Brian Leathem, JBoss by Red Hat

Realize the full potential of JSF with Seam Faces and CDI

What's this all about?

Seam Faces

Seam 3

JSF 2 / CDI integration is not as thorough is it could have been.

How does Seam Faces solve these problems?

How are these solutions implemented in Seam Faces?

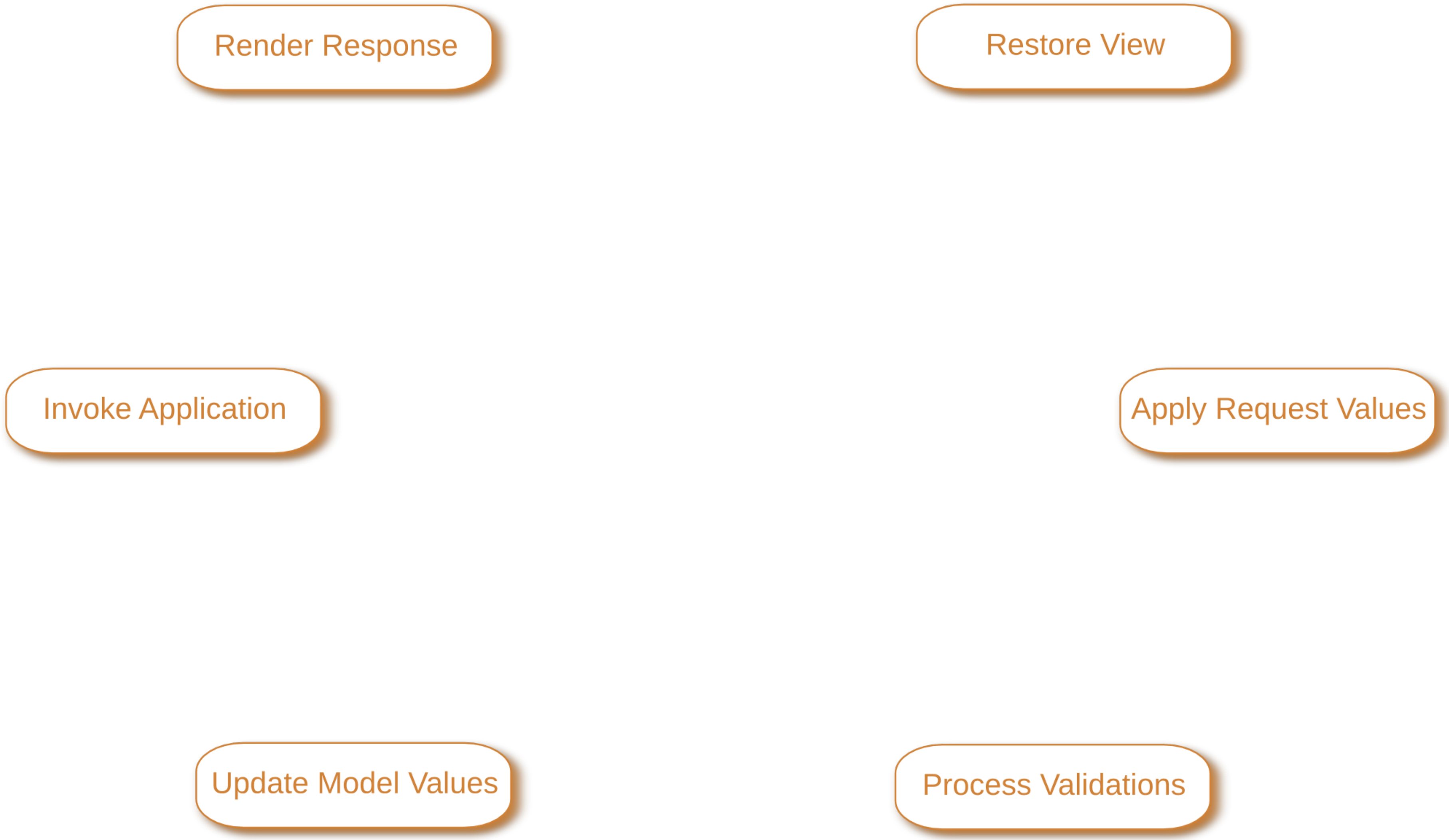
Who am I?

