<? xml ?>

hussein suleman
uct csc3002f 2007

Outline

- Markup Languages and XML
- XML Structure
- XML Parsing
- Namespaces
- XML Schema
- Metadata in XML
- XPath
- XSL – XSLT
- XSL – FO
- XQuery
- XML Databases
- References

Markup Languages and XML

Markup

- Markup refers to auxiliary information (a.k.a. tags) that is interspersed with text to indicate structure and semantics.
- Examples:
  - LaTeX uses markup to specify formatting (e.g., \hspace)
  - HTML uses markup to specify structure (e.g., <p>)
- A markup language specifies the syntax and semantics of the markup tags.

Is LaTeX outdated because of its markup language

Markup Example

- Plain text
  - The brown fox jumped over the lazy dog.
- Marked up text
  - `*paragraphstart*The *subjectstart*quick brown fox*subjectend* *verbstart*jumped*verbend* over the *objectstart*lazy dog*objectend*.*paragraphend*`
- Advantages:
  - Aids semantic understanding.
  - Supports automatic translation to other formats.

Can we build a parser for this ML?

SGML

- Standard Generalised Markup Language (SGML) specifies a standard format for text markup. All SGML documents follow a Document Type Definition (DTD) that specifies the structure.

```
<!DOCTYPE uct PUBLIC "-//UCT//DTD SGML//EN">
<title>test SGML document
<author email='pat@cs.uct.ac.za' office=410 lecturer
>Pat Pukram
<version>
   <number>1.0
</version>
```

Why don't we need a closing title tag?

HTML

- HyperText Markup Language (HTML) specifies standard structure/formatting for linked documents on the WWW, as a subset of SGML.
- SGML defines general framework – HTML defines semantics for a specific application.

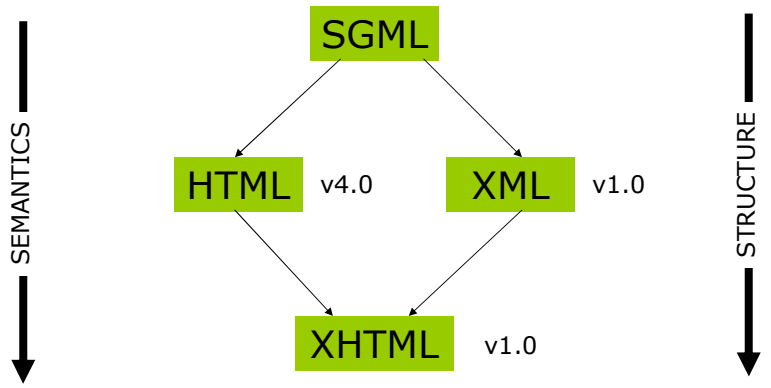
```
<html><head><title>test HTML document</title></head>
<body>
<h1>Author</h1>
<p>Pat Pukram
<br>Lecturer
<br>Email: pat@cs.uct.ac.za
<br>Office: 410
</p>
<h1>Version</h1>
<p>1.0</p>
</body>
</html>
```

XML

- eXtensible Markup Language (XML) is a subset of SGML to ease adoption, especially for WWW use.

```
<uct>
<title>test XML document</title>
<author email="pat@cs.uct.ac.za" office="410"
type="lecturer">Pat Pukram</author>
<version>
   <number>1.0</number>
</version>
</uct>
```

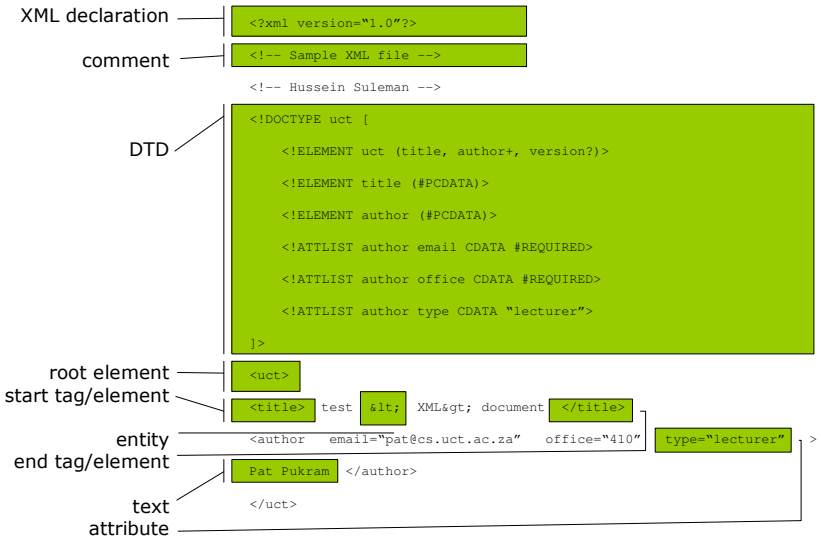
Relationship



XML Primer

- An XML document is a serialised segment of text which follows the XML standard.
  - (http://www.w3.org/TR/REC-xml)
- Documents may contain
  - XML declaration
  - DTDs
  - text
  - elements
  - processing instructions
  - comments
  - entity references

XML Sample

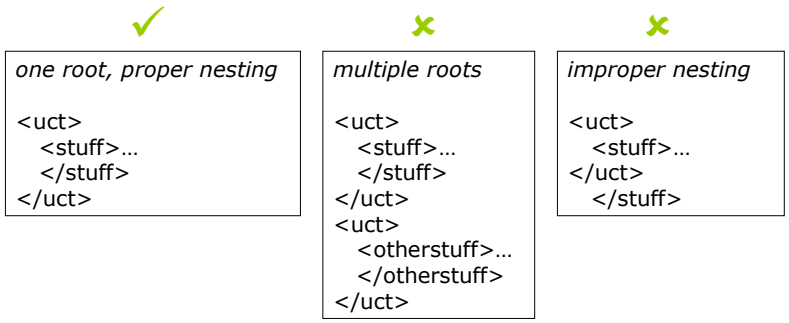


Exercise 1: View XML

- Start the Firefox WWW browser.
- Load the file `uct1.xml` from the workshop folder.
- Use the – and + buttons to collapse and expand subsections of the XML.

Well-formedness

- Well-formed XML documents have a single root element and properly nested matching start/end tags.



Validity

- Valid XML documents strictly follow a DTD (or other formal type definition language).
- Well-formedness enforces the fundamental XML structure, while validity enforces domain-specific structure!
- Why validate? Catch errors, quality assurance, allow structural assumptions …
- SGML parsers, in contrast, had no concept of well-formedness so domain-specific structure had to be incorporated into the parsing phase.

