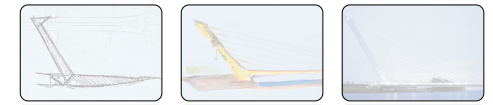


Bitter JavaServer™ Faces

An adopter's report

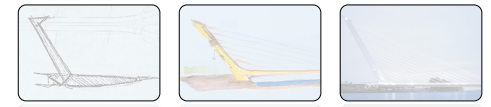
Stijn Van den Enden
s.vandenenden@aca-it.be





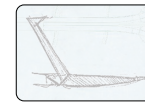
Overall Presentation Goal

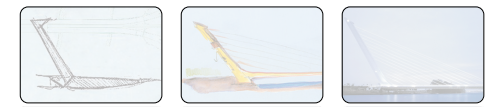
Learn how to avoid some common issues people face when adopting JavaServer Faces technology (JSF™)



Speaker's Qualifications

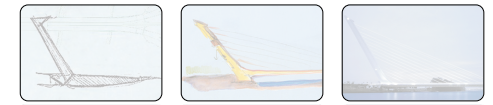
- Stijn Van den Enden is a J2EE architect for ACA IT-Solutions
- Has been using J2EE technology extensively since 1999
- Frequently conducts classes on J2EE, Architecture and Development Methodologies
- Specializes in component based software engineering, workflow and SOA





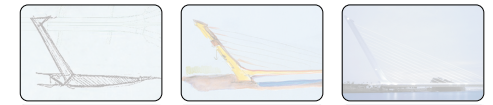
Agenda

- Common pitfalls when starting with JSF
 - Lifecycle Subtleness
 - Mismatch JSP and JSF rendering
 - Problems Handling Tabular Data
- Tips and Tricks
 - Handling Page Flow
 - How to Save State
 - Avoid scattered validation logic
 - Testing JSF Applications

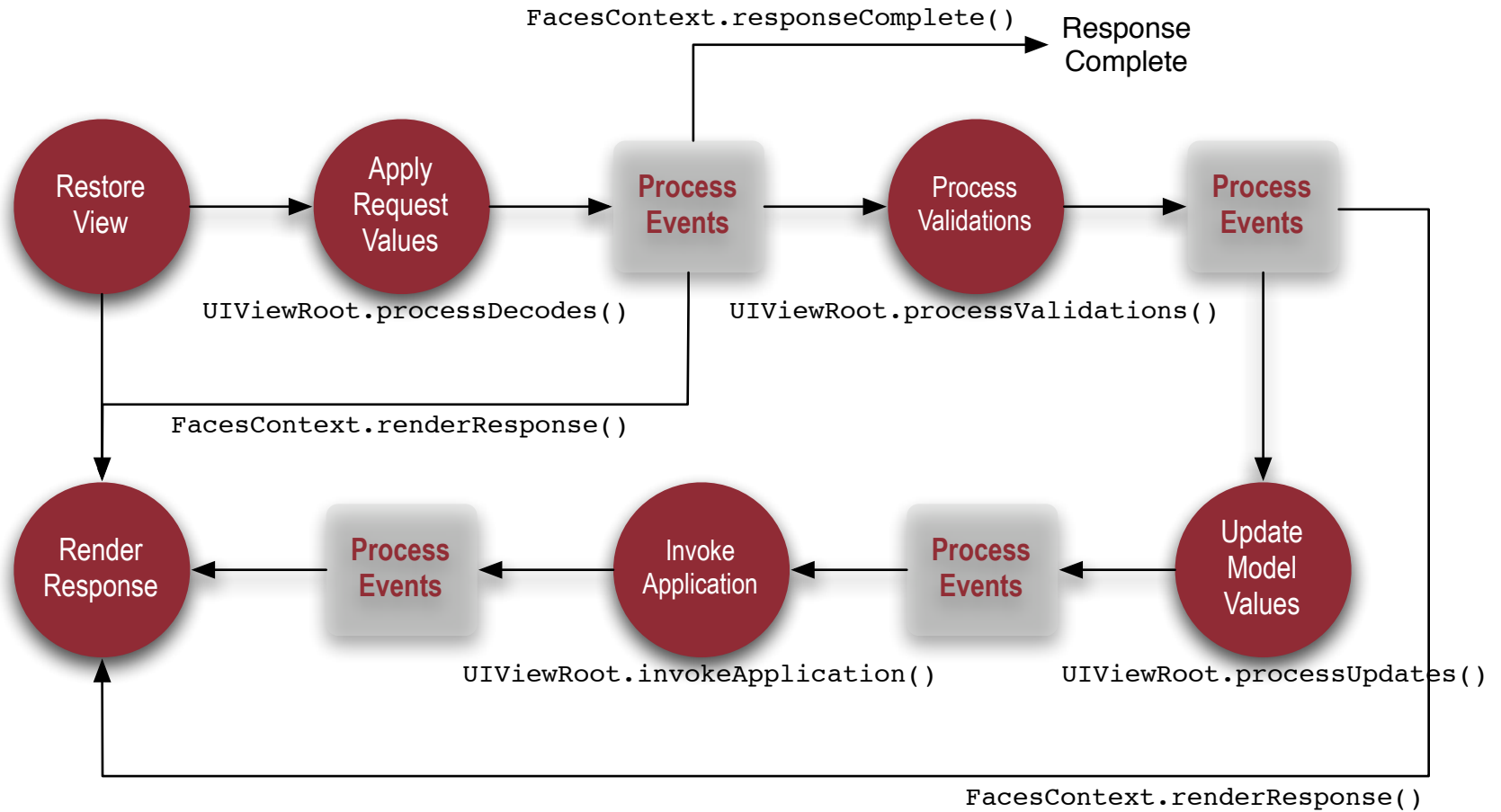


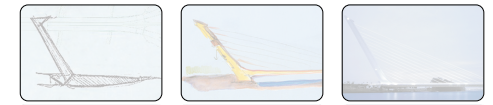
Agenda

- **Common pitfalls when starting with JSF**
 - **Lifecycle Subtleness**
 - Mismatch JSP and JSF rendering
 - Problems Handling Tabular Data
- Tips and Tricks
 - Handling Page Flow
 - How to Save State
 - Avoid scattered validation logic



Lifecycle Subtleness



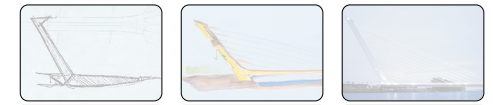


Lifecycle Subtleness

Avoid HTTP Centric Thinking

- Usage of hidden fields in forms

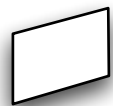
```
<h:form>
  ...
  <h:inputHidden value="#{bean.property}" />
  ...
  <h:commandButton id="click"
                    action="#{bean.click}"/>
</h:form>
```



Lifecycle Subtleness

Avoid HTTP Centric Thinking

- Usage of hidden fields in forms
 - Not a generic state saving mechanism
 - Doesn't work if `property` is needed before *Update Model Phase*



```
<h:form>  
  ...  
  <h:inputHidden value="#{bean.property}"/>  
  ...  
</h:form>
```

