VIRTUAS SOLUTIONS

Spring MVC

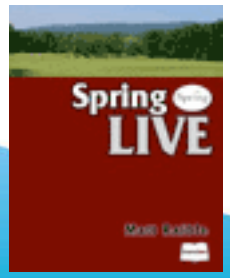
Matt Raible
mraible@virtuas.com

Introductions

- Your experience with Spring?
- Your experience with J2EE?
- What do you hope to learn today?
- Open Source experience: Ant, XDoclet, Hibernate?
- Favorite IDE? Favorite OS?

Who is Matt Raible?

- Developing websites since 1994 - Developing J2EE webapps since 1999
- Committer on several open source projects: AppFuse, Roller Weblogger, XDoclet, Struts Menu, Display Tag
- J2EE 5.0 and JSF 1.2 Expert Group Member
- Author: Spring Live (SourceBeat) and contributor to Pro JSP (Apress)



Part I
Spring MVC Overview

Controller Interface

- Has *handleRequest()* method that returns a ModelAndView
- Base interface for all controllers: *handleRequest()* can be called in unit tests
- **ModelAndView**: a class that holds both Model and a View
- **AbstractCommandController**: use for populating a command object with request parameters
- **MultiActionController**: allows for many methods in same class

ModelAndView

- Many constructor options make it easy to use
- View names are logical names that are configured by ViewResolvers
- Model can be Map or a JavaBean object

```
public ModelAndView(String viewName) {
    this.viewName = viewName;
}

public ModelAndView(String viewName, Map model) {
    this.viewName = viewName;
    this.model = model;
}

public ModelAndView(String viewName, String modelName, Object modelObject) {
    this.viewName = viewName;
    addObject(modelName, modelObject);
}
```

Controller

```
public class UserController implements Controller {
    private final Log log = LogFactory.getLog(UserController.class);
    private UserManager mgr = null;

    public void setUserManager(UserManager userManager) {
        this.mgr = userManager;
    }

    public ModelAndView handleRequest(HttpServletRequest request,
                                      HttpServletResponse response)
    throws Exception {

        return new ModelAndView("userList", "users", mgr.getUsers());
    }
```

JSP

```
<display:table name="users" class="list" requestURI="" id="userList" export="true">
```

ControllerTest

```
public class UserControllerTest extends MockObjectTestCase {
    private UserController c = new UserController();
    private Mock mockManager = null;

    protected void setUp() throws Exception {
        mockManager = new Mock(UserManager.class);
        c.setUserManager((UserManager) mockManager.proxy());
    }

    public void testGetUsers() throws Exception {
        // set expected behavior on manager
        mockManager.expects(once()).method("getUsers")
                   .will(returnValue(new ArrayList()));

        ModelAndView mav =
            c.handleRequest((HttpServletRequest) null,
                            (HttpServletResponse) null);
        Map m = mav.getModel();
        assertNotNull(m.get("users"));
        assertEquals(mav.getViewName(), "userList");

        // verify expectations
        mockManager.verify();
    }
}
```

Configuration

- Configured as a bean definition in *action-servlet.xml* where *action* is the name of the DispatcherServlet in web.xml

```
<bean id="/users.html" class="org.appfuse.web.UserController">
    <property name="userManager"><ref bean="userManager"/></property>
</bean>
```

OR

```
<bean id="userController" class="org.appfuse.web.UserController">
    <property name="userManager"><ref bean="userManager"/></property>
</bean>
```

URL Mapping

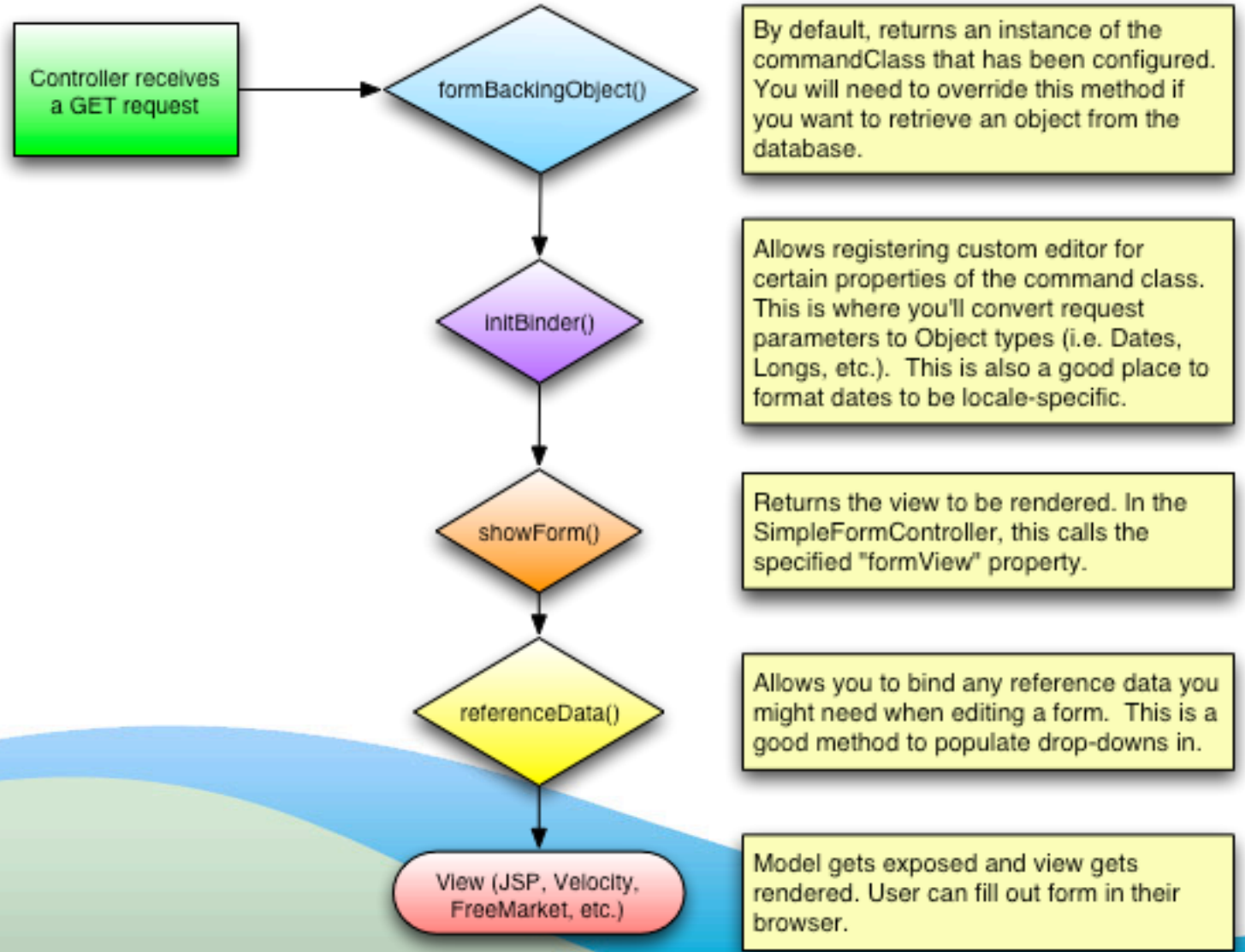
- **BeanNameUrlHandlerMapping** is the default - where URLs are matched to bean names
- **SimpleUrlHandlerMapping** provides central means of configuring URLs and allows interceptors

```
<bean id="urlMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/editUser.html">userFormController</prop>
            <prop key="/users.html">userController</prop>
        </props>
    </property>
</bean>
```

