Application Development with XML and Java

Lecture 2

Review of XML Syntax, DTD, SML Schema

Validity

- One of the important innovations of XML is the ability to place preconditions on the data the programs read, and to do this in a simple declarative way.
- *XML allows you to say that every Order element must contain exactly one Customer element, that each Customer element must have an id attribute that contains an XML name token, that every ShipTo element must contain one or more Streets, one City, and one Postcode, and so forth.*
- Checking an XML document against this list of conditions is called validation. Validation is an optional step but an important one.
- There is more than one language in which you can express such conditions. Generically, these are called *schema languages*, and the documents that list the constraints are called *schemas*.

DTD- Element Declarations

- In order to be valid according to a DTD, each element used in the document must be declared in an *ELEMENT declaration*. For example,

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

- Elements that can have children are declared by listing the names of their children in order, separated by commas. For example,

```
<!ELEMENT Order (Customer, Product,
  Subtotal, Tax, Shipping, Total)>
```

- Attach
  - a question mark after an element name in the content model to indicate that the element is optional
  - an asterisk after the element name to indicate that zero or more instances of the element may occur at that position,
  - a plus sign to indicate that one or more instances of the element must occur at that position

```
<!ELEMENT ShipTo (GiftRecipient?,
  Street+, City, Postcode)>
```


DTD - Attributes

- A DTD also specifies which attributes may and must appear on which elements. Each attribute is declared in an *ATTLIST declaration* which specifies:
  - The element to which the attribute belongs
  - The name of the attribute
  - The type of the attribute
  - The default value of the attribute

```
<!ATTLIST Customer id ID #REQUIRED>
```

- DTDs define ten different types for attributes
- Most parsers and APIs will tell you what the type of an attribute is if you want to know, but in practice this knowledge is not very useful.
- DTDs allow four possible default values for attributes

Document Type Declarations

- Documents are associated with particular DTDs using *document type declarations*.
- The document type declaration is placed in the instance document's prolog, after the XML declaration but before the root element start-tag. For example,

```
<?xml version="1.0" encoding="ISO-
   8859-1"?>
<!DOCTYPE Order SYSTEM "order.dtd">
   <Order> ...
```

- DTDs are not just about validation. They can also affect the content of the instance document itself. In particular, they can:
  - Define entities
  - Define notations
  - Provide default values for attributes

Schemas

- The W3C XML Schema Language addresses several limitations of DTDs.
- First schemas are written in XML instance document syntax, using tags, elements, and attributes.
- Secondly, schemas are fully namespace aware.
- Thirdly, schemas can assign data types like integer and date to elements, and validate documents not only based on the element structure but also on the contents of the elements.

What is an XML Schema

- The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.
- An XML Schema:
  - defines elements that can appear in a document
  - defines attributes that can appear in a document
  - defines which elements are child elements
  - defines the order of child elements
  - defines the number of child elements
  - defines whether an element is empty or can include text
  - defines data types for elements and attributes
  - defines default and fixed values for elements and attributes

Why Schemas?

- XML Schemas use XML Syntax
  - You don't have to learn a new language
  - You can use your XML editor to edit your Schema files
  - You can use your XML parser to parse your Schema files
  - You can manipulate your Schema with the XML DOM
  - You can transform your Schema with XSLT
- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
  - With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- XML Schemas are extensible, because they are written in XML. Therefore you can:
  - Reuse your Schema in other Schemas
  - Create your own data types derived from the standard types
  - Reference multiple schemas in the same document

Simple Example

- XML document called "note.xml":

```
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this
  weekend!</body>
</note>
```

- The following example is a DTD file called "note.dtd"

```
<!ELEMENT note (to, from, heading,
  body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

- The first line defines the note element to have four child elements: "to, from, heading, body".
- Line 2-5 defines the to, from, heading, body elements to be of type "#PCDATA".

Simple Example

- The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/X
  MLSchema"
targetNamespace="http://www.w3schools
  .com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
  <xs:element name="to"
  type="xs:string"/>
  <xs:element name="from"
  type="xs:string"/>
  <xs:element name="heading"
  type="xs:string"/>
  <xs:element name="body"
  type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element></xs:schema>
```

Simple Example

- A Reference to a DTD

```
<?xml version="1.0"?><!DOCTYPE note
  SYSTEM
"http://www.w3schools.com/dtd/note.dt
  d"><note>

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this
  weekend!</body>
</note>
```

- A Reference to an XML Schema

```
<?xml version="1.0"?><note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XML
  Schema-instance"
xsi:schemaLocation="http://www.w3scho
  ols.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this
  weekend!</body>
</note>
```


Schema Element

- The <schema> element is the root element of every XML Schema:

```
<?xml version="1.0"?><xs:schema>...
...</xs:schema>
```

- The <schema> element may contain some attributes. A schema declaration often looks something like this:

```
<?xml version="1.0"?><xs:schema
  xmlns:xs="http://www.w3.org/2001/X
  MLSchema"
targetNamespace="http://www.w3schools
  .com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">...
...</xs:schema>
```

```
xmlns:xs="http://www.w3.org/2001/XMLS
  chema"
```

- indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace.

