THILO FROTSCHER
Software Architect & Trainer

Enterprise Java, Web Services and XML

Java Web Frameworks

- Overview of latest trends -

Thilo Frotscher

thilo@frotscher.com
http://www.frotscher.com

PLEASE NOTE:

The following slides document the results of an effort to find the most suitable web framework for a particular organization and a specific set of requirements.

Therefore, some of the statements you'll find in this presentation are very subjective. This applies in particular to the slides at the end of each section that summarize the pros & cons of a specific framework. When assessing the same framework, you may find that some of the cons we've found are actually pros in your case and vice versa.

As in most cases, there is no one-fits-all solution that is better than any other. It all depends on your specific requirements.

Agenda

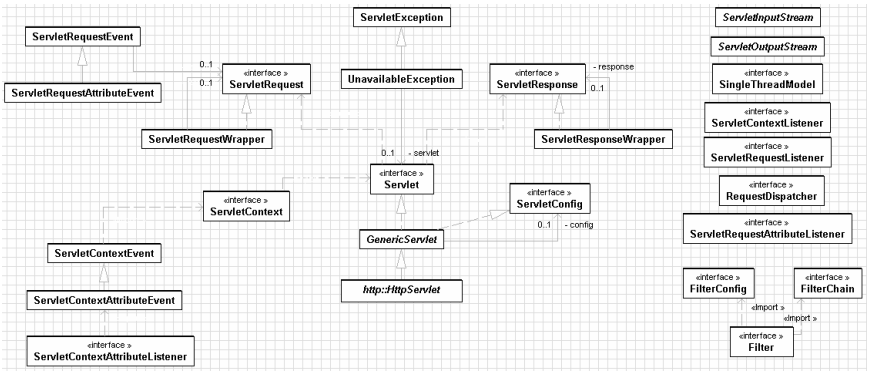
- Brief history of Web Technologies
- Action-based frameworks
- Component-based frameworks
- Summary

Brief history of Web Technologies

- First web servers: static content only
- CGI (Common Gateway Interface)
  - Extension for dynamic content: write little programs which generate content at request time
  - Typically implemented using Perl, C or shell scripts
  - Problem: spawning a new process for every request is very expensive
- CGI's poor scalability inspired more efficient technologies such as PHP or mod_perl
  - script interpreters integrated directly into web servers
- 1995: J.Gosling introduces idea of Servlets
- 1997: Servlet specification 1.0 finalized

Java Servlets

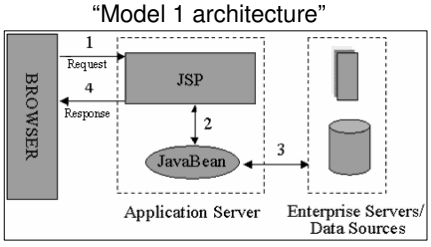
"A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessible via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes."



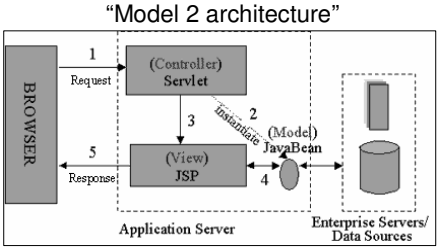
JavaServer Pages

- Generating dynamic content using servlets is not exactly a lot of fun: HTML output with `println()`
- A little later JSP technology was created
  - Define resulting pages as mix of static content and and JSP elements, which construct dynamic content
  - JSPs are compiled into servlets

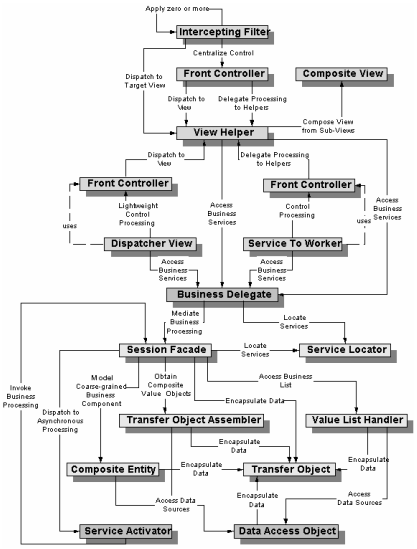
"Model 1 architecture"



"Model 2 architecture"



Java EE Patterns



- Servlets and JSP are just very basic technologies – but how to use them?
  - How many servlets do I need?
  - Where should I put the business logic?
  - How to pick the right view?
- Best practices for Java EE development, published by Sun
- Many experts consider some of these patterns "workarounds for J2EE issues"
- Some patterns are obsolete with Java EE 5
- Why implement the same patterns and architecture again and again?
- Over the years an incredible number of web frameworks has emerged

Java Web Frameworks

"An in-depth comparison of all the relevant Web frameworks is a PHD thesis, not a 45 minute presentation."