Render Response

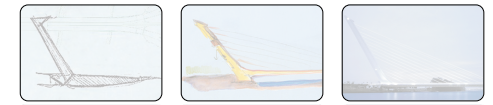


Update Model



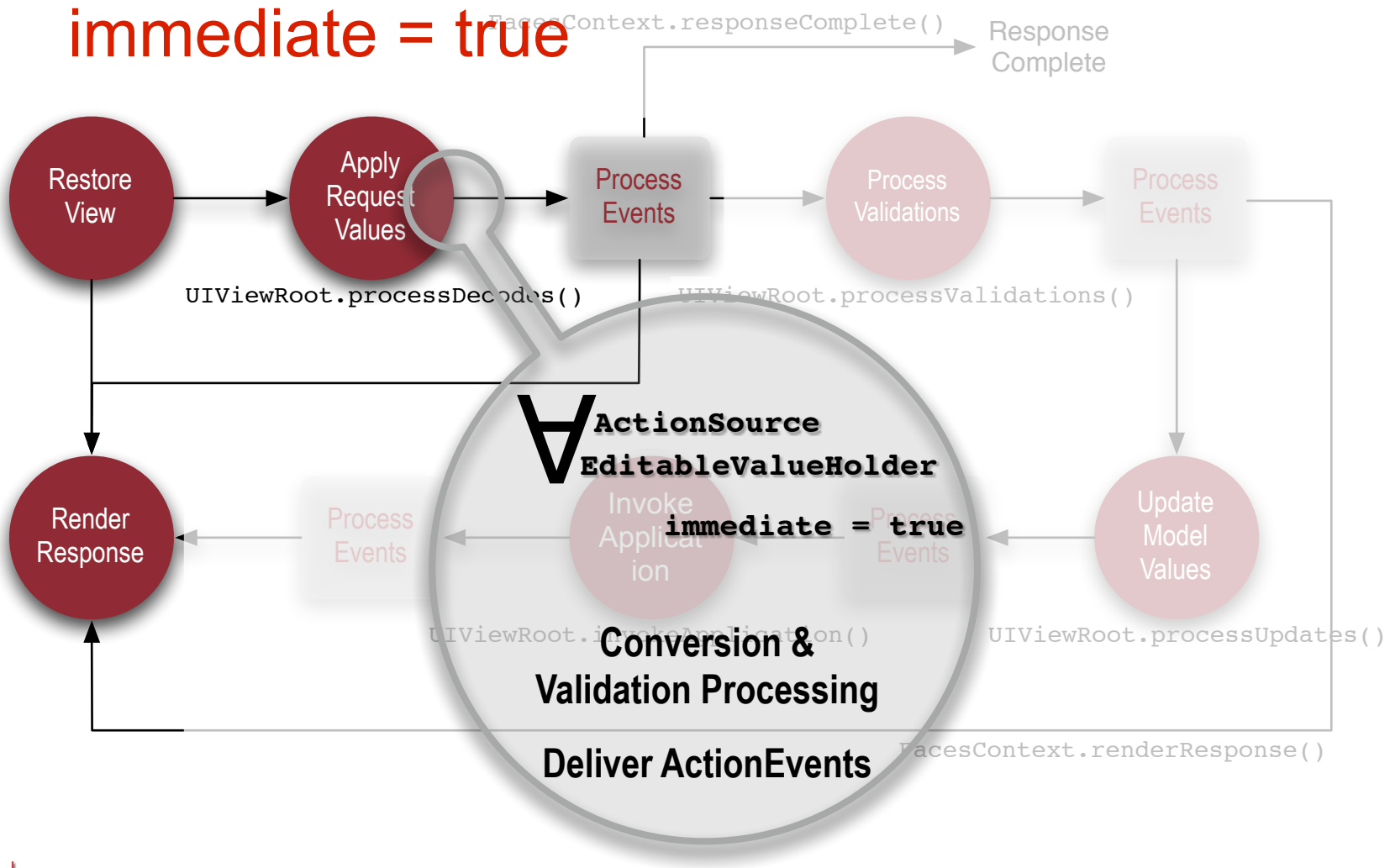
property

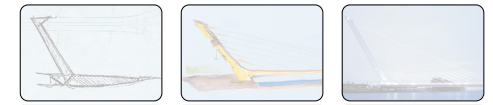
BackingBean
property
getProperty():String
setProperty():String



Lifecycle Subtleness

immediate = true





Lifecycle Subtleness

And then it goes wrong

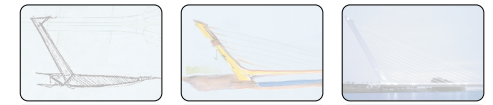
...

ZipCode

Validate

City

```
<h:inputText id="zipCode"
  value="#{bean.zipcode}"
  immediate="true"/>
<h:commandButton id="validate"
  action="#{bean.validate}"
  immediate="true"/>
...
<h:inputText id="city"
  value="#{bean.city}"/>
```



Lifecycle Subtleness

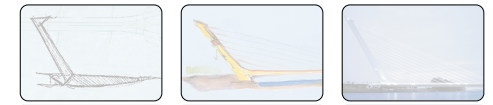
And then it goes wrong

```
<h:inputText id="zipCode"
    value="#{bean.zipcode}"
    immediate="true"/>
<h:commandButton id="validate"
    action="#{bean.validate}"
    immediate="true"/>
```

```
...
<h:inputText id="city"
    value="#{bean.city}"/>
```

```
private int zipCode;
private String city;
```

```
public String validate() {
    try{
        setCity(
            zipCodeHelper.getCity(getZipCode());
        );
    }
    catch(InvalidZipCodeException e) {
        //Add Message ...
    }
    return null;
}
```



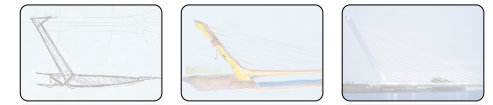
DEMO



Lifecycle example

www.aca-it.be





Lifecycle Subtleness

Problems

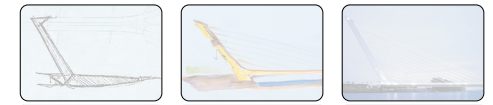
- ① `getZipCode()` will always return 0
- ② `validate()` will always be called even if conversion fails

```
private int zipCode;  
private String city;
```

②

```
public String validate() {  
    try{  
        setCity(  
            zipCodeHelper.getCity(  
                getZipCode() ) )  
        );  
    }  
    catch(InvalidZipCodeException e) {  
        //Add Message ...  
    }  
    return null;  
}
```

①



Lifecycle Subtleness

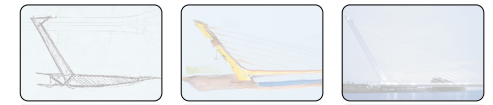
Problems

- if city would get initialized,
③ the value binding will not be called

```
private int zipCode;  
private String city;
```

②

```
public String validate() {  
    try{  
        setCity( ③  
                zipCodeHelper.getCity( ①  
                                     getZipCode() ) )  
    };  
    }  
    catch(InvalidZipCodeException e) {  
        //Add Message ...  
    }  
    return null;  
}
```



Lifecycle Subtleness

Cause of ③

Apply
Request
Values

submittedValue = <value>

City

Process
Validation

submittedValue = null
value = converted<value>

Update
Model

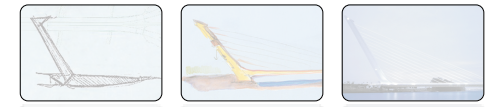
submittedValue = null
value = null

Render
Response

getValue() — returns
↓
submittedValue
value
evaluate value binding

EditableValueHolder

```
getLocalValue():Object  
isLocalValueSet():boolean  
isValid:boolean  
  
getValue():Object  
setValue(Object):void  
  
getSubmittedValue():Object  
setSubmittedValue(Object):void
```

