JSF 2 in WebSphere Portal 8: Having the Best of Both Worlds

Graham Harper
Application Architect
IBM Software Services for Collaboration

Idea behind this session

- WebSphere Portal 8 comes with integrated support for version 2 of JavaServer Faces (JSF)

- How can you make best use of the strengths of the two technologies...
  - …whilst simultaneously avoiding some of the potential pitfalls of integration?

- Take advantage of others' experiences:
  - Patterns* and recommendations
  - Draw upon real-world projects using Portal and JSF together to solve business problems

** I freely admit to abusing this term (as is common these days); including ignoring the "rule of three"*

Agenda

- **Introductions**
- **Why JavaServer Faces (JSF) and WebSphere Portal?**
- **How does the integration work?**
- **Introducing the example**
- **Integration Patterns**
- **State Management Patterns**
- **Navigation Patterns**
- **Performance Patterns**
- **Questions and discussion**

Approach

- Since we have longer slots at this User Group...
- … I'm going to try making changes on the fly to an example application to show some of the patterns …
- … what could possibly go wrong?

Introductions

I'll go first...

- Worked in IBM Software Group since the acquisition of Lotus in 1995
- Developed solutions for customers on WebSphere Portal for over 10 years
- Used JavaServer Faces (JSF) 2 (with PrimeFaces) in WebSphere Portal 8 since last summer

Your turn – a quick survey

- **So, who here has:**
  - Developed with JavaServer Faces (JSF) before?
  - Used JSF 2 specifically?
  - Developed WebSphere Portal applications?
  - Used JSF **in** Portal before?

Why JavaServer Faces (JSF) and WebSphere Portal?

Why JavaServer Faces (JSF)?

- **Productivity**
  - Quick user interface creation
  - Built-in support for things you might otherwise have to code yourself:
    - User interface components
    - Ajax support
    - Validation
    - Page navigation
    - etc.

- **It's a standard (JSF 2.0 is JSR 314)**
  - Skills already in many organisations and in the market
  - IBM support: used for many supplied portlets (and XPages)
  - Supported by tools like Rational Application Developer

Why JavaServer Faces (JSF)?

- **Extensibility**
  - Many plug-in points to enhance the framework
    - Support your own way of working
    - Create reusable custom components and the like
  - Covering a few in this session as we look at ways to address specific situations

- **Third-party enhancements**
  - Many people have embraced the extensibility enthusiastically
  - Particularly for user interface components
  - *Examples:* PrimeFaces, MyFaces Trinidad, RichFaces...

What's new in JSF 2.0?

- **Facelets (not JSP) as the view standard**
- **Composite components**
- **Built-in Ajax support**
- **Improved navigation**
- **More bean scope options**
- **System events**

Why WebSphere Portal?

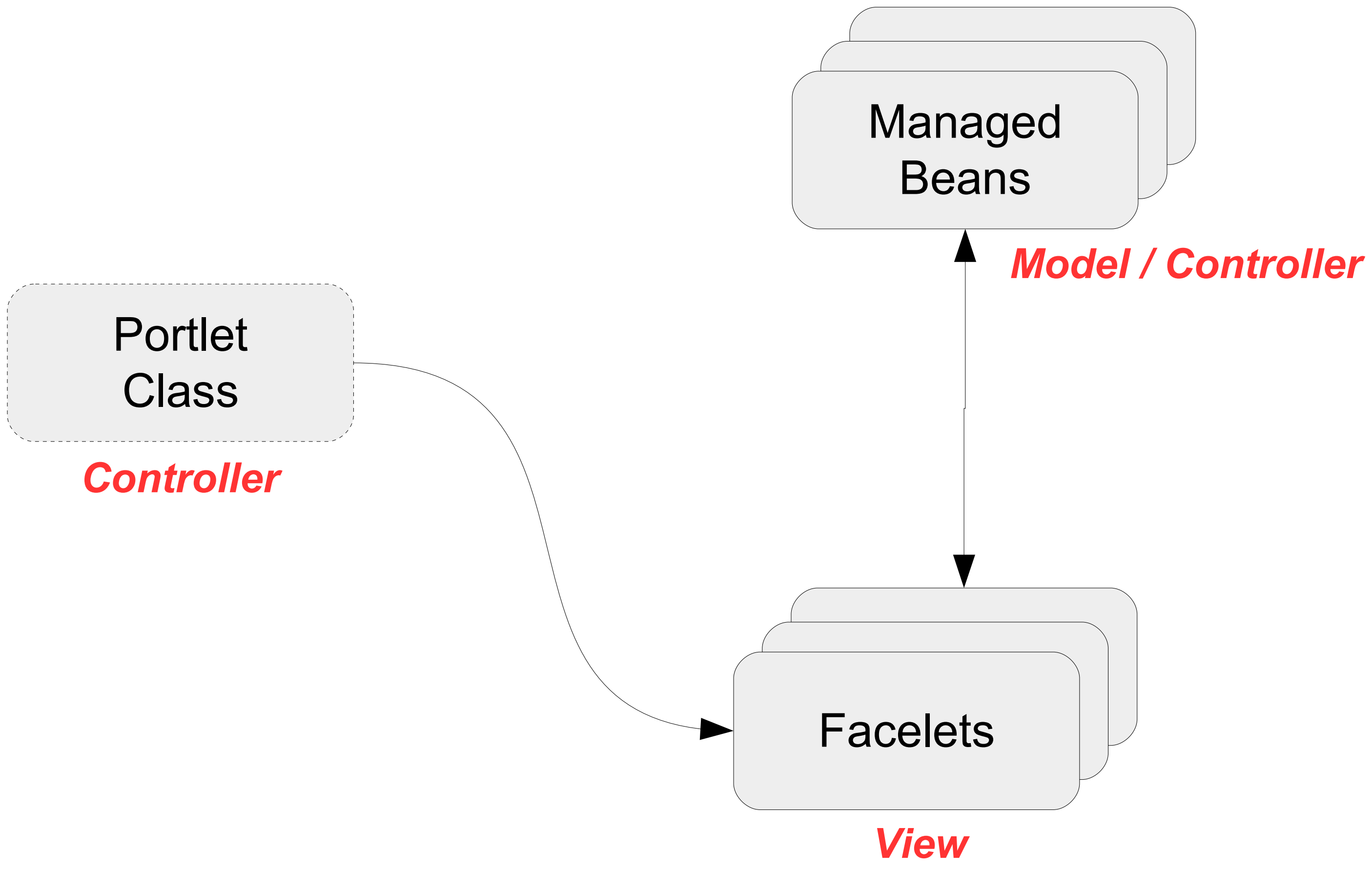
- **Integration “on the glass”**
  - JSF and non-JSF portlets
- **Robust web content management**
  - Sites are invariably a mixture of function and content
- **Security and user management**
- **Personalisation**
- **… and many more**

