Information Management

Markup Languages<? xml ?>

hussein suleman

uct cs 303 2004

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Is LaTeX outdated because of its markup language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.

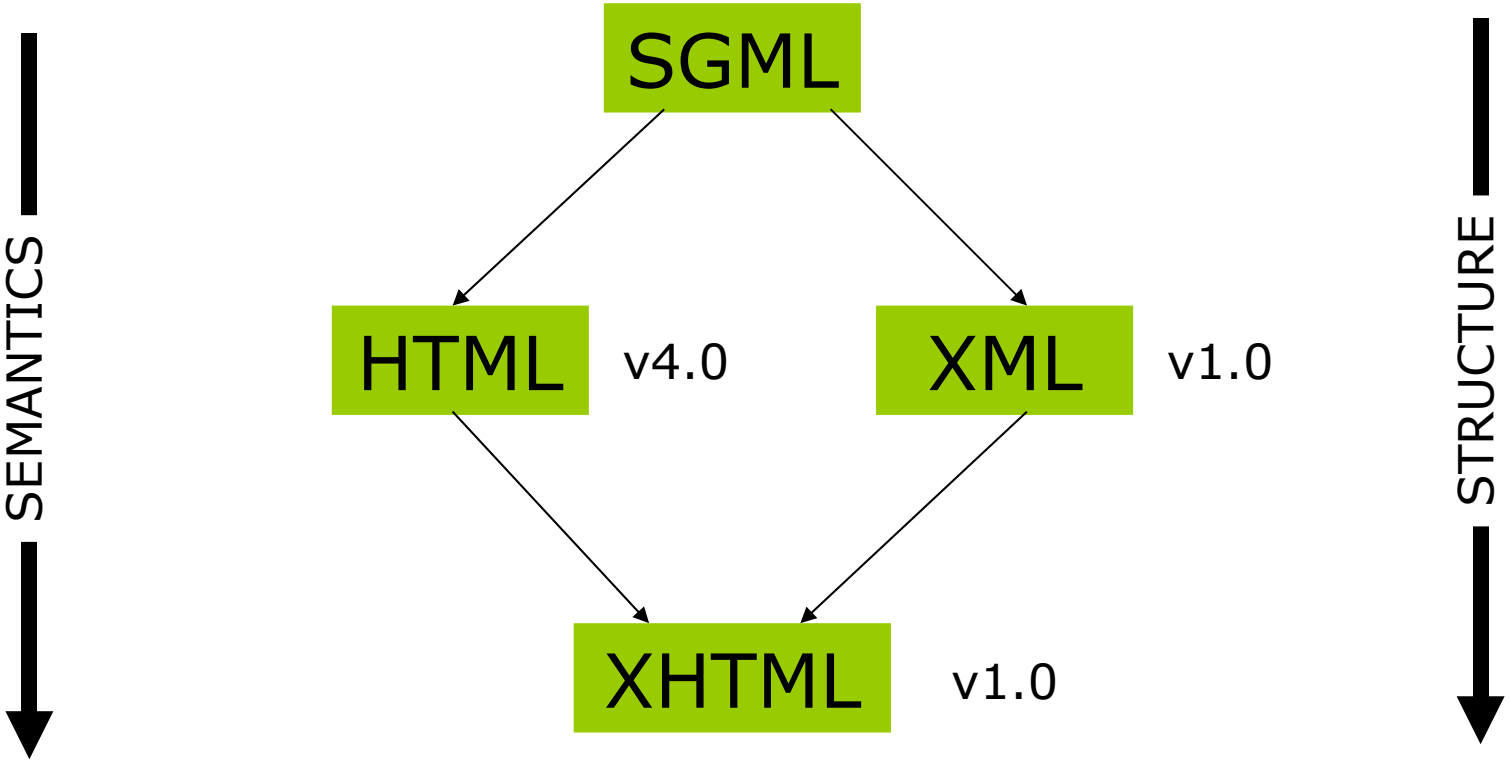
```
<html><head><title>test HTML document</title></head>
<body>
<h1>Author</h1>
<p>Pat Pukram
<br>Lecturer
<br>Email: pat@cs.uct.ac.za
<br>Office: 410
</p>
<h1>Version</h1>
<p>1.0</p>
</body>
</html>
```

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

XML declaration

```
<?xml version="1.0"?>
```

comment

```
<!-- Sample XML file -->
<!-- Hussein Suleman -->
```

DTD

```
<!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">
]>
```

root element

```
<uct>
```

start tag/element

```
<title> test &lt; XML&gt; document </title>
```

entity

end tag/element

```
<author email="pat@cs.uct.ac.za" office="410" type="lecturer" >
Pat Pukram </author>
</uct>
```

text

attribute

Validity and Well-formedness

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

- SGML parsers, in contrast, had no concept of well-formedness so domain-specific structure had to be incorporated into the parsing phase.

