Open Source for the Enterprise

COMPARING WEB FRAMEWORKS

Greg Hinkle

Erin Mulder

Brian McCallister

Agenda

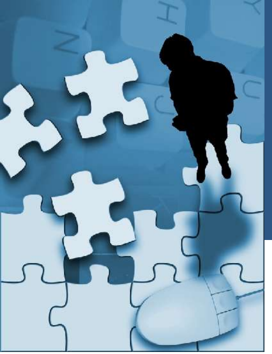
- Introduction to each architecture

Struts	Cocoon	Portlets
WebWork2	Tapestry	Rich Web Clients
Spring MVC	Java Server Faces	

- Recognizing common patterns and knowing when to use them

Model 2 Separation	Transformation	Components
Externalized Flow	Validation	Client-side Logic
		Continuations

Copyright 2005 Chariot Solutions



Struts - Overview

- “Model 2” web architecture (separates presentation from data and controller)
- Provides:
  - Front Controller Servlet
  - Tag Libraries for logic, form binding, etc.
- You write:
  - JSPs
  - Action classes
  - Form Bean classes
  - Flow configuration file (struts-config.xml)

Copyright 2005 Chariot Solutions

Struts - Diagram

HTTP
Request

Front
Controller
Servlet

gets flow info
struts-config.xml

finds or creates
Form
Bean

updates
reads

calls
Action
class

Tag
Library

forwards*
uses

returns response
JSP

HTTP
Response

** Note: This diagram is simplified. Actions actually use the framework to forward, and can forward along to JSPs or other actions.*

Copyright 2005 Chariot Solutions

Struts - Examples

- Example struts-config.xml:

```
<struts-config>

  <form-beans>
    <form-bean name="agencyForm" type="acme.web.agency.AgencyForm"/>
  </form-beans>

  <action-mappings>
    <action path="/agency/NewAgency"
            type="acme.web.agency.NewAgencyAction"
            name="agencyForm"
            input="/WEB-INF/jsps/agencyList.jsp">
        <forward name="success" path="/WEB-INF/jsp/editAgency.jsp"/>
    </action>
    ...
  </action-mappings>

</struts-config>
```


Copyright 2005 Chariot Solutions

Struts - Examples

- Example Form Bean:

```
public class AgencyForm extends ActionForm {

  private String name;
  private String description;

  public String getName() {
    return name;
  }
  public void setName(String name){
    this.name = name;
  }
  public String getDescription() {
    return description;
  }
  public void setDescription(String description){
    this.description = description;
  }
}
```

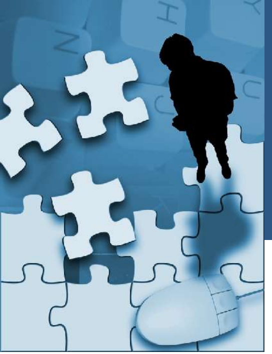
Copyright 2005 Chariot Solutions

Struts - Examples

- Example Action Class

```
public class SaveAgencyAction extends Action {

  public ActionForward perform(ActionMapping mapping, ActionForm actionForm,
       HttpServletRequest request, HttpServletResponse response) throws ... {

    AgencyForm form = (AgencyForm) actionForm;
    Agency agency = new Agency ();
    agency.setName(form.getName());
    agency.setDescription(form.getDescription());

    if (form.isNew()){
      getAgencyManager().add(agency);
    }
    else {
      getAgencyManager().update(agency);
    }

    form.clear();
    return mapping.findForward("success");
  }
}
```


Copyright 2005 Chariot Solutions



Struts - Examples

- Example JSP

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html>
<head><title>New Agency</title></head>
<body>
  <html:form action="SaveAgency">
    Name: <html:text property="name" size="50" maxlength="50"/><br>
    <logic:equal name="agencyForm" property="action" value="Add">
      Description:
      <html:text property="description" size="100" maxlength="100"/>
    </logic:equal>
  <html:submit value="Save Agency"/>
</html:form>
</body>
</html>
```


Copyright 2005 Chariot Solutions

Struts – Review

- Pros
  - Very well-known among developers
  - Supported by many tools
  - Good separation of concerns
- Cons
  - Rigid inheritance structure
  - Form beans duplicate existing objects
  - Errors can happen at inconvenient points
  - Difficult to test
  - Confusing naming scheme

Copyright 2005 Chariot Solutions

WebWork2 - Overview

- “Model 2” web architecture (separates presentation from data and controller)
- Provides:
  - Front Controller Servlet
  - Tag Libraries for logic, form binding, etc.
  - Component System
- You write:
  - JSPs
  - Action classes
  - Components
  - Flow configuration file (xwork.xml)
  - Component configuration file (components.xml)

Copyright 2005 Chariot Solutions

WebWork2 - Diagram

HTTP
Request

Front Controller Servlet

gets flow info → xwork.xml

calls → Interceptors → Action classes → Interceptors → JSP

returns response

HTTP
Response

Copyright 2005 Chariot Solutions

WebWork2 - Examples

- Example xwork.xml

