JBoss-Features und -Tools, Teil 4: JBoss und JBuilder X

von Marcus Redeker

In zwanzig Klicks zur EJB

Mit dieser Ausgabe des *Java Magazins* geht die Artikelserie über den JBoss Application Server in die vierte Runde. Im vorigen Teil haben wir uns die JBoss-Unterstützung innerhalb der freien Entwicklungsumgebung Eclipse angeschaut (*Java Magazin* 3.2004). Dies ist Grund genug, auch einmal zu betrachten, was ein kommerzieller Mitbewerber in diesem Bereich zu bieten hat.

Zunächst war eigentlich geplant, über das „Voyager JBoss OpenTool" [1] zu schreiben, da es zu dem Zeitpunkt, als die JBoss-Artikelserie besprochen wurde, noch keine offizielle Unterstützung in JBuilder für JBoss gab. Mittlerweile ist allerdings JBuilder X [2] auf dem Markt und Borland unterstützt jetzt auch JBoss in der gleichen Art und Weise wie die anderen kommerziellen Application Server. Das bedeutet, dass es entsprechende Wizards für die unterschiedlichen J2EE-Applikationstypen gibt und JBuilder X die JBoss-spezifischen Deployment-Deskriptoren erzeugt. Des Weiteren kann JBoss innerhalb von JBuilder X gestartet und debuggt werden und Anwendungen können direkt deployt werden. Zudem werden die JBoss-spezifischen Service Archives zum Deployen von MBeans unterstützt und es gibt einen eigenen Wizard für diese Archive.

Für die Entwicklung von EJBs stellt Borland den EJB-Designer zur Verfügung. Dies ist ein 2-Wege-Tool, welches erlaubt, EJB 2.0- und EJB 1.1-Komponenten grafisch zu modellieren, und JBuilder generiert den benötigten Sourcecode. Dabei lassen sich Änderungen an dem Design entweder über den Designer oder direkt in dem generierten Sourcecode machen. JBuilder synchronisiert dann das Design, je nachdem, wo geändert wurde. Parallel zur Source-Generierung werden auch die Deployment-Deskriptoren immer an das Design angepasst. Der EJB-Designer ist das Hauptziel dieses Artikels und ihn werden wir uns daher etwas genauer anschauen. Auf alle Möglichkeiten von JBuilder im Bereich Enterprise-Java einzugehen, würde den Rahmen dieses Artikels überschreiten.

Konfiguration

Nachdem JBuilder X und JBoss installiert sind, muss im JBuilder X über den Menüpunkt TOOLS | CONFIGURE SERVER... der JBoss als verfügbarer Server konfiguriert werden. Dazu reicht es aus, das Installationsverzeichnis von JBoss anzugeben. Es können in dem entsprechenden Dialog (Abb. 1) noch weitere Parameter eingegeben werden, die allerdings für eine einfache Standardinstallation nicht benötigt werden. Wer sich für seinen JBoss eine eigene Konfiguration angelegt hat, kann diese auf dem *Custom*-Tab auswählen. Dort lässt sich auch das Deployment-Verzeichnis ändern, wenn nicht das Standardverzeichnis benutzt werden soll.

Jetzt muss noch in dem zu benutzenden Projekt JBoss als Server eingestellt werden. Dies geschieht über die Projekteigenschaften und dort über den Eintrag *Server* in der linken Navigation. Hier kann auch noch festgelegt werden, welche Dienste der ausgewählte Server für das ausgewählte Projekt zur Verfügung stellen soll. Man hat dabei die Möglichkeit, mehrere Server für ein Projekt zu konfigurieren. So könnte man z.B. einen JBoss für die EJB-Services und einen Borland Application Server für die JSP/Servlet-Services benutzen.

