Why Another Framework?

© Garth Gilmour 2008
garth@ggilmour.com

The Evolution of Web Frameworks

- JSP based Web Apps have slowly evolved
  - Initially server pages held lengthy blocks of code
  - Business logic was moved into controller components
    - Within frameworks such as Struts and WebWork
  - Presentation code was moved into Tag Libraries
    - Libraries like the JSTL encapsulated ‘wiring’
    - Other libraries focused on generating complex HTML / CSS
      - To represent ‘widgets’ like calendars, charts and maps
- The design of the Web App becomes much neater
  - But the users experience is still much less rich than with a traditional thick client written in a GUI library like Swing

© Garth Gilmour 2008

The Evolution of Web Frameworks

- As Web Apps become more popular we increasingly want them to behave like normal GUI's
  - This means 'Web Widgets' must work like 'GUI Widgets'
- Key characteristics of GUI components are:
  - They are arranged in a tree structure
    - The window acts as the root of the tree
  - They hold and display their own state
    - E.g. a table maintains a model of the data within it
  - They both generate and respond to events
    - A component generates events as it is used
    - Other components can listen for these events

© Garth Gilmour 2008

The Evolution of Web Frameworks

- Component tree based Web Frameworks have emerged
  - The HTML and Web Control libraries in ASP .NET
  - JavaServer Faces in Enterprise Java
- Both use the ‘Page Controller’ pattern:
  - Each time the page is requested a tree of components is created
  - The tree of components generates a complex user interface
  - When the user makes a choice a ‘postback’ occurs
    - The Web Page generates a POST request back to the server page that generated it, containing information about the event
  - The event is processed on the server side
  - A new version of the page is sent to the browser

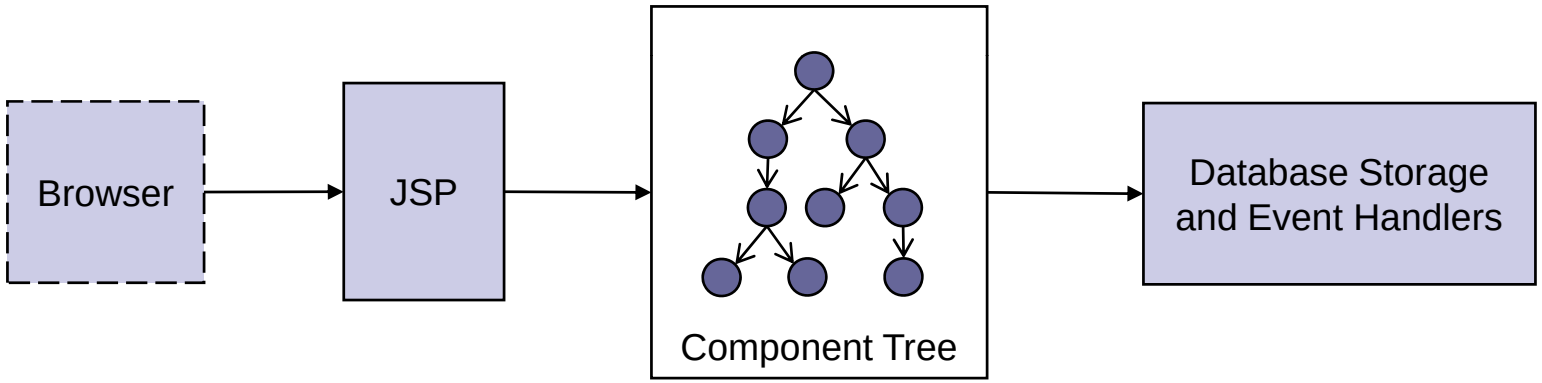
© Garth Gilmour 2008

Introducing JavaServer Faces

- JSF is an architecture for building web controls
  - Components that render a sophisticated GUI, expose a standard event model and manage their own state
- Components to generate basic HTML forms are provided
  - The intention is for vendors to use the framework to develop interoperable libraries of more complex widgets
- JSF is not necessarily tied to the JSP API
  - JSF will initially be used as a layer on top of JSP, but there is no reason why the JSP layer could not be replaced…

© Garth Gilmour 2008

Introducing JavaServer Faces



© Garth Gilmour 2008

Goals of JavaServer Faces

- JSF components are designed to:
  - Manage their own state across requests
  - Encapsulate browser specific rendering issues
  - Provide an event handling infrastructure that allows server side code to respond to browser events
  - Convert and validate the data received from clients
  - Handle navigation between pages and error reporting
- The specification is arranged into thirteen packages
  - Reflecting the complexity and modularity of JSF
  - You don’t need in-depth knowledge of how the components are implemented in order to make use of JSF

