Using Ajax With POJC (Plain Old JavaServer™ Faces Components)

Craig McClanahan, Matthew Bohm, Jayashri Visvanathan

Sun Microsystems, Inc.

TS-9511

Goal of this Talk

To answer the burning question…

How can I add Ajax behaviors to my JavaServer™ Faces technology-based application, without throwing away my investment in existing component libraries?

Agenda

Problem Statement

Background

Issues to Be Addressed

- Low Level Concerns
- Medium Level Concerns
- High Level Concerns

Summary and Q&A

The Problem Statement

What are we trying to accomplish?

- ***I have***
  - Existing Java technology-based web applications…
  - Plus new applications on the drawing board…
  - Based on existing JavaServer Faces component libraries…
  - In which I have a considerable investment
- ***I want***
  - To add Ajax functionality…
  - To my existing applications as well as new ones…
  - Without throwing away my existing libraries
- And it needs to work with my favorite IDEs too

Agenda

Problem Statement

Background

Issues to Be Addressed

- Low Level Concerns
- Medium Level Concerns
- High Level Concerns

Summary and Q&A

Background—JavaServer Faces Technology

Very brief introduction to JavaServer Faces technology

- *JavaServer Faces technology is:*
  - *A server-side user interface component framework*
    - ***Components** modelled as Java objects*
    - *Expressions bind components to **values** and **methods***
    - ***Renderers** emit HTML (or other markup) and/or JavasScript™ technology*
    - *Support for additional features less relevant to this discussion*
      - ***Converters, Validators, Navigation Handler***
  - *A runtime front controller framework*
    - *Well defined **request processing lifecycle** for HTTP POSTs*
    - *Simple dependency injection mechanism (“managed beans”)*
- *Original design centered on form submit handling*

Background—JavaServer Faces Technology

JavaServer Faces components

- *Server side components organized into a **tree***
  - *Single root node provided by the framework*
  - *Content nodes assembled by the application developer*
- *For HTML, the shape of the component tree is **generally** the same as the resulting DOM tree*
  - *Many components write start and end elements*
  - *While delegating nested elements to child components*
- *JavaServer Faces technology maintains the **state** of the tree across HTTP requests*
  - *Simplifies applications—just react to events*

