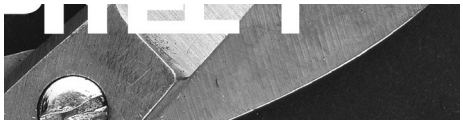

1 Einführung

1.1 Motivation

J2EE und JBoss – beides zwei der Hauptthemen heutiger Enterprise-Applikationen. J2EE (alias Java 2 Platform, Enterprise Edition) ist ein De-facto-Standard, der Einzug in die Entwicklung vieler Unternehmensapplikationen gehalten hat. So gibt es kaum ein Großunternehmen, das nicht diese Technologie einsetzt. Aber J2EE ist nichts ohne einen geeigneten Server. Ein Unternehmen, das sich also für J2EE entschieden hat, muss sich gleichzeitig auch für einen Server entscheiden. Und hier präsentiert sich die sprichwörtliche Qual der Wahl. Der Markt der J2EE-Server ist zwar überschaubar, dennoch gibt es viele Bewerber um die Gunst des Kunden. Die hierbei am häufigsten eingekauften J2EE-Server sind beispielsweise BEA WebLogic oder IBM WebSphere – mit Betonung auf „eingekauft“. Denn ebenso häufig wird auch ein Konkurrent dieser beiden Server eingesetzt: JBoss. Ungeachtet der Tatsache, dass der JBoss Applicationserver fantastische Performancewerte besitzt und ungeheuer handlich ist, besitzt er einen entscheidenden Vorteil gegenüber seinen kommerziellen Konkurrenten: Er ist ein Open-Source-Produkt, kostenlos und kann von jedermann unter der Adresse `www.jboss.org` heruntergeladen werden.

Da JBoss ein Open-Source-Produkt ist, das unter der Gnu Lesser General Public License (LGPL) vertrieben wird, darf ein Unternehmen, das JBoss einsetzt, folgende Handlungen durchführen:

1. Die JBoss Software darf als Komponente der eigenen Geschäftsapplikationen in jeglicher Form verwendet werden. Die eigenen oder die eingekauften Applikationen dürfen auf die JBoss Software zugreifen bzw. mit dieser in Bezug gesetzt werden.
2. Die JBoss Software darf unlimitiert kopiert werden, ohne dass irgendwelche Lizenz-Zahlungen zu leisten wären.
3. Die JBoss Software darf unbeschränkt distribuiert werden.
4. Sofern der Bedarf besteht, kann die JBoss Software für die eigenen Zwecke abgeändert werden.
5. Die ggf. abgeänderte JBoss Software darf distribuiert werden, sofern der modifizierte Source Code jedermann unter den Bedingungen der LGPL (Punkte 1-5) zur Verfügung gestellt wird.

J2EE ist primär eine herstellerunabhängige Technologie. Jedoch müssen die J2EE-Applikationen ab einem gewissen Punkt an den jeweils verwendeten Applicationserver angepasst werden. Und genau an diesem Punkt unterscheidet sich dieses J2EE-Buch von ande-

ren. Es wird nicht nur in das Thema J2EE eingeführt, sondern es wird auch jeweils erläutert, wie die eigene J2EE-Applikation anzupassen ist, sodass sie innerhalb eines JBoss Applicationservers lauffähig ist.

Die bei diesem Buch verwendete Version der J2EE ist 1.4. Der eingesetzte JBoss Applicationserver ist 4.x. Im Gegensatz zu vielen seiner kommerziellen Konkurrenten ist JBoss in der 4. Generation 100%ig J2EE 1.4 zertifiziert. Zum Zeitpunkt der Niederschrift dieses Buches wurden mehr als 5.000.000 Downloads gezählt.

