Java Web MVC Frameworks: Background, Taxonomy, and Examples

M.Sc. Certificate Module

Ian F. Darwin
M Sc Candidate
Staffordshire University
https://darwinsys.com/contact?subject=jwf
Dr. Cathy French
Faculty Advisor

Abstract: This research presents some technological background on Java-based World Wide Web applications, develops a taxonomy of web application development frameworks that is more fine-grained than previous discussion, offers examples of each type, and provides implementation samples of a “hello world”-type Web Application implemented using several of the frameworks.

Biographical Note: Ian Darwin has worked in the area of software development for three decades. He began working with Java prior to the first public release and wrote one of the world’s first commercial Java developer training programs. He is the author of the widely-used *Java Cookbook* from O’Reilly & Associates (2001, revised 2004). He is currently working on a web-based database of MRI data for researchers at the Toronto Centre for Phenogenomics.

Acknowledgements: I should like to express my deep gratitude to my Supervisor at Staffordshire University School of Computing, Dr. Cathy L. French, for her enthusiasm and assistance.

In addition to journal resources, the web sites *http://www.husted.com*, *http://www.wafer.com*, and *Google.com* were used in the search for open source Java MVC Web Frameworks.

This research was not primarily supported by research funding. A lot of it was done in my personal time; I must therefore thank my wife and children for the time away.

Some of the evaluations, and the sample implementations using dbforms, JCorporate Expresso and SOFIA - perhaps 15-20 per cent of the total work described in this report - were carried out as part of work performed for the Toronto Centre for Phenogenomics (TCP, *http://www.phenogenomics.ca/*); the following applies to that portion of the work:

“This work is part of the Mouse Imaging Centre (MICe) at the Hospital for Sick Children and the University of Toronto. The infrastructure has been funded by the Canada Foundation for Innovation (CFI) and Ontario Innovation Trust (OIT). The research has been funded by an Ontario Research and Development Challenge Fund (ORDCF) grant to the Ontario Consortium for Small Animal Imaging (OCSAI).”

I would like to thank everyone I worked with at TCP for being so cooperative and supportive. Especial thanks to my supervisor John Cargill for encouraging my research and for careful reading of an earlier draft of this report.

Finally, I am grateful to the many developers whose continuing efforts to make open source software viable made this research possible. Special thanks to Craig McClanahan of the Apache Software Foundation for developing Struts concurrently with being the lead developer of version 4 of Tomcat, the web server and official reference implementation for the Servlet and JSP Specification.

1.0 Introduction

The state of Web Application Development is undergoing constant change. Within the sphere of Java-based web applications, there is an ongoing trend to reducing the use of raw Java coding in “JavaServer Pages”, fostering code re-use by placing Java code in re-usable components such as JavaBeans and Java Custom Tags. Many development frameworks - intended to automate this or that facet of web application development - are being developed, often with duplication and overlap. There is now a peer-reviewed journal dedicated to Web Technology Methodologies (Uden, 2002).

The research described in this paper had as its aim to catalog all Java-based Web Application Frameworks (approximately forty have been found so far). A new taxonomy was developed to assist in classifying the frameworks. A “sample” or “hello world” type application was developed in representative frameworks. A Summary Catalog will be maintained over time on the author’s web site.

The “sample” application comprises a simple entry form - a subset of the “CRUD” (Create, Read, Update and Delete) that is the basis of so much repetitive Web programming[1]. The example was developed using no framework at all, using the well-known Struts and JSF frameworks, and using half a dozen other frameworks drawn from the Catalog.

What is a “framework”? Among the many definitions I like these best: “In object-oriented systems, a set of classes that embodies an abstract design for solutions to a number of related problems.” (FOLDOC, 1995), and “A reusable, semi-complete application that can

1. Turau (2002) describes a framework for automating generation of CRUD forms; niggle, dbforms and SOFIA also provide similar facilities.

be specialized to produce custom applications" (Johnson, 1998). In common usage, a Framework is differentiated from a Toolkit in being generally more comprehensive, and by "who calls whom": an application generally starts off by invoking a Toolkit, whereas a Framework generally provides the "main program" and invokes the user's applications. This informal definition is supported by traditional software applications such as "The X Window System Toolkit" (MIT-XT, 1988). By this definition, the Java Applet and Servlet mechanisms could be classified as Frameworks (although this term is not commonly employed), while Java's client-side User Interface package, Swing, would be a Toolkit. This report concentrates on Frameworks used to build web applications under the Java Servlet/JSP APIs.

Another term of relevance is that of the "hollow API". A "hollow API" is one that provides an interface but not a particular implementation. The best-known example is Sun's JDBC package (java.sql), which specifies in some detail how relational database-related objects are to behave, but does not provide the actual behaviour. The external "JDBC Driver" for a particular database will provide implementations of the interfaces in the JDBC package. Other "hollow APIs" include the Java Message Service.

The effectiveness of web application development frameworks is of critical import to those building and maintaining electronic commerce web sites, because of issues like time-to-market, ease of updating the site, and customer perception of how well the site meets the customer's needs.

Creating and developing successful web applications requires a balanced partnership between web page designers (whose job traditions arise out of Commercial Artists and Designers), and software developers, programmers. Since very few practitioners can claim truly to be expert in both of these areas, one of the goals of a good web application framework is to allow each of these professions to operate without interfering with the good works of the other. An important feature, then, is separating (in the case of Java web applications) the Java developer's files from the web designer's files, so that changes made by one do not corrupt information provided by the other. Since JavaServer Pages (see below) are the dominant (and officially recommended) View technology for Java Web Applications, this separation of function somewhat paradoxically requires that we "get the Java out of the JavaServer Pages", a theme that will be visited several times throughout this report.

2.0 Background - Java/J2EE and Web Architecture

Java is well known as one of the leading programming languages today. The J2EE or Java 2, Enterprise Edition, is Sun's collection of Application Programmer Interfaces (APIs) for

developing server-side applications in Java[1]. Included are most of the major server-side APIs, including those listed in Table 1 on page 4; new ones are added periodically.

TABLE 1. Major J2EE APIs

Acronym	Expansion	Applicability
JAF	Java Activation Framework	Mime-type-based activation (similar to double-clicking on a file on the client)
EJB	Enterprise JavaBeans	Scalable distributed business process
JDBC	Unofficially Java DataBase Connectivity; officially not acronym	Accessing Relational Databases
JMS	Java Message Service	Access to Message-Oriented Middleware (MOM) such as IBM's MQ Series software
JTA	Java Transaction API	Transactions involving JDBC and/or EJB
Mail	Java Mail API	Sending e-mail notifications (also provides a client API for reading mail)
Servlets	Servlets	Processing for dynamic HTML
JSP	JavaServer Pages	Presentation for Dynamic HTML
WS	Web Services	Remote Procedure Call via HTTP and XML
XML	Extensible Markup Language (DOM & SAX)	Self-describing data

While each of these APIs has its place in developing distributed applications, this paper focusses on web applications, especially those based on Servlets and JSPs.

See also the Sun J2EE Documentation (Sun, 2004).

2.1 HTTP, the Lorries of the Web

There are many protocols that can be used on the Internet. At the lower level is a series of frame protocols, such as the Internet Protocol Version 4 with its transport protocols TCP (RFC793) and UDP (RFC768), or the newer IP Version 6. Most current internet application protocols are layered on top of IPV4 and TCP. One of the most-used of these is the standard Web protocol, HTTP or HyperText Transport Protocol (RFC2616), originated by Tim Berners-Lee at CERN (Berners-Lee, 2000).

HTTP is a request-response protocol; the client program (normally a web browser) sends a request, and gets a response. The request begins with a sub-type called a "method", like GET (for normal links) or POST (for web forms that should not be repeated). The request is sent in three parts:

- A request line, itself in three parts;
- Optional Header lines, in the same format as the familiar e-mail headers;

1. Indeed, Java's major competitor in the enterprise software realm is Microsoft ".Net" - which bills itself as a framework.

- (a required null line)
- Optionally a body (only with POST or other specialized methods).

The request line contains three space-separated fields: the method (usually GET or POST), the resource requested (a file name, a Servlet or JSP name, a directory, etc.,), and the highest HTTP version number understood by the browser.

The response is in the same general format as the request:

- A response line, itself in three parts;
- Header lines, in the same format as the familiar e-mail headers;
- (a required null line)
- Usually a body (typically HTML codes to be displayed by the browser).

The response line consists of the HTTP version being used by the server, a numeric response code, and a textual response code. The numeric codes are in five general categories:

- 100-199 Informational;
- 200-299 Success (200 is most common);
- 300-399 Redirections (used when a page is moved, and by some servers when the client requests a directory to re-direct the browser to its index page);
- 400-499 Client Error (404 is the most famous of these);
- 500-599 Server error - something wrong with the server or a component on it

Since both the request and the response are pure text (at least where an HTML file is being requested), it is possible to demonstrate the protocol in action using a program such as a `telnet` client. In this exercise I request a file named *index.jsp* from the default directory running on the local computer; the response has been partly elided to save space.

```
telnet localhost http
Trying::1...
Connected to localhost.
Escape character is '^]'.
GET /index.jsp HTTP/1.0

HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=DF35B53D5D1D951FA81B1216FD2F3426;
Path=/
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 2539
Date: Tue, 19 Aug 2003 03:19:31 GMT
Server: Apache Coyote/1.0
Connection: close
```

```
<!DOCTYPE html PUBLIC ...>
<html>
<head>
        <title>Welcome to my computer!</title>
        <meta http-equiv="Content-Type"
             content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>Welcome to my computer!</h1>
...
</body>
</html>
```

Note that I sent a GET request using HTTP/1.0, and the server responded in HTTP/1.1, exemplifying the upwards-compatibility capability build into the various revisions of the HTTP protocol.

Although the content of the response body can be in almost any format, the most common is HTML, the HyperText Markup Language, a subset of SGML (ISO8879) based loosely on a set of SGML tags used at CERN (Berners-Lee, 2000). The HTML and HTTP protocols have grown and matured together, as mentioned in the following section.

2.2 Web Applications

A Web Application is simply a program or series of programs whose primary user interface is a web browser and with which it communicates using the HTTP protocol described above. Well-known examples of commercial web applications include the eBay auction site, the Google and AltaVista web search engines, and the amazon.com consumer shopping site. There are many thousands of lesser-known web applications in use today.

A Web Application is not to be confused with a Web Service; Web Services are inter-program communication services rather than browser-to-server. Web Services promises to provide the first truly pan-platform distributed computing environment; previous attempts such as CORBA[1] (OMG, undated) have failed due either to complexity or to a significant vendor failing to accept the technology as a standard. Web Services are often implemented as Web Applications, but not necessarily. Web Services are supported by the J2EE, by Microsoft .Net, and by other languages as well (e.g., Perl's SOAP::Lite module).

1. CORBA, the Object Management Group's Common Object Request Broker Architecture, a platform-neutral, language-neutral Remote Procedure Call mechanism, implemented for Java, C++ and a dozen other languages. It ultimately failed in the market because of complexity and because Microsoft refused to sanction it.

The Web protocol was originally designed primarily as a user-friendly replacement for the FTP protocol (Berners-Lee, 2000). As such, HTTP had no requirement for interactivity or "web applications". Capabilities have been added in subsequent versions of HTTP and HTML. Version 2 of HTML RFC1866) added the `<form>` tag and a variety of input field types; at the same time, the NCSA Web Server (NCSA, 1995) was extended with a mechanism called "CGI", the Common Gateway Interface (NCSA, 1995). CGI allows almost any general-purpose programming or scripting language to be used as a back-end to the web server. CGI provides a common format for passing request parameters and sending a response. Here, for example, is a UNIX shell script which the author previously used to provide a "Contact form" facility (it has since been replaced, of course, by a Java servlet). Though this example is trivial, it has within it the two actions which make up the core of all web applications: *process the input data* in some way, and *generate an HTML response* to the user.

```
#!/bin/sh

DEST=contact@mycompany.com  # where to send the mail

# Part One: send the message to the system contact person
(
echo "Environment:"
echo "REMOTE_ADDR=${REMOTE_ADDR}"
echo "HTTP_REFERER=${HTTP_REFERER}"
echo "HTTP_USER_AGENT=${HTTP_USER_AGENT}"

echo ""
echo "HTTP Body:"
tr '&' '\n' | fmt
) | Mail -s "darwinsys contact form" ${DEST}

# Part Two: generate an HTML response to client's browser
echo "Content-type: text/html"
echo ""

echo "<html><head><title>Thanks</title></head>
echo "<body>"
echo "<H1>Thank you...</H1>"
echo "<p>Thanks for the input.
We will try get back to you shortly!"
echo "</p></body></html>"
```

The first part saves some "evidence" such as the Remote IP address, along with the content of the message being sent from the browser, via the Unix `mail` command, to a given contact person. In a realistic application, this part might represent, e.g., the order processing.

The second part uses the UNIX `echo` command to generate an HTML response. The standard output of the CGI script is connected (by the web server; the Web Application doesn't have to be concerned with this detail) to the network socket that leads back to the browser, thus the echo output (from `<html>` to `</html>`) is delivered back to the user's browser and is formatted to control what the user sees.

The simplest web applications consisted of only a single interaction: the client filled in a simple form and clicked the Submit button, and got the results. HTTP is intrinsically a "stateless" protocol: after one request-response cycle (see "HTTP, the Lorries of the Web" on page 4), the connection between client and server is torn down. To extend the reach of web functionality, *session management* was added to HTTP to track multiple request-response interactions between a given browser instance (hence, presumably, a given person) and a given server.

Although CGI has been all but abandoned for e-commerce sites due to the overhead of running a separate system-level process to handle each user request, it remains in use on personal and hobbyist sites. As well, CGI has been formative on all subsequent software used for developing web applications, including commercial offerings from Netscape (NSAPI) and Microsoft (ISAPI), and of course Servlets and JavaServer Pages.

2.3 The Servlet API

A servlet is a small segment of Java code run in a "servlet container". I shall somewhat circularly define a "servlet container" as "that part of the web server that processes servlets and JSPs." Servlets and their close kin JavaServer Pages take the place of CGIs in the Java 2 Enterprise Edition. Because invoking a method on a servlet that is already in memory is very much faster than invoking an external system-level process, and because Java servlets normally run "multi-threaded" (Sun Tutorial, 1995), servlets tend to be substantially faster than CGI-based processing.

In this section I discuss the API used in normal Servlet and JSP development, with some discussion of the operations of the "servlet container". The Java Servlet API has been relatively stable for several years, and the specification is currently at Version 2.3. This specification is developed by the Java Community Process (jcp.org), is shepherded by Sun Microsystems (*http://www.sun.com/*; Sun owns the Java trademark and provides the JCP project management), and the reference implementation is provided by the Apache Software Foundation's Jakarta Project, Tomcat group. The Servlet API provides, in package `javax.servlet`, a generic outline for Internet-based interactions between a client and a server (See Figure 1 on page 9). There can be Servlets for various protocols such as FTP, but the most interesting ones are for HTTP and, indeed, there is an entire sub-package in this API for such things, the package `javax.servlet.http`. While the original Servlet mechanism defined a method `service(ServletRequest, ServletResponse)` as the primary entry point to be called once for each user request, the HTTP servlet overloads this as `service(HttpServletRequest, HttpServletResponse)`, but also defines a series of methods such as `doGet(HttpServletRequest, HttpServletResponse)` - there is one method here for each HTTP method (see "HTTP, the Lorries of the Web" on page 4). In normal use, an HTTP servlet

writer will only implement `doGet()` or `doPost()` (or possibly both), leaving the higher-level HTTP servlet to sort out which HTTP method was used and therefore which HTTPServlet method should be invoked.

