Contents

Part I

XML introduction

Chapter 1

Introduction

A useful reference for this chapter is the book by Goldfarb and Prescod [4].

1.1 What is XML?

XML, or Extensible Markup Language, is a markup language that you can use to create your own tags.

It was created by the World Wide Web Consortium (W3C) to overcome the limitations of HTML, the Hypertext Markup Language that is the basis for all Web pages. Like HTML, XML is based on SGML – Standard Generalized Markup Language.

XML was designed with the Web in mind [5].

1.2 Motivation for XML

- Widespread adoption of the world-wide web has provided a communication platform that is available to more users than any other.
- Initially the world-wide web was used to display information requested by the user.

- To improve the flexibility and quality of information transfer on the world-wide web a standardised, scalable data format was needed.
- XML is a markup language that provides a standardised data format that is suitable for transfer on the world-wide web. Its flexibility has resulted in many application areas.

1.2.1 The limitations of HTML

Consider for example: [5]

Given the success of HTML, why did the W3C create XML? To answer that question, consider this document:

```
 <p><b>Sir John Citizen</b>
<br>
Gosta Green
<br>
Birmingham, West Midlands, B4 7ET</p>
```

1.2.2 Processing HTML

Consider the task of extracting the postal code from this address. Here's an (intentionally brittle) algorithm for finding the postal code in HTML markup:

If you find a paragraph with two
 tags, the postal code is the second word after the first comma in the second break tag.

Although this algorithm works with this example, there are any number of perfectly valid addresses worldwide for which this simply wouldn't work.

With XML, you can assign some meaning to the tags in the document.

```
<address>
 <name>
   <title>Sir</title>
   <first-name>
     John
   </first-name>
   <last-name>
     Citizen
   </last-name>
 </name>
 <street>
   Gosta Green
 </street>
 <city>Birmingham</city>
 <county>West Midlands</county>
 <postal-code>
   B4 7ET
 </postal-code>
</address>
```

More importantly, it's easy for a machine to process the information as well.

You can extract the postal code from this document by simply locating the content surrounded by the <postal-code> and </postal-code> tags, technically known as the <postal-code> element.

1.2.3 How XML is changing the Web

XML simplifies data interchange.

XML enables smart code.

XML enables smart searches.

1.3 XML is a markup language

The online dictionary of computing [3] defines markup as:

> In computerised document preparation, a method of adding information to the text indicating the logical components of a document, or instructions for layout of the text on the page or other information which can be interpreted by some automatic system.

XML files are text data with internal data structuring information - a separate application is required to do anything with the data, including display of the data.

1.4 markup

XML belongs to the tradition of text processing systems, which aim to automate parts of the document creation and publishing process.

1.4.1 Markup for formatting

1.4.2 A document is data

1.4.3 Data need not ever exist in the form of a document

Data can be transferred and manipulated automatically, without ever taking the form of a document - in the sense of a structure that is to be readable to humans.

1.5 XML document rules

XML documents must be complete and correct, according to the rules of XML markup.

This can be contrasted with HTML, which is forgiving (or flexible, depending on your point of view).

XML is checked with a *parser*, a program which goes through the xml file checking that all the rules are met.

1.6 Data transfer on the world wide web

Chapter 2

XML basics

What is XML?

XML: e**X**tensible**M**arkup**L**anguage

XML is a **framework** for defining markup languages:

- N**o fixed collection of markup tags**: Define your own tags, tailored for the specific application domain
- Common set of **generic tools** for processing documents

XML is designed to:

- separate **syntax** from **semantics** to provide a common framework for structuring information (browser rendering semantics is completely defined by stylesheets);
- allow **tailor-made markup** for any imaginable application domain
- support **internationalization** (Unicode) and **platform independence**
- be the future of structured information, including **databases**

```
<recipe id="117" category="dessert">

<title>Rhubarb Cobbler</title>

<author><email>M.Harris@bbs.mhv.org</email></author>

<date>Wed, 14 Jun 99</date>

<description> Rhubarb Cobbler with bananas as sweetener. </description>

<ingredients>

   <item><amount>2 1/2 cups</amount><type>diced rhubarb</type></item>

   <item><amount>2 tablespoons</amount><type>sugar</type></item>

   <item><amount>2</amount><type>fairly ripe bananas</type></item>

   <item><amount>1/4 teaspoon</amount><type>cinnamon</type></item>

   <item><amount>dash of</amount><type>nutmeg</type></item>

</ingredients>

<preparation>
    Combine all and use as cobbler or pie.
  </preparation>

</recipe>
```

Figure 2.1: Recipe markup language

XML Example:

We define our own ”recipe markup language” where the markup tags directly correspond to concepts in the world of recipes - see Fig 2.1

This example illustrates:

- the markup tags are chosen purely for **logical structure**
- this is just one choice of markup detail level

- we need to define which XML documents we regard as ”recipe collections”
- we need a stylesheet to define browser presentation semantics
- we need to express queries in a general way

Later:

- **XML Schema** will later be used to define our class of recipe documents
- **XSLT** will be used to transform the XML document into XHTML (or HTML), including automatic construction of index, references, etc.
- **XLink**, **XPointer**, and **XPath** could be used to create cross-references
- **XQuery** will be used to express queries

2.0.1 More XML Examples:

```
<?xml version=”1.0”?>
  <dining-room>
  <manufacturer>The Wood Shop</manufacturer>
  <table type=”round” wood=”oak”>
  <price>£199.99</price>
  </table>
  <chair wood=”oak”>
  <quantity>6</quantity>
  <price>£39.99</price>
  </chair>
  </dining-room>
```

