



The Shale Framework

Craig R. McClanahan
Senior Staff Engineer
Sun Microsystems, Inc.



Agenda

- **Background**
- **JavaServer Faces and Other Frameworks**
- **Tour of Shale Features**
- **Shale and Struts Action Framework**
- **Current Status and Roadmap**



Background

- **JavaServer Faces 1.0 released in March, 2004:**
 - Initial focus was on getting the component APIs right
 - Hidden inside is a *front controller* to handle each request
 - Not enough time to address framework aspects as well
 - Selected approach – provide extension points for adding desired functionality
- **Extension points can be used by:**
 - Components – to provide specialized services
 - Frameworks – to provide additional functionality
 - Applications – to meet specific needs



JavaServer Faces And Other Frameworks

- **JSF came into being in an environment filled with frameworks**
- **Desire to leverage new and old capabilities together**
- **Two fundamental approaches to framework integration:**
 - **Treat JSF as a *view* tier only**
 - **Treat JSF as a *controller* and a *view* tier**
- **First approach is available for several frameworks already:**
 - **Spring**
 - **Struts**
 - **Beehive**
- **And is easily added to others**



JavaServer Faces And Other Frameworks

- This first approach has two overlapping sets of issues:
 - ***Resulting application architecture:***
 - Typically a front controller “in front of” a front controller
 - JSF handles UI events, delegates form submit events
 - ***Overall architectural elegance:***
 - Redundant functionality – conversion, validation, page navigation, invoking actions
 - Impedance mismatches – expression language syntax, lifecycle differences
- Treating JSF as *view tier only* is recommended primarily as a migration strategy, not as an endgame



JavaServer Faces And Other Frameworks

- **Building a framework on top of JavaServer Faces has advantages:**
 - **Smaller – skip implementing all the redundant functionality**
 - **Easier to use – do not have to learn “old” and “new” version of (say) navigation**
 - **Enables a focus on *adding features* and *improving ease of use***
- **Started working on Shale in Fall 2005, focused on:**
 - **Adding ease of use APIs inspired by Java Studio Creator**
 - **Integrate functionality that existing Struts developers expect:**
 - **Client-side validation, Tiles layout management**
 - **Integrate new functionality enabled by JSF**
 - **(Later) Layer that leverages Java SE 5 annotations**

JavaServer Faces And Other Frameworks

- To date, I am aware of only one other framework taking this tack:
 - JBoss Seam
 - Focused on tying JSF to EJB3 tier
 - Also includes features for workflow orchestration
- But the extension capabilities are becoming widely used
 - Clay / Facelets – alternative view representation
 - AJAX component libraries – inject phase listeners without requiring extra web.xml configuration
- Treating JSF as a *controller* and a *view* tier is the recommended basis for new projects that incorporate JSF



JavaServer Faces Extension Points

- **VariableResolver** – Customize evaluation of the first token in a value binding or method binding expression
- **PropertyResolver** – Customize evaluation of the “.” operator in value binding or method binding expressions
- **NavigationHandler** – Customize execution of navigation decisions
- **ViewHandler** – Customize creation and restoration of new views
- In addition, **PhaseListener** instances can participate in (and modify) the standard request processing lifecycle

Tour of Shale Features

- **Key Functionality:**

- **View Controller**
- **Dialog Handler**
- **Clay Plug In**
- **Tiger Extensions**
- **Remoting**

- **Other Features:**

- **Application Controller**
- **JNDI and Spring Integration**
- **Unit testing framework**
- **Struts Feature Integration (Commons Validator, Tiles)** ⁹

View Controller

- **A common pattern in JSF is one backing bean per page**
- **Must know the JSF request processing lifecycle to understand where to inject some types of application logic**
- **Example -- “expensive” database query needed to populate a table**
 - **Only want to perform this query if it will actually be used**
 - **Skip it if we navigated elsewhere**
- **Example – need a transactional resource available through rendering, but then need to clean up**
 - **Need to regain control after rendering is completed**

View Controller

- **Shale provides an optional interface for your backing bean**
 - **Must also use a naming convention for managed bean name**
- **Implements the “Hollywood Principle”:**
 - *Don't call us, we'll call you*
- **Four application oriented callbacks provided:**
 - **init() -- Called when view is created or restored**
 - **preprocess() -- Called when about to process a postback**
 - **prerender() -- Called when about to render the response**
 - **destroy() -- Called after rendering, if init() was called**
- ***AbstractViewController* – Convenience base class available**

