Developing Applications With Seam and Eclipse

Gavin King

JBoss, A Division of Red Hat

gavin@hibernate.org

TS-4092

My Dilemma

- If I talk about the really unique stuff in Seam, everyone thinks it must be complicated and difficult to use

My Dilemma

- If I talk about the really unique stuff in Seam, everyone thinks it must be complicated and difficult to use
- If I show how easy it is to use, people think it's just another action/CRUD framework

My Dilemma

- If I talk about the really unique stuff in Seam, everyone thinks it must be complicated and difficult to use
- If I show how easy it is to use, people think it's just another action/CRUD framework
- Usually I go down the first path, because at least that way the presentation is interesting

My Dilemma

- If I talk about the really unique stuff in Seam, everyone thinks it must be complicated and difficult to use
- If I show how easy it is to use, people think it's just another action/CRUD framework
- Usually I go down the first path, because at least that way the presentation is interesting
- Today I'm going to try the second path, and risk being boring

My Dilemma

- If I talk about the really unique stuff in Seam, everyone thinks it must be complicated and difficult to use
- If I show how easy it is to use, people think it's just another action/CRUD framework
- Usually I go down the first path, because at least that way the presentation is interesting
- Today I'm going to try the second path, and risk being boring
- …because I'm sick to death of hearing people who think that Java technology has to be complicated, and has to involve reams and reams of XML files, and has to mean waiting around for the server to restart for hours each day

Getting Started

- After the first day of development, I want to have something to show the users of my system

Getting Started

- After the first day of development, I want to have something to show the users of my system
- Most Java projects spend weeks arguing about frameworks, project structures, architecture, programming in PowerPoint (ugh) or rational rose (worse), writing build scripts, etc.

Getting Started

- After the first day of development, I want to have something to show the users of my system
- Most Java projects spend weeks arguing about frameworks, project structures, architecture, programming in PowerPoint (ugh) or rational rose (worse), writing build scripts, etc.
- The user doesn't care about this sh*t

Getting Started

- After the first day of development, I want to have something to show the users of my system
- Most Java projects spend weeks arguing about frameworks, project structures, architecture, programming in PowerPoint (ugh) or rational rose (worse), writing build scripts, etc.
- The user doesn't care about this sh*t
- I don't have to be an obsessive unit-testing-pair-programming-no-overtime-collective-code-ownership-time-boxed-releases fetishist to think that it's good to spend time on sh*t that users care about

Getting Started

- seam-gen can get you started in minutes…

What Was Generated?

- A basic project skeleton
- With an Ant build script
- That deploys to a WAR or exploded directory
- With support for persistence via JPA
  - With test, dev, and prod database profiles
- Basic login/logout
- A Facelets template
  - What if we don't like it?
- And a welcome page
  - What if we don't like the welcome message?

Let's Create Our Own Page

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                template="layout/template.xhtml">

<ui:define name="body">
      Hello!
</ui:define>

</ui:composition>
```

Now for Some Dynamic Content…

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                template="layout/template.xhtml">

<ui:define name="body">
      Hello!
      The time is #{currentTime}.
</ui:define>

</ui:composition>
```

Using JSF Components

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:s="http://jboss.com/products/seam/taglib"
                template="layout/template.xhtml">

<ui:define name="body">
   <rich:panelBar>
      <rich:panelBarItem label="hello">Hello!</rich:panelBarItem>
      <rich:panelBarItem label="time">
          The time is
          
         <h:outputText value="#{currentTime}">
            <s:convertDateTime type="time" timeStyle="long"/>
         </h:outputText>.
      </rich:panelBarItem>
      <rich:panelBarItem label="nothing">nothing to see here</rich:panelBarItem>
   </rich:panelBar>
</ui:define>