(Craig R. McClanahan, creator of the Apache Struts framework)

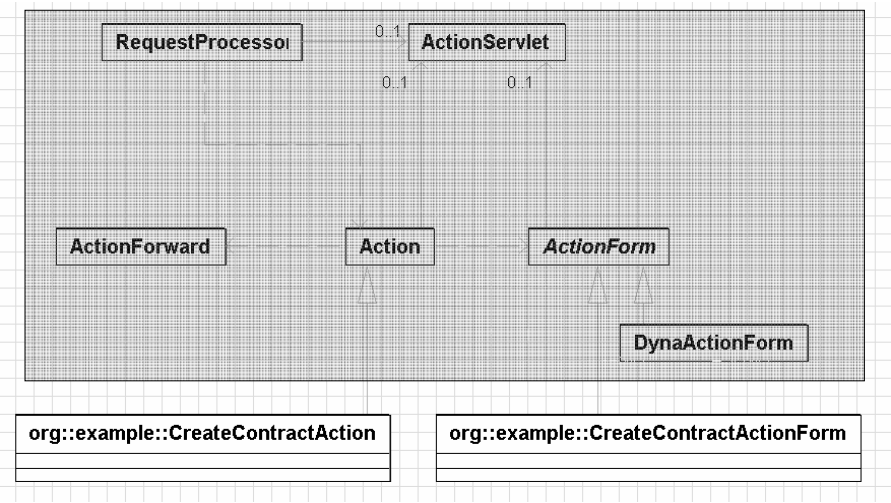
Categorizing Web Frameworks

- Action-based frameworks (ABFs)
  - combine servlets and JSPs, split request processing into *processing* logic and *presentation* logic
  - MVC pattern, or more recently known as the *front controller*, or in Sun parlance Model 2
  - Servlet is the front controller, maps request URL to a unit of work known as an *action*
  - *Action* performs specific functionality for a given URL and maps response into a model
  - Action returns a result, which is mapped (via configuration files) to a JSP to render the view.
  - Prominent examples: Struts, Struts2, Spring MVC

Apache Struts

- Recognized as the most popular Java Web framework of all times
- Has been used in the vast majority of web apps for more than six years
- Lots of experienced developers available
- Widely regarded as outdated, annoying shortcomings identified over several years
- As a result, a number of new ABFs have emerged, based on experiences made

Struts overview



Struts issues

- Inheritance-driven API
  - Action classes have to inherit from `Action`
  - Form classes have to inherit from `ActionForm`
  - Consequences
    - makes it awkward to test or reuse code
    - difficult to keep the domain independent of the web framework (encourages developers to pass down and depend on things like ServletContext deep into the object domain)
- Verbosity and indirection
  - Even simple applications require a number of actions, controllers and form handlers
  - Large apps end up with very lengthy *struts-config.xml* files that in turn drive a demand for visual tools to understand them in maintenance

Struts issues

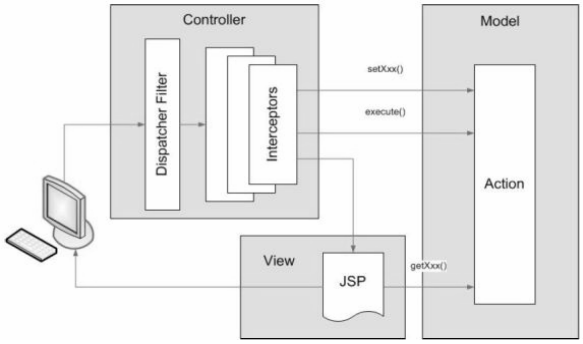
- Complexity
  - walking up to a Struts app and understanding where all the redirects are going is often a challenge
  - keeping Struts apps clean and well organized over time requires effort to fight down entropy

Apache Struts 2

- Originally known as WebWork 2
  - communities have merged, but creator of Struts has left
- Features
  - Reduces XML configuration via intelligent defaults
  - utilizes annotations and “convention over configuration”
  - No Form classes anymore, Actions are now POJOs
    - increased testability, reduces coupling with the framework
  - Dependency injection
  - Automatic type conversion (HTTP requests → Java)
  - More modular request processing through interceptors
  - Flexible validation framework, decouples validation rules from action code

Architecture

- Struts2 is a pull-MVC (or MVC2) framework
  - slightly different from traditional MVC frameworks
  - Action takes role of the model rather than the controller
  - "Pull" comes from view's ability to pull data from an action, rather than having a separate model object



Code Example

- Actions with a single result / outcome

```
class MyAction {
  public String execute() throws Exception {
    // do something meaningful
    return "success";
  }
}
```

```
<action name="myAction" class="com.example.MyAction" >
   <result>view.jsp</result>
</action>
```

- Things to note:
  - Action doesn't need to extend another class or implement an interface – it's a simple POJO
  - Action class has one method named `execute`
    - this name is the one used by convention
    - Multi-action (i.e. multi-method) actions are possible, too
  - There are no ActionForm classes anymore!
  - There's no dependency to the Servlet API!