View Controller – Example Use Case

- **Shale SQL Browser – analog to SQL command console:**
 - **Allow user to perform arbitrary SQL SELECT statements**
 - **Dynamically reconfigure table columns based on results**
 - **In prerender(), execute query and rebuild component tree**
 - **In destroy(), clean up the JDBC resources that were used**



Dialog Handler

- **NavigationHandler – Standard page navigation criteria:**
 - **What view am I currently processing?**
 - **Which execute action was invoked?**
 - **What logical outcome was returned by this action**
- **Issue – Modelling of a conversation is *ad hoc***
- **Issue – How do we deal with “conversational” state?**
 - **Pass information in hidden fields**
 - **Can be unwieldy when numerous fields required**
 - **Store information in a session**
 - **Occupies server memory if not cleaned up**

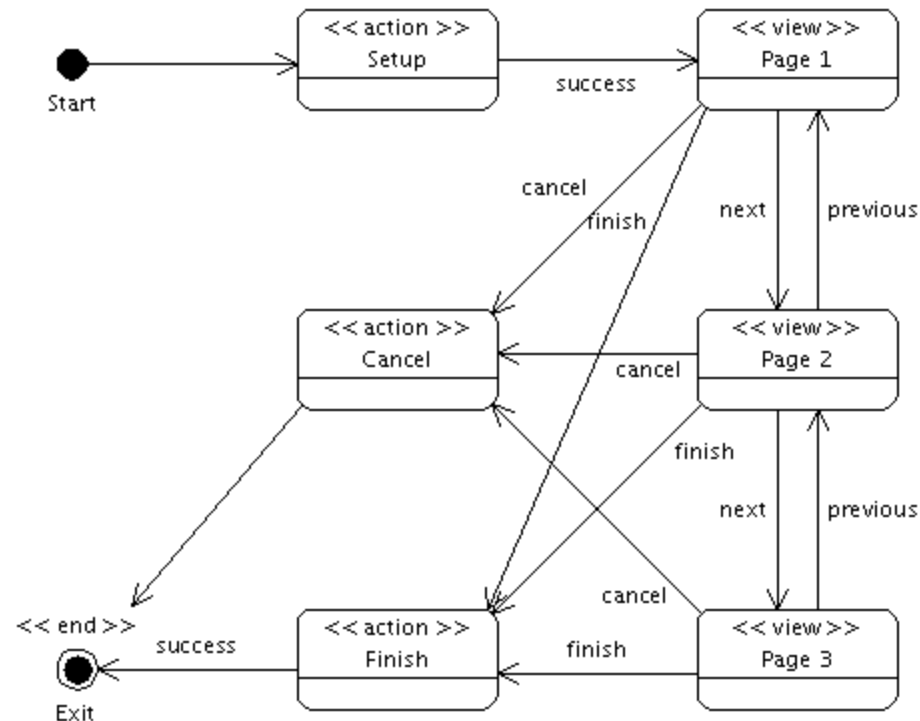


Dialog Handler

- ***Dialog Handler*** abstraction provided to deal with these issues
 - Models conversations as a state machine
 - Provides “conversational state” storage mechanism
 - Heavily inspired by Spring Web Flow, but “JSF-ized”
- States represent processing activities that can occur:
 - ***Action*** – Call an arbitrary method binding expression
 - ***View*** – Display a view, plus subsequent form submit
 - ***Subdialog*** – Call a named dialog as a black box
 - ***Exit*** – Exit from the current dialog
- Transitions between states driven by logical outcome strings

Dialog Handler

- Dialogs can be modelled with UML State Diagrams:



Created with Poseidon for UML Community Edition. Not for Commercial Use.



Dialog Handler

- **Dialog definitions are configured at application startup**
 - **Via parsing an XML file**
 - **Or by programmatic means**
- **Dialog is entered by returning a special logical outcome:**
 - **Return *dialog:foo* to enter a dialog named “foo”**
- **Implementation is a custom *NavigationHandler* extension**



Dialog Handler – Example Use Case

- **“Use Cases” Demo Application logon dialog:**
 - **Log on with existing username and password**
 - **Create user profile and log on**
 - **Edit existing user profile**
 - **Optionally support “remember me” cookies**



Clay Plug-In

- **JavaServer Faces mandates that standard components support JavaServer Pages (JSP) for view representation**
 - **Optional but recommended for third party components**
 - **Requires developing tag handler classes and TLD**
- **Issue – Interoperability issues with template text and other tags**
 - **Mostly resolved with JSF 1.2 and JSP 2.1 (part of Java EE 5)**
- **Issue – Techniques to reuse portions of page layout are hard**
 - **Can be addressed by JSF components focused on this**
- **Issue – Some developers prefer a more “pure HTML” representation of the view portion of an application**

