

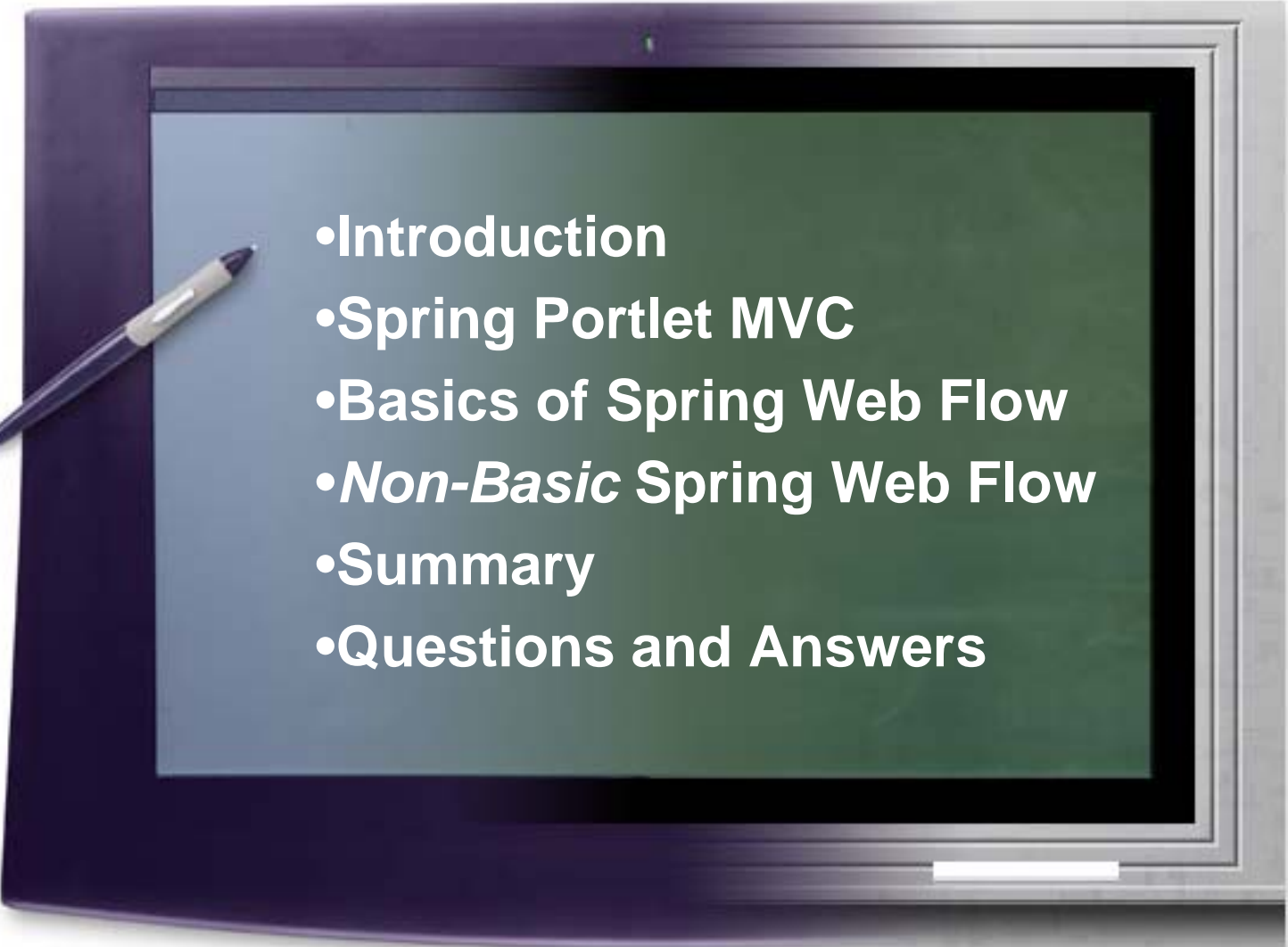


Spring Web Flow

Cris J. Holdorph (holdorph@unicon.net)
Lennard Fuller (lfuller@unicon.net)

Unicon, Inc.
<http://www.unicon.net/>

Agenda

- 
- Introduction
 - Spring Portlet MVC
 - Basics of Spring Web Flow
 - *Non-Basic* Spring Web Flow
 - Summary
 - Questions and Answers