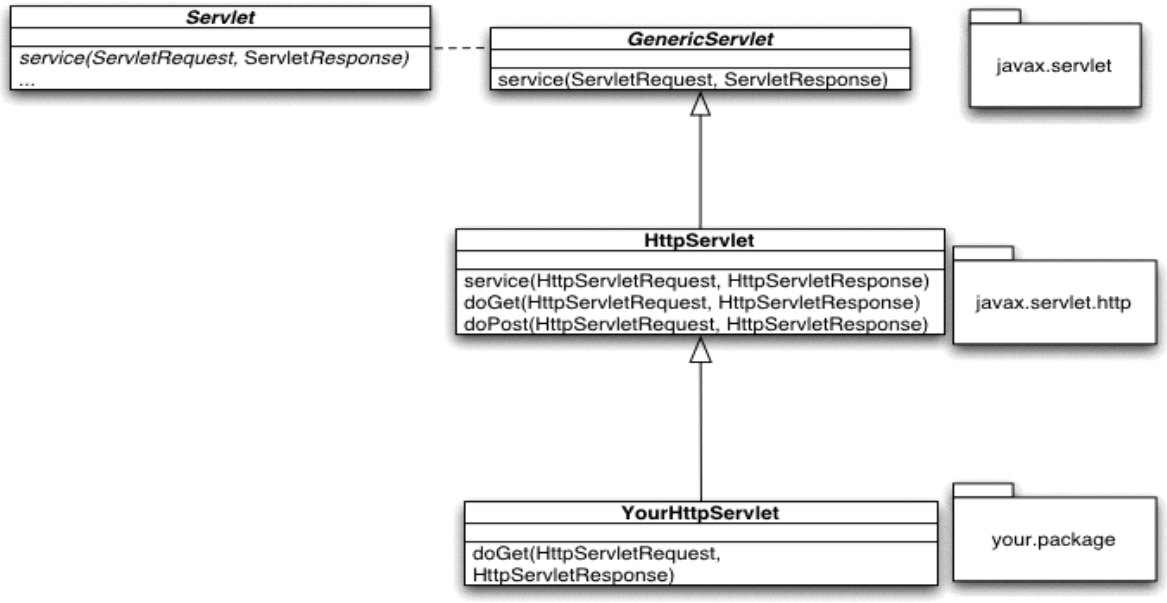


FIGURE 1. Servlet Class Hierarchy

Figure 2 on page 9 shows the interaction between the client browser and the server; the Servlet Container accumulates the state of the input request in an `HttpServletRequest` object and information on the return path to the browser in an `HttpServletResponse` object. Note in passing that the `HttpServletRequest` has methods directly analogous to the CGI variables discussed in “Web Applications” on page 6, such as `getRemoteAddr()`, and methods which access these values, such as `getHeader(“Referer”)` and `getHeader(“User-agent”)`.

After creating the `HttpServletRequest` and `HttpServletResponse`, the servlet container then sends these to the `service()` method of the Servlet corresponding to the URL.

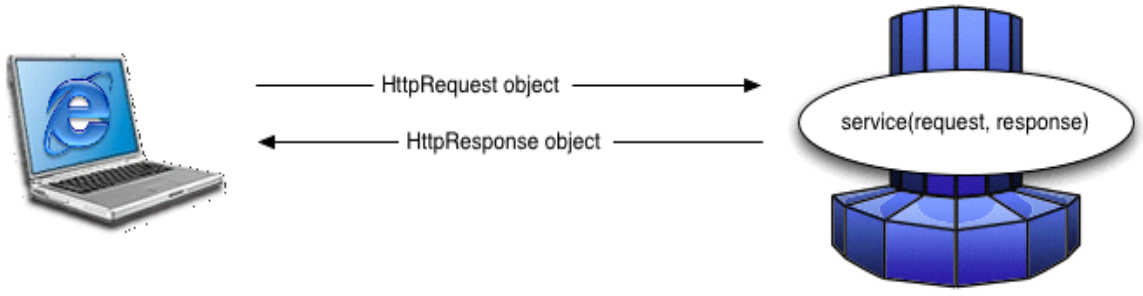


FIGURE 2. Browser-Server-Servlet Interaction

Servlet is an interface in package java.servlet, and GenericServlet is an abstract class which implements it. Servlet's most important method is service(), called with two arguments, a ServletRequest and a ServletResponse. The former represents the entire state of the inputs, such as the full URL, the request parameters, the remote IP address, and more. The ServletResponse contains all the state needed to generate a valid response and have it sent back to the user, including setting headers and providing a binary or text file handle for writing the response data. There are also `init()` and `destroy()` methods, for initialization and finalization respectively.

In the package javax.servlet.http, the class HttpServlet extends Servlet, and HttpServletRequest and HttpServletResponse extend Request and Response respectively. HttpServlet is the most important class for us; almost all Java web servlets, and the Servlets in the frameworks discussed here, extend this HttpServlet. There is, for example, a JSP servlet in most servlet containers which will subclass HttpServlet and add the functionality of compiling and invoking JavaServer pages. If a Servlet container is also a Web Server, as most are, it will also provide an HtmlServlet or WebPageServlet which simply outputs static HTML pages upon request.

In the HttpServlet class, the service method is specialized to look at the HTTP request method and farm it out to doGet(), doPost(), doPut(), etc., as appropriate. A minimal HTTP servlet is shown in Figure 3 on page 10.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServletMinimal extends HttpServlet{

  public void doGet(HttpServletRequest request,
       HttpServletResponse response) throws IOException {

       PrintWriter out = response.getWriter();

       response.setContentType("text/html");

       out.println("<h1>Hello from a Servlet</h1>");
       out.println("<p>Server time is now ");
       out.println(new Date());
  }
}
```

FIGURE 3. HelloServletMinimal.java

2.4 The JSP API

A JavaServer Page is an HTML (or XML[1]) text file that can contain a variety of Java constructs, including executable code, declarations, and printable expressions. One way of understanding a JSP is that it is "a servlet turned inside-out": a self-contained servlet will contain executable Java code, and the HTML output will be embedded in calls to out.println. A JSP, conversely, contains ordinary HTML, and any Java statements it needs will be embedded in special tags. Indeed, JSPs began as a more convenient means of writing Servlets. The *HelloServletMinimal* shown above can be rewritten in as little as the following *hellominimal.jsp*:

```
<h1>Hello from a Servlet</h1>
<p>Server time is now <% new java.util.Date() %>
```

The standard JSP mechanism is based on HTTP Servlets as described above. The JSP source file is stored in the normal web server hierarchy, along with static HTML pages, images, and other resources. When a URL validly referring to a JSP file is first encountered, the file is converted into a Java servlet extending `HttpJspBase`, using the obvious (and some not-so-obvious) transformations, such as surrounding text with quotes and embedding in an `out.printnln()` call, and embedding expressions (such as `new java.util.Date()`) in calls on `out.println()`. The resulting Java file is then compiled into a Java bytecode class file. The resulting servlet is from this point on largely indistinguishable from an HTTP servlet written by a developer.

The executable code of the JSP-based servlet runs in the `_jspService()` method, which is of course called from the `service()` method of the Servlet, and called with the same two arguments, an `HttpServletRequest` and an `HttpServletResponse`. The JSP page developer has these, and several other built-in server objects, such as the `PrintWriter` "out", the `ServletContext`, `PageContext`, and `Session` objects.

The developer may thus be tempted to write a lot of Java code into JSPs, but this is discouraged. Indeed, one of the minor themes of this paper, like the focus of evolving roles in JSP development, is "getting the Java out of the JavaServer Pages". Doing so is intended both to make the JSP more readable and to promote code re-use by moving Java code into components (JavaBeans), JSP Custom Tags, and model-view-controller frameworks.

2.4.1 Getting the Java Out I: JavaBeans

JavaBeans were first promulgated as a client-side API; they were historically intended to allow Java tools builders to create Rapid Application Development tools (the "drag and drop" part of an Integrated Development Environment), such that Java tools would become directly competitive with Microsoft's Visual Basic. While this goal has been achieved technically, it has not had the market acceptance that Sun had hoped - many peo-

1. Actually almost any text-based format can be used, including Microsoft RTF, Adobe Framemaker MML, and others.

ple still think first of using Microsoft Visual Basic for simple desktop applications. However, the JavaBeans pattern has turned out to be much more widely useful than first anticipated; this has led to their acceptance in diverse environments, including use as data model components in JavaServer pages.

JavaBeans are just classes that follow a few simple conventions:

- provide a no-argument constructor;
- implement the (empty) java.io.Serializable interface, and
- follow the standard setter/getter pattern for property accessors.

In the JavaServer Pages API, there are convenient methods for e.g., passing all the HTML forms parameter from the input request into a JavaBean instance: the JSP need only contain

```
<jsp:useBean id="myBean" class="com.darwinsys.Person"/>
<jsp:setProperty name="myBean" property="*"/>
```

to instantiate a `Person` object, storing a reference to it in a page-specific container object called the `PageContext`, *and* populate it with all HTML forms parameters whose name matches a setter method in the MyBean class. For example, if the HTML form contains `Your Name: <input type="text" name="name">` and the MyBean class has a setter method `public void setName(String);` then the `jsp:setProperty` call will cause the JSP mechanism to automatically call this method with the value the user type in the name field. But it will also handle setAddress(), setCity(), setHomePhone(), and so on, again assuming that the HTML form names match the Bean property names. It will even do certain type conversions automatically; if there is a text field named age and a setAge(int) method, the JSP mechanism will convert the string returned from the HTML parameter into an integer and then call setAge(). This works for all standard types (the eight built-in types and their wrapper classes).

JavaBeans are a good tool for separating out Java code, and they promote code re-use by packaging Java code into self-contained components. But they don't go far enough. It is still often necessary to invoke a Java Scriptlet (raw Java code) to invoke methods that do not fit the set/get pattern. Handling a collection (array or List) of Bean objects, for example, requires a Scriptlet containing a Java `for` loop. To go further in simplifying things, we have to resort to a mechanism that is more tightly integrated with the JSP processor.

2.4.2 Getting the Java Out II: JSP Custom Tags

JSP Custom Tags were introduced with the JSP 1.1 API specification, and allow the developer to write a small module which interacts with the JSP mechanism in clearly defined ways. The JSP then need contain only a reference to the tag. No Java scriptlet code is needed, even for a Collection. Consider the hypothetical ForEachUser iterator tag, for example:[1]

```
<myco:ForEachUser user="u">
```

```
Name: <jsp:getProperty name=”u” property=”fullName”>
</myco:ForEachUser>
```

which might print a list of user names such as:

```
Name: Ian Darwin
Name: Donald Duck
Name: Albert Einstein
Name: Henry Ford
```

All the code related to locating the user data, and storing it in the `PageContext`, is hidden in the class that implements the `ForEachUser` tag; typically no Java code at all is needed in the JSP that uses Custom Tags. As concrete example, I converted a JSP for displaying a list of club members from Scriptlet code to use JSP Tags provided by the Struts framework (see “Struts” on page 24). The iteration code changed from this:

```
<%  for (int i = 0; i < list.size(); i++) {
    out.println(i%2==0 ? "<tr>" : "<tr bgcolor='#ddd'>");
    String TD = "<td>", STD = "</td>";
    // For browsers that don't believe a cell is a cell
    // unless it has something in it...
    String NBSP = " ";
    Member m = (Member)list.get(i);
    String email = m.getEmail();
    if (!("".equals(email))) {
        email = "<a href='mailto:" + email + "'>" +
            email + "</a>";
     }
     out.println(
         TD + m.getName() +
         TD + email + NBSP +
         TD + m.getBusPhone() + NBSP +
         TD + m.getHomePhone() + NBSP +
         TD + m.getCellPhone() + NBSP +
         TD + m.getFaxNumber() + NBSP +
         "</tr>");
    }
 %>
```

to this:

```
<logic:iterate name="listaction.results" id="m" indexId="i">
```

1. I omit here the <%@taglib directive and the corresponding TLD file, both of which would be needed in a complete example; such matters may be gleaned from the formal J2EE documentation.

```
<% out.println(i.intValue()%2==0 ? "<tr>" :
     "<tr bgcolor='#ddd'>");
 %>
     <td><bean:write name="m" property="name"/></td>
     <td><logic:notEmpty name="m" property="email">
          <a href='mailto:<jsp:getProperty
               name="m" property="email"/>'>
          <jsp:getProperty name="m" property="email"/></a>
          </logic:notEmpty> 
        </td>
        <td><bean:write name="m" property="busPhone"/> 
        <td><bean:write name="m" property="homePhone"/> 
        <td><bean:write name="m" property="cellPhone"/> 
        <td><bean:write name="m" property="faxNumber"/> 
        </tr>
    </logic:iterate>
```

And all Java code was eliminated from the page except for the single conditional statement (`i%2==0`) used to print alternating table rows with a shaded background; I consider this an acceptable compromise.

The current JSP standard, 1.3, includes the Java Standard Template Library. JSTL provides many powerful features such as direct access to SQL databases, iterations, logic, and others that can be useful in writing sophisticated scripts in a JSP without resorting to actual Java coding.

2.5 The Trouble With JSPs

Despite the clear benefits of JavaBeans and JSP Custom Tags, there is nothing to prevent a developer from using arbitrary amounts of Java code directly in the JSP. This Java would be inscrutable to, and possibly subject to accidental (or even malicious) modification by the Web Page Designer working on the file. This is the main objection to the entire JSP mechanism raised by Jason Hunter (Hunter, 2000). In order to ensure maintainability, it is desirable to impose one or more Design Patterns, such as the Model-View-Controller pattern.

Design Patterns (GOF, 1995) are recurring patterns in development. “A design pattern systematically names, motivates and explains a general design that addresses a recurring design problem in object oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve[s] the problem. The solution is customized and implemented to solve the problem in a particular context.” (GOF, 1995, p. 360). In other words, using Design Patterns gives developers and managers a common language for describing how to implement a given section of code, and pro-

vides the developers with proven, pre-fabricated implementations. It's a little bit like building an electronic gadget by choosing parts from a catalog of integrated circuits rather than soldering transistors and diodes together individually; developers who don't use Design Patterns tend to "reinvent the flat tyre" rather than using proven solutions. For example, the *Factory* pattern (used throughout the standard Java API) is used when a method creates instances of a class that are tailored for the user (in Java, consider all the static methods called getInstance() - they are Factory methods). The developers of the early Java API and the GOF authors did not, as far as I can tell, directly borrow from each other; the notion of Design Patterns was "in the air" in computing research in the early 1990's (GOF, 1995, p. 356-7). Of the twenty or so patterns in the book, the one that most interests us is the Model-View-Controller pattern (which the GOF call Observer).

2.5.1 The Model-View-Controller Design Pattern

There are several methods of building Java-powered web applications. The best one seems to be a scheme adapted from the client-side Model-View-Controller or MVC design pattern (Burbeck, 1987). MVC originated to allow a client-side GUI application to be developed in a maintainable way giving heed to the need for multiple displays of common data. In Smalltalk-80, the world's second significant Object-Oriented language[1], the names View and Controller referred to actual classes that a developer would directly extend, while the Model could be anything from a String class object (for a simple text editor) up to a complex data structure.

MVC is a pervasive idea; indeed, the modern application convention of having a `View` menu derives directly from the notion of object-oriented software having multiple Views of a model (Hiltzik, 2000) For example, a Spreadsheet might have both a table view and a chart view on-screen side by side. A Presentation program might have both a text view and a slide sorter view showing concurrently. It is obviously not desirable for the user to have to click a menu item such as `View->Refresh` in order to have changes he has entered in one view be displayed correctly in the other. The MVC pattern, also known as the Observer pattern (GOF, 1995, p. 293), divides the code into three clearly-defined sections:

- The Model represents the data being modelled; the text of a word processor or slide show; the data in a spreadsheet.
- The View represents the visible display.
- The Controller is the code that responds to user input; there might be a Mouse controller, a Keyboard controller and a Menu Controller in a typical application. In a Java GUI implementation, these might be a series of Inner Classes delegating to a back-end, or they might be implemented as a single class that implements the appropriate Listener interfaces.

These three parts must be kept synchronized, of course. The Model must notify all registered Views whenever its data changes. The Controller must notify the Model when the user makes a request that involves changes to the data. The View must request current data

1. Simula-67 was the first.

from the Model when notified that the model has changed. The user, seeing the updated view, will make more changes, and so back to the Controller. This is summarized in Figure 4 on page 16.

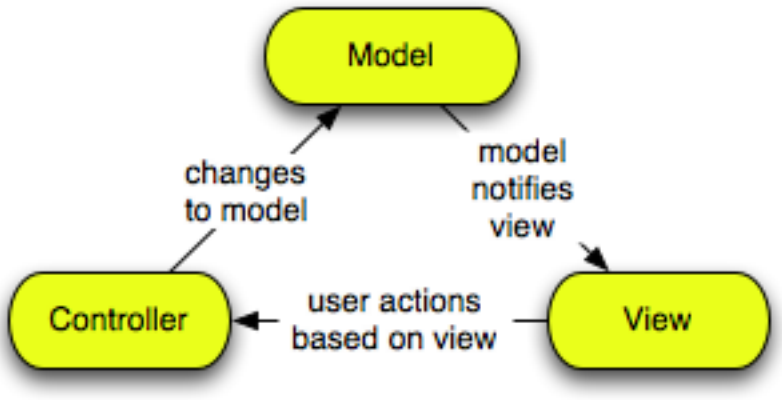


FIGURE 4. Model-View-Controller Pattern

I have written (Darwin, 1999) about the implementation of MVC in a Java GUI application. In the Java context, the Controllers are most often constructed as classes that implement sub-interfaces of the `java.util.EventListener` Interface. A well-known client-side example is `java.awt.event.ActionListener`, used as a Controller to handle user actions in buttons, menu items, text fields and other action-based components. The Controller registers (or is registered by the main part of the application) with the event source (e.g, the JButton) by calling the latter's `addActionListener` method. Using anonymous inner classes, the code for this might be:

```
JButton applyButton = new JButton("Apply");
applyButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
       model.applyChanges();
  }
});
```

While this might seem to have extra overhead compared to commingling the action handling code along with the data handling code, in a large application the benefits outweigh the slight increase in code. This is all the more so when there are, or might be, more than one View at work. There is a significant increase in maintainability when the views are isolated from each other; each is simpler to implement on its own, and there are fewer "hidden interactions" among the different parts of the code. If MVC is not used, the code for implementing multiple views over a single model quickly becomes an unmaintainable tangle of special cases and complex logic.

For these reasons, over time, MVC has become the definitive way of building large client-side GUI-based applications.

2.5.2 MVC on the Web?

How can the MVC pattern be applied on the web, where there is no direct coupling between the user and the server? Doing so should be expected to yield the same sort of benefits as on a client side: greater separation among different parts of the application, greater code re-use, etc. Typically the intent is that the View will be represented by an HTML page, generated by a View component such as a JSP. The Controller will be a Servlet that responds to user requests and forwards them to a Model component. The Model may be stored in JavaBeans or EJBs or any similar data object. Some of the frameworks discussed in the next section provide special intermediaries, e.g. Struts' "Action" classes, to be called from the Controller Servlet, do some processing, and pass control either to another Action or to a final View component (e.g., JSP). Thus the MVC pattern is simulated: the HTTP request invokes the Servlet Controller, the Actions update the Model, and the JSP causes a new View to be displayed in the user's browser. As will be seen in the next section, some Frameworks go further, and attempt to carry over the client-side Layout and Event Listener patterns from client-side GUI development into the realm of Web Application development. Leff (2001) describes a framework for building MVC web applications as a single running application and then partitioning them, offering more flexibility than dividing into Model, View and Controller from inception. Seshadri (1999) offers an early presentation on MVC on the web; the term "Model 2" stems from the description of Web-based MVC in Sun documentation preceding Seshadri's article.

2.5.3 Other Relevant Design Patterns

Other Design Patterns figure prominently in web applications.

Facade (GOF 1995) is a general term for the use of one object as a front-end for one or more others.

Data Access Object or DAO (Alur 2001) is the use of a Facade for data access. Rather than having the main application code be concerned with the details of the JDBC API (see Figure 1, "Major J2EE APIs," on page 4), a DAO is constructed which provides methods for fetching application-specific objects that represent rows in the database or other persistent entities. Done properly, this completely isolates the application from knowledge of the underlying persistence structure, making it easier to change either the application or the persistence without affecting the other. A DAO is used in the demonstration application in Section 4.3, "Servlet and JSP (MVC, not using a Framework)," on page 39).

A Singleton (GOF 1995) is a class of which at most one single instance may be present in a given application (more precisely in Java terms, a given "ClassLoader context"; unrelated Web Applications might unknowingly each instantiate their own instances of the same Singleton class). Java's Preferences (`java.util.prefs.Preferences`, new in JDK1.4) is a Singleton to provide centralized access to storing and retrieving users' preferences from multiple points within an application. The Singleton pattern is very easy to implement in Java:

```
public class SingletonDemo {
```

```
    private static SingletonDemo singleton;

    /** Prevents any other class from instantiating us. */
    private SingletonDemo() {
    }

    /** A static 'factory' method */
    public static SingletonDemo getInstance() {
        if (singleton == null)
            singleton = new SingletonDemo();
        return singleton;
    }
    // methods protected by singleton-ness here...
}
```

The application using this does not call the constructor, but instead calls `SingletonDemo.getInstance()`. Indeed, applications are unable to instantiate the class directly due to the only constructor being private; the only construction of this class that can possibly occur is in the first statement of getInstance(). An alternate implementation is to instantiate the Singleton in a static code block. In a full example, the constructor might check that it has only been called once (e.g., by testing that the variable `singleton` is still `null`) and throw an exception if not; the online source version of `oo/Singleton` does this, and features a JUnit (JUnit 2001) test case for ensuring that it works correctly.

Front Controller (Alur 2001) is the use of a Facade for dispatching from a Singleton object to the appropriate one of a list of other objects. Many of the MVC frameworks use a Front Controller, e.g. Struts' "Controller Servlet", to receive all user requests and dispatch them to the appropriate action handler.

Inversion of Control (IoC), also called Dependency Injection (Fowler, 2004) attempts to de-couple the various parts of an application, in our case the Model, View and Controller. Instead of having, say, the View code explicitly register itself with the View, an IoC framework allows the application to specify that a Model can accept one or more View registrations; these are declared as Java interfaces. Two leading IoC frameworks are PicoContainer (http://www.picocontainer.org/) which is not web-specific, and Spring (http://www.springframework.org/) which aims to be a "lightweight J2EE framework" (see "Spring Framework" on page 37).

3.0 The Research: Java MVC Web Frameworks Catalog

3.0.1 Brief Taxonomy of Web Frameworks

In researching web frameworks, I have found approximately 40 software projects that claim to simplify web development and implement the MVC design pattern. In attempting

to compare them, I found comparisons among some frameworks impossible due to radically different feature sets, so I came to realize that our current nomenclature- considering them all to be “MVC frameworks” - was inadequate. Starting with the observation that Struts, the best-known “MVC framework”, actually does not implement any of the Model code for you, was the starting point for developing a new taxonomy, more fine-grained than simply calling them all “MVC frameworks”. I now believe that all such frameworks can be categorized into one of these major categories:

TABLE 2. Taxonomy of Frameworks

Short form.	See	Description	Examples and notes
M	N/A	Model(data) only	OR layers: JDO, Castor, Hibernate, OHB, Entity EJB, etc. **Not detailed in this report.**
V	Section 3.2	View layers (template engines)	WebMacro,Velocity. Note 1.
C	Section 3.3	Controller only; rely on JSP or a V framework	ActionServlet, Weaver. Note 2.
VC	Section 3.4	View-Controller Frameworks	A front controller and a View layer. Tea Servlet; Struts
MC	Section 3.5	Model & Controller only	Niggle, dbforms. Note 2.
MVC	Section 3.6	Full frameworks	Turbine, Expresso
MVCO	Section 3.7	Full framework + OO (HTML Components)	SOFIA, JSF, webonswing. Note 3.
Meta	Section 3.8	Full MVC or MVCO + more	Spring, Keel

1. Template Engines (Velocity, Web Macro) - these provide content substitution within pre-defined “templates”, operating primarily at the View layer level;
2. Template-agnostic Frameworks - frameworks which provide some MVC functionality themselves, but allow use of one or more of the Template engines instead of or in addition to JavaServer Pages for the view components;
3. OO Frameworks - certainly most Java programs are Object-oriented; OO Frameworks in this context go further and use component-based development for “GUI” layout of the resultant web pages, or for “Listener”-style event modelling, or both.

   Most OO frameworks are also full MVC frameworks, with the prominent exception of Java Server Faces (JSF). I did not wish to complicate the taxonomy by further subdividing based on this one case.

The frameworks are discussed under these headings in the following sections; an alphabetical list of all known frameworks appears at the very end of this document.

3.0.2 Observation: You Cannot Escape Programming

Templates do attempt to separate programming from display, however, in general the view must have some programming in it, even if not in Java. Consider the problem of displaying a “simple” table, such as a “sign-up” page for assignments at a meeting. Assume the User

Requirements for this page to be something akin to the following (taken from a real example for one of my consulting clients):

To print e.g., the role of meeting Chairperson:

- If the role of Chairperson for Tuesday's meeting is assigned to Robin, then print Robin's name;
- Else if the user logged in to the web site is in charge of meeting assignments for this particular meeting, then display the list of persons eligible to take on this role;
- Else if the user is logged in as a "member", display a "Sign up" button;
- Failing all of the above (i.e, a guest user), just print the word "Unassigned".

Repeat the above algorithm for each meeting role.

The only reasonable way to express all of that is in traditional if-else logic, so any really capable View layer that abolishes Java must reinvent at least a goodly part of it.

3.1 M-type (Model-only) Toolkits

Other than low-level coding of JDBC and SQL statements, how do web application frameworks store or persist their data into permanent storage? Frameworks that do not provide their own Model layer tend to utilize one of the following data access technologies or toolkits. Some of these are "O/R Mapping" (Object-Relational Mapping) in that they provide a (bidirectional) mapping between Objects and a Relational Database.

- JavaBeans - also known as POJOs (Plain Old Java Objects), these simply store properties with paired set/get methods. No data access in their own right, but often used to transfer or transport data into one of the other layers.
- JDO and Hibernate are OR mapping projects similar in their general approach. They provide POJO-like semantics with DAO functionality. I have described and given examples of JDO in (Darwin, 2004) and Hibernate in (Darwin, 2004a) JDO is a "hollow API" for which multiple implementations are available; Hibernate is both a specification and the only implementation of that specification. JDO resources can be found at *http://www.jdocentral.com/*, while Hibernate can be found at *http://www.hibernate.org/*.
- OJB (Object-Java Bridge) from the Apache Foundation provides two OR-mapping APIs, one for JDO and one that is compliant with the Object Management Group specification.
- Castor is yet another OR-mapping, and can be found at http://www.castor.org/.
- Entity EJB is part of the J2EE framework (see "J2EE" in Section 2.0).

These are not discussed at length in this report.

3.2 V (View) Frameworks - Template Engines

Template Engines to not necessarily implement a full Model-View-Controller pattern. They tend to implement primarily the View portion, leaving the Model and Controller to others. For standalone use as Web applications, however, they must provide - or the developer using them must provide - at least a Servlet. As Freemarker says of itself (on *http://freemarker.sourceforge.net/* as of the time of writing): “FreeMarker is not a Web application framework. It is suitable for [use as] a component in a Web application framework...”.

Like JSPs, most of the Template engines are not restricted to HTML but can output XML, RTF, and other textual formats.

3.2.1 WebMacro

Among the oldest of the Java Servlet macro languages is WebMacro, a simple scripting language. The developer writes a Java class extending WMServlet, and the web designer writes a Template file. The servlet creates a Context object (part of the WebMacro API, not the Servlet/JSP API) which contains “variables” (Java objects). The template refers to these using the dollar sign to denote a variable and the C/C++/Java “dot” notation for attributes within a variable. The template might, for example, refer to

```
$club.member.name
```

which refers to “name” in attribute “member” in variable “club”. Despite claiming to avoid programming in the View layer, it seems inescapable, and WebMacro provides a large collection of syntax. See *http://www.webmacro.org/WMScriptDirectives* and *http://www.webmacro.org/WMScriptOperators* for a summary of the “Directives” - commands like “if” and “foreach” - and “Operators” - many of them identical to their Java counterparts.

WebMacro has been used in a number of web site projects, can be used with the Jakarta Turbine project, and has been interfaced to Jakarta Struts. Its flagship user is the AltaVista search engine (*http://www.altavista.com*), long one of the top-ten sites on the Internet

3.2.2 FreeMarker

FreeMarker is similar to WebMacro. It is less widely known and used, but this may change as it is the default View layer for Niggle (see “Niggle” on page 28).

An important difference is that WebMacro allows any Java object to be put out for display on the page, whilst FreeMarker requires that the objects being “exposed” for display implement one of several interfaces that are part of FreeMarker; this provides a level of type-safeness that makes it more likely that objects will display correctly, when contrasted with the default use of the Java language `toString()` method in WebMacro.

FreeMarker has its own simple programming language (see “Observation: You Cannot Escape Programming” on page 19). The construct ‘<#’ is used to introduce a programming command. A simple page to display a bean (both via “toString” and by iterating over

a property named “array” in the bean) previously set in the Servlet PageContext by a servlet might look something like the following:

```
<fm:template>
<#assign mybean = page.mybean>

<p>Here is the bean: ${mybean.string}
<ul>
<#list mybean.array as item>
  <li>${item}</li>
</#list>
</ul>
</fm:template>
```

3.2.3 Velocity

A moderately well-known view layer, Velocity has the imprimatur of the Apache Jakarta project. Velocity is a small, simple view layer that has no strong dependencies on the Servlet/JSP mechanism - it can be used quite outside of web applications. However it is best known as a template engine for providing a view of Servlet output. Whether used as an Application or as a Servlet view, the general operation is that the Java code places one or more objects into a Velocity Context, and a “template” file contains references to its name (preceded by a dollar sign), resulting in merge substitution in the output. A VelocityContext behaves like a java.util.Dictionary, with put(name, value) and get(name) methods. Typically the former are invoked by the Java code and that latter invoked implicitly by the template.

Velocity also provides a simple programming language (see “Observation: You Cannot Escape Programming” on page 19). Velocity uses commands beginning with “#” in an attempt to avoid conflicts with e.g., Java code in a JSP scriptlet.

Assuming that a previous Login Servlet invocation might have set the value of “loggedinusername” in the Velocity context, a Velocity Template for a “welcome page” might look like the following:

```
#if $(loggedinusername)
<p>Welcome back, $(loggedinusername)</p>
#else
<p>I see you are not logged in. Won’t you please be nice
and either <a href=”login.html”>login</a> or
<a href=”register.html”>register</a>?
#end
```

Velocity is not a full MVC framework, but it is used as the View portion of some full frameworks, such as Turbine (see “Turbine” on page 29).

3.3 C (Controller-only) Frameworks

3.3.1 ActionServlet

ActionServlet is a small Front Controller servlet that forwards to Java components; in this sense it is similar to Struts. It differs in being able to forward to a variety of different types of components; the actions “POJOs” (“Plain Old Java Objects”, that is, they are not required to extend a given type nor expose any given method).The class and method to invoke for each web page are specified in an XML configuration file, as are the pages to display on success and on any of a list of Exception types. ActionServlet does not provide JSP custom tags and, indeed, does not recommend use of JSPs, favoring use of either WebMacro or Velocity. It is a small, reasonably easy-to-learn package and is accompanied by adequate documentation and a dozen or so examples of usage.

Like many of the other frameworks, an ActionServlet template uses # for programming, $ for retrieving fields from objects.

See also *http://www.actionframework.org/*.

3.3.2 Weaver

Weave is another small Front Controller; it works with JSP, including use of the JSP2 Expression Language and JSTL tags. Weaver pages are written in pure XML, which maps to HTML tags and JSP tags. Or, as the web site puts it, Weaver “exploits your existing knowledge of the JSP Expression Language (EL) and of JSTL via a consistent XML tag set.”

Weaver provides no model of its own, but a simple Java interface is used to hook any Model functionality that you like into Weaver.

JSP is the default View layer, but others may be substituted.

See also: *http://www.oldlight.com/weaver/*.

3.4 VC (View-Controller) Frameworks

3.4.1 Tea Servlet

Another interesting template environment is the Tea Servlet, originated by Walt Disney Internet Group (*http://www.dig.com*). Unlike the simpler template frameworks, Tea is comprised of the Tea language (including a bytecode compiler) used for writing Tea Templates, the Tea servlet itself, and Kettle, an IDE for generating Templates.

The Tea language is small, simple, and typesafe. In the name of enforcing the separation of View and Processing, it does not allow any direct access to Java APIs, including the Servlet API. Tea compiles simple expressions into Java bytecode.

Each page must have two classes written for it, an Application class and a Context class. The Application class is mostly boilerplate but includes the `init` and `destroy` functionality of Servlets. The Context class is not, as the name suggests, the page's interface to the Servlet Context, but is instead the page's interface to the Request object.

The developer will write in the Context class code for the most trivial methods, e.g., what most Servlets/JSPs would write as request.getParameter("name") becomes getName() in the Template, but then must be written as something like

```
getName() { return request.getParamer("name"); }
```

in the Context class. I would suggest that this process could be automated using introspection on the request object, similar to the DynaActionForm feature of Struts, which avoids the need to write simple data classes.

Overall, the Tea Servlet provides very complete separation of view templates from Java. The Tea language doesn't allow access to Java, only to functions provided in the runtime or functions provided by the developer. This means that the template designer ('web designer") has no access at all to Java, and cannot mess things up for you.

Another nice aspect is the administrative servlet, which provides the ability to reload the Tea servlet configuration files, without unduly perturbing the surrounding Servlet Engine or Web Server.

3.4.2 Struts

Struts is one of the best known Java Web Application frameworks at this time; a quick look on Amazon.com revealed a dozen textbooks with Struts in the title. Struts focusses most of its effort on the Controller portion of the MVC triad; the View is usually JSP although Velocity may be substituted, and the developer provides the Model using any M-type technology that is appropriate. Struts excels at making the Controller simple to use and very maintainable. Bishop *et al*. (2002) discuss an advanced application build upon Struts.

For each web form or other action, the user provides an Action class, and lists the action in the struts-config.xml file. This configuration file is the organizational centre of a Struts application, as it specifies the mappings among URL paths, Data Holders, Action classes, and display pages. The configuration file is read by the Struts ActionServlet, a singular class which acts as a Facade (GOF 1995, p 185) or Front Controller (Alur 2001) to all the Action classes, using information in the configuration to map URLs to the correct Action.

The Data Holder classes or Form Beans are, like JavaBeans when these are used with a <jsp:useBean...><jsp:setProperty property="*"...>, intended to accumulate the HTML forms parameters. These could be considered part of the Model, but Struts experts tend to consider them as temporary holder classes for the benefit of the Controller. They must subclass the base class ActionForm, so "ordinary" JavaBeans are not eligible. The key point is that one can write a Java class to serve as an ActionForm but, often, one does not have to. The DynaActionForm class can be created just by specifying the forms parameters as

Form Bean fields in XML, as in this extract from the Struts Implementation in my Catalog.

```
<struts-config>

<form-beans>
   <!-- Used in the Demo, for inserting a Person -->
   <form-bean name="addPersonDynaForm"
     type="org.apache.struts.action.DynaActionForm">
     <form-property name="firstName" type="java.lang.String"/>
     <form-property name="lastName" type="java.lang.String"/>
     <form-property name="address1" type="java.lang.String"/>
     <form-property name="address2" type="java.lang.String"/>
     <form-property name="city" type="java.lang.String"/>
     <form-property name="province" type="java.lang.String"/>
     <form-property name="postcode" type="java.lang.String"/>
     <form-property name="country" type="java.lang.String"/>
     <form-property name="email" type="java.lang.String"/>
    </form-bean>
</form-beans>
```

Struts then uses the Introspection API (`java.lang.Class`) to create an ActionForm class dynamically. The HTML forms parameters are copied from the HTML form into the ActionForm bean, and this is passed into the execute() method of the Action class. This works best when the HTML form is created using the Struts “JSP Custom Tags”, so the Struts mechanism knows which form fields are associated with which HTML elements.

The Action class is the page-specific portion of the Controller. It responds to the forms parameters and any other arguments in the request, performs some action (such as inserting a person’s information into, or getting a list of items from, a database), and then forwards control to a View component. A nice feature is that Actions and Views are decoupled; the struts-config file specifies named mappings, such as that the “displayUsers” forwarding might map to “listusers.jsp”. Thus if you change e.g., the names of the JSP files, you do not have to recompile the Actions, but simply edit the “forward” mappings in the struts-config file.

Struts Form Beans support the concept of Validation. If, after loading all the parameters, the Form Bean considers itself invalid (as evidenced by the return from its validate() method if it is a custom bean, or by use of an external, XML-scripted Validator Plug-In), then Struts will simply forward back to the input form listed in the Struts configuration. Since the HTML form is being generated by Struts JSP Custom Tags, any fields (other than passwords!) that the user entered will automatically be remembered (from the Form bean) and put back into the HTML form, saving the user from having to re-enter the ones that are valid (or saving the developer from having to write code to re-populate the form). This same mechanism can be used to write an “edit”-type form: just fetch the Form Bean

contents from the database, attach it to the HttpServletRequest object, and forward to the input form. Presto! Instant editing page. Almost: you still have to write an Action to put the changed data back into the database, but this is simplified by the near-universal use of the Data Accessor pattern (Alur et. al., 2001). In the source for the Framework Catalog web application (see Figure 5.0, “The Frameworks Catalog Online,” on page 50), this can be seen in the AddFramework form and the EditFramework form. In all my examples the View components are JavaServer pages, but the Struts documentation claims that other View technologies such as Velocity can be used.

Here is part of a Struts-config file from one of my web sites. It shows both a dynamic form-bean (no Java code required) and a conventional JavaBean used as a Form Bean (so called because the framework will populate either type from the contents of a form). It also demonstrates the “global forwards” which make maintenance of a larger site easier. And it shows one example of an Action Mapping; this is the real tie-in from the URL namespace to the action handling code in the Actions.

```
<struts-config>

  <form-beans>
        <!-- DynaFormBean for person table -->
        <form-bean name="person"
        type="org.apache.struts.validator.DynaValidatorForm">
          <form-property name="id" type="java.lang.String"/>
          <form-property name="email" type="java.lang.String"/>
              <!-- many more String fields ->
        </form-bean>

        <!-- Java-based form bean -->
        <form-bean name="product" type="jabacart.Product">
        </form-bean>
  </form-beans>

  <global-forwards type="org.apache.struts.action.ActionForward">
        <forward name="showcart" path="/showcart.jsp"
              redirect="false"/>
        <forward name="checkoutstart" path="/sales/checkoutstart.do"
              redirect="false"/>
        <forward name="checkoutfinish"
              path="/sales/checkoutfinish.do" redirect="false"/>
        <forward name="cartisempty" path="/cartisempty.jsp"
              redirect="false"/>
  </global-forwards>

  <action-mappings>
        <!-- Get a listing of products -->
        <action path="/catalog"
              input="/"
```

```
                parameter="searchType"
                type="actions.CatalogAction">
                <forward name="success" path="/catalog.jsp"/>
        </action>
    </action-mappings>

</struts-config>
```

Struts provides convenient features for internationalization and validation; these are discussed in the Implementation section. Internationalization provides the means for users to view a site in their choice of human language. A Validation service makes the site easier for people to use (client-side validation in particular, since it stops errors before you can submit the form, saving the time of a round-trip to the server for simple errors).

There are a number of add-on tools for Struts. James Holmes (*http://www.jamesh-olmes.com*) provides the Struts Console, a free tool for editing the XML configuration files required for Struts.

Exadel Struts Studio is, as the name implies, a sophisticated front-end development tool for building Struts applications. There is a free “Community” edition, a “Standard” edition and a “Professional” edition available for purchase. Pricing and other details may be obtained from *http://www.exadel.com/products/strutsstudio.htm*, which also contains a link to a detailed comparison of various versions (a legacy Professional edition, and the current Standard and Professional editions).

While it does not provide WYSIWYG development of JSPs, it does provide a convenient work-flow-like view of the “forwards” within a Struts project. The Pro edition can import and work with an existing Struts project.

What is impressive about Struts Studio is that it does not provide any proprietary extentions. With the exception of the struts-config.exadel file that stores the state of the image file and the (project-name)_exadel_project.xml (both files are XML text, so can be stored as text in a source code repository), there does not appear to be anything non-standard that is inserted into the Struts project.

Struts Studio would appear to be a good tool for building Struts-based web applications.

Despite progress in other frameworks, Struts remains very popular among developers, and has become a resume/C.V. requirement for many employment opportunities in the J2EE area.

See Also: “Implementation using Struts MVC Framework” on page 44.

3.5 MC (Model-Controller) View-Agnostic Frameworks

Several frameworks are relatively agnostic about their View layer: some can use one or most of the Template engines in “V (View) Frameworks - Template Engines” on page 21 instead of or in addition to JavaServer Pages.

3.5.1 Niggle

Niggle is a pleasant surprise to most developers in the amount of work it does for you. Niggle aims to be a very-high-level processor for building database-backed web sites. Once the database schema have been described in an XML format and "Niggle HTML" (*.nhtml*) templates set up, the complete Java code[1] to retrieve and validate all the HTML fields, and insert them into a database, and show the view page acknowledging the insertion, is shown here (this is excerpted from the file *niggle/MyNiggleServletInteraction* in the code sections):

```
/** The AddPerson handler, invoked for action="addperson" */
public void execAddPerson() throws IOException {
      Record person = getNewRecord("people");
      fillInFields(person);
      OreoDataSource people = getDataSource("peopledb");
      people.insert(person);
      // Acknowledge addition into DB back to the user.
      page = getPage("niggle/thanks.nhtml");
      page.expose("title", "Thank you for registering");
      page.expose("firstname", person.get("firstname"));
      page.expose("lastname", person.get("lastname"));
      page.expose("email", person.get("email"));
}
```

While Niggle aims to be view-agnostic, the primary maintenance person on Niggle recently became the primary maintainer of FreeMarker ("FreeMarker" on page 21) so it will become better supported. In the View page *thanks.nhtml* that is used to acknowledge the action, variables like `${title}` and `${firstname}` will be substituted with the values given them in the `expose()` method.

Niggle takes advantage of the object-oriented nature of Java to enable easy data validation. For example, in a RecordSet, each field can have a different type, and the defined types can be used to validate or transform ("normalize") the data. Predefined normalizations include all-upper case, all-lower case, capitalize, trim leading/trailing space, and collapse multiple spaces. Predefined validations - subclasses of the `StringField` class - include email addresses and URL fields. My implementation defines a CountryCodeField class to validate a two-letter lower-case ISO country code, ensuring that the user has selected a country from an HTML <select> element where the display values are country names and the parameter values are the corresponding ISO country codes.

Niggle is surprisingly easy to use, once you get over a few initial hurdles. The tutorial documentation is a bit sparse and there are a few strange points (such as the need to install the

1. Apart from a three-line Servlet, which just calls a superclass constructor; this could be obviated by designing a generic dispatcher servlet as Struts and other frameworks have done.

XML configuration files not in *WEB-INF* but in *WEB-INF/classes*, so that they can be loaded using the ClassLoader mechanism). But its ideas are good, and its code is very high level. Niggle deserves to be better known and used; this attention would also presumably lead to improvements in the documentation; I have submitted a couple of minor documentation changes in the few hours it took to write the Sample Implementation.

See Also: “Implementation using the Niggle Framework” on page 39

3.5.2 DBForms

dbforms provides a relatively light-weight interface to a database. It is fairly easy to use, but is limited to fairly straightforward data-related applications (hence the name). Indeed, the project’s developers do not attempt to expand dbforms’ scope beyond this, saying on their web site:

> You may use DbForms in conjunction with common JSP-pages, Struts-based pages, etc. This means that you are free to use DbForms where it brings you the most benefits (dramatically reduce development efforts, etc.) and to use other techniques if you think that they offer a better solution.

The dbforms package is hosted at *http://jdbforms.sourceforge.net/*.

See Also: “Implementation Using DBForms” on page 40

3.6 Full MVC Frameworks

The “full” frameworks either use one of the above-mentioned View technologies or implement their own.

3.6.1 Turbine

Turbine is the MVC framework of the Apache Jakarta project, and can be found at http://jakarta.apache.org/turbine/. It offers a full MVC framework, and operates at several layers:

- Model layer defaults to “Torque”, Turbine’s own persistence layer, but can also use other persistence layers such as OJB or Hibernate.
- View layer can use either Velocity or “standard” JSP pages
- Controller is provided by Turbine.
- HTML Forms Validation using Turbine’s own “Intake Service” (see below).
- The Logging layer uses the Jakarta Commons Logging and can therefore work with logging frameworks such as Log4J, the J2SE 1.4 standard logging, or simple file logging

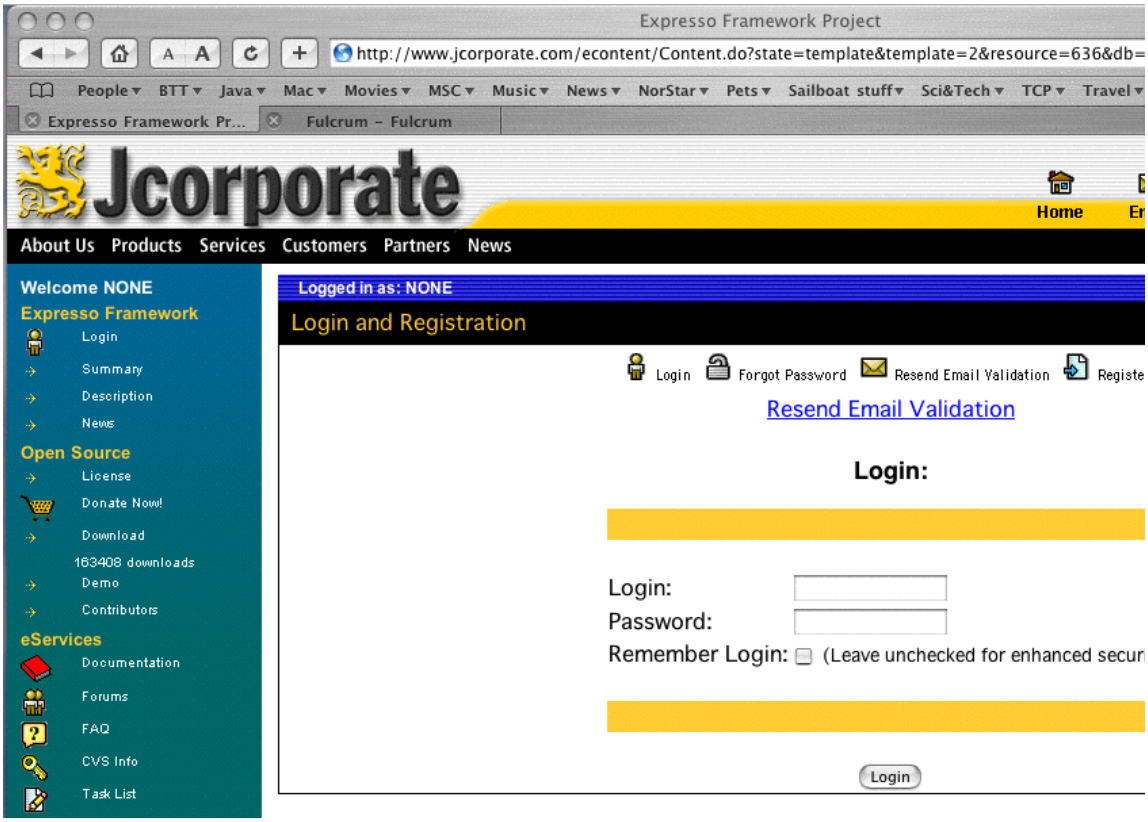
As well, Turbine offers a plug-in based “services” framework. Some of the distributed services include:

- The SecurityService provides a “user, group, role” permission system similar to that of the Servlet standard “container managed security” but with finer-grained control. Can be driven from user information in a database or via LDAP.
- The IntakeService offers forms validation and automatic mapping of forms parameters to a bean, including validation, based on XML configuration files.
- The UploadService manages the uploading of files based on the standard HTML/HTTP browser upload mechanism.
- The XSLTService can transform XML documents into other forms such as PDF or HTML, using XSLT stylesheets

3.6.2 Expresso

Expresso is a heavy-duty framework. Expresso uses Struts underneath, but it seems quite a distance below the surface. An Expresso developer typically implements at least an Expresso Controller component (subclassed from the Struts Action class) an Expresso model object (subclassed from Expresso DBObject), and an Expresso “Schema” which ties these pieces together.

Expresso provides many nice facilities, like a user login mechanism, numerous setup and testing capabilities, and more (see screen shot below).



The cost of all these is that Expresso applications, unlike simple web-apps and unlike applications deployed with most of the other frameworks, are not self-contained and cannot be deployed self-containedly. You can only deploy an Expresso-based application into an existing Expresso deployment. The JSPs for a hypothetical module “demo” must be copied into (Expresso WEB-ROOT)/components/demo, while the classes must be merged into (Expresso WEB-ROOT)/WEB-INF/classes, and the supplemental jars merged into (Expresso WEB-ROOT)/WEB-INF/lib. This latter process is the most error-prone, because JAR file conflicts can cause either the web server to not start up or, more commonly, to cause the Struts ActionServlet not to start, which in turn causes the entire Expresso service to become inoperative.

Further, if you decide to use some of their mechanisms but not others, you will have to implement a lot of functionality. For example, in a self-registration web page such as the sample implementations done in the research phase of this project, to use the fancy Registration controller, your “DBObject” implementation must either extend their User class, or implement their UserInfo interface with approximately thirty methods (many of them simple setter/getter methods for the “standard” fields they have defined); you would basically be writing a conventional Data Accessor or DAO from scratch.

Another error-prone aspect is that, in most examples, methods in the user-defined "Schema" class are shown using introspection ("class.forName(String)") to load the DBObject and Action classes (addDBAction and addController methods); if the class names passed are incorrect, the developer must go looking through server logs to find the cause of the error (the browser simply reports a NullPointerException), then recompile, reinstall, and re-start the web application. Since the 5.0 release of Expresso, this can be ameliorated by using overloaded forms which accept a Class object instead of a String, that is, changing

