About Faces: The JavaServer™ Faces API and how it relates to Struts

Ed Burns

Staff Engineer

J2EE Platform

Sun microsystems

We make the net work.

Presentation Goals

Demonstrate how JavaServer Faces makes it easier for you to develop Java web applications that are both usable and scalable.

Cover some differences, similarities, and an integration strategy with Struts

Educational Goals

- A feel for the value faces brings to the table
- An understanding of the core faces concepts
- An insight into the architecture of Faces
- An explanation of how faces relates to Struts

Agenda

- Faces Value Add
- Faces Architecture
- Core Concepts
  - workflow
  - components
  - converters and validators
  - value binding expressions
  - events
  - navigation handler
  - rendering
  - value style vs component style
- Struts
  - Overview
  - Major players in Struts in Faces
  - Similarities and differences
- Q&A

JavaServer Faces: What is it?

- A specification, reference implementation, and TCK for a web application development framework
  - Components
  - Events
  - Validators
  - Back-end-data integration
  - Designed to be added to tools

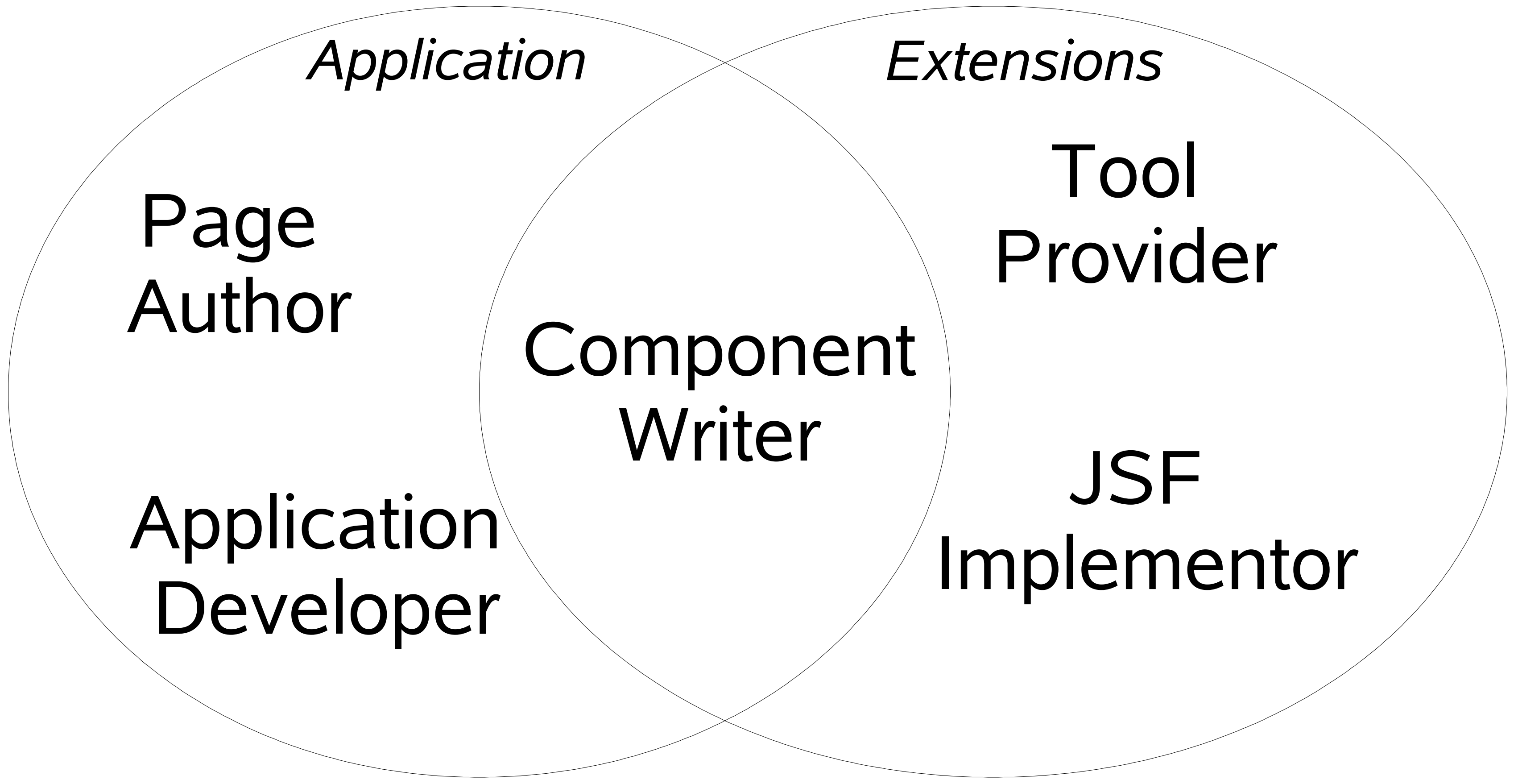
JavaServer Faces: Why do I need it?

