

CMSC 433 – Programming Language Technologies and Paradigms Spring 2003

XML and Ant
May 8, 2003

Alphabet Soup

XSL JAX-RPC SOAP
DTD XML XPath
DOM HTML
JAXM JAXB W3C XSLT
UDDI CSS
XPointer SGML
JAXP XLink ebXML JAXR
XMLP SAX

2

Background

- Applications tend to represent data in proprietary internal formats
 - E.g., Word, Excel, Quicken
- Difficult to share data
 - Between different software versions
 - Between different applications
 - Between different platforms

3

Extensible Markup Language

- Goal: Provide a universal external format for application data
- Siméon and Wadler:
 - “[T]he essence of XML is this: the problem it solves is not hard, and it does not solve the problem well.”

4

HTML

- You’re probably familiar with HTML:

```
<html>
<head><title>CMSC 433</title></head>
<body><p>Hello, world!</p></body>
</html>
```
- Pretty good for display, but won’t really work as a file format

5

HTML Not Good for Data

- Not extensible
 - Fixed set of tags like <a>, <p>, <h4>, etc.
- No semantic structure
 - Divides document into headings, paragraphs, tables etc.
 - ...which doesn’t match a lot of file formats
 - E.g., spreadsheets

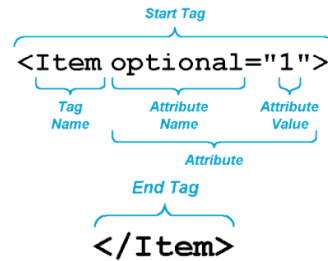
6

XML Example

```
<?xml version="1.0"?>
<reading>
  <book>
    <title>Thinking in Java</title>
    <author>Bruce Eckel</author>
  </book>
  <book>
    <title>Program Development in Java</title>
    <author>Barbara Liskov</author>
    <authorwith>John Guttag</authorwith>
  </book>
</reading>
```

7

Notation



from <http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml-p3.html>

8

Attributes

- Tags can have attributes
`<book binding="hardcover">...</book>`
- Attributes don't add any expressiveness
 - Could also have
`<hardcoverbook>...</hardcoverbook>`
 - But attrs sometimes make things easier

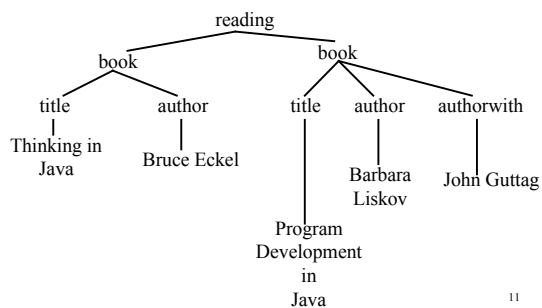
9

Syntactic Notes

- Begin with magic xml incantation
- Must have root element
- All elements need a closing tag
 - Shorthand:
 - Instead of `<Foo attr="value"></Foo>`
 - Can write `<Foo attr="value"/>`
- Elements must be properly nested

10

XML = Tree Structured Data



11

Parsing XML

- XML files are just text files
 - Can write with your favorite text editor
- Easy to parse XML into its tree structure
 - Can tell if a document is well-formed
 - But does it make sense?
 - Can't have two `<title>`s in a `<book>`
 - An `<author>` outside of a `<book>` doesn't make sense
 - Need to *validate* the document

12

Document Type Definition

- Defines a set of legal XML files

```
<!ELEMENT reading (book*)>
<!ELEMENT book (title, author, authorwith?)>
<!ATTLIST book binding CDATA "paperback">
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT authorwith (#PCDATA)>
```
- XML files can specify DTD

```
<?xml version="1.0"?>
<!DOCTYPE reading SYSTEM "reading.dtd">
```

13

XML Schema

- The replacement for DTDs

```
<?xml version="1.0"?>
<xsd:schema - start a schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  - Anything prefixed with xsd: is from ...w3.org... namespace
  targetNamespace="http://www.cs.umd.edu"
  - The tag's we're defining are for this namespace
  xmlns="http://www.cs.umd.edu">
  - The default namespace for tags
  ...
</xsd:schema>
```

