## JBoss-Features und Tools, Teil 1: Der JBoss Application Server

von Dirk Weil und Marcus Redeker

# Just JBoss

Der JBoss ist der freiverfügbare Application Server. Grund genug, ihm eine Serie zu widmen, in der künftig verschiedene JBoss-Features und Tools vorgestellt werden. Wir starten mit einem Überblick über die Features und wie man JBoss konfigurieren und administrieren kann.

### JBoss und die JBoss Group

JBoss [1] ist ein J2EE-konformer Application Server, der frei verfügbar ist und sich in letzter Zeit immer mehr an Beliebtheit erfreut. J2EE-konform bedeutet, dass er alle in der J2EE 1.3 erforderlichen Spezifikationen erfüllt, aber nicht von Sun zertifiziert ist. Dies soll sich allerdings in Kürze ändern, da die JBoss Group [2] momentan in Verhandlungen mit Sun steht. Die JBoss Group ist eine kommerzielle Firma, die Dienstleistungen rund um den Application Server anbietet. Sie wurde von Marc Fleury gegründet, der 1999 das JBoss-Projekt ins Leben gerufen hat. Durch die erwartete Zertifizierung erhofft sich die JBoss Group entsprechend mehr Umsatz, da sich momentan noch viele Firmen auf Grund der nicht vorhandenen Zertifizierung gegen JBoss entscheiden. Auch das Argument, dass es für freie Software keinen Support gibt, versucht die JBoss Group mit entsprechenden Supportverträgen, inklusive 24x7-Support, aus dem Weg zu räumen.

### JBoss Historie

Mittlerweile befindet sich JBoss in der dritten Generation inklusive diverser Maintenance-Releases und Version vier befindet sich in der Entwicklung. Die Historie der unterschiedlichen JBoss Releases ist in Tabelle 1 aufgeführt. Wir werden uns in diesem Artikel bereits auf die kommende Version 4.0 beziehen, wobei sich die Installation und Konfiguration nicht von der aktuellen Version 3.2 unterscheidet.

### Features

Wie schon erwähnt ist JBoss J2EE 1.3-konform. Das heißt, er unterstützt die dort geforderten APIs, wie z.B. einen Container für Enterprise JavaBeans, einen eigenen Transaktionsmanager, einen Container für Web-Anwendungen (Servlets

| Version | Datum | Typ | Kommentar |
|---|---|---|---|
| 2.2 | 10.04.2001 | Major Release | – |
| 2.2.1 | 18.04.2001 | Bugfix Release | – |
| 2.2.2 | 30.05.2001 | Bugfix Release | Aktuelles 2.2 Release |
| 2.4 | 21.08.2001 | Major Release | – |
| 2.4.1 | 10.09.2001 | Bugfix Release | – |
| … | … | … | … |
| 2.4.11 | 16.04.2003 | Bugfix Release | Aktuelles 2.4 Release |
| 3.0 | 30.05.2002 | Major Release | J2EE 1.3-Unterstützung, komplettes Redesign, Clustering |
| 3.0.1 | 06.08.2002 | Bugfix Release | – |
| … | … | … | … |
| 3.0.8 | 06.06.2003 | Bugfix Release | Aktuelles 3.0 Release |
| 3.2 | 11.04.2003 | Major Release | kleine Änderungen, Erweiterung des Clustering, vereinfachte Konfiguration |
| 3.2.1 | 02.06.2003 | Bugfix Release | – |
| 3.2.2 | 18.10.2003 | Bugfix Release | Aktuelles 3.2 Release |
| 4.0.0 | Nicht bekannt | Major Release | Voraussichtlich Ende des Jahres mit J2EE 1.4-Unterstützung |
| 4.0.0 DR2 | 03.07.2003 | Preview | Developer Release |

Tab. 1: Überblick der JBoss-Versionen

und JSPs) und die Möglichkeit, JCA-Connectoren zu deployen, um nur die Wichtigsten zu nennen. Bei dem Web-Container kann sich der Benutzer entscheiden, ob er JBoss mit der Servlet-Engine Tomcat oder mit Jetty benutzen möchte. Seit Version 3.2.2 wird Tomcat bei dem JBoss-Download als Standard mitgeliefert. Bis Version 3.2.1 wurde Jetty standardmäßig integriert.

