B336 Advanced Internet Computing

XML Parsing and Processing With Perl

Learning Objectives

- Learn some of the available modules in Perl to parse and process XML documents.

- Look at some example code in using these modules.

Learning Objectives

- In the scheme of what we are doing in this unit:
    - We are studying how to use XML as an important set of Internet technologies to use as solutions in different areas (for example, the problem defined in Assignment 2).
    - A major part of developing an XML solution is to design and implement software to deal with the information in these XML documents. To do this, we must know how to parse and process XML documents.
    - Perl has a set of packages for XML, and we will use that as an introduction to tools available in a programming language to handle XML.

Lecture Outline

- The XML::Parser module
- The XML::DOM module
- Other modules in Perl

Parsing XML in Perl

- As mentioned in the previous lecture, 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
  - put the relevant data in appropriate data structure for program to use

- In Perl, there is the XML::Parser module which can do this.

XML::Parser

- XML::Parser is a basic Perl module we can use 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 is hidden in parser such as XML::Parser.

- Expat and XML::Parser are non-validating parsers.

Event handlers in XML::Parser

- XML::Parser is event-driven.
    - It works by allowing you to specify event handlers for different situations the Parser encounters during parsing.
    - You do not have to deal with the low-level character-by-character parsing (and verifying). You only what happens when you encounter certain “components” of an XML document (eg. a start tag).

Some Event Handlers in XML::Parser

- For encountering any start tags.
  - Form: `start_handler (expat, element, [attr_name, attr_value])`
- For encountering any end tags.
  - Form: `end_handler (expat, element)`
- For encountering any comments.
  - Form: `comment_handler (expat, comment_data)`

