The Evolution of Web Application Architectures

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

O'Reilly Open Source Convention
August 1 - 5, 2005

The 7th Annual
O'REILLY
OPEN
SOURCE
CONVENTION

Session Agenda

- Background and Introduction
- Variations On A Theme
- Compare and Contrast Overview:
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- Brief Peeks
- Summary and Q & A

Session Agenda

- **Background and Introduction**
- Variations On A Theme
- Compare and Contrast Overview:
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- Brief Peeks
- Summary and Q & A

Background

- Web tier APIs were among the first standardization efforts outside the base JDK
  - Servlet (initially released in 1996)
  - JSP (initially released in 1999)
- But the standards stopped at the foundations ...
  - Low level abstraction of HTTP APIs
  - Easy mechanism for combining dynamic markup
- And did not address application architecture
  - At least until JavaServer Faces (2004)
- Resulting in much innovation in OSS space

Servlet API – The Foundation

- Abstracting the basic concepts:
  - Servlet, HttpServletRequest, HttpServletResponse
- Adding a concept to deal with statelessness:
  - HttpServletRequest
- Later versions fleshed out functionality:
  - RequestDispatcher, Filter, Event Listeners
- It is possible to create apps with just servlets:
  - writer.println(“<td>Customer Name:</td>”);
  - writer.println(“<td>” + customer.getName() + “</td>”);
- But this approach has several issues

Servlet API – Issues

- All the code is in Java
- Markup generation spread throughout the code
- Difficult to visualize appearance
- Common look and feel hard to create
- Markup generation and business logic intermixed

JSP 1.0 – Inside Out Servlets

- In a dynamic web application, much content is actually static
- Servlets embed static content generation in code
  - writer.println()
- What if we could embed dynamic content generation in static markup instead?
- JSP 1.0 supported three types of markers:
  - Variables (<%! String foo; %>)
  - Expressions (<%= foo %>)
  - Scriptlets (<% foo = “Name: “ + name; %>)

JSP 1.1 – Reduce Embedded Java

- But embedded Java code still has issues
  - Still requires Java familiarity
  - Still intermixes markup and business logic
- JSP 1.1 provides custom tags
  - Page author deals with markup elements
  - Java code abstracted to separate classes
  - JSP Standard Tag Library (JSTL) for common cases
- JSP 2.0 (2003) addresses more of the issues
- But JSP's reputation for intermix could not be easily shaken

Session Agenda

- Background and Introduction
- **Variations On A Theme**
- Compare and Contrast Overview:
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- Brief Peeks
- Summary and Q & A

Web Application Frameworks

- While standards were evolving, innovative solutions were explored:
  - Application architecture frameworks
  - User interface component models
- To meet specific needs:
  - “Hello, world” examples do not help build real apps
  - Most developers did not wish to deal with low level server functionality
  - Many people building web apps were newcomers to Java, as well as newcomers to the web

Variations On A Theme

- Stepping away from the nitty gritty details, these frameworks generally offer a variety of solutions to some common problems
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- It is useful to compare approaches
  - There are lessons to learn from all frameworks

Frameworks To Look At Today

- Struts (http://struts.apache.org)
- WebWork (http://www.opensymphony.com/webwork)
- Spring MVC (http://www.springframework.org)
- Tapestry (http://jakarta.apache.org/tapestry)
- JavaServer Faces
  - (http://java.sun.com/j2ee/javaserverfaces)
- With brief peeks at:
  - Beehive (http://incubator.apache.org/beehive)
  - Cocoon (http://cocoon.apache.org)
  - Shale (http://struts.apache.org/shale)

But Where Is My Favorite???