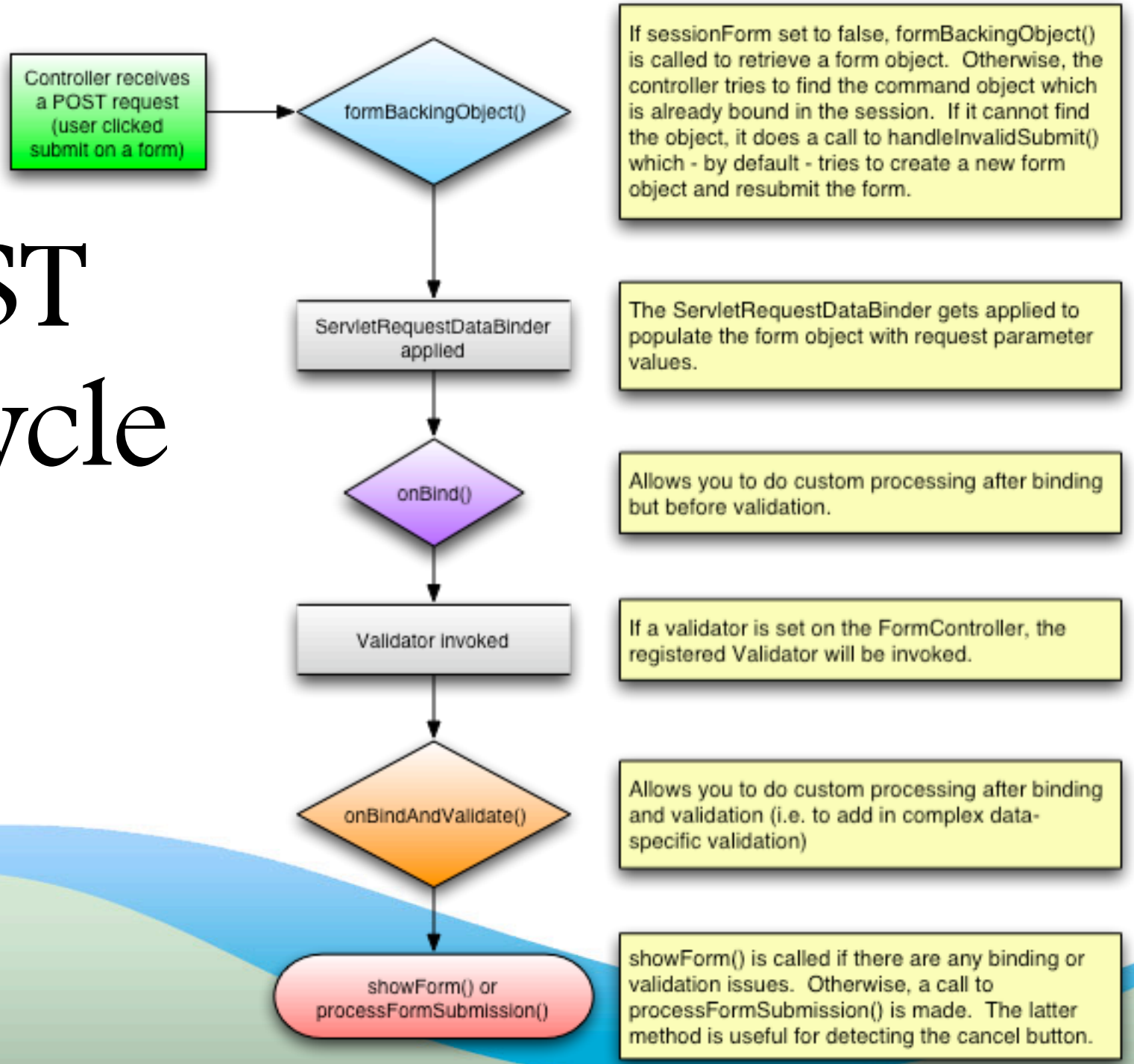
Form Controllers

- **SimpleFormController**: best to use for processing forms
- **AbstractWizardFormController**: use for processing wizards
- **AbstractFormController**: parent of both Simple/AbstractWizardFormControllers. Requires before/after view names to be configured programmatically

HTTP GET Lifecycle



POST Lifecycle



SimpleFormController

- Initializing form object:

```
protected Object formBackingObject(HttpServletRequest request)
        throws ServletException {
    String userId = request.getParameter("id");

    if ((userId != null) && !userId.equals("")) {
        User user = mgr.getUser(userId);

        if (user == null) {
            return new User();
        }

        return user;
    } else {
        return new User();
    }
}
```

SimpleFormController

- Data Binding:

```
protected void initBinder(HttpServletRequest request,
                          ServletRequestDataBinder binder) {
    // convert java.util.Date
    SimpleDateFormat dateFormat = new SimpleDateFormat(getText("date.format"));
    dateFormat.setLenient(false);
    binder.registerCustomEditor(Date.class, null,
            new CustomDateEditor(dateFormat, true));

    // convert java.lang.Long
    NumberFormat nf = NumberFormat.getNumberInstance();
    binder.registerCustomEditor(Long.class, null,
            new CustomNumberEditor(Long.class, nf, true));
}
```

SimpleFormController

- Processing a valid command object:

```
public ModelAndView onSubmit(HttpServletRequest request,
                             HttpServletResponse response, Object command,
                             BindException errors)
        throws Exception {
    if (log.isDebugEnabled()) {
        log.debug("entering 'onSubmit' method...");
    }

    User user = (User) command;

    if (request.getParameter("delete") != null) {
        mgr.removeUser(user.getId().toString());
        request.getSession().setAttribute("message",
                getText("user.deleted", user.getFullName()));
    } else {
        mgr.saveUser(user);
        request.getSession().setAttribute("message",
                getText("user.saved", user.getFullName()));
    }

    return new ModelAndView(getSuccessView());
}
```

SimpleFormControllerTest

```
public class UserFormControllerTest extends AbstractTransactionalDataSourceSpringContextTests {
    private UserFormController c;

    public void setUserFormController(UserFormController userFormController) {
        this.c = userFormController;
    }

    protected String[] getConfigLocations() {
        return new String[] {"/WEB-INF/action-servlet.xml", "/WEB-INF/applicationContext*.xml"};
    }

    public void testEdit() throws Exception {
        // add a test user to the database
        UserManager mgr = (UserManager) applicationContext.getBean("userManager");
        User user = new User();
        user.setFirstName("Matt");
        user.setLastName("Raible");
        mgr.saveUser(user);

        // verify controller can grab user
        MockHttpServletRequest request = new MockHttpServletRequest("GET", "/editUser.html");
        request.addParameter("id", user.getId().toString());
        ModelAndView mv = c.handleRequest(request, new MockHttpServletResponse());
        assertEquals("userForm", mv.getViewName());
        Map model = mv.getModel();
        assertEquals(user, model.get(c.getCommandName()));
    }
}
```

Configuration

```
<bean id="userFormController" class="org.appfuse.web.UserFormController">
    <property name="commandName"><value>user</value></property>
    <property name="commandClass"><value>org.appfuse.model.User</value></property>
    <property name="validator"><ref bean="beanValidator"/></property>
    <property name="formView"><value>userForm</value></property>
    <property name="successView"><value>redirect:users.html</value></property>
    <property name="userManager"><ref bean="userManager"/></property>
</bean>
```

- **TIP**: Set commandClass and commandName in constructor - since they're unlikely to change

View Options

- **JavaServer Pages:** includes support for JSTL (i18n, etc.)
- **Tiles:** allows you to use Tiles like you would with Struts - excellent for page composition
- **Velocity:** includes convenience macros to simplify form development
- **FreeMarker:** macros for form development
- **XSLT, PDF, Excel:** create classes to render view
- **JasperReports:** nice open-source reporting engine

ViewResolvers

- Bean definition that defines how Spring MVC should resolve views
- Provide de-coupling between Controllers and view technology
- Implementations provides for each of the previous view options

ViewResolver Examples

```
<!-- View Resolver for JSPs -->
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/"/>
    <property name="suffix" value=".jsp"/>
</bean>

<!-- Velocity Configurer and View Resolver -->
<bean id="velocityConfig" class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">
    <property name="resourceLoaderPath" value="/"/>
</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">
    <property name="dateToolAttribute" value="date"/>
    <property name="exposeSpringMacroHelpers" value="true"/>
    <property name="requestContextAttribute" value="rc"/>
    <property name="cache" value="true"/>
    <property name="prefix" value="/"/>
    <property name="suffix" value=".vm"/>
</bean>
```

ViewResolver Examples

```
<!-- FreeMarker Configurer and View Resolver -->
<bean id="freemarkerConfig"
    class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">
    <property name="templateLoaderPath" value="/"/>
    <property name="freemarkerSettings">
        <props>
            <prop key="datetime_format">MM/dd/yyyy</prop>
        </props>
    </property>
</bean>

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
    <property name="exposeSpringMacroHelpers" value="true"/>
    <property name="requestContextAttribute" value="rc"/>
    <property name="prefix" value="/">
    <property name="suffix" value=".ftl"/>
</bean>
```

