

Web Services

A Three Part Tutorial

Part I

Overview and XML

Why Study Web Services?

- Because we are really interested in Grid Services!
 - The OGSF specification says that a grid service is merely a web service that conforms to specific interface and behavior conventions that define how clients interact with that service.
 - In OO-speak – a Grid Service “is-a” Web Service.
- There is much more \$\$\$ wrapped up in Web Services and therefore lots of tools, programmers, experience, books, magazines, etc. - Great leverage!

Three Part Tutorial

- Web services tend to be complicated – partly due to the clever re-use of existing technologies, so we'll tackle it in bite sized chunks
 - Overview and XML
 - What the paradigm is and the underpinning of everything
 - Data Transfer
 - Too many ways to simply move data. Why?
 - Tomcat as one implementation
 - Creating and finding web services
 - Directories and such.

Language and O/S Warning

- Web Services are language neutral.
 - Implementations of various components exist in Java, C#, Perl, Python (Jython), C++, etc.
- However, my bias is toward Java and all examples will be presented using Java
- There are ways to use .Net or EJB and Web Services together – however, I have no interest in .Net and don't know enough about EJB so I can't provide much useful information.

Web Services

- Definition 1
 - A web service is any service that is available over the internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language.
- Definition 2
 - A web service is a piece of business logic, located somewhere on the internet, that is accessible through standard-based Internet protocols such as HTTP or SMTP.

Best Definition I Found

- A web service is a software system
 - identified by a URI
 - with public interfaces and bindings are defined and described using XML
 - whose definition can be discovered by other software systems.
 - and these other software systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet Protocols.

Characteristics of Web Services

XML

- XML based – everything is XML, from the data transferred, to the service description to the “make” files used to create the programs that implement the service.
 - You will probably find that you use XML even outside of the Web Services interface simply because it is quite useful!
 - These slides are created in StarOffice and the file they are stored in is (of course) XML based.

Characteristics of Web Services

Loosely Coupled

- There is a service and a client.
 - Location of the client and service may be variable
 - The language they were written in is un-important
 - The service need not exist when the client is written
 - The service may change between invocations
- More manageable and simpler integration albeit slower.
- Separate in your mind, the client and server. Use different programmers. Use CRC types of design.

Characteristics of Web Services

Coarse Grained

- Web Services should do a lot because the connectivity is fairly expensive
 - Computing a sin of an angle is probably a bad idea.
 - Asking the current load level of a computer is probably a bad idea
 - Asking for the optimal way to access a resource given a complex environment is a better candidate.
- This is not Corba and it isn't RMI. You should see services rather than atomic methods.

Characteristics of Web Services

Synchronous or Asynchronous

- This should be part of your design
 - With the coarse grained and loosely coupled nature of the system, some services may take a long time to respond – one should not sit waiting for a response
- It really requires the designer to think bigger than the immediate problem and in a fashion different from “ordinary” programming.
- This is another good opportunity for multiple person design sessions

Characteristics of Web Services Support for Document Exchange

- Since
 - XML is the mode of communication between clients and servers
 - XML can represent simple as well as complex data
 - All major office automation systems will be in XML within the next year
- Web Services provides a means of sharing large documents without the need to know what is inside of them.

Characteristics of Web Services

Self Describing

- If a system is loosely coupled, then there has to be a way for a client to find a service.
 - If we rely upon human documentation – we are lost
 - There has to be a way to automate the extraction of the essence of a service from the code that implements it.
 - There has to be an automated means of reading the essence of a service and then take advantage of it.

Characteristics of Web Services

Discoverable

- Again – to support the loosely coupled nature of web services, you cannot require that a program knows where the service is nor the exact means by which one invokes it!
- The use of the self-describing nature of web services with some simple directories accomplishes this.

