



the
POWER
of
JAVA™

 **JBoss™** The Professional
Open Source Company



Introducing Seam

Gavin King

JBoss

gavin.king@jboss.com

<http://jboss.com/products/seam>

TS-3352

DEMO

Hibernate Tools

Java™ Platform, Enterprise Edition 5 Programming Model

- JavaServer™ Faces 1.2 technology
 - Template language
 - Extensible component model for widgets
 - “Managed bean” component model
 - JavaBeans™ specification with dependency injection
 - XML-based declaration
 - Session/request/application contexts
 - Defines interactions between the page and managed beans
 - Fine-grained event model (true MVC)
 - Phased request lifecycle
 - EL for binding controls to managed beans
 - XML-based “Navigation rules”
 - Ad hoc mapping from logical outcomes to URL

Java EE 5 Programming Model

- Enterprise JavaBeans™ (EJB™) 3.0 specification
 - 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

Surrogate Key Identifier Attribute

`@Entity`

```
public Document {
    @Id @GeneratedValue private Long id;
    private String title;
    private String summary;
    private String content;

    //getters and setters...
}
```

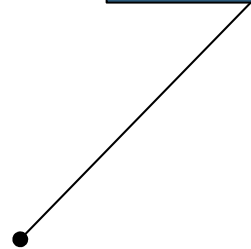
Search Page

```

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

```

A JSF-EL value binding



```

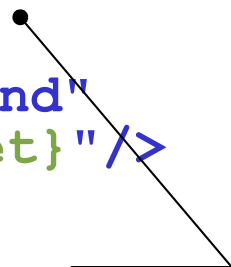
  <h:commandButton value="Find"
action="#{documentEditor.get}" />
</h:form>

```

A JSF control



A JSF-EL method binding



Edit Page

```

<h:form>
  <table>
    <tr>
      <td>Title</td>
      <td>
        <h:inputText value="#{documentEditor.title}">
          <f:validateLength maximum="100"/>
        </h:inputText>
      </td>
    </tr>
    <tr>
      <td>Real Name</td>
      <td>
        <h:inputText value="#{documentEditor.summary}">
          <f:validateLength maximum="1000"/>
        </h:inputText>
      </td>
    </tr>
    <tr>
      <td>Password</td>
      <td><h:inputText value="#{documentEditor.content}"/></td>
    </tr>
  </table>

  <div><h:messages/></div>

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

```



A JSF validator

We Could Use a Stateless Session Bean

`@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 the value bindings

Action listener methods bound to controls via the method bindings

JSF outcome

Declare the Managed Bean

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

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

This is a session-scoped component!

JavaServer Faces Technology

Navigation Rules

```

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

```

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

```

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

```

The outcome returned by the action listener method

Compared to J2EE™ Technology

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

Compared to J2EE Technology

- **Much simpler code**
 - Fewer artifacts (no DTO, for example)
 - Less noise (EJB specification boilerplate, Struts boilerplate)
 - More transparent (no direct calls to HttpSession, HttpRequest)
 - Much simpler ORM (even compared to Hibernate)
 - Finer grained components
- **Also more powerful for complex problems**
 - JavaServer Faces technology is amazingly flexible and extensible
 - EJB specification interceptors support a kind of “AOP lite”
 - Powerful ORM engine

Compared to J2EE Technology

- Much simpler code
 - Fewer artifacts (no DTO, for example)
 - Less noise (EJB specification boilerplate, Struts boilerplate)
 - More transparent (no direct calls to HttpSession, HttpRequest)
 - Much simpler ORM (even compared to Hibernate)
 - Finer grained components
- Also more powerful for complex problems
 - JavaServer Faces is amazingly flexible and extensible
 - EJB specification interceptors support a kind of “AOP lite”
 - Powerful ORM engine
- Unit testable
 - All these components (except the JavaServer Pages™ specifications) may be unit tested using JUnit or TestNG

Room for Improvement

- The managed bean is just noise—its concern is pure “glue”
 - And it accounts for more LOC than any other component!
 - It doesn't really decouple layers, in fact the code is more coupled than it would otherwise be

Room for Improvement

- The managed bean is just noise—its concern is pure “glue”
 - And it accounts for more LOC than any other component!
 - It doesn't really decouple layers, in fact the code is more coupled than it would otherwise be
- This code does not work in a multi-window application
 - And to make it work is a major architecture change!

