JSF 2 and beyond: Keeping progress coming

Andy Schwartz - *Oracle Corporation*

Dan Allen - *Red Hat, Inc.*

See how far JSF 2 has come,

explore the community's role and

take a glimpse at JSF 2.next

Join in!

Twitter hashtag: **#jsf2next**

Andy Schwartz

http://andyschwartz.wordpress.com

- Software engineer at **Oracle Corporation**
- Architect on ADF Faces project team
- Member of the JSR-314 (JSF 2) Expert Group

Dan Allen

http://mojavelinux.com

- Senior Software Engineer at **Red Hat, Inc**
- Author of Seam in Action
- Seam and Weld project member
- Member of the JSR-314 (JSF 2) Expert Group

Many faces of JSF 2


Ed Burns


Roger Kitain


Andy Schwartz


Jim Driscoll


Matthias Wessendorf


Martin Marinschek


David Geary


Ted Goddard


Craig McClanahan


Jacob Hookom


Kito Mann


Alexander Jesse


Alexandr Smirnov


Keith Donald


Nick Belaevski


Çağata Çivici


Ryan Lubke


Dan Allen


Lincoln Baxter


Stan Silvert


Emmanuel Bernard


Pete Muir


Max Katz


Amy Fowler


Neil Griffin


Jason Lee


Roger Keays


Hazem Saleh


Dennis Byrne


Gavin King


Ken Paulsen


Rick Hightower


Adam Winer


Joe Ottinger


Yara Senger


Jeremy Grelle


Imre Oßwald

Topic areas

- View
  - Facelets and VDL
  - Ajax & behaviors
  - State saving
- Controller
  - Bookmarkability
  - Navigation
  - Resource loading
- Model
  - Components and EL
  - Validation
  - Error handling
- Pain relief
- Community

View declaration

Facelets, View Declaration Language API

The problem

JSP pain points

- Content vs component tree creation
- Grunge
  - Tag class
  - Tag library
- Mixing presentation with logic
- Translation/compilation
- Stateful tags

The solution

Facelets

(Thanks, Jacob!)

Breaking free with Facelets

- View definition optimized for JSF
- XHTML + tags (no scriptlets)
- Default, stateless tag handling
- Simplified tag library configuration
- No more translation/compilation
- Templating



The problem revisited

But, Facelets isn’t standard :(

The solution revisited

Now it *is*!

The solution 2.0

- JSF 2.0 includes Facelets in the spec
- Same features, some enhancements
- Facelets is now preferred over JSP
  - Most new functionality not available in JSP
- Also new: View Declaration Language APIs



Facelets and VDL: JSF2.next

- Facelets XHTML vs. XML
- XSD for Facelets
- Facelets/JSP compatibility
- Whitespace handling
- Are Facelets APIs complete?
- Are VDL APIs complete?

Component development

Java components, composite components

The problem

Component development is hard!

The problem in detail

- Too many artifacts
  - UIComponent class
  - Renderer class
  - Tag class
  - .tld
  - lots of faces-config.xml
- *Ouch*!

The solution: Take 1

Simplify Java component development

The solution: Take 1

- Annotations replace faces-config.xml
- Default handlers replace tag classes
- Facelets taglib.xml replaces tld grunge
- Simplified state saving
  - More on this in a bit...
- Better, but good enough?



The solution: Take 2

Composite components!

Composite components

- Easy component creation (via Facelets)
  - It's not just for JSF gurus any more
- Defined using a single Facelets file
- No external configuration
- Conventions define tag namespace/name
- No Java code required

Composite component definition

/resources/foo/greeting.xhtml

```
<composite:interface>
   <composite:attribute name="name" default="World"/>
</composite:interface>

<composite:implementation>
   Hello, #{cc.attrs.name}!
</composite:implementation>
```

Composite component usage

/hello.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:foo="http://java.sun.com/jsf/composite/foo">
  <body>
    <foo:greeting name="Devoxx"/>
  </body>
</html>
```

Composite components

- Definitions live in web root or JAR
- Optional Java/Groovy backing file
- Optional .properties file
- Optional supporting resources
- Attach listeners, converters, validators, behaviors

Component development: JSF2.next

- Possible to simplify further?
- Hybrid tag libraries (composites + Java)
- Resource location (WEB-INF/resources)
- Java/Groovy backing class naming

Ajax

jsf.ajax.request(), <f:ajax>, Ajax Java APIs, and tree visiting

The problem

Tomahawk	Tobago	Trinidad	ICEfaces	RCFaces	Netadvantage	WebGalileoFaces	QuipuKit	BluePrints	Woodstock	JBoss RichFaces	Oracle ADF	Simplica	PrimeFaces	Oper
JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF
URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL
URL	URL	URL	URL	URL	URL	URL	URL		URL	URL	URL	URL	URL	
⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕		⊕	⊕		⊕	⊕	
⊘	⊖	⊕	⊕	http://primefaces.prime.com.tr		⊘	⊕		⊖	⊕		⊕		
URL	URL	URL	URL	URL	URL	URL	URL		URL	URL		URL	URL	
		URL	URL		URL		URL			URL	URL	URL	URL	
⊕	⊕	⊕										⊖	⊕	
2	?	2										0		
81.300	17.800	46.700										630		
Tomahawk	Tobago	Trinidad									DF	Simplica	PrimeFaces	Oper
⊕	⊕ AJAX	⊕ AJAX										⊕ AJAX	⊕ AJAX	
⊕	⊕	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊖	⊕	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	
⊕	⊖	⊕ AJAX	⊖	⊖	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊖	⊖	⊖	⊕ AJAX			
⊖	⊖	⊖	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊖	⊖	⊖	⊖	⊕ AJAX	⊕ AJAX	⊕ AJAX		
⊕	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕	⊖	⊕	⊕ AJAX		⊕	⊕ AJAX	
⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊖	⊖	⊕ AJAX	⊕ AJAX			⊕	

JSF/Ajax Overload!

Where things went wrong

- Everyone has a solution
- No two solutions are compatible
- Sad application developers

The solution

Standard Ajax APIs

The solution in detail

- Start with a programmatic API
  - jsf.ajax.request()
- Add in some declarative support
  - <f:ajax>
- Don't forget about the server side
  - PartialViewContext
  - PartialResponseWriter
  - Behaviors

jsf.ajax.request()

- Java EE's *first* JavaScript API!
- Performs a partial page update
- Caller specifies execute/render ids
  - Or keywords: @all, @form, @this, @none
- jsf.ajax.request() takes care of the rest
- Supports notifications of events/errors

jsf.ajax.request()

