DOM Interface subset 1/3

- Document
  - attributes
    - `documentElement` – top element in document tree
  - methods
    - `createElement (tag)` – creates and returns element 'tag'
    - `createElementNS (ns, tag)` – creates and returns element 'tag' in namespace 'ns'
    - `createTextNode (text)` – creates and returns text node with content 'text'
    - …

DOM Interface subset 2/3

- Node
  - attributes
    - `nodeName` – name of any node
    - `nodeValue` – value of text or comment node
    - `nodeType` – type of node
    - `parentNode` – node one level higher up in the tree
    - `childNodes` – list of children nodes
    - `firstChild` – first child of current node
    - `lastChild` – last child of current node
    - `previousSibling` – previous node with same parent
    - `nextSibling` – next node with same parent
    - `attributes` – list of name/value pairs
  - methods
    - `insertBefore (newchild, pos)` – inserts newchild before pos
    - `replaceChild (new, old)` – replaces child node old with new
    - `removeChild (old)` – removes child node old
    - `appendChild (new)` – adds child node new to end of list
    - `hasChildNodes` – returns whether or not there are children

DOM Interface subset 3/3

- Element (which is also a Node)
  - methods
    - `getAttribute (name)` – returns value associated with name
    - `setAttribute (name, val)` – sets value for name
    - `getElementsByTagName (tag)` – returns list of nodes from among children that match tag
- NodeList
  - attributes
    - `length` – number of nodes in list
  - methods
    - `item (pos)` – returns the node at position pos
- CharacterData (which is also a Node)
  - attributes
    - `data` – textual data

Schema structure

- Elements are defined by
  - `<element name="…" type="…" minOccurs="…" maxOccurs="…">`
    - *name* refers to the tag.
    - *type* can be custom-defined or one of the standard types. Common predefined types include *string*, *integer* and *anyURI*.
    - *minOccurs* and *maxOccurs* specify how many occurrences of the element may appear in an XML document. *unbounded* is used to specify no upper limits.
- Example
  - `<element name="title" type="string" minOccurs="1" maxOccurs="1"/>`

Sequences

- Sequences of elements are defined using a *complexType* container.

```
<complexType>
   <sequence>
      <element name="title" type="string"/>
      <element name="author" type="string"
               maxOccurs="unbounded"/>
   </sequence>
</complexType>
```

- Note: Defaults for both *minOccurs* and *maxOccurs* are 1

Nested Elements

- Instead of specifying an atomic type for an element as an attribute, its type can be elaborated as a structure. This is used to correspond to nested elements in XML.

```
<element name="uct">
   <complexType>
      <sequence>
         <element name="title" type="string"/>
         <element name="author" type="string"
                  maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>
```

Extensions

- Extensions are used to place additional restrictions on the content of an element.
  - Content must be a value from a given set:

```
<element name="version">
   <simpleType>
      <restriction base="string">
         <enumeration value="1.0"/>
         <enumeration value="2.0"/>
      </restriction>
   </simpleType>
</element>
```

  - Content must conform to a regular expression:

```
<element name="version">
   <simpleType>
      <restriction base="string">
         <pattern value="[1-9]\.[0-9]+"/>
      </restriction>
   </simpleType>
</element>
```

Attributes

- Attributes can be defined as part of *complexType* declarations.

```
<element name="author">
   <complexType>
      <simpleContent>
         <extension base="string">
            <attribute name="email" type="string"
                       use="required"/>
            <attribute name="office" type="integer"
                       use="required"/>
            <attribute name="type" type="string"/>
         </extension>
      </simpleContent>
   </complexType>
</element>
```

Named Types

- Types can be named and referred to by name at the top level of the XSD.
  - `<element name="author" type="uct:authorType"/>`

```
<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string"
                    use="required"/>
         <attribute name="office" type="integer"
                    use="required"/>
         <attribute name="type" type="string"/>
      </extension>
   </simpleContent>
</complexType>
```

Other Content Models

- Instead of *sequence,*
  - *choice* means that only one of the children may appear.
  - *all* means that each child may appear or not, but at most once each.

Many more details about content models can be found in specification!

Schema Namespaces

- Every schema should define a namespace for its elements, and for internal references to types
  - `<schema xmlns="http://www.w3.org/2001/XMLSchema"`

```
        targetNamespace="http://www.uct.ac.za"
        xmlns:uct="http://www.uct.ac.za">

<element name="author" type="uct:authorType"/>

<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string"
                    use="required"/>
         <attribute name="office" type="number"
                    use="required"/>
         <attribute name="type" type="string"/>
      </extension>
   </simpleContent>
</complexType>

</schema>
```

XPath Shorthand

Expression	What it selects in current context
title	"title" children
*	All children
@office	"office" attribute
author[1]	First author node
/uct/title[last()]	Last title within uct node at top level of document
//author	All author nodes that are descendent from top level
.	Context node
..	Parent node
version[number]	Version nodes that have "number" children
version[number='1.0']	Version nodes for which "number" has content of "1.0"

XSLT Language 1/3

- *value-of* is replaced with the textual content of the nodes identified by the XPath expression.
  - Example:
    - `<value-of select="uct:title"/>`
- *text* is replaced by the textual content. Usually the plain text is sufficient.
  - Example:
    - `<text>1.0</text>`
      `1.0`
- *element* is replaced by an XML element with the indicated tag. Usually the actual tag can be used.
  - Example:
    - `<element name="dc:publisher">UCT</element>`
      `<dc:publisher>UCT</dc:publisher>`

XSLT Language 2/3

- *apply-templates* explicitly applies templates to the specified nodes.
  - Example:
    - `<apply-templates select="uct:version"/>`
- *call-template* calls a template like a function. This template may have parameters and must have a *name* attribute instead of a *match*.
- Example:

```
<call-template name="doheader">
   <with-param name="lines">5</with-param>
</call-template>

<template name="doheader">
   <param name="lines">2</param>
   …
</template>
```

XSLT Language 3/3

- *variable* sets a local variable. In XPath expressions, a $ prefix indicates a variable or parameter instead of a node.
  - Example:
    - `<variable name="institution">UCT</variable>`
      `<value-of select="$institution"/>`
      `<place institution="{$institution}"/>`
- Selection and iteration examples:
  - `<if test="position()=last()">…</if>`
  -
```
<choose>
   <when test="$val=1">…</when>
   <otherwise>…</otherwise>
</choose>
```
  - `<for-each select="uct:number">…</for-each>`