Form data and business logic

- Request and form data
  - This is done following the JavaBeans paradigm
  - Create a setter method on the action class for each request parameter or form value
  - Data type does not always have to be `String`: Struts2 will convert from `String` to the type on the action class
  - Struts2 also handles the navigation of values into more complex objects graphs, i.e. *person.address.postcode*
- Access to Business Services
  - To provide a loosely coupled system, Struts2 uses dependency injection for service objects (i.e. Spring)

Code example

```
public class ProviderSearchAction {
  private SearchService providerSearchService;
  private String providerName;
  private Integer serviceType;
  private List searchResult;
  public void setProviderSearchService(SearchService service) {
    providerSearchService = service;
  }
  public void setProviderName(String providerName) {
    this.providerName = providerName;
  }
  public String getProviderName() {
    return providerName;
  }
  public void setServiceType(Integer serviceType) {
    this.serviceType = serviceType;
  }
  public Integer getServiceType() {
    return serviceType;
  }
  public String execute() {
    searchResult = providerSearchService.search(providerName, serviceType);
  }
}
```

Business service (will be injected)

Form data

Action

Interceptors

- Many features provided by the framework are implemented using interceptors
  - Exception handling
  - File uploading
  - Lifecycle callbacks
  - Validation
- Conceptually the same as servlet filters: provide a way of pre/post-processing around the action
- Can be layered and ordered
- Have access to the action being executed

There's more to come…

- AJAX theme
  - tags look and feel just like standard Struts tags but provide greater interactivity
  - backed by Dojo Toolkit
- Zero Configuration (optionally)
  - further reduce or even eliminate XML configuration with convention and annotation (still experimental)
- Support for Struts 1.x
  - allows you to use existing Struts 1.x Actions and ActionForms in Struts 2 applications

Pros & Cons

+ Action classes can be implemented so that they are independent of Struts2 and Servlet API
+ Less XML, more conventions / annotations for configuration
+ Easy learning curve, especially if you know Struts 1

- Still in an early stage – risk of significant changes?
- Requires Java 5, backported Java 1.4 JARs are available but backward compatibility not assured

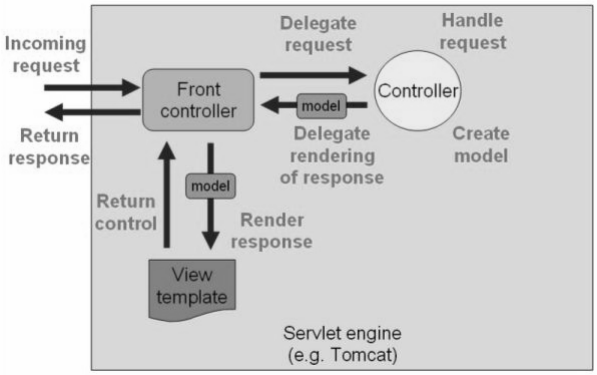
- Documentation needs a bit more detail (but 2 books are available, 3 more coming soon)
- Not a Java EE standard

Spring MVC

- Integral part of the Spring framework
  - Note: any other web frame work can be integrated with the rest of the Spring framework easily
- Features
  - Clear separation of roles: controller, form, validator, command, model can be fulfilled by specialized objects
  - Customizable binding, validation, handler mapping and view resolution
  - Flexible model transfer that supports integration with any view technology
  - Tag library for features such as bindings and themes