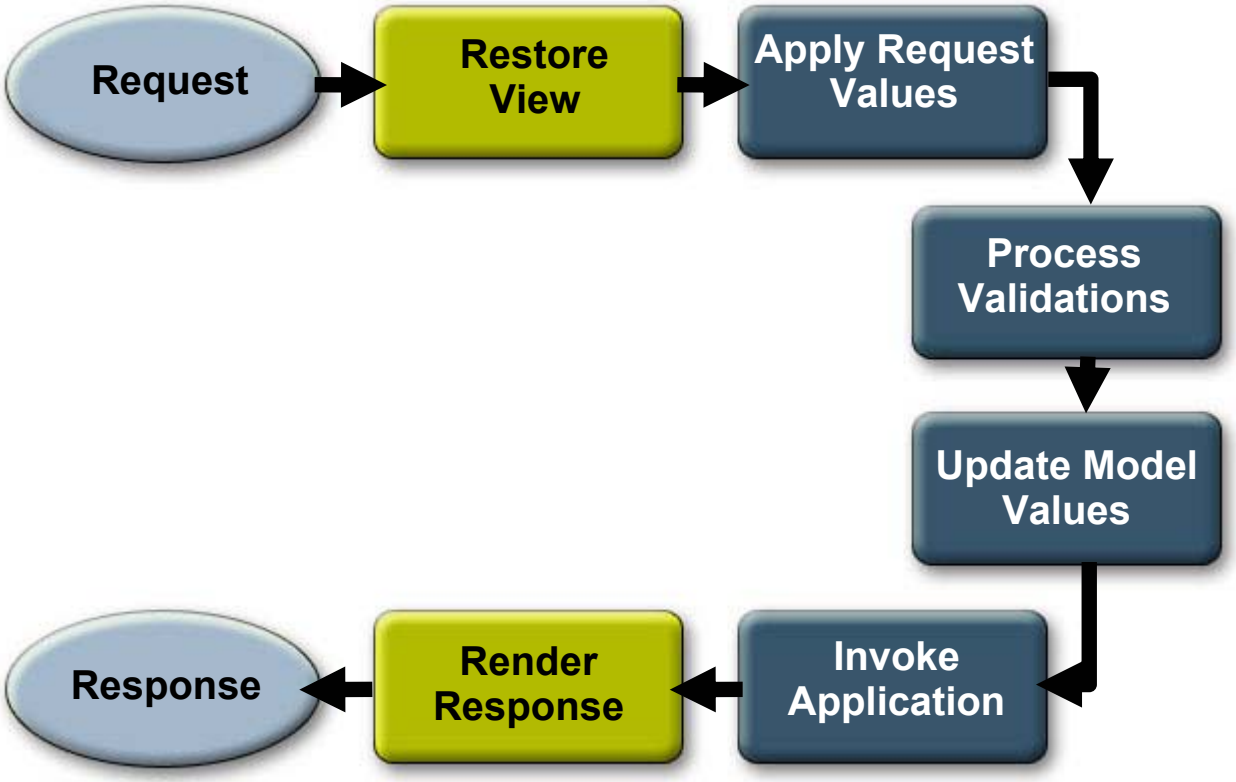
Background—JavaServer Faces Technology

Component Representation In Source [JavaServer Pages™ (JSP™ pages)]:

```
<f:view>
 <h:form id="logonForm">
  <h:panelGrid columns="2">
   <h:outputLabel for="username" value="Username:"/>
   <h:inputText id="username"/>
   <h:outputLabel for="password" value="Password:"/>
   <h:inputSecret id="password"/>
   <h:outputText value=""/>
   <h:commandButton id="logon" value="Log On"/>
  </h:panelGrid>
 </h:form>
</f:view>
```

Background—JavaServer Faces Technology

Standard request processing life-cycle



Background—JavaServer Faces Technology

JavaServer Faces technology today

- *A **Java Community Process℠ (JCP℠)** Standard*
  - *Java Server Faces 1.2 technology (April 2006)—http://jcp.org/jsr/detail?id=252*
  - *Required component of the Java Platform, Enterprise Edition (Java EE 5 platform)*
- *Basis for a rich marketplace of JavaServer Faces component libraries*
  - *Commercial and open source*
  - *General purpose and very specialized*
- *Supported by many popular IDEs*
- *Coming Soon! JavaServer Faces 2.0 technology Java Specificiation Request (JSR) to be filed*

Background—Ajax

Very brief introduction to Ajax

- *Term created about two years ago*
  - *To denote approaches to client-server interactions without conventional page submissions*
  - *"Sort of" an acronym:*
    - *"Asynchronous JavaScript technology and XML"*
- *Underlying technical concepts are nothing new:*
  - *Dynamic HTML modifies client-side DOM on the fly*
  - *Asynchronous (to the user viewing a page) interactions with the server*
    - *Originally performed with tricks like hidden <IFRAME>s*
    - *XMLHttpRequest introduced in IE5, picked up by others*

Background—Ajax

Very brief introduction to Ajax

- *What **is** new is a synergy:*
  - *Browser JS+DHTML that does not crash once an hour*
  - *Increasing demand for better user experience:*
    - ***Zero install** is a compelling advantage for web applications*
    - *Rich client applications and OSs have raised expectations*
  - *Emergence of **"Web 2.0"** or **next generation web** programming models:*
    - *REST-based services*
    - *Client side mashups*
  - *Availability of client-side JavaScript technology **widget** libraries:*
    - *Similar in spirit to JavaServer Faces **components***
    - *Support for client-side events, asynchronous processing*

Background—Ajax

Ajax today

- At the peak of a hype curve:
  - Beyond the early adopters, entered the mainstream
  - Beware of the “gulf of disillusionment”
- Wide range of technology solutions:
  - JavaScript technology client-side only libraries
    - Independent of server implementation technology or language
  - Embedded in server-side technologies
    - PHP templates, Ruby on Rails helper methods, JSP pages/JavaServer Faces technology tags
  - Complete end to end development platforms
    - Flash, Flex, Adobe, GWT, many others…

Agenda

Problem Statement

Background

Issues to Be Addressed

- Low Level Concerns
- Medium Level Concerns
- High Level Concerns

Summary and Q&A

Back to the Problem Statement

What are we trying to accomplish?

