Murdoch
UNIVERSITY

XML Parsing and Processing

+ NodeJS XML Parsing and Processing

XML Advance

Lecture 7 (A)

Learning Objectives

- Learn some Node.js classes and methods available for parsing and processing XML documents
- We will also look at some Perl classes and methods available for parsing and processing XML documents
- NOTE: you will **NOT** be examined on any Perl material
- Look at some example code using these modules

Learning Objectives

- In the context of this unit:
  - XML is an important set of Internet technologies for use in different solutions in different areas
  - A major part of developing an XML solution is to design and implement software that can process the information in XML documents **automatically** (i.e. **XML applications/programs**)
  - Thus, we must learn to parse and process XML documents

Learning Objectives

- In the context of this unit:
  - Node.js and Perl have external modules to assist with such tasks, so we introduce them as examples of tools available in programming languages to handle parsing and processing XML documents

Learning Outline

- What is a parser?
- Why do we need to parse?
- The XML::Parser module in Perl
- The XML::DOM module in Perl
- The DOMParser module in Node.js
- The XMLSerializer module in Node.js
- DOM methods available for parsing and processing XML documents

What is a Parser?

- Parsing is more formally known as *syntactic analysis*
- In computer science and linguistics, parsing is the process of analyzing a sequence of tokens to determine their grammatical structure with respect to a given formal grammar

What is a Parser?

- The word parser in compiler parlance, implies a module that reads and interprets the source code
- In a compiler, the parser creates a parse tree, which is an in-memory representation of the source code
- The second half of the compiler – known as the backend – uses parse trees to generate object files (compiled modules)

Parsers in XML

- A parser is one of the most important XML tools
- **Every XML application** includes a parser
- The parser is positioned between the XML application and XML documents
- Its goal is to shield the developer from the intricacies of the XML syntax
- It is a low-level tool that is almost invisible to everybody but programmers

Why Do You Need Parsers?

- To this point, we have used XML within XML environments??
- However, we also need to be able to design and implement software that can process the information in XML documents (i.e. write **XML applications**)
  - That is, so that Web software operates over the Internet **automatically**

Why Do You Need Parsers?

- Imagine you are given an XML file with product information, including product code, description, prices, suppliers, etc.
- You are asked to write an application to convert the prices from dollars to Euros
- How to do it?
  - http://www.informit.com/articles/article.aspx?p=29389

First Approach

- At first, it looks like a simple assignment
- Algorithm:

```
Loop through the price list
Multiply each price by the exchange rate
```

What About XML Syntax?

- Yet on closer consideration, you need to remember the prices are in an XML document
- To loop through the prices means to read the entire document and interpret the XML syntax
- That doesn't seem too difficult because elements in the document are tagged
  - They are designated by an element name inside angled brackets

Do You Remember Entities?

- However, the XML syntax is not just about angled brackets
- There might be entities in the price list
  - Therefore, the application must read and interpret the Document Type Definition or Schema to be able to resolve entities
- While it reads the DTD/Schema, it might as well read element definitions and validate the document
  - This is required to ensure the correct information is being accessed

What About Other XML Features?

- What about Character encodings, namespaces?
- And do not forget to consider errors!!
  - How would the software recover from a missing closing tag?

How “simple” is XML Syntax?

- Yet XML has an eXtensible syntax so XML applications have to be ready to cope with many of these issues – and possibly more
- As it turns out, writing a software library to decode XML files may be a one month long assignment
- If you were to write such a library, after one month you would have written your own parser and more

How “simple” is XML Syntax?

- So the question has to be asked:
  - Is it productive to spend one month writing a parser library when you need only a quarter of a day's work to actually process the data?
- Of course not!
- It is more sensible to download a parser from the Internet or use one that ships with your favourite development tool

XML Parsing

- As just mentioned, before your program is able to do anything with the data in an XML document, it must first
  - Parse the document and extract the relevant data
  - Store the relevant data in an appropriate data structure for a program to utilize

XML::Parser

- In Perl, XML::Parser is a basic module that can be used to parse an XML document
- It is based on James Clark's Expat library
  - Expat is a very important C library used extensively to implement XML parsers
  - Most of the low level details of Expat are hidden in a parser such as XML::Parser
  - http://www.jclark.com/xml/expat.html

