Java Server Faces (JSF)

7th October, 2006
Hyderabad, India

We appreciate

Overview

- One-tier solution, presentation, business logic, and data are all integrated in one monolithic application.
- The multi-tier architecture breaks this type of application into three layers
  model (data access),
  view (presentation),
  controller (logic).
- This programming paradigm, representing the split between these layers, is known as the **Model-View-Controller (MVC)** architecture

Overview

- A typical multi tier software solution—serving a retail company, for example—it might include support for multiple agents such as

  - Web browsers,
  - mobile devices etc..

- Application developer forced to provide one application that should support all 3 agents - will be a maintenance nightmare, and may cause issues with security and scalability.

Overview

"How many technologies do I
have to learn in order to successfully
build a complete solution for my project?"

Framework

open source communities

Struts (an open source controller framework);

TopLink and Hibernate (model frameworks);

Tiles, Tapestry, XUL, and ADF UIX (view frameworks).

Benefits of application frameworks – modularity, reusability, and inversion of control (IoC)

Restrictions: Frameworks are also incompatible

with each other, which makes integration hard to handle.

JSF Overview

- Why you need yet another framework ?
- The differentiator that JSF brings, which other similar frameworks do not have, is the backing of a standard specification (JSR-127).
- Because JSF is part of the J2EE standard specification, it is a top priority for every major J2EE tools vendor in the market (including Oracle, IBM, Borland, and Sun) to support it, which in turn will guarantee a wide adoption and good tools support.

What is JSF?

- A set of Web-based GUI controls and associated handlers
- A Component-based model - similar to the model that's been used in standalone GUI applications for years
- A device-independent GUI control framework
- JSF uses UI components that hide most of the grunt work of integrating richer functionality into Web applications.
- Provides an easy way to construct UIs from a set of reusable UI components.

JSF - Component Model

- JSF component model is similar to the AWT GUI component model.
- Events and Properties
- Component functionality typically centers around two actions: decoding and encoding data.
- **DIRECT IMPLEMENTATION** approach and **DELEGATED IMPLEMENTATION** approach
- JSF components consist of two parts: the **COMPONENT** and the **RENDERER**