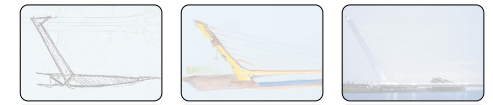


Lifecycle Subtleness

immediate = true

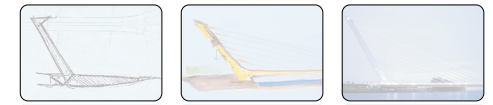
- Unexpected behavior can be hard to trace
- Use it with care, ideal for
 - bypassing validation when navigating
 - enabling/disabling parts of the screen

- But be cautious for the effect on `EditableValueHolder`



Agenda

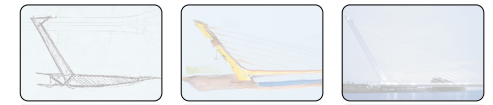
- **Common pitfalls when starting with JSF**
 - Lifecycle Subtleness
 - **Mismatch JSP and JSF rendering**
 - Problems Handling Tabular Data
- Tips and Tricks
 - Handling Page Flow
 - How to Save State
 - Avoid scattered validation logic



Mismatch JSP and JSF Rendering

Symptoms

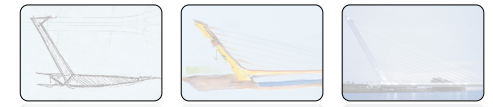
```
<body>
  <f:view>
    <h:panelGroup styleClass="portlet-menu">
      <ul>
        <li>
          <h:outputText
            value="#{bb.message}" />
        </li>
        <li>Second, template text</li>
      </ul>
    </h:panelGroup>
  </f:view>
</body>
```



Mismatch JSP and JSF Rendering

Symptoms

```
<body>
  <ul>
    <li></li>
    <li>Second, template text</li>
  </ul>
  <span class="portlet-menu">
    First, h:output
  </span>
</body>
```



Mismatch JSP and JSF Rendering

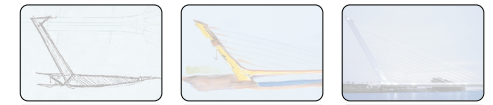
Cause

JSP

```
<body>
  <f:view>
    <h:panelGroup styleClass="portlet-menu">
      <ul>
        <li><h:outputText
          value="#{bb.message}" /></li>
        <li>Second, template text</li>
      </ul>
    </h:panelGroup>
  </f:view>
</body>
```

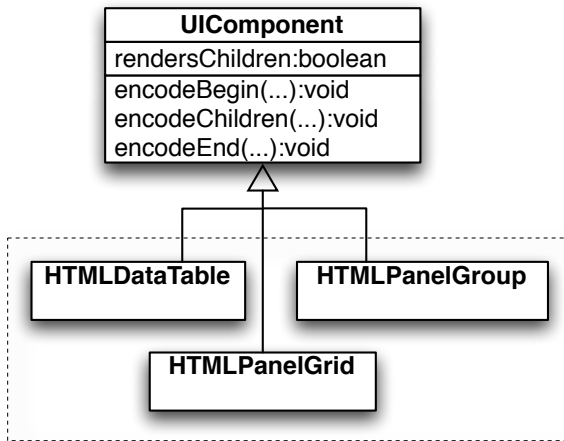
Servlet

```
panelGroup.doStartTag();
out.write("<ul>\n");
out.write("<li>\n");
  outputText.doStartTag();
  outputText.doEndTag();
out.write("</li>\n");
out.write("<li>Second, temp");
out.write("</ul>\n");
panelGroup.doEndTag();
```

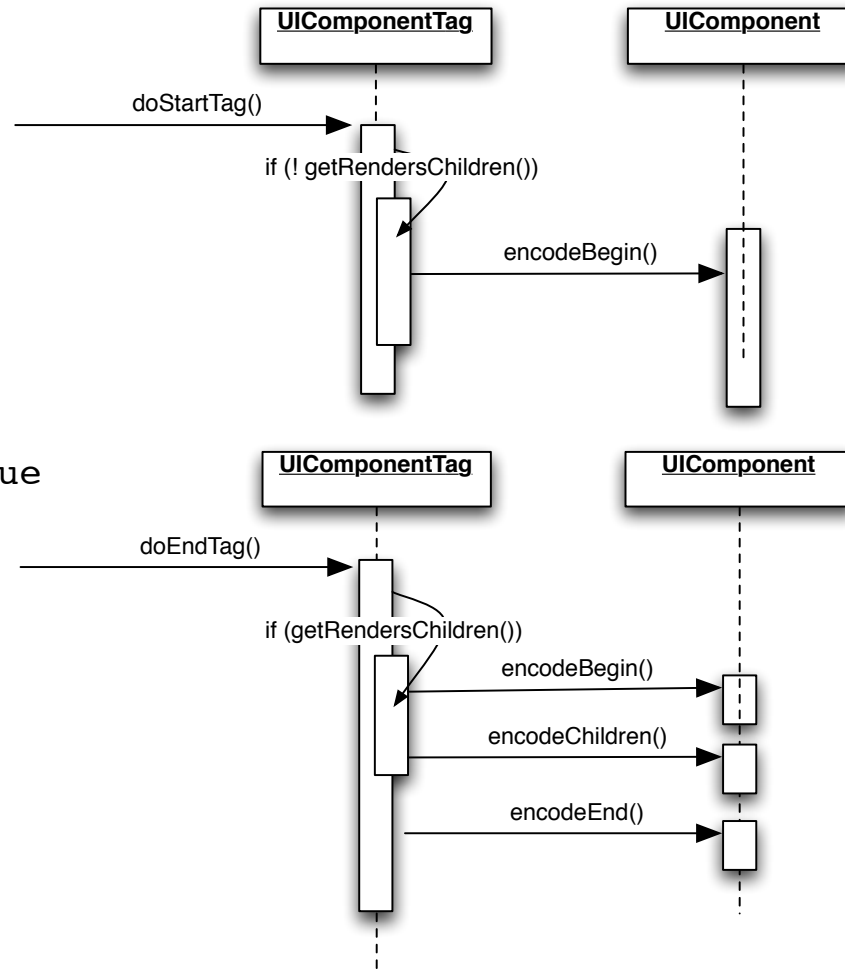


Mismatch JSP and JSF Rendering

Cause



rendersChildren = true





Mismatch JSP and JSF Rendering

Cause

JSP

```
<body>
  <f:view>
    <h:panelGroup styleClass="portlet-menu">
      <ul>
        <li><h:outputText
          value="#{bb.message}" /></li>
        <li>Second, template text</li>
      </ul>
    </h:panelGroup>
  </f:view>
</body>
```

Servlet

```
panelGroup.doStartTag();
out.write("<ul>\n");
out.write("<li>\n");
  outputText.doStartTag();
  outputText.doEndTag();
out.write("</li>\n");
out.write("<li>Second, temp");
out.write("</ul>\n");
panelGroup.doEndTag();
```

HTML

```
<ul>
  <li>      </li>
  <li>Second, template text</li>
</ul>
<span class="portlet-menu">
  First, h:output
</span>
```

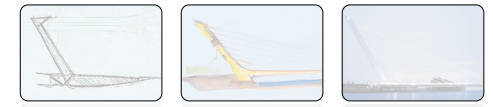


Mismatch JSP and JSF Rendering

Solution

- Use the `<f:verbatim>` custom action

```
<body>
  <f:view>
    <h:panelGroup styleClass="portlet-menu">
      <f:verbatim>
        <ul>
          <li>
            </f:verbatim>
            <h:outputText
              value="#{bb.message}" />
            <f:verbatim>
              </li>
              <li>Second, template text</li>
            </ul>
          </f:verbatim>
        </h:panelGroup>
      </f:view>
    </body>
```



