
JBoss Seam

By : Yanai Franchi, Chief Architect Tikal



Agenda

- ▶ JEE5 pros and cons
- ▶ Seam components
- ▶ Conversations
- ▶ Integration with Seam
 - » Seam AF
 - » Spring
 - » Remoting and JMS
- ▶ Integration Tests with Seam

Lightweight Framework For JEE5

- ▶ What does that mean?
- ▶ Isn't JEE5 itself a collection of "frameworks"?
- ▶ Why do you need another one that is outside the official specification?
- ▶ Seam – The "missing framework" that should have been included in Java EE 5.0.

JEE5 Programming Model – JSF

- ▶ Template language
 - » Extensible component model for widgets
- ▶ *Contextual* “Managed Bean” component model
- ▶ Defines interactions between the page and managed beans
- ▶ True MVC – No httpRequest and httpResponse
- ▶ Validation
- ▶ XML-based “Navigation Rules”

JEE5 Programming Model – EJB3

- ▶ Component model for transactional components
 - » Dependency injection
 - » Declarative transaction and persistence context demarcation
 - » Sophisticated state management
- ▶ ORM for persistence
- ▶ Annotation-based programming model

Let's Suppose We Have Some Data

```
create table Document (  
  id bigint not null primary key,  
  title varchar(100) not null unique,  
  summary varchar(1000) not null,  
  content clob not null  
)
```

We'll Use an Entity Bean

```
@Entity
public Document {
    @Id @GeneratedValue private Long id;
    private String title;
    private String summary;
    private String content;
    //getters and setters...
}
```

Surrogate key
identifier
attribute

Search Page

```
<h:form>
  Document Id
  <h:inputText value="#{documentEditor.id}"/>

  <h:commandButton type="submit" value="Find"
                    action="#{documentEditor.get}"/>
</h:form>
```

JSF value
binding

JSF control

JSF method
binding

Edit Page

```
<f:form>
  Title
  <h:inputText value="#{documentEditor.title}">
    <f:validateLength maximum="100"/>
  </h:inputText>

  Summary
  <h:inputText value="#{documentEditor.summary}">
    <f:validateLength maximum="1000"/>
  </h:inputText>

  Content
  <h:inputText value="#{documentEditor.content}"/>

  <h:messages/>

  <h:commandButton type="submit" value="Save"
                    action="#{documentEditor.save}"/>
</f:form>
```

JSF validator

Should We Use SLSB?

```
@Stateless
public EditDocumentBean implements EditDocument {
    @PersistenceContext
    private EntityManager em;

    public Document get(Long id) {
        return em.find(Document.class, id);
    }

    public Document save(Document doc) {
        return em.merge(doc);
    }
}
```

And a “Backing Bean”?

```
public class DocumentEditor {
    private Long id;
    private Document document;

    public String getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getTitle() { return document.getTitle(); }
    public void setTitle(String title){ document.setTitle(title); }
    //etc...

    private EditDocument getEditDocument() {
        return (EditDocument) new InitialContext().lookup(...);
    }

    public String get() {
        document = getEditDocument().get(id);
        return document==null ? "notFound" : "success";
    }

    public String save() {
        document = getEditDocument().save(document);
        return "success";
    }
}
```

