Hibernate - EJB 3.0 - JBoss Seam

Christian Bauer

christian.bauer@jboss.com

christian@hibernate.org

Agenda

- What is Object/Relational Mapping?
- Why Hibernate?
- Current modules and releases
- The new EJB3 programming model
- Towards stateful application design
- Introduction *JBoss Seam*

How does ORM work?

Persistence

State outlives the process that created it.

- In object-oriented applications:
  - ✓ interconnected objects are transient, not only single instances
  - ✓ some subgraph becomes persistent and can outlive power failure
  - ✓ the persistent subgraph can be recreated at a later point in time
  - ✓ in the same or another process

Object/Relational mismatch

- Loading and storing graphs of objects using a tabular relational database exposes us to the mismatch problem

- We identified 5 generic issues
  - ✓ Granularity of types
  - ✓ Type inheritance / Subtypes
  - ✓ Node identity
  - ✓ Node associations (directionality + cardinality)
  - ✓ Node access paths (n+1 selects)

Transparent ORM

- Object/relational mapping (ORM)
  - ✓ solves the mismatch problems in middleware through transformation
  - ✓ metadata governs this transformation
- Elements of an ORM middleware
  - ✓ a *transparent* programming model for domain classes and relationships
  - ✓ an API for performing CRUD operations
  - ✓ query facilities
  - ✓ metadata facilities

POJO programming model

- The rules for **P**lain **O**ld **J**ava **O**bjects are best practices
  - ✓ loosely follow JavaBean specification
  - ✓ accessor methods (getters and setters)
  - ✓ declare `Serializable` for externalization

- Hibernate requirements
  - ✓ no-arg constructor - `package` visibility
  - ✓ interface types for collection properties (`Map`, `Set`, `List`, `Collection`)

Simple persistent POJO

```
public class Customer
  implements Serializable {

  private String name;
  private Address address;

  Customer() {}

  public String getName() { return name; }
  public void setName(String n) { name = n; }

  public Address getAddress() { return address; }
  public void setAddress(Address adr) { address = adr; }

  public MonetaryAmount calcShippingCosts(Address from) {
      // Business method implementation
  }

}
```

Simple Hibernate Mapping

```
<hibernate-mapping package="shop.model">
 <class name="Customer" table="CUSTOMER"
   <id>... Primary Key Mapping ...</id>

   <property name="name" column="C_NAME" not-null="true"/>

   <component name="address" class="shop.model.Address">
      <property name="street" column="ADR_STREET"/>
      <property name="city" column="ADR_CITY"/>
      <property name="zipCode" column="ADR_ZIPCODE"/>
   </component>

   <set name="orders" table="ORDER">
      <key column="CUSTOMER_ID"/>
      <one-to-many class="Order"/>
   </set>

</class>
</hibernate-mapping>
```

EJB 3.0 / JDK 5.0 Annotations

```
@Entity
@org.hibernate.annotations.BatchSize(10)
public class Customer implements Serializable {

  private String name;

  @Dependent
  private Address address;

  @Transient
  private Integer age;

  @OneToMany(cascade=CascadeType.ALL")
  @JoinColumn(name="CUSTOMER_ID")
  Set<Order> orders;

  Customer() {}

  // Accessor and business methods
}
```

Hibernate CRUD interfaces

```
Customer c = new Customer();

// Store a Customer
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction()

session.save(c);
c.setName("John Doe");

tx.commit();
session.close();
```

- ✓ SessionFactory - configuration object
- ✓ Session - single unit of work
- ✓ Transaction shields code from the underlying transaction system