Brian Leathem

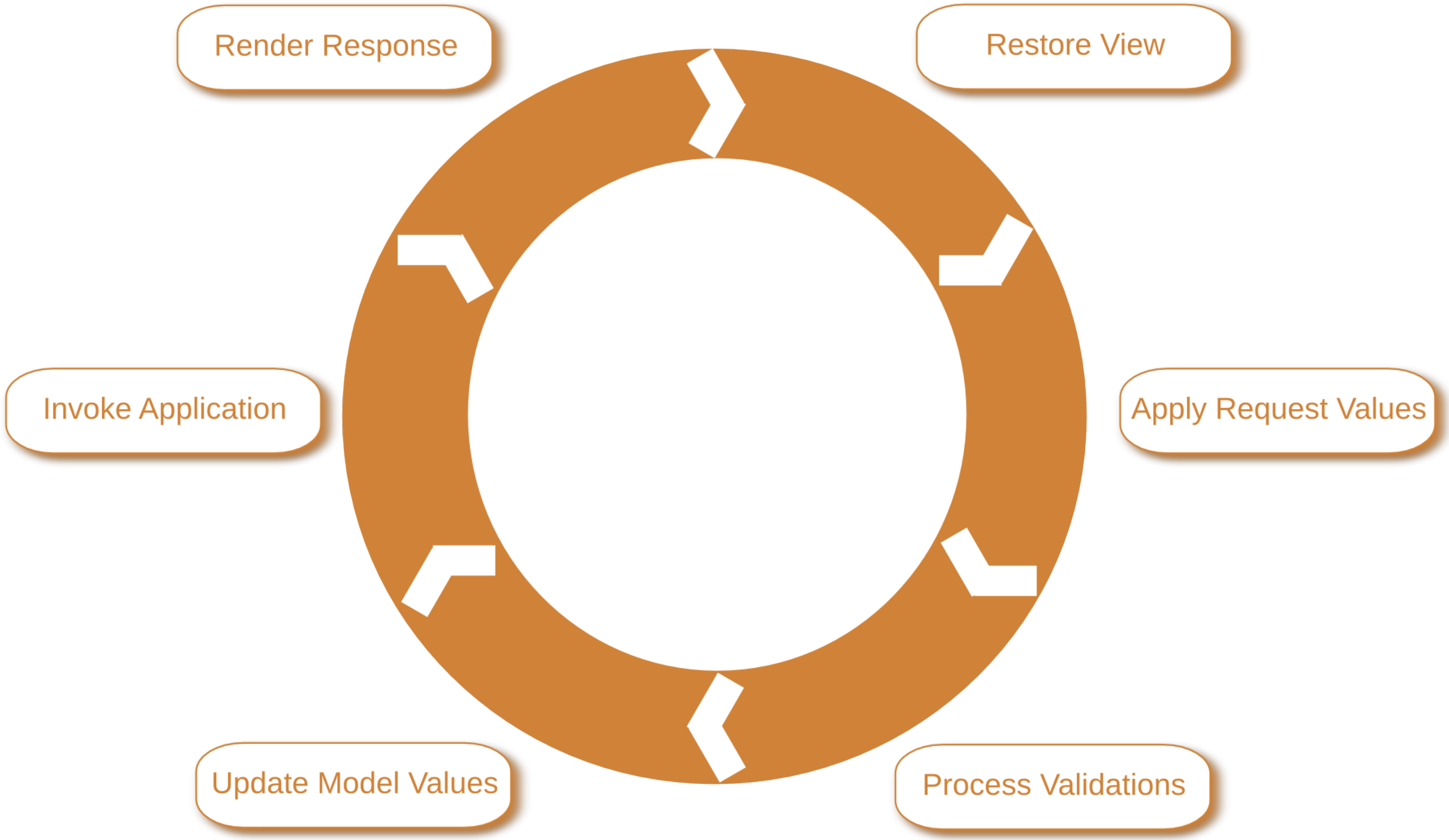
- Senior Software Engineer, JBoss by Red Hat
- RichFaces Core Developer
- Seam Faces Module Lead
- JSR 344: JavaServer Faces 2.2