Eine EJB aus dem Designer

Um mit JBuilder eine EJB zu erzeugen, muss zunächst ein neues EJB-Modul angelegt werden. Dafür werden über den Menüpunkt FILE | NEW... aus der Objektgalerie der Eintrag EJB MODULE ausgewählt

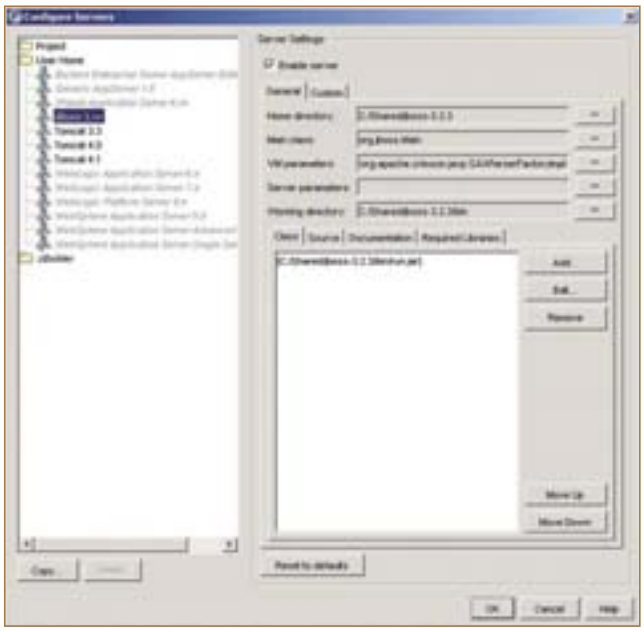

Abb. 1: Server konfigurieren

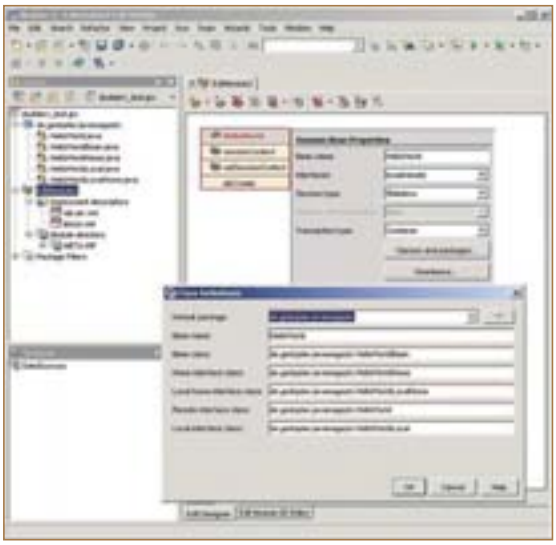

Abb. 3: DD-Editor (allgemeine EJB-Einstellungen)

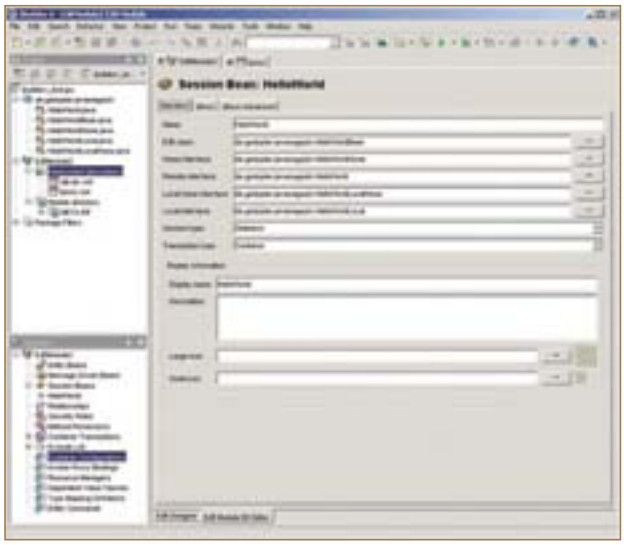

Abb. 2: EJB-Eigenschaften bearbeiten

und bei dem darauf folgenden Dialog die entsprechenden Einstellungen vorgenommen. Hierbei kann man sich entscheiden, ob ein bereits bestehender Ordner benutzt oder ein neues Verzeichnis angelegt werden soll. Außerdem wird festgelegt, welchem Standard die EJBs in dem Modul folgen sollen. Zur Auswahl stehen die EJB-Versionen 1.1 und 2.0. Ein EJB-Modul ist die JBuilder-interne Form eines EJB-JAR und kann eine oder mehrere EJBs beinhalten.

