CSI 3140

WWW Structures, Techniques and Standards

Representing Web Data: XML

XML

- Example XML document:

```
<text>
  Hello World!
</text>
```

- An XML document is one that follows certain syntax rules (most of which we followed for XHTML)

XML Syntax

- An XML document consists of
  - Markup
    - Tags, which begin with < and end with >
    - References, which begin with & and end with ;
      - Character, e.g. `&#x20;`
      - Entity, e.g. `<`
        - The entities `lt`, `gt`, `amp`, `apos`, and `quot` are recognized in every XML document.
        - Other XHTML entities, such as `nbsp`, are only recognized in other XML documents if they are defined in the DTD
  - Character data: everything not markup

XML Syntax

- Comments
  - Begin with <!--
  - End -->
  - Must not contain –
- CDATA section
  - Special element the entire content of which is interpreted as character data, even if it appears to be markup
  - Begins with <![CDATA[
  - Ends with]]> (illegal except when ending CDATA)

XML Syntax

- The CDATA section

```
<![CDATA[
  <message>This & that</message>
]]>
```

is equivalent to the markup

```
&lt;message&gt;This &amp; that&lt;/message&gt;
```

XML Syntax

- < and & must be represented by references except
  - When beginning markup
  - Within comments
  - Within CDATA sections

XML Syntax

- Element tags and elements
  - Three types
    - Start, e.g. <message>
    - End, e.g. </message>
    - Empty element, e.g.
  - Start and end tags must properly nest
  - Corresponding pair of start and end element tags plus everything in between them defines an element
  - Character data may only appear within an element

XML Syntax

- Start and empty-element tags may contain attribute specifications separated by white space
  - Syntax: *name* = *quoted value*
  - *quoted value* must not contain <, can contain & only if used as start of reference
  - *quoted value* must begin and end with matching quote characters (‘ or “)

XML Syntax

- Element and attribute names are case sensitive
- XML white space characters are space, carriage return, line feed, and tab

XML Documents

- A well-formed XML document
  - follows the XML syntax rules and
  - has a single root element
- Well-formed documents have a tree structure
- Many XML parsers (software for reading/writing XML documents) use tree representation internally

XML Documents

- An XML document is written according to an XML vocabulary that defines
  - Recognized element and attribute names
  - Allowable element content
  - Semantics of elements and attributes
- XHTML is one widely-used XML vocabulary
- Another example: RSS (rich site summary)

XML Documents

```
<rss version="0.91">
  <channel>

    <title>www.example.com</title>
    <link>http://www.example.com/</link>
    <description>
      www.example.com is not a site that changes often...
    </description>
    <language>en-us</language>

    <item>
      <title>Announcing a Sibling Site!</title>
      <link>http://www.example.org/</link>
      <description>
        Were you aware that example.com is not the
        only site in the example family?
      </description>
    </item>

    <item>
      <title>We're Up!</title>
```

XML Documents

```
      <link>http://www.example.net/</link>
      <description>
        Our new RSS feed is up.  Visit us today!
      </description>
    </item>
  </channel>
</rss>
```

XML Documents

- Valid names and content for an XML vocabulary can be specified using
  - Natural language
  - XML DTDs (Chapter 2)
  - XML Schema (Chapter 9)
- If DTD is used, then XML document can include a document type declaration:

```
<!DOCTYPE rss
    SYSTEM "http://my.netscape.com/publish/formats/rss-0.91.dtd">
```

XML Documents

- Two types of XML parsers:
  - Validating
    - Requires document type declaration
    - Generates error if document does not
      - Conform with DTD and
      - Meet XML validity constraints
        - Example: every attribute value of type ID must be unique within the document
  - Non-validating
    - Checks for well-formedness
    - Can ignore external DTD

XML Documents

- Good practice to begin XML documents with an XML declaration
  - Minimal example: `<?xml version="1.0"?>`
  - If included, < must be very first character of the document
  - To override default UTF-8/UTF-16 character encoding, include encoding declaration following version:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

XML Documents

- Internal subset of DTD

```
<!DOCTYPE rss
    SYSTEM "http://my.netscape.com/publish/formats/rss-0.91.dtd"
 [
<!ENTITY vsn "0.91">
<!ENTITY unused "This entity is not used.">
 ]
>
<rss version="&vsn;">
```

