Distributed Software Development

XML

Chris Brooks

Department of Computer Science
University of San Francisco
5-0: Outline

- About XML
- Structuring XML documents
- Validating XML with schema
- Using CSS to display XML
- Parsing with DOM
XML is a language for describing data
   Really more of a meta-language
XML itself provides metadata
   Data types, relations between data objects, etc.
Designed to be read, created, and consumed by programs.
5-2: Advantages of XML

- Well-defined, easy-to-manipulate structure
- Human-readable
- Extensible
- Metadata can be included directly with data
- Widely used
5-3: Things to note

6 An XML document has two components:
   △ tags (metadata)
   △ content (data)

6 Metadata serves to help an application make sense of the data.
<?xml version="1.0"?>
<book>
  <author> J.R.R. Tolkien </author>
  <title> The Lord of the Rings </title>
  <volumes>
    <volume> Fellowship of The Ring </volume>
    <volume> The Two Towers </volume>
    <volume> Return of the King </volume>
  </volumes>
  <price> 14.95 </price>
  <publisher> Ballantine </publisher>
  <isbn> 0345340426 </isbn>
</book>
An XML document can also be represented as a tree.

This makes XML very easy to parse.

The outermost element is the root element, and elements contained within it are children of that element.

Content is stored at the leaves.

What would the tree for our Tolkien example look like?
5-6: Outline

- About XML
- Structuring XML documents
- Validating XML with schema
- Using CSS to display XML
- Parsing with DOM
XML requires that every starting tag have a corresponding closing tag.

Everything between a starting tag and a closing tag is called an *element*

For example, `<volume>Return of The King</volume>` is an element

So is everything between `<volumes>` and `</volumes>`

As is everything between `<book>` and `</book>`.

This means that elements must be nested.
Tags form the boundaries of elements, and give processing instructions to parsers.

- **Empty elements**: `<coAuthor />` All information is contained in the tag.

- **Container elements**: `<price> 14.95 </price>`

- **Comments**: `<!-- here’s a comment -->`

- **Declaration**: `<!ENTITY jrrt “J.R.R. Tolkien”>` This provides a way to define variables or constants in a single location.

- **Entity reference**: `<author> &jrrt </author>`
You can also specify that an element has *attributes*

These attributes can take on *values*

This is helpful when you want to specify that an object belongs to one of a few types.

```xml
<book genre="fantasy" size="large"> ... 
</book>
```
5-10: Attributes vs. Sub-elements

- We could rewrite the example above using subelements instead of attributes.
- When to use one over the other is largely stylistic.
  - Can always transform one into the other
- If a feature can only take on one of a few values, an attribute might make more sense.
- If we expect to extend the number of genres, a subelement is preferable.
- Also, order is preserved for subelements
  - Semantically, attribute/value pairs are treated as a dictionary.
- So, a list of authors should be done as subelements
A particularly helpful attribute is ID - this lets you assign a reference to an element and refer to it later in the document.

```
<volume id="book1"> Fellowship of the Ring </volume>
<volume id="book2"> The Two Towers. Read this book after you’ve finished <volumeref idref="book1" />
```

The ref tag refers to a previous volume.

This provides the XML parser with the information that this is a reference to a previous volume with id “book1”.
If you’ve looked at XML that’s used by other applications, you’ve probably noticed a lot of messy-looking stuff at the top.

This is called the **document prolog**.

This tells a client that the document is in XML and refers it to other document that indicate which tags are valid.

```xml
<?xml version="1.0" encoding="US-ASCII" standalone="no">
<!DOCTYPE book
  PUBLIC "-//USF //DTD Book 1.8//EN"
  "http://www.foobar.com/DTDs/lotr.dtd"
[
  <!ENTITY jrrt "J.R.R. Tolkien">
  <!ENTITY elvish-key "elvish.xml">
]>
5-13: Document Prolog

<?xml version="1.0" encoding="US-ASCII" standalone="no">

- This is the XML declaration.
- It indicates that the document is XML, the encoding schema, and whether or not the client will need to fetch supporting documents.
<!DOCTYPE book>

This is the document type declaration - it indicates that the root element in the XML document is a book.
PUBLIC "-//USF //DTD Book 1.8//EN"
"http://www.foobar.com/DTDs/lotr.dtd"

These lines designate a document type definition.

Basically, this points to a separate document (called a DTD) that describes what elements books are allowed to have.
These lines declare an *internal subset*. These are sort of like C macros; they give a shorthand for elements that occur repeatedly throughout the document.

All of the lines in the prolog except for the first are optional.
We could then use our entity definitions later in the document by prepending a ’&’ to them

<book> ... 
<description> the Author of The Lord of the Rings is &jrrt; he invented a grammar and semantics for Elvish, which can be found at &elvish-key; 
</description>
About XML
Stucturing XML documents
Validating XML with schema
Using CSS to display XML
Parsing with DOM
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?
- 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.
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.
5-21: An example

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- document element -->
  <xs:element name="book">
    <xs:complexType>
      <xs:all>
        <xs:element ref="title"/>
        <xs:element ref="author"/>
        <xs:element ref="price"/>
        <xs:element ref="publisher"/>
        <xs:element ref="ISBN"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <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>
  </xs:element>
</xs:schema>
5-22: 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` - like IDs in DTDs
  - `xs:boolean` - 'true' or 'false'
  - `xs:dateTime` - 2004-11-03T11:03:00-10:00
Many interesting XML elements are not just simple data types, but are compositions of simple types.

For example, let’s say we want a date element that looks like this:

```xml
<date>
  <month>12</month>
  <day>13</day>
  <year>1972</year>
</date>
```
A Schema for this would look like:

```xml
<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'>
```
We might also want to specify that months must be between 1 and 12.

We do this by making a new type.

```xml
<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"/>
```
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.

```xml
<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>
```
We can also define particular values that an element can take on with an enumeration.

```xml
<xs:simpleType name="genderType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="female"/>
    <xs:enumeration value="male"/>
  </xs:restriction>
</xs:simpleType>
```
XML Schema provide a number of tools for grouping elements.

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

- `xs:all`
- `maxOccurs=n`
- `xs:enumeration`
- `xs:sequence`
XML Schema allow you to validate an XML document.

This ensures that all the data is formatted correctly.

Validation allows heterogeneous processes to exchange data with confidence.
5-30: Outline

- About XML
- Structuring XML documents
- Validating XML with schema
- Using CSS to display XML
- Parsing with DOM
CSS can also be used to display XML documents.

Control is limited to laying out a complete XML document.

If we want filtering or sorting, we’ll need to use XSLT.
Let’s say we have an XML-based CD database:

We can use CSS to display it in a web browser.

(see separate examples)
5-33: Outline

- About XML
- Structuring XML documents
- Validating XML with schema
- Using CSS to display XML
- Parsing with DOM
XML also has the advantage of being easy for programs to parse and construct.

There are two different approaches to parsing and manipulating XML.

**SAX: Simple API for XML**
- Event-driven parser
- User defines actions to take when an element is found during parsing.
DOM: Document Object Model

- Tree parser: Entire document is instantiated in memory as a tree.
- Nice for random-access applications
- Large documents may consume a large amount of memory

Most languages provide support for both. We’ll focus on DOM.
Java
- `javax.xml.parsers` built into Java 1.5
- Apache's Xerces parser provides support for both SAX and DOM.
  - Xerces also has C++ and Perl implementations
- JDOM is also a popular tool for parsing and creating XML in Java.

Python
- Built-in support for SAX, DOM, and minidom

Perl
- LibXML provides SAX and DOM functionality.

C#  
- .NET has built-in support for SAX and DOM
Example:

```python
from xml.dom import minidom
doc = minidom.parse('library.xml')
```

Reads in and parses a document
creates a Document object.
toxml() show the XML version.
5-38: Traversing the tree

- childNodes, firstChild, lastChild, parentNode
- childNodes can have childNodes.
- Leaves are text nodes,
  - Respond to ‘data’, which gives up the data they store.
- This is useful if you need to process an entire document, but annoying if you’re searching.
5-39: Finding specific elements

- `getElementsByTagName` finds all elements according to name:
  ```javascript
  eltlist = doc.getElementsByTagName('key')
  ```
- Can search at any node
Nodes have a dictionary-like structure that holds attribute/value pairs:

```python
eltlist = doc.getElementsByTagName('key')
nodel = eltlist[0]
attrs = eltlist[0].attributes
keys = eltlist[0].attributes.keys()
```
Let's build a simple program for reading and displaying XML

```python
#!/usr/bin/python

from xml.dom import minidom
import sys

doc = minidom.parse('./cdcat.xml')
def showCD(cd):
    for item in cd.childNodes:
        if not item.nodeType == item.TEXT_NODE:
            print '<p>', item.tagName, item.firstChild.data, '<p>'

print '<html><body>'
print 'CDs in my catalog: '
cds = doc.getElementsByTagName('cd')
for item in cds:
    showCD(item)

print '</body></html>'
```