JBoss Seam

Gavin King & Thomas Heute

JBoss

October 11th 2005

Why do we need it ?

- EJB 3.0 and JSF are great, but how do they work together ?
- Clustering technology has improved to the point where stateful architectures can be efficiently scaled

How is it different ?

- Layered architecture
- Web tier calls EJB
- Stateless components
- XML
- Dependency injection
- UI validation
- Request-oriented
- Shared, second-level data cache
- State management in code

- Don't repeat yourself
- Web tier *is* EJB
- Stateful components
- Annotations
- Bijection
- Model constraints
- Conversations
- Natural cache of conversational objects
- Contextual, declarative state management

Seam component model

- Seam unifies the component models of JSF and EJB 3.0
  - ✓ Allows you to use EJB components as JSF managed beans
  - ✓ "One kind of stuff"
- Component types
  - ✓ Any JavaBean
  - ✓ Stateful session beans
  - ✓ Entity beans
  - ✓ Stateless session beans
- Component type limitations
  - ✓ Stateless session beans always belong to **STATELESS** pseudo-context
  - ✓ Entity beans are not intercepted, so they can't have bijection, context demarcation, etc.

Context model

- "Session" is not a meaningful construct in terms of the application
  - ✓ We need new, logical contexts
- Seam defines the following contexts
  - ✓ **EVENT** (request)
  - ✓ **CONVERSATION** (logical sequence of requests)
  - ✓ **SESSION** (servlet session)
  - ✓ **PROCESS** (the long-running business process)
  - ✓ **APPLICATION** (servlet context)
  - ✓ **STATELESS** (all stateless components)

State management

- Component instances are associated with a *context variable*
  - ✓ Component name defined by the `@Name` annotation
  - ✓ Component scope given by `@Scope`
- So, we can refer to the **User** instance by name
  - ✓ Seam might even instantiate it automatically
  - ✓ `@In(create=true) private User currentUser;`
  - ✓ `<h:inputText value="#{currentUser.name}"/>`
- The state of the object is cleaned up when the context ends
  - ✓ Reduces memory leakage
- Lifecycle methods
  - ✓ `@Create` when Seam instantiates the component
  - ✓ `@Destroy` when the context ends

Example code – Java bean

```
@Entity
@Name("user")
@Scope(SESSION)
public class User implements Serializable
{
   private String username;
   private String password;
   private String name;

   public User() {}

   public String getName() { return name; }

   public void setName(String name) { this.name = name; }

   public String getPassword() { return password; }

   public void setPassword(String password) { this.password = password; }

   @Id
   public String getUsername() { return username; }

   public void setUsername(String username) { this.username = username; }
}
```


Example code - JSF

```
<div class="entry">
  <h:outputLabel for="username">Username:</h:outputLabel>
  <h:inputText id="username" value="#{user.username}"/><br/>
  <span class="errors"><h:message for="username" /></span>
</div>
<div class="entry">
  <h:outputLabel for="name">Real Name:</h:outputLabel>
  <h:inputText id="name" value="#{user.name}" /><br/>
  <span class="errors"><h:message for="name" /></span>
</div>
<div class="entry">
  <h:outputLabel for="password">Password:</h:outputLabel>
  <h:inputSecret id="password" value="#{user.password}" /><br/>
  <span class="errors"><h:message for="password" /></span>
</div>
<div class="entry">
  <h:outputLabel for="verify">Verify Password:</h:outputLabel>
  <h:inputSecret id="verify" value="#{register.verify}" /><br/>
  <span class="errors"><h:message for="verify" /></span>
</div>
<div class="entry errors"><h:messages globalOnly="true" /></div>
<div class="entry">
  <h:commandButton value="Register" action="#{register.register}" class="button"/>
  <h:commandButton value="Cancel" action="login" class="button"/>
</div>
```


Conversation context

- A conversation is a logical scope, demarcated by the application
  - ✓ Bigger than a request, smaller than a login session
  - ✓ Multiple concurrent conversations per user (multiple windows)
  - ✓ Provides isolation of work done in different windows!
  - ✓ For now, demarcation done by annotation of action listener methods: **@Begin @End**
  - ✓ You should define the scope based upon functional requirements and performance considerations
  - ✓ Eventually, we will provide client-side SFSB

Conversation context

- Server-side conversations
  - ✓ Conversation state held in **HttpSession**
  - ✓ Destroyed when conversation ends
  - ✓ Or by conversation timeout
- Client-side conversations
  - ✓ Conversational state serialized to browser
  - ✓ Use JavaBeans instead of SFSBs
  - ✓ Eventually, we will provide client-side SFSB

