Lecture Notes 14
XML Schemas

Anand Tripathi

CSci 4131

Internet Programming

XML Schemas

- An XML defines the validity rules for structuring a document.
- Relationships between elements and attributes
- A schema is itself an XML document, so it must also conform to some structuring rules in order to be valid.
- It allows one to define the data types that can be contained in an element or assigned to an attribute.
  - Restrictions can be specified on data types.
  - For example, a social security contains 9 digits

XML Schema

- A set of predefined basic data types are available for schema definition.
- New complex data types can be defined as structures containing basic data types.
- Restrictions can be defined on the basic data types.
- It also provides constructs to define the rules according to which elements can be nested.
  - Order of nested elements
  - Limitations on the number of times an element can be or should be present

Structure of a Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.skatestown.com/ns/po">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
                  Purchase order schema for SkatesTown.
           </xsd:documentation>
      </xsd:annotation>

          ... More things here
</xsd:schema>
```

Schema Definition Namespace

- **xsd** is used to represent the namespace of the "meta level schema" which is used for defining any new schema:
  - It is associated with the namespace
    - **http://www.w3.org/2001/XMLSchema**
    - **It defines various tags such as "schema", "annotation", "documentation" and many more other elements, structuring rules, and several data type definitions along with ways to specify restrictions.**
- targetNamespace identifies the namespace of the elements in a document which should conform to this schema.

Associating a Schema with a Namespace

- A schema definition document is stored in a file with extension "xsd"
- A document making use of this schema and namespace makes the following declaration

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.skatestown.com/ns/po
                    http://www.skatestown.com/schema/po.xsd"
```

Here:

http://www.skatestown.com/ns/po is namespace id

http://www.skatestown.com/schema/po.xsd is schema location

Simple Types

- Prior to the arrival of schemas, one of the biggest problems with XML processing was that XML had no notion of datatypes,
- Because of this limitation, XML applications included a large amount of validation code. For example, even a simple PO requires the following validation rules, which are outside the scope of the XML Specification:
- Attributes id and customerId of the po element must be positive integers.
- Attribute submitted of the po element must be a date in the format yyyy-mm-dd.
- Attribute quantity of the item element must be a positive integer.
- Attribute sku (stock keeping unit) of the item element must be a string with this format: three digits, followed by a dash, followed by two uppercase letters.

Simple Types

TYPES	EXAMPLES
String	Confirm this is electric
Base64	BinaryGpM7
hexBinary	0FB7
Integer	-126789, -1, 0, 1, 126789
long	
short	-32768 to 32767
float	0.12, -105.7
positiveInteger	1, 126789
negativeInteger	-126789, -1
nonNegativeInteger	0, 1, 126789
nonPositiveInteger	-126789, -1, 0
Decimal	-1.23, 0, 123.4, 1000.00
Boolean	true, false, 1, 0

Simple Types

- Time 13:20:00.000 13:20:00.000-05:00
- dateTime 1999-05-31T13:20:00.000-05:00
- duration P1Y2M3DT10H30M12.3S
  - (1 year, 2 months, 3 days, 10 hours, 30 minutes, and 12.3 seconds) date1999-05-31
- Date 1999-05-31
- Name shipTo
- QName po:USAddress
- anyURI http://www.example.com/
- ID
- IDREF

Extending Simple Types

- A new type must be derived from a base type: a predefined schema type or another already-defined simple type.
- The base type is restricted along a number of facets to obtain the new type. The facets identify various characteristics of the types, for example:
  - length, minLength, maxLength— The exact, minimum, and maximum character length of the value
  - pattern— A regular expression pattern for the value
  - enumeration— A list of all possible values
  - whiteSpace— The rules for handling whitespace in the value
  - minExclusive, minInclusive, maxInclusive, maxExclusive— The range of numeric values that are allowed
  - totalDigits— The number of decimal digits in a numeric value
  - fractionDigits— The number of decimal digits after the decimal point

Simple Types and Facects

	Type Facets					
Simple types	**length**	**minLength**	**maxLength**	**pattern**	**enumeration**	**whiteSpace**
string	x	x	x	x	x	x
base64Binary	x	x	x	x	x	x
hexBinary	x	x	x	x	x	x
integer				x	x	x
positiveInteger				x	x	x
negativeInteger				x	x	x
nonNegativeInteger				x	x	x

The above is only a partial table of the simple data types and the facets of restrictions possible on them.