- Best yet MVC for webapps
- Clean separation of roles
- Easy to use
- Extensible Component and Rendering architecture
- Support for client device independence

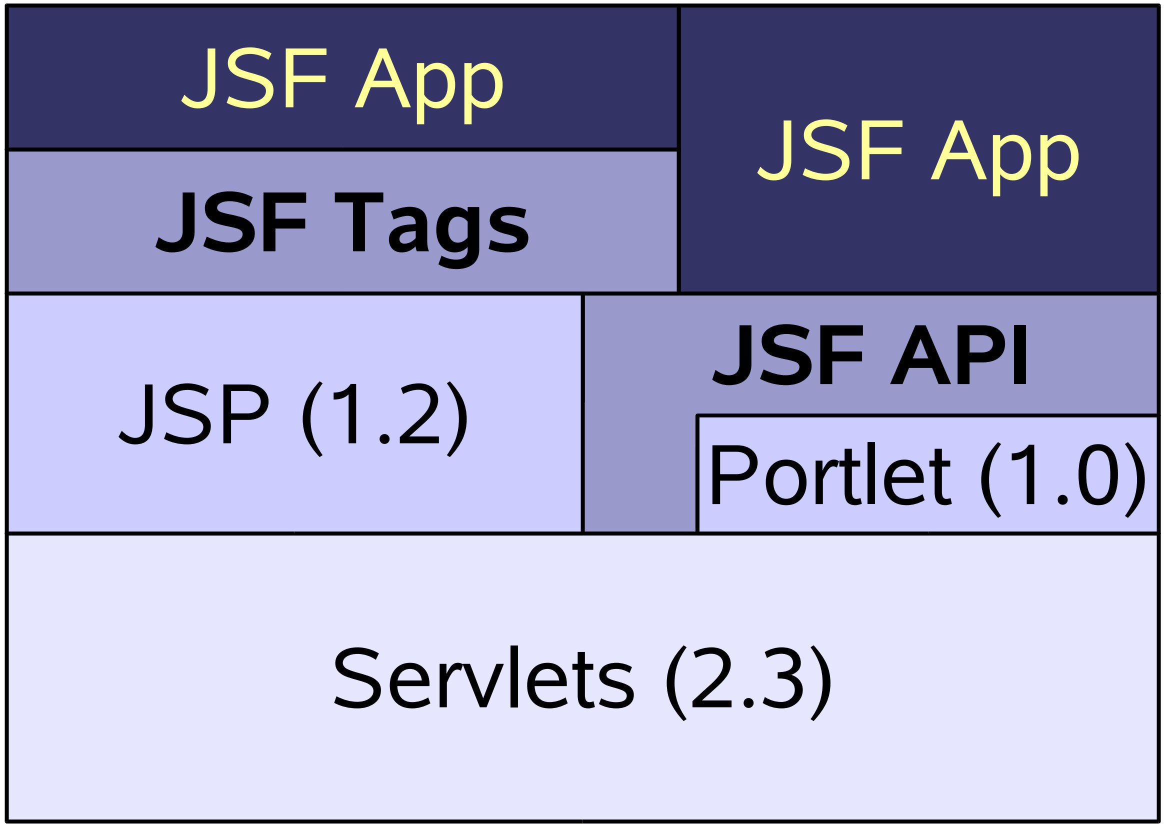
JSR Requirements

1)Tools can leverage Faces
2)Client device independence
3)MVC
4)State management
5)UI components
6)Events
7)Validation and error handling
8)Lightning fast performance