XML::Parser

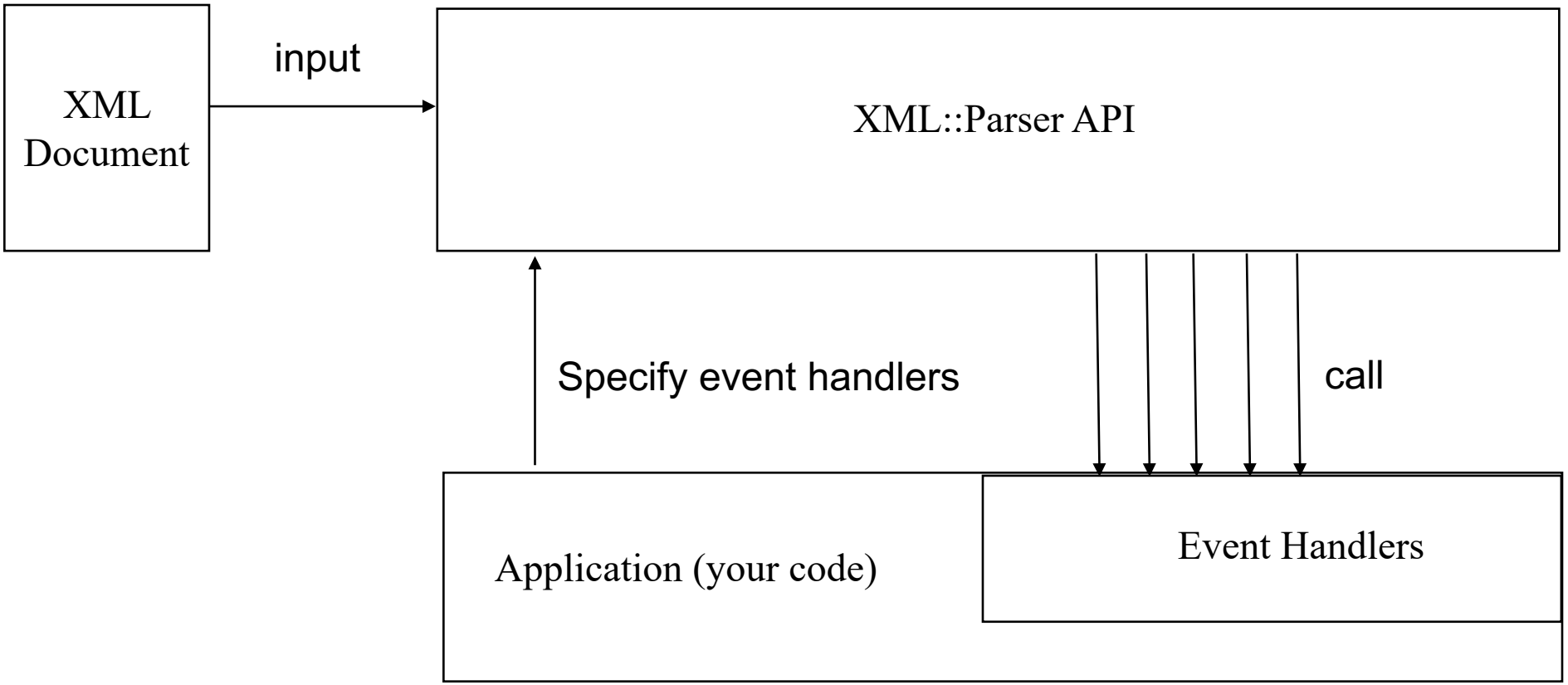
- Expat and XML::Parser are non-validating parsers
  - That is, the parser does not check a document against any Document Type Definition or Schema
  - It only checks that the document is **well-formed** (i.e. that it is properly marked up according to XML syntax rules)

Event Handlers in `XML::Parser`

- Perl's `XML::Parser` is **event-driven**
  - It works by allowing you to specify event handlers for different situations the Parser might encounter during parsing
  - You do not have to deal with the low-level character-by-character parsing, you only specify what happens when the parser encounters certain "components" of an XML document (eg: a start tag)

http://search.cpan.org/~msergeant/XML-Parser-2.36/Parser.pm

XML::Parser Processing



Example XML Document


```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="poetry.xsl"?>
<poetry>
   <anthology>
     <poem>
          <title>The SICK ROSE</title>
          <author>William Blake</author>
          <stanza>
               <line>O Rose thou art sick.</line>
               <line>The invisible worm,</line>
               <line>That flies in the night</line>
               <line>In the howling storm,</line>
          </stanza>
          <stanza>
              <line></line>
              <line></line>
              <line />
              <line />
          </stanza>
     </poem>
   </anthology>
</poetry>
```

Murdoch UNIVERSITY

An Example Using `XML::Parser`

```
use XML::Parser;

my @tags;
my $parser = new XML::Parser;
$parser->setHandlers (Start => \&my_start_handler);

die "Unable to parse XML document\n"
    unless $parser->parsefile("poetry.xml");

# Do something with @tags

sub my_start_handler
{
    my ($expat, $item, %attr) = @_;
    print "Encountered node type: $item \n";
    push (@tags, $item);
}
```

Every time a start tag is encountered call subroutine `&my_start_handler`

`&my_start_handler` will take node-type value from subroutine parameter list `@_`, print a message, and put the node-type into the `@tags` array

Node.js XML Parser

- In Node.js, the `node-xml` module provides an event-driven interface `xml.SaxParser()` to parse XML documents
- There are numerous methods to assist parsing
- More information on the parser, methods, and an example parser are available at:
  - https://www.npmjs.com/package/node-xml

Non-Validating Parsers

- Like Perl's `XML::Parser`, `node-xml` is a non-validating parser
  - That is, it does not check a document against any DTD or Schema
  - It only checks that the document is **well-formed** (i.e. that it is properly marked up according to XML syntax rules)

No Data Structures

- `node-xml` and Perl's `XML::Parser` **do not** construct a useful object or data structure to represent the XML document
  - It is left to the programmer using the parser to abstract any structure and access the required data
  - We mentioned earlier that `node-xml` and Perl's `XML::Parser` are event-driven parsers
  - So when using these parsers, the programmer thinks more in terms of the events that occur rather than the objects to be manipulated

DOM vs SAX

- Parsers that are event driven offer a Simple API for XML (**SAX**) approach to processing XML documents
- The Document Object Model (**DOM**) is an alternative approach to processing XML documents, where an object tree representation is constructed based on the XML definition

XML::DOM

- Perl's `XML::DOM` is an alternative module to `XML::Parser`, that **does** construct a useful object / data structure for processing and manipulation purposes
  - http://search.cpan.org/~tjmather/XML-DOM-1.45/
- Node.js has DOM parser modules
  - `xmldom`
  - `libxmljs-dom`

Node.js XML DOM Parser

- In Node.js, the `xmldom` module provides a DOM interface `xml.DOMParser()` to parse and process XML documents
- There are numerous methods to assist parsing and processing
- More information on such methods is available at:
  - https://www.npmjs.com/package/xmldom

In This Unit...