Seit der Version 3.0 bietet JBoss allerdings noch einige andere interessante Features, die nicht in den Spezifikationen aufgeführt sind. So bietet die CMP-Engine die Möglichkeit, Finder auf Attribute automatisch anzulegen und auch eine Erweiterung der EJB-QL in Form von JBossQL ist vorhanden. Ein weiteres Feature, was von vielen im Standard vermisst wird, ist die Möglichkeit, CMP-Finder dynamisch generieren zu lassen. Dafür bietet JBoss seine DynamicQL an, die zur Laufzeit JBossQL erzeugt und ausführt. Auch die oft bemängelte schlechte Performance von CMP Entity Beans bekommt man mit JBoss in den Griff. Dafür werden diverse Optimierungseinstellungen angeboten, die beim Zugriff auf die Datenbank die abgesetzten SQLs minimieren und optimieren. Die CMP 2.0-Engine von JBoss werden wir uns in einer der kommenden Ausgaben etwas genauer ansehen.

Bleibt noch zu erwähnen, dass JBoss seit Version 3.0 auch Clustering unterstützt. Dieses Feature, das bei vielen kommerziellen Produkten herausgestellt wird, ist immer weiter entwickelt worden und in der aktuellen Version 3.2.2 stabil und ausgereift. Es wird bereits in mehreren unserer Kundenprojekte erfolgreich eingesetzt. Die folgenden Eigenschaften sind bei einem JBoss-Cluster vorhanden:

**Der JBoss 4.0 soll J2EE 1.4 unterstützen**

- Alle Instanzen in einem Cluster finden sich automatisch.
- Load Balancing und Fehlertoleranz für die Dienste JNDI, RMI, Entity EJBs, Stateless Session EJBs und Stateful Session EJBs mit in-memory Status-Replizierung.
- HTTP Session Replizierung.
- Verteilter JNDI-Baum im Cluster.
- Verteiltes Cluster-Deployment.

Die kommende Version 4.0 soll zudem die noch nicht verabschiedete J2EE 1.4-Spezifikation unterstützen. So kann man z.B. im Developer Release schon den Timer-Service von EJBs ausprobieren oder auch Web Services deployen. Allerdings ist wohl das von den JBoss-Entwicklern propagierte AOP-Framework (siehe Kasten „AOP“) die interessanteste Neuerung, die in Version 1.4 enthalten sein wird. Sie erlaubt dem Entwickler, einfache Java-Objekte um Dienste wie Transaktionsmanagement, Persistenz, Logging und andere angebote Aspekte zu erweitern, ohne seine Klasse zu modifizieren. Dabei kann man in der mitgelieferten Konsole sehen, welche Klassen vom AOP-Framework geladen und um welche Aspekte sie erweitert wurden.

### Wo bekommt man JBoss

Links zu den Final Releases findet man immer auf der JBoss-eigenen Website [1]. Alle anderen Downloads rund um JBoss (Minor Releases, freie Doku, …) findet man bei SourceForge [6], der Open Source-Community, die das JBoss-Projekt verwaltet. Alle diejenigen, die sich an der Entwicklung beteiligen wollen, finden dort auch eine Liste von Bugs, Mailinglisten und Informationen, wie man das CVS Repository von JBoss auschecken kann, um JBoss selber zu builden.

### Unkomplizierte Installation

Zur Installation von JBoss wird das Download-File einfach an der gewünsch-

#### AOP – Aspect Oriented Programming

Bei der aspektorientierten Programmierung geht es darum, bestehende Programm-Module (Java-Klassen) um Aspekte zu erweitern. Diese Aspekte können zum Beispiel wiederkehrende technische Anforderungen (Logging-Mechanismus) kapseln. Der Trick besteht darin, dass der Programmierer beim Erzeugen seiner Java-Klasse keine zusätzlichen Methoden definieren muss. Die Module, die durch Aspekte zur Verfügung gestellt werden, werden nachträglich in die Klasse eingefügt. Dies kann entweder über einen speziellen Compiler geschehen (z.B. AspectJ [5]) oder – wie bei JBoss – durch Modifikation der Klasse beim Laden. AOP bietet dem Entwickler somit ganz neue Möglichkeiten im Bezug auf Modularisierung und Erweiterung von Programmen.

