Seam & Web Beans

Pete Muir

JBoss, a division of Red Hat

http://in.relation.to/Bloggers/Pete

pete.muir@jboss.org

Road Map

- Background
- Seam
- Web Beans

Advantages of JSF/JPA over Struts/EJB 2

- Fewer, finer grained artifacts
  - No DTOs required
  - Clean MVC
- Less noise
  - No Struts/EJB 2.x boilerplate code
  - No direct calls to HttpSession or HttpRequest
- Simple ORM
  - Even simpler than the Hibernate API!

Advantages of JSF/JPA over Struts/EJB 2

- JSF is flexible and extensible
  - Custom UI widget suites (open source)
  - Good AJAX support
- JPA
  - Powerful object/relational mapping, far beyond EJB 2.x CMP entity beans
- All components are POJO so easily testable with TestNG or JUnit

But, still some problems

- JSF
  - Backing bean couples layers and is just noise
  - Hard to refactor all the XML and String outcomes
  - No support for the business layer
  - Validation breaks DRY
  - XML is too verbose
- How do we write our business layer
  - EJB3? - can’t be used directly by JSF
  - EJB3? - no concept of scopes

And some more challenges

- Workflow
  - Ad-hoc back buttoning not suppored
  - No stateful navigation
  - Long running business processes?
- Multi-tab/window support is not built in
  - All operations happen in the session - leakage
  - No support for a conversation context
  - Memory leak - objects don’t get cleaned up quickly

Simple example

```
 <h:form>

Item:        <h:outputText value="#{itemEditor.id} />
Name:        <h:inputText value="#{itemEditor.item.name}">
               <f:validateLength maximum="255" />
             </h:inputText>
Price (EUR): <h:inputText value="#{itemEditor.item.price}"/>
               <f:convertNumber type="currency" pattern="$###.##" />
             </h:inputText>
<h:messages />
<h:commandButton value="Save" action="#{itemEditor.save}" />
</h:form>
```

JSF validation still

JSF converter

JSF provides a way of outputting messages (e.g. error, info)

Calling the business layer directly

Adding Seam

```
@Name("itemEditor") @Scope(CONVERSATION)
public class EditItemBean implements EditItem {

   @In EntityManager entityManager;

   Long id;
   Item item;
   // getter and setter pairs

   @Begin public String find(Long id) {
     item  = entityManager.find(Item.class, id);
     return item == null ? "notFound" : "success";
   }

   @End public String save(Item item) {
     item = entityManager.merge(item);
     return "success";
   }
}
```

A **conversation** scoped Seam component

Inject a Seam Managed Persistence Context (more later)

Begin and **End** a conversation - state is maintained over multiple requests between these methods

Road Map

- Background
- Seam
- Web Beans

Contextual variables

- Contexts available in Seam

- Event
- Page
- Conversation
- Session
- Business Process
- Application

JSF lifecycle - quick review

- RESTORE VIEW: Restore the tree of UI components
- APPLY REQUEST VALUES: Synchronize request parameters with UI components
- PROCESS VALIDATIONS: Validate state of UI components
- UPDATE MODEL: Synchronize UI components with bound backing bean properties
- INVOKE APPLICATION: Notify action listeners, call action methods
- RENDER RESPONSE: Render a new tree of UI components

Application lifecycle

How is state stored?

Depends on the context:

- Conversation context
  - Segmented HttpSession - times out if not used
- Page context
  - Stored in the component tree of the JSF view (page)
  - Can be stored in HttpSession or serialized to client
- Business Process context
  - Persisted to database, handled by jBPM

Bijection

- Seam provides hierarchical, stateful contexts
- (Dependency) Injection fine for stateless applications BUT stateful applications need bidirectional wiring. Think about aliasing a stateful object into a context

```
@Name("passwordChanger") public class PasswordChanger {

  @In EntityManager entityManager;

  @In @Out User currentUser;

  public void changePassword() {
    entityManager.merge(currentUser);
  }
}
```

Bijection: before the method call, inject the current user; after the method call, save it back into the context.

