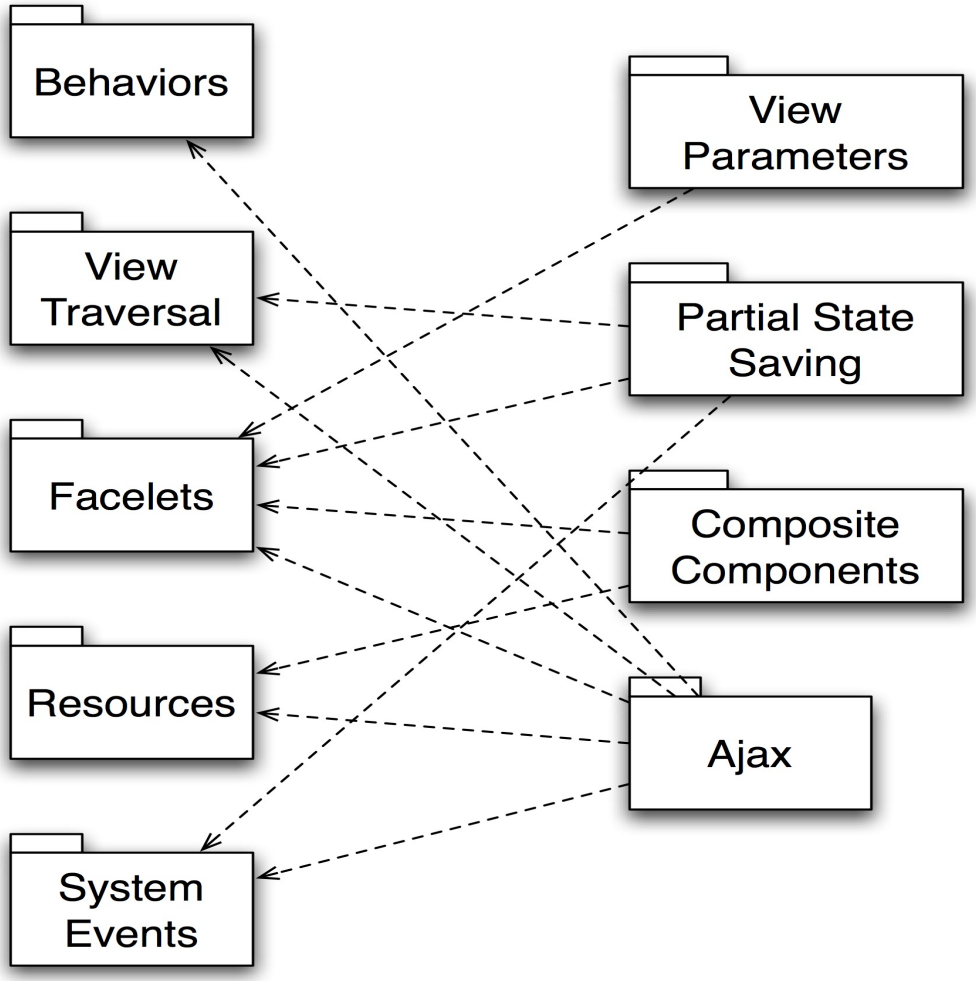
JavaServer Faces 2.0 Overview

Roger Kitain
Specification Lead

Hi Level Categories of Features

Foundational New Features



Large New Features

System Events

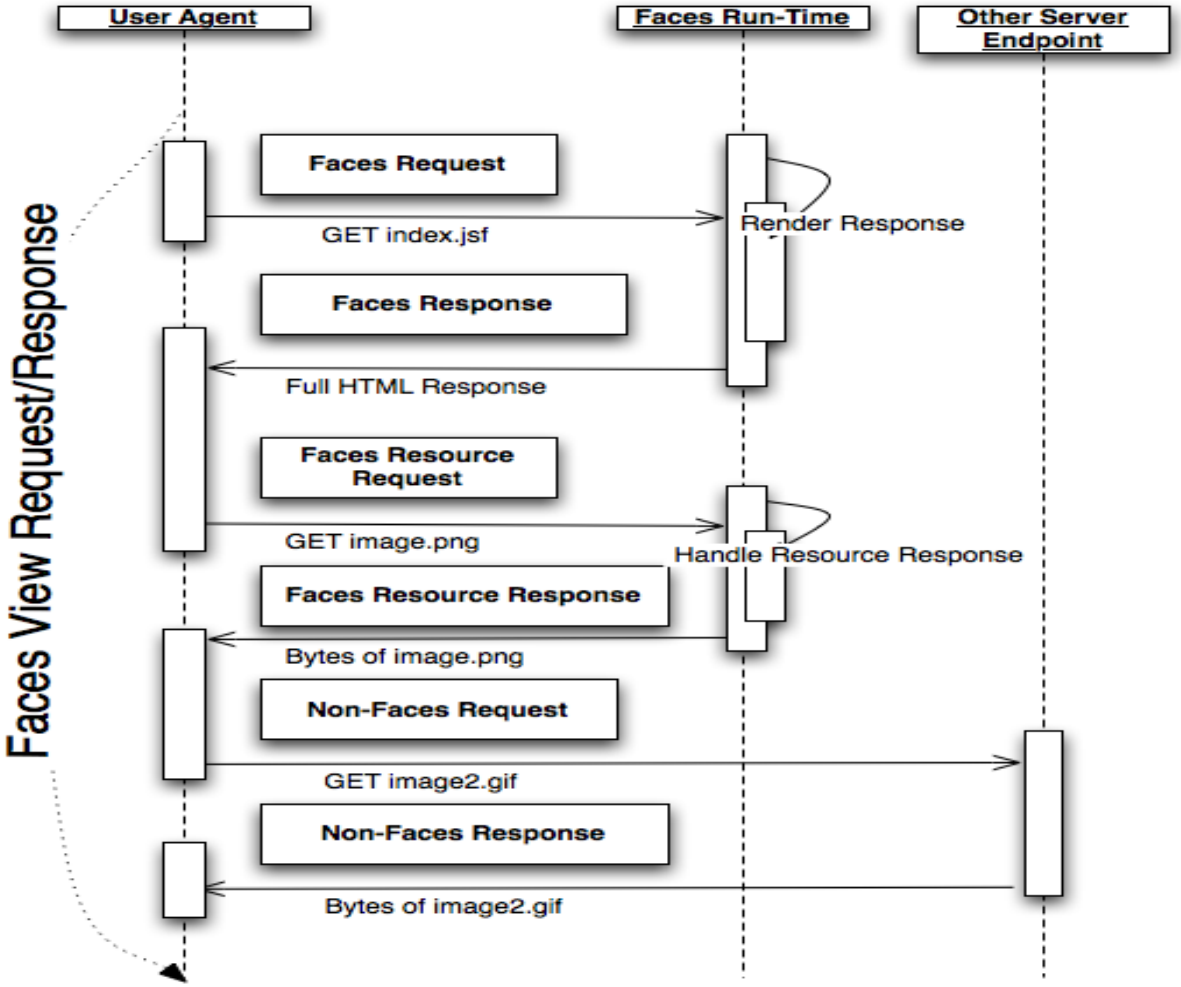
- Publish/Subscribe event model
- JSF Lifecycle events
- Listeners registered at 3 scopes:
  - Component: UIComponent.subscribeToEvent()
  - View: UIViewRoot.subscribeToEvent()
  - Application: Application.subscribeToEvent()
- Publish using:
  - Application.publishEvent()
- Declarative component event listener registration:

```
<h:inputText>
   <f:event type=”preValidate” listener=”#{bean.preVal}”/>
</h:inputtext>
```