XML describes Structure and Semantics, Not Formatting

<SONG>
<TITLE>Va Pensiera</TITLE>
<COMPOSER>G Verdi</COMPOSER>
<PRODUCER>P Waterman</PRODUCER>
<PUBLISHER>Choral Music Society</PUBLISHER>
<LENGTH>4:46</LENGTH>
<YEAR>1991</YEAR>
<ARTIST>Aston Opera</ARTIST>
</SONG>

Self-Describing Data

<?xml version=”1.0” encoding=”UTF-8”?>
<DOCUMENT>
<GREETING>Hello from XML</GREETING>
<MESSAGE>Welcome to Programing XML</MESSAGE>

</DOCUMENT>

Structured and Integrated Data

<?xml version=”1.0”?> <SCHOOL>
<CLASS type=”seminar”>
<CLASS_TITLE>XML In The Real World</CLASS_TITLE>
<CLASS_NUMBER>311.8</CLASS_NUMBER>
<SUBJECT>XML</SUBJECT>
<START_DATE>6/1/2002</START_DATE>
<STUDENTS>
<STUDENT status=”attending”>
<FIRST_NAME>E</FIRST_NAME>
<LAST_NAME>Sampson</LAST_NAME>
</STUDENT>
<STUDENT status=”withdrawn”>
<FIRST_NAME>Delilah</FIRST_NAME>
<LAST_NAME>Smith</LAST_NAME>
</STUDENT>
</STUDENTS>
</CLASS>
</SCHOOL>

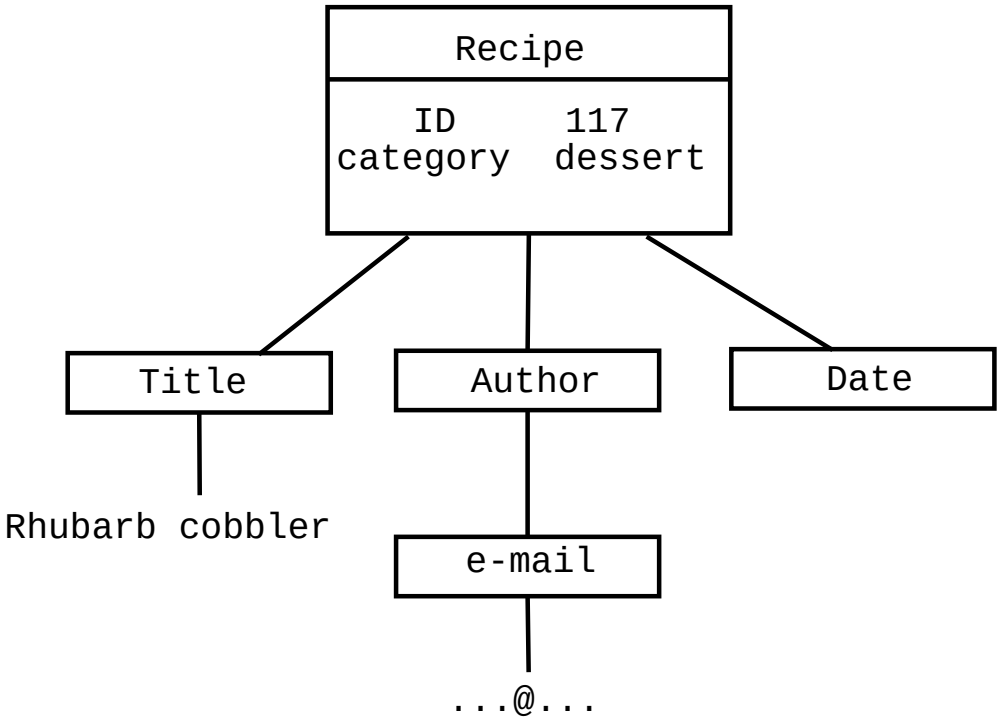

Figure 2.2: XML tree

A conceptual view of XML

An XML document is an **ordered, labeled tree**:

- **character data** leaf nodes contain the actual data (text strings)
  - usually, character data nodes must be **non-empty** and **non-adjacent to other character data nodes**
- **elements** nodes, are each labeled with
  - a name (often called the element *type*), and
  - a set of **attributes**, each consisting of a name and a value,

  and these nodes can have child nodes

2.0.2 A tree view of the XML recipe collection:

The tree structure of a document can be examined in the Explorer browser.

see rhubarb.xml

In addition, XML trees may contain other kinds of leaf nodes:

- **processing instructions** - annotations for various processors
- **comments** - as in programming languages
- **document type declaration** - described later...

... <fooattr1= ”val1” ... attrn=”valn”>... </foo>...
| | | |
| | | a matching*element end*tag
| | the *contents* of the element
| an attribute with name`attr` and value`val`, enclosed by `'` or `"`
an*element start*tag with name`foo`

A concrete view of XML

An XML document is a text with **markup tags** and other meta-information.

2.0.3 An XML document must be well-formed:

- Follow the syntax rules setup for XML by W3C in The XML 1.0 Specification [2]
- Contain one or more elements
- Root element must contain all the other elements
- start and end tags must match
- element tags must be properly nested
- + . . .

Note: XML is *case sensitive*!

