Migration einer J2EE-1.4-Anwendung von JBoss 4 bis 7

Von 4 über 5 nach 7

Natürlich sammelt jeder seine eigenen Erfahrungen bei einer Migration. Es kann daher sein, dass Ihre Anwendung ohne oder mit nur wenigen Änderungen auf allen JBoss-Versionen lauffähig ist. Wahrscheinlicher ist es jedoch, dass Sie bei der Migration auf ähnliche Probleme wie der Autor stoßen. In diesem Fall sollten Sie folgende Lösungen konsultieren.

von Marcus Schießer

In diesem Artikel werden Praxiserfahrungen beschrieben, die der Autor bei der Migration einer J2EE-1.4-Altanwendung von JBoss 4.2.2 GA nach JBoss 5.0.1 EAP machte. Anschließend erläutert der Autor beispielhaft die Migration der Anwendung zu der neuen JBoss-Version 7.0.0. Bei JBoss 4.2.2 GA und 7.0.0 handelte es sich um die frei verfügbare Community Edition. Sie kann unter http://www.jboss.org heruntergeladen werden. JBoss 5.0.1 EAP war jedoch die Enterprise Edition, die auf der Version 5.1 der Community Edition aufbaut. Der Ende letzten Jahres veröffentlichte JBoss 6 wird hier nicht behandelt, da es sich um ein wenig dokumentiertes Zwischenrelease handelt und die nächste Enterprise Edition auf der Community Edition 7 aufbauen wird. Die zu migrierende Anwendung basiert auf einer Drei-Schicht-Architektur, die Hibernate 3.3.2 in der Persistenzschicht, Session Beans nach EJB 2.1 in der Businesslogik und schließlich Swing für den Rich Client verwendet. Die serverseitigen Komponenten der Anwendung wurden als ein einziges EAR gepackt und im JBoss verteilt. Die Swing-Anwendung wurde über eine eigenentwickelte Komponente ähnlich Java WebStart unternehmensweit verteilt und kommuniziert mit den Session Beans über den HTTP-Invoker [1] von JBoss. Als Besonderheit verfügt die Anwendung noch über ein eigenes Managed Bean (MBean), das über Quartz [2] zeitgesteuerte Abläufe startet. Da es sich um ein MBean handelt, ist dieses über das JBoss-eigene SAR-Format innerhalb des EARs verpackt. Außerdem existiert noch ein WAR, das ein kleines Servlet enthält, über das der Administrator den Zustand der Anwendung kontrolliert. Zur Verdeutlichung zeigt **Abbildung 1** ein vereinfachtes Deployment-Diagramm der Architektur.

Bei der Migration wurde die Anwendung nicht nach JEE5 oder JEE6 migriert, sondern lediglich derart angepasst, dass sie mit dem JBoss EAP 5.0.1 fehlerfrei läuft. Auch für diesen vermeintlich kleinen Schritt waren einige Änderungen notwendig, nicht zuletzt weil sich zentrale Komponenten wie der Transaktionsmonitor oder die Security zwischen beiden JBoss-Versionen geändert haben. Die Angabe der Dateien ist in dem Artikel relativ zu dem Verzeichnis der Default-Server-Konfiguration, die jeweils auch Grundlage für die Migration war. Diese existiert für beide JBoss-Versionen (4.2.2 und 5.0.1) und wird standardmäßig beim Start von JBoss verwendet. Die Default-Server-Konfigurationen befindet sich vom Hauptverzeichnis aus gesehen in dem Unterverzeichnis *server/default*.

