LESSON 9

SPRING WEB FLOW

Continuity

Spring Web Flow

- Spring Web Flow is a workflow engine designed exclusively for Web page navigation.

work·flow
sequence of steps through which a piece of work passes from initiation to completion.

Basic characteristics of a web flow:

There is a clear start and an end point.

The user must go through a set of screens in a specific order.

The changes are not finalized until the last step.

Once complete it shouldn't be possible to repeat a transaction accidentally

Example Use Cases

Airline Flight Checkin
Loan Application
Shopping Cart Checkout

Spring Web Flow
States, Transition and Data

A Web Flow consists of a set of states

and the transitions between the states

States

When an action happens OR Decision made OR View is displayed

The result is a state [Action State; Decision State; View State]

Transitions

Connection between two states.

Data

Information that is managed between states

Spring Web Flow States

View State -

Any view defined in Spring MVC. (e.g. JSP)

Action State –

Invoke an action, then transition to another state

Decision State -

Action State alternative - True/False determination resulting in two alternate transitions

Subflow state

Invoke another flow as a subflow

End State

Examples of States

Invoke action[addCartToOrder] to transition to another state

```
<action-state id="addCartToOrder">
   <evaluate expression="cartServiceImpl.validate(cartId)“ result="order.cart" />
   <transition to="collectCustomerInfo" />
</action-state>
```

view is based on id (collectCustomerInfo.jsp)

```
<view-state id="collectCustomerInfo" model="order">
<transition on="customerInfoCollected" to="collectShippingDetail" />
<transition on="checkoutCancelled" to="cancelCheckout" validate= "false"/>
</view-state>
```

Action State alternative - True/False determination resulting in two alternate transitions

```
 <decision-state id="createOrEdit">
    <if test="homeAddress == null" then="cancelCheckout" else="getBillingAddress" />
 </decision-state>

<subflow-state id="creditCard" subflow="creditCard">
...
</subflow-state>

<end-state id="endState"/>
```

More Action Execution Scenarios [Besides action-state]

Start Of Flow [<on-start>] ---- **End of Flow** [<on-end>]

On Entering a State [<on-entry>] ---- **On Exiting a State** [<on-exit>]

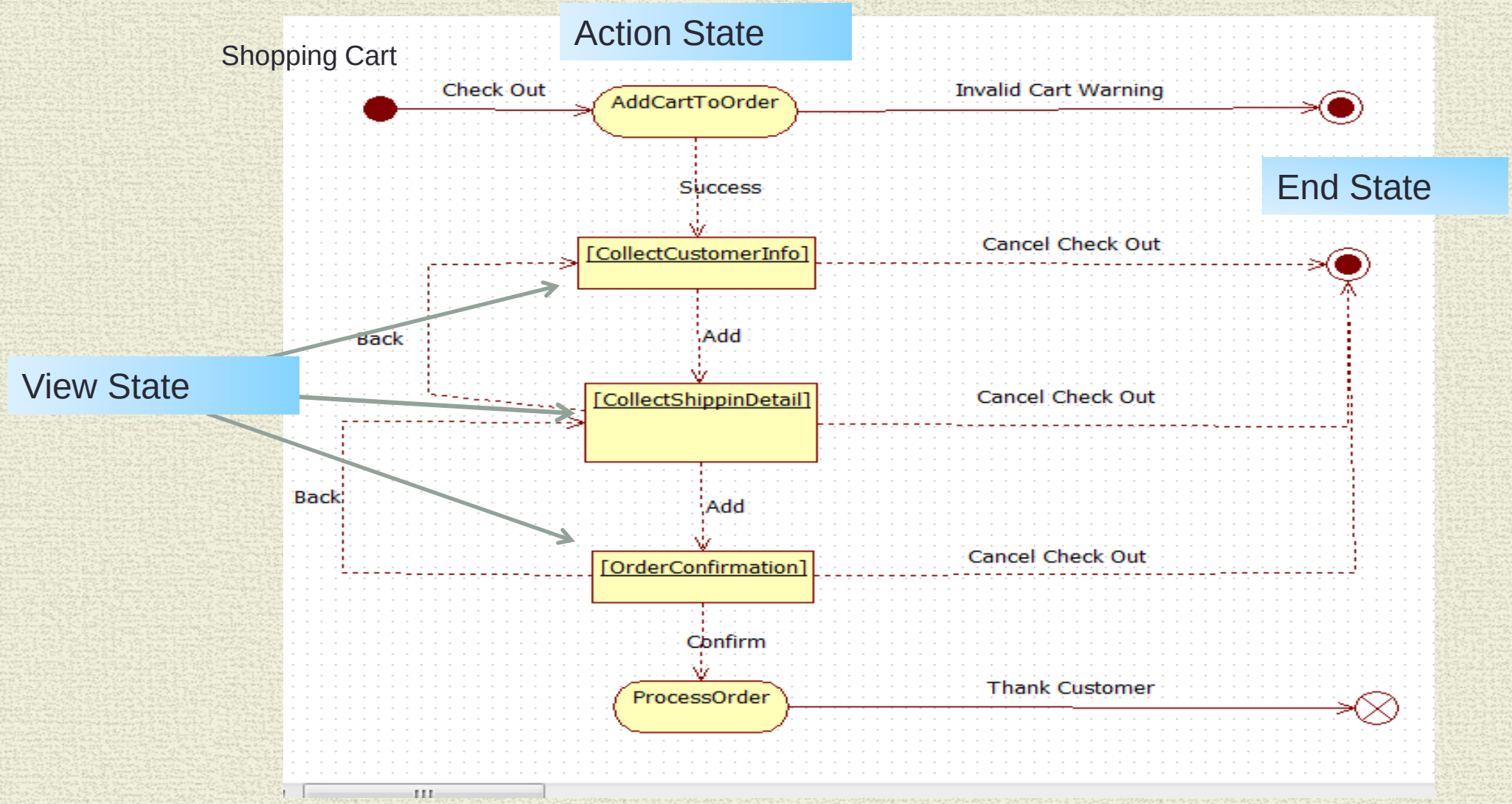
On the rendering of a view [<on-render> in view-state]

On transition [<transition>]

EXAMPLE:

```
<transition on="cardComplete" to="creditCardCompleted">
   <evaluate expression= creditCard.setBillingAddress(billingAddress())"/>
 </transition>
```

Order Check Out Example



Spring Web Flow
User Conversational Flows

A Spring Web Flow is a blueprint for user conversations that drive a business process.

Fundamental capability is ***Managing Data*** in the back & forth process involving multiple input screens.