```
<xwork>
  <package name="default" extends="webwork-default">
    <interceptors>
      <interceptor-stack name="defaultComponentStack">
        <interceptor-ref name="component"/>
        <interceptor-ref name="defaultStack"/>
      </interceptor-stack>
    </interceptors>
    <default-interceptor-ref name="defaultStack"/>
  </package>
  <package name="examples" extends="default">
    <action name="SimpleCounter" class="foo.SimpleCounter">
      <result name="success" type="dispatcher">
        <param name="location">/success.jsp</param>
      </result>
    </action>
  </package>
</xwork>
```

Copyright 2005 Chariot Solutions

WebWork2 - Examples

- Example Action

```
public class SelectExampleAction extends ActionSupport {
    private Map selectMap;
    private Long selected;

    public String execute() throws Exception {
        selectMap = new HashMap();
        selectMap.put(new Long(1), "Value 1");
        return SUCCESS;
    }
    public Map getSelectMap() { return selectMap; }
    public void setSelectMap(Map selectMap) {
        this.selectMap = selectMap;
    }
    public Long getSelected() { return selected; }
    public void setSelected(Long selected) {
        this.selected = selected;
    }
}
```

Copyright 2005 Chariot Solutions

WebWork2 - Examples

- Example JSP

```
<%@ taglib prefix="ww" uri="webwork" %>
<html><body>
<ww:if test="errorMessages != null">
  <p><font color="red"><b>ERRORS:</b><br><ul>
  <ww:iterator value="errorMessages">
    <li><ww:property/></li>
  </ww:iterator>
  </ul></font></p>
</ww:if>
<ww:form name="'myForm'" action="'formTest.action'"
         method="POST">
  <ww:token name="'myToken'"/><br>
  <ww:textfield label="'Foo'" name="'foo'" value="foo"/><br>
  <input type="submit" value="Test With Token"/>
</ww:form>
</body></html>
```


Copyright 2005 Chariot Solutions

WebWork2 - Review

- Pros
  - Fairly well-known among developers
  - Clean separation from Servlet API
  - Good separation of concerns
  - Strong component model for services
  - Easy to unit test
  - Simplifies session management
- Cons
  - Less well known and supported
  - Some questionable implementation decisions

Copyright 2005 Chariot Solutions

SpringMVC - Overview

- “Model 2” web architecture (separates presentation from data and controller)
- Provides:
  - Front Controller Servlet
  - Built-in controllers (for forms, wizards, etc.)
  - Built-in validation when binding request data
- You write:
  - JSPs (or Velocity, Freemarker, etc.)
  - Controller classes
  - Configuration file (foo-servlet.xml)

Copyright 2005 Chariot Solutions

SpringMVC - Diagram

HTTP Request → Front Controller Servlet

Front Controller Servlet — gets flow info → foo-servlet.xml

Front Controller Servlet — calls → Handler Mapper

Handler Mapper → Controller Classes

Controller Classes → View Resolver

View Resolver → JSP

JSP — returns response → Front Controller Servlet

Front Controller Servlet → HTTP Response

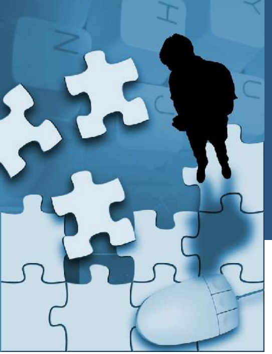
Copyright 2005 Chariot Solutions

SpringMVC - Examples

- Example JSP:

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
<head><title><fmt:message key="title"/></title></head>
<body>
  <h1><fmt:message key="heading"/></h1>
  <p>
    <fmt:message key="greeting"/> <c:out value="${model.now}"/>
  </p>
  <h3>Products</h3>
  <c:forEach items="${model.products}" var="prod">
    <c:out value="${prod.description}"/>
    <i>$<c:out value="${prod.price}"/></i><br><br>
  </c:forEach>
</body>
</html>
```