</ui:composition>
```

What Do Actions Look Like?

- It's time to write some Java code!

What Do Actions Look Like?

- It's time to write some Java code!
- There are no "actions" in Seam

What Do Actions Look Like?

- It's time to write some Java code!
- There are no "actions" in Seam
- Or, if you prefer, **all** Seam components are "actions"
  - Seam-gen pretends that there is something called an "action", just to make it easy for Struts and Ruby on Rails guys

What Do Actions Look Like?

- It's time to write some Java code!
- There are no "actions" in Seam
- Or, if you prefer, **all** Seam components are "actions"
  - Seam-gen pretends that there is something called an "action", just to make it easy for Struts and Ruby on Rails guys
- A traditional action framework binds a class to a URL, and request parameters to attributes, unidirectionally—JavaServer™ Faces lets you bind buttons to methods and fields to attributes, bidirectionally
  - Let's make an "action" now…

What Was Generated?

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                ...
                template="layout/template.xhtml">

<ui:define name="body">

   <h:messages globalOnly="true" styleClass="message"/>

   <rich:panel label="ping">

      <h:form id="pingForm">
         <h:commandButton id="ping" value="ping!"
                          action="#{ping.ping}"/>
      </h:form>

   </rich:panel>

</ui:define>

</ui:composition>
```

What Was Generated?

```
@Name("ping")
public class Ping {

    @Logger private Log log;

    @In FacesMessages facesMessages;

    public void ping()
    {
        log.info("ping.ping() action called");
        facesMessages.add("ping");
    }

}
```

What Was Generated?

```
public class PingTest extends SeamTest {

    @Test
    public void test() throws Exception {

      new FacesRequest() {
            @Override
            protected void invokeApplication() {
                  invokeMethod("#{ping.ping}");
            }
      }.run();

    }
}
```

Not a Unit Test!

- Unit tests test a component in isolation (supposedly)

Not a Unit Test!

- Unit tests test a component in isolation (supposedly)
- Hence they are next to useless

Not a Unit Test!

- Unit tests test a component in isolation (supposedly)
- Hence they are next to useless
- Useful tests test the interaction between a component and its collaborators and container environment
  - Please, please stop writing StringUtilTest classes

Not a Unit Test!

- Unit tests test a component in isolation (supposedly)
- Hence they are next to useless
- Useful tests test the interaction between a component and its collaborators and container environment
  - Please, please stop writing StringUtilTest classes
- Or they test that the external behavior of the system is correct, from the point of view of the user

Debugging

- Debugging is easy!
  - Let's try debugging the test case…
  - Let's try debugging the running app…
  - Now, let's meet the Seam debug page…

Using Seam Security

```
@Name("ping")
public class Ping {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   @Restrict("#{identity.loggedIn}")
   public void ping()
   {
      log.info("ping.ping() action called");
      facesMessages.add("pong!");
   }

}
```

Using EL in the Action

```
@Name("ping")
public class Ping {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   @Restrict("#{identity.loggedIn}")
   public void ping()
   {
      log.info("ping.ping() action called by #{identity.username}");
      facesMessages.add("pong to #{identity.username}");
   }

}
```

Calling the Action by Ajax

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                ...
                template="layout/template.xhtml">
<ui:define name="body">

   <s:span id="msg">
      <h:messages globalOnly="true" styleClass="message"/>
   </s:span>

   <rich:panel>
      <f:facet name="header">ping</f:facet>

      <h:form id="pingForm">

         <a:commandButton id="ping" value="ping!"
                          action="#{ping.ping}" reRender="msg"/>
      </h:form>

   </rich:panel>

</ui:define>

</ui:composition>
```

Drag and Drop

```
<s:span id="msg">
   <h:messages id="msg" globalOnly="true" styleClass="message"/>
</s:span>

<rich:panel label="ping">

   <h:form id="pingForm">

      <rich:panel>
            <rich:dragSupport dragType="text"/>
            drag me!
      </rich:panel>

      <rich:panel>
            <rich:dropSupport id="drop" acceptedTypes="text"
                        dropListener="#{ping.ping}" reRender="msg"/>
        drop me!
      </rich:panel>

      ...
   </h:form>

</rich:panel>
```

Navigation From an Action

```
@Name("ping")
public class Ping {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   @Restrict("#{identity.loggedIn}")
   public String ping()
   {
      log.info("ping.ping() action called by #{identity.username}");
      facesMessages.add("pong to #{identity.username}");
      return "/home.xhtml";
   }

}
```

Using #xternal Navigation Rules

```
<!DOCTYPE page PUBLIC
        "-//JBoss/Seam Pages Configuration DTD 1.2//EN"
        "http://jboss.com/products/seam/pages-1.2.dtd">