Clay Plug-In

- Clay enables grafting a component subtree onto an existing tree
- Sounds simple, but provides three compelling features:
 - *HTML Views* – Can separate views into pure HTML pages, with pointers to component definitions
 - Similar capabilities found in Tapestry and Facelets
 - *Metadata Inheritance* – Component definitions can extend previous definitions
 - Similar in spirit to how Tiles can extend other Tiles
 - Can create fine grained reusable “components” with no coding (address form)
 - *Symbol Replacement* – When components are used, customize to current use case (managed bean name)

Clay Plug-In – Logon Form (Using JSP)

```
<h:form>
  <table border="0">
    <tr><td>Username:</td>
      <td><h:inputText      id="username"
                          value="#{logon.username}" /></td></tr>
    <tr><td>Password:</td>
      <td><h:inputSecret   id="password"
                          value="#{logon.password}" /></td></tr>
    <tr><td><h:commandButton id="logon"
                          action="#{logon.authenticate}" /></td></tr>
  </table>
</h:form>
```



Clay Plug-In – Logon Form (Using Clay)

```
<form jsfid="logonForm">
  <table border="0">
    <tr><td>Username:</td>
      <td><input type="text" name="username"
        jsfid="username" /></td></tr>
    <tr><td>Password:</td>
      <td><input type="password" name="password"
        jsfid="password" /></td></tr>
    <tr><td><input type="submit" value="Log On"
      jsfid="logon" /></td></tr>
  </table>
</form>
```



Clay Plug-In – Component Definition

```
<component jsfid="username"
    extends="inputText"
    id="username">
    <attributes>
        <set name="required" value="true" />
        <set name="value" value="{logon.username}" />
    </attributes>
</component>
```

Clay Plug-In

- So why do I want this?
 - Pure HTML can be easily built with standard HTML editors
 - Graphic artist can include “sample” data that will be replaced

```
<table jsfid="addressList">
    ... dummy columns and data values ...
</table>
```
- Four approaches are supported:
 - Strictly XML that uses composite components (addressForm)
 - Tapestry style separate HTML (as illustrated above)
 - Subtree dynamically calculated at runtime `<clay:clay>` tag
 - XML similar to HTML approach



Clay Plug-In – Use Case Examples

- **“Use Cases” example includes four implementations of a simple example application (Rolodex)**



Tiger Extensions

- **JSF and Shale use XML for configuration files:**
 - But XML configuration is going out of fashion :-)
 - Can we reduce or eliminate this stuff?
- **Java SE 5 (codenamed “Tiger”) includes *annotations*:**
 - Provide metadata, not functionality
 - In source code, can annotate classes, methods, and fields
 - Can be examined at compile time (i.e. for code generation)
 - Can be processed at runtime
- **NOTE – not every config element should become an annotation!**
- **Tiger Extensions library adds optional features to Shale using annotations**

Tiger Extensions

- **Three categories of annotations are currently supported:**
 - *Annotated managed beans*
 - *Annotated view controllers*
 - *Annotated JSF component registration*
- **All of these annotations are processed at runtime**
- **Searches for annotated classes in a web application:**
 - **/WEB-INF/classes**
 - **Jar files in /WEB-INF/lib that have a META-INF/faces-config.xml**

Tiger Extensions – Managed Beans

- Managed beans typically defined in faces-config.xml

```
<managed-bean>
```

```
  <managed-bean-name>foo</managed-bean-name>
```

```
  <managed-bean-class>...</managed-bean-class>
```

```
  <managed-bean-scope>request</managed-bean-scope>
```

```
  <!-- optional property initializations -->
```

```
</managed-bean>
```

- Replaced by annotations in the Java source code:
 - `@Bean(name="foo", scope=Scope.REQUEST) public class Foo`
 - `@Value("#{bar}") private int bar;`

Tiger Extensions – View Controllers

- **Basic Shale requires your backing beans to implement the *ViewController* interface to receive these services**
 - Therefore requires implementing all callback methods
 - Optional base class contains empty definitions
- **Tiger Extensions allow you to annotate classes**
 - **@View public class Foo { ... }**
- **And the callback methods (no required method names):**
 - **@Init public void myInit() { ... }**
 - **@Preprocess public void setup() { ... }**
 - **@Prerender public void justBeforeRendering() { ... }**
 - **@Destroy public void destroy() { ... }**