Nachdem das Modul angelegt ist, befindet man sich in dem EJB-Designer, über den man nun EJBs erzeugen und konfigurieren kann. Dazu wählt man aus dem über einen Rechtsklick geöffneten Kontextmenü CREATE EJB und dann den EJB-Typ, der erzeugt werden soll. Die EJB wird dann im Designer angezeigt und gleichzeitig der Sourcecode für die Default-Einstellungen generiert. Anschließend wird automatisch ein Panel angezeigt, in das man die Bean-Eigenschaften einstellen kann. Dort wird der Name eingegeben, welche Interfaces (local, remote oder beide) es geben soll, dann der Transaktionstyp und über den Button CLASSES AND PACKAGES… wird festgelegt, in welchem Package die generierten Interfaces und die Bean-Klasse liegen sollen (Abb. 2). Alle hier gemachten Änderungen werden automatisch im Sourcecode nachgezogen.

Nun können die Methoden der EJB im Designer definiert werden. Dazu genügt ein Rechtsklick auf die grafische Darstellung der EJB und in dem Kontextmenü wählt man dann ADD | METHOD. In dem darauf folgenden Panel werden der Name der Methode, ihre Parameter und der Returnwert festgelegt. Außerdem wird über eine Dropdown-Box festgelegt, in welchen Interfaces (local, remote oder beide) die Methode verfügbar sein soll. JBuilder generiert daraufhin automatisch die Signatur der Methode in den gewählten Interfaces und erzeugt eine leere Methode in dem Bean-Sourcecode. Man muss jetzt lediglich in dem Sourcecode seine Logik implementieren und fertig ist die EJB.

Jetzt fehlen nur die Deployment-Deskriptoren für unsere Bean, damit wir sie anschließend im JBoss deployen können. Aber auch diese sind mithilfe des Deployment-Deskriptor-Editors von JBuilder schnell erzeugt. Dazu klickt man wiederum mit der rechten Maustaste auf die EJB im Designer und wählt OPEN DD EDITOR. Es wird dann der Deployment-Deskriptor-Editor für die ausgewählte EJB geöffnet und alle Einstellungen lassen sich komfortabel

Listing 1

ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC „-//Sun Microsystems,
                Inc.//DTD Enterprise JavaBeans 2.0//EN"
            "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
 <display-name>EJBModule2</display-name>
 <enterprise-beans>
  <session>
   <display-name>HelloWorld</display-name>
   <ejb-name>HelloWorld</ejb-name>

<home>de.gedoplan.javamagazin.HelloWorldHome</home>
   <remote>de.gedoplan.javamagazin.HelloWorld</remote>
   <local-home>de.gedoplan.javamagazin.HelloWorld
                                LocalHome</local-home>
   <local>de.gedoplan.javamagazin.HelloWorldLocal</local>
   <ejb-class>de.gedoplan.javamagazin.
                              HelloWorldBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
 </enterprise-beans>
 <assembly-descriptor>
  <container-transaction>
   <method>
    <ejb-name>HelloWorld</ejb-name>
    <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
  </container-transaction>
 </assembly-descriptor>
</ejb-jar>
```

Abb. 4: DD-Editor (Standard-JBoss-Einstellungen)

über entsprechende Eingabefelder vornehmen. Die Abbildungen 3 und 4 z.B. zeigen unterschiedliche Bereiche des Deployment-Deskriptor-Editors für eine Session-EJB. In den Listings 1 und 2 ist abgedruckt, was JBuilder aus den gemachten Einstellungen generiert.

Builden und Deployen

Nachdem die erste EJB mit einigen Klicks und dem Eingeben von den benötigten Werten für die Deployment-Deskriptoren erstellt worden ist, fehlt noch das JAR-File, welches in JBoss deployt werden kann. Wenn man keine Tool-Unterstützung hätte, müsste man nun die kompilierten Klassen und die Deployment-Deskriptoren der EJB in ein Verzeichnis kopieren, in die richtige Struktur bringen und das Verzeichnis dann mit dem JAR-Kommando zusammenpacken. JBuilder nimmt einem diese Arbeit ab und es genügt ein Klick auf den Menüpunkt REBUILD im Kontextmenü, welches man mit einem Rechtsklick auf das EJB-Modul im Projektbaum erhält. Über dasselbe Kontextmenü lässt sich das von JBuilder gepackte JAR-File auch in den vorher konfigurierten JBoss deployen. Dazu wählt man den Menüpunkt DEPLOY OPTIONS… | DEPLOY und JBuilder kopiert das JAR-File automatisch an die richtige Stelle.

Starten von JBoss

Um JBoss direkt im JBuilder zu starten, wird eine Runtime-Konfiguration benötigt. Diese wird über das Menü RUN | CONFIGURATIONS… angelegt. Dort wählt man den Button NEW und im anschließenden Dialog werden die Einstellungen für die entsprechende Runtime-Konfiguration vorgenommen (Abb. 5). Nun lässt sich JBoss ganz einfach im normalen Modus oder auch im Debug-Modus über das RUN-Menü starten. Wenn JBoss im Debug-Modus gestartet ist, lassen sich EJBs mit Breakpoints versehen und JBuilder stoppt die Ausführung der EJB an der definierten Stelle und der EJB-Code lässt sich Zeile für Zeile abarbeiten.

Listing 2

jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.2//EN"
        "http://www.jboss.org/j2ee/dtd/jboss_3_2.dtd">
<jboss>
 <enforce-ejb-restrictions>true</enforce-ejb-restrictions>
 <security-domain>test</security-domain>
 <enterprise-beans>
  <session>
   <ejb-name>HelloWorld</ejb-name>
   <jndi-name>ejb/HelloWorld</jndi-name>
   <local-jndi-name>ejb/HelloWorldLocal</local-jndi-name>
   <clustered>True</clustered>
   <cluster-config>
    <partition-name>Gedo_Cluster1</partition-name>
   </cluster-config>
  </session>
 </enterprise-beans>
</jboss>
```