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) ;
}
```

XML::Parser and Data Structure

- Although XML::Parser is adequate in parsing and verifying the document, it does not construct a useful object or data structures to represent the XML document.
    - That is left to the programmer using XML::Parser.
    - When using XML::Parser, we think about the events that occur rather than the objects to be manipulate.
- XML::DOM is an alternative module which does construct a useful data structure to manipulate.

The Document Object Model (DOM)

- The DOM is a platform- and language-independent describing the content, structure and style of documents, by putting them in specified *objects*.

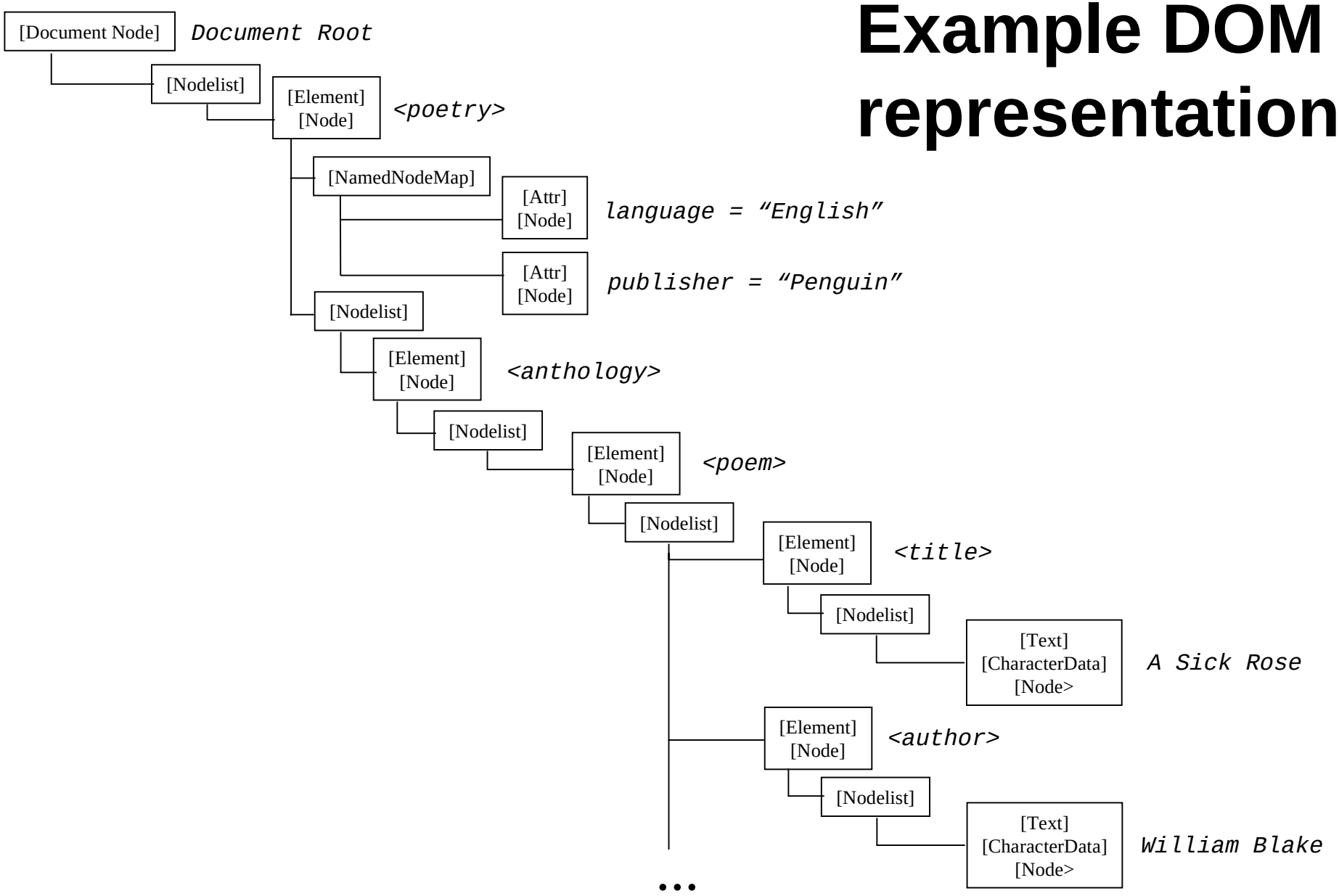
- DOM is currently a W3C standard
  - Currently developing level 3
  - Based originally on Netscape's, and later Microsoft's, efforts to allow client-side scripting (eg JavaScript) to manipulate HTML pages.

The DOM Structure Model

- The DOM presents documents as a hierarchy of ***node*** objects, some with ***child*** nodes of various types, and others with ***leaf*** nodes that cannot have anything below them.
    - This basic idea is the basis behind the structure of XML documents.

- Some example DOM objects:
    - `Document` - consisting of `Element`, `ProcessingInstruction`, `Comment`, `DocumentType`
    - `Element` - consisting of `Element`, `Text`, `Comment`, `ProcessingInstruction`, `CDATASection`, `EntityReference`.

Example DOM representation



XML::DOM

- XML::DOM is a DOM level 1 compliant parser module in Perl
    - It extends XML::Parser by constructing a DOM object when parsing.
    - The programmer can then manipulate this object after calling the parsing method.

Constants in XML::DOM

- XML::DOM has the following predefined constants:
  - see Table 10.2 in section 5 of the unit reader

Some example subclasses of XML::DOM

- XML::DOM::Node
  - class describing any node in the DOM structure (ie. the XML tree)
- XML::DOM::Element
  - class describing any element
- XML::DOM::Attr
  - class describing any attribute
- XML::DOM::CharacterData
  - class describing any CDATA section
- XML::DOM::XMLDecl
  - class describing the opening `<?xml ...?>` declaration

Some example methods in XML::DOM::Node

- Access data in the DOM object:
  - getNodeType
  - getNodeName
  - getFirstChild
  - getAttributes
  - ...
- Change the DOM object:
  - replaceNode (newnode, oldnode)
  - removeChild (childnode)
  - ...

An 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(“poetry.xml”) ;

my @nodes = $dom_obj->getElementsByTagName(“line”) ;

foreach $elem (@nodes) {

    if ($elem->getNodeType == ELEMENT_NODE) {
        print $elem->getTagName, “\n” ;
    }

    # Do other things with $elem
    # ...
}
```

An Example of other XML modules

```
use XML::XSLT;

my $xsl_file = “default.xsl” ;
my $xml_file = “poetry.xml”

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

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

References

- Introduction to XML processing in Perl, and to XML::Parser and XML::DOM is found in section 5 of the unit reader.
- More details (including XML::XSLT) are in the Perl documentation, under the XML library
    - Assuming you have installed the packages
- For all available XML packages, search at www.cpan.org.

In the remaining few lectures on XML...

- Alternative methods to processing XML in Java and other programming languages.

- Putting everything together by looking at example XML applications.