Neuer Transaktionsmonitor

Zunächst betrachten wir die Konfiguration der internen Services von JBoss zwischen Version 4.2.2 und 5.0.1. Diese sind in der Datei *conf/jboss-service.xml* konfiguriert. Hier fällt auf, dass im Vergleich zur vorherigen Version die Services für Security, der Transaktionsmonitor und der Deployment Scanner in JBoss 5.0.1 wegfallen. Das liegt daran, dass JBoss seit Version 5 auf einer sehr reduzierten Microcontainer-Architektur basiert, die Services als POJO-Komponenten und nicht mehr als MBeans implementiert wie bei dem JMX-Kernel von JBoss 4. Daher sind die genannten Services in Version 5 POJOs, die im Verzeichnis *deploy* installiert und konfiguriert werden. An der von JBoss gelieferten Microcontainer-Konfiguration mussten für Version 5.0.1 keine Anpassungen vorgenommen werden, sodass die Standardkonfiguration (*conf/jboss-service.xml*) für die Migration übernommen wurde.

Bezüglich des Transaktionsmonitors war für die Anwendung in der alten JBoss-4.2.2-Konfiguration

jedoch der Fast In-Memory Transaction Manager konfiguriert. Er war bereits in JBoss 4.2.2 veraltet und wurde nun ab Version 5 endgültig entfernt. Für Version 5 wurde der Transaktionsmanager von der Firma Arjuna aufgekauft und als JBossTS dem JBoss hinzugefügt. Konfiguriert werden kann er in der Datei *deploy/transaction-jboss-beans.xml*. Da die Anwendung sehr langläufige Transaktionen enthält, musste der Time-Out des Transaktionsmonitors angepasst werden. Hierzu wurde das Property *transactionTimeout* auf 3600 Sekunden (oder eine Stunde) in dieser Datei folgendermaßen erhöht: *<property name="transactionTimeout">3600</property>*. Der neue Transaktionsmanager bereitete zunächst keine Probleme, bei genauerem Testen gab es jedoch während der Laufzeit Ausnahmefehler, wenn eine Transaktion zwei Datasources verwendete. Hintergrund war, dass in der Deployment-Datei der Datasources (in diesem Fall Oracle-basiert *oracle-ds.xml*) die beiden an der Transaktion beteiligten Datasources als lokale TX-Datasources (über das Tag *local-tx-datasource*) konfiguriert waren. Das Problem trat nicht mehr auf, nachdem der Autor beide Datenquellen als XA-Datasources (über das Tag *xa-datasource*) konfiguriert hatte. Das war erstaunlich, da sich beide in derselben Datenbankinstanz befanden und daher eigentlich keine XA-Datasource notwendig sein müsste, die im Gegensatz zur lokalen TX-Datasource verteilte Transaktionen erlaubt. Möglicherweise ist der Transaktionsmanager an dieser Stelle nun strikter, sodass er generell keine Transaktion mit zwei lokalen TX-Datasources erlaubt.

Änderungen an der Security

Gleich vorab: Dieser Abschnitt behandelt nur die notwendigen Änderungen an der JBoss-5.0.1-Konfiguration, um die Anwendung fehlerfrei lauffähig zu bekommen – der Artikel behandelt nicht die Absicherung einer JBoss-Instanz. Die eingesetzte Anwendung verwendete schon in der Konfiguration für JBoss 4.2.2 eine eigene *conf/standardjboss.xml*. In dieser Datei werden generell Parameter für die EJBs abhängig von ihrem Typ (z. B. Session Bean oder Entity Bean) festgelegt. Gültig ist diese Datei nur für EJB 2.1 Beans – aus Kompatibilitätsgründen ist sie jedoch ebenfalls in JBoss 5.0.1 enthalten, um EJB-2.1-basierte Anwendungen, wie die unsere, verwenden zu können. Um EJBs ab der Version 3 zu konfigurieren, muss man gegebenenfalls die Dateien *deploy/ejb3-connectors-jboss-beans.xml, deploy/ejb3-container-jboss-beans.xml* und *deploy/ejb3-interceptors-aop.xml* anpassen.