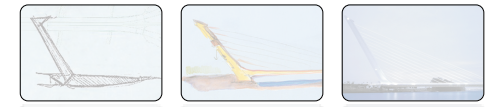
Mismatch JSP and JSF Rendering

Solution

- Use the HTML Layout tag library*

```
<body>
  <f:view>
    <h:panelGroup styleClass="portlet-menu">
      <htm:ul>
        <htm:li>
          <h:outputText
            value="#{bb.message}"/>
        </htm:li>
        <htm:li>
          <h:outputText
            value="Second, template text"/>
        </htm:li>
      </htm:ul>
    </h:panelGroup>
  </f:view>
</body>
```

* See <http://www.jsftutorials.net/htmlLib>



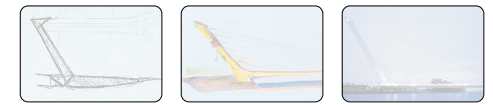
Mismatch JSP and JSF Rendering

Solution

- Use another ViewHandler*

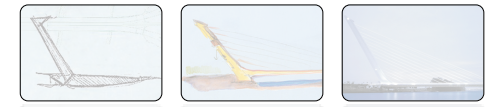
```
<?xml version="1.0" encoding="UTF-8"?>
...
<body>
  <span jsfc="h:panelGroup" style="portlet-menu"
    id="portletMenu">
    <ul>
      <li>
        ${bb.message}
      </li>
      <li>
        Second, template Text
      </li>
    </ul>
  </span>
</body>
</html>
```

* See <http://facelets.dev.java.net>



Agenda

- **Common pitfalls when starting with JSF**
 - Lifecycle Subtleness
 - Mismatch JSP and JSF rendering
 - **Problems Handling Tabular Data**
- Tips and Tricks
 - Handling Page Flow
 - How to Save State
 - Avoid scattered validation logic

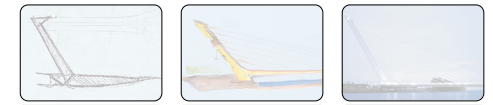


Handling Tabular Data

Symptoms

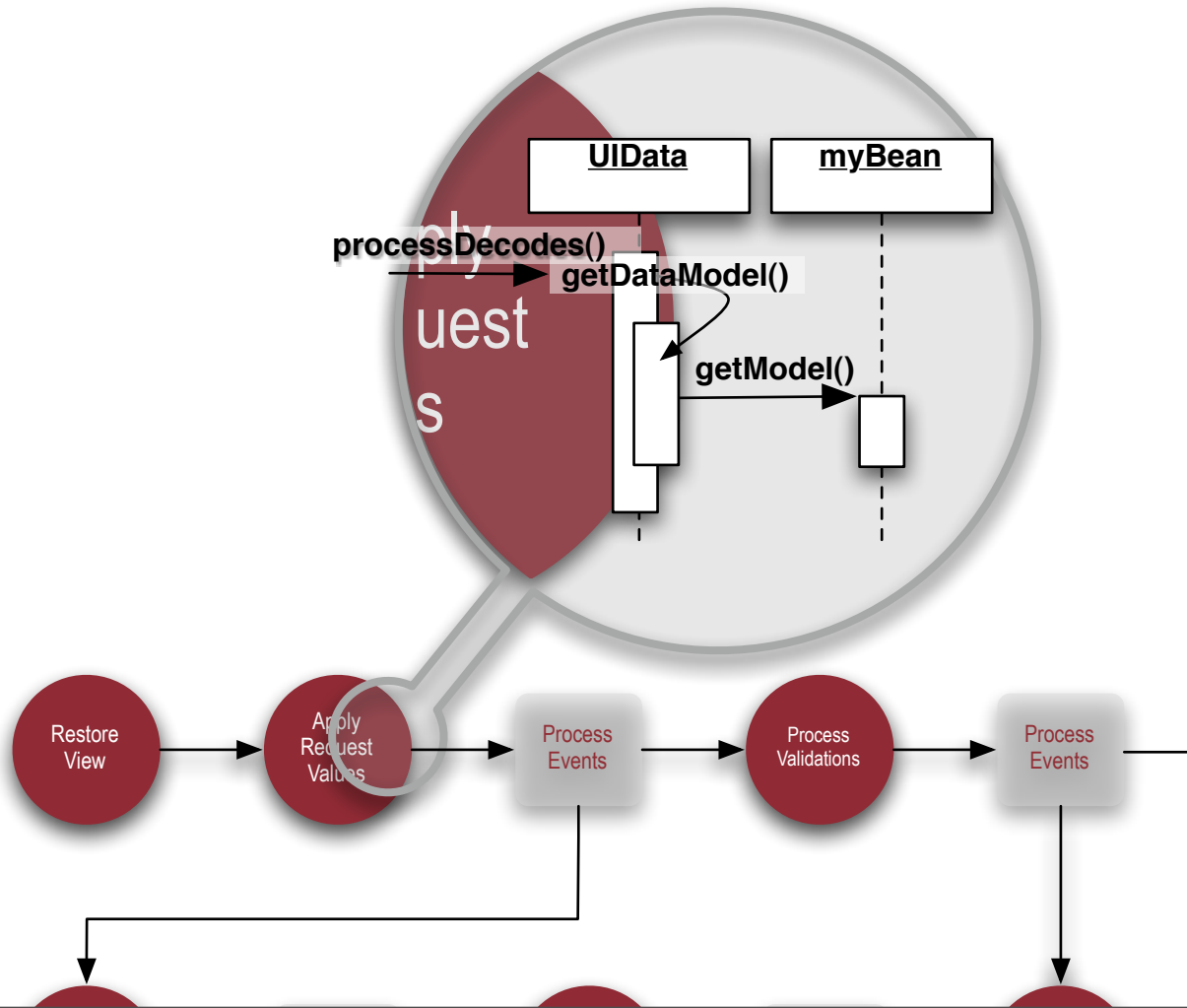
- Hides the complexity of handling Tabular Data
- However issues arise:
 - `getRowData()` returns wrong data
 - No events get fired
 - Error during response rendering
 - ...

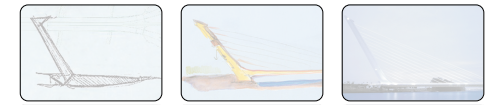
DataModel
<code>getRowData():Object</code>
<code>getRowIndex():int</code>
...