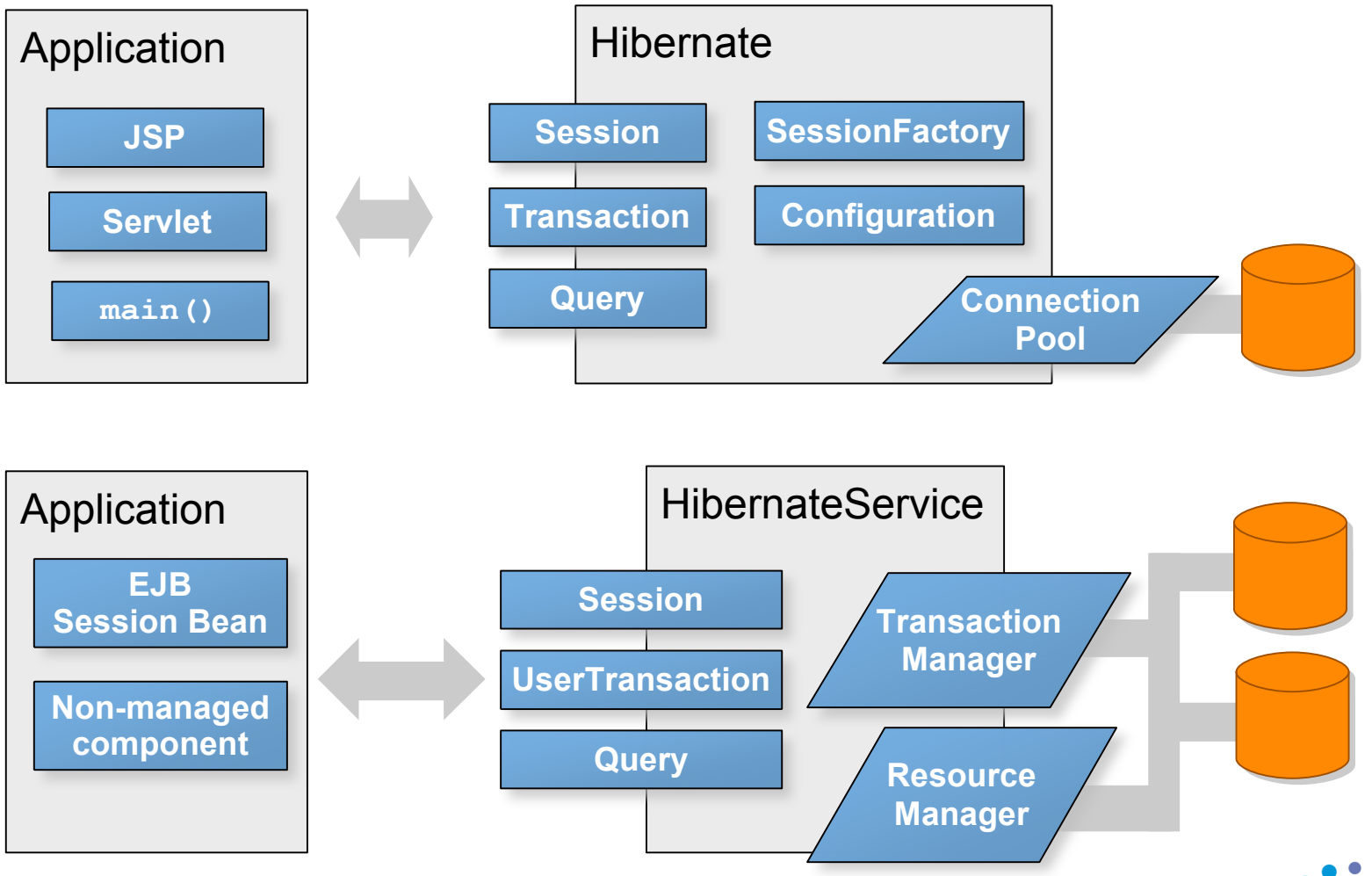
Hibernate query interfaces

```
Query q = session.createQuery("from Customer c where c.address.zipCode = :zip");
q.setParameter("zip", givenZipCode);
List result = q.list();

List result = session.createCriteria(Customer.class)
                .add( Expression.eq("zipCode", givenZipCode) )
                .add( Example.create( new Customer("David") ) )
                .list();
```

- ✓ Query - string-based in HQL or SQL
- ✓ Criteria - type-safe programmatic queries
- ✓ Support for query by example and native SQL (all SQL can be customized)

Hibernate deployment



Why Hibernate?