```
<h:outputScript name="jsf.js" library="javax.faces"/>
...
<h:commandButton value="Do something Ajaxy"
   onclick="jsf.ajax.request(this, event, {render: 'out'}); return false;"/>
...
<h:outputText id="out" value="Update me!"/>
```

<f:ajax>

- Declarative mapping for jsf.ajax.request()
- Attach via nesting or wrapping

<f:ajax> nesting

```
<h:commandButton value="Do something Ajaxy">
   <f:ajax render="out"/>
</h:commandButton>
...
<h:outputText id="out" value="Update me!"/>
```

<f:ajax> wrapping

```
<f:ajax render="out"/>
  <h:commandButton value="Do something Ajaxy"/>
  <h:commandButton value="Do something else"/>
  <h:commandButton value="One more here"/>
</f:ajax>
...
<h:outputText id="out" value="Update me!"/>
```

<f:ajax> client events

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

Ajax: JSF2.next

- Fallback
- ID round-tripping
- Out-of-band/GET requests
- Event collapsing
- File upload



State saving

Partial state saving, state helper

The problem

State saving is nasty

State saving lunacy

```
public Object saveState(FacesContext ctx) {
  if (_values == null) {
    _values = new Object[10];
  }
  _values[0] = super.saveState(ctx);
  _values[1] = accesskey;
  _values[2] = alt;
  _values[3] = dir;
  _values[4] = disabled;
  _values[5] = image;
  _values[6] = label;
  _values[7] = lang;
  _values[8] = onblur;
  _values[9] = onchange;
  return _values;
}
```

```
public void restoreState(
    FacesContext ctx, Object _state) {
  _values = (Object[]) state;
  super.restoreState(ctx, _values[0]);
  this.accesskey = (java.lang.String) _values[1];
  this.alt = (java.lang.String) _values[2];
  this.dir = (java.lang.String) _values[3];
  this.disabled = (java.lang.Boolean) _values[4];
  this.image = (java.lang.String) _values[5];
  this.label = (java.lang.String) _values[6];
  this.lang = (java.lang.String) _values[7];
  this.onblur = (java.lang.String) _values[8];
  this.onchange = (java.lang.String) _values[9];
}
```

Another problem

State saving is expensive

State overhead

- State saving == component developer tax
  - Do I really need to implement saveState and restoreState?
- Full component tree state not small
  - Where do you want it? Session? Client?

The solution

Partial state saving for smaller state.

State helper utilites make happier component developers.

Partial state saving

- Why save the full component tree?
- Initial component tree is accessible
  - Just need to re-execute the tags
- Initial component tree isn't sufficient
- Also need any state deltas.

Partial state saving

- Build the component tree
- Lock it down (mark initial state)
- Subsequent modifications saved
- On restore, build component tree again
- Apply previously saved deltas
- No need to save *full* state!

State saving 2.0

- PartialStateHolder
  - StateHolder that can lock down state
- StateHelper
  - Manages state, tracks deltas
- No more custom saveState/restoreState
- Significantly smaller saved state!

State saving: JSF2.next

- Further optimizations?
- Better support for edge cases
  - Non-rexecution of tags after invoke app
- Target high scalability cases
  - Fully stateless?

Controller

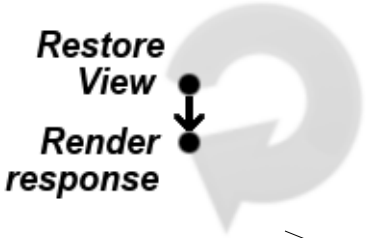
GET support, bookmarkable URLs,
navigation and redirects,
and resource loading

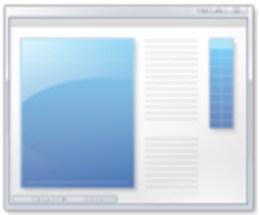
GET support

View metadata, view parameters, pre-render event listeners and bookmarkable URL components

Initial request lifecycle

http://acme.org/catalog.jsf?cat=electronics&page=3&layout=grid





Initial state

http://acme.org/catalog.jsf?cat=electronics&page=3&layout=grid

↓

view ID

↓

/catalog.xhtml

Initial state

http://acme.org/catalog.jsf?cat=electronics&page=3&layout=grid

↓

request parameters

↓

cat=electronics
page=3
layout=grid

Bean property mapping

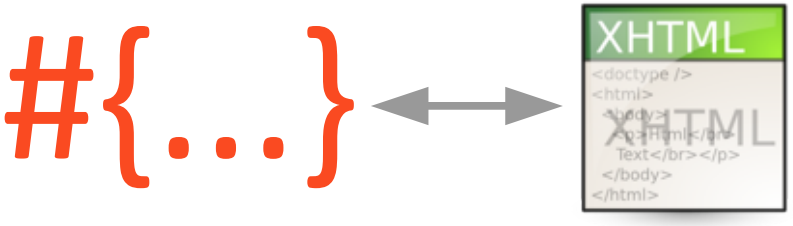
```
<managed-bean>
  ...
  <managed-bean-property>
    <property-name>category</property-name>
    <value>#{param['cat']}</value>
  </managed-bean-property>
</managed-bean>
```

Bean property mapping limitations

- Assignment occurs when bean is used
  - What if mapping differs based on current view?
- Implicit conversion only
  - What if property type is java.util.Date?
  - What about validation?
- What about a post-mapping listener?

Need more sophisticated, view-oriented mapping

View metadata



- Yet another XML schema? (YAXS!)
- Need elements for:
  - matching view ID(s)
  - describing EL binding
  - conversion
  - validation
  - post-mapping listener
  - …

Reuse the tree

View metadata facet