Roles in JSF


Application
Extensions
Page Author
Application Developer
Component Writer
Tool Provider
JSF Implementor

Architecture: dependencies



Core Concepts: Workflow

- Map closely to the webapp workflow
  - JSP pages composed of components
  - Page flow described by flexible XML syntax
  - Data Integration with the help of JavaBeans concepts

Core Concepts: Components

- A view is a tree of components
  - familiar concept in UI design
  - maps well to XML
  - Faces components are JavaBeans, they have properties, methods and events/listeners
- Things attached to components
  - validators
  - converters
  - listeners

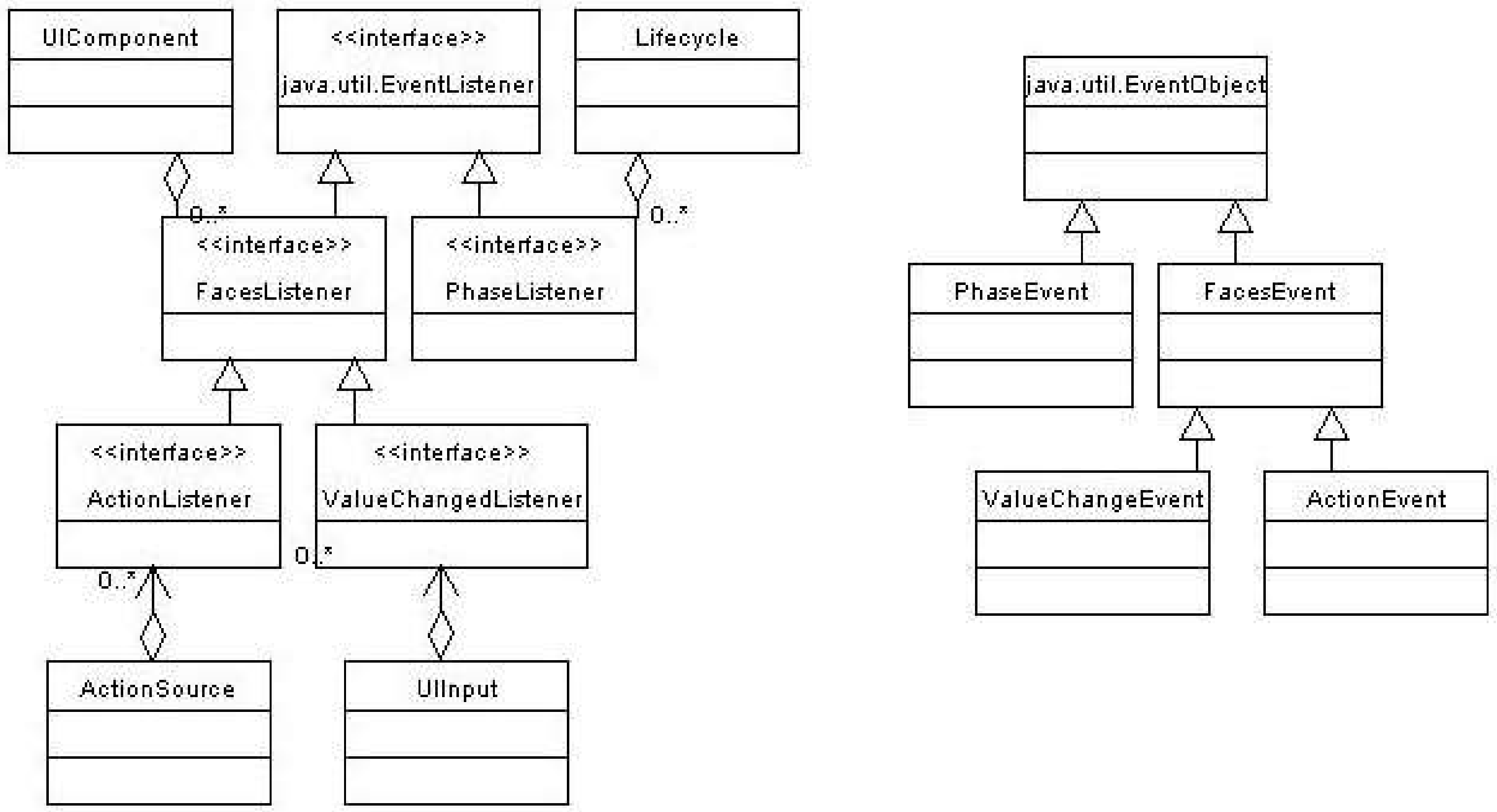
Core Concepts: Converters and Validators

