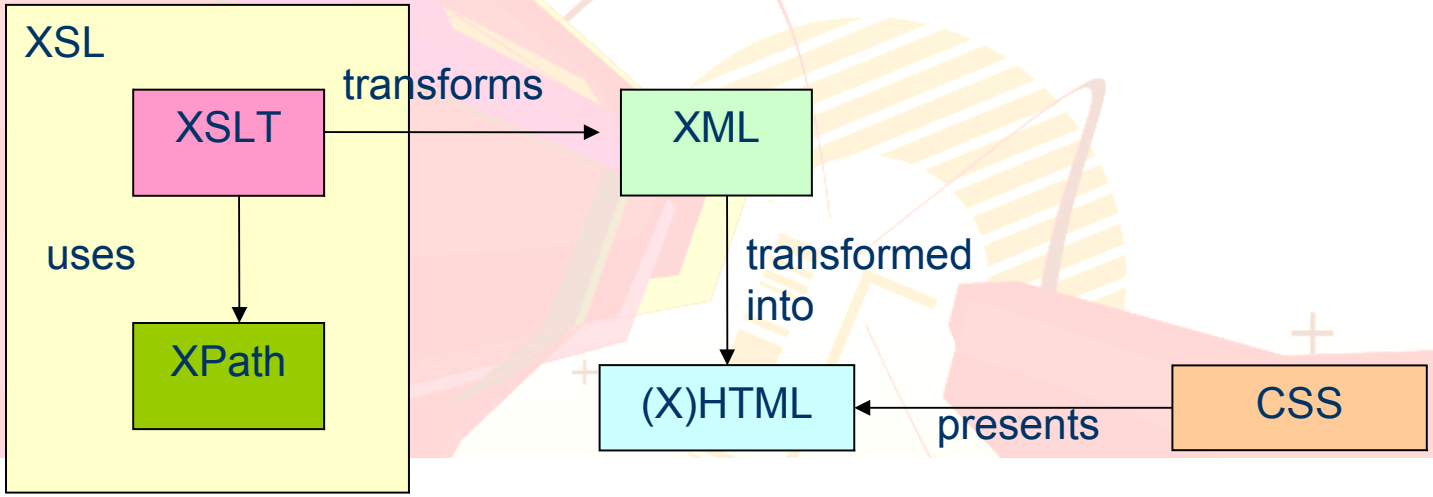
```
/policy-claims/policy[@type = "buildings"][claims]
```

eXtensible Stylesheet Language Transformations (XSLT)

- XSLT can be used to generate web pages from XML documents.
- XSLT is part of the Extensible Stylesheet Language (XSL), a set of standards for XML document transformation and presentation.
- It consists of three parts; XSLT, XPath and XSL-FO
  - XSL Transformations (XSLT)
    - can transform XML documents into various types of other document
  - XML Path (XPath)
    - an expression language that can select certain parts of an XML document.
  - XSL Formatting Objects (XSL-FO)
    - a way of formatting XML in presentational formats other than markup, e.g. PDF (Portable Document Format)

XSLT, HTML and CSS

- Although XSL refers to stylesheets, it does not replace CSS
  - XSLT, HTML and CSS are complementary



Processing XSLT

- An XSLT stylesheet, or transform, consists of a number of aspects.
- XPath is used to identify content from the input document that will be included in the output document.
- There will also be other parts of the transform that are meant to be used directly in the output document, for example (X)HTML tags.
- XSLT uses template matching to process different parts of the input document in different ways
  - Nodes that match the template's XPath expression are included in the output document.

XSL Namespace

- The first element of an XSL document consists of a version number (1.0 or 2.0) and a namespace reference
  - The usual prefix for the XSLT namespace is ‘xsl’:

```
<elementname version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
…
</elementname>
```

  - The namespace reference is a good example of a URN. It does not represent a downloadable resource. However if you put the URN into a browser it identifies its purpose.