© Garth Gilmour 2008

Packages in JavaServer Faces

Package	Description
javax.faces	Contains the base class for exceptions and an object factor locator
javax.faces.application	Classes which control how a JSP application executes
javax.faces.component	The infrastructure used to create User Interface Components
javax.faces.component.html	UI Components representing HTML elements
javax.faces.context	Classes representing the current request and streams for replying
javax.faces.convert	Conversion classes for numbers, booleans and dates
javax.faces.el	The API for the JSF version of Expression Language
javax.faces.event	Classes and interfaces for JSF event handling
javax.faces.lifecycle	Holds the lifecycle manager class for JSF requests
javax.faces.model	Bean classes representing different kinds of data sources
javax.faces.render	The rendering framework for JSF
javax.faces.validator	The Validator interface and some basic implementations
javax.faces.webapp	Classes which integrate JSF into JSP based Web Applications

© Garth Gilmour 2008

Comparing JSF to ASP .NET

- ASP .NET controls have several advantages
  - They are fully integrated with Visual Studio .NET
  - ASP .NET has always included a library of controls that generate rich content (e.g. the calendar control)
  - It is relatively easy to write new controls
- JSF is slowly catching up
  - It is the official Web Framework in JEE 5
  - It is increasingly supported by IDE's like NetBeans
  - Open source and commercial JSF libraries are emerging
    - Such as Oracle ADF Faces and MyFaces
    - But there is still no official library

© Garth Gilmour 2008

Comparing JSF to ASP .NET

```
<form runat="server" style="border: solid black thin; background-color: cyan;
   margin-left: 5em; width: 20em; padding-width: 2em;">
   <p style="margin-left:2em; margin-top: 1em;">
         <asp:Label ID="salaryInstructions" RunAt="Server"
                        Text="Please enter your gross salary"/><br/>
         <asp:TextBox ID="grossSalary" RunAt="Server" Text="10000"/>
   </p>
   <p style="margin-left:2em;">
         <asp:Label ID="taxInstructions" RunAt="Server" Text="Please enter your tax rate"/><br/>
         <asp:RadioButtonList ID="taxList" RunAt="server">
               <asp:ListItem Value="20" Text="Basic Rate (20%)" Selected="true" RunAt="server"/>
               <asp:ListItem Value="30" Text="Medium Rate (30%)" RunAt="server"/>
               <asp:ListItem Value="40" Text="High Rate (40%)" RunAt="server"/>
         </asp:RadioButtonList>
   </p>
   <p style="margin-left:2em;"><asp:Label ID="netSalary" RunAt="Server"/></p>
   <asp:Button Text="Calculate" OnClick="CalculateSalary"
                        RunAt="Server" style="margin-bottom: 1em;"/>
</form>
```

© Garth Gilmour 2008

Comparing JSF to ASP .NET

```
<script language="C#" runat="server">
   void CalculateSalary(Object sender, EventArgs e) {
         try {
               //get the values selected by the user
               // NB no need to muck about with session or request objects
               double gross = Double.Parse(grossSalary.Text);
               int taxAmount = Int32.Parse(taxList.SelectedItem.Value);

               //change the text to be displayed
               salaryInstructions.Text = "You entered the following salary";
               taxInstructions.Text = "You selected the following tax level";

               //display the net salary
               netSalary.Text = "Your net salary is: " + (gross - ((gross*taxAmount)/100));
         } catch(FormatException ex) {
               salaryInstructions.Text = "Please enter a salary level!";
         }
   }
</script>
```

© Garth Gilmour 2008

Performance Issues With JSF

- Component tree based frameworks have issues with performance and scalability
  - The richer the user experience provided by the server-side components the greater the number of postbacks required
  - Every postback forces the user to wait for the page to be regenerated and places an additional load on the server
  - Architects have to balance usability with efficiency
- One possible solution is to combine JSF with AJAX
  - AJAX technologies allow sections of the web page to be repainted via partial postbacks sent from JavaScript functions
  - Again this mirrors how a regular GUI operates

© Garth Gilmour 2008

JavaServer Faces and AJAX

- JSF and AJAX can work well together
  - JSF components can encapsulate the generation of the complex client-side JavaScript that AJAX requires
  - Most JSF based libraries now offer some AJAX enabled widgets
- JavaScript generation should be kept to a minimum
  - Common functions should be held in '.js' files
  - Third party script libraries for AJAX are available
- AJAX enabled JSF widgets should be easy to use
  - They should have the same 'drag and drop' usability
  - Note that there is no standard JEE toolkit for AJAX

© Garth Gilmour 2008

JEE Design, Seam and Web Beans

- JSF does not yet tie-in with other JEE specifications
  - Data entered into JSF components is stored in normal JavaBeans, which also hold the event handlers
  - There is no standard way of combining these with:
    - Business components like Enterprise JavaBeans
    - Data mapping technologies like the Java Persistence API
- The Seam framework shows how this can be done
  - It makes it easy to join JSF and EJB components together
  - This is particularly useful for generating CRUD web apps
- A JSR is underway to make this functionality part of JEE
  - This is JSR 299, commonly known as ‘Web Beans’