Doch wann sollte JBoss in einem Projekt eingesetzt werden? Die JBoss Group gibt hierzu folgende Punkte an. Sofern einer der Punkte zutrifft, wäre dies bereits ein Indikator für das Produkt:

- Es soll eine Open-Source-Software eingesetzt werden.
- Es wird Wert auf einen J2EE 1.4 zertifizierten Server gelegt.
- Die entwickelten Applikationen sollen geclustert werden. Clustering ist ein Feature, das bei kommerziellen Applicationservern teilweise mit einem rund 80%igen Aufschlag der Lizenzkosten zu Buche schlägt.
- Ein Applicationserver verfügt über eine hohe Anzahl administrativer Programmlogik. Die Ausführung dieser administrativen Programmlogik erfordert Rechenzeit. Und diese Rechenzeit fehlt der eigenen Applikation. So haben Benchmark-Tests ergeben, dass JBoss wesentlich performanter ist als andere Applicationserver.
- Der Applicationserver soll architektonisch so implementiert sein, dass einzelne Dienste leicht zu ergänzen bzw. zu administrieren sind – z. B. mittels JMX.
- Die Applikationen sollen quasi Just-In-Time auf dem Applicationserver installiert werden können, ohne dass der Server angehalten werden muss. Man spricht von sog. Hot-Deployment.
- Die selbst entwickelten Applikationen sollen dem aspektorientierten Paradigma (AOP) folgen.

Doch JBoss kann noch mehr. Genau wie seine kommerziellen Konkurrenten bietet die JBoss Group Erweiterungsprodukte für den Applicationserver an. Dies sind u. a.

- Java Business Process Management (jBPM): Hierbei handelt es sich um eine Servererweiterung, mit der sog. Serviceorientierte Architekturen (SOA) implementiert werden können.
- JBoss Portal: Das JBoss-Portal ist ein Open-Source-basiertes Content Management System (CMS).

Diese Erweiterungsprodukte sind jedoch nicht Bestandteil dieses Buches. Vielmehr wird auf den Applicationserver eingegangen.

Abbildung 1.1 Die Willkommensseite der JBoss Group

1.2 Grundlage von J2EE: die 4-Tier-Architektur

Webbasierte Anwendungen sind Applikationen, bei denen der Anwender über einen Standard-Browser (wie z. B. den Internet Explorer von Microsoft) mit einem Softwaresystem in Kontakt tritt. Sofern ein Unternehmen heutzutage seinen Kunden einen internetbasierten Zugang zum betrieblichen Informationssystem anbieten möchte, stehen dem realisierenden IT-Projektteam dazu eine Vielzahl von verschiedenen Technologien zur Verfügung. Doch vor der Auswahl einer konkreten Technologie ist die Entscheidung über die grundsätzliche Architektur einer webbasierten Anwendung zu treffen. Dazu werden Schichten (oder Ebenen) bestehend aus Systemkomponenten in Form von sog. Schichten-Architekturen betrachtet.

1.2.1 2-Tier-Architekturen

Die ersten Software-Systeme waren Anwendungen auf zentralen Großrechnern (Mainframes). Ende der 80er, Anfang der 90er waren die meisten betrieblichen Informationssysteme getreu einer 2-Tier-Architektur (tier (engl.) = Schicht, Ebene) aufgebaut. Ein bekannte-

re Umsetzung dieser Architektur ist das Client/Server-Modell. Bei der ursprünglichen 2-Tier-Architektur wird die Gesamtapplikation (also Ausführungslogik und Datenhaltung) auf zwei Ebenen aufgeteilt: Auf Ebene 1 befindet sich die Präsentationslogik zusammen mit der Applikationslogik, die auch als Geschäfts(prozess)logik bezeichnet wird. Die Applikation auf Ebene 1 (meist der Client-PC) kommuniziert über ein Netzwerk mit einem Datenbankserver, der auf Ebene 2 angesiedelt ist. Auf dem Datenbankserver werden die kritischen Unternehmensdaten gespeichert, die innerhalb der Geschäftsprozesse geschrieben bzw. gelesen werden. Der Zugriff auf diese Daten erfolgt z. B. bei relationalen Datenbanken über die Abfragesprache SQL (Structured Query Language).