How does the integration work?

Where does JSF sit?

- **Some portlet applications (WARs) will be JSF applications**
  - Of course, not all need be
- **Portlets in those applications can use JSF features**
  - JSF Portlet Bridge and MyFaces provide the implementation
  - Portlet lifecycle integrated with JSF lifecycle
    - Portal phases:
      - Render, Action and Event
    - JSF phases:
      - Restore View, Apply Request Values, Process Validations, Update Model Values, Invoke Application, Render Response

Typical structure of a JSF portlet



Minimum recommended versions

- **WebSphere Portal 8.0**

- **WebSphere Application Server 8.0.0.5**
  - Important JSF Portlet Bridge fixes
  - *Can (though it's probably unsupported) copy 8.0.0.5 JAR into an 8.0.0.4 install, if really necessary (I only know because I had to...)*

Recommended configuration options

- **In the “web.xml” file of your portlet application, include the following to avoid odd behaviour (RAD does not include these settings by default, unfortunately):**

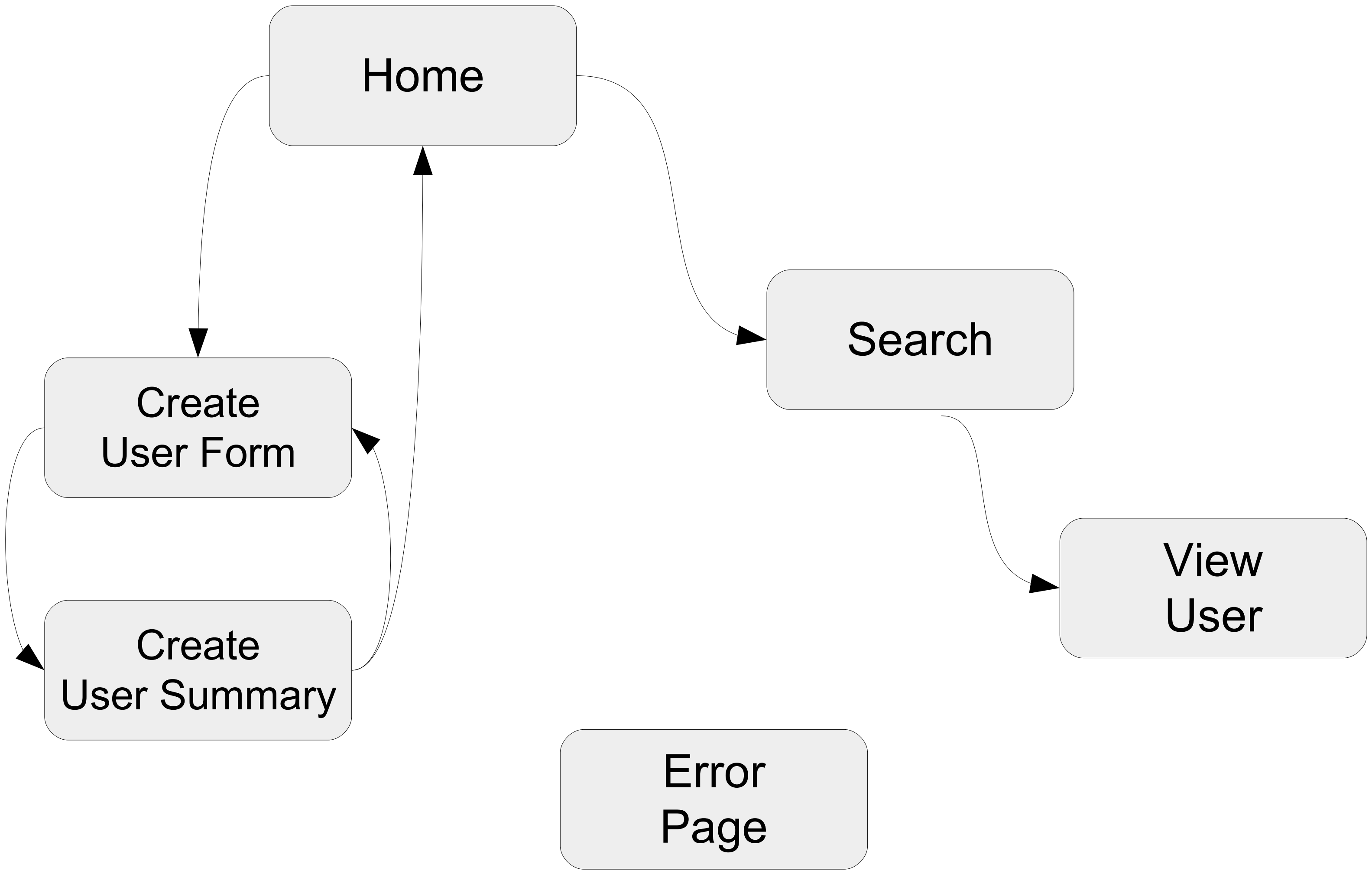
```
<context-param>
        <param-name>org.apache.myfaces.ERROR_HANDLING</param-name>
        <param-value>false</param-value>
</context-param>

<context-param>
        <param-name>javax.faces.PARTIAL_STATE_SAVING</param-name>
        <param-value>false</param-value>
</context-param>
```