Für die zu migrierende Anwendung wurde von der Standardkonfiguration abgewichen, unter anderem um eigene Log Interceptors für die Session Beans zu verwenden. Ein Interceptor ist ein Konzept aus der aspektorientierten Programmierung, das es erlaubt, bei einem Methodenaufruf einen weiteren Aufruf vorweg- und/oder nachzuschalten. Bei unserer Anwendung wurde bei jedem Aufruf einer Methode einer Session Bean eine Log-Meldung erzeugt.

Um die alte Konfigurationsdatei verwenden zu können, musste der Autor zunächst die DTD für JBoss 5.0.1 aktualisieren. Die DTD musste auch an weiteren Stellen aktu-

alisiert werden, in jedem Fall wurde dabei einfach der DOCTYPE im Header der Datei ausgetauscht:

```
<!DOCTYPE jboss PUBLIC"-//JBoss//DTD JBOSS 5.0//EN" "http://www.jboss.
                                                    org/j2ee/dtd/jboss_5_0.dtd">
```

Danach konnte die Datei zwar verwendet werden, es trat jedoch ein seltsamer Fehler auf, der erst nach einiger Analysezeit dieser Datei zugeordnet werden konnte. Das Problem innerhalb der Anwendung war, dass das Login immer erfolgreich war, auch wenn ein falsches Passwort angegeben wurde, die Autorisierung war also immer erfolgreich. Die Ursache war, dass in JBoss 5 die Security über neue Interceptors geregelt wird und diese zwingend in *conf/standardjboss.xml* konfiguriert werden müssen. In der alten Konfigurationsdatei von JBoss 4.2.2, die als Basis für die verwendete Konfigurationsdatei diente, waren diese natürlich nicht konfiguriert. Das Problem wurde behoben, indem der Autor in der Datei für alle Invoker-Konfigurationen den *SecurityContextInterceptor* hinzufügte:

```
<interceptor>org.jboss.proxy.ejb.SecurityContextInterceptor</interceptor>
```

Für alle Bean-Konfigurationen wurde *PreSecurityInterceptor* hinzugefügt:

```
<interceptor>org.jboss.ejb.plugins.security.PreSecurityInterceptor
                                                          </interceptor>
```

Zusätzlich für alle Stateful Session Beans wurde *StatefulSessionSecurityInterceptor ergänzt*:

```
<interceptor>org.jboss.ejb.plugins.StatefulSessionSecurityInterceptor
                                                          </interceptor
```

Da wir gerade beim Thema Security sind: In der Datei *login-config.xml* werden üblicherweise die Security Domains für den Java Authentication and Authorization Service (JAAS) konfiguriert. Pro Security Domain werden dabei die Loginmodule, die für eine Anwendung die Authentifizierung übernehmen, und deren Konfigurationsparameter festgelegt. In einer Anwendung wird dann lediglich referenziert, zu welcher Security Domain diese gehört, dadurch wird die eigentliche Autorisierung von der Anwendung getrennt.

Vergleicht man die Standardkonfiguration der Datei *login-config.xml* zwischen JBoss 4.2.2 und 5.0.1, so wurde die DTD aktualisiert und die Security Domain für JBoss MQ entfernt, das in Version 5.0.1 durch JBoss Messaging ersetzt wurde. Für die Migration übernahm der Autor einfach die Security Domain der Anwendung in die Standardkonfiguration von JBoss 5.0.1.

Ein weiteres Securityproblem trat in Zusammenhang mit dem HTTP-Invoker auf. Dieser ermöglicht es, Aufrufe der EJBs über HTTP zu tätigen, hierbei dient der HTTP-Invoker als Adapter, der die eigentlichen RMI-Aufrufe kapselt. Standardmäßig verwendet dieser HTTP-Invoker in Version 5.0.1 die Security Domain der JMX Console (*jmx-console*), anstatt, wie noch in Version 4.2.2, die Security Domain der EJB-Konfiguration aus der Datei *jboss.xml*. Insofern entsprach das Verhalten in der alten Version eher der Erwartung.

