Web Development using Java, JSP, and Web Services

XML & XML Schema

Lecture #9 2008

1. XML
2. XML Parsers
   - SAX
   - DOM
   - XPath
   - XQuery
3. XML Schema

Extensible Markup Language (XML)

- Designed to structure and describe data
- A family of related technologies
- Widely used in a range of technologies today
- Often used for data validation and migration

XML

- Tags are user specified (not predefined)
- XML documents are extensible
  - applications are still able to parse extended documents
  - unrecognized extensions are ignored
- XML is platform, software and hardware independent
- XML is used to structure, store and send information
- XML is both human and machine-readable

XML Syntax

- XML document = hierarchy of elements
- Document version and encoding stated in prolog (optional)
- A single root element
- All elements (except root) has a parent
- Elements must be closed and properly nested
- Each element may have a set of nested children
- Each element can have a set of attributes
- Attributes are name-value pairs
- Attribute names are unique for the element
- Attribute values are escaped and quoted
- Element names are case sensitive
- Comments exclude elements from document content

XML Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rootElement>
  <childElement1 attribute1="value1" attribute2="value2">
    <!-- Element names are not necessarily unique -->
    <subChild>Value1</subChild>
    <subChild>Value2</subChild>
  </childElement1>
  <childElement2>
    <subChild3>Value3</subChild3>
  </childElement2>
  <!-- childElement3 is an empty element -->
  <childElement3/>
  <!- childElement4 is an empty element with attribute -->
  <childElement4 attribute4="value4"/>
</rootElement>
```

XML namespaces

- Defines a collection of elements (a "vocabulary")
- Usually has an associated syntax (e.g., a Schema)
- Implies a semantic (user "knows what is means")
- Namespaces resolves naming conflicts
- Namespaces gives elements given fully qualified names
- Namespaces are declared as attributes and givens prefixes
- A namespaces is simply a unique name (URI)
- All child elements with the same prefix are associated with the same namespace
- A document can assign a default namespace (no prefix)

XML Namespace Example

```
<addresslist
     xmlns:us="http://us.addresses"
     xmlns:swe="http://swe.addresses">

  <us:address>
    <us:street>E-street</us:street>
    <us:city>L.A</us:city>
    <us:state>California</us:state>
    <us:zip>12345</us:zip>
  </us:address>

  <swe:address>
    <swe:street>Kungsgatan 50</swe:street>
    <swe:city>Stockholm</swe:city>
    <swe:postcode>12345</swe:postcode>
  </swe:address>

</addresslist>
```

XML Interpretation

- Well-formed: conforms to XML syntax specifications
  e.g., tags properly nested, tags closed, attributes quoted
- Valid: conforms to a namespace-defined syntax
  i.e., document matches a syntax definition (schema)

XML Parsers

- Programs that traverse XML documents and retrieve data
- Usually transforms data into a custom format
- Can ignore data and insert default values
- Usually slow
- Should incorporate extensive fault tolerance techniques

Simple API for XML (SAX)

- Uses an event-driven parse model
- Performs online parsing of data streams
- Requires very little memory
- Somewhat cumbersome to use

Document Object Model (DOM)

- Constructs a node tree from a document
- Allows off-line processing of data
- Requires large amounts of memory
- Simple to use

XML Path Language (XPath)

- Expression language for addressing parts of XML documents
- Used for pattern-based searching of XML documents
- Possible to search on elements, attributes and tag data
- Integrates well with the DOM API

XML Query Language (XQuery)

- An query language for XML documents (SQL for XML)
- Used for structured data access in XML documents
- Uses the same data model as XPath
- Contains simple programming language constructs
- Integrates well with the DOM API

XML Schema

- Describes the structure of a XML document
- Is used to validate XML documents
- Is itself a XML document
- Typically difficult to read
- Defines elements, attributes and data types

Terminology

- A XML Schema *validates* a XML document
- A XML document *matches* a XML Schema
- A XML document is an *instance of* a XML Schema

XML Schema Documents

- Root tag called schema
- Matches the XML Schema document namespace http://www.w3.org/2000/08/XMLSchema
- Usually declares its own namespace
- All defined elements etc belongs to this namespace
- Instance elements with this namespace are validated against the schema

XML Schema Namespace

```
<?xml version="1.0"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://my.language.ns"
     xmlns="http://my.language.ns"
     elementFormDefault="qualified">

...