Introducing the Example

A user management example

- **“Business” requirements:**
  - Create new portal users
    - Put them into groups representing departments and projects
  - Search for and view existing portal users
- **Example requirements:**
  - Must show the situations I want to talk about
  - … so please excuse me if some of it feels a bit artificial!

Example application logical flow



Example approach

- **Will concentrate on front-end and JSF integration**
  - Pre-written helper classes to interact with PUMA etc.
  - Won't bother showing you the code for these (unless you ask)

- **We have limited time**
  - One simple portlet cannot demonstrate all the patterns I want to talk about
    - So I have prebuilt most of the front-end parts too
    - But I'll show you the code
    - And we'll enhance it on the fly in the session (what could possibly go wrong?)

- **I'll use core JSF tags only**
  - There are many choices of extra component libraries, so I won't push any in particular

Integration Patterns:
Playing to the strengths of the technologies

Integration approach

- ***High Level Problem***
  - We want to use the most appropriate technology for each aspect of solution creation

- ***Solution***
  - Well, let's break it down a bit...

Data capture and validation

- ***Problem***
  - Accept user input then convert, validate and process it

- ***Solution***
  - JSF provides data capture components, converters and declarative validation
  - Also many nice additions in this area (though you many need third party libraries):
    - Typeahead
    - Dialogs
    - Attractive “widgets”

Data capture and validation example

- Creating a user

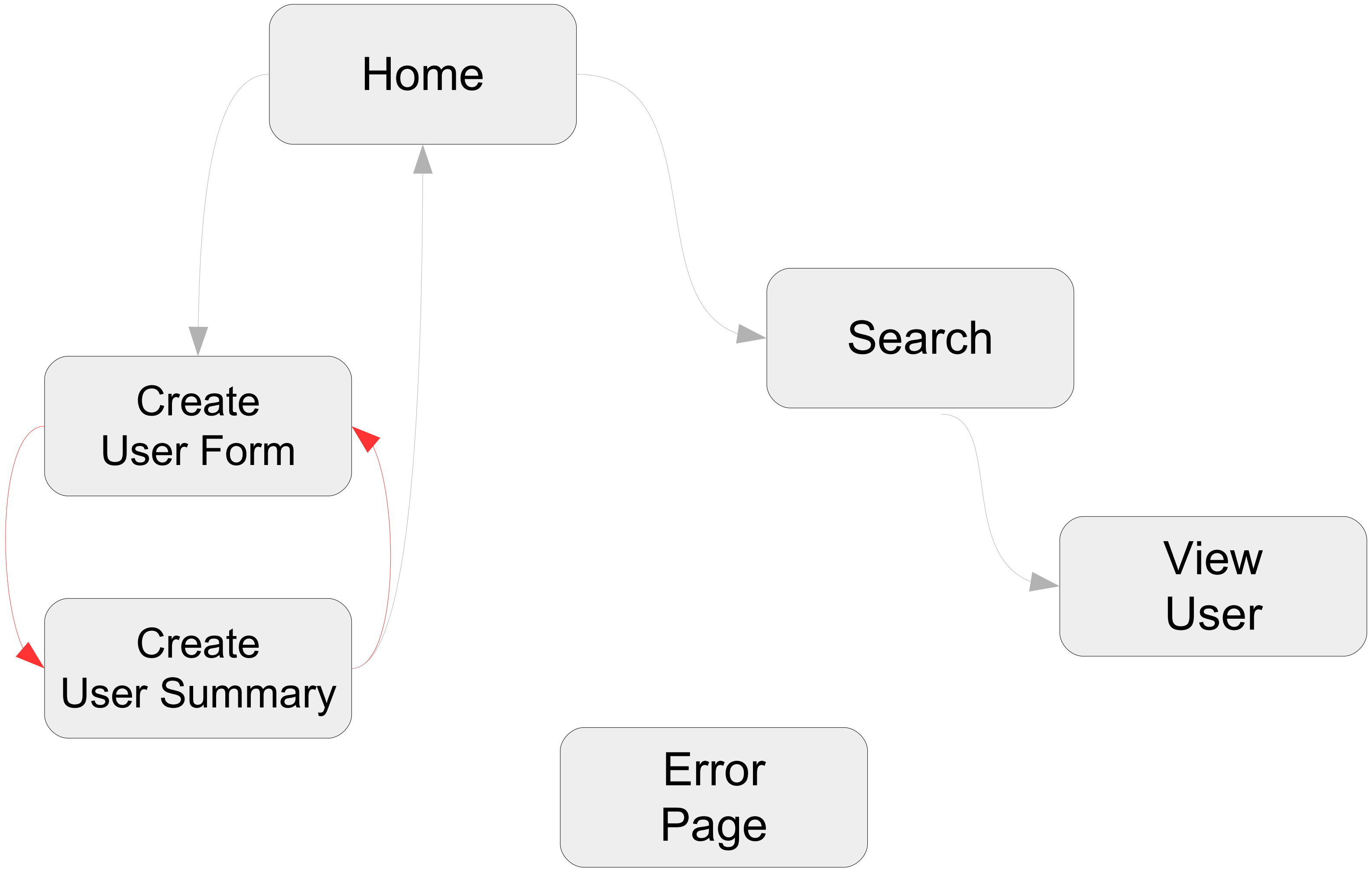
```
<h:outputLabel value="First Name:" styleClass="outputLabel"
                 id="label1" for="text1"></h:outputLabel>
<h:inputText styleClass="inputText" id="text1"
                 value="#{userUnderCreation.firstName}"
                 binding="#{pc_CreateUserView.text1}">
        <f:validateRequired></f:validateRequired>
</h:inputText>
<h:message for="text1"></h:message>
```