Stylesheets and Transforms

- The root element for an XSL transform can be either <xsl:stylesheet…>

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
```

- or <xsl:transform…>

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
```

- Both mean exactly the same thing
  - The ‘xsl’ can also be replaced by something else, but is the usual naming convention

Template Matching

- An XSL transform contains one or more <xsl:template…> elements
- <xsl:template…> elements have a 'match' attribute, the value of which is an XPath expression

```
<xsl:template match="XPath expression">
```

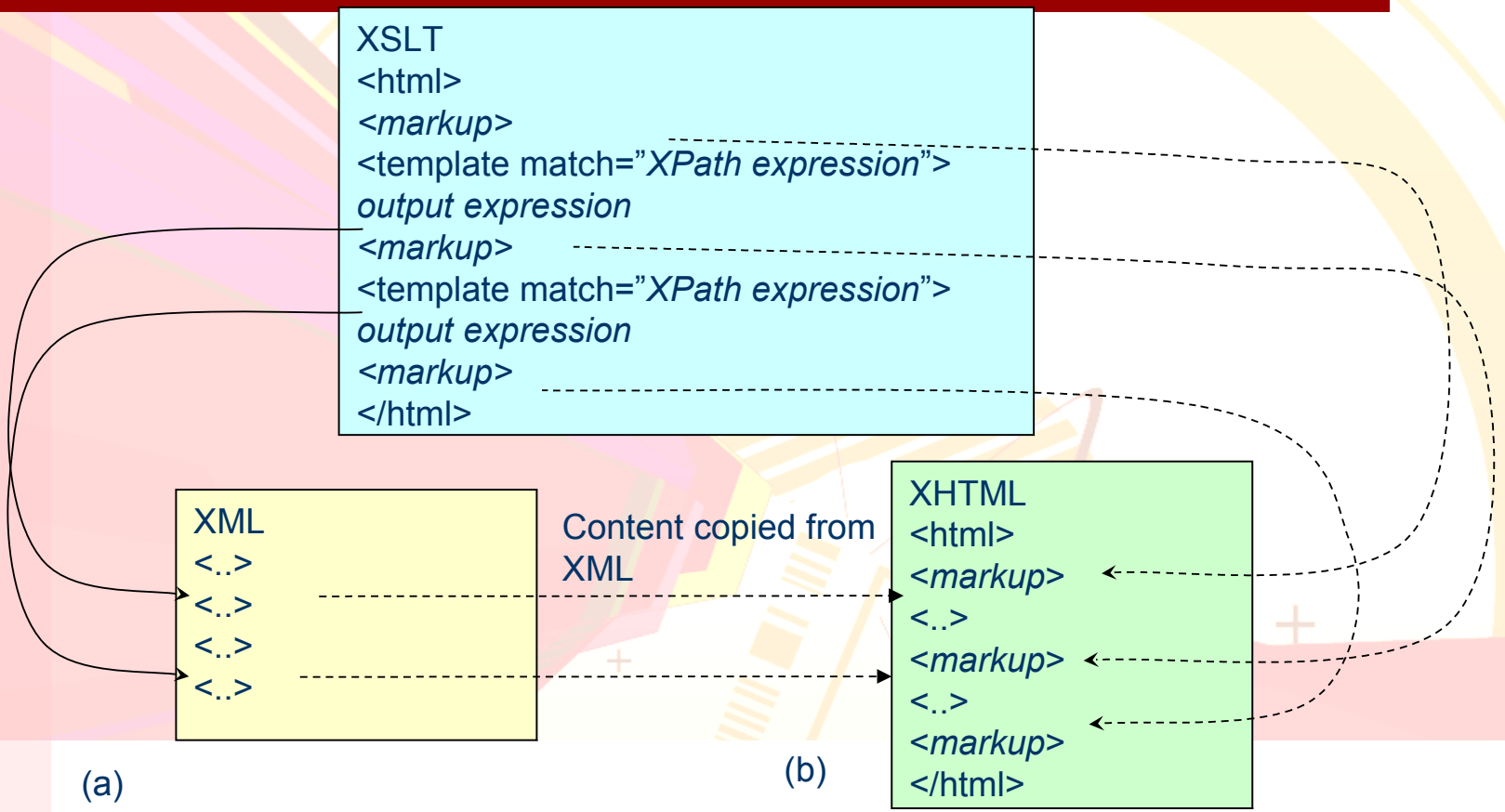
- This expression must match something in the XML document being processed