JSP 2.0 + JSTL

```
<spring:bind path="user.*">
    <c:if test="${not empty status.errorMessages}">
    <div class="error">
        <c:forEach var="error" items="${status.errorMessages}">
            <c:out value="${error}" escapeXml="false"/><br />
        </c:forEach>
    </div>
    </c:if>
</spring:bind>

...

<form method="post" action="<c:url value='/editUser.html'/>"
    onsubmit="return validateUser(this)" name="userForm">
<spring:bind path="user.id">
<input type="hidden" name="id" value="${status.value}"/>
</spring:bind>
<table class="detail">
<tr>
    <th><label for="firstName"><fmt:message key="user.firstName"/>:</label></th>
    <td>
        <spring:bind path="user.firstName">
        <input type="text" name="firstName" id="firstName" value="${status.value}"/>
        <span class="fieldError">${status.errorMessage}</span>
        </spring:bind>
    </td>
</tr>
```

Velocity

```
#set($springXhtmlCompliant = true)

...

#springBind("user.*")
#if ($status.error)
<div class="error">
    #foreach ($error in $status.errorMessages)
        ${error}<br/>
    #end
</div>
#end

...

<form method="post" action="#springUrl('editUser.html')">
#springFormHiddenInput("user.id")
<table>
<tr>
    <th><label for="firstName">#springMessage("user.firstName"):</label></th>
    <td>
        #springFormInput("user.firstName" 'id="firstName"')
        #springShowErrors("<br/>" "fieldError")
    </td>
</tr>
```

FreeMarker

```
<#import "/spring.ftl" as spring/>
<#assign xhtmlCompliant = true in spring>

...

<@spring.bind "user.*"/>
<#if spring.status.error>
<div class="error">
    <#list spring.status.errorMessages as error>
        ${error}<br/>
    </#list>
</div>
</#if>

...

<form method="post" action="<@spring.url '/editUser.html'/>">
<@spring.formHiddenInput "user.id"/>
<table>
<tr>
    <th><label for="firstName">
        <@spring.message "user.firstName"/></label>:</th>
    <td>
        <@spring.formInput "user.firstName", 'id="firstName"'/>
        <span class="fieldError">${spring.status.errorMessage}</span>
    </td>
</tr>
```

Part II
Spring MVC vs. Struts

Struts

- Version 1.0 released in June 2001
- De-facto standard for Java web application frameworks because it was first
- Much better than developing with only JSPs and Servlets
- Widely adopted and used in many applications
- Many developers familiar with programming Struts

Spring MVC

- Invented around the same time as Struts, but version 1.0 released in March 2004
- Inversion of Control is built-in, making Controller's easy to test
- Supports wide range of view options: JSP, Tiles, Velocity, FreeMarker, XSLT, PDF, Excel, JasperReports
- Lots of documentation and examples

Actions vs. Controllers

- Action has a single **execute()** method

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
        throws Exception {
}
```

- Controller has a single **handleRequest()** method

```
public ModelAndView handleRequest(HttpServletRequest request,
                                  HttpServletResponse response)
        throws Exception {

}
```

Multiple Methods

- Both frameworks allow you to have multiple methods in an Action/Controller - and control which ones are invoked
- Struts: subclass **DispatchAction**, LookupDispatchAction, or MappingDispatchAction
  - Specify “parameter” in action-mapping
- Spring: subclass **MultiActionController**
  - Configure “methodNameResolver” property

Method Configuration

- Struts:

```
<action path="/user" type="org.appfuse.web.UserAction"
    name="userForm" scope="request" parameter="method" validate="false">
    <forward name="list" path="/userList.jsp"/>
    <forward name="edit" path="/userForm.jsp"/>
</action>
```

- Spring:

```
<bean id="methodResolver"
   class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
    <property name="paramName" value="method"/>
    <property name="defaultMethodName" value="list"/>
</bean>

<bean id="userController" class="org.appfuse.web.UserController">
    <property name="methodNameResolver" ref="methodResolver"/>
    <property name="userManager" ref="userManager"/>
</bean>
```

Configuration

- Struts Actions are configured in /WEB-INF/*struts-config.xml*
- Spring Controllers are configured in /WEB-INF/*action-servlet.xml* (where *action* is the name of the Dispatcher servlet in web.xml)
- Both allow success/failure views to be configured in XML, or specified programmatically

ActionForms vs. POJOs

- **Struts:** forms must be backed by an ActionForm (ValidatorForm for validation) - which is a Java representation of the HTML form
  - Contains all form properties in String form
  - Use BeanUtils.copyProperties to convert from ActionForm to POJO
  - Register Converters with ConvertUtils
- **Spring**: forms use POJOs as "command objects"
  - Register PropertyEditors to handle complex types

View Resolution

- Called **forwards** in Struts and **views** in Spring
- To redirect in Struts, use **redirect="true"** in <forward> configuration
- To redirect in Spring, use **redirect:viewName.html** in "successView" property
- Programmatic configuration:
  - Struts**:** new **ActionForward**(URL, true) or new **ActionRedirect**()
  - Spring**:** new **RedirectView**(viewName)
  - Both allow parameters to be added to redirected URL

Validation

- Can use Commons Validator with both frameworks
- **Struts:** programmatic validation in your ValidatorForm's *validate()* method
- **Spring:** programmatic validation in a Validator class
  - Validators can be chained by specifying a <list> in your Controller's "validators" property

```
public class UserValidator implements Validator {
    private Log log = LogFactory.getLog(UserValidator.class);
    public boolean supports(Class clazz) {
        return clazz.equals(User.class);
    }

    public void validate(Object obj, Errors errors) {
        User user = (User) obj;

        ValidationUtils.rejectIfEmptyOrWhitespace(errors,
            "lastName", "errors.required", "Value required.");        }
    }
}
```

Migrating from Struts to Spring MVC

- web.xml front-dispatcher servlet
- i18n bundle configuration
- action-mappings to bean definitions
- forwards to views
- Struts JSP tags to be Spring JSP tags

web.xml

- Struts:

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

- Spring:

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Mappings to Definitions

- Struts Actions configured as "mappings" in **struts-config.xml**:

```
<action path="/user" type="org.appfuse.web.UserAction"
    name="userForm" scope="request" parameter="method" validate="false">
    <forward name="list" path="/userList.jsp"/>
    <forward name="edit" path="/userForm.jsp"/>
</action>
```

- Spring Controllers configured as "beans" in **action-servlet.xml**:

```
<bean id="userController" class="org.appfuse.web.UserController">
    <property name="userManager"><ref bean="userManager"/></property>
</bean>

<bean id="userFormController" class="org.appfuse.web.UserFormController">
    <property name="commandName"><value>user</value></property>
    <property name="commandClass"><value>org.appfuse.model.User</value></property>
    <property name="validator"><ref bean="beanValidator"/></property>
    <property name="formView"><value>userForm</value></property>
    <property name="successView"><value>redirect:users.html</value></property>
    <property name="userManager"><ref bean="userManager"/></property>
</bean>
```

Internationalization

- Can use JSTL and <fmt:message/> with both
- struts-config.xml:

```
<message-resources parameter="messages"/>
```

- action-servlet.xml:

```
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages"/>
</bean>
```

JSP Tags ~ Struts

```
<html:form action="/user" focus="user.firstName"
    onsubmit="return validateUserForm(this)">
<input type="hidden" name="method" value="save"/>
<html:hidden property="user.id"/>
<table class="detail">
<tr>
    <th><label for="user.firstName">
        <fmt:message key="user.firstName"/>:</label></th>
    <td><html:text property="user.firstName" styleId="user.firstName"/></td>
</tr>
<tr>
    <th><label for="user.lastName" class="required">
        * <fmt:message key="user.lastName"/>:</label></th>
    <td>
        <html:text property="user.lastName" styleId="user.lastName"/>
        <span class="fieldError">
            <html:errors property="addressForm.address"/></span>
    </td>
</tr>
```

JSP Tags ~ Spring

