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('oscar.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) :
    nds = doc.getElementsByTagName("cd")
    for item in nds :
        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) :
    nds = doc.getElementsByTagName("cd")
    print nds
    for item in nds :
        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 nds[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 heterogeneous 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+ , publisher?,  title, price?, publication?, isbn,
(volumes | description)+)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT volumes volume*>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT publication (#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> 10 </month>
  <day> 10 </day>
  <year> 1973 </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"/>
```