XML Schema Notes

Lecture 13, 07/16/02

<?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:simpleType name="ISBNType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}="/>
      <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}="/>
      <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}="/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:element name="BookStore">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Book" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="Title" type="xsd:string"/>
                  <xsd:element name="Author" type="xsd:string"/>
                  <xsd:element name="Date" type="xsd:gYear"/>
                  <xsd:element name="ISBN" type="ISBNType"/>
                  <xsd:element name="Publisher" type="xsd:string"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
</xsd:schema>

Here we are defining a new (user-defined) data-type, called ISBNType.

I hereby declare a new type called ISBNType. It is a restricted form of the string type. Elements declared of this type must conform to one of the following patterns:

- First Pattern: 1 digit followed by a dash followed by 5 digits followed by another dash followed by 3 digits followed by another dash followed by 1 more digit, or
- Second Pattern: 1 digit followed by a dash followed by 3 digits followed by another dash followed by 5 digits followed by another dash followed by 1 more digit, or
- Third Pattern: 1 digit followed by a dash followed by 2 digits followed by another dash followed by 6 digits followed by another dash followed by 1 more digit.

These patterns are specified using Regular Expressions. In a few slides we will see more of the Regular Expression syntax.

Built-in Datatypes

- Primitive Datatypes
  - string
  - boolean
  - decimal
  - float
  - double
  - duration
  - dateTime
  - time
  - gYearMonth
  - gYear
  - gMonthDay

- Atomic, built-in
  - "Hello World"
  - true, false, 1, 0
  - 12.3456
  - 5.678
  - 12.3456, 12.3456, a, INF, -INF, NaN
  - P1Y2M3DT10H30M12.3S
  - format: CCYY-MM-DDThh-mm-ss
  - format: hh:mm:ss.sss
  - format: CCYY-MM-DD
  - format: CCYY-MM
  - format: CCYY
  - format: MM-DD

Note: 'T' is the date/time separator
INF = infinity
NAN = not-a-number

Built-in Datatypes (cont.)

- Primitive Datatypes (cont.)
  - gDay
  - gMonth
  - hexBinary
  - base64Binary
  - anyURI
  - QName
  - NOTATION

- Atomic, built-in (cont.)
  - format:
    - ---DD
    - --MM--
  - a hex string
  - a base64 string
  - http://www.xfront.com
  - a namespace qualified name
  - a NOTATION from the XML spec

Built-in Datatypes (cont.)

- Derived types
  - normalizedString
  - token
  - language
  - IDREFS
  - ENTITIES
  - NMTOKENS
  - ID
  - ENTITY
  - image
  - xinclude

- Subtype of primitive datatype

- integer
- nonPositiveInteger
- negative infinity to 0
Built-in Datatypes (cont.)

- Derived types
  - negativeInteger
  - long
  - int
  - short
  - byte
  - nonNegativeInteger
  - unsignedLong
  - unsignedInt
  - unsignedShort
  - unsignedByte
  - positiveInteger

Note: the following types can only be used with attributes (which we will discuss later): ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, and ENTITIES.

Creating your own Datatypes

- A new datatype can be defined from an existing datatype (called the "base" type) by specifying values for one or more of the optional facets for the base type.
- Example. The string primitive datatype has six optional facets:
  - length
  - minLength
  - maxLength
  - pattern
  - enumeration
  - whitespace (legal values: preserve, replace, collapse)

Example of Creating a New Datatype by Specifying Facet Values

```xml
<xsd:simpleType name="TelephoneNumber">
  <xsd:restriction base="xsd:string">
    <xsd:length value="8"/>
    <xsd:pattern value="\d{3}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

1. This creates a new datatype called 'TelephoneNumber'.
2. Elements of this type can hold string values,
3. But the string length must be exactly 8 characters long and
4. The string must follow the pattern: ddd-dddd, where 'd' represents a 'digit'.
   (Obviously, in this example the regular expression makes the length facet redundant.)

Another Example