Automates State Management

Common MVC Technologies:

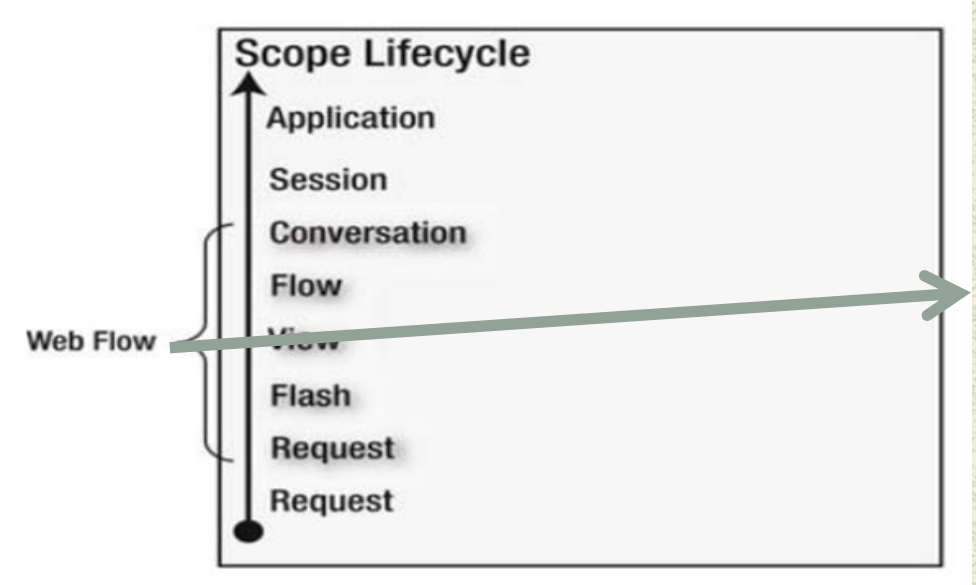
@SessionAttributes facilitates conversational flows

Flash Attributes are also a building block

PLUS

Additional Scopes [beyond Request,Session & Application]

SWF Additional Scopes for Managing Data



Web Flow Specific
Stored in HTTP Session
Managed by SWF
[Creation & Deletion]

- **Conversation** - The conversation scope starts when a flow starts and ends when the flow ends.
  It is available in sub flows
- **Flow -** Available within a flow. **Not available in sub flows**
- **Request -** Available during the life of a request in a flow [related to Flow NOT HTTP]
- **Flash -** Available during the lifetime of a flow. However, once a state is rendered, the variable is cleared.
- **View -** Available only during the lifetime of a view. Created when a view is created and destroyed once a view is destroyed [e.g. Ajax requests]

Main Point

- The Spring WebFlow Controller is characterized by clearly defined separate and distinct states.

- *Consciousness is also defined by unique and separate states*

Spring MVC with Spring Web Flow

Spring Web Flow is built

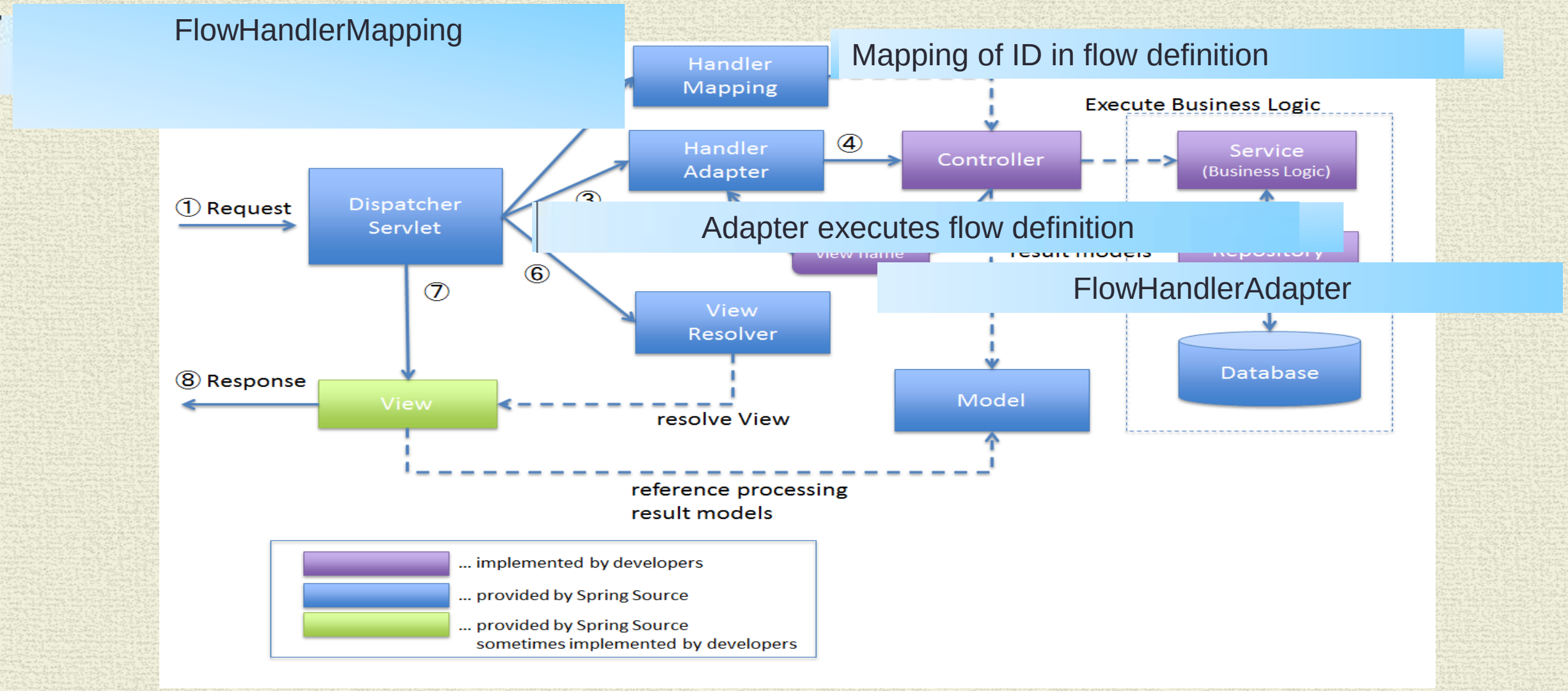
on top of Spring MVC

Spring Web Flow acts as the

“C”[Controller] in Spring MVC

Spring MVW [Whatever]

Spring MVC Flow



Spring Web Flow Configuration

Configure Handler Mapping

```
<bean id="flowHandlerMapping"
        class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
        <property name="flowRegistry" ref="flowRegistry" />
</bean>
```

Configure Handler Adapter

```
<bean id="flowHandlerAdapter"
         class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
      <property name="flowExecutor" ref="flowExecutor" />
</bean>
```

flowExecutor Drives the execution of flows across a variety of environments e.g., Spring MVC, Struts, and Java Server Faces (JSF)

Configure Executor

```
<webflow-config:flow-executor id="flowExecutor"flow-registry="flowRegistry" />
```

Configure Flow definition

```
<webflow-config:flow-registry id="flowRegistry"base-path="/WEB-INF/flows">
     <webflow-config:flow-location id="checkout" path="/checkout/checkout-flow.xml" />
</webflow-config:flow-registry>
```

Initiating the flow

Cart

All the selected products in your cart

Clear Cart

Check out

Product	Unit price	Quantity	Price	Action
P1234-iPhone 5s	500	1	500	Remove
		Grand Total	500	

```
<a href="<spring:url value="/checkout?cartId=${cartId}"/>">Check out</a>
```

“Base” URL “checkout” comes from location of the flow registry:

- `<webflow-config:flow-registry id="flowRegistry"base-path="/WEB-INF/flows">`

```
    <webflow-config:flow-location id="checkout" path="/checkout/checkout-flow.xml" />
```