Special characters can be escaped using Unicode *character references*:

Applications of XML

There are already hundreds of serious applications of XML.

XHTML

W3C's XMLization of HTML 4.0. Example XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head><title>Hello world!</title></head>
 <body><p>foobar</p></body>
</html>
```

CML Chemical Markup Language. Example CML document snippet:

```
<molecule id="METHANOL">
 <atomArray>
  <stringArray builtin="elementType">C O H H H H</stringArray>
  <floatArray builtin="x3" units="pm">
-0.748 0.558 -1.293 -1.263 -0.699 0.716
  </floatArray>
 </atomArray>
</molecule>
```

WML Wireless Markup Language for WAP services:

```
<?xml version=”1.0”?>
<wml>
<card id=”Card1” title=”Wap-UK.com”>
<p>
Hello World
</p>
</card>
</wml>
```

2.0.4 And many more:

- Mathematical Markup Language (MathML)
- Channel Definition Format (CDF)
- Synchronized Multimedia Integration Language (SMIL)
- Scalable Vector Graphics (SVG)
- XML Databases
- Office packages such as Sun’s Open Office
- MusicML
- VoxML
- . . .

Chapter 3

Defi ning document content

3.1 Overview

Need to define the elements used to represent data.

- One method is to use a Document Type Definition, or DTD.
- The other method is to use an XML Schema.

3.2 Introduction to DTD

3.2.1 Creation of Document Type Definitions

Document Type Definition: Specify Structure and Syntax of XML Document

```
<!ELEMENT DOCUMENT (CUSTOMER)*>
<!ELEMENT CUSTOMER (NAME,DATE,ORDERS)>
<!ELEMENT NAME (LAST_NAME,FIRST_NAME)>
<!ELEMENT LAST_NAME (#PCDATA)>
<!ELEMENT FIRST_NAME (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT ORDERS (ITEM)*>
<!ELEMENT ITEM (PRODUCT,NUMBER,PRICE)>
<!ELEMENT PRODUCT (#PCDATA)>
<!ELEMENT NUMBER (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
```

PCDATA Parsed character data

***** Iterations - a number or ITEMs may appear in an order

Example DTD

A DTD for the recipe collections,
`recipes.dtd`:
<!ELEMENT collection (description,recipe*)>
<!ELEMENT description ANY>
<!ELEMENT recipe (title,ingredient*,preparation,comment?,nutrition)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT ingredient (ingredient*,preparation)?>
<!ATTLIST ingredient name CDATA #REQUIRED
amount CDATA #IMPLIED
unit CDATA #IMPLIED>
<!ELEMENT preparation (step*)>
<!ELEMENT step (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT nutrition EMPTY>
<!ATTLIST nutrition protein CDATA #REQUIRED
carbohydrates CDATA #REQUIRED
fat CDATA #REQUIRED
calories CDATA #REQUIRED
alcohol CDATA #IMPLIED>
By inserting:
<!DOCTYPE collection SYSTEM ”recipes.dtd”>
in the headers of recipe collection documents, we state that they are intended to conform to `recipes.dtd`.

Alternatively, the DTD can be given locally with <!DOCTYPE collection [...]>

Mini Example of valid XML Document and DTD

```
<?xml version="1.0"?>
<!DOCTYPE BOOK [
    <!ELEMENT BOOK (P*)>
    <!ELEMENT P (#PCDATA)>
]>

<BOOK>
    <P>chapter 1 - Intro</P>
    <P>chapter 2 - Conclusion</P>
    <P>Index</P>
</BOOK>
```

Creating Document Type Declarations

External DTDs (Privates)

- File Name:

<?xml version=”1.0” standalone=”no”?>
<!DOCTYPE BOOK SYSTEM ”book.dtd”>
<BOOK>
.....
</BOOK>

- URL:

<?xml version=”1.0” standalone=”no”?>
<!DOCTYPE BOOK SYSTEM ”http://www.library.org/book.dtd”>
<BOOK>
.....
</BOOK>

External DTDs (Publics)

<!DOCTYPE name PUBLIC ”FPI”
”URL”>
Formal Public Identifier (FPI)

- First Field : - no standard, + standard DTD
- Second Field : name of group or person responsible for DTD
- Third Field : type of document and version number
- Fourth Field : language (2 characters)
- Fields separated by //

<!DOCTYPE BOOK PUBLIC ”-//Joseph Smith//General Book Version 5.3//EN” ”http://www.library.org/book.dtd”>

XHTML Document

```
<?xml version=”1.0” encoding=”UTF-8”?> <!DOCTYPE html PUBLIC ”-//W3C//DTD XHTML 1.0 Strict//EN” ”DTD/xhtml1-strict.dtd”>
  <html xmlns=”http://www.w3.org/1999/xhtml” xml:lang=”en” lang=”en”>
  <head>
  <title>Virtual Library</title>
  </head>
  <body>
  <p>Moved to <a href=”http://vlib.org/”>vlib.org</a>.</p>
  </body>
  </html>
```

Attributes

3.3 XML schemas

With XML schemas, you have more power to define what valid XML documents look like. They have several advantages over DTDs:

XML schemas use XML syntax.

XML schemas support datatypes.

XML schemas are extensible.

XML schemas have more expressive power.

3.3.1 A sample XML schema

Here's an XML schema that matches the original name and address DTD. It adds two constraints: The value of the <county> element must be exactly two characters long and the value of the <postal-code> element must match the regular expression [0-9]5(-[0-9]4)?. Although the schema is much longer than the DTD, it expresses more clearly what a valid document looks like. Here's the schema:

```
 <?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="address">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name"/>
        <xsd:element ref="street"/>
        <xsd:element ref="city"/>
        <xsd:element ref="county"/>
        <xsd:element ref="postal-code"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title" minOccurs="0"/>
        <xsd:element ref="first-Name"/>
        <xsd:element ref="last-Name"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="title"      type="xsd:string"/>
  <xsd:element name="first-Name" type="xsd:string"/>
  <xsd:element name="last-Name"  type="xsd:string"/>
  <xsd:element name="street"     type="xsd:string"/>
  <xsd:element name="city"       type="xsd:string"/>

  <xsd:element name="county">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:length value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>

  <xsd:element name="postal-code">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>
```

Chapter 4

Programming interfaces

4.1 The Document Object Model

The Document Object Model, commonly called the DOM, defines a set of interfaces to the parsed version of an XML document.

The DOM was created by the W3C, and is an Official Recommendation of the consortium.

The DOM provides a rich set of functions that you can use to interpret and manipulate an XML document, but those functions come at a price.

The DOM creates objects that represent everything in the original document, including elements, text, attributes, and whitespace. If you only care about a small portion of the original document, it's extremely wasteful to create all those objects that will never be used.

A DOM parser has to read the entire document before your code gets control. For very large documents, this could cause a significant delay.

4.2 The Simple API for XML

SAX interface

To get around the DOM issues, the XML-DEV participants created the SAX interface.

A SAX parser sends events to your code.

A SAX parser doesn't create any objects at all, it simply delivers events to your application.

A SAX parser starts delivering events to you as soon as the parse begins.

Both SAX and DOM have their place.

4.2.1 SAX issues

SAX parsers also have issues that can cause concern:

SAX events are stateless.

SAX events are not permanent.

SAX is not controlled by a centrally managed organization.

4.3 JDOM

JDOM is a Java technology-based, open source project that attempts to follow the 80/20 rule: Deliver what 80% of users need with 20% of the functions in DOM and SAX.

JDOM works with SAX and DOM parsers, so it's implemented as a relatively small set of Java classes.

The main feature of JDOM is that it greatly reduces the amount of code you have to write.

4.4 The Java API for XML Parsing

Sun has released JAXP, the Java API for XML Parsing. This API provides common interfaces for processing XML documents using DOM, SAX, and XSLT.

JAXP provides interfaces such as the DocumentBuilderFactory and the DocumentBuilder that provide a standard interface to different parsers.

4.5 Which interface?

Will your application be written in Java?

How will your application be deployed?

Once you parse the XML document, will you need to access that data many times?

Do you need just a few things from the XML source?

Are you working on a machine with very little memory?

Be aware that XML APIs exist for other languages; the Perl and Python communities in particular have very good XML tools.

Chapter 5

XLINK -Linking XML Documents

5.1 XLink

The HTML link model:

Construction of a hyperlink:

- <a name="here"> is placed at the destination
- <a href="URL#here"> is placed at the source

Problems when using the HTML model for general XML:

- Link recognition:
  - in HTML, links are recognized by element names (a, img, ..) - we want a generic XML solution (XLink)
  - the "semantics" of a link is defined in the HTML specification - we want to identify abstract semantic features, e.g. link actuation
- Limitations:
  - an anchor must be placed at every link destination (problem with read-only documents) - we want to express relative locations (XPointer)

  - the link definition must be at the same location as the link source (outbound) - we want inbound and third-party links
  - only individual nodes can be linked to - we want links to whole tree fragments
  - a link always has one source and one destination - we want links with multiple sources and destinations

The usual point: generic solutions allow generic tools!

5.2 ATTRIBUTES in a SPECIAL NAMESPACE

XLink is part of a three-layers extension of HTML link-related notions:

- XLink
  - a generalization of the HTML link concept
  - higher abstraction level (intended for general XML - not just hypertext)
  - more expressive power (multiple destinations, special behaviours, linkbases, ...)
  - uses XPointer to locate resources
- XPointer
  - an extension of XPath suited for linking
  - specifies connection between XPath expressions and URIs
- XPath

  - a declarative language for locating nodes and fragments in XML trees
  - used in both XPointer (for addressing), XSL (for pattern matching), XML Schema (for uniqueness and scope descriptions), and XQuery (for selection and iteration)

These technologies are standardized but not all widely implemented yet.

5.3 The XLink linking model

Basic XLink terminology:

- Link : explicit relationship between two or more resources.
- Linking element : an XML element that asserts the existence and describes the characteristics of a link.
- Locator : an identification of a remote resource that is participating in the link.

One linking element defines a set of traversable arcs between some resources.

A local resource comes from the linking element's own content.

- Outbound: the source is a local resource
- Inbound: the destination is a local resource
- Third-party: none of the resources are local

An example

A linking element defining a third-party "extended" link involving two remote resources:

```
<mylink xmlns:xlink="http://www.w3.org/1999/xlink" x
  <myresource xlink:type="locator"
              xlink:href="students.xml#Fred" xlink:l
  <myresource xlink:type="locator"
              xlink:href="teachers.xml#Joe" xlink:la
  <myarc xlink:type="arc"
         xlink:from="student" xlink:to="teacher"/>
</mylink>
```

- the namespace http://www.w3.org/1999/xlink is used to recognize XLink information in general XML documents
  - host language : elements and attributes not belonging to this namespace are ignored by XLink processors
  - all XLink information is defined in attributes (in host language elements)
- xlink:type=”extended” indicates a linking element
- xlink:type=”locator” locates a remote resource
- xlink:type=”arc” defines traversal rules

5.4 Linking elements

- defining links

All elements with XLink information contain an xlink:type attribute.

A general linking element is defined using an xlink:type=”extended attribute; this element can contain the following:

- a local resource is defined withxlink:type=”resource”

- a remote resource is defined with xlink:type=”locator” and with an xlink:href attribute (an XPointer expression locating the resource)
- arcs (traversal rules) are defined with xlink:type=”arc”:
  - both ”resource” and ”locator” elements can have xlink:label attributes
  - an arc element has an xlink:from and an xlink:to attribute
  - the ”arc” element defines a set of arcs: from each resource having the ”from” label to each resource having the ”to” label

5.5 Behaviour

- link semantics

Arcs can be annotated with abstract behaviour information using the following attributes:

- xlink:show
- Possible values:
- xlink:actuate
- Semantic attributes: describe the meaning of link resources and arcs
  - xlink:title
  - xlink:role
  - xlink:arcrole

5.6 Simple vs. Extended links

- for compatibility and simplicity
  Two kinds of links:

  - extended
  - simple

  Shorthand notation for simple links:

```
<mylink xlink:type="simple" xlink:href="..." xlink:s
<mylink xlink:type="extended">
  <myresource xlink:type="resource"
              xlink:role="local"/>
  <myresource xlink:type="locator"
              xlink:role="remote" xlink:href="..."/>
  <myarc xlink:type="arc"
         xlink:from="local" xlink:to="remote" xlink:
</mylink>
```

Chapter 6

XSL -XML Stylesheets

6.1 What Does a Stylesheet Do?

CSS stylesheets are limited:

- cannot change the structure of the document tree (rearrange, insert or remove elements);
- cannot ‘compute’ with data in the document.

CSS can only decorate the tree. What is needed is a real transformation engine, providing the translation:

XML document according to DTD1 - > XML document according to DTD2

The solution is XSL: Extensible Stylesheet Language.

A stylesheet specifies the presentation of XML information using two techniques:

- An optional transformation of the input document into another structure.
- A description of how to present the transformed information.

Transformation capabilities include:

- generation/suppression/duplication/moving/sorting of content;

- ”compute” new information in terms of the existing information.

The full XSL language consists of three component languages:

- XPath : XML Path Language
- XSLT : XSL Transformations
- XSLFO : XSL Formatting Objects

6.1.1 The Structure of a Stylesheet

XSLT Stylesheets are XML documents rooted at <xsl:stylesheet> or <xsl:transform>

The namespace
http://www.w3.org/1999/XSL/Transform
identifies semantically relevant tags.

Stylesheet Examples

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  ...
</xsl:stylesheet>
```

For example:
medskip xsl:template (defines a transformation rule)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
 <xsl:template match="/">
  <HTML>
   <HEAD><TITLE>New XML Document</TITLE></HEAD>
   <BODY>DNA</BODY>
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

An XSLT stylesheet consists of a number of template rules:
template rule = pattern + template.

The source tree is processed by processing the root node. A node list is processed by processing each node in order and concatenating the results. A node is processed by:

- finding the template rule with the best matching pattern
- instantiating its template

Apply templates to the node's children:
xsl:apply-templates (recursive invocation of the transformation)

```
<?xml version='1.0'?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Tran

  <xsl:template match="para">
   <p><xsl:apply-templates /></p>
  </xsl:template>

  <xsl:template match="emphasis">
   <i><xsl:apply-templates /></i>
  </xsl:template>

 </xsl:stylesheet>
```

With this stylesheet, the following XML document:

```
<?xml version='1.0'?>
 <para>This is a <emphasis>test</emphasis>.</para>
```

Would be transformed into:

```
<?xml version="1.0" encoding="utf-8"?>
 <p>This is a <i>test</i>.</p>
```

Get the value of nodes: xsl:value-of (compute the value of XPATH expressions)

XML Student Document

```
<?xml version="1.0"?>
 <course>
  <name id="CS311">DNA</name>
  <teacher id="bsd">BS Doherty</teacher>
  <student id="ft">
   <name>Fred Tong</name>
   <cw1>30</cw1> <cw2>70</cw2> <project>80</project> <
  </student>
  <student id="gb">
   <name>Garry Brown</name>
   <cw1>80</cw1> <cw2>90</cw2> <project>100</project>
  </student>
  <student id="vc">
   <name>Vincent Chandler</name>
   <cw1>60</cw1> <cw2>95</cw2> <project>50</project> <
  </student>
 </course>
```

XSLT Style Sheet (note the select attribute that filters the recursive application of template rules)

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
  <xsl:template match="course">
   <HTML>
    <HEAD><TITLE>Name of students</TITLE></HEAD>
    <BODY>
           <xsl:apply-templates select="student"/>
    </BODY>
   </HTML>
  </xsl:template>

  <xsl:template match="student">
   <P>
      <xsl:value-of select="name"/>
   </P>
  </xsl:template>
 </xsl:stylesheet>
```

Output File