© Garth Gilmour 2008

Building JavaServer Faces Based Web Applications

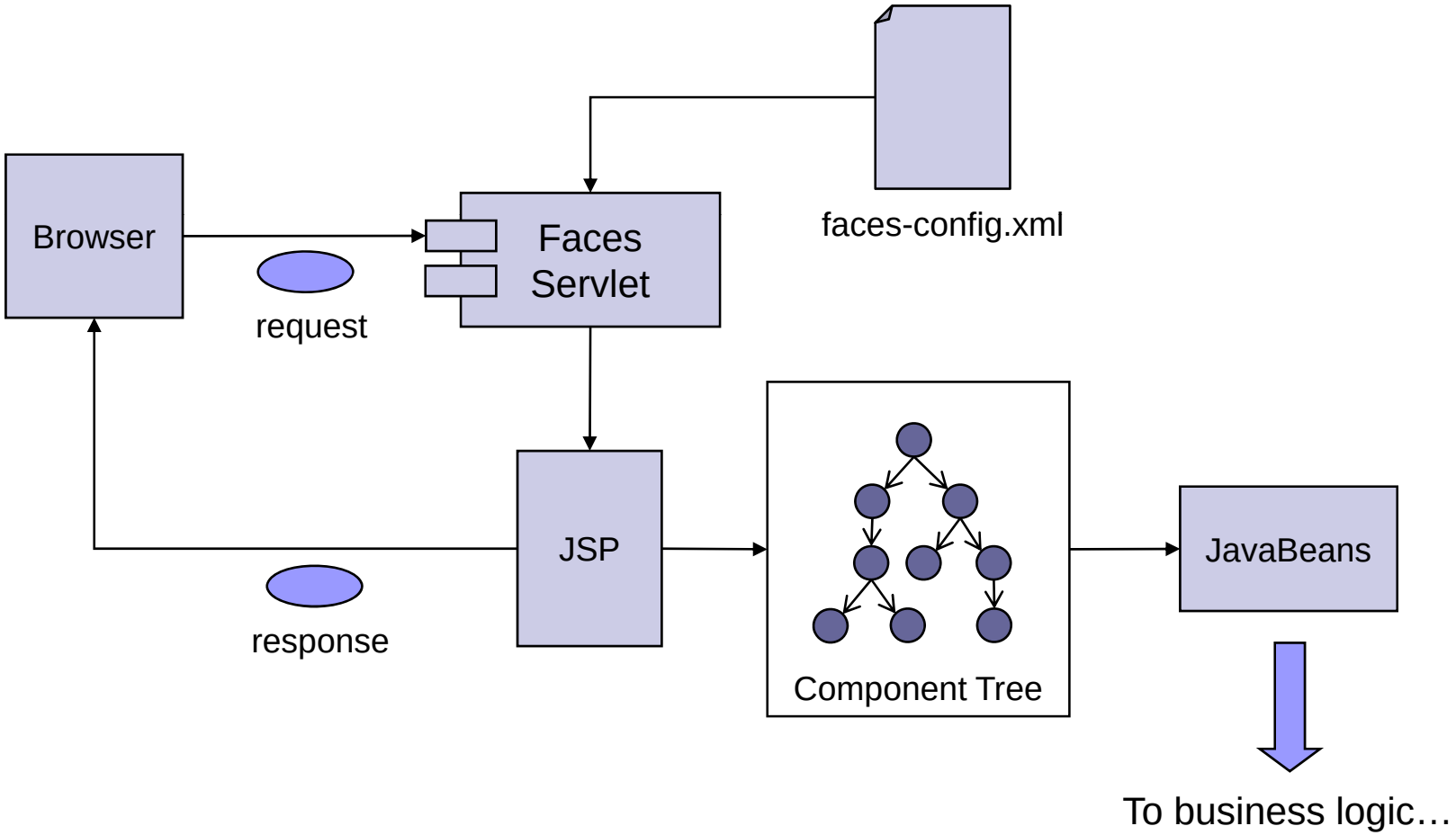
© Garth Gilmour 2008
garth@ggilmour.com

JavaServer Faces Basics

- JSF components appear on JSP's as custom actions
  - All the actions inherit from the class 'UIComponentTag'
- Two tag libraries are referenced at the top of the JSP
  - One library for general purpose actions
    - <%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
  - One library for the HTML rendering actions
    - <%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
- All requests go through a Front Controller Servlet
  - The full name of which is "javax.faces.webapp.FacesServlet"
- As in Struts the Front Controller has its own config file
  - By default this is called "faces-config.xml"

© Garth Gilmour 2008

JavaServer Faces Basics



© Garth Gilmour 2008

Elements in the JSF Config File

Element	Description
application	Configures the locale, adds application listeners and registers plug-ins
factory	Registers replacements for JSF object factories
component	Registers 3rd party JSF Components
converter	Registers 3rd party classes for data type conversions
managed-bean	Registers JavaBeans used to hold state
navigation-rule	Configures navigation rules for JSF forms
referenced-bean	Registers a JavaBean as accessible to JSF but not controlled by it
render-kit	Replaces the default rendering kit or adds custom rendering classes
validator	Registers validator classes
lifecycle	Registers listener classes for the phases of the JSF lifecycle