Der architektonische Nachteil der beschriebenen 2-Tier-Architektur ist, dass die Geschäftslogik nicht klar von der Präsentationslogik getrennt ist. Wenn die Geschäftslogik einer Applikation beispielsweise in einer anderen Applikation wieder verwendet werden sollte, dann muss die Applikationslogik innerhalb der zweiten Applikation zwangsweise neu implementiert werden.

Für die Trennung von Präsentationslogik und Geschäftslogik lassen sich weitere Gründe identifizieren, die aus folgenden Nachteilen einer 2-Tier-Architektur resultieren:

Die Datenbankintegrität kann leicht gestört werden: Da jedes Clientprogramm über seine eigene Geschäftslogik verfügt, kann ein Fehler in einem Clientprogramm die Integrität anderer Clientprogramme gefährden.

Die Administration in einem großen Unternehmen ist schwierig zu handhaben: Bei dieser Architektur wird die Geschäftslogik auf den Clients ausgelagert, welche meist von einer zentralen Administration ausgeliefert werden. Wird ein Geschäftsprozess geändert, der auf 10 verschiedenen Clientprogrammen separat implementiert worden ist, müssen alle 10 Clientprogramme diesbezüglich abgeändert werden. Der Änderungsaufwand innerhalb der Wartungs- und Weiterentwicklungsphase der Software ist unverhältnismäßig hoch.

Die Gefahr von Sicherheitslücken wächst: Die auf einem Client distribuierten Programme beinhalten die Geschäftslogik, die z.B. durch Dekompilieren analysiert und modifiziert werden kann.

Die Skalierbarkeit ist limitiert: Jede Applikation muss normalerweise Kontakt zur Datenbank aufnehmen. Hierbei ist jedoch die Anzahl der Datenbankverbindungen durch die natürlichen Grenzen der Server- und Netzwerk-Hardware meist limitiert. Dies hätte zur Folge, dass nicht alle Clients Zugriff auf die Daten hätten.

Der Präsentationstyp ist limitiert: Da die Geschäftslogik bei der 2-Tier-Architektur fest mit der Präsentationslogik verknüpft ist, ist es häufig schwierig, die gleiche Implementation der Geschäftslogik für andere Präsentationstypen (wie z. B. Browser oder PDA) wieder zu verwenden.

1.2.2 3-Tier-Architekturen

Das 2-Tier-Architekturmodell hat für viele Applikationstypen durchaus seine Daseinsberechtigung. Einige Autoren zählen hierzu beispielsweise Applikationen, die nur Daten vi-

sualisieren und nicht manipulieren lassen. Jedoch ist bei der Einführung webbasierter Applikationen die Trennung von Präsentations- und Geschäftslogik zwingend.

Bei einer 3-Tier-Architektur wird die Geschäftslogik auf einem eigenen Server implementiert, um von den verschiedenen Clienttypen verwendet zu werden. Die Schicht, auf der die Geschäftslogikebene implementiert ist, wird auch als sog. Middle-Tier bezeichnet. Mit der Einführung dieser separaten Geschäftslogikebene ist auch die Geburt des Begriffs „Applicationserver„ (Anwendungsserver) verbunden.

Die Einführung einer 3-Tier-Architektur bringt diverse Vorteile mit sich: So wird beispielsweise die Skalierbarkeit erhöht, da kostspielige Verbindungen zur Datenbankebene wieder verwendet werden, unabhängig von der Anzahl und Art der Clients. Zudem erhöht sich die Datenintegrität, wenn die Datenbankzugriffslogik in einer für die Clients zentralen Applikation implementiert ist. Dieses Architekturmodell ist auch die Basis vieler Enterprise Resource Planning (ERP)-Systeme und derjenigen Systeme, die viele Transaktionen handhaben müssen (wie z. B. Tuxedo).

Die 3-Tier-Architektur besitzt jedoch immer noch einige architektonische Nachteile:

Die Komplexität der Gesamtanwendung steigt: Je höher der Abstraktionsgrad, d.h. je mehr Ebenen innerhalb eines Architekturmodells eingefügt werden, desto komplexer ist die Applikationsentwicklung. Generell sind verteilte Applikationen wesentlich aufwändiger zu implementieren als zentrale Applikationen.