- Converters – Plugins for conversion:
  - Maximum of one converter per component
  - Explicit (by id) or implicit (by type) registration
  - Output – Object to string
  - Input – String to object
  - Standard implementations included
- Validators – Correctness checks on input values:
  - Built in (to component) validation
  - Register one or more external Validators
  - Standard implementations included

Core Concepts: ValueBinding

- All properties and attributes can be expressed as:
  - Literal value ("foo", 123")
  - Calculated on demand ("#{customer.name}")
  - Accessible via Java API and as custom tag attributes
- Expression syntax and semantics based on JSTL 1.1 / JSP 2.0 EL
- Input components have a local value representing the server-side state of the client field
- Local values on input components are propogated to the model if an expression is defined
  - Must be an "lvalue" expression

Core Concepts: Event Model



Core Concept: Events

- Standard JavaBeans design patterns
- Strongly typed events
- Strongly typed listener registration
  - Register one or more listener implementations
- Standard events and listeners:
  - ActionEvent – User interface component was activated
  - ValueChangeEvent – Input field has been validated and changed value

Core Concept: MethodBinding

- As we have seen, you can register external listeners and validators
- Each listener or validator is an instance of a separate class
- It would be convenient if we could reference methods in a single "backing bean" class
- Method binding expressions enable this:
  - Syntax like a value binding expression (#{mybean.mymethod})
  - Last element identifies a public method
  - API allows restriction based on parameter signatures

Core Concept: MethodBinding

- UICommand – actionListener
  - Method that acts like ActionListener.processAction()
- UIInput – validator
  - Method that acts like a Validator.validate()
- UIInput – valueChangeListener
  - Method that acts like a ValueChangeListener.processValueChange()
- UICommand – action
  - Method that is called when an action event occurs

Core Concept: NavigationHandler

- Navigation decisions externalized to a pluggable NavigationHandler
- Decision outcome is the name of the view to be displayed next:
  - Details are dependent on ViewHandler instance in use
  - Most common use case – “view” == “JSP page”
- Default NavigationHandler implementation bases decision on:
  - Which view (page) is being processed?
  - Which application action was invoked?
  - Which logical outcome was returned by the invoked action?
- Navigation rules configured in faces-config.xml

Core Concept: Navgation

```
<navigation-rule>

  <!-- Search Button on every page -->
  <from-view-id> * </from-view-id>
  <navigation-case>
    <from-action>
      searchHandler.go
    </from-action>
    <from-outcome> success </from-outcome>
    <to-view-id> /search-results.jsp </to-view-id>
  </navigation-case>

</navigation-rule>
```

Core Concept: Rendering

- Renderers – Adapt components to a specific markup language:
  - Encoding – Create markup to represent component value
  - Decoding – Interpret request parameters to update component value
- RenderKits – Library of Renderers:
  - Extensible at runtime
  - For JSP, represented as custom tag libraries

Sample JSP Page – Value Style

```
<f:view>
  <f:form id=”logonForm”>
    <h:panelGrid    columns=”2”>
      <h:outputText   value=”Username:”/>
      <h:inputText       id=”username”
                        value=”#{logonBean.username}”/>
      <h:outputText   value=”Password:”/>
      <h:inputSecret     id=”password”
                        value=”#{logonBean.password}”/>
      <h:commandButton type=”submit”
                        label=”Log On”
                    action=”#{logonBean.logon}”/>
      <h:commandButton type=”reset”
                        label=”Reset”/>
    </h:panelGrid>
  </f:form>
</f:view>
```

Managed Bean Creation Facility

- In the previous example, we saw references to logonBean
- JavaServer Faces tries to find this bean in any scope:
  - Request
  - Session
  - Application
- If not found, optionally:
  - Instantiate a bean of a specified class
  - Configure bean property values
  - Store bean instance in specified scope
- Configuration rules in faces-config.xml