Properties
bound to
controls
via value
bindings

Action
listener
methods
bound to
controls
via the
method
bindings

JSF
outcome

We Also Need the JSF XML

```
<managed-bean>
  <managed-bean-name>documentEditor</managed-bean-name>
  <managed-bean-class>
    com.tikal.docs.DocumentEditor
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<navigation-rule>
  <from-view-id>/getDocument.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>editDocument.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>

<navigation-rule>
  <from-view-id>/editDocument.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>findDocument.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

The name of a contextual variable we can refer to in the EL

This is a session scoped component!

Navigation rules map logical, named "outcomes" to URL of the resulting view

JEE5 Compared to J2EE

- ▶ Much simpler code
 - » Fewer artifacts (no DTO, for example)
 - » Less noise (EJB boilerplate, Struts boilerplate)
 - » More transparent (no direct calls to `HttpSession`, `HttpRequest`)
 - » Much simpler ORM (even compared to Hibernate)
 - » Finer grained components

JEE5 Compared to J2EE Cont.

- ▶ More powerful for complex problems
 - » JSF is amazingly flexible and extensible
 - » EJB interceptors support a kind of “lightweight AOP”
 - » Powerful ORM engine
- ▶ Unit testable
 - » All these components (except the xhtml pages) may be unit tested using JUnit or TestNG

Room For Improvement in JEE5

- ▶ The managed bean is just noise – its concern is pure “glue”
 - » Accounts for more LOC than any other component!
 - » It doesn't really decouple layers
- ▶ Does not work in a multi-window application
 - » Make it work is a major architecture change!

Room For Improvement Cont.

- ▶ The application leaks memory
 - » The backing bean sits in the session until the user logs out
 - » In more complex apps, this is often a source of bugs!
- ▶ “Flow” is weakly defined
 - » Navigation rules are totally ad hoc and difficult to visualize
 - » How can this code be aware of the long-running business process?
- ▶ JSF XML is too noisy – especially the first part
- ▶ Don't repeat yourself (DRY) regarding validation

Stateful Session Bean ?

- ▶ Let's assume we need “Transparent Failover”
 - » Put and reconstruct state into the DB for every request
 - Lack of scalability.
 - Added latency
 - Introduce 2nd Level Cache ?
 - LRU instead of user interaction
 - Transactional synch with DB
 - » Hold your state in HttpSession
 - A lot more difficult than it sounds – i.e. Cluster Bugs
 - You must remember to call `session.setAttribute()`
 - » Stateful Session Bean is your solution to hold state!!!

JBoss Seam

- ▶ Unify the two component models
 - » Simplify Java EE 5, filling a gap
 - » Improve usability of JSF
- ▶ Integrate jBPM
 - » BPM technology for the masses
- ▶ Deprecate so-called stateless architecture
- ▶ Enable richer user experience

Contextual Components

- ▶ Most of the problems relate directly or indirectly to state management
 - » Servlet spec contexts are not meaningful in terms of the application
 - » EJB itself has no strong model of state management
 - » We need a richer context model that includes “logical” contexts that are meaningful to the application
 - » Bind EJB component directly to our view!

Bijection Components

- ▶ DI is broken for stateful components
 - » A contextual variable can be written / read
 - » Its value changes over time
- ▶ DI was designed with stateless services in mind
- ▶ **bijection – both injection and outjection**
 - » **Dynamic, contextual, bidirectional**
 - » **Don't think of this in terms of “dependency”**
 - » **Think about this as aliasing a contextual variable into the namespace of the component**

Our First Seam Component

```
@Entity
@Name("document")
public Document {
    @Id @GeneratedValue private Long id;
    private String title;
    private String summary;
    private String content;

    @NotNull
    @Length(max=100)
    public String getTitle() { return title; }
    public void setTitle(String t) { title = t; }

    @NotNull
    @Length(max=1000, message = "This summary is too long")
    public String getSummary() { return summary; }
    public void setSummary(String s) { summary = s; }
    ...
}
```

Component
name

Validation
Rules

Error
message



The Edit Page

```
<f:form>
  <s:validateAll>
    Title: <h:inputText value="#{document.title}"/>
    Summary: <h:inputText value="#{document.summary}"/>
    Content: <h:inputText value="#{document.content}"/>
    <h:messages/ >
  </s:validateAll>
  <h:commandButton type="submit" value="Save"
                    action="#{documentEditor.save}"/>
</f:form>
```

Bind view to the entity bean directly

Error messages will display here

```
<f:facet name="afterInvalidField">
  <s:message/ >
</f:facet>
...