Dieses Problem führte während der Laufzeit zu einer *SecurityException*, die verhindert werden konnte, indem für den HTTP-Invoker generell die Security Domain der Anwendung gesetzt wurde. In unserem Fall war das kein Problem, da nur unsere Anwendung den HTTP-Invoker in JBoss verwendet hat. Hier muss man gegebenenfalls den HTTP-Invoker mehrmals, d. h. einmal pro Anwendung deployen. Alternativ kann man hier auch eine aktuellere EAP-Version als 5.0.1 verwenden, da dieses Problem laut JBoss mittlerweile behoben ist. Um die Security Domain für den HTTP-Invoker anzupassen, wurde diese in der Datei *deploy/http-invoker.sar/inwoker.war/WEB-INF/jboss-web.xml* geändert (im Beispiel hat die Anwendung die Security Domain *MySecurityDomain*):

```
<jboss-web>
  <security-domain>java:/jaas/MySecurityDomain</security-domain>
</jboss-web>
```

Sonstige Konfigurationsanpassungen

Ansonsten wurde die Datei für die Log4J-Konfiguration von *log4j.xml* in *jboss-log4j.xml* umbenannt. Das entspricht dem Standard für JBoss 5 und wird so in der *jboss-service.xml* referenziert (die unverändert von der JBoss-5.0.1-Standardkonfiguration übernommen wurde). Außerdem musste der JBoss-eigene Scheduling-Service aus der Serverkonfiguration entfernt werden. Für diesen Zweck müssen die Dateien *scheduler-manager-service.xml, scheduler-service.xml* und der Resource-Adapter *quartz-ra.rar* aus dem Verzeichnis *deploy* gelöscht werden. Möchte man den Scheduling-Service nur deaktivieren, können die Dateien alternativ auch in ein anderes Verzeichnis (z. B. mit dem Namen *nodeploy*) verschoben werden. Dieser Schritt war notwendig, da die Anwendung und der Scheduling-Service beide Quartz verwenden und die Anwendung für Quartz eine eigene Konfiguration mitliefert. Sie wurde von dem Resource-Adapter *quartz-ra.rar* fälschlicherweise als ihre eigene angesehen (obwohl in einer eigenen EAR verpackt) und geladen. Da in der Quartz-Konfiguration eine Datasource verwendet wird, die zu diesem Zeitpunkt noch nicht geladen war, gab es einen Ausnahmefehler beim Starten des Application Servers.

Anpassungen an der Anwendung

Bisher wurde nur die Konfiguration des Application Servers behandelt, aber für die Migration waren auch Änderungen an der Anwendung selbst notwendig. Zunächst musste die DTD in der *jboss-web.xml* des WAR-Archivs aktualisiert werden. Hierzu wurde folgender Header hinzugefügt:

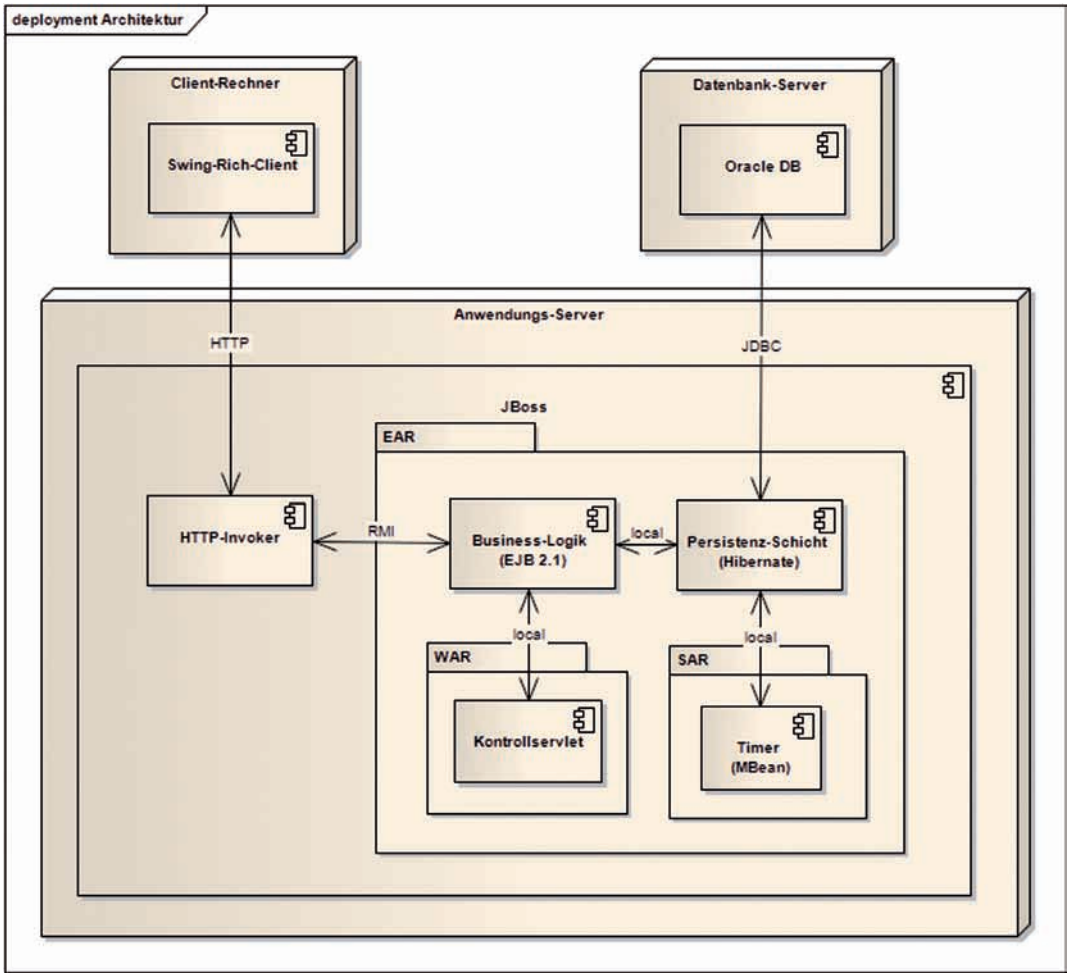

Abb. 1: Deployment-Diagramm der zu migrierenden Anwendung

```
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 5.0//EN"
                           "http://www.jboss.org/j2ee/dtd/jboss-web_5_0.dtd">
```

Wie am Anfang erwähnt, liefert die Anwendung ein eigenes MBean mit, das sich in einem SAR-Archiv befindet. Hier hat sich offensichtlich im Vergleich zur JBoss-Version 4.2.2 das SAR-Archivformat geändert, da die zu dem MBean abhängigen JARs nicht mehr wie zuvor in dem Ordner *lib* des SARs gefunden wurden. Nachdem diese in das Root des SAR-Archivs kopiert wurden, konnten sie jedoch korrekt referenziert werden. Außerdem konnte das MBean seine Datasource nicht verwenden, da diese zu dem Zeitpunkt des Ladens der MBean noch nicht geladen war. Hintergrund war, dass eine Referenz zu der Datasource in der Konfigurationsdatei *jboss-service.xml* des SAR-Archivs fehlte. Diese musste in der MBean-Definition innerhalb des *<mbean>* Tags hinzugefügt werden mit:

```
<depends>jboss.jca:service=DataSourceBinding,name=AppDS</depends>
```

Hierbei steht *AppDS* für den Namen der Datasource, die das MBean referenzieren soll. Das Deployment für JBoss 4.2.2 hat ohne diese Referenzierung funktioniert.