- ***I have***
  - Existing Java technology-based web applications…
  - Plus new applications on the drawing board…
  - Based on existing JavaServer Faces component libraries…
  - In which I have a considerable investment
- ***I want***
  - To add Ajax functionality…
  - To my existing applications as well as new ones…
  - Without throwing away my existing libraries
- And it needs to work with my favorite IDEs too

Out of Scope for this Discussion

Interesting problems for another session

- *Create Ajax enabled JavaServer Faces components*
  - ***TS-9516—Using Project jMaki In A Visual Development Environment***
  - ***TS-6178—Simplifying JavaServer Faces Component Development***
  - ***TS-9782—Ajax and JavaServer Faces Technology Tooling in Eclipse***
  - ***TS-6824—JavaServer Faces Technology, Ajax, and Portlets: It's Easy If You Know How***
  - ***LAB-4460—Building Ajax-Enabled JavaServer Faces Components and Web Applications With Project jMaki, Dynamic Faces, and the NetBeans™ IDE***

In Scope for this Discussion

Interesting problems for this session

- *Low level concerns:*
  - *Triggering JavaScript technology events for client side changes*
  - *Performing asynchronous server interactions*
  - *Dynamic updates to related client-side elements*
- *Medium level concerns:*
  - *Modifying existing components for Ajax behaviors*
  - *Synchronizing the server-side component state*
- *High level concerns:*
  - *Performing **partial page submit** operations*
  - *Performing **partial page refresh** operations*

Agenda

Problem Statement

Background

Issues to Be Addressed

- **Low Level Concerns**
- Medium Level Concerns
- High Level Concerns

Summary and Q&A

Low Level Concerns

Motivating example

- *Let's walk through a simple example use case:*
  - ***Coordinated dropdowns***
  - *First dropdown—select a US state*
  - *Second dropdown—select a large city from that state*
  - *Use Ajax to dynamically change second dropdown options when first dropdown value changes*
- *We will be using JavaServer Faces standard components*
  - *But techniques will work with most component libraries*

Low Level Concerns

Triggering JavaScript technology events for client-side changes

- *Problem:*
  - *Need to gain control when interesting events occur*
- *Solution:*
  - *HTML provides a rich variety of event attributes*
  - *Most commonly used:*
    - ***onchange*—*value in an input element has changed***
    - ***onclick*—*element has been clicked***
    - ***onfocus*—*input element gains focus***
    - ***onblur*—*input element loses focus***
- *Most JavaServer Faces components provide **pass through** attributes to add JavaScript technology handlers*

Low Level Concerns

Triggering JavaScript technology events for client-side changes

```
<h:form id="form1">
   ...
   <h:selectOneMenu id="state" value="#{MyBean.state}"
      onchange="switchCities(this)">
     <f:selectItems value="#{MyBean.states}"/>
   </h:selectOneMenu>
   ...
   <h:selectOneMenu id="city" value="#{MyBean.city}">
     <f:selectItems value="#{MyBean.cities}"/>
   </h:selectOneMenu>
   ...
</h:form>
```

Low Level Concerns

Performing asynchronous server interactions

- *Problems:*
  - Initiate asynchronous callback to the server
  - Map request URL to backing bean logic
- *Solutions:*
  - *Use XMLHttpRequest (directly or indirectly)*
    - *We will utilize **dojo.io.bind()** to perform asynchronous I/O*
  - *Map request URL to a Servlet or JavaServer Faces technology handler*
    - *We will use **Shale Remoting***
      - *Avoids requiring an explicit servlet mapping*
      - *Leverages managed beans facility to invoke server logic*

Low Level Concerns

Performing asynchronous server interactions

```
<script type="text/javascript">
  function switchCities(state) {
    ... // Flesh out JavaScript to:
    ... // (a) compose URL .../MyBean/updateState
    ... //     with parameter for new state value
    ... // (b) initiate asynchronous callback
    ... // (c) delegate to updateCities()
  }
</script>
```