```
addDBObject("jwfdemo.dbobj.PersonDB");
```

to

```
addDBObject(jwfdemo.dbobj.PersonDB.class);
```

will cause any errors to be detected at compile time. However, this can not be applied to the names of the fields in the database, which can only be checked at run time.

As well, the name “expresso” is normally visible in all the path names, which doesn't look very professional. Expresso Project lead developer Mike Nash claimed, in a posting on their support forum, that you could eliminate this by installing Expresso as the "ROOT" application; this sounds plausible for an installation using nothing but Expresso, but I have not tested it.

3.6.3 M7

M7 is not just an MVC framework, but it is a framework implemented in the constraints of the M7 Application Assembler program. It does not use Struts, but provides its own MVC framework. I chose not to use it on the “TCP” project due to the fact that it is such a

heavy-weight environment; it seems that once you start using it, you have to keep using it. There are a number of tags which it inserts into JSPs, for example, that are only in its library. And the web site says something like “No runtime royalties for now” but does not explain what the runtime royalties will be or when they will be charged.

There are also a few non-standard JSP elements that it uses.

For these reasons, we decided against using it in my work at the Toronto Centre for Phenogenomics.

3.7 Object-Oriented Frameworks

These frameworks use an object-oriented similar to Java’s AWT/Swing on the client side for layout purposes, for event-based interactions, or both. They typically provide “HTML Components”, Java objects which both prepare the HTML for rendering and handle the storage of the resulting data into a database. To give an idea of the difference between traditional approaches such as Struts, and these component-based frameworks, consider the notion of a Country selector in a name and address form. In traditional frameworks, you have to set up the HTML for the <select>, then an <option> for each country. For example:

```
<select name="country">
  <option selected="true" value="/">Choose a country</option>
  <option value="us">Canada</option>
  <option value="us">USA</option>
  ... 200+ other entries ...
</select>
```

Then you still have to write code in the Servlet or Struts Action to retrieve the value of the HTML forms parameter “country” and put it into a DAO or into a JDBC insert/update.

With the component-based frameworks such as JSF, SOFIA, or Velocity, each component is responsible for generating its own HTML. In Sofia, you might replace the 225 lines of HTML needed for a country code chooser, and the action handling to get it into the database, with this one tag:

```
<salmon:input type="state" listtype="countries" name="country"
datasource="dsPerson:PERSON.COUNTRY">
```

The “type” attribute identifies that you want a list of state-related names. The “listtype” attribute lets you choose from among several pre-fabricated lists (countries, US states, Canadian provinces, etc.; the Canadian and Country lists were one of my contributions to this open source framework that is included with Version 2.2 of SOFIA). The “datasource” attribute is better thought of as "data destination",but the name comes from database technology, e.g., javax.sql.DataSource, where a DataSource is simply a conveniently means of getting a connection to a database. In this example, the datasource attribute tells SOFIA to put the selected country into the COUNTRY column in the PERSON table of the relational database identified by the dsPerson data source.

The only downside, if there is one, to these frameworks is that, since the components render the HTML, the components must all be activated before the page can be displayed for the first time, resulting in less “lazy loading” than simpler frameworks. In my experience with SOFIA, this cost is more than offset in the increased productivity that such frameworks offer.

3.7.1 SOFIA

SOFIA, the Salmon (LLC) Open Framework for Internet Applications, is a comprehensive MVC framework. While the SOFIA framework is completely open-sourced (under the GPL), it ties into “best of breed” tools in a couple of areas, DreamWeaver for JSP development and either the open-source Eclipse or the commercial IntelliJ IDEA for object-oriented code engineering. Dreamweaver has been provisioned with a plug-in that lets it understand the SOFIA JSP Tags, making a very powerful editing tool. SOFIA's maintainers did mention (private E-mail communication, February, 2004) that the framework can be used without these proprietary tools, albeit with less convenience. I have had the opportunity to work with SOFIA on a large web project at the Toronto Centre for Phenogenomics, and found it very productive. Its extensive use of JSP tags and HTML components means that one can build powerful web applications with fewer lines of hand-written code than other frameworks might require.

SOFIA Model classes are very powerful; they provide a complete interface to an SQL table in a database, and support joined tables. This has been extended recently, with funding from my client the Toronto Centre for Phenogenomics, to provide “table inheritance” so that, for example, common fields between Person and Company, such as Street Address and Primary Telephone, can be placed in a parent table (called, say, Entity); the child tables Person and Company then add only the unique fields (such as Mobile Phone and Company Name respectively). In SOFIA 2.3, the inheritance of the Model classes can exactly reflect the inheritance structure of the tables.

SOFIA Controller classes provide action handling. There are numerous Controller methods, such as

```
pageRequested();
pageSubmitted();
```

These indicate, respectively, that the page is being displayed, and that the user has clicked a Form Submit button on the given page. As well, the HTML form components have an addSubmitListener() method which takes a SubmitListener argument. The one method in the SubmitListener interface is:

```
public boolean submitPerformed(SubmitEvent e)
throws Exception;
```

This is intentionally very similar to the client-side GUI ActionListener interface, whose one method’s signature is

```
public boolean actionPerformed(ActionEvent e);
```

Both Model and Controller classes are generated by SOFIA, using one of the IDEs mentioned above or using a program called IDETool directly. The Models are generally complete; the Controllers are skeletons into which you need to write the actual action handling code. Either class can be re-generated and your local changes will be preserved if they are made in accordance with comments contained in the file.

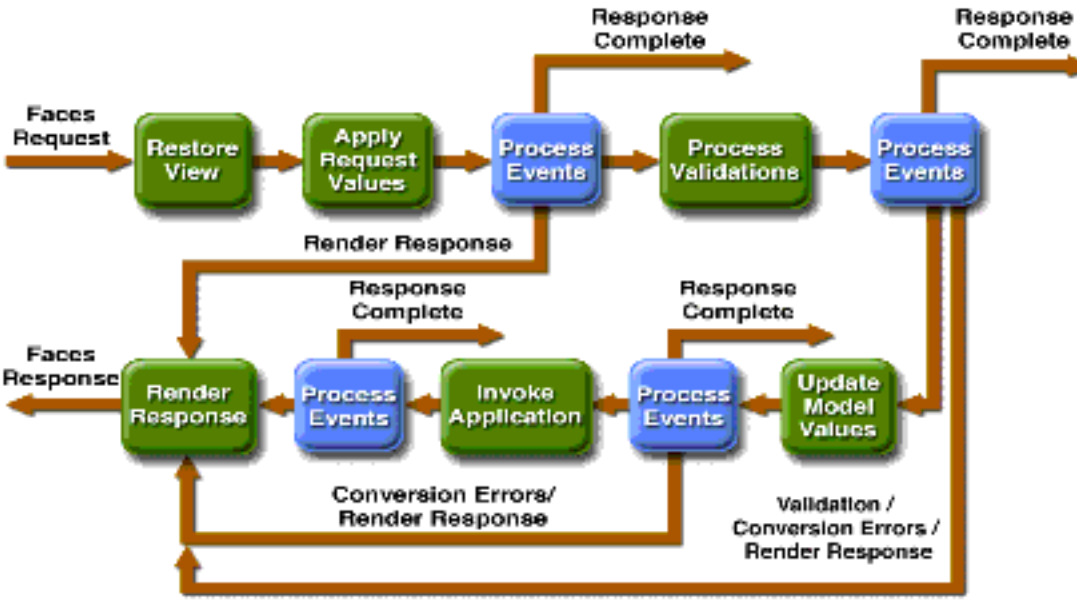
See Also: “Implementation Using SOFIA” on page 40

3.7.2 JavaServer Faces

A relative newcomer to the Frameworks arena, JavaServer Faces (JSF for short) has the official blessing of Sun Microsystems, for which reason it will continue to grow in importance. JavaServer Faces provides a component-based Object model for “GUI” components. As well, Sun’s Creator Studio product (Sun 2004) provides a Visual Basic-like graphical tool for building pages, and is clearly positioned as a competitor to Microsoft’s ASP.Net for providing an easy-to-use tool for building web applications.

JSF itself provides an object model framework similar in some ways to SOFIA. A single Front Controller receives all requests for “.face” files and arranges for the correct processing of each request, based upon the URL and any forms parameters. The view rendering is done by components in the framework. Where it differs is that JSF does not provide data model support, though it does provide for POJO JavaBeans to receive the data from a request and a means to activate code in the bean or in a handler upon activation of an HTML form. This means more work for developers building a complete application from scratch, compared to SOFIA’s model generation code.

JSF also differs from SOFIA in providing an explicitly documented flowchart for the inter-operations among the various components. This diagram is reprinted from Figure 17-4 of Sun's documentation (Sun JSF, 2004)



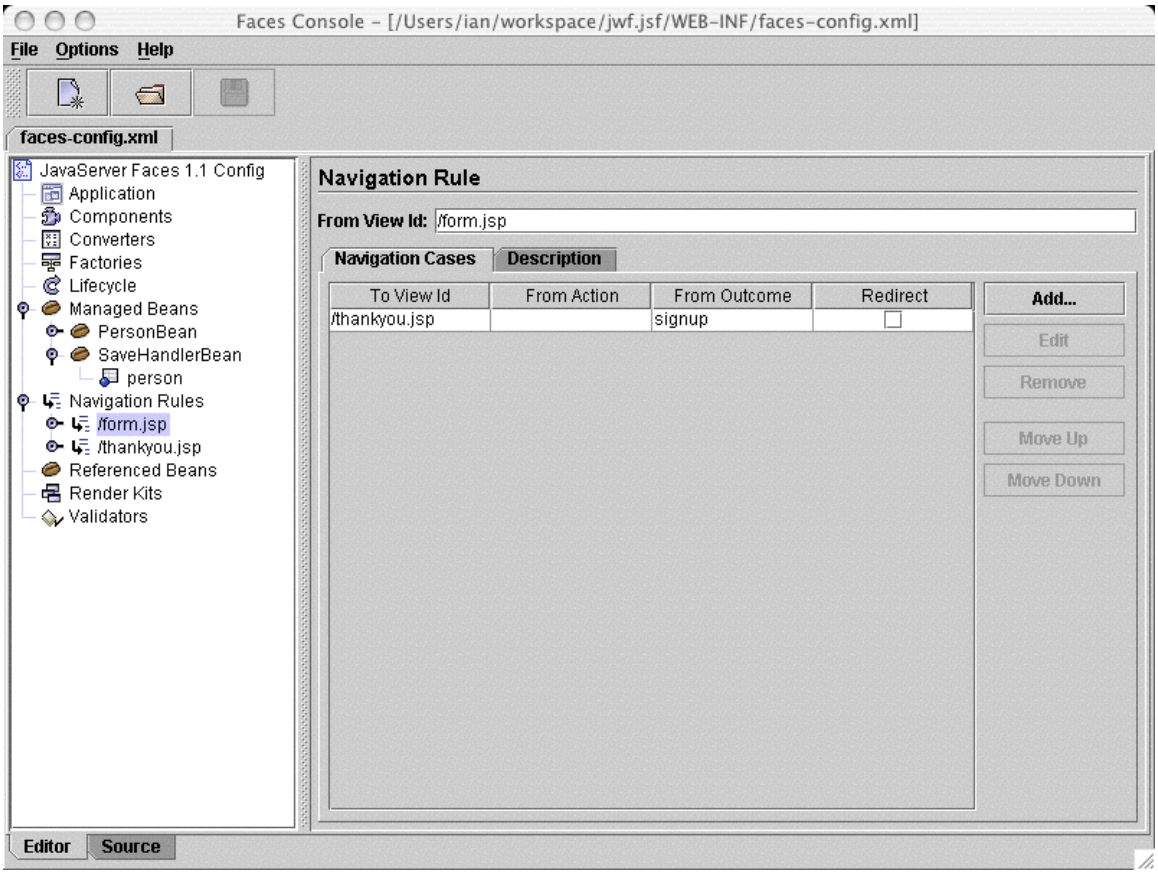
Even when a page is only being displayed, the OO components still must render the view, including any that have initial or saved values. If a form is being submitted, the validators and application logic must also be invoked.

JSF uses an ActionListener framework similar to that of client-side Swing GUIs and of SOFIA above.

JSF also provides clear documentation for subclassing components. A good explanation for practicing developers is in Dudney (2004), where an example is presented of a custom component for credit card validation.

Because of JSF's importance, there is a growing body of free and commercial support tools and add-ons. James Holmes (*http://www.jamesholmes.com*) has written the Faces

Console, a free tool for editing JSF's XML configuration files; here the Faces Console is shown editing the configuration for my JSF Sample Implementation.



JSF also has a public community web site, *http://www.jsfcentral.com/*. The source for two implementations of JSF is publicly available: Sun's master site is hosted at *http://java.net/* , and the open source project *myfaces.org* is a free, open-source re-implementation of the JSF specification. Companies such as Otrix (*http://www.otrix.com/*) are now starting to market components specifically-built for use with JSF.

See also: "Implementation Using JavaServer Faces" on page 46.

3.7.3 webonswing

This project ambitiously aims to allow direct use of Java Swing (client side GUI) to render HTML pages. Unfortunately, as of September, 2003 their web site had posted neither any source code nor any design documents, so it is difficult to see what direction this project will be moving in.

Accordingly my implementation using this framework is the most efficient from a coding point of view, and the most minimal, consisting of exactly zero lines of code. However it

cannot be proven that implementation using this technology conforms precisely to the specification.

I recently discovered that they did finally release in July, 2004 but found out too recently to examine their first release.

See also: *http://webonswing.sourceforge.net/*

3.8 Meta-Frameworks

3.8.1 Keel Framework

The Keel Framework provides a meta-framework into which many other frameworks can be plugged, allowing the developer to "pick and choose". Frameworks supported in Keel 2.0 include Avalon, Cocoon, Struts, Hibernate, Velocity, WebWork2, Axis, Maverick, JBoss, OpenJMS, Turbine, Lucene, BSF, Jelly, JFreeChart, Quartz and others. Many of these are mentioned in this report; of those that are not, Avalon is a server framework (the original basis of Keel), Cocoon is a web publishing framework, JBoss is a J2EE server, Axis is a Java Web Services / SOAP toolkit and OpenJMS is an implementation of the Java Message Service API, Lucene is a Java Search Engine, BSF is the Bean Scripting Framework and Jelly is a Java scripting language, JFreeChart is of course a charting program, and Quartz is a scheduling program (analogous to the well-known Unix "cron" facility. So it is clear that Keel provides a very wide range of services, and is much more widely construed than just a Web Application Framework.

Keel makes extensive use of Java interfaces to isolate these frameworks from the application and from each other; Keel calls this "Component Oriented Programming" or COP. Keel also uses "Aspect Oriented Programming" or AOP (Elrad, 2001) and the Inversion of Control (IOC, see above under "Other Relevant Design Patterns" on page 17) to reduce coupling among parts of the application and the framework. The connections are established in a pair of XML-based configuration files that can easily be edited to provide major changes in the implementation, without having to change a line of code in the user-written part of the application.

Information about Keel can be found at *http://www.keelframework.org/*.

3.8.2 Spring Framework

While Keel provides "the most of all frameworks", Spring provides 'the minimum needed to work successfully", but does offer some support for co-existence with other frameworks (e.g., Struts). Spring is a light-weight framework that promises lower overhead for developing all types of J2EE applications. It provides a "Web MVC" interface to the framework. Similar in some ways to Struts, Spring's Web MVC offers a controller servlet that dispatches to actions, supports rendering via "views", and can support various "M" frameworks for the Model, such as Hibernate.

The main web site for Spring is *http://www.springramework.org/*. A contributed tutorial on their Web MVC can be viewed at *http://www.springframework.org/docs/MVC-step-by-step/Spring-MVC-step-by-step.html*, while the "official" documentation for Spring Web MVC can befound at *http://www.springframework.org/docs/reference/mvc.html*.

4.0 Implementing a Simple Web Application

For the evaluation portion of the research, I chose to implement the processing for a simple web-based HTML form using a variety of Java-based technologies:

- Servlet-only ("worst case")
- JSP-only ("worst case");
- Servlet and JSP ("servlet dispatcher"; MVC, not using a Framework)
- MVC using the Niggle Framework
- MVC using dbforms
- MVC using Struts MVC Framework
- MVC using SOFIA
- MVC using JavaServer Faces

Because of the conflicting requirements for Jar files, each implementation was done as a separate Web Application context. Each implementation is stored in a Web Application structure for ease of use by those wishing to replicate any of my work. Each is also set up as an Eclipse Project (Eclipse.org, 2004) by having project metadata stored in each project directory.

4.1 Servlet-only ("worst case")

Having a Servlet generating HTML is a bad idea. Every bit of HTML must be wrapped in a call to `out.println()`. Quoted attributes must be escaped, or alternating double and single quotes used; either is a mess:

```
out.println("<form action=\"/myservlet\">");
```

Nonetheless, it is certainly possible, and it's what we did before JavaServer Pages were available. This implementation of the Servlet code to receive and acknowledge the contents of the Registration form consists of 129 lines (jwf.servletonly/Servlet.java); this includes code to locate the JDBC DataSource object and pre-compile a JDBC insert statement (Connection.prepareCall()), but has no error checking, and provides a crudely-formatted HTML response.

4.2 JSP-only ("worst case");

This Implementation goes to the opposite extreme, and does all the work inside a JavaServer page. This is how JSPs were used immediately after their introduction, before the divi-

sion of labor became common knowledge. It consists of 95 lines (jwf.jsponly/process.jsp), of which about two thirds are Java code borrowed from the Servlet-only implementation. It is similarly short on error handling.

4.3 Servlet and JSP (MVC, not using a Framework)

In this version of the Servlet, I use the `javax.servlet.RequestDispatcher` interface to explicitly forward from the Servlet to the "acknowledgement" JSP. I also provide an error-formatting page for the case where the JavaBean reports that not all its inputs are valid. The Servlet size is only 67 lines, but the "thank you" page (27 lines) and the error handler (36 lines) bring the total to 130 lines. This provides a minimal but workable implementation of the Model-View-Controller pattern. The Servlet is the Controller, the Model is the Person bean, and the View consists of the two JSP files.

This version and those following use an external JavaBean class, called Person, to represent the data or state of the application - the Model, in MVC terms. This is simply a class with fields for name, address, phone number, and so on, plus trivial get/set methods. It has one additional method, for validation, simply to ensure that the data are adequate - certain fields are required, and this logic is imbedded here. As well, the JDBC code has been excerpted into an additional class, called PersonDAO after the Data Accessor Object pattern (see Section 2.5.3, "Other Relevant Design Patterns," on page 17).

4.4 Implementation using the Niggle Framework

Niggle (see "Niggle" on page 28) is a little-known framework that probably deserves wider acclaim. One little annoyance of the current version of Niggle is that, for each "action" (e.g., add a customer, add an item to the shopping basket, etc.), in addition to writing the action handler class, you must write both a dummy NiggleServlet sub-class - which is invariably null, e.g.,

```
public class MyNiggleServlet extends NiggleServlet {
}
```

By contrast, Struts and most other frameworks provide a "generic" controller Servlet. This could be provided in a future version of Niggle.

The Niggle action handlers, which are called "InterAction" classes, typically get a Form object (which is constructed dynamically from an XML descriptor) to hold the input data, then call the inherited method fillInFields() to cause Niggle to extract the HTML forms data and populate the Form object. Then the forms data can be processed in whatever manner is appropriate to the application. In my example, I insert it into the database using the OreoDataSource (the Oreo package is part of Niggle). Then I get the acknowledgement page and fill in its fields using page.expose(); the resulting page will be displayed by Niggle as its way of displaying the results of this action.

Niggle thus implements the MVC pattern. The Form object is the model or data, the NiggleServletInteraction subclass is the controller, and the output pages (which are by convention named *.nhtml, for Niggle HTML) represent the View.

In the Niggle implementation I also demonstrate use of a custom validator class. Niggle is one of the frameworks provided here that supports automatic validation. These typically provide for “required” fields in an HTML form, and also provide a limited set of predefined validations for common field types such as an E-mail address. To explore the extensibility of Niggle validations, I wrote a Niggle-specific validator subclass for validating two-letter ISO country codes (“uk”, “ca”, “us” and the like). The benefit of such validators is further extraction of logic and code re-use: this validator can be used in any Niggle web application that uses a country code in an input form.

4.5 Implementation Using DBForms

To add dbforms support to an existing Web Application, one must perform the following steps:

- add dbforms_config.xml to WEB-INF, and tailor it for your web application;

- add dbforms_errors.xml to WEB-INF, (may not need tailoring initially);

- add validation.xml to WEB-INF: note that this conflicts with Struts' validation.xml, so if you are using both frameworks in the same Web Application you'd need either to merge them and hope for the best :-) or pick a new name, like dbforms-validation.jar;

- add the dbforms servlet in WEB-INF/web.xml, tailoring the servlet for the locations of most of the above files;

- add dbforms.jar to the WEB-INF/lib directory

- add dbforms.tld to WEB-INF (do not modify);

- add log4j.jar, log4j.properties, commons-beanutils.jar, commons-logging.jar to WEB-INF/lib;

Of course, if you are starting afresh, you would probably begin with one of the provided demos, which has all these pieces in place.

The dbforms package is primarily controlled by one configuration file at runtime

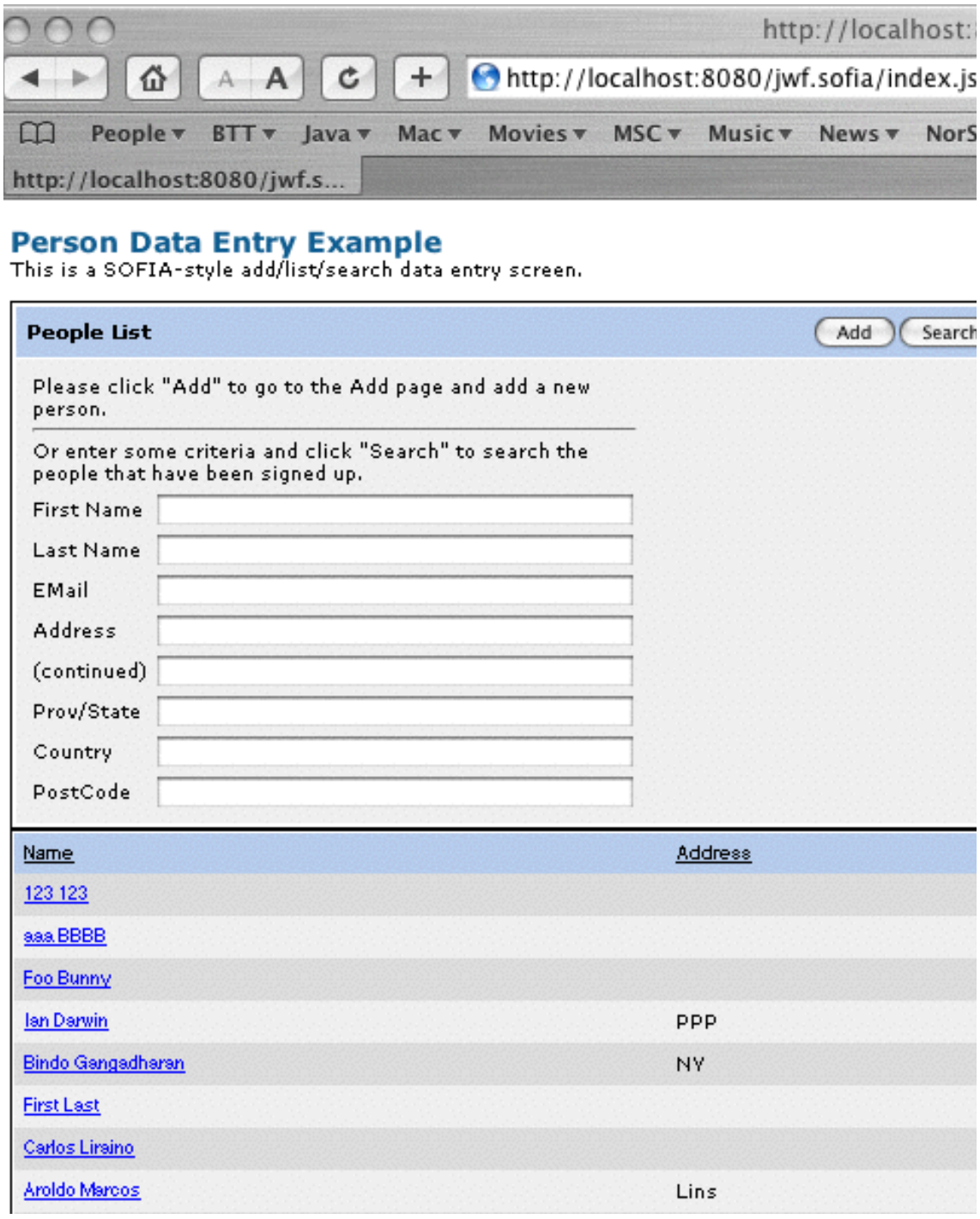
4.6 Implementation Using SOFIA

SOFIA, the Salmon Open Framework for Internet Applications, provides a powerful MVC interface, including extensive support for a data Model in a relational database. Its IDE integration allows you to generate both the model for a database table (or join), and the skeleton of the Controller for a given SOFIA JavaServer Page. My implementation of the

simple Name-Address form for SOFIA is based on SOFIA's "normal" input form rather than the simple form used in the other examples. SOFIA has extensive support for what they call "List/Detail" pages; the list page consists of a search-criteria list and a partial list of the table, while the "detail" page shows the detail for one record being added, updated, or deleted.

The following screen shot shows the List page from my SOFIA implementation, whose filename is index.jsp. Note the Add and Search buttons at the top, followed by the search fields in the top half, and the partial list in the bottom half, with a page navigator. Because SOFIA is an HTML-OO framework, most of this page is provided by components. The

list at the bottom, for example, is provided by a DataTable, the JSP code for which is shown after the illustration.



SOFIA provides an extensive JSP Tag Library of high-level constructs. The list box in this figure was produced in its entirety by a few JSP tags. While there appears to be a lot of code because most of the default attributes were provided by the IDE tool which generated this, in essence all that is needed is:

```
<salmon:datatable>
```

```
  <salmon:datatableheader>
       ...<salmon:tr> and <salmon:td> tags ...
  <salmon:datatablerows>
       ...<salmon:tr> and <salmon:td> tags ...
</salmon:datatable>
```

The full form is shown here (slightly reformatted to fit this page)

```
<salmon:datatable name="people" datasource="jwf"
        clicksort="true"
        rowsperpage="10" pageselectortype="text"
        rowsperpageselector="false"
        width="100%" align="Left" cellpadding="5"
        cellspacing="0">
      <salmon:datatableheader>
          <salmon:tr>
             <salmon:td valign="top">
             <salmon:font type="TableHeadingFont">Name
             </salmon:font></salmon:td>
              <salmon:td valign="top">
                   <salmon:font type="TableHeadingFont">
                     Address</salmon:font>
             </salmon:td>
          </salmon:tr>
      </salmon:datatableheader>
      <salmon:datatablerows>
            <salmon:tr>
             <salmon:td valign="top">
             <salmon:a name="nameLink" href="details"
             datasource="jwf:'%details?id='+people.id">
            <salmon:text name="rowName" text="Name here"
    datasource="jwf:people.firstname+' '+people.lastname"
             font="LargeLinkFont"/></salmon:a>
             </salmon:td>
            <salmon:td><salmon:text name="rowAddr"
             text="City Here"
             datasource="jwf:people.city"/></salmon:td>
             </salmon:tr>
      </salmon:datatablerows>
      </salmon:datatable>
```

The “Detail” form is not shown but it is similary powerful, and is used to add, delete, or update listings.

The important parts of the implementation are the two JSPs, one Model, and a Controller for each of the two pages. These are in the “jwf.sofia” subdirectory of the source code.

Difficulties with SOFIA arise from their non-standard Properties class (which is shared by all web applications within a server instance). This has been corrected in the upcoming version 2.3 (Dubinsky, private correspondence).

Also, SOFIA’s error reporting is lax, on the theory that you normally use the IDE tools and external tools such as DreamWeaver for all editing. For traditional Java code developers this can be frustrating, as a simple change that isn’t quite right will often cause a page to reload continuously with no real indication of the error, requiring perusal of log files to deduce the underlying cause of the error. While these are less common when working within the supported tools, they do still occur on occasion, and can be vexing.

4.7 Implementation using Struts MVC Framework

Struts provides explicit support for the MVC pattern. A single controller Servlet acts as the Controller for all Actions. Action handlers are written as subclasses of the class org.apache.struts.actions.Action. Adjunct “forms beans” for holding data from HTML forms are also provided; these are explicitly not part of “the Model” but simply data helper objects that can be passed into the application-specific “model” code. Forms beans can be written by the user in Java, but it is also common to use dynamically-generated forms beans, build by Struts from an XML configuration file.

Struts applications are often internationalized from the start by use of series of JSP custom tags which simply extract the relevant strings from a standard ResourceBundle. For example, the JSP for the Struts implementation page starts off (after a bit of error checking) like this (file *jwf.struts/index.jsp*):

