XML and SOAP

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 7818 — Lecture 2 — 09/03/2008

Credit Where Credit is Due

- Portions of this lecture are derived from material in “Web Services: Principles and Technology” by Michael P. Papazoglou and its accompanying instructors materials. As such, they are Copyright 2008 by Pearson/Prentice Hall

Course Prep.

- To keep you informed
  - I'm going to be practicing "Just in Time" lecture prep for this class
    - Be prepared for lectures to be "rough around the edges"
      - Be prepared to ask questions and participate in discussions to smooth things out
      - Volunteer to follow-up on week N's topic in week N+1
- Many reasons
  - ABET, OO A&D, Programming, Lab Remodel, Family, etc.
- My apologies and please bear with me!

Lecture Goals

- Present an Overview of Web Technologies
  - Lay foundation for discussion of Web Services Technologies
- Present an Overview of XML
  - XML is the underlying foundation of nearly all Web Service specifications (known collectively as WS-*)
- Structure:
  - Multiple presentations from Ken (Web Tech/XML*)
  - One presentation by Jude Allred on XML
  - One presentation by Matt Novinger on parsing/generating XML

Web Technologies

- Will make use of slides located here:
  - <http://www.cs.colorado.edu/~kena/classes/7818/f06/lectures/06/index.html>
- for this portion of the lecture

Chapter 3: XML

- XML stands for Extensible Markup Language
    - It is a “markup language generator” in that it can be used to define many different markup languages
- What is a markup language?
    - It is a method for distinguishing text from instructions in typesetting systems
        - Example: This is a very serious matter.
        - This is a *very serious* matter
- and are called tags. Tags in XML have a clear start () and a clear end () if they contain content. If they do not contain content, then they both start and end with a special syntax (<br />).

Background

- XML was developed to address concerns about HTML
  - In particular, HTML mixes document structure and document presentation in one language
  - This makes it difficult to change a document's presentation while keeping its structure the same
    - Note: the situation has changed now with XHTML/CSS/Javascript
- Originally, HTML was meant to address the same concern; it was just supposed to specify document structure, not presentation
  - but the browser wars quickly changed that!
  - In particular, users cared about the presentation of their information, and quickly demanded presentation features
    - <font>, , <margin>, etc.

An additional problem

- An additional problem can be seen by viewing the HTML source of the the CNN website
  - This page is filled with “headlines” and text/images that support those headlines
  - A “major” headline looks like this
    - `<H3><A href="..." class="t1">Earliest certified election results in Florida: 6 p.m. EST</A></H3>`
  - A “minor”headline looks like this
    - `  &#149; <a href="...">Bush sues 4 counties over absentee ballots</a><br>`
  - Is the difference intuitive? :-)
  - Disclaimer: the above code is taken from a few years back

The problem explained

- The problem is that
  - presentation concerns (i.e. making the web page look good)
  - are overriding structural concerns (i.e. this information is a headline)
- The fact that one paragraph is a headline and another is supporting text is completely lost in the HTML
- If you wanted to write a program to search this web page and list all headlines, you would need to code knowledge of CNN's presentation rules to figure out where the headlines are hiding
  - To make matters worse, if CNN changes its presentation, you would have to change your program!

The XML approach

- Imagine if the source for CNN's webpage looked like this
  - <story>
    - <headline class="important">Election returns due at 6 PM EST.</headline>
    - <supportingText>Blah Blah Blah…</supportingText>
  - </story>
- Here, structure is preserved
  - It would be very easy to write a program to grab the headlines out of this document
- So, how do we handle presentation?
  - XSLT, which is covered later in this lecture

Software Engineering Benefits

- XML attacks an accidental difficulty of software engineering
  - Having to define your own file formats
  - Having to write parsers for these formats
- With XML, you can define file formats in a standard way, and any XML parser can be used to parse the file
  - You never have to write a parser again!
  - I threw out hundreds of lines of code from my hypermedia system when I converted my preference files to XML!

XML definitions

- An XML document consists of the following parts
  - a Document Type Definition (or DTD)
  - Data
- The DTD defines the structure of the data. A parser can read the DTD and know how to parse the data that follows it
  - As such, XML documents are said to be “self-describing”: all the information for parsing the data is contained in the document itself

Note

- This lecture is presenting a simplified view of the XML standard
  - In particular, the standard supports a number of ways of associating a DTD with an XML document
  - We will cover only one of these mechanisms in this lecture, known as the internal DTD
  - For more information, buy a book on XML, visit <http://www.xml.com/>, or read the XML standard at:
    - <http://www.w3.org/TR/2000/REC-xml-20001006>
    - Note: the spec is not for the “faint of heart”. I would recommend starting with an XML book

XML Syntax Rules

- XML imposes a number of syntax rules that make it easier to parse than HTML
  - All tags must be closed, e.g.
    - HTML lets you skip the closing p tag, XML does not.
    - Note: the closing tag must match the opening tag!
    -
 - In HTML, you can have single tags like <br> to introduce a horizontal break in the document. The <br> tag has no content associated with it; XML requires tags with no content to explicitly end with a trailing slash, hence <br />.