© Garth Gilmour 2008

Basic JSF Processing

- JSF Components get and set the value of JavaBeans
  - Using an Expression Language similar to JSP 2.0 EL
  - The syntax is '#{ … }' rather than '${ … }'
- The beans are configured in 'faces-config.xml'
  - Using the '<managed-bean>' element
  - Initial values can be set for the bean properties
- The configuration file also contains navigation rules
  - Using the '<navigation-rule>' element
  - These specify where a request is directed when a form submits
  - Each rule specifies the input page, one or more keys associated with submit buttons and one or more destination URL's

© Garth Gilmour 2008

Requesting JSF Pages

- URL's used by browsers do not map directly to the locations of JSP's using JSF
  - Instead the 'url-pattern' parameter of the Front Controller specifies a mapping to be used by clients
- This mapping can be based on a prefix or an extension
  - '*.faces' is recommended for an extension mapping
    - So if a browser requests 'jsfapp/jsp/mypage.faces' the 'FacesServlet' will find and include 'jsfapp/jsp/mypage.jsp'
  - For a prefix mapping '/faces/*' is recommended
    - So is a browser requests 'jsfapp/faces/mypage.jsp' the 'FacesServlet' will find and return 'jsfapp/mypage.jsp'

© Garth Gilmour 2008

JSF Web App - ‘web.xml’

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi=“…"  xsi:schemaLocation=“…">
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
    <listener>
        <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
    </listener>
    <welcome-file-list>
        <welcome-file>jsp/startup.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

© Garth Gilmour 2008

JSF Web App - ‘faces-config.xml’

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
      <!-- Configure navigation between pages -->
      <navigation-rule>
            <from-view-id>/jsp/jsf/salary.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>ok</from-outcome>
                  <to-view-id>/jsp/jsf/result.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>
      <!-- Configure JavaBeans -->
      <managed-bean>
            <description>The Employee Bean</description>
            <managed-bean-name>emp</managed-bean-name>
            <managed-bean-class>demos.jsf.beans.EmployeeBean</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
</faces-config>
```

© Garth Gilmour 2008

The Component Tree

- JSF Components must be arranged in a tree
  - The top level component must be of type ‘UIViewRoot’
  - Its children are the ‘top level’ components
  - These in turn may contain other components and so on
- This tree structure is reflected in a JSF page
  - All the actions on the page must be within a ‘<c:view>’ tag
  - Actions for an HTML form must be inside a ‘<h:form>’ tag
- As with JSP some JSF pages may be modular
  - They may be designed to be included by other pages
  - In this case the actions can be put inside a ‘<c:subview>’

© Garth Gilmour 2008

JSF Web App - Input Page

```
<%@page language="java" contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
   <body><h2>Please enter:</h2>
        <f:view>
            <h:form>
                <p>Your basic salary:<h:inputText size="10" value="#{emp.salary}"/></p>
                <p>Your monthly deductions:<h:inputText size="10" value="#{emp.deductions}"/></p>
                <p>
                    Your marital status:<br/>
                    <h:selectOneRadio value="#{emp.maritalStatus}">
                        <f:selectItem  itemValue="married" itemLabel="married"/>
                        <f:selectItem  itemValue="not married" itemLabel="not married"/>
                    </h:selectOneRadio>
                </p>
                <p><h:commandButton action="ok" value="CalculateSalary"/></p>
            </h:form>
        </f:view>
  </body>
</html>
```

© Garth Gilmour 2008

JSF Web App - JavaBean

```
public class EmployeeBean {
	//accessor methods for properties
	public void setSalary(double salary) { this.salary = salary; }
	public double getSalary() { return salary; }
	public void setDeductions(double deductions) { this.deductions = deductions; }
	public double getDeductions() { return deductions; }
	public void setMaritalStatus(String maritalStatus) { this.maritalStatus = maritalStatus; }
	public String getMaritalStatus() { return maritalStatus; }

	public double getWage() {
		boolean isMarried = "married".equalsIgnoreCase(maritalStatus);
		double tax_rate = isMarried ? married_tax_rate : single_tax_rate;
		double tax = salary * (tax_rate / 100);
		return ((salary - tax) / 12) - deductions;
	}
	private double salary;                                  //yearly gross salary
	private double deductions;                              //monthly deductions
	private String maritalStatus;                           //marital status
	private static final double single_tax_rate = 20;       //tax rate if not married
	private static final double married_tax_rate = 15;      //tax rate if married
}
```

© Garth Gilmour 2008

JSF Web App - Output Page