Structured process screenflows

- ***Problem***
  - Multi-screen process (e.g. wizard) needed

- ***Solution***
  - JSF provides declative control of navigation between facelets
  - Validation framework stops progress with invalid data
  - Cannot “jump in” to a page part way through the process

Structured process screen flow example



Page composition and layout

- ***Problem***
  - Need to integrate processes and content from multiple sources

- ***Solution***
  - Create multiple JSF portlets where appropriate
    - Gives flexibility in layout
    - Allows reuse
    - Can take advantage of personalisation based on role or other factors
  - Add non-JSF portlets to portal pages where needed
    - Decoration of core process portlets with content and other portlets

Page composition and layout example

- Home page
  - Multiple JSF portlets and one not
  - Skins included to easily show portlet delineation

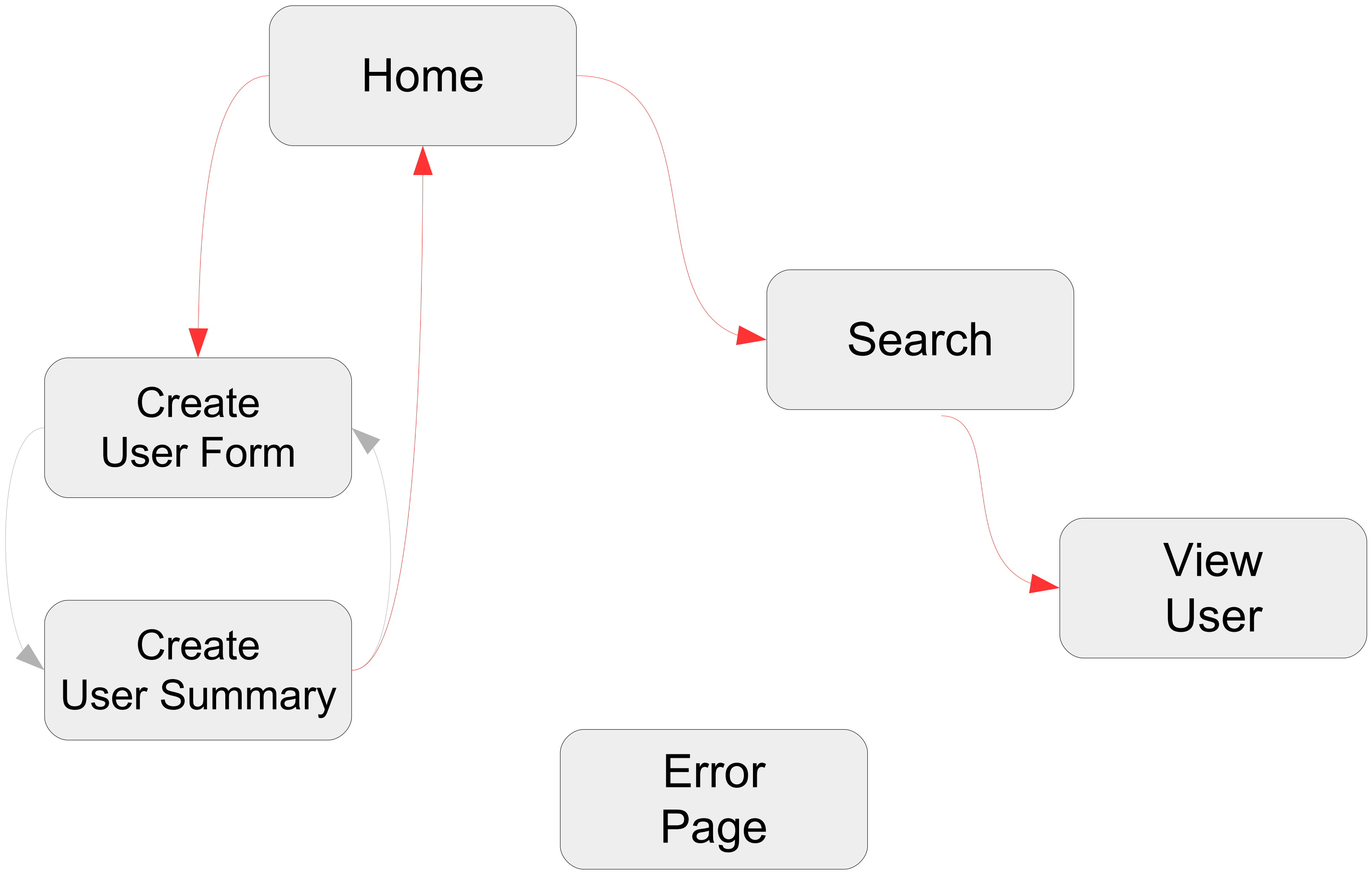
- Search portlet reuse

Higher level navigation

- ***Problem***
  - Need to navigate to around an application, outside individual processes

- ***Solution***
  - Have multiple portal pages and use portal navigation between them
    - Navigation links in theme
    - Cross-page wires with events
    - Selection Model API in portlets
  - Allows direct entry to different parts of a complex application
    - Bookmarking
    - URI resolvers