Resources

- No need for separate Filter or Servlet
- Loaded from Classpath or filesystem
- Library / i18n / Versioning Support
- Java API:
  - @ResourceDependency ; @ResourceDependencies
    - Attributes: library name; resource name
    - “Target” Attribute: head, body, form
- Markup:
  - <h:head> <h:body> <h:outputScript> <h:outputStylesheet>

Resources


User Agent
Faces Run-Time
Other Server Endpoint
Faces View Request/Response
Faces Request
GET index.jsf
Render Response
Faces Response
Full HTML Response
Faces Resource Request
GET image.png
Handle Resource Response
Faces Resource Response
Bytes of image.png
Non-Faces Request
GET image2.gif
Non-Faces Response
Bytes of image2.gif

View Traversal

- Multiple components in the view visited for a single traversal
- Useful for component(s) processing
- VisitContext
  - Specifies components to visit
  - createVisitContext()
- UIComponent.visitTree()
  - Performs component(s) visit
- Usages: Ajax ; State Saving

Partial State Saving

- JSF 1.x State Saving:
  - Tedious/error prone for component developers
  - Performance issues(session bloat)
- JSF 2.0: Partial State Saving
  - Only save what's changed since initial view creation
  - Per view state size up to 4 x smaller
  - Default for JSF 2.0 components (written with Facelets)
  - PartialStateHolder
    - signals when initial state is configured : markInitialState()
  - StateHelper
    - Storage map for component state (attributes,listeners,...)
    - No need for saveState/restoreState

GET Request Support

View Parameters

- Map incoming GET request parameter values to any EL bean/property
- Mappings specified with <f:viewParam> component:

```
<f:metadata>
  <f:viewParam name="user" value="#{bean.userName}"/>
</f:metadata>
```

- GET Request: page1.xhtml?user=johndoe
- <f:viewParam> is EditableValueHolder
  - converters/validators can be used on view paramaters

GET Request Support
<h:link> <h:button>

- Issue GET requests without hand coding destination URL
- Uses JSF navigation system for determining destination

```
<h:link outcome="success"/>
```

- Destination determined at render time
  - > Outcome mapped to target view identifier
- Integrate with view parameters:

```
<h:link outcome="success" includeViewParams="true"/>
```

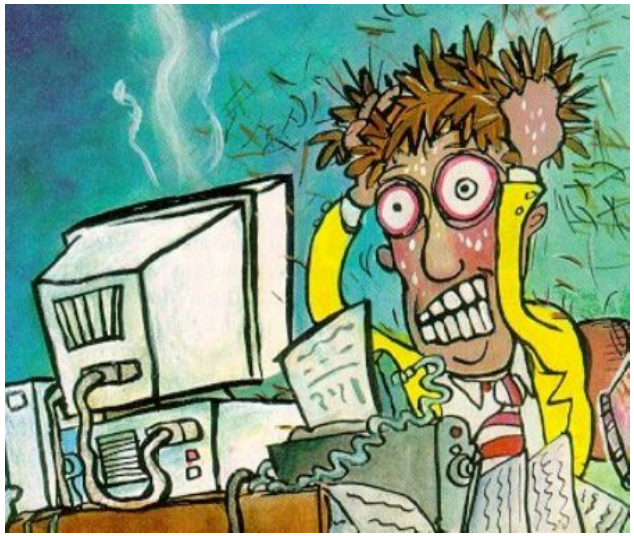
- Can use <f:param> to specify parameters too

JSF 2.0 Component Model

Why Do We Need It?

- With the 1.x component model – too many artifacts, hooks:
  - Component class
  - Renderer
  - Tag class
  - Tld
  - faces-config.xml

  .....