Zum Abschluss stellten sich noch zwei kleinere Classloading-Probleme heraus: Einerseits lieferte die Anwendung ihre eigene Version der Quartz Library (Version 1.6.0) in seinem EAR mit – diese wurde vom Classloader jedoch nicht gefunden, sondern die Version 1.5.2, die JBoss in dem Pfad *common/lib* (vom Hauptverzeichnis aus gesehen) ausliefert. Um unserer eigenen Version Präferenz beim Classloading zu geben, wurde die Datei *quarz-all-1.6.0.jar* in den Library-Ordner der Serverkonfiguration (*server/default/lib*) kopiert. Ein weiteres Classloading-Problem trat beim Laden einer anwendungseigenen Konfigurationsdatei (Java Properties) auf. Diese wurde bei JBoss 4.2.2 korrekt aus dem Verzeichnis *conf* von JBoss geladen. Das Problem war nun, dass sich diese Datei mit anderen Konfigurationsparametern fälschlicherweise zusätzlich in einem JAR innerhalb des SARs befand. Diese Datei bekam in JBoss 5.0.1 die Präferenz gegenüber der eigentlichen Konfigurationsdatei im Verzeichnis *conf* und wurde daher mit den falschen Konfigurationsparametern geladen. Nachdem das Problem gefunden wurde, musste die doppelte Konfigurationsdatei innerhalb des SARs gelöscht werden. Danach wurde die richtige Konfigurationsdatei aus dem Verzeichnis *conf* verwendet.

Aus 5 mach 7

Damit ist der Teil der Migration der Anwendung auf JBoss 5.0.1 abgeschlossen. Der Autor schaute sich in

diesem Zusammenhang noch die Verteilung der Anwendung auf der neuen JBoss-Version 7 an. Dieser Abschnitt erklärt jedoch nicht die neuen Funktionen, konsultieren Sie hierzu bitte den Artikel „Ride the Lightning“ auf Seite 16, in dem Sie eine ausführliche Einführung zu JBoss 7 finden.

Zunächst trat während des Deployments eine Vielzahl an Ausnahmefehlern auf, weil eine Klasse nicht gefunden wurde (*ClassNotFoundException*). Der Hintergrund war, dass mit JBoss 7 das komplette Classloading auf JBoss Modules [3] umgestellt wurde. Dabei werden Klassen zu Modulen zusammengefasst und Abhängigkeiten zwischen den einzelnen Modulen explizit festgelegt. Besteht zu einem Deployment (z. B. einem EAR) keine explizite Abhängigkeit zu einem Modul, so werden die Klassen dieses Moduls nicht geladen. In unserem Fall fehlte die Abhängigkeit zu der verwendeten Hibernate-Version und zu Log4J. Erstere wurde einfach in das Verzeichnis *lib* des EARs kopiert. Da Log4J als Modul jedoch schon vom Application-Server zur Verfügung gestellt wird, muss man das JAR nicht zusätzlich in das EAR kopieren. Besser ist es stattdessen zu definieren, dass das EAR von dem von JBoss mitgelieferten Log4J-Modul (mit dem Namen *org.apache.log4j*) abhängig ist. Um JBoss das mitzuteilen, kann einfach folgende Zeile in die Manifest-Datei (*MANIFEST.MF*) des EARs eingefügt werden: *Dependencies: org.apache.log4j*. Anschließend waren alle Classloading-Probleme behoben. Für andere Deployments funktioniert die Definition einer Abhängigkeit analog.

Danach trat beim Deployment der Session Beans ein Parsing-Fehler in dem EJB-Deployment-Deskriptor *ejb-jar.xml* auf. Der Parser erwartete, dass innerhalb der <*method*>- und <*method-permission*>-Tags keine <*description*>-Tags auftreten. Da diese nur Kommentare darstellen und keine Bedeutung zur Laufzeit haben, wurden sie für diesen Test entfernt. Darauf wurden die Session Beans fehlerfrei im JBoss 7 deployt, obwohl es sich noch um EJB 2.1 Beans handelte. Sie fanden sich nun lediglich in dem mit EJB 3.1 eingeführten standardisierten JNDI Namespace [4], sodass der Service Locator [5] der Anwendung angepasst werden musste. Nach Aussage von JBoss sollen diese Probleme ab Version 7.1 behoben sein, da diese Version EJB 2.1 Beans wieder komplett unterstützt.