JSF - Navigation Model

- JSF provides a declarative navigation model
- JSF provides a set of navigation rules to define the navigation from one view to another
- Navigation rules in JSF are defined inside the JSF configuration file, faces-config.xml, and are **page-based**

```
<navigation-rule>
    <from-view-id>/login.jsp</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/result.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

JSF - Application Lifecycle

- JSF framework helps manage **UI STATE** across server requests. Instead of having to take care of user elections and passing these selections from page to page, the framework will handle this
- JSF framework also has built-in processes in the Lifecycle to assist with **VALIDATION**, **CONVERSION**, and **MODEL UPDATES**.
- JSF provides a simple model for delivering client-generated **EVENTS** to server-side application code.

Example

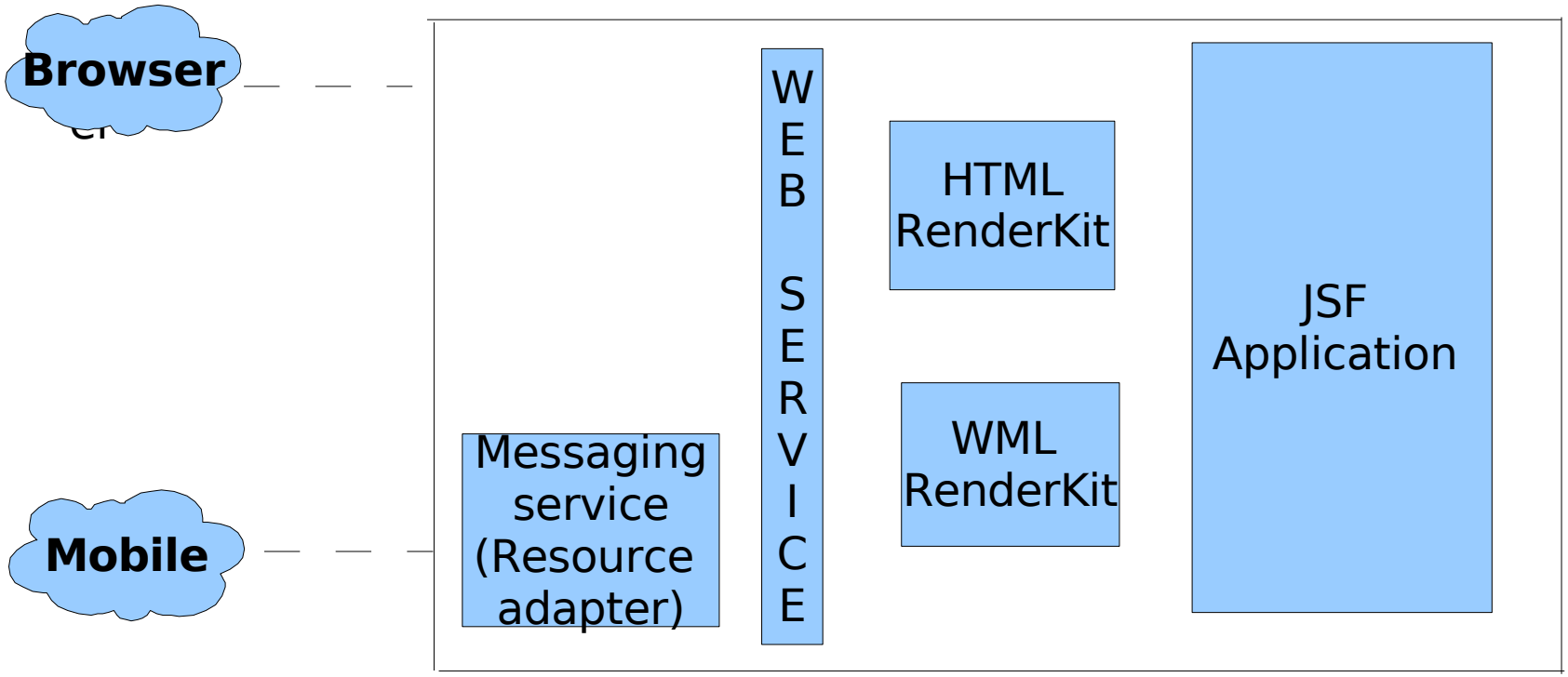
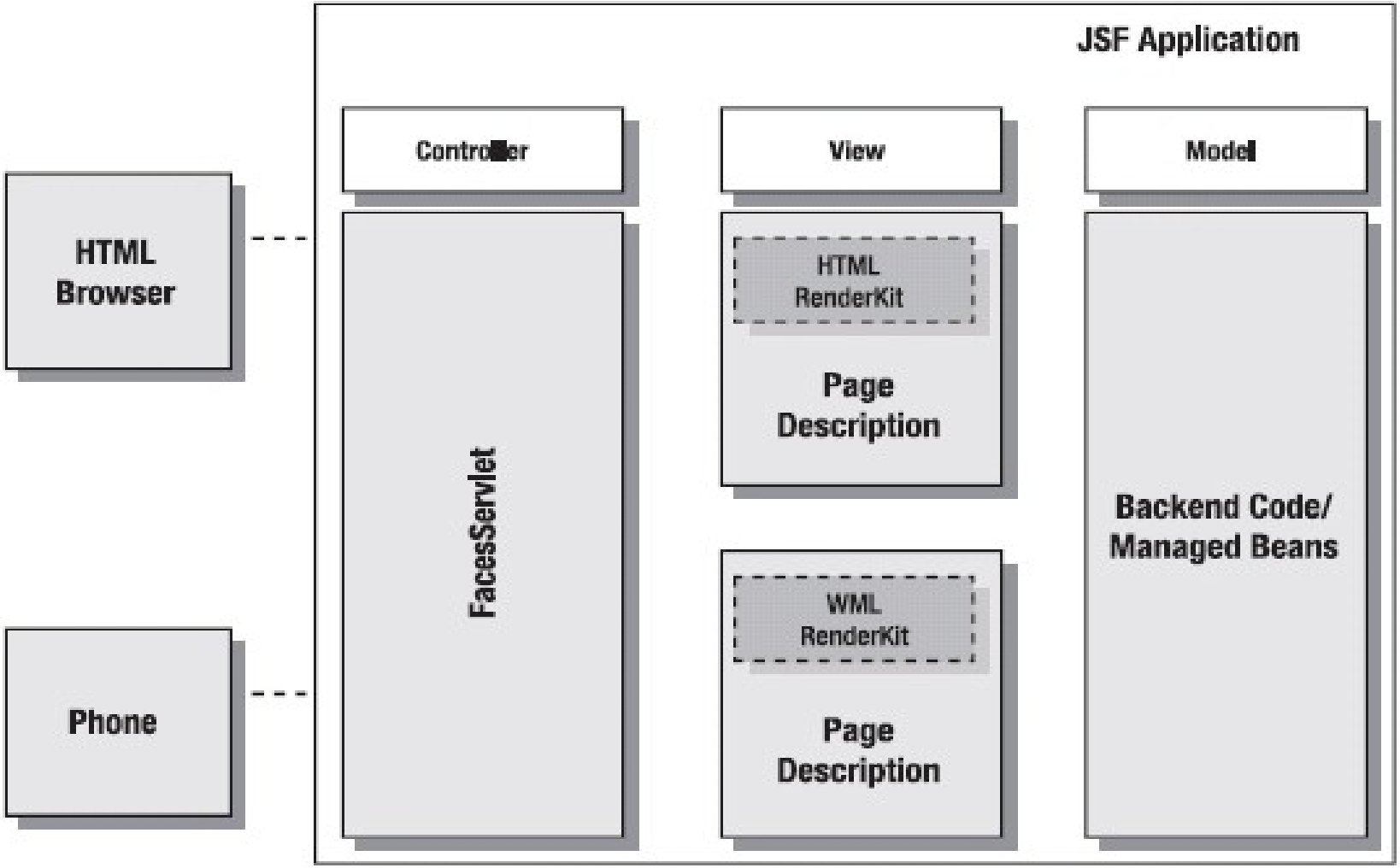


Figure J2EE architecture using JSF for a typical multi-tier software solution

JSF Architecture

- JSF implements - Model 2 pattern, which is based on the MVC architecture.
- Consists of three elements—the view, the navigation model, and the application logic



Model

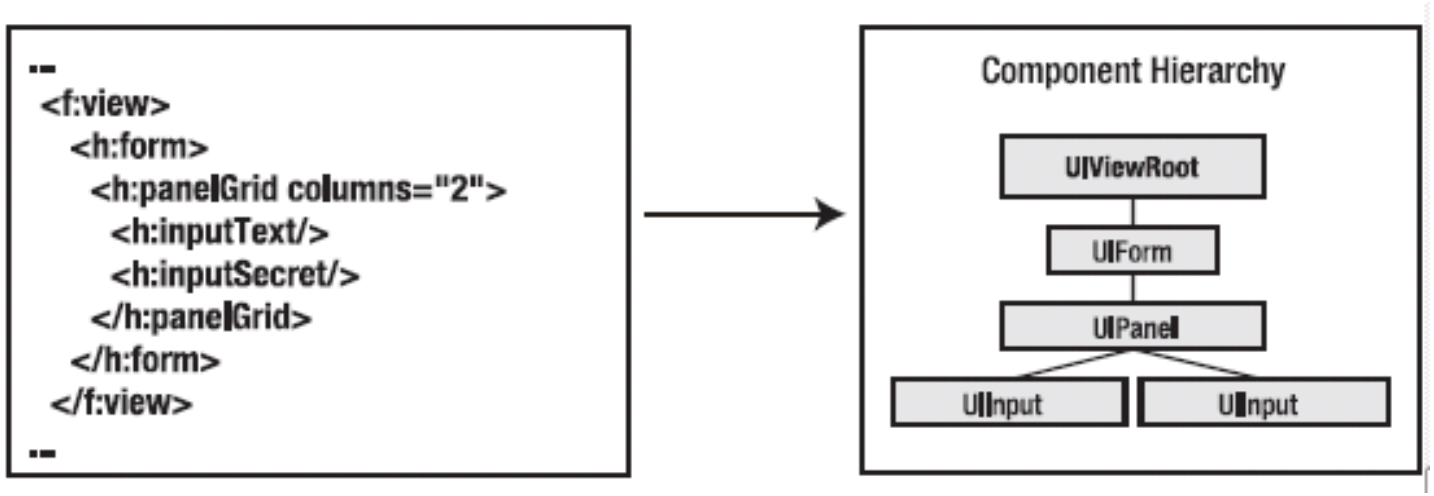
- Managed bean is the glue to the application logic—backing code or backing bean.
- In JSF it is the component that is aware of which action, or method, to call on a particular user event.
- The managed bean facility is responsible for creating the backing beans