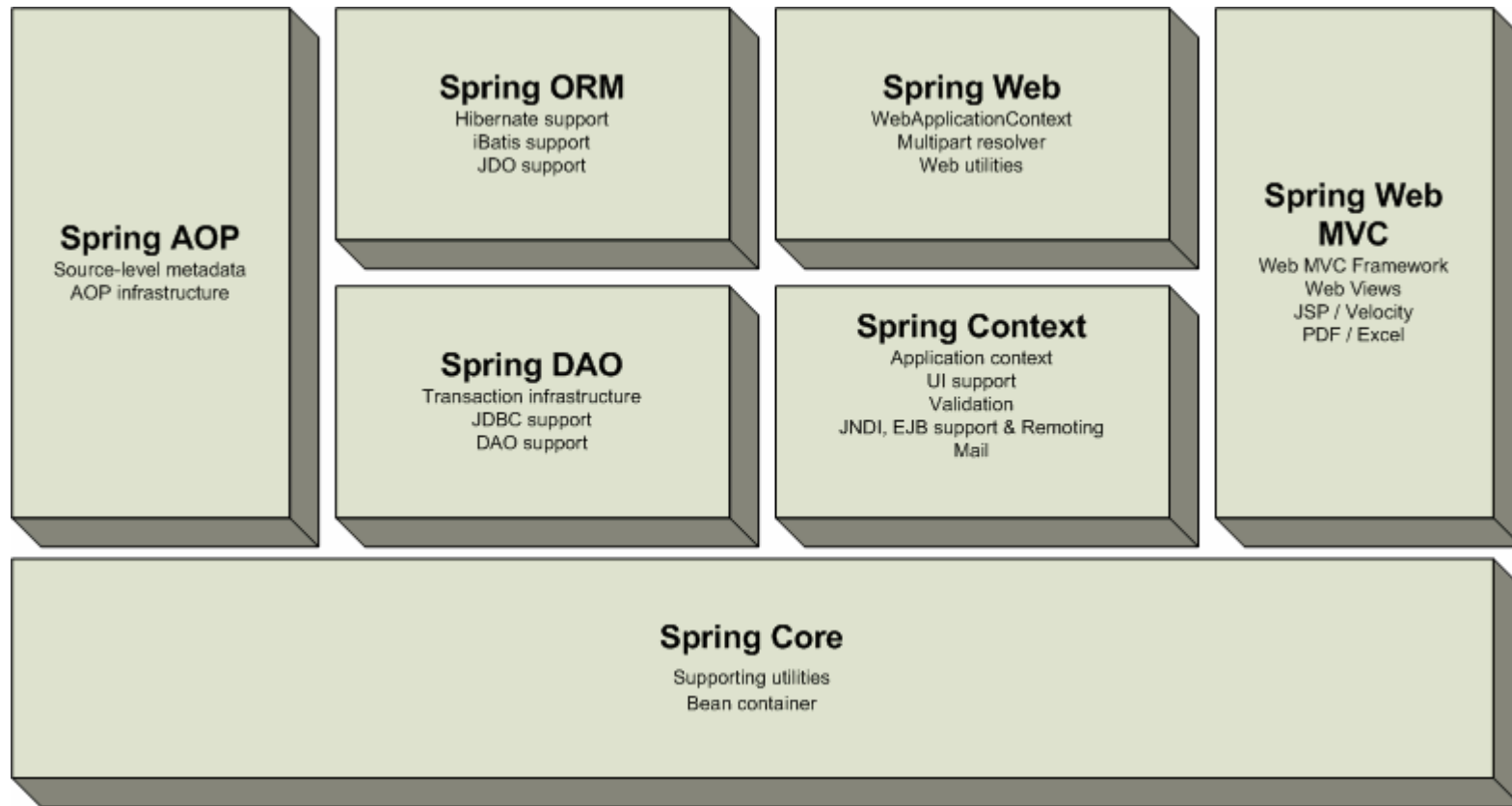
Introduction

- Spring Web Flow
 - Spring Web Flow (SWF) is a component of the Spring Framework's web stack focused on the definition and execution of page flow within a web *or Portlet* application.
(<http://static.springframework.org/spring-webflow/docs/1.0-ea/reference/introduction.html>)
- Speakers
 - Cris J. Holdorph, Unicon, Inc.
 - Lennard Fuller, Unicon, Inc.

History

- Spring
 - Started in 2002 to make J2EE development easier
- Spring Web
 - Part of the original Spring system, contains general servlet/web tier tools
- Spring Web MVC
 - Lightweight MVC framework, allows easy integration to other frameworks
- Spring Portlet MVC
 - Contributed initially by Rutgers in April 2004
 - John Lewis started contributing changes in June 2005
 - Added to main Spring code base for the Spring 2.0 release
- Spring Web Flow
 - Started as a sub-project, first preview release march 2005
 - Will be released as an independent project, 1.0 release soon
 - Depends on Spring 2.0 for Portlet support (otherwise Spring 1.2)

Spring Overview



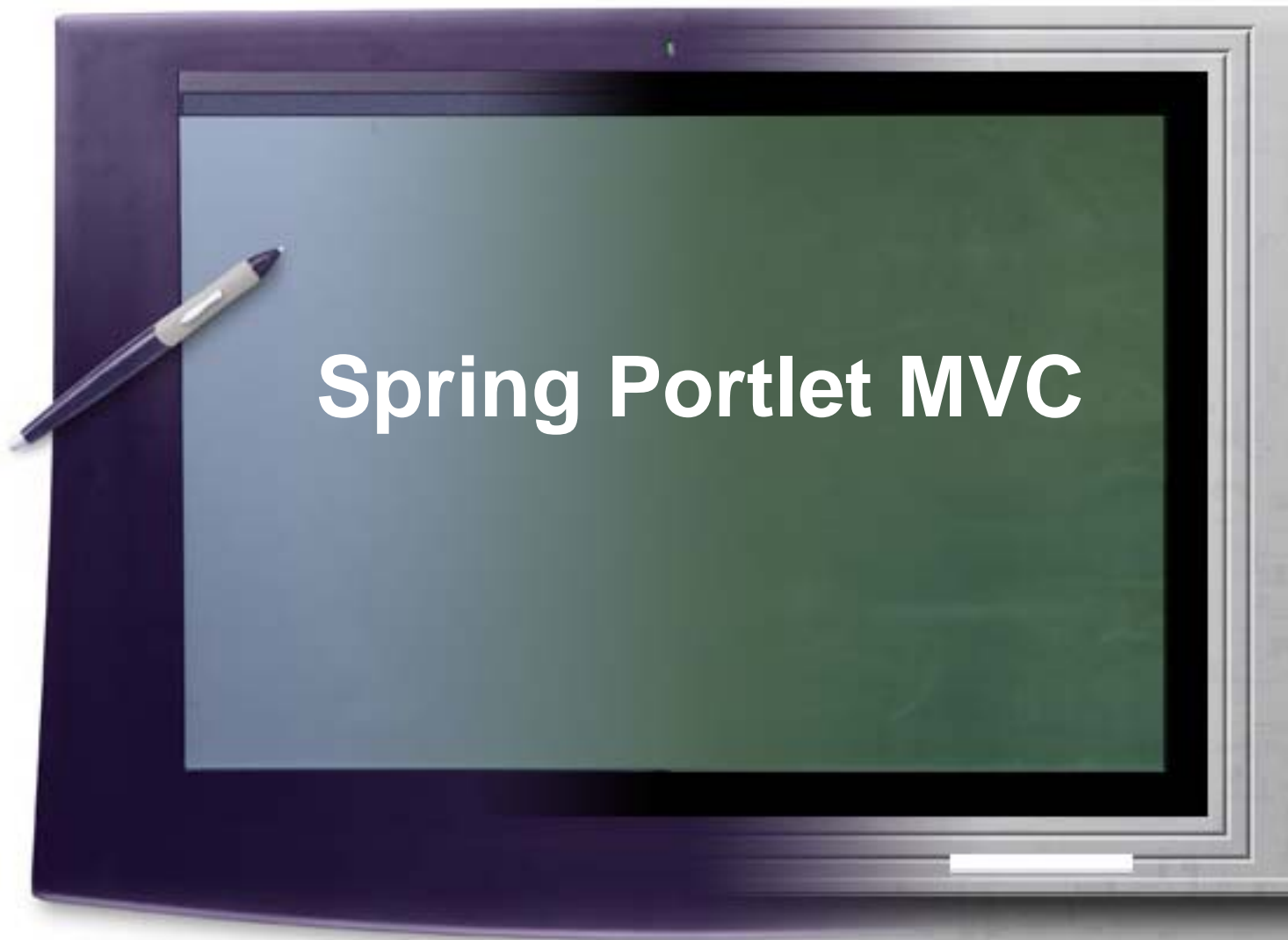
<http://static.springframework.org/spring/docs/2.0.x/reference/introduction.html>

Resources

- Books
- Online Reference Manual
 - Spring
 - Spring Portlet MVC Chapter Just Finished
 - Spring Web Flow
- Wiki (<http://opensource.atlassian.com/confluence/spring>)
 - Spring Portlet MVC
 - Spring Web Flow
- Forums
 - <http://forum.springframework.org/>
- Sample Apps
 - Spring Portlet MVC
 - Spring Web Flow
- This presentation