Customer

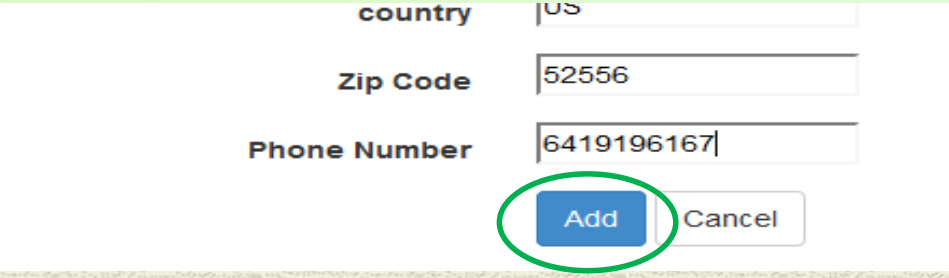
On click of JSP ADD button:

```
<input type="submit" value="Add" name="_eventId_customerInfoCollected" />
```

"_eventId_" signals Spring to activate event with name:

customerInfoCollected

```
<button id="btnCancel" name="_eventId_checkoutCancelled">Cancel</button>

<view-state id="collectCustomerInfo" view="collectCustomerInfo.jsp" model=order>
  <transition on="customerInfoCollected" to="collectShippingDetail" />
  <transition on="checkoutCancelled" to="cancelCheckout" validate= "false"/>
</view-state>
```



There is a Hidden Field on a page contained in a Flow that ID's to Spring the view-state the page is associated with in order to continue the flow:

```
<input type="hidden" name="_flowExecutionKey" value="${flowExecutionKey}"/>
```

Shipping

Shipping details

Shipping Details

Name	Fred Tool
shipping Date (dd/mm /yyyy)	22/12/1967
Street Name	West St.
State	IA
country	US
Zip Code	51234

back Add Cancel

```
<view-state id="collectShippingDetail" model="order">
    <transition on="shippingDetailCollected" to="orderConfirmation" />
     <transition on="backToCollectCustomerInfo" to="collectCustomerInfo" />
</view-state>
```

Order

Order Confirmation

```
<button type="submit" class="btn btn-success“ name="_eventId_orderConfirmed"> Confirm

<transition on="orderConfirmed" to="processOrder" />
```

Fred Tool
West St.
IA ,51234
US

Billing Address
Joe
Coral Lane
IA,52556
US
P: 6419196167

Product	#	Price	Total
iPhone 5s	1	$500	$500
		Grand Total:	**$500**

back | Confirm | Cancel

```
<action-state id="processOrder">
<evaluate expression="orderServiceImpl.saveOrder(order)" result="order.orderId"/>
  <transition to="thankCustomer" />
</action-state>
```

Thank you

Thanks for the order. your order will be delivered to you on Dec 22, 1967!

Your Order Number is 1000

products

Replace Service implementation with DI

```
<evaluate expression="cartServiceImpl.validate(cartId)“ result="order.cart" />
```

- **Create Wrapper Class for WebFlow Controller Service calls**

```
@Component
public class CheckoutControllerHelper {

@Autowired
OrderService orderService;
@Autowired
CartService cartService;

public void saveOrder(Order order) {
        orderService.saveOrder(order);
}
public Cart validateCart(String cartId) {
        return cartService.validate(cartId);
}
```

```
<evaluate expression="checkoutControllerHelper.validateCart(cartId)“ />
```

Add Validation

Configure Validator

```
• <webflow-config:flow-builder-services id="flowBuilderServices"
                                        validator="validator" />
<bean id="validator“ class="org.springframework....LocalValidatorFactoryBean" />
```

- Annotate Domain Objects with JSR 303

```
• public class Order  implements Serializable{
• @Valid
• private Customer customer;
•
```

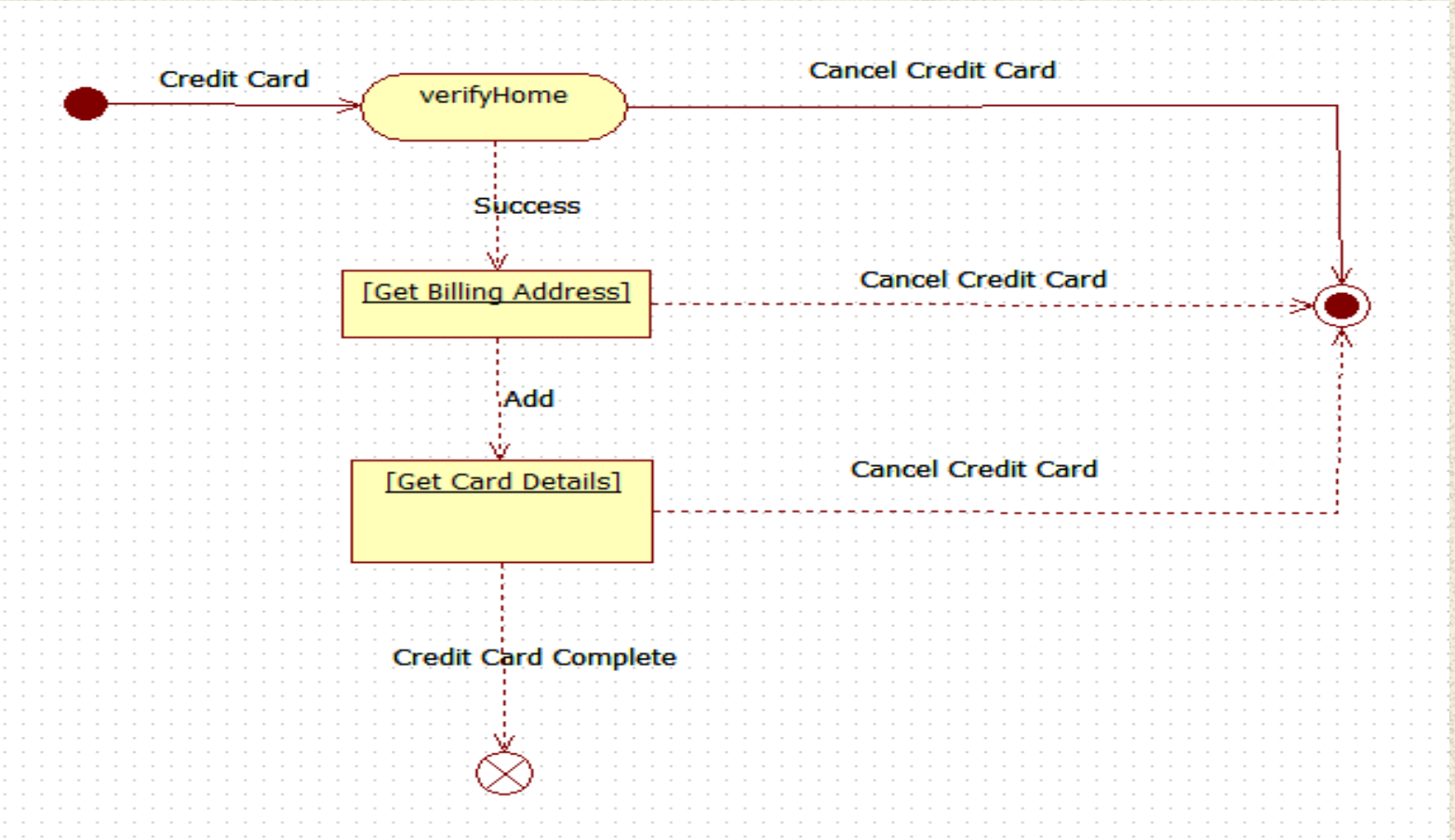
Modify collectCustomerInfo.jsp:

```
<form:errors path="*" cssStyle="color : red;" />
```