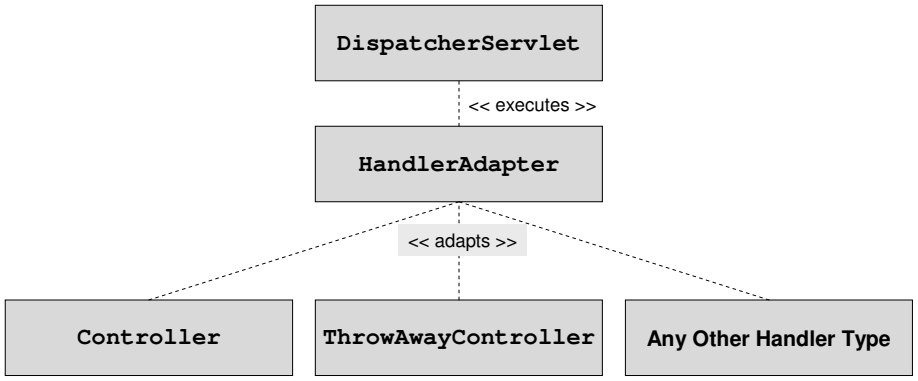
Architecture

- Typical Model2 architecture
- *Handlers* process incoming requests and create a model
  - selection of handlers is based on criteria defined by *handler mappings*
- All kinds of handler types are possible – the default type are *controllers* (`Controller` is an interface defined by the framework)



Handler Adapters

- Any class can be a handler… how is this possible?



Spring controllers

- The Controller interface is very simple:

```
public interface Controller {
  /**
   * Process the request and return a ModelAndView object which the
   * DispatcherServlet will render.
   */

  ModelAndView handleRequest( HttpServletRequest req,
                              HttpServletResponse resp)
                              throws Exception;
}
```

- In addition, the framework provides several implementations of this interface
  - provide common basic functionality, application controllers will typically be subclasses of those
  - e.g. `AbstractController` (introduces the template method `handleRequestInternal`)

Code example

```
public class SampleController extends AbstractController {

 public ModelAndView handleRequestInternal(HttpServletRequest request,
                                           HttpServletResponse response)
                                           throws Exception {
  // do something meaningful...
  // ...
  ModelAndView mav = new ModelAndView("hello");
  mav.addObject("message", "Hello World!");
  return mav;
 }
}
```

```
<bean id="sampleController" class="foo.SampleController">
  <property name="requiresSession" value="true"/>
</bean>
```

- Things to note
  - The view name is hard-coded, other controller types allow external configuration for this
  - Multi-Action controllers are possible as well
  - Controller has dependency to Spring and Servlet API

Command controllers

- This type of controllers creates a *command* object on receipt of a request
- Command is populated with request parameters
- Parameters can be validated by *Validators*
- *PropertyEditors* can be used to transform parameters into specific types or formats
- *BaseCommandController* provides basic functionality, several sub-classes add further stuff
- Powerful and flexible approach, lots of control
- But also lots of configuration, less automatic functionality than for instance in Struts2

Code example

```
public class ProviderSearchFormController extends SimpleFormController {
 private static final DateFormat DATE_FORMAT = new SimpleDateFormat("dd.MM.yyyy");
 private ProviderService providerService;
 public ProviderService getProviderService() {
  return providerService;
 }
 public void setProviderService(ProviderService providerService) {
  this.providerService = providerService;
 }
 @Override
 protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
                                                                  throws ServletException {
  binder.registerCustomEditor(Date.class, "nextMonitoringVisitDate",
                              new CustomDateEditor(DATE_FORMAT, true));
 }
 @Override
 protected ModelAndView onSubmit(Object command, BindException errors) throws Exception {
  ProviderSearchCriteria sc = (ProviderSearchCriteria) command;
  List<Provider> providerList = providerService.search(sc);
  ModelAndView mav = new ModelAndView(getSuccessView());
  mav.addObject("message", "This is a sample message!");
  mav.addObject("providerList", providerList);
  return mav;
}
```

Business service (will be injected)

Form data

Model is stored in a `Map` → abstraction from view technology

Code example

```
<bean id="providerService" class="com.example.services.ProviderServiceImpl"/>
<bean id="providerSearchCritValidator" class="com.example.web.ProviderSearchCritValidator" />

<bean name="/searchProvider.form" class="com.example.web.ProviderSearchFormController">
 <property name="commandClass" value="com.example.web.ProviderSearchCriteria"/>
 <property name="validator"> <ref bean="providerSearchCritValidator"/> </property>
 <property name="providerService"> <ref bean="providerService"/> </property>

 <property name="formView" value="providerSearch" />
 <property name="successView" value="providerSearchResult" />
</bean>
```

- Using command controllers you can
  - use any class as a form/command class
  - use any class to implement the business logic (service)
  - use any class to represent the model for the view
- But you still have to implement a controller, which depends on Spring framework and Servlet API

Mapping requests to controllers

- Define Handler Mappings in Spring configuration file
  - criteria (e.g. URL patterns)
  - handler
  - interceptors (optional)
- `BeanNameUrlHandlerMapping` maps the URLs of incoming requests to the names of Spring beans

```
<bean id="handlerMapping" class="org.springframework.web.servlet.handler.
                                 BeanNameUrlHandlerMapping" />
  <bean name="/searchProvider.form"
        class="com.example.web.ProviderSearchFormController">
 ...
</bean>
```