- An in depth comparison of all the relevant frameworks:
  - Is a PHD thesis, not a 45 minute presentation
  - Requires someone with in depth knowledge of all of the frameworks
  - Experts in the covered frameworks will undoubtedly tell me when I botch the following descriptions :-)
- But you should be able to examine your favorites on the same criteria, to see how they compare

Session Agenda

- Background and Introduction
- Variations On A Theme
- **Compare and Contrast Overview:**
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- Brief Peeks
- Summary and Q & A

O'REILLY

Overall Application Architecture

- All the frameworks like the basic idea of a "Model View Controller" architecture
  - Although Smalltalkers and people building rich client apps still bristle at web folks usurping the term :-)
- More modern name for the design pattern used is *front controller*
- Key feature: all requests into the application flow through a common application level controller

Overall Application Architecture

- Primary controller entry points:
  - Struts – ActionServlet
  - WebWork – ServletDispatcher
  - Spring MVC – DispatcherServlet
  - Tapestry – ApplicationServlet
  - JSF – FacesServlet
- We will see a couple of variations in our “quick peeks” later on

Static and Dynamic Markup

- Basic implementation characteristics:
  - JSP supported or not?
  - Alternative approaches supported or not?
- Struts:
  - Rich support for JSP (including custom tags)
  - Third party mixins for Velocity, others
  - Layout Management with Tiles
- WebWork:
  - Rich support for JSP, Velocity
  - Layout management with SiteMesh, others

Static and Dynamic Markup

- Spring MVC:
  - OOB support for JSP, Velocity
  - Easy rendering plugin for others
  - Layout management with SiteMesh, Tiles
- Tapestry:
  - Prefers HTML markup with “jwcid” tags linking to component definitions
  - Layout management with SiteMesh

Static and Dynamic Markup

- JavaServer Faces:
  - Rich support for JSP
  - API supports *ViewHandler* extensibility for alternative approaches
  - Layout management with Tiles, layout components

View Tier Component Model

- Basic concepts to understand:
  - How are view tier elements represented?
  - How are view tier elements bound to model data?
  - How are conversion and validation handled?
- Struts:
  - View tier state represented in an *ActionForm*
    - Conversion is the application's responsibility
  - JSP tags for rendering common HTML elements
  - Primitive data binding syntax (BeanUtils)
  - Jakarta Commons validator (client + server side)

View Tier Component Model

- WebWork:
  - View tier state represented as typesafe properties in an ActionSupport subclass
  - JSP tags for rendering common HTML elements
  - Data binding via Object Graph Navigation Language
  - XWork validation framework (server side)
- Spring MVC:
  - Variety of strategies for page level controllers
  - Variety of strategies for view resolution
  - Data binding via JSTL EL (for JSP pages)
  - Validation set up in mapping to actions

View Tier Component Model

- Tapestry:
  - UI components with typesafe properties
  - HTML markup with “jwcid” references to components
  - Data binding via OGNL or several other options
  - Validation via delegate bound to form
- JavaServer Faces:
  - UI components optionally bind to backing bean properties with typesafe accessors
  - In JSP, custom tag per component
  - Data binding via JSF EL (superset of JSTL EL)
  - Validation via validators attached to components

Mapping Requests To Business Logic

- Key features to evaluate:
  - Logical mapping from request URL to logic class
  - Standard request processing lifecycle
  - Customizations that are possible
- Struts:
  - XML configration maps URL to *Action* instance
    - Also has associated form bean for view state
    - Actions are singletons, so no instance variables
  - Standard *RequestProcessor* implementation
  - Customization by subclassing
    - This is changing in Struts 1.3, using Commons Chain

Mapping Requests To Business Logic

- WebWork:
  - Uses XWork facilities to map requests to Actions
    - Form properties set on same Action instance
      - Works because actions are per-request instances
  - Configuration for default processing flow provided
  - Customize via interceptor stacks, other techniques
- Spring MVC:
  - Flexible strategies based on configured controllers
  - Standard controller implementations provided
  - Customize via IoC configuration of controllers

Mapping Requests To Business Logic

- Tapestry:
  - Tapestry *ApplicationServlet* manages lifecycle
  - Requests are mapped to *listeners* on a Java class via suitable component property values
  - Listeners implemented in a page class with *abstract* typesafe getters for corresponding form properties
  - Customize page class behavior by injecting configuration information and services

Mapping Requests To Business Logic

- JavaServer Faces:
  - JSF manages a standard request lifecycle
    - Customized via phase listeners
  - Requests are mapped to *action* methods on some backing bean class
    - No particular base class needed
  - Action methods typically implemented on backing bean class with either (or both):
    - Typesafe values for form properties
    - Bindings to component instances for direct manipulation
  - Customize action behavior by injecting configuration information and services

