JavaServer Faces

Gary Murphy
Hilbert Computing, Inc.
http://www.hilbertinc.com/
glm@hilbertinc.com
JSF Presentations at CSS

- “Extending JSF to Build a Product-specific UI Framework” - Bryan Basham
- “Comparing Java Frameworks” - Matt Raible
What is JSF?

- JavaServer Faces is a Java-based user-interface framework
  - Creates a model for user interface components
  - Manages the state of user interface components
  - Creates an event model for notification of user interface changes, similar to Swing or SWT
  - Handles input validation and conversion
What is JSF?

- JavaServer Faces is *(continued)*
  - Handles page navigation
  - Handles the construction of POJOs via dependency injection
- JSF is the J2EE web development standard
JSF is NOT...

- ...an application framework
  - No controller (as in MVC)
  - Scope is primarily user-interface
  - Some support for navigation
  - Some support for POJO construction

- ...JSPs
  - The reference implementation *uses* JSPs for rendering, but that is not a requirement
  - JSF is at a much higher level of abstraction
Comparison to Struts

- Model-View-Controller
  - Struts includes an MVC controller
  - JSF does not have a controller, but provides a mechanism for page navigation

- UI Model
  - Struts is form-based. That is, it processes one page at a time.
  - JSF is component based. Each component (e.g. button, list box) can have associated actions and events.
Brief History

- JSF 1.0 (2004-03-11) under JSR 127
- JSF 1.1 (2004-05-27)
  - Bug fixes
  - Note two *month* span between 1.0 and 1.1
- JSF 1.2 (2006-05-11) under JSR 252
  - Support for multi-frame, multi-window designs
  - Facelet support
  - Bug fixes, especially for Portlets, duplicate press
Brief History

- JSF 1.2_03 (2006-12-04)
  - Bug Fixes
- JSF 1.2_04 (2007-03-05)
- JSF 1.2_05 (2007-10-17)
Criticisms

- Initial implementation was weak.
  - Need to distinguish architecture from implementation.
  - Architecture is very robust
  - Initial implementation was very weak

- Heavier Memory Requirements
  - Must manage a component tree
  - Heavier than Struts where mainly a wrapper over the POST data is maintained
JSF Implementations

- Open Source choices exist other than the reference implementation:
  - ICEfaces
  - JBoss RichFaces
  - MyFaces Tomahawk from Apache

- Commercial choices:
  - Infragistics NetAdvantage
  - IBM Rational Application Developer
Architecture

- Models user-interface at the component level
  - Supports component-level actions and events
  - Similar conceptually to Swing or SWT
- Separates the description of a component from the implementation
  - Reference implementation uses JSP
  - Implementations could render WML, HTML, XHTML, XUL, etc.
Architecture

- Component Model
  - Class that implements the tag (*e.g.* SpinnerTag)
  - Class that implements the component class (*e.g.* UISpinner):
    - Maintains state
    - Renders the implementation of user interface
    - Processes the user input (*e.g.* HTTP POST values)
Architecture

- Components can delegate rendering and input processing
  - Allows for plugging in different UI technologies such as WML, XUL
  - Highly recommended for custom components
Architecture

- Data Conversion Framework
  - HTTP POST contains String values
  - Built-in conversions for Java data types
    - `int`, `Integer`, `Date`, `etc.`
  - Supports custom converters. For example:
    - Phone number. Parsing and formatting special chars
    - Credit cards. Parsing and formatting blanks
  - Declared in `faces-config.xml`
Architecture

- Data Validation Framework
  - Similar to data conversion, but can validate data as well
Architecture