Room for Improvement

- The managed bean is just noise—its concern is pure “glue”
 - And it accounts for more LOC than any other component!
 - It doesn't really decouple layers, in fact the code is more coupled than it would otherwise be
- This code does not work in a multi-window application
 - And to make it work is a major architecture change!
- 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!

Room for Improvement

- The managed bean is just noise—its concern is pure “glue”
 - And it accounts for more LOC than any other component!
 - It doesn't really decouple layers, in fact the code is more coupled than it would otherwise be
- This code does not work in a multi-window application
 - And to make it work is a major architecture change!
- 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?

Room for Improvement

- The managed bean is just noise—its concern is pure “glue”
 - And it accounts for more LOC than any other component!
 - It doesn't really decouple layers, in fact the code is more coupled than it would otherwise be
- This code does not work in a multi-window application
 - And to make it work is a major architecture change!
- 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?
- JavaServer Faces technology is still using XML where it should be using annotations

The Case for SFSB

- “Stateful session beans are unscalable”... why?
 - Replicating conversational state in a clustered environment (needed for transparent failover) is somewhat expensive

The Case for SFSB

- “Stateful session beans are unscalable”... why?
 - Replicating conversational state in a clustered environment (needed for transparent failover) is somewhat expensive
- Solution 1: keep all state in the database

The Case for SFSB

- “Stateful session beans are unscalable”... why?
 - Replicating conversational state in a clustered environment (needed for transparent failover) is somewhat expensive
- Solution 1: keep all state in the database
 - Traffic to/from database is even more expensive (database is the least scalable tier)
 - So, inevitably, end up needing a second-level cache
 - Second-level cache must be kept transactionally consistent between the database and every node on the cluster—even more expensive!

The Case for SFSB

- “Stateful session beans are unscalable”... why?
 - Replicating conversational state in a clustered environment (needed for transparent failover) is somewhat expensive
- Solution 1: keep all state in the database
 - Traffic to/from database is even more expensive (database is the least scalable tier)
 - So, inevitably, end up needing a second-level cache
 - Second-level cache must be kept transactionally consistent between the database and every node on the cluster—even more expensive!
- Solution 2: keep state in the HttpSession

The Case for SFSB

- “Stateful session beans are unscalable” ... why?
 - Replicating conversational state in a clustered environment (needed for transparent failover) is somewhat expensive
- Solution 1: keep all state in the database
 - Traffic to/from database is even more expensive (database is the least scalable tier)
 - So, inevitably, end up needing a second-level cache
 - Second-level cache must be kept transactionally consistent between the database and every node on the cluster—even more expensive!
- Solution 2: keep state in the HttpSession
 - Totally nuts, since HttpSession is exactly the same as a SFSB
 - But it does not have dirty-checking
 - And methods of a JavaBeans specification in the session can't be transactional

JBoss Seam

- Unify the two component models
 - Simplify Java EE 5 technology, filling a gap
 - Improve usability of JavaServer Faces technology

JBoss Seam

- Unify the two component models
 - Simplify Java EE 5 technology, filling a gap
 - Improve usability of JavaServer Faces technology
- Integrate jBPM
 - BPM technology for the masses

JBoss Seam

- **Unify the two component models**
 - Simplify Java EE 5 technology, filling a gap
 - Improve usability of JavaServer Faces technology
- **Integrate jBPM**
 - BPM technology for the masses
- **Deprecate so-called stateless architecture**
 - Managed application state—more robust, more performant, richer user experience
 - Take advantage of recent advances in clustering technology