```
<f:view>
  <f:metadata>
    ...
  </f:metadata>
  ...
</f:view>
```

View metadata facet

- Built-in facet of UIViewRoot
  - Known place to find metadata
  - Can be built separate from tree
- Reuses UI component infrastructure
  - Metadata is described using UI components
  - Manifests as UIPanel component
  - Easy to extend

View metadata lifecycle

- Initial request is now a full postback
  - UI component tree only contains view metadata
  - Activated if view parameters are present
- A postback is just a postback
  - Metadata components == UI components

View parameter

UIViewParameter

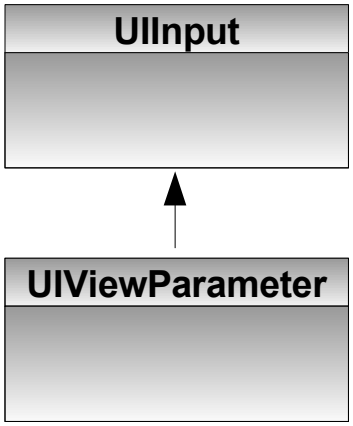
```
<f:view>
  <f:metadata>
    <f:viewParam name="cat" value="#{catalogBean.category}"/>
  </f:metadata>
  ...
</f:view>
```

View parameter w/ converter

UIViewParameter

```
<f:view>
  <f:metadata>
    <f:viewParam name="cat" value="#{catalogBean.category}">
      <f:converter converterId="com.acme.converter.Category"/>
    </f:viewParam>
  </f:metadata>
  ...
</f:view>
```

View parameter assignment



- name – request parameter name
- value – bean property described w/ EL
- Specialization of UIInput
  - Initial value transfered from request parameter
  - Submitted value stored in component state
  - Request parameter can override value on postback
- Foundation of bookmarkable URLs

View metadata templating

```
<f:view>
  <f:metadata>
    <ui:include src="/WEB-INF/metadata/catalog.xhtml"/>
    [ or ]
    <acme:catalogMetadata/>
  </f:metadata>
  ...
</f:view>
```

More powerful & flexible than a matching pattern

Post-processing

The values are set, now what?

Component system events

- Fine-grained event system in JSF 2
  - Publish/subscribe pattern (3 tiers)
- PostAddToViewEvent
  - After component is created (e.g., UIViewRoot)
- PreRenderViewEvent
  - Before component tree is rendered
  - ||: Lifecycle :|| if view ID is changed by listener

Post-mapping event listener

Declarative system event

```
<f:view>
  <f:metadata>
    ...
    <f:event type="preRenderView" listener="#{catalogBean.onRender}"/>
  </f:metadata>
  ...
</f:view>
```

No-args method or method that accepts `ComponentSystemEvent`

Hold the rendering!

```
public void onRender() {
  FacesContext ctx = FacesContext.getCurrentInstance();
  if (ctx.isValidationFailed() || !loadDataAttempt()) {
    ctx.getApplication().getNavigationHandler()
        .handleNavigation(ctx, null, "invalid");
  }
}
```

Force navigation if preconditions not met

View actions

Wouldn't it be nice if we had...?

```
<f:view>
  <f:metadata>
    ...
    <f:viewAction execute="#{catalogBean.onLoad}" onPostback="false"/>
  </f:metadata>
  ...
</f:view>
```

View actions

...followed by built-in navigation?

```
<navigation-rule>
  <from-view-id>/catalog.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{catalogBean.onLoad}</from-action>
    <from-outcome>failure</from-outcome>
    <to-view-id>/search.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

View actions vs PreRenderView

- It’s about timing
- PreRenderView
  - Executes before *rendering* component tree
- View action
  - Executes before *building* component tree
  - Why build it just to throw it away?

How about this URL?

http://acme.org/catalog/category/electronics

Pretty URLs proposal

```
<rewrite-rule>
  <rewrite-view-id>/catalog.xhtml</rewrite-view-id>
  <rewrite-case>
    <url-pattern>/catalog</url-pattern>
    <url-pattern>/catalog/category/{cat}</url-pattern>
    <url-pattern>/catalog/category/{cat}/{page}</url-pattern>
  </rewrite-case>
</rewrite-rule>
```

View parameter mappings

Producing

UIOutputLink

```
<h:outputLink value="/home.jsf">Home</h:outputLink>
```

- Basic hyperlink-generating component
- Not aware of:
  - context path,
  - view ID extension → servlet mapping, or
  - navigation rules
- Manual query string creation
  - Does at least support <f:param>

UIOutcomeTarget

```
<h:link outcome="home" value="Home"/>
```

- *Intelligent* hyperlink-generating component
- Aware of:
  - context path,
  - uses navigation handler to derive view ID, and
  - can encode view parameters into query string
- Parameter overrides
  - Can use <f:param> to set parameter explicitly

Generating bookmarkable links

```
<h:link value="Previous" includeViewParams="true">
  <f:param name="page" value="#{catalogBean.previousPage}"/>
</h:link>
```

```
<a href="http://acme.org/catalog.jsf?q=portable+hole&page=3">Previous</a>
```

/catalog.xhtml

```
<f:metadata>
  <f:viewParam name="q" value="#{catalogBean.query}"/>
  <f:viewParam name="page" value="#{catalogBean.page}"/>
</f:metadata>
```

Navigation

Implicit, conditional and preemptive navigation, queryable navigation rules and redirect parameters

Implicit navigation

- Fall-through case catering to prototypes
- Logical outcome => view ID
- Applies to:
  - return value of action method,
  - action of UICommand (<h:commandButton>),
  - outcome of UIOutcomeTarget (<h:link>), or
  - NavigationHandler.handleNavigation() method

A navigation shorthand

```
<h:commandButton action="#{productBean.save}" value="Save"/>
```

```
public String save() {
   // perform save logic, then...
  return "/catalog.xhtml";
}
```