The Seam Faces Emblem

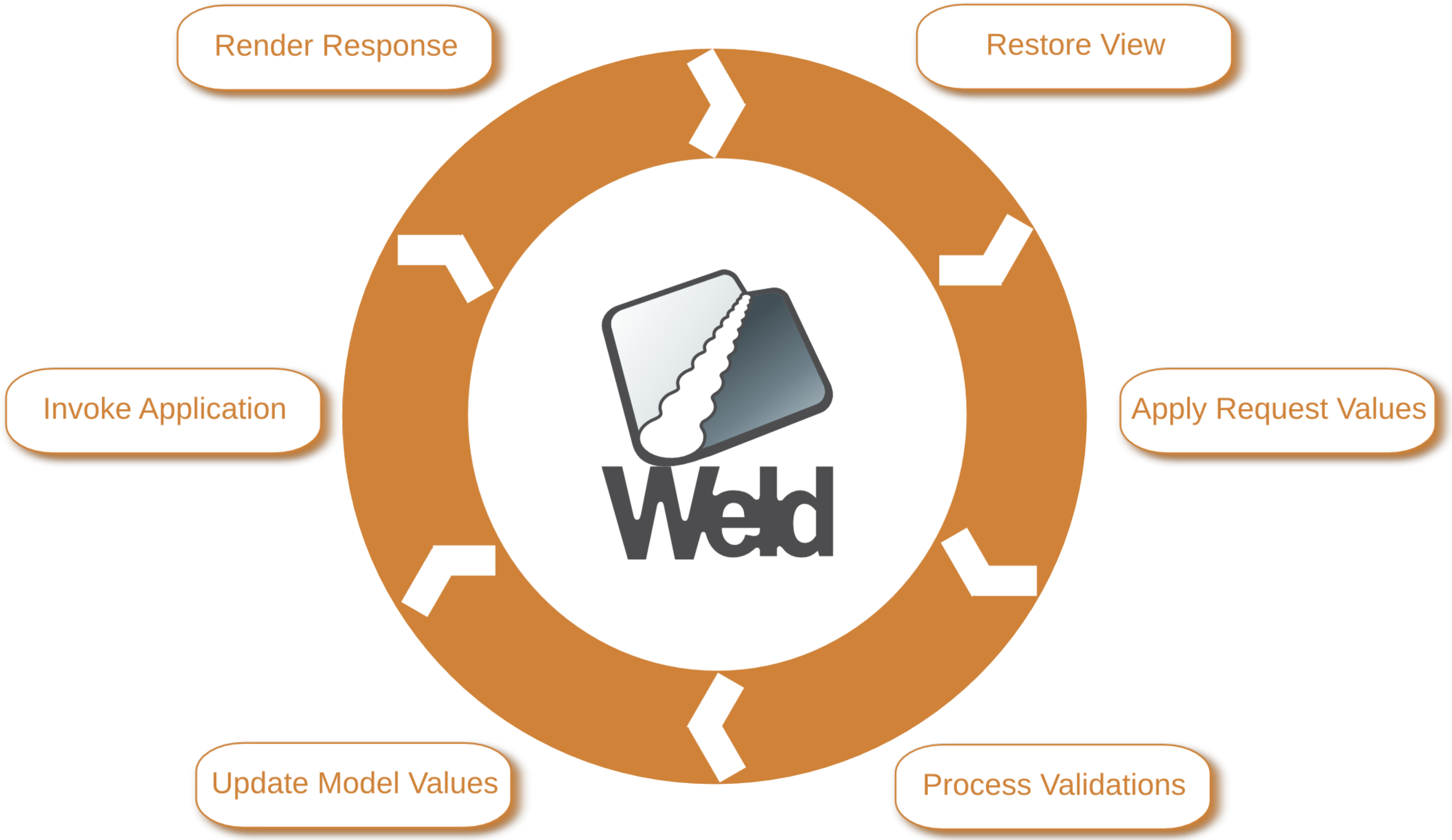
The Seam Faces Emblem



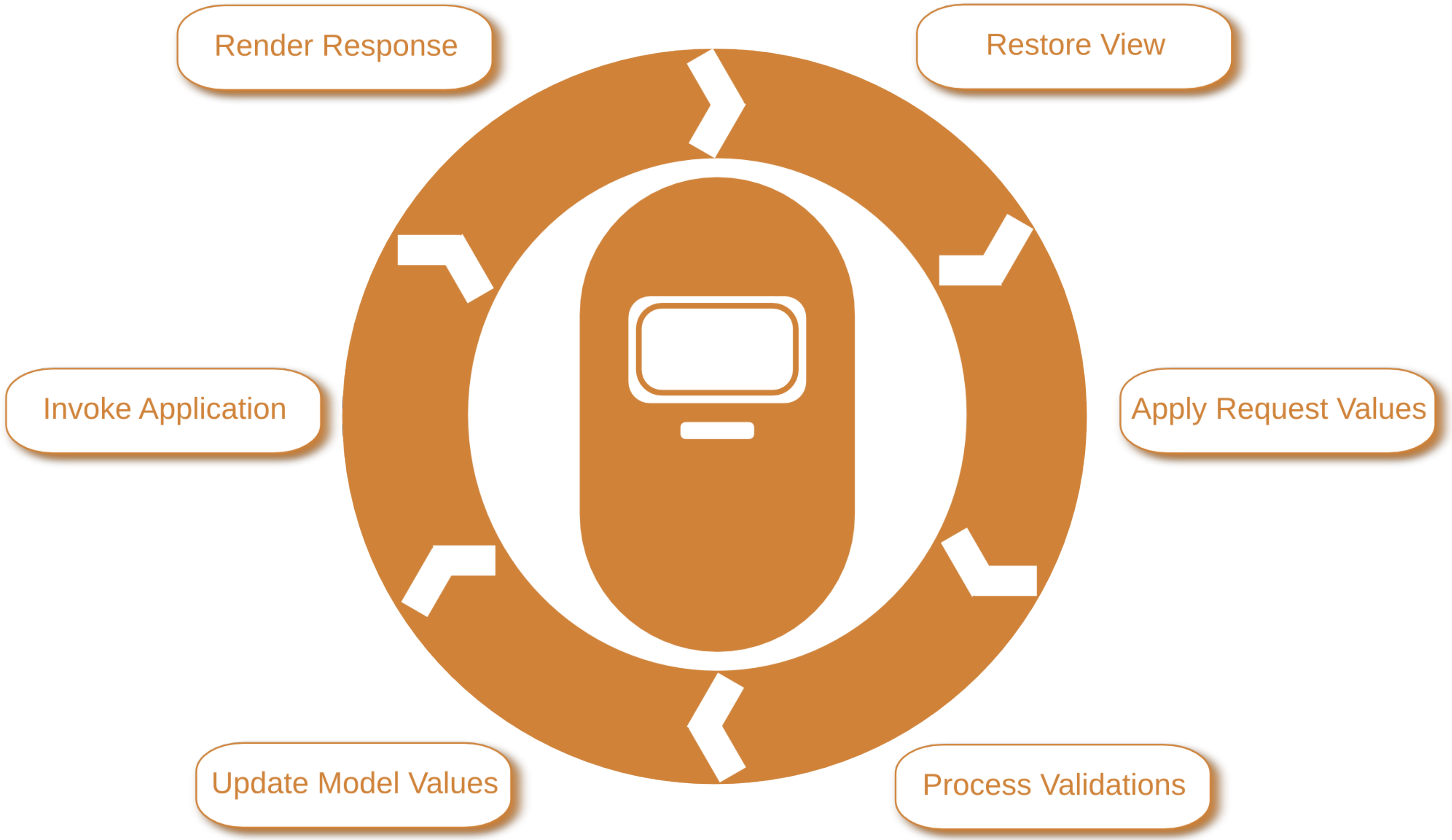
The Seam Faces Emblem



The Seam Faces Emblem



The Seam Faces Emblem



JSF? CDI? What?

Show of hands for those in the audience who:

- are familiar with JSF?
- actively develop with JSF?
- JSF 2?
- are familiar with CDI?
- are familiar with Seam 2?