Model Tier Resource Access

- Typical features available:
  - Standard J2EE resource access API (JNDI)
  - Dependency injection or IoC facilities
  - Integration with alternative frameworks
- Struts:
  - “Bring your own model” (BYOM :-)
- WebWork:
  - XWork component framework, Spring, Pico

Model Tier Resource Access

- Spring MVC:
  - Spring, HiveMind, XWork
    - Spring used internal to the framework as well
    - Robust implementations for many external services
- Tapestry:
  - HiveMind, Spring
    - HiveMind used internal to the framework as well
- JavaServer Faces:
  - Managed Beans
    - Can integrate with other DI/IoC frameworks via extensibility APIs

Page Navigation

- Approach alternatives:
  - Logical outcomes versus view identifiers
  - Mapping to alternative view technologies
- Struts:
  - Each *Action* returns an *ActionForward*
  - An *ActionForward* is logically mapped to a view
    - Globally or per-*Action*
  - Default rendering via RequestDispatcher.forward()

Page Navigation

- WebWork:
  - Each *Action* returns a (String) result
  - A String is logically mapped to a view
    - Globally or per-*Action*
  - Can navigate to a variety of destinations
    - Action, RD.forward(), ...
- Spring MVC
  - Controller returns a *ModelAndView*
  - *ViewResolver* maps view to specific technology
    - XML, ResourceBundle, URL, Velocity, ...

Page Navigation

- Tapestry:
  - *Listener* returns void (stay on same page), String (URL), or instance of *IPage* representing the new page to be rendered
  - Values for *IPage* returns can be injected
- JavaServer Faces:
  - *Action* method returns logical outcome
  - *NavigationHandler* maps outcome to next view id
    - Default uses current viewId and action method also
  - *ViewHandler* creates next view
    - Default implementation: RequestDispatcher.forward()

Session Agenda

- Background and Introduction
- Variations On A Theme
- Compare and Contrast Overview:
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- **Brief Peeks**
- Summary and Q & A

Brief Peeks

- Beehive:
  - Just graduating from incubation at Apache
  - Presumes Java 5 (“Tiger”) as base platform
    - Aggressive use of annotations vs. configuration
  - Three major components:
    - NeUI Page Flow – Annotations driven web framework built on top of Struts
    - Controlls – Lightweight metadata based component framework
    - Web Services – Implementation of JSR-181, annotations driven web services

Brief Peeks

- Cocoon:
  - Very different focus from other Java frameworks
  - Build XML-based pipelines for
    - Processing incoming requests
    - Composing rendered response
  - Embedded Rhino (JS interpreter) for continuations

Brief Peeks

- Shale:
  - Accepted as a Struts community sub-project
  - Approaching 1.0.0 milestone release
  - Architecturally an *extension* of JSF
    - Avoid implementating redundant features
    - Leverage extensibility points
  - Add value to base JSF 1.x platform
    - ViewController architecture (page level controllers)
    - Dialogs (like Spring Web Flow)
    - Commons Validator and Tiles integrations
    - Clay Plug-in (Tapestry-like views)

Session Agenda

- Background and Introduction
- Variations On A Theme
- Compare and Contrast Overview:
  - Overall Application Architecture
  - Static and Dynamic Markup
  - View Tier Component Model
  - Mapping Requests to Business Logic
  - Model Tier Resource Access
  - Page Navigation
- Brief Peeks
- **Summary and Q & A**

Summary and Q & A

- We've briefly reviewed five popular Java based web application frameworks
  - And peeked at three more
- Provided a taxonomy of key architectural features on which frameworks can be compared
- Each framework has many additional (and sometimes unique) features to recommend it
- Each framework is worthy of consideration
  - For use, for learning, or for both
- Q & A

The 7th Annual

O'REILLY®

OPEN SOURCE CONVENTION

AUGUST 1-5 2005

The 7th Annual

O'REILLY®

OPEN SOURCE CONVENTION

AUGUST 1-5 2005