Die Portabilität der Gesamtapplikation ist enorm eingeschränkt. Da Hersteller komplette Produkte für 3-Tier-Architekturen anbieten, ist es schwierig von den Vorgaben dieses Herstellers bzgl. der Entwicklung abzuweichen. Die Applikation wird sozusagen für die Plattform eines einzelnen Herstellers maßgeschneidert. Der Wechsel eines Herstellers hat somit zur Folge, dass die gesamte Applikation neu implementiert werden muss und nicht portiert werden kann. Ebenfalls ist es häufig schwierig, Applikationen, die auf dem System des Herstellers X deployt sind, mit Applikationen, die auf dem System des Herstellers Y deployt sind, kommunizieren zu lassen. Webbasierte Anwendungen vom Typ der 3-Tier-Architektur sind somit weder plattform- noch herstellerunabhängig.

1.2.3 4-Tier-Architektur einer webbasierten Anwendung

Im Zuge des Internet-Booms in der Mitte der 90er Jahre, bei dem jedes Unternehmen (selbst „Tante-Emma-Läden“) unbedingt seine Waren über das Internet verkaufen wollte, wurde eine weitere Schicht eingeführt. Die so entstandene 4-Tier-Architektur gilt seitdem als sog. State-of-the-art-Architektur webbasierter Applikationen. Sie kann wie in Abbildung 1.1 dargestellt visualisiert werden.

Hierbei teilen sich die 4 Ebenen wie folgt auf:

- Ebene I ist die *Visualisierungsebene (Web-Client)*. Sie visualisiert die Benutzeroberfläche. Hier werden diejenigen Daten mittels eines Browsers visualisiert, die auf der Ebene II präsentationstechnisch zusammengestellt werden. Gleichzeitig nimmt der Browser die Eingaben des Anwenders entgegen und sendet diese zu Verarbeitungszwecken an die Applikationen auf Ebene II.

