ZK vs. Struts vs. JSF – ein Erfahrungsbericht

Auf welches Pferd setzen Sie?

VON DANIEL SEILER

Als Entwickler Java-basierender Webanwendungen steht man seit jeher vor der Entscheidung: welches Framework passt am besten zu meinen Zwecken? Soll man auf ein altbewährtes Model-View-Controllver-Framework à la Struts setzen, oder sich schon den moderneren JavaServer Faces (JSF) zuwenden? Oder möchte man gar auf der aktuellen Web 2.0-Welle mitreiten und ganz neue Wege gehen?

Viele Unternehmen haben in den letzten Jahren zur Entwicklung der internen, webbasierten Geschäftsapplikationen meist auf Struts-basierende Model-View-Controller-Frameworks gesetzt. Das so zum defacto Standard avancierte Struts gewährleistet dabei eine konsistente Architektur und sorgt für ein akzeptables Maß an Wartbarkeit. Dieser Ansatz funktioniert soweit ganz gut, wenn auch die Komplexität der Benutzeroberfläche und die damit verbundenen Anforderungen an die Entwickler in überschaubarem Rahmen bleiben. HTML, JavaScript, CSS, Java, JSP und verschiedene Tag-Bibliotheken gilt es zu beherrschen, um der Aufgabe, eine ansprechende Benutzeroberfläche zu entwickeln, gewachsen zu sein.

Der Anspruch der Benutzer auf Interaktivität und ein professionelles Erscheinungsbild geht dabei meist zulasten der Wartbarkeit der JSP-Seiten und auch die Produktivität in der Entwicklung lässt häufig zu wünschen übrig. Scheinbar einfache und selbstverständliche Funktionen wie sortierbare Tabellen oder das Verteilen von großen Tabellen auf mehrere Seiten, genannt ‚Paging', erweisen sich häufig als zeitaufwändige Probleme, die jedes Mal von Neuem gelöst werden müssen.

So steht mit den Java Server Faces (JSF) seit 2004 ein Standard zur Verfügung, der die Web-Entwicklung unter Java auf ein nächst höheres Level heben soll und die mittlerweile bekannten Schwachpunkte von Struts ausmerzen. Und gerade im

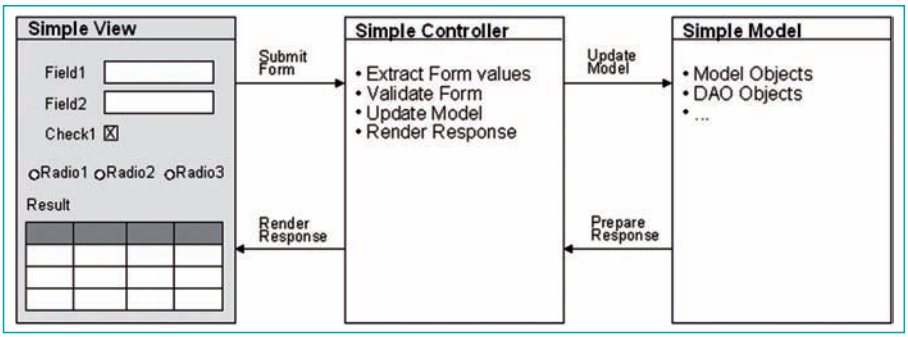


Abb. 1: Architektur der Beispielapplikation

Kontext des aktuellen Web 2.0-Hypes ist die Diskussion um das beste und wahre Framework zur Entwicklung von Java Web-Applikationen neu entbrannt. Das Ziel dabei ist, die Entwicklung solcher Applikationen weiter zu vereinfachen und das Benutzererlebnis zu steigern.

Als Entwickler oder Technologieverantwortlicher von webbasierten Java-Anwendungen steht man also vor der schwierigen Entscheidung, auf welches Framework-Pferd man im nächsten Projekt setzen soll. Welche Webtechnologien werden sich in Zukunft durchsetzen? Sind es die Java Server Faces, JBoss Seam, Grails, Laszlo, Dojo, DWR, Google Web Toolkit, ICEFaces, Ajax4JSF, Spring MVC, Struts, ZK etc. oder eine Kombination aus den verschiedenen Ansätzen?

Um dieser Frage nachzugehen, habe ich beschlossen, eine möglichst einfache Beispielapplikation, die dennoch die wichtigsten Elemente einer ausgewachsenen Geschäftsapplikation enthält, mit drei verschiedenen Frameworks – ZK, Struts und JSF – zu entwickeln.

Verschiedene Gründe haben zu dieser Wahl beigetragen: So ist ZK konsequent komponentenorientiert und beinhaltet eine große Anzahl von Standardkomponenten. ZK verfügt über eine überdurchschnittlich umfangreiche Dokumentation, ist Open Source und wird auf Sourceforge aktiv weiterentwickelt. Ebenfalls für ZK spricht das durchdachte, serverseitige Modell und die Tatsache, dass die zur Darstellung verwendeten ZUL-Seiten auf dem XML User Interface Language (XUL) Standard basieren.

Der folgende Artikel erhebt nicht den Anspruch einer kompletten Analyse der drei Frameworks Struts, Apache MyFaces und ZK. Es ist auch kein Tutorial, wie man Applikationen mit Struts, JSF oder ZK entwickelt. Das Ziel dieses Artikels ist es, anhand einer einfachen, aber aussagekräftigen Beispielapplikation, Stärken, Schwächen und Potenzial der ausgewählten Frameworks zu beleuchten.