JSF 2.0 Component Model

- Powerful tools:
  - Templating
- Facelets is the foundation
  - Optimized for JSF
  - XHTML and tags
  - Eliminates translation/compilation
  - Templating
  - Composite Components

JSF 2.0 Component Model

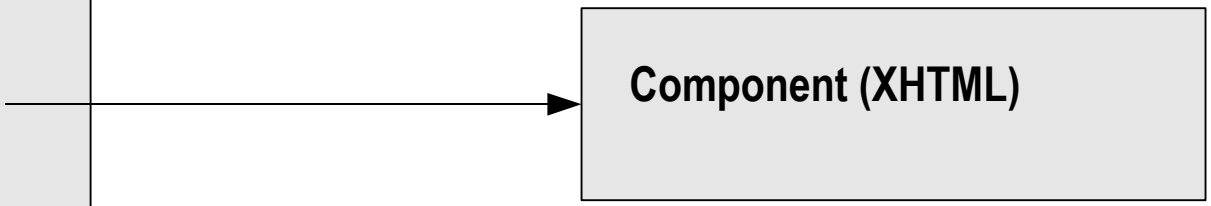
- Facelets 2.0
  - Contains most of the standard Facelets features
  - Enhanced to work with JSF 2.0 component building
    - Easily attach listeners, validators, converters
    - Namespaces created “automagically” – no more taglibs
  - Supports Composite Components
  - It's in the specification !

Composite Components

- Turns page markup into a JSF UI component with attached validators, converters, listeners
- True abstraction:
  - Reuseable component

Using Page (XHTML)

```
<html …
xlms:my="http...."">
<my:comp
 value="yes" />
</html>
```



Composite Components

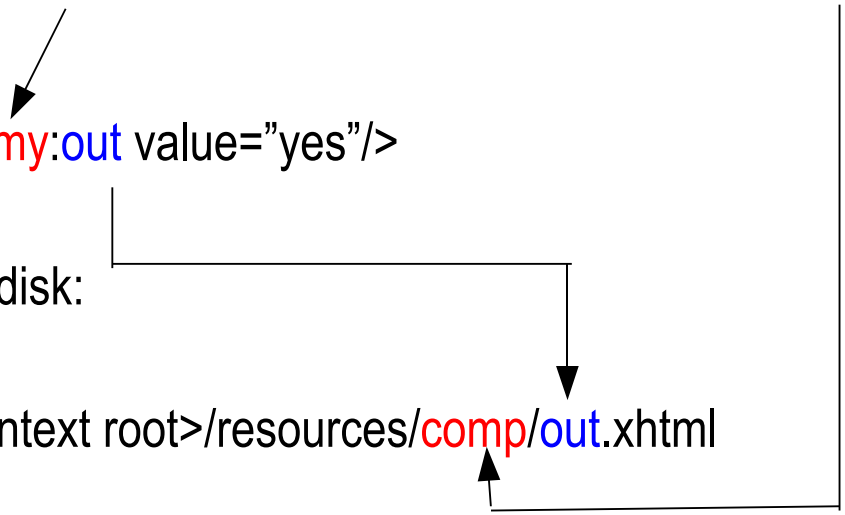
- Convention over configuration
  - Resources dir; Library name is directory name;
  - Tag name is file name

```
<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:my="http://java.sun.com/jsf/composite/comp">

     <my:out value="yes"/>

   On disk:

   <context root>/resources/comp/out.xhtml
```



Composite Components
What's Inside The Black Box?

- Interface
  - The usage contract
  - Everything page author needs to know to use component
- Implementation
  - Markup used to create component

Composite Components

<context-root>resources/ezcomp/LoginPanel.xhtml

```
<html... xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:cc="http://java.sun.com/jsf/composite">
...
  <cc:interface>
     <cc:attribute name=”userVal” required=”true” ”/>
     <cc:attribute name=”passValue” required=”true” />
  </cc:interface>
  <cc:implementation>
    <div> Username: <h:inputText id=”userId”” value=”#{cc.attrs.userVal}”/> </div>
    <div>Password:  <h:inputSecret id=”passId” value=”#{cc.attrs.passVal}” /></div>
    <div> <h:commandButton value=”Login” id=”loginButton” /> </div>
  </cc:implementation>
....
```