<s:decorate id="content">
  <h:inputText value="#{document.content}"/>
  <a:support event="onblur" reRender="content"/>
</h:inputText>
</s:decorate>
...
```

Message after invalid field

Use Seam with ajax for re-rendering



Let's Create a "Conversation"

Binds the component to a contextual variable

Injection & Outjection

Starts a conversation

End the conversation

```
@Stateful
@Name("documentEditor")

public EditDocumentBean implements EditDocumentBean {
    @PersistenceContext(type=EXTENDED)
    private EntityManager em;
    private Long id;

    @Out @In(create=true)
    private Document document;

    @Begin
    public String get() {
        document = em.find(Document.class, id);
        return document==null ? "notFound" : "success";
    }

    @End
    public String save() {
        //NO NEED TO MERGE!
        return "success";
    }

    @Destroy @Remove
    public void destroy(){}
}
```

The Managed Beans have states so we use SFSB

Extended persistence context.

Improve performance

Seam Managed Persistence Context

- ▶ Outside of a Java EE 5 environment
- ▶ Loosely coupled components that collaborate together in the scope of a single conversation
- ▶ Inject the Persistence Context with `@In` annotation
- ▶ No more `LazyInitializationException`
 - » Two system transactions to handle one JSF request
 - From Restore View phase Invoke Application phase
 - Read Only – Spans the Render Response phase of a JSF request.

Atomic Conversations

```
@In(create = true, value = "claimEM")
private EntityManager em;

@Begin(flushMode=FlushModeType.MANUAL)
public void beginClaimWizard() {
    claim = em.find(Claim.class, claimId);
}
}
```

Claim
remains
managed after
we finish the
method

```
public void addPartyToClaim() {
    Party party = ....;
    claim.addParty(party);
}
}
```

Improve
performance -
Changes will
NOT be flushed

```
@End
public void commitClaim() {
    em.flush();
}
}
```

...until we
get here



Booking Hotel Conversation Demo

Seam AF - Home Objects

- ▶ Provides persistence operations for a particular entity class: persist(), remove(), update() and getInstance().
- ▶ No need for the Session Bean nor the Backing Bean

Factory method

```
<framework:entity-home name="documentHome" entity-class="eg.Document">
  <factory name="document" value="#{documentHome.instance}"/>
</framework:entity-home>
```

Bind variable to instance created by the factory method

```
<h1>Create Document</h1>
<h:form>
  Title: <h:inputText value="#{document.title}"/>
  Summary: <h:inputText value="#{document.summary}"/>
  Content: <h:inputText value="#{document.content}"/>
  <h:commandButton value="Create Document"
    action="#{documentHome.persist}"/>
</h:form>
```

Using the Home Object to persist the document

Seam AF - CRUD with Home Objects

```
<pages>
  <page viewid="/editDocument.xhtml">
    <param name="documentId" value="#{documentHome.id}"/>
  </page>
</pages>
```

Pass the entry
identifier to the
DocumentHome

Seam AF - CRUD with Home Objects

```
<h1>
  <h:outputText rendered="#{!documentHome.managed}" value="Create
Document" />
  <h:outputText rendered="#{documentHome.managed}" value="Edit Document" />
</h1>
<h:form>
  Title: <h:inputText value="#{document.title}" />
  Summary: <h:inputText value="#{document.summary}" />
  Content: <h:inputText value="#{document.content}" />

  <h:commandButton value="Create Document" action="#{documentHome.persist}"
rendered="#{!documentHome.managed}" />
  <h:commandButton value="Update Document" action="#{documentHome.update}"
rendered="#{documentHome.managed}" />
  <h:commandButton value="Delete Document" action="#{documentHome.remove}"
rendered="#{documentHome.managed}" />
</h:form>
```

Edit or
Create will
be
determined if
we supply id

Seam AF - Query Objects

```
<framework:entity-query name="documents"  
    ejbql="select d from Document d"/>
```

Run the
query

```
<h1>List of documents</h1>  
<h:dataTable value="#{documents.resultList}" var="document">  
    <h:column>  
        <s:link view-id="/editDocument.xhtml"  
            value="#{document.title}"  
            #{document.summary}">  
            <f:param name="documentId" value="#{document.id}"/>  
        </s:link>  
    </h:column>  
</h:dataTable>
```

Page
parameter
will be used
by the Home
object

Binding the
document
variables

Injecting Components into Spring Beans

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:seam="http://jboss.com/products/seam/spring"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://jboss.com/products/seam/spring
    http://jboss.com/products/seam/spring-1.2.xsd">/

...
<seam:instance id="seamManagedEM" name="someManagedEMComponent"
  proxy="true"/>

<bean id="someSpringBean" class="SomeSpringBeanClass">
  <property name="entityManager" ref="seamManagedEM">
</bean>

...
</beans>
```