The Template Element

- The body of the template element defines what is to be sent to the output document if the element is matched
  - This can be a combination of XML from the document and other markup

```
<xsl:template match="XPath expression">
… specify what goes to the output document here
… this may be markup, and/or XSLT elements that
… process the input XML document
</xsl:template>
```

Combining XML and XHTML Markup



Matching elements

- The ‘match’ attribute of an ‘xsl:template’ start tag must contain a valid XPath expression.
  - To apply a template to the root element, for example, the value of the ‘match’ attribute is the XPath expression for the root element, which is ‘/’

```
<xsl:template match="/">
… define the transform for the root element here
</xsl:template>
```

  - Another example from the ‘policy-claims.xml’ document might match the ‘policy’ node, again using standard XPath

```
<xsl:template match="/policy-claims/policy">
… define the transform for the policy element here
</xsl:template>
```

Output Types

- XSL Transformations can generate output using three different methods:
  - xml, html or text
- The method can be specified by using the 'method' attribute of the 'xsl:output' element

```
<xsl:output method="xml" version="1.0"/>
```

- The default is XML document, so HTML or text must be specified for those types of output

```
<xsl:output method="html" version="4.0"/>
```

- However If the first non-XSL child node is <html>, then the output is automatically HTML instead of XML (not XHTML!).

```
<xsl:stylesheet version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
```

Linking an XSLT Stylesheet

- To link an XSLT to an XML document, we can add an XML stylesheet processing instruction to the top of the document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="policy-claims.xsl"?>
```

- This can be used in, for example, a browser
  - Not all processing applications need this instruction in the XML - some can apply the transform to the XML externally
  - Other stylesheet types can be used as well, for example CSS

Selecting Values From the XML

- In XSLT the <xsl:value-of..> element is used to select element or attribute values from the source document
- The 'select' attribute contains an XPath expression

```
<xsl:value-of select="XPath expression"/>
```

- The value returned from the XPath expression is inserted into the output document
- A single 'xsl:value-of' element will only match a single node from the source document, which will be the first one that it matches in the document order.

xsl:value-of (element)

- In this example, the value of the (first) policy-holder element is selected

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <HTML>
   <HEAD>
    <TITLE>Insurance Claims</TITLE>
   </HEAD>
   <BODY>
    <H1>Claimants</H1>
    <H2>
     <xsl:value-of select="//policy-holder"/>
    </H2>
   </BODY>
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

xsl:value-of, which uses the value of the element

Example6-1.xsl

an XPath expression

Resulting Document

- The result of the transformation in XMLSpy is:

```
<HTML>
 <HEAD>
  <META http-equiv="Content-Type" content="text/html; charset=UTF-16">
  <TITLE>Insurance Claims</TITLE>
 </HEAD>
 <BODY>
  <H1>Claimants</H1>
  <H2>A. Liu</H2>
 </BODY>
</HTML>
```

This was the value of the first matching element

Selecting Attributes

- 'xsl:value-of' can be used to select either element or attribute values

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<HTML>
 <HEAD>
  <TITLE>Insurance Claims</TITLE>
 </HEAD>
 <BODY>
  <H1> Claimants and policy types </H1>
  <H2>
    Name: <xsl:value-of select="policy-claims/policy/policy-holder"/> <br />
   Policy type: <xsl:value-of select="policy-claims/policy/@type"/>
  </H2>
 </BODY>
</HTML>
```