```
<form method="post" action="<c:url value="/editUser.html"/>"
    onsubmit="return validateUser(this)" name="userForm">
<spring:bind path="user.id">
<input type="hidden" name="id" value="${status.value}"/>
</spring:bind>
<table class="detail">
<tr>
    <th><label for="firstName">
        <fmt:message key="user.firstName"/>:</label></th>
    <td>
        <spring:bind path="user.firstName">
        <input type="text" name="${status.expression}"
            id="firstName" value="${status.value}"/>
        <span class="fieldError">${status.errorMessage}</span>
        </spring:bind>
    </td>
</tr>
<tr>
    <th><label for="firstName" class="required">
        * <fmt:message key="user.lastName"/>:</label></th>
    <td>
        <spring:bind path="user.lastName">
        <input type="text" name="${status.expression}"
            id="lastName" value="${status.value}"/>
        <span class="fieldError">${status.errorMessage}</span>
        </spring:bind>
    </td>
</tr>
```

Validation Errors

- **Struts:**

```
<logic:messagesPresent>
    <div class="error">
        <html:messages id="error">
            ${error}<br/>
        </html:messages>
    </div>
</logic:messagesPresent>
```

- **Spring:**

```
<spring:bind path="user.*">
    <c:if test="${not empty status.errorMessages}">
    <div class="error">
        <c:forEach var="error" items="${status.errorMessages}">
            <c:out value="${error}" escapeXml="false"/><br />
        </c:forEach>
    </div>
    </c:if>
</spring:bind>
```

Part III
Advanced Spring MVC

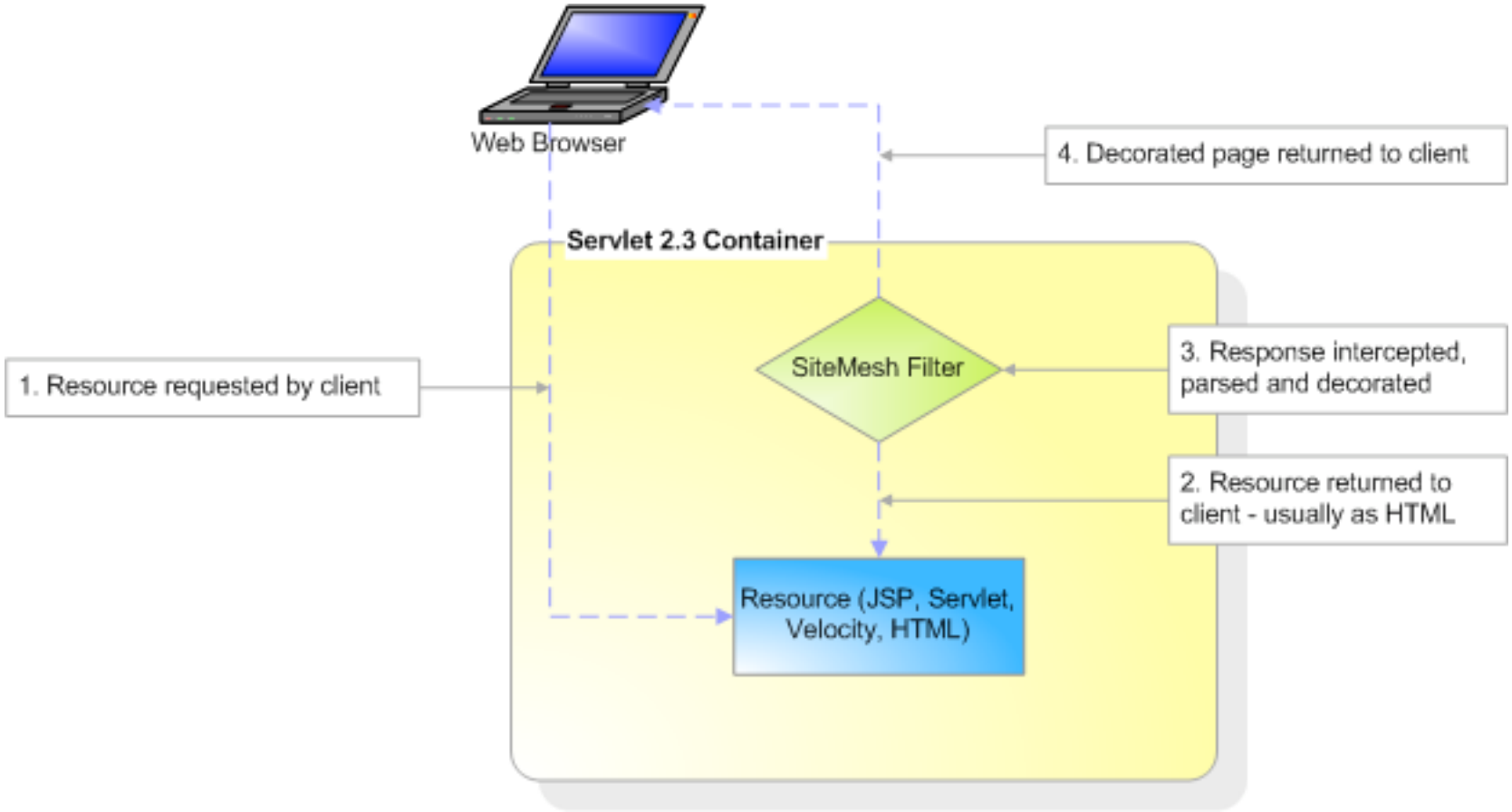
Concept Overview

- Page Decoration and Composition
- Validation with Commons Validator
- Exception Handling
- File Upload
- Data Binding
- Indexed Properties
- Developing Wizards

Page Decoration

- Tiles can be used for page composition
  - Supports JSP and Velocity
- SiteMesh is an excellent tool for page decoration
- SiteMesh can be used with any Java web framework since it's a ServletFilter
  - Supports Velocity, JSP and FreeMarker (maybe Tapestry/JSF in next release)

How SiteMesh Works



SiteMesh: web.xml

```
<filter>
    <filter-name>sitemesh</filter-name>
    <filter-class>com.opensymphony.module.sitemesh.filter.PageFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>sitemesh</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

/WEB-INF/sitemesh.xml

```
<sitemesh>
    <property name="decorators-file" value="/WEB-INF/decorators.xml"/>
    <excludes file="${decorators-file}"/>
    <page-parsers>
        <parser default="true"
            class="com.opensymphony.module.sitemesh.parser.FastPageParser"/>
        <parser content-type="text/html"
            class="com.opensymphony.module.sitemesh.parser.FastPageParser"/>
        <parser content-type="text/html;charset=ISO-8859-1"
            class="com.opensymphony.module.sitemesh.parser.FastPageParser"/>
    </page-parsers>

    <decorator-mappers>
        <mapper class="com.opensymphony.module.sitemesh.mapper.ConfigDecoratorMapper">
            <param name="config" value="${decorators-file}"/>
        </mapper>
    </decorator-mappers>
</sitemesh>
```