Levels of Correctness

1. Unicode encoding must not contain erroneous characters
2. XML documents **must** be well-formed
   - if there is no single root, then it is an XML fragment
   - if elements are not properly nested, it is not really XML!
3. XML **can** be valid, conforming to a DTD, Schema or other formal description

Exercise 2: View XML Error

- Start the Firefox WWW browser.
- Load the file `uct_error1.xml` from the exercise folder.
- Take note of the error and try to understand what it means.

XML Structure

XML declaration

- `<?xml encoding="UTF-8" version="1.0" standalone="yes" ?>`

- Appears (optionally) as first line of XML document.
- "encoding" indicates how the individual bits correspond to character sets.
- "version" indicates the XML version (usually 1.0).
- "standalone" indicates if external type definitions must be consulted in order to process the document correctly.

recommended for all: standalone
recommended for most European languages: UTF-8

Unicode

- Most XML is encoded in ISO 10646 Universal Character Set (UCS or Unicode).
- Unicode at first supported 16-bit characters, as opposed to ASCII's 8-bits – implying 65536 different characters from most known languages.
- This has since been expanded to 32 bits. The simplest encoding mapping this to 4 fixed bytes is called UCS-4.
- To represent these characters more efficiently, variable length encodings are used: UTF-8 and UTF-16 are standard.

Common characters should take less space to store/transmit - less common characters can take more space!

UTF-16

- Basic Multilingual Plane (characters in the range 0-65535) can be encoded using 16-bit words.
- Endianness (if there are 2 bytes, which one is stored first) is indicated by a leading Byte Order Mark (BOM) e.g., FF FE = little endian UTF-16.
- For more than 16 bits, characters can be encoded using pairs of words and the reserved D800-DFFF range.
  - `D800DC00 = Unicode 0x00010000`
    `D800DC01 = Unicode 0x00010001`
    `D801DC01 = Unicode 0x00010401`
    `DBFFDFFF = Unicode 0x0010FFFF`
- UTF-16 → UCS-4
  - D801-D7C0 = 0041, DC01 & 03FF = 0001
    (0041 << 10) + 0001 = 00010401
- UCS-4 → UTF-16 ?

Ouch!

UTF-8

- Optimal encoding for ASCII text since characters < #128 use 8 bits.
- Variable encoding thereafter
  - Unicode 7-bit = 0vvvvvvv
    Unicode 11-bit = 110vvvvv 10vvvvvv
    Unicode 16-bit = 1110vvvv 10vvvvvv 10vvvvvv
    Unicode 21-bit = 11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
    etc.
- UCS-4 → UTF-8
  - 0001AB45 = 11010 101100 100101
    11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
    = 11110000 10011010 10101100 10100101
    = F09AACA5
- UTF-8 → UCS-4 ?
- UTF-8, like UTF-16, is self-segregating to detect code boundaries and prevent errors.

You mean we can't actually write XML with Notepad/vi ?

Document Type Definition (DTD)

- Defines structure of XML documents.
- Optionally appears at top of document or at externally referenced location (file).

```
<!DOCTYPE uct [
    <!ELEMENT uct (title, author+, version?)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
    <!ATTLIST author email CDATA #REQUIRED>
    <!ATTLIST author office CDATA #REQUIRED>
    <!ATTLIST author type CDATA "lecturer">
    <!ELEMENT version (number)>
    <!ELEMENT number (#PCDATA)>
]>
```

- ELEMENT defines structure of elements.
  - ()=list of children, +=one or more, *=zero or more, ?=optional, PCDATA=text
- ATTLIST defines attributes for each element.
  - #REQUIRED=required, "lecturer"=default, CDATA=text

Elements / Tags

- Basic tagging or markup mechanism.
- All elements are delimited by < and >.
- Element names are case-sensitive and cannot contain spaces (full character set can be found in spec).
- Attributes can be added as space-separated name/value pairs with values enclosed in quotes (either single or double).
  - `<sometag attrname="attrvalue">`

Element Structure

- Elements may contain other elements in addition to text.
- Start tags start with "<" and end with ">".
- End tags start with "</" and end with ">".
- Empty tags start with "<" and end with "/>".
  - Empty tags are a shorthand for no content.
  - Example:
</br> is the same as
  - To convert HTML into XHTML, all
 tags must be in either of the forms above!
- Every start tag must have an end tag and must be properly nested.
- Not well-formed:
  - `<x><a>mmm<b>mmm</a>mmm</b></x>`
- Well-formed:
  - `<x><a>mmm<b>mmm</b></a><b>mmm</b></x>`
- Elements may be repeatable!

Does this work in HTML?

Special attributes

- xml:space is used to indicate if whitespace is significant or not.
  - In general, assume all whitespace outside of tag structure is significant!
- xml:lang indicates the language of the element content.
  - Example
    - `<p xml:lang="en">I don't speak</p> Zulu`
      `<p xml:lang="es">No hablo</p> Zulu`

Entities

- Entities begin with "&" and end with ";".
- Named entity references refer to (are macros for) previously defined textual content – usually defined in an external or internal DTD.
  - Example: © is assumed in HTML but in XML it can only be used if the ISOLat1 entity list is included
- Character entities correspond to Unicode characters.
  - Example: &#23; refers to decimal character number 23 A refers to hex character number 41
- Predefined escape sequence entities:
  - <(<), > (>), ' ('), " ("), &(&)

Byte Order Marker

- The Byte Order Marker is an optional code at the very beginning of the file primarily to indicate endianness of UTF-16.
  - FF FE = little endian
    - Unicode "0102 0304" stored as "02 01 04 03"
  - FE FF = big endian
    - Unicode "0102 0304" stored as "01 02 03 04"
- Since it is the first code, it also suggests the base encoding (to be used in conjunction with the more specific encoding attribute).
  - EF BB BF = UTF-8
  - FF FE 00 00 = UCS-4, little endian
- It is usually possible to use heuristics to determine encodings and endianness automatically from the first 4 bytes.

Exercise 3a: XML to store data

- Encode the following relational data in XML:

title	Markup & XML
date	2006
users	

name	machine
vusi	12
john	24
nithia	36

Exercise 3b: Handwritten XML

- Open a text editor and type in your XML document.
  - Start with an XML declaration!
  - Leave out the BOM and use UTF-8 encoding.
- Save the file in the exercise folder with a ".xml" extension.
- Open the file in Firefox to make sure it loads properly and is well-formed.

Namespaces

XML Namespaces

- Namespaces are used to partition XML elements into well-defined subsets to prevent name clashes.
- If two XML DTDs define the tag "title", which one is implied when the tag is taken out of its document context (e.g., during parsing)?
- Namespaces disambiguate the intended semantics of XML elements.