Note XHTML tag – what happens?

element

Example6-2.xsl

attribute

HTML Tag Output

- The HTML output is not XHTML
  - The
 tag is converted into 'legacy' html

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<TITLE>Insurance Claims</TITLE>
</HEAD>
<BODY>
 <H1> Claimants and policy types </H1>
 <H2>
 Name: A. Liu
 <br>
 Policy type: contents
 </H2>
</BODY>
</HTML>
```

Generating XHTML

- We can generate XHTML by setting the output method to ‘xml’ and adding public and system doctypes
  - Tags also need to be XHTML

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"
 doctype-public="-//W3C//DTD XHTML 1.1//EN"
 doctype-system=" http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd "/>
 <xsl:template match="/">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
   <head>
```

Example6-3.xsl

Modified Resulting Document

- The result of the modified transformation is an XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC
 "-//W3C//DTD XHTML 1.1//EN“
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd ">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Insurance Claims</title>
 </head>
 <body>
  <h1> Claimants and policy types </h1>
  <h2> Name: A. Liu
   <br />
   Policy type: contents
  </h2>
 </body>
</html>
```

Main changes to document

Iteration With <xsl:for-each…>

- So far, we have only been getting the first match from each <xsl:valueof…> element
- The <xsl:for-each…> element enables us to iterate over all the matching nodes in the XML document
  - Like <xsl:valueof…>, its 'select' attribute uses an XPath expression to find all the matching nodes

Iteration Example

- Here, the <xsl:for-each…> element selects all the policy nodes

```
<body>
  <h1>Claimants and policy types</h1>
    <xsl:for-each select="policy-claims/policy">
     <h2>Name: <xsl:value-of select="policy-holder"/>,
      Policy type: <xsl:value-of select="@type"/>
     </h2>
    </xsl:for-each>
  </h2>
</body>
```

Example6-4.xsl

These XPath expressions are relative to the policy node

Iteration Example Output

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd ">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Insurance Claims</title>
 </head>
 <body>
  <h1>Claimants and policy types</h1>
  <h2>Name: A. Liu, Policy type: contents</h2>
  <h2>Name: B. Singh, Policy type: contents</h2>
  <h2>Name: C. Jones, Policy type: buildings</h2>
  <h2>Name: D. Umaga, Policy type: contents</h2>
  <h2>Name: E. Tolstoy, Policy type: buildings</h2>
 </body>
</html>
```

Claimants and policy types

Name: A. Liu, Policy type: contents

Name: B. Singh, Policy type: contents

Name: C. Jones, Policy type: buildings

Name: D. Umaga , Policy type: contents

Name: E. Tolstoy , Policy type: buildings

Selection With <xsl:if…>

- We can use <xsl:if..> to conditionally include elements or attributes in the output document
- The ‘test’ attribute contains a conditional XPath expression

```
<xsl:if test="XPath expression">
…
</xsl:if>
```

<xsl:if…> example

- In this example, we add a condition to our iteration that only selects policies with claims since 2003

```
<xsl:for-each select="policy-claims/policy">
 <xsl:if test="claims/claim[year > 2003]">
   Name: <xsl:value-of select="policy-holder"/>,
   <br/>
   Claim dates:
   <xsl:for-each select="claims/claim">
    <xsl:value-of select="year"/>,
   </xsl:for-each>
   <hr/>
 </xsl:if>
</xsl:for-each>
```

Example6-5.xsl

Example Output

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd ">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head> <title>Insurance Claims</title> </head>
 <body>
  <h1>Claimants since 2003</h1>
  <h2>
  Name: C. Jones,
  <br />
  Claim dates:2004,
  </h2>
  <hr />
  <h2>Name: D. Umaga,
  <br />
  Claim dates:1998,2005,
  </h2>
  <hr />
 </body>
</html>
```

Alternative Actions

- There is no alternative action that can be specified with <xsl:if…>
- To provide an alternative we must use *choose*, *when* and *otherwise*