Low Level Concerns

Performing asynchronous server interactions

```
<!-- Managed Bean declaration in faces-config.xml -->
<managed-bean>
  <managed-bean-name>MyBean</managed-bean-name>
  <managed-bean-class>
    mycompany.mypackage
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Low Level Concerns

Performing asynchronous server interactions

```
// Managed bean named MyBean (Page 1)
package mycompany.mypackage;
public class BackingBean {

  // "state" -- currently selected state abbreviation
  private String state;
  public String getState() { return this.state; }
  public void setState(String state) {
    this.state = state;
    ... // Make getCities() return updated list
  }

  ...
```

Low Level Concerns

Performing asynchronous server interactions

```
// Managed bean named MyBean (Page 2)

  // "city" -- currently selected city name
  private String city;
  public String getCity() { return this.city; }
  public void setCity(String city) {
    this.city = city;
  }

  ...
```

Low Level Concerns

Performing asynchronous server interactions

```
// Managed bean named MyBean (Page 3)

  // "states" -- selection items for all states
  public SelectItem[] getStates() { ... }

  // "cities" -- selection items for all cities
  // in the currently selected state
  public SelectItem[] getCities() { ... }

  ...
```

Low Level Concerns

Performing asynchronous server interactions

```
// Managed bean named MyBean (Page 4)

  // Update selected state and return revised set of
  // cities to the client
  public void updateState() {
    // ... See next example ...
  }

}
```

Low Level Concerns

Dynamic updates to related client-side elements

- *Problems:*
  - Respond with data or markup (or both)
  - Translate into client-side DOM updates
- *Solutions:*
  - *Transfer data representing the new cities list*
    - *We will use JavaScript Object Notation (JSON)*
  - *Identify client-side DOM element to be updated*
    - *Based on JavaServer Faces technology **client id** of the destination element*
  - *Transform data into new set of <option> elements*

Low Level Concerns

Dynamic updates to related client-side elements

```
// Backing Bean updateState() method (Page 1)

  // Update selected state and return revised set of
  // cities to the client
  public void updateState() {

    // Update the currently selected state
    String state = ...;
    setState(state);
    // Get new list of related cities
    SelectItem[] cities = getCities();

    ...
```

Low Level Concerns

Dynamic updates to related client-side elements

```
// Backing Bean updateState() method (Page 2)

    // Acquire reference to ResponseWriter
    ...

    // Render response as JSON structure
    for (int i = 0; i < cities.length; i++) {
      ...
    }