Default Namespaces

- Every element has a default namespace if none is specified.
- The default namespace for an element and all its children is defined with the special "xmlns" attribute on an element.
  - Example: `<uct xmlns="http://www.uct.ac.za">`
- Namespaces are URIs, thus maintaining uniqueness in terms of a specific scheme.

Universal Resource Locator (URL) = location-specific
Universal Resource Name (URN) = location-independent
Universal Resource Identifier (URI) = generic identifier

Explicit Namespaces

- Multiple active namespaces can be defined by using prefixes. Each namespace is declared with the attribute "xmlns:*ns*", where *ns* is the prefix to be associated with the namespace.
- The containing element and its children may then use this prefix to specify membership of namespaces other than the default.
-
```
<uct xmlns="http://www.uct.ac.za"
     xmlns:dc="http://somedcns">
   <dc:title>test XML document</dc:title>
</uct>
```

Can you rewrite the last example?

- For example
  -
```
<uct:uct xmlns:uct="http://www.uct.ac.za">
   <dc:title xmlns:dc="http://somedcns">test XML
document</dc:title>
</uct:uct>
```

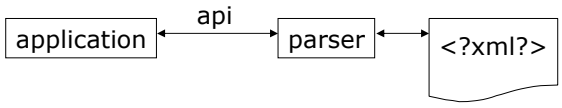
Exercise 4: Namespaces

- Edit your XML file from Exercise 3 to include namespaces as follows:
  - title and date are in the namespace `http://purl.org/dc/elements/1.1/`
  - all other date is in the namespace `http://www.cs.uct.ac.za/XMLworkshop/`
- Minimise the size of your XML by using only one definition of each namespace and shorter prefixes thereafter.
- Make sure your XML file loads into Firefox.

XML Parsing

Parsing XML

- XML parsers expose the structure as well as the content to applications, as opposed to regular file input where applications get only content or linear structure.
- Applications are written to manipulate XML documents using APIs exposed by parsers.



- Two popular APIs:
  - Simple API for XML (SAX)
  - Document Object Model (DOM)

XML, SAX, DOM … is everything a TLA?

SAX

- Simple API for XML (SAX) is event-based and uses callback routines or event handlers to process different parts of XML documents.
- To use SAX:
  - Register handlers for different events
  - Parse document
- Textual data, tag names and attributes are passed as parameters to the event handlers.

SAX Example

- Using handlers to output the content of each node, the following output can be trivially generated:
  - start document
  - start tag : uct
  - start tag : title
  - content : test XML document
  - end tag : title
  - start tag : author
  - content : Pat Pukram
  - end tag : author
  - start tag : version
  - start tag : number
  - content : 1.0
  - end tag : number
  - end tag : version
  - end tag : uct
  - end document

What happened to the attributes?

pseudo-code:

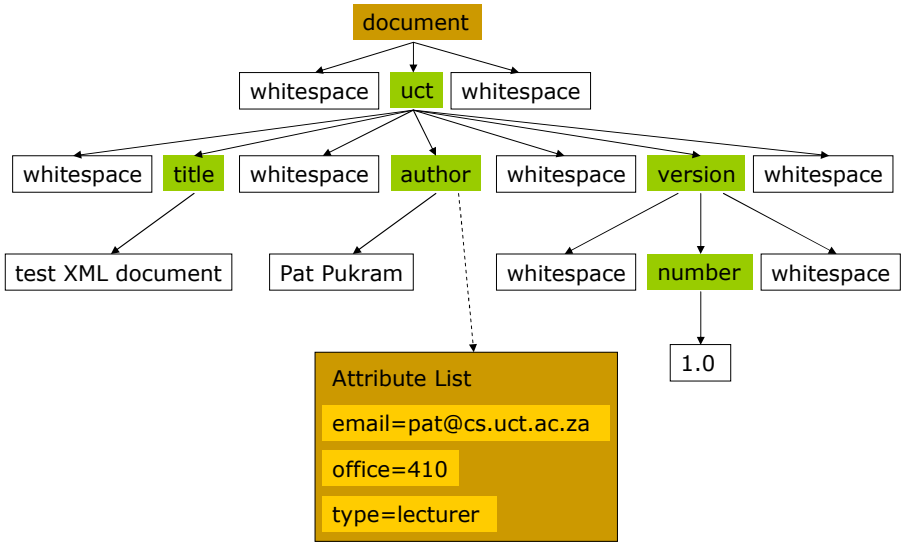
```
startCallback
{ output "start tag: ",tag }
…
main_program
{
  register_starthandler (startCallback)
  …
  do_parse
}
```

DOM

- Document Object Model (DOM) defines a standard interface to access specific parts of the XML document, based on a tree-structured model of the data.
- Each node of the XML is considered to be an object with methods that may be invoked on it to set/retrieve its contents/structure or navigate through the tree.
- DOM v1 and v2 are W3C standards. DOM3 is a (newer) standard as of April 2004.

W3C?

DOM Tree



DOM Example

- Step-by-step parsing

```
# create instance of parser
my $parser = new DOMParser;
# parse document
my $document = $parser->parsefile ('uct.xml');
# get node of root tag
my $root = $document->getDocumentElement;
# get list of title elements
my $title = $document->getElementsByTagName ('title');
# get first item in list
my $firsttitle = $title->item(0);
# get first child - text content
my $text = $firsttitle->getFirstChild;
# print actual text
print $text->getData;
```

Perl is popular for its text-processing capabilities.
Java is popular because of its libraries and servlet support.

- Quick-and-dirty approach

```
my $parser = new DOMParser;
my $document = $parser->parsefile ('uct.xml');
print $document->getDocumentElement->getElementsByTagName
('title')->item(0)->getFirstChild->getData;
```

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

DOM Bindings

- DOM has different bindings (correspondence between abstract API and language-specific use) in different languages.
- Each binding must cater for how the document is parsed – this is not part of DOM.
- In general, method names and parameters are consistent across bindings.
- Some bindings define extensions to the DOM e.g., to serialise an XML tree.

SAX vs. DOM

- DOM is a W3C standard while SAX is a community-based "standard".
- DOM is defined in terms of a language-independent interface while SAX is specified for each implementation language (with Java being the reference).
- DOM requires reading in the whole document to create an internal tree structure while SAX can process data as it is parsed.
- In general, DOM uses more memory to provide random access.

there is another … actually, others

XML Schema

XML Schema

- XML Schema specifies the type of an XML document in terms of its structure and the data types of individuals nodes.
- It replaces DTDs – it can express everything a DTD can express plus more.
- Other similar languages are RELAX and Schematron, but XML Schema is a W3C standard so has more support.