Books

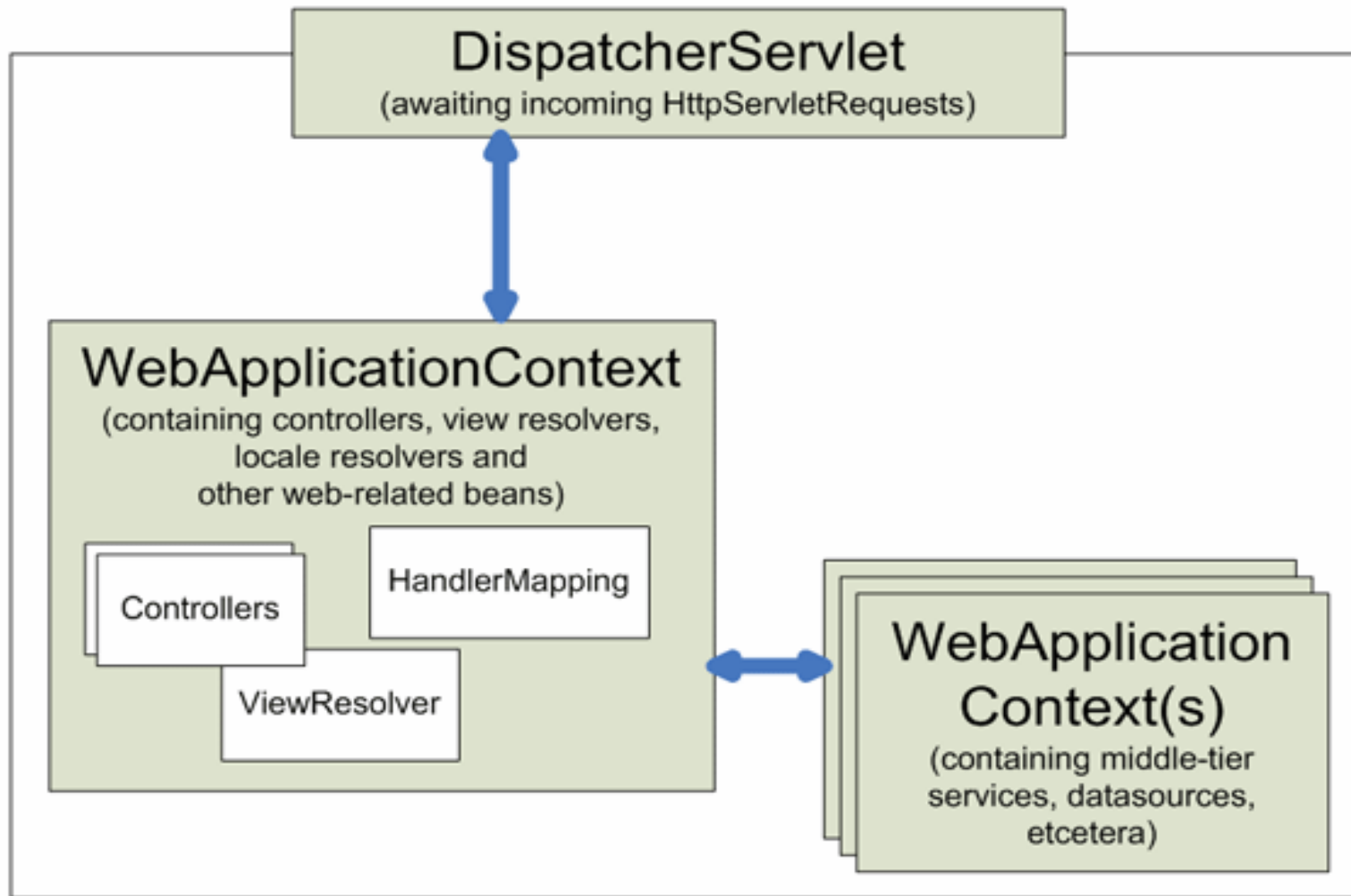
- Expert One-on-One J2EE Development without EJB
- Pro Spring
- Spring in Action
- Spring Live
- Expert Spring MVC and Web Flow



Spring Web/Portlet MVC

- *Spring Web*
 - *WebApplicationContext*
 - *MultiPart Resolver*
 - *Web Utilities*
- *Spring Web MVC*
 - *Controller, View, ModelAndView*
- *Spring Portlet MVC*
 - *Controller, View, ModelAndView*

Spring Web MVC



Spring Web MVC

- WebApplicationContext Bean Types
 - Handler Mapping
 - Controller
 - View Resolver
 - Locale Resolver
 - Theme Resolver
 - Multipart Resolver
 - Handler Exception Resolver

web.xml - DispatcherServlet

```
<web-app>
  ...
  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherS
      ervlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
</web-app>
```

View

```
<bean id="jspViewResolver"  
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
  <property name="viewClass"  
    value="org.springframework.web.servlet.view.JstlView" />  
  <property name="prefix" value="/WEB-INF/jsp/" />  
  <property name="suffix" value=".jsp" />  
</bean>
```

Controller

```
package samples;
public class SampleController extends AbstractController {
    public ModelAndView handleRequestInternal(
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception {
        ModelAndView mav = new ModelAndView("foo");
        mav.addObject("message", "Hello World!"); return mav; }
}
```

```
<bean id="sampleController" class="samples.SampleController">
    <property name="cacheSeconds" value="120"/>
</bean>
```

Handler Mappings

Using a handler mapping, incoming web requests are mapped to appropriate handlers

- Some basic handlers are provided
 - SimpleUrlHandlerMapping
 - This mapping is configurable in the application context and has Ant-style path matching capabilities
 - BeanNameUrlHandlerMapping
 - Maps HTTP requests to named beans in the web application context.

Spring Portlet MVC - Similarities

Very similar to Spring Web MVC

- Driven by a Dispatcher (Portlet)
- Similar beans in `WebApplicationContext`:
 - Handlers, Controllers, View Resolvers, Multipart Resolvers, Exception Handler....
- Supports same view technologies
 - JSP & JSTL, Tiles, Velocity, Freemarker, XSLT, and document views (pdf, excel...)

Spring Portlet MVC - Differences

Developed purely to comply with JSR-168...

- Preserves Action and Render phase separation.
 - If Spring Web MVC has one method, typically Spring Portlet MVC has two.
- Special URL & parameter handling
- No locale or theme resolution support.
 - responsibility of portal/portlet-container
- NOT bundled in core module like Spring MVC, Portlet MVC is instead in the extended module.
- Successfully used with Jetspeed2, uPortal, Gridsphere, Liferay, JBoss, Weblogic, Websphere, and Vignette

Spring Portlet MVC

- WebApplicationContext Bean Types
 - Handler Mapping
 - Controller
 - View Resolver
 - Multipart Resolver
 - Handler Exception Resolver

portlet.xml - DispatcherPortlet

```
<portlet>
    <portlet-name>sample</portlet-name>
    <portlet-
class>org.springframework.web.portlet.Dispatcher
rPortlet</portlet-class>
</init-param>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>view</portlet-mode>
        <portlet-mode>edit</portlet-mode>
        <portlet-mode>help</portlet-mode>
    </supports>
    <portlet-info>
        <title>Sample Portlet</title>
    </portlet-info>
</portlet>
```

web.xml – Spring Portlet MVC - View

```
<web-app>
...

<servlet>
<servlet-name>ViewRendererServlet</servlet-name>
<servlet-
  class>org.springframework.web.servlet.ViewRendererServlet</
  servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

...

<servlet-mapping>
<servlet-name>ViewRendererServlet</servlet-name>
<url-pattern>/WEB-INF/servlet/view</url-pattern>
</servlet-mapping>

...
</web-app>
```