Seam booking demo

Example code (1/2)

```
@Stateful
@Name("hotelBooking")
@Interceptor(SeamInterceptor.class)
@Conversational(ifNotBegunOutcome="main")
@LoggedIn
public class HotelBookingAction implements HotelBooking, Serializable
{
   private String searchPattern;
   public void setSearchPattern(String pat) { searchPattern=pat; }
   …
   @Begin
   public String find()
   {
      hotel = null;
      hotels = em.createQuery("...")
            .setParameter("search", searchPattern)
            .setMaxResults(50)
            .getResultList();
      return "main";
   }
   …
```

Example code (2/2)

```
   public String selectHotel()
   {
      if ( hotels==null ) return "main";
      setHotel();
      return "selected";
   }
   …
   @End
   public String confirm()
   {
      if (booking==null || hotel==null) return "main";
      em.persist(booking);
      log.info("booking confirmed");
      return "confirmed";
   }
}
```


Conversational page flow

- JSF navigation rules define page flow
  - ✓ But navigation rules are totally ad hoc
  - ✓ There is no "demarcation" of what user interaction a rule belongs to
  - ✓ Conversation demarcation is in the annotations
- Much better solution
  - ✓ Define page flow using jBPM
  - ✓ Then, a jBPM process instance will naturally demarcate the conversation
  - ✓ We could have *nested* conversations

Business process context

- jBPM process instance defines a scope
  - ✓ Spans multiple conversations with multiple users
  - ✓ In the context of a business process, usually, a conversation is a jBPM *task*
- jBPM engine provides:
  - ✓ Process flow and demarcation in XML
  - ✓ Provides the mechanism for persisting process state (ie. Seam components in the **PROCESS** context)
  - ✓ User task list
  - ✓ Transition events (these should have their own Seam contexts)
- Seam provides:
  - ✓ Transparency
  - ✓ Abstraction

Bijection

- Dependency injection does not work for stateful components
  - ✓ Stateful instances are not interchangeable
  - ✓ Components in wider scopes need to use components in narrower scopes
- Bijection is
  - ✓ Dynamic (invocation-time)
  - ✓ Contextual
  - ✓ Bidirectional (read + write)
- Don't think about dependency!
  - ✓ Think in terms of aliasing *context variables* to attributes of the component:
    `@In @Out private User currentUser;`
  - ✓ Especially useful for entities:
    `currentUser = em.merge(currentUser);`
  - ✓ IoC addresses dependencies among stateless services; bijection addresses collaboration of stateful components in various contexts

Data model constraints

- Most "validation" is really just enforcing constraints that apply to the data model
  - ✓ Don't repeat yourself applies here
- Hibernate Validator provides a set of annotations for expressing constraints directly on the entity
  - ✓ Or on any other object
- These constraints will now apply at all level of the application
  - ✓ When receiving user input
  - ✓ Before writing to database
  - ✓ When generating DDL
  - ✓ (Anywhere else you like!)
- You can add "extra" validation only when you need it
  - ✓ In the JSF form
  - ✓ In the action listener method

Example code (1/3)

```
@Entity
@Name("booking")
public class Booking implements Serializable
{
   private Long id;
   private User user;
   private Hotel hotel;
   private Date checkinDate;
   private Date checkoutDate;
   private String creditCard;

   public Booking() {}

   @Id(generate=GeneratorType.AUTO)
   public Long getId()
   {
      return id;
   }
   public void setId(Long id)
   {
      this.id = id;
   }

   @NotNull
   @Basic(temporalType=TemporalType.DATE)
   public Date getCheckinDate()
   {
      return checkinDate;
   }
```

Example code (2/3)

```
public void setCheckinDate(Date datetime)
{
   this.checkinDate = datetime;
}

@ManyToOne @NotNull
public Hotel getHotel()
{
   return hotel;
}
public void setHotel(Hotel hotel)
{
   this.hotel = hotel;
}

@ManyToOne @NotNull
public User getUser()
{
   return user;
}
public void setUser(User user)
{
   this.user = user;
}
```


Example code (3/3)

```
@Basic(temporalType=TemporalType.DATE)
@NotNull
public Date getCheckoutDate()
{
   return checkoutDate;
}
public void setCheckoutDate(Date checkoutDate)
{
   this.checkoutDate = checkoutDate;
}

@NotNull(message="Credit card number is required")
@Length(min=16, max=16, message="Credit card number must 16 digits long")
@Pattern(regex="\\d*", message="Credit card number must be numeric")
public String getCreditCard()
{
   return creditCard;
}

public void setCreditCard(String creditCard)
{
   this.creditCard = creditCard;
}
}
```


