Love & Peace & JSF!

Es existieren viele Wege, JavaServer Faces mit Ajax zu verheiraten. Allerdings ist die Vielzahl an Möglichkeiten dafür durchaus verwirrend, zumal die zahlreichen Ansätze alle untereinander inkompatibel sind. Hier kommt das Projekt Woodstock ins Spiel. Als Voraussetzung benötigt man lediglich eine JSF-1.2-konforme Implementierung.

von Oliver Wolff

Die Woodstock-Komponenten [1] basieren auf dem Projekt JSF Extensions Dynamic Faces [2] (Kasten: *Dynamic Faces*). Die einzelnen Komponenten wurden ursprünglich für die Administrationsoberfläche des GlassFish-Servers [3] entwickelt. Auf dieser Basis hat Sun für NetBeans [4] das VisualWeb Plug-in geschaffen, das einerseits die mächtigen Woodstock-Komponenten enthält, andererseits aber auch die Möglichkeit bietet, Webanwendungen mithilfe von Drag-and-Drop zu entwickeln. Das visuelle Entwickeln bezieht sich nicht nur auf das Zusammenklicken der Komponenten, sondern auch auf das Verknüpfen mit anderen Programmteilen. Über Drag-and-Drop kann man eine Datenquelle, z.B. eine Datenbank, direkt mit der Tabelle verknüpfen. Alles Weitere wird durch ein paar Assistenten und etwas „Zauberei" im Hintergrund erledigt. Die Komponenten bieten eine hohe Funktionalität, aber auch eine erstaunliche Qualität. Dies reicht von der Tatsache, dass jede Komponente genau spezifiziert ist, bis hin zum erprobten produktiven Einsatz in vielen Projekten. Eine saubere Internationalisierung versteht sich hierbei von selbst. Im Zuge der Open-Source-Orientierung von Sun wurde Woodstock unter der CDDL-Lizenz [5] veröffentlicht. Die kommerzielle Herkunft und das anhaltende Interesse von Sun sorgen für eine hohe Qualität, die Open-Source-Lizenz vereinfacht den Einsatz in Projekten. Nebenbei wurde durch diesen Prozess die Abhängigkeit von NetBeans gelockert. Wenn man auf das Drag-and-Drop verzichtet, kann man Woodstock auch mit anderen Entwicklungsumgebungen wie Eclipse verwenden.

Dynamic Faces

Vor dem Erscheinen von JSF 1.2 gab es keinen einheitlichen Weg, um Ajax mit JSF zu verwenden. Die Komponentenhersteller kochten alle ihr eigenes Süppchen und die verschiedenen Bibliotheken konnten nur sehr schwer zusammen verwendet werden. Dynamic Faces bietet eine einheitliche Basis für die Integration, mit der man Komponenten unterschiedlicher Bibliotheken parallel verwenden kann. Technisch bieten sie eine JavaScript-Bibliothek für den asynchronen Zugriff und einen eigenen Lebenszyklus. Sie verwenden außerdem sehr stark die mit JSF 1.2 eingeführte Funktion *UIComponent.invokeOnComponent(...)* [6]. Die ursprüngliche Idee namens Avatar stammt von Jacob Hookom [7].

Artenvielfalt und Daten

Jenseits aller technischen Aspekte lebt eine Komponentenbibliothek von der Anzahl und Qualität der einzelnen Komponenten. Auch hier hat Woodstock viel zu bieten. Eine Übersicht findet sich bei Sun [8]. Die interessanteste Komponente stellt immer die Datentabelle dar. Viel Funktionalität, die man bei der Standard-JSF-Tabelle mühsam per Hand programmieren muss, bekommt man hier geschenkt: Paginierung, (mehrfache) Sortierung, Filter, Voreinstellungen und vieles mehr. Bei manchen genügt die entsprechende Deklaration. Bei anderen benötigt man einfache Java-Klassen zur Unterstützung, z.B. für die Filter. Diese kann man freundlicherweise komplett der Javadoc-Dokumentation zur Table-Komponente entnehmen (Abb. 1).