Managed Bean Creation Facility

```
<managed-bean>
  <!-- Customer Bean created on demand -->
  <managed-bean-name>customer</managed-bean-name>
  <managed-bean-class>
    mypackage.CustomerBean
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>creditLimit</property-name>
    <value>#{initParam.defaultCreditLimit}
      </value>
  </managed-property>
</managed-bean>
```

Sample Business Bean – Value Style

```
package com.mycompany.mypackage;

public class MyLogonBean { // No required base
  class

   // The usual             property
   private String username;
   public String getUsername() { return
  username; }
   public void setUsername(String username)
     { this.username = username; }

   // The usual password property
   private String password;
   public String getPassword() { return
  password; }
   public void setPassword(String password)
     { this.password = password; }
```

Sample Business Bean – Value Style

```
// The business logic for a logon

public String logon() {
  if (isValidLogon(username,
password)) {
    ... record successful logon ...
    return “success”;
  } else {
    ... enqueue error message ...
    return “failure”;
  }
}

}
```

Value Style and Component Style

- In the previous example, the component values are bound to properties in the data model:
  - Assumption – page author is in charge of the properties and attributes of the components
  - Backing bean has zero API dependencies on JavaServer Faces APIs
  - JavaServer Faces understands model data type, and can provide implicit conversions
- This approach will be very familiar to users of frameworks like Struts:
  - Backing Bean == ActionForm + Action
  - Still possible to separate these concepts if appropriate

Value Style and Component Style

- A different style (“component style”) has been popularized by other frameworks:
  - UI components themselves are bound to properties in the backing bean
  - Backing bean can programmatically modify component attributes directly
  - Backing bean can optionally instantiate components instead of letting the page do so
  - Any required converters must be explicitly registered
- This approach will be very familiar to users of frameworks like ASP.Net's “code behind files”
  - Will likely be the approach taken by many tools built on top of JavaServer Faces

JSP Page – Component Style

```
<f:view>
  <f:form id=”logonForm”>
    <h:panelGrid    columns=”2”>
      <h:outputText   value=”Username:”/>
      <h:inputText         id=”username”
                    binding=”#{logonBean.username}”/>
      <h:outputText   value=”Password:”/>
      <h:inputSecret       id=”password”
                    binding=”#{logonBean.password}”/>
      <h:commandButton type=”submit”
                       label=”Log On”
                    action=”#{logonBean.logon}”/>
      <h:commandButton type=”reset”
                       label=”Reset”/>
    </h:panelGrid>
  </f:form>
</f:view>
```

Backing Bean – Component Style

```
package com.mycompany.mypackage;
import javax.faces.component.UIInput;

public class MyLogonBean { // No required base class

  // The username component
  private UIInput username;
  public UIInput getUsername() { return username; }
  public void setUsername(UIInput username)
    { this.username = username; }

  // The password component
  private UIInput password;
  public UIInput getPassword() { return password; }
  public void setPassword(UIInput password)
    { this.password = password; }
```

Backing Bean – Component Style

```
    // The business logic for a logon

     public String logon() {
       String user = (String) username.getValue
     ();
       String pass = (String) password.getValue
     ();
       if (isValidLogon(user, pass)) {
         ... record successful logon ...
         return “success”;
       } else {
         ... enqueue error message ...
         return “failure”;
       }
     }

   }
```