Copyright 2005 Chariot Solutions



SpringMVC - Examples

- Example Controller:

```
public class MyController implements Controller{

  public ModelAndView handleRequest(HttpServletRequest request,
                                    HttpServletResponse response)
            throws ServletException, IOException {
        String now = (new java.util.Date()).toString();
        Map myModel = new HashMap();
        myModel.put("now", now);
        myModel.put("products",getProductMgr().getProducts());
        return new ModelAndView("hello", "model", myModel);
    }
}
```


Copyright 2005 Chariot Solutions

SpringMVC - Examples

- Example foo-servlet.xml:

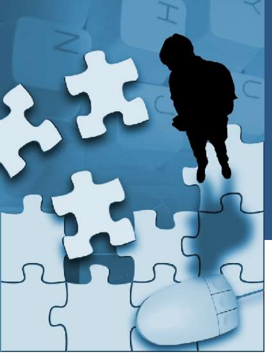
```
<beans>
  <bean id="myController" class="MyController"/>
  <bean id="urlMapping" class="o.s.w.s.h.SimpleUrlHandlerMapping"
    <property name="mappings">
      <props><prop key="/hello.htm">myController</prop></props>
    </property>
  </bean>
  <bean id="viewResolver" class="o.s.w.s.v.InternalResourceViewResolver">
    <property name="viewClass">
      <value>o.s.w.s.v.JstlView</value>
    </property>
    <property name="prefix">
       <value>/WEB-INF/jsp/</value>
    </property>
    <property name="suffix"><value>.jsp</value></property>
  </bean>
</beans>
```


Copyright 2005 Chariot Solutions

SpringMVC - Review

- Pros
  - No form beans (just use your business objects)
  - Nice built-in validation and error-handling
  - Flexible mapping to controllers (not just URL patterns)
  - Builds on top of Spring, letting you use one framework / configuration scheme in all tiers
  - Easy to use AOP and other Spring features in web tier
- Cons
  - Less well-known among developers
  - More abstract, bigger learning curve than some others
  - Verbose configuration files

Copyright 2005 Chariot Solutions



Cocoon - Overview

- XML Publishing Framework
- Provides
  - Everything, including the kitchen sink
    - Model 2 Framework, Continuation Based Framework, Portlet Container, Search Engine, PDF Transforms, WML Transforms, XSLT Engines, SVG Generator, MP3 Indexing System
  - Continuation Based “Modal” Controllers
- You Write
  - Java or JavaScript Controller Scripts
  - XSLT Transforms
  - XML Templates
  - Form Definition Files (CocoonForms)

Copyright 2005 Chariot Solutions

Cocoon - Diagram

HTTP Request → Front Controller Servlet

Front Controller Servlet — get flow/pipeline info → sitemap.xml

Front Controller Servlet — calls or continues → Flow Script

Flow Script → Pipeline

Pipeline:

- XML Generator (JX, etc…)
- Transforms…
- Serializer

Pipeline → Front Controller Servlet

Front Controller Servlet → HTTP Response

Copyright 2005 Chariot Solutions

Cocoon - Examples

- Example Flow Script

```
cocoon.load("resource://org/apache/cocoon/forms/flow/javascript/Form.js");

function registration() {
    var form = new Form("forms/registration.xml");

    // The showForm function will keep redisplaying
    // the form until everything is valid

    form.showForm("reg-display-pipeline");

    var model = form.getModel();
    var bizdata = { "username" : model.name }
    cocoon.sendPage("reg-success-pipeline.jx", bizdata);
}
```


Copyright 2005 Chariot Solutions

Cocoon - Examples

- Example sitemap.xml

```
...
<map:match pattern="*-display-pipeline">
  <map:generate src="forms/{1}_template.xml"/>
  <map:transform type="forms" label="content1"/>
  <map:transform type="i18n">
    <map:parameter name="locale" value="en-US"/>
  </map:transform>
  <map:call resource="simple-page2html">
    <map:parameter name="file" value="forms/{1}_template.xml"/>
  </map:call>
  <map:transform src="resources/forms-samples-styling.xsl"/>
  <map:serialize/>
</map:match>
...
```