Example of Data Type Restriction

This example specifies that the type skuType is a string restricted such that it has 3 digits, followed by dash, and then followed by two letters.

```
<xsd:simpleType name="skuType“ >
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
</xsd:simpleType>
```

Examples of Data Type Restriction

```
<xsd:simpleType name="poIdType">
    <xsd:restriction base="xsd:integer">
          <xsd:minExclusive value="10000"/>
          <xsd:maxExclusive value="100000"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="stateType">
     <xsd:restriction base="xsd:string">
          <xsd:enumeration value="AK"/>
          <xsd:enumeration value="AL"/>
          <xsd:enumeration value="AR"/>
            ...
          </xsd:restriction>
 </xsd:simpleType>
```

Complex Types

- *Complex types* , on the other hand, define complex content models, such as those of elements that can have attributes and nested children.
- Complex type definitions address both the sequencing and multiplicity of child elements as well as the names of associated attributes and whether they're required or optional.
- The syntax for defining complex types is straightforward:

```
<xsd:complexType name="typeName">
    <xsd:someTopLevelModelGroup>
     <!-- Sequencing and multiplicity constraints for child elements
          defined using xsd:element -->
    </xsd:someTopLevelModelGroup>
    <!-- Attribute declarations using xsd:attribute -->
</xsd:complexType>
```

Model Groups of the Complex Type

- xsd:sequence— A sequence of elements
- xsd:choice— Allows one out of a number of elements
- xsd:all— Allows a certain set of elements to appear once or not at all, but in any order
- xsd:group— References a model group that is defined elsewhere

Specifying multiplicities of elements

- minOccurs and maxOccurs attributes of xsd:element is used for this purpose.
- The value of zero for minOccurs renders an element's presence optional.
  - The default value for minOccurs is 1.
- The special maxOccurs value "unbounded" is used for an element to indicate that at least one must be present (+ in the document structure diagrams).
- minOccurs=0 implies 0 or 1 occurrence, similar to ? In regular expressions
- maxOccurs=unbounded implies 1 or more occurrences, + in regular expresssions
- minOccurs=0 and maxOccurs=unbounded is similar to *

Example – Defining a new type: addressType

```
<xsd:complexType name="addressType">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string" minOccurs="0"/>
        <xsd:element name="company" type="xsd:string" minOccurs="0"/>
         <xsd:element name="street" type="xsd:string"
                        maxOccurs="unbounded"/>
         <xsd:element name="city" type="xsd:string"/>
         <xsd:element name="state" type="xsd:string" minOccurs="0"/>
         <xsd:element name="postalCode" type="xsd:string"
                        minOccurs="0"/>
         <xsd:element name="country" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="href" type="xsd:IDREF"/>
</xsd:complexType>
```

Schema Example 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns="http://www.cs.umn.edu/4131/ns/bookstore"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.cs.umn.edu/4131/ns/bookstore">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Schema for Book Description
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="book" type="bookType"/>
```

Schema Example 1

```
<xsd:complexType name="bookType">
   <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="author" type="authorType"  maxOccurs="5"/>
      <xsd:element name="description" type="xsd:string"  minOccurs="0"/>
      <xsd:element name="isbn" type="isbnType"  />
      <xsd:element name="price" type="priceType"  />
      <xsd:element name="publisher" type="xsd:string"  />
      <xsd:element name="year" type="yearType" minOccurs="0"  />
      <xsd:element name="edition" type="editionNumber" minOccurs="0"  />
   </xsd:sequence>
   <xsd:attribute name="id" type="xsd:ID" use="optional" />
   <xsd:attribute name="cover" type="coverType" use="optional" />
 </xsd:complexType>
```

Schema Example 1

```
<xsd:complexType name="authorType">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string" />
      <xsd:element name="lastname" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="title" type="personTitle" use="optional" />
  </xsd:complexType>

  <xsd:simpleType name="personTitle">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Doctor" />
      <xsd:enumeration value="Professor" />
    </xsd:restriction>
  </xsd:simpleType>
```

Schema Example 1