Declaration of internal subset of DTD

  - Entity `vsn` will be defined by any XML parser, validating or not

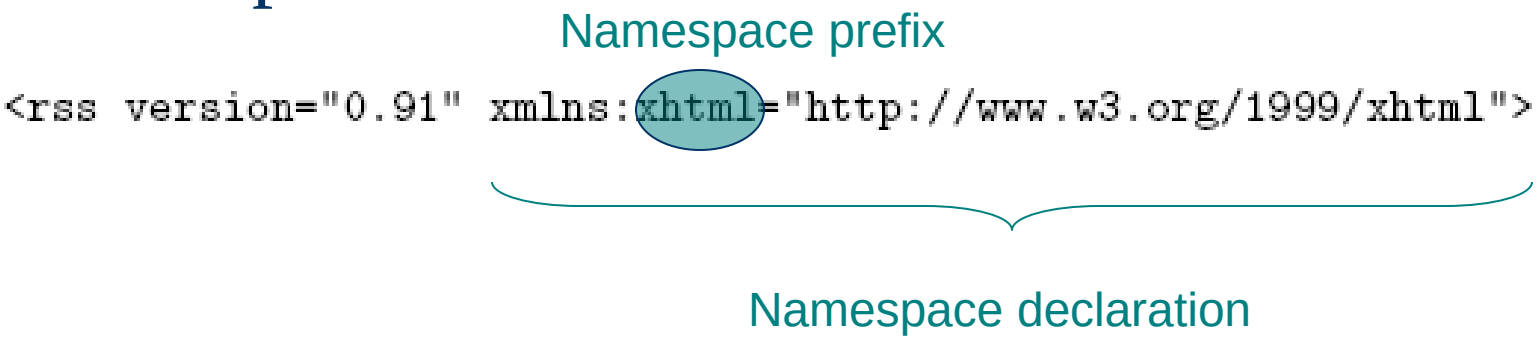
XML Namespaces

- XML Namespace: Collection of element and attribute names associated with an XML vocabulary
- Namespace Name: Absolute URI that is the name of the namespace
  - Ex: http://www.w3.org/1999/xhtml is the namespace name of XHTML 1.0
- Default namespace for elements of a document is specified using a form of the `xmlns` attribute:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

XML Namespaces

- Another form of xmlns attribute known as a namespace declaration can be used to associate a namespace prefix with a namespace name:



XML Namespaces

- Example use of namespace prefix:

```
<item>
  <title>Announcing a Sibling Site!</title>
  <link>http://www.example.org/</link>
  <description>Were you aware that
     <xhtml:a href="example.com">example.com</xhtml:a>
     is not the only site in the example family?</description>
</item>
```

XML Namespaces

- In a namespace-aware XML application, all element and attribute names are considered qualified names
  - A qualified name has an associated expanded name that consists of a namespace name and a local name
  - Ex: `item` is a qualified name with expanded name `<null, item>`
  - Ex: `xhtml:a` is a qualified name with expanded name `<http://www.w3.org/1999/xhtml, a>`

XML Namespaces

- Other namespace usage:

A namespace can be declared and used on the same element

```
<ns1:elt1 xmlns:ns1="http://www.example.com/ns">
  <ns1:elt2 xmlns:ns1="http://www.example.org/namespace">
    <ns1:elt3 />
  </ns1:elt2>
</ns1:elt1>
```

XML Namespaces

- Other namespace usage:

```
<ns1:elt1 xmlns:ns1="http://www.example.com/ns">
  <ns1:elt2 xmlns:ns1="http://www.example.org/namespace">
    <ns1:elt3 />
  </ns1:elt2>
</ns1:elt1>
```

A namespace prefix can be redefined for an element and its content

These elements belong to `http://www.example.org/namespace`

JavaScript and XML

- JavaScript DOM can be used to process XML documents
- JavaScript XML Dom processing is often used with `XMLHttpRequest`
  - Host object that is a constructor for other host objects
  - Sends an HTTP request to server, receives back an XML document

JavaScript and XML

- Example use:
  - Previous visit count servlet: must reload document to see updated count
  - Visit count with `XMLHttpRequest`: browser will automatically update the visit count periodically without reloading the entire page

JavaScript and XML

Document generated by GET request to `VisitCountUpdate` servlet

```
<html xmlns='http://www.w3.org/1999/xhtml'>
  <head>
    <title>
      VisitCountUpdate.java
    </title>
    <script type='text/javascript' src='/VisitCountUpdate.js'>
    </script>
    <meta http-equiv='Content-Script-Type' content='text/javascript' />
  </head>
  <body onload='init();'>
    <p>
       Hello World!
    </p>
    <p>
       This page has been viewed
       <span id='visits'>12</span>
       times since the most recent server restart.
    </p>
  </body>
</html>
```

JavaScript file using XMLHttpRequest object

span that will be updated by JavaScript code

JavaScript and XML

```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response)
  throws ServletException, IOException
    {
      // Set the HTTP content type in response header
      response.setContentType("application/xml; charset=\"UTF-8\"");

      // Obtain a PrintWriter object for creating the body
      // of the response
      PrintWriter servletOut = response.getWriter();

      // Output the count
      servletOut.println(
"<?xml version='1.0' encoding='UTF-8'?> \n" +
"<count>" + visits + "</count>");
      servletOut.close();
    }
```