- In the tutorials for this unit, we will be using the Perl `XML::DOM` and the Node.js `xmldom` modules instead of Perl's `XML::Parser` and `node-xml`
  - `node-xml` may prove helpful for the 2nd assignment
  - However, be aware that the Node.js XML libraries are not as mature as Perl XML libraries, and there is not as much documentation for the use of Node.js XML libraries
  - Perl's XML libraries are simple and well supported

The Document Object Model (DOM)

- The Document Object Model (DOM) is a cross-platform and language-independent Application Programming Interface (API) for representing and interacting with objects in HTML, XHTML and XML
- The DOM describes the content, structure and style of documents, by putting them into specified **objects**

The Document Object Model (DOM)

- The objects of every document are organized in a tree-like structure called the DOM tree
- DOM is currently a W3C standard
- So regardless of the language being used for development of an application, objects in the DOM tree may be accessed and manipulated by using appropriate methods available which comply with the W3C specifications

The Document Object Model (DOM)

- The DOM is an interface that allows an application to discover information about an XML document by navigating around it
  - Many XML parsers implement the DOM interface

The Document Object Model (DOM)

- In the DOM specification, the term "document" is used in the broad sense
  - XML is used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as **data** rather than as **documents**
- XML presents this data as documents, and the DOM may be used to manage this data
- With the DOM, programmers can build documents, navigate their structure, and add, modify, or delete elements and content

The DOM Structure Model

- The DOM presents documents as a hierarchy of **node** objects, some with **child** nodes of various types, and others as **leaf** nodes that cannot have other nodes below them
  - This fundamental idea is the basis behind the structure of XML documents, and so it is very well suited to XML applications
  - XPath is also built on the same concept

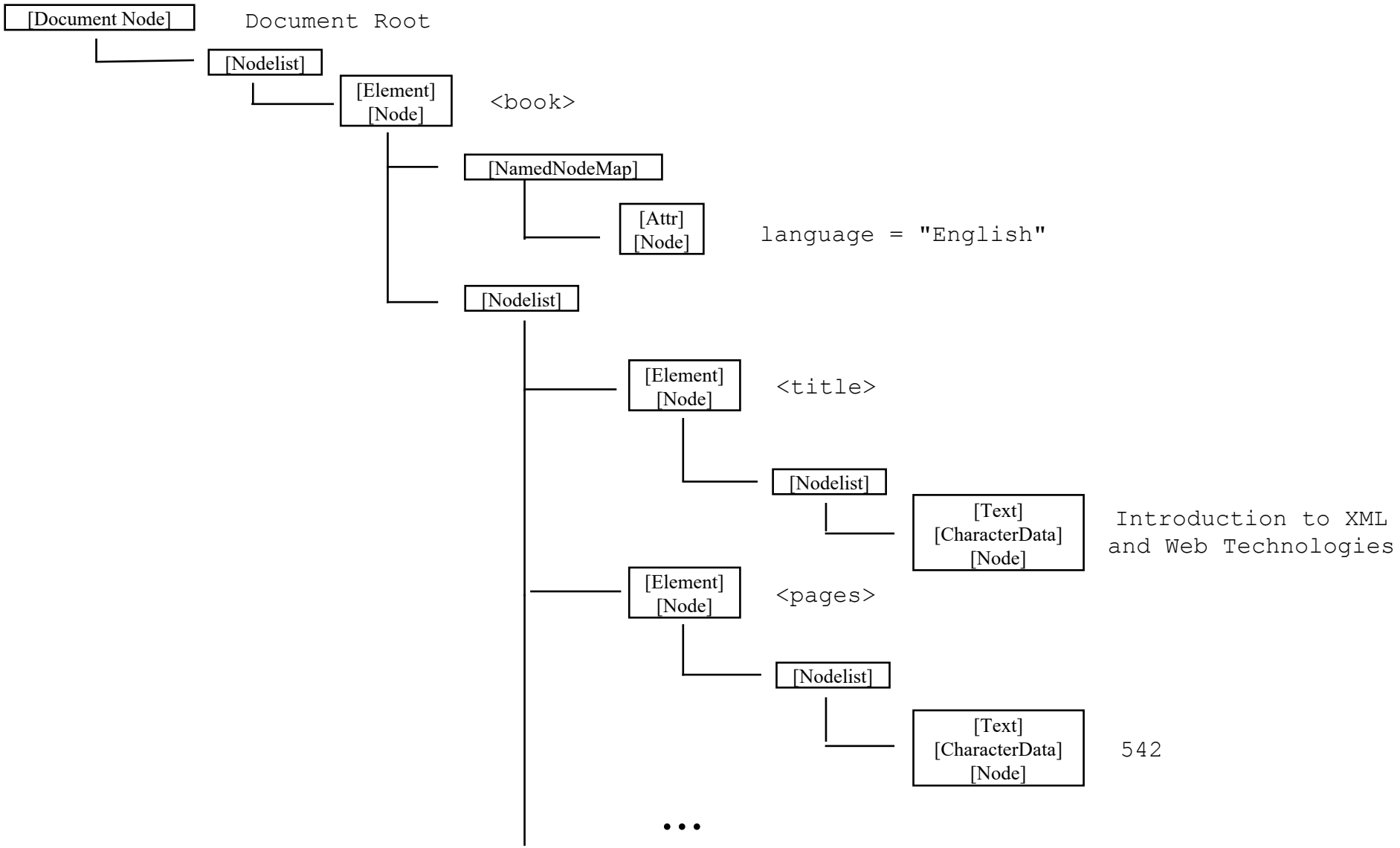
The DOM Structure Model

- Some example DOM objects:
  - `Document` - consisting of `Element`, `ProcessingInstruction`, `Comment`, `DocumentType`
  - `Element` - consisting of `Element`, `Text`, `Comment`, `ProcessingInstruction`, `CDATASection`, `EntityReference`
- For a full set of DOM level 2 objects, see:
  - http://www.w3.org/TR/DOM-Level-2/