Handling Tabular Data

Cause

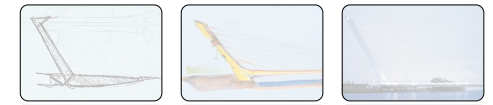




Handling Tabular Data

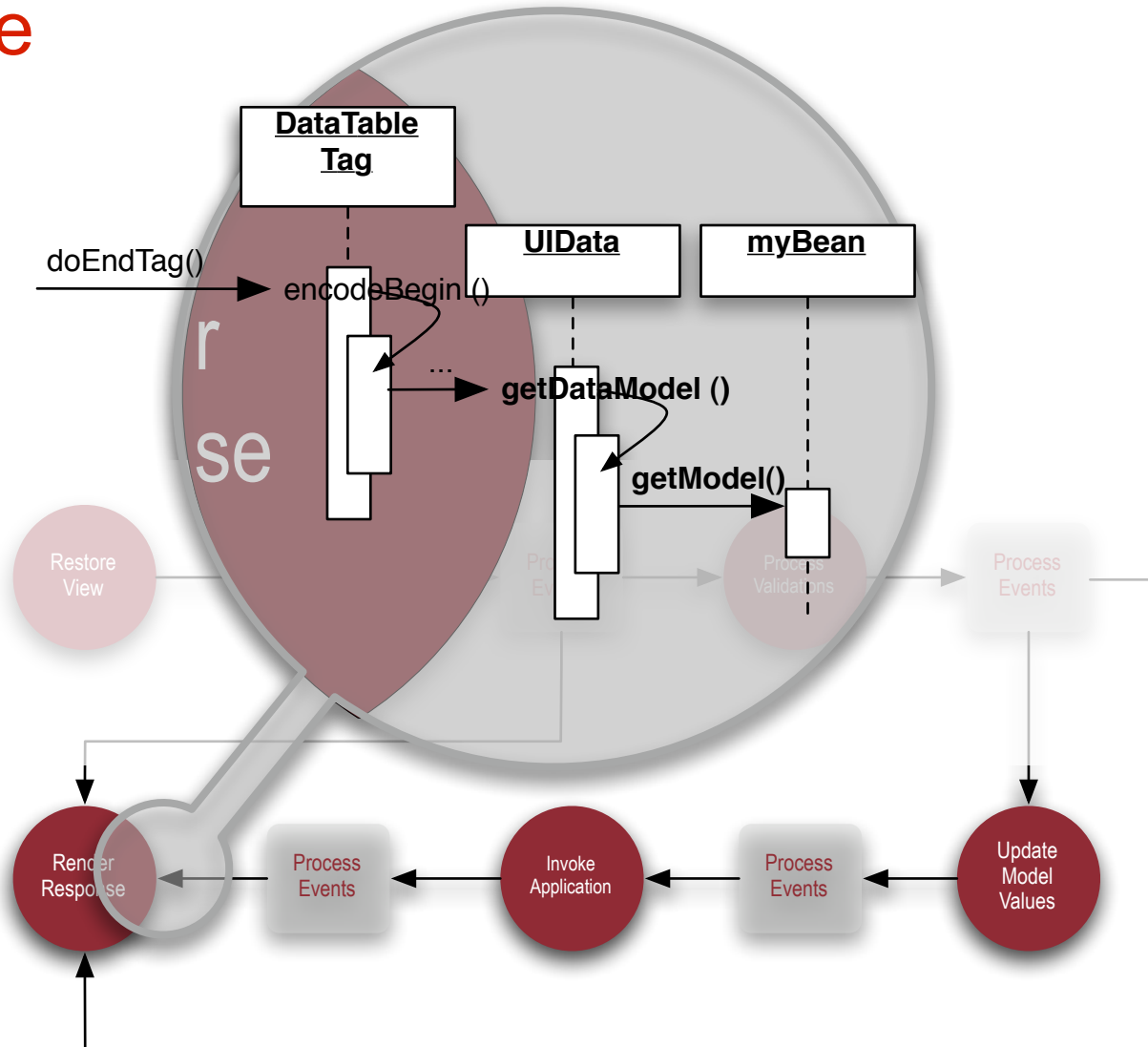
Cause

- DataModel required in Apply Request Values Phase:
 - Used to iterate the model
 - May require backing bean to be initialized
 - Failure results in:
 - Events not getting queued
 - Wrong values returned by `getRowData()`
- ◎ *Reminder* - Backing Bean only gets values in the *Update Model phase*



Handling Tabular Data

Cause

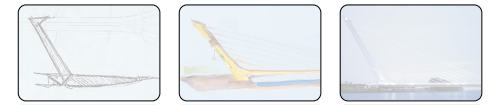




Handling Tabular Data

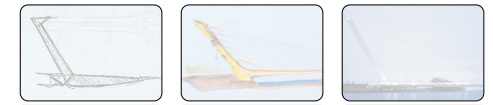
Solution

- Make sure the backing bean returns the *right* model
 - Properties needed to retrieve the model should be present
 - Session scope often the only option
 - `saveState` can do the trick
- Don't depend on `getRowData()`
 - Retrieve Identifier using other means (cfr. Tips and Tricks Section)



Agenda

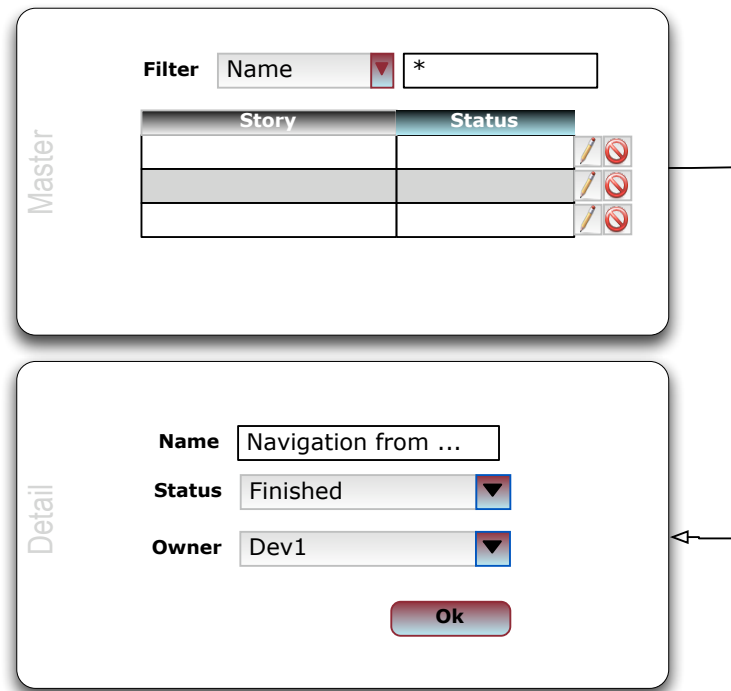
- Common pitfalls when starting with JSF
 - Lifecycle Subtleness
 - Mismatch JSP and JSF rendering
 - Problems Handling Tabular Data
- **Tips and Tricks**
 - **Handling Page Flow**
 - How to Save State
 - Avoid scattered validation logic
 - Testing JSF Applications

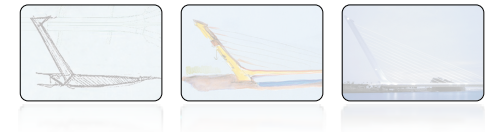


Page Flow

Problem

- Navigate from Master to Detail





Page Flow

Using f:param

- Pass identifier using `f:param`

Master

Filter Name

Story	Status	

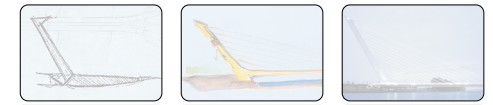
Detail

Name

Status

Owner

```
...
<h:dataTable value="#{storyOverview.stories}"
             var="story" style="overviewTable">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Title" />
    </f:facet>
    ...
  </h:column>
  <h:column>
    <h:commandLink action="#{storyDetail.select}">
      <f:param name="id" value="#{story.id}" />
      <h:outputText value="View" />
    </h:commandLink>
  </h:column>
</h:dataTable>
...
```