- Handler Mapping returns *HandlerExecutionChain* to the *DispatcherServlet*, which executes handler and interceptors

Spring MVC vs. Struts 2

- Key differences
  - Spring MVC has more differentiated object roles
    - Supports the notion of a controller, and optional command (or form) object and a model that get's passed to the view
    - Struts2 combines controller and form object in one object. Optionally the same object serves as a model as well.
  - Dependency to APIs and frameworks
    - Spring MVC controller* classes have dependencies to the Spring framework and the Servlet API
    - Struts2 controller / form / model classes can be developed so that they don't have any dependencies

* This could be fixed by using a different handler type.

Pros & Cons

+ Good documentation, could be more detailed though
+ Fairly mature API
+ Maximum control: controller, validator, command, model etc can be fulfilled by a specialized objects
+ Command (form) objects don't have dependencies to the Spring framework
- Spring 2.5 is still compatible with JDK 1.4.2+ and J2EE 1.3+
- Configuration intensive (lots of XML) and almost too flexible
  - requires more development / configuration effort than Struts2 / Stripes
  - Spring 2.5 introduces some "Convention over Configuration" though
- Controllers dependent on Spring and Servlet API
  - Spring 2.5 introduces annotation-based controller configuration
- Not a Java EE standard

Spring 2.5 controller

```
@Controller
public class ClinicController {

 private final Clinic clinic;

 @Autowired
 public ClinicController(Clinic clinic) { this.clinic = clinic; }

 /**
  * Custom handler for the welcome view.
  */
  @RequestMapping("/welcome.do")
  public void welcomeHandler() { }

 /**
  * Custom handler for displaying vets.
  * @return a ModelMap with the model attributes for the view
  */
  @RequestMapping("/vets.do")
  public ModelMap vetsHandler() { return new ModelMap(this.clinic.getVets()); }

  /**
   * Custom handler for displaying an owner.
   * @param ownerId the ID of the owner to display
   * @return a ModelMap with the model attributes for the view
   */
   @RequestMapping("/owner.do")
   public ModelMap ownerHandler(@RequestParam("ownerId") int ownerId) {
     return new ModelMap(this.clinic.loadOwner(ownerId));
   }
```


Stripes

- The ABF many people rave about at the moment
- Makes significant use of several features in Java 5, such as Annotations and Generics
  - also relies heavily on Servlet 2.4/JSP 2.0 features
- Designed to require as little configuration as possible (note the difference to Spring MVC)
- Very little configuration needed to get started
- Some exciting features
  - Auto-discovers Action beans at deployment time by scanning your web application's classpath!
  - automatic auto-wiring of request URLs with action classes and views, automatic input validation
  - strong focus of convention over configuration

Some of these features are now supported by Spring 2.5 as well

Code example

```
public class CalculatorActionBean implements ActionBean {
  private ActionBeanContext context;
  private double numberOne;
  private double numberTwo;
  private double result;

  public ActionBeanContext getContext() { return context; }
  public void setContext(ActionBeanContext context) { this.context = context; }

  public double getNumberOne() { return numberOne; }
  public void setNumberOne(double numberOne) { this.numberOne = numberOne; }

  public double getNumberTwo() { return numberTwo; }
  public void setNumberTwo(double numberTwo) { this.numberTwo = numberTwo; }

  public double getResult() { return result; }
  public void setResult(double result) { this.result = result; }

  @DefaultHandler
  public Resolution addition() {
    result = getNumberOne() + getNumberTwo();
    return new ForwardResolution("/quickstart/index.jsp");
  }
}
```

ActionBeans receive the data submitted in requests and process the user's input

Things to note

- Action beans define both the properties of the form and the processing logic (similar to Struts2)
- There's no need for external configuration to
  - let Stripes know about the ActionBean implementations
  - tie together the JSP page and ActionBean

  All of the information needed is in the Action bean itself!
- `ActionBean` is an interface, so your action beans can extend any class
- Stripes populates the action bean's properties automatically with the values of request parameters
- Automatic type conversion
  - Stripes will happily set nested properties within your existing domain objects, many levels deep, and instantiate objects along the way as necessary.

How do ActionBeans get bound to a URL?