```
<xsd:simpleType name="priceType">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="\d{1,3}.\d{2}" />
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="yearType">
   <xsd:restriction base="xsd:positiveInteger">
     <xsd:minInclusive value="1900" />
     <xsd:maxInclusive value="2020" />
   </xsd:restriction>
</xsd:simpleType>
```

Schema Example 1

```
 <xsd:simpleType name="editionNumber">
   <xsd:restriction base="xsd:positiveInteger">
     <xsd:minInclusive value="1" />
     <xsd:maxInclusive value="10" />
   </xsd:restriction>
 </xsd:simpleType>
 <xsd:simpleType name="coverType">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Hardcover" />
     <xsd:enumeration value="Softcover" />
   </xsd:restriction>
 </xsd:simpleType>
 <xsd:simpleType name="isbnType">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="\d{3}-\d{5}" />
   </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

Book --- Example Instance

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bks:book  xmlns:bks="http://www.cs.umn.edu/5131/ns/bookstore"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.cs.umn.edu/4131/ns/bookstore
      http://www.cs.umn.edu/~tripathi/schema/bookType-v2.xsd"
      id="A123"   cover="Softcover" >
  <title> Introduction to XML </title>
  <author     title="Doctor">
    <firstname> Mickey </firstname>
    <lastname>  Mouse  </lastname>
  </author>
  <description> This is about XML   </description>
  <isbn>    123-45678    </isbn>
  <price>   219.99   </price>
  <publisher>   MBI Publishers  </publisher>
  <edition>    5   </edition>
</bks:book>
```

Complex Type Example: Purchase order

```
<xsd:complexType name="poType">
   <xsd:sequence>
         <xsd:element name="billTo" type="addressType"/>
         <xsd:element name="shipTo" type="addressType"/>
         <xsd:element name="order">
              <xsd:complexType>
                   <xsd:sequence>
                        <xsd:element name="item" type="itemType"
                             maxOccurs="unbounded"/>
                   </xsd:sequence>
              </xsd:complexType>
         </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" use="required" type="xsd:positiveInteger"/>
    <xsd:attribute name="submitted" use="required" type="xsd:date"/>
    <xsd:attribute name="customerId" use="required" type="xsd:positiveInteger"/>
</xsd:complexType>
```

Schema Example 2

Complete Purchase Order (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.skatestown.com/ns/po">
   <xsd:annotation>
        <xsd:documentation xml:lang="en">
          Purchase order schema for SkatesTown.
        </xsd:documentation>
    </xsd:annotation>

    <xsd:element name="po" type="poType"/>
```

Complete Purchase Order Schema (2)

```
<xsd:complexType name="poType">  …  </xsd:complexType>  See slide 24
<xsd:complexType name="addressType">  … </xsd:complexType> See slide 17
<xsd:complexType name="itemType">
   <xsd:sequence>
        <xsd:element name="description" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="sku" use="required">
        <xsd:simpleType>
           <xsd:restriction base="xsd:string">
              <xsd:pattern value="\d{3}-[A-Z]{2}"/>
           </xsd:restriction>
        </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute name="quantity" use="required"  type="xsd:positiveInteger"/>
</xsd:complexType>
</xsd:schema>
```

Defining a Schema for Global Priority

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns=http://www.skatestown.com/ns/priority
      targetNamespace="http://www.skatestown.com/ns/priority"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:attribute name="priority" use="optional" default="medium">
          <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="low"/>
                  <xsd:enumeration value="medium"/>
                  <xsd:enumeration value="high"/>
              </xsd:restriction>
          </xsd:simpleType>
      </xsd:attribute>
</xsd:schema>
```

Specifying multiplicities of elements

- The presence of attributes can be required through the use="required" attribute-value pair. To give default and fixed values to attributes, you can also use the aptly named default and fixed attributes of xsd:attribute.

Schema Reusability

- Element references
- Content model groups
- Attribute groups
- Schema includes
- Schema imports

Element references

```
<xsd:element name="comment" type="xsd:string"/>

<xsd:complexType name="personType">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element ref="comment" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="taskType">
    <xsd:sequence>
        <xsd:element name="toDo" type="xsd:string"/>
        <xsd:element ref="comment" minOccurs="0"/>
    </xsd:sequence>
 </xsd:complexType>
```

Content Model Group

Define a group called “comments”. It has a sequence of elements.