Enabling
Seam

Inject a proxy of
the Seam
component, and
resolve the
reference when the
proxy is invoked

Injecting Spring Beans into Components

```
<application>
  <variable-resolver>
    org.springframework.web.jsf.DelegatingVariableResolver
  </variable-resolver>
</application>
```

```
@In("#{bookingService}")
private BookingService bookingService;
```

Inject bookService
Spring bean into
Seam component

Making Spring Beans to Components

```
<bean id="someSpringBean" class="SomeSpringBeanClass" scope="prototype">  
  <seam:component/>  
</bean>
```

Will create a
Stateless Seam
component

Seam Remoting - Server Side

- ▶ Seam provides a convenient method of remotely accessing components from a web page, using AJAX.
- ▶ Client-side interaction with your components is all performed via the Seam Javascript object.

```
@Stateless
@Name("helloAction")
public class HelloAction implements HelloLocal {
    public String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

Component
name

```
@Local
public interface HelloLocal {
    @WebRemote
    public String sayHello(String name);
}
```

A Remote
Service

Seam Remoting - Client Side

```
<script type="text/javascript"
src="seam/resource/remoting/resource/remote.js"></script>
  <script type="text/javascript"
src="seam/resource/remoting/interface.js?helloAction"></script>
```

```
<button onclick="javascript:sayHello()">
  Say Hello
</button>
```

```
<script type="text/javascript">
  //
  function sayHello() {
    var name = prompt("What is your name?");
    Seam.Component.getInstance("helloAction").
      sayHello(name, sayHelloCallback);
  }

  function sayHelloCallback(result) {
    alert(result);
  }

  // ]]&gt;
&lt;/script&gt;</pre><p>The remote call</p><p>Creates a proxy</p><p>The callback function for results</p></div><div data-bbox="184 961 213 976" data-label="Page-Footer">Hoste</div>
```



JMS Subscription

```
function subscriptionCallback(message) {  
  if (message instanceof Seam.Remoting.TextMessage)  
    alert("Received message: " + message.getText());  
}  
  
Seam.Remoting.subscribe("topicName", subscriptionCallback);
```

Subscribe to a topic

The callback function to invoke when a message is received.