```
<managed-bean>
    <managed-bean-name>sample</managed-bean-name>
    <managed-bean-class>com.application.SampleBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>

</managed-bean>
```

View

- JSF view layer describes the intended layout, behavior, and rendering of the application.



- UIComponents nested structure will at runtime be represented as a component hierarchy

Controller

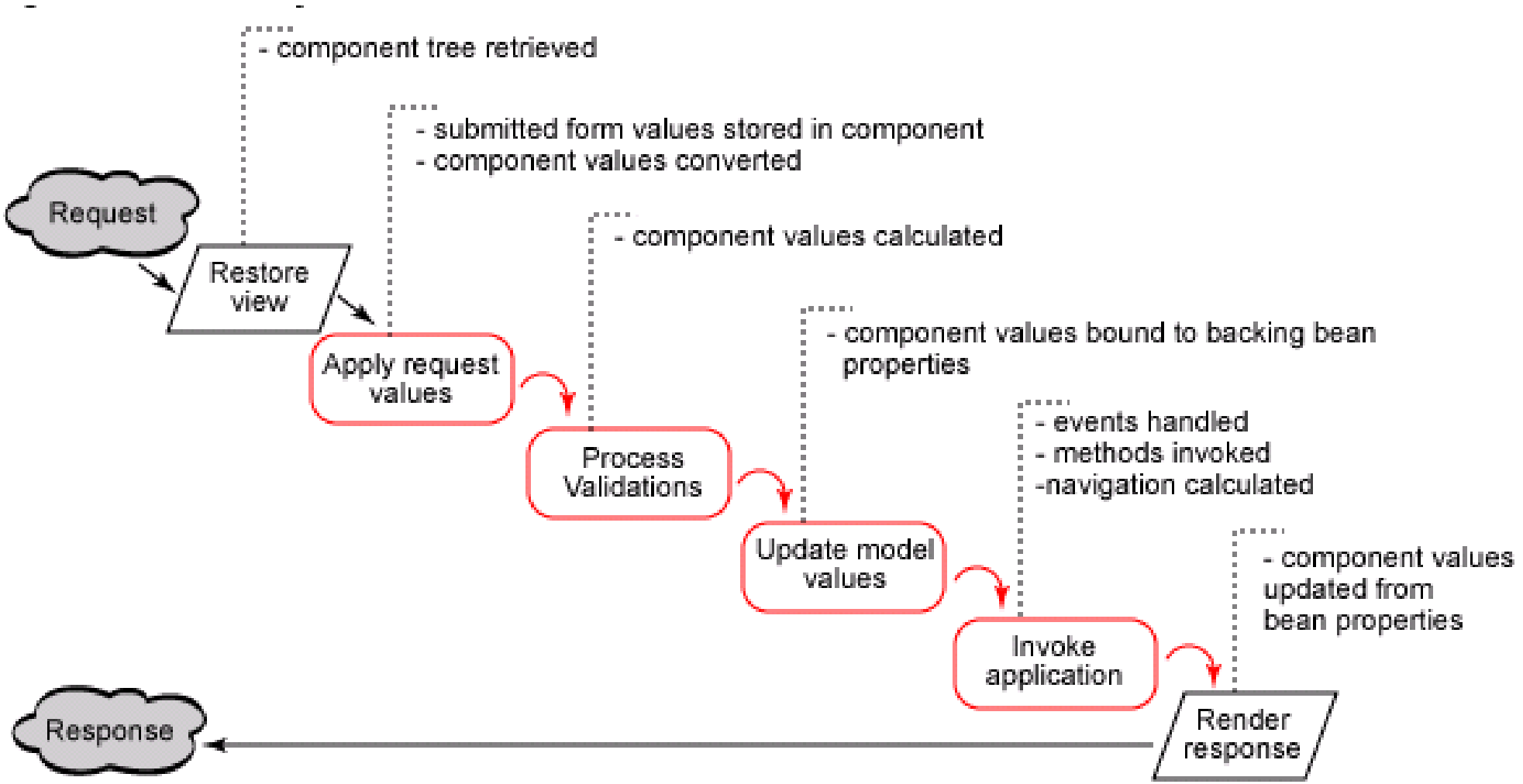
- JSF comes with a simple controller—the FacesServlet.
- Controlls Navigation Flow
- Dispatch Requests to appropirate page

A Simple JSF Application

Developing a simple JavaServer Faces application usually requires these tasks:

- Create the pages using the UI component and core tags.
- Define page navigation in the application configuration resource file. (faces-config.xml)
- Develop the backing beans. (backingbean/ managedbean)
- Add managed bean declarations to the application configuration resource file (faces-config.xml)
  - Refer example here....

The JSF lifecycle



Things to keep in mind.....

- FacesContext
  - The FacesContext object contains all the state information JSF needs to manage the GUI component's state for the current request in the current session.
  - The FacesContext stores the view in its viewRoot property
  - viewRoot contains all the JSF components for the current view ID.

Phase 1: Restore view

- In the first phase of the JSF lifecycle -- restore view -- a request comes through the FacesServlet controller.
- The controller examines the request and extracts the view ID, which is determined by the name of the JSP page.
- The JSF framework controller uses the view ID to look up the components for the current view.
- If the view doesn't already exist, the JSF controller creates it. If the view already exists, the JSF controller uses it. The view contains all the GUI components.

Phase 1: Restore view

- This phase - three view instances:

  new view,

  initial view, and