Funktionalität der Beispielapplikation

Wie oben erwähnt, soll die Beispielapplikation die wichtigsten Elemente einer ausgewachsenen Geschäftsapplikation enthalten. In der Praxis hat sich gezeigt, dass sehr oft die Hauptaufgabe solcher Applikationen darin besteht, Daten in strukturierter Form zu erfassen, abzuspeichern und wieder darzustellen. Dies ist daher genau die Funktionalität, die unser Beispiel abdecken sollte.

Dem Benutzer wird ein Formular angezeigt, auf dem er Personendaten wie Vorname, Nachname etc. eingeben kann. Sobald er auf *Submit* drückt, werden die Daten an den Server übermittelt und an ein ‚Data Access Object' (DAO) übergeben, welches das Speichern der Daten in eine Datenbank simuliert. Als Resultat wird dem Benutzer die aktuelle Liste der gespeicherten Personen in Form einer Tabelle angezeigt.

Was zeigt uns dieses Beispiel? Zum einen werden verschiedene Standard-Formularkomponenten benutzt, wie z.B. Textfeld, Datumsfeld, Checkbox, Selectbox und Radiobuttons. Zum anderen wird das Übermitteln und wieder Anzeigen der Formulardaten sowie das Aufbauen und Aktualisieren einer Tabelle mit den Resultaten demonstriert.

Rahmenbedingungen

Um die verschiedenen Frameworks auch möglichst aussagekräftig vergleichen zu können, habe ich mir folgende Rahmenbedingungen auferlegt:

1. Es soll jeweils dieselbe Architektur verwendet werden.
2. Code, welcher nicht zur Präsentationsschicht gehört, soll von allen drei Frameworks gemeinsam genutzt werden.
3. Es soll keine Zeit in das Verschönern der Benutzeroberfläche investiert werden.
4. Es soll für die Implementierung bei jedem Framework in etwa gleichviel Zeit investiert werden.

Architektur

In der Praxis hat sich der Model-View-Controller (MVC)-Ansatz bewährt und soll daher als Architektur für alle drei Frameworks zum Einsatz kommen. Dabei beschreibt das Modell die Geschäftsobjekte und die Geschäftslogik, die View präsentiert dem Benutzer das Formular und die Daten, während der Kontroller sich um die korrekte Verarbeitung der Benutzereingaben kümmert (Abb. 1).

Die Daten, die der Benutzer in die Formularmaske eingibt, werden an den Controller geschickt. Dieser extrahiert und validiert die Formulardaten und aktualisiert das Modell der Applikation. In unserem Fall besteht das Modell, das von allen drei Frameworks gemeinsam genutzt wird, aus einem einfachen Personen-Objekt (Listing 1).

Struts

Struts ist das ‚Urgestein' der drei Frameworks. Die Grundidee von Struts ist einfach: Ein zentrales Action-Servlet empfängt die Http-Requests des Benutzers, füllt, falls gewünscht, automatisch die Formulardaten in ein entsprechendes Action-Form Objekt ab und leitet die Anfrage an ein spezifisches Struts-Action-Objekt weiter. Dieses Action-Objekt erfüllt die Rolle des Controllers, indem es die im Form-Objekt gespeicherten Daten in das entsprechende Personen-Objekt abfüllt, die Liste mit allen Personen-Objekten aktualisiert und anschließend die JSP aufruft, die zur Darstellung des Formulars und der Liste mit den Personendaten bestimmt ist. Die Entwicklung der Beispielapplikation mit Struts ist schnell und einfach zu realisie-

ren. Es werden dabei folgende Komponenten entwickelt:

Der Controller, vertreten durch die SimpleAction-Java-Klasse (Listing 2), die SimpleForm-Klasse, die hier nicht abgebildet ist, da sie nur alle Felder des Formulars und die dazugehörigen Getter- und Setter-Methoden enthält sowie die JSP-Seite *simple.jsp* als View (Listing 3). Was ebenfalls noch erstellt werden muss, ist die *struts-config.xml*-Datei, welche die Definitionen der Struts-Actions, der Struts-Forms sowie die Navigationsregeln enthält.

Man beachte, dass die Formulardaten automatisch von Struts in die SimpleAction-Form abgefüllt werden. Man muss jetzt allerdings noch die Formdaten in ein PersonBean-Objekt kopieren. Dies geschieht mithilfe der Jakarta Commons BeanUtils Bibliothek durch folgenden Aufruf:

```
BeanUtils.copyProperties(personBean,form);
```

Das gefüllte PersonBean-Objekt wird dann an das Data Access-Objekt überge-

Listing 1

PersonBean.java

```
package com.ebpm.webdemo.common;

import java.util.Date;

public class PersonBean {
 private String firstname;
 private String lastname;
 private String address;
 private String color;
 private Date birthdate;
 private float weight;
 private boolean married;
 private String rating;

 public String getAddress() {
  return address;
 }
 public void setAddress(String address) {
  this.address = address;
 }
 public Date getBirthdate() {
  return birthdate;
 }
 public void setBirthdate(Date birthdate) {
  this.birthdate = birthdate;
 }
 public String getFirstname() {
  return firstname;
 }
 public void setFirstname(String firstname) {
  this.firstname = firstname;
 }
 public String getLastname() {
  return lastname;
 }
 public void setLastname(String lastname) {
  this.lastname = lastname;
 }
 public boolean isMarried() {
  return married;
 }
 public void setMarried(boolean married) {
  this.married = married;
 }
 public float getWeight() {
  return weight;
 }
 public void setWeight(float weight) {
  this.weight = weight;
 }
 public String getColor() {
  return color;
 }
 public void setColor(String color) {
  this.color = color;
 }
 public String getRating() {
  return rating;
 }
 public void setRating(String rating) {
  this.rating = rating;
 }
}
```