```
<%@page language="java" contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
  <head>
      <title>JavaServer Faces Demos</title>
  </head>
  <body>
      <f:view>
          <h:form>
              <p>Your monthly wage is <h:outputText value="#{emp.wage}"/></p>
          </h:form>
      </f:view>
  </body>
</html>
```

© Garth Gilmour 2008

The Tag Libraries in Detail

- The core library contains tags:
  - To serve as containers
  - To represent the built in validators and converters
  - To introduce your own validators and converters
  - To attach extra features to components
    - Such as registering a class as an event handler
  - To configure items within selection components
  - To load resource bundles
- The HTML library contains tags:
  - To represent the inputs of an HTML form
  - To represent output formatted by CSS and tables

© Garth Gilmour 2008

JSF Core Tag	Description
view	Represents the container holding all components on the page
subview	Represents a nested view (like a JPanel in Swing)
facet	Adds a facet to a component
attribute	Adds an attribute to a component
param	Adds a parameter to a component
actionListener	Registers a listener for Action Events
valueChangeListener	Registers a listener for Value Change Events
converter	Adds a converter class to a component
convertDateTime	Adds the built in converter for dates
convertNumber	Adds the built in converter for numbers
validator	Adds a validator class to a component
validateDoubleRange	Adds the built in double range validator
validateLength	Adds the built in length validator
validateLongRange	Adds the built in long range validator
loadBundle	Loads a resource bundle into a Map
selectItems	Specifies multiple items for a tag that offers a choice box
selectItem	Specifies a single item for a tag that offers a choice box
verbatim	Adds literal characters to a JSF page

© Garth Gilmour 2008

The JSF HTML Tag Library Part 1

JSF Structural HTML Tags	
form	Creates a new form
panelGrid	Arranges components inside a table
panelGroup	Groups multiple components as a unit

JSF Input HTML Tags	
inputText	Single line text box
inputTextArea	Multi line text box
inputSecret	Password style text box
inputHidden	A hidden HTML field
commandButton	A form button
commandLink	A link acting as a button
selectOneListbox selectManyListbox	List boxes allowing one or multiple selections
selectOneRadio	A group of radio buttons