Eine Alternative hierzu ist die Ajax-Tabelle. Sie erledigt die Paginierung über einen Scrollbalken. Sobald neue Daten benötigt werden, werden sie asynchron vom Server geladen und angezeigt. Es gibt noch weitere Komponenten zum strukturierten Darstellen

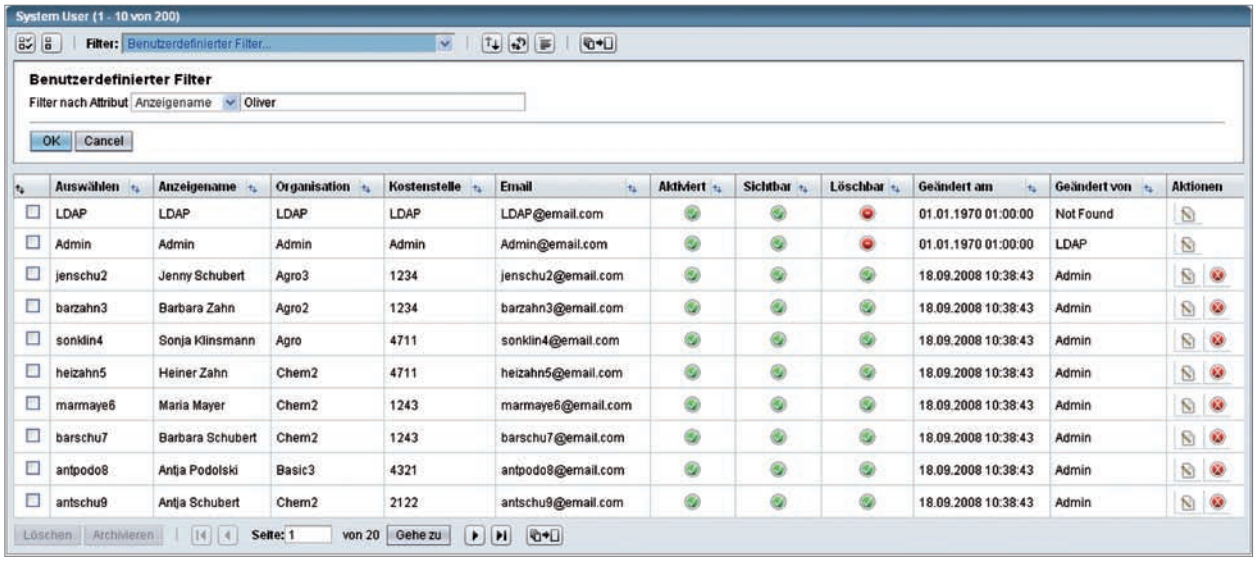

Abb. 1: Table-Komponente mit Filter-Panel

von Daten: AjaxTree, Ordable-/Editable List und einiges mehr.

Struktur und Layout

Auch für den Seitenaufbau gibt es viele praktische Komponenten: Accordion, Alert, Bubble, LogIn, Wizard, PopUp-Menu, Breadcrumbs sind nur einige Beispiele.

Eingabe und andere Elemente

Neben diversen Varianten von Standard-Eingabe-Elementen, z.B. eigenen Textfeldern, Buttons, Calendar & Co, gibt es aber auch speziellere wie ProgressBar, Rating, dndContainer (Drag-and-Drop), FileUpload und weitere. Mit diesen Elementen kann man desktopartige Webanwendungen entwickeln.

Unter die Haube geschaut

Interessant ist das Zusammenspiel zwischen Client und Server: Für jede Komponente, die im Browser angezeigt wird, existiert neben der Serverkomponente ein Client Widget, das auf der Dojo-Bibliothek basiert. Das eigentliche Rendering der Komponente wird also im Browser [9] erledigt. Diese Trennung geht sogar so weit, dass man nicht auf JSF für die Serverseite beschränkt ist, sondern auch andere Frameworks verwenden kann. Die Integration funktioniert über eine DynaFaces Bridge, die die Kommunikation zwischen Server und Client abwickelt (Abb. 2). Zusätzlich gibt es im Browser noch ein EventListener JavaScript, das im Zusammenspiel mit der DynaFaces Bridge Daten und Events zwischen JSF und dem Dojo/JavaScript Widget überträgt.