```
<xsl:choose>
  <xsl:when test="XPath selection expression">
    Action for all selected nodes
  </xsl:when>
  <xsl:otherwise>
    Action for all other nodes
  </xsl:otherwise>
</xsl:choose>
```

<xsl:choose> example

- Here we display a message that indicates if a customer has recent claims or not

```
<xsl:for-each select="policy-claims/policy">
 <xsl:choose>
  <xsl:when test="claims/claim[year > 2003]">
   customer has recent claims<br/>
   Claim dates:
   <xsl:for-each select="claims/claim">
    <xsl:value-of select="year"/>,
   </xsl:for-each> <br/>
  </xsl:when>
  <xsl:otherwise>
   customer has no recent claims <br/>
  </xsl:otherwise>
 </xsl:choose>
 Name: <xsl:value-of select="policy-holder"/>,
 Policy type: <xsl:value-of select="@type"/> <hr/>
</xsl:for-each>
```

Example6-6.xsl

Example Output

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd ">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head> <title>Insurance Claims</title> </head>
 <body>
   <h1>Claimant History</h1>
   <h2> customer has no recent claims<br />
   Name: A. Liu, Policy type: contents</h2> <hr />
   <h2> customer has no recent claims<br />
   Name: B. Singh, Policy type: contents</h2> <hr />
   <h2> customer has recent claims<br />
   Claim dates:2004,<br />
   Name: C. Jones, Policy type: buildings</h2> <hr />
   <h2> customer has recent claims<br />
   Claim dates:1998,2005,<br />
   Name: D. Umaga, Policy type: contents</h2> <hr />
   <h2> customer has no recent claims<br />
   Name: E. Tolstoy, Policy type: buildings</h2> <hr />
 </body>
</html>
```

Claimant History

customer has no recent claims
Name: A. Liu, Policy type: contents

---

customer has no recent claims
Name: B. Singh, Policy type: contents

---

customer has recent claims
Claim dates: 2004,
Name: C. Jones, Policy type: buildings

---

customer has recent claims
Claim dates: 1998, 2005,
Name: D. Umaga , Policy type: contents

---

customer has no recent claims
Name: E. Tolstoy , Policy type: buildings

Sorting with <xsl:sort>

- <xsl:sort> has several attributes
- select
  - The XPath expression that identifies the sort key
- data-type
  - States whether the sort key is ‘text’ or a ‘number’
- order
  - Determines the sort order. Can be 'ascending' or 'descending',
- case-order
  - For text sorting. Determines which case is sorted first. Can be ‘upper-first’ or ‘lower-first’

Sorting with <xsl:sort>

- <xsl:sort> can appear either as a child of an <xsl:apply-templates> element (described later) or an <xsl:for-each> element
  - If it is a child of a <xsl:for-each>, it must be the first child
- If using more than one <xsl:sort> in a single node, the primary sort key is given by the first <xsl:sort> instruction, the secondary key by the second and so on.

Sorting Example

- Here, we sort the resulting nodes according to the alphabetical order of their insurance 'type' attribute

```
<xsl:for-each select="policy-claims/policy">
  <xsl:sort select="@type" data-type="text" order="ascending"/>
  <p>
  Name: <xsl:value-of select="policy-holder"/>,
  Policy type: <xsl:value-of select="@type"/>
  </p>
</xsl:for-each>
```

Example6-7.xsl

Sorted Output Document

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd ">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Insurance Claims</title>
  </head>
  <body>
    <h1>Claimants and policy types</h1>
    <p>Name: C. Jones, Policy type: buildings</p>
    <p>Name: E. Tolstoy, Policy type: buildings</p>
    <p>Name: A. Liu, Policy type: contents</p>
    <p>Name: B. Singh, Policy type: contents</p>
    <p>Name: D. Umaga, Policy type: contents</p>
  </body>
</html>
```

Claimants and policy types

Name: C. Jones, Policy type: buildings

Name: E. Tolstoy , Policy type: buildings

Name: A. Liu, Policy type: contents

Name: B. Singh, Policy type: contents

Name: D. Umaga , Policy type: contents

‘buildings’ policies appear before ‘contents’ policies

Writing Attributes to the Output Document

- Attributes can be written using 'xsl:attribute' elements
  - e.g. to add a 'class' attribute to a paragraph