Higher level navigation example



State Management Patterns

State management

- ***High Level Problem***
  - Applications need to store user state that is not persisted as business data, such as:
    - Navigational choices
    - Progress in processes
    - Unsaved entered data (e.g. with validation errors)

- ***Solution***
  - We'll look at some of the options and then the best approach for certain situations

JSF state storage

- **View state persistence**
  - State of components (e.g. entered values in fields, messages) saved by the framework
  - Similar to portlet session in scope

- **Managed bean scopes**
  - Request
  - View
  - Conversation
  - Session
  - Application
  - Custom

Portal state storage

- **Render parameters**
  - Stored in URL and passed to portlet(s)
    - Persist in bookmarks
    - Should be very limited in size (i.e. keys only)
  - Can be declared “public” and shared between portlets
  - Can only be set in Action Phase
- **Portlet preferences**
  - Less transitory storage, not really “state”
  - Only updateable in Edit Mode

Storing state data for individual portlets

- ***Problem***
  - Portlets need to store data that persists between page refreshes and is easily accessible by JSF

- ***Solution***
  - Use JSF managed beans
  - Use session scope for most beans with non-transient portlet data
    - Saved across portal page refreshes (needed due to actions in other portlets and page navigations)
    - Saved across different facelets within a portlet
  - Have encountered issues trying to use view scope in portal
    - Injection of common beans seems to break, for instance

Storing state data for individual portlets example

- Search criteria persist

- faces-config.xml is where scope is configured in this example
  - Java annotations are also an option

```
<managed-bean>
        <managed-bean-name>searchCriteria</managed-bean-name>
        <managed-bean-class>
                com.ibm.wug.model.SearchCriteria
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Shared state between portlets for a given user

- ***Problem***
  - Need to share user state between two portlets or two instances of the same portlet (e.g. on different pages)

- ***Solution***
  - In portal, JSF “session” scope applies to a **particular portlet instance** for a specific user, so not suitable
  - JSF “application” scope applies to all portlets in the WAR for **all users**, so not suitable
  - JSR 286 has another kind of session - the portlet application session - that applies to all portlets in the WAR for a specific user
  - We can make this available to JSF using one of its extensibility features: custom scopes

Portlet application session scope

- We can create a new scope for JSF beans which I've called "Portlet Application Session Scope" (not being very creative)

- Just requires:
  - The creation and registration of an Expression Language (EL) resolver
  - And the creation of a custom map class

- Already imported into our sample application

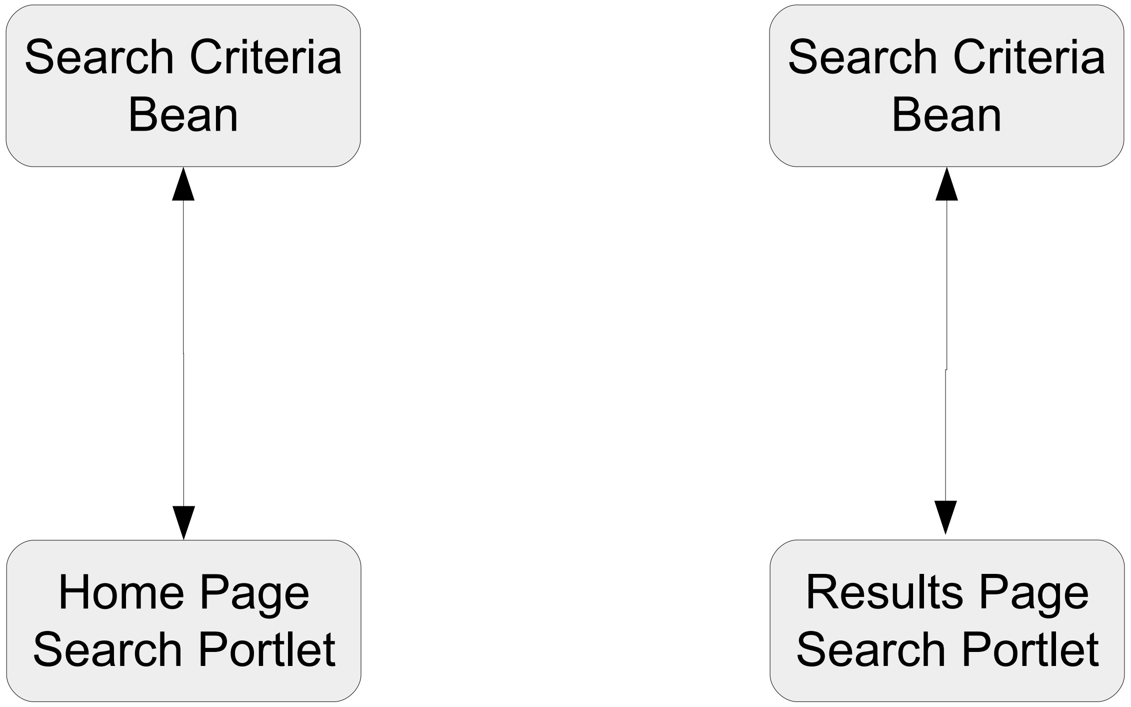
- Used by specifying a scope of "#{portletApplicationSessionScope}" in the "faces-config.xml" file

Portlet application session scope example

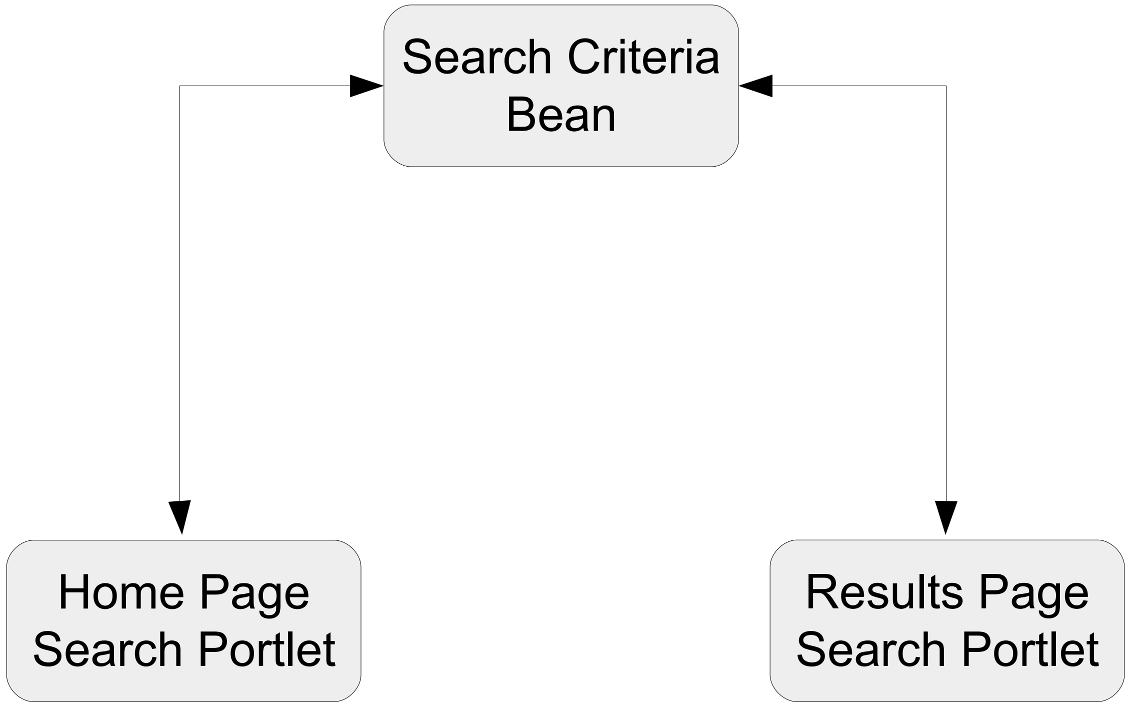
- Search criteria are not currently persisted between the instances of the search portlet on the “Home” and “Search Results” pages