- Default: Stripes examines ActionBeans and determines their URL based on their class and package names:
  - remove any package names up to and including packages called `web`, `www`, `stripes` and `action`
  - remove `Action` and `Bean` (or `ActionBean`) if it is the last part of the class name
  - convert it to a path and appends '.action'
- So `net.sourceforge.stripes.examples.quickstart.CalculatorActionBean` becomes:
  - *examples.quickstart.CalculatorActionBean*
  - *examples.quickstart.Calculator*
  - */examples/quickstart/Calculator.action*
- To overwrite this default all you have to do is annotate the class with the `@UrlBinding` annotation.

Validation

- Simply annotate the properties of the ActionBean themselves, or their respective getters/setters

```
@Validate(required=true)
private double numberOne;
```

- Type validation is done automatically by Stripes
  - it knows that `numberOne` and `numberTwo` were of type `double` and it applies validations that are applicable
- Custom or additional validation can be performed in methods marked with @ValidationMethod

```
@ValidationMethod(on="division")
public void avoidDivideByZero(ValidationErrors errors) {
 if (this.numberTwo == 0) {
  errors.add("numberTwo", new SimpleError("Dividing by zero isn't allowed."));
  }
}
```

Handler methods

- Because method `addition` is public and returns a `Resolution`, Stripes will identify it as a handler method.
- When a request comes to the ActionBean, and the user hit a submit button or image button with the name "addition", this method will be invoked.
- @DefaultHandler annotation tells Stripes which method should be invoked if it cannot determine which button the user hit
  - this often happens because the user hit ENTER instead of clicking a button

A sample ActionBean

```
public class CalculatorActionBean implements ActionBean {
  private ActionBeanContext context;
  private double numberOne;
  private double numberTwo;
  private double result;

  public ActionBeanContext getContext() { return context; }
  public void setContext(ActionBeanContext context) { this.context = context; }

  public double getNumberOne() { return numberOne; }
  public void setNumberOne(double numberOne) { this.numberOne = numberOne; }

  public double getNumberTwo() { return numberTwo; }
  public void setNumberTwo(double numberTwo) { this.numberTwo = numberTwo; }

  public double getResult() { return result; }
  public void setResult(double result) { this.result = result; }

  @DefaultHandler
  public Resolution addition() {
    result = getNumberOne() + getNumberTwo();
    return new ForwardResolution("/quickstart/index.jsp");
  }
}
```

Stripes Tag Library

- First part very closely mirrors the different HTML input tag variants, but comes with nice features
- Examples:
  - <stripes:form> has a focus attribute. If it's left empty, Stripes automatically sets the focus into the first visible field (or the first field with errors when there are errors).
  - Tags for input fields provide functionality for pre-populating, re-populating, and changing display when there are validation errors
- Second part is a "layout" system that is to be the most useful 80% of Tiles in 20% of it's complexity

Interceptors

- Each request goes through a lifecycle of 6 stages
  - each of these can be intercepted
- A class must be written which implements the `Interceptor` interface
- Using the `@Intercepts` annotation you can define which stages an interceptor intercepts
- Interceptors intercept *around* the lifecycle stage
  - they can execute code before it and after it
- The configuration of interceptors is done in *web.xml* (using a special init parameter).

Interceptors - Example

```
@Intercepts({LifecycleStage.ActionBeanResolution,
             LifecycleStage.HandlerResolution,
             LifecycleStage.BindingAndValidation,
             LifecycleStage.CustomValidation,
             LifecycleStage.EventHandling,
             LifecycleStage.ResolutionExecution})
public class NoisyInterceptor implements Interceptor {

  public Resolution intercept(ExecutionContext ctx) throws Exception {
    System.out.println("Before " + ctx.getLifecycleStage());
    Resolution resolution = ctx.proceed();
    System.out.println("After " + ctx.getLifecycleStage());
    return resolution
  }
}
```


Pros & Cons

+ Strong focus on convention over configuration
+ no XML :-)
+ Good documentation, enthusiastic community
+ Action classes don't have to extend Stripes classes (but depend on Stripes through interfaces and annotations)
+ Probably the framework with the best productivity

• Configuration **must** be done using annotations, no XML option

- Interceptors are global and cannot be configured per ActionBean
- rather small user community (yet) compared to Spring MVC and Struts
- hard-coded URLs in ActionBeans (workarounds possible)
- not a Java EE standard

Categorizing Web Frameworks

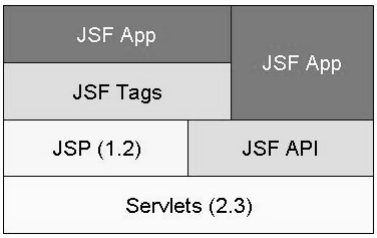
- Component-based frameworks (CBF)
  - Over time web apps have become more complex; it was realized that a page is actually not the logical separation
    - Multiple forms per page
    - Links for content updates
    - Custom widgets
    - ➢ Need processing logic to perform their tasks
  - CBFs provide a close tie between UI components and classes that represent them
  - CBFs are event-driven and more object-oriented than action-based frameworks
  - A component can be an input field, a form, or complex custom widget like a date picker or a foldable tree