Page Flow

Using f:param

- Initialize DetailBean's properties

Master

Filter Name

Story	Status	

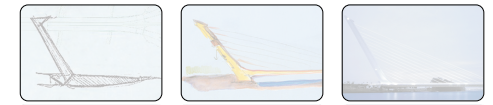
Detail

Name

Status

Owner

```
<managed-bean>
  <managed-bean-name>storyDetail</managed-bean-name>
  <managed-bean-class>
    be.aca.bitterjsf.xp.jsf.StoryDetail
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>id</property-name>
    <property-class>java.lang.String</property-class>
    <value>#{param.id}</value>
  </managed-property>
</managed-bean>
```



Page Flow

Using f:param

- Implement DetailBean action method

```
public class StoryDetail {
```

```
    private Story story;
```

```
    private String id; ← Initialized immediately  
                        after creation
```

```
    ...
```

```
    public String select() {
```

```
        story = storyManager.
```

```
            lookupStory(id);
```

```
        return "detail";
```

```
    }
```

```
    ...
```

```
}
```

Master

Filter Name

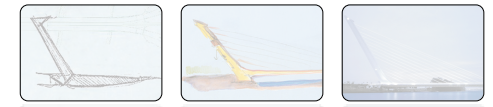
Story	Status	
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>

Detail

Name

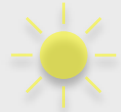
Status

Owner



Page Flow

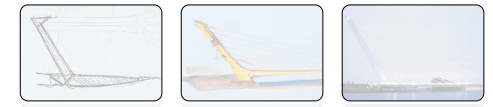
Using f:param



- Navigation and initialization handled by JSF
- Loose coupling between backing beans



- Requires DetailBean to be in request scope
 - Due to initialization of the managed property
- Only usable when `f:param` is supported
 - Not the case for `h:commandButton`

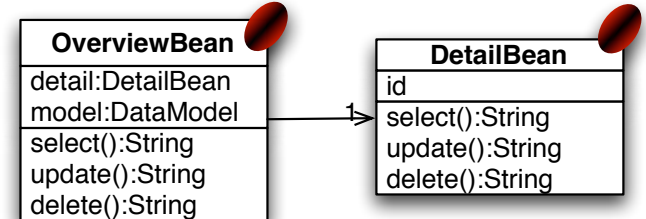


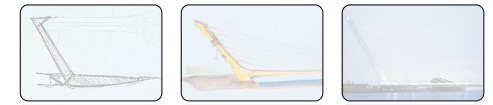
Page Flow

Using Backing Bean Injection

- Inject DetailBean in OverviewBean

```
<managed-bean>
  <managed-bean-name>storyOverview</managed-bean-name>
  <managed-bean-class>
    be.aca.bitterjsf.xp.jsf.StoryOverviewBean
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>detail</property-name>
    <property-class>
      be.aca.bitterjsf.xp.jsf.StoryDetailBean
    </property-class>
    <value>#{storyDetail}</value>
  </managed-property>
</managed-bean>
```



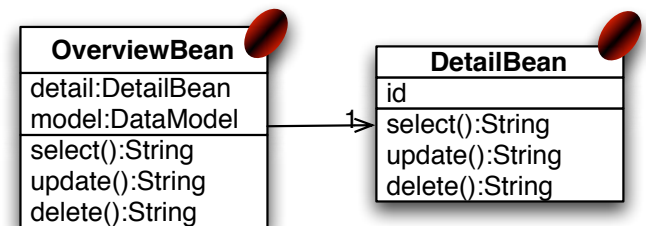


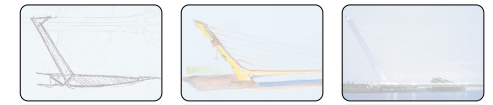
Page Flow

Using Backing Bean Injection

- Initialize `DetailBean` with appropriate values

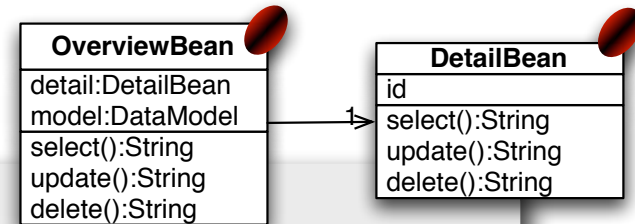
```
public String select(){
    (Story) selected = (Story) model.getRowData();
    getDetailBean().setId(selected.getId());
    return getDetailBean().select();
}
```





Page Flow

Using Backing Bean Injection



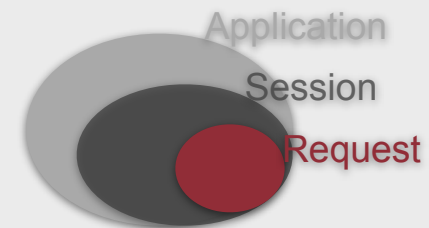
- Complex content can be passed to detail

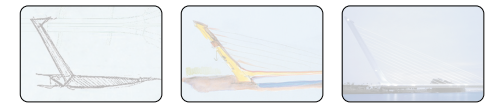
- Adds scope dependency

– $\text{Scope}^{\text{detail}} \geq \text{Scope}^{\text{overview}}$



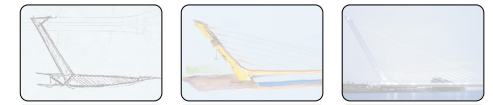
- Reintroduces '*Row Data problem*'





Agenda

- Common pitfalls when starting with JSF
 - Lifecycle Subtleness
 - Mismatch JSP and JSF rendering
 - Problems Handling Tabular Data
- **Tips and Tricks**
 - Handling Page Flow
 - **How to Save State**
 - Avoid scattered validation logic
 - Testing JSF Applications



Save State

Problem

- Conversational state must be saved

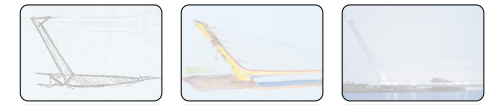
Session



- Easy to get started with



- Back button problem
- Bookmarks don't work
- Handling Session timeout
- Scalability



Save State

Problem

- Conversational state must be saved

Request

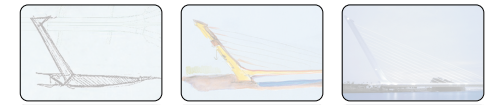


- Scales well
- Eliminates Back Button problem
- No complex session timeout handling