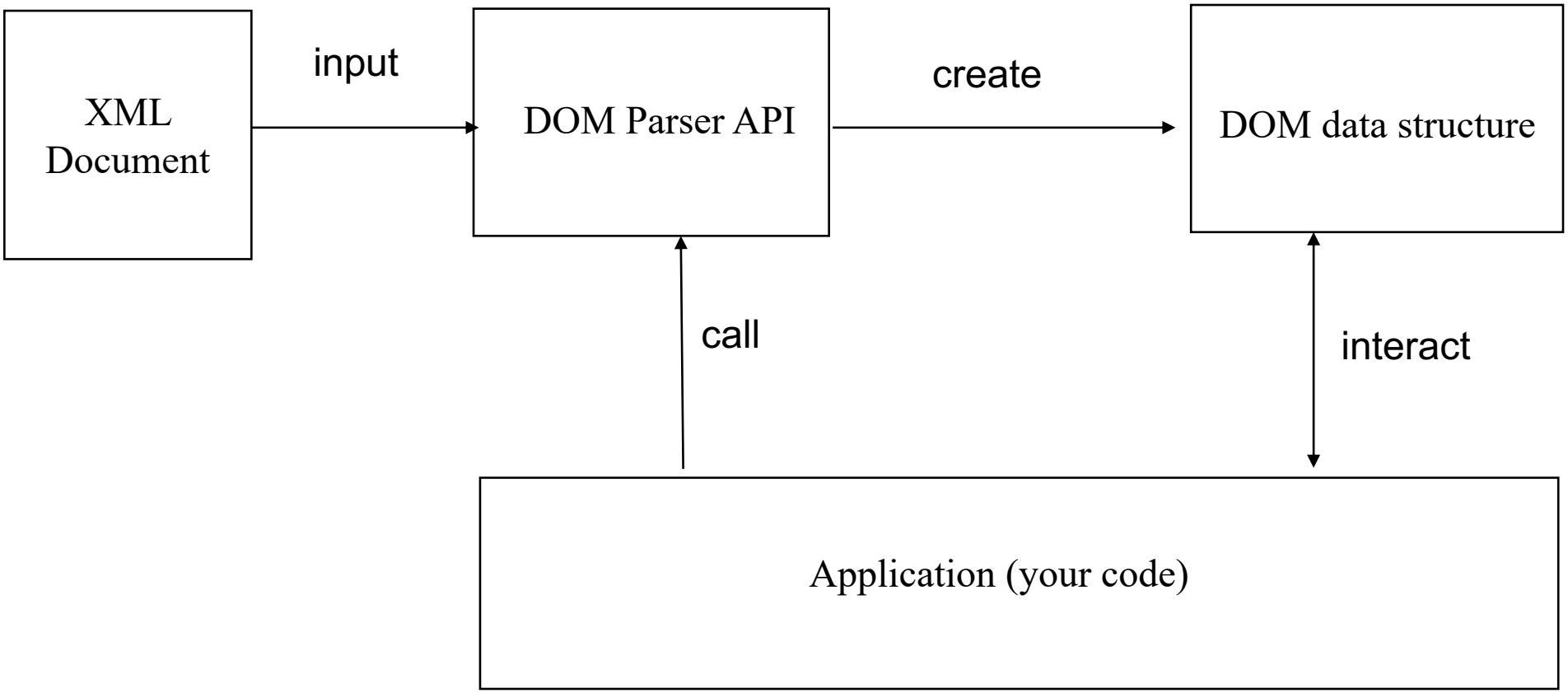
Example XML Document

```
<?xml version="1.0"?>
<book language="English">
  <title>Introduction to XML and Web Technologies
  </title>
  <pages>542</pages>
  <publisher>
    <name>Addison Wesley</name>
    <city>Sydney</city>
  </publisher>
</book>
```

Example DOM Representation



XML DOM Processing



Node Type Constants in XML::DOM

- XML::DOM has the following pre-defined constants to refer to the types of nodes:

Constant	Actual Value
UNKNOWN_NODE	0
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENITITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
…	
XML_DECL_NODE	15
ATTLIST_DECL_NODE	16

Some Example Sub-classes of `XML::DOM`

- `XML::DOM::Node`
  - Class describing any node in the DOM structure (i.e. the XML tree)
- `XML::DOM::NodeList`
  - Class describing a collection of nodes
- `XML::DOM::Element`
  - Class describing any element
- `XML::DOM::Attr`
  - Class describing any attribute
- `XML::DOM::Text`
  - Class describing text nodes (these nodes that contain the text between the start and end tags)

Example Methods in `XML::DOM::Node`

- Access data in the DOM object:
  - `getNodeType`
    - Returns the integer constant indicating the type of node
  - `getNodeName`
    - Returns the name of the node as a string
  - `getChildNodes`
    - Returns a list of all the children of the node
  - `getFirstChild`
    - Returns the first child of the node as an object

Example Methods in `XML::DOM::Node`

- `getLastChild`
  - Returns the last child of the node as an object
- `hasChildNodes`
  - Returns true if the node has child nodes
- `getAttributes`
  - Returns an object containing all the attributes of the node
- `getElementsByTagName("string")`
  - Returns a list of all the elements under the node with the name `"string"`