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>

Full Schema 1/2

- <schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.uct.ac.za"
  xmlns:uct="http://www.uct.ac.za"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  >

  <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>

  <complexType name="versionType">
  <sequence>
  <element name="number">
  <simpleType>
  <restriction base="string">
  <pattern value="[1-9]\.[0-9]+"/>
  </restriction>
  </simpleType>
  </element>
  </sequence>
  </complexType>

Full Schema 2/2

```
<complexType name="uctType">
   <sequence>
      <element name="title" type="string"/>
      <element name="author" type="uct:authorType"/>
      <element name="version" type="uct:versionType"/>
   </sequence>
</complexType>

<element name="uct" type="uct:uctType"/>

</schema>
```

Binding XML Instances to Schemata

- In order to specify the XML Schema for a particular XML document, use the `schemaLocation` attribute in the root tag (and elsewhere if necessary).
- `schemaLocation` contains a space-separated list of pairs of namespaces and the associated URLs of XML Schema definitions.
  - `schemaLocation="namespace schemaURL"`
- `schemaLocation` is defined in the W3C's XMLSchema-instance namespace so this must be defined as well.
  - `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
    `xsi:schemaLocation="namespace schemaURL"`

Qualified Valid XML

```
<uct xmlns="http://www.uct.ac.za"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.uct.ac.za
       uct.xsd"
>

   <title>test XML document</title>
   <author email="pat@cs.uct.ac.za"
           office="410"
           type="lecturer">Pat Pukram</author>
   <version>
      <number>1.0</number>
   </version>

</uct>
```

cool trick: use one of Xerces's sample programs, like dom.Counter with a "-v" parameter, to do Schema validation!

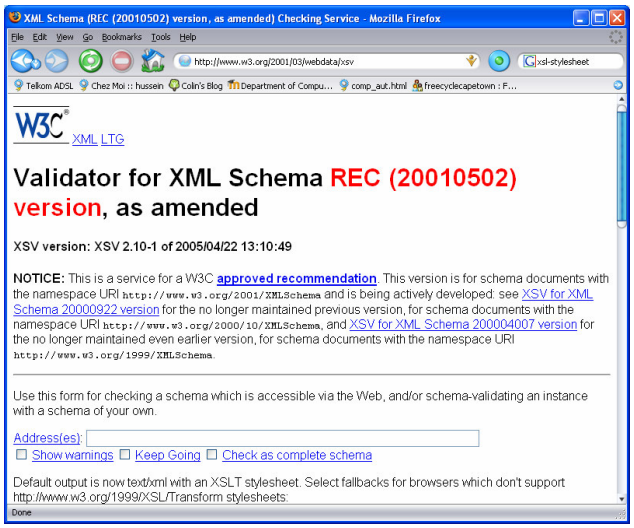
Validating XML (using Schema)

- Using an online service
  - http://www.w3.org/2001/03/webdata/xsv
- Running validator from command-line

```
#!/bin/sh
export CLASSPATH=/usr/local/share/xerces-
  2_4_0/xmlParserAPIs.jar:/usr/local/share/xerces
  -2_4_0/xercesImpl.jar:/usr/local/share/xerces-
  2_4_0/xercesSamples.jar
/usr/local/jdk1.4.2/bin/java -
  DproxyHost=cache.uct.ac.za -DproxyPort=8080
  dom.Counter -s -v -f -p dom.wrappers.Xerces $1
```

- Embedding validator in program
  - Parse the document with a validation switch turned on – validation is a core part of the parser (e.g., Xerces).

W3C Schema Validator



Exercise 5a: XML Schema Validation

- Open a Command Prompt window (usually from Accessories on WinXP).
- Change directory to the exercise folder.
- Type the command (on one line):
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct1.xml`
  - Output should be:
    - `[Error] uct1.xml:1:6: cvc-elt.1: Cannot find the declaration of element 'uct'.`
      `uct1.xml: 731;40;0 ms (5 elems, 3 attrs, 0 spaces, 56 chars)`
  - This is because the validator cannot find a schema.
- Note that the second line prints statistics on the XML since that is the function of dom.Counter – this is not part of the validation.

Exercise 5b: XML Schema Validation

- Type the command:
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct2.xml`
  - Output should be:
    - `[Error] uct2.xml:1:35: cvc-elt.1: Cannot find the declaration of element 'uct'.`
      `uct2.xml: 731;30;0 ms (5 elems, 4 attrs, 0 spaces, 56 chars)`
  - Now, even though there is a namespace, there is still no schema declared.

Exercise 5c: XML Schema Validation

- Type the command:
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct3.xml`
  - Output should be:
    - `uct3.xml: 821;30;0 ms (5 elems, 6 attrs, 0 spaces, 56 chars)`
  - This time no errors are reported because the XML is well-formed, valid and connected to its Schema using the right namespace and Schema URL.

Exercise 5d: XML Schema Validation

- Type the command:
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct_error1.xml`
  - Output should be:
    - `[Error] uct_error1.xml:1:6: cvc-elt.1: Cannot find the declaration of element 'uct'.`
      `[Fatal Error] uct_error1.xml:7:6: The element type "number" must be terminated by the matching end-tag "</number>".`
  - The first error occurs because there is no namespace and schemaLocation.
  - The second error is fatal because the XML is not well-formed!

Exercise 5e: XML Schema Validation

- Type the command:
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct_error2.xml`
  - Output should be:
    - `[Error] uct_error2.xml:11:14: cvc-complex-type.2.4.d: Invalid content was found starting with element 'abstract'. No child element is expected at this point.`
      `uct_error2.xml: 911;40;0 ms (6 elems, 6 attrs, 0 spaces, 63 chars)`
  - The XML is invalid because "abstract" is not defined in the schema.

Exercise 5f: XML Schema Validation

- Type the command:
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct_error3.xml`
  - Output should be:
    - `[Error] uct_error3.xml:6:66: cvc-complex-type.2.4.a: Invalid content was found starting with element 'author'. One of '{"http://www.uct.ac.za":title}' is expected. uct_error3.xml: 891;40;0 ms (4 elems, 6 attrs, 0 spaces, 35 chars)`
  - The XML is invalid because the title element is required but is missing.

Exercise 5g: XML Schema Validation

- Type the command:
  - `java -classpath xercesImpl.jar;xercesSamples.jar dom.Counter -v -s -f uct_error4.xml`
  - Output should be:
    - `[Error] uct_error4.xml:7:11: cvc-complex-type.2.4.a: Invalid content was found starting with element 'title'. One of '{"http://www.uct.ac.za":author}' is expected. uct_error4.xml: 901;30;0 ms (6 elems, 6 attrs, 0 spaces, 73 chars)`
  - The XML is invalid because there is a second title and only one is defined in the schema.