JBoss Seam

- Unify the two component models
 - Simplify Java EE 5 technology, filling a gap
 - Improve usability of JavaServer Faces technology
- Integrate jBPM
 - BPM technology for the masses
- Deprecate so-called stateless architecture
 - Managed application state—more robust, more performant, richer user experience
 - Take advantage of recent advances in clustering technology
- Decouple the technology from the execution environment
 - Run EJB 3-based apps in Tomcat
 - Or in TestNG
 - Use Seam with JavaBeans specifications and Hibernate

JBoss Seam

- Unify the two component models
 - Simplify Java EE 5 technology, filling a gap
 - Improve usability of JavaServer Faces technology
- Integrate jBPM
 - BPM technology for the masses
- Deprecate so-called stateless architecture
 - Managed application state—more robust, more performant, richer user experience
 - Take advantage of recent advances in clustering technology
- Decouple the technology from the execution environment
 - Run EJB 3-based apps in Tomcat
 - Or in TestNG
 - Use Seam with JavaBeans specifications and Hibernate
- Enable richer user experience

Contextual Components

- Most of the problems relate directly or indirectly to state management
 - The contexts defined by the servlet spec are not meaningful in terms of the application
 - EJB technology itself has no strong model of state management
 - We need a richer context model that includes “logical” contexts that are meaningful to the application

Contextual Components

- Most of the problems relate directly or indirectly to state management
 - The contexts defined by the servlet spec are not meaningful in terms of the application
 - EJB technology itself has no strong model of state management
 - We need a richer context model that includes “logical” contexts that are meaningful to the application
- We also need to fix the mismatch between the JavaServer Faces technology and EJB 3.0-based component models
 - We should be able to use annotations everywhere
 - An EJB specification should be able to be a JavaServer Faces-based managed bean (and vice versa)

Contextual Components

- Most of the problems relate directly or indirectly to state management
 - The contexts defined by the servlet spec are not meaningful in terms of the application
 - EJB technology itself has no strong model of state management
 - We need a richer context model that includes “logical” contexts that are meaningful to the application
- We also need to fix the mismatch between the JavaServer Faces technology and EJB 3.0-based component models
 - We should be able to use annotations everywhere
 - An EJB specification should be able to be a JavaServer Faces-based managed bean (and vice versa)
- It makes sense to think of binding EJB-based components directly to the JavaServer Faces-based view
 - A session bean acts just like a backing bean, providing event listener methods, etc.
 - The entity bean provides data to the form, and accepts user input

Slight Change to the Edit Page

```

<h:form>
  <table>
    <tr>
      <td>Title</td>
      <td>
        <h:inputText value="#{documentEditor.document.title}">
          <f:validateLength maximum="100"/>
        </h:inputText>
      </td>
    </tr>
    <tr>
      <td>Real Name</td>
      <td>
        <h:inputText value="#{documentEditor.document.summary}">
          <f:validateLength maximum="1000"/>
        </h:inputText>
      </td>
    </tr>
    <tr>
      <td>Password</td>
      <td><h:inputText value="#{documentEditor.document.content}"/></td>
    </tr>
  </table>

  <h:messages/>

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

```

Bind view to the entity bean directly

Our First Seam Component

`@Stateful`

`@Name("documentEditor")`

```
public EditDocumentBean implements EditDocument {
```

```
    @PersistenceContext
```

```
    private EntityManager em;
```

```
    private Long id;
```

```
    public void setId(Long id) { this.id = id; }
```

```
    private Document document;
```

```
    public Document getDocument() { return document; }
```

`@Begin`

```
    public String get() {
```

```
        document = em.find(Document.class, id);
```

```
        return document==null ? "notFound" : "success";
```

```
    }
```

`@End`

```
    public String save(Document doc) {
```

```
        document = em.merge(doc);
```

```
        return "success";
```

```
    }
```

The `@Name` annotation binds the component to a contextual variable—it's just like `<managed-bean-name>` in the JSF XML

The `@Begin` annotation defines the beginning of a logical scope—it starts a *conversation*

The `@End` annotation ends the conversation—a conversation can also end by being timed out