  postback, with each one being handled differently.

- New view - JSF builds the view of the Faces page and wires the event handlers and validators to the components. The view is saved in a FacesContext object.
- In the case of an initial view (the first time a page is loaded), JSF creates an empty view.

Phase 1: Restore view

- In the case of a postback (the user returns to a page that has been accessed previously), the view corresponding to the page already exists, so it needs only to be restored.
- In this case, JSF uses the existing view's state information to reconstruct its state.

Phase 2: Apply request values

- The purpose of the apply request values phase is for each component to retrieve its current state.
- The components must first be retrieved or created from the FacesContext object, followed by their values.
- Component values are typically retrieved from the request parameters, although they can also be retrieved from cookies or headers.

Phase 3: Process validation

- The first event handling of the lifecycle takes place after the apply request values phase.
- Values entered by the user are compared to the validation rules.
- If an entered value is invalid, an error message is added to FacesContext
- If there are no validation errors, JSF advances to the update model values phase
- If validation fails, JSF calls render response phase, which will display the current view with the validation error messages.

Phase 4: Update model values

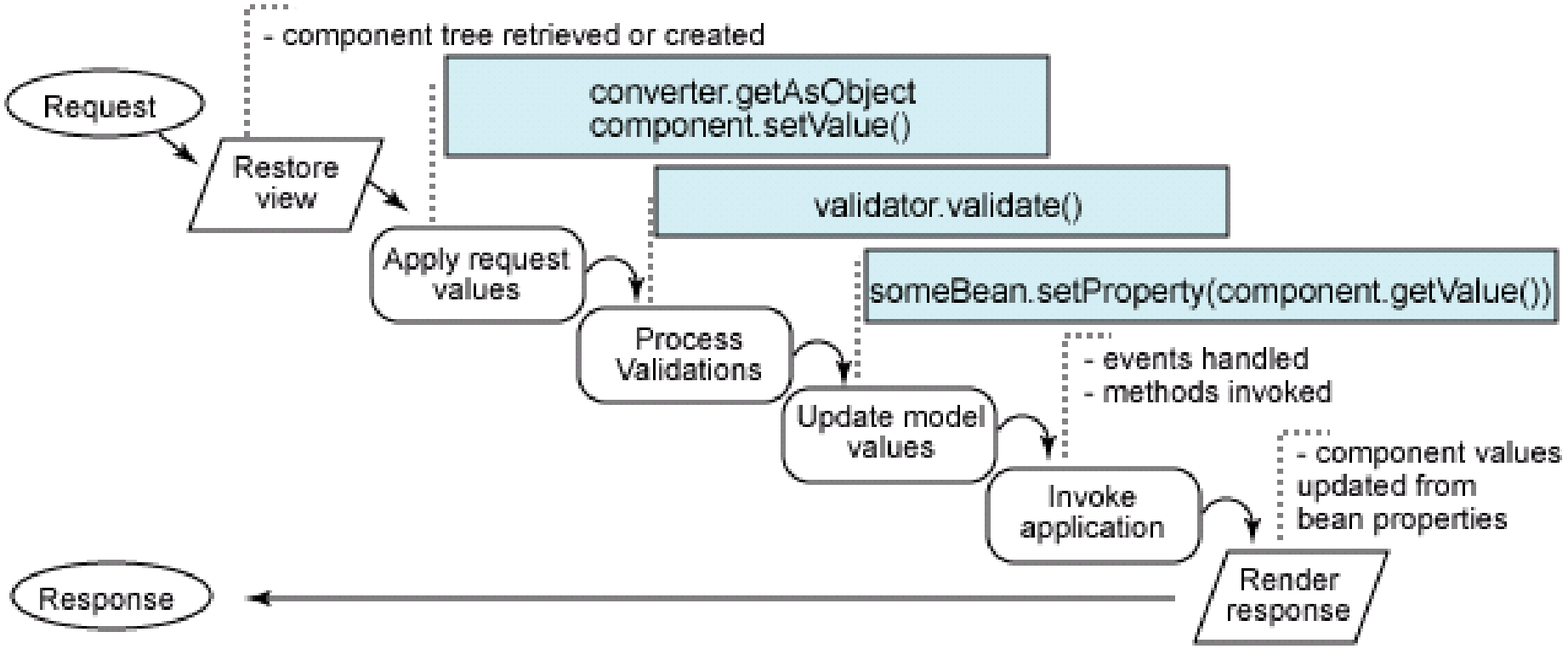
- Updates the actual values of the server-side model
- This is done by updating the properties of backing beans (managed beans).

Phase 5: Invoke application

- JSF controller invokes the application to handle Form submissions.
- At this phase, you also get to specify the next logical view for a given
- Define a specific outcome for a successful form submission and return that outcome

Phase 6: Render response