JPA Persistence Context

- What is the Persistence Context?
  - “a HashMap of all the objects I’ve loaded and stored”
  - holds (at most) one in-memory object for each database row while the PC is active
  - a natural first-level cache
  - can do dirty checking of objects and write SQL as late as possible (automatic or manual flushing)
- The Persistence Context has a flexible scope
  - default: same scope as the system transaction (JTA)
  - extended: the PC is bound to a stateful session bean

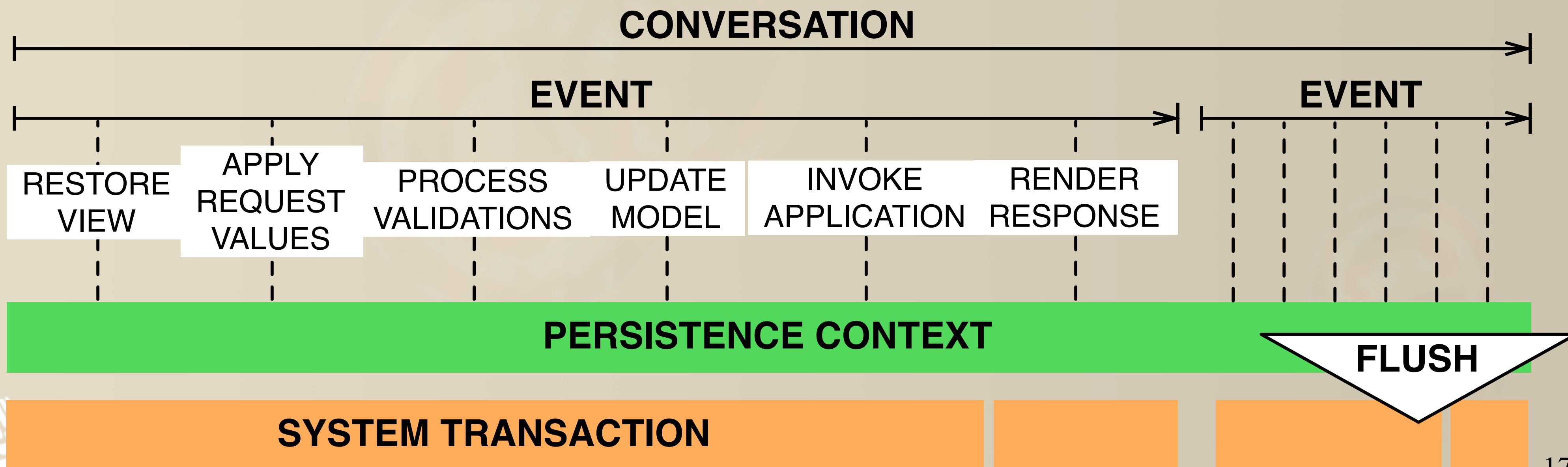
Which PC scope to use?

- Transaction scoped & detached objects
  - `LazyInitializationException`
  - `NonUniqueObjectException`
  - Less opportunity for caching
- An extended persistence context of a SFSB is
  - not available during view rendering (LIE again)
  - very complicated propagation rules
- No concept of a conversation

Seam managed persistence and transactions

Seam managed PC is conversation scoped

- Remains active through conversation,
- Inject using @In
- Allows use of manual flush mode



Road Map

- Background
- Seam concepts
- Web Beans

Web Beans Goal’s

Web Beans provides a unifying component model:

- A programming model for stateful, contextual components compatible with EJB 3.0 and JavaBeans
- An extensible context model
- Component lookup, injection and EL resolution
- Conversations
- Lifecycle and method interception
- An event notification model
- Persistence context management for optimistic transactions
- Deployment-time component overriding and configuration

Platform integration

- Web Beans may be EJB 3.0 session beans
  - take advantage of EJB declarative transactions, security, etc.
- Web Beans may be used seamlessly from JSF
  - as a replacement for JSF managed beans
  - request, session, application, conversation contexts
- Web Beans are usable from servlets
  - request, session, application contexts
  - reuses Common Annotations and **javax.interceptor**
- Web Beans will integrate tightly with JPA
  - conversation-scoped extended persistence contexts