#### JMX – Java Management eXtension

JMX ist eine Spezifikation, die eine Architektur und ein API für das Management von Anwendungen beschreibt. Wie schon bei anderen Java-Spezifikationen werden nur die Schnittstellen beschrieben, die Implementierung ist jedem selbst überlassen, wobei Sun eine Referenzimplementierung und eine CTS (Compatibility Test Suite) anbietet. Im Falle von JBoss heißt die JMX-Implementierung JBossMX.
Die JMX-Architektur ist in vier wesentliche Bestandteile untergliedert:

- Instrumentation Level: Definiert, wie JMX-gemanagte Ressourcen (z.B. Dienste, Anwendungen, Softwaremodule) implementiert werden können. Die Implementierung erfolgt in Java oder sie hat einen Java-Wrapper. Über Managed Beans (MBeans) kann der Zustand einer Ressource abgefragt oder Operationen auf ihr durchgeführt werden.
- Agent Level: Definiert, wie JMX-Agenten implementiert werden sollen. Zentrale Komponente eines Agenten ist der MBean-Server, bei dem alle MBeans registriert werden können. Der JMX-Agent stellt eine Brücke zwischen den MBeans und den Management-Applikationen dar.
- Distributed Services Level: Ist in der aktuellen Version (1.2) nicht spezifiziert. Soll Connectoren und Protokoll-Adapter für den remote-Zugriff auf den MBean-Server bereitstellen.
- Zusätzliche APIs: Definieren, wie JMX mit anderen Managementtechnologien wie z.B. SNMP, CIM/WBEM integriert werden soll. Details dazu findet man in zusätzlichen JSRs (Java Specification Request).

ten Stelle entpackt, z.B. nach *C:* unter Windows oder */opt* unter Unix. Dies erzeugt dort die in Abbildung 1 gezeigte Verzeichnisstruktur. Im Folgenden bezeichnen wir das oberste Verzeichnis davon mit *JBOSS_HOME*. Gehen wir die Verzeichnisse unter *JBOSS_HOME* einmal kurz durch:

- *bin*: Diverse Kommandoskripte, darunter die oben erwähnten zum Starten und Stoppen des Servers, sowie davon benutzte JAR-Files.
- *client*: Bibliotheken für stand-alone-Clients.
- *docs*: DTDs für diverse Konfigurations-Files und Deployment Descriptoren sowie beispielhafte Service-Definitionen.
- *lib*: Bibliotheken für den Bootstrap des Servers.
- *server*: Verzeichnis für Configuration Sets, Unterverzeichnisse mit jeweils einer Server-Konfiguration.

Vor dem Lauf des Servers muss ggf. noch JDK 1.3 (oder höher) installiert und sein Installationsverzeichnis in der Umgebungsvariable *JAVA_HOME* abgelegt werden. Dann steht dem ersten Start mittels *%JBOSS_HOME%\bin\run.bat* (Windows) bzw. *$JBOSS_HOME/bin/run.sh* (Unix) nichts mehr entgegen. Zum Stoppen des Servers ist im gleichen Verzeichnis ein Script namens *shutdown.bat* bzw. *shutdown.sh* vorhanden. Das Killen des Prozesses tut es allerdings auch.

## Configuration Sets enthalten die Einstellparameter

Die Einstellung des Servers und das Deployment von Diensten und Anwendungen geschieht ausschließlich in den Configuration Sets. Werfen wir auch hier einen zunächst groben Blick auf ihren Inhalt:

- *Conf:* Basiskonfiguration des Servers sowie Konfigurations-Files einzelner Dienste.
- *Deploy:* Hier werden Services und Applikationen platziert.
- *Lib*: Bibliotheken für Services und Applikationen.

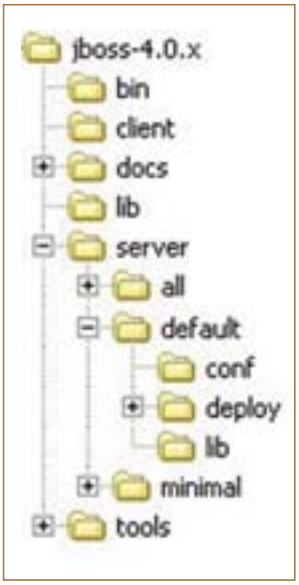
Während des Laufs des Servers werden noch weitere Verzeichnisse angelegt, z.B. *log* für Protokolldateien.

Das Installationsfile enthält die Konfigurationen *minimal*, *default* und *all*. Die gewünschte Konfiguration kann beim Starten des Servers gewählt werden: *run -c ConfigSetName*. Standardmäßig wird *default* verwendet. Es können beliebige neue Konfigurationen durch Erstellen eines entsprechenden Unterverzeichnisses von *server* erzeugt werden; zweckmäßigerweise kopiert und modifiziert man eine der schon vorhandenen Konfigurationen.