```
<remoting:remoting poll-timeout="1" poll-interval="5"/>
```

Integration Tests

- ▶ How can we emulate user interactions ?
- ▶ Where to put our assertions ?
 - » Test the whole application by reproducing user interactions with the web browser is not appropriate for use at development time.
- ▶ Solution – Script your components while running inside a pruned down container environment
 - » You get to pretend you are the JSF implementation!

Integration Tests Cont.

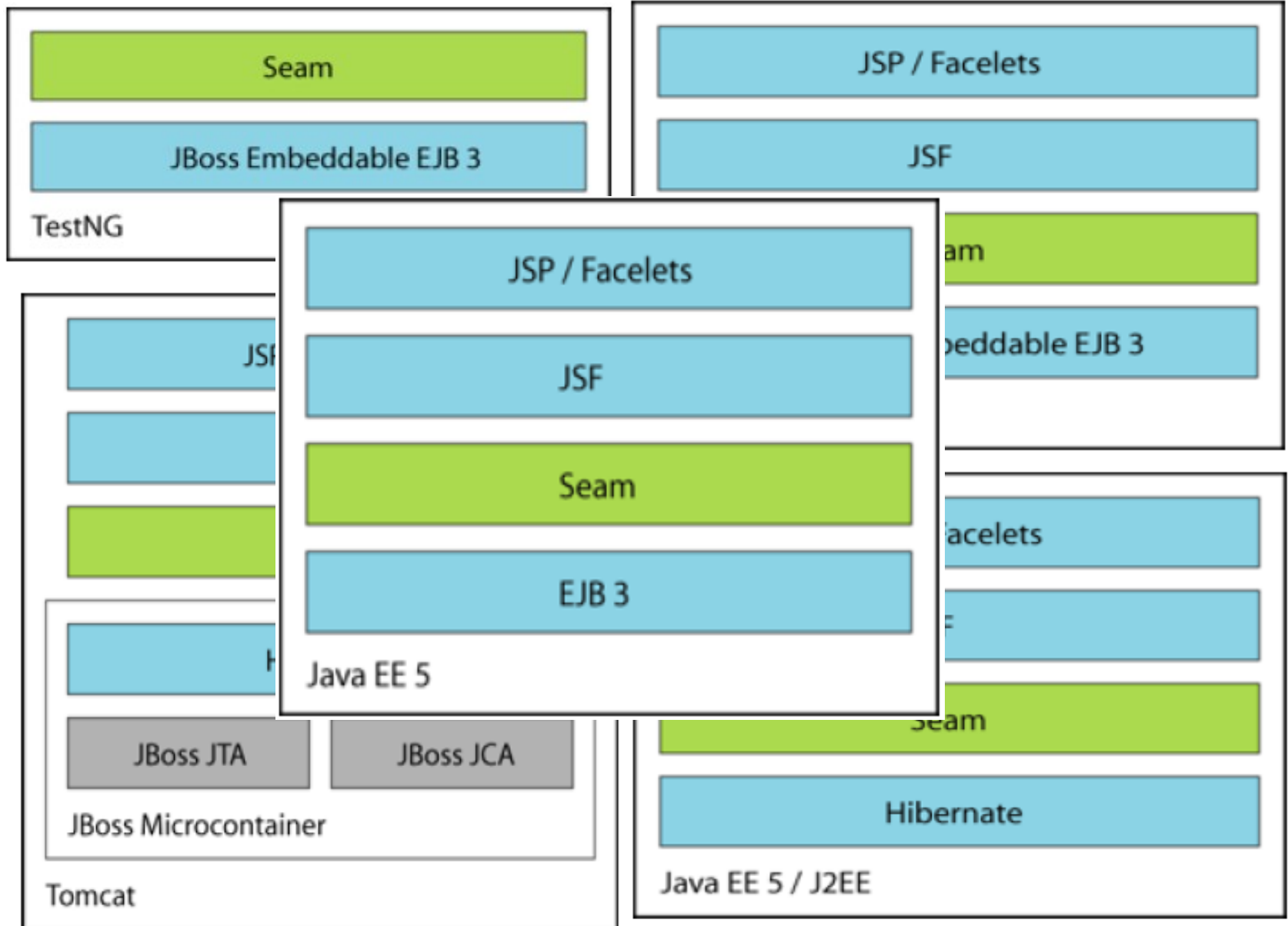
```
public class CreateDocumentTest extends SeamTest{
    @Test public void testCreate() throws Exception {
        new FacesRequest() {
            @Override protected void processValidations() throws Exception {
                validateValue("#{document.title}", "Some Document Title");
                validateValue("#{document.summary}", "A short summary");
                validateValue("#{document.content}", "Blah Blah");
                assert !isValidationFailure();
            }

            @Override protected void updateModelValues() throws Exception {
                setValue("#{document.title}", "Some Document Title");
                setValue("#{document.summary}", "A short summary");
                setValue("#{document.content}", "Blah Blah");
            }

            @Override protected void invokeApplication() {
                assert invokeMethod("#{documentEditor.save}").equals("success");
            }

            @Override protected void renderResponse() {
                assert getValue("#{document.title}", "Some Document Title");
                assert getValue("#{document.summary}", "A short summary");
                assert getValue("#{document.content}", "Blah Blah");
            }
        }.run();
    }
}
```

Runtime Environments



JBoss Seam Summary

- ▶ Provides a consistent programming model for all components in an enterprise web application.
- ▶ Other covered areas by Seam:
 - » Email, iText, Security (Authentication, Authorization), Jboss Rules, Caching, Exception Handling, I18N, Interceptors, jBPM
- ▶ Tested: GlassFish, WebLogic WebSphere, Jboss, Oracle
- ▶ Future enhancements
 - » Web Services
 - » Web Beans – JSR–299 may be introduced into JEE6



Q&A



Thank You

yanai@tikalk.com