Der Widget-Ansatz hat einige Konsequenzen. So besitzt jede Komponente eine umfangreiche JavaScript-API. Dies ist in vielen Fällen sehr praktisch, so kann man z.B. für die meisten Komponenten mit einem *domNode.refresh()*-Aufruf eine asynchrone Aktualisierung auslösen. Im Vergleich zu anderen Ajax-Bibliotheken wie RichFaces [10] muss man bei Woodstock deutlich mehr mit JavaScript arbeiten. Da für viele JSF-Entwickler JavaScript eher unbekannt bis böse ist, bedeutet dies eine große Umstellung. Man wird jedoch mit einer höheren Funktionalität und Flexibilität belohnt.

Ein Blick in die Praxis

Nachdem nun einige Geheimnisse um Woodstock gelüftet wurden, ist es an der Zeit, sich auf die Praxis zu stürzen und zwar am Beispiel einer der wichtigsten Ajax-Anwendungen: *AutoComplete* (Abb. 3). Dies ist nicht nur ein gutes Beispiel, um Woodstock zu demonstrieren, sondern auch eine schöne Möglichkeit, Anwender glücklich zu machen. Wie funktioniert das Ganze? Nach jedem Tastendruck muss der Inhalt des betreffenden Textfeldes an den Server gesendet werden. Dieser verarbeitet den Teil-String und sendet Daten zurück, die als Auswahlliste unterhalb des Eingabefeldes eingeblendet werden. Mit den Pfeiltasten und der Eingabetaste oder der Maus kann der Benutzer dann einen Eintrag auswählen. Jeder, der so etwas schon „zu Fuß" implementiert hat, weiß, dass dies alles andere als einfach ist. Mit Woodstock ein Kinderspiel. JSP-Seite:

```
xmlns:w="http://www.sun.com/webui/webuijsf"
.....
<w:textField id="organisationInput"
autoComplete="true" autoCompleteExpression=
        "#{autoComplete.getOrganisationChoices}"
text="#{modifyUserBean.updateUser.organization}"
/>
```

Um das *AutoComplete* einzuschalten, braucht man nur die Attribute *autoComplete="true"* und *autoCompleteExpression="methodName"*. Die *autoCompleteExpression* sorgt über ein *MethodBinding* für den Aufruf der entsprechenden Methode. Diese muss die Signatur *public com.sun.webui.jsf.model.Option[] methodName(String searchString)* haben. Im konkreten Beispiel sieht es wie folgt aus:

```
public Option[] getOrganisationChoices(String
                                    searchString) {
 // Verfügbare Organisationen zur Auswahl
 List<String> organizationList = getAvailable
                                    Organizations();
 // Ergebnis Optionen
 ArrayList<Option> resultList = new
ArrayList<Option>();
 // Einige Filter-Operationen
 .....
 return resultList.toArray(new Option
                                    [resultList.size()]);
```

Abb. 2: Vereinfachte Darstellung der Client-Server-Kommunikation

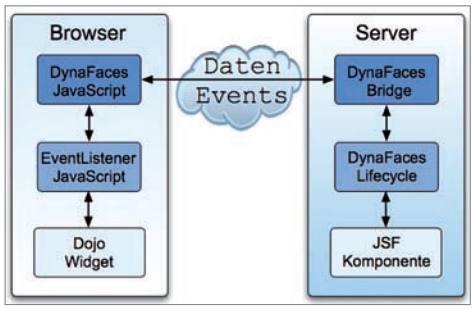

Abb. 3: AutoComplete bei der Eingabe des Feldes Organisation

Mit Hilfsmethoden kann man das Ganze noch deutlich vereinfachen. So stellt man sich das Arbeiten mit JSF vor. Eine Komponente, zwei einfache Attribute und eine Methode für die eigentliche Arbeit. Der Rest wird vom Framework erledigt.