Listing 2

SimpleAction.java

```
package com.ebpm.webdemo.struts;

import java.util.Collection;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.*;
import com.ebpm.webdemo.common.*;

public class SimpleAction extends Action {

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

    String subaction = request.getParameter("subaction");
    // If user pressed 'Cancel' button, return to home page
    if (isCancelled(request)) {
      return mapping.findForward("home");
    }

    if(subaction != null && "addperson".
                                  equals(subaction)) {
      // validate the data
      ActionErrors errors = validate
                          ((SimpleActionForm)form);
      if(errors.size() == 0) {
       // copy the form data into the person bean
       PersonBean personBean = new PersonBean();
       BeanUtils.copyProperties(personBean,form);
       // pass the data to the persistency layer
       PersonDAO.getInstance().addPerson(personBean);
      } else {
       this.addErrors(request, errors);
      }
    }
    // get all the PersonBeans from the persistence layer
    Collection personBeans = PersonDAO.getInstance().
                                        getAllPersons();
    request.setAttribute("personBeans", personBeans);
    // forward to the result page
    return mapping.findForward("success");
  }

}
```

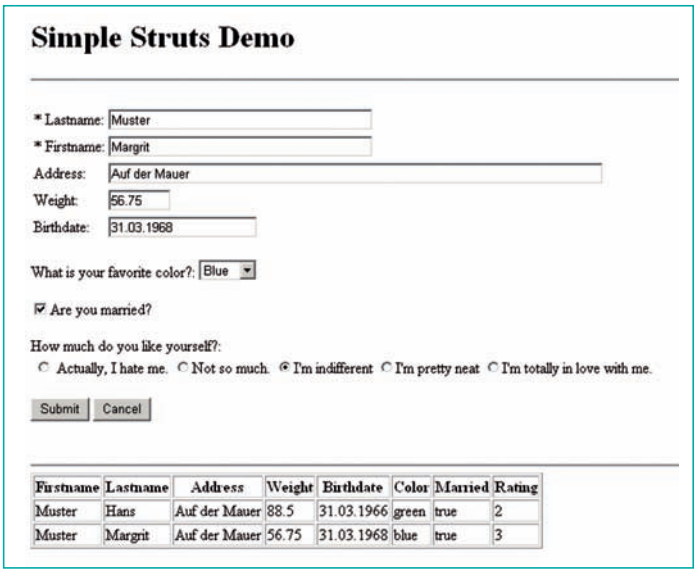


Abb. 2: Struts-Resultat

ben, welches je nach Implementierung die Personendaten in einem Datenspeicher ablegt. Wieder über das Data Access-Objekt werden dann alle gespeicherten Personendaten abgefragt und im Http-Request unter dem Namen *personBeans* abgelegt. Der Controller hat nun seine Arbeit erledigt und leitet die Anfrage weiter an die *simple.jsp*-View zur Darstellung des Resultats.

Auf dem Bildschirm sieht die Ausgabe dann wie in Abbildung 2 aus. Gemäß den definierten Rahmenbedingungen wird keine Zeit ins Design investiert, dementsprechend ist auch das Resultat optisch wenig ansprechend. Meine Beobachtungen beim Entwickeln dieser Beispielapplikation mit Struts lassen sich wie folgt zusammenfassen: Einerseits ist Struts stabil, schnell und nach dem Überwinden der üblichen Konfigurationshürden, intuitiv zu benutzen und ausgereift. Es erlaubt eine saubere Trennung von Darstellung, Kontrollfluss und Geschäftslogik, wobei allerdings auch einiges an Disziplin auf der Entwicklerseite gefordert ist, um diese Trennung wirklich konsequent einzuhalten. Es ist grundsätzlich auch möglich, die JSP mit Geschäftslogik ‚anzureichern' oder die gesamte Geschäftslogik im Controller zu verpacken.

Listing 3

simple.jsp

```
<html:form action="/simple">
<table>
 <tr>
  <td>* Lastname: </td>
  <td><html:text property="lastname" size="40"
                                maxlength="50"/></td>
 </tr>
 <tr>
  <td>* Firstname: </td>
  <td><html:text property="firstname" size="40"
                                maxlength="50"/></td>
 </tr>
 <tr>
  <td> Address: </td>
  <td><html:text property="address" size="80"
                                maxlength="80"/></td>
 </tr>
 <tr>
  <td> Weight: </td>
  <td><html:text property="weight" size="5"/></td>
 </tr>
 <tr>
  <td> Birthdate: </td>
  <td><html:text property="birthdateString" size="20"
                                              /></td>
 </tr>
</table>
<p>What is your favorite color?:
 <html:select property="color">
  <html:option value="red">Red</html:option>
  <html:option value="green">Green</html:option>
  <html:option value="blue">Blue</html:option>
 </html:select>
</p>
<p><html:checkbox property="married"/>Are you
                                        married?</p>
<p>How much do you like yourself?: <br/>
<table>
 <tr>
  <td><html:radio property="rating"
                value="1">&amp;#160;Actually, I hate me.
   </html:radio></td>
  <td><html:radio property="rating" value="2">Not so
                                    much.</html:radio></td>
  <td><html:radio property="rating" value="3">I'm
                            indifferent.</html:radio></td>
  <td><html:radio property="rating" value="4">I'm
                            pretty neat.</html:radio></td>
  <td><html:radio property="rating" value="5">I'm
                                totally in love with me.
   </html:radio></td>
 </tr>
</table>
</p>
<p>
<html:submit>
   <bean:message key="button.submit" />
</html:submit>
<html:cancel/>
</p>
<input type="hidden" name="subaction"
                                value="addperson"/>
</html:form>
...
<table border="1">
 <tr>
  <th>Firstname</th><th>Lastname</th><th>Address</
                                th><th>Weight</th>
  <th>Birthdate</th><th>Color</th><th>Married</
                                th><th>Rating</th>
 </tr>
 <logic:present name="personBeans">
 <logic:iterate id="personBean" name="personBeans">
 <tr>
  <td>${personBean.lastname}</td>
  <td>${personBean.firstname}</td>
  <td>${personBean.address}</td>
  <td>${personBean.weight}</td>
  <td>
   <bean:write name="personBean" property="birthdate"
                                format="dd.MM.yyyy" />
  </td>
  <td>${personBean.color}</td>
  <td>${personBean.married}</td>
  <td>${personBean.rating}</td>
 </tr>
 </logic:iterate>
 </logic:present>
</table>
...
```