Why validate anyway ?

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.

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.

UTF-16

- Basic Multilingual Plane (characters in the range 0-65535) can be encoded using 16-bit words.
- Endianness is indicated by a leading Byte Order Mark (BOM) e.g., FF FE = little endian.
- 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

Ouch!

- UCS-4 → UTF-16 ?

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 <br> 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>`

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
    - I don't speak Zulu
      No hablo Zulu

Entities

- Entities begin with “&” and end with “;”.
- 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:
  - <(<), > (>), ' (‘), " (“), &(&)

Invent your own ML based on XML

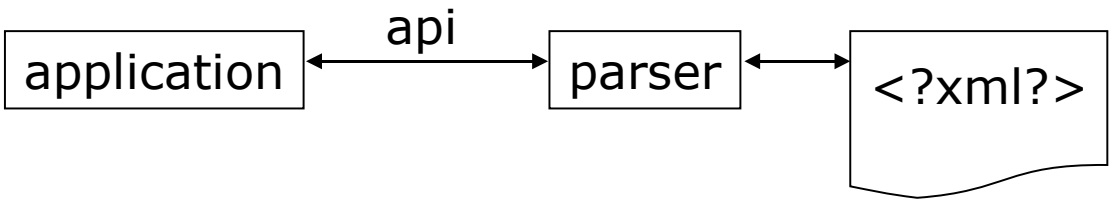
- Encode the following relational data in XML:

class	CS&614
students	3
marks	

vusi	12
john	24
nithia	36

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

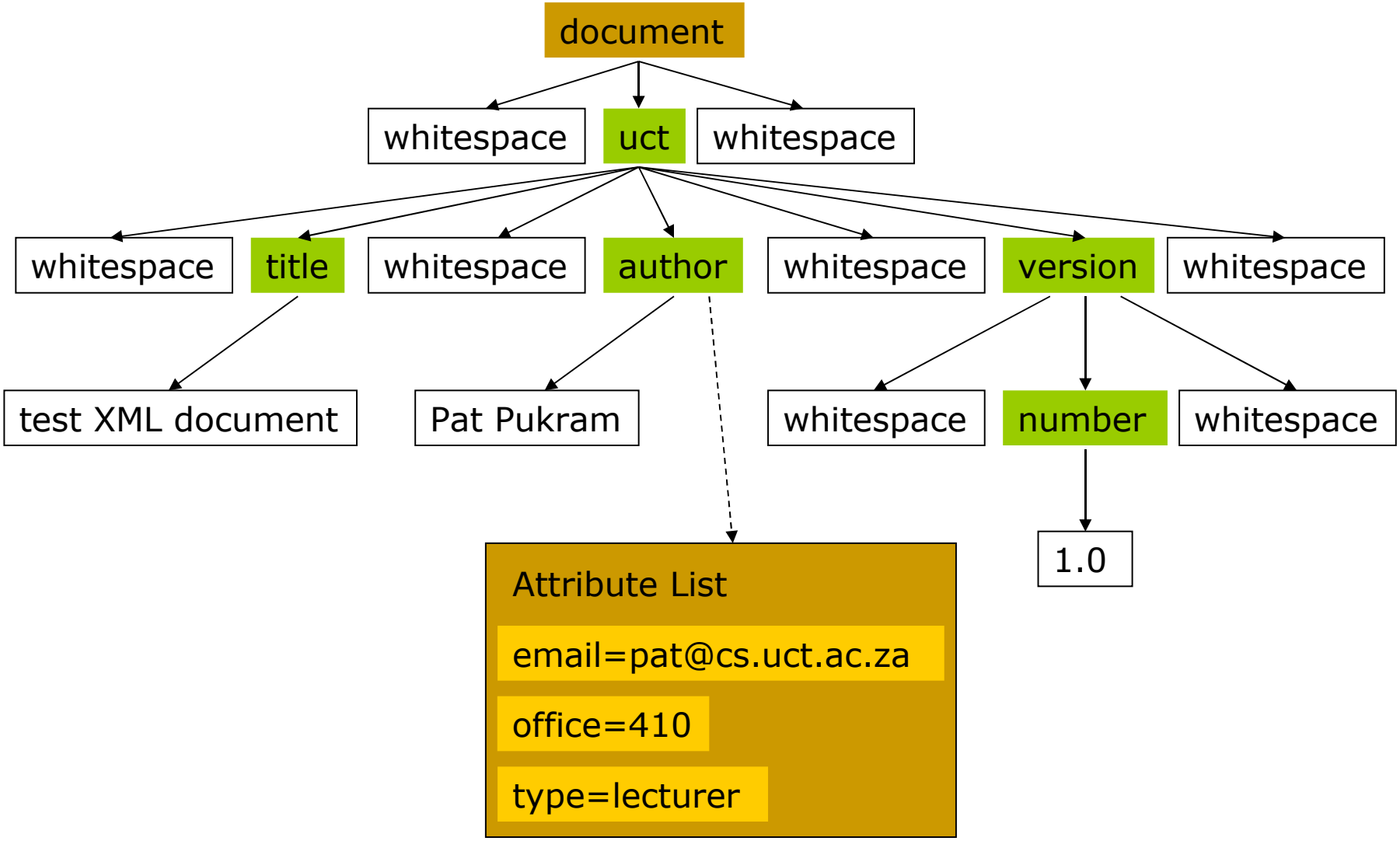
What happened to the attributes?