The Seam Context Model

- Seam defines a rich context model for stateful components, enabling container-management of application state
- The contexts are:
 - **EVENT**
 - **PAGE**
 - **CONVERSATION**
 - **SESSION**
 - **PROCESS**
 - **APPLICATION**
- Components are assigned to a scope using the **@Scope** annotation
- The highlighted “logical” contexts are demarcated by the application itself
 - For now, this is always done with annotations like **@Begin**, **@End**, **@BeginProcess**, **@BeginTask**

DEMO

Seam Hotel Booking

Conversations

- Conversations are not that exciting until you really start thinking about them:
 - Multi-window operation
 - Back button support
 - “Workspace management”
- Nested conversations
 - Multiple concurrent inner conversations within an outer conversation
 - A stack of continuable states
- How is state stored between requests?
 - Server-side conversations (HttpSession + conversation timeout)
 - Client-side conversations (serialize into the page)
 - Business process state is made persistent by jBPM

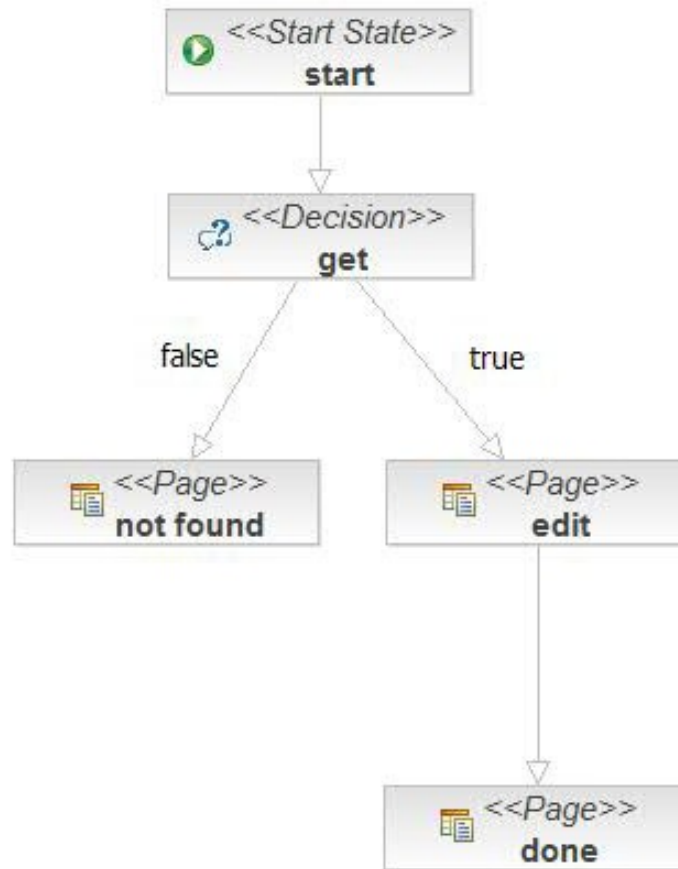
DEMO

Seam Issue Tracker

Pageflow

- Two models for conversational pageflow
 - The stateless model: JavaServer Faces technology navigation rules
 - Ad hoc navigation (the app must handle backbutton)
 - Actions tied to UI widgets
 - The stateful model: jBPM pageflow
 - No ad hoc navigation (back button bypassed)
 - Actions tied to UI widgets **or called directly from pageflow transitions**
- Simple applications only need the stateless model
- Some applications need both models

jBPM Pageflow Definition

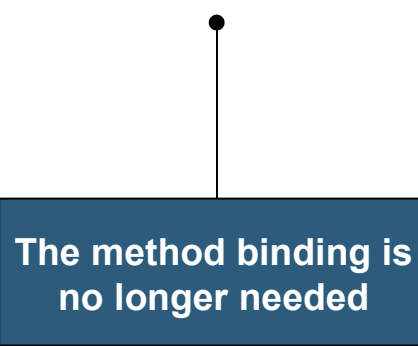


Search Page

```

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

  <h:commandButton value="Find" />
</h:form>
  