Zauberlehrling

Wizards dienen dem strukturierten Erfassen von Daten. Gerade bei Produktivanwendungen sind sie wichtige Elemente, um die Effizienz und Qualität zu steigern. Dies führt dazu, dass in vielen Projekten handgestrickte Lösungen verwendet werden. Die Woodstock-Wizard-Komponente vereinfacht dies. Die Einbindung ist relativ einfach. Die JSP-Seite:

```
<w:wizard id="createUserWizard" title="Neuen Nutzer
hinzufügen">
```

Dies ist das oberste Element. Es kann unterschiedliche Elemente aufnehmen:

- Normale *<w:wizardStep>*-Elemente für einfache Schritte
- *<w:wizardBranch>*-Schritte für Verzweigungen
- Für komplexere Wizards gibt es auch noch die Möglichkeit von *<w:wizardSubstepBranch>*-Elementen

Das Wizard-Element wird an einen *WizardEventListener* angebunden. Die entsprechende *handleEvent*-Methode wird bei jedem Übergang aufgerufen. Damit hat man eine einfache programmatische Kontrolle über den Ablauf des Wizards. Mit diesen Elementen kann man effizient sehr funktionale Assistenten erstellen.

Datenschleuder

Wie schon erwähnt, vereinfacht Woodstock den Umgang mit Datentabellen enorm. Erweitertete Funktionalität wie die Sortierung der Daten kann man jedoch schwer mit der Table-Komponente abbilden. Auch die Drag-and-Drop-„Zauberei" aus NetBeans ist damit kaum zu implementieren. Das Interface *DataModel* aus dem JSF-Standard ist für den komfortablen Umgang mit Tabellendaten nicht ausreichend. Die Lösung hierfür ist die DataProvider-API. Sie stellt einen vereinfachten Zugriff für Data Bindings dar. DataProvider sind eine Abstraktion für DataSources. Klassische DataSources beziehen sich auf Datenbanken, DataProvider verallgemeinern diesen Ansatz [11]. Eine spezielle Form sind die *TableDataProvider*. Das Interface beschreibt Funktionen, die die Woodstock-Tabelle befähigen, fortgeschrittene Techniken wie Sortierung rein deklarativ einzubinden. Ein Beispiel: Die Klasse *com.sun.data.provider.impl.ObjectListDataProvider* ist ein *TableDataProvider* mit einer Liste als interne Datenstruktur. Die Table-Komponente wird über das Kindelement *<w:tableRowGroup>* mit dem *TableDataProvider* verknüpft:

```
<w:tableRowGroup sourceData="#{listSystemUser
 Bean.group.provider}" .... sourceVar="systemUser">
```

Nun kann man die einzelnen Spalten einfach erzeugen:

```
<w:tableColumn id="shortname"
            headerText="Auswählen" sort="shortname"
rowHeader="true">
  <w:staticText text="#{systemUser.value.
                                    shortname}" />
</w:tableColumn>
```

Die Tabelle iteriert über die vom *TableDataProvider* bereitgestellten Objekte mit dem Element *systemUser*, das eine Zeile repräsentiert. Da die einzelnen Objekte JavaBeans sind, kann man mit der kurzen Schreibweise "*#{systemUser.value.shortname}*" auf die Attribute zugreifen. Die Sortierung erfolgt durch Angabe des Bean Property, nach dem sortiert werden soll: *sort="shortname"*. Man muss nichts weiter tun. Wenn mehrere Spalten sortierbar sind, erhält man automatisch eine Mehrfachsortierung, inklusive aufklappbarem Panel. Wenn man den Provider gegen eine andere Implementierung, z.B. den *CachedRowSetDataProvider*, austauscht, kann man sehr einfach CRUD-Anwendungen schreiben. Bei komplexeren Szenarien würde man sich die entsprechenden Zugriffsklassen *MyCRUDServiceTableProvider* erzeugen lassen. Durch die einfache Struktur der Table-Komponente kann man sich auch mit wenig Aufwand einen Codegenerator bauen, der komfortabel JSF-Seiten mit hoch funktionalen Tabellen erzeugt.

Facelets

Auch wenn Facelets [12] als ViewHandler keinen JSF-Standard darstellen, erfreuen sie sich weiter Verbreitung. Deshalb haben inzwischen die meisten Komponentenbibliotheken, auch Woodstock, eine Unterstützung für die Technologie eingebaut. Hierbei gibt es jedoch leider ein großes Aber: Woodstock verwendet sehr intensiv die mit JSF 1.2 eingeführten EL-Expressions für *Me-*

