

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 Syntax

- Expressions are separated by “/”.
- In general, each subexpression matches one or more nodes in the DOM tree.
- Each sub-expression has the form:
 - `axis::node[condition1][condition2]..`
 - where axis can be used to select children, parents, descendants, siblings, etc.
- Shorthand notation uses symbols for the possible axes.

XPath Shorthand

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

XSLT

- XSLT is a declarative language, written in XML, to specify transformation rules for XML fragments.
- XSLT can be used to convert any arbitrary XML document into XHTML or other XML formats (e.g., different metadata formats).
- Example:
 - ```
<template match="uct:author">
 <dc:creator>
 <value-of select="."/>
 </dc:creator>
</template>
```

## 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>`
 - `<for-each select="uct:number">...</for-each>`