XMLHttpRequest request is processed by doPost() method of servlet

Response is XML document

Current visit count is returned as content of count XML element

JavaScript and XML

```
// VisitCountUpdate.js

// Start a timer that every three seconds retrieves from the
// server the current number of visitors to our site.
function init() {
  window.setInterval("getVisits()", 3000);
  return;
}
```

JavaScript and XML

```
// Use XMLHttpRequest to request the current number of
// visitors to our site.
function getVisits() {
  var connection;  // Object used to send HTTP requests to server
                   // and receive HTTP responses from server

  // Create the connection object using the appropriate constructor.
  if (window.XMLHttpRequest) {
    connection = new XMLHttpRequest();
  }
  else if (window.ActiveXObject) {
    try {
      connection = new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (e) {
    }
  }
```

Typical code for creating an instance of XMLHttpRequest

JavaScript and XML

```
if (connection) {

  // Associate this XMLHttpRequest object with a specific URL.
  connection.open("POST",
                  "/servlet/VisitCountUpdate",
                  true);
  // Send an HTTP request to the server after specifying the
  // function that should be called when the response is received.
  connection.onreadystatechange =
    function () { updateVisits(connection); };
  connection.setRequestHeader("Content-Type",
                              "application/x-www-form-urlencoded");
  connection.send("");
}
return;
}
```

Return immediately after sending request (asynchronous behavior)

Function called as state of connection changes

Body of request (empty in this example)

JavaScript and XML

```
// Update the associated HTML document when the HTTP response
// containing the visit count is received.
function updateVisits(connection) {
  if (connection.readyState == 4 && connection.status == 200) {
    var visits = document.getElementById("visits");
    var count = connection.responseXML.documentElement;
    visits.childNodes[0].data = count.childNodes[0].data;
  }
  return;
}
```

Indicates response received successfully

Root of returned XML document

JavaScript and XML

- Ajax: Asynchronous JavaScript and XML
- Combination of
  - (X)HTML
  - XML
  - CSS
  - JavaScript
  - JavaScript DOM (HTML and XML)
  - `XMLHttpRequest` in asynchronous mode

Java-based DOM

- Java DOM API defined by `org.w3c.dom` package
- Semantically similar to JavaScript DOM API, but many small syntactic differences
  - Nodes of DOM tree belong to classes such as `Node`, `Document`, `Element`, `Text`
  - Non-method properties accessed via methods
    - Ex: `parentNode` accessed by calling `getParentNode()`

Java-based DOM

- Methods such as `getElementsByTagName()` return instance of `NodeList`
  - `getLength()` method returns # of items
  - `item()` method returns an item

```
document.getElementsByTagName("link").item(0)
```

Java-based DOM

- Example: program to count `link` elements in an RSS document:

```
DocumentBuilderFactory docBuilderFactory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();

Document document = parser.parse(new File(args[0]));
NodeList links = document.getElementsByTagName("link");
System.out.println("Input document has " +
                   links.getLength() +
                   " 'link' elements.");
```

Java-based DOM

- Imports:

From Java API for XML Processing (JAXP)

```
// JAXP classes
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

// DOM classes
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

// JDK classes
import java.io.File;
```

Java-based DOM

- Default parser is non-validating and non-namespace-aware.
- Overriding:

```
DocumentBuilderFactory docBuilderFactory =
    DocumentBuilderFactory.newInstance();
docBuilderFactory.setNamespaceAware(true);
DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();
```

- Also `setValidating(true)`

Java-based DOM

- Namespace-aware versions of methods end in NS:

```
NodeList links =
  document.getElementsByTagNameNS(null, "link");
```

Namespace name

Local name

SAX

```
// JAXP classes
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;

// SAX classes
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
```

SAX

```
/** Count the number of link elements in an XML document */
class SAXCountLinks {

  /** Source for RSS feed */
  static String FEED_URL = "http://today.java.net/rss/21.rss";

  /** Initialize XMLReader and set up event handlers */
  static public void main(String args[]) {
    try {
      // JAXP-style initialization of SAX parser
      SAXParserFactory saxFactory = SAXParserFactory.newInstance();
      XMLReader parser = saxFactory.newSAXParser().getXMLReader();

      // SAX-style processing of RSS document at FEED_URL
      parser.setContentHandler(new CountElementsHelper());
      parser.parse(FEED_URL);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    return;
  }
```

SAX

```
/** Helper class containing SAX event handler methods */
private static class CountElementsHelper extends DefaultHandler {

  /** Number of 'p' elements seen so far */
  int numElements;

  /** Constructor (allows for superclass initialization) */
  CountElementsHelper() {
    super();
  }
```

SAX

```
/** Perform initialization for this instance */
public void startDocument() throws SAXException {
  numElements = 0;
  return;
}

/** Process the start of an element */
public void startElement(String namespaceURI, String localName,
                         String qName, Attributes atts)
    throws SAXException
{
   if (qName.equals("link")) {
     numElements++;
   }
   return;
}
```

Used if not namespace-aware or if qualified name does not belong to any namespace.