- In the sixth phase of the lifecycle -- render response -- you display the view with all of its components in their current state.



User Interface Component Model

JSF provides a rich, flexible component architecture that includes:

- A set of UICOMPONENT classes for specifying the state and behavior of UI components
- A RENDERING MODEL that defines how to render the components in various ways
- An EVENT and LISTENER MODEL that defines how to handle component events
- A conversion model that defines how to register data CONVERTERS onto a component
- A VALIDATION MODEL that defines how to register validators onto a component

UI Component class

JSF provides a set of

- UI COMPONENT CLASSES
- ASSOCIATED BEHAVIORAL INTERFACES

that specify all the UI component functionality like

- HOLDING COMPONENT STATE,
- MAINTAINING A REFERENCE TO OBJECTS
- EVENT HANDLING
- RENDERING

UI Component class

All JSF UI component classes extend *UIComponentBase*

refer block diagram in "JSF Custom Components"

UI component classes is includes: UIColumn, UICommand, UIData, UIForm, UIGraphic, UIInput, UIMessage, UIMessages, UIOutput, UIPanel, UIParameter, UISelectBoolean, UISelectMany, UISelectOne, UIViewRoot

Behavioral Interfaces include: ActionSource, EditableValueHolder, NamingContainer, StateHolder, ValueHolder

UI Component class

ex:

UICommand implements ActionSource and StateHolder.

UIOutput implement StateHolder and ValueHolder.

UIInput implement EditableValueHolder, StateHolder, and ValueHolder.

UIComponentBase implements StateHolder

Component Rendering Model

Renderer class defines a different way to render the particular component to the output defined by the render kit.

For example,

a UISelectOne component has three different renderers.

- One of them renders the component as a set of radio buttons.
- Another renders the component as a combo box.
- The third one renders the component as a list box.

refer table

UISelectOne

selectOneRadio