Characteristics of Web Services

Support for RPC

- You could have a web service architecture without this. (e.g. mail, web pages, etc.) where the user defines the handshaking and the protocol for the client's use of the service
- Or, you can simply make a function call and be done with it.
- Web services permits both and in fact this is the typical way of accessing EJB or .Net

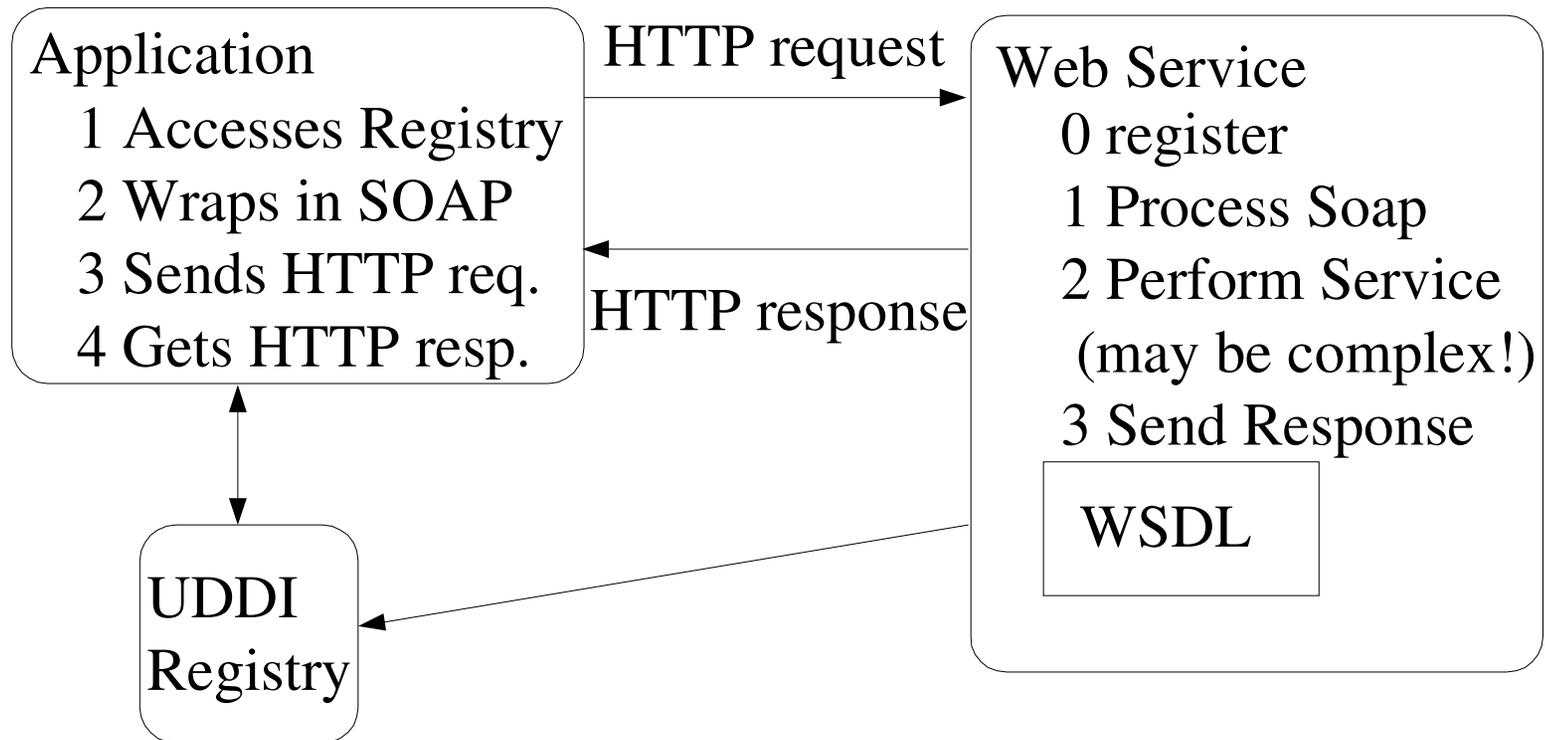
Summary

- Programming web services is not the same type of programming you usually do
 - Utmost degree of flexibility
 - Listening to Client designers (and vice-versa)
 - Understand that clients will change over time as will services
 - Design using interfaces
 - Design using Design Patters (more on this later!)