```
<HTML>
 <HEAD><TITLE>Name of students</TITLE></HEAD>
 <BODY>
  <P>Fred Tong</P>
  <P>Garry Brown</P>
  <P>Vincent Chandler</P>   </BODY>
</HTML>
```

Part II

XML applications

Chapter 7

XHTML

XHTML: The Extensible HyperText Markup Language

A Reformulation of HTML 4 in XML 1.0. XHTML is a family of current and future document types and modules that reproduce, subset, and extend HTML4. Some estimates indicate that by the year 2002, 75% of Internet document viewing will be carried out on these platforms.

7.1 Strictly Conforming Documents

1. Must conform to the constraints expressed in one of the three DTDs found in AppendixA.
2. The root element of the document must be <html>.
3. The root element of the document must designate the XHTML namespace using the xmlns attribute.

   The namespace for XHTML is defined to be http://www.w3.org/1999/x
4. There must be a DOCTYPE declaration in the document prior to the root element.

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

```
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict

   <!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transi

   <!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frames
```

The DTD subset must not be used to override any parameter entities in the DTD.

Here is an example of a minimal XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="e
  <head>
    <title>Virtual Library</title>
  </head>
  <body>
    <p>Moved to <a href="http://vlib.org/">vlib.org</a
  </body>
</html>
```

The following example shows the way in which XHTML 1.0 markup could be incorporated into another XML namespace:

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

```
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
    xmlns:isbn='urn:ISBN:0-395-36341-6' xml:lang="en" lang=
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML default namespace for hypertext commenta
    <p xmlns='http://www.w3.org/1999/xhtml'>
        This is also available <a href="http://www.w3.org/"
    </p>
  </notes>
</book>
```

7.2 Differences with HTML4

1. Documents must be well-formed

   Well-formedness is a new concept introduced by [XML]. Essentially this means that all elements must either have closing tags or be written in a special form (as described below), and that all the elements must nest properly.

   CORRECT: nested elements.

   here is an emphasized paragraph.

   INCORRECT: overlapping elements

   here is an emphasized paragraph.

2. Element and attribute names must be in lower case

   XHTML documents must use lower case for all HTML element and attribute names. This difference is necessary because XML is case-sensitive e.g. and <LI> are different tags.

3. For non-empty elements, end tags are required

   CORRECT: terminated elements

   here is a paragraph.

   here is another paragraph.

   INCORRECT: unterminated elements

   here is a paragraph. here is another paragraph

4. Attribute values must always be quoted

   All attribute values must be quoted, even those which appear to be numeric.

   CORRECT: quoted attribute values

   <table rows=”3”>

   INCORRECT: unquoted attribute values

   <table rows=3>

Chapter 8

Scalable Vector Graphics

8.1 Introducing SVG

The SVG specification used here is the version 1.0, available at the W3C Web site [7].

SVG is a language for describing two-dimensional graphics in XML.

Because SVG describes graphics in terms of lines, curves, text, and other primitives, SVG images can be scaled to any arbitrary degree of precision.

8.2 Tools

SVG can be prepared using a text editor, a drawing package which produces SVG output or a file conversion from another graphics format.

The graphics can be viewed with a viewer, which may be standalone or a web browser plug-in.

A full list of tools is available from the w3 web site [8]

8.3 Definitions and concepts

8.3.1 Explaining the name: SVG

SVG stands for Scalable Vector Graphics, an XML grammar for stylable graphics, usable as an XML namespace.

8.3.2 Scalable

8.3.3 Vector

8.3.4 Graphics

8.3.5 XML

8.3.6 Namespace

8.3.7 Stylable

8.4 Important SVG concepts

8.4.1 Graphical Objects

8.4.2 Symbols

8.4.3 Raster Effects

8.4.4 Fonts

8.4.5 Animation

8.5 Options for using SVG in Web pages

There are a variety of ways in which SVG content can be included within a Web page. Here are some of the options:

8.5.1 A stand-alone SVG Web page

8.5.2 Embedding by reference

Three options:

- The HTML/XHTML 'img' element is the most common method for using graphics in HTML pages.
- The HTML/XHTML 'object' element can contain other elements nested within it, unlike 'img', which is empty.
- The HTML/XHTML 'applet' element which can invoke a Java applet to view SVG content within the given Web page.

8.5.3 Embedding inline

In this case, SVG content is embedded inline directly within the parent Web page. An example is an XHTML Web page with an SVG document fragment textually included within the XHTML.

8.5.4 External link, using the HTML 'a' element

This allows any stand-alone SVG viewer to be used, which can (but need not) be a different program to that used to display HTML.

8.5.5 Referenced from a CSS2 or XSL property

8.6 A user view

8.6.1 What is SVG? [1]

Data-driven graphics

SVG Data	Meaning
M 10 10	Moves the current point to (10, 10). M stands for the move command.
L 90 10	Draws a line from the current point to (90, 10). L stands for the lineto command.
..	The lineto command draws a line and moves the current point.
L 90 40	Draws a line from the current point to (90, 40).
L 10 40	Draws a line from the current point to (10, 40).
Z	Closes the path and fi lls it according to the value of the style attribute.

Table 8.1: SVG Data for simple square

Interactive graphics

Personalized graphics

8.7 Transforming XML into SVG

ibm tutorial [6]

8.7.1 <path> elements

To define a shape in SVG, use a <path> element.

Here's an SVG document that contains a sample <path> element:

```
<svg height="50" width="100">
  <path style="stroke:black; stroke-width:2; fill:red"
        d="M 10 10 L 90 10 L 90 40 L 10 40 Z"/>
</svg>
```

The meaning of the d attribute is explained in the following table 8.1:

The x- and y-coordinates of SVG graphics begin in the upper left corner. The point (10, 10) is 10 pixels down and 10 pixels to the right, measured from the upper left corner. See the SVG specification for complete definitions of all drawing commands.