Spring Portlet MVC - controller

```
package sample;

import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import org.springframework.web.portlet.mvc.AbstractController;
import org.springframework.web.portlet.ModelAndView;

public class SampleController extends AbstractController {
    public ModelAndView handleRenderRequestInternal(
        RenderRequest request,
        RenderResponse response)
        throws Exception {
        ModelAndView mav = new ModelAndView("foo");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}

<bean id="sampleController" class="sample.SampleController">
    <property name="cacheSeconds" value="120"/>
</bean>
```

Spring Portlet MVC Handler Mappings

Using a handler mapping, incoming portlet requests are mapped to appropriate handlers

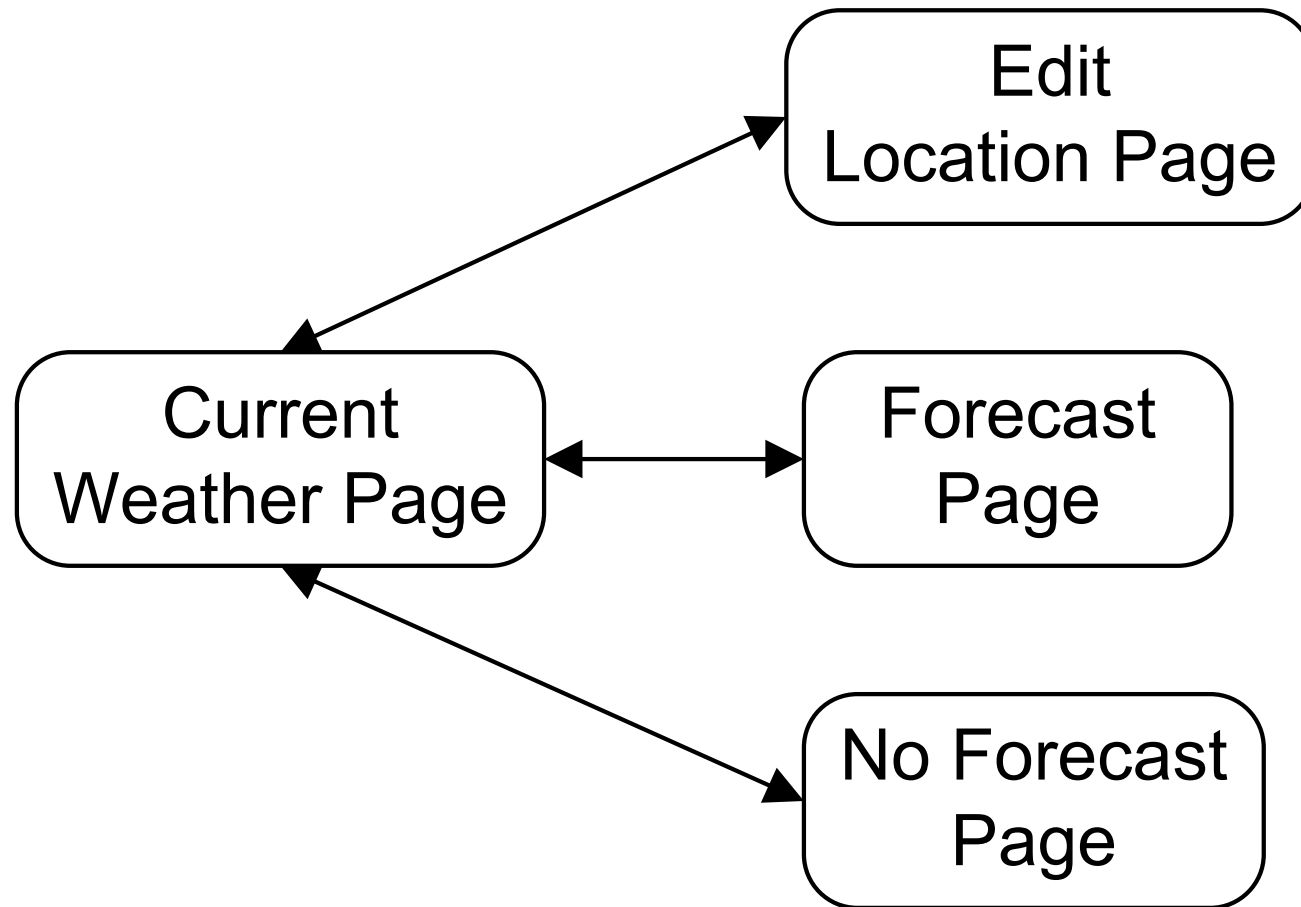
- Some basic handlers are provided
 - PortletModeHandlerMapping
 - Maps requests based on portlet mode ('view', 'edit', 'help',.....)
 - ParameterHandlerMapping
 - Uses a specific request parameter to control mapping (default is 'action').
 - PortletModeParameterHandlerMapping
 - Combines the previous mappings to allow different navigation within each portlet mode..



Spring Web Flow Basics

- *The Flow is King*
- Spring Web Flow (SWF) defines controlled page flows
- Spring Web Flow orchestrates the clients progress through the flow that has been defined
- The main input a programmer gives Spring Web Flow is the *Flow Definition*

Sample Application – Weather Portlet



State Types

- **View State**
 - Renders a view allowing the user to participate in the flow by entering data or viewing a message
- **Action State**
 - Executes your application code, typically delegating to a business service in the middle tier
- **Decision State**
 - Evaluates a condition to drive a transition to a new state
- **End State**
 - Terminates a flow
- **Subflow State**
 - Spawns another flow as a subflow. The spawning flow is suspended until the subflow reaches an end state, at which point the subflow will end and the spawning flow will resume.

Spring Web Flow Building Blocks

- Flows
- States
- Transitions
 - All states except end states are transitionable and maintain a set of one or more transitions that define “allowed paths” to other states. A transition is triggered on the occurrence of an event.
- Events
 - The result of the state’s execution
- Actions
 - The logic executed within a state
 - On flow start
 - On flow end
 - On state enter
 - On state exit
 - Before transition

Defining the Flow

- Define the Flow
- Define the States
- Define the Transitions Between the States

Flow.xml – basic flow element

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE flow PUBLIC "-//SPRING//DTD WEBFLOW
1.0//EN"
"http://www.springframework.org/dtd/spring-
webflow-1.0.dtd">

<flow start-state="currentWeather">
  <view-state id="currentWeather"
view="currentWeather">
  ...
  </view-state>
  ...
</flow>
```

Flow.xml – View States

```
<flow start-state="currentWeather">
  <view-state id="currentWeather"
    view="currentWeather">
    ...
  </view-state>

  <view-state id="forecast" view="forecast">
    ...
  </view-state>

  <end-state id="editLocation"
    view="editLocation" />
  ...
</flow>
```

Flow.xml – Transitions