Tiger Extensions – Components

- **JSF allows component libraries and applications to register custom objects at runtime**
 - **User interface components**
 - **Converters**
 - **Renderers**
 - **Validators**
- **Tiger extensions provide annotations to “self register” them:**
 - **@FacesComponent(“componentType”)**
 - **@FacesConverter(“converterId”)**
 - **@FacesRenderer(renderKitId=“x”, componentFamily=“y”, rendererType=“z”)**
 - **@FacesValidator(“validatorId”)**

Tiger Extensions -- Example

- The *Query.java* class in the SQL Browser example that we looked at before uses two categories of these annotations

- Class level annotations:

```
@Bean (name="query" , scope=Scope.REQUEST)
```

```
@View
```

```
public class Query { ... }
```

- Method level annotations:

```
@Prerender public void prerender() { ... }
```

```
@Destroy public void destroy() { ... }
```

Remoting

- It is common for applications to respond to *programs* as well as to *humans*:
 - Web services
 - AJAX-based asynchronous requests
- Shale provides features to make this easier:
 - For application developers
 - For JSF component authors
- Packaged as a small (40k) JAR, no dependencies except JSF:
 - Zero configuration if you accept the defaults
 - Implemented as a JSF PhaseListener

Remoting

- Primary concept is the idea of a *Processor*:

```
public interface Processor {  
    public void process(FacesContext context,  
        String resourceId) throws IOException;  
}
```

- Processors examine a “resource identifier” and construct the entire response
- Processors are registered to a URL pattern, similar to servlets:
 - Path mapping and extension mapping supported
 - Creates a *FacesContext* so you can use EL, managed beans



Remoting

- **Processor architecture is extensible**
 - Each processor mapped to a URL pattern
 - Application specific processors can be configured
- **Standard processor implementations are provided:**
 - **Classpath resource serving:**
 - <http://localhost:8080/myapp/static/org/apache/foo.css.faces>
 - **Webapp resource serving:**
 - <http://localhost:8080/myapp/webapp/resources/foo.js.faces>
 - **Map to a dynamically generated method binding:**
 - <http://localhost:8080/myapp/dynamic/foo/bar.faces>
 - Executes method binding `#{foo.bar}` to return the response

Remoting

- **Helper classes to assist developers:**
 - **Two way mapping of resource id <--> URL**
 - **Create *ResponseWriter* implemenations for dynamic output**
- **Example – Auto Complete Text Component**
 - **Available as a sample with Java Studio Creator**
 - <http://developers.sun.com/jscreator/>



Other Shale Features

- **Application Controller:**
 - **Configured as a Servlet Filter**
 - **Supports decoration of request processing lifecycle**
 - **Uses “chain of responsibility” pattern (Commons Chain)**
 - **Similar in spirit to how Struts 1.3 is being implemented**
- **JNDI and Spring Integration:**
 - **Custom JSF variable and property resolvers**
 - **Allows transparent access to JNDI contexts and Spring created beans, via EL expressions**
- **Unit testing framework:**
 - **Mock objects for building unit tests**



Other Shale Features

- **Struts Functionality Equivalents Integration:**
 - **Commons Validator for client side validation**
 - **Implemented as a JSF validator**
 - **Tiles support**
 - **Based on “standalone” version of Tiles being developed**
 - **No dependency on struts.jar**
 - **Can navigate to a page or to a tile**



Shale and the Struts Action Framework

- **Shale was originally proposed to the Struts developers as a “revolutionary” basis for Struts 2.0**
 - **Not accepted as that role**
 - **Was accepted as a formal subproject of the Struts project**
 - **To avoid confusion, the existing Struts framework is rebranded as the *Struts Action Framework***
- **More recently, the developers of WebWork 2 expressed interest in merging with another project, and Struts community agreed**
 - **WebWork codebase will be used as basis of Struts Action Framework 2**
 - **Currently in the Apache Incubator**
- **Both frameworks will be developed and released**



Current Status And Roadmap

- **As of today, voting continues on a 1.0.2 release:**
 - **Following standard Struts labelling policy for releases**
 - **After test build, vote to label Alpha, Beta, General Avail**
 - **This release will likely be approved as an “Alpha”**
 - **Relies on unreleased version of Standalone Tiles**
 - **Dialog functionality is not yet complete**
- **Most APIs in Shale should be considered stable enough to use:**
 - <http://struts.apache.org/struts-shale/api-stability.html>
- **When will it go General Availability?**
 - **As soon as I stop flying on airplanes 3 weeks out of 4 :-)**
 - **As soon as we get user feedback that it is ready**



Questions