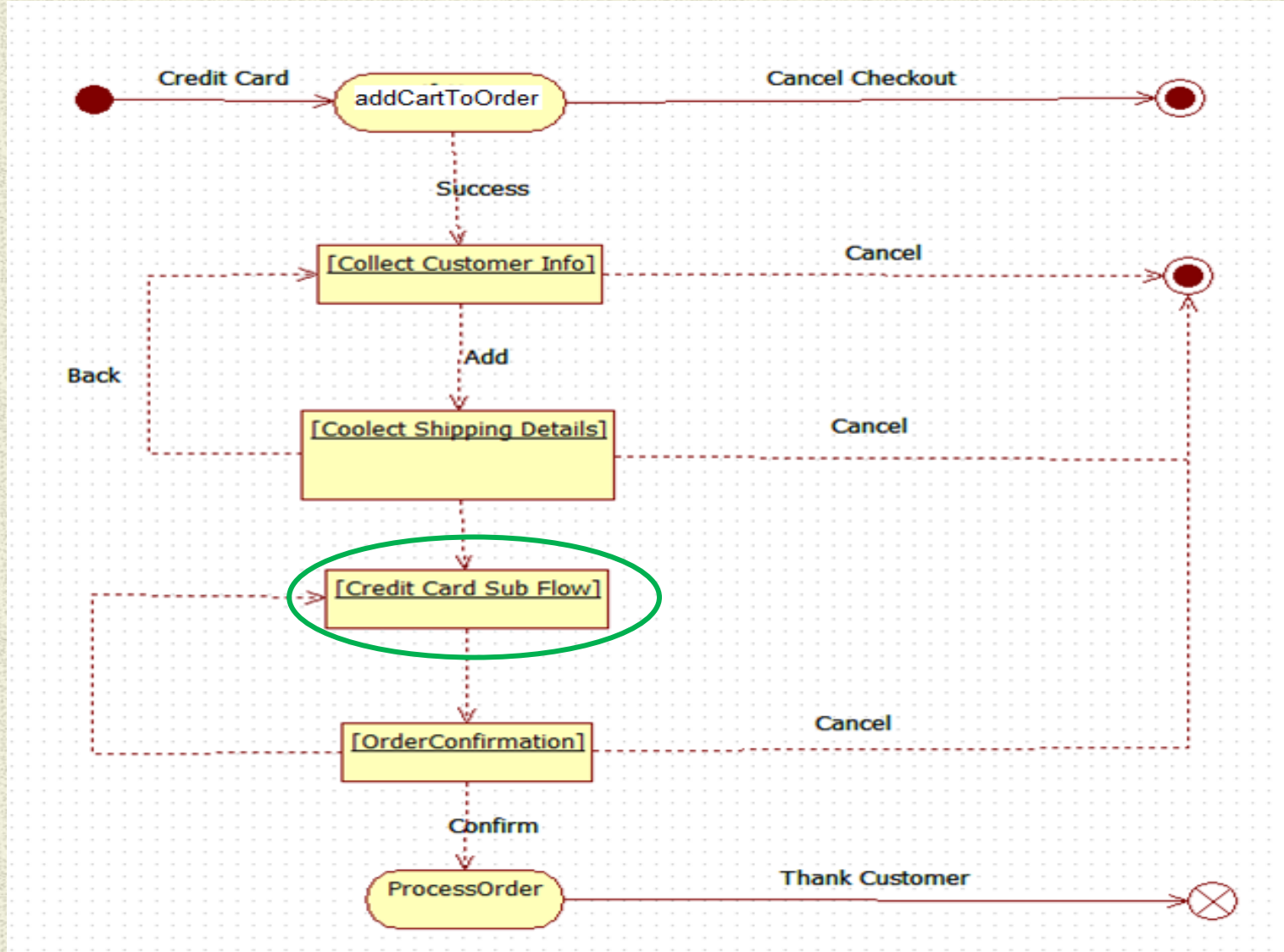
```
• public class Customer implements Serializable{
• @NotEmpty(message = "Name required")
• private String name;
• @Valid
• private Address billingAddress;
•
• public class Address implements Serializable{
• @NotEmpty(message="Street Name Required")
•  private String streetName ;
```

Adding a subflow

- Making use of a re-usable credit card flow…



“new” Checkout flow



Sub Flow Declaration in Checkout-flow

```
<subflow-state id="creditCard" subflow="creditCard">
<!-- INPUT to subFlow -->
  <input value="order.customer.address" name="homeAddress"/>
  <input value="order.shippingDetail.shippingAddress" name="shippingAddress"/>
 <!-- OUTPUT From subFlow -->
  <output name="creditCard"   type="com.packt.webstore.domain.CreditCard"/>
  <!- When creditCard sub flow is done-->
  <transition on="creditCardCompleted" to="orderConfirmation">
          <set name="flowScope.order.creditCard" value="creditCard"/>
  </transition>

<transition on="cancelCreditCard" to="cancelCheckout"  />
</subflow-state>
```

Set credit card in order

Credit Card Sub Flow

- **Input Parameters:**

```
• <input name="homeAddress" type="com.packt.webstore.domain.Address" />
• <input name ="shippingAddress" type="com.packt.webstore.domain.Address"/>
•
• <!-- First "state" listed is first executed - unless <on-start> -->
<decision-state id="verifyHome">
 <if test="homeAddress == null" then="cancelCheckout" else="getBillingAddress" />
</decision-state>
```

...Upon completion...

Output Parameter:

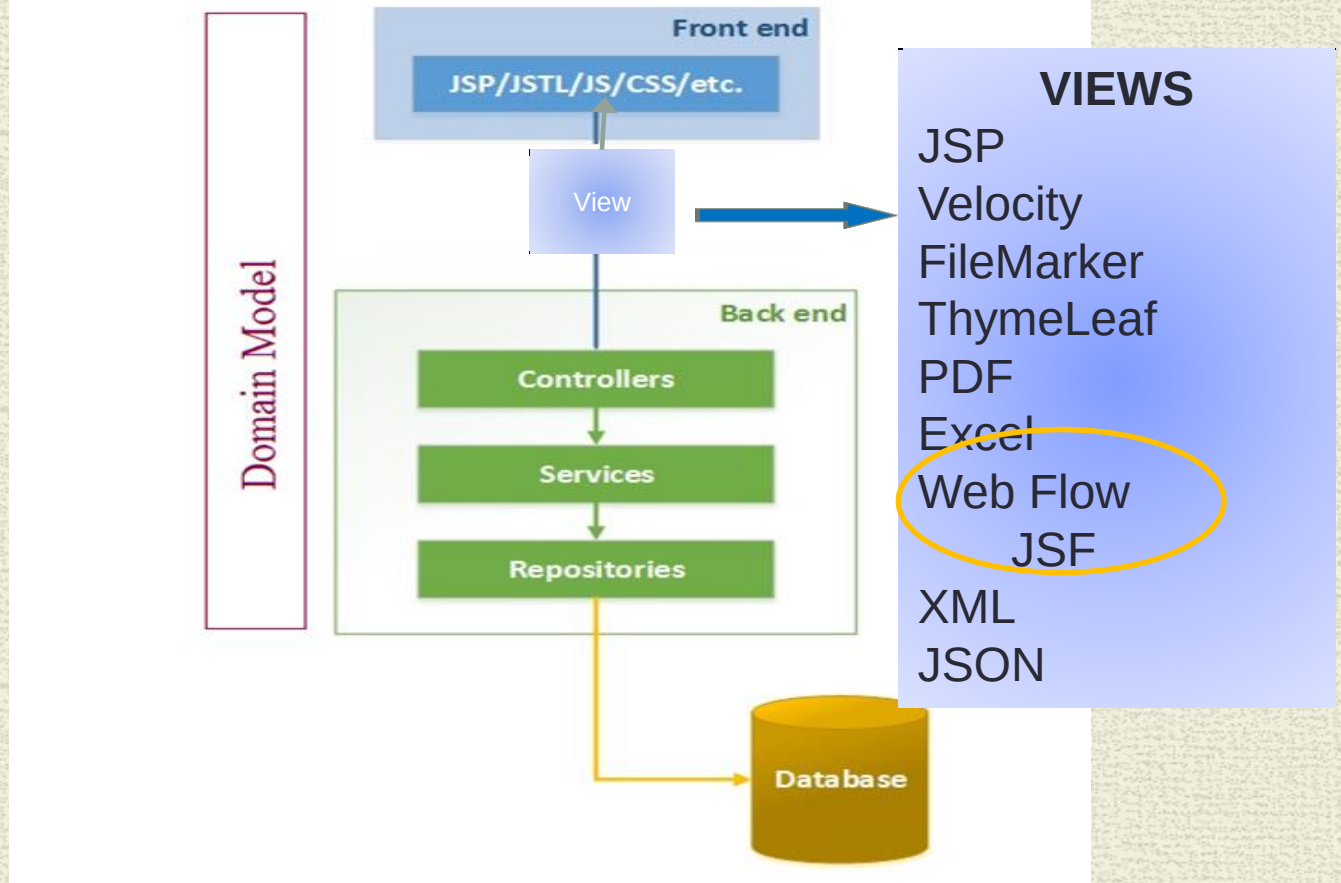
```
•  <end-state id="creditCardCompleted">
•          <output name="creditCard" value="creditCard"/>
•       </end-state>
```