selectBooleanCheckBox selectManyCheckbox	Check boxes allowing one or multiple selections
selectOneMenu selectManyMenu	A menu allowing one or multiple selections

© Garth Gilmour 2008

The JSF HTML Tag Library Part 2

JSF Output HTML Tags	
outputLabel	A label for another component
outputLink	A hypertext link
outputFormat	A single line built from multiple inputs
outputText	A single line of output
message	The most recent message
messages	All messages
graphicImage	An image
dataTable	A complex table
column	A column in a dataTable

© Garth Gilmour 2008

JSF Expression Language in Detail

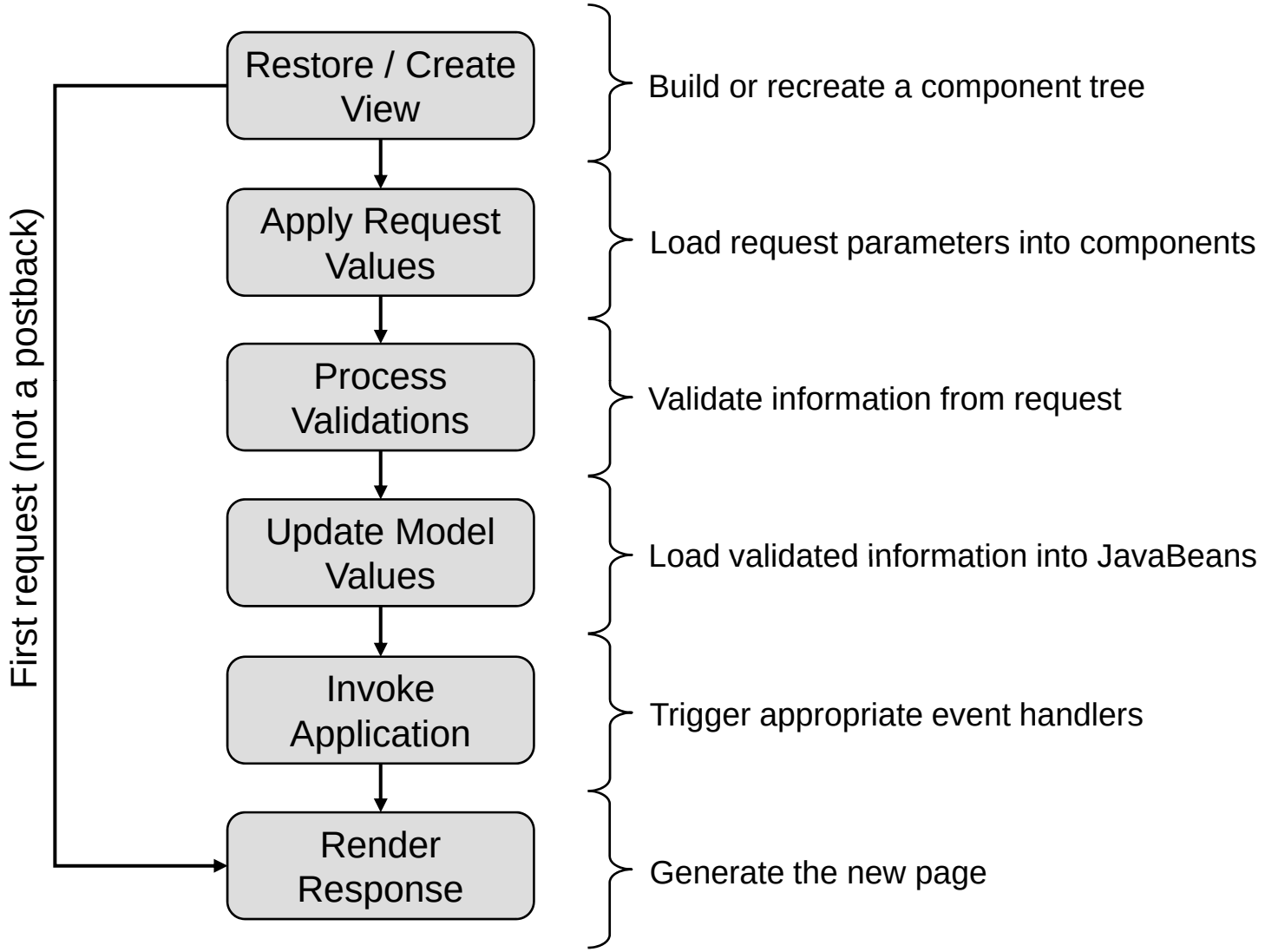
- JSF needed its own EL for two reasons:
  - Expressions need to be able to specify arbitrary methods
    - Which will be used as handlers for component events
  - The evaluation of expressions needs to be deferred till the component tree is being built and populated
    - As opposed to when the Server Page is called
- The two languages are being merged
  - JSF 1.2, JSP 2.1 and JSTL 1.2 will use a unified expression language with the capabilities of both
  - The '${ }' and '#{ }' syntaxes are still used to distinguish between immediate and deferred processing

© Garth Gilmour 2008

Events in the JSF Lifecycle

- The JSF Lifecycle has six phases:
  - Restore View
  - Apply Request Values
  - Process Validations
  - Update Model Values
  - Invoke Application
  - Render Response
- The lifecycle is simplified if:
  - A non JSF request has been received
  - A JSF request has been received for a non JSF page
  - In both these cases most of the phases can be skipped

© Garth Gilmour 2008



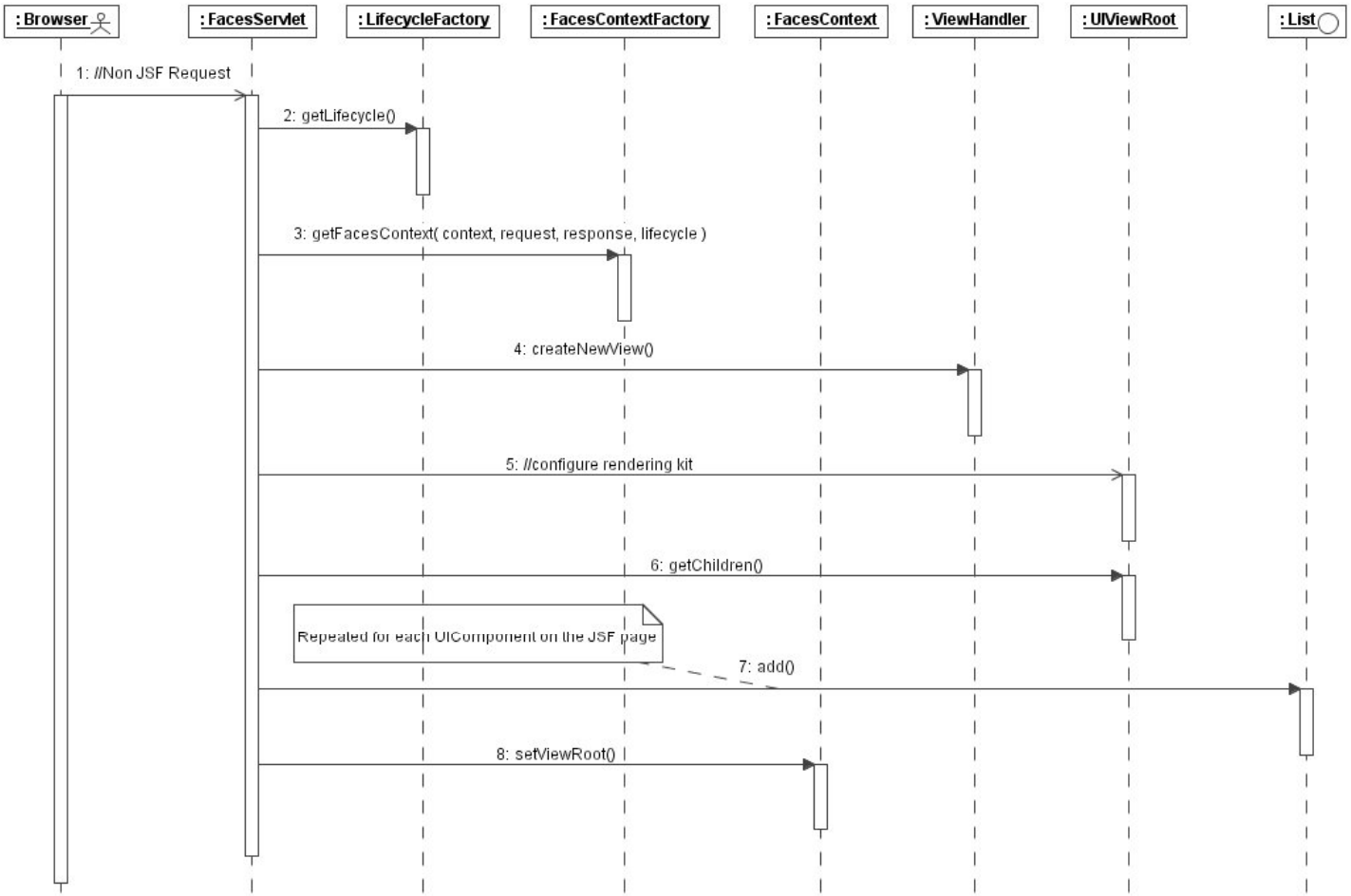
© Garth Gilmour 2008

Responding to a Non JSF Request

- When a non JSF request arrives we must:
  - Create a ‘Lifecycle’ object to manage state transitions
  - Create a ‘FacesContext’ object to represent the current state
    - Passing the lifecycle, context, request and response objects
  - Create a new ‘UIViewRoot’ object
    - To contain all the components required to build the client view
  - Populate the ‘UIViewRoot’ with ‘UIComponent’ objects
    - Represented by the corresponding actions on the JSF Page
  - Add the ‘UIViewRoot’ object to the ‘FacesContext’
- We can then proceed to the ‘Render Response’ phase

© Garth Gilmour 2008

Responding to a Non JSF Request



© Garth Gilmour 2008

Restore View

- If a request arrives for a previously viewed page we must restore its component tree