State and clustering

- Traditional SFSB implementation:
  - ✓ Stickiness or cluster-wide replication (or write to database!)
  - ✓ Replicate whole bean at end of transaction
- JBoss 5 SFSB implementation:
  - ✓ Stickiness with replication to n-of-m nodes
  - ✓ Replicate only the attributes which actually changed
- Can't I just use the HttpSession?
  - ✓ Fine-grained passivation and passivation policies
  - ✓ Automatic change detection (no need to call **`setAttribute()`** to force replication)
  - ✓ Potentially, a stateful bean can outlast a login session (state associated with the long-running business process)

Conversations and caching

- Traditional web architecture avoids stateful components
  - ✓ All state goes to database or client on each request
  - ✓ Database is the least scalable tier
  - ✓ Serializing state to the client is also expensive
- To improve performance, people add a shared second-level cache
  - ✓ Oops, we just became stateful
  - ✓ Managing consistency of a shared cache with the database is a virtually intractable problem in full generality
  - ✓ Keeping unshared data in a shared cache has inefficiencies (LRU algorithm is suboptimal)
- Instead, conversations give you a natural cache of data associated with the user
  - ✓ Consistency is well-defined (optimistic locking) without overhead or cluster-wide replication
  - ✓ Eviction is efficient (when conversation ends)
- In practice, a combination of the two strategies makes sense
  - ✓ Some data is truly shared

Conversations and persistence

- EJB goes some way toward a solution
  - Transaction scoped persistence context (LIE still possible when rendering view, or in the "next" transaction)
  - Extended persistence context for SFSB (LIE still possible when rendering view)
- Seam completely solves this problem
  - You can easily have a conversation scoped Seam-managed persistence context
  - The Seam-managed context spans the entire request cycle, including render response
  - Two transactions per request: one during update model values/invoke application, the next during render response
  - This ensures that all write operations are successful before displaying page to the user
  - Objects are never detaches, so no need to use **`merge()`** or **`saveOrUpdate()`**
  - As long as you access your entities within the scope of the conversation, you will never get **`LazyInitializationException`** or his friend **`NonUniqueObjectException`**

Conversations and persistence

- Hibernate users all complain about **`LazyInitializationException and NonUniqueException`**
  - ✓ *No, you absolutely cannot just start fetching data from the database outside of a persistence context!*
  - ✓ You would totally break association integrity and expose your application to far more insidious problems with data aliasing
  - ✓ Yes, you do have to end the persistence context somewhere, otherwise your object graph will gradually expand, as more and more associations are fetched, until you get OOME
  - ✓ *This is a basic limitation of all data access technologies in an online environment, not a bug in Hibernate!*
- Three solutions to this problem
  1. Do like JDBC: don't have associations
  2. Do like EJB2/DTO: use an *assembly phase* (this is implicit in the DTO pattern)
  3. Use a conversation-scoped persistence context

Interceptors

- EJB 3.0 has a nice way to define interceptors for session bean components
  - ✓ Annotate the session bean with `@Interceptor(LoggedInInterceptor.class)`
  - ✓ But, on second thoughts, it's a bit noisy
- Instead, apply the **`@Interceptor`** annotation as a meta-annotation

```
@Interceptor(LoggedInInterceptor.class)
public @interface LoggedIn {}
```

- Oh, and you can use this for plain JavaBeans, too

Seam outside Java EE 5

- Seam is conceived for use in a Java EE 5 environment, but:
  - ✓ you can use Seam with JBoss Embeddable EJB3, in any appserver
  - ✓ Yes, even in Tomcat
  - ✓ If you are scared of EJB, you can use Seam with JavaBeans components and Hibernate
  - ✓ If you want to do this in Tomcat, you need to use the JBoss Microcontainer to provide JTA/JNDI/JCA

Testing Seam

- You can unit test Seam components in TestNG or JUnit
  - ✓ They are all just POJOs
- You can integration test Seam applications in TestNG or JUnit
  - ✓ Embeddable EJB3 and Microcontainer can run inside a unit test
  - ✓ Seam includes a framework for integration testing
  - ✓ Basically, you write a test script that reproduces the operations performed by JSF when the form is submitted (setting model values, invoking action listener method) and then makes a set of assertions
  - ✓ This tests the entire application, with the exception of the view template
  - ✓ Its actually *really* easy

Roadmap

- Seam 1.0 beta 2
  - ✓ Improved jBPM integration including task list JSF component
  - ✓ Tomcat integration
  - ✓ Support in Hibernate Tools
- Seam 1.0 final
  - ✓ jBPM conversation flow definition
  - ✓ Portal integration
  - ✓ More improvements to jBPM integration
- Future
  - ✓ Seam for web services
  - ✓ Seam for rich clients