<page>
      <navigation from-action="#{ping.ping}">
        <redirect view-id="/home.xhtml"/>
      </navigation>
</page>
```

“One Kind of Stuff”

- A single component model that unifies JavaServer Faces, Enterprise JavaBeans™ (EJB™) 3, BPM, WS, ESB…
- A single rendering technology for HTML, XML, email, PDF…
- A single EL for binding rendering, orchestration, configuration to the component model

Sending an Email

```
<m:message xmlns="http://www.w3.org/1999/xhtml"
        xmlns:m="http://jboss.com/products/seam/mail">

   <m:from name="Seam" address="seam@hibernate.org" />
   <m:to name="#{identity.username}">#{identity.username}@hibernate.org</m:to>
   <m:subject>Hello...</m:subject>
   <m:body>
       Hello #{identity.username}!
   </m:body>
</m:message>
```

Sending an Email

```
@Name("ping")
public class Ping {

   @Logger private Log log;

   @In FacesMessages facesMessages;
   @In Renderer renderer;

   @Restrict("#{identity.loggedIn}")
   public void ping()
   {
      log.info("ping.ping() action called by #{identity.username}");
      facesMessages.add("pong to #{identity.username}");
      renderer.render("/email.xhtml");
   }

}
```

What About Persistence?

- Remember how we said there are no actions in Seam?

What About Persistence?

- Remember how we said there are no actions in Seam?
- There are no DAOs either
  - At least not in the traditional sense

What About Persistence?

- Remember how we said there are no actions in Seam?
- There are no DAOs either
  - At least not in the traditional sense
- In Seam it is common to bind components which access the database via JPA/Hibernate—and the entities themselves—directly to the view

What About Persistence?

- Remember how we said there are no actions in Seam?
- There are no DAOs either
  - At least not in the traditional sense
- In Seam it is common to bind components which access the database via JPA/Hibernate—and the entities themselves—directly to the view
- Seam solves a bunch of persistence-related problems (LazyInitializationException, etc.) by providing persistence context management and transaction management that is aware of the boundaries of the user interaction
  - A conversation-scoped persistence context
  - We havn't met conversations yet, so let's start with something simple…

The Entity Bean

```
@Entity
public class User {

    @Id
    private String username;
    private String name;

    public String getName() {
      return name;
    }
    public void setName(String name) {
      this.name = name;
    }
    public String getUsername() {
      return username;
    }
    public void setUsername(String username) {
      this.username = username;
    }

}
```

Using a Seam Persistence Context

```
@Name("ping")
public class Ping {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   @In EntityManager entityManager;

   @Restrict("#{identity.loggedIn}")
   public void ping()
   {
      User user = (User) entityManager
         .createQuery("from User where username = #{identity.username}")
         .getSingleResult();

      log.info("ping.ping() action called by #{identity.username}");
      facesMessages.add("pong to #0", user.getName());
   }

}
```

What About Form Processing?

- This is the one thing JavaServer Faces does **really** well!
  - Of course, we'll use seam-gen to get us started…

What Was Generated?

```
<h:form id="adderForm">

    <rich:panel>
      <f:facet name="header">adder</f:facet>

      <s:decorate id="valueDecoration" template="layout/edit.xhtml">
        <ui:define name="label">Value</ui:define>
        <h:inputText id="value" required="true"
                     value="#{adder.value}"/>
      </s:decorate>

      <div style="clear:both"/>

    </rich:panel>

    <div class="actionButtons">
      <h:commandButton id="adder" value="adder"
            action="#{adder.adder}"/>
    </div>

</h:form>
```

What Was Generated?

```
@Name("adder")
public class Adder {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   private String value;

   public void add()
   {
      log.info("adder.add() action called with: #{adder.value}");
      facesMessages.add("add #{adder.value}");
   }

   //get/set pair for value...

}
```

What Was Generated?

```
public class AdderTest extends SeamTest {
      @Test
      public void test() throws Exception {
        new FacesRequest() {
              @Override
              protected void updateModelValues() throws Exception {
              setValue("#{adder.value}", "seam");
          }
              @Override
              protected void invokeApplication() {
                    invokeMethod("#{adder.add}");
              }
              @Override
              protected void renderResponse() {
              assert getValue("#{adder.value}").equals("seam");
              }
        }.run();
      }
}
```

Implementing an “adder”

```
@Name("adder")
public class Adder {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   private BigDecimal value;
   private BigDecimal total = BigDecimal.ZERO;

   public void add()
   {
      total = total.add(value);
      log.info("adder.add() action called with: #{adder.value}");
      facesMessages.add("total: #{adder.total}");
   }

   //get/set pair for value and total...