Main Point

A Sub Flow is specialized, re-usable Spring WebFlow functionality that supports composition & modularity.

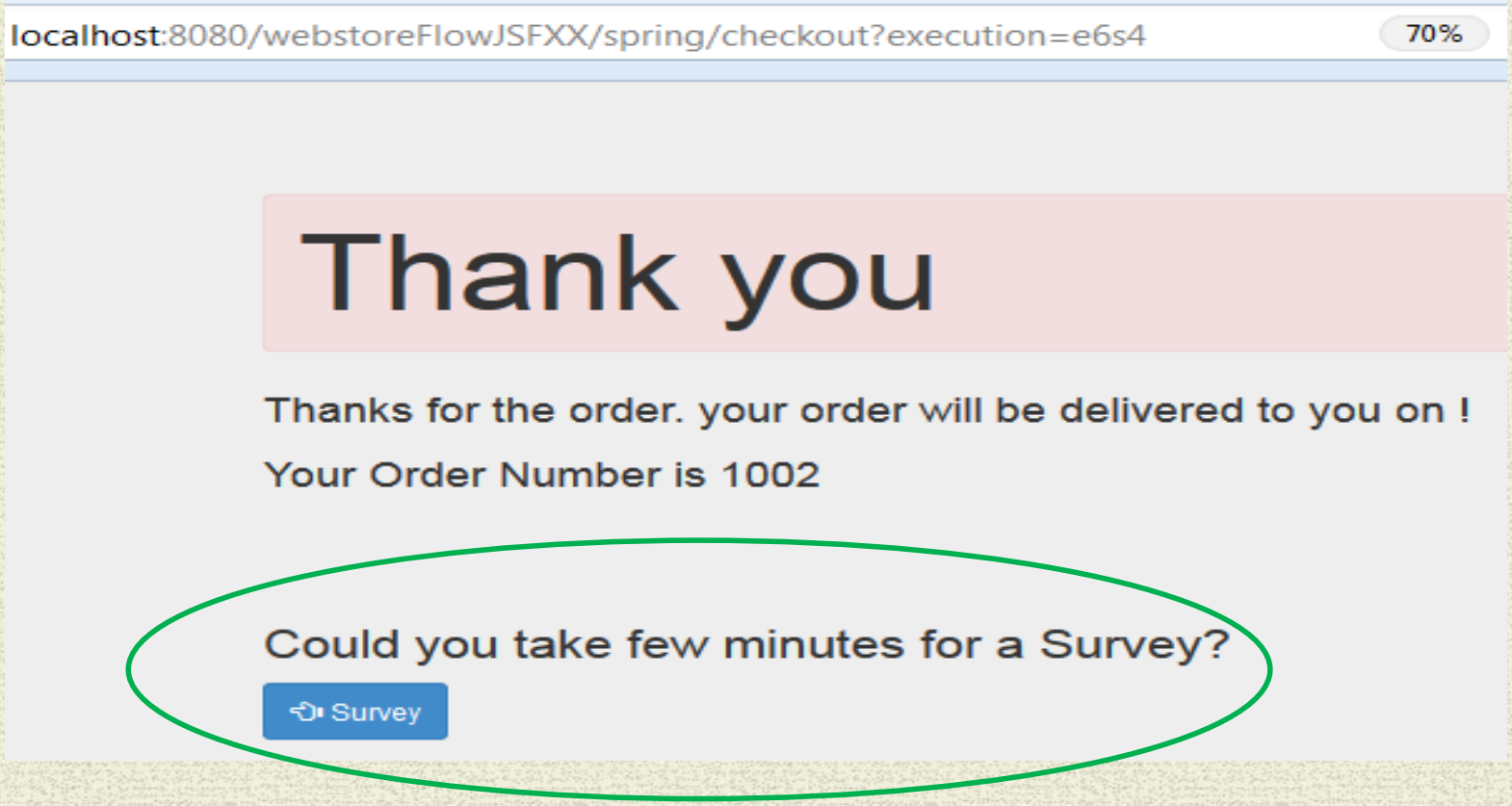
Nature is efficient. It re-uses building blocks within its own structure

Spring – JSF Integration



Survey Web Flow integrating Spring & JSF Technologies

- JSF UI Component Model can be used with Spring MVC / Spring Web Flow Controllers.
- Spring Web Flow can be used in JSF environments
- Spring Security can be used in JSF environments



Java Server Faces [JSF]