Migration

- Any existing EJB3 session bean may be made into a Web Bean by adding annotations
- Any existing JSF managed bean may be made into a Web Bean by adding annotations
- New Web Beans may interoperate with existing EJB3 session beans
  - via **@EJB** or JNDI
- New EJBs may interoperate with existing Web Beans
  - Web Beans injection and interception supported for *all* EJBs
- New Web Beans may interoperate with existing JSF managed beans

Loose Coupling

- decouple server and client via well-defined APIs and “binding types”
  - server implementation may be overridden at deployment time
- decouple lifecycle of collaborating components
  - components are contextual, with automatic lifecycle management
  - allows stateful components to interact like services
- decouple orthogonal concerns
  - via interceptors
- completely decouple message producer from consumer
  - via events

What is a Web Bean?

Kinds of components:

- Any Java class
- EJB session and singleton beans
- Resolver methods
- JMS components
- Remote components

Essential Ingredients:

- Deployment type
- Binding types (optional)
- Name
- Implementation

Example

```
@Component
public class Hello {

    public String hello(String name) {
        return "hello " + name;
    }

}
```

A simple Web Bean

Example

```
@Component
public class Printer {

   @Current Hello hello;

   public void hello() {
      System.out.println( hello.hello(“world”) );
   }

}
```

A simple client

Example

```
@Component
public class Printer {
   private Hello hello;

   public Printer(@Current Hello hello) { this.hello=hello; }

   public void hello() {
      System.out.println( hello.hello(“world”) );
   }

}
```

A client using constructor injection

Example

```
@Component
public class Printer {

   private Hello hello;

   @Initializer
   initPrinter(@Current Hello hello) { this.hello=hello; }

   public void hello() {
      System.out.println( hello.hello("world") );
   }
}
```

A client using an initializer method

Example

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

Access from EL

Deployment Types

- A *deployment type* identifies a class as a Web Bean:
  - Deployment types may be enabled or disabled, allowing whole sets of components to be easily enabled or disabled at deployment time
  - Deployment types have a precedence, allowing the container to choose between different implementations of an API
  - Deployment types replace verbose XML configuration documents

Deployment Type example

```
@French
@Component
public class Hi extends Hello {
    public String hello(String name) {
        return "Bonjour " + name;
    }
}
```

Declare the deployment type

```
<web-beans>
    <deployment-types>
        <deployment-type>javax.webbeans.Standard</deployment-type>
        <deployment-type>javax.webbeans.Component</deployment-type>
        <deployment-type>org.jboss.i18n.Spanish</deployment-type>
    </deployment-types>
</web-beans>
```

Enable deployment types

Binding Types

- A *binding type* is an annotation that lets a client choose between multiple implementations of an API
  - Binding types replace lookup via string-based names
  - **@Current** is the default binding type

Binding Type example

```
@Casual
@Component
public class Hi extends Hello {
    public String hello(String name) {
        return "hi " + name;
    }
}
```

Declare a component which can be resolved by injection

```
@Component
public class Printer {
    @Casual Hello hello;
    public void hello() {
        System.out.println( hello.hello("London") );
    }
}
```

Inject, resolving against binding types

Scopes and Contexts

- Extensible context model
  - A scope type is an annotation
  - A context is associated with the scope type
  - Custom scopes
- Dependent scope, **@Dependent**
- Built-in scopes:
  - Any servlet
    - **@ApplicationScoped**, **@RequestScoped**, **@SessionScoped**
  - JSF requests
    - **@ConversationScoped**
  - Web service request, RMI calls...

Scope example

```
@ConversationScoped
@Component
public class ChangePassword {
   @UserDatabase EntityManager em;
   @Current Conversation conversation;
   private User user;
   public User getUser(String userName) {
      conversation.begin();
      user = em.find(User.class, userName);
   }
   public User setPassword(String password) {
      user.setPassword(password);
      conversation.end();
   }
}
```