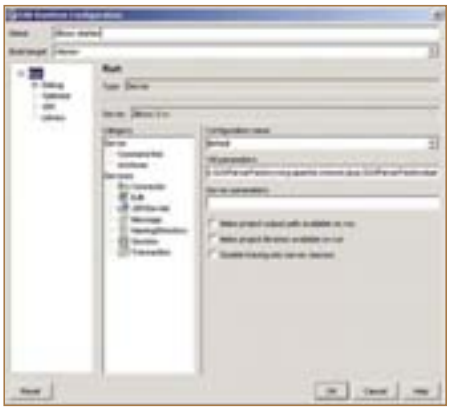

Abb. 5: Definieren der JBoss-Runtime-Konfiguration

Fazit

Worauf viele Entwickler bereits lange gewartet haben, ist nun also geschehen: JBuilder unterstützt von Hause aus den Open Source Application Server JBoss und es wird kein Open-Tool mehr benötigt. Wer bereits mit JBuilder und einem anderen Application Server gearbeitet hat, wird sich schnell an die JBoss-spezifischen Dialoge gewöhnen. Existierende J2EE-Anwendungen lassen sich so, ohne größeren Aufwand, auch für JBoss erzeugen. Es muss lediglich der neue Server in den Projekteigenschaften eingestellt und der Build-Vorgang muss neu vorgenommen werden.

Es besteht allerdings weiterhin das Problem, dass von Hand editierte Deployment-Deskriptoren von JBuilder überschrieben werden. Man ist also darauf angewiesen, nur die Tags zu benutzen, die auch von JBuilder unterstützt werden. Gerade bei JBoss, an dem sehr viele Entwickler aktiv mitentwickeln, kann es allerdings passieren, dass es neue Tags in den Deskriptoren gibt, bevor sie auch von JBuilder angeboten werden. Hier wäre eine Merge-Funktionalität, wie sie z.B. XDoclet [3] anbietet, sehr hilfreich. Im Großen und Ganzen muss man sagen, dass die Enterprise-Unterstützung von JBuilder wirklich gelungen ist und dem J2EE Entwickler viel Arbeit abgenommen wird. Im nächsten Artikel unserer JBoss-Reihe gehen wir auf die Geheimnisse von CMP 2.0 ein und was JBoss über die Spezifikation hinaus bietet, um eine optimale Performance zu erzielen. ■

Marcus Redeker ist Senior Java Consultant bei der GEDOPLAN GmbH in Bielefeld. Er ist zertifizierter JBoss Consultant und seit mehr als fünf Jahren als Berater und Trainer im Bereich Java und Enterprise Java tätig.

Links & Literatur

[1] JBoss OpenTool für JBuilder: www.sourceforge.net/projects/jboss-opentool/

[2] Borland JBuilder X: www.borland.de/jbuilder/

[3] XDoclet: xdoclet.sourceforge.net/