A navigation short(er)hand

```
<h:commandButton action="#{productBean.save}" value="Save"/>
```

```
public String save() {
   // perform save logic, then...
  return "catalog";
}
```

Relative to current path and view ID

Can link to navigation case later

Logical outcomes aren’t logical

- Leak into business logic
- Reuse is difficult
- Void methods don’t work

Conditional navigation

- Navigation case matched based on state
- Promotes loose coupling
  - Action methods don't return "logical outcome"

Web tier ▪ ▪ ▪ ▪ *Transactional tier*

- Can reduce number of navigation cases
- Navigation cases not skipped on void outcome

A conditional case

```
<navigation-case>
  <from-action>#{registration.register}</from-action>
  <if>#{currentUser.registered}</if>
  <to-view-id>/account.xhtml</to-view-id>
  <redirect include-view-params="true"/>
</navigation-case>
```

Matching a void outcome

```
<navigation-case>
  <from-action>#{catalog.search}</from-action>
  <if>#{true}</if>
  <to-view-id>/results.xhtml</to-view-id>
</navigation-case>
```

Preemptive navigation

- Evaluated at render time
- Outcome translated into bookmarkable URL
- Key elements:
  - UIOutcomeTarget (<h:link>, <h:button>)
  - implicit navigation
  - view parameters

Bookmarkable link

```
<h:link outcome="product" value="View">
  <f:param name="id" value="#{product.id}"/>
</h:link>
```

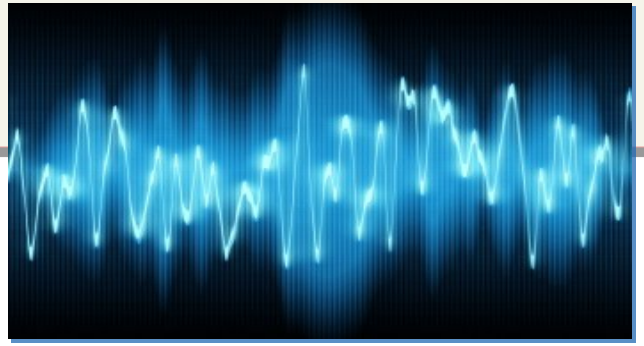
```
<a href="/product.jsf?id=3">View</a>
```

Redirect parameters

- No support in JSF 1.x
  - Made redirect after POST difficult
  - Limited usefulness of declarative navigation
- Two solutions in JSF 2
  - Explicit redirect parameters
  - View parameters

Redirect after POST the **hard** way

```
FacesContext ctx = FacesContext.getCurrentInstance();
ExternalContext extCtx = ctx.getExternalContext();
String url = ctx.getApplication().getViewHandler()
   .getActionURL(ctx, "/product.xhtml") + "?id=" + getProductId();
try {
   extCtx.redirect(extCtx.encodeActionURL(url));
} catch (IOException ioe) {
   throw new FacesException(ioe);
}
```



Redirect after POST the *easier* way

```
<navigation-case>
  <from-action>#{productBean.save}</from-action>
  <if>#{productBean.id != null}</if>
  <to-view-id>/product.xhtml</to-view-id>
  <redirect>
    <view-param>
      <name>id</name>
      <value>#{productBean.id}</value>
    </view-param>
  </redirect>
</navigation-case>
```

Redirect after POST the *best* way

```
<navigation-case>
  <from-action>#{productBean.save}</from-action>
  <if>#{productBean.id != null}</if>
  <to-view-id>/product.xhtml</to-view-id>
  <redirect include-view-params="true"/>
</navigation-case>
```

Navigation: JSF 2.next

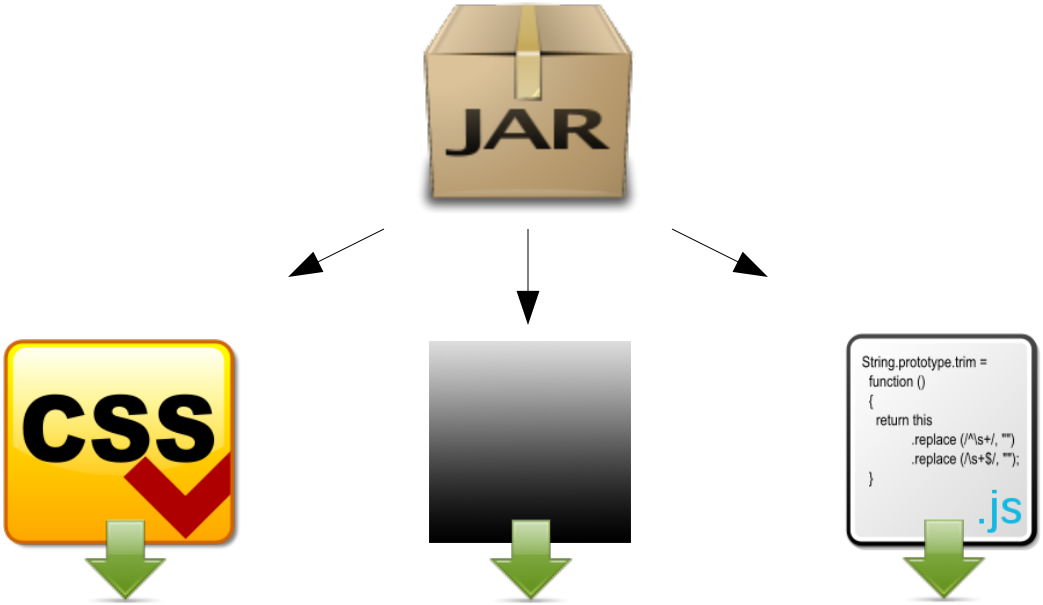
- Include view parameters automatically
- <if>#{true}</if> to match void outcome is *ugly*
- Navigation rules are XML hell
  - A more conscise DSL?
  - Java-based configuration?
- Other ideas?

Resource handling

Native resource handling,

packaging and resource relocation

No more ”bonus” servlet!