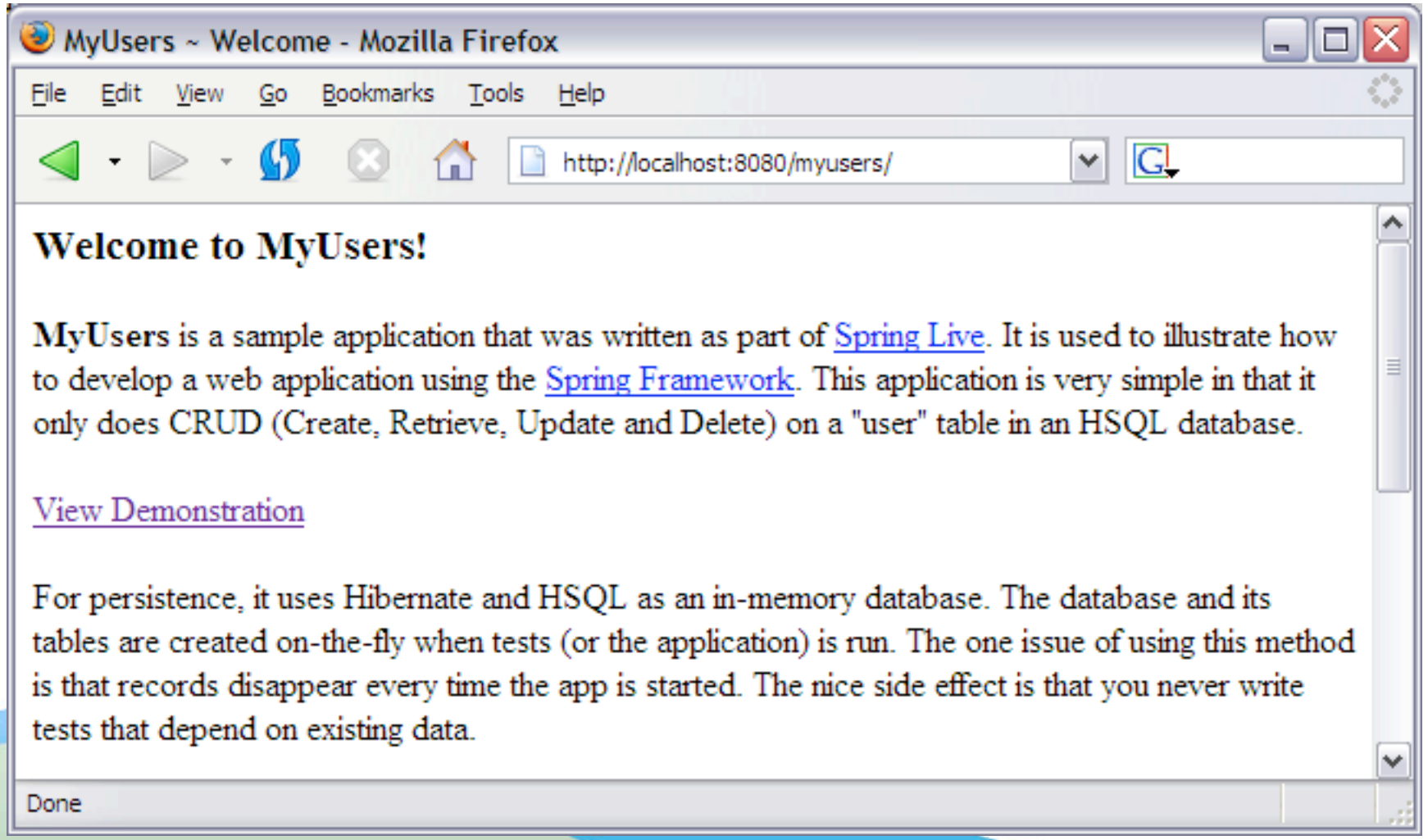
/WEB-INF/decorators.xml

```
<decorators defaultdir="/decorators">
    <excludes>
        <pattern>/demos/*</pattern>
        <pattern>/resources/*</pattern>
    </excludes>
    <decorator name="default" page="default.jsp">
        <pattern>/*</pattern>
    </decorator>
</decorators>
```

Sample Decorator

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<%@ include file="/taglibs.jsp"%>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title><decorator:title default="Equinox"/></title>
    <meta http-equiv="Cache-Control" content="no-store"/>
    <meta http-equiv="Pragma" content="no-cache"/>
    <meta http-equiv="Expires" content="0"/>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <link href="${ctx}/styles/global.css" type="text/css" rel="stylesheet"/>
    <link href="${ctx}/images/favicon.ico" rel="SHORTCUT ICON"/>
    <script type="text/javascript" src="${ctx}/scripts/global.js"></script>
    <script type="text/javascript" src="${ctx}/scripts/fade.js"></script>
    <decorator:head/>
</head>
<body<decorator:getProperty property="body.id" writeEntireProperty="true"/>>
<a name="top"></a>

    <div id="content">
        <%@ include file="/messages.jsp"%>
        <decorator:body/>
    </div>

</body>
</html>
```

Before SiteMesh



After SiteMesh

Commons Validator

- Spring support created by Daniel Miller in April 2004
- Moved from Spring CVS sandbox to Spring Modules project in April 2005
- Validation rules specified in /WEB-INF/*validation.xml*
- Validators (client and server-side) configured in /WEB-INF/*validator-rules.xml*

Spring Configuration

```
<bean id="validatorFactory"
    class="org.springmodules.commons.validator.DefaultValidatorFactory">
    <property name="validationConfigLocations">
        <list>
            <value>/WEB-INF/validation.xml</value>
            <value>/WEB-INF/validator-rules.xml</value>
        </list>
    </property>
</bean>

<bean id="beanValidator"
    class="org.springmodules.commons.validator.DefaultBeanValidator">
    <property name="validatorFactory" ref="validatorFactory"/>
</bean>
```

validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">

<form-validation>
    <formset>
        <form name="user">
            <field property="lastName" depends="required">
                <arg0 key="user.lastName"/>
            </field>
        </form>
    </formset>
</form-validation>
```

Client-side validation

- Form's *onsubmit* handler:

```
<form method="post" action="editUser.html"
    onsubmit="return validateUser(this)" name="userForm">
```

- JavaScript tags after form:

```
<v:javascript formName="user" staticJavascript="false" xhtml="true" cdata="false"/>
<script type="text/javascript" src="<c:url value="/scripts/validator.jsp"/>"></script>
```

- /scripts/validator.jsp

```
<%@ page language="java" contentType="javascript/x-javascript" %>
<%@ taglib uri="http://www.springmodules.org/tags/commons-validator" prefix="v" %>

<v:javascript dynamicJavascript="false" staticJavascript="true"/>
```

Exception Handling

- action-servlet.xml:

```
<bean id="exceptionResolver"
    class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="org.springframework.dao.DataAccessException">
                dataAccessFailure
            </prop>
        </props>
    </property>
</bean>
```

- dataAccessFailure.jsp:

```
<%@ include file="/taglibs.jsp" %>

<h3>Data Access Failure</h3>
<p>
    <c:out value="${requestScope.exception.message}"/>
</p>

<!--
<% Exception ex = (Exception) request.getAttribute("exception");
   ex.printStackTrace(new java.io.PrintWriter(out)); %>
-->

<a href="<c:url value='/'/>">&#171; Home</a>
```

File Upload

```
<%@ include file="/taglibs.jsp"%>

<h3>File Upload</h3>

<c:if test="${not empty model.filename}">
<p style="font-weight: bold">
    Uploaded file (click to view): <a href="${model.url}">${model.filename}</a>
</p>
</c:if>

<p>Select a file to upload:</p>
<form method="post" action="fileUpload.html" enctype="multipart/form-data">
    <input type="file" name="file"/><br/>
    <input type="submit" value="Upload" class="button"
        style="margin-top: 5px"/>
</form>
```

Command Class

```
package org.appfuse.web;

public class FileUpload {
    private byte[] file;

    public void setFile(byte[] file) {
        this.file = file;
    }

    public byte[] getFile() {
        return file;
    }
}
```

FileUploadController

```
public class FileUploadController extends SimpleFormController {

    public FileUploadController() {
        super();
        setCommandClass(FileUpload.class);
    }

    protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
    throws ServletException {
        binder.registerCustomEditor(byte[].class,
                                    new ByteArrayMultipartFileEditor());
    }

    protected ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response,
                                    Object command, BindException errors)
    throws ServletException, IOException {

        FileUpload bean = (FileUpload) command;
        byte[] bytes = bean.getFile();

        // cast to multipart file so we can get additional information
        MultipartHttpServletRequest multipartRequest =  (MultipartHttpServletRequest) request;
        CommonsMultipartFile file =  (CommonsMultipartFile) multipartRequest.getFile("file");

        // process the file
        return new ModelAndView(getSuccessView(), "model", model);
    }
}
```