Composite Components

<context-root>resources/ezcomp/LoginPanel.xhtml

```
<html... xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:cc="http://java.sun.com/jsf/composite">
...
  <cc:interface>
     <cc:attribute name=”userVal” required=”true” ”/>
     <cc:attribute name=”passValue” required=”true” />
     <cc:actionSource name="loginAction” targets=”loginButton” />
  </cc:interface>
  <cc:implementation>
    <div> Username: <h:inputText id=”userId”” value=”#{cc.attrs.userVal}”/> </div>
    <div>Password:  <h:inputSecret id=”passId” value=”#{cc.attrs.passVal}” /></div>
    <div> <h:commandButton value=”Login” id=”loginButton” /> </div>
  </cc:implementation>
....
```

Composite Components
“Using Page”

```
    xmlns:ez="http://java.sun.com/jsf/composite/ezcomp">
    <html...xmlns:ui="http://java.sun.com/jsf/facelets"
…
      <h:form>
        <div id="compositeComponent" class="grayBox"
          style="border: 1px solid #090;">
          <ez:loginPanel >
            <f:actionListener for=”loginAction” binding=”#{bean.action}” />
          </ez:loginPanel>
        </div>
       <p><h:commandButton value="reload" /></p>
      </h:form>
```

Enhancing JSF 2.0 Components

- Other prominent JSF 2.0 additions:
  - Ajax
  - Behaviors
- Work well with composite components

Enhancing JSF 2.0 Components
Ajax

- JavaScript api in the specification
  - jsf.ajax.request .....
  - Useful for making JSF Ajax calls from JavaScript
  - Control component processing on the server and component rendering at the client

Enhancing JSF 2.0 Components
Ajax : Declarative Solution : <f:ajax />

- Serves two roles depending on placement:
  - Nested within single component
    - “ajaxifies” that component
  - Wrapping approach – tag is placed around a group of components
    - “ajaxifies” all components that support the “events” attribute

Enhancing JSF 2.0 Components

Ajax : Declarative Solution : <f:ajax />

```
<h:commandButton ...>
   <f:ajax event="action"/>
</h:commandButton>
```

```
<h:commandButton...>
   <f:ajax />
</h:commandButton>
```

Do exactly the same thing since "action" is the default event for commandButton components.

```
<h:commandButton...>
 <f:ajax event="mouseover"/>
</h:commandButton>
```

→ JavaScript events supported too.

Enhancing JSF 2.0 Components

Ajax : Declarative Solution : <f:ajax /> Regions

```
<f:ajax>
  <h:panelGrid>
    <h:inputText id="text1"/>
    <h:commandButton id="button1" />
  </h:panelGrid>
</f:ajax>
```

Ajax applied to “text1” and “button1”.

Ajax not applied to panelGrid : no default event associated with it.

Enhancing JSF 2.0 Components
Adding Ajax

- Method 1: <f:ajax/> in composite component implementation

```
<cc:interface...>
…
</cc:interface>
<cc:implementation...>
…
   <h:commandButton id=”button1” value=”login”>
      <f:ajax />
   </h:commandButton>
…
</cc:implementation>
```

Enhancing JSF 2.0 Components
Adding Ajax

- Method 2: JSF Ajax api call in script

```
<cc:implementation...>
…
   <h:commandButton id=”button1” value=”login”
        onclick=”return myscript.login('#{cc.clientId'), event);”/>
   </h:commandButton>
…
</cc:implementation>
script.js:
   myscript.login=function login(componentId, event) {
      jsf.ajax.request(document.getElementById(subButton),
       event);
```

Enhancing JSF 2.0 Components
Behaviors

- Additional functionality applied to components
- Ajax is a behavior (AjaxBehavior)
- Not just about Ajax
  - Tool tips, client validation, …

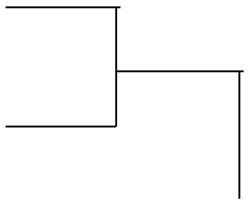
Enhancing JSF 2.0 Components
Behaviors : Client Behavior(s)