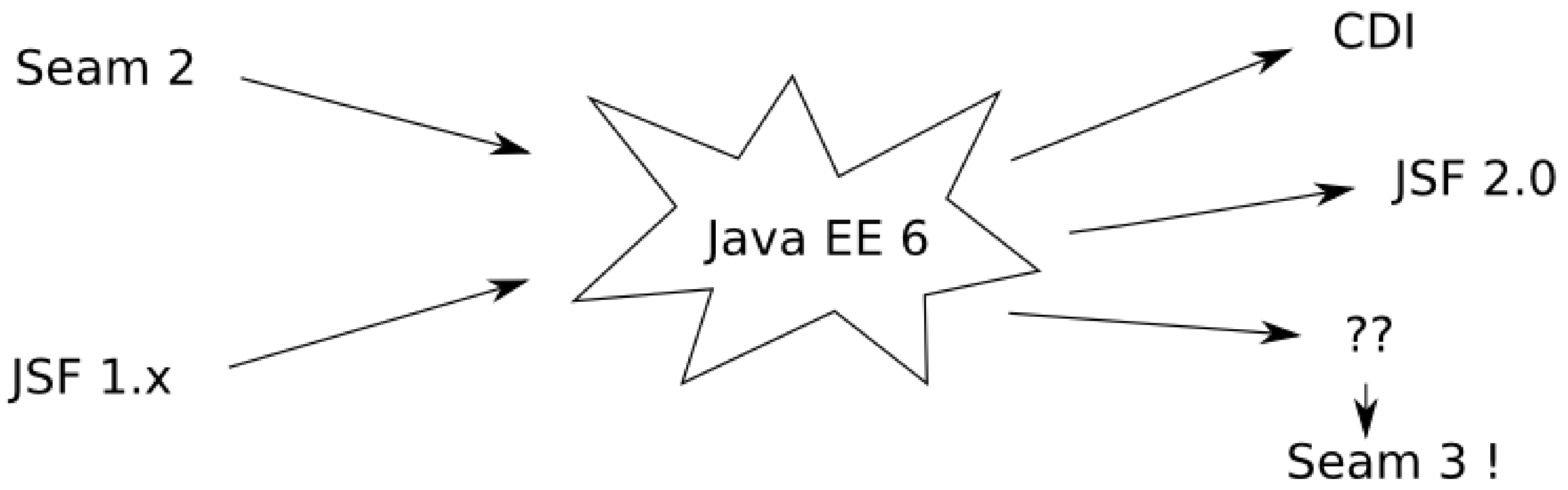
JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

Seam: Innovation, Standardization

Seam 2 married JSF 1.x with EJB 3

Much of Seam 2 was standardized in JSF 2.0 and CDI



But why bother?

Why did the Seam project bother improving JSF, and not just write yet another Java Web framework?

Why is the Seam 2 project being re-written as a set of CDI components?

JSF is Great! – A lot going for it

Part of the Java EE standard

Many groups working to make things easier for developers.

These improvements are turned back into the platform as appropriate

- – As we saw earlier with Seam 2
- – As we saw with AJAX and JSF 1.2

The JSF Ecosystem

Two JSF implementations:

1. Oracle's Mojarra	http://java.net/projects/mojarra/
2. Apache MyFaces	http://myfaces.apache.org/

JSF component suites:

1. JBoss RichFaces	http://www.jboss.org/richfaces
2. IceFaces	http://www.icefaces.org/
3. PrimeFaces	http://www.primefaces.org/
4. Apache MyFaces Tomahawk / Trinidad / Tobago	http://myfaces.apache.org/tomahawk/
5. OpenFaces	http://openfaces.org/

CDI is Awesome!

CDI: Contexts and Dependency Injection for Java EE 6

Contexts:
Request, Session, Conversation, etc.

Dependency Injection:
@Inject, @Produces, etc.

Events:

```
public void myObserver (@Observes MyEvent myEvent) { ... }
```

Finally! A consistent programming model for working with all parts of Java EE, *almost*...

JSF + CDI ⇒ Only OK.

Working with CDI and JSF 2 together is not as *Seam*less as it should be

- The scopes cannot easily interact
- Not all JSF classes are "Injectable"
- Disconnected event mechanisms

“The developer bears the burden of integrating these potentially highly-complimentary technologies.”

Enter Seam Faces

The Seam Faces goal:

> *“Build further on the integration of JSF and CDI, providing developers with a truly worthy web framework.”*

Seam Faces aims to:

- Reduce boilerplate
- Make CDI the managed bean facility in JSF
- Ease integration with other technologies
- Fill in gaps in the specification

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

Colour code

JSF style “developer” code

Seam Faces style “developer” code

Seam Faces framework code

EL short-cut refs, and Injection

		@Inject
`#{javax.enterprise.conversation}`	`#{conversation}`	✓
`#{facesContext}`	-	✓
`#{facesContext.externalContext}`	`#{externalContext}`	✓
`#{flash}`	-	✓
`#{facesContext.application. navigationHandler}`	`#{navigationHandler}`	✓
`#{facesContext.application. projectStage}`	`#{projectStage}`	✓

@Named @Produces

These EL references are made using the named CDI Producers:

```
public class ProjectStageProducer {
    @Named
    @Produces
    public ProjectStage getProjectStage(final FacesContext context) {
        return context.getApplication().getProjectStage();
    }
}
```

Conversation Boundaries

@ConversationScoped

- provided with CDI
- transient: Scope bounded by the JSF Lifecycle
- long-running: Spans multiple requests

Conversations are made “long-running” programatically

```
@Inject Conversation conversation;

public void method() {
    conversation.begin();
    ....
}
```

@Begin/@End Conversation

Seam Faces provides annotations for conversation demarcating conversation boundaries

- @Begin – begins the conversation when the method is invoked
- @End – ends the conversation when the method is invoked

```
@Begin
public void method() {
    ....
}
```

JSF Converters: Boilerplate

JSF Converters work with Object, one ends up casting in every converter created.

- These casts result in boilerplate code!

```
public class MyConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String string) {
        Object object;
        // convert string into an object
        return object;
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object object) {
        MyClass instance = (MyClass) object;
        String string;
        // convert object into a string
        return string;
    }
}
```

<Generic> Converters

Seam Faces provides <Generic> Converters:

```
public abstract class Converter<T> implements javax.faces.convert.Converter {

    @Override
    public Object getAsObject(final FacesContext context, final UIComponent comp, final
String value) {
        this.context = context;
        return toObject(comp, value);
    }

    @Override
    public String getAsString(final FacesContext context, final UIComponent comp, final
Object value) {
        this.context = context;
        return toString(comp, (T) value);
    }

    public abstract T toObject(UIComponent comp, String value);
    public abstract String toString(UIComponent comp, T value);
}
```

Converter – The Seam Faces way

The Seam Faces Converter has no casts:

```
public class MyConverter extends Converter<MyClass> {

    @Override
    public MyClass toObject(UIComponent comp, String value) {
        MyClass myClass;
        // convert string into an object
        return myClass;
    }

    @Override
    public String toString(UIComponent comp, MyClass value) {
        String string;
        // convert object into a string
        return string;
    }

}
```

Further improvements coming in a new Seam 3 module, and later in a JSR

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

Can't use DI in some JSF classes

JSF 2 has the @ManagedProperty annotation for DI

```
@ManagedProperty(value="#{userManager}")
private UserManager userManager;
```

Whereas with CDI, we use the @Inject annotation

```
@Inject
private ItemServiceFacade itemService;
```

Unfortunately neither is supported out of the box throughout all JSF classes

JSF/CDI does not support injection in Converters & Validators

DI for Everyone!