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 standard as of April 2004.

W3C?

DOM Tree


document
whitespace
uct
whitespace
whitespace
title
whitespace
author
whitespace
version
whitespace
test XML document
Pat Pukram
whitespace
number
whitespace
1.0
Attribute List
email=pat@cs.uct.ac.za
office=410
type=lecturer

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

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

DOM Interface subset 1/2

- Document
  - attributes
    - documentElement
  - methods
    - createElement, createTextNode, …
- Node
  - attributes
    - nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes
  - methods
    - insertBefore, replaceChild, appendChild, hasChildNodes

DOM Interface subset 2/2

- Element
  - methods
    - getAttribute, setAttribute, getElementsByTagName
- NodeList
  - attributes
    - length
  - methods
    - item
- CharacterData
  - attributes
    - data

DOM Bindings

- DOM has different bindings 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 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>

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>
```

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
       http://www.husseinsspace/teaching/uct/2003/csc400dl/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!

Data and Metadata

- Data refers to digital objects that contain useful information for information seekers.
- Metadata refers to descriptions of objects.
- Many systems manipulate metadata records, which contain pointers to the actual data.
- The definition is fuzzy as metadata contains useful information as well and in some cases could contain all the data e.g., metadata describing a person.

Metadata Standards

- To promote interoperability among systems, we use popular metadata standards to describe objects (both semantically and syntactically).
  - Dublin Core
    - 15 simple elements to describe anything.
  - MARC
    - Comprehensive system devised to describe items in a (physical) library.
  - RFC1807
    - Computer science publications format.
  - IMS Metadata Specification
    - Courseware object description.
  - VRA-Core
    - Multimedia (especially image) description.
  - EAD
    - Library finding aids to locate archived items.

Why didn't the CS folks use MARC?

Dublin Core

- Dublin Core is one of the most popular and simplest metadata formats.
- 15 elements with recommended semantics.
- All elements are optional and repeatable.

Title	Creator	Subject
Description	Publisher	Contributor
Date	Type	Format
Identifier	Source	Language
Relation	Coverage	Rights

Dublin Core in XML