```
<xsd:group name="comments">
   <xsd:sequence>
        <xsd:element name="publicComment" type="xsd:string" minOccurs="0"/>
        <xsd:element name="privateComment" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
</xsd:group>
```

Using “comments” group in a type definition

```
<xsd:complexType name="personType">
    <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:group ref="comments"/>
   </xsd:sequence>
</xsd:complexType>
```

Content Model Groups

```
<xsd:complexType name="taskType">
    <xsd:sequence>
        <xsd:element name="toDo" type="xsd:string"/>
        <xsd:group ref="comments"/>
    </xsd:sequence>
</xsd:complexType>
```

Attribute Groups

- Reusable attribute groups can be defined.
- For example, ID and IDREF pair of attributes:

```
<xsd:attributeGroup name="referenceable">
    <xsd:attribute name="id" type="xsd:ID"/>
   <xsd:attribute name="href" type="xsd:IDREF"/>
</xsd:attributeGroup>
```

Attribute Groups

```
<xsd:complexType name="personType">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="referenceable"/>
</xsd:complexType>
```

Attribute Groups

```
<xsd:complexType name="taskType">
    <xsd:sequence>
        <xsd:element name="toDo" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="referenceable"/>
</xsd:complexType>
```

Schema Example 3

This is from Deitels' Book – Figure 14-4 (4th Edition)

This schema is used of describing a computer, as shown in the following XML example

```
<computer:laptop
   xmlns:computer = "http://www.deitel.com/computer"
  manufacturer = "IBM">
     <processor   model = "Centrino">  Intel  </processor>
     <monitor>  17   </monitor>
     <CPUSpeed>   2.4  </CPUSpeed>
     <RAM>  256  </RAM>
</computer:laptop>
```

Schema Example 3

This is from Deitels' Book – Figure 15-13 (5th Edition)

```
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
        xmlns:computer = "http://www.deitel.com/computer"
        targetNamespace = "http://www.deitel.com/computer">

   <simpleType name = "gigahertz">
      <restriction base = "decimal">
         <minInclusive value = "2.1"/>
      </restriction>
   </simpleType>

   <complexType name = "CPU">
      <simpleContent>
         <extension base = "string">
          <attribute name = "model" type = "string" />
          </extension>
      </simpleContent>
   </complexType>
```

Schema Example 3 (continued)

This is from Deitels' Book – Figure 15-13 (5th Edition)

```
<complexType name = "portable">
   <all>
      <element name = "processor" type = "computer:CPU"/>
      <element name = "monitor" type = "int"/>
      <element name = "CPUSpeed" type = "computer:gigahertz"/>
      <element name = "RAM" type = "int"/>
   </all>
   <attribute name = "manufacturer" type = "string"/>
</complexType>

<element name = "laptop" type ="computer:portable"/>

</schema>
```

Type Extensions

- Extension mechanism adds new elements and attributes to an existing type.
  - The content model of extended types contains all the child elements of the base type plus any additional elements added by the extension. Any attributes in the extension are added to the attribute set of the base type.

```
<xsd:complexType name="...">
    <xsd:complexContent>
        <xsd:extension base="someBaseType">
            <!-- Optional extension content model -->
            <!-- Optional extension attributes -->
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Example of Type Extension

This example extends the itemType to include an attribute named unitPrice.

```
<?xml version="1.0" encoding="UTF-8"?> <xsd:schema
    xmlns="http://www.skatestown.com/ns/invoice"
    targetNamespace="http://www.skatestown.com/ns/invoice"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:po="http://www.skatestown.com/ns/po">
<xsd:complexType name="itemType">
  <xsd:complexContent>
      <xsd:extension base="po:itemType">
          <xsd:attribute name="unitPrice" use="required"
                         type="priceType"/>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

Type Restrictions

- Restriction mechanism (seen earlier for simple types) also applies to complex types. Several kinds of restrictions can be specified:
  - Multiplicity restriction
  - Deletion of optional elements
  - Tighter limits on occurrence constraints
  - Specification of default values
  - Specification of types where there were none

Example: Corporate Address which restricts the addressType

- A corporate address does not contain a name.
- Delete the optional name element by adding maxOccurs=0 to the element.