}
```

Low Level Concerns

Dynamic updates to related client-side elements

```
<script type="text/javascript">
  function updateCities(...) {
    ... // extract JSON from response and eval
    ... // remove old <option> elements
    ... // dynamically create new ones
  }
</script>
```

Agenda

Problem Statement

Background

Issues to Be Addressed

- Low Level Concerns
- **Medium Level Concerns**
- High Level Concerns

Summary and Q&A

Medium Level Concerns

Modifying existing components for Ajax behaviors

- *Is this idea cheating?*
  - *We ruled creating new components out of scope…*
  - *But extending existing components? Hmm*
- *Key to understanding:*
  - *JavaServer Faces technology APIs are **designed** to be extended*
  - *At very fine grained levels*
- *Relevant extension points for Ajax:*
  - *Tag class (to add properties)*
  - *Renderer class (to add default and custom behavior)*

Medium Level Concerns

Modifying existing components for Ajax behaviors

- *Example comes from the Java BluePrints Solutions Catalog:*
  - *https://blueprints.dev.java.net/bpcatalog/ee5/ajax/extendingRenderFunctionality.html*
- *Compose an Ajaxified file upload component*
  - *Based on the standard UIForm component*
  - *Leveraging the existing renderer for basic output*
  - *Configuring default property values for extended behavior*
- *Due to time constraints, we will not have time to examine this solution in detail:*
  - *All the necessary code is available in the Java BluePrints Solutions Catalog entry referenced above*

Medium Level Concerns

Synchronizing the server-side component state

- *Hey wait a minute!*
  - *There is a component tree on the server (JavaServer Faces component)*
  - *There is a component tree on the client (DOM)*
  - *Shouldn't they **always** be synchronized?*
- *When should we care about synchronization:*
  - *The user might press the browser **reload** button*
    - *Expectation—the **current** **state of the page will be displayed***
  - ***New components (and potentially new behavior) have been dynamically added***
  - ***Need to leverage existing components to process part of the component tree***

Medium Level Concerns

Synchronizing the server-side component state

- *When should we **not** **care about synchronization?***
  - *Dynamic changes do not affect the set of components*
  - ***Can deliver reload behavior without a synchronization***
  - ***Cannot afford the extra performance overhead***
- ***Performance overhead?***
  - ***JavaServer Faces components processing per Ajax call not just per POST***
  - ***Entire component tree is restored***
  - ***Perform life-cycle on portions of the component tree***
  - ***Re-render portions of the client DOM***

Medium Level Concerns

Synchronizing the server-side component state

- ***This is a test:***
  - ***In the coordinated dropdowns use case…***
  - ***Did we worry about synchronizing the server-side component tree with changes in the client DOM?***

Medium Level Concerns

Synchronizing the server-side component state

- ***This is a test:***
  - ***In the coordinated dropdowns use case…***
  - ***Did we worry about synchronizing the server-side component tree with changes in the client DOM?***
- ***No—the component tree was not restored***

Medium Level Concerns

Synchronizing the server-side component state

- ***This is a test:***
  - ***In the coordinated dropdowns use case…***
  - ***Did we worry about synchronizing the server-side component tree with changes in the client DOM?***
- ***No—the component tree was not restored***
- ***Yes—server data behind view was synchronized***
  - ***Every change to first dropdown is sent to server***
  - ***Server data is saved in session scope***
  - ***A reload will render the current state and cities***

Medium Level Concerns

Synchronizing the server-side component state

- ***But what if I need:***
  - ***Partial page submit—gather up a particular set of input element values, and send them to a bit of server-side business logic***
  - ***Partial page refresh—the business logic needs to refresh the content of one or more subtrees of the client-side DOM***
  - ***Synchronization—the benefits of synchronizing the server-side state***
  - ***Don't repeat yourself (DRY)—reuse existing components and renderers for partial page updates***
- ***This constellation of needs is very common***

Agenda

Problem Statement

Background

Issues to Be Addressed

- Low Level Concerns
- Medium Level Concerns
- **High Level Concerns**

Summary and Q&A

High Level Concerns

Very common requirements

- ***JavaServer Faces technology is a component oriented architecture***
- ***Common in component based applications:***
  - ***Respond to a user interface event by…***
  - ***Accumulating input values, then…***
  - ***Performing some business logic, and…***
  - ***Notifying view that state has changed, finally…***
  - ***Asking the view to rerender itself***
- ***Traditionally, web UI granularity was a page***
- ***With Ajax, web UI granularity can be an event***

High Level Concerns

Very common requirements

- ***Implementing this strategy can be complicated***
- ***Likely to be a key component of JavaServer Faces 2.0 technology***
- ***In the mean time, use an add on framework:***
  - ***Ajax4JSF***
  - ***Dynamic Faces***
- ***To provide Ajax functionality***
  - ***Partial page submit***
  - ***Partial page refresh***
- ***To plain old JavaServer Faces components (POJC)***

High Level Concerns

Ajax4JSF

- ***Exadel's Ajax4JSF open sourced on jboss.org as JBoss Ajax4JSF***
- ***Component library that adds Ajax capability to existing JSF applications***
  - ***Without any Javascript code***
  - ***Takes full advantage of benefits of JSF framework***
- ***Key benefits***
  - ***Page-wide Ajax support instead of traditional component wide support***
  - ***Access to managed bean facility, server-side convertors, validators etc.***

High Level Concerns

Ajax4JSF

- Two tags of interest for this talk
  - <a4j:support>
  - <a4j:region>
- Also includes other tags with built in Ajax behavior
  - <a4j:commandLink>
  - <a4j:commandButton>
  - <a4j:poll>
  - <a4j:form>
  - <a4j:repeat>

Ajax4jsf—Steps to Add Ajax Behavior

Step 1: Add the required libraries

- `oscache-2.3.2`
- `ajax4jsf-1.1.0`
- `commons-digester`
- `commons-collections`
- `commons-logging`
- `commons-beanutils`

Ajax4jsf—Steps to Add Ajax Behavior

Step 2: Add the following to WEB-INF/web.xml