XML Syntax Rules, continued

- Additional syntax rules
  - All attribute values must be quoted
    - e.g. HTML allows the following
      - blah blah blah
    - XML requires the following
      - blah blah blah
- There are many others
  - concerning legal characters, comments, etc. See the spec for details.

Well-Formed XML Documents

- XML documents are considered well-formed if they conform to the XML Syntax rules
- Well-formed documents can be parsed by any XML Parser without the need for a DTD
    - It can use the syntax rules to parse the document cleanly, but without the DTD it does not know if the document is valid

Valid XML Documents

- An XML document is considered “valid” if
  - (1) it is well-formed and
  - (2) it conforms to the rules specified in its associated DTD
    - That is, if the DTD says that a <p> tag can only contain <b> tags and plain text, then a <p> tag which contains an tag would be considered invalid

Parts of an XML document

- XML declaration
- Document declaration
    - We will be showing a document declaration with an embedded DTD
        - This is only one type of XML document declaration
        - There are various ways of linking XML docs to DTDs
        - You can now ignore DTDs altogether and use XML Schemas instead
- Data

XML Declaration

- An XML document begins with this tag
  - <?xml version=“1.0”?>
- The question marks denote a “processing instruction”
- This instruction is for an XML parser
  - Its provides the parser with additional information about the XML document
- An XML document can contain additional processing instructions
  - The parser will pass these instructions to the client that asked the parser to parse the document
- Can contain other attributes such as **encoding** and **standalone**

Document Declaration

- The document declaration comes after the XML Declaration
- Its tag name is DOCTYPE
    - There are two forms
    - internal
        - <!DOCTYPE greeting [...DTD Goes Here…]>
    - external
        - <!DOCTYPE greeting SYSTEM “greeting.dtd”]>
    - We will cover the first form

DTD Syntax

- The DTD is where you declare the elements (a.k.a. tags) and attributes that will appear in your XML document
- In defining elements, you use regular expressions to declare the order in which elements are to appear
- Attributes can be associated with elements and can have default values associated with them
- NOTE: DTD syntax does NOT follow XML formatting rules
  - This is the primary motivation behind XML Schema: to allow the schema of an XML document BE another valid, well-formed XML document
    - “Its turtles all the way down…”
- Lets look at an example

DTD for a Class Gradebook

- <!DOCTYPE gradebook [
  - <!ELEMENT gradebook (class, student*)>
  - <!ELEMENT class (name, studentsEnrolled)>
  - <!ATTLIST class semester CDATA #REQUIRED>
  - <!ELEMENT name (#PCDATA)>
  - <!ELEMENT studentsEnrolled (#PCDATA)>
  - <!ELEMENT student (name, grade*)>
  - <!ELEMENT grade (#PCDATA)>
  - <!ATTLIST grade name CDATA #REQUIRED>
-]>

What does this mean?

- This DTD defines a document whose root element is called **gradebook**
- The first element in gradebook has to be a **class** element followed by zero or more **student** elements
- A class element contains a **name** and the **number of student's enrolled**
    - It has a required attribute called **semester**
- A student contains a name and zero or more **grades**
- A name, a grade, and the studentsEnrolled are declared as having PCDATA or "Parsed Character Data" as their content => this means that they contain strings
    - The grade element also has an attribute called name

An example

- <?xml version="1.0" ?>
- <!DOCTYPE gradebook […insert DTD from slide 19 here]>
- <gradebook>
  - <class semester="Fall 2004">
    - <name>CSCI 3308</name>
    - <studentsEnrolled>36</studentsEnrolled>
  - </class>
  - <student>
    - <name>Ken Anderson</name>
    - <grade name="lab0">10</grade>
    - <grade name="lab1">9</grade>
  - </student>
- </gradebook>

Element Declarations

- Empty Elements
    - <!ELEMENT BR EMPTY>
- Non-Empty Elements
    - <!ELEMENT NAME (CONTENT)>
        - Content contains a regular expression of element names and/or Character Data
        - #PCDATA - strings are parsed for embedded elements (like searching for a <b> tag within a <p> tag in HTML)
        - #CDATA - strings are not parsed for embedded elements