## Modulare Architektur

Bestechend an der Architektur von JBoss ist die Modularität des Servers. Kern des Systems ist ein JMX-Server, der als Basis für alle weiteren Dienste in Form von

Abb. 1: JBoss-Verzeichnisstruktur




MBeans dient (siehe Kasten „JMX“). Das Laden der Dienste übernimmt dabei eine Reihe von Deployern:

- *MainDeployer:* Einstiegspunkt für jedes Deployment, delegiert an den passenden Deployer.
- *JARDeployer*: Lädt Java-Bibliotheken (**.jar*) in den Classpath.
- *SARDeployer:* Lädt JBoss-Services *(*.sar*).
- *CMDeployer:* Lädt Datasources *(*-ds.xml*).
- *RARDeployer:* Lädt Resource-Adapter *(*.rar*).
- *EJBDeployer:* Lädt EJBs *(*.jar*).
- *WARDeployer:* Lädt Web-Applikationen *(*.war*).

- *EARDeployer:* Lädt Enterprise-Applikationen *(*.ear)*.

Alle Deployer können anstatt der mit *jar* gepackten Archive auch gleich benannte Verzeichnisse mit entsprechendem Inhalt deployen.

Die Funktionalität der letzten vier dieser Liste dürfte dem Leser hinlänglich bekannt sein: Sie sind für die bekannten J2EE-Deployment-Einheiten zuständig. Auch der *JARDeployer* ist leicht erklärt: Er erweitert den Classpath um den Inhalt eines JAR-Files.

## JBoss-Services hinzuladen

Interessanter wird es bei der Betrachtung des *SARDeployers*. Er kümmert sich um Dateien mit der Endung *.sar*. Diese Service Archives sind ähnlich wie andere Deployment-Einheiten JAR-Files, die neben Java-Code einen Deployment Descriptor namens *META-INF/jboss-service.xml* enthalten. Häufig benötigt ein Dienst keinen eigenen Java-Code, weil die entsprechenden Klassen bereits geladen wurden, z.B. als Bestandteil des JBoss-Kerns. Das SAR-File, das dann ausschließlich den Deployment Descriptor enthielte, kann in diesem Fall durch den Descriptor als allein stehende Datei ersetzt werden. Der Name der Datei muss auf *-service.xml* enden. Der Aufbau dieser Datei kann der freie bzw. käuflich erwerbbaren Dokumentation der JBoss Group entnommen werden.

Main-, JAR- und SARDeployer sind Bestandteil des JBoss-Kerns, die restlichen Deployer werden abhängig von der Konfiguration dazu geladen.

Die erste Service-Beschreibung, die der SAR-Deployer verarbeitet, ist die Datei *conf/jboss-service.xml* des aktuell verwendeten Configuration Sets. In der *default*-Konfiguration werden dadurch zunächst alle Bibliotheken aus dem *lib*-Verzeichnis dem Classpath hinzugefügt, Logging, JNDI etc. aktiviert und schließlich der Deployment-Scanner geladen. Diese MBean vom Typ *org.jboss.deployment.scanner.URLDeploymentScanner* hat die Aufgabe, eine oder mehrere URLs zu überwachen, wobei diese lokal (*file:*) oder remote (*http:*) sein können. URLs, die auf / enden, werden als Verzeichnisse interpretiert, die nach Deployment-Einheiten durchsucht werden sollen. Neu erkannte

## XPetstore: Eine komplexe J2EE-Anwendung für JBoss

Damit Sie das hier vermittelte Wissen über Installation und Konfiguration des JBoss Application Server direkt in der Praxis anwenden können, wollen wir Ihnen noch kurz den XPetstore vorstellen. XPetstore ist eine auf XDoclet [3] basierende Re-Implementierung der von Sun veröffentlichten Petstore-Anwendung [4]. Dabei handelt es sich um einen Online-Shop für Haustiere auf Basis von EJBs, Servlets und JSPs. Auf Grund der Tatsache, dass sich JBoss 4.0 momentan noch in der Entwicklung befindet, sollten Sie den XPetstore noch auf dem aktuellen Produktivstand 3.2.2 ausprobieren. Die hier beschriebenen Schritte beziehen sich auf XPetstore-EJB. Es gibt auch noch ein XPetstore-Servlet, das ohne EJBs und mit JDO arbeitet.