Resource handling

- Load resources out of web root or JAR
- Associate resources with UIComponent
  - Resources loaded if component is rendered
- Resource loading API
- Localization

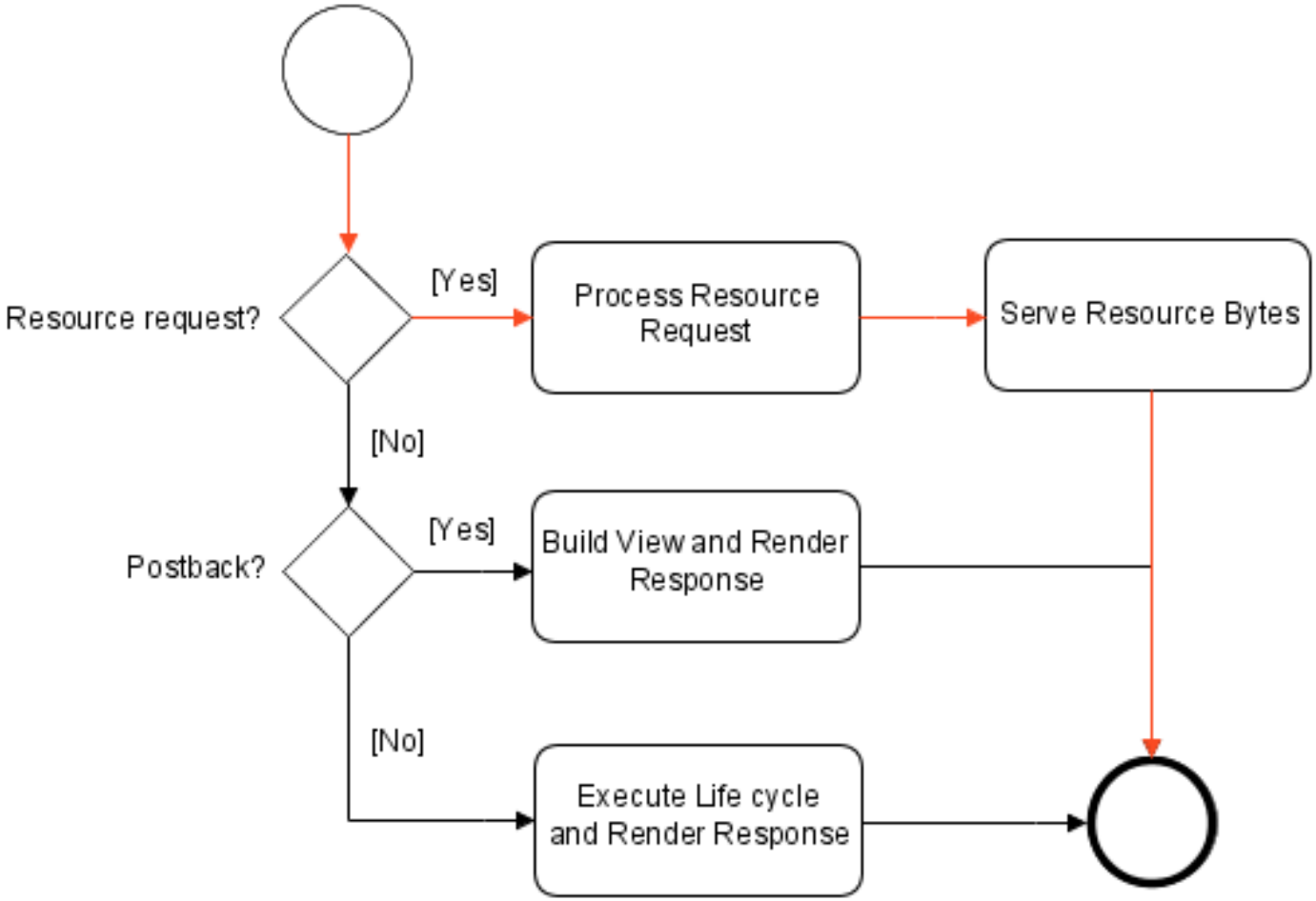
Declarative component resources

```
@ResourceDependency(
  name = "jsf.js", library = "javax.faces", target = "head")
public class MyComponent extends UIOutput { ... }
```

A resource at a glance

- Structure
  - Name
  - Library
  - Locale
  - Version
- Packaging
  - Web root
    - /resources
  - Classpath
    - META-INF/resources

A third request processing scenario



Resolving a resource

localePrefix/libraryName/libraryVersion/resourceName/resourceVersion

Path segments in gray are optional

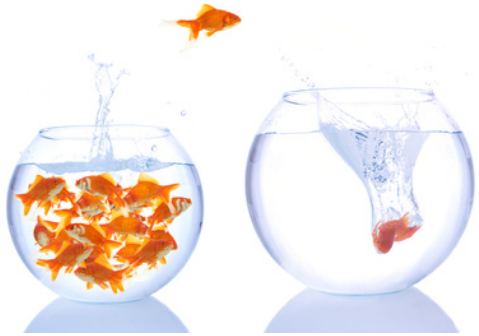
- Served from web root

```
<h:graphicImage name="visa.png"/>
```

- Served from classpath of creditcards.jar

```
<h:graphicImage name="visa.png" library="creditcards"/>
<h:graphicImage value="#{resources['creditcards:visa.png']}"/>
```

Resource relocation



- Resources can target section of document
- Essential for templating

```
<html>
  <h:head>
    <title>Resource Relocation Example</title>
  </h:head>
  <h:body>
    <h:outputScript name="script.js" target="head"/>
  </h:body>
</html>
```

Model

Java EE 6 component model,
Bean Validation, error handling
and resource loading

Java EE 6: Newcomers

- Managed Beans (part of JSR-316)
- Contexts and Dependency Injection - JSR-299
- Bean Validation - JSR-303
- JAX-RS (RESTful Web Services) - JSR-311
- Web Profile