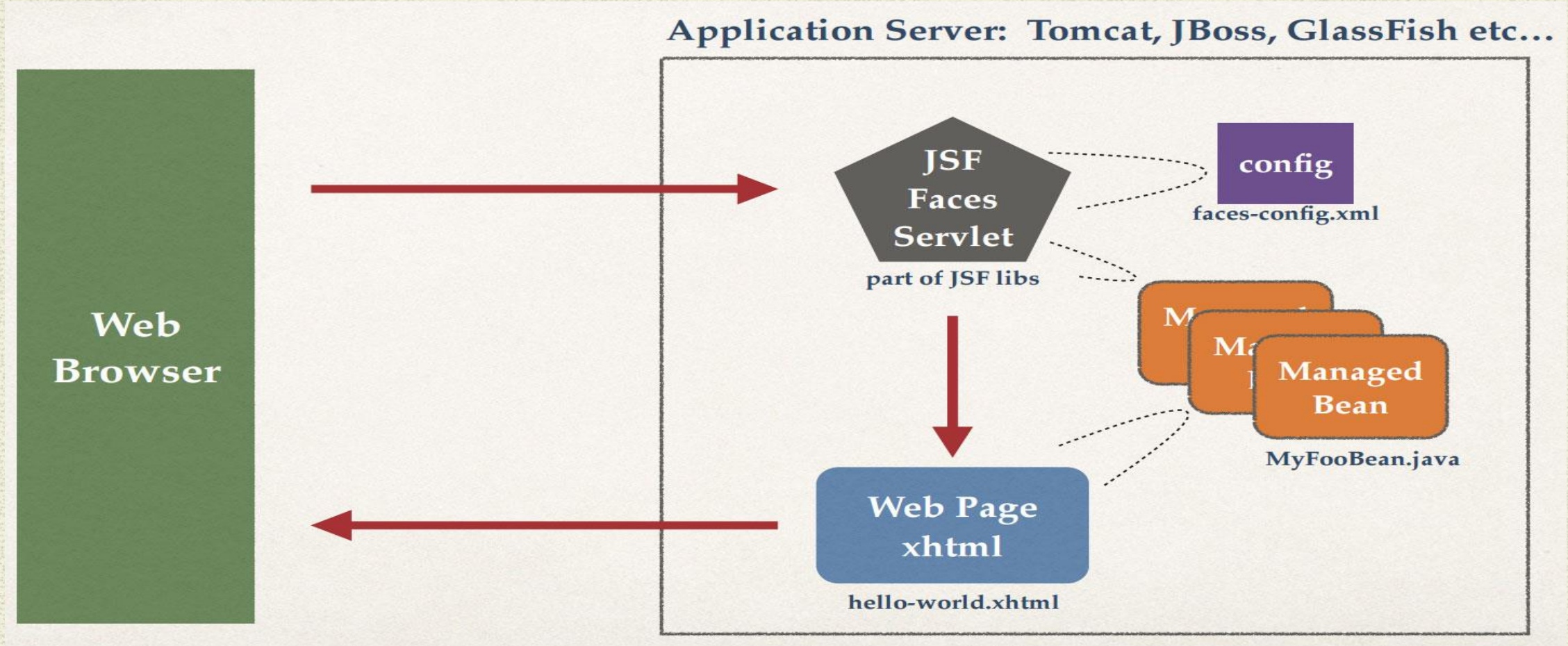
- Develop Rich Internet Applications [RIA]

  Achieve the behavior of desktop application on web browser.
- Response to the success of Microsoft's ASP.NET WebForms

- According to Sun Microsystems:
- ***"Java Server Faces technology simplifies building user interfaces for Java Server applications. Developers of various skill levels can quickly build web applications by: assembling reusable UI components in a page…"***

  It is a UI-driven component framework
  - GOAL: factor away much of the complexity of web UI development, in particular appealing to 4GL developers

JSF MVC Framework



100+ UI tags for basic web & reusable UI component development

JSF Component [Tag] Libraries

JSF Tag Libraries

Library	Namespace Identifier	Commonly Used Prefix	Number of Tags
Core	http://java.sun.com/jsf/core	f:	27
HTML	http://java.sun.com/jsf/html	h:	31
Facelets JSF 2.0	http://java.sun.com/jsf/facelets	ui:	11
Composite Components JSF 2.0	http://java.sun.com/jsf/composite	composite:	12
JSTL Core JSF 2.0	http://java.sun.com/jsp/jstl/core	c:	7
JSTL Functions JSF 2.0	http://java.sun.com/jsp/jstl/functions	fn:	16

Actions: Listener,Event , Ajax..

UI Templating

Specialized Enhanced 3rd party Component libraries PrimeFaces OpenFaces ICE Faces RichFaces

4GL type development Downside

High level abstraction requires a specialized non-transferrable skillset. And the learning curve can be long… and the tool set complex…

January 2014 "Technology Radar" publication, ThoughtWorks :

"We continue to see teams run into trouble using JSF …and are recommending you avoid this technology.

We think JSF is flawed because it tries to abstract away HTML, CSS …, exactly the reverse of what modern web frameworks do…

We are aware of the improvements in JSF 2.0, but think the model is fundamentally broken.

We recommend teams use simple frameworks and embrace and understand web technologies including HTTP, HTML and CSS."

JSF - UI-driven component framework

JSF Expression Language

provides ***method binding to UI [NOT URL]***

```
<h:commandButton action="#{surveyController.submit(survey)}" value="Done"/>
```

Built-in PRG pattern – Flash Attributes

Controller:

```
return "surveyDone?faces-redirect=true";
```

HTML

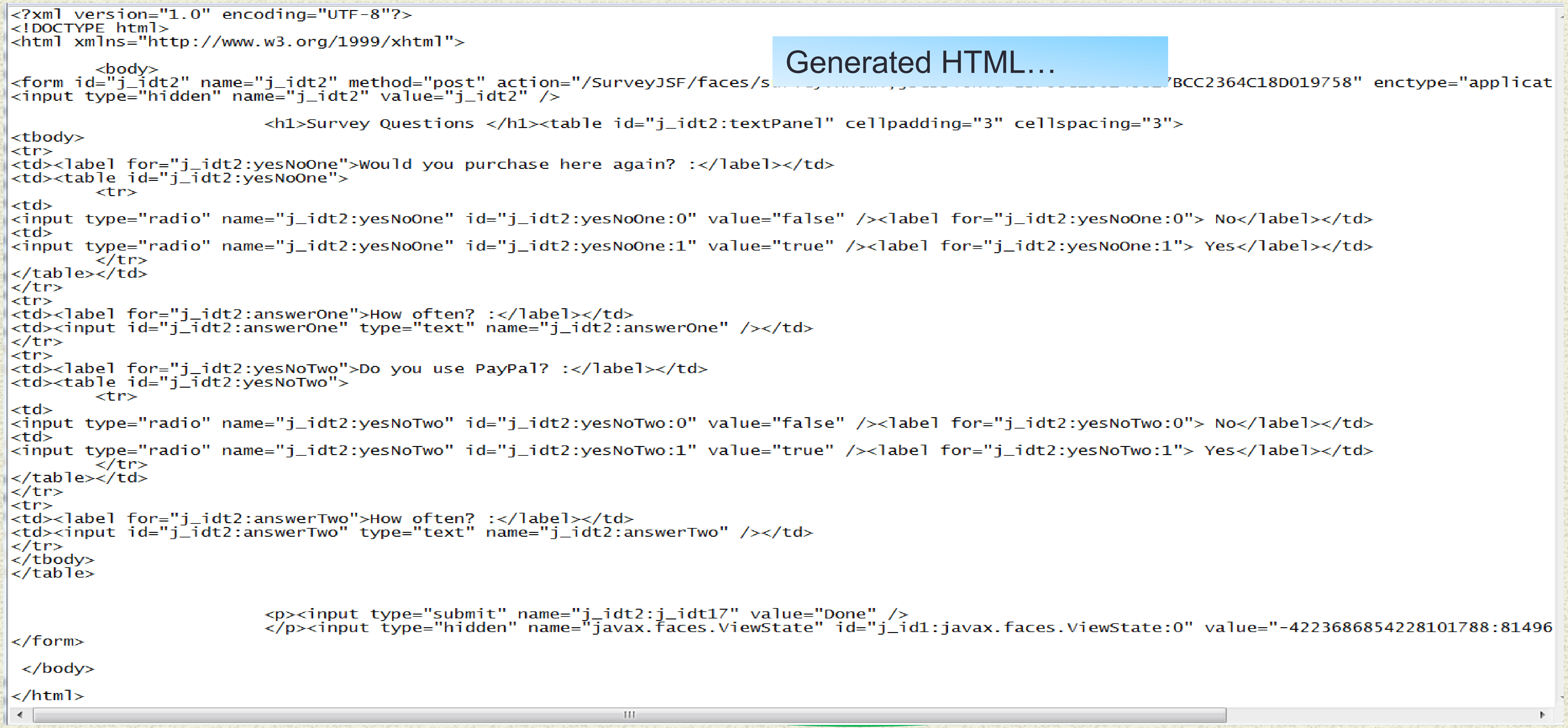
When Button *"Tell Me"* is clicked –
`execute inputText-"inputName"` to store name in *userController*; *render* `outputText-"outputMessage"` which renders the response: `userController.welcomeMessage` for display

```
<p>
<h:inputText id = "inputName" value= "#{userController.name}"/>
<h:commandButton value = "Tell Me">
    <f:ajax execute = "inputName" render = "outputMessage" />
 </h:commandButton>
  <h:outputText id = "outputMessage"
                value = "#{userController.welcomeMessage}"/>
```