Copyright 2005 Chariot Solutions

Cocoon - Examples

- Example JX template

```
<jx:template xmlns:jx="http://apache.org/cocoon/templates/jx/1.0“
             xmlns:i18n="http://apache.org/cocoon/i18n/2.1">
<html><body>
  <jx:choose>
    <jx:when test="size &lt; 1"><strong>No team.</strong></jx:when>
    <jx:otherwise>
      <jx:forEach var="person" items="${list.getTeam()}">
        <jx:set var="name" value="${person.getName()}"/>
        <jx:set var="position" value="${person.getPosition()}"/>
        <jx:set var="country" value="${person.getCountry()}"/>
        <img src="flags/${country}.gif"/>
        ${name} (<i18n:text>${position})
      </jx:forEach>
    </jx:otherwise>
  </jx:choose>
</body></html>
```


Copyright 2005 Chariot Solutions

Cocoon - Review

- Pros
  - Very flexible (transforms, Model 2, modal, etc.)
  - Supports continuations
  - Allows complex flow
  - Fast transformation pipeline
- Cons
  - Steep learning curve
  - Few experienced developers
  - Lots of XML
  - Overwhelming array of architecture options

Copyright 2005 Chariot Solutions

Tapestry - Overview

- MVC Component-Oriented Architecture
- Provides:
  - Framework for building complex user interfaces
  - Base set of components
  - Component model
- You write:
  - Pages and other Components
  - Callbacks
  - Templates
  - Component configuration (Home.page, Table.jwc)

Copyright 2005 Chariot Solutions

Tapestry - Diagram

HTTP
Request

Front
Controller
Servlet

loads info for
requested page

.page files

if
necessary

Page Implementation

Components pull data from page impl

View
Components

View
Components

uses

HTML
Template

renders page

HTTP
Response

Copyright 2005 Chariot Solutions

Tapestry - Examples

- Example: HTML Template

```
<body jwcid="@Body">

<form jwcid="@Form">

  <span msg="add-an-artist">Add an Artist</span>

  <span msg="artist-name">Artist Name:</span>
  <input jwcid="artistName" size="50>

  <span msg="date-of-birth">Date of Birth:</span>
  <input jwcid="dateOfBirth"/>

  <input type="submit" jwcid="saveArtistButton" value="Add Artist"/>

</form>

</body>
```


Copyright 2005 Chariot Solutions

Tapestry - Examples

- Example: Page Specification

```
<page-specification class="cayenne.tutorial.tapestry.pages.AddArtistPage">
  <component id="hasErrors" type="Conditional">
    <binding name="condition" expression="hasErrorMessage"/>
  </component>

  <component id="dateOfBirth" type="DatePicker">
    <binding name="value" expression="artist.dateOfBirth"/>
  </component>

  <component id="saveArtistButton" type="Submit">
    <binding name="listener" expression="listeners.saveArtistAction"/>
  </component>

  <property-specification name="artist"
     type="cayenne.tutorial.tapestry.domain.Artist" persistent="yes"/>
</page-specification>
```


Copyright 2005 Chariot Solutions

Tapestry - Examples

- Example: Page Implementation

```
public abstract class AddArtistPage extends EditorPage {
  public abstract void setArtist(Artist value);
  public abstract Artist getArtist();

  public void saveArtistAction(IRequestCycle cycle) {
    Artist artist = getArtist();
    context.registerNewObject(artist);
    context.commitChanges();

    BrowseArtistsPage nextPage =
            (BrowseArtistsPage) cycle.getPage("BrowseArtistsPage");

    nextPage.setArtist(artist);

    cycle.activate(nextPage);
  }
}
```


Copyright 2005 Chariot Solutions

Tapestry - Review

- Pros
  - Easy to build complex user interfaces
  - Sophisticated UI components available
  - More traditional event based user interfaces
- Cons
  - (Un)Learning Curve
  - Does not use standard view technologies
  - Ugly URL’s
  - Less well known in developer community

Copyright 2005 Chariot Solutions

JSF - Overview

- Component-based web standard
- Implemented by vendors and open source projects
- Provides:
  - Components (input fields, date pickers, tables, etc.)
  - Validation and event handling flow
- You write:
  - JSPs
  - Controller classes
  - Configuration file (faces-config.xml)