Categorizing Web Frameworks

- Component-based frameworks (cont'd)
  - Events, such as form submits, are mapped to methods of the class representing the component
  - Typically the state of UI components is saved when the client requests a new page
    - …and then is restored when the request is returned
  - Additional benefit: approach allows you to re-use visual components across multiple web applications
  - For each of the most popular CBFs there are component libraries available
    - some of them offer very advanced AJAX widgets
  - Prominent examples: JSF, Tapestry, Wicket

JavaServer Faces

- "Framework for building rich user interfaces"
- Standard technology for implementing web applications using Java EE – this has some advantages!
- Initial release of the specification in March 2004
- Latest release: JSF 1.2 (November 2006)
- Has often been criticized for being immature
- Based on Servlet API (and JSP):
  - Set of APIs for representing UI components, managing their state, handling events, validation etc.
  - Two JSP custom tag libraries for expressing UI components within a JSP page



Basic concepts

- Build web applications by assembling reusable UI components in a page
  - Connect these components to an application data source
  - Wire client-generated events to event handlers
  - ➢ Similar to Swing, but event-handlers are on the server-side!
- The UI code runs on the server!
  - responds to events generated on the client
  - renders the user interface back to the client
- A typical JSF application consists of
  - JSP pages with JSF components representing the UI
  - JavaBeans to hold the model data ("managed beans")
  - Configuration files specifying the JSF controller servlet, managed beans and navigation handlers

Basic concepts

- Managed beans (or backing beans)
  - Typical application couples a backing bean with each page in the application
  - Defines the properties and methods associated with the UI components on the page
  - Page author binds the component's value to a bean property using the component's `value` attribute
  - Can also define a set of methods that perform functions, such as validating a component's data

Code example


```
<html>
  <head>
    <title>Hello</title>
  </head>
  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
  <body bgcolor="white">
    <f:view>
      <h:form id="helloForm" >
        <h2>
        Hi. My name is Duke. I'm thinking of a number
        from <h:outputText value="#{UserNumberBean.minimum}"/>
        to   <h:outputText value="#{UserNumberBean.maximum}"/>.
        Can you guess it?
        </h2>
        <h:graphicImage id="waveImg" url="/wave.med.gif" />
        <h:inputText id="userNo" value="#{UserNumberBean.userNumber}"
                     validator="#{UserNumberBean.validate}"/>
        <h:commandButton id="submit" action="success" value="Submit" />
        <p>
        <h:message id="errors1" for="userNo"/>
      </h:form>
    </f:view>
  </body>
</html>
```

UserNumberBean is a *managed bean*

Code example: Managed Bean

```
public class UserNumberBean {
  Integer userNumber = null;
  Integer randomInt = null;
  private int maximum;
  private int minimum;
  String response = null;

  public UserNumberBean() {
    Random randomGR = new Random();
    randomInt = new Integer(randomGR.nextInt(10));
  }

  // getters and setters for userNumber, maximum and minimum omitted

  public String getResponse() {
    if (userNumber != null && userNumber.compareTo(randomInt) == 0) {
      return "Yay! You got it!";
    } else {
      return "Sorry, " + userNumber + " is incorrect.";
    }
  }

  public void validate(FacesContext context, UIInput component) {
    // validate if minimum <= user's guess <= maximum
    // throw ValidatorException if not
  }
}
```

Initial values are specified in the configuration file

Custom validation (default validation available through tags)

Events and Listeners

- Event and listener model is similar to the JavaBeans event model
  - strongly typed event classes and listener interfaces
  - `Event` object identifies the component that generated the event and stores information about the event
- To be notified of an event, the application must
  - provide an implementation of interface `EventListener` (as a class or as a method of the backing bean)
  - register it on the component that generates the event
- Example: event is fired when user clicks a button
  - causes JSF to invoke the listener that processes it
- That way, web-based UIs are event-driven, in contrast to action-based frameworks which are based on requests

JSF configuration

```
<faces-config>
  <application>
    <locale-config>
      <default-locale>en</default-locale>
      <supported-locale>de</supported-locale>
      <supported-locale>fr</supported-locale>
    </locale-config>
  </application>

  <navigation-rule>
    <from-view-id>/greeting.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/response.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <navigation-rule>
    <from-view-id>/response.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/greeting.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
  ...
```

Depending on the outcome of action (or event) handlers in the backing bean, JSF determines the page to display next according to the navigation rules.