JSF Example

Survey Custom Component

Generated HTML…



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">

        <body>
<form id="j_idt2" name="j_idt2" method="post" action="/SurveyJSF/faces/s[illegible]BCC2364C18D019758" enctype="applicat
<input type="hidden" name="j_idt2" value="j_idt2" />

                        <h1>Survey Questions </h1><table id="j_idt2:textPanel" cellpadding="3" cellspacing="3">
<tbody>
<tr>
<td><label for="j_idt2:yesNoOne">Would you purchase here again? :</label></td>
<td><table id="j_idt2:yesNoOne">
        <tr>
<td>
<input type="radio" name="j_idt2:yesNoOne" id="j_idt2:yesNoOne:0" value="false" /><label for="j_idt2:yesNoOne:0"> No</label></td>
<td>
<input type="radio" name="j_idt2:yesNoOne" id="j_idt2:yesNoOne:1" value="true" /><label for="j_idt2:yesNoOne:1"> Yes</label></td>
        </tr>
</table></td>
</tr>
<tr>
<td><label for="j_idt2:answerOne">How often? :</label></td>
<td><input id="j_idt2:answerOne" type="text" name="j_idt2:answerOne" /></td>
</tr>
<tr>
<td><label for="j_idt2:yesNoTwo">Do you use PayPal? :</label></td>
<td><table id="j_idt2:yesNoTwo">
        <tr>
<td>
<input type="radio" name="j_idt2:yesNoTwo" id="j_idt2:yesNoTwo:0" value="false" /><label for="j_idt2:yesNoTwo:0"> No</label></td>
<td>
<input type="radio" name="j_idt2:yesNoTwo" id="j_idt2:yesNoTwo:1" value="true" /><label for="j_idt2:yesNoTwo:1"> Yes</label></td>
        </tr>
</table></td>
</tr>
<tr>
<td><label for="j_idt2:answerTwo">How often? :</label></td>
<td><input id="j_idt2:answerTwo" type="text" name="j_idt2:answerTwo" /></td>
</tr>
</tbody>
</table>


                        <p><input type="submit" name="j_idt2:j_idt17" value="Done" />
                        </p><input type="hidden" name="javax.faces.ViewState" id="j_id1:javax.faces.ViewState:0" value="-4223686854228101788:81496
</form>

 </body>

</html>
```

Ajax Example

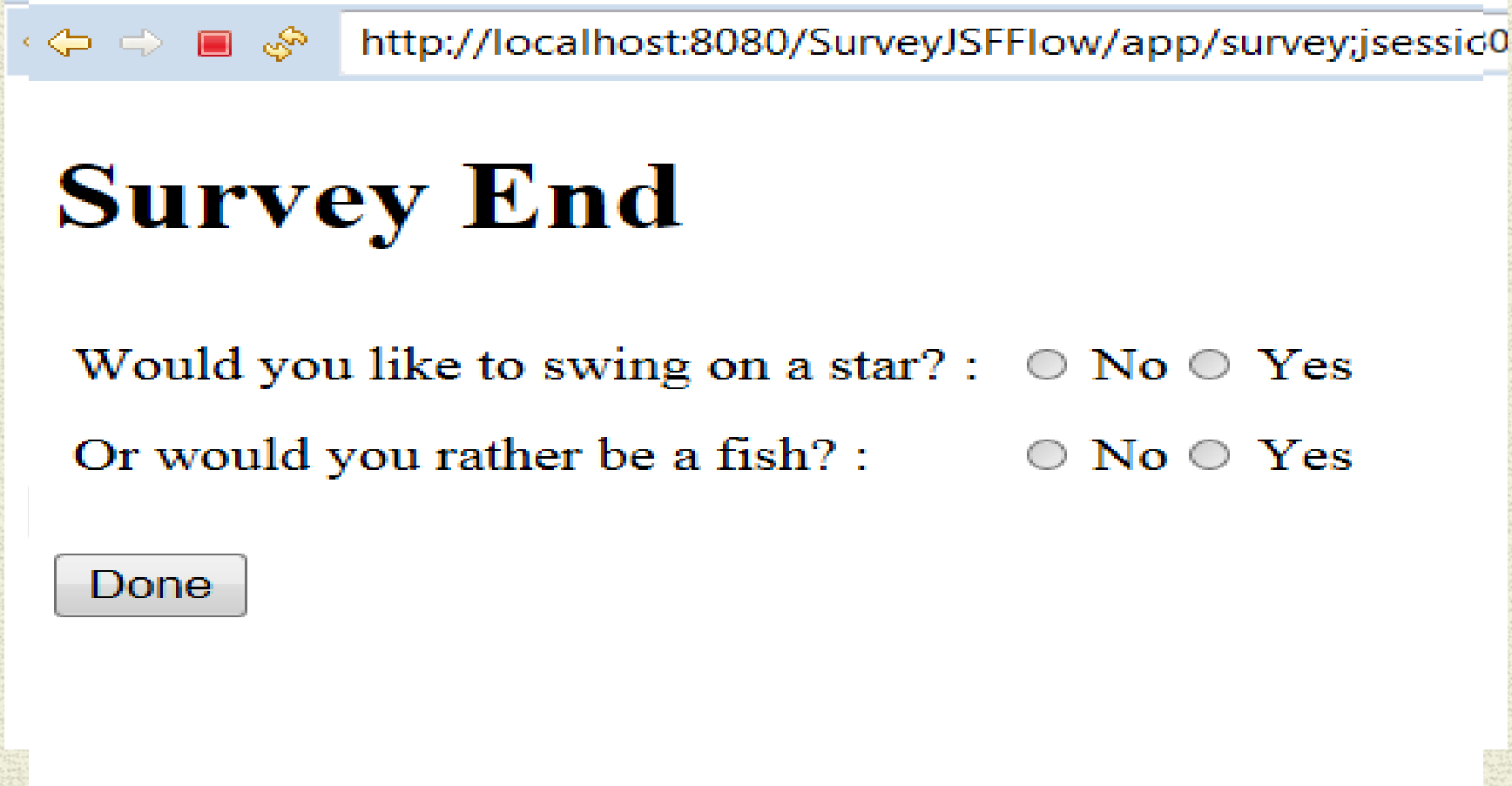
```
@ManagedBean
@RequestScoped
public class UserController implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name ="";