```
<xsd:complexType name="corporateAddressType">
  <xsd:complexContent>
    <xsd:restriction base="addressType">
      <xsd:sequence>
     <xsd:element name="name" type="xsd:string"
                    minOccurs="0" maxOccurs="0"/>
      …
      …
```

Use of xsi:type

- In an instance sometimes it is helpful for the document processing software to know that an instance is a derived-type of a base type expected according to the schema.

```
<?xml version="1.0" encoding="UTF-8"?> <po:po
    xmlns:po="http://www.skatestown.com/ns/po"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.skatestown.com/ns/po
    http://www.skatestown.com/schema/po.xsd" id="43871"
    submitted="2004-01-05" customerId="73852">
<billTo xsi:type="po:corporateAddressType">
  <company>The Skateboard Warehouse</company>
   <street>One Warehouse Park</street>
   <street>Building 17</street>
   <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
</billTo>
```

Schema Includes and Imports

- The “include” mechanism allows one to combine and use different schemas, possibly defining different namespaces.

- The “import” mechanism allows one to use the elements and attributes defined in another schema in its own namespace.

Example of “include”

- Suppose that the schema for “addressType” is stored in a separate file, which is available at
  http://www.skatestown.com/schema/address.xsd

- A new schema with the same namespace, say for purchase-order, can include it as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.skatestown.com/ns/po">
   <xsd:include
    schemaLocation="http://www.skatestown.com/schema/address.xsd"/
    >
    ...
</xsd:schema>
```

Including a Schema of Different Namespace

Following schema is developed for a “mailing-list”. It includes addressType from SkatesTown.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.skatestown.com/ns/mailingList">
     <xsd:include schemaLocation="http://www.skatestown.com/schema/address.xsd"/>
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
       Mailing list schema for SkatesTown.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="mailingList">
     <xsd:sequence>
         <xsd:element name="contact" type="addressType" minOccurs="0"
                                                    maxOccurs="unbounded"/>
         </xsd:sequence>
  </xsd:element>
</xsd:schema>
```

A Mailing-list Document

- All elements and attributes defined in addressType schema have to be clearely marked to indicate that they belong to a different namespace, i.e. the SkatesTown namespace.
- The next example shows a mailing-list document, which references elements in two namespaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<list:mailingList xmlns:list="http://www.skatestown.com/ns/mailingList"
    xmlns:addr="http://www.skatestown.com/ns/po"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.skatestown.com/ns/mailingList
    http://www.skatestown.com/schema/mailingList.xsd
    http://www.skatestown.com/ns/po
    http://www.skatestown.com/schema/address.xsd">
   <contact>
      <addr:company>The Skateboard Warehouse </addr:company>
      <addr:street>One Warehouse Park</addr:street>
      <addr:street>Building 17</addr:street>
       <addr:city>Boston</addr:city>
       <addr:state>MA</addr:state>
       <addr:postalCode>01775</addr:postalCode>
    </contact>
</list:mailingList>
```

Importing a Schema

- This makes the elements and attributes defined in the imported names space part of the new schema's namespace.

```
<xsd:import namespace="http://www.skatestown.com/ns/po"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.skatestown.com/ns/po"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:addr="http://www.skatestown.com/ns/po"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.skatestown.com/ns/po
    http://www.skatestown.com/schema/address.xsd"
    targetNamespace="http://www.skatestown.com/ns/mailingList">
<xsd:import namespace="http://www.skatestown.com/ns/po"/>
<xsd:annotation>
      <xsd:documentation xml:lang="en">
          Mailing list schema for SkatesTown.
      </xsd:documentation> <
/xsd:annotation>
<xsd:element name="mailingList">
     <xsd:sequence>
          <xsd:element name="contact" type="addr:addressType" minOccurs="0"
                                                  maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:element>
</xsd:schema>
```

RSS Feed Aggregator
Example of XML Processing

- RSS
  - RDF Site Summary or (Really Simple Syndication)
  - RSS feed provides a collection of summarized news items.
  - It is an XML document, with a specific structure
  - An aggregator or feed reader can collect feed from many sources and select items on a specific topic
    - You will do this in Assignment 4
- See Harvard Law School site
- See Wikipedia

RSS Structure

(See Wikipedia)