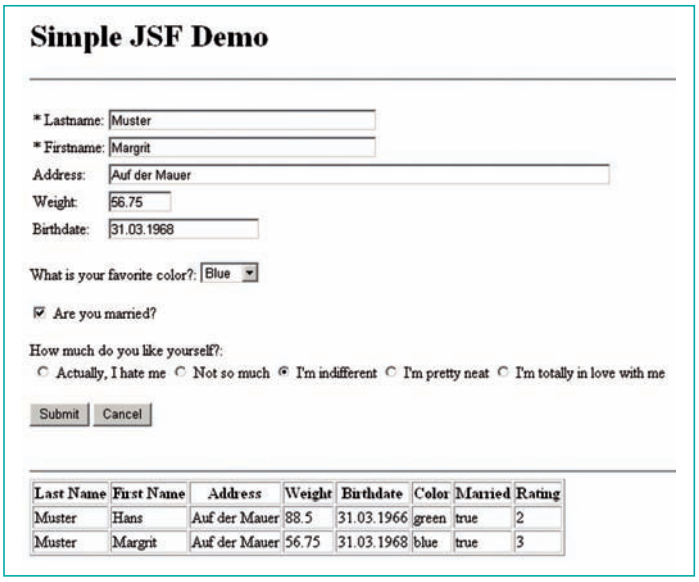
Andererseits handelt es sich bei der JSP-Programmierung immer noch um harte Knochenarbeit. Trotz vielen Bemühungen der Struts- Community wird es ohne fundierte HTML-, Java-, JavaScript- und Tag-Library-Kenntnisse schwierig, eine ansprechende Seite zu gestalten. Es gibt auch wenig (bis keine) Standardkomponenten wie z.B. einen Kalender, den ich mir in dieser Anwendung gewünscht hätte. Mit der neuen Expression Language, die in die Version 2.0 der JSP-Spezifikation Einzug gefunden hat, lassen sich die JSPs zwar relativ kompakt und übersichtlich darstellen – nur wenn die Daten formatiert ausgegeben werden sollen, muss noch auf das ‚alte' *<bean:write>*-Tag zurückgegriffen werden. In unserem Beispiel wird dies am Datums-Feld *birthdate* demonstriert. Das Kopieren der Form-Daten ins Personen-Objekt hinterlässt einen etwas seltsamen Nachgeschmack, obwohl mit der Jakarta Commons BeanUtils Bibliothek diese Aufgabe größtenteils automatisiert werden kann.

JSF (MyFaces)

Kommen wir nun zu JSF, seit Mai 2006 in der Version 1.2 verfügbar. Im Gegensatz zu Struts handelt sich hier um einen ‚richtigen' Standard, der vom Java Community Process autorisiert ist. Was dies für die Entwicklung von Applikationen bedeutet, werden wir in diesem Abschnitt noch sehen. Das Ziel der JSF-Spezifikation ist es, die Webentwicklung, mehr als dies bei Struts der Fall ist, zu vereinfachen und dank einem durchdachten Komponentenmodell von der JSP-Programmierung so weit wie möglich zu abstrahieren.

Was bei Struts die Action-Klassen sind, sind bei JSF die Managed Beans. Die Managed Beans werden in der Datei *faces-config.xml* definiert und können, anders als bei Struts die Actions, direkt aus der View über eine Expression Language angesprochen werden. Das Managed Bean SimpleManagedBean übernimmt in unserem Beispiel die Rolle des Controllers (Listing 4). Als View kommen auch bei den JSF standardmäßig JSPs zum Einsatz (Listing 5). Gemäß dem JSF-Standard wird die Technologie bei der View nicht vorgeschrieben. Das Facelets Projekt [8] liefert hier einige sehr interessante Ansätze, um die JSPs durch eine auf Templates basierende Technologie zu ersetzen. In unserem Beispiel wird die JSP-Seite *simple.jsp* zur Darstellung benutzt.

Abb. 3: JSF-Resultat



Was beim Managed Bean auffällt, ist das Fehlen eines Form-Objektes, wie wir es bei Struts gesehen haben. Die View greift über die Methode *getPersonBean()* direkt auf das *PersonBean* zu und bindet die Attribute dieser Beans an die entsprechenden Komponenten der View. Aus diesem Grund braucht man kein Form-Objekt als Vermittler zwischen View und Controller (Listing 5).