Yet, we still need to reference business objects.

This results in nasty EL lookups to retrieve managed beans:

```
public MyClass getMyClass() {
    ELContext context = FacesContext.getCurrentInstance().getELContext();
    MyClass myClass = (MyClass) context.getELResolver().getValue(context, null, "myClass");
    return myClass;
}
```

Or even worse, JNDI lookups:

```
private MyClass getMyClass() {
    try {
        InitialContext ctx = new InitialContext();
        return (TodoDaoInt) ctx.lookup("jsfejb3/MyClass/local");
    } catch (Exception e) {
        ...
    }
}
```

Seam Faces DI solution

Seam Faces enables @Inject in JSF Converters & Validators

```
public class MyConverter implements Converter {

    @Inject
    private MyClass myClass;

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String string) {
        ...
    }

    ...
}
```

But how?

Seam Faces implements a JSF ApplicationWrapper that intercepts all calls for Converters and Validators.

Seam Faces then provides Bean Managed instances of Converters and Validators.

```
public class SeamApplicationWrapper extends ApplicationWrapper {
    ...

    @Override
    public Converter createConverter(final String converterId) {
        log.debugf("Creating converter for converterId %s", converterId);
        Converter result = parent.createConverter(converterId);
        result = attemptExtension(result);
        return result;
    }
    ...
}
```

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

JSF Events

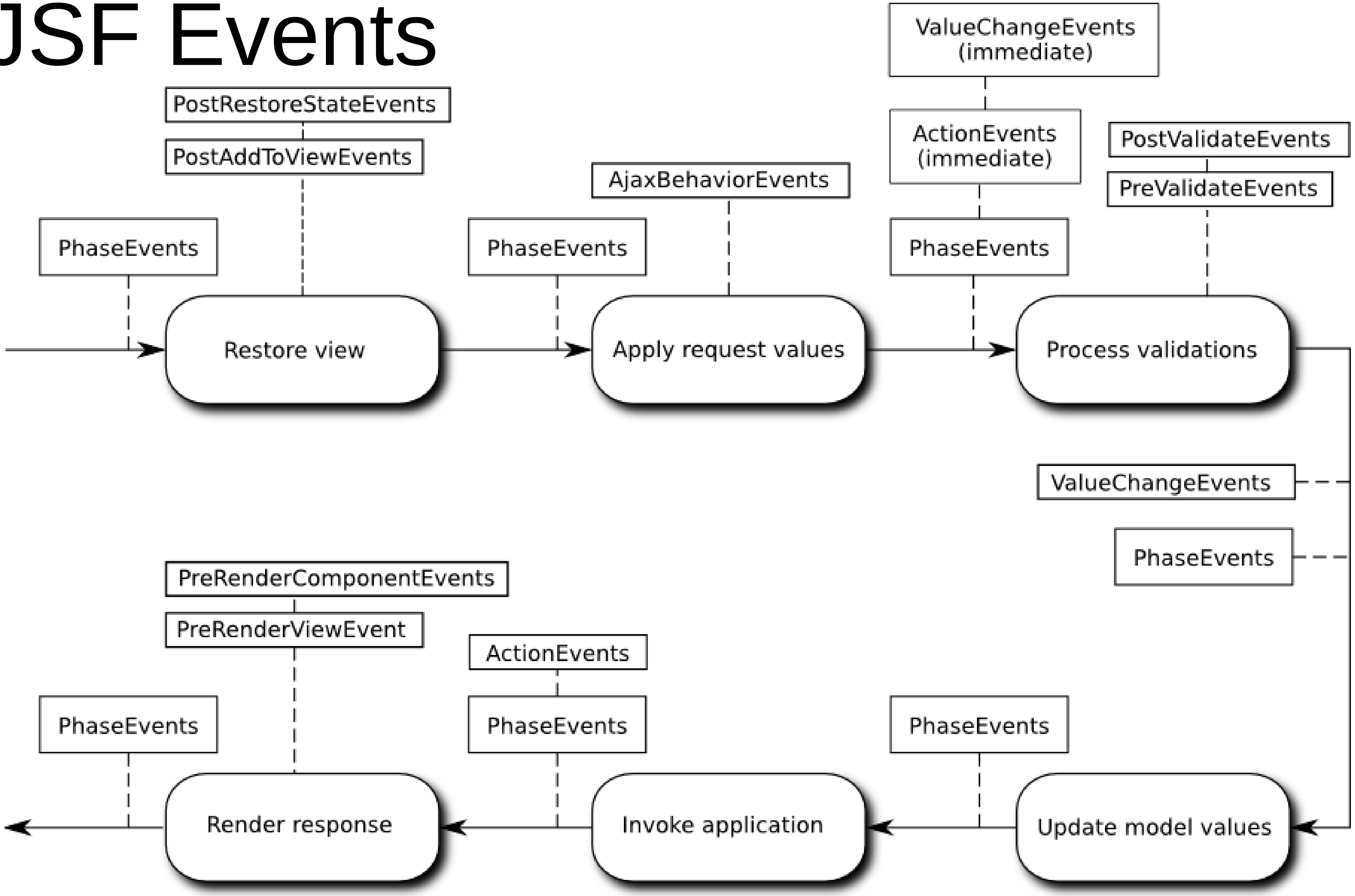


Figure: Copyright 2010 Ed Burns

JSF Phase Listeners

Implement a JSF Phase Listener - cumbersome

```
public class MyPhaseListener implements PhaseListener {

    @Override
    public void afterPhase(final PhaseEvent e) {
        // do stuff
    }

    @Override
    public void beforePhase(final PhaseEvent e) {
        // do stuff
    }

    @Override
    public PhaseId getPhaseId() {
        return PhaseId.RESTORE_VIEW;
    }
}
```