}
```

It Doesn't Work!

- Our component doesn't "remember" the total across multiple requests

It Doesn't Work!

- Our component doesn't "remember" the total across multiple requests
- We could keep state in the http session
  - But what if we wanted to be keeping multiple totals at the same time?

It Doesn't Work!

- Our component doesn't "remember" the total across multiple requests
- We could keep state in the http session
  - But what if we wanted to be keeping multiple totals at the same time?
- Let's try using a **conversation**
  - Finally, this is our first baby-step into the world of Seam…

Implementing an “adder”

```
@Name("adder")
@Scope(ScopeType.CONVERSATION)
public class Adder {

   @Logger private Log log;

   @In FacesMessages facesMessages;

   private BigDecimal value;
   private BigDecimal total = BigDecimal.ZERO;

   @Begin(join=true)
   public void add()
   {
      total = total.add(value);
      log.info("adder.add() action called with: #{adder.value}");
      facesMessages.add("total: #{adder.total}");
   }

   //get/set pair for value and total...

}
```

Validation

- The word “validation” implies a process that belongs in the user interface

Validation

- The word “validation” implies a process that belongs in the user interface
- But most “validation” enforces constraints that exist all the way to the database
  - It is an aspect of the data model

Validation

- The word “validation” implies a process that belongs in the user interface
- But most “validation” enforces constraints that exist all the way to the database
  - It is an aspect of the data model
- Hibernate Validator provides model-based constraints
  - These are enforced by Seam at the user interface
  - And by Hibernate, before writing to the database
  - You can even ask Hibernate to generate DDL that includes the constraint

Validation

- The word “validation” implies a process that belongs in the user interface
- But most “validation” enforces constraints that exist all the way to the database
  - It is an aspect of the data model
- Hibernate Validator provides model-based constraints
  - These are enforced by Seam at the user interface
  - And by Hibernate, before writing to the database
  - You can even ask Hibernate to generate DDL that includes the constraint
- There are some nice built-in constraints
  - But we want higher-level semantics for our constraints…

Adding Validation— The Model

```
@Name("adder")
@Scope(ScopeType.CONVERSATION)
public class Adder {

    ...

    @Money
    public BigDecimal getValue()
    {
        return value;
    }

}
```

Adding Validation—The Annotation

```
@Retention(RetentionPolicy.RUNTIME)
@ValidatorClass(MoneyValidator.class)
public @interface Money {
      String message() default "please enter an amount in dollars";
}
```

Adding Validation—The Validator

```
public class MoneyValidator implements Validator<Money> {

    public void initialize(Money money) {}

    public boolean isValid(Object value)
    {
      return ((BigDecimal) value).scale() <= 2;
    }

}
```

Adding Validation—Customizing the Message

```
@Name("adder")
@Scope(ScopeType.CONVERSATION)
public class Adder {

   ...

   @Money(message="you can only add dollar amounts")
   public BigDecimal getValue()
   {
      return value;
   }

}
```

Adding Interactive Validation

```
<s:div id="total">Total: #{adder.total}</s:div>

<h:form id="adderForm">
      <rich:panel>
        <f:facet name="header">adder</f:facet>

        <s:decorate id="valueDecoration" template="layout/edit.xhtml">
          <ui:define name="label">Value</ui:define>
          <h:inputText id="value" required="true"
                       value="#{adder.value}">
            <a:support event="onblur" reRender="valueDecoration"/>
          </h:inputText>
        </s:decorate>

        <div style="clear:both"/>

      </rich:panel>

      <div class="actionButtons">
        <a:commandButton id="adder" value="adder"
              action="#{adder.add}" reRender="total,valueDecoration"/>
      </div>