```
<html:html locale="true">
<head>
    <title><bean:message key="insert.title"/></title>
</head>
```

The `<html:html>` tag invokes the JSP Custom Tag named “html” from the “html” JSP tag library provided with Struts; the `locale="true"` attribute informs the tag to get the preferred Locale from the user’s browser and use it to find the correct language resources. The matching key and value in the English-language ResourceBundle text file (*ApplicationResources.properties*) is:

```
insert.title=Please sign up with us!
```

while the Spanish version (*ApplicationResources_es.properties*) contains this:

```
insert.title=¡Registrarse con nosotros!
```

The page will thus view correctly[1] for either of these locales (the only two that I have prepared localization ResourceBundles for).

The Internationalization feature appears to work nicely, but making any changes does require reloading the web application (see “Slower to live web site” on page 52), because the “properties” files containing the internationalized text are only read when the Struts ActionServlet is loaded or reloaded.

Struts supports forms bean validation in several ways. User-defined forms classes can override the validate() method. Alternately, the “Validator Plug-in”, a standard component of Struts despite the name, can be used. This is controlled by another XML configuration file, and provides validation for any Forms bean (whether dynamic or hand-coded).

The Struts Validator provides extensive validation capabilities. And unlike some of the other frameworks such as Niggle (see “Niggle” on page 28), Struts' validator embeds a significant knowledge base of JavaScript validation.

This can potentially eliminate a lot of HTTP round-trip interactions by performing the validation in the client's browser.

To configure the validator, one need only:

- put a few lines in struts-config.xml, referring to the (generic) validation-rules.xml and the application-specific validation.xml
- write the validation.xml for the forms that need validation

1. Barring any errors in my Spanish translations; for these, I apologize in advance to any Hispanic readers.

For an example of client-side validation, see Figure 5 on page 46.

FIGURE 5. Struts Client-Side Validation

In my Struts sample implementation, the Action subclass InsertAction, called after the form has been filled in and submitted by the user and optionally validated, is 68 lines long. There is a 20-odd line configuration for the validator. This implementation uses the Person bean class and PersonDAO database code from the previous implementations. Because the Struts version is internationalized, there are also hundred-line-long files containing the language inserts for each language. This code is in the jwf.struts subdirectory.

4.8 Implementation Using JavaServer Faces

The Hello Form implementation in JSF is shown here.

The implementation is stored in the jwf.jsf subdirectory, and consists of:

- A trivial index.jsp which redirects to /form.face; it is a requirement that the default page either transfer to, or contain a link to, a URI with the extension.face, using the default page type mappings in web.xml. It is also a requirement that .jsp and .face be distinct, since each type must be processed by the correct servlet (this is analogous to the use of .do and.jsp within the Struts framework).
- Form.jsp (which is the visual part of form.face), which uses two JSP tag libraries provided by Faces to specify the input form fields. Extracts from this JSP appear here:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<f:view>
<h:form id="personForm">
    <table>
    <tr><td>Name: *</td>
       <td><h:inputText id="firstName"
          value="#{PersonBean.firstName}" size="10"/>
         <h:inputText id="lastName"
          value="#{PersonBean.lastName}" size="18"/>
```

```
          </td></tr>
          <tr><td>Email: *</td><td>
          <h:inputText id="email"
           value="#{PersonBean.email}" size="30"/></td></tr>
      ...
      <h:commandButton id="submit"
        action="#{SaveHandlerBean.doSave}" value="Register!"/>
```

This page is, as it is in most of these frameworks, a mixture of “pure HTML” and JSP tags. Note also the use of the value substitution syntax #{value...} which was presumably chosen to avoid conflict with the JSP 2 expression language (el), which uses ${value...}.

- WEB.XML/faces-config.xml, which specifies the navigation rules, such as this one which says to go from the form to the acknowledgement page if the action returns the status value “signup”:

```
<navigation-rule>
     <from-view-id>/form.jsp</from-view-id>
     <navigation-case>
          <from-outcome>signup</from-outcome>
          <to-view-id>/thankyou.jsp</to-view-id>
     </navigation-case>
</navigation-rule>
```

This file also contains information about the two managed beans, the Person Bean (reused from the earliest Servlet) and a custom-written action handler to save a valid submitted form’s content to the database.

```
<managed-bean>
      <managed-bean-name>PersonBean</managed-bean-name>
       <managed-bean-class>beans.Person</managed-bean-class>
       <managed-bean-scope>session</managed-bean-scope>
       <managed-property>
           <property-name>firstName</property-name>
           <property-class>java.lang.String</property-class>
           <value></value></managed-property>
        <managed-property>
             <property-name>email</property-name>
             <property-class>java.lang.String</property-class>
             <value>@</value>
  </managed-property>
</managed-bean>

<managed-bean>
     <managed-bean-name>SaveHandlerBean</managed-bean-name>
```

```
      <managed-bean-class>jwfdemo.SaveHandler</managed-bean-
class>
      <managed-bean-scope>request</managed-bean-scope>
      <managed-property>
            <property-name>person</property-name>
             <property-class>beans.Person</property-class>
             <value>#{PersonBean}</value>
      </managed-property>
</managed-bean>
```

The first managed bean is the Person, kept in Session scope to avoid extra object creation and to keep values from one invocation to the next. Note that each property must be listed here; this is tedious but there are tools to automate this. The email property is given a default value of "@" to remind the user what form of address is expected; this is shown in the screen shot above.

The second managed bean is the Action Handler. It is created at request scope since it is only used for submission, and it has a Person property which causes its setPerson() method to be called with a reference to the Person object. The handler's doSave() method is invoked when the submit button is pressed; this is specified in the commandButton JSP tag of the form JSP.

- The Java code for the Action Handler is in file jwfdemo/SaveHandler.java. It does not use the ActionHandler interface because it is "loosely coupled", using the faces-config file. Tighter coupling typically involves Java code calling the add action listener methods to register created instances of listeners. The code is basically as follows:

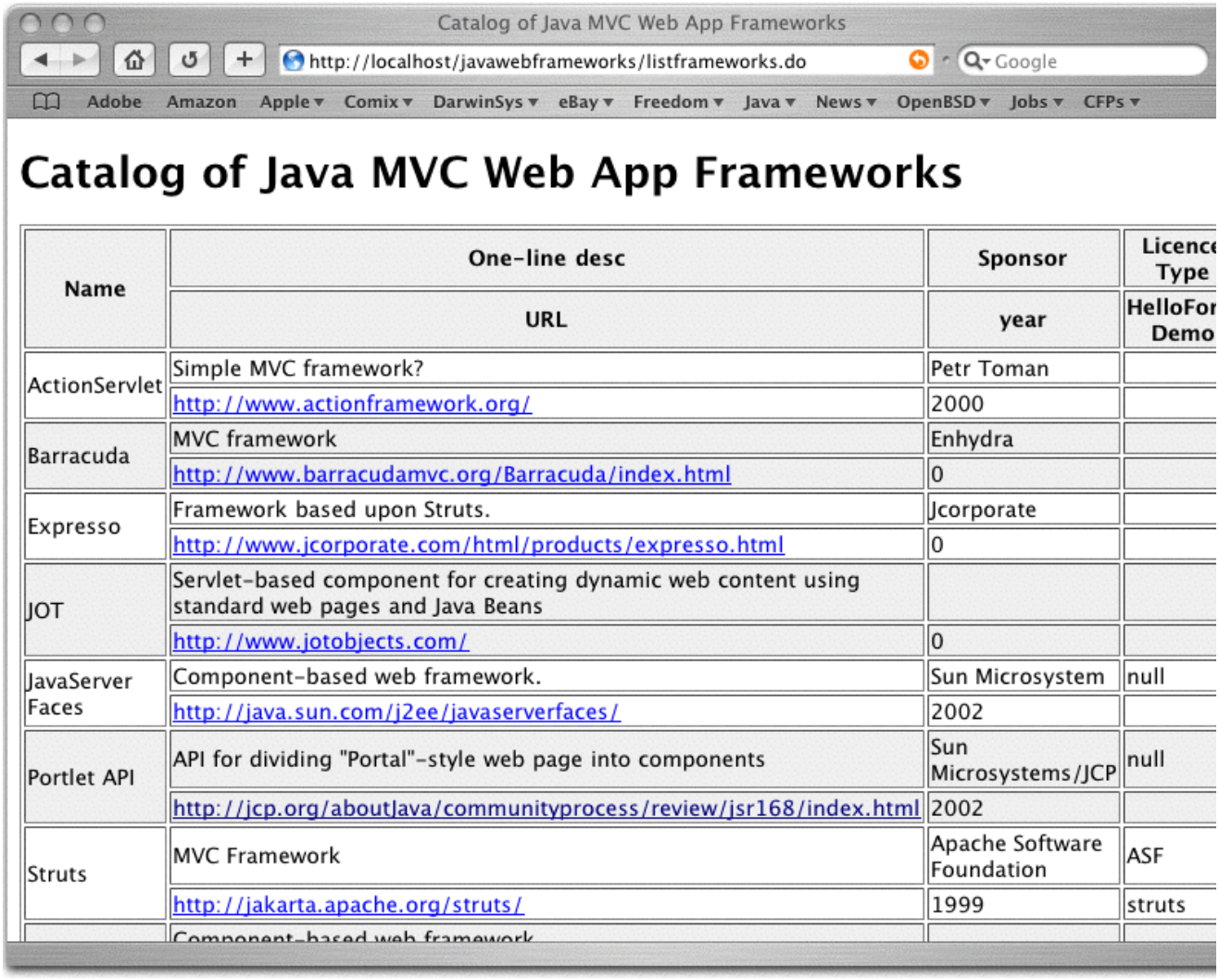
```
import beans.*;
public class SaveHandler {
        private Person thePerson;
        private PersonDAO dao;
        public SaveHandler() {
            try {
                dao = new PersonDAO();
            } catch (NamingException e) {
                // standard error handling omitted
             }
        }

        public void setPerson(Person p) {
             thePerson = p;
        }
        public String doSave() {
                dao.insert(thePerson);
                return "signup"; //indicate success
        }
```

```
}
```

This, plus the standard *web.xml* file to provide the mapping to the Faces Servlet, and the simple acknowledgement page *thankyou.jsp*, rounds out the implementation using JSF.

5.0 The Frameworks Catalog Online

In addition to the Implementations described above and suggested in my original research proposal for this Certificate Module, I have also implemented a Web Frameworks Catalog web application which will make much of this research available to the Java Developer community. This site consists of the summary information about each Framework along with links to its home page and the Sample Application implementation, if available. This was implemented entirely as a Struts application. There is a form submission page used for inputting and updating pages (this section is password protected), and a View Catalogs page which looks something like Figure 6 on page 50.



Name	One-line desc	Sponsor	Licence Type
	URL	year	HelloFor Demo
ActionServlet	Simple MVC framework?	Petr Toman	
	http://www.actionframework.org/	2000	
Barracuda	MVC framework	Enhydra	
	http://www.barracudamvc.org/Barracuda/index.html	0	
Expresso	Framework based upon Struts.	Jcorporate	
	http://www.jcorporate.com/html/products/expresso.html	0	
JOT	Servlet-based component for creating dynamic web content using standard web pages and Java Beans		
	http://www.jotobjects.com/	0	
JavaServer Faces	Component-based web framework.	Sun Microsystem	null
	http://java.sun.com/j2ee/javaserverfaces/	2002	
Portlet API	API for dividing "Portal"-style web page into components	Sun Microsystems/JCP	null
	http://jcp.org/aboutJava/communityprocess/review/jsr168/index.html	2002	
Struts	MVC Framework	Apache Software Foundation	ASF
	http://jakarta.apache.org/struts/	1999	struts
	Component-based web framework		

FIGURE 6. Catalog Web Site

6.0 Frameworks: Summary and Conclusion

6.1 Drawbacks of Frameworks

The use of frameworks continues to grow, thus confirming the benefits they provide to developers. However, these benefits are not without cost: frameworks impose some overhead.

6.1.1 Debugging

Debugging is generally harder when using a Framework than when working directly with Servlets and JSP. For example, Struts creates a single point of failure, the ActionServlet. If the ActionServlet fails to initialize due to, say, a parsing error in its main file (struts-config.xml), then the entire web application will be disabled until the error is rectified and the context or the web server reloaded. Needless to say, this requires that development organizations place a higher value on testing and on configuration management than is sometimes the case in smaller shops.

Both Struts and DbForms, and probably other frameworks that provide framework-specific configuration mechanisms are vulnerable to failure here. DbForms, for example, simply reports when there is trouble with the DBForms Configuration “Can not find CONFIG object in Application Context” It does not report the actual name the object should have been stored under. This makes it difficult to track down the root cause of the error. WIth Struts, there is occasionally confusion between the string name used to store a value in the Http Session, and the name of the Java constant used to specify the Java name as a String constant.

As well, difficulties or curious design choices in the underlying server can affect reliability. Tomcat 4.x, for example, is vulnerable to “Realm” failures. The “Realm” is the unit of authentication namespace for a given web application using J2EE “container-managed authentication”. Each different web app may have a different Realm by which users of that App must authenticate. If one of these is a JDBC-based realm, and the database can not be contacted, the resulting exception will cause the entire Tomcat server to fail to start up. As you can imagine, this “feature” is not appreciated by, for example, Internet Service Providers trying to server multiple clients with a single Tomcat instance. Nor, I might add, by web developers who often run a large number of web applications on a notebook...

Speaking of Tomcat, one of the most annoying features is the failure to print filenames. Both when a Zip file specified by name is missing, or when an XML file contains syntax errors leading to parse failures, the server prints a stack trace indicating in great and gory detail where the error occurred, in terms of Java method calls, but gives not one iota of information regarding the name of the offending file. Both are issues of the underlying software used by Tomcat, but Tomcat really should catch these errors and include the filename in the report.

6.1.2 Slower to live web site

With most of these frameworks, the developer loses the “instant recompile” flexibility of the original JavaServer Pages concept. A JSP is a straight HTML file with some Java embedded into it. To update a JSP, you can simply edit the “master” copy of the file right in the WebServer directory (or update a master copy, commit it to a source archive such as CVS, and then copy it to the web directory); the very next access to the page via a URL will result in immediate recompilation. It is thus common (generally only while testing!) to run a web site directly out of one’s “source” directory; this author has a number of “in development” web sites that are Tomcat Contexts running directly from subdirectories of his home directory on his development machine.

This approach breaks down with frameworks such as Struts, in which, for example, almost the entire textual content of a JSP is not embedded in the page, but output from a Java “properties” file by use of a number of “JSP tags” embedded in the page. As these properties files are loaded into memory when the Web Context is initialized, it is typical to have to rebuild the files in the Context for even the simplest change, or else to “reload” the web server (which causes delays that are unacceptable on production servers) or at least, to reload of the particular “web context”. The difficulty of this is mitigated somewhat by the Ant task for connecting to the Tomcat “Manager” servlet and reloading a given context; this is usually called from an “install” or “deploy” target to copy the updated files and immediately reload the server.

6.1.3 More Artifacts

The use of any of these frameworks imposes an increase of complexity, in that a greater number of individual artifacts must be assembled, compared to writing a straight Servlet or JSP solution. Even the simplest framework has some baggage; to add dbforms support (see Section 3.5.2, “DBForms,” on page 29) to an existing Web Application, you must perform the following steps:

- add dbforms_config.xml to WEB-INF, and tailor it for your web app;

- add dbforms_errors.xml to WEB-INF, (may not need tailoring initially);

- add validation.xml to WEB-INF: note that this conflicts with Struts' validation.xml, so either merge them and hope for the best :-) or pick a new name, like dbforms-validation.jar;

- add the dbforms servlet in WEB-INF/web.xml, tailoring the servlet for the locations of most of the above files;

- add dbforms.jar to the WEB-INF/lib directory

- add dbforms.tld to WEB-INF (do not modify);

- add log4j.jar, log4j.properties, commons-utils.jar, commons-logging.jar to WEB-INF/lib;

Of course, if you are starting fresh, you would probably begin using the “blank” demo, which has all these pieces in place.