  - If the request is a post-back we use the current ‘UIViewRoot’
  - Otherwise we retrieve the appropriate ‘UIViewRoot’
    - By calling the ‘restoreView’ method of ‘ViewHandler’
    - The context object and view id are passed as parameters
      - For web applications the view id is the relative URL
    - If no ‘UIViewRoot’ can be found a new one must be created
- This completes the ‘Restore View’ phase of the life cycle
  - We have a component tree ready to be updated

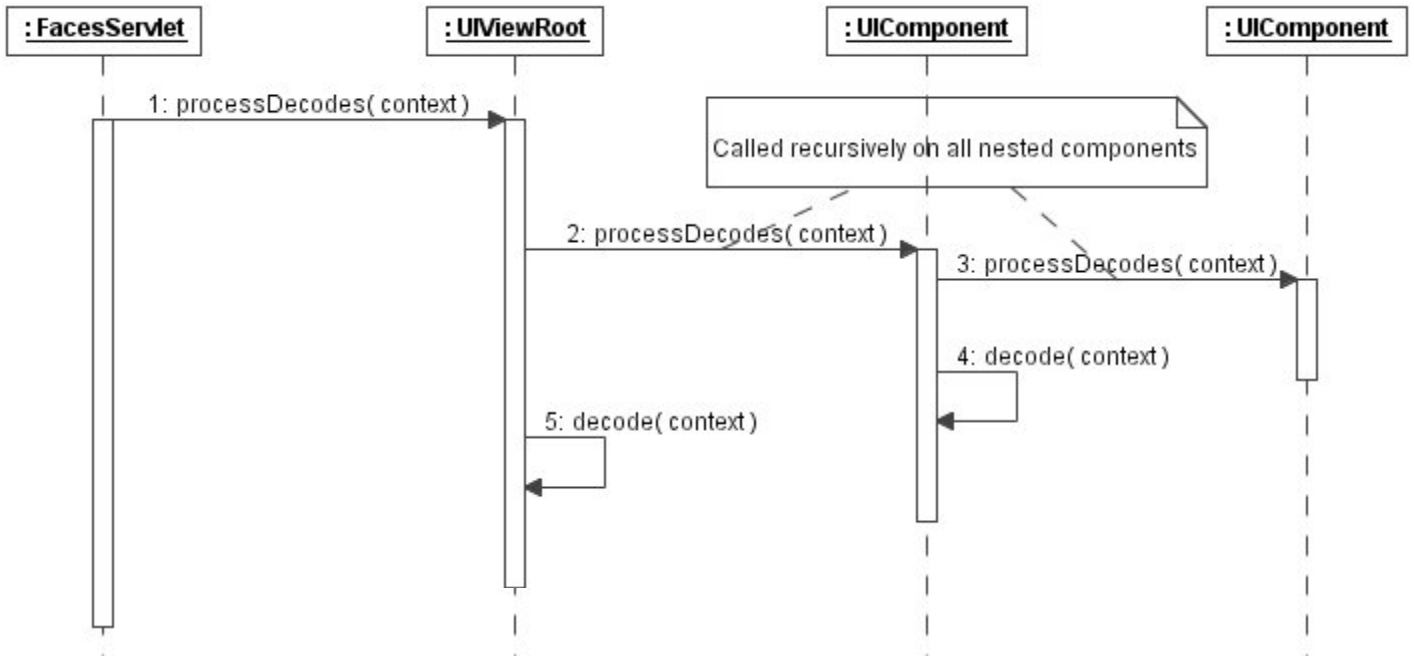
© Garth Gilmour 2008

Apply Request Values

- In this phase the ‘UIViewRoot’ calls ‘processDecode’ on each child component
  - This call is repeated recursively by the children
  - After the children have been notified each component calls its own ‘decode’ method
    - This extracts any information associated with the components state from the parameters of the request object
- At this stage all values are represented as strings
  - Conversion and validation happen in the next phase
  - An exception is if an input component has its immediate property set to true, in which case validation is carried out at once

© Garth Gilmour 2008

Activities on Apply Request Values



© Garth Gilmour 2008

Process Validations

- In this phase any conversion and/or validation objects associated with a component are applied
  - The state information is converted to an appropriate type and checked by the validator
  - If either of these fail the ‘valid’ property of the component is set to false and messages are logged with the context object
- The phase is initiated via the ‘processValidators’ method
  - As with ‘processDecodes’ this method call is recursively passed around the component tree
- Note that the model classes (JavaBeans) are still unused
  - These are updated during the next phase

© Garth Gilmour 2008

Update Model Values

- This phase is initiated via ‘processUpdates’
  - Which again is called on each component in the tree
- After passing the call on to its children a component calls its own ‘updateModel’ method
  - The updates the associated JavaBean with the converted and validated data from the request
  - This was set on the JSF page via the expression language
    - For example ‘<h:inputText value="#{emp.salary}"/>’
- At this point we have finished processing the request
  - The final phases are devoted to creating a response

© Garth Gilmour 2008