Web profile contents

Persistence

- JPA 2.0
- JTA

Component model

- EJB 3.1 (Lite)
- Bean Validation
- **CDI** (formerly Web Beans)

Presentation

- JSF 2.0
- Servlet 3.0

JSR-299: Essential ingredients

- Beans types
- Qualifier annotations
- Scope
- Alternatives
- An EL name (optional)
- Interceptors and decorators
- Bean implementation

JSF managed bean replacement

```
@Named("hello")
public class Hello {
  private String name; // getters and setters not shown
  public void sayHello() {
    System.out.println("Hello, " + name);
  }
}
```

`@Named` makes bean available to EL

```
@Named
public class Hello {
  ...
}
```

Named can be defaulted to simple name of class

JSF view

Invoking a bean via EL

```
<h:inputText value="#{hello.name}"/>
<h:commandButton value="Say Hello" action="#{hello.sayHello}"/>
```

Scopes and contexts

- Built-in scopes:
  - Any servlet request: @ApplicationScoped, @RequestScoped, @SessionScoped
  - JSF requests - @ConversationScoped
  - Dependent scope (Default): @Dependent
- Custom scopes
  - Define scope type annotation (e.g., @FlashScoped)
  - Context impl defines where bean is stored

Parameterized EL methods

- Syntax similar to Java method calls
- Method arguments are EL expressions
- Arguments resolved at different times:
  - Value expression: at render time
  - Method expression: when event is fired

```
<h:commandButton action="#{hello.sayHello('Devoxx')}" .../>
<h:commandButton action="#{hello.sayHello(currentConference)}" .../>
```

Validation

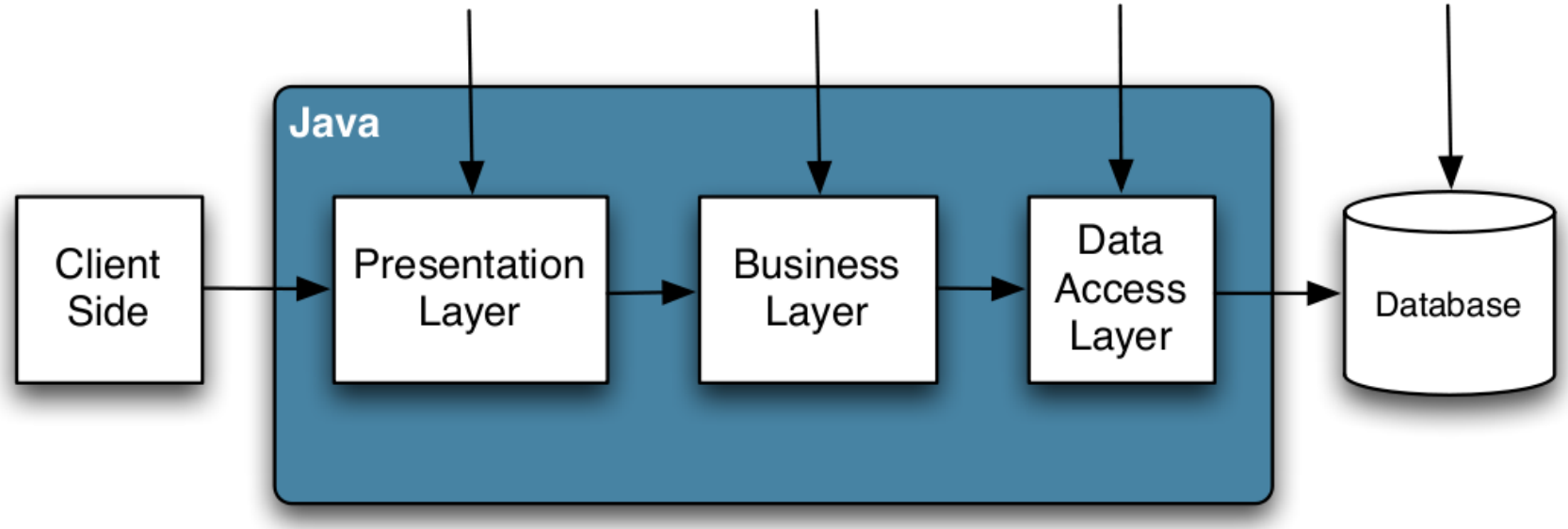
Bean Validation integration,
validating empty fields and multi-field
validation with post-validate events

Constraints in the enterprise

One model...

User
String username String email

...validated across multiple layers



Bean Validation (JSR-303)

- Constrain once, validate anywhere
- Centrally define constraints in model class
  - Constraints described using annotations
- JSF integration
  - Enforce constraints in presentation layer
  - Replaces existing JSF validators
  - Zero configuration!