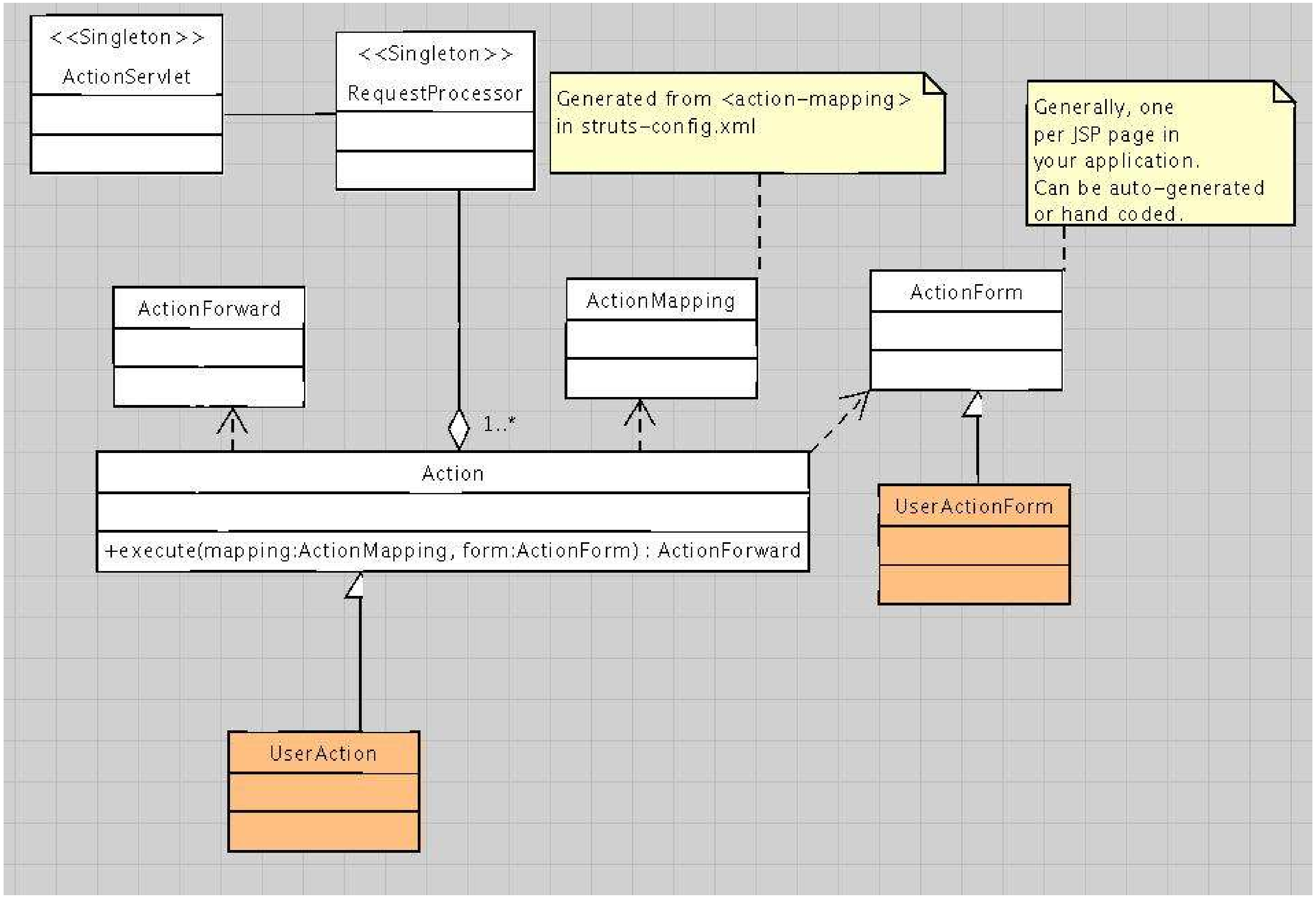
Value Style and Component Style

- Both value style and component style approaches are legitimate use cases
- Value style is preferred when:
  - Generally do not manipulate component properties in event handlers (although it is still possible)
  - You want to leverage the implicit converter capabilities of JavaServer Faces
  - You want to minimize the API dependencies of your backing bean classes (for unit testing, etc.)
- Component style is preferred when:
  - You want to create components programmatically
  - You regularly want to manipulate component properties in event handlers
  - You are using development tools using this approach
- Hybrid approaches are also possible