### Download & Installation

Um den XPetstore life zu sehen, müssen die folgenden Schritte durchgeführt werden:

- Entpacken von JBoss 3.2.2 und das Setzen der Umgebungsvariablen JBOSS_HOME entsprechend Ihres Verzeichnisses (befindet sich auf der Heft-CD).
- Eine eigene JBoss-Konfiguration mit dem Namen *xpetstore* erzeugen.
- 2 JMS Queues mit den JNDI-Namen *queue/order* und *queue/mail* anlegen. Diese werden in der Konfigurationsdatei *jbossmq-destinations-service.xml* definiert, die sich im *deploy*-Verzeichnis im Unterverzeichnis *jms* befindet. Diese Datei sollte so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
 <mbean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination:service=Queue,name=order">
  <attribute name="JNDIName">queue/order</attribute>
  <depends optional-attribute-name="DestinationManager">
   jboss.mq:service=DestinationManager
  </depends>
 </mbean>
 <mbean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination:service=Queue,name=mail">
  <attribute name="JNDIName">queue/mail</attribute>
  <depends optional-attribute-name="DestinationManager">
   jboss.mq:service=DestinationManager
  </depends>
 </mbean>
</server>
```

- Erzeugen einer Datasource mit dem JNDI-Namen *xpetstoreDS*. XPetstore unterstützt alle gängigen Datenbanken. Im Falle einer SAP DB kann das Beispiel genutzt werden.
- Konfiguration des JBoss Mail-Service mit dem JNDI-Namen *Mail*.
- Entpacken von XPetstore (befindet sich auf der Heft-CD).
- Anpassen der Datei *XPETSTORE_HOME/conf/as/appserver.properties* und setzen der Property *app.server=jboss*. Evtl. überprüfen, ob die Einstellungen in *XPETSTORE_HOME/conf/as/jboss.properties* Ihrer Konfiguration entsprechen.
- Anpassen der Datei *XPETSTORE_HOME/conf/db/database.properties* und die Property *db.name* entsprechend ihrer Datenbank setzen.
- In der Datei *XPETSTORE_HOME/conf/db/<db.name>.properties* die Einstellungen für Ihre Datenbank vornehmen.
- Starten Sie JBoss mit der Konfiguration *xpetstore*.
- Wechseln Sie in das Verzeichnis *XPETSTORE_HOME/xpetstore-ejb* und führen Sie der Reihe nach die Kommandos *build.bat*, *build.bat deploy* und *build.bat db* aus.
- Sie erreichen Ihren Petstore nun mit der URL *http://localhost:8080/xpetstore-ejb*.

Einträge werden automatisch deployed, Änderungen und Löschungen von Einträgen führen zum Redeployment bzw. Undeployment der entsprechenden Anwendung. Standardmäßig überwacht der Deployment-Scanner das Verzeichnis *deploy* des Configuration Sets. Dies geschieht in Abständen von 5 Sekunden, wobei diese Zeit durch das MBean-Attribut *ScanPeriod* verändert werden kann. Mit Hilfe eines Deployment-Sorters kann erreicht werden, dass Anwendungen und Dienste in einer bestimmten Reihenfolge deployed werden, wenn sie gleichzeitig erkannt werden. Es stehen zwei Varianten zur Verfügung, die als Wert des MBean-Attributs *URLComparator* verwendet werden können: Eine, die anhand der Dateiendung (und damit des Typs der Anwendung oder des Dienstes) sortiert, und eine, die zusätzlich einen evtl. vorhandenen, numerischen Präfix im Dateinamen mit zur Sortierung heranzieht.

**Listing 1**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE server>
<server>
 ...

 <mbean code="org.jboss.deployment.scanner.
                                URLDeploymentScanner"
         name="jboss.deployment:type=
                        DeploymentScanner,flavor=URL">
  <depends optional-attribute-name="Deployer">
   jboss.system:service=MainDeployer
  </depends>
  <attribute name="URLComparator">
   org.jboss.deployment.scanner.PrefixDeploymentSorter
  </attribute>
  <attribute name="Filter">
   org.jboss.deployment.scanner.DeploymentFilter
  </attribute>
  <attribute name="ScanPeriod">3000</attribute>
  <attribute name="URLs">deploy/,applications/
  </attribute>
 </mbean>
</server>
```