```
<p><xsl:attribute name="class"> …
```

- This element can be combined with an 'xsl:value-of' element to supply the attribute value

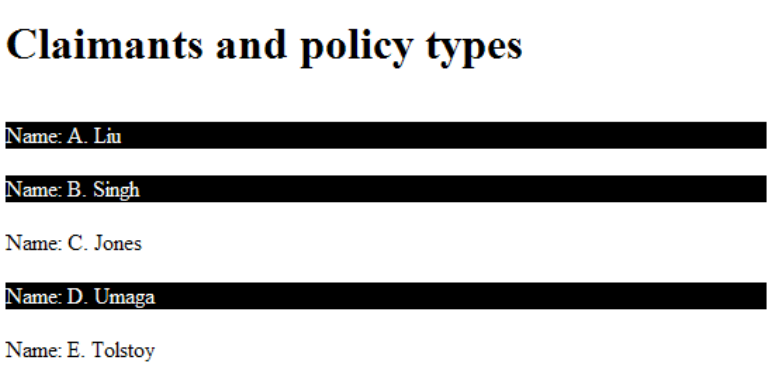
```
<p>
 <xsl:attribute name="class">
   <xsl:value-of select ="@type"/>
 </xsl:attribute>
</p>
```

Example6-8.xsl

Generated Markup

- The ‘class’ attribute is added to each paragraph, in each case given with the value of a ‘type’ attribute from the XML input document:

```
<p class="contents">Name: A. Liu</p>
<p class="contents">Name: B. Singh</p>
<p class="buildings">Name: C. Jones</p>
<p class="contents">Name: D. Umaga</p>
<p class="buildings">Name: E. Tolstoy</p>
```



- CSS could then be applied

```
.contents{color:white; background-color:black}
.buildings{color:black; background-color:white}
```

Other Attribute Examples

- Other non-presentational attributes that might come from an XML transform include anchors and image files

```
<xsl:when test="@type='contents' ">
  <img>
    <xsl:attribute name="src">
      <xsl:value-of select="/policy-claims/contents-image/" />
    </xsl:attribute>
   <xsl:attribute name="alt">
      contents
   </xsl:attribute>
  </img>
```

Example6-9.xsl

```
<a>
 <xsl:attribute name="href">
  <xsl:value-of select="policy-claims/company-domain"/>
 </xsl:attribute>Company home page
</a>
```

XML Special Characters

- In HTML we can use special *entity* characters such as:
  -   for a non breaking space
  - © for a copyright symbol (©)
- These are not recognised in XML so cannot be used in XSL Transformations
- We have to use their number codes instead
  -   for a non breaking space
  - © for a copyright symbol (©)

Using a Special Character

- In this anchor element we add the copyright symbol to the hyperlink text

```
<a>
  <xsl:attribute name="href">
   <xsl:value-of select="policy-claims/company-domain"/>
  </xsl:attribute>
  &#169;WebHomeCover
</a>
```

Some Special Character Codes

- ! Exclamation mark !
- " Quotation mark "
- # Number sign #
- $ Dollar sign $
- % Percent sign %
- & Ampersand &
- ' Apostrophe '
- (Left parenthesis (
-) Right parenthesis)
- * Asterisk *
- + Plus sign +
- , Comma ,
- - Hyphen -
- . Period (full stop) .
- : Colon :
- ; Semi-colon ;
- < Less than <
- = Equals sign =
- > Greater than >
- ? Question mark ?
- @ Commercial at @
- [Left square bracket [
-] Right square bracket]
-   Non-breaking Space
- ¢ Cent sign ¢
- £ Pound sterling £
- © Copyright ©
- ® Registered trademark ®

Character Encoding

- The problem is that the numbers may be interpreted differently depending on the character encoding being used
- Since XML defaults to utf-8, we can specify this in a meta element of a generated document