Struts: Overview

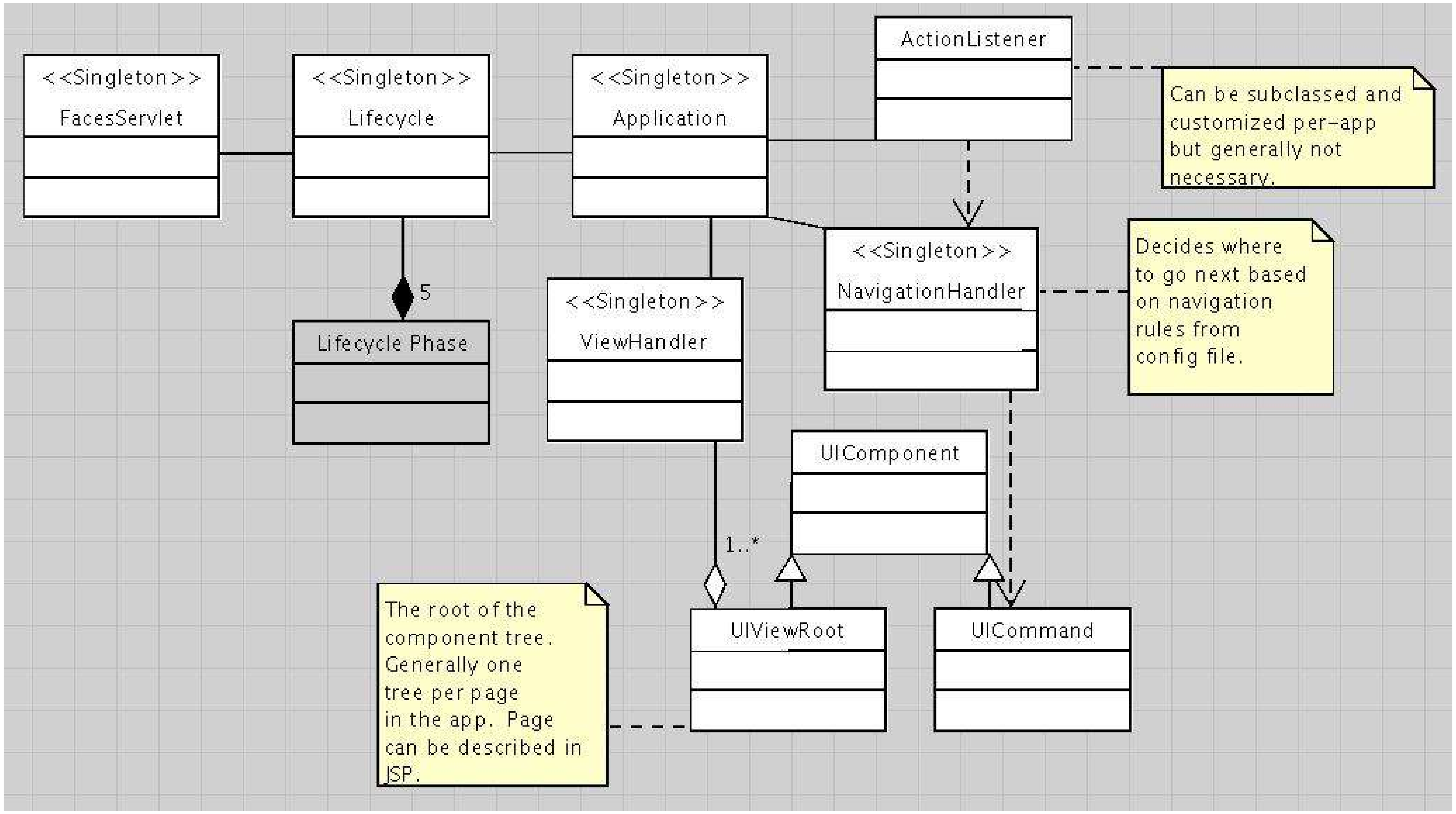
- De-facto standard for Java web-apps
- Pretty basic support: gets the job done
  - MVC framework
  - Simple navigation
  - Validation
  - Basic conversion
  - Form centric
  - I18N

Major Players in Struts



<<Singleton>>
ActionServlet
<<Singleton>>
RequestProcessor
Generated from <action-mapping> in struts-config.xml
Generally, one per JSP page in your application. Can be auto-generated or hand coded.
ActionForward
ActionMapping
ActionForm
1..*
Action
+execute(mapping:ActionMapping, form:ActionForm) : ActionForward
UserActionForm
UserAction

Major Players in Faces



<<Singleton>>
FacesServlet
<<Singleton>>
Lifecycle
<<Singleton>>
Application
ActionListener
Can be subclassed and customized per-app but generally not necessary.
5
Lifecycle Phase
<<Singleton>>
ViewHandler
<<Singleton>>
NavigationHandler
Decides where to go next based on navigation rules from config file.
UIComponent
1..*
The root of the component tree. Generally one tree per page in the app. Page can be described in JSP.
UIViewRoot
UICommand

Similarities and Differences (interactive)

- Flexibility
- Foundation technologies
- Model Tier access
- Components, Events
- Conversion and Validation
- Request Processing and Navigation

Similarities and Differences

- Flexibility
  - Struts form-beans can span pages, but the concept gets muddied when you do that.
  - Faces has client device independence, Struts doesn't
  - Struts tags aren't as well suited to complex widgets such as trees and tab panels as are Faces components
  - Faces supports a “code behind files” concept, Struts does not.