```
<flow start-state="currentWeather">
  <view-state id="currentWeather"
    view="currentWeather">
    <transition on="submit" to="forecast"/>
  </view-state>

  <view-state id="forecast" view="forecast">
    <transition on="submit" to="forecast"/>
  </view-state>

  <end-state id="editLocation"
    view="editLocation"/>
  ...
</flow>
```

Flow.xml – Actions

```
<flow start-state="currentWeather">
  <view-state id="currentWeather" view="currentWeather">
    <transition on="submit" to="forecast"/>
  </view-state>

  <view-state id="forecast" view="forecast">
    <transition on="submit" to="forecast"/>
  </view-state>

  <view-state id="editLocation" view="editLocation">
    <transition on="submit" to="currentWeather">
      <action bean="formAction" method="bindAndValidate"/>
    </transition>
  </view-state>
  ...
</flow>
```

Flow.xml – Action States

```
<flow start-state="currentWeather">
  <view-state id="currentWeather" view="currentWeather">
    <transition on="submit" to="forecast"/>
  </view-state>

  <view-state id="forecast" view="forecast">
    <transition on="submit" to="forecast"/>
  </view-state>

  <view-state id="editLocation" view="editLocation">
    <transition on="submit" to="applyLocation">
      <action bean="formAction" method="bindAndValidate"/>
    </transition>
  </view-state>

  <action-state id="applyLocation">
    <action bean="weatherService" method="changeLocation()" />
    <transition on="success" to="currentWeather"/>
  </action-state>
  ...
</flow>
```

Flow.xml – Decision States

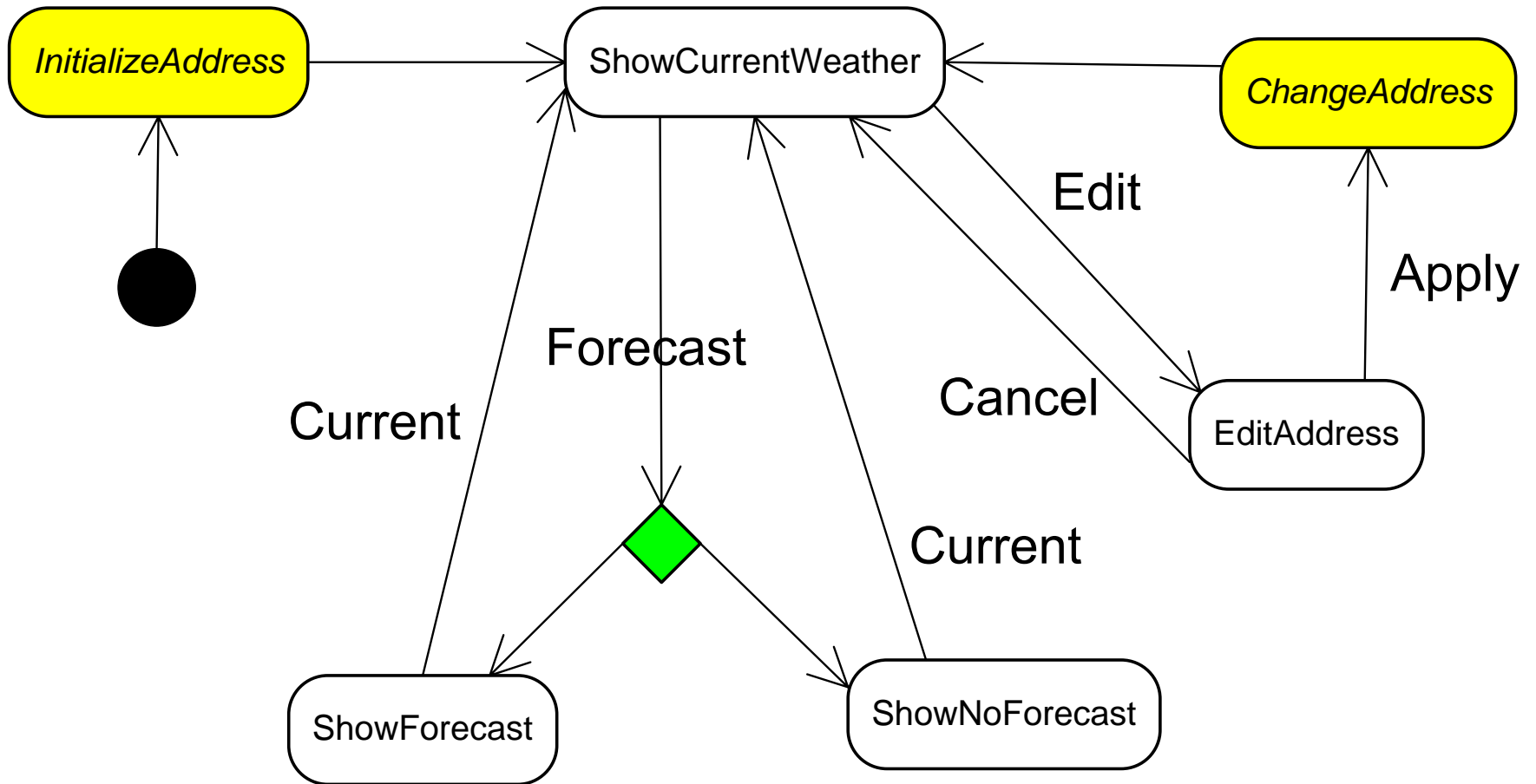
```
<flow start-state="currentWeather">
  <view-state id="currentWeather" view="currentWeather">
    <transition on="submit" to="forecast"/>
  </view-state>

  <decision-state id="forecastAvailable">
    <if test="{flowscope.weather.forecast}" then="forecast"
    else="noForecast"/>
  </decision-state>

  <view-state id="forecast" view="forecast">
    <transition on="submit" to="currentWeather"/>
  </view-state>

  <view-state id="noForecast" view="noForecast">
    <transition on="submit" to="currentWeather"/>
  </view-state>
  ...
</flow>
```

State Diagram for Weather Portlet



Weather Portlet - weather-flow.xml

```
<flow start-state="initializeAddress">

  <action-state id="initializeAddress">
    <action bean="weatherService" method="getWeather()"
    resultName="weather" resultScope="flow"/>
    <transition on="success" to="showCurrentWeather"/>
    <transition on="error" to="changeAddress"/>
  </action-state>

  <view-state id="showCurrentWeather" view="currentWeather">
    <transition on="forecast" to="forecastAvailable"/>
    <transition on="edit" to="editAddress"/>
  </view-state>

  <decision-state id="forecastAvailable">
    <if test="\${flowScope.weather.hasForecast()}"
    then="showForecast" else="showNoForecast"/>
  </decision-state>

</flow>
```

Weather Portlet - weather-flow.xml

```
<view-state id="showForecast" view="displayForecast">  
    <transition on="current" to="showCurrentWeather"/>  
</view-state>
```