Auch hier verrichtet die Expression Language, welche ein direktes Ansprechen der Managed Beans und deren Methoden erlaubt, sehr gute Dienste. Was ebenfalls auffällt, sind die JSF- Standardkomponenten, wie z.B. die Tabellen-Komponente *<h:dataTable>*. Das manuelle Iterieren über eine Liste von Elementen wie bei Struts entfällt, dafür wird die Liste, welche als Tabelle angezeigt werden soll, über das Managed Bean referenziert und die benötigten Kolonnen definiert. Die Ausgabe auf dem Bildschirm sieht dann aber, wie in Abbildung 3 zu sehen ist, sehr ähnlich der mit Struts erstellten Applikation aus (Abb. 3).

JSF bietet standardmäßig einige Unterstützung, um mithilfe eines so genann-

Listing 4

SimpleManagedBean.java

```
package com.ebpm.webdemo.jsf;

import java.util.Collection;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import com.ebpm.webdemo.common.*;

public class SimpleManagedBean {
 private PersonBean personBean = new PersonBean();

 public PersonBean getPersonBean() {
  return personBean;
 }

 public String addPerson() {
  if(validate(FacesContext.getCurrentInstance()))
  {
   PersonDAO.getInstance().addPerson(personBean);
  }
  return "success";
 }

 public String cancel() {
  return "home";
 }

 public Collection getAllPersons() {
  return PersonDAO.getInstance().getAllPersons();
 }

 ...
}
```

ten Converters das Geburtsdatum in ein Datum-Objekt umzuwandeln und bei der Anzeige wieder entsprechend zu formatieren. Das durchdachte Komponentenmodell sowie das direkte Aufrufen der Managed Beans über die Expression Language bieten einige Vorteile gegenüber Struts. Leider sind die verfügbaren Komponenten noch nicht soweit ausgereift, wie sie sein könnten und das Suchen und Integrieren der passenden Komponente aus den vielen verschiedenen Komponenten-Bibliotheken gestaltet sich als zeitraubendes Unterfangen. Schaut man sich die JSP der JSF-View an, so sieht man, dass immer noch einige HTML-Kenntnisse von Nöten sind und man kommt auch um den Eindruck nicht herum, dass sich bezüglich der Komplexität der Seite gegenüber Struts nicht allzu viel verbessert hat.

ZK

Nach Struts und JSF schauen wir in diesem Kapitel als letztes Framework einen

Listing 5

simple.jsf

```
<html>
...
<body>
<f:view>
<h1>Simple JSF Demo</h1>

<h:messages/>
<h:form id="personForm">
<table>
<tr>
 <td>* Lastname: </td>
 <td><h:inputText id="lastname" value="#
          {SimpleManagedBean.personBean.lastname}"
   size="40"/></td>
</tr>
<tr>
 <td>* Firstname: </td>
 <td><h:inputText id="firstname" value="#
          {SimpleManagedBean.personBean.firstname}"
   size="40"/></td>
</tr>
<tr>
 <td>Address: </td>
 <td><h:inputText id="address" value="#
          {SimpleManagedBean.personBean.address}"
   size="80"/></td>
</tr>
<tr>
 <td>Weight: </td>
 <td><h:inputText id="weight" value="#
          {SimpleManagedBean.personBean.weight}"
   size="5"/></td>
</tr>
<tr>
 <td>Birthdate: </td>
 <td>
  <h:inputText id="birthdate" value="#
          {SimpleManagedBean.personBean.birthdate}"
   size="20">
   <f:convertDateTime type="date" pattern=
                                   "dd.MM.yyyy"/>
  </h:inputText>
 </td>
</tr>
</table>

<p>What is your favorite color?:
 <h:selectOneMenu id="color" value="#
          {SimpleManagedBean.personBean.color}">
  <f:selectItem itemValue="red" itemLabel="Red" />
  <f:selectItem itemValue="green" itemLabel="Green" />
  <f:selectItem itemValue="blue" itemLabel="Blue" />
 </h:selectOneMenu>
</p>

<p>
<h:selectBooleanCheckbox id="married"
   value="#{SimpleManagedBean.personBean.married}"
                                   /> Are you married?</p>
<p>How much do you like yourself?: <br/>
 <h:selectOneRadio id="rating" value="#
          {SimpleManagedBean.personBean.rating}" >
  <f:selectItem itemValue="1" itemLabel="Actually, I
                                   hate me" />
  <f:selectItem itemValue="2" itemLabel="Not so much" />
  <f:selectItem itemValue="3" itemLabel="I'm
                                   indifferent" />
  <f:selectItem itemValue="4" itemLabel="I'm pretty
                                   neat" />
  <f:selectItem itemValue="5" itemLabel="I'm totally in
                                   love with me" />
 </h:selectOneRadio>
</p>
<p>
<h:commandButton id="submit" action="#
   {SimpleManagedBean.addPerson}" value="Submit" />
<h:commandButton id="cancel" action="#
        {SimpleManagedBean.cancel}" value="Cancel" />
</p>
</h:form>
...
<h:dataTable value="#{SimpleManagedBean.allPersons}"
                                   var="row" border="1">
  <h:column>
   <f:facet name="header">
    <h:outputText value="Last Name"/>
   </f:facet>
   <h:outputText value="#{row.lastname}"/>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="First Name"/>
   </f:facet>
   <h:outputText value="#{row.firstname}"/>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="Address"/>
   </f:facet>
   <h:outputText value="#{row.address}"/>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="Weight"/>
   </f:facet>
   <h:outputText value="#{row.weight}"/>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="Birthdate"/>
   </f:facet>
   <h:outputText value="#{row.birthdate}">
    <f:convertDateTime type="date"
                                   pattern="dd.MM.yyyy"/>
   </h:outputText>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="Color"/>
   </f:facet>
   <h:outputText value="#{row.color}"/>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="Married"/>
   </f:facet>
   <h:outputText value="#{row.married}"/>
  </h:column>
  <h:column>
   <f:facet name="header">
    <h:outputText value="Rating"/>
   </f:facet>
   <h:outputText value="#{row.rating}"/>
  </h:column>
</h:dataTable>
</f:view>
</body>
</html>
```