```
<filter>
  <display-name>Ajax4jsf Filter</display-name>
  <filter-name>ajax4jsf</filter-name>
  <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>

<filter-mapping>
   <filter-name>ajax4jsf</filter-name>
   <servlet-name>Faces Servlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

Ajax4jsf—Steps to Add Ajax Behavior

Step 3: Import ajax4jsf tag library in your JSP page

```
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax"
prefix="a4j"%>
```

Ajax4jsf—Steps to Add Ajax Behavior

Step 4: Use the tags in your JSP page

```
<f:view>
  <h:form>
    <h:inputText value="#{bean.text}">
       <a4j:support event="onkeyup" reRender="rep"/>
    </h:inputText>

    <h:outputText value="#{bean.text}" id="rep"/>
  </h:form>
</f:view>
```

DEMO

Ajax4JSF Example

High Level Concerns

Dynamic Faces

- ***Part of the JavaServer Faces Extensions project***
- Adds Ajax support to JavaServer Faces
- Easily configure Ajax calls, specifying:
  - Web page **inputs** to send
  - Server-side nodes over which to **execute**
  - Web page DOM nodes to re-**render**
- Mechanisms of interest:
  - AjaxZone component
  - DynaFaces.fireAjaxTransaction JavaScript function
  - AjaxTransaction component (visual web complib only)

High Level Concerns

Dynamic Faces—AjaxZone component

- ***Container component***
- Renders JavaScript that “arms” particular children
  - <input>, <option>, <button> armed by default (this is customizable)
  - Default event type is “click” (this is customizable)
- Default interactions (all are customizable):
  - Send inputs within this zone only
  - Execute over inputs in this zone only
  - Re-render this zone only
- A good solution when the elements to arm and the inputs to send can share a common parent

High Level Concerns

Dynamic Faces—AjaxZone component

- **Properties of interest:**
  - **inspectElement** (children to arm)
  - **eventType** (what triggers Ajax call)
  - **collectPostData** (inputs to send)
  - **execute** (server-side nodes over which to execute)
  - **render** (DOM nodes to re-render)
  - **replaceElement** (re-rendering behavior)
  - **postReplace** (behavior after re-rendering occurs)

Source: jsf-extensions.dev.java.net

High Level Concerns

Dynamic Faces—fireAjaxTransaction/AjaxTransaction

- ***fireAjaxTransaction JavaScript function***
  - ***Makes an Ajax call when invoked—no "arming" of components ahead of time***
  - Great for easily configuring virtual any Ajax operation—is not based on containment
- ***AjaxTransaction*** component
  - ***Component version of fireAjaxTransaction***
  - ***Value proposition: visually configure the inputs to send and DOM nodes to re-render via design time color coding***
  - ***Makes an Ajax call when DynaFaces.Tx.fire is invoked, which invokes fireAjaxTransaction***

DEMO

Dynamic Faces Examples

Agenda

Problem Statement

Background

Issues To Be Addressed

- Low Level Concerns
- Medium Level Concerns
- High Level Concerns

Summary and Q&A

Summary

- JavaServer Faces applications abound:
  - Initially designed around HTML “page” paradigm
  - Increasing desire to incorporate Ajax functionality
- Existing investment in JavaServer Faces component libraries
  - Cannot afford to throw away and start over
- Techniques to add Ajax functionality
  - Low level—handwritten JavaScript technology
  - Medium level—extend existing components
  - High level—partial page submit/update frameworks
- Future JavaServer Faces technology versions will standardize here

Resources

- JavaServer Faces technology
  - http://java.sun.com/javaee/javaserverfaces/
- Apache Shale
  - http://shale.apache.org/
- Java blueprints solutions catalog
  - https://bpcatalog.dev.java.net/
- Ajax4JSF
  - http://labs.jboss.com/portal/jbossajax4jsf
- Dynamic Faces
  - https://jsf-extensions.dev.java.net/

Q&A

Craig McClanahan, Matthew Bohm, Jayashri Visvanathan

Using Ajax With POJC (Plain Old JavaServer™ Faces Components)

Craig McClanahan, Matthew Bohm, Jayashri Visvanathan

Sun Microsystems, Inc.

TS-9511