```
<h:selectOneRadio
 value="#|carBean.currentCar]">
  <f:selectItems
   value="#|carBean.carList]" />
</h:selectOneRadio>
```

selectOneMenu

```
<h:selectOneMenu id="selectCar"
 value="#|carBean.currentCar]">
  <f:selectItems
   value="#|carBean.carList]" />
</h:selectOneMenu>
```

selectOneListbox

```
<h:selectOneListbox id="pickCar"
 value="#|carBean.currentCar]">
  <f:selectItems
   value="#|carBean.carList]" />
</h:selectOneListbox>
```

Conversion Model

- JSF implementation automatically converts component data between these model view and presentation view

For example,

```
<h:inputText value="#{sample.date}" >
   <f:convertDateTime pattern="yyyy-MMM-dd" />
   </h:inputText>
```

java.util.Date - represented as a text string in the format mm/dd/yy etc..

UISelectBoolean component is associated with a bean property of type java.lang.Boolean,
JSF implementation will automatically convert the component's data from String to Boolean.

Event and Listener Model

JSF supports three kinds of events: value-change events, action events, and data-model events.

- An action event occurs when the user activates a component that implements ActionSource. ex: buttons and hyperlinks.
- A value-change event occurs when the user changes the value of a component represented by UIInput or one of its subclasses ex: checkbox.
- A data-model event occurs when a new row of a UIData component is selected

```
<h:commandButton
        styleClass="text"
        id="start"
        action="#{MyNewAction.startAction}"
        value="#{msgs['deploy.start']}"
        rendered="#{MyNewAction.startRender}"
/>
```

```
<h:selectOneListbox
    styleClass="selectListBoxStyle" size="10"
      value="#{MyNewAction.selectedCategoryId}"
      valueChangeListener="#{MyNewAction.onSelectCategoryValueChange}"
      id="categoryId1"
      onchange="form.submit();" required="true"

       <f:selectItems value="#{MyNewAction.categorySelectItems}"
      id="categorySelectItem1"/>
</h:selectOneListbox>
```

Mixing JSTL and JSF

- If you're using JSP or JSTL expressions with managed beans, you need to ensure that the beans have been created first,
  - either by a JSF expression,
  - Java code, or
  - custom tag
- older expression languages don't know about JSF's Managed Bean Creation facility

Mixing JSTL and JSF

```
...
<f:view>
  <jsp:useBean class="org.jia.examples.TestForm" id="exampleBean" scope="session"/>
  <h:form>
    <h:inputText id="inputInt" value="#{sessionScope.exampleBean.number}"/>
    <h:commandButton value="Go!"/>
      <c:if test="${sessionScope.exampleBean.number > 0}">
        <c:forEach begin="0" end="${sessionScope.exampleBean.number - 1}" var="count">
          Queen Tracey will achieve world domination.<br>
        </c:forEach>
      </c:if>
  </h:form>
</f:view>
```

Resources

- Javaserver Faces in Action – Kito D.mann
- Mastering Java Server Faces – Bill Dudney
- JSF - OReilly
- Pro JSF and AJAX – Jonas and John - APress
- http://www-128.ibm.com/developerworks/java

Special thanks

CommVault Systems (India) Pvt. Ltd.

http://www.commvault.com/

CommVault is a Storage Management, Backup and Disaster Recovery company incorporated in USA with its Global Development Centre in Hyderabad, AP.

About us ...

Charter
Promote, Develop and Showcase Open Source software.

website
www.twincling.org

more info
info@twincling.org

irc
#twincling

mailing list
groups.yahoo.com/group/twincling

helpline
+91-99494 96414

forum (software --> twincling)
http://www.nabble.com/twincling-f15741.html