Vertreter der neuen Web-2.0-Generation an. ZK ist ein Open-Source-Projekt und wurde 2004 ins Leben gerufen. Es steht unter der GPL-Lizenz, bietet aber, analog zu MySQL, ein Dual Licensing- Modell an, sodass ZK auch für kommerzielle Applikationen, die nicht unter die GPL gestellt werden sollen, eingesetzt werden kann. Gemäß Wikipedia ist ZK ein Ajax-basiertes Webapplikation-Framework, das eine interaktive Benutzeroberfläche ermöglicht und dies alles ohne JavaScript zu programmieren. ZK wird von der Firma Potix Corporation aktiv unterstützt und weiterentwickelt. Es ist strikt komponentenbasiert, d.h. alles was im Browserfenster angezeigt wird, sind Komponenten. Dabei besteht jede Komponente aus einem serverseitigen Modell und einer im Browser sichtbaren Darstellung. Die Komponenten können dabei entweder in einer XUL-basierten XML-Sprache, in so genannten ZUL-Seiten, beschrieben oder aber komplett in Java programmiert werden. Welche Variante im konkreten Fall gewählt wird, XML oder Java, ist zum Teil eine Geschmacksfrage, aber auch Gegenstand von Erfahrung und Architekturüberlegungen. Dem Autor hat es auf jeden Fall viel Spaß gemacht, nach den vielen JSPs die Benutzeroberfläche, ähnlich wie bei Swing, in Java zu programmieren.

Die Hauptkomponente einer ZK-ZUL-Seite ist die Window-Komponente. Auf der Seite wird diese Komponente durch das <*window*>-Tag beschrieben. Standardmäßig wird das <*window*>-Tag auf dem Server durch die Klasse *org.zkoss.zul.Window* implementiert. Um nun unsere MVC-Architektur in ZK abzubilden, besteht die Möglichkeit, durch das XML Attribut *use* des *window*-Tags eine eigene Version der *org.zkoss.zul.Window*-Klasse zu definieren. In unserem Beispiel ist es die *SimpleWindow*-Klasse in Listing 6, welche die Rolle des Controllers übernimmt. Die Komponenten einer ZUL-Seite können über das jeweilige *id*-Attribut referenziert werden. So bewirkt ein Klicken auf den *Submit*-Button, dass die Methode *addPerson()* der *SimpleWindow*-Klasse aufgerufen wird.

Wie bei den meisten ZK-Requests handelt es sich auch bei dem Aufruf der *addPerson()*-Methode um einen Ajax-Request. Dieser wird asynchron durch

Listing 6

SimpleWindow.java

```
package com.ebpm.webdemo.zk;

import java.text.SimpleDateFormat;
import java.util.*;

import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zul.*;

import com.ebpm.webdemo.common.PersonBean;
import com.ebpm.webdemo.common.PersonDAO;

public class SimpleWindow extends Window
{
 public static SimpleDateFormat sdf = new
                    SimpleDateFormat("dd.MM.yyyy");

 public void onCreate()
 {
  createPersonsGrid();
 }

 public void addPerson()
 {
  // extracting the person data
  Textbox firstName = (Textbox)Path.getComponent
                    ("/simpleWindow/firstname");
  Textbox lastName = (Textbox)Path.getComponent
                    ("/simpleWindow/lastname");
  Textbox address = (Textbox)Path.getComponent
                    ("/simpleWindow/address");
  Textbox weight = (Textbox)Path.getComponent
                    ("/simpleWindow/weight");
  Datebox birthdate = (Datebox)Path.getComponent
                    ("/simpleWindow/birthdate");
  Combobox color = (Combobox)Path.getComponent
                    ("/simpleWindow/color");
  Checkbox married = (Checkbox)Path.getComponent
                    ("/simpleWindow/married");
  Radiogroup rating = (Radiogroup)Path.getComponent
                    ("/simpleWindow/rating");

  PersonBean personBean = new PersonBean();
  personBean.setFirstname(firstName.getValue());
  personBean.setLastname(lastName.getValue());
  personBean.setAddress(address.getValue());
  personBean.setWeight(new Float(weight.getValue().
                    floatValue());
  personBean.setBirthdate(birthdate.getValue());
  personBean.setColor(color.getValue());
  personBean.setMarried(married.isChecked());
  personBean.setRating(rating.getSelectedItem().
                    getValue());
  PersonDAO.getInstance().addPerson(personBean);
  Grid personsGrid = Path.getComponent
                    ("/simpleWindow/personsGrid");
  Rows rows = personsGrid.getRows();
  addRow(personBean,rows,rows.getChildren().size());
 }

 private void addRow(PersonBean personBean, Rows rows)
 {
  Row row = new Row();
  rows.appendChild(row);
  row.getChildren().add(new Label
                    (personBean.getFirstname()));
  row.getChildren().add(new Label
                    (personBean.getLastname()));
  row.getChildren().add(new Label
                    (personBean.getAddress()));
  row.getChildren().add(new Label(Float.toString
                    (personBean.getWeight())));
  row.getChildren().add(new Label(sdf.format
                    (personBean.getBirthdate())));
  row.getChildren().add(new Label
                    (personBean.getColor()));
  Checkbox cb = new Checkbox();
  cb.setChecked(personBean.isMarried());
  cb.setDisabled(true);
  row.getChildren().add(cb);
  row.getChildren().add(new Label
                    (personBean.getRating()));
 }

 private void createPersonsGrid()
 {
  Grid personsGrid = Path.getComponent("/simpleWindow/
                    personsGrid");
  // add all the personsBeans
  Collection personBeans = PersonDAO.getInstance().
                    getAllPersons();
  for(Iterator it = personBeans.iterator(); it.hasNext();)
  {
   PersonBean personBean = (PersonBean)it.next();
   addRow(personBean,rows);
  }
 }
}
```