- New type of attached object
- Attached by “event”
- Contributes script content for rendering
  - AjaxBehavior is a Client Behavior that formulates the jsf.ajax.request script content for rendering
- All Html standard components implement ClientBehaviorHolder interface

Enhancing JSF 2.0 Components
Adding Behaviors

- Similar to adding Ajax...

```
<cc:interface..>...</cc:interface>
<cc:implementation>
  <h:graphicImage value...>
    <be:tip event=”mouseover”/>
    <be:tip event=”mouseout/>
  </h:graphicImage>
```



be:tip is custom behavior

Other Enhancements

- Navigation
  - Implicit Navigation : navigation logic in page (instead of faces-config xml)
  - Conditional Navigation : more complex navigation rules
- Exceptions
  - Central Exception Handler subscribes to exception events
  - Can be decorated
- EL : enhancements for component processing:
  - #{component} #{compositeComponent}
  - #{component.clientId}

Other Enhancements

- Validation
  - Integration with JSR 303: Bean Validation
  - <f:validateRequired> <f:validateRegexp>
- New Scopes
  - “Conversation” : from Contexts and Dependency Injection (JSR 299)
  - “Flash”
    - useful for storing data for use on the “next” request
    - Single view transition
  - “View”
    - Longer than request ; shorter than session
    - State preserved until interaction with current view complete

Other Enhancements

- Annotations
  - Managed Bean
  - Component (and associated artifacts)
    - @FacesComponent, @FacesRenderer
    - @FacesConverter, @FacesValidator
    - @FacesBehavior

Summary

- JSF 2.0 : Big improvements address community needs
  - Easier component development
  - Ajax : Dynamic Components
  - Less configuration
  - Performance (State Saving)

Resources

Java EE 6 and GlassFish v3

- Java EE 6 Home
  java.sun.com/javaee
- Java EE 6 Downloads
  java.sun.com/javaee/downloads
- Upcoming Training
  java.sun.com/javaee/support/training

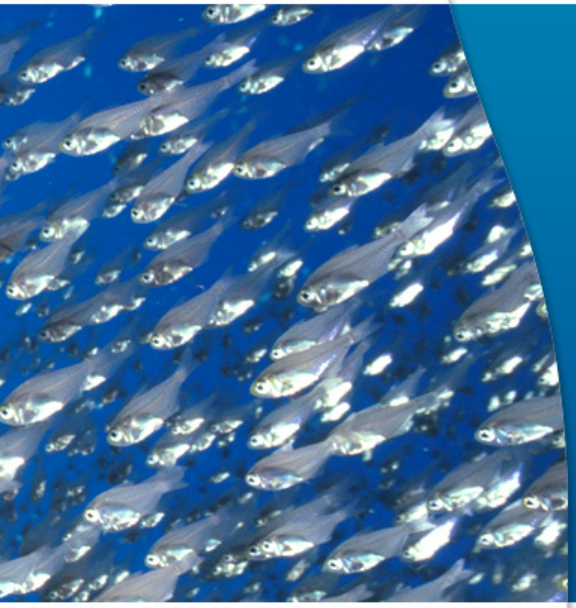
Java EE 6

- Sun GlassFish Enterprise Server v3 Home
  www.sun.com/glassfishv3
- Community Page
  glassfish.org
- The Aquarium Blog
  blogs.sun.com/theaquarium
- White Papers/Webinars
  http://www.sun.com/glassfish/resources

GlassFish

New JSF 2.0 Training and Certification

- Brand new training course and certification for JSF 2.0
  - Developing Web Applications using JSF Technologies (SL-340-EE6) – 3 Days
- Includes coverage of:
  - Using JSF within a Web Container
  - Designing views using JSF, EL, and custom components using Facelets
  - Integrating external resources such as JPA within Web Applications
  - Testing, packaging and deploying applications and more
- Register your interest!
  - https://dct.sun.com/dct/forms/reg_us_1611_480_0.jsp



Q&A