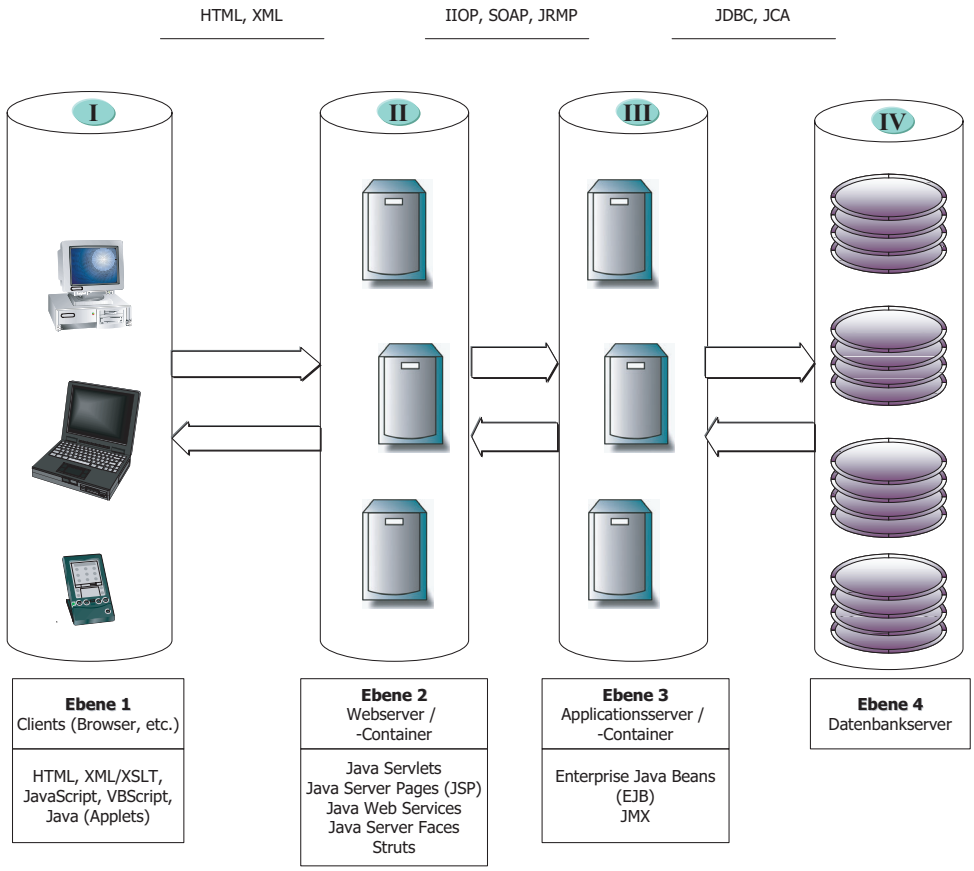

Abbildung 1.2 Überblick über die 4-Tier-Architektur und die J2EE-spezifischen Komponentenmodelle

- Ebene II ist die *Präsentationsebene (Web-Server)*. Hier werden die Daten applikationstechnisch ausgewertet, die von der Ebene I – speziell dem Browser – an diese Ebene gesendet werden. Die eingehenden Daten werden analysiert und ausgewertet, wobei die involvierte Geschäftsprozesslogik auf Ebene III angesprochen wird. Nach der Auswertung werden die Daten an die Ebene I zurückgesendet.
- Ebene III ist die *Geschäftslogikebene (Anwendungsserver)*. Hier ist die Geschäftslogik implementiert. Die Daten, die für die Geschäftsprozesse benötigt werden, werden von der Ebene IV bezogen.
- Ebene IV ist die *Datenbankebene (Datenbankserver)*. Hier werden die Daten zur Verfügung gestellt, die für die Geschäftsprozesse benötigt werden, bzw. die in den Geschäftsprozessen produziert werden.

Dem Nachteil, dass mit der wiederum höheren Anzahl der verwendeten Schichten auch die Komplexität der Gesamtapplikation zunimmt, stehen – neben den anderen Vorteilen der 3-Tier-Architektur – zudem der Vorteil der möglichen Plattform- und Herstellerunabhängig-

keit gegenüber. Dies soll anhand einer konkreten Implementierungstechnologie im abschließenden Abschnitt erläutert werden.

1.2.4 Webbasierte Anwendungen mit J2EE

Auf dem heutigen Markt für webbasierte Anwendungsentwicklung lassen sich zwei Technologie-Plattformen identifizieren, die die zuvor beschriebene 4-Tier-Architektur unterstützen. Dies sind zum einen Microsofts .NET-Plattform und zum anderen Sun Microsystems J2EE-Plattform. Bei größeren Unternehmen, wie z. B. Banken, hat in den letzten Jahren vor allem die letztgenannte Technologie, J2EE (Java 2 Platform, Enterprise Edition) besondere Aufmerksamkeit und einen hohen Verbreitungsgrad gefunden.

Bei J2EE handelt es sich zum einen um eine Plattform, mit der plattformunabhängige Programme ausgeführt werden können, die in der Programmiersprache Java geschrieben wurden. Zum anderen ist es eine Ansammlung sog. Komponentenmodelle, die die Arbeit eines Softwareentwicklers unterstützen. Komponentenmodelle sind Programmbibliotheken, die bei der Entwicklung eigener Applikationen verwendet werden können, um den eigenen Entwicklungsaufwand zu reduzieren und den erzeugten Programmcode an einen Standard anzulehnen. Durch die konsequente Anlehnung der eigenen Programme an die in J2EE enthaltenen Komponentenmodelle wird neben der Plattformunabhängigkeit auch das Kriterium der Herstellerunabhängigkeit erreicht.