Use IE or another SVG viewer to look at redbox.svg

8.8 <text> elements

To draw text in an SVG image, use the SVG <text> element. The style attribute is used to define the properties of the text, and x and y attributes supplied to tell the SVG rendering engine where to begin drawing.

Here's an SVG document that contains a sample <text> element:

```
<svg height="50" width="100">
  <text style="font-size:36; font-family:Times Roman; fill:blue"
        x="10" y="40">
    The quick brown fox
  </text>
</svg>
```

8.9 <g> elements

The final element user for this SVG transformations is the <g> element.

Here's an SVG document that contains three <path> elements. The first two <path> elements inherit all their properties from the <g> element, while the third overrides one of them.

```
<svg height="50" width="100">
  <g style="stroke:black; stroke-width:2; fill:blue">
    <path d="M 10 10 L 30 10 L 30 40 L 10 40 Z"/>
    <path d="M 40 10 L 60 10 L 60 40 L 40 40 Z"/>
    <path style="fill:green"
          d="M 70 10 L 90 10 L 90 40 L 70 40 Z"/>
  </g>
</svg>
```

When rendered, this SVG image looks like this:
Use IE or another viewer to look at threebox.svg

8.9.1 Arcs

An arc is drawn as part of a <path>, using the A command:

```
<svg height="200" width="200">
  <path style="fill:blue; stroke:black; stroke-width:2;
               fillrule:evenodd; stroke-linejoin:miter;"
        transform="translate(100, 100)
                   rotate(-72.24046757584189)"
        d="M 80 0 A 80 80 0 0 0 25.26622959787925
           -75.90532024770893 L 0 0 Z"/>
</svg>
```

When rendered, this SVG image looks like this:
Use IE or another viewer to look at arc.svg

Chapter 9

XML Databases

9.1 Introduction

9.2 Is XML a Database?

9.3 Why Use a Database?

9.4 Data versus Documents

9.5 Data-Centric Documents

For example, the following sales order document is data-centric:

```
<SalesOrder SONumber="12345">
   <Customer CustNumber="543">
      <CustName>Edsel Industries</CustName>
      <Street>123 High St.</Street>
      <City>Staines</City>
      <PostCode>ST1 1AA</PostCode>
   </Customer>
   <OrderDate>20021215</OrderDate>
   <Item ItemNumber="1">
      <Part PartNumber="123">
         <Description>
            <p><b>Apple corer:</b><br />
            Stainless steel, one-piece construction,
            lifetime guarantee.</p>
```

```
            </Description>
            <Price>19.95</Price>
         </Part>
         <Quantity>10</Quantity>
      </Item>
      <Item ItemNumber="2">
         <Part PartNumber="456">
            <Description>
               <p><b>Stuffing maker:<b><br />
               Aluminum, one-year guarantee.</p>
            </Description>
            <Price>13.27</Price>
         </Part>
         <Quantity>5</Quantity>
      </Item>
   </SalesOrder>
```

For example, consider the following document describing a flight:

```
<FlightInfo>
   <Airline>Fast Airways</Airline> provides
   <Count>three</Count> non-stop flights daily
   from <Origin>Birmingham</Origin> to
   <Destination>Oulu</Destination>. Departure
   times are <Departure>09:15</Departure>,
   <Departure>11:15</Departure>,
   and <Departure>13:15</Departure>.
   Arrival times are three hours later.
</FlightInfo>
```

This could be built from the following XML document and a simple stylesheet:

```
<Flights>
    <Airline>Fast Airways</Airline>
```

```
      <Origin>Birmingham</Origin>
      <Destination>Oulu</Destination>
      <Flight>
         <Departure>09:15</Departure>
         <Arrival>12:15</Arrival>
      </Flight>
      <Flight>
         <Departure>11:15</Departure>
         <Arrival>14:15</Arrival>
      </Flight>
      <Flight>
         <Departure>13:15</Departure>
         <Arrival>16:15</Arrival>
      </Flight>
   </Flights>
```

9.6 Document-Centric Documents

Document-centric documents are usually written by hand in XML or some other format, such as RTF, PDF, or SGML, which is then converted to XML. Unlike data-centric documents, they usually do not originate in the database. (Documents built from data inserted into a template are data-centric; for more information, see the end of section 4.1.) For information on software you can use to convert various formats to XML, see the links to various lists of XML software.

For example, the following product description is document-centric:

```
<Product>
```

```
<Intro>
The <ProductName>Apple corer</ProductName> from
<Developer>Acme Labs, Inc.</Developer> is <Summary>
suited to Bramley apples.</Summary>
</Intro>

<Description>

<Para>The apple corer, which comes in <i>both
right- and left-handed versions</i>, is made of
the <b>finest stainless steel</b>. The wooden
handle is smooth in your hands</Para>

<Para>You can:</Para>

<List>
<Item><Link URL="Order.html">Order your apple
corer</Link></Item>
<Item><Link URL="Corers.htm">Read more about
corers</Link></Item>
<Item><Link URL="Catalog.zip">Download the
catalog</Link></Item>
</List>

<Para>The apple corer costs <b>just 19.95</b>
and, if you order now, comes with a <b>pip
collector</b> as a bonus gift.</Para>

</Description>
```

```
</Product>
```