- Backing Bean often needs to be initialized properly early in JSF Lifecycle

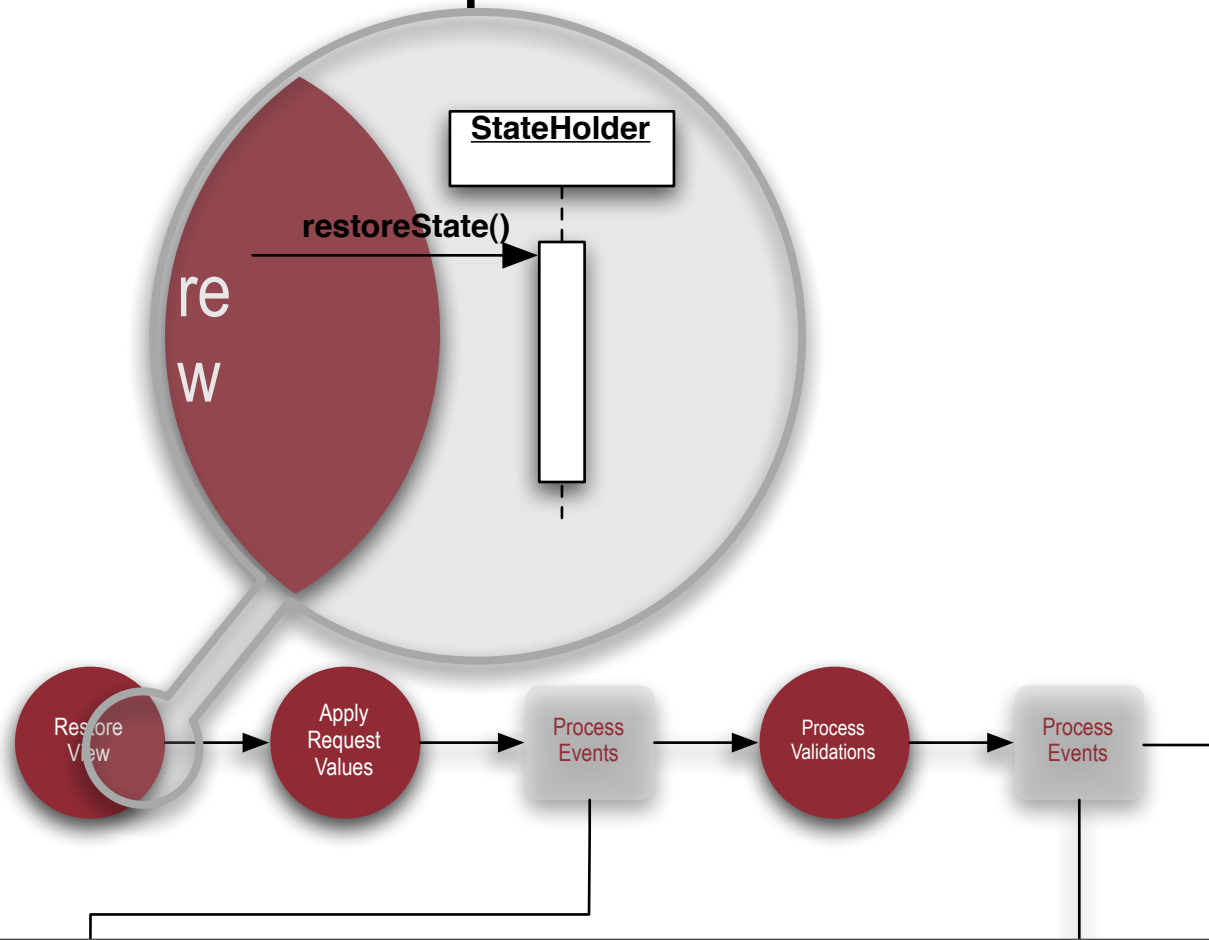


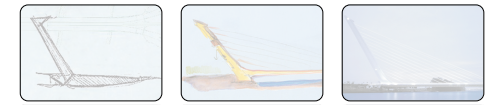


Save State

Solution

- Save State Component

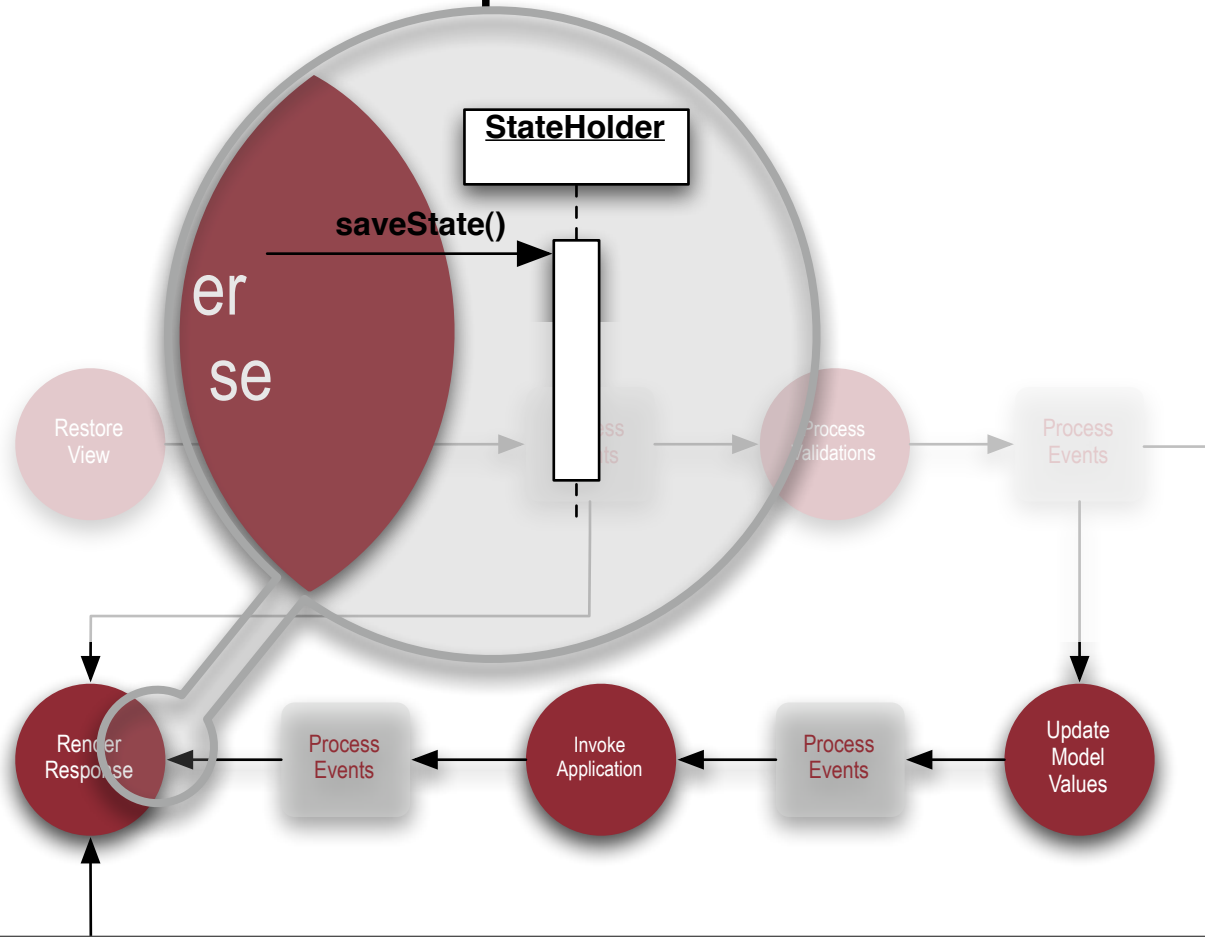




Save State

Solution

- Save State Component





Save State

Solution

- Save State Component
 - `saveState()` is used to save any component value
 - `restoreState()` restores previously saved component value



Save State

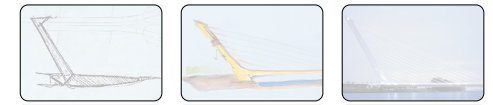
Solution

- Component Implementation*

```
@Override
public Object saveState(FacesContext context) {
    Object values[] = new Object[2];
    values[0] = super.saveState(context);
    values[1] = getValue();
    return values;
}
```

```
@Override
public void restoreState(FacesContext context,
                        Object state) {
    Object values[] = (Object[]) state;
    super.restoreState(context, values[0]);
    Object savedValue = values[1];
    ValueBinding vb = getValueBinding("value");
    if (vb != null) {
        vb.setValue(context, savedValue);
    }
}
```