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
    <title>RSS Title</title>
    <description>This is an example of an RSS feed</description>
    <link>http://www.someexamplerssdomain.com/main.html</link>
    <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
    <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
    <ttl>1800</ttl>
    <item>
        <title>Example entry</title>
        <description>Here is some text description.   </description>
        <link>http://www.wikipedia.org/</link>
         <guid>unique string per item</guid>
        <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
    </item>
</channel>
 </rss>
```

RSS/XML Parsing

See javax.xml.parsers documentation page

Also see org.w3c.dom package for Node documentation

XML document is parsed and a DOM (Document Object Model) tree is constructed using a parser.

One can then examine the nodes in this tree, which represent the parts of the document.

Java RSS/XML Parser example

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
public class RSSparser {
public static void main ( String arg[] ) {
  String filePath = arg[0];
  Document  xmlDOM  =  null;
   try {
      DocumentBuilderFactory docBuilderFactory   =
                DocumentBuilderFactory.newInstance();
      DocumentBuilder   docBuilder  =  docBuilderFactory.newDocumentBuilder();
              File  fileObj  =  new File ( filePath );
      xmlDOM  = docBuilder.parse ( fileObj );
   }  catch ( Exception ex )  {
         ex.printStackTrace();
  }
```

Java RSS/XML Parser example

```
NodeList      itemNodeList  =  xmlDOM.getElementsByTagName( "item" );

System.out.println( "Number of items found = " + itemNodeList.getLength() );


 for (int i = 0; i<  itemNodeList.getLength();  i++ )  {
       Node      item = itemNodeList.item( i );
       NodeList      itemContentList = item.getChildNodes();
```

Java RSS/XML Parser example

```
for (int j = 0; j< itemContentList.getLength();  j++ )  {
Node content =  itemContentList.item( j );
  if (content.getNodeName().equals( "title" ) )  {
       Node  child = content.getFirstChild();
       System.out.println(  );
       System.out.println( "----------------------------------------------------------------" );
       System.out.println( "Title  "  +  i + " :"  +   child.getNodeValue() );
    }
     if (content.getNodeName().equals( "description" ) )  {
       Node  child = content.getFirstChild();
       System.out.println( "Description "  +  i + " :"  +   child.getNodeValue() );
    }
  }
 }
 }
}
```

Example of XML Parsing: DOMPrinter.java

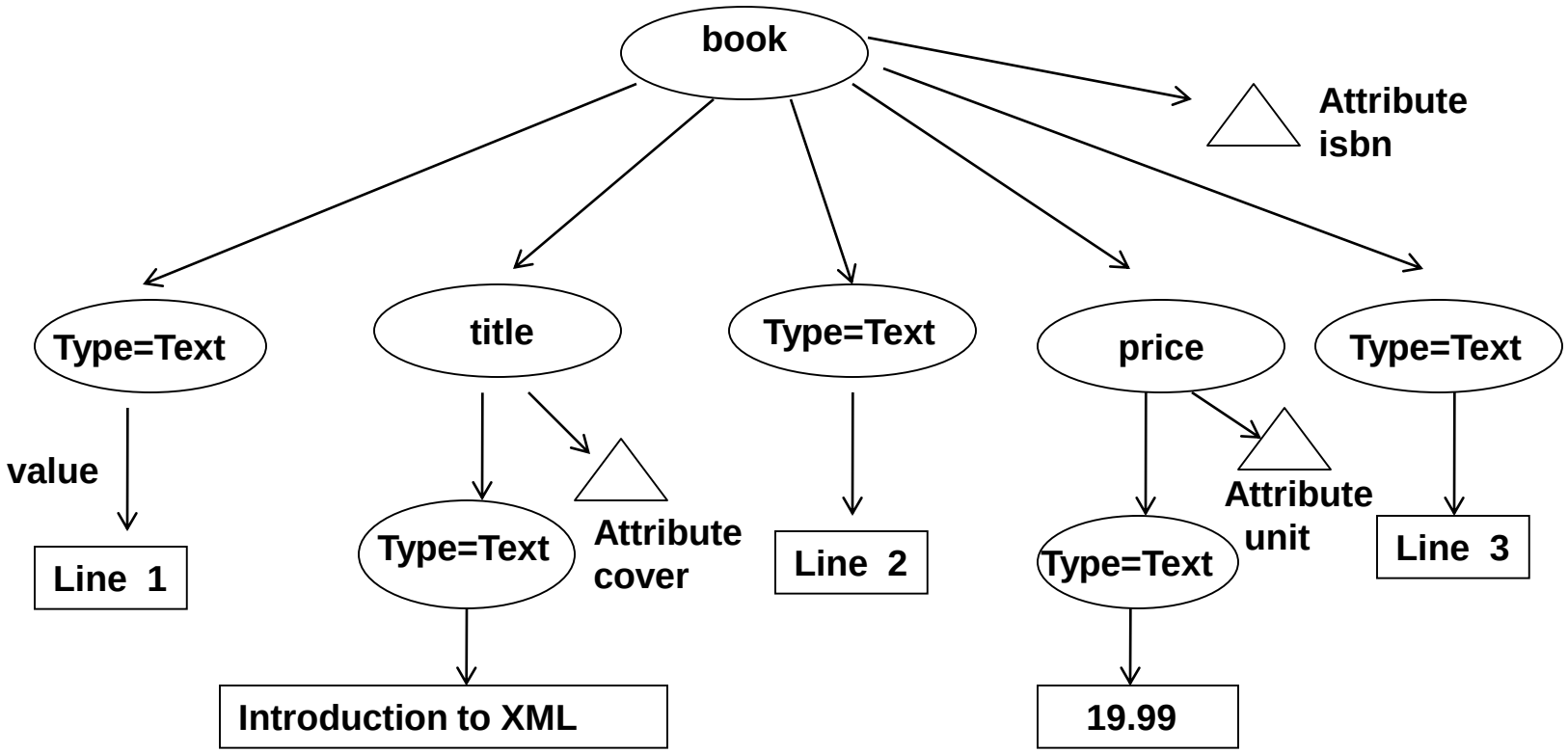
- Input to this program is an XML document file
- This program reads and parses the XML
- Creates DOM (Document Object Model tree)
- It then traverses the tree in depth-first order and prints the information contained in each node.
  - It also prints the level of the node in the tree, with root node at level 0