Etwas Aufwand erforderte auch die Anpassung der Datasource-Konfiguration. Sie ist nun in einem neuen Format und wird direkt in der Datei für die gesamte Serverkonfiguration konfiguriert. Außerdem wird in ihr nun auch der von der Datasource verwendete JDBC-Datenbanktreiber als JBoss Module referenziert. Wie die Konfiguration genau verläuft, würde den Rahmen des Artikels sprengen, eine Beschreibung für eine MySQL Datasource befindet sich jedoch unter [6].

Soweit lief die Migration zu JBoss 7 relativ problemlos. Schwierigkeiten hat dann jedoch die Tatsache bereitet, dass sich das SAR-Archiv der Anwendung nicht mehr deployen ließ. Zwar unterstützt Version 7 noch MBeans, in unserem Fall ließ sich der Deskriptor jedoch auch nach einer Aktualisierung der DTD nicht mehr parsen. Hier müsste man genauer analysieren, was die Ursache ist und den Deskriptor entsprechend anpassen. Da das MBean für unsere Anwendung zeitgesteuerte Aufgaben ausführt, könnte man es alternativ auch in einen EJB 3.1 Timer konvertieren.

Ein weiteres Problem war, dass JBoss 7 nicht mehr den HTTP-Invoker enthält. Die Version des HTTP-Invokers von JBoss 5.0.1 EAP ließ sich ebenfalls nicht auf dem neuen JBoss deployen. Für diesen Migrationsschritt sollte man ganz auf den HTTP-Invoker verzichten und den Rich Client auf einen Transport mit RMI umstellen.

Fazit

An dieser Stelle wurde die Migration nach JBoss 7 unterbrochen, da das Deployment des MBeans und die Reimplementierung der Transportschicht zu zeitaufwändig gewesen wären. Fazit ist, dass sich die Migration einer Altanwendung (J2EE 1.4) von JBoss 4 auf 5 aufwändiger gestaltet hat, als zunächst vermutet. Zwar ist es einfach, Konfigurationsdateien an ein neues Format anzupassen, aber für das geänderte Security- und Transaktionsverhalten lohnt es sich auf jeden Fall, einen höheren Analyse- und Testaufwand einzuplanen. Diesen Aufwand sollte man sicherlich auch für eine Migration von JBoss 5 nach 7 einplanen, auch wenn die Migration in den bereits durchgeführten Schritten relativ problemlos verlief. Verwendet man wie im Beispiel EJB 2.1 Beans, lohnt es sich außerdem, auf Version 7.1 zu warten, da hier eine bessere Unterstützung angekündigt ist. Ebenso sollte man beim Einsatz des HTTP-Invokers beachten, dass dieser nicht mehr unterstützt wird und daher auch Anpassungen an der Anwendung selbst notwendig sind.

Marcus Schießer ist freiberuflicher Softwarearchitekt und Consultant aus Karlsruhe. Seit seiner Diplomarbeit im Jahr 2002 verwendet er JBoss, damals noch in der Version 2.4. Aktuell arbeitet er in einem Projekt als Technischer Architekturverantwortlicher bei der DB Systel GmbH in Frankfurt.

Links & Literatur

[1] http://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/JNDI_over_HTTP-Accessing_JNDI_over_HTTP.html

[2] http://www.quartz-scheduler.org/

[3] http://planet.jboss.org/view/post.seam?post=modularized_java_with_jboss_modules

[4] https://docs.jboss.org/author/display/AS7/Developer+Guide#DeveloperGuide-JNDIPortableJNDISyntax

[5] http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html

[6] https://docs.jboss.org/author/display/AS7/Developer+Guide#DeveloperGuide-DataSourceConfiguration