JSF configuration

```
  <managed-bean>
    <managed-bean-name>UserNumberBean</managed-bean-name>
    <managed-bean-class>guessNumber.UserNumberBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>

    <managed-property>
      <property-name>minimum</property-name>
      <property-class>int</property-class>
      <value>0</value>
    </managed-property>

    <managed-property>
      <property-name>maximum</property-name>
      <property-class>int</property-class>
      <value>10</value>
    </managed-property>
  </managed-bean>
</faces-config>
```

Component libraries for JSF

- One of the central concepts of JSF: provide an industry standard API for UI components
- Apache Tomahawk (subproject of MyFaces)
  - set of JSF components that go beyond the JSF spec
- Apache Trinidad (subproject of MyFaces)
  - set of over 100 AJAX-enabled JSF components
  - f.k.a. Oracle ADF Faces
- ICEfaces (commercial, free community edition)
  - Demo (Progress Bar, Effects, Drag & Drop, Charts)
- RichFaces (Exadel/JBoss, LGPL licence)
  - Demo (Drop-Down Menu, Drop support, Number slider, Simple Toggle Panel, Tree)

Problems with JSF

- Focused on the view tier of MVC architectures
  - As a result, typical "controller" features or infrastructure seem to be missing
- Several add-on frameworks have emerged which try to solve this issue using different approaches
  - Interestingly the two most important ones were created by two very popular experts for Java technology:
    - Apache Shale: Craig McClanahan (original creator of Struts)
    - JBoss Seam: Gavin King (founder of the Hibernate project)
- General advice: don't use JSF by itself, always add one of the other frameworks
- Problem: you have to learn 2 new frameworks…

Pros & Cons

+ JSF is the industry standard for building web-based UIs with Java
+ tool support is good, e.g. JBossTools, BEA Workshop, JDeveloper, Eclipse plugins from Instantiations, MyEclipse, NetBeans, IntelliJ IDEA
+ very nice component libraries for building rich UIs are available
  → trees, date pickers, drop-down menus, drag & drop, reporting, charts & graphs, navigational components etc.
+ state of the art in JSF is advancing rapidly

• JavaServer Faces 2.0 is on the way (JSR 314)
  – expected to be finalized by mid 2008 (in time with Java EE 6)
• Component libraries and frameworks built on top of JSF solve most issues with "plain-JSF"
  – some of these add-ons are regarded as being of very high quality

- XML-based configuration files can get complex
- technology is still being regarded as immature by many people

Apache Tapestry

+ Very productive once you've learned it
+ Templates are HTML → great for designers
+ Distribution contains >50 components, and it's pretty simple to create new ones
+ Lots of innovation between releases

• Tapestry seems to be mainly developed by a single person

- Tapestry 3 is incompatible with Tapestry 4 and Tapestry 4 is incompatible with Tapestry 5...
- Documentation very conceptual, rather than pragmatic
- Steep learning curve
- Long release cycles
- Major upgrades every year

Apache Wicket

+ Great for Java developers, not web developers
+ Tight binding between pages and views
+ Active community - support from the creators

- Need to have a good grasp of OO
- The Wicket Way
- Absolutely no XML configuration

- Almost everything done in Java
- HTML templates live next to Java code!
  - Requirement: actual html file and class name are equal
  - They also need to be in the same place on the classpath (i.e. same folder)
- Poor online documentation

Conclusion in our project

- Team believed that the component-based approach is conceptually better that the action-based approach
- Nonetheless it was decided to use **Spring MVC** in the near future and to review this decision regularly
- This decision was based on
  - prior experience of developers
    → leverage existing investment in skills
  - current state of component-based technology in general and particularly of JSF
  - time restrictions for upcoming projects
- It was also influenced by SSC guidelines
  - UIs “need to degrade gracefully if JavaScript is disabled”
  - this basically means: no funky AJAX features

Conclusion in our project

- View Technology
  - Continue using **JSP** to be on the safe side
  - Try **FreeMarker** in one project, if successful use it for all future projects
- Layouting
  - **SiteMesh** recommended (rather than Tiles)

Summary

- There are so many web frameworks around, it's really hard to get an overview
- Even learning the differences of the most popular ones takes quite some time
- Some criteria
  - Action-based vs. component-based approach
  - Maturity of the framework
  - Prior knowledge, project deadline, budget
  - Support for Java 1.4.x
  - Quality of documentation, size of community
  - Project requirements (e.g. AJAX)
- A general recommendation is not possible

Java Web Frameworks

Thank you very much
for your attention!

Any questions?