```
<head>
 <meta http-equiv="Content-Type" content="text/html" charset="utf-8" />
</head>
```

Transform Encoding

- You can set the encoding of the generated document to something else, but you must match it in the META element

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="ISO-8859-1"
doctype-public="'-//W3C//DTD XHTML 1.1//EN"
doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
<xsl:template match="/">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html" charset="ISO-8859-1" />
<title>…</title>
</head>
```

ISO-8859-1 is a 'legacy' encoding for HTML pages

UTF-8

- Best option is to always use UTF-8

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="utf-8"
doctype-public="'-//W3C//DTD XHTML 1.1//EN"
doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
<xsl:template match="/">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html" charset="utf-8" />
<title>…</title>
</head>
```

Transforming From XML to XML

- The examples we have seen so far have been from XML to (X)HTML
- We may also want to transform one XML document into another
- In this case we may want to keep whole XML elements from the source document
- To include parts of the source document as the original XML (rather than as the text values of elements or attributes), use the <xsl:copy-of…> element

<xsl:copy-of…>

- This XSLT stylesheet copies XML elements:

Example6-10.xsl

```
<?xml version="1.0"?>
<policy-holders xsl:version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:copy-of select="/policy-claims/policy/policy-holder"/>
</policy-holders>
```

- The XPath expression selects policy holders
- The full <policy-holder> element of each one (including tags) will be copied to the output document

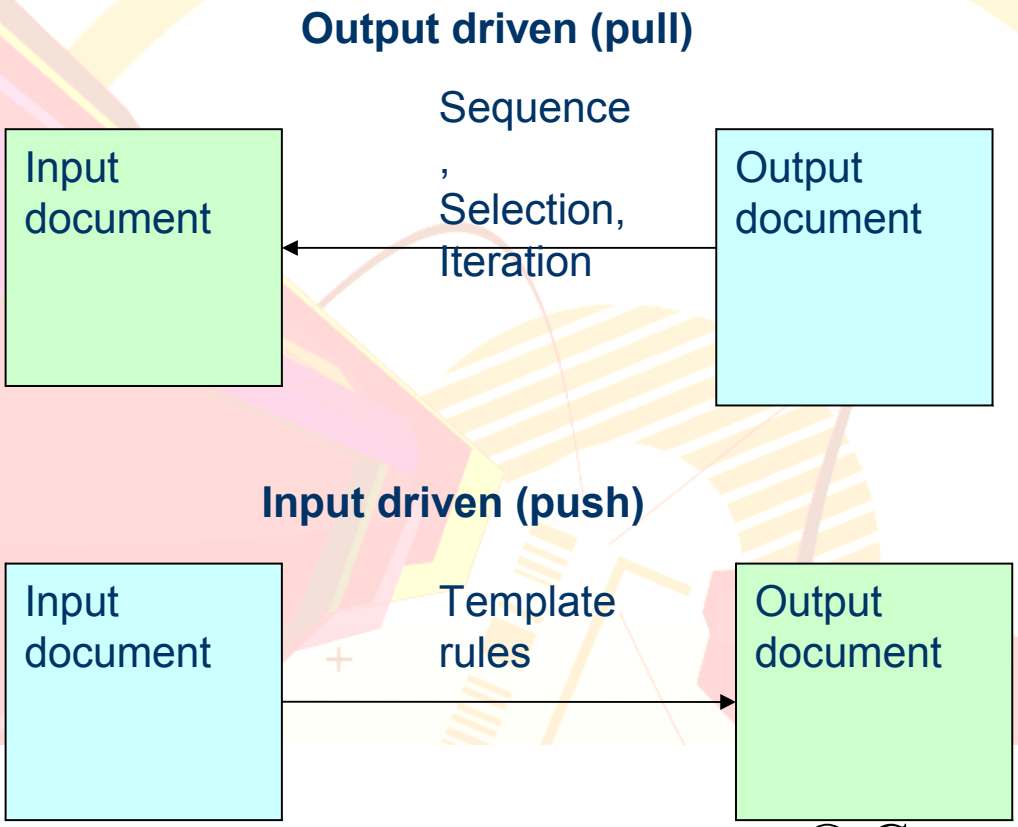
Output XML Document