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getWelcomeMessage() {
        String outMessage;
        if (!this.name.isEmpty())
            outMessage = "Hello " + name + "!   "
                    + "Thanks for taking the survey!!";
        else outMessage = null;

        return outMessage;
    }
}
```

```
#{userController.name}
```

```
{userController.welcomeMessage}
```

JSF Custom Component[Composite] Use Case

- Multiple Survey pages Follow a pattern…



- Prime target for a Re-usable component

Survey Custom Component Attribute definition [survey.xhtml]

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
       xmlns:composite="http://java.sun.com/jsf/composite"
```

Attributes are “variables”/placeholders in the custom component

```
    <composite:interface>
        <composite:attribute name="value" />
        <composite:attribute name="action"
            method-signature="java.lang.String action()" />
```

Allows for 4 possible questions:

```
            <composite:attribute name="yesNo1Label" />
            <composite:attribute name="yesNo2Label" />
            <composite:attribute name="question1Label" />
            <composite:attribute name="question2Label" />
```

Answers go here:

```
        <composite:attribute name="survey"
                    type="edu.mum.domain.Survey" required="true" />
    </composite:interface>
```

Survey Custom Component Implementation [survey.xhtml continued]

```
<composite:implementation>
    <h:form>
        <h:panelGrid columns="2" id="textPanel" cellpadding= "3" cellspacing= "3" >
            <c:if test="#{cc.attrs.yesNo1Label != null}">
                #{cc.attrs.yesNo1Label} :
                <h:selectOneRadio id="yesNoOne" value ="#{cc.attrs.survey.yesNoOne}">
                    <f:selectItem itemValue="#{false}" itemLabel="No" />
                    <f:selectItem itemValue="#{true}" itemLabel="Yes"/>
                </h:selectOneRadio>
```

possible question #1

We have “generalized” the original JSF Survey Example from JSF Example

```
            <c:if test="#{cc.attrs.question1Label != null}">
                #{cc.attrs.question1Label} :
                <h:inputText id="answerOne" value="#{cc.attrs.survey.answerOne}" />
            </c:if>

            <c:if test="#{cc.attrs.yesNo2Label != null}">
                #{cc.attrs.yesNo2Label} :
                <h:selectOneRadio id="yesNoTwo" value ="#{cc.attrs.survey.yesNoTwo}">
                    <f:selectItem itemValue="#{false}" itemLabel="No" />
                    <f:selectItem itemValue="#{true}" itemLabel="Yes"/>
                </h:selectOneRadio>
            </c:if>

            <c:if test="#{cc.attrs.question2Label != null}">
                #{cc.attrs.question2Label} :
                <h:inputText id="answerTwo" value="#{cc.attrs.survey.answerTwo}" />
            </c:if>
        </h:panelGrid>
        <p> <h:commandButton action="#{cc.attrs.action}" value="#{cc.attrs.value}" /> </p>
    </h:form>
</composite:implementation>
```

possible question #2

possible question #3

possible question #4

Survey Custom Component Usage

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:custom="http://java.sun.com/jsf/composite/components"
      >

    <h:body>

        <h1>Survey Start </h1>

        <custom:survey survey="#{surveyPages.surveyList[0]}"
        yesNo1Label = "Will you Shop here again?"
        question1Label = "How often?"
        yesNo2Label = "Do you use PayPal?"
        question2Label = "How often?"
        value="Next" action="#{surveyController.submit('survey2')}" />

    </h:body>

  </html>
```

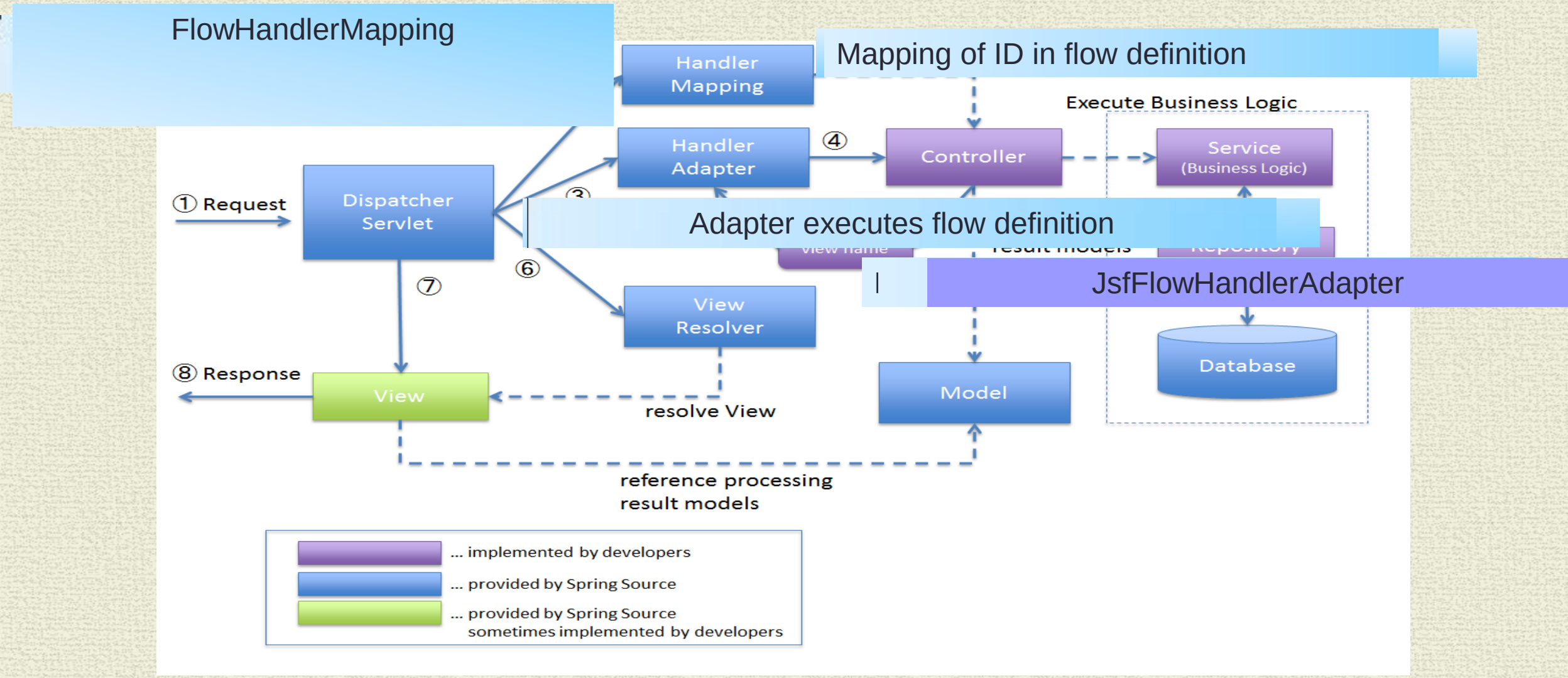
This is the First Page

Where to store answers

Set the Attributes

Page to go to next

Spring MVC Flow



Survey Custom Component in Spring Web Flow

We can “externalize” the question info [survey.properties]

& further simplify/generalize the pages…

```
<section>
    <div class="jumbotron">
        <div class="container">
    <h1>Survey Start </h1>

        </div>
    </div>
</section>

<div class="container">
    <custom:survey survey="#{survey1}" />
</div>
```

See survey-flow.xml in webstoreFlowJSF Demo

Main Point

- The flexibility of the Spring MVC/Web Flow architecture accommodates the use of a JSF custom component within the Spring Framework.

- *Flexibility is a characteristic of a healthy stress-free nervous system.*