</h:form>
```

Why Ajax Needs Conversations

- Ajax changes the interaction model of the web

Why Ajax Needs Conversations

- Ajax changes the interaction model of the web
- From few, coarse-grained, requests

Why Ajax Needs Conversations

- Ajax changes the interaction model of the web
- From few, coarse-grained, requests
- To many, fine-grained, requests

Why Ajax Needs Conversations

- Ajax changes the interaction model of the web
- From few, coarse-grained, requests
- To many, fine-grained, requests
- This is going to kill your database if you don't have somewhere to keep conversational state
  - You can keep **some** state in the client, but there is plenty you can't keep there
  - For example, you don't want to serialize persistence contexts back and forth, if you can possibly avoid it

Why Ajax Needs Conversations

- Ajax changes the interaction model of the web
- From few, coarse-grained, requests
- To many, fine-grained, requests
- This is going to kill your database if you don't have somewhere to keep conversational state
  - You can keep **some** state in the client, but there is plenty you can't keep there
  - For example, you don't want to serialize persistence contexts back and forth, if you can possibly avoid it
- With asynchronicity comes concurrency
  - The servlet engine offers no construct for managing concurrent access to state held in the web tier
  - Seam solves this problem by serializing requests for a particular conversation, and by serializing access to session-scoped components

Conversations and Bookmarks

- Conversations are a fantastic feature, but they are all about server-side state

Conversations and Bookmarks

- Conversations are a fantastic feature, but they are all about server-side state
- Usually, this is the most efficient place to keep state
  - “Shared nothing” architectures simply don’t scale, no matter how long you all spend sitting around in circles telling each other that they do

Conversations and Bookmarks

- Conversations are a fantastic feature, but they are all about server-side state
- Usually, this is the most efficient place to keep state
  - “Shared nothing” architectures simply don’t scale, no matter how long you all spend sitting around in circles telling each other that they do
- The disadvantage of keeping state on the server is that you can’t bookmark it, or send a link in an email

Conversations and Bookmarks

- Conversations are a fantastic feature, but they are all about server-side state
- Usually, this is the most efficient place to keep state
  - “Shared nothing” architectures simply don’t scale, no matter how long you all spend sitting around in circles telling each other that they do
- The disadvantage of keeping state on the server is that you can’t bookmark it, or send a link in an email
- Seam provides page parameters, a bidirectional binding from request parameters to the model
  - So they are automagically propagated across links, redirects, etc.
  - And the model doesn’t need to know anything about them

Using Page Parameters

```
<!DOCTYPE page PUBLIC
        "-//JBoss/Seam Pages Configuration DTD 1.2//EN"
        "http://jboss.com/products/seam/pages-1.2.dtd">

<page>
   <param name="total" value="#{badder.total}"/>
   <navigation from-action="#{badder.add}">
     <redirect/>
   </navigation>
</page>
```

Finale

- The common case in enterprise development is that we have an existing database
  - If we do, seam-gen can give you basic CRUD, virtually for free…

What Was Generated?

```
@Entity
@Table(name = "User")
public class User implements java.io.Serializable {

        private String username;
        private String name;

        //constructors...

        @Id
        @Column(name = "username", unique = true, nullable = false)
        @NotNull
        public String getUsername() {
           return this.username;
        }

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

        @Column(name = "name")
        public String getName() {
           return this.name;
        }

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

}
```

What Was Generated?

```
@Name("userHome")
public class UserHome extends EntityHome<User> {

        public void setUserUsername(String id) {
           setId(id);
        }

        public String getUserUsername() {
           return (String) getId();
        }

        @Override
        protected User createInstance() {
           User user = new User();
           return user;
        }

        public void wire() {
        }

        public boolean isWired() {
           return true;
        }

        public User getDefinedInstance() {
           return isIdDefined() ? getInstance() : null;
        }

}
```

What Was Generated?

```
@Name("userList")
public class UserList extends EntityQuery {

        private static final String[] RESTRICTIONS = {
                    "lower(user.username) like concat(lower(#{userList.user.username}),'%')",
                    "lower(user.name) like concat(lower(#{userList.user.name}),'%')",};

        private User user = new User();

        @Override
        public String getEjbql() {
          return "select user from User user";
        }

        @Override
        public Integer getMaxResults() {
          return 25;
        }

        public User getUser() {
          return user;
        }

        @Override
        public List<String> getRestrictions() {
          return Arrays.asList(RESTRICTIONS);
        }

}
```

What Was Generated?

- Search, view, and edit pages
- With association management
- With pagination and sorting
- Bookmarkable search results
- With interactive validation via Ajax
- Conversational editing (easy optimistic locking)

Q&A

Developing Applications With Seam and Eclipse

Gavin King

JBoss, A Division of Red Hat

gavin@hibernate.org

TS-4092