Defining constraints on the model

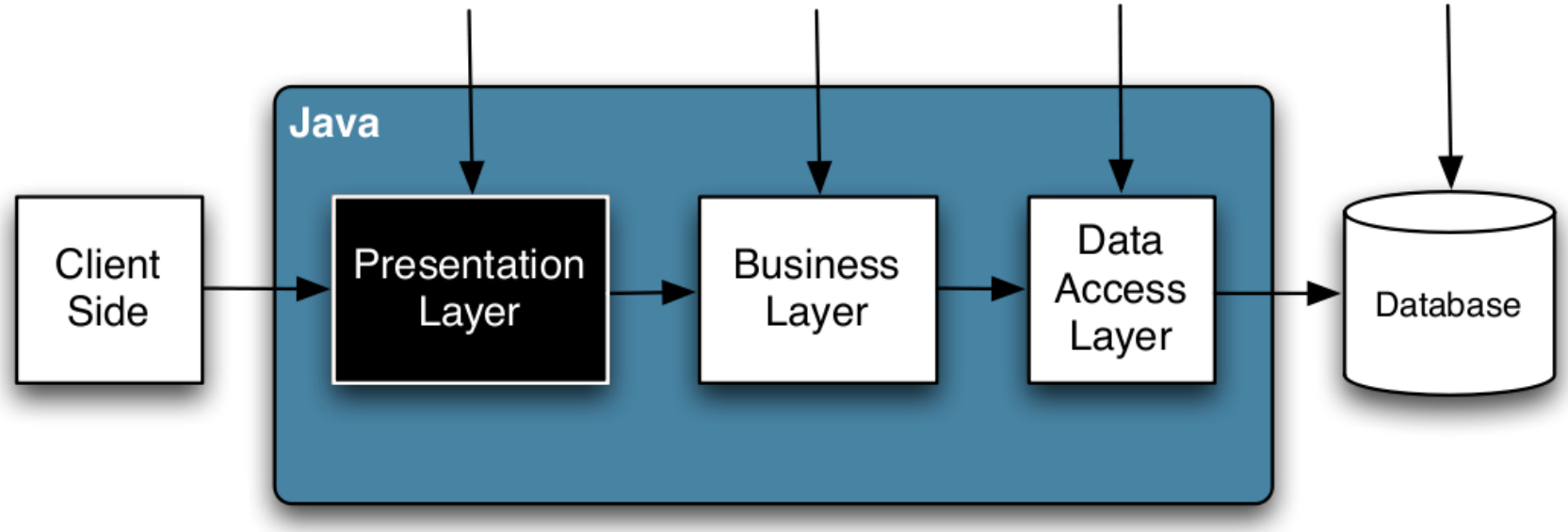
```
public class User {
  ...
  @NotNull @Size(min = 3, max = 25)
  public String getUsername() { return username; }

  @NotNull @Email
  public String getEmail() { return email; }
}
```

Constraints in JSF

One model...

User
String username String email

...validated across multiple layers



Enforcing constraints in the UI

```
<h:inputText id="username" value="#{user.username}"/>
```

```
<h:inputText id="email" value="#{user.email}"/>
```

Zeroconf!

Constraining partially

```
<h:inputText id="username" value="#{user.username}">
  <f:validateBean disabled="true"/>
</h:inputText>
```

```
<f:validateBean validationGroups="com.acme.BareMinimum">
  <h:inputText id="email" value="#{user.email}"/>
</:validateBean>
```

The case of the empty field

- Validation skipped if value is:
  - null
  - a zero-length string
- *Unless...*
  - Bean Validation is present or
  -

```
<context-param>
  <param-name>javax.faces.VALIDATE_EMPTY_FIELDS</param-name>
  <param-value>true</param-value>
</context-param>
```

Do you mean null?

- *Problem*: user can’t enter null in text field
- *Side-effect*: inadvertant database updates
- *Solution*: interpret empty strings as null
  -
```
<context-param>
  <param-name>
    javax.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL
  </param-name>
  <param-value>true</param-value>
</context-param>
```

Multi-field validation

- A tougher problem than it seems
- Two approaches:

Before model update

- Compare UIInput values
- PostValidateEvent

After model update

- Validate populated model
- Bean Validation

Listening for post validate

```
<h:form>
  <f:event type="postValidate" listener="#{minMax.validate}"/>
  <h:inputText id="min" value="#{bean.min}"
    binding="#{minMax.minInput}"/>
  <h:inputText id="max" value="#{bean.max}"
    binding="#{minMax.maxInput}"/>
  <h:commandButton value="Submit"/>
</h:form>
```

Validating across fields

```
@Inject FacesContext ctx;
private UIInput minInput, maxInput; // accessors hidden
public void validate() {
  if (ctx.isValidationFailed()) { return; }
  if ((Integer) maxInput.getValue() < (Integer) minInput.getValue()) {
    ctx.addMessage(maxInput.getClientId(ctx),
      new FacesMessage("cannot be less than min value"));
    ctx.validationFailed();
    ctx.renderResponse();
  }
}
```

Validation JSF.next

- What about postModelUpdate?
- Adding FacesMessages is tedious
- Graph Validation (Bean Validation on object)

Error handling

Exception handlers, exception events, servlet errors and the default error page

The good news

No more swallowed exceptions!

The bad news

You're still going to get exceptions

Exception handler

- Hub for handling *unexpected* exceptions
- When exception is thrown:
  - ExceptionQueuedEvent is published
  - Exception handler queues exception
- After each phase:
  - Exception handler unwraps first exception, rethrows as FacesException

Bubbling over in production

- Exceptions → servlet error handler (web.xml)
  -
```
<error-page>
  <exception-type>com.acme.SecurityException</exception-type>
  <location>/accessDenied.jsf</location>
</error-page>
```
- Several problems:
  - Error page is outside of JSF life cycle
  - Error page must include servlet mapping
  - Context of request is left behind

Declarative error handling *in JSF*

Wouldn't it be nice if we had...?

```
<exception class="javax.persistence.EntityNotFoundException">
  <redirect view-id="/error/404.xhtml">
    <message severity="warn">Record not found</message>
  </redirect>
</exception>
```

Default error page

An Error Occurred:

Error Parsing /index.xhtml: Error Traced[line: 4] The prefix "h" for element "h:head" is not bound.

