Distributed Software Development

XML Schema

Chris Brooks

Department of Computer Science
University of San Francisco

Modifying XML programmatically

- Last week we saw how to use the DOM parser to read an XML document.
- The DOM parser can also be used to create and modify nodes.

Creating new nodes

- The top-level Node in an XML document is of class Document
- It contains a set of factory methods that allow you to create new nodes.
  - doc = minidom.parse('cdcat.xml')
  - doc.createElement('songs')
  - doc.createTextNode('Tomorrow Never Knows')
  - doc.createAttribute('encoding')
- After creating a node, it can be added to the tree.

Adding nodes

- We then insert nodes by attaching them to existing nodes.
  - node.appendChild(newNode)
  - node.insertBefore(newNode, childAfter)
  - node.replaceChild(newNode, oldNode)
- We can also remove nodes:
  - node.removeChild(nodename)

Example

```
def changeRating(artist, newrating) :
    cds = doc.getElementsByTagName('cd')
    for item in cds :
        a = item.getElementsByTagName('artist')
        if a[0].firstChild.data.strip() == artist.strip() :
            r = item.getElementsByTagName('rating')
            r[0].replaceChild(doc.createTextNode(newrating), r[0].firstChild)
    return doc
```


Example

```
def addLabel(artist, label) :
    cds = doc.getElementsByTagName('cd')
    print cds
    for item in cds :
        print item
        a = item.getElementsByTagName('artist')
        if a[0].firstChild.data.strip() == artist.strip() :
            n = doc.createElement('Label')
            n.appendChild(doc.createTextNode(label))
            item.appendChild(n)
    print cds[1].toxml()
    return doc
```

Defining XML documents

- Recall that, unlike HTML, an XML author can declare any tags he or she wants.
- If you're just making your own simple documents, an ad hoc approach can work fine.
- If you're building more complex applications, need to incorporate legacy data, or need to exchange data with others, this may not be suitable.
- XML Schema are a way to formally define legal XML documents.

Data Interchange

- A challenge in exchanging data between heterogenous systems is ensuring that all participants agree on the meaning and representation of the data.
  - Is author a sub-element of book, or the other way around?
  - Do all books have to have an ISBN tag, or is it optional? What is the format of a valid ISBN number?
  - Must price be a float?
  - Is there an order that elements must occur in?
- XML allows users of data to validate this data against a *schema*.

DTDs vs Schema

- There are (at least) two different mechanisms for specifying the legal structure of an XML document.
  - DTDs
  - XML Schema
- DTDs are an older technology
  - Less flexible, but still found in many documents.
- XML Schema are a newer, W3C-backed standard.

DTDs

- A Document Type Definition is information about the legal structure of an XML document.
- A DTD allows you to specify the set of allowable elements (the vocabulary), how they fit together (the grammar), and the legal values that can be assigned to them (their semantics).

DTD example

```
<!-- BTW, here's how to add a comment. This is our book DTD  --!>
<!ELEMENT book (author+ , subtitle?,  title, price+, publisher+, isbn,
(volumes | description)*)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT volumes volume+>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ATTLIST book
    genre   (fantasy | sci-fi | mystery | horror) ''fantasy''
    id      ID                                    #REQUIRED
>
```


Weaknesses of DTDs

- DTDs are still used, often for backward compatibility.
- Weaknesses:
  - Not XML - require a different parser.
  - Can't constrain ranges
  - Overly restrictive about order.

XML Schema

- XML Schema are one of several proposed techniques for describing how elements can be arranged.
  - DTDs are the other common way to do this.
  - Schemata are more flexible and expressive than DTDs
  - Backed by W3C
- Essentially an XML document that describes XML documents.
  - Allow you to specify order, data types, number of occurrences, etc.

An example

(see external example)

Using a schema

- We can then use the schema to *validate* an XML document.
- This lets us programmatically ensure that the document is well-formed.
  - Helps with data integration, testing output, verifying received data.
- Schemata also serve as a form of documentation
- Can also be used to provide application-level and parsing guidance.

Schema datatypes

- Schema let us specify what data types an element can have:
  - xs:string - any text
  - xs:token - tokens separated by whitespace
  - decimal - float
  - xs:integer - integer
  - xs:ID - - provides a unique identifier
  - xs:boolean - 'true' or 'false'
  - xs:dateTime - 2004-11-03T11:03:00-10:00

Complex types

- Many interesting XML elements are not just simple data types, but are are compositions of simple types.
- For example, let's say we want a date element that looks like this:

```
<date>
<month> 12 </month>
<day> 13 </day>
<year> 1972 </year>
</date>
```


Complex types

- A Schema for this would look like:

```
<xs:element name=''date''>
<xs:complexType>
  <xs:all>
    <xs:element ref=''year''/>
    <xs:element ref=''month''/>
    <xs:element ref=''day''/>
  </xs:all>
 </xs:complexType>
</xs:element>
<xs:element name=''year'' type=''xs:integer''/>
<xs:element name=''month'' type=''xs:integer''/>
<xs:element name=''day'' type=''xs:integer''>
```

Value Restrictions

- We might also want to specify that months must be between 1 and 12.
- We do this by making a new type.

```
<xs:simpleType name="monthNum">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="12"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="month" type="monthNum"/>
```


Value Restrictions

- We can also specify patterns that an element must follow, using a regular expression.
- For example, let's say we want to specify that a price is one or more numbers, followed by a decimal point, followed by two numbers.

```
<xs:element name=''price'' type=''priceval''>
<xs:simpleType name=''priceval''>
 <xs:restriction base=''xs:token''>
   <xs:pattern value=''[0-9]+.[0-9]2''/>
 </xs:restriction>
</xs:simpleType>
```


Value Restrictions

- We can also define particular values that an element can take on with an enumeration.

```
<xs:simpleType name="genderType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="female"/>
    <xs:enumeration value="male"/>
  </xs:restriction>
</xs:simpleType>
```


Groupings

- XML Schema provide a number of tools for grouping elements.
  - xs:choice

```
<xs:choice minOccurs="0">
  <xs:element ref="junior"/>
  <xs:element ref="senior"/>
</xs:choice>
```

  - xs:all
  - maxOccurs=n
  - xs:enumeration
  - xs:sequence

Referencing Schema

- Once your schema is defined, it can be referenced from within an instance document.

```
<book
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://www.cs.usfca.edu/xml-schema/tolkien.xsd"
>
```

- The first line provides a reference to the definition of 'xsi:schemaLocation'.
- The second line points to the actual URI of the schema.

Namespaces

- A big advantage of schemata is the ability to incorporate multiple schemata within a single document.
- Each schema has its own *namespace*, indicated by a prefix on the element.

```
<book
 xmlns:usf1 = "http://www.cs.usfca.edu/xml-schema"
 xmlns:usf2 = "http://www.cs.usfca.edu/newSchema/"
 >
  <usf1:title> Here's the book title </usf1:title>
  <usf2:title> Here's another type of title </usf2:title>
</book>
```

- Note that, in practice, there's no promise that the URIs will resolve to actual schema.

Summary

- Schemata provide a mechanism to formally define the legal structure of an XML document.
- Allow you to specify data types, required elements, and values.
- This allows you to *validate* an XML document.
- Like many things on the WWW, standards are evolving and incompletely implemented.