CS433 – Technology Overview

Scott Selikoff
Cornell University
November 13, 2002

Outline

- I. Introduction
- II. Stored Procedures
- III. Java Beans
- IV. JSPs/Servlets
- V. JSPs vs. Servlets
- VI. XML Introduction
- VII. XSLT Introduction
- VIII. JSP vs. JSP/XML/XSLT Combo
- IX. Closing

I. Introduction

- The focus of this lecture is to get you familiar with each of the technologies you have at your disposal.
- Afterwards, you should have a general notion of when to use each technology and why.
- There is a myriad of information available on the web for each technology to help you out. If you are having trouble, try looking for help on the web first, before using the newsgroups or e-mailing a TA, since this is what you would have to do in practice.

II. Stored Procedures

- Your next project will be on SPs and will be out shortly.
- SPs offer many efficiency advantages because they execute on the server computer and are precompiled. This means there is no network delay when they are executed, nor is there any runtime compilation required.
- They can be written in a variety of languages and often DB's will differ greatly on the precise syntax of a SP.
- To use a SP, you must place it in the special SP area of the DB Management System.

Types of SPs

- All SPs have input and output variables.
- SQL SP: A prewritten SQL Statement as so:

```
CREATE PROCEDURE GetAllProducts
     AS
              SELECT *
              FROM PRODUCTS
              ORDER BY PID
```

- Java SP: This just a SP written in Java the same way you would write it in a JSP file.
- See HW for instructions and examples on how to call a SP

When to Prefer a SP

- When you need to do a complex query over and over again, a SP is a definite advantage.
- Many SPs can be accessed like database Views in that you can return them as a result set and merge them with other results sets
- Especially useful for setting up simple Database security quickly and easily. Just allow users access to specific SPs and no direct DB access.

III. Java Beans

- Used to Pass Information From an HTML form to a JSP/Servlet
- Often, you have a JSP generate an HTML form. The HTML then submits its information to a JSP (could be the same JSP) that parses the data using the Bean class
- Beans can be used as simple I/O for forms or contain more complex functionality

More on Java Beans

- First, you must compile the Servlet .java file to be a .class file.
- Then, place the class file in the directory:

  \WEB-INF\Classes
- To Use a Java Bean in a JSP/Servlet, include it in your list of header files.

Example Java Bean

```
public class LoginData {
        String username = "";
        String password = "";

        public void setUsername( String value ) {
                if (value != null)
                                user = value;
        } // setUsername()

        public void setPassword( String value ) {
                if (value != null)
                        userID = value;
        } // setPassword()


        public String getUserName() {
                return(username.toLowerCase());
        } // getUserName()

        public String getPassword() {
                return(password);
        } // getPassword()
}
```

Example Use of Java Bean

```
<form METHOD=POST ACTION="login.jsp">
        <INPUT TYPE = "text" NAME="User" SIZE="20" MAXLENGTH="20">
        <INPUT TYPE = "password" NAME="Pass" SIZE="20" MAXLENGTH="20">
        <INPUT TYPE = "submit" VALUE = "Login">
</form>
```

```
<%@page import="LoginData"%>

<jsp:useBean id="userParams" class="LoginData">
  <jsp:setProperty name="userParams"
    property="username"
    param="User"/>
  <jsp:setProperty name="userParams"
    property="password"
    param="Pass"/>
</jsp:useBean>

<%
        out.println(userParams.getUserName()+", "+userParams.getPassword())
%>
```

IV. JSPs/Servlets

- JSPs/Servlets: You should all know how to make a JSP from the last assignment.
- JSPs and Servlet are functionally identical. Anything you can do with a JSP, you can do with a Servlet and visa versa.
- The difference lies with how you call them.

JSP → Servlet Conversion

```
import javax.servlet.*;
import javax.servlet.http.*;

public class INSERT CLASS NAME HERE extends HttpServlet
{
    // initialize the servlet
    public void init(ServletConfig config) throws ServletException {
           super.init(config);
    } // Init()

    // INSERT CLASS FUNCTIONS HERE

    // when someone requests this servlet, it will return any outputed response
    public void doGet (HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException, java.net.MalformedURLException {
           try {
                       // The servlet returns HTML (could be XML instead!)
                       response.setContentType("text/html; charset=UTF-8");

                       // Output goes in the response stream.
                       PrintWriter out = response.getWriter();

                       /* INSERT CODE HERE */

           } catch (Exception e) {
                       PrintWriter out = response.getWriter();
                       out.println(e.toString());
           }
    } // doGet()
```

How To Call a Servlet

- First, you must compile the Servlet .java file to be a .class file.
- Then, place the class file in the directory:

  \WEB-INF\Classes
- Restart Tomcat
- Access the Servlet as:

  http://localhost/servlet/ClassName

V. JSPs vs. Servlets

- Many reasons for choosing one over the other, although most of them are based on issues of style.

Why/When to Prefer a JSP

- **Can use HTML code directly.**
- Often Easier to Use. Tomcat will recompile them for you.
- **Especially useful if your page is functionally simple or contains a great deal of HTML**
- Often preserves relative paths better than a Servlet does.