Start and end a conversation through programs

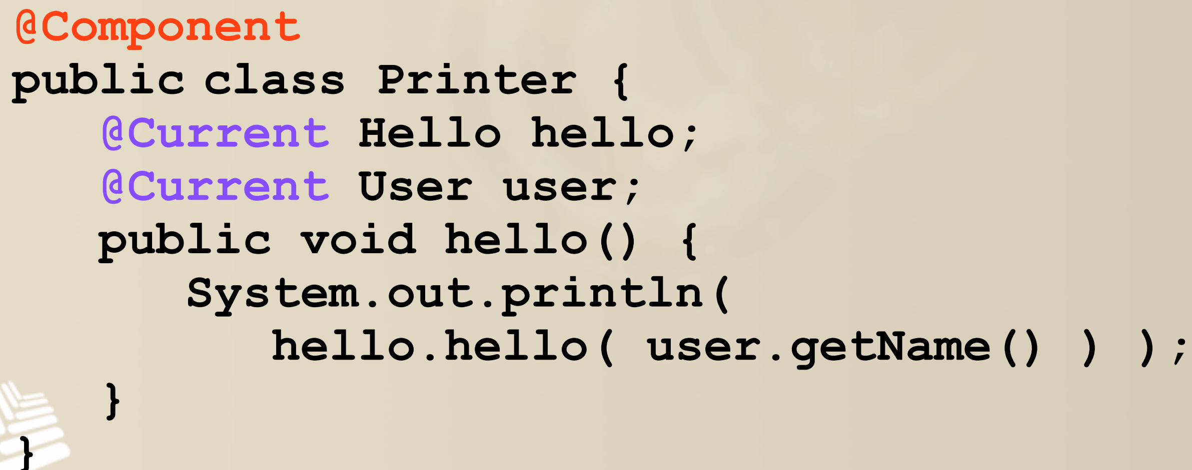
Producer Methods

- Producer methods allow control over the production of a component instance
  - For runtime polymorphism
  - For control over initialization
  - For Web-Bean-ification of classes we don't control
  - For further decoupling of a "producer" of state from the "consumer"

Producer Methods example

```
@SessionScoped
@Component
public class Login {
   private User user;
   public void login() {
      user = ...;
   }

   @Produces User getUser() { return user; }
}
```



```
@Component
public class Printer {
   @Current Hello hello;
   @Current User user;
   public void hello() {
      System.out.println(
         hello.hello( user.getName() ) );
   }
}
```

Stereotypes

- Composite annotations
  - Binding types
  - Deployment types
  - Interceptors
  - Scope type
  - requiredType & supportedScope

Stereotype

```
@Stereotype(requiredTypes=Animal.class)
@Target( { TYPE })
@Retention(RUNTIME)
@RequestScoped
@British
@Production
public @interface BritishAnimals {}
```

Require these classes to be implemented by any class annotated with this stereotype

The component is *Request* scoped (each component can have only one scope

Declare a binding type

The deployment type, highest precedence is used

More Web Beans

- Easy to use and declare Interceptors
- Decorator (delegate pattern) - *Still under discussion*
- Simple event bus
- Transaction control - *Still under discussion*
- Validation from JSR-304 - *Still under discussion*
- Specification status?
    - Still some open issues, but getting there
    - Aimed at EE, but pressure for it to be in SE

Web Beans RI

- Standalone version
  - Apache License (very liberal)
  - Run in any app server or standalone
- Microcontainer based
  - LGPL (as with rest of JBoss projects)
  - Run in any app server which the MC runs in, or standalone
  - Container will be very extendable
  - Provide “legacy” Seam compatibility layer
- Get involved!

Q&A

- http://in.relation.to/Bloggers/Pete
- http://www.seamframework.org
- http://www.seamframework.org/WebBeans

Seam provides...

- Security
- Email templates
- PDF templates
- JavaScript Remoting
- Asynchronicity (Java SE, EJB3 or Quartz)
- “Google your app” using Hibernate Search
- Integration and Unit Testing
- JSF components (deep integration into JPA)
- Components in groovy
- Webservices
- > 25 examples
- Portal support
- Validation
- BPM support
- Stateful navigation

Seam 2.1 Roadmap

- Wicket as a view layer
- GWT as a view layer
- First class support for other other containers (e.g. Websphere)
- Identity Management
- SSO for security
- Deeper integration with JBoss Portal (inter-portlet communication)