```
<view-state id="showNoForecast" view="displayNoForecast">  
    <transition on="current" to="showCurrentWeather"/>  
</view-state>
```

```
<view-state id="editAddress" view="editAddress">  
    <transition on="apply" to="changeAddress">  
        <action bean="formAction"  
method="bindAndValidate"/>  
    </transition>  
    <transition on="cancel" to="showCurrentWeather"/>  
</view-state>
```

Weather Portlet - weather-flow.xml

```
<action-state id="changeAddress">  
    <action bean="weatherService"  
method="getWeather( ${flowScope.newAddress} )  
resultName="weather"  
resultScope="flow" />
```

```
    <transition on="success"  
to="showCurrentWeather" />
```

```
</action-state>
```

```
<import resource="weather-flow-beans.xml" />
```

```
</flow>
```

Weather Portlet - weather-flow-beans.xml

```
<beans>
```

```
<!-- Purchase form action that setups the form and  
processes form submissions -->
```

```
<bean id="formAction"  
class="org.springframework.webflow.action.FormAction">
```

```
    <property name="formObjectName" value="newAddress" />
```

```
    <property name="formObjectClass"  
        value="demo.weather.domain.Address" />
```

```
    <property name="formObjectScope" value="FLOW" />
```

```
    <property name="validator">
```

```
        <bean class="demo.weather.domain.AddressValidator" />
```

```
    </property>
```

```
</bean>
```

```
</beans>
```

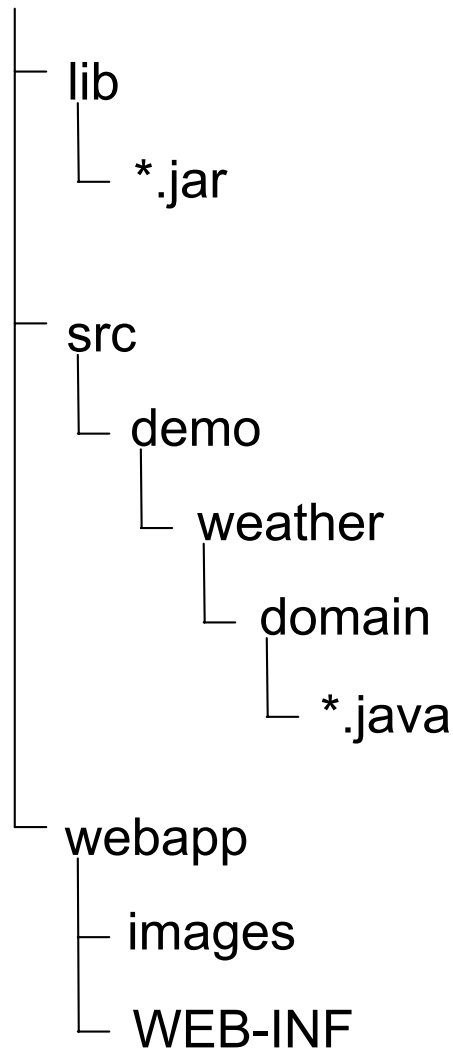
Spring Web Flow – uPortal 2006

Questions and Answers

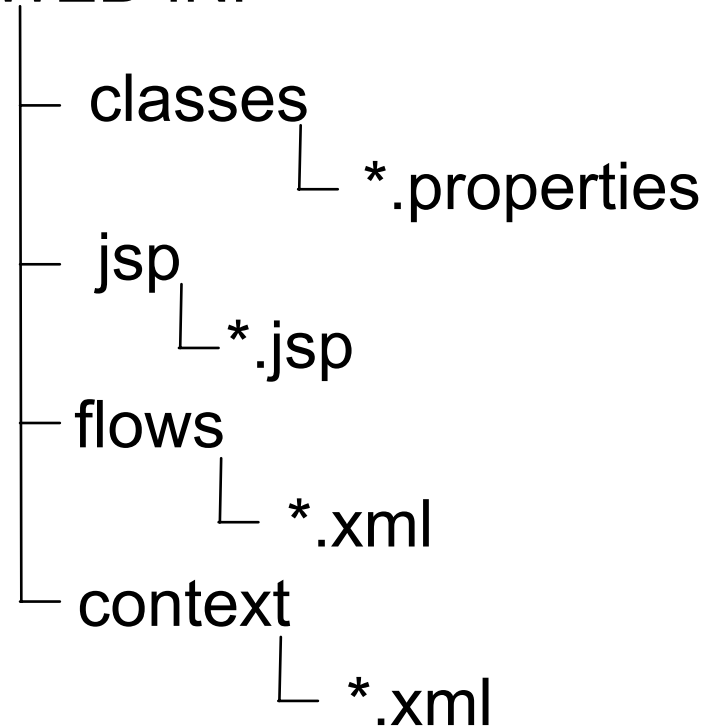


Weather Portlet Directory Structure

WeatherSWF



WEB-INF



Spring Web Flow Portlet

- Specify Spring Portlet MVC configuration in web.xml and portlet.xml
- Specify a Spring Web Flow *FlowController* as the controller for this portlet
- Specify a FlowRegistry for the FlowController to use
- Specify a ViewResolver to resolve the views

Spring Web Flow Portlet XML Files

- ***web.xml*** – the main web application file
- ***portlet.xml*** – the main portlet application file
- ***ApplicationContext.xml*** – the main global Spring application context
- ***Weather-portlet.xml*** – the Spring Portlet MVC spring context for the Weather Portlet
- ***weather-flow.xml*** – the main flow definition
- ***weather-flow-beans.xml*** – the Spring bean configuration for beans local to the weather-flow



Non-Basic Spring Web Flow

- Subflows
- Writing your own Actions
- Complicated Decision States
- Chain of Responsibility Action States
- Testing Flows
- Continuations
- Spring IDE/Spring Web Flow plugin for Eclipse

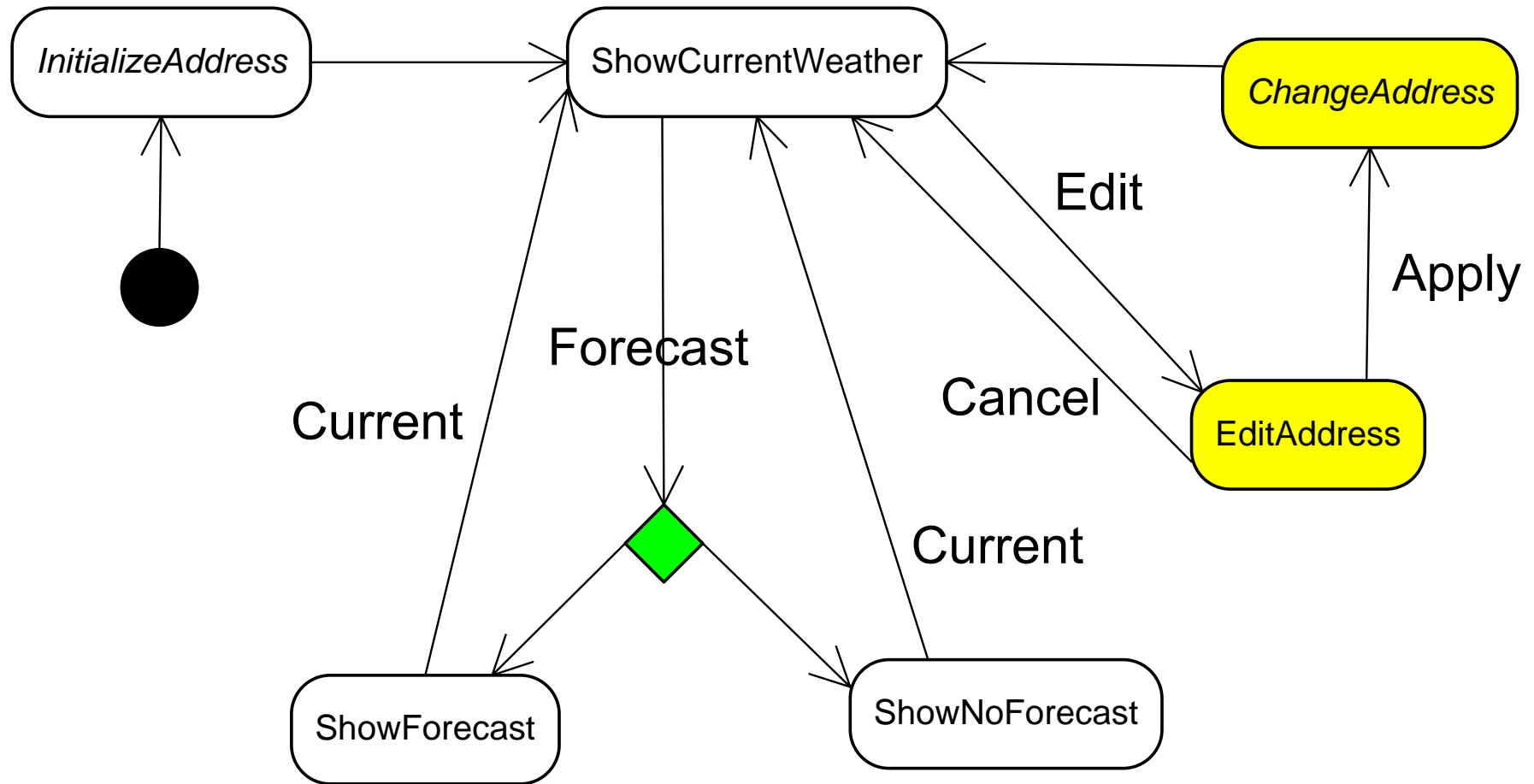
Subflows

- Any flow called by another flow is a ‘subflow’.
- Any flow calling another flow is a ‘parent flow’
- All subflows are flows themselves
- All parent flows can be subflows themselves
- There is no limit to depth of subflow nesting
- Best used to model a sequence of steps when reuse is possible.
- Key feature of Spring Web Flow

Subflow – Refactoring Steps

1. Isolate webflow fragment and dependant beans.
2. Define webflow fragments and dependant beans in their own flow.
3. Define appropriate end states for each outcome of the new flow.
4. In the original flow replace the refactored webflow fragments with a call to the new subflow.

Subflow – Step 1a



Subflow – Step 1b