Some Terminology

- XML – a data format (think HTML)
- SOAP – a standard way to wrap up XML with an envelope etc. (Think mail and attachments)
- WSDL (Web Services Description Language)
 - Think about the best program library documentation you ever saw – one that permitted you to use the methods almost effortlessly.
- Universal Description, Discovery and Integration (UDDI)
 - Think of a dynamic name service on steroids.

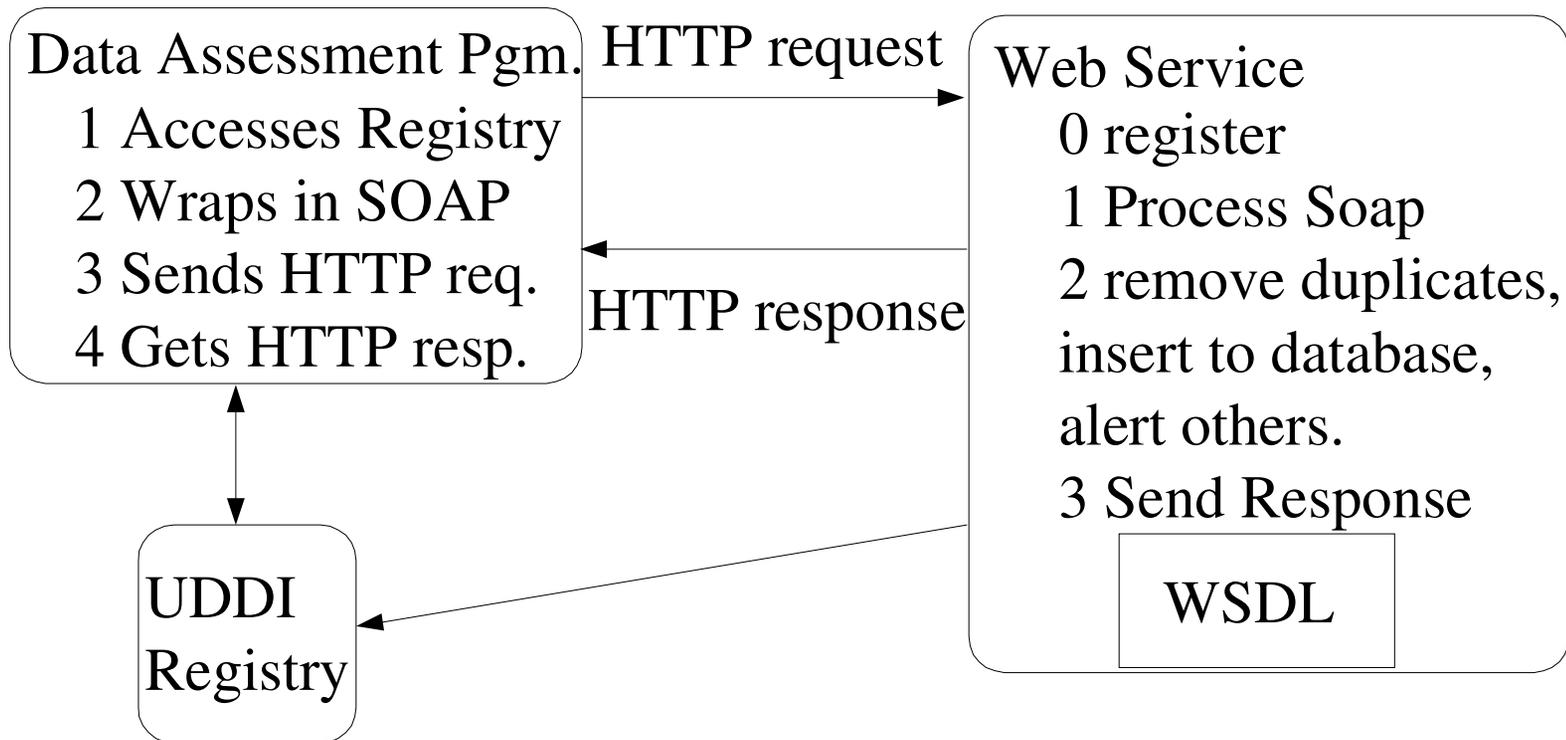
A Simple Architecture Example



Real Example

- ARM data collection.
 - Meta data is produced for all of the data – commentary, equipment reports, etc.
 - Meta data is filtered for duplicates, stored in the database, and transmitted to interested parties
 - Meta data is then attached to all data that is transmitted and is used to form a “color map”
- Experience shows that databases change and collection details change. Needs a clean break to permit independent development.

A Real Example



First Step – Why XML?

- The concept of “well formedness”
 - How many examples do you need of programs going out of control due to a missing “,” (or something) in its data file.
- The concept of “validity”
 - There is the idea that every program is a compiler, accepting as valid “sentences” properly structured input and (oh if it were only so!) rejecting improperly structured input. XML permits validity (syntax) checks
- These two facts alone make XML worth while independent of Web Services!

A Sample XML

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Document   : DesignPatterns.xml
  Created on : September 7, 2003, 12:52 PM
  Author    : Dave Stampf
  Description:
    Purpose of the document follows.
-->

<book isbn="0-201-633511-2">
  <title>Design Patterns</title>
  <subtitle>Elements of Reusable Object-Oriented Software</subtitle>
  <author>Eric Gamma</author>
  <author>Richard Helm</author>
  <author>Ralph Johnson</author>
  <author>John Vlissides</author>