- Reduce application code by 30%
  - ✓ Less code is better maintainable
- Improve performance
  - ✓ Sophisticated performance optimizations, such as aggressive caching
- Advantage of relational technology
  - ✓ A key differentiator of Hibernate is its focus upon relational data access

Why Hibernate?

- Maturity
  - ✓ As the most-used ORM solution, Hibernate is also extremely well tested in a huge range of environments
- Support and Documentation
  - ✓ Professional support & training, active user community, and rich documentation
- Clear migration path to EJB 3.0

Hibernate Modules

- **Hibernate Core for Java**
  - ✓ Native CRUD/Query API, XML metadata
- **Hibernate Annotations**
  - ✓ JDK 5.0 annotations, EJB 3.0 compatible
- **Hibernate EntityManager**
  - ✓ EJB 3.0/Java Persistence API
- **NHibernate for .NET**

Hibernate 3.2

- Currently in alpha release
- Focus on
  - ✓ HQL/EJB-QL improvements (union, etc.)
  - ✓ Final EJB3/JPA compatibility
- Final release in Q2/2006

EJB 3.0

- JSR-220 is a two-part specification:
  - ✓ The new EJB programming model, radically simplified and improved for session beans, message driven beans, etc.
  - ✓ The new entity bean and *Java Persistence* interfaces, including object/relational mapping
- Strategy: Make things easier, use proven concepts that worked in practice
- All vendors are involved in writing the standard: Sun, Oracle, JBoss, BEA, IBM, ...

Stateless session bean

```
@Stateless
public class EditCustomerBean implements EditCustomer {

  @PersistenceContext
  EntityManager em;

  public Customer getCustomer(Long id) {
       return (Customer) em.find(Customer.class, id);
  }

  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void saveCustomer(Customer c) {
       em.merge(c);
  }
}
```

- ✓ Annotation-based metadata
- ✓ Dependency injection

Stateful session bean

```
@Stateful
public class CreateCustomerBean implements CreateCustomer {
  Customer c;
  ...
  public void addPersonalDetails(Map details) {
    c.addPersonalDetails(details);
  }

  public void addAddressDetails(Map details) {
    c.addAddressDetails(details);
  }

  @Remove
  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void completeConversation() {
    em.persist(c);
  }
}
```

- ✓ Conversation-aware components

Microcontainer deployment

```
public class JBossEJB3Microcontainer {

   @Configuration(groups = "integration.ejb3", beforeTest = true)
   public void startup(String deployDir) {
         EJB3StandaloneBootstrap.boot(null);

         // Deploy infrastructure (transaction manager, datasources, etc.)
         EJB3StandaloneBootstrap.deployXmlResource("META-INF/jboss-beans.xml");

         // Deploy all EJBs found in this directory (incl. scanning of JARs)
         EJB3StandaloneBootstrap.scanClasspath(deployDir);

         // Create JNDI context from jndi.properties
         initialContext = new InitialContext();
   }
}
```

- TestNG for functional/integration testing

Hibernate and EJB 3.0

- Hibernate is at the core of the JBoss EJB 3.0 application server (now in a preview release)
- Pluggable Java Persistence provider:

  Hibernate Core +

  Hibernate Annotations +

  Hibernate EntityManager
- Vendor-specific annotations and APIs for tuning and advanced use cases

Introducing JBoss Seam

What is Seam?

- An integration framework for JSF, EJB3, jBPM, JBoss AOP, …
- A *unified* programming model for components and event handling
- A uniform model for contextual dependency injection
- A declarative approach to context definition
- … in one sentence: "Seam enables you to write stateful applications."