SAX

```
/** Done with document; output final count */
public void endDocument() throws SAXException {
  System.out.println("Input document has " +
                     numElements +
                     " 'link' elements.");
  return;
}
```

Transformations

- JAXP provides API for transforming between DOM, SAX, and Stream (text) representations of XML documents
- Example:
  - Input from stream to DOM
  - Modify DOM
  - Output as stream

Transformations

```
// JAXP classes
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
```

Transformations

```
// Input an RSS document into a DOM Document object
DocumentBuilderFactory docBuilderFactory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder parser = docBuilderFactory.newDocumentBuilder();
Document document = parser.parse(new File(args[0]));

// Use the DOM API to remove the first item element
// (this code assumes that there is at least one item...)
NodeList items = document.getElementsByTagName("item");
items.item(0).getParentNode().removeChild(items.item(0));

// Use JAXP methods to output the modified Document object
TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer();
transformer.transform(new DOMSource(document),
                      new StreamResult(System.out));
```

Transformations

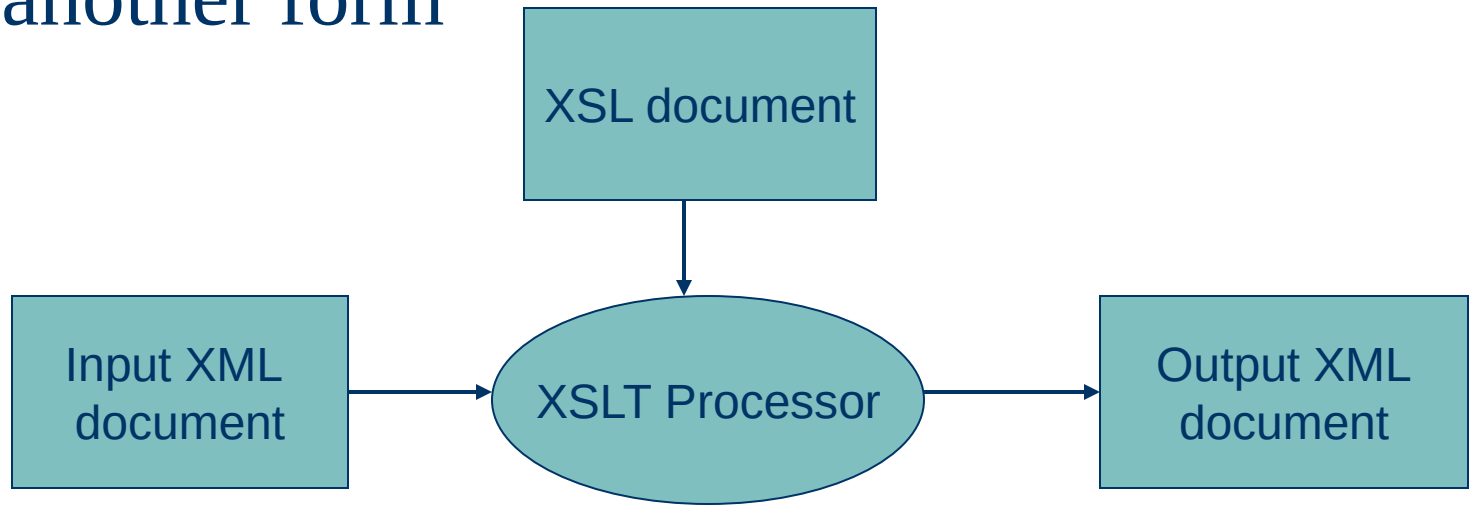
- "SAX output" means that a SAX event handler is called:
  - Example: the code

```
transformer.transform(new DOMSource(document),
                      new SAXResult(new CountElementsHelper()));
```

feeds the XML document represented by DOM `document` through the SAX event handler `CountElementsHelper()`

XSL

- The Extensible Stylesheet Language (XSL) is an XML vocabulary typically used to transform XML documents from one form to another form



XSL

Example XSL document
`HelloWorld.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>