  <publisher>Addison-Wesley Publishing Company</publisher>
  <copyright>1995</copyright>
  <hardcover />

</book>
```

Notes on the Simple XML

- It is wordy – the better to be read by humans and the machines don't mind.
- Much more strict than html
 - one “root” (otherwise, not well formed)
 - case matters
 - attributes must be quoted
 - all open elements need to be closed
- No formatting information – just info info (see XSLT!)
- Well-formedness

Minutia

- You can't use &, <, >, ', “
 - &
 - <
 - >
 - etc.
 - Or, use CDATA sections
 - <![CDATA[<anything your heart desires]]>

But...

- How to insist that a title be there?
- How to indicate that a subtitle is optional?
- How to indicate that the `<hardcover>` tag must be empty?
- How to indicate that there must be 1 or more authors?
- How to indicate that the isbn number have the right form?

Some Answers

- All but the last can be handled with a “DTD”
- Note – the format of the DTD was a blunder! It should have been in XML.
- DTD is very limited – looks like a quick hack – can't count, has trouble with namespaces.
- But, it is universally recognized while the more robust versions are slowly gaining acceptance.

Simple XML with DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "Book.dtd">
<!--
  Document   : DesignPatterns.xml
  Created on : September 7, 2003, 12:52 PM
  Author    : Dave Stampf
  Description:
    Purpose of the document follows.
-->

<book isbn="0-201-633511-2">
  <title>Design Patterns</title>
  <subtitle>Elements of Reusable Object-Oriented Software</subtitle>
  <author>Eric Gamma</author>
  <author>Richard Helm</author>
  <author>Ralph Johnson</author>
  <author>John Vlissides</author>
  <publisher>Addison-Wesley Publishing Company</publisher>
  <copyright>1995</copyright>
  <hardcover />

</book>
```

The Associated DTD

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<!--
```

An example how to use this DTD from your XML document:

```
<?xml version="1.0"?>
```

```
<!DOCTYPE book SYSTEM "Book.dtd">
```

```
<book>
```

```
...
```

```
</book>
```

```
-->
```

```
<!ELEMENT book ( title, subtitle?, author+, publisher, copyright, hardcover? ) >
```

```
<!ATTLIST book
```

```
  isbn CDATA #IMPLIED
```

```
>
```

```
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT subtitle (#PCDATA)>
```

```
<!ELEMENT author (#PCDATA)>
```

```
<!ELEMENT publisher (#PCDATA)>
```

```
<!ELEMENT copyright (#PCDATA)>
```

```
<!--
```

Tools

- What is so remarkable about having a “data language” is that you can develop tools similar to those that front end compilers to work with the data.
- Netbeans is one of many

Name Collisions

- I'm probably the 1,000,000th person to think of this book example, and many have probably used tags like book, title, etc. before.
- Also, “title” is overloaded with html
- So, “namespaces” are added (in a rather bizarre fashion)
 - Note – this is a young technology – support for namespaces is NOT universal yet!!!

How to Work with XML

- If the XML format existed (like CSV) without good libraries to verify, validate, input, manipulate and output, it would be of little value.
 - One benefit of XML is that it is accompanied by tons of support programs and libraries that
- There are 2 standard models for interacting with XML (and a number of nonstandard ones as well)
 - Simple API for XML (SAX)
 - Document Object Model (DOM)

Simple API for XML

- Scans the document, top to bottom and “calls-back” a function for everything interesting
 - start of document
 - start of tag
 - characters
 - end of tag
 - end of document
 - etc.
- Only useful for infinitely long documents or other special purposes.

How SAX Sees XML

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!--
```

```
Document : DesignPatterns.xml
```

```
Created on : September 7, 2003, 12:52 PM
```

```
Author : Dave Stampf
```

```
Description:
```

```
    Purpose of the document follows.
```

```
-->
```

```
[startDocument]
```

```
[startElement]
```

```
<book isbn="0-201-633511-2">
```

```
[startElement]
```

```
  <title>[characters]Design Patterns[endElement]</title>[endElement]
```

```
[startElement]
```

```
  <subtitle>[characters]Elements of Reusable Object-Oriented Software[endElement]</subtitle>
```

```
[startElement]
```

```
  <author>[characters]Eric Gamma[endElement]</author>
```

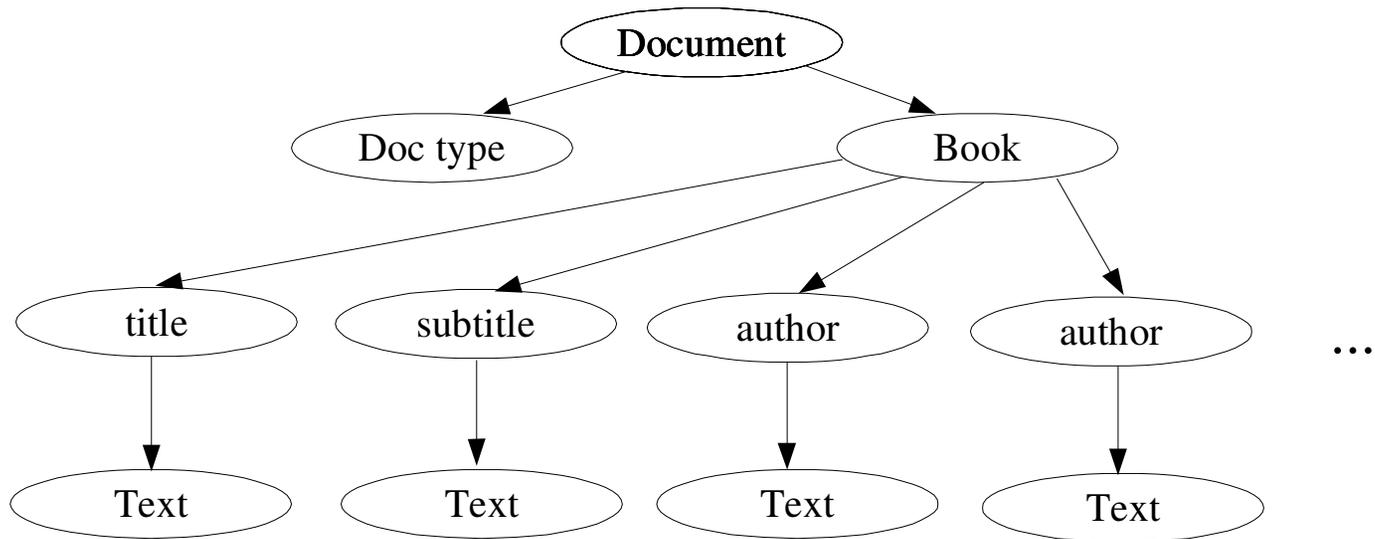
```
[startElement]
```

```
  <author>[characters]Richard Helm[endElement]</author>
```