Example Methods in `XML::DOM::Node`

- Change the DOM object:
  - `insertBefore(newnode, referencenode)`
    - Insert the `newnode` immediately before the `referencenode`
  - `replaceChild(newnode, oldnode)`
    - Replace `oldnode` with `newnode`
  - `removeChild(childnode)`
    - Delete `childnode` from the tree
  - `appendChild(childnode)`
    - Append `childnode` to the end of the node's children

Example Methods in `XML::DOM::NodeList`

- `item(i)`
  - Returns the content of the i^{th} item in the list of nodes; `i` starts from 0
- `getLength`
  - Returns the number of items in the `nodelist`

Methods in `XML::DOM::Element`

- `XML::DOM::Element` inherits from `XML::DOM::Node`, and so has access to all the methods available to `XML::DOM::Node`
- Most of the nodes in a DOM tree are of the `XML::DOM::Element` type

Methods in `XML::DOM::Element`

- Example extra methods:
  - `getTagName`
    - Returns name of the element as a string
  - `setTagName(newname)`
    - Change the name of the element
  - `getAttribute(attributename)`
    - Returns the value of the attribute as a string
  - `setAttribute(attributename, value)`
    - Gives an attribute a new value
  - `removeAttribute(attributename)`
    - Remove an attribute from the element node

Methods in `XML::DOM::Text`

- `XML::DOM::Text` is the second most common node type in a DOM tree (after `XML::DOM::Element`)
- As with `XML::DOM::Element`, `XML::DOM::Text` inherits from `XML::DOM::Node`, and so also has access to all of the methods available to `XML::DOM::Node`

Methods in `XML::DOM::Text`

- Examples of extra `XML::DOM::Text` methods:
  - `getData`
    - Returns the text contained in the node
    - Can also use **`getNodeValue`**
  - `setData(text)`
    - Set the data in the node to the `text` given

Our Course XML Document

```
<course>
     <name>Bachelor of Science - Internet Computing</name>
     <duration>3 years</duration>
     <unit>
          <title>ICT375 Advanced Web Programming</title>
          <lecturer>
               <surname language="English">Xie</surname>
               <othernames language="English">Hong</othernames>
               <email>H.Xie@murdoch.edu.au</email>
          </lecturer>
     </unit>
     <unit>
          <title>ICT283 Data Structures And Abstraction</title>
          <lecturer>
               <surname>Rai</surname>
               <othernames>Shri</othernames>
               <email>s.rai@murdoch.edu.au</email>
          </lecturer>
     </unit>
</course>
```

Murdoch UNIVERSITY

A Simple Example Using XML::DOM

```
use XML::DOM;
my $parser = new XML::DOM::Parser;
my $dom_obj;

die "Unable to parse XML document\n"
    unless $dom_obj = $parser->parsefile("course.xml");
# at this point the DOM tree is stored in $dom_obj

my @nodes = $dom_obj->getElementsByTagName("unit");
foreach $elem (@nodes)
{
    if ( $elem->getNodeType == ELEMENT_NODE ) {
        print $elem->getTagName, "\n";
    }
    # Do other things with $elem
    # ...
}
```

What the Script Does

- Attempt to open and parse `course.xml`
- If the document is not well-formed then stop with an error message
- If document is well-formed, then create a DOM object and assign it to the variable `$dom_obj`
- Get all elements with name “unit” from `$dom_obj` and assign to the array `@nodes`
- For each of those elements:
  - Print the tag name if it is an ELEMENT_NODE
  - Do other things...

Another Example Using XML::DOM

```
use XML::DOM;
my $dom_obj;
my $xml_file = shift; # access commandline argument
my $parser = new XML::DOM::Parser;

die "Unable to parse XML document\n"
    unless $dom_obj = $parser->parsefile($xml_file);
# at this point the DOM tree is stored in $dom_obj

foreach $elem ($dom_obj->getElementsByTagName("title"))
{
   foreach $child ($elem->getChildNodes) {
      print $child->getNodeValue;
   }
   print "\n";
}
```

What the Script Does

- Read a file name from the command line
- Attempt to open and parse the file
- If document is not well-formed then stop with an error message
- If document is well-formed, then create a DOM object and assign it to the variable `$dom_obj`
- Get all “title” elements from `$dom_obj`
- For each of these:
  - Get all its child nodes, ie. the text nodes containing the “title”, and print the value

Text Nodes in Elements

- The text in the tags belongs to a child Text node of an element node, instead of the element node itself
  - Eg: In the previous code, the text in the "title" tag belongs to the *child* of the "title" node, instead of the "title" node itself

Another Example Using XML::DOM

- Convert the whole constructed DOM object into a string, and print it

```
use XML::DOM;

my $dom_obj;
my $xml_file = shift;
my $parser = new XML::DOM::Parser;

die "Unable to parse XML document\n"
  unless $dom_obj = $parser->parsefile($xml_file);

print $dom_obj->toString;
```

Another Example Using XML::DOM

- Remove every "lecturer" element from every "unit" element

```
use XML::DOM;
my $xml_file = shift;
my $parser = new XML::DOM::Parser;
my $course = $parser->parsefile($xml_file);
my @units = $course->getElementsByTagName("unit");
foreach $unit (@units){
   foreach $child ($unit->getChildNodes){
      if ($child->getNodeName eq "lecturer"){
         $unit->removeChild($child);
      }
   }
}
print $course->toString;
```