The Implementation directories contain approximately the following artifacts. Each entry of the form n/m shows first the number of files, then the number of lines of text. The HTML-based pages have been accumulated in the HTML column; error handling pages have been excluded as they have not been uniformly provided in all the examples. Java source files for a minimal implementation are accumulated under the heading Java; optional validation classes have been excluded. All XML configuration files have been accumulated into the last, except that build.xml files have been excluded.

TABLE 3. Framework Artifacts

Framework	HTML, JSP	Java	Jar Files	XML Files
Servlet Only	1/42	1/143	0	1
JSPOnly	2/137	0	0	1
Servlet+JSP	3/111	1/70	1	1
JSP+JavaBean	2/99	0	1	1
dbforms	2/83	0	4	5/1127
Expresso	2/47	3/196	10	5/510
Groovy	1/30	1/25 (groovy)	5	1/39
JSF	3/66	1/39	9	2/104
Niggle	2/55	2/66	??	2/50
SOFIA	2/182	3/600(!)	3	2/79
Struts	2/99	1/68	8	4/1134

6.2 Frameworks Benefits

The primary benefit of using a Framework is the significant amount of code that you do not have to write and maintain.

A framework provides many facilities that you would otherwise have to write yourself. A single servlet which can pass control to one of a dozen or so smaller Action classes is easier to write (and maintain) than a dozen full-function self-contained servlets. A call to a method that looks up a Struts ActionForward and returns it is more convenient than finding a ServletDispatcher and invoking it. A framework that provides objects to hold values across multiple invocations - so the user need not re-enter the fields that previously passed validation - is much more convenient than one that does not.

A second benefit is the “shared knowledge” phenomena. Developers working on a project that is based upon one of these frameworks will use the framework as a vocabulary, as in “Use a DynaActionForm for that page” or “use an XML form for that list page”. As well, when the need arises to hire additional developers, for one of the mainstream Frameworks (particularly Struts), it is often possible to find developers already experienced in use of

that particular framework; these developers can be integrated much more quickly into the development process than those who required training in the framework. This tends to provide a self-reinforcing accumulation of expertise around particular frameworks such as Struts, not because they are better, but because managers perceive them as relatively "safe" because other managers have already chosen them. Moore (2002) discusses of the "safe adopter" or "herd mentality" that drives this.

6.3 Reflection

In this report I have developed a new Taxonomy for classifying Java Web Application Frameworks, based upon which parts of the MVC trinity a framework actually implements and on whether it uses "HTML Objects" to represent HTML elements on the page. Some of these distinctions may be seen as arbitrary. On first glance, the distinction between clever JSP tags such as those used by Struts and the HTML Components used by SOFIA or JSF might appear contrived. However, the distinction is valid: HTML Components are held as fields in a Controller class and thus have an ongoing state, whereas JSP Tag handlers are instantiated upon each use, so they are basically stateless. Further, JSP tags do not have an event model associated with them,while HTML Components, at least in SOFIA and JSF, each have such a model. So this distinction has merit.

Where the Taxonomy gets in a bit more trouble is in the "MVCO" and "Meta" categorizations. I have not further divided these categories to indicate which frameworks provide all three parts of the MVC model. For example, the "Meta" Spring Framework does not provide its own model, so it should be "VCO", but my reliance on Occam's Razor to avoid a huge number of categories lets me plead only that Spring does make it easy to interface with a variety of other M-type frameworks.

Accordingly, while it is not perfect, I believe that my Taxononomy will provide a useful tool in categorizing and comparing Java Web Application Frameworks and a good basis for continuing analysis by myself and others.

6.4 Recommendations

I would advise any organization undertaking a Java web application development project today to consider adopting Struts if they want the greatest pool of already-trained developers; JSF if they want to get in on "the next wave" and maximize technology lifespan, and SOFIA if they want the best path tobuilding a working web site. This recommendation would be accompanied by a caveat that they also evaluate the many other quality frameworks that are freely available, especially the Spring and Keel Frameworks for their reduced overhead and increased flexibility.

6.5 Conclusion

The use of Frameworks in Web Application Development continues to increase because of the benefits outlined in Section 6.2. As in technology as a whole--where users have settled on MS-Windows, Mac and UNIX platforms--there tends to be a settling-out around some

of the main frameworks. Struts in particular, now that it has been rendered almost obsolescent by newer frameworks such as JSF, is becoming "established":many recruiting agencies now require Struts competence for Java Web hirings, and a major technology training vendor is planning on launching a course on Developing Web Applications with Struts in early 2005.

I believe that this trend will continue and accelerate, with Struts and JSF becoming the two major commercially-utilized frameworks for Java MVC web applications into the 2005-2006 time frame and beyond.

The pace of change in the Web Applications world will keep going, and good tools will always be a source of value.

6.5.1 Future Directions

I would like to do a more detailed implementation of larger applications using some of the leading frameworks, such as Struts and JSF, to ascertain whether the notions of scalability continue as larger applications are developed.

I would like to find research funding to develop the following modules; these might be useful deliverables in future modules of this MSc program, and would be generally useful to the open source Java web application community:

- Eclipse plug-ins for 1) building Web application projects quickly (setting build classpath, build.xml, and so on), and 2) visual editing of JavaServer pages. The latter would work with any of the packages that are based on JSPs, and could perhaps be built quickly on top of Swing's HTML Editor Kit.
- A complete MVC web framework based on lessons learned from all of the other frameworks.

References

Alur et. al., 2001 — *Core J2EE Patterns: Best Practices and Design Strategies* book, 2001, by Deepak Alur, John Crupi, Dan Malks. Patterns are online at the book's web site, *http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/*, as of June 2003.

Barracuda, 2002 — Comparison paper online at *http://barracuda.enhydra.org/cvs_source/Barracuda/docs/landscape* compares Barracuda with Struts, Turbine, Velocity, and many more.

Berners-Lee, 2000 — *Weaving the Web*, Harper Business; 1st edition November, 2000, ISBN: 006251587X. See also his personal site, currently online at *http://www.w3.org/People/Berners-Lee/*.

Bishop *et al*. 2002 Kevin W. Bishop, Arlen Johnson and Deb Wentorf, *From dated to dynamic: a campus newsletter unfolds as a web service*, in *Proceeding of the 30th annual ACM SIGUCCS fall conference on User services conference*, 2002, Providence, Rhode Island, USA. Note that their use of Web Service in the title is misleading under current terminology; it is a Web Application, not a SOAP-based Web Service.

Brittain/Darwin, 2003 *Tomcat: The Definitive Guide*, Jason Brittain and Ian Darwin, O'Reilly, 2003.

Burbeck, 1987 Model-View-Controller, original paper, available online at *http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html* (as of June 2003).

Darwin, 1999 Ian Darwin, "GUI Development with Java", in *Linux Journal*, May, 1999. SSC, Seattle, WA, USA. ISSN 1075-3583

Darwin, 2001 *The Java Cookbook*, O'Reilly, 2001. JabaDot is discussed at the end of Chapter 18, "Servlets and JSP".

Darwin, 2004 *TheJava Cookbook*, Second Edition, 2004. JDO is discussed in Chapter 20, "Databases".

Darwin, 2004a "Keeping Up with the Java Joneses", article on O'Reilly.net at *http://www.onjava.com/pub/a/onjava/2004/07/28/javackbk2.html* discusses a variety of "current" technologies including Hibernate.

DataMonitor, 2003 The Register.UK, available online at *http://www.theregister.co.uk/content/53/31021.html*, posted June 4, 2003.

Dudney, 2004 Bill Dudney, *Creating JSF Components*, July 2004, online at *http://today.java.net/pub/a/today/2004/07/16/jsfcustom.html*.

Elrad, 2001 Tzilla Elrad, Robert E. Filman, and Atef Bader, "Aspect-oriented programming: Introduction", in *Communications of the ACM*, Volume 44 , Issue 10 (October 2001), pp 29 - 32. ACM Press New York, NY, USA

FOLDOC, 1995 Free Online Dictionary of Computing, Imperial College, Department of Computing. The definition for Framework is online at *http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=framework&action=Search*

Fowler, 2004 Martin Fowler, *Inversion of Control Containers and the Dependency Injection pattern*, revised January 2004, online at *http://www.martinfowler.com/articles/injection.html*.

GOF, 1995 — Gamma, Helm, Johnson and Vlissides (widely referred to as the "Gang of Four"), Addison-Wesley, January, 1995.

Hiltzik, 2000 — *Dealers of Lightning: XEROX PARC and the Dawn of the Computer Age*, HarperCollins, ISBN 0887309895. This overview of early work at the XEROX Palo Alto Research Center and how so many of the OO and GUI ideas we use today originated with a small band of researchers puts us in mind of Churchill's famous remark about the Battle of Britain.

Husted, 2002 — A large collection of Java-centric Web links; start at *http://www.husted.com/struts/links.html#mvc* (viewed June, 2003)

Husted, 2003 — "Is Struts Performant?" article reasoning that Struts will gain in performance over non-MVC solutions because of Java API details such as the way Servlets run multi-threaded. Online at *http://husted.com/struts/resources/performant.htm* (as of June, 2003).

ISO8879:1986 — *Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML)* , International Organization for Standardization (*http://www.iso.org*). Standards document available for purchase in hardcopy or electronically.

Johnson, 1988 — Ralph E. Johnson and B. Foote. Designing reusable classes. J. Object-Oriented Programming, 1(5):22--35, June/July 1988.

JUnit, 2001 — JUnit, from *http://www.junit.org*, is a widely-used "unit testing" facility for Java applications.

Leff, 2001 — "Web-application development using the Model/View/Controller design pattern", Leff, A. and Rayfield, J.T. of IBM Thomas J. Watson Research Center, Hawthorne, NY. Appears in: *Enterprise Distributed Object Computing Conference*, 2001. EDOC '01, Proceedings Fifth IEEE International Conference, 09/04/2001 -09/07/2001, 2001, Seattle, WA, USA. pages 118-127

McClanahan, 1999 — Struts Framework, discussed in online video presentation at *http://www.theserverside.com/events/videos/CraigMcClanahan/dsl/interview.html* (MS-Windows Media Format)

MIT-XT, 1988 — *An overview of the X toolkit*, Joel McCormack and Paul Asente in *Proc 1st annual ACM SIGGRAPH symposium on User Interface Software*, Alberta, Canada, Pages: 46 - 55,1988. ISBN:0-89791-283-7

Moore 2002 — Geoffrey Moore, *Crossing the Chasm* (HarperBusiness; Revised edition, August, 2002, ISBN 0060517123)

NCSA 1995	The CGI specification, online at *http://hoohoo.ncsa.uiuc.edu/cgi/intro.html* with a last-modified date in 1995.
OMG	CORBA information is at *http://www.omg.org/gettingstarted/specintro.htm#CORBA*.
RFC793	Transmission Control Protocol (TCP), online at *http://www.ietf.org/rfc/rfc0793l.txt*
RFC768	User Datagram protocol (UDP), online at http://www.ietf.org/rfc/rfc0768.txt.
RFC2616	HyperText Transport Protocol (HTTP), Tim Berners-Lee, CERN
Seshadri, 1999	Govind Seshadri, *Understanding JavaServer Pages Model 2 architecture: Exploring the MVC design pattern*, December, 1999 - an early explanation of MVC as it applies to Java Web frameworks.
Sun, 1995	*The Java Tutorial*, Threads chapter, online at *http://java.sun.com/docs/books/tutorial/essential/threads/*. Provides a straightforward introduction to the notion of threaded code in the Java environment.
Sun, 2004	Sun's *The J2EE Tutorial* 1.4, June 17,2004, online at *http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html*.
Turau, 2002	Volker Turau, *A framework for automatic generation of web-based data entry applications based on XML*, in *Proceedings of the 2002 ACM symposium on Applied computing*, 2002 , Madrid, Spain. ISBN 1-58113-445-2.
Uden, 2002	IJWET, the International Journal of Web Engineering and Technology. Dr. Lorna Uden of Staffordshire is Editor. Information at *http://www.inderscience.com/catalogue/w/ijwet/indexijwet.html*
uidesign.net, 1999	An early paper on MVC, first published in October, 1999. Author's name not given. Online at *http://www.uidesign.net/1999/papers/webmvc_part1.html* as of June, 2003.

Alphabetical List of Known Frameworks

This table lists all the frameworks I have come across and, for those that are discussed references to the sections in which they are discussed.,

TABLE 4.

Framework name	Framework URL or Reference	Reference
ActionServlet	http://www.actionframework.org/	3.3.1, p. 23
BEA Page Flows	http://dev2dev.bea.com/products/wlworkshop81/technicalguides/pgflow_portability.jsp	
Barracuda	http://www.barracudamvc.org/Barracuda/index.html	
Bento	http://www.bentodev.org/	
Bishop	http://bishop.sourceforge.net/	
Cocoon	http://cocoon.apache.org/	
DBforms	http://jdbforms.sourceforge.net/	3.5.2, p. 29; 4.5, p. 40
Echo	http://www.nextapp.com/products/echo/	
Expresso	http://www.jcorporate.com/html/products/expresso.html	3.6.2, p. 30
fwap	Leff, 2001	
Freemarker	http://freemarker.sourceforge.net/	3.2.2, p. 21
Groovy	http://groovy.codehaus.org/	
JOIST	http://joist.tigris.org/	
JOT	http://www.jotobjects.com/	
JPublish	http://www.jpublish.org/	
JStateMachine	http://www.jstatemachine.org/products.html	
JWIG	http://www.brics.dk/JWIG/	
Japple	http://www.japple.org/	
JavaServer Faces	http://java.sun.com/j2ee/javaserverfaces/	3.7.2, p. 34; 4.8, p. 46
Keel	http://www.keelframework.org/	3.8.1, p. 37
Kona	http://www.aki.com/kona/	
M7	http://www.m7.com	3.6.3, p. 31
Maverick	http://mav.sourceforge.net/	
Melati	http://www.melati.org/	
Millstone	http://millstone.org/	
Niggle	http://www.niggle.org/	3.5.1, p. 28; 4.4, p. 39
OpenSymphony	http://www.opensymphony.com/	
Portlet API	http://jcp.org/aboutJava/communityprocess/review/jsr168/index.html	

TABLE 4.

Framework name	Framework URL or Reference	Reference
Ruby Web Framework	http://ruby-waf.sourceforge.net/	
SOFIA	http://www.salmonllc.com/sofia/	3.7.1, p. 33; 4.6, p. 40
Spring	http://www.springframework.org/	3.8.2, p. 37
SWAF	http://www.sysoft.com/swaf	
Struts	http://jakarta.apache.org/struts/	3.4.2, p. 24; 4.7, p. 44
Tapestry	http://sourceforge.net/projects/tapestry/	
Tea Servlet	http://teatrove.sourceforge.net/	3.4.1, p. 23
Theseus	http://www.brainopolis.com/theseus/	
Thin Client Framework	http://alphaworks.ibm.com/tech/tcf	
Tiles	http://jakarta.apache.org/struts/	3.4.2, p. 24
Turbine	http://jakarta.apache.org/turbine/	3.6.1, p. 29
Velocity	http://jakarta.apache.org/velocity/	3.2.3, p. 22
WakeSoft	http://www.wakesoft.com/	
WebMacro	http://www.webmacro.org/	3.2.1, p. 21
Weaver	http://www.oldlight.com/weaver/	3.3.2, p. 23
WebWork	http://www.opensymphony.org/webwork	
wings	http://wings.mercatis.de/tiki-index.php	
wizard	Turau, 2002	