FileUpload Configuration

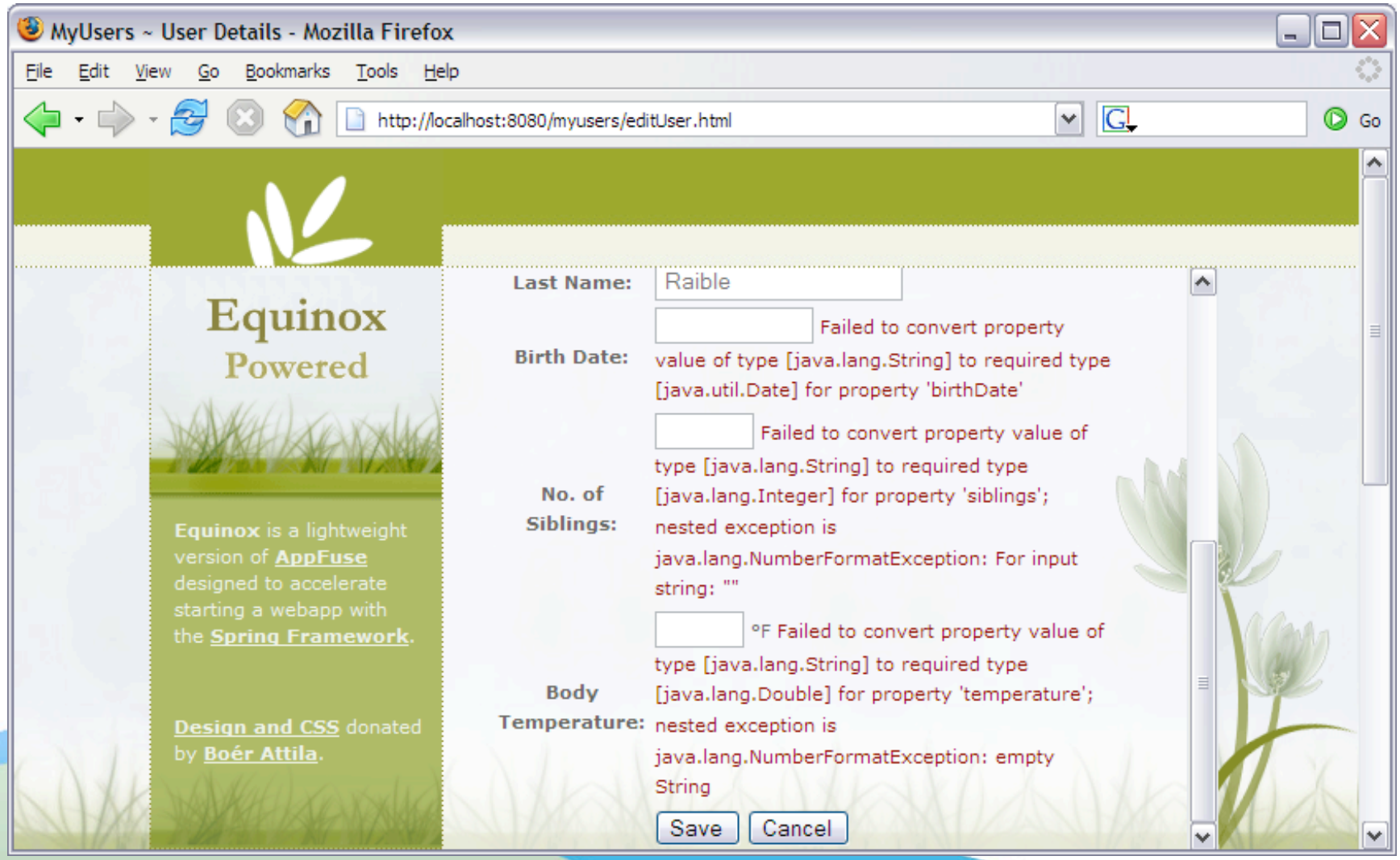
```
<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>

<bean id="fileUploadController" class="org.appfuse.web.FileUploadController">
    <property name="formView" value="fileUpload"/>
    <property name="successView" value="fileUpload"/>
</bean>

<bean id="urlMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            ...
            <prop key="/fileUpload.html">fileUploadController</prop>
        </props>
    </property>
</bean>
```

Data Binding

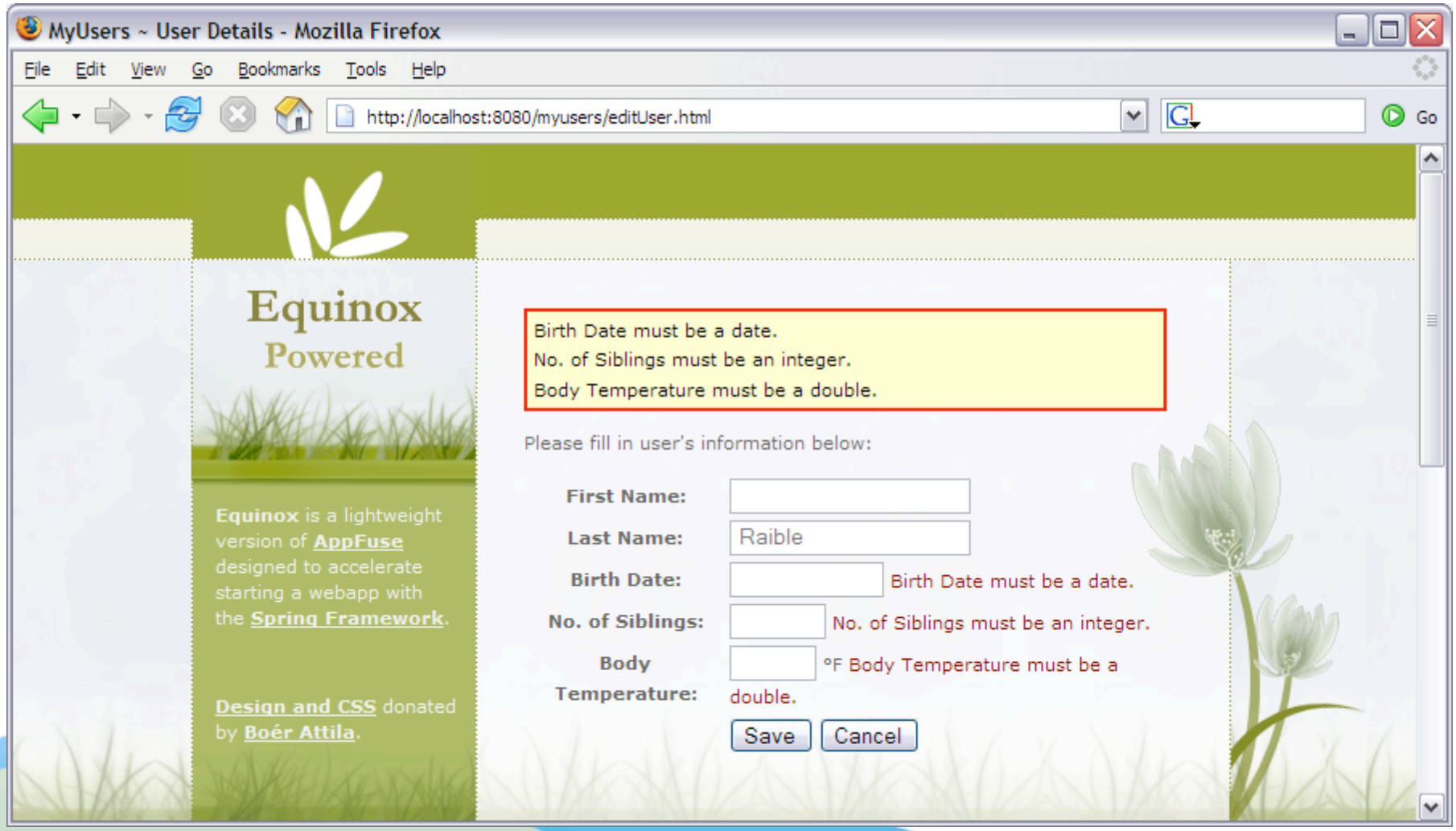
- Complex types (i.e. Integer, Double, Date) require you to register a custom **PropertyEditor**
- Custom PropertyEditors can be registered in *initBinder()* method of SimpleFormController
- Conversion failures can be managed easily, just add messages to your i18n bundle:

```
typeMismatch.java.lang.Integer={0} must be an integer.
typeMismatch.java.lang.Double={0} must be a double.
typeMismatch.java.util.Date={0} must be a date.
```

Before typeMismatch Messages



After typeMismatch Messages



Register Custom Editors

```
protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder) {
    // convert java.lang.Long
    NumberFormat nf = NumberFormat.getNumberInstance();
    binder.registerCustomEditor(Long.class, null,
            new CustomNumberEditor(Long.class, nf, true));

    // convert java.util.Date
    SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");
    dateFormat.setLenient(false);
    binder.registerCustomEditor(Date.class, null,
            new CustomDateEditor(dateFormat, true));

    // convert java.lang.Integer
    binder.registerCustomEditor(Integer.class, null,
            new CustomNumberEditor(Integer.class, nf, true));

    // convert java.lang.Double
    binder.registerCustomEditor(Double.class, null,
            new CustomNumberEditor(Double.class, nf, true));
}
```