**Listing 2**

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
 <local-tx-datasource>
  <jndi-name>SapdbDS</jndi-name>
  <connection-url>jdbc:sapdb://127.0.0.1/DB_NAME
  </connection-url>
  <driver-class>com.sap.dbtech.jdbc.DriverSapDB
  </driver-class>
  <user-name>x</user-name>
  <password>y</password>
  <transaction-isolation>TRANSACTION_READ_COMMITTED
  </transaction-isolation>
  <min-pool-size>5</min-pool-size>
  <max-pool-size>100</max-pool-size>
 </local-tx-datasource>
</datasources>
```

Eine sinnvolle Modifikation der Konfiguration zeigt die Abwandlung der Datei *conf/jboss-service.xml* in Listing 1. Hier wird neben *deploy* auch das Verzeichnis *applications* im Abstand von 3 Sekunden überwacht und Deployment-Einheiten nach numerischem Präfix und Dateiendung sortiert.

Nach diesen Vorbereitungen wird JBoss also nach dem Starten seines Mini-Kerns weitere Dienste aus den Deploy-Verzeichnissen laden. Diese sind – wie bereits erwähnt – als Dateien oder Verzeichnisse mit der Endung *.sar* oder als Dateien, deren Name auf *-service.xml* endet, abgelegt. Beispiele finden Sie in *deploy jbossweb.sar* oder *deploy/counter-service.xml*.

Der Deployment-Scanner wird die gefundenen Einträge übrigens immer an den *MainDeployer* geben, der dann bei allen Sub-Deployern anfragt, welcher den jeweiligen Typ verarbeiten kann. Im Falle von JBoss-Services ist dies wiederum der *SARDeployer*.

## Datenbankverbindungen definieren

Nahezu jede Applikation benötigt Zugriff auf eine Datenbank. J2EE-Anwendungen sollten dazu Datasources verwenden, da damit die konkreten Verbindungsparameter wie Treiber, URL, Username etc. nicht in der Anwendung ‚eingebrannt' werden, sondern sich an zentraler Stelle im Server befinden. Zudem kommt die Anwendung über Datasources leicht in den Genuss von DB-Verbindungspools. Lassen Sie uns deshalb nun noch abschließend schauen, wie man in JBoss Datasources definiert.

Jede Datasource wird – wie andere Dienste auch – intern durch eine entsprechende MBean repräsentiert. Daher sind hier auch wieder Definitionen von JBoss-Services gefragt. Seit JBoss 3.2 steht eine vereinfachte Möglichkeit zur Definition von DB-Verbindungen zur Verfügung. Descriptoren mit der Dateiendung *-ds.xml* werden durch den zugehörigen Deployer (*CMDeployer*) einer XSL-Transformation unterzogen und dann als JBoss-Services deployed. Die Descriptoren sind durch diese Änderung recht übersichtlich geworden. Listing 2 zeigt die Anbindung einer SAP DB.

Weitere Beispiele finden sich im Verzeichnis *JBOSS_HOME/doc/examples/jca*. Darin findet sich auch eine Datei namens *generic-ds.xml*, die alle möglichen Parameter enthält.

## Fazit

Zusammenfassend kann festgehalten werden, dass JBoss ein sehr robuster und ausgereifter J2EE Application Server ist, der darüber hinaus für die Entwicklung neuer Standards und Technologien richtungsweisend ist, wie das Beispiel AOP zeigt. Zwar fehlt zur Zeit noch eine ausgereifte grafische Konfigurations- und Management-Konsole, aber auf Grund der Open Source-Eigenschaft und den kommenden Spezifikationen von Sun (JSR-77 J2EE Management, JSR-88 J2EE Deployment) kann erwartet werden, dass bald entsprechende Werkzeuge vorhanden sind. In der nächsten Ausgabe des *Java Magazin*s werden wir zeigen, wie JBoss im Cluster betrieben werden kann und was in diesem Bereich unterstützt wird. ■

*Dirk Weil ist Geschäftsführer der Gedoplan GmbH in Bielefeld [7]. Marcus Redeker ist Senior Java Consultant im gleichen Unternehmen. Beide sind zertifizierte JBoss Consultants und seit mehr als 5 Jahren als Berater und Trainer im Bereich Java und Enterprise Java tätig.*

### Links & Literatur

[1] www.jboss.org/

[2] www.jbossgroup.com/

[3] Stefan Edlich: „Beans aufblasen", *Java Magazin* 01.2003

[4] java.sun.com/blueprints/code/jps131/docs/index.html

[5] eclipse.org/aspectj/

[6] sourceforge.net/projects/jboss/

[7] www.gedoplan.de/