JSF System Event Listener

Implement a JSF SystemEventListener

```
public class MySystemEventListener implements SystemEventListener {

    @Override
    public boolean isListenerForSource(final Object source) {
        return true;
    }

    @Override
    public void processEvent(final SystemEvent e) throws AbortProcessingException {
        // do stuff
    }
}
```

- Non-uniform approach to listening to events
- Lots of LoC

CDI Events: An Elegant Solution

On the other hand the CDI event mechanism, is much simpler, and easier to use

Publish an event:

```
@Inject Event<MyClass> myClassEvents;

public void someMethod() {
    MyClass myClass = new MyClass();
    myClassEvents.fire(myClass);
}
```

Listen for an event:

```
public void myObserver (@Observes MyEvent myEvent) {
    // do stuff
}
```

Publish JSF Event as CDI Event

Seam Faces writes these JSF listeners for you, and publishes the Phase Events on the CDI event bus.

```
public class SystemEventBridge implements SystemEventListener {
    @Inject
    BeanManager beanManager;

    @Override
    public boolean isListenerForSource(final Object source) {
        return true;
    }

    @Override
    public void processEvent(final SystemEvent e) throws AbortProcessingException {
        Object payload = e.getClass().cast(e);
        Annotation[] qualifiers = getQualifiers(e);
        beanManager.fireEvent(payload, qualifiers);
    }

    …
}
```

Now Listening to Events is Easy!

With Seam Faces, listening to JSF events is much easier

```
public class MyClass {

    public void aMethod(@Observes @After @RestoreView PhaseEvent event) {
        // do stuff
    }

    ...

    public void bMethod(@Observes PostConstructApplicationEvent event) {
        // do stuff
    }
}
```

Additional events

Seam Faces adds to the already rich list of JSF events available:

`@Observes PreNavigate event`

- useful for intercepting and altering JSF navigation

```
@Observes PreLogin event
@Observes PostLogin event
```

- Provide a session map where values can be stored/retrieved during the login process

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

View Configuration

A **type-safe** mechanism to configure the behaviour of your JSF views.

So far these configurable behaviors include:

1. Restricting view access by integrating with Seam Security
2. Configuring URL rewriting by integrating with PrettyFaces (or any other pluggable URL rewriting framework
3. Configuring Transactions via Seam Persistence
4. JSF internals: set “?faces-redirect=true” when navigating to a view

@ViewConfig: configuration enum

```
@ViewConfig
public interface MyAppViewConfig {

    static enum Pages {

        @ViewPattern("/admin.xhtml")
        @Admin
        ADMIN,

        @ViewPattern("/item.xhtml")
        @UrlMapping(pattern="/item/#{id}/")
        @Owner
        ITEM,

        @ViewPattern("/*")
        @FacesRedirect
        @AccessDeniedView("/denied.xhtml")
        @LoginView("/login.xhtml")
        ALL;

    }
}
```

https://issues.jboss.org/browse/CDI-127

@ViewConfig annotations

```
@ViewPattern("/*")
@FacesRedirect
@LoginView("/login.xhtml")
@AccessDeniedView("/denied.xhtml")
ALL;
```

`@FacesRedirect`

sets faces-redirect=true for all associated JSF navigations

`@LoginView`

the view to navigate to when login is required

`@AccessDeniedView`

the view to navigate to when access is denied

@ViewConfig - Securing Views

Use Annotations to link view patterns to Securing methods

```
@ViewPattern("/item.xhtml")
@Owner
ITEM,


…

@SecurityBindingType
@Retention(RetentionPolicy.RUNTIME)
public @interface Owner {
}

…

public @Secures @Owner boolean ownerChecker(Identity identity, @Current Item item) {
    if (item == null || identity.getUser() == null) {
        return false;
    } else {
        return item.getOwner().equals(identity.getUser().getId());
    }
}
```

@ViewConfig - URL Rewriting

Use Annotations to rewrite URLS

```
@ViewPattern("/item.xhtml")
@UrlMapping(pattern="/item/#{id}/")
ITEM,
```

“#{id}” tells the rewriting-engine to treat the last portion of the URL as a the value of the query-parameter named “id”

urls like:

    /item/1/

get mapped into:

    /item.jsf?item=1

Courtesy of PrettyFaces (http://ocpsoft.com/prettyfaces/)

ViewConfig - Into the guts

The ViewConfigStore:

An API to programatically add and remove view configuration from the data store.

This API is used by Seam Faces to read the annotation data from the @ViewConfig enum

This API is source agnostic: the plan is to provide additional ways to feed data into the ViewConfig data store

ViewConfig Extension

The CDI extension API makes annotation scanning easy:

```
public class ViewConfigExtension implements Extension {

    private final Map<String, Set<Annotation>> data = new HashMap<String, Set<Annotation>>();

    public <T> void processAnnotatedType(@Observes ProcessAnnotatedType<T> event) {
        AnnotatedType<T> type = event.getAnnotatedType();
        if (type.isAnnotationPresent(ViewConfig.class)) {
            // add the annotations to the View Config Data Store
        }
    }
    ...
}
```

ViewConfig Data Store

The ViewConfigStore consumes the ViewConfigExtension

```
@ApplicationScoped
public class ViewConfigStoreImpl implements ViewConfigStore {

    @Inject
    public void setup(ViewConfigExtension extension) {
        for (Entry<String, Set<Annotation>> e : extension.getData().entrySet()) {
            for (Annotation i : e.getValue()) {
                addAnnotationData(e.getKey(), i);
            }
        }
    }
    ...
}
```

This ViewConfigStore is configuration source agnostic

- <f:metadata> configuration
- XML configuration
- Database configuration

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

Exception Handling

Java Applications throw exceptions. Fact of life.