DOM

- DOM is based on the fact that an XML file can be viewed as a “tree”.
- Most implementations of DOM use SAX to build the tree (smart design!)
- When you use DOM to process an XML file, you are returned a “Document” object and you are free to walk the tree yourself.

The Tree as Seen by DOM



As an OOProgrammer...

- Every oval “is-a” Node
 - Document Node “is-a” Node
 - Element Node (e.g. author) “is-a” Node
 - Text Node “is-a” Node
- Nodes have Nodelists (sequences of Nodes) beneath them.
- Nodes also have “values”
- Basically – everything you should have learned in Data Structures 1!

Time for some Programs!

- Lets set some tasks
 - Read in a Book xml file
 - Validate it
 - Extract the title, main author, publisher, and copyright date
 - output to standard output

Read and Validate

- There is no “magic” XML parser. Many companies and students write their own and either give them away or sell them. Some parsers come with other components (Tomcat)
- In addition, the XML spec says nothing about how one gets (instantiates) a parser
- Two major techniques
 - Hardwired
 - Factory generated – we'll work with this one

JAXP – the Acronyms Mount

- JAXP – Java API for XML Processing specifies how to get a parser and how the parser behaves.
- It provides a default parser
- If you have a better parser, you can still gain access to it through the means that JAXP provides and when a better one appears, swapping it in is a configuration issue, not a compiler issue
- Very good use of “Interfaces”

Read and Validate (and explain!)

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.InputSource;
import org.w3c.dom.*;

public class Book {

    public Book(Reader r) throws Exception {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setValidating(true);
        DocumentBuilder db = dbf.newDocumentBuilder();
        System.out.println(db.getClass());
        InputSource is = new InputSource(r);

        Document doc = db.parse(is);
        process(doc);
    }

    private void process(Document doc) {
        System.out.println("Have the document");
    }

    public static void main(String[] args) throws Exception{
        Reader r = new FileReader("C:/WebServicesTutorial/DesignPatternsWithDTD.xml");
        Book b = new Book(r);
    }
}
```

Factory Pattern

- JAXP uses the Factory Pattern to produce a parsers (document builder). This permits the parse actual parser to be determined at run time.
 - use the default
 - run with the definition
`javax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl`
 - or having a configuration file...
- The `DocumentBuilderFactory` is a Factory that produces “parsers” or `DocumentBuilders`
- The returned parser understands input sources

Messy?

- While the code is messy, switching parses
 - does not require any recompilation
 - can be done by applications asynchronously
 - can be done on the fly as well
- The factory pattern is used many times in the Java library to handle varying databases and window systems as well. (Its worth adding to your bag of tricks.)

So – Extracting the Data...