Similarities and Differences

- Foundation Technologies
  - Both leverage XML, JSP, Servlet and JavaBeans
  - Both support high quality MVC architecture
  - JSF will be in J2EE 1.5
  - Both have a “config file” concept.
  - JSF 1.1 apps can run in a JSR 168 portlet, Struts apps cannot.

Similarities and Differences

- **Model Tier Access**
  - Struts uses commons-beanutils for bean hierarchy navigation
  - JSF uses the ValueBinding API.
  - Struts can create "FormBeans" for you. With DynaActionForms, you can pre-configure the initial properties of the form.
  - JSF has a much richer bean creation story. Bonus: I'm working with the Spring Framework people to make sure their bean factory can integrate well with the faces managed bean facility.

Similarities and Differences

- **Components and Events**
  - Struts has no notion of components, but the struts-faces integration library allows you to use the JSF component model, and keep your Struts based back end logic.
  - Since Struts has no notion of components, it has no notion of component state. Faces has an excellent state management story supporting saving the state in the client or on the server.
  - Faces brings a JavaBeans like event model to the web, Struts has nothing similar to this.
  - JSF has dataTable support, struts does not, but you can approximate it with Struts + JSTL.
  - JSF was intended from the beginning to create a market for third party components. See JavaOne 2004.

Similarities and Differences

- **Conversion and Validation**
  - Both have support for validation.
  - Both support type Conversion, but the Faces story is more powerful
  - Struts Action class tightly coupled to ActionServlet, can call its methods. Nothing in JSF calls the FacesServlet.
  - Struts DynaActionForm instances can be author automatically. There is no support in the JSF framework to author backing beans automatically.
  - Struts has support for client side validation in the framework. JSF does not. This will be in JSF 2.0.

Similarities and Differences

- **Request Processing and Navigation**
  - JSF uses logical outcomes from a java method to feed into a rule base. Struts uses the retruned ActionForward instance.
  - Struts Action concept is similar to what method bindings, and listeners give you in JSF.
  - In Struts, The ActionForm bean is passed to the Action and the action can do with it what it wants. In JSF, the ValueBinding mechanism exposes the entire managed-bean namespace to anywhere in the app that needs it.

Similarities and Differences

- **Request Processing and Navigation**
  - The Struts ActionForm beans and the JSF managed-beans are both intended to be proxy objects to the real model tier data, not to be that data themselves. However, this is only a recommendation.
  - The Struts "Action.execute() generates an ActionForward" concept is similar to the Faces "command generates an outcome that is used by the NavigationHandler.
  - In Struts, the decision of where to go to next, is based on the outcome of processing the form. That is, the which ActionForward is returned from the Action. In JSF the decision of where to go next is based on which UICommand in the page was activated.
  - Both can use Tiles to manage layout

Recommendations from the Struts Team

- Existing projects
  - Continue with Struts. If you want Faces support, for the component model only, use the jakarta-struts-faces integration library
- New projects
  - Use Faces. Better technology, and it's a standard.

Web References

- http://java.sun.com/J2EE/javaserverfaces/
  - Official site from Sun
  - Links to content rich discussion forum
- http://www.jsfcentral.com/
  - Links to implementations, renderkits, components
  - News stories featuring JSF.
- http://nagoya.apache.org/wiki/apachewiki.cgi?StrutsMoreAboutJSF
  - Recommendations from Struts development team: new development == use JSF
- http://www.sun.com/jscreator/

Shipping Plans

- Bundled into AppServer 8
- Bundled with Sun Java Studio Creator (project RAVE)
- vendor plans
  - IBM
  - Oracle

Ed Burns

ed.burns@sun.com

Sun microsystems
We make the net work.