9.7 Data, Documents, and Databases

9.8 Mapping Document Schemas to Database Schemas

9.8.1 Table-Based Mapping

The table-based mapping is used by many of the middleware products that transfer data between an XML document and a relational database. It models XML documents as a single table or set of tables. That is, the structure of an XML document must be as follows, where the <database> element and additional <table> elements do not exist in the single-table case:

```
<database>
     <table>
        <row>
           <column1>...</column1>
           <column2>...</column2>
           ...
        </row>
        <row>
           ...
        </row>
        ...
     </table>
     <table>
        ...
     </table>
     ...
```

```
</database>
```

9.8.2 Object-Relational Mapping

The way in which the object-relational mapping is supported varies from product to product. For example:

- All products support the mapping of complex element types to classes and simple element types and attributes to properties.
- All(?) products allow you to designate a root element that is not mapped to the object model or database.
- Most products allow you to specify whether properties are mapped to attributes or child elements in the XML document.
- Most products allow you to use different names in the XML document and the database
- Most products allow you to specify the order in which child elements appear in their parent, although in such a way that it is impossible to build many content models.
- Some products allow you to map complex element types to scalar properties.
- Few products support the mapping of PCDATA in mixed content.

9.9 Query Languages

9.9.1 Template-Based Query Languages

The most common query languages that return XML from relational databases are template-based.

SELECT statements are embedded in a template and the results are processed by the data transfer software.

For example, the following template (not used by any real product) uses <SelectStmt> elements to include SELECT statements and column-name values to determine where the results should be placed:

```
<?xml version="1.0"?>
<FlightInfo>
   <Introduction>The following flights have
   available seats:</Introduction>
   <SelectStmt>SELECT Airline, FltNumber, Depart,
   Arrive FROM Flights</SelectStmt>
      <Flight>
         <Airline>Airline</Airline>
         <FltNumber>FltNumber</FltNumber>
         <Depart>Depart</Depart>
         <Arrive>Arrive</Arrive>
      </Flight>
   <Conclusion>We hope one of these meets your
   needs</Conclusion>
</FlightInfo>
```

The result of processing such a template might be:

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

```
      <Introduction>The following flights have
      available seats:</Introduction>
      <Flights>
         <Flight>
            <Airline>Fast</Airline>
            <FltNumber>FS10</FltNumber>
            <Depart>Dec 12, 2002 13:43</Depart>
            <Arrive>Dec 13, 2002 01:21</Arrive>
         </Flight>
         ...
      </Flights>
      <Conclusion>We hope one of these meets your
      needs.</Conclusion>
   </FlightInfo>
```

Template-based query languages can be tremendously flexible. Although the feature set varies from product to product, some commonly occurring features are:

- The ability to place result set values anywhere in the output document, including as parameters in subsequent SELECT statements.
- Programming constructs such as for loops and if statements.
- Variables and function definitions.
- Parameterization of SELECT statements through HTTP parameters.

Template-based query languages are used almost exclusively to transfer data from relational databases to XML documents.

9.9.2 SQL-Based Query Languages

SQL-based query languages use modified SELECT statements, the results of which are transformed to XML. The simplest of these uses nested SELECT statements, which are transformed directly to nested XML according to the object-relational mapping.

9.9.3 XML Query Languages

Unlike template-based query languages and SQL-based query languages, which can only be used with relational databases, XML query languages can be used over any XML document. To use these with relational databases, the data in the database must be modeled as XML, thereby allowing queries over virtual XML documents.

With XQuery, either a table-based mapping or an object-relational mapping can be used.

With XPath, an object-relational mapping must be used to do queries across more than one table.

9.10 Storing Data in a Native XML Database

It is also possible to store data in XML documents in a native XML database.

9.11 Storing and Retrieving Documents

9.12 Storing Documents in the File System

9.13 Storing Documents in BLOBs

Native XML databases are most clearly useful for storing document-centric documents.

Native XML databases are also useful for storing documents whose ”natural format” is XML, regardless of what those documents contain.

Several other uses for native XML databases are to store semi-structured data, to increase retrieval speed in certain situations, and to store documents that do not have DTDs (well-formed documents).

Query Languages

Almost all native XML databases support one or more query languages. The most popular of these are XPath and XQL.

In the future, most native XML databases will probably support XQuery from the W3C.

Updates and Deletes

Transactions, Locking, and Concurrency

Application Programming Interfaces (APIs)

9.14 XML Database Products

Bibliography

[1] Adobe. Svg zone. http://www.adobe.com/svg/.

[2] W3 Consortium. http://www.w3.org/TR/REC-xml.

[3] Denis Howe (ed). Free online dictionary of computing. http://foldoc.doc.ic.ac.uk/.

[4] Charles F Goldfarb and Paul Prescod. *The XML Handbook*. The Definitive XML series. Prentice-Hall PTR, Upper Saddle River, NJ, 2nd edition, 2000.

[5] IBM. Introduction to xml. http://www6.software.ibm.com/developerworks/education/xmlintro/.

[6] Doug Tidwell. Transforming xml into svg. http://www-106.ibm.com/developerworks/education/transforming-xml/.

[7] w3 Consortium. Scalable vector graphics 1.0 specification. http://www.w3.org/TR/SVG/.

[8] w3 Consortium. Svg implementations. http://www.w3.org/Graphics/SVG/SVG-Implementations.