An Example of Other XML Modules

- Open and parse the XSLT file `default.xsl`
- Transform `course.xml` using XSLT templates according to `default.xsl`
- Print the result on screen

```
use XML::XSLT;
my $xsl_file = "default.xsl";
my $xml_file = "course.xml";

my $xslt = XML::XSLT->new($xsl_file);

$xslt->transform($xml_file);
print $xslt->toString;
```

Murdoch UNIVERSITY

XML Parsing and Processing Using Node.js

- **Note:** to this point we have used Perl to demonstrate XML parsing and processing
- However, with the Node.js `xmldom` package we can parse and process XML documents using the **same DOM methods** that we have seen using Perl's `XML::DOM` module
- The `DOMParser()` method (from the `xmldom` package) can be used to parse XML or HTML source stored in a string into a DOM Document

XML Parsing and Processing Using Node.js


```
// import module
var xmldom = require("xmldom").DOMParser;

// constructs a new DOMParser object
var parser = new xmldom();
// constructor must return a new DOMParser object

// call parseFromString method to return Document object
var doc = parser.parseFromString( str, type );

Example:
var parser = new xmldom();
var doc = parser.parseFromString(XMLSource,
                                 "application/xml");
```

XML Parsing and Processing Using Node.js

- To parse a document:
  - Import the `xmldom` module
  - Use the `DOMParser` constructor to create an object of that type
  - Use the `parseFromString` method to access and process the document
    - The method has two parameters: `str` the data to be parsed, and the MIME `type` (XML, HTML, or Text)
    - A Document object containing the parsed content is returned if successful, otherwise an error

Example XML File: books.xml

```
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>Creating Apps with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>Architect Battles.</description>
   </book>
</catalog>
```

Example: Reading from XML File

```
// import parser from xmldom package
var xmldom = require('xmldom').DOMParser;
var fs = require('fs');

var parser, doc, targetNodes, i, targetObj, fcObj;

// use fs to read xml document
fs.readFile('books.xml', 'utf-8', function (err, data) {
    if (err) { throw err; }

    // construct parser
    parser = new xmldom();
    // call method to parse document - not the type
    doc = parser.parseFromString(data, 'application/xml');

    // use DOM Node method
    targetNodes = doc.getElementsByTagName('genre');
```

Example: Reading from XML File

```
    // go through all returned nodes
    for (i in targetNodes) {
        // process current ith node
        targetObj = targetNodes[i];
        // if it has a firstchild
        if (targetObj.firstChild) {
            // obtain the node value
            fcObj = targetObj.firstChild.nodeValue;
            // compare this to the target genre
            if (fcObj === 'Computer') {
                // display book 'title' corresponding to target
                console.log(targetObj.parentNode.
                                getElementsByTagName('title')[0].
                                firstChild.nodeValue);
            }// end fcObj if
        }// end targetObj if
    }
); // end readFile function
```

Murdoch UNIVERSITY

Example: Reading from XML File

- Open a file as a string using `fs.readFile`
- Use `xmldom` to parse the XML into a DOM object using `parser.parseFromString`
- This returns a DOM object to traverse the tree using common DOM methods like `getElementsByTagName`
  - We can use such methods to find elements and perform whatever task we require
  - In the example, the script displays all book titles where the genre is listed as `'Computer'`

Writing DOM Tree to String Using `XMLSerializer`

- To write the DOM tree back to a string we use `XMLSerializer`

```
// Constructs a new XMLSerializer object
var serializer = new XMLSerializer();

// use the serializeToString method
var str = serializer.serializeToString(doc);

// Serializes doc into a string using an XML
// serialization
// Throws TypeError exception if doc is not a Node
// or an Attr object
```

Example: Using XMLSerializer

```
var fs = require('fs');
var DOMParser = require('xmldom').DOMParser;
var XMLSerializer = require('xmldom').XMLSerializer;

fs.readFile("myFile.xml", "utf-8", function(err, data) {
    // CREATE/PARSE XML OBJECT FROM STRING
    var CC = new DOMParser().parseFromString(data);

    // SET "test" VALUE (<name>default</name> TO <name>test</name>)
    //CC.getElementsByTagName("name")[0].childNodes[0].nodeValue = "test";
    CC.getElementsByTagName("name")[0].childNodes[0].data = "test";
    // THIS OUTPUTS "test"
    console.log(CC.getElementsByTagName("name")[0].childNodes[0].nodeValue);

    // SERIALIZE TO STRING
    var xmlString = new XMLSerializer().serializeToString(CC);
    console.log(xmlString);
    // OR WRITE TO FILE USING fs.writeFile
});
```

Murdoch UNIVERSITY

References

- Perl online documentation for `XML::DOM`
- For other XML packages, search www.cpan.org
- Node.js documentation for `xmldom`:
- https://www.npmjs.com/package/xmldom

Ok, that is the link to XML!

- The whole “web development” scene is based on many technologies that work together, or alongside each other
- Of necessity, many parts of the picture are based on similar methods and techniques
  - The DOM (Document Object Model) is one of the links connecting these technologies
- We cannot cover everything!
  - But we can introduce, and show the links between the technologies … then its up to you!