Die J2EE-Plattform bietet dabei eine spezielle Unterstützung für die Applikationsentwicklung einer webbasierten 4-Tier-Architektur an, denn die Komponentenmodelle der J2EE-Plattform orientieren sich am Vorbild dieser Architektur. Wie aus Abbildung 1.2 hervorgeht, wird auf Ebene I ein Browser vorausgesetzt. Ein Standardbrowser unterstützt die Auszeichnungssprache HTML (im Fall des Internet Explorers auch XML). HTML (Hypertext Markup Language) ist eine spezielle Sprache zur Auszeichnung von Dokumenten, die letztendlich vom Browser visualisiert werden. Da HTML-Dokumente statisch sind, werden sie häufig mit ein wenig Präsentationslogik angereichert, die in einer vom Browser verstandenen Skriptsprache geschrieben ist. Dies ist primär JavaScript. Alternativ kann dies auch VBScript sein.

Auf Ebene II wird ein Webserver eingesetzt. Da die Visualisierung auf Ebene I durch einen separaten Client erfolgt, muss auf Ebene II „lediglich“ die Präsentationslogik implementiert werden. Damit das Entwicklungsteam nicht mehr sämtliche Grundfunktionalitäten eines Webservers implementieren muss, wird der Webserver bei J2EE von einem beliebigen Hersteller bezogen, der ein J2EE-kompatibles Produkt anbietet. Häufig wird jedoch statt eines kommerziellen Produktes der kostenlose Open Source Webserver Tomcat der Apache Foundation eingesetzt. Damit selbst geschriebene Programme in den Webserver integriert werden können, muss die eigene Präsentationslogik die Standards beachten, die die Komponentenmodelle Java Servlets bzw. JavaServer Pages festlegen. Hierdurch wird Herstellerunabhängigkeit auf Ebene II erzielt. Da auf Ebene II wiederum eine Trennung von Logik und Daten erfolgen kann, haben sich sog. Frameworks entwickelt, mit denen die Web-Applikationen der Ebene II wartungsfreundlicher entwickeln lassen. Hierzu

zählen u. a. die in J2EE enthaltenen JavaServer Faces (JSF) oder das von Apache publizierte Struts.

Die Geschäftslogik wird auf Ebene III implementiert. Hierzu bedarf es grundlegender administrativer Programmlogik (wie z. B. Handhabung konkurrierender Zugriffe). Genau wie auf Ebene II hat sich auf Ebene III ein Markt für fertige Produkte herauskristallisiert, der diese grundlegende Programmlogik zur Verfügung stellt. Die Server werden als Applicationserver bezeichnet und bieten eine Umgebung für die selbst geschriebenen Applikationen an. Bekannteste kommerzielle Produkte sind der IBM WebSphere oder der BEA Applicationserver. Ein ebenso verbreitetes, aber kostenloses Produkt ist der JBoss Applicationserver, dem dieses Buch gewidmet ist. Damit die eigenen Applikationen innerhalb dieser Server deployt werden können, bedarf es wiederum eines herstellerunabhängigen Komponentenmodells. Für die Ebene III ist dies das Modell der Enterprise JavaBeans (EJBs), dem wohl bekanntesten Komponentenmodell der J2EE-Plattform.

Damit die Applikationen auf Ebene II mit denen auf Ebene III kommunizieren können, wird Middleware benötigt. Unter Middleware versteht man eine Dienstleistungsschicht zwischen Applikation und Betriebssystem. Die von J2EE unterstützten Middlewareprodukte sind CORBA, RMI und Web Services. Middleware impliziert ein Protokoll: CORBA nutzt das Protokoll IIOP, RMI nutzt die Protokolle JRMP und IIOP und Web Services verwenden SOAP.

Da auf Ebene IV keine Eigenimplementation erfolgt, sondern ein Datenbankserver eines favorisierten Herstellers eingesetzt wird, gibt es von Seiten Sun Microsystems auch keine Unterstützung für die Implementation eines Datenbankservers auf der Ebene IV. Um jedoch mit dem favorisierten Datenbankserver kommunizieren zu können, bietet J2EE ein herstellerunabhängiges Komponentenmodell namens JDBC an. Jeder Datenbankserver, für den es einen JDBC-kompatiblen Treiber gibt, kann somit an die eigene Applikation ohne große Aufwände angedockt werden. Sofern es sich nicht um eine relationale Datenbank auf Ebene IV handelt (z. B. um ein ERP-System), kann auf dieses System mittels Java Connector Architecture (JCA) zugegriffen werden.