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

  <xsl:template match="/">
    <html>
      <head>
        <title>
          HelloWorld.xsl (transformed)
        </title>
      </head>
      <body>
        <p><xsl:value-of select="child::message" /></p>
      </body>
    </html>
  </xsl:template>
</xsl:transform>
```

XSL markup

Everything in the body of the document that is not XSL markup is *template data*

XSL

- Input XML document `HelloWorld.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>Hello World!</message>
```

- Output XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"><head><title>
          HelloWorld.xsl (transformed)
        </title></head><body><p>Hello World!</p></body></html>
```

XSL

```
class XSLTransform {
    public static void main(String args[]) {

        try {
            TransformerFactory tFactory = TransformerFactory.newInstance();
            Transformer transformer =
                tFactory.newTransformer(
                    new StreamSource(new File(args[0])));
            transformer.transform(
              new StreamSource(new File(args[1])),
              new StreamResult(System.out));
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return;
    }
}
```

```
java XSLTransform HelloWorld.xsl HelloWorld.xml
```

XSL

- Components of XSL:
  - XSL Transformations (XSLT): defines XSL namespace elements and attributes
  - XML Path Language (XPath): used in many XSL attribute values (ex: `child::message`)
  - XSL Formatting Objects (XSL-FO): XML vocabulary for defining document style (print-oriented)

XPath

- XPath operates on a tree representation of an XML document
  - Similar to DOM tree in that nodes have different types (element, text, comment, etc.)
  - Unlike DOM, attributes are also nodes in the XPath tree
  - Root of XPath tree called document root
  - One of the children of the document root, called the document element, is the root element of the XML document

XPath

- Location path: expression representing one or more XPath tree nodes
  - `/` represents document root
  - `child::message` is an example of a location step and has two parts:

Axis name *Node test*

XPath

XSLT specifies *context node*

TABLE 7.2: Some XPath 1.0 axis names.

Name	Relationship with context node
`self`	The context node itself
`child`	Any immediate descendant
`descendant`	Any proper descendant
`descendant-or-self`	Any descendant, including the context node itself
`parent`	Immediate ancestor
`ancestor`	Any proper ancestor, including the document root (unless the context node is the document root)
`ancestor-or-self`	Any ancestor, including the context node itself
`preceding-sibling`	Any sibling of the context node that precedes the context node in the document
`following-sibling`	Any sibling of the context node that follows the context node in the document
`attribute`	Any attribute defined for the context node

Attribute nodes are
only seen along the attribute axis

XPath

- Node test:
  - Name test: qualified name representing an element (or attribute, for attribute axis) type
    - Example: `child::message` uses a name test
    - May use * as wildcard name test
  - Node-type test:
    - `text()`: true if node is a text node
    - `comment()`: true if node is a comment node
    - `node()`: true of any node

XPath

- A location step can have one or more predicates that act as filters:

```
child::chapter[attribute::display="visible"][position()=last()]
```

This predicate applied first

XPath

TABLE 7.3: Some XPath predicates.

Predicate type	Example	Example predicate is true if...
Related node exists	`child::title`	Node has child element of type `title`
Related node exists with certain text content	`child::title="Overview"`	Node has child element of type `title` which has string value (concatenation of all text content of descendants) of `Overview`
Attribute exists	`attribute::display`	Node has attribute named `display`
Attribute exists with certain value	`attribute::display='visible'`	Node has attribute named `display` that has value `visible`
Node is at a certain position in the list of nodes being filtered	`position()<=3`	Node is one of the first three in the node list (see text concerning node ordering).

XPath

- Abbreviations:
  - Axis defaults to `child` if not specified
    - `child::para` = `para`
  - `@` can be used in place of `attribute::`
    - `attribute::display` = `@display`
  - `parent::node()` = `..`
  - `self::node()` = `.`

XPath

- A location path is one or more location steps separated by `/`
  - Ex: `child::para/child::strong` (or just `para/strong`)
  - Ex: `para/strong/emph`

XPath

- Evaluating a two-step location path:
  - Evaluate first location step, producing node list L_1
  - For each node n_i in L_1
    - Temporarily set context node to n_i
    - Evaluate second location step, producing node list L_i
  - Result is union of L_is
- Continue process for paths with more steps

XPath