14

XML Schema Example

```
<xsd:element name="reading">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="book" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="author" type="xsd:string"/>
            <xsd:element name="authorwith" type="xsd:string" minOccurs="0"/>
          </xsd:all></xsd:complexType></xsd:element></xsd:sequence>
      <xsd:attribute name="binding" type="xsd:string" fixed="paperback"/>
    </xsd:complexType>
  </xsd:element>
```

15

Advantages of Schemas

- Schemas are written in XML
 - So schemas can describe schemas
- Schemas have type information
 - Can introduce new named types

```
<xsd:simpleType name="ssnumber">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-\d{2}-\d{4}"/>
  </xsd:restriction></xsd:simpleType>
```

(Brill, CodeNotes for XML)

16

Using XML in Java

- Can use the Simple API for XML (SAX)
 - org.xml.sax.Parser
- ```
public interface ContentHandler {
 void startElement(String namespaceURI, String
 localName, String qName, Attributes attrs);
 void characters(char[] ch, int start,
 int length);
 void endElement(String namespaceURI, String
 localName, String qName);
 ...}

```

17

## SAX approach

- Essentially, a visitor pattern for XML documents
  - performs DFS traversal of XML tree

18

## Document Object Model

- Parse entire XML document
- Build tree
- Can examine the tree in any order

19

## SAX vs. DOM

- DOM requires entire document be in memory
- SAX keeps only a small part of document in memory
- DOM allows any traversal order
  - tree can be updated

20

## XSLT and XPATH

- grep / search replace tools for XML

21

## Ant

- Portable makefile system
  - Provides ways to compile, build, and distribute Java projects
  - Contrast to automake, autoconf, make
- Ant is written in Java
  - Available at <http://ant.apache.org>
- Ant build files are written in XML

22

## A Simple Example

```
<project name="p5" default="comp">
 <target name="comp">
 <javac srcdir=".">
 </target>
</project>
```

- Running ant with no args builds “comp”
- comp target runs javac on .java files in .
  - Recursively
  - Only if .class file older than .java file

23

## Tasks

- javac is an example of a task
  - Code that is executed
- Lots of tasks built-in to ant
  - javac, mail, echo, jar, ...
  - And you can write your own
- Supposed to be more portable than make
  - Doesn't use shell

24

## Dependencies

```
<project name="p5" default="stubs">
 <target name="comp">
 <javac srcdir=".">
 </target>
 <target name="stubs" depends="comp">
 <rmic classname="Portal" base="."/>
 <rmic classname="Portal.Peer" base="."/>
 </target>
</project>
```

- comp task must be executed before stubs
- Won't do same task more than once
  - But ant doesn't seem to smart about need for some tasks

25

## Properties

- `${propertyName}` replaced by property value when ant is run

```
<javac srcdir="${src}" destdir="${src}/obj"/>
```
- Can set
  - With `-D` on the command line
    - `ant -Dsrc="Source"`
  - `<property>` tag in ant file
    - `<property name="src" location="Source">`

26

## Other Features of Ant

- Ant can mangle your source code
  - Use `<filter>` or `<replace>`
  - Handy for substituting in things like version numbers, or disabling debugging
- Has nice ways to build up sets of files

```
<path id="base">
 <fileset dir="lib">
 <include name="**/*.jar"/>
 </fileset>
</path>
```

27

## Writing Your Own Tasks

```
import org.apache.tools.ant.*;
public class MyTask extends Task {
 private String msg;
 public void execute() {
 System.out.println(msg);
 }
 public void setMessage(String msg) {
 this.msg = msg;
 }
}
```

from ant.apache.org

28

## Calling Your Own Task

```
<project default="test">
 <taskdef name="mytask" classname="MyTask">
 <target name="test">
 <mytask message="Hello, world!"/>
 </target>
</project>
```

(Set CLASSPATH before running ant on example)

- How does ant find MyTask?

29

## Design Goals of XML

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

30

## Design Goals of XML (cont'd)

6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.