3.2 Document Object Model (DOM)

- How to provide uniform access to structured documents in diverse applications (parsers, browsers, editors, databases)?
- Overview of W3C DOM Specification
  - DOM standardisation began in Spring 1997
  - the second recommendation in the “XML-family” of standards
    - Level 1, W3C Rec, Oct. 1998
    - Level 2, W3C Rec, Nov. 2000
    - Level 3, (an early) W3C Working Draft

DOM: What is it?

DOM Specification:
“...a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents...”

DOM: What is it? (2)

- An API for XML and HTML documents
  - allows programs and scripts to build documents, navigate their structure, add, modify or delete elements and content
  - Provides a foundation for developing querying, filtering, transformation, rendering etc. applications on top of DOM implementations

DOM structure model (1)

- Roughly similar to the XSLT/XPath data model (to be discussed later)
  - a parse tree
- Example document:

```
<article>
  <para>Written by the lecturer.</para>
  <fig file="pekka.jpg" caption="The Lecturer"/>
</article>
```

DOM structure model (2)

- Corresponding structure model:

DOM structure model (3)

- The tree-like structure
  - is implied by the abstract relationships defined by the programming interfaces;
  - does not necessarily reflect data structures used by an implementation (but probably does)
- Based on O-O concepts:
  - methods (to access or change object’s state)
  - interfaces (declaration of a set of methods)
  - objects (encapsulation of data and methods)
Object-based document modelling

- Object model covers
  - structure of a document
  - behaviour of a document and its constituent objects
- DOM defines
  - interfaces and objects for representing and manipulating documents
  - semantics of these interfaces
  - relationships between interfaces and objects

DOM: Background & support

- Background:
  - JavaScript, “Dynamic HTML”
  - APIs for structured document editors and repositories
- Implementations
  - Java-based parsers (e.g., Sun Project X, IBM XML4J, Apache Xerces)
  - MS IE5 browser: COM programming interfaces for C/C++ and MS Visual Basic, ActiveX object programming interfaces for script languages
  - Others? Non-parser implementations? (Vendor involvement in DOM WG was quite impressive.)

Structure of DOM Level 1

- Two parts:
  I: DOM Core Interfaces
    - Fundamental interfaces
      - low-level interfaces to structured documents
      - XML specific: CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction
  II: DOM HTML Interfaces
    - more convenient to access HTML documents
    - (we ignore these)

DOM Level 2

- Level 1 intentionally limited to representation and manipulation of document structure and content
  - document instance only; no access to the contents of a DTD
- DOM Level 2 adds
  - support for namespaces
  - accessing elements by ID attribute values
  - optional features
    - interfaces to document views and stylesheets
    - an event model (for, say, user actions on elements)
    - methods for traversing the document tree and manipulating regions of document (e.g., selected by the user of an editor)

DOM Language Bindings

- DOM interfaces are defined using OMG Interface Definition Language (IDL; Defined in Corba Specification)
- Language bindings (implementations of DOM interfaces) defined for
  - Java and
  - ECMAScript (standardised JavaScript/JScript)

Overview of DOM Core Interfaces

- All document components can be accessed through the base interface Node, or through interfaces derived from it:
  - Document, DocumentFragment, Element, Attr, Text, Comment (fundamental interfaces)
  - CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction (extended interfaces)
Additional Core Interfaces

- to handle ordered lists of nodes: NodeList
  - e.g. from Node.childNodes or Element.getElementsByTagName("name")
    - all descendant elements of type "name" in document order
  - "*" a wildcard matching any element type
- to access unordered sets of nodes by name: NamedNodeMap
  - e.g. from Node.attributes

NodeLists and NamedNodeMaps are "live":
- changes to the document structure reflected to their contents

Object Creation in DOM

- Each DOM object X lives in the context of a Document: X.ownerDocument
- Objects implementing interface Y are created by factory methods
  D.createElement(...)
where D is a Document object. E.g:
- createElement("A"), createAttribute("href"),
  createTextNode("Hello!")
- Creation and persistent saving of Documents left to be specified by implementations

Object manipulation in DOM

- Manipulating children of a node:
  - node.insertBefore(newChild, refChild)
  - node.replaceChild(newChild, oldChild)
  - node.removeChild(oldChild)
  - node.appendChild(newChild)
- Accessing a specific child, or iterating over children by indexing items of a NodeList:
  - Using the Java binding of DOM:
    for (i=0; i<node.getChildNodes().getLength(); i++)
    process(node.getChildNodes().item(i));

Content and element manipulation

- Manipulating CharacterData D:
  - D.substringData(offset, count)
  - D.appendData(string)
  - D.insertData(offset, string)
  - D.deleteData(offset, count)
  - D.replaceData(offset, count, string)
    (= delete + insert)
- Accessing attributes of an Element object E:
  - E.getAttribute(name)
  - E.setAttribute(name, value)
  - E.removeAttribute(name)

Accessing properties of a Node

- Node.nodeName
  - for an Element = tagName
  - for an Attr: the name of the attribute
  - for Text = #text etc
- Node.nodeValue
  - content of a text node, value of attribute, ...;
    null for an Element.
  - (in XSLT/Xpath: the full text content of elem)
- Node.nodeType:
  - numeric constants (1, 2, 3, ...
  - ELEMENT_NODE, ATTRIBUTE_NODE, ...
  - TEXT_NODE, NOTATION_NODE

Accessing Context of a Node

- Node.parentNode
  - for all nodes except for:
    - Document, DocumentFragment, Attr nodes
    - Entity, Notation nodes
  - DocumentFragments used as temporary holders of forests; Attrs, Entities and Notations hang in a NamedNodeMap
- Node.firstChild, Node.lastChild,
  Node.previousSibling, Node.nextSibling
  - null if the child or sibling does not exist