1.3 Vorgehensweise im Buch

Das Buch ist mit seinen 26 Themenkapiteln sehr umfangreich. Wichtig war es den Autoren, ein möglichst umfassendes Wissen über die beiden Themen J2EE und JBoss zu liefern. Im Verlauf des Buches wird eine Gesamtapplikation erstellt, die der webbasierten 4-Tier-Architektur entspricht.

Das Buch beinhaltet folgende Kapitel:

1. In der Einleitung (also in diesem Kapitel) wurde die 4-Tier-Architektur eingeführt, an der sich die weiteren Applikationsentwicklungen orientieren. Sie ist Grundlage für den Sinn und Aufbau der nachfolgenden Applikationsteile.
2. Bevor die einzelnen Applikationsteile erstellt werden, wird in Kapitel 2 vorgeführt, wie JBoss zu „installieren“ bzw. einzurichten ist. Ebenfalls wird die JBoss IDE präsentiert,

mit deren Hilfe JBoss-spezifische J2EE-Entwicklung mit Eclipse durchgeführt werden kann.

3. Sofern nicht auf relationale Datenbanken zugegriffen wird, kann mit Hilfe des Komponentenmodells JCA einheitlich auf andersartige Ressourcen zugegriffen werden.
4. Grundlage der 4-Tier-Architektur ist der Zugriff auf Datenbanken. Hierzu stellt J2EE das Komponentenmodell JDBC zur Verfügung. Kapitel 4 zeigt, wie mit Hilfe von Java auf relationale Datenbanken zugegriffen werden kann.
5. JBoss ist ein Applicationserver, der verschiedene Services startet. Ein Entwicklungsteam kann jedoch mit Hilfe des Kompontenmodells JMX ebenfalls eigene Services schreiben.
6. Es muss nicht immer Middleware zum Einsatz kommen, wenn zwei Applikationen miteinander kommunizieren. Falls nicht, dann bietet Java die Möglichkeit an, über Sockets mit anderen Applikationen zu kommunizieren.
7. Falls keine Sockets verwendet werden, sondern Middleware, dann steht in J2EE das Komponentenmodell RMI zur Verfügung, das sowohl JRMP- als auch IIOP-sprechende Client/Server-Applikationen erstellen lässt. Letzteres sind CORBA-Applikationen.
8. Im 8. Kapitel wird ein genereller Überblick über EJBs gegeben.
9. Im 9. Kapitel werden die Session Beans als eine Form der EJBs vorgestellt.
10. Im 10. Kapitel werden die Entity Beans in Form der CMPs dargestellt.
11. Im 11. Kapitel werden die Entity Beans in Form der BMPs beschrieben.
12. Neu in J2EE 1.4 ist die Einführung der Timer. Diese können in jedem Typ der EJBs eingesetzt werden.
13. Nachrichtengesteuerte Applikationen können bei Verwendung des Komponentenmodells EJB in Form der MDBs geschrieben werden. Kapitel 13 führt in dieses Thema ein.
14. Geschäftslogik basiert meist auf Transaktionen. Hierzu steht das Komponentenmodell JTA zur Verfügung.
15. Nach dem 14. Kapitel sind alle Grundlagen bekannt, mit denen die Geschäftslogik auf Ebene III mit Hilfe der EJBs implementiert werden kann. Daher wird im 15. Kapitel die Geschäftslogik des Buchbeispiels präsentiert.
16. Interessant ist nun, wie sich die Theorie in die Praxis, d. h. mit JBoss, umsetzen lässt. Dies ist Aufgabe von Kapitel 16.
17. Wenn eine Applikation erst einmal entwickelt wurde, stellt sich die Frage, wie die Last der Clients auf die Ebene III verteilt werden kann. Clustering mit JBoss ist also das Stichwort in Kapitel 17.
18. Jetzt wird es Zeit, die Präsentationslogik der Beispielapplikation zu implementieren. Hierzu bietet J2EE die Komponentenmodelle JSP und Servlets