Invoke Application

- The ‘Invoke Application’ phase triggers event handling
  - Action Listeners are triggered first, followed by actions
- Both types of event correspond to bean methods
  - As specified in the attributes of actions on the JSF page
    - ‘<h:commandButton actionListener=“#{b.op1}” action=“#{b.op2}”>’
- Action Listeners respond to User Interface events
  - The associated methods take an ‘ActionEvent’ parameter
- Actions trigger business logic and affect navigation
  - The method returns a String which is mapped to the output page
  - This is similar to the ‘perform’ method of an Action class in Struts

© Garth Gilmour 2008

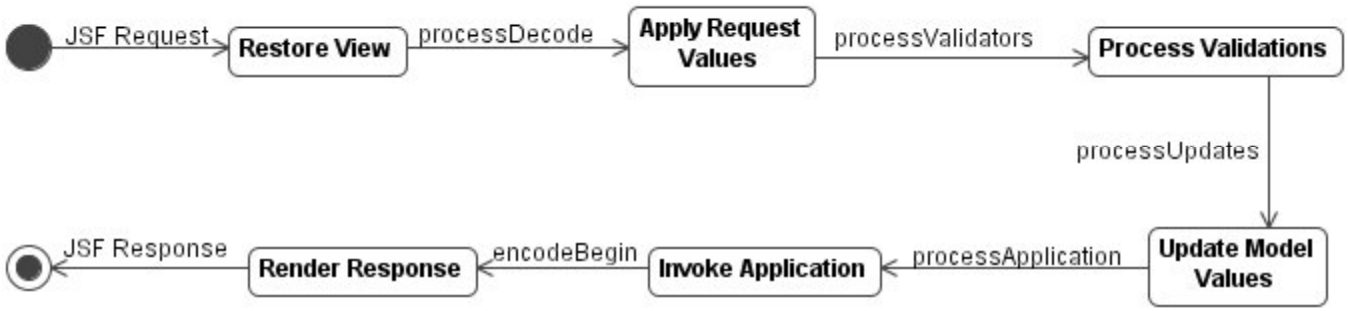
Render Response

- The details of this phase are implementation specific
  - When JSF is used in Web Applications the ‘ViewHander’ will forward the request to the output page
- Two activities occur in this phase
  - The output page is rendered into the response
    - Remember this is held within the ‘FacesContext’ object
  - The new component tree is saved
    - So it can be accessed again during the ‘Restore View’ phase
    - The ‘StateManager’ controls the saving and restoring of views
- Rendering the page means rendering each component
  - The methods ‘encodeBegin’, ‘encodeChildren’ and ‘encodeEnd’ are called on each component to let it write to the response

© Garth Gilmour 2008

The JSF Lifecycle

- The diagram below summarizes the lifecycle
- In addition to the normal state transitions:
  - Calling ‘responseComplete’ terminates the lifecycle
  - Calling ‘renderResponse’ transfers control to the final phase
  - Both these methods belong to the ‘FacesContext’ object



© Garth Gilmour 2008

Additional Types of Event

- Each phase of the lifecycle has its management method
  - These are ‘processDecodes’, ‘processValidators’, ‘processUpdates’ and ‘processApplication’
  - Before one of these methods is called any events queued in the previous phase must be dispatched
- One common event is a ‘ValueChangeEvent’
  - Components can use methods or classes as listeners
    - Methods of beans are registered using an attribute
    - Classes are registered using the ‘valueChangeListener’ action
      - The class must implement the ‘ValueChangeListener’ interface
  - The handler can be triggered after either ‘Apply Request Values’ or ‘Process Validation’, depending on the type of the component

© Garth Gilmour 2008