JSFTemplating

Obwohl Facelets der wohl am häufigsten verwendete alternative ViewHandler ist, ist er nicht der Einzige. Das Projekt JSFTemplating stellt eine ernsthafte Alternative dar. Neben der Unterstützung aller JSF-1.2-Features bietet es nicht nur eine Sprache zur Beschreibung der Views, sondern gleich eine ganze API. Aktuell gibt es zwei Implementierungen. Neben der proprietären versteht es auch die Facelets-Syntax, zumindest die wichtigsten Tags. Damit ist auch eine Migration relativ einfach. Zusätzlich kann man auch Template-basierte Renderer schreiben, was deutlich komfortabler ist, als der Umgang mit dem ResponeWriter.

thodBinding, z.B. *autoCompleteExpression*. Facelets sind dagegen noch nicht richtig bei JSF 1.2 angekommen, unterstützen diese Version bestenfalls rudimentär. Es gibt jedoch einen eleganten Ausweg, und zwar das Projekt JSFTemplating [13] (Kasten: *JSFTemplating*). Das Projekt wird aktiv weiterentwickelt und unterstützt auch die Facelets-Syntax. Damit hat man die Möglichkeit, eine bestehende Facelets-Anwendung sanft zu migrieren.

Love & Peace & JSF

Mit Woodstock haben Sun und die dahinter stehende Community hervorragende Arbeit geleistet. Wo viel Licht ist, ist aber auch Schatten. Die starke Verwendung von JavaScript stellt für manchen Entwickler durchaus eine Umstellung dar. Man wird jedoch mit einer höheren Funktionalität und Flexibilität belohnt. Die zögerliche Unterstützung von JSF 1.2 durch Facelets macht die Kombination nicht gerade einfach, allerdings kann man die Schwierigkeiten mit JSFTemplating elegant umschiffen. Aber es überwiegen die Vorteile. Die Menge der Komponenten und vor allem deren Qualität sind beeindruckend. Es gibt nicht viele Open-Source-Bibliotheken, bei denen die einzelnen Komponenten so sauber ausspezifiziert sind, inklusive Testspezifikation. Diese Qualität zieht sich durch alle Bereiche, vor allem auch die Dokumentation. Die einzelnen Komponenten sind durch eine umfangreiche JavaScript-API sehr flexibel. Es gibt viele elegante Lösungen wie das *AutoComplete* oder die Wizards. Das eigentliche Schmankerl stellt jedoch die DataProvider-API dar. Was nützen die hübschesten Ajax-Features, wenn die Integration der Daten mühsam ist? Mit passenden DataProvidern und einer klugen Architektur lassen sich leicht automatisiert CRUD-Anwendungen erzeugen, die über eine hohe Bandbreite an Funktionalität verfügen. ■

Oliver Wolff ist Entwickler, Berater und Trainer bei der Orientation in Objects GmbH. Die technischen Schwerpunkte seiner Arbeit bilden JavaServer Faces, Portale, Ajax, XML-Verarbeitung mit Java und Softwaretechnik. Seine Spezialgebiete sind JSF Framework-Aspekte, besonders im Hinblick auf die Ajax-Integration und die Architektur von J2EE-Anwendungen, mit dem Fokus auf Frontends.

Links & Literatur

[1] Woodstock: woodstock.dev.java.net/index.html
[2] JSF-Extensions: jsf-extensions.dev.java.net
[3] GlassFish: https://glassfish.dev.java.net
[4] NetBeans: www.netbeans.org
[5] CDDL-Lizenz: www.sun.com/cddl
[6] JavaDoc UIComponent.invokeOnComponent(...): java.sun.com/javaee/javaserverfaces/1.2/docs/api/javax/faces/component/UIComponent.html
[7] Jacob Hookom Avatar-Blog: weblogs.java.net/blog/jhook/archive/2005/09/jsf_avatar_vs_m_1.html
[7] Woodstock-Showcase: webdev2.sun.com/example/faces/index.jsp
[8] Woodstock Java-Architektur: https://woodstock.dev.java.net/docs/specs/ajax-architecture-spec.html
[9] Browser Support Matrix: https://woodstock.dev.java.net/docs/SupportedBrowers.htm
[10] RichFaces: www.jboss.org/jbossrichfaces
[11] Kurze Einführung DataProvider: blogs.sun.com/winston/entry/creator_data_provider_a_gentle
[12] Facelets: facelets.dev.java.net
[13] JSFTemplating: jsftemplating.dev.java.net