- Support for internationalization via resource bundles.

```<f:loadBundle basename="com.acme.msgs" var="msgs" />
<f:outputText value="#{msgs.useridLabel}"/>```
Architecture

- Support for visual development environments
  - Eclipse has a JSF project in development
  - MyEclipse has a visual development environment for MyFaces
  - IBM RAD has a visual development environment for their JSF implementation
    - Not open source
    - Poorly documented
    - Some components poorly implemented
Architecture

- Managed beans
  - Uses the JavaBean specification
  - Configured in the `faces-config.xml`
    - Name
    - Scope
    - Class
  - Automatically factory-constructed with dependency injection
  - Name is used in EL in the JSP:
    ```html
    <h:outputText value="#{user.name}" />
    ```
Architecture

- Backing beans
  - Java object that contains the UIComponent objects that are the Java representation of the components on a given page
  - Typically generated by a visual development environment (VDE).
  - Should be used as a binding between the component as HTML and the component as a Java object.
    - Should not contain business rules
Architecture

- JSF Lifecycle consists of six phases:
  - Restore view
  - Apply request values
  - Process validations
  - Update model values
  - Invoke application (actions and events)
  - Render response

- Applications can get callbacks before and after each phase by implementing a phase listener
Architecture

- JSF includes an event model:
  - Value change events are attached via the `valueChangeListener` attribute of a component.
    - These should submit the form via JavaScript `onchange` events.
    - Provides access to the old value and the new value
  - Action listeners called during normal HTTP POST via button or command hyperlink
Architecture

- Action Events
  - These are generated via submit buttons or command links.
  - These action events are bound to any no-arg method on any Java object in any scope.
    - This should be bound to an explicit controller method, but there is no controller as a part of JSF.
  - This is the primary way in which information gets from the browser to the application.
Architecture

- Navigation

- Actions on user interface components run code that returns a String that is an outcome name. (Similar to Struts)

- The string is matched to an outcome in the Faces configuration along with the current view ID (i.e. JSP)

- Also can map to global outcomes, which map a logical name to a JSP. (More like Struts).
Architecture

- Navigation
  - An invalid or null outcome will stay on the same page
  - *I will show an example of Faces config...*
Facelets

- Introduced in JSF 1.2
- Provides templating
- Does not use JSPs
  - JSF and JSP lifecycle were in conflict at times.
  - Uses the new EL-API that will be in JSP 2.1 without requiring JSP 2.1
- Eliminates `view` and `subView`
  - Compensated for JSP binding issues
Configuring Facelets

- `jsf-facelets.jar` must be in classpath
- `faces-config.xml` override view handler:

```xml
<faces-config>
  <application>
    <view-handler>
      com.sun.facelets.FaceletViewHandler
    </view-handler>
  </application>
</faces-config>
```
Configuring Facelets

- **web.xml** - Change file type for defining views from JSPs (*jsp*) to XHTML:

```
<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>
```
Facelet Aliasing

- You can specify references to JSF components in the facelet much like a JSP.
- If HTML comes from a design group, you can alias

```html
<input type="text"
    jsfc="h:inputText"
    value="#{credentials.userid}"/>
```
Facelets Reference

Let's Take a Look

- Code with Facelets and JSF 1.2 RI
Techniques

- Overcoming some diagnostic deficiencies
  - Reference implementation has verbose debug logging using commons-logging
  - Recommend a phase listener
Techniques

- JSF is supportive of MVC, but doesn't provide a controller implementation.
  - You should implement an explicit controller
  - You can (but shouldn't) implement implicit controllers, which is what most JSF examples show.
Techniques

- The controller should:
  - Provide a navigation outcome
  - Become the anchor point for non-functional requirements:
    - Logging, authorization, performance metrics, serialization checking, etc.
Techniques

- Use JSP includes appropriately as you would in a Struts or plain JSP application. (Better to use facelets):
  - Headers
  - Footers
  - Navigation bars
  - Groups of controls for which you don't want to build a custom component
  - Use the `<f:view>` tag to avoid rendering multiple `<body>` tags.
Techniques

- Backing bean vs. Updating state on view objects.
  - Backing beans are typically generated by visual development environments (VDEs)
  - VDEs may update the backing bean with business rules entered in the VDE.
  - My recommendation is to not use them.
Techniques

- Favor global navigation outcomes over view-to-view navigations:
  - This pushes navigation into your controller implementation
  - Avoids the combinatorial explosion problem when supporting multiple destinations
Techniques

- Strive for 100% JSF tags in the JSP.
  - Mixing JSF tags and other JSP tags, like JSTL, cause problems because both write to the response stream.
  - Don't use tables – use DataTable JSF objects instead.
  - Wrap HTML in `<f:verbatim>` tags.
  - Almost no HTML is needed.
    - `<script>` tags, `<link>` to CSS only, `<meta>`, `<title>`

Note some VDEs drop HTML into the page
Techniques

- Push styling into CSS
  - All JSF components have references to one or more style sheets
  - Some JSF components directly support HTML styling, such as width, but I tend to stay away from those so that look-and-feel is external to the application.

- That's a recommendation for non-JSF web applications as well.

Techniques

- Most web applications involve some DHTML.
- These aren't JSF-specific, but are worth mentioning:
  - Firefox with the FireBug plugin is useful for debugging JavaScript
  - JSF confuses FireBug because the URL doesn't change with the view change.
Discussion

- Any questions?
- Comments or advice from your experiences?
- Comparison with Struts or other web frameworks.

- Thank you for taking the time to learn about JSF.