Schema Element

- This fragment:

```
targetNamespace="http://www.w3schools
.com"
```

  - indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "http://www.w3schools.com" namespace.

- This fragment:

```
xmlns="http://www.w3schools.com"
```

  - indicates that the default namespace is "http://www.w3schools.com".

- This fragment:

```
elementFormDefault="qualified"
```

  - indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Simple Element

- A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.
- The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.
- The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

- XML Schema has a lot of built-in data types. The most common types are:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

Example

- Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>

<dateborn>1970-03-27</dateborn>
```

- And here are the corresponding simple element definitions:

```
<xs:element name="lastname"
  type="xs:string"/>
<xs:element name="age"
  type="xs:integer"/>
<xs:element name="dateborn"

  type="xs:date"/>
```


Attributes

- How to Define an Attribute?
- The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

- XML Schema has a lot of built-in data types. The most common types are:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time
- Here is an XML element with an attribute:

```
<lastname lang="EN">Smith</lastname>
```

- And here is the corresponding attribute definition:

```
<xs:attribute name="lang"
  type="xs:string"/>
```

Restrictions

- The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

```
<xs:element
  name="age"><xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>

</xs:simpleType></xs:element>
```


Complex Elements

- A complex element is an XML element that contains other elements and/or attributes.
- There are four kinds of complex elements:
  - empty elements
  - elements that contain only other elements
  - elements that contain only text
  - elements that contain both other elements and text
- Note: Each of these elements may contain attributes as well!
- An empty complex XML element:

```
<product pid="1345"/>
```

- A complex element, which contains only other elements:

```
<employee>
<firstname>John</firstname>
<lastname>Smith</lastname>
</employee>
```

Empty Elements

```
<product prodid="1345" />
```

- The "product" element above has no content at all. To define a type with no content, we must define a type that allows only elements in its content, but we do not actually declare any elements, like this:

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction
  base="xs:integer">
        <xs:attribute name="prodid"
  type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

</xs:element>
```

- We define a complex type with a complex content. The complexContent element signals that we intend to restrict or extend the content model of a complex type, and the restriction of integer declares one attribute but does not introduce any element content.

Containing Elements

```
<person>
 <firstname>John</firstname>
 <lastname>Smith</lastname>
 </person>
```

- You can define the "person" element in a schema, like this:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname"
  type="xs:string"/>
      <xs:element name="lastname"
  type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

</xs:element>
```

- Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

```
<letter>
Dear Mr.<name>John Smith</name>.
Your order <orderid>1032</orderid>
will be shipped on <shipdate>2001-07-
  13</shipdate>.
</letter>
```

- The following schema declares the "letter" element:

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name"
type="xs:string"/>
      <xs:element name="orderid"
type="xs:positiveInteger"/>
      <xs:element name="shipdate"
type="xs:date"/>
    </xs:sequence>
  </xs:complexType></xs:element>
```

- Note: To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "letter" element.

XML Protocols

- *XML protocols* are XML applications used for machine-to-machine exchange of information across the Internet over HTTP.

Protocol applications of particular interest.

- **RSS:** RSS is used to exchange headlines and abstracts between different Web news sites.
- **XML-RPC:** XML-RPC supports remote procedure calls across the Internet by passing method names and arguments embedded in an XML document over HTTP.
- **SOAP:** Whereas XML-RPC uses only elements, SOAP adds attributes and namespaces as well. SOAP even lets the body of the message be an XML element from some other vocabulary

Message Format

- One of the major uses of XML is for exchanging data between heterogenous systems.
- Since XML is natively supported on essentially any platform of interest, you can send data encoded in such an XML application from point A to point B without worrying about whether point A and point B agree on how many bytes there are in a float, or any of other issues that arise when moving data between systems.
- As long as both ends of the connection agree on the XML application used, they can exchange information without worrying about what software produced the data.
  - One side can use Perl and the other Java. One can use Windows and the other Unix. One can run on a mainframe and the other on a Mac.

Envelopes

- two systems communication
  - they only talk to each other, and they always send the same type of message, an envelope may not be needed. It's enough for one system to send the other the message in the agreed upon XML format.
- many systems communication
  - exchanging many different kinds of messages in many different ways, it's useful to have some standards that are independent of the content of the message. This offers up some hope that when a message in an unrecognized format is received, it can still be processed in a reasonable fashion.
- In XML-RPC, essentially all the mark-up is the envelope and all the text content is the data inside the envelope. SOAP and RSS are a little more complex.