Murdoch UNIVERSITY

XML Parsing and Processing in Other Programming Languages

Lecture 7 (B)

Learning Objectives

- In the scheme of what we are doing in this unit:
  - A major part of developing an XML solution is to design and implement software to deal with the information in XML documents
  - To do this, we must know how to parse and process XML documents
  - For us to select the appropriate development environment to use, we should have an idea how much support different languages have for XML parsing and processing

Lecture Outline

- An overview of XML parsers and processors available in other programming languages besides Perl
- Widely-used languages with good XML support
- DOM versus SAX approaches
- Examples of available tool-kits

Programming Language Support for XML in Perl

- In the previous lecture, we predominantly used Perl (with some Node.js) examples to illustrate XML parsing and processing
- We did so because:
  - Perl is the most powerful language around for text processing, which at the introductory level should be what students concentrate on in order to understand XML structures
  - Perl has a very simple set of modules for XML parsing, and it is complete enough to serve our needs

Programming Language Support for XML in Other Languages

- The above reasons do not mean we ignore available support for XML in other programming languages
- There is comprehensive support for XML in other languages like Java, C/C++, and Microsoft's C# language

JAVA's Support for XML

- Some example Java XML APIs:
  - Sun's Java APIs for XML (JAX)
  - Xerces parser from Apache XML Project
  - IBM's XML4J
- See other examples at:
  - http://www.xml.com/pub/rg/Java
  - http://www.xml.com/pub/rg/Java_Parsers

Example Code from JAX

```
DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse("priceList.xml");

NodeList list = document.getElementsByTagName("name");
Node thisNode = list.item(0);// loop through list
Node thisChild = thisNode.getChildNode();
if (thisNode.getFirstChild() instanceof org.w3c.dom.TextNode){
   String data = thisNode.getFirstChild().getData();
}
if (data.equals("Mocha Java")) {
   // new node will be inserted before Mocha Java
   Node newNode = document.createElement("coffee");
   Node nameNode = document.createElement("name");
   TextNode textNode = document.createTextNode("Kona");
   nameNode.appendChild(textNode);

   Node priceNode = document.createElement("price");
   TextNode tpNode = document.createTextNode("13.50");
   priceNode.appendChild(tpNode);

   newNode.appendChild(nameNode);
   newNode.appendChild(priceNode);
   thisNode.insertBefore(newNode, thisNode);
}
```

- Note that the DOM method names are the same as the Node and Perl XML::DOM APIs
- These names are defined in **DOM specification**, and are **language independent**

Code Source: http://java.sun.com/webservices/docs/ea2/tutorial/doc/IntroWS5.html#63986

Programming Language Support for XML in Other Languages

- C and C++ also have a large support base for XML
  - Most core system-based processes are still implemented in C/C++
  - The popular Expat parser by James Clark is implemented in C

Programming Language Support for XML in Other Languages

- For scripting languages other than Perl, Python has gained popularity as a language for XML programming
- Most languages mentioned, such as C, C#, Java, and Python, have major libraries supporting XML

Programming Language Support for XML in Other Languages

- JavaScript is also another popular choice for writing XML parsers and processors
- There is a large base of JavaScript programmers who picked up the language through web-page design
  - Most XML documents are directed towards being served over the Web

Programming Language Support for XML in Other Languages

- JavaScript is very lightweight once a browser is started and it doesn't involve large resource requirements when starting up and processing
- Until recently, JavaScript did not have the large amount of support, in terms of data types and libraries, as some other programming languages
  - Due to Node.js, JavaScript libraries are now quite numerous, but support can still be a bit limited