```



The method binding is no longer needed

jBPM Pageflow Definition

```

<pageflow-definition name="editDocument">

  <start-page name="start" page="/search.jsp">
    <transition to="get">
      <action expression="#{documentEditor.get}"/>
    </transition>
  </start-page>

  <decision name="get" expression="#{documentEditor.found}">
    <transition name="false" to="not found"/>
    <transition name="true" to="edit"/>
  </decision>

  <page name="not found" page="/notFound.jsp">
    <end-conversation/>
  </page>

  <page name="edit" page="/editDocument.jsp">
    <transition to="done">
      <action expression="#{documentEditor.save}"/>
    </transition>
  </page>

  <page name="done" page="/findDocument.jsp">
    <end-conversation/>
  </page>

</pageflow-definition>

```

A jBPM state transition action, instead of a JSF action listener

A jBPM decision node, instead of a JSF navigation rule

Each <page> node is a jBPM wait state – the pageflow “waits” for user input

No Process Logic in Business Logic!

`@Stateful`

`@Name("documentEditor")`

```
public EditDocumentBean implements EditDocument {
    @PersistenceContext private EntityManager em;
    private Long id;
    public void setId(Long id) { this.id = id; }

    private Document document;
    public Document getDocument() { return document; }
```

When the component is first created, the pageflow execution begins

`@Create @Begin(pageflow="editDocument")`

```
public void start() {}
```

```
public void get() {
    document = em.find(Document.class, id);
}
```

Notice that the outcomes have disappeared from the component code

```
public boolean isFound() {
    return document != null;
}
```

`@End`

```
public void save(Document doc) {
    document = em.merge(doc);
}
```

DEMO

Seam DVD Store (1)

What About the Business Process?

- Different from a conversation
 - Long-running (persistent)
 - Multi-user
 - (The lifespan of a business process instance is longer than the process definition!)
- A conversation that is significant in terms of the overarching business process is called a “task”
 - Driven from the jBPM task list screen
- We demarcate work done in a task using `@BeginTask` / `@ResumeTask` and `@EndTask`
- Work done in the scope of a task also has access to the `PROCESS` scope
 - In addition to the task’s `CONVERSATION` scope

Start a Business Process

```
@Name ("documentSubmission")
```

```
@Stateful
```

```
public class DocumentSubmissionBean implements DocumentSubmission {
```

```
    @PersistenceContext entityManager;
```

```
    @Out(scope=PROCESS) Long documentId;
```

```
    private Document document;
```

```
    //some conversation ...
```

```
    @CreateProcess(definition="DocumentSubmission")
```

```
    public String submitDocument() {
```

```
        documentId = document.getId();
```

```
        return "submitted";
```