HTTP

- XML is just a document format. If two systems want to exchange messages in XML format, it is not hard for them to do so.
- XML doesn't care how documents are moved from point A to point B, therefore we pick the simplest broadly supported protocol HTTP (Hypertext Transport Protocol)
- Using HTTP to transport XML has a number of advantages, among them:
- HTTP is well supported by libraries in Java, Perl, C, and other languages.
- HTTP is platform independent.
- HTTP connections normally pass through firewalls.
- The HTTP header provides a convenient place to store information such as the document size and encoding.
- HTTP is very well understood in the developer community.

HTTP in Java

- Java lets you write programs that connect to and retrieve information from web sites with hardly any effort.
- Once you have a document in memory you can do whatever you want with it: search it, sort it, transform it, etc.
- Java example to retrieve such information and dump it to the console.
- This example is a Java class that uses the java.net.URL class to load documents via HTTP.
  - getDocumentAsInputStream() connects to a server and returns the unread stream after stripping off the HTTP header.
  - getDocumentAsString() actually reads the entire document, stores it in a string buffer, and then returns a string containing the document at the URL.
  - The method that retrieves the document as an input stream would be used if you want to process the document as it arrives. The string version would be used if the document wasn't too big and you wanted to make sure the entire document was available before working with it.

Example

- It does not have a main() method. It is not intended to be used directly by typing **java URLGrabber** at the command line. Rather this is a library class meant for other programs to use.
- The following is a simple program designed just to test URLGrabber with a very basic command line user interface.
- Here's a simple example of using URLGrabberTest to download the XML document from http://www.slashdot.org/slashdot.xml:

java URLGrabberTest http://www.slashdot.org/slashdot.xml

<?xml version="1.0" encoding="ISO-8859-1"?><backslash xmlns:backslash="http://slashdot.org/backslash.dtd"> <story>

RSS

- RDF Site Summary or Rich Site Summary (sometimes referred to as Really Simple Syndication); a set of XML communication standards created by Netscape.
- RSS allows a web developer to share the content on his/her site. RSS repackages the web content as a list of data items, to which you can subscribe from a directory of RSS publishers.
- RSS content usually includes news stories, headlines, content from discussion lists, or corporate announcemnets and is primarily used by news websites and weblogs.
- RSS "feeds" can be read with a web browser or special RSS reader called a content aggregator.

XML-RPC

- XML-RPC is an XML application designed to enable *remote procedure calls (RPC)* over the Internet. (In Java, a *procedure call* is just a method invocation. In some other languages like C it might be called a function call.)
- It just means that some named chunk of code somewhere is invoked with a list of argument values of particular types. The procedure may or may not return a single value of a known type;
- A *remote procedure call* is one in which the called procedure is not necessarily running on the same host as the calling procedure.
- XML-RPC was hardly the first effort to invent a syntax for remote procedure calls. There have been numerous attempts previously including *CORBA* and Java's own *Remote Method Invocation (RMI)*.

XML - RPC

- XML-RPC bits off the 90% of the problem that gave developers the features they actually needed. It ignored the 10% of the problem that caused 90% of the complexity in previous RPC systems.
- At a very high level, the fundamental idea of XML-RPC is this: An XML document that contains a method name and some arguments is sent to a Web server using HTTP POST. The server invokes the method with the specified arguments. Then it wraps up the return value of the method in another XML document, and sends that back to the client.

SOAP

- XML-RPC did not go through any sort of standardization process. For instance, in XML-RPC the string type is defined as an "ASCII string".
- XML documents are Unicode, not ASCII. Modern programming languages like Java can handle Unicode without any trouble.
- There is no good reason to limit XML-RPC strings to ASCII.
- A more serious effort to enable remote procedure calls by passing XML documents over HTTP is known as the Simple Object Access Protocol.
- SOAP has been developed by a committee of XML experts from various companies including IBM and Microsoft.

- SOAP is a much more robust protocol than XML-RPC. It is much better designed from an XML standpoint as well. It takes advantage of numerous features of XML such as attributes, Unicode, and namespaces that XML-RPC either ignores or actively opposes.
- XML-RPC is adequate for simple tasks.
- SOAP can take you a lot farther. Although there are some basic services available using XML-RPC, the future clearly lies with SOAP.
- The biggest conceptual difference between SOAP and XML-RPC is that XML-RPC exchanges a limited number of parameters of six fixed types, plus structs and arrays. However, SOAP allows you to send the server arbitrary XML elements. This is a much more flexible approach.