```
<?xml version="1.0"?>
<policy-holders>
  <policy-holder>A. Liu</policy-holder>
  <policy-holder>B. Singh</policy-holder>
  <policy-holder>C. Jones</policy-holder>
  <policy-holder>D. Umaga</policy-holder>
  <policy-holder>E. Tolstoy</policy-holder>
</policy-holders>
```

Transforms Using Template Matching

- XLST can do two different types of transformation
  - Output driven (pull)
  - Input-driven (push)
- So far all our examples have been output-driven
  - Style sheets based on the structure of the output document
  - Using sequence, selection and iteration from the root
- An input driven approach applies ‘template rules’ to particular elements
  - More flexible for semi structured data

Pull and Push Transformations



Example

- We might provide an XSL transformation for a document by template matching several different nodes:
  - <xsl:template match="/">
  - <xsl:template match="heading">
  - <xsl:template match="subheading">
  - <xsl:template match="paragraph">
- Each node will have its own transform

document.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Example6-11.xsl"?>
<document>
  <heading>My first heading</heading>
  <subheading>My first subheading</subheading>
  <paragraph>Para 1</paragraph>
  <paragraph>Para 2</paragraph>
  <heading>My second heading</heading>
  <subheading>My second subheading</subheading>
  <paragraph>Para 3</paragraph>
  <subheading>My third subheading</subheading>
  <paragraph>Para 4</paragraph>
…etc..
</document>
```

Template Matching

- This is a ‘push’ transformation.
  - Instead of imposing the overall structure, we respond to template matches in the input document

```
<xsl:template match="document/paragraph">
  <p><xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="subheading">
  <h2><xsl:value-of select="."/></h2>
</xsl:template>
<xsl:template match="heading">
  <h1><xsl:value-of select="."/></h1>
</xsl:template>
```

Example6-11.xsl

Invoking Other Templates

- From one template, we can apply other templates to other nodes

```
<xsl:apply-templates/>
```

  - This will insert output from other template matches at that point in the output document
- By default, all children of the current node will have templates applied, but we can specify individual nodes using the 'select' attribute

```
<xsl:apply-templates select="policies/policy"/>
```

Example Template Matching (1)

- Root template

Example6-12.xsl

```
<xsl:template match="/">
…XHTML markup
    <xsl:apply-templates select="policy-claims/policy"/>
…XHTML markup
</xsl:template>
```

Ignore other child nodes

- Policy template

```
<xsl:template match="policy">
…XHTML markup
<xsl:apply-templates select="claims"/>
</xsl:template>
```

Ignore other child nodes (do not apply a template to 'policy-holder')

Example Template Matching (2)

- Claims template

```
<xsl:template match="claims">
…XHTML markup
  <xsl:apply-templates/>
…XHTML markup
</xsl:template>
```

Process all child nodes. The only child of ‘claims’ is ‘claim’

- Claim template

```
<xsl:template match="claim">
…XHTML markup
</xsl:template>
```

No ‘apply-templates’, so no processing for the child nodes ‘year’ and ‘details’

Chapter Summary

- XPath expressions
  - Picking out parts of an XML documents
- XSLT for transforming documents from one (type) to another
- Transforming XML into HTML and XHTML
- Transforming XML into XML
- Input-driven and output driven transformations