```
<oaidc:dc xmlns="http://purl.org/dc/elements/1.1/"
   xmlns:oaidc="http://www.openarchives.org/OAI/2.0/oai_dc/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
   http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
   <title>02uct1</title>
   <creator>Hussein Suleman</creator>
   <subject>Visit to UCT </subject>
   <description>the view that greets you as you emerge from the tunnel
   under the freeway - WOW - and, no, the mountain isnt that close - it
   just looks that way in 2-D</description>
   <publisher>Hussein Suleman</publisher>
   <date>2002-11-27</date>
   <type>image</type>
   <format>image/jpeg</format>
   <identifier>http://www.husseinsspace.com/pictures/200230uct/02uct1.jpg
   </identifier>
   <language>en-us</language>
   <relation>http://www.husseinsspace.com</relation>
   <rights>unrestricted</rights>
</oaidc:dc>
```

Why is there a separate namespace for the root element?

Metadata Transformation

- Use XML parser to parse data.
- Use SAX/DOM to extract individual elements and generate new format.
- Example (to convert UCT to DC):

```
my $parser = new DOMParser;
my $document = $parser->parsefile ('uct.xml')->getDocumentElement;
foreach my $title ($document->getElementsByTagName ('title'))
{
   print "<title>".$title->getFirstChild->getData."</title>\n";
}
foreach my $author ($document->getElementsByTagName ('author'))
{
   print "<creator>".$author->getFirstChild->getData."</creator>\n";
}
print "<publisher>UCT</publisher>\n";
foreach my $version ($document->getElementsByTagName ('version'))
{
   foreach my $number ($version->getElementsByTagName ('number'))
   {
      print "<identifier>".
            $number->getFirstChild->getData."</identifier>\n";
   }
}
```

Come on, there must be an easier way!

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”

XSL

- XML Stylesheet Language (XSL) is used to convert structured data in XML to a "human-friendly" representation.
- 2-step process:
  - Transform XML data
  - Process data and stylesheet

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 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:author" tag will be replaced by the contents of the template.
- Special tags in the XSL namespace are used to perform additonal customisation.
  - Example: value-of

XSLT Special Tags

- 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.

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="$institution"/>`
- Selection and iteration examples:
  - `<if test="position()=last()">…</if>`
  -
```
<choose>
    <when test="$val=1">…</when>
    <otherwise>…</otherwise>
</choose>
```
  - `<for-each select="uct:number">…</for-each>`

Full XSLT 1/2

```
<stylesheet version='1.0'
 xmlns='http://www.w3.org/1999/XSL/Transform'
 xmlns:oaidc='http://www.openarchives.org/OAI/2.0/oai_dc/'
 xmlns:dc='http://purl.org/dc/elements/1.1/'
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
 xmlns:uct='http://www.uct.ac.za'
>

<!--
   UCT to DC transformation
   Hussein Suleman
   v1.0 : 24 July 2003
-->

   <output method="xml"/>

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

Full XSLT 2/2

```
   <template match="uct:uct">
      <oaidc:dc xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
                http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
         <dc:title><value-of select="uct:title"/></dc:title>
         <apply-templates select="uct:author"/>
         <element name="dc:publisher">
            <value-of select="$institution"/>
         </element>
         <apply-templates select="uct:version"/>
      </oaidc:dc>
   </template>

   <template match="uct:author">
      <dc:creator>
         <value-of select="."/>
      </dc:creator>
   </template>

   <template match="uct:version">
      <dc:identifier>
         <value-of select="uct:number"/>
      </dc:identifier>
   </template>

</stylesheet>
```

note: this is not the simplest XSLT for this problem

Transformed XML

```
<?xml version="1.0"?>
<oaidc:dc
   xmlns:oaidc="http://www.openarchives.org/OAI/2.0/oai_dc/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:uct="http://www.uct.ac.za"
          xsi:schemaLocation=
                "http://www.openarchives.org/OAI/2.0/oai_dc/
                 http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
  <dc:title>test XML document</dc:title>
  <dc:creator>Pat Pukram</dc:creator>
  <dc:publisher
   xmlns:dc="http://purl.org/dc/elements/1.1/">UCT</dc:publisher>
  <dc:identifier>1.0</dc:identifier>
</oaidc:dc>
```

why all the extraneous “xmlns”s?

References 1/2

- 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
- 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 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/

References 2/2

- 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.htm
- 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/
- SAX Project (2003) Quickstart. Available http://www.saxproject.org/?selected=quickstart
- Visual Resources Association Data Standards Committee (2002) VRA Core Categories, Version 3.0. Available http://www.vraweb.org/vracore3.htm