Why/When to Prefer a Servlet

- **Better if your code is functionally complex.**
- Better if the page is being used a stand-alone function.
- Servlets are safer for copy protection. You can give someone a Servlet class file to use without giving out your source code.

VI. XML Introduction

- XML is a data format, not a language
- The structure of XML is extremely simple since in its most general form, it has very little structure at all. You impose the structure to meet your needs.
- HTML is a dirty subset of XML. I use the term 'dirty' since strict HTML is not often practiced.

Example XML File

```
<Shopping Basket>
        <Product PID = "1">
                <PID>1</PID>
                <Title>Coca-Cola Soda</Title>
                <Company>Coca-Cola</Company>
                <Price>0.75</Price>
        </Product>
        <Product PID = "3">
                <Title>TV-Guide</Title>
                <PID>3</PID>
                <Price></Price>
                <Title>TV-Guide</Title>
        </Product>
        <Product/>
</Shopping Basket>
```

General XML Structure

- **All Open tags MUST have a closed tag or be empty.** The "/" at the end of the tag signifies an empty tag.
- **Most interpreters consider < and > characters to be reserved.**
- **Tags can contain data.** In the previous slide, the PID is listed in two places, but only one is actually necessary. Most interpreters would consider both implementations equivalent.
- Some interpreters also require that if you embed data in tags, you use double quotes around the data.

XML Structure Continued

- In general, XML is line-break and space insensitive, but **tags are often case sensitive**.
- In general, XML does not care about data consistency. For example: missing tags, data, or values is fine since XML does not have any structure except that which you impose.
- **Tags on the same level are usually order insensitive.** Meaning an items attributes can be listed in any order for any item.

VII. XSLT Introduction

- XSLT: An XML file converts XML → XML
- An XSLT only considers special XSLT tags as being in the XML Language. (all other XML tags are completely ignored)
- An XSLT can be applied to ANY XML file. If any information is unavailable in the XML file, the XSLT will simply either ignore it or return blank strings.
- Text or tags not handled by the XSLT directly is always outputted to the final XML file exactly as it appears in the XSLT file.

Example XSLT

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<HTML><BODY>
<TITLE>Search Results</TITLE>
XSLTs will ignore anything not starting with a xsl tag!!!

<xsl:template match="/Shopping Basket">
        <xsl:for-each select="/Product">
                You have <xsl:value-of select="string(./Title)"/> in your cart.<BR>
                The price of this item is: <xsl:value-of select="string(./Price)"/><BR>
                Pid # <xsl:value-of select="string(./Pid)"/><BR>
                <HR>
        </xsl:for-each>
</BODY></HTML>

</xsl:template>
</xsl:stylesheet>
```

XSLT Use

- **The most important strategy you will be using XSLTs for is converting XML data from your database into HTML.**
- Strategy: **First**, create a JSP that reads from the database and outputs XML. **Second**, create an HTML file of how you would like the page you are creating to look using made-up data.
- Once you like how the HTML file looks, **create the XSLT file** by adding an XSLT header to the HTML file and adding xsl-select statements where the data is. Finally, **have your original JSP call the XSLT to parse the XML data** and display the output to the user.

XSLT Use Continued

- XSLTs can have complex logic but should only contain the bare minimum. *xsl:select, xsl:for-each. and xsl:if* should be all you need. If you need a lot more, you probably should be doing it in the JSP.
- Conversions of Data types into specially formatted strings like Date/Time should be done in the JSP.

VIII. JSP/HTML vs. the JSP/XML/XSLT/HTML Combo

- One might ask why go through all this work of using JSP's, XMLs, XSLTs, and HTML if you can just output directly to HTML as you did in the JSP assignment?
- Certainly JSPs seem easy!
- And if the Combo is better, should we always use it?

Why to Prefer the Combo

- **It's a million times easier to update later on.** Anyone who's done a little of both styles on a major project can verify this.
- **The most crucial code is extremely sensitive to error.** The JSP is the single most complex component of the Combo. But because the JSP initially just produces XML, it is extremely easy to debug and verify correctness.
- **The least important crucial code is extremely insensitive to error.** The XSLT will work under a variety of circumstances and won't break your entire site if something small is missing.

When to Prefer the Combo

- When you need to read a lot of information from the database and you aren't doing anything too complex.
- Examples include: Search Results, User Home Pages, Product Information Pages, etc.
- In general, use it in READONLY environments with the exception of checking for the user to be logged in or for his username.

When NOT to Prefer the Combo

- When you processing a lot information and are not reading a lot from the database that needs to be displayed to the user.
- Examples include: order confirmation pages, new user forms, new product forms, login pages, etc.

IX. Closing

- This presentation has been designed with the intention of giving you a general overview of all the different technologies you have at your disposal.
- Because this is a project course, it is your responsibility to use resources like the web to learn more about each technology and the precise syntax needed to use them.
- Keep in mind: All these new technologies are wonderful but sometimes plain, old HTML will suffice. Example: Search Pages