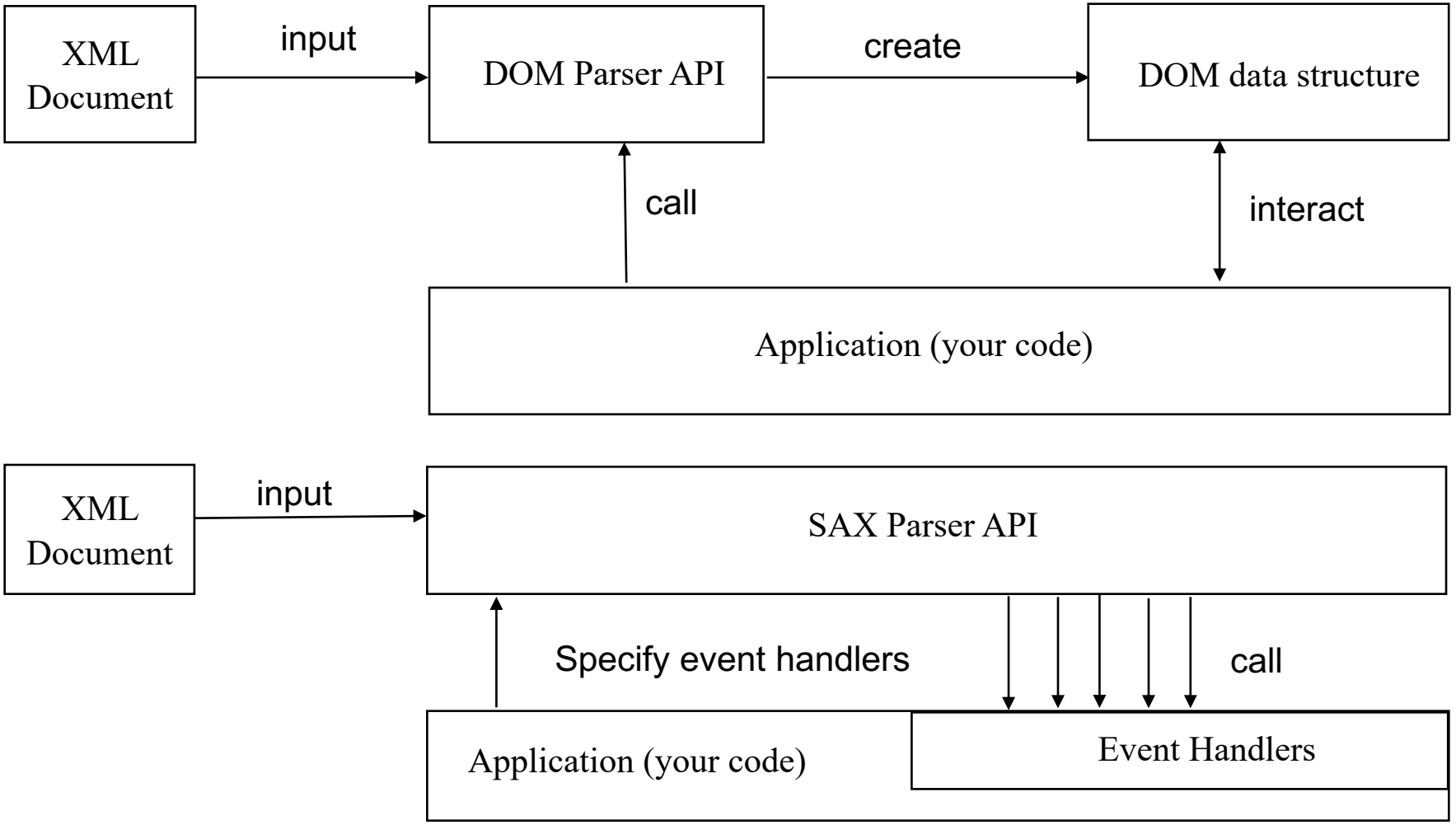
DOM vs SAX

- In the last lecture, we looked at the parsing and processing approach using DOM
- Another approach (briefly mentioned previously) for parsing and processing XML is SAX (Simple API for XML), which is just as popular as the DOM approach
  - Written by David Megginson (http://www.saxproject.org/)

The SAX Approach

- SAX approaches parsing in the same vein as `Expat` (and as a consequence Perl's `XML::Parser` and the Node.js module `node-xml` from the last lecture)
- SAX provides facilities to define event handlers for handling different parts of the XML document as the parser encounters them

DOM vs SAX


XML Document
input
DOM Parser API
create
DOM data structure
call
interact
Application (your code)
XML Document
input
SAX Parser API
Specify event handlers
call
Application (your code)
Event Handlers

XML Processing with SAX

- A parser which implements SAX (i.e. a SAX Parser) functions as a stream parser, with an event-driven API
- The user defines a number of callback methods that will be called when events occur during parsing
- These include SAX events for:
  - XML Text nodes
  - XML Element nodes
  - XML Processing Instructions
  - XML Comments

XML Processing with SAX

- Events are fired when each of these XML features are encountered, and again when the end of the features are encountered
- XML attributes are provided as part of the data passed to element events
- SAX parsing is uni-directional
  - That is, previously parsed data cannot be re-read without starting the whole parsing operation from the very beginning

DOM vs SAX

- The SAX approach has its advantages because it:
  - Is efficient in dealing with large files
    - Does not build a large memory map of the whole document
    - Only parses what it has been instructed to parse
    - Therefore, it can begin processing even before the parser finishes reading the whole document eg: an event handler can start a new thread

DOM vs SAX

- The SAX approach has its advantages because it:
  - Concentrates on the content rather than the layout
  - When there are a lot of external API calls (eg: database specific system calls) your event handlers can deal with the data

DOM vs SAX

- The DOM approach has its advantages because it:
  - Returns a data-structure that a programmer can store and easily manipulate
  - Encapsulates all information about the structure of the document, some of which SAX does not return (eg: order of attributes)

A Question of Efficiency

- The SAX approach is an important part of XML parsing due to efficiency
  - It is an efficient approach when parsing very large documents, especially when you only want a small part of the data
- DOM on the other hand is also important due to the data and object-centric way we approach most programming today
  - We need to have adequate facilities to store and easily manipulate data

Popular XML Parsing Tool-kits Today

- Those which have a long history behind them, upon which many new XML software libraries are built:
  - Expat by James Clark (www.jclark.com/xml/expat.html)
  - SAX: not defined in any language, and has implementations in Perl, Java, C/C++ and many other languages
  - XP: also by James Clark (www.jclark.com/xml/xp/index.html)

Popular XML Parsing Tool-kits Today

- Continued:
  - Lark by Tim Bray, in Java (www.textuality.com/Lark/)
  - LT XML by Language Technology Group at University of Edinburg (www.ltg.ed.ac.uk/software/xml/)
  - XML::Parser by Larry Wall (see Perl documentation)
  - SXP: Silfide XML Parser (www.loria.fr/projets/XSilfide/EN/sxp/

Popular XML Parsing Tool-kits Today

- Widely used today:
  - XML for Java (XML4J) by IBM AlphaWorks - widely used and conforms well to W3C standards (www.alphaworks.ibm.com/tech/xml4j)
  - Microsoft XML Parser (MSXML) - implemented as a COM component (msdn.microsoft.com/xml/general/xmlparser.asp)
  - Ælfred in Java concentrates on optimising speed / size, especially good to use in applets (http://www.opentext.com/services/content_management_services/xml_sgml_solutions.html#aelfred_and_sax)

Popular XML Parsing Tool-kits Today

- Widely used today:
  - Java Standard Extensions for XML by Sun Microsystems (java.sun.com/products/xml)
  - Perl XML modules (http://cpan.valueclick.com/modules/by-module/XML/)
  - Python parser (www.python.org/topics/xml)
  - Node.js libraries: node-xml, xmldom, xml2js
  - … and many, many more

What Language Should You Choose?

- Depends on trade-offs between:
  - Whether the languages and their libraries have support for features which will enhance your particular applications/solutions
  - What languages you and your project team are familiar/comfortable working with
  - What language the developers in the problem area is mostly using - for better integration
  - What resources (especially software) your organization owns, has access to, or is planning to obtain - again for integration

Reference

- XML can be validated at:
  - https:xmlvalidation.com