```
private void process(Document doc) {
    System.out.println("Have the document");
    Node book = doc.getDocumentElement();

    // find the title, first author, publisher and copyright date

    String title=null, author = null, pub=null, copy=null;
    NodeList nl = book.getChildNodes();
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        String s = n.getNodeName();
        if (s.equals("title")) {
            title = n.getFirstChild().getNodeValue();
        } else if (s.equals("author")) {
            if (author == null) {
                author = n.getFirstChild().getNodeValue();
            } else if (!author.endsWith(", et al")) {
                author += ", et al";
            }
        } else if (s.equals("publisher")) {
            pub = n.getFirstChild().getNodeValue();
        } else if (s.equals("copyright")) {
            copy = n.getFirstChild().getNodeValue();
        }
    }
    System.out.println(title + " by " + author + " published by " + pub + " in " + copy);
}
```

Method Patterns

- In the tree, everything is a Node
- Nodes understand:
 - getNodeName – a String
 - getNodeValue – a String
 - getNodeChildren – a List of Nodes
 - and other “editing” methods
 - append, remove, replace

A More Typical Application...

- This was a bit too special purpose
 - More typically, you only know that you will find nodes, but you don't know the real type.
 - It is easy to write a tree walking program. Everything in the tree is a Node and all nodes have Children.
- This was also a bit “un-Java” like. If you are buying into Java 100%, you should invest some time with JDOM. If you are multi-lingual, you should probably stick with DOM.

Modify an XML and Output

- Lets modify the XML by removing all but the first author and adding “et al” if needed.

Shorten the XML

```
private void process(Document doc) throws Exception {
    System.out.println("Have the document");
    Node book = doc.getDocumentElement();

    int numAuthors = 0;

    NodeList nl = book.getChildNodes();
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        String s = n.getNodeName();
        if (s.equals("author")) {
            numAuthors++;
            if (numAuthors == 1) continue;
            else if (numAuthors == 2) {
                n.getFirstChild().setNodeValue("et al");
            } else {
                book.removeChild(n);
            }
        }
    }

    // ok - output new XML - you've seen this pattern b4

    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer t = tf.newTransformer();
    t.transform(new DOMSource(doc), new StreamResult(System.out));
}
```

The Shortened XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document    : DesignPatterns.xml
  Created on  : September 7, 2003, 12:52 PM
  Author      : Dave Stampf
  Description:
    Purpose of the document follows.
--><book isbn="0-201-633511-2">
  <title>Design Patterns</title>
  <subtitle>Elements of Reusable Object-Oriented Software</subtitle>
  <author>Eric Gamma</author>
  <author>et al</author>

  <publisher>Addison-Wesley Publishing Company</publisher>
  <copyright>1995</copyright>
  <hardcover/>
</book>
```

XML's Other Tricks

- XSLT – a language to transform XML documents to something else (you saw the identity transformation in the last example)
- XML Schemas – should provide a much better syntax specification than DTD
- Database Connection – every major database provides data transformations to and from XML
- But, all of these are outside the realm of “web services”

Next Time

- SOAP
- Tomcat
- A simple request/response

Homework

- Find the proper XML libraries for your favorite language, get them installed and create a Hello World XML file. (If Java, install the “jwsdp”)
- Write a program to “walk the tree” and pretty-print out the tags. I'll show you mine next time
- Find any data format application you have now and re-phrase it in terms of XML. Write a simple output method and some useful access methods. Then, stand back.

Annotated Bibliography

Design Patterns – Elements of Object Oriented Software by Gamma, Helm, Johnson & Vlissides (Gang of 4). I don't know how you can make sense of OO software today without this book.

Java Web Services by Chappell & Jewell – an O'Reilly Book. Useful in conjunction with other books.

Web Services – Essentials by Cerami – an O'Reilly Book. As above – less of a Java spin, but still, plenty of Java.

Java Web Services in a Nutshell by Topley – an O'Reilly Book. Very good reference work. You need it.

Professional Java XML by Ahmed, et al. - a Wrox book. Very good collection of tutorials. Highly recommended.