XPath

XPath

- XML Path Language (XPath) is a language to address particular nodes or sets of nodes of an XML document.
- Using XPath expressions we can write precise expressions to select nodes without procedural DOM statements.
- Examples:
  - uct/title
  - uct/version/number
  - uct/author/@office

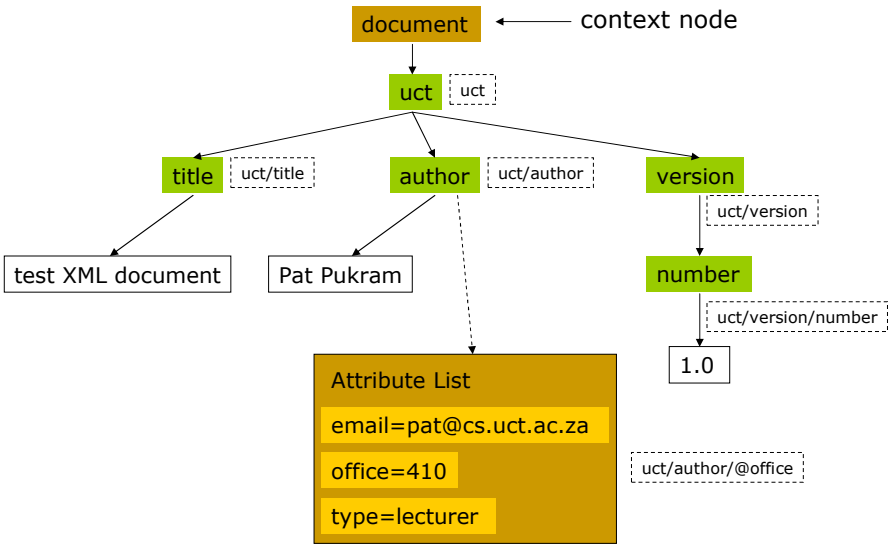
XPath Syntax

- 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, descendents, siblings, etc.
- Shorthand notation uses symbols for the possible axes.

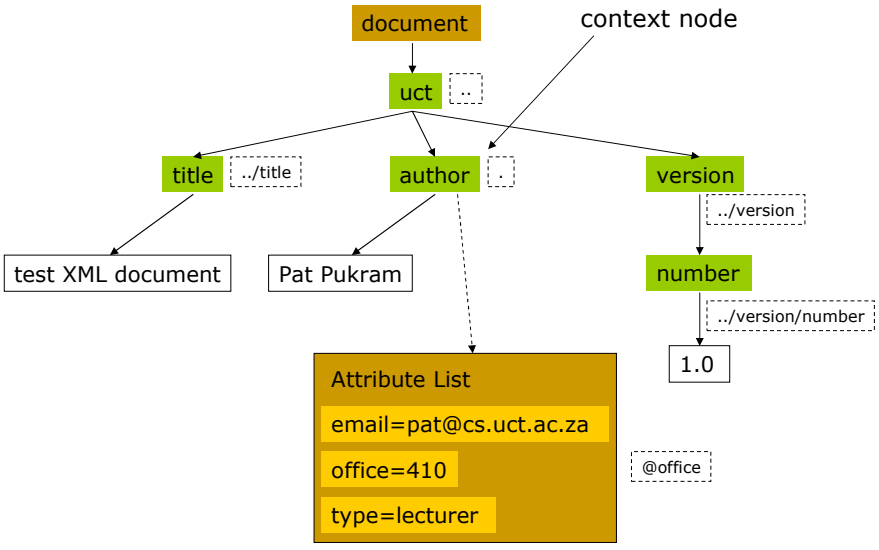
XPath Shorthand

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"

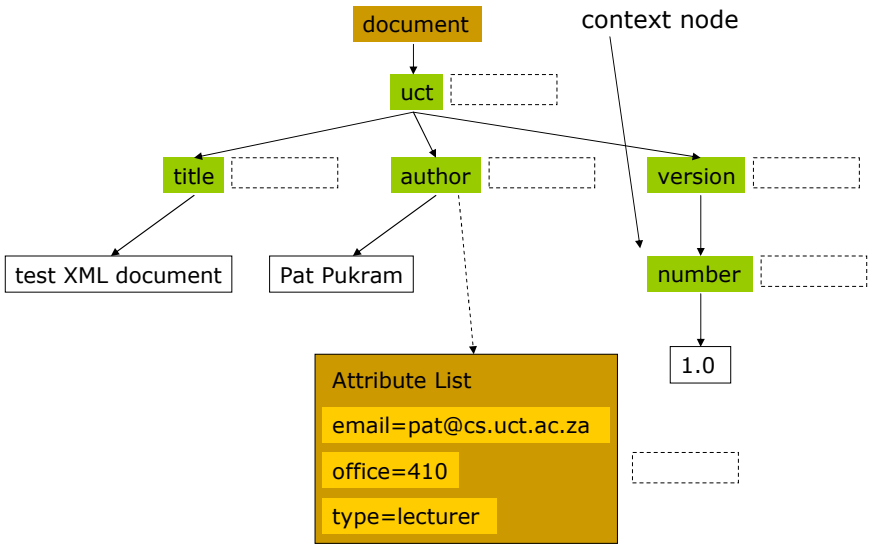
XPath Example 1



XPath Example 2



XPath Exercise



XSL - XSLT

XSL

- XML Stylesheet Language (XSL) is used to convert structured data in XML to a "human-friendly" representation.
- 2-step process:
  - Transform XML data (XSLT)
  - Process formatting instructions and generate output (XSL-FO)

Philosophically, besides programmers, nobody should ever have to read/write XML!

- In systems that are WWW-based, the first step is more useful – XSL Transformations (XSLT) – as XHTML is directly "processed" by browsers.

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 Basic Idea

source XML

```
<uct xmlns="http://www.uct.ac.za">
   <title>test XML document</title>
</uct>
```

XSLT

```
<xsl:stylesheet version='1.0'
 xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
 xmlns:uct='http://www.uct.ac.za'
 xmlns:uwc='http://www.uwc.ac.za'
>
   <xsl:template match="uct:uct">
      <uwc:uwc>
         <uwc:title>test XML document</uwc:title>
      </uwc:uwc>
   </xsl:template>
</xsl:stylesheet>
```

transformed XML

```
<?xml version="1.0"?>
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
  <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Applying XSLT Transformations

- Running processor from command-line
  - xsltproc uct.xsl uct.xml
- Running processor from within browser (static page)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="uct.xsl"?>
```

- Embedding processor in program

```
var processor = new XSLTProcessor ();
var dataXML =
  document.implementation.createDocument("", "", null);
dataXML.async = false;
dataXML.load("uct.xml");
var dataXSL =
  document.implementation.createDocument("", "", null);
dataXSL.async = false;
dataXSL.load('uct.xsl');
processor.reset();
processor.importStylesheet(dataXSL);
```

XSLT Templates

- Templates of replacement XML are specified along with criteria for matching in terms of XPath expressions.
- XSLT processors attempt to match the root XML tag with a template. If this fails they descend one level and try to match each of the root's children, etc.
- In the previous example, all occurrences of the "uct:uct" tag will be replaced by the contents of the template.