Indexed Properties

- A User can have many phone numbers:

```
private Set phones;

public Set getPhones() {
    return phones;
}

public void setPhones(Set phones) {
    this.phones = phones;
}
```

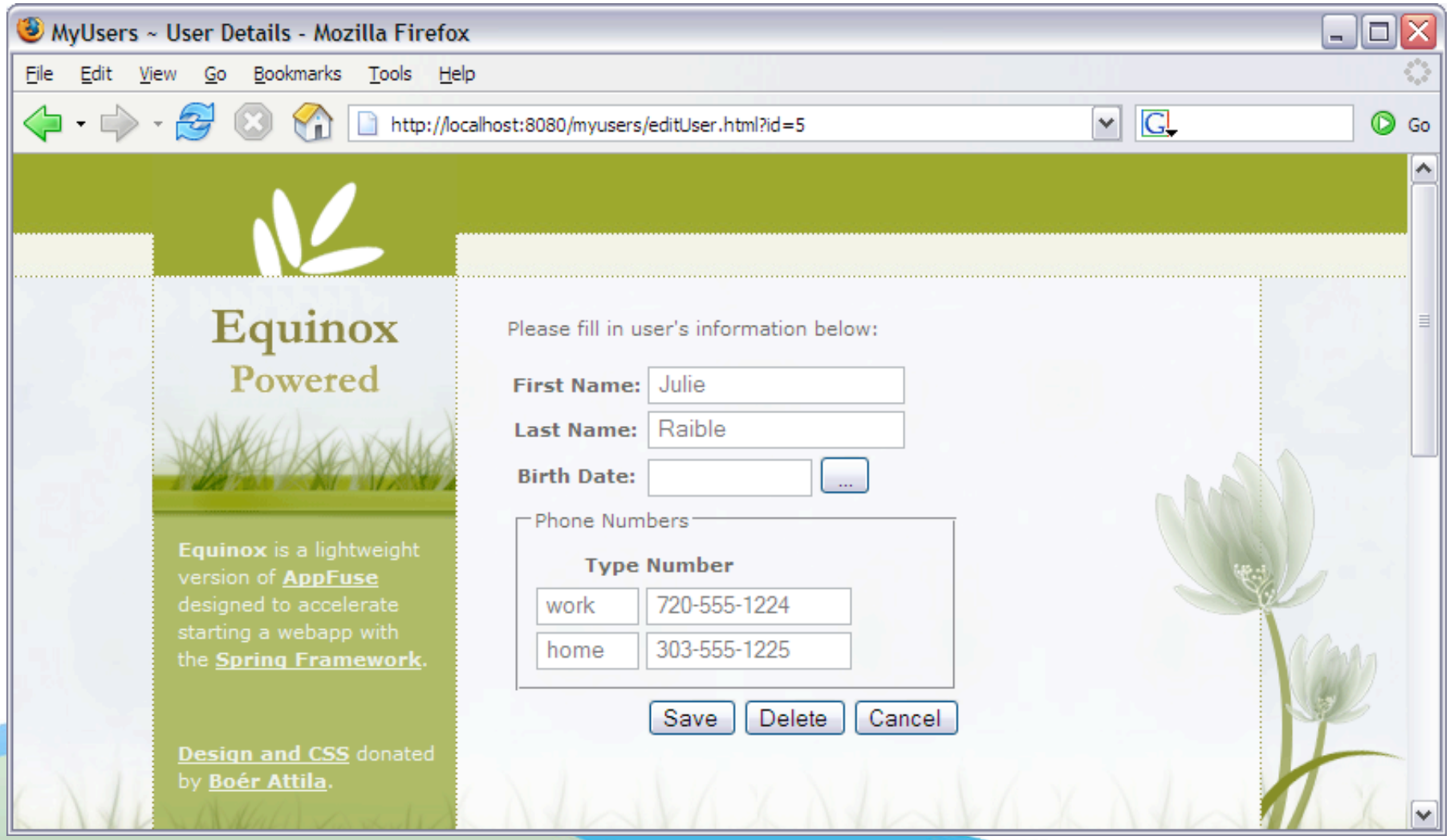
- Hibernate mapping:

```
<set name="phones" cascade="all">
    <key column="user_id"/>
    <one-to-many class="org.appfuse.model.Phone"/>
</set>
```

Indexed Properties, cont.

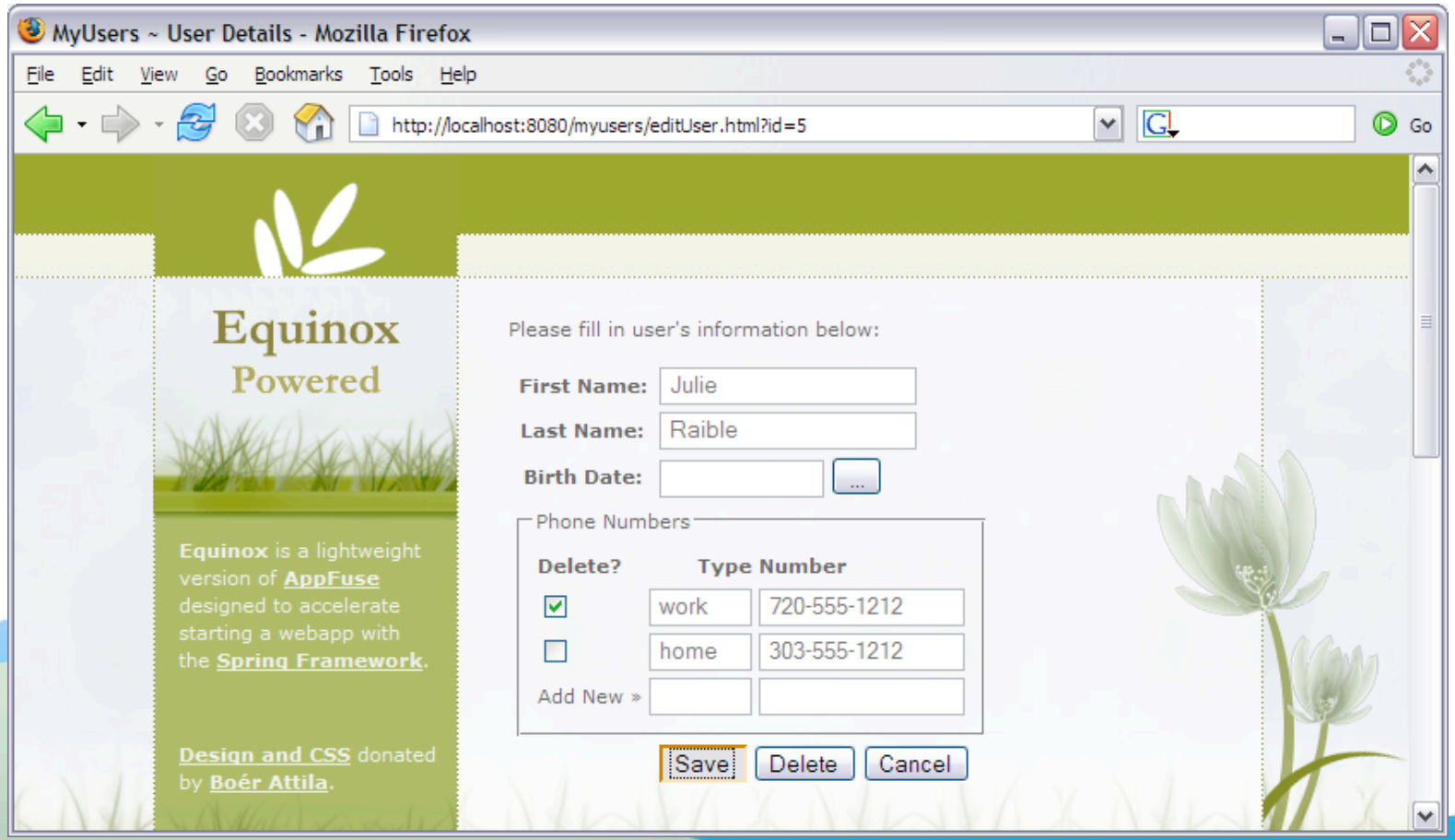
```
<table>
    <tr>
        <th style="text-align: right">Type</th>
        <th style="text-align: left">Number</th>
    </tr>
    <c:forEach var="no" items="${user.phones}" varStatus="s">
    <spring:bind path="user.phones[${s.index}].id">
        <input type="hidden" name="${status.expression}" value="${status.value}"/>
    </spring:bind>
    <tr>
        <td style="text-align: right">
            <spring:bind path="user.phones[${s.index}].type">
            <input type="text" name="${status.expression}" value="${status.value}" size="5"/>
            <span class="fieldError">${status.errorMessage}</span>
            </spring:bind>
        </td>
        <td>
            <spring:bind path="user.phones[${s.index}].number">
            <input type="text" name="${status.expression}" value="${status.value}" size="15"/>
            <span class="fieldError">${status.errorMessage}</span>
            </spring:bind>
        </td>
    </tr>
    </c:forEach>
</table>
```