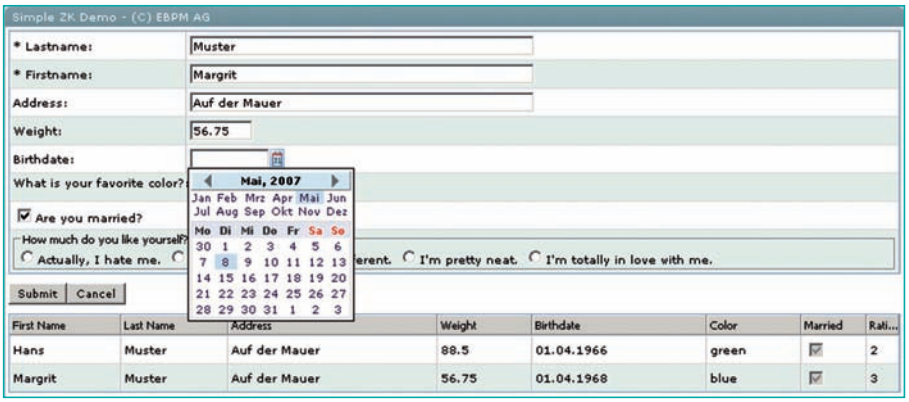


Abb. 4: ZK Resultat

einen JavaScript-Aufruf vom Client an den Server gesendet. Als Antwort auf diesen Request wird nicht mehr die ganze ZUL-Seite zurückgegeben (Listing 7) und erneut aufgebaut, sondern es wird in der *addRow()*-Methode lediglich der Teil der Darstellung aktualisiert, der durch den Aufruf betroffen ist. D.h. es wird nur eine zusätzliche Zeile in der Personentabelle eingefügt. Da nicht die ganze Seite neu aufgebaut werden muss, wird das Resultat entsprechend schnell wieder angezeigt - eine schöne interaktive Ajax-Anwendung eben. In der *addRow()*-Methode sieht man auch, wie eine grafische Komponente auf der Serverseite in Java aufgebaut wird.

Was in diesem Beispiel mit ZK auffällt, ist die aufgeräumte und übersichtliche ZUL-Seite. Die Seite enthält keinerlei Programmierlogik, wie zum Beispiel das Iterieren über eine Liste, sondern beschreibt lediglich die darzustellenden Komponenten. Dafür ist die Controller-Klasse *SimpleWindow* kein reiner Controller mehr, da sie auch GUI-spezifische Funktionalität enthält. Ob dies nun gut oder schlecht ist, kann natürlich diskutiert werden. ZK würde es auf jeden Fall auch erlauben, die Applikation anders zu strukturieren und das Aufbauen der Personentabelle auf der ZUL-Seite durchzuführen. Das sauberste Design für diese Applikation wäre gewesen, die Personentabelle als eigenständige ZK-Komponente zu implementieren: Einer der mächtigsten Eigenschaften von ZK ist nämlich die Möglichkeit, auf einfache Weise eigene Komponenten zu definieren, die dann über ein entsprechendes XML-Tag in die ZUL-Seiten eingebunden werden können. Was bei ZK auch überzeugt hat, sind die verfügbaren Standardkomponenten: Ob Kalender, Comboboxen oder sortierbare Tabellen, ZK lässt diesbezüglich fast keine Wünsche offen, wie das Resultat in Abbildung 4 zeigt.

Natürlich ist auch bei ZK nicht alles Gold, was glänzt. So entpuppte sich das Extrahieren der Formulardaten über die

Listing 7

simple.zul

```
<?page id="simplePage" title="Simple ZK Application"?>
<zk>
<window id="simpleWindow" use="com.ebpm.webdemo.
zk.SimpleWindow" title="Simple ZK Demo"
                    border="normal" width="800px"
                                height="500px">
  <caption label="(C) EBPM AG"/>
  <grid>
  <rows>
  <row>* Lastname: <textbox id="lastname"
                        width="300px" /></row>
  <row>* Firstname: <textbox id="firstname"
                        width="300px"/></row>
  <row> Address: <textbox id="address"
                        width="300px"/></row>
  <row> Weight: <textbox id="weight"
                          width="50px"/></row>
  <row> Birthdate: <datebox id="birthdate"
                    format="dd.MM.yyyy"/></row>
   <row spans="2">
     <hbox>
       What is your favorite color?:
       <combobox id="color">
         <comboitem label="Red" description="Red
                                    means Red ;-)"/>
         <comboitem label="Green"/>
         <comboitem label="Blue"/>
       </combobox>
     </hbox>
   </row>
   <row spans="2">
     <hbox>
       <checkbox id="married" label="Are you married?"/>
     </hbox>
   </row>
   <row spans="2">
     <groupbox>
       <caption label="How much do you like yourself?:"/>
       <radiogroup id="rating">
         <radio value="1" label="Actually, I hate me."/>
         <radio value="2" label="Not so much."/>
         <radio value="3" label="I'm indifferent."/>
         <radio value="4" label="I'm pretty neat."/>
         <radio value="5" label="I'm totally in love with
me."/>
       </radiogroup>
     </groupbox>
   </row>
  </rows>
  </grid>
   <separator/>
   <button label="Submit" onClick="simpleWindow.
                                    addPerson()"/>
   <button label="Cancel" onClick="Executions.
            sendRedirect(&amp;quot;/index.jsp&amp;quot;)"/>
   <separator/>
   <grid id="personsGrid">
     <columns sizable="true">
       <column label="First Name"/>
       <column label="Last Name"/>
       <column label="Address"/>
       <column label="Weight"/>
       <column label="Birthdate"/>
       <column label="Color"/>
       <column label="Married"/>
       <column label="Rating"/>
     </columns>
     <rows id="personsGridRows"/>
   </grid>
</window>
</zk>
```