```
<flow start-state="initializeAddress">

...
<view-state id="editAddress" view="editAddress">
  <transition on="apply" to="changeAddress">
    <action bean="formAction" method="bindAndValidate"/>
  </transition>
  <transition on="cancel" to="showCurrentWeather"/>
</view-state>

<action-state id="changeAddress">
  <action bean="weatherService"
    method="getWeather(${flowScope.newAddress})"
    resultName="weather" resultScope="flow"/>
  <transition on="success" to="showCurrentWeather"/>
</action-state>

<import resource="weather-flow-beans.xml"/>
</flow>
```

Subflow – Step 2

```
<flow start-state="editAddress">
  <view-state id="editAddress" view="editAddress">
    <transition on="apply" to="changeAddress">
      <action bean="formAction" method="bindAndValidate"/>
    </transition>
    <transition on="cancel" to="showCurrentWeather "/>
  </view-state>
  <action-state id="changeAddress">
    <action bean="weatherService"
      method="getWeather(${flowScope.newAddress})"
      resultName="weather" resultScope="flow"/>
    <transition on="success" to="showCurrentWeather "/>
  </action-state>

  <import resource="weather-flow-beans.xml"/>
</flow>
```

Subflow – Step 3

```
<flow start-state="editAddress">
<view-state id="editAddress" view="editAddress">
<transition on="apply" to="changeAddress">
<action bean="formAction" method="bindAndValidate"/>
</transition>
<transition on="cancel" to="finish"/>
</view-state>
<action-state id="changeAddress">
<action bean="weatherService"
  method="getWeather(${flowScope.newAddress})"
  resultName="weather" resultScope="flow"/>
<transition on="success" to="finish"/>
</action-state>
<end-state id="finish"/>
<import resource="weather-flow-beans.xml"/>

</flow>
```

Subflow – Step 4

```
<flow start-state="initializeAddress">
<view-state id="showCurrentWeather"
  view="currentWeather">
  <transition on="forecast"
    to="forecastAvailable"/>
  <transition on="edit" to="editAddress"/>
</view-state>
<subflow-state id="editAddress" flow="edit-
  address">
<transition on="finish" to="showCurrentWeather"/>
</subflow-state>
...
</flow>
```

Writing your own Actions

Two approaches:

1. Implement the Action interface
2. Use spring to adapt your POJOs to use the Action interface.

Implementing the Action interface

```
Public interface Action {  
    Event execute(RequestContext context);  
}
```

Action definition in flow.xml:

```
<flow start-state="initializeAddress">  
...  
<action-state id="changeAddress">  
<action bean="getWeatherAction"/>  
<transition on="success"  
    to="showCurrentWeather"/>  
</action-state>  
...  
</flow>
```

POJO Actions

POJO bean referenced in flow.xml:

```
<flow start-state="initializeAddress" >
...
<action-state id="changeAddress" >
<action bean="weatherService"
  method="getWeather( ${flowScope.newAddress} )"
  resultName="weather" resultScope="flow" />
<transition on="success"
  to="showCurrentWeather" />
</action-state>

...
</flow>
```

Complicated Decision States

- SWF is capable of supporting complicated decision states.
- SWF syntax + developer intuition may provide unexpected results.

Example: Bad Decision State

```
<decision-state>  
  <if test="{flowscope.object.booleanProperty}"  
    then="stateA" else="stateB" />  
  <if test="{this.will.never.be.called}"  
    then="neverGetCalled" />  
</decision-state>
```

An else in an if statement will ALWAYS be evaluated to true, therefore any following ifs will never be called.

Example: Poor Decision State

```
<decision-state>
  <if test="\${flowscope.object.booleanProperty}"
  then="stateA" />
  <if test="\${called if above evaluates to
  false}" then="stateB" />
  <if test="\${called if above evaluates to
  false}" then="stateC" />
</decision-state>
```

If all tests are found to be false, then a `NoMatchingTransitionException` is thrown.

Example: Good Decision State