Regular Expressions in Element Declarations

- Element1, Element2
  - Element2 must follow Element1
- Element1?
  - Element1 is optional
- Element1+
  - At least one Element1 tag must appear
- Element1*
  - Zero or more Element1 tags may appear
- Element1 | Element2
  - Either Element1 or Element2 may appear

Examples

- <!ELEMENT p ((#PCDATA|B|I|EM)+)>
  - A p tag may contain text, or a B element, or an I element, or …
- <!ELEMENT name (first, middle?, last)
  - A name consists of a first and last name and may contain a middle name
- <!ELEMENT shoppinglist (item+)
  - A shopping list contains one or more items

Attribute Declarations

- Declaring attributes requires that you first declare the associated element
  - You then use the ATTLIST element to declare the attributes
  - <!ELEMENT name (first, middle?, last)>
  - <!ATTLIST name
    - age CDATA #REQUIRED
    - height CDATA #IMPLIED
    - gender (male|female) “female”>
- This example declares three attributes, one required and two implied (optional), if no gender attribute is specified, it defaults to “female”
- See the spec. for complete details on ATTLIST tag

Entities

- XML needs a way for characters that indicate markup (such as the “<”and the “>”) to be included in the content of an XML document
  - This mechanism also needs to allow the inclusion of other chars. such as:
    - characters from languages around the world
    - symbols
    - non-printing characters
    - etc.
- XML uses entities for this purpose. Entities have several ways in which they can be specified, but the most common look like this:
  - , &, “, >, <, etc.
  - That is &NAME;
- An XML processor (such as a Web browser) will expand these chars in place before displaying the document

Namespaces

- Sometimes it is necessary to combine the tags of two DTDs or schemas (discussed next) into one XML document
  - Since XML DTDs/schemas (hereafter schemas) can be created independently, it is very easy for different schemas to choose the same name for an element (tag)
  - When the tags of these conflicting schemas are merged, a name clash occurs and it becomes ambiguous as to which element is being referenced in the merged XML document
- Namespaces were added to XML 1.1 to address this problem
  - Essentially, a schema's tags can be associated with a namespace
  - A namespace is given a prefix string and a URL (which must be unique)
  - In an XML document, if a tag "foo" comes from namespace "x" then all references to "foo" in the document appear as "<x:foo></x:foo>"

WSDL makes extensive use of namespaces; the document below makes use of 4 namespaces

```
<?xml version="1.0"?>
<definitions name="Procurement"
  targetNamespace="http://example.com/procurement/definitions"
  xmlns:tns="http://example.com/procurement/definitions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/" >

  <message name="OrderMsg">
    <part name="productName" type="xs:string"/>
    <part name="quantity" type="xs:integer"/>
  </message>

  <portType name="procurementPortType">
    <operation  name="orderGoods">
      <input message = "OrderMsg"/>
    </operation>
  </portType>

  <binding name="ProcurementSoapBinding" type="tns:procurementPortType">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="orderGoods">
      <soap:operation soapAction="http://example.com/orderGoods"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="ProcurementService">
    <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
      <soap:address location="http://example.com/procurement"/>
    </port>
  </service>
</definitions>
```

abstract part

messages

operation and port type

concrete part

binding

port and service

Default Namespace

- If an XML document includes an attribute of the form:
  - xmlns="URI"
- that is, the xmlns attribute does not define a prefix, then this attribute is asserting that all tags within the document that do not have a prefix belong to this namespace
  - In the previous example, all tags belonging to the WSDL namespace appeared with no prefix on them
- Technically, this declaration defines the default namespace for the node it appears on and all its children
  - In practice, since namespace declarations often appear on the root node, it defines the default namespace for the entire document

XML Schema Language (XSL)

- The XML Schema language is designed to take the place of DTDs and
  - provide a standardized way to define XML document structures in XML
    - no weird syntax to deal with as with DTDs
    - standard located at: <http://www.w3.org/XML/Schema>
      - allow the specification of more complicated XML structures
- It does this by providing a set of simple types and a set of composition rules that allow the specification of complex types
  - Lets look at a simple example

Example (DTD)