```xml
<xsd:simpleType name="shape">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="circle"/>
    <xsd:enumeration value="triangle"/>
    <xsd:enumeration value="square"/>
  </xsd:restriction>
</xsd:simpleType>
```

This creates a new type called shape.
An element declared to be of this type must have either the value circle, or triangle, or square.

Facets of the integer Datatype

- The integer datatype has 8 optional facets:
  - totalDigits
  - pattern
  - whitespace
  - enumeration
  - maxInclusive
  - maxExclusive
  - minInclusive
  - minExclusive

Example

```xml
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
```

This creates a new datatype called 'EarthSurfaceElevation'.
Elements declared to be of this type can hold an integer. However, the integer is restricted to have a value between -1290 and 29035, inclusive.
Element Containing a User-Defined Simple Type

Example. Create a schema element declaration for an elevation element. Declare the elevation element to be an integer with a range -1290 to 29035

```xml
<elevation>-1290</elevation>
```

Here’s one way of declaring the elevation element:

```xml
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
```

```xml
<xsd:element name="elevation" type="EarthSurfaceElevation"/>
```

Regular Expressions

- Recall that the string datatype has a pattern facet. The value of a pattern facet is a regular expression. Below are some examples of regular expressions:

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Chapter \d</td>
<td>- Chapter 1</td>
</tr>
<tr>
<td>- Chapter&amp;#x020;d</td>
<td>- Chapter 1</td>
</tr>
<tr>
<td>- a*b</td>
<td>- b, ab, aaab, aaab, ...</td>
</tr>
<tr>
<td>- [xyz]b</td>
<td>- xb, yb, zb</td>
</tr>
<tr>
<td>- a?b</td>
<td>- b, ab</td>
</tr>
<tr>
<td>- a+b</td>
<td>- ab, aab, aaab, ...</td>
</tr>
<tr>
<td>- [a-c]x</td>
<td>- ax, bx, cx</td>
</tr>
</tbody>
</table>

Regular Expressions (cont.)

- a{1,3}x
- a[2,]x
- w/aw

- ax, aax, aaaa, ...
- aax, aaaa, aaaa, ...

- word character (alphanumeric plus dash) followed by a space followed by a word character

- [a-zA-Z-]+ * A string comprised of any lower and upper case letters, except “O” and “I”
- \ \ The period “.” (Without the backward slash the period means "any character")

- \n
- \r
- \t
- \\ 
- \-
- \^ 
- \?  
- * 
- + 
- { 
- } 
- ( 
- ) 
- [ 
- ] 

- regular character
- carriage return
- tab
- backslash
- hyphen
- question mark
- asterisk
- plus
- open curly brace
- close curly brace
- open parenthesis
- close parenthesis
- open square bracket
- close square bracket

Regular Expressions (cont.)

```xml
<xsd:simpleType name="money">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[a-zA-Z-][L]\p{Nd}+(\p{Nd}?)?"/>
  </xsd:restriction>
</xsd:simpleType>
```

```xml
<xsd:element name="cost" type="money"/>
```

"Currency sign from any language, followed by one or more digits from any language, optionally followed by a period and two digits from any language"
Example R.E.


0 to 99 100 to 199 200 to 249 250 to 255

This regular expression restricts a string to have values between 0 and 255. … Such a R.E. might be useful in describing an IP address …

IP Datatype Definition

```xml
<xsd:simpleType name="IP">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="(([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.)\4\4\4\4">
    </xsd:pattern>
  </xsd:restriction>
</xsd:simpleType>
```

Flash-range-description

Wide: 0.5 - 3.2m (1.6 - 10.5ft); Tele: 0.5 - 2.3m (1.6 - 7.5ft)

```xml
<xsd:simpleType name="range">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="Wide: \d\d\d\d (\d\d-\d\d))">
    </xsd:pattern>
  </xsd:restriction>
</xsd:simpleType>
```

Constraints on using <all>

- Elements declared within <all> must have a maxOccurs value of "1" (minOccurs can be either "0" or "1")
- If a complexType uses <all> and it extends another type, then that parent type must have empty content.
- The <all> element cannot be nested within either <sequence>, <choice>, or another <all>
- The contents of <all> must be just elements. It cannot contain <sequence> or <choice>