Applications need to properly handle exceptions, provide appropriate logging, and direct the user accordingly.

Handling Exceptions in JSF is ugly.

- Implement a ExceptionHandler
- Provide a ExceptionHandlerFactory
- Register this factory via the faces-config.xml

Incredible power and flexibility, but too verbose

JSF ExceptionHandler

One has to write the ExceptionHandler:

```
public class MyHandler extends ExceptionHandlerWrapper {

    @Override
    public ExceptionHandler getWrapped() {
        return this.wrapped;
    }

    @Override
    public void handle() throws FacesException {
        ...
    }
}
```

JSF ExceptionHandlerFactory

Then One must extend the ExceptionHandlerFactory:

```
public class MyHandlerFactory extends ExceptionHandlerFactory {

    public MyHandlerFactory(ExceptionHandlerFactory parent) {
        super();
        this.parent = parent;
    }

     @Override
    public ExceptionHandler getExceptionHandler() {
        ...
    }
}
```

And register it in the faces-config.xml

```
     <factory>
       <exception-handler-factory>
          my.package.MyHandlerFactory
       </exception-handler-factory>
    </factory>
```

Integration with Seam Catch

Seam Faces publishes all JSF Exceptions using the CDI event mechanism

One writes Exception Handlers to handle the exceptions:

```
@HandlesExceptions
public class MyHandlers {
   void printExceptions(@Handles CaughtException<Throwable> evt) {
       System.out.println("Something bad happened: " + evt.getException().getMessage());
      evt.markHandled();
    }
}
```

This is a much “cleaner” way of handling exceptions

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

Non-visual Components

See RichFaces for UI Components:

However some non-visual components are available in Seam Faces:

- <s:viewAction>
- <s:validateForm>
- <sc:inputContainer>

<s:viewAction>

Like observing the preRenderView event on steroids

```
<f:metadata>
    <f:viewParam name="id" value="#{itemManager.itemId}"/>
    <f:event name="preRenderView" listener="#{itemManager.loadEntry}" />
</f:metadata>
```

- Doesn't perform navigation
- Can't conditionally choose the execution phase
- Can't disable for a POST

```
@Named
@RequestScoped
public class ItemManager {
    private Item itemId;

    public Item getItemId() {
        return itemId;
    }

    public void setItemId(Integer id) {
        ItemId = id;
    }

    private void loadItem() {
       // do stuff
   }
}
```

<s:viewAction>

A view action is a UICommand for an initial (non-faces) request.

```
<f:metadata>
   <f:viewParam name="id" value="#{itemManager.itemId}"/>
   <s:viewAction action="#{itemManager.loadItem}"/>
</f:metadata>
```

Can perform navigation - a full fledged UICommand

Can conditionally choose the execution phase

Can disable for a POST

<s:validateForm>

Perform cross-field form validation is simple
Place the <s:validateForm> component in the form you wish to validate, then attach your custom Validator.

```
 <h:form id="locationForm">
     <h:inputText id="cityId" value="#{bean.city}" />
     <h:inputText id="stateId" value="#{bean.state}" />
     <h:inputText id="zip" value="#{bean.zip}" />
     <h:commandButton id="submit" value="Submit" action="#{bean.submitPost}" />

     <s:validateForm fields="city=cityId state=stateId" validatorId="locationValidator" />
 </h:form>


@FacesValidator("locationValidator")
public class MyValidator implements Validator {

    @Inject
    private InputElement<String> city;
    @Inject
    private InputElement<String> state;
    ...
}
```

<sc:inputContainer>

Wraps any input component (EditableValueHolder)

```
xmlns:sc="http://java.sun.com/jsf/composite/components/seamfaces"

<sc:inputContainer label="name" id="name">
    <h:inputText id="input" value="#{person.name}"/>
</sc:inputContainer>
```

Reduces Facelet boiler plate, by creating:

- The JSF label, with required flag
- The message associated with the input
- matches up the "id" and the "for" attributes as required

<sc:inputContainer>

Define your own InputContainer to control the layout

```
<cc:interface componentType="org.jboss.seam.faces.InputContainer">
   <cc:attribute name="label" required="true"/>
   <cc:attribute name="required" required="false"/>
   <cc:attribute name="ajax" required="false" default="false"/>
   <cc:attribute name="inputs" required="false" default="1"/>
</cc:interface>
<cc:implementation>

   <div class="entry" id="#{cc.clientId}">
      <h:outputLabel id="label" for="" value="#{cc.attrs.label}:"
                      styleClass="#{cc.attrs.invalid ? 'label errors' : 'label'}">
         <h:panelGroup styleClass="required" rendered="#{cc.attrs.required}">*</h:panelGroup>
      </h:outputLabel>
      <span class="#{cc.attrs.invalid ? 'input errors' : 'input'}">
         <cc:insertChildren/>
      </span>
      <h:panelGroup rendered="#{cc.attrs.invalid}">
         <c:forEach var="i" begin="1" end="#{cc.attrs.inputs}">
            <h:message id="message#{i}" for="" styleClass="error errors"/>
         </c:forEach>
      </h:panelGroup>
   </div>

</cc:implementation>
```

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

JSF & CDI Scopes

There is a disconnect between the JSF and CDI scopes:

JSF Provided Scopes	CDI Provided Scopes
javax.faces.bean.**RequestScoped**	javax.enterprise.context.**RequestScoped**
// No JSF equivalent	javax.enterprise.context.**ConversationScoped**
javax.faces.bean.**SessionScoped**	javax.enterprise.context.**SessionScoped**
javax.faces.bean.**ViewScoped**	// No CDI equivalent
javax.faces.bean.**ApplicationScoped**	javax.enterprise.context.**ApplicationScoped**

Seam Faces overrides the JSF scopes with the corresponding CDI scopes.