DOMprinter.java

Example input:

```
<book isbn="980125465">
    Line  1
    <title cover="soft"> Introduction to XML </title>
   Line 2
   <price  unit="dollar">  19.99 </price>
   Line 3
</book>
```

Example input's DOM tree structure



DOMprinter.java

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
public class DOMprinter {
public void visitNode( Node v,  int level ) {
    System.out.println("*** Visiting node at level "  + level);
     v.normalize();
     short nodeType = v.getNodeType();
     if (nodeType == 1  ){
       System.out.println("Node type is Element with name " + v.getNodeName() );
        NamedNodeMap  nMap =  v.getAttributes();
        for (int i=0; i<nMap.getLength(); i++)  {
            Node child   = nMap.item(i);
            nodeType  = child.getNodeType();
            if (nodeType == 2 ){
               System.out.println("Node type is Attribute with name " + child.getNodeName() + " value
      " + child.getNodeValue() );
    } } }
```

DOMprinter.java

```
if (nodeType == 3 ) {
    System.out.println("Node type is Text with text " + v.getNodeValue() );
  }
  if (nodeType == 9 ) {
    System.out.println("Node type is Document root with " + v.getNodeValue() + "  " +
    v.getNodeName()  );
  }


  NodeList childNodesList = v.getChildNodes();
 for (int i=0; i< childNodesList.getLength(); i++)  {
   Node child   = childNodesList.item(i);
   visitNode( child,  level+1  );
 }
} /* end of visitnode */
```

DOMprinter.java

```
public static void main ( String arg[] ) {
  String filePath = arg[0];
  Document  xmlDOM  =  null;
  try {
      DocumentBuilderFactory docBuilderFactory =   DocumentBuilderFactory.newInstance();
      DocumentBuilder   docBuilder  =  docBuilderFactory.newDocumentBuilder();
      File  fileObj  =  new File ( filePath );
      xmlDOM  = docBuilder.parse ( fileObj );
   }  catch ( Exception ex )  {
         ex.printStackTrace();
   }
    short nodeType = xmlDOM.getNodeType();
    if (nodeType == 9 ) {
       System.out.println("Node type is Document root with " + xmlDOM.getNodeValue() + "  " +
        xmlDOM.getNodeName()  );
     }
   Element root = xmlDOM.getDocumentElement();
   DOMprinter  DP =  new DOMprinter();
   DP.visitNode( root, 0 );
 }
}
```