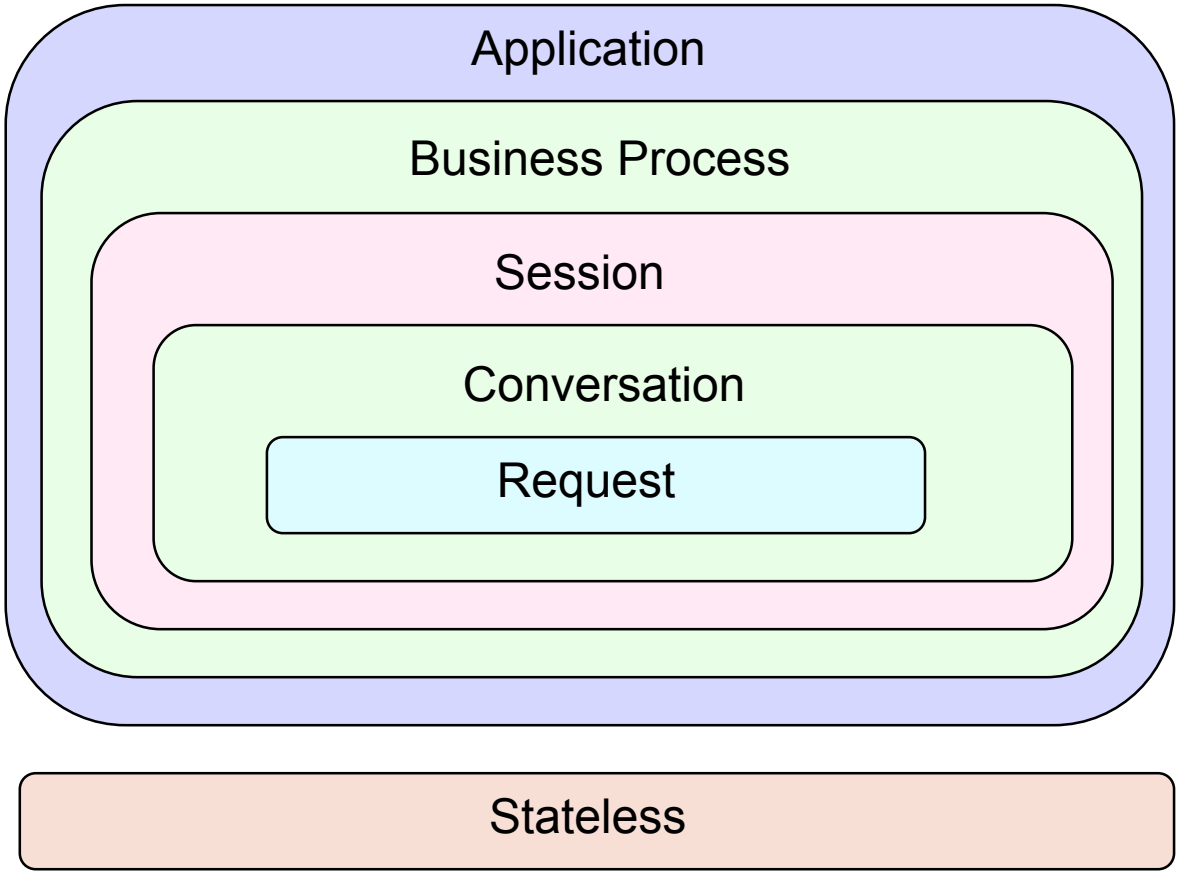
Seam components

- Any JavaBean
- In particular, any EJB3 session bean
- In particular, any JSF managed bean
- Many Seam components will be *both* EJBs and JSF managed beans *simultaneously*

Contexts

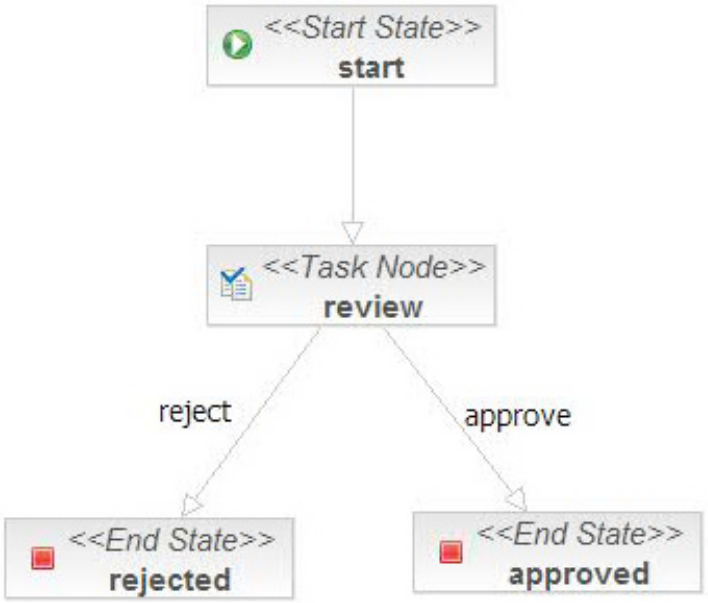
- There are many different contexts available in a J2EE application, some explicit, some implicit:
  - ✓ JSF request, Login session, Application context
  - ✓ Portlet session
  - ✓ EJB
    - A stateless context for stateless services (JNDI)
    - Transactional context for entities
  - ✓ jBPM (workflow)
    - Business process
    - Task
  - ✓ Application
    - User interaction (conversation)