* see <http://myfaces.apache.org/>



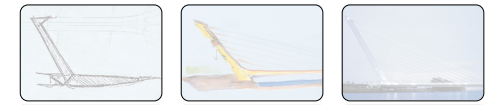
Save State

Solution

- Usage

```
<a:saveState id="filterState"  
            value="#{storyOverview.filter}"/>
```

- Stores the current value of the `filter` property
- Injects the saved value in the `storyOverview` bean on the next request



Save State

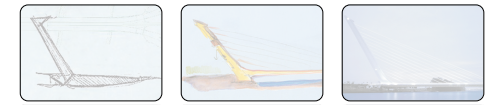
Remarks



- Powerful and flexible mechanism
- Eliminated initialization problem:
 - Values are restored in the beginning of the lifecycle
- Can save simple values and whole beans
 - as long as they are `Serializable`



- Use it with care:
 - Saved state can become hard to trace
- Does not work with redirects



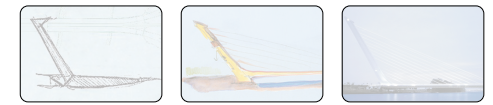
Save State

Remarks

According to the JSF 1.1 Specs:

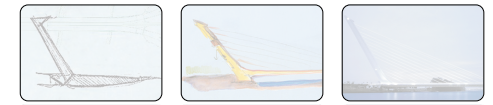
“If the state saving method is server, these methods may not be called.”

- `javax.faces.STATE_SAVING_METHOD` must be set to `client` for the JSF RI
- In MyFaces both `client` and `server` can be used



Agenda

- Common pitfalls when starting with JSF
 - Lifecycle Subtleness
 - Mismatch JSP and JSF rendering
 - Problems Handling Tabular Data
- **Tips and Tricks**
 - Handling Page Flow
 - How to Save State
 - **Avoid scattered validation logic**
 - Testing JSF Applications



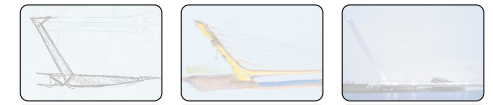
Custom Data Types

Problem

- Duplicate validation constraints
 - Multiple pages contain the same constraint

```
<h:inputText id="accountNumber" required="true"
              value="#{accountDetail.accountNumber}">
  <t:validateRegExpr
                    pattern='\d{3}-\d{7}-\d{2}' />
</h:inputText>
```

- Back-end business logic implements this constraint as well



Custom Data Types

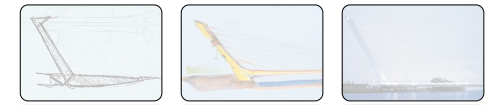
Solution

- Use a `Converter` for custom data types
 - Components can be bound to the custom data type

```
<h:inputText id="accountNumber" required="true"
              value="#{accountDetail.accountNumber}"/>
```

```
public AccountDetailBean {
    private AccountNumber accountNumber;
    ...
}
```

- Data type encapsulates all constraints
- `Converter` handles the conversion details

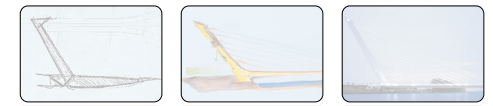


Custom Data Types

Solution

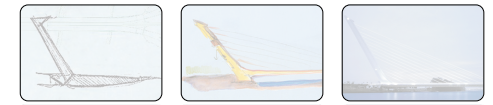
- Use a Converter for custom data types
 - Converter handles the conversion details

```
public final class AccountNumberConverter
    implements Converter {
    ...
    public Object getAsObject(FacesContext context,
        UIComponent component,
        String value) {
        //Delegate conversion to datatype
    }
    public String getAsString(FacesContext context,
        UIComponent component,
        Object value) {
        ...
    }
}
```



Agenda

- Common pitfalls when starting with JSF
 - Lifecycle Subtleness
 - Mismatch JSP and JSF rendering
 - Problems Handling Tabular Data
- **Tips and Tricks**
 - Handling Page Flow
 - How to Save State
 - Avoid scattered validation logic
 - **Testing JSF Applications**



Testing JSF Applications

Problem

- Presentation Layer typically very error prone





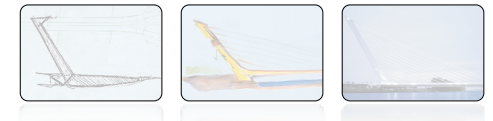
Testing JSF Applications

Solution

①

Use *Mock Objects* to emulate the JSF container API

- Allows JSF related classes to be tested as an isolated unit
- See:
 - <http://struts.apache.org/struts-shale>
 - <https://javaserverfaces.dev.java.net>



Testing JSF Applications

Solution

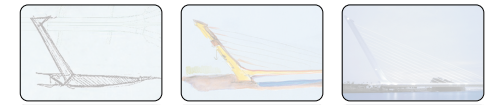
①

Use *Mock Objects* to emulate the JSF container API

```
public class StoryPointConverterTest extends
    AbstractViewControllerTestCase {
    ...
    public void testGetAsObjectIncorrectStoryPoint() {
        try {
            getConverter().getAsObject(this.facesContext,
                getComponent(), "2.2");
            fail("ConversionException not thrown");
        }
        catch (ConverterException e) {
            checkMessageCount(
                getComponent().getClientId(facesContext),
                1);
        }
    }
}
```

[exit to code](#)

eclipse



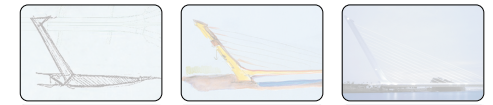
Testing JSF Applications

Solution

②

Use *Stubs* for backing beans and/or services

- Stubs can be activated by custom `VariableResolver`
 - Does not require any change to the actual pages



Testing JSF Applications

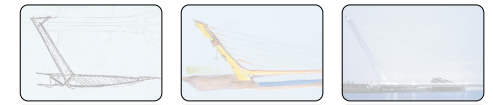
Solution

②

Use *Stubs* for backing beans and/or services

```
public class DecoratedVariableResolver extends
                                   VariableResolver {

    @Override
    public Object resolveVariable(FacesContext ctx,
                                   String name)
                                   throws EvaluationException {
        ...
        return this.resolver.resolveVariable(ctx,
                                   stubMap.lookupStub(name));
        ...
    }
}
```

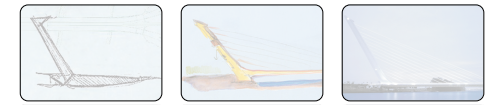


Testing JSF Applications

Solution

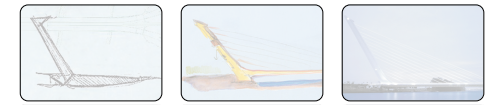
③ Web Test Frameworks

- Ideally for end-to-end integration testing
 - Selenium (<http://www.openqa.org/selenium/>)
 - JWebUnit (<http://jwebunit.sourceforge.net/>)
 - ...



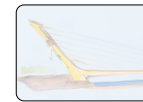
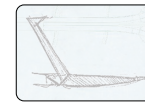
Sweet JavaServer Faces

- Keep your experiences sweet
 - Value the lifecycle
 - Consider Test First
 - Choose the right component libraries
 - Customize the framework to fit your needs
- And ... it only will get better with tools emerging that leverage on JSF



If You Only Remember One Thing...

JavaServer™ Faces offers an extremely powerful and flexible framework. Use its power to fit your application's needs. And don't forget that it's all about the lifecycle.



Q&A

www.aca-it.be