Copyright 2005 Chariot Solutions

JSF - Diagram

HTTP
Request

Front Controller Servlet

loads config → faces-config.xml

- find/build components tree & controller
- apply request values to components
- call component validation
- apply values to model (Java Beans)
- call form/link listeners (controller methods)
- build response (JSP + new component tree)

HTTP
Response

Copyright 2005 Chariot Solutions

JSF - Examples

- Example faces-config.xml

```
<faces-config>
  <navigation-rule>
    <from-view-id>/people.jsp</from-view-id>
    <navigation-case>
      <from-action>#{PersonManager.register}</from-action>
      <from-outcome>success</from-outcome>
      <to-view-id>/register.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
  <managed-bean>
    <managed-bean-name>PersonManager</managed-bean-name>
      <managed-bean-class>foo.PersonManager</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
  </managed-bean>
</faces-config>
```

Copyright 2005 Chariot Solutions

JSF - Examples

- Example JSP

```
<f:view>
  <h:form>
    <h:dataTable id="people"
        value="#{PersonManager.people}"
        var="person"
        rowClasses="oddRow, evenRow"
        headerClass="tableHeader">
      <h:column>
        <f:facet name="header">
          <h:outputText value="Name"/>
        </f:facet>
        <h:commandLink action="#{PersonManager.editPerson}">
          <h:outputText value="#{person.name}"/>
        </h:commandLink>
        ...
```


Copyright 2005 Chariot Solutions

JSF - Examples

- Example controller

```
public class PersonManager {
  private Person currentlyEditablePerson;
  private String foo;
  private DataModel personModel = new ListDataModel();
  public DataModel getPeople(){
        return personModel;
  }
  public String register(){
        currentlyEditablePerson = new Person();
        return "success";
  }
  public void fooChanged(ValueChangeEvent event){
       foo = (String) event.getNewValue();
       FacesContext.getCurrentInstance().renderResponse();
  }
  ...
```


Copyright 2005 Chariot Solutions

JSF - Review

- Pros
  - It's a JCP-blessed standard
    - Eventual competing component libraries?
    - Increased developer mindshare over time
  - More traditional event handling
  - Flexible navigation model (externalized)
- Cons
  - JSPs are very tag intensive
  - No clear standard implementation
  - Limited developer base and design experience

Copyright 2005 Chariot Solutions

Portlets - Overview

- Platform for integrating web applications
- Implementing Servers Provide:
  - Standard Portlet API similar to Servlet API
    - Supports render vs. action, window states, view modes
    - User preferences
  - Layout, Navigation, Skins (all customizable)
  - User and Portal Management Features
- You write:
  - Portlet classes (usually with JSP views)
  - Portlet descriptor

Copyright 2005 Chariot Solutions

Portlets - Diagram

HTTP Request

Portal Server

decodes request

Portal Config

calls action

calls render

Active Portlet

Inactive Portlet

each portlet renders part of response

JSP

JSP

HTTP Response

Copyright 2005 Chariot Solutions

Portlets - Examples

- Example portlet:

```
import javax.portlet.*;

public class HelloWorldPortlet extends GenericPortlet {

   public void doView(RenderRequest request,
                      RenderResponse response)
                      throws IOException, PortletException {
      response.setContentType("text/html");
      response.getWriter().print("Hello World!");
   }

   public void processAction(ActionRequest request,
                             ActionResponse response)
                      throws IOException, PortletException {
      // do nothing for now
   }
}
```


Copyright 2005 Chariot Solutions

Portlets - Examples

- Example portlets.xml:

```
<portlet>
    <portlet-name>hello</portlet-name>
    <display-name>Hello World</display-name>
    <portlet-class>HelloWorldPortlet</portlet-class>
    <expiration-cache>0</expiration-cache>
    <supports><mime-type>text/html</mime-type></supports>
    <portlet-info>
        <title>Hello World</title>
        <short-title>Hello World</short-title>
        <keywords>Hello World</keywords>
    </portlet-info>
    <security-role-ref>
        <role-name>Administrator</role-name>
    </security-role-ref>
</portlet>
```


Copyright 2005 Chariot Solutions

Portlets - Review

- Pros
  - Can integrate applications at web tier
  - Great for applications that are highly customized for specific user groups
  - Nice separation of action and render functionality
  - Centralizes web application management & branding
- Cons
  - Restrictive programming environment
  - No easy inter-portlet communication
  - Best practices are still emerging
  - Difficult to find experienced developers
  - Speed/performance can suffer