Seam Context Hierarchy



jBPM process definition

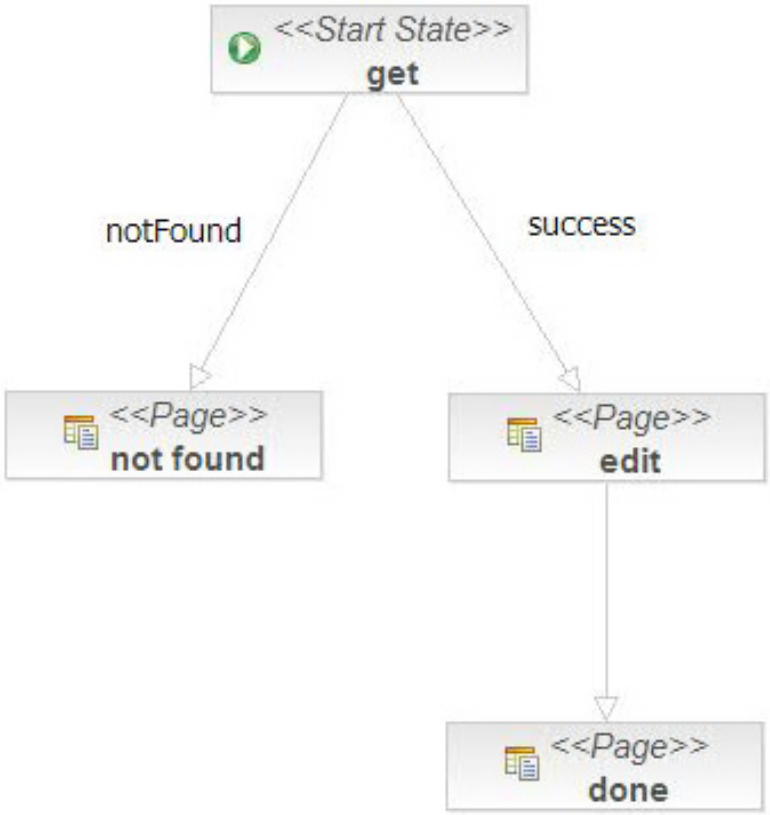
- JBoss Eclipse IDE



- saves XML process descriptor...

jBPM conversation pageflow

- JBoss Eclipse IDE saves XML descriptor...



'Edit' page in JSF and JSP

```
Welcome user '<h:outputText value="#{currentUser.loginName}“/>', please review:
<table>
   <tr>
      <td>Title:</td>
      <td><h:inputText value="#{documentEditor.title}“/></td>
   </tr>
   <tr>
      <td>Summary:</td>
      <td><h:inputText value="#{documentEditor.summary}“/></td>
   </tr>
   <tr>
      <td>Content:</td>
      <td><h:inputText value="#{documentEditor.content}"/></td>
   </tr>
</table>
<h:messages/>

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

Seam component as backing bean

```
@Stateful
@Name("documentEditor")
public EditDocumentBean implements EditDocument {
        @PersistenceContext
        private EntityManager em;

        @Valid
        private Document document;
        public Document getDocument() { return document; }

        @Begin(pageflow="review")
        public void get() {
          document = em.find(Document.class, id);
        }