```
<managed-bean>
        <managed-bean-name>searchCriteria</managed-bean-name>
        <managed-bean-class>
                com.ibm.wug.model.SearchCriteria
        </managed-bean-class>
        <managed-bean-scope>
                #{portletApplicationSessionScope}
        </managed-bean-scope>
</managed-bean>
```

Using session scope



Using portlet application session scope



Bookmarkable multi-portlet state

- ***Problem***
  - Need state that can persist within a bookmark or be encoded in a URL emailed to a user
    - Therefore needs to persist between user sessions
    - *Example:* URL to display details for a specific customer

- ***Solution***
  - JSF scopes are not encoded in a portal URL
  - URL query string parameters are not supported by JSR 286
  - Use render parameters and particularly **public** render parameters which are shared by multiple portlets
    - Keep number and size of parameters small

Bookmarkable multi-portlet state example

- Search Results portlet sets public render parameter "SelectedUser" when a user from the results list is clicked
- View User portlet reads the parameter to determine which user to display

```
portletResponse.setRenderParameter(DisplayedUserWrapper.PRP_DISPLAY_USER,
                                   user.getId());
```

Navigation Patterns

Allow declarative portal navigation from JSF

- ***Problem***
  - JSF allows declarative navigation among facelets using outcomes and navigation rules
  - We would like to do the same for portal navigation (sending an event or an explicit page change) from JSF for consistency and flexibility

- ***Solution***
  - Create a custom navigation handler for JSF that interprets a particular destination syntax as portal navigation and uses the portal APIs to get there
  - Note that portal events can only be generated in response to an action, so that form of navigation only possible on JSF actions that do a POST

JSF declarative navigation for portal

- Create a custom navigation handler (extend the JSF 2 ConfigurableNavigationHandler)
- Register it in “faces-config.xml”

```
<application>
          <view-handler>com.ibm.faces20.portlet.FaceletPortletViewHandler</view-handler>
          <el-resolver>com.ibm.faces20.portlet.PortletELResolver</el-resolver>
          <resource-handler>
                    com.ibm.faces20.portlet.httpbridge.PortletResourceHandler
          </resource-handler>
          <navigation-handler>
                    com.ibm.wug.jsf.navigation.PortalNavigationHandler
          </navigation-handler>
</application>
```

JSF declarative navigation for portal

- Create navigation rules with portal destinations in “faces-config.xml”

```
<navigation-rule>
          <from-view-id>/ButtonPortletView.xhtml</from-view-id>
          <navigation-case>
                    <from-outcome>NewUser</from-outcome>
                    <to-view-id>portal-event:CreateUser</to-view-id>
          </navigation-case>
</navigation-rule>