JSP Screenshot



Add/Delete Properties

- Custom Logic required in *onBind()* for add/delete



AbstractWizardFormController

- Must implement *processFinish()* method to process results of wizard
- "pages" property can be specified in XML:

```
<property name="pages">
    <list>
       <value>wizard/name</value>
       <value>wizard/address</value>
       <value>wizard/phone</value>
    </list>
</property>
```

- Or in your constructor:

```
public WizardFormController() {
    setPages(new String[] {"wizard/name", "wizard/address",
                           "wizard/phone"});
}
```

Determining Workflow

- Page to call is determined by *getTargetPage()* method
- Method uses request parameters to determine page flow

Key Request Parameters

Parameter	Description
_target#	The # is the page's index in the list of pages. It specifies which page the controller should show when the current page is submitted.
_finish	If this parameter is present in the request, the processFinish() method is called and the command object is removed from the session.
_cancel	If this parameter is present in the request, the processCancel() method is called. The default implementation removes the command object from the session and throws a ServletException stating that processCancel() is not implemented.
_page	Indicates the index of the current page. Its recommended you specify this parameter in a hidden field.

UserWizardController

```
public class UserWizardController extends AbstractWizardFormController {

    private UserManager userManager;

    public UserWizardController() {
        setCommandClass(User.class);
    }

    public void setUserManager(UserManager userManager) {
        this.userManager = userManager;
    }

    protected ModelAndView processFinish(HttpServletRequest request,
                                         HttpServletResponse response,
                                         Object command,
                                         BindException errors)
    throws Exception {
        User user = (User) command;
        userManager.saveUser(user);
        request.getSession()
               .setAttribute("message", "User added successfully.");

        return new ModelAndView(new RedirectView("users.html"));
    }
}
```

Bean Definition

```
<bean id="userWizardController" class="org.appfuse.web.UserWizardController">
    <property name="commandName" value="user"/>
    <property name="userManager" ref="userManager"/>
    <property name="pages">
        <list>
           <value>wizard/name</value>
           <value>wizard/address</value>
           <value>wizard/other</value>
        </list>
    </property>
</bean>
```

name.jsp

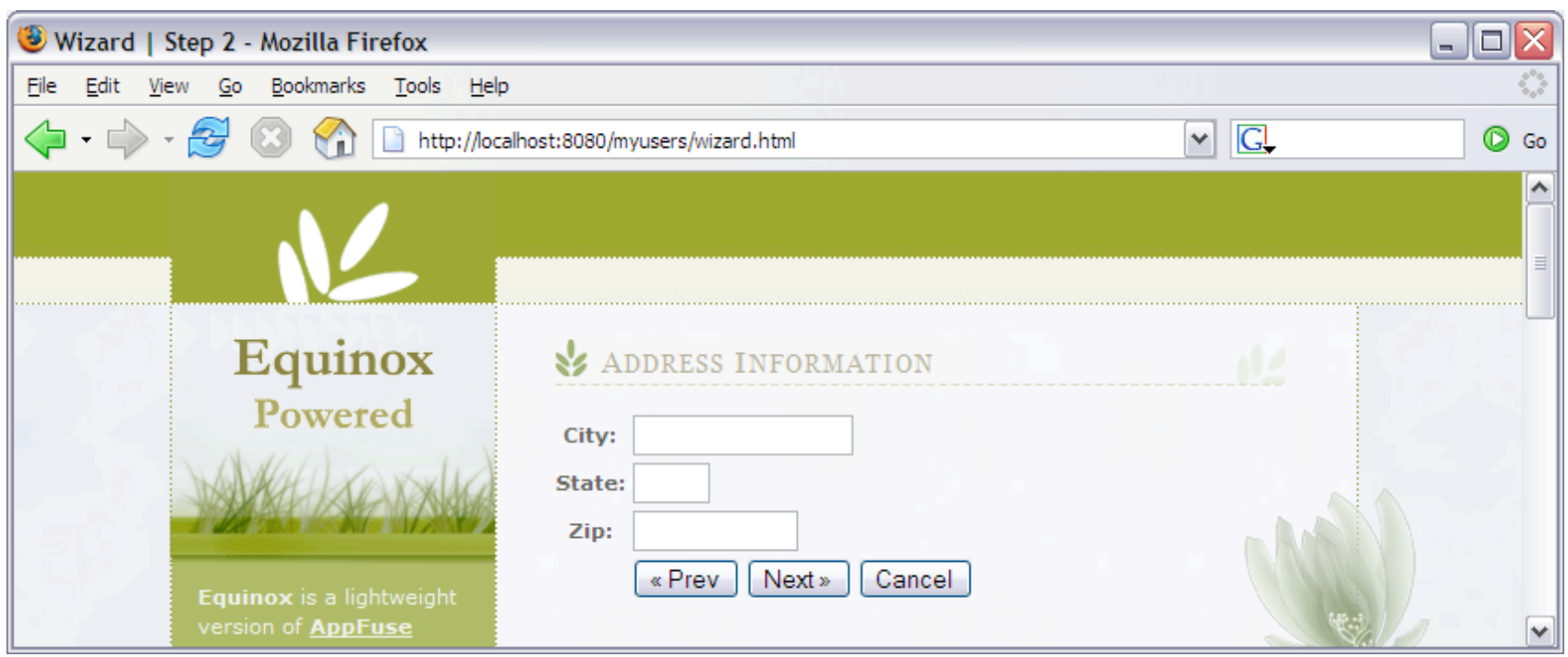
```
<title>Wizard | Step 1</title>

<h3>Name</h3>

<form method="post" action="wizard.html">
<input type="hidden" name="_page" value="0"/>
...
    <input type="submit" class="button" name="_target1" value="Next &raquo;"/>
    <input type="submit" class="button" name="_cancel" value="Cancel"/>
```

- Need to add *processCancel()* to handle cancelling

```
protected ModelAndView processCancel(HttpServletRequest request,
                                     HttpServletResponse response,
                                     Object command, BindException errors)
throws Exception {
    return new ModelAndView(new RedirectView("users.html"));
}
```

address.jsp

```
<title>Wizard | Step 2</title>

<h3>Address Information</h3>

<form method="post" action="wizard.html">
<input type="hidden" name="_page" value="1"/>
...
    <input type="submit" class="button" name="_target0" value="&laquo; Prev"/>
    <input type="submit" class="button" name="_target2" value="Next &raquo;"/>
    <input type="submit" class="button" name="_cancel" value="Cancel"/>
```



other.jsp

```
<title>Wizard | Step 3</title>

<h3>Other Information</h3>

<form method="post" action="wizard.html">
<input type="hidden" name="_page" value="2"/>
...
    <input type="submit" class="button" name="_target1" value="&laquo; Prev"/>
    <input type="submit" class="button" name="_finish" value="Finish"/>
    <input type="submit" class="button" name="_cancel" value="Cancel"/>
```

Validating a Wizard

- Use “page” attribute for Commons Validator
  - validation.xml

```
<field property="lastName" depends="required" page="1">
```

  - name.jsp

```
<input type="hidden" name="page" value="1"/>
```

- Use *validatePage()* method in WizardController to validate programmatically

VIRTUAS SOLUTIONS

Questions?

mraible@virtuas.com