XSLT Special Tags

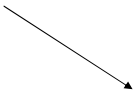
- Special tags in the XSL namespace are used to control transformation.

- value-of, text, element
  - Create nodes in result document.
- apply-templates, call-template
  - Apply template rules explicitly.
- variable, param, with-param
  - Local variables and parameter passing.
- if, choose, for-each
  - Procedural language constructs.

Creating Element nodes

- *element* is replaced by an XML element with the indicated tag.
  - Example:
    - `<element name="dc:publisher">UCT</element>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <xsl:element name="uwc:title">
      </xsl:element>
   </uwc:uwc>
</xsl:template>
```

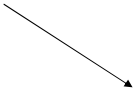


```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title/>
</uwc:uwc>
```

Creating Text nodes

- *text* is replaced by the textual content.
  - Example:
    - `<text>1.0</text>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <xsl:element name="uwc:title">
         <xsl:text>test XML document</xsl:text>
      </xsl:element>
   </uwc:uwc>
</xsl:template>
```

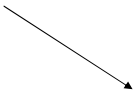


```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Element and Text Shorthand

- Elements and text nodes can usually be included directly in templates.
  - Example
    - Instead of `<element name="xxx"/>`
    - Use `<xxx/>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <uwc:title>
         test XML document
      </uwc:title>
   </uwc:uwc>
</xsl:template>
```

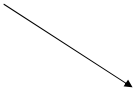


```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Copying values across

- *value-of* is replaced with the textual content of the nodes identified by the XPath expression.
  - Example:
    - `<value-of select="uct:title"/>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <uwc:title>
         <xsl:value-of select="uct:title"/>
      </uwc:title>
   </uwc:uwc>
</xsl:template>
```



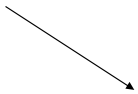
```
<uwc:uwc
     xmlns:uwc='http://www.uwc.ac.za'
     xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
</uwc:uwc>
```

Applying Templates Explicitly

- *apply-templates* explicitly and recursively applies templates to the specified nodes.
  - Example:
    - `<apply-templates select="uct:version"/>`

```
<xsl:template match="uct:uct">
   <uwc:uwc>
      <uwc:title>
         <xsl:value-of select="uct:title"/>
      </uwc:title>
      <xsl:apply-templates select="uct:author"/>
   </uwc:uwc>
</xsl:template>

<xsl:template match="uct:author">
   <uwc:author>
      <xsl:value-of select="."/>
   </uwc:author>
</xsl:template>
```



```
<uwc:uwc
      xmlns:uwc='http://www.uwc.ac.za'
      xmlns:uct='http://www.uct.ac.za'
>
   <uwc:title>test XML document</uwc:title>
   <uwc:author>Pat Pukram</uwc:author>
</uwc:uwc>
```

Calling Templates

- *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>
```

Variables

- *variable* sets a local variable in a template or globally. In XPath expressions, a $ prefix indicates a variable or parameter instead of a node.
  - Example:
    - `<variable name="institution">UCT</variable>`
      `<value-of select="$institution"/>`
- Expressions also can be inserted into attributes in generated nodes. Surround the expression with { and } to tell XSLT it is not a literal string.
  - Example:
    - `<place institution="{$institution}"/>`

Procedural Constructs

- Generate a tree of nodes if a condition holds.
  - `<if test="position()=last()">…</if>`
- Generate a different tree of nodes, depending on which of a number of a conditions holds.
  - ```
    <choose>
       <when test="$val=1">…</when>
       <otherwise>…</otherwise>
    </choose>
    ```
- Iterate over a set of nodes matching an expression and generate a tree for each. This has the same effect as *apply-templates*.
  - `<for-each select="uct:number">…</for-each>`
    ```

Full XSLT 1/2

```
<xsl:stylesheet version='1.0'
 xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
 xmlns:uct='http://www.uct.ac.za'
 xmlns='http://www.w3.org/1999/xhtml'
 exclude-result-prefixes='xsi uct'
>

<!--
   UCT to HTML transformation
   Hussein Suleman
   v1.0 : 10 May 2007
-->

   <xsl:output method="xml" omit-xml-declaration="yes"
      omit-namespace="html"/>

   <xsl:variable name="institution">
      <xsl:text>UCT</xsl:text>
   </xsl:variable>
```

Full XSLT 2/2

```
    <xsl:template match="uct:uct">
      <html>
         <head>
            <title>UCT Information Page</title>
         </head>
         <body>
            <h1><xsl:value-of select="uct:title"/></h1>
            <hr/>
            <xsl:apply-templates select="uct:author"/>
            <h2>Publisher</h2><p><xsl:value-of select="$institution"/></p>
            <xsl:apply-templates select="uct:version"/>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="uct:author">
      <h2>Author</h2>
      <p><xsl:value-of select="."/></p>
   </xsl:template>

   <xsl:template match="uct:version">
      <h2>Version</h2>
      <p><xsl:value-of select="uct:number"/></p>
   </xsl:template>

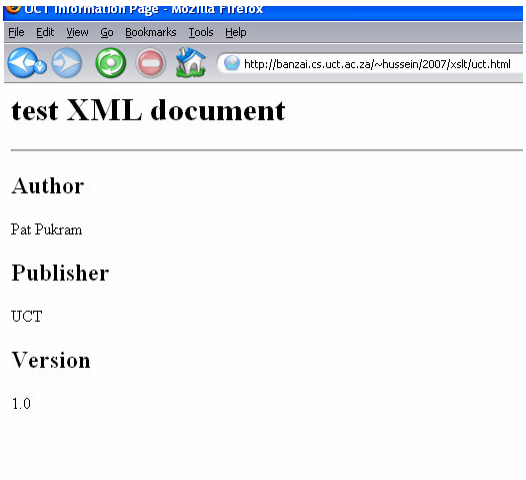
</xsl:stylesheet>
```

note: this is not the simplest XSLT for this problem

Transformed XML (XHTML Source)