Same Faces provides a CDI enabled @ViewScoped

Dynamic Annotation Replacement

Seam Faces scans for JSF Scope annotations, and replaces them with CDI Scope annotations

Again, taking advantage of the CDI Extenstion mechanism

```
public class FacesAnnotationsAdapterExtension implements Extension {
    ...
    public void aliasJsfScope(@Observes final ProcessAnnotatedType<Object> annotatedType) {
        for (Class<? extends Annotation> scope : scopeAliasMapping.keySet()) {
            if (annotatedType.getAnnotatedType().isAnnotationPresent(scope)) {
                annotatedType.setAnnotatedType(
                    decorateType(annotatedType.getAnnotatedType(), scope));
                break;
            }
        }
    }
}
```

@RenderScoped

Seam Faces provides a new scope, the Render Scope

> *Beans placed in the @RenderScoped context are scoped to, and live through but not after, the next "Render Response" phase*

What other scopes would be useful?

Come, engage the Seam community and help to define the “killer scope”

JSF and CDI – Full Potential

- Overview: JSF, CDI, Seam 3
- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing with Arquillian and JSFUnit

In container testing required

How can we test the result of dependency injection in our unit tests?

- With object mocking?
- Full system tests?

How about independent and isolated in-container tests?

Aka, “Real tests”!

Arquillian & JSFUnit

Arquillian drives the in container tests. The container in turn:

- Resolves DI
- Manages transactions, the persistence context, and all other container services

JSFUnit drives the UI, tests:

- Action Commands
- Page navigation
- Message Creation
- etc.

Shrinkwrap

We use the Shrinkwrap API to build the deployment artifact:

```
public static WebArchive createDeployment() {
    WebArchive war = ShrinkWrap.create(WebArchive.class).addAsWebInfResource(
        new File("src/test/webapp/WEB-INF/faces-config.xml"), "faces-config.xml");
    war.addAsWebResource(new File("src/test/webapp", "index.xhtml")).addAsWebResource(
        new File("src/test/webapp", "inputcontainerform.xhtml"));
    war.addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
    return war;
}
```

Arquillian

Arquillian manages the container lifecycle, and deploys the Shrinkwrap built bundle

```
@Deployment
public static WebArchive createDeployment() {
    WebArchive war = Deployments.createCDIDeployment();
    return war;
}
```

JSFUnit driving the test

JSFUnit drives the test by:

1) loading the page
2) filling in the values,
3) clicking a button
4) verifying the rendered response

```
@Test
@InitialPage("/inputcontainerform.xhtml")
public void checkComponentRenderAfterSuccess(
    JSFServerSession server, JSFClientSession client) throws IOException {
    Assert.assertEquals("/inputcontainerform.xhtml", server.getCurrentViewID());
    client.setValue(AGE_INPUT_CLIENT_ID, "100");
    client.setValue(NAME_INPUT_CLIENT_ID, "jose_freitas");
    client.click("submitInputContainer");

    Assert.assertTrue(client.getPageAsText().contains(
        "The test succeeded with jose_freitas of 100 years old"));
}
```

OPENSHIFT™

PaaS by Red Hat® Cloud

JBoss® by Red Hat

- Developing new mobile, web or Java EE apps?
- Deploy easily to the cloud w/ auto-scaling built-in
- Supports Java EE, PHP, Ruby and Python
- Check out a demo at the JBoss booth #313
- Pick up some limited edition swag

***try-it*: openshift.com** ***twitter*: @openshift**

Upcoming features

Killer Scope

Type safe navigation rules

@ViewConfig controls in <f:metadata>

Global Protection against XSRF attacks

<s:debug>

JSF 2.2

The next iteration of the JSF spec is being hashed out.

- CDI Integration is on the table:
  http://java.net/jira/browse/JAVASERVERFACES_SPEC_PUBLIC-976
- s:viewAction no longer requires f:viewParam tags

  http://java.net/jira/browse/JAVASERVERFACES_SPEC_PUBLIC-872
- s:viewAction is also being considered:
  http://java.net/jira/browse/JAVASERVERFACES_SPEC_PUBLIC-758

JBoss, Seam & CDI @JAXConf

- Java EE 6 secrets: Wield CDI like a Ninja Master

  (Dan Allen & Lincoln Baxter III)
- Forge new Ground in Rapid Enterprise Java Development

  (Dan Allen & Lincoln Baxter III)
- Seam 3 brings Java EE improvements of tomorrow, today (Dan Allen)
- The future of Java enterprise testing (Dan Allen)
- 7 reasons to love JBoss AS 7 (Dan Allen @ JBoss Day)
- RichFaces 4.0 Component Deep Dive (Jay Balunus)
- The Mobile Web Revealed For The Java Developer

  (Jay Balunus @ JBoss Day)
- Java EE on Google App Engine: CDI to the Rescue! (Ales Justin)

Conclusion

Seam Faces improves upon the JSF 2 / CDI integration, providing JSF developers with a full featured framework for developing Web Applications

- Productivity shortcuts
- Dependency Injection for Everyone!
- JSF/CDI Event Bridging
- View Configuration
- Exception Handling
- Seam Faces – non visual components
- JSF & CDI Scopes
- Testing

Get Involved

Get involved with Seam Faces, and help make the JSF platform into what you need it to be.

http://seamframework.org/Seam3/FacesModule

http://seamframework.org/Community

Twitter: #SEAM, #SEAMFACES

http://twitter.com/brianleathem

Github:

https://github.com/seam/faces