- Stack Trace

```
javax.faces.view.facelets.FaceletException: Error Parsing /index.xhtml: Error Traced[line: 4] The prefix "h" for element "h:head" is not bound.
        at com.sun.faces.facelets.compiler.SAXCompiler.doCompile(SAXCompiler.java:390)
        at com.sun.faces.facelets.compiler.SAXCompiler.doMetadataCompile(SAXCompiler.java:373)
        at com.sun.faces.facelets.compiler.Compiler.metadataCompile(Compiler.java:122)
        at com.sun.faces.facelets.impl.DefaultFaceletFactory.createMetadataFacelet(DefaultFaceletFactory.java:325)
        at com.sun.faces.facelets.impl.DefaultFaceletFactory.getMetadataFacelet(DefaultFaceletFactory.java:214)
        at com.sun.faces.facelets.impl.DefaultFaceletFactory.getMetadataFacelet(DefaultFaceletFactory.java:147)
        at com.sun.faces.application.view.ViewMetadataImpl.createMetadataView(ViewMetadataImpl.java:102)
        at com.sun.faces.lifecycle.RestoreViewPhase.execute(RestoreViewPhase.java:239)
        at com.sun.faces.lifecycle.Phase.doPhase(Phase.java:97)
        at com.sun.faces.lifecycle.RestoreViewPhase.doPhase(RestoreViewPhase.java:110)
        at com.sun.faces.lifecycle.LifecycleImpl.execute(LifecycleImpl.java:118)
        at javax.faces.webapp.FacesServlet.service(FacesServlet.java:310)
        at org.mortbay.jetty.servlet.ServletHolder.handle(ServletHolder.java:511)
        at org.mortbay.jetty.servlet.ServletHandler.handle(ServletHandler.java:390)
        at org.mortbay.jetty.security.SecurityHandler.handle(SecurityHandler.java:216)
        at org.mortbay.jetty.servlet.SessionHandler.handle(SessionHandler.java:182)
        at org.mortbay.jetty.handler.ContextHandler.handle(ContextHandler.java:765)
        at org.mortbay.jetty.webapp.WebAppContext.handle(WebAppContext.java:418)
        at org.mortbay.jetty.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:230)
        at org.mortbay.jetty.handler.HandlerCollection.handle(HandlerCollection.java:114)
        at org.mortbay.jetty.handler.HandlerWrapper.handle(HandlerWrapper.java:152)
        at org.mortbay.jetty.Server.handle(Server.java:326)
        at org.mortbay.jetty.HttpConnection.handleRequest(HttpConnection.java:536)
        at org.mortbay.jetty.HttpConnection$RequestHandler.headerComplete(HttpConnection.java:915)
        at org.mortbay.jetty.HttpParser.parseNext(HttpParser.java:539)
        at org.mortbay.jetty.HttpParser.parseAvailable(HttpParser.java:212)
        at org.mortbay.jetty.HttpConnection.handle(HttpConnection.java:405)
        at org.mortbay.io.nio.SelectChannelEndPoint.run(SelectChannelEndPoint.java:409)
        at org.mortbay.thread.QueuedThreadPool$PoolThread.run(QueuedThreadPool.java:582)
```

+ Component Tree

+ Scoped Variables

Nov 11, 2009 12:21:20 AM - Generated by Mojarra/Facelets

Ajax error handling

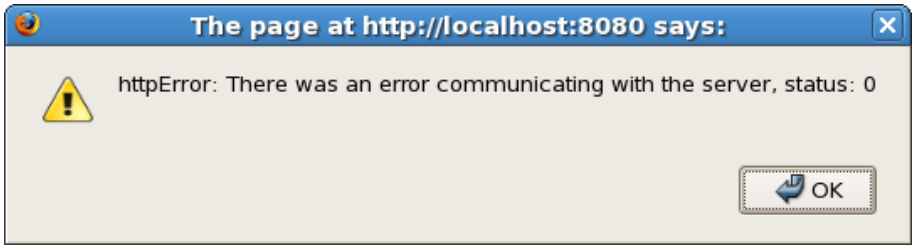
- JavaScript error callback for single request

```
<f:ajax ... onerror="handle_specific_error"/>
```

- Global JavaScript error listener

```
jsf.ajax.addOnError(handle_all_errors);
```

- Alert window fallback in development



Pain relief

Select items from collections,

validation failed flag, API improvements,

varStatus on ui:repeat, and more...

From collection to select items

```
<h:selectOneMenu value="#{product.category}">
  <f:selectItems value="#{catalogBean.categories}" var="cat"
    itemLabel="#{cat.name}" itemValue="#{cat}"
    noSelectionValue="#{catalogBean.defaultCatalog}"/>
</h:selectOneMenu>
```

```
@Named
public class CatalogBean {
   public List<Category> getCategories() {
      return ...;
  }
}
```

Minor improvements that add up

- Retrieve faces messages as java.util.List
  - FacesContext.getMessageList() (can filter by client ID)
- Preserve faces messages across redirect
  - ExternalContext.getFlash().setKeepMessages(true)
- Flag indicating whether validation failed
  - FacesContext.isValidationFailed()
- ActionEvent optional for action listeners
- Deterministic ordering of descriptors

Pain relief: JSF 2.next

- UIData components
  - java.util.Collection
  - varStatus
  - row state
- Standard components
  - h:inputDate
  - Separate spec?
- Facelets from JAR
- EL
  - Static methods
  - Enum support
- Rendered attribute
- Generated ids
- Container injection

Community

JSR-314-OPEN mailinglist,

javaserverfaces-spec-public project,

JCP.org and ***you*!**

Steps towards openness

- *Semi*-public mailinglist – JSR-314-OPEN
  - http://archives.java.sun.com/jsr-314-open.html
  - Free registration required to view
  - Must be EG member to post
- Public issue tracker – java.net project
  - https://javaserverfaces-spec-public.dev.java.net
  - No registration required to view
  - Free java.net account required to edit

Next steps

- Anonymous read access to JSR-314-OPEN
  - Allow community to follow along
  - Make sharing links easier
  - Indexable by search engines
- Non-EG member invites to JSR-314-OPEN
  - Prime candidates – implementation team members
- Read-write jsr-314-public forum @ jcp.org

JSF community home page

http://www.javaserverfaces.org

- Root node of the JSF ecosystem
  - Specification and API docs
  - Mailinglists and forums
  - Issue tracker
  - FAQs and guides
  - Implementations, component libraries

Summary

- JSF 2 is a drastic improvement
- Embraced de-facto community standards
- JSR-314 seeks to be role model for openness
- Still lots of room for innovation in #jsf2next
- *You* can be part of the process!

Dive into JSF 2

Learn

http://tinyurl.com/jsf2new

http://tinyurl.com/jsf2devworks

http://tinyurl.com/jsf2dzone

http://tinyurl.com/jsf2driscoll

http://tinyurl.com/jsf2ryan

Try

http://tinyurl.com/jsf2ri

http://tinyurl.com/jsf2issue

Get involved

http://tinyurl.com/jsr-314-public