</xsd:schema>
```

XML Schema Basics

- Schemas contain
  - Type definitions
  - Element declarations
  - Attribute declarations
- Simple type
  - Does not have sub-elements
  (no child elements or attributes)
  - Predefined type or derived from predefined type
- Complex type
  - Have sub-element(s) - elements and/or attributes
- Element declarations can reference both simple and complex types
- Attributes can only reference simple types

Predefined Types

- string
- QName
- int
- long
- decimal
- double
- boolean
- date
- time
- dateTime
- anyURI

Simple Types

Declarations:

```
<xsd:element name="lastname" type="xsd:string"/>
<xsd:element name="age" type="xsd:int"/>
<xsd:element name="birthdate" type="xsd:date"/>
```

Instances:

```
<lastname>Andersson</lastname>
<age>37</age>
<birthdate>1970-01-01</birthdate>
```

Attributes

Declaration:

```
<xsd:attribute name="nationality" type="xsd:string"/>
```

Instance:

```
<person nationality="Sweden">Kalle Kula</person>
```

Derived Simple Types

- Derived from existing simple types (predefined or derived)
- Typically restricts existing simple types
- Use *restriction* elements with facets to restrict value ranges
- Facets are rules of restriction - Integer: minInclusive, maxInclusive, minExclusive, etc
  - String: pattern, whitespace
  - Any type: enumeration (lists valid values)

Derived Simple Types

Declaration:

```
<xsd:simpleType name="CarType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Audi"/>
    <xsd:enumeration value="BMW"/>
    <xsd:enumeration value="Volvo"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="car" type="CarType"/>
```

Instance:

```
<car>Audi</car>
<car>BMW</car>
<car>Volvo</car>
```

Complex Types

Contains

- Element declarations
- Element references
- Attribute declarations

Model (of how elements occur in the type)

- Sequence - each component in specified order
- Choice - exactly one of the specified components
- All - each component in any order

Complex Types

Declaration:

```
<xsd:complexType name="USAddressType" >
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="street" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="state" type="xsd:string" />
    <xsd:element name="zip" type="xsd:decimal" />
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="address" type="USAddressType"/>
```

Complex Types

Instance:

```
<address country="US">
  <name>John</name>
  <street>E-street</street>
  <city>L.A</city>
  <state>California</state>
  <zip>12345</zip>
</address>
```

Derived Complex Types

Declaration:

```
<xsd:complexType name="BaseType">
  <xsd:sequence>
    <xsd:element name="a"/>
    <xsd:element name="b"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ExtendedType">
  <xsd:complexContent>
    <xsd:extension base="BaseType">
       <xsd:sequence>
         <xsd:element name="c" type="xsd:string"/>
         <xsd:element name="d" type="xsd:string"/>
       </xsd:sequence>
     </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Derived Complex Types

Instance:

```
<baseTypeInstance>
  <a>...</a>
  <b>...</b>
</baseTypeInstance>

<extendedTypeInstance>
  <a>...</a>
  <b>...</b>
  <c>...</c>
  <d>...</d>
</extendedTypeInstance>
```

Schema Composition

- include - add multiple schemas with the same target namespace to a document
- import - add multiple schemas with differing target namespaces to a document

any

Allows documents to use elements not specified by schema

```
<xs:element name="person" type="PersonType"/>
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="firstname"type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:any minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

anyAttribute

Allows documents to use attributes not specified by schema

```
<xs:element name="person" type="PersonType"/>
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="firstname"type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="married" type="xs:boolean" use="required"/>
  <xs:anyAttribute/>
</xs:complexType>
```

Next Time

- Web Services