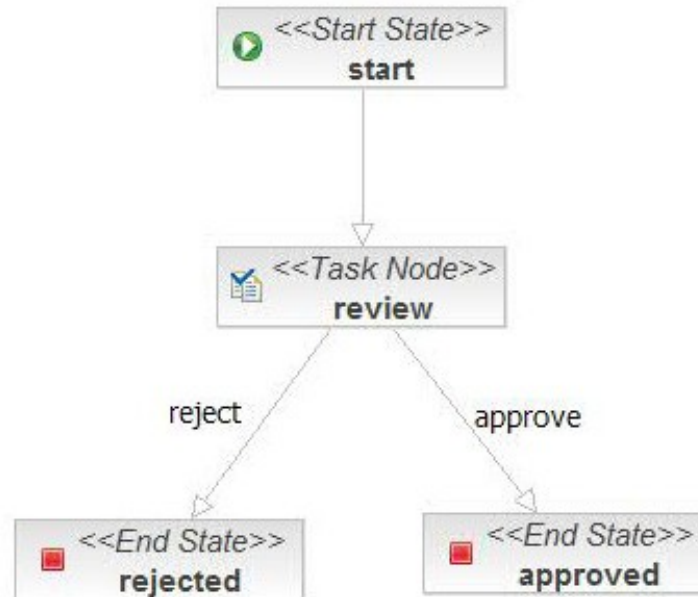
```
    }
```

```
}
```

Output documentId to the business process context

Create a new business process instance

jBPM Process Definition



jBPM Process Definition

```
<process-definition name="DocumentSubmission">

  <start-state name="start">
    <transition to="review"/>
  </start-state>

  <task-node name="review">

    <task name="review">
      <assignment actorId="#{user.manager.id}" />
    </task>

    <transition name="approve" to="approved">
      <action expression="#{email.sendApprovalEmail}"/>
    </transition>

    <transition name="reject" to="rejected"/>

  </task-node>

  <end-state name="approved"/>
  <end-state name="rejected"/>

</process-definition>
```

In this case, the wait states are <task> nodes, where the process execution waits for the user to begin work on a task

A jBPM *task assignment*, via EL evaluated in the Seam contexts

Perform the Task

```
@Name ("reviewDocument")
```

```
@Stateful
```

```
public class ReviewDocumentBean implements ReviewDocument {
```

```
    @PersistenceContext entityManager;
```

```
    @In Long documentId;
```

```
    @Out Document document;
```

documentId injected from
business process context

```
@BeginTask
```

```
public String getDocument() {
```

```
    document = entityManager.find(Document.class, documentId);
```

```
    return "reviewDocument";
```

```
}
```

End the task, specifying
a transition name

```
@EndTask (transition="approve")
```

```
public String approve() { return "documentApproved"; }
```

```
@EndTask (transition="reject")
```

```
public String approve() { return "documentRejected"; }
```

```
}
```

DEMO

- Seam DVD Store (2)

What About Dependency Injection?

- Dependency injection is broken for stateful components
 - A contextual variable can be written to, as well as read!
 - Its value changes over time
 - A component in a wider scope must be able to have a reference to a component in a narrower scope
- Dependency injection was designed with J2EE technology-style **stateless services** in mind—just look at that word “dependency”
 - It is usually implemented in a static, unidirectional, and non-contextual way
- For stateful components, we need **bijection**
 - 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**

Bijection

```

@Stateless
@Name("changePassword")
public class ChangePasswordBean implements Login {
    @PersistenceContext
    private EntityManager em;

    @In @Out
    private User currentUser;

    public String changePassword() {
        currentUser = em.merge(currentUser);
    }
}

```

The `@In` annotation injects the value of the contextual variable named `currentUser` into the instance variable each time the component is invoked

The `@Out` annotation “outjects” the value of the instance variable back to the `currentUser` contextual variable at the end of the invocation

Conversations and Persistence

- The notion of persistence context is central to ORM
 - A canonicalization of pk—Java-based instance
 - Without it, you lose referential integrity
 - It is also a natural cache
- A process-scoped persistence context is evil
 - Requires in-memory locking and sophisticated deadlock detection
- A transaction-scoped persistence context has problems if you re-use objects across transactions
 - LazyInitializationException navigating lazy associations
 - NonUniqueObjectException reassociating detached instances
 - Less opportunity for caching (workaround: use a second-level cache, which is quite unscalable)
- EJB 3 specification-style component-scoped persistence context is nice, but...
 - Not held open for entire request (while rendering view)
 - Problems propagating across components
- Solution: conversation-scoped persistence contexts
 - Much, much better than well-known “open session in view” pattern!

DEMO

Seam Remoting

Roadmap

- Seam 1.0 CR 1 out now
 - JSR 168 Portlet Specification
 - Seam Remoting
 - i18n enhancements
- Seam 1.0 final in May
- Seam 1.1 in Q3
 - Asynchronicity/Calendar
 - JBoss Rules integration
- Seam 1.5 in Q4
 - Seam for SOA/ESB
- Future
 - Seam for rich clients?

Q&A

DEMO



the
POWER
of
JAVA™

 **JBoss™** The Professional
Open Source Company



Introducing Seam

Gavin King

JBoss

gavin.king@jboss.com

<http://jboss.com/products/seam>

TS-3352