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
    <title>UCT Information Page</title>
  </head>
  <body>
     <h1>test XML document</h1>
     <hr/>
     <h2>Author</h2>
     <p>Pat Pukram</p>
     <h2>Publisher</h2>
     <p>UCT</p>
     <h2>Version</h2>1.0</body>
</html>
```

XHTML Rendered



Exercise 6: XSLT

- View the uct.xsl stylesheet in your browser.
- In the workshop folder, copy uct3.xml to uct4.xml.
- Edit uct4.xml and add the following line just below the XML declaration (or as the top line if there is no declaration).
  - `<?xml-stylesheet type="text/xsl" href="uct.xsl"?>`
- View the uct4.xml file in your browser.
  - It should appear in its transformed state (as HTML).
  - View source to see the original file.
  - Note that this is XML→XHTML (because the end result is to view in a browser) but you do not always do this…

XSL - FO

XSL Formatting Objects

- XSL-FO is a language to specify the layout of elements on pages.
- Page masters (templates) are first defined and then content is flowed onto the pages.
  - Formatting attributes are similar to CSS!
- XSLT is typically used to convert XML into XSL-FO, then an FO processor (such as Apache FOP) converts the FO into a document format (such as PDF).

Example XSL-FO

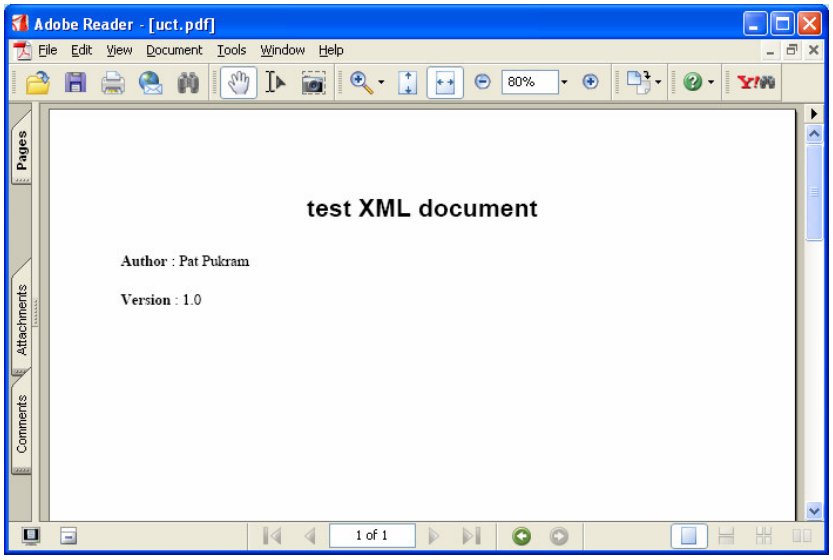
```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

<fo:layout-master-set>
   <fo:simple-page-master margin-right="1cm" margin-left="1cm" margin-top="1cm" margin-bottom="1cm" page-
width="210mm" page-height="297mm" master-name="first">
      <fo:region-after extent="1cm"/>
      <fo:region-body margin-top="1cm" margin-bottom="2cm" margin-left="1cm" margin-right="1cm"/>
   </fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="first">
   <fo:flow flow-name="xsl-region-body">
      <fo:block margin="0" padding="12px 0 12px 0" font-weight="bold" text-align="center" font-
size="20pt" font-family="sans-serif">test XML document</fo:block>
      <fo:block margin="0" padding="12px 0 6px 0" font-size="12pt" font-family="serif"><fo:inline font-
weight="bold">Author</fo:inline> : Pat Pukram</fo:block>
      <fo:block margin="0" padding="12px 0 6px 0" font-size="12pt" font-family="serif"><fo:inline font-
weight="bold">Version</fo:inline> :
      1.0</fo:block>
   </fo:flow>
</fo:page-sequence>

</fo:root>
```

XSL-FO→PDF Output



Example XSLT (XSL-FO) 1/3

```
<!--
   XSL FOP stylesheet to convert the UCT metadata record
   into
   FO suitable for FOP to convert into a PDF

   Hussein Suleman
   1 August 2005
-->

<xsl:stylesheet
 version='1.0'
 xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
 xmlns:source='http://www.uct.ac.za'
 xmlns:fo='http://www.w3.org/1999/XSL/Format'
 xmlns:html='http://www.w3.org/1999/xhtml'
>

<xsl:output method="xml" omit-xml-declaration="yes"/>
```

Example XSLT (XSL-FO) 2/3

```
<xsl:template match="source:uct">
   <fo:root>
      <fo:layout-master-set>
         <fo:simple-page-master margin-right="1cm"
                                margin-left="1cm"
                                margin-top="1cm"
                                margin-bottom="1cm"
                                page-width="210mm"
                                page-height="297mm"
                                master-name="first">
            <fo:region-after extent="1cm"/>
            <fo:region-body margin-top="1cm" margin-bottom="2cm"
                            margin-left="1cm" margin-right="1cm"/>
         </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-reference="first">
         <fo:flow flow-name="xsl-region-body">
            <xsl:apply-templates select="*"/>
         </fo:flow>
      </fo:page-sequence>
   </fo:root>
</xsl:template>
```

Example XSLT (XSL-FO) 3/3

```
<xsl:template match="source:title">
   <fo:block margin="0" padding="12px 0 12px 0" font-weight="bold"
             text-align="center" font-size="20pt" font-family="sans-serif">
      <xsl:value-of select="."/>
   </fo:block>
</xsl:template>

<xsl:template match="source:author">
   <fo:block margin="0" padding="12px 0 6px 0"
             font-size="12pt" font-family="serif">
      <fo:inline font-weight="bold">Author</fo:inline> :
      <xsl:value-of select="."/>
   </fo:block>
</xsl:template>

<xsl:template match="source:version">
   <fo:block margin="0" padding="12px 0 6px 0"
             font-size="12pt" font-family="serif">
      <fo:inline font-weight="bold">Version</fo:inline> :
      <xsl:value-of select="source:number"/>
   </fo:block>
</xsl:template>

</xsl:stylesheet>
```

XQuery

XQuery

- XQuery specifies advanced functional queries over XML documents and collections.
- XQuery is a superset of XPath 1.0, and parallel specification for XPath 2.0 and XSLT 2.0.

XQuery Expressions 1/2

- Primary expressions
  - `12.1, "Hello world"` (literals)
  - `$firstauthor` (variable)
  - `xq:string-concat ()` (function call)
- Path expressions
  - `document("test.xml")//author`
  - `para[5][@type="warning"]`
  - `child::chapter[child::title='Intro']`

XQuery Expressions 2/2

- Arithmetic/Comparison/Logic expressions
  - `$unit-price - $unit-discount`
  - `//product[weight gt 100]`
  - `1 eq 1 and 2 eq 2`
- Sequence expressions
  - `(1, 2, (3))`
  - `(10, 1 to 4)`
  - `(1 to 100)[. mod 5 eq 0]`
  - `$seq1 union $seq2`

FLWOR Expressions

- For-Let-Where-OrderBy-Return
- Iterates over a sequence of nodes, with intermediate binding of variables.
- Most useful for database-like "join" operations.

FLWOR Example