        @Remove @End @IfInvalid(outcome=REDISPLAY)
        public void save(Document doc) {
          document = em.merge(doc);
        }
}
```

Default context for components

```
@Entity
@Name("currentUser")
@Scope(SESSION)
public User {

   @NotNull
   @Length(min=5, "Login name requires at least 5 characters.")
   private String loginName;
   public String getLoginName() { return loginName; }
   public void setLoginName(String ln) { loginName = ln; }
   ...
}
```

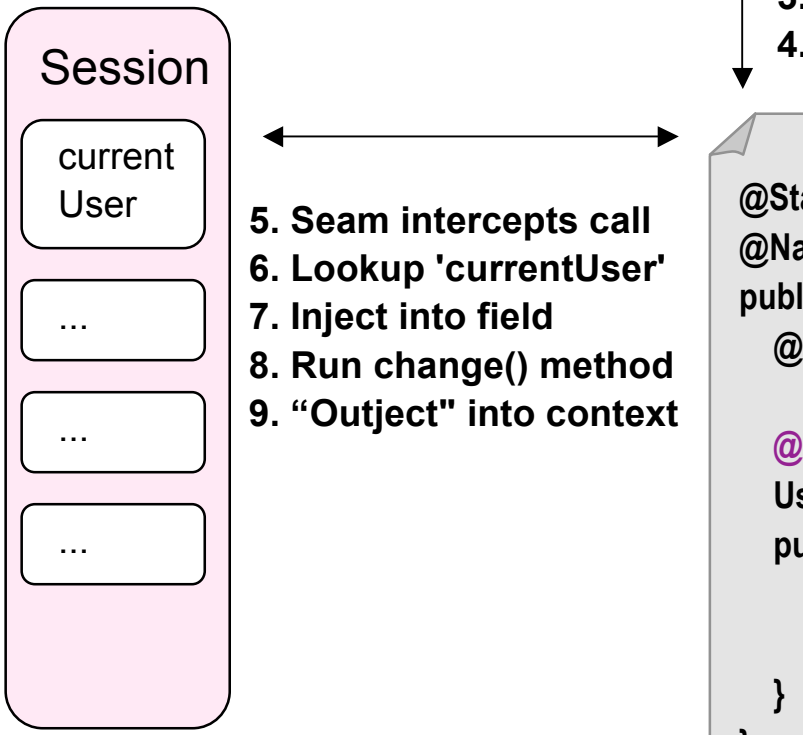
- When Seam tries to resolve the "currentUser" variable, it creates an instance of this component with binding to the SESSION context, if needed.

Dependency "Bijection"

```
<h:inputText value="#{currentUser.password}"/>
<h:commandButton type="submit" value="Save" action="#{changePassword.change}"/>
```

1. Lookup 'currentUser.password'

2. JSF: apply input to field on bean
3. JSF: lookup 'changePassword' component
4. JSF: call change() method



5. Seam intercepts call
6. Lookup 'currentUser'
7. Inject into field
8. Run change() method
9. “Outject" into context

```
@Stateless
@Name("changePassword")
public class ChangePassword {
   @PersistenceContext EntityManager em;

   @In @Out
   User currentUser;
   public String change() {
      currentUser = em.merge(currentUser);
      return "success"
   }
}
```

Features and road map

- Seam is much more...
  - ✓ no Hibernate LazyInitializationExceptions
  - ✓ BPM and workflow technology for the masses
  - ✓ deprecates so-called stateless architecture
  - ✓ runs in any EJB3 container
  - ✓ runs in Tomcat/TestNG JBoss EJB3 container
  - ✓ use Seam, JavaBeans and Hibernate, no EJB3
- Future of Seam
  - ✓ JBoss Seam 1.0 beta 2, final in early Q3
  - ✓ Seam for desktop applications/rich clients?
  - ✓ Check website for many demo applications!

Q & A

Hibernate

http://www.hibernate.org/

JBoss EJB3

http://www.jboss.com/products/ejb3

JBoss Seam

http://www.jboss.com/products/seam