BookStore.dtd

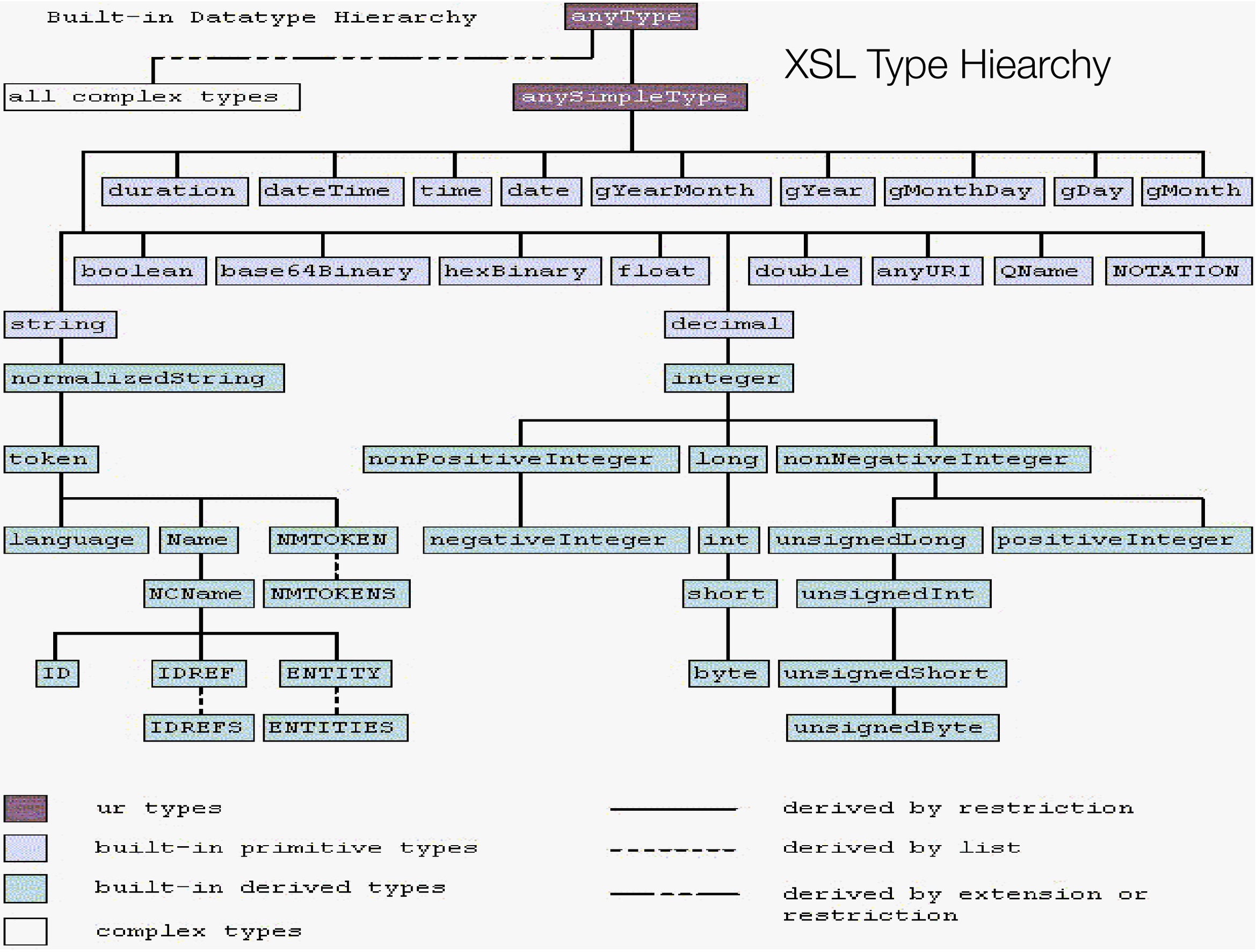
```
<!ELEMENT BookStore (Book)+>

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

<!ELEMENT Title (#PCDATA)>

<!ELEMENT Author (#PCDATA)>

<!ELEMENT Date (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

<!ELEMENT Publisher (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.books.org"
                    xmlns="http://www.books.org" elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Same specification using XSL

See how much BETTER it is!

XSL Type Hiearchy



An Instance Document (using previous schema)

```
“”<?xml version="1.0"?>
<BookStore xmlns=“http://www.books.org”
           xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”
           xsi:schemaLocation="http://www.books.org/BookStore.xsd">
        <Book>
            <Title>My Life and Times</Title>
            <Author>Paul McCartney</Author>
            <Date>July, 1998</Date>
            <ISBN>94303-12021-43892</ISBN>
            <Publisher>McMillin Publishing</Publisher>
        </Book>
        ...
</BookStore>
```

XSLT

- Will use a separate set of slides to discuss XSLT

Summary: XML

- XML provides the ability to create your own tagged language
  - Provides ways to define tags, attributes, entities, etc.
    - Either via DTDs or schemas
  - Namespaces are used to avoid name clashes between schemas
- XSLT provides a way to specify (in XML) how XML documents can be transformed
  - relies on XSLT processor to do the actual transformation

What's Next?

- Chapter 4: SOAP
- Chapter 5: WSDL

- Volunteers?