<navigation-rule>
          <from-view-id>/CreateUserComplete.xhtml</from-view-id>
          <navigation-case>
                    <from-outcome>GoHome</from-outcome>
                    <to-view-id>portal-page:wug.page.home</to-view-id>
          </navigation-case>
</navigation-rule>
```

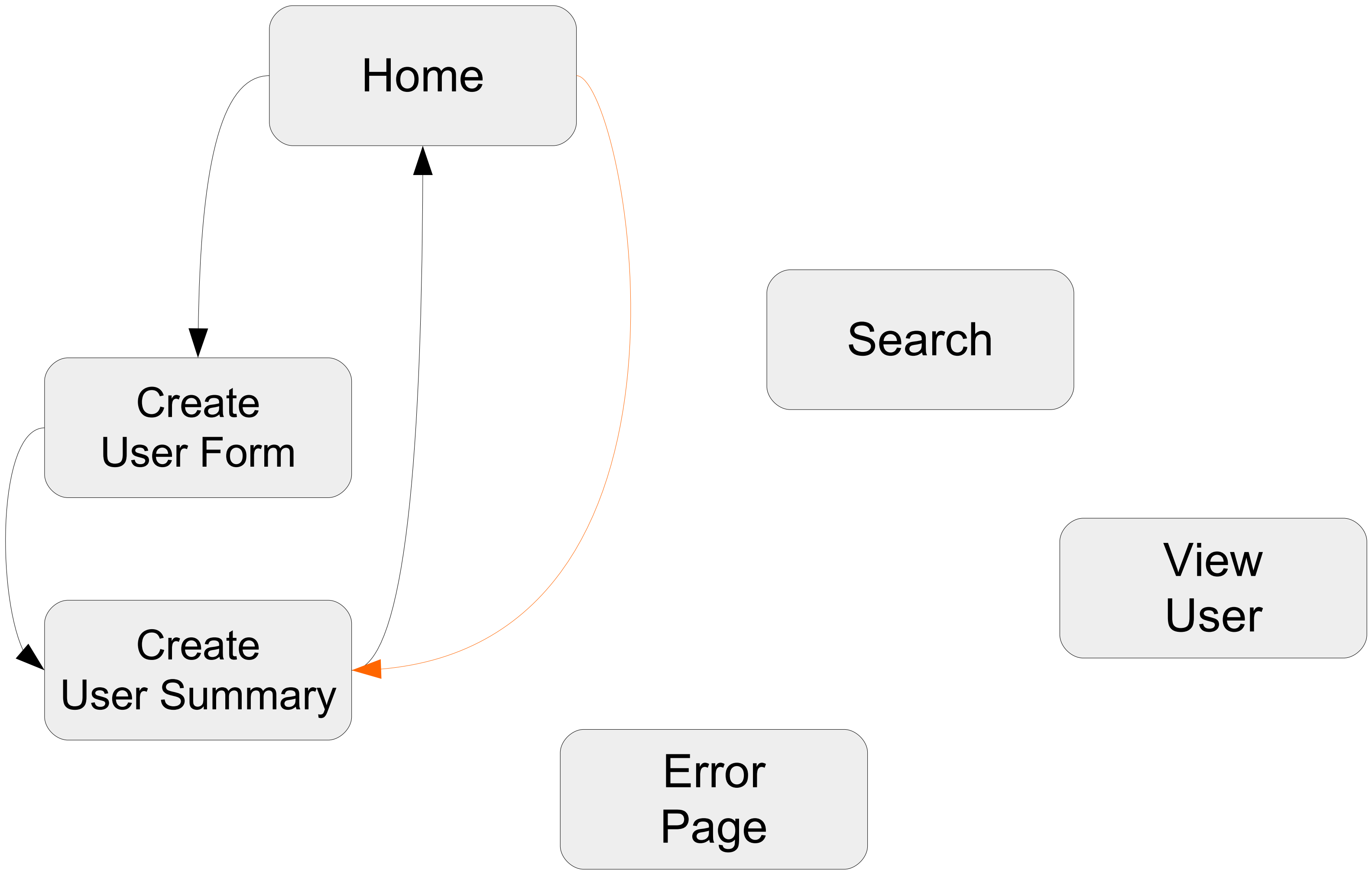
JSF declarative navigation for portal example

- Let's create some links from the Create User summary page
  - Back to the Create User first page
    - A standard JSF navigation to a different facelet
  - To the home page
    - An explicit portal page navigation

Reflect portal navigation in JSF saved state

- ***Problem***
  - Portal navigation actions not triggered from a JSF portlet are not visible to JSF, so JSF state might not be as desired when return to portlet
    - *Example:* go to home page from Create User summary page via theme navigation links, press “New User” button – still on summary page
- ***Solution***
  - Use portal events to navigate to places where this will be an issue and reset JSF state on receipt
    - Needs a custom portlet class to override “processEvent”
    - Several different things you might want to reset

Navigation if state not reset



Resetting JSF state on received event

- **Reset model data in managed bean**
  - Call setters or a reset method

- **Reset user input**
  - Call “resetValue” on UIInput components
    - Easiest to bind them to properties in a managed bean
  - Reset faces messages to clear errors
  - JSF and portal lifecycle interactions mean this has to be done during rendering
    - So just set a flag in a managed bean during event processing

- **Reset current JSF facelet**
  - *Don't* override getView() in portlet class as stated in portal documentation
  - Use navigation to change the view instead
  - Save the state

Resetting JSF state on received event example

- On receipt of the CreateUser event, call model bean reset method, set view bean reset flag and reset the current view (i.e. facelet):

```
// Reset the current data values in the model
ELContext elContext = facesContext.getELContext();
User controller = (User) elContext.getELResolver().
                              getValue(elContext, null, "userUnderCreation");
controller.reset();

// Reset the fields on the form
CreateUserView view = (CreateUserView) elContext.getELResolver().
                              getValue(elContext, null, "pc_CreateUserView");
view.setResetNeeded(true);

// Reset the current portlet facelet
NavigationHandler nav = facesContext.getApplication().getNavigationHandler();
nav.handleNavigation(facesContext, null, "/CreateUserView.xhtml");

facesContext.renderResponse();
super.saveViewState(facesContext);
```

Resetting JSF state on received event example

- The model bean reset method:

```
public void reset() {
        this.firstName = null;
        this.lastName = null;
        this.password = null;
        this.department = null;
        this.project = null;
}
```

Resetting JSF state on received event example

- On the “preRenderView” event during the event phase, the following method is called from a tag and resets the fields etc.:

```
<f:event type="preRenderView" listener="#{pc_CreateUserView.checkReset}"/>
```

```
public void checkReset(ComponentSystemEvent event) throws UserManagementException {
        if (isResetNeeded()) {
                resetFields();
                setResetNeeded(false);
        }
}
```

```
                public void resetFields() {
                        if (getText1() != null) {
                                getText1().resetValue();
                        }
                        if (getText2() != null) {
                                getText2().resetValue();
                        }
                        ...
                        resetFacesMessages();
                }
```

Resetting JSF state on received event example

- Resetting the faces messages is a little non-intuitive:

```
public void resetFacesMessages() {
        FacesContext context = getFacesContext();
        if (context != null) {
                Iterator<FacesMessage> messages = context.getMessages();
                while (messages.hasNext()) {
                        messages.next();
                        messages.remove();
                }
                Iterator<String> itIds =
                                context.getClientIdsWithMessages();
                while (itIds.hasNext()) {
                        List<FacesMessage> messageList =
                                context.getMessageList(itIds.next());
                        if (!messageList.isEmpty()) {
                                messageList.clear();
                        }
                }
        }
}
```

Handle public render parameter changes in portlets

- ***Problem***
  - Portlets using session scope beans might be holding data inconsistent with the current values of public render parameters
  - Especially likely if arrived at using a saved bookmark or a portal page navigation not involving an event

- ***Solution***
  - During the render phase the portlet needs to check the public render parameters it consumes and refresh its data if inconsistent
  - In a complex application this refresh can potentially be moved to a base class and handled using callbacks

Handle public render parameter changes example

- The ViewUser portlet already does this in the “DisplayedUserWrapper” bean class

```
public User getUser() throws UserManagementException {
          String displayUserId = getPublicRenderParameter(PRP_DISPLAY_USER);
          if ((displayUserId != null) && (user == null ||
                                    !user.getId().equalsIgnoreCase(displayUserId))) {
                    user = retrieveUser(displayUserId);
          }
          return user;
}
```

Capture JSF errors and redirect to an error page

- ***Problem***
  - JSF exceptions are not handled very elegantly by default – not something you would want a user to see
    - Instead we would like to go to a new “we're sorry” page in portal

- ***Solution***
  - Don't want to put this functionality everywhere exceptions might be thrown
  - So, create a custom JSF exception handler and let it do the redirect (and any logging etc.)
    - Redirection on the server is fine when an action is being processed, unfortunately the handler needs to deliver JavaScript redirect code to the browser instead when catching a render phase exception

Creating a custom exception handler

- Extend “ExceptionHandlerWrapper” class
- Extend “ExceptionHandlerFactory” class to create instances of the above
- Register the factory in “faces-config.xml”:

```
<factory>
          <exception-handler-factory>
                    com.ibm.wug.jsf.error.PortalExceptionHandlerFactory
          </exception-handler-factory>
</factory>
```

Capture JSF errors and redirect to an error page example

- “Easter egg” in Search Results portlet throws an exception in JSF

Performance Patterns

Responsive user interface

- ***Problem***
  - Page refresh is slow due to size of markup or slow portlets on the page
- ***Solution***
  - Use JSF's built in Ajax support to only refresh those parts of the page that are affected by a change (such as a user selection)

Responsive user interface example

- Filter search results via department
  - Change update to use Ajax

```
<f:ajax event="valueChange" execute="@this" render="@form"></f:ajax>
```

Efficient retrieval of backend data

- ***Problem***
  - Data should be loaded by at most one back-end call per rendering of a portlet (and then only if needed)
  - Data may be accessed by multiple components on a single facelet
- ***Solution***
  - Can't use a “Post Create” method of a bean, if using session scope
    - Could theoretically if using request scope, but I've had issues
  - The order of method calls from the page is not stable when the user interface is changed and we would like to avoid adding boilerplate code in multiple “getters” to check if data is up to date
  - Use a “pre-render” event handler tag and bean method

Efficient retrieval of backend data example

- Lists of groups for drop-downs in Create User portlet could be retrieved in this way
  - Luckily we are already calling the “checkReset” method on the “preRenderView” event to reset the form, so add a line to get the groups in here

```
<f:event type="preRenderView" listener="#{pc_CreateUserView.checkReset}"/>
```

```
public void checkReset(ComponentSystemEvent event) throws UserManagementException {
        if (isResetNeeded()) {
                resetFields();
                setResetNeeded(false);
        }
        groups = PumaHelper.getInstance().getGroupsPrivileged();
}
```

Questions and discussion

References

- **Articles on Rational tooling for JSF and Portal**
  - http://public.dhe.ibm.com/software/dw/rational/zip/Consumption_of_JSF_2_0_in_a_JSR_286_Portlet_Application_using_RAD_8_5_Portal_Toolkit.zip
  - http://public.dhe.ibm.com/software/dw/rational/zip/Utilizing_the_power_of_JSF2-0_and_JSR_286_technology_on_WebSphere_Portal_using_RAD_v8-5_portal_tools.zip
- **Core JavaServer Faces (3rd edition) book**
  - http://www.amazon.co.uk/Core-JavaServer-Faces-Sun-Series/dp/0137012896/ref=sr_1_1?ie=UTF8&qid=1363794733&sr=8-1

Backup slides

Dynacache for backend data caching

- Keeps data out of session
- Shared across portlets (reduces duplication)
- Can be synchronised across nodes if desired
- Can be cleared and viewed from external tool
- Support for expiry times / max entries etc.
- Can inject values for local testing!
- Classes for keys and values must be in classloader shared across all using components
- Requires some setup (in WAS Console or application properties file)