id-Attribute der entsprechenden Komponenten als recht mühsam. Ein eleganterer Ansatz wäre gewesen, für jede Komponente einen Change-Listener zu registrieren oder mithilfe so genannter Annotations ein automatisches Binden der Formulardaten an das Personen-Bean zu realisieren.

Wenn man also bedenkt, dass der Autor sich von Grund auf mit dem ZK-Framework vertraut machen musste und dabei unwesentlich mehr Zeit in die Beispielapplikation mit ZK investiert hat, als mit Struts oder JSF, so ist das Resultat in Abbildung 4 doch ziemlich beeindruckend.

Fazit

Die parallele Evaluation dieser drei Frameworks hat mir einige überraschende Erkenntnisse gebracht. Eigentlich hatte ich bislang JSF, als Weiterentwicklung von Struts, als die Web-Zukunft schlechthin betrachtet. Dabei hatte ich den Ajax- und Web-2.0-Hype für die Entwicklung von geschäftskritischen, datenzentrierten Geschäftsapplikationen bislang eher als Spielerei belächelt. Doch das ganze Komponentenmodell von ZK und die Tatsache, dass der Entwickler eigentlich nie mit Ajax bzw. JavaScript direkt in Berührung kommt, sondern auf einem viel höheren Abstraktionsniveau programmiert, haben mich sehr positiv überrascht. Der reiche Satz an fertigen Komponenten und die intuitive und logische Architektur des ZK-Frameworks haben mir beim Entwickeln der Beispielapplikation viel Spaß bereitet.

JSF hat für mich nach wie vor ein gewisses Potenzial, doch war ich auch etwas erstaunt zu entdecken, wie ähnlich sich die Beispielapplikationen mit der JSF-Implementation MyFaces und mit Struts geworden sind. In gewissen Bereichen war Struts sogar übersichtlicher und einfacher. Ich bin überzeugt, dass ein entscheidender Faktor bezüglich der Zukunft von JSF die Komponentenbibliotheken sein werden. Gelingt es der JSF-Community, genügend ausgereifte und einfach zu benutzende Komponenten bereitzustellen, die auch Ajax-Funktionalitäten beinhalten, so sehe ich eine positive Zukunft für JSF.

Das gute alte Struts hat meiner Meinung nach seine Dienste getan. Es ist solide, stabil und ausgereift und erfüllt seinen Zweck. Die Entwicklung mit Struts ist aber auf lange Sicht, aufgrund des fehlenden Komponentenmodells und der zu beherrschenden Technologievielfalt zu teuer. Die Benutzer sind auch je länger desto weniger bereit, die Unzulänglichkeiten heutiger Webapplikationen, wie das ständige Nachladen der ganzen Seite nach jedem Mausklick oder die spartanischen GUI-Elemente, zu akzeptieren.

Für mich hat sich durch die Evaluation des ZK-Frameworks eine neue Welt aufgetan und mir ist plötzlich bewusst geworden: so geht es auch! Wenn man sich einmal von der Vorstellung verabschiedet hat, den erzeugten Quelltext einer Webseite verstehen zu wollen, so steht einem nichts mehr im Weg, um in die faszinierende Web-2.0-Welt einzutauchen. Natürlich muss das ZK-Framework sich noch in vielen Geschäftsapplikation beweisen, bis es den Reifegrad von Struts erlangt. Da ZK sehr viel Arbeit dem Server überlässt, wird vor allem auch die Frage nach der Performance einen zentralen Stellenwert bekommen.

Daniel Seiler, Dipl. El. Ing. ETH, ist Gründer der EBPM AG (www.ebpm.ch) mit Sitz in Zürich. Zuvor war er mehrere Jahre in einem Finanzdienstleistungsunternehmen tätig, wo er sich u.a. mit der Entwicklung von prozessorientierten Web-Applikationen beschäftigte Kontakt: daniel.seiler@ebpm.ch.

Links & Literatur

[1] www.zkoss.org
[2] myfaces.apache.org
[3] struts.apache.org
[4] www.icefaces.org
[5] www.jboss.com/products/seam
[6] incubator.apache.org/adffaces
[7] tobago.atanion.net/tobago-example-demo
[8] https://facelets.dev.java.net
[9] grails.codehaus.org
[10] www.openlaszlo.org
[11] https://ajax4jsf.dev.java.net
[12] getahead.org/dwr/
[13] code.google.com/webtoolkit
[14] dojotoolkit.org
[15] de.wikipedia.org/wiki/Struts
[16] de.wikipedia.org/wiki/JavaServer_Faces
[17] en.wikipedia.org/wiki/ZK_Framework