```
<decision-state>
  <if test="{flowscope.object.booleanProperty}"
  then="stateA" />
  <if test="{called if above evaluates to
  false}" then="stateB" />
  <if test="{called if above evaluates to
  false}" then="stateC" else = "stateD" />
</decision-state>
```

POJO Return Values

POJO Return Value	Event Identifier
null	null
true	yes
false	no
Enum	String representation of Enum

Example: boolean POJO in a decision state

```
<decision-state id="determineTaxable">  
  <action bean="orderClerk"  
    method="isTaxable(${flowScope.taxationRule})" />  
  <transition on="yes" to="enterTaxDetails" />  
  <transition on="no" to="lucky" />  
</decision-state>
```

Chain of Responsibility Action States

SWF supports two strategies for chaining a number of logical actions together:

- 1) Create distinct action states for each of the actions in the chain, transitioning from state to state. (flexible, but verbose)
- 2) Invoke multiple actions within a single action state via the use of named actions. (concise but less flexible)

Chain of Responsibility – Strategy 1

```
<action-state id = "exposedFormObject" >
<action bean="formAction"
  method="exposedFormObject" /><transition
  on="success" to="setupReferenceData" />
</action-state>
<action-state id="setupReferenceData" > <action
  bean="formAction" method="setupReferenceData" />
  <transition on="success"
    to="setupDateChooser" />
</action-state>
<action-state id="setupDateChooser" > <action
  bean="formAction" method="setupDateChooser" />
  <transition on="success" to="nextState" />
</action-state>
```

Chain of Responsibility – Strategy 2

```
<action-state id="setupForm">
  <action name="exposeFormObject"
    bean="formAction" method="exposeFormObject" />
  <action name="setupReferenceData"
    bean="formAction" method="setupReferenceData" />
  <action name="setupDateChooser"
    bean="formAction" name="setupDateChooser" />
  <transition on="setupDateChooser.success"
    to="nextState" />
</action-state>
```

Testing Flows – SWF Support

Class	Purpose
AbstractExternalizedFlowExecutionTests	Base class for flow integration tests that verify an externalized flow definition executes as expected.
AbstractFlowExecutionTests	Base class for integration tests that verify a flow executes as expected.
AbstractXmlFlowExecutionTests	Base class for flow integration tests that verify a XML flow definition executes as expected.
MockExternalContext	Mock implementation of the ExternalContext interface.
MockFlowExecutionContext	A stub implementation of the flow execution context interface.
MockFlowExecutionControlContext	Mock implementation of the FlowControlContext interface to facilitate standalone Flow and State unit tests
MockFlowServiceLocator	A stub flow service locator implementation suitable for a test environment.
MockFlowSession	Mock implementation of the FlowSession interface.
MockParameterMap	A extension of parameter map that allows for mutation of parameters.
MockRequestContext	Mock implementation of the RequestContext interface to facilitate standalone Action unit tests.

Continuations

- Continuations allow SWF to capture a snapshot of the systems state at a point in time. These snapshots can then be resumed.
- Useful in resolving backbutton issues in web applications
- Doesn't SWF's FlowExecution already solve the back button issue?
 - FlowExecution does capture the state... but a single state is not enough.

Continuations – How it works

- Continuations store multiple FlowExecutions for logical points in the user's conversation with the web application.
 - When a user clicks the back button they are taken to a page that references an older FlowExecution. If the user clicks submit, then the old FlowExecution is restored and the request is processed from there.
- Continuations repository can be placed on either server side or client side.

Continuations - Tradeoffs

- Increased Memory usage for additional copies of FlowExecution.
- Increased CPU usage for serialization of additional FlowExecution objects.
- Some users may not understand the behavior

Continuations – Tradeoffs continued

- Some view states should NOT support Continuations.
- Server side continuation repositories requires additional memory
- Client side continuation repositories eliminates the need for continuation server side state, yet introduces an additional security risk.

Continuations – The Big Question

Will SWF Portlet Continuations
resolve the dreaded portal back
button issue?

*No, unfortunately this is still an issue
that the portal must resolve. However,
it does provide a nice building block
for a portal to resolve such an issue.*

Spring Web Flow – uPortal 2006

Spring IDE

The screenshot displays the Eclipse IDE environment for configuring a Spring Web Flow. The main window is titled "Java - sellItem-flow.xml - Eclipse Platform".

Package Explorer: Shows the project structure with a package named "sellItem" containing "src", "dispatcher-servlet.xml", and "sellItem-flow.xml".

Spring WebFlow: Lists the "sellItem" WebFlow Config Set and its associated "sellItem" Beans Config Set.

Outline: Lists the states and actions in the flow:

- setupForm [Action State]
- id=test [Property]
- sellItemAction [Action]
- enterPriceAndItemCount [View State]
- enterCategory [View State]
- requiresShipping [Decision State]
- enterShippingDetails [View State]
- showCostOverview [End State]

Flow Diagram: A visual representation of the flow. It starts with an "Action State" named "setupForm" containing "sellItemAction". This leads to a "View State" "enterPriceAndItemCount", then to another "View State" "enterCategory". From "enterCategory", the flow goes to a "Decision State" "requiresShipping" which contains the expression "\${flowScope.sale.shipping}". This decision state branches into two paths: one leading to "enterShippingDetails" (View State) and another leading to "showCostOverview" (End State).

XML Editor: Shows the XML configuration for the flow. The visible code includes:

```
5 <webflow id="sellItem" start-state="setupForm"
6
7   <action-state id="setupForm">
8     <action bean="sellItemAction"/>
9     <transition on="success" to="enterPr
10    <!-- uncomment to experiment with pro
11    <!-- <property name="role" value="ma
12  </action-state>
13
14  <view-state id="enterPriceAndItemCount" v
15    <transition on="submit" to="enterCate
16    <action bean="sellItemAction" met
17    <property name="validatorMeth
18  </action>
19  </transition>
20 </view-state>
21
22  <view-state id="enterCategory" view="cate
23    <transition on="submit" to="requiresS
24    <action bean="sellItemAction" met
25  </transition>
26 </view-state>
27
28  <decision-state id="requiresShipping">
29    <if test="${flowScope.sale.shipping}"
30  </decision-state>
31
32  <view-state id="enterShippingDetails" vie
33    <transition on="submit" to="showCostO
```

Properties Table: Located at the bottom, it has columns for "Property" and "Value".



Summary

- Spring Framework makes developing in Java easier
- Spring Portlet MVC enables other Portlet Development Frameworks
- Spring Web Flow is one of the best Portlet Development Frameworks available today
- Spring Portlet MVC and Spring Web Flow are extensible

Summary

- Spring Web Flow supports the definition and execution of page flow within a portlet application
- Spring Web Flow lets you program your portlet without importing any spring classes
- Flows are defined in a readable flow definition xml
- Flows can be modeled as state transition diagrams

Questions and Answers