```
for $d in fn:doc("depts.xml")//deptno
let $e := fn:doc("emps.xml")//emp[deptno = $d]
where fn:count($e) >= 10
order by fn:avg($e/salary) descending
return
   <big-dept>
      {
      $d,
      <headcount>{fn:count($e)}</headcount>,
      <avgsal>{fn:avg($e/salary)}</avgsal>
      }
   </big-dept>
```

(from specification)

FLWOR For, Let

- `for` and `let` create a sequence of tuples with bound variables.
- Can have multiple `for`s and multiple `let`s.
- Multiple `for`s result in a Cartesian product of the sequences.
  - ```
    for $car in ("Ford", "Chevy"),
        $pet in ("Cat", "Dog")
    ```
- Multiple `let`s result in multiple intermediate variable bindings per tuple of nodes.

FLWOR Where, OrderBy, Return

- `where` filters the list of tuples, by removing those that do not satisfy the expression.
- `return` specifies result for each tuple.
- `order by` specifies the expression to use to order the tuples – the expression can use nodes not included in the result.
  - ```
    for $e in $employees
    order by $e/salary descending
    return $e/name
    ```

FLWOR for DB Joins

```
<ucthons>
   {
      for $stud in fn:doc("students.xml")//student
      for $proj in
        fn:doc("projects.xml")//project[id = $stud/id]
      order by $stud/name
      return
         <honsproj>
           <studentname>{$stud/name}</studentname>
           <projectname>{$proj/name}</projectname>
         </honsproj>
   }
</ucthons>
```

XML Databases

XML Databases

- Databases must be Unicode-compliant! (usually UTF-8)
- Options:
  - Blob: Store XML documents or fragments in tables.
  - Tree: Store XML as sequence of nodes with child relationships explicitly indicated.
  - Relation: Store XML in specialised tables/relations as defined by XML structure.
  - Flat files: Store each XML document in a file.

Blob/Clob/etc.

Id	XMLBlob
TestXML	<uct> <title>test XML document</title> <author email="pat@cs.uct.ac.za" office="410" type="lecturer">Pat Pukram</author> <version> <number>1.0</number> </version> </uct>

Tree Representation

Nodes

Id	Type	Label	Value
1	Element		uct
2	Element		title
3	Text		test XML document
4	Element		author
5	Attribute	email	pat@cs.uct.ac.za
6	Attribute	office	410
7	Attribute	type	lecturer
8	Text		Pat Pukram
9	Element		version
10	Element		number
11	Text		1.0

Links

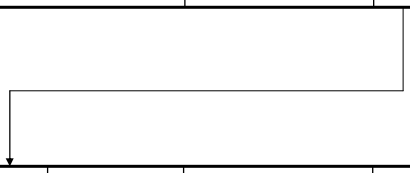
Parent id	Child id
1	2
2	3
1	4
4	5
4	6
4	7
4	8
1	9
9	10
10	11

Note: Whitespace nodes have been ignored!

Relation Representation

main table

Institute	Title	VersionNumber	id
uct	test XML document	1.0	1



id	Author	Email	Office	Type
1	Pat Pukram	pat@cs.uct.ac.za	410	lecturer

author table

Evaluation

- Blob: fast insert/select for XML documents, but slow querying.
- Tree: fast location of single nodes and sequences of nodes, but slow to enforce structure of XML.
- Relation: fast data query and extraction, but could be many tables and thus slow to insert/select XML documents.
- Flat file: fast load/store, but slow queries.

Are we only interested in relational queries? Google-like queries?

that's all folks!

References 1/3

- Adler, Sharon, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman and Steve Zilles (2001) Extensible Stylesheet Language (XSL) Version 1.0, W3C. Available http://www.w3.org/TR/xsl/
- Berners-Lee, Tim, Roy Fielding and Larry Masinter (1998) Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, Network Working Group. Available http://www.ietf.org/rfc/rfc2396.txt
- Boag, Scott, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie and Jérôme Siméon (2005). XQuery 1.0: An XML Query Language, W3C Working Draft 4 April 2005, W3C. Available http://www.w3.org/TR/xquery/
- Bourret, Ronald (1999), Declaring Elements and Attributes in an XML DTD. Available http://www.rpbourret.com/xml/xmldtd.htm
- Bradley, Neil (1998) The XML Companion, Addison-Wesley.
- Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen and Eve Maler (2000) Extensible Markup Language (XML) 1.0 (Second Edition), W3C. Available http://www.w3.org/TR/REC-xml
- Clark, James (1999) XSL Transformations (XSLT) Version 1.0, W3C. Available http://www.w3.org/TR/xslt
- Clark, James (1999) Associated Style Sheets with XML Documents, W3C Recommendation. Available http://www.w3.org/TR/xml-stylesheet/

References 2/3

- Clark, James and Steve DeRose (1999) XML Path Language (XPath) Version 1.0, W3C. Available http://www.w3.org/TR/xpath
- Czyborra, Roman (1998), Unicode Transformation Formats: UTF-8 & Co. Available http://czyborra.com/utf/
- Dublin Core Metadata Initiative (2003) Dublin Core Metadata Element Set, Version 1.1: Reference Description, DCMI. Available http://dublincore.org/documents/dces/
- Fallside, David C. (editor) (2001) XML Schema Part 0: Primer, W3C. Available http://www.w3.org/TR/xmlschema-0/
- IMS Global Learning Consortium, Inc. (2001) IMS Learning Resource Meta-Data Information Model Version 1.2.1 Final Specification, http://www.imsglobal.org/metadata/imsmdv1p2p1/imsmd_infov1p2p1.html
- Lasher, R. and D. Cohen (1995) A Format for Bibliographic Records, RFC 1807, Network Working Group. Available http://www.ietf.org/rfc/rfc1807.txt
- Le Hors, Arnaud , Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, Steve Byrne (2000), Document Object Model Level 2 Core, W3C. Available http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

References 3/3

- SAX Project (2003) Quickstart. Available http://www.saxproject.org/?selected=quickstart
- Thomson, Henry S. and Richard Tobin (2005) Validator for XML Schema, W3C. Available http://www.w3.org/2001/03/webdata/xsv
- Visual Resources Association Data Standards Committee (2002) VRA Core Categories, Version 3.0. Available http://www.vraweb.org/vracore3.htm