Copyright 2005 Chariot Solutions

Rich Web - Overview

- Shifts a lot of controller and presentation logic to client side
- First page load uses traditional HTTP request and HTML response
- Response includes lots of code to run in client-side container (JavaScript runtime, Flash, Applet, XUL)
- Subsequent requests bring back raw data, which is formatted/displayed on the client

Copyright 2005 Chariot Solutions

Rich Web - Diagram

Copyright 2005 Chariot Solutions

Rich Web - Examples

- Too many variations to list
- Go to “Rich Web Clients” talk this afternoon

Copyright 2005 Chariot Solutions

Rich Web - Review

- Pros
  - More responsive applications
  - Rich widgets
  - Reduced bandwidth needs
  - Makes “impossible” web requirements possible
- Cons
  - First page load takes longer
  - Flaky browser support
  - Can break web paradigm (back button, text selection, tabbing, scrolling, etc.)
  - Difficult to develop, test, debug and maintain

Copyright 2005 Chariot Solutions

So far, we've seen…

- Front controllers, navigation and page flow models, validation and error handling, components, tag libraries, etc. etc.
- Lots of overlap
- Several frameworks try to do it all
- Others have dozens of plug-ins
- Don't start by choosing a framework…

…start by looking at your design needs!

Copyright 2005 Chariot Solutions

Model 2 Separation

- Keeps presentation logic separate from data and control logic
- Easier to develop and maintain for anything larger than a small website
- Most modern frameworks work this way, though you can still mess them up with scriptlets and custom tags
- Some frameworks are better than others at making life easy for your web UI designers
- Component frameworks moving back to traditional MVC and event handling

Copyright 2005 Chariot Solutions

Externalized Flow

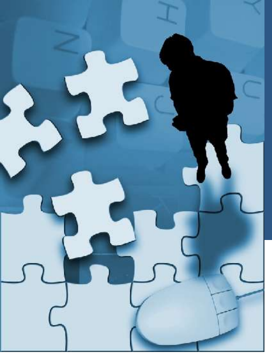
- Applications used to explicitly pass control using specific URLs
- Most frameworks support at least logical separation
- Some pull out navigation and flow into a separate model that can be maintained outside of code
- Trade-off between complexity and flexibility

Copyright 2005 Chariot Solutions

Transformation

- Great for applications that serve lots of data in lots of ways (e.g. CMS)
- Great for i18n and multi-channel delivery
- Not as big a win for highly transactional form-based applications
- XSLT is a natural choice
- Can still use any of the web frameworks to generate XML before transformation

Copyright 2005 Chariot Solutions



Validation

- Challenges:
  - Validate on all tiers
  - Keep validation rules in sync across tiers
  - Highlight errors and give meaningful error messages
- Some frameworks have much better support than others
- Component frameworks are great at highlighting errors
- Lots of pluggable validation libraries that work with several web frameworks

Copyright 2005 Chariot Solutions

Components

- Smart components encapsulate a chunk of reusable control and presentation logic
- Easy to accumulate library of useful building blocks
- Higher returns from usability investment
- Component frameworks are designed to make this easy
- Can do this with other frameworks with custom tags and control-side logic

Copyright 2005 Chariot Solutions

Client Side Logic

- Taking usability to the next level…
  - Minimize page refreshes in click-intensive apps
  - Load cascading drop-downs through Javascript
  - Use interactive/animated components to provide more natural user experience
- Can complicate accessibility, i18n, security and a lot of other design goals
- Check out the “Rich Web Clients” talk for an extensive run-down of available technologies and their pros & cons

Copyright 2005 Chariot Solutions

Continuations

- Your framework may support continuations if…
  - Your controller code reads like a use case
  - You're not checking a "currentAction" field every other line
  - You don't redirect a user to a registration page and forget where they were
  - You actually have more business logic than state management
  - Your application is implemented in LISP

Copyright 2005 Chariot Solutions

Parting thoughts…

- Try to investigate and experience each of the technologies we covered
- During design, think in terms of patterns, not specific frameworks, e.g.
  - “This is a great place for transformation”
  - “We could really use a smart component for this”
- Choose the framework that is best at the things your organization needs to do
- Mix and match additional pieces where appropriate (Tiles, SiteMesh, Validators…)

Copyright 2005 Chariot Solutions