```
<body>
  <para id="p1">
    This is <strong id="s1">important</strong> to know.
    And I do mean <strong id="s2">important</strong>.
  </para>
  <para id="p2">
    This is not as important.
    <em><strong id="s3">Is this?</strong></em>
  </para>
  <strong id="s3">What about this?</strong>
  <para id="p3">
    <strong id="s4">Is anyone listening?</strong>
  </para>
</body>
```

- If `body` is context node, then:
  - `para/strong` represents {`s1`,`s2`,`s4`}
  - `para[strong]` represents {`p1`, `p3`}

XPath

- An absolute location path begins with `/` and uses the document root as the context node
  - Ex: `/body/para` represents all `para` nodes that are children of `body` which is child of document root
  - Ex: `/` represents list consisting only of the document root
- A relative location path does *not* begin with `/` and uses an element determined by the application (e.g., XSLT) as the context node
  - Ex: `body/para`

XPath

- Another abbreviation:
  - `/descendant-or-self::node()/` = `//`
  - Examples:
    - `//strong`: All `strong` elements in document
    - `.//strong`: All `strong` elements that are descendants (or self) relative to the context node

XPath

- Combining node lists:
  - Use | to represent union of node lists produced by individual location paths
  - Ex: `strong|descendant::emph` represents all nodes that are either
    - children of the context node of type `strong`; or
    - descendants of the context node of type `emph`

XSLT

```
<?xml version="1.0" encoding="UTF-8"?>

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

  <xsl:template match="/">
    <html>
      <head>
        <title>
          HelloWorld.xsl (transformed)
        </title>
      </head>
      <body>
        <p><xsl:value-of select="child::message" /></p>
      </body>
    </html>
  </xsl:template>
</xsl:transform>
```

Pattern of template rule

Template rule

Template of template rule

XSLT

- XSLT processor deals with three XPath trees:
  - Input trees: source and style-sheet
    - Elements containing only white space are normally not included in either input tree (exception: `xsl:text` element)
    - White space retained within other elements
  - Output tree: result

XSLT

- XSLT processing (high level):
  - Construct input trees
  - Initialize empty result tree
  - Search source tree for a node that is matched by a template rule, i.e., a node that is contained in the node list represented by the pattern of some template rule
  - Instantiate the template of the matching template rule in the result tree
    - Context node for XPath expressions is matched node

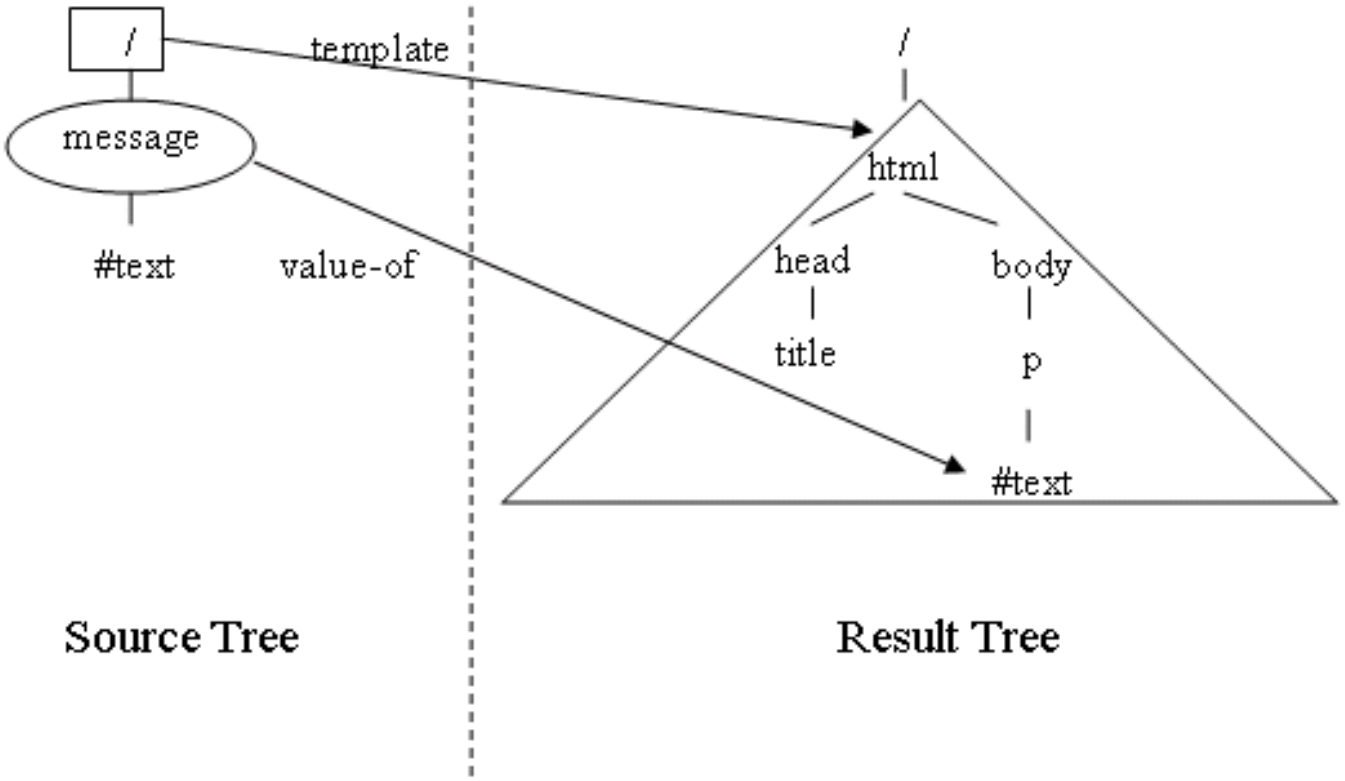
XSLT

```
<?xml version="1.0" encoding="UTF-8"?>

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

  <xsl:template match="/">
    <html>
      <head>
        <title>
          HelloWorld.xsl (transformed)
        </title>
      </head>
      <body>
        <p><xsl:value-of select="child::message" /></p>
      </body>
    </html>
  </xsl:template>
</xsl:transform>
```

Matches source tree document root

Context node for relative location path is source tree document root

XSLT


/
template
message
#text
value-of
/
html
head
body
title
p
#text
Source Tree
Result Tree

XSLT

- Restrictions on XPath in template rule pattern (value of `match` attribute):
  - Only `child` and `attribute` axes are allowed directly (can indirectly use `descendant-or-self` axis via `//` notation)
  - XPath expression must evaluate to a node list (some XPath expressions are functions that produce string values)

XSLT

```
<rss version="0.91">
  <channel>

    <title>www.example.com</title>
    <link>http://www.example.com/</link>
    <description>
      www.example.com is not a site that changes often...
    </description>
    <language>en-us</language>

    <item>
      <title>Announcing a Sibling Site!</title>
      <link>http://www.example.org/</link>
      <description>
        Were you aware that example.com is not the
        only site in the example family?
      </description>
    </item>

    <item>
      <title>We're Up!</title>
      ...
```

XSLT

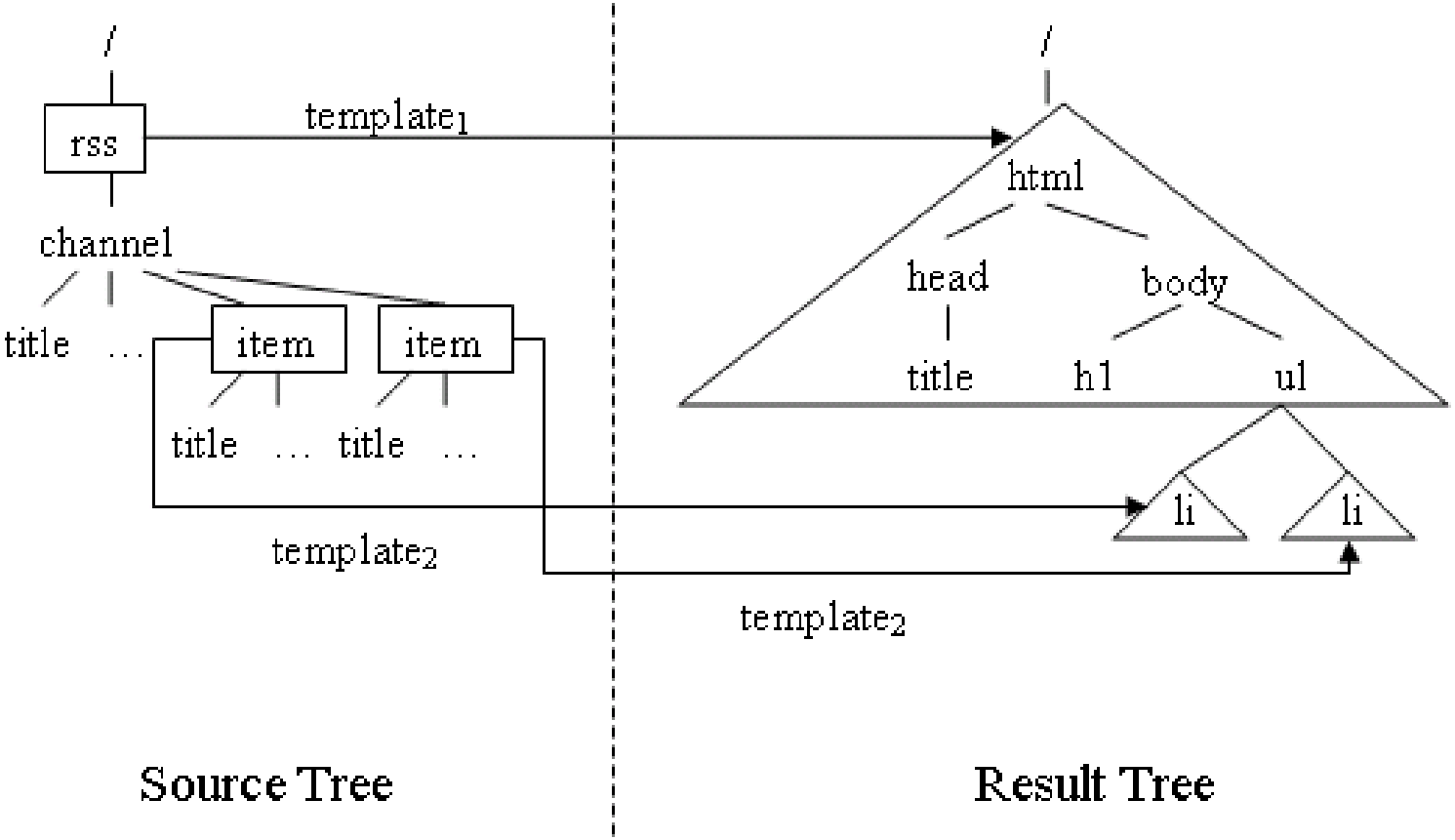
```
<xsl:template match="/rss">
  <html>
    <head>
      <title>
        RSStoXHTML
      </title>
    </head>
    <body>
      <h1>
        <xsl:value-of select="channel/title" />
        RSS feed links:
      </h1>
      <ul>
        <xsl:apply-templates select="channel/item" />
      </ul>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="item">
  <li>
    <xsl:value-of select="title" />
  </li>
</xsl:template>
```

XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"><head><title>
          RSStoXHTML
        </title></head><body><h1>www.example.com
          RSS feed links:
        </h1><ul><li>Announcing a Sibling Site!</li>
        <li>We're Up!</li></ul></body></html>
```

XSLT


/
rss
template1
channel
title
...
item
item
title
...
title
...
template2
template2
/
html
head
body
title
h1
ul
li
li
Source Tree
Result Tree

XSLT

- Adding attributes


```
<xsl:template match="item">
  <li>
    <a>
      <xsl:attribute name="href">
        <xsl:value-of select="link" />
      </xsl:attribute>
      <xsl:attribute name="title">
        <xsl:value-of select="description" />
      </xsl:attribute>
      <xsl:value-of select="title" />
    </a>
  </li>
</xsl:template>
```

⟺ `<a href="{link}"`

XSLT

Source document elements are in a namespace

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      H1ExtractTest.html
    </title>
  </head>
  <body>
    <h1>First heading</h1>
    <h2>Lower heading</h2>
    <h1>Second
      <span style="font-size:larger">major</span>
       heading
    </h1>
    <h2>Another lower heading</h2>
    <h3>And even lower</h3>
    <h1>Last heading</h1>
  </body>
</html>
```

We want to copy all h1 elements *plus* all of their descendants

XSLT

```
<xsl:template match="/xhtml:html">
  <html>
    <head>
      <title>
        H1Extract
      </title>
    </head>
    <body>
      <xsl:apply-templates select=".//xhtml:h1" />
    </body>
  </html>
</xsl:template>

<xsl:template match="xhtml:h1">
  <xsl:copy-of select="." />
</xsl:template>
```

XPath expressions must use qualified names (because source includes namespace)

Copy element and content

XSLT

- Template markup

```
<ul>
  <li> </li>
```

in the result tree becomes `<ul><li/>`

  - Most browsers will not accept this notation!
- XSLT does not recognize ` `
- Solution: `<li><xsl:text> </xsl:text></li>`

XSLT

- Adding XML special characters to the result
  - Template:

```
<li>
  <xsl:text
    disable-output-escaping="yes">&amp;nbsp;</xsl:text>
</li>
```

Becomes & on input

  - Result:

```
<li> </li>
```

- `disable-output-escaping` also applies to `value-of`

XSLT

- Output formatting:
  - Add `xml:space="preserve"` to `transform` element of template to retain white space
  - Use xsl:output element:

```
<xsl:output
  method="xml"
  version="1.0"
  encoding="UTF-8"
  doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
  omit-xml-declaration="yes"
/>
```

XML and Browsers

- An XML document can contain a processing instruction telling a browser to:
  - Apply XSLT to create an XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- HelloWorldStyled.xml -->
<?xml-stylesheet type="text/xsl" href="HelloWorld.xsl"?>
<message>Hello World!</message>
```

XML and Browsers

- An XML document can contain a processing instruction telling a browser to:
  - Apply CSS to style the XML document:

```
/* HelloWorld.css */
message { display: block;
          margin: 8px;
          font-weight: bolder }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- HelloWorldStyledCSS.xml -->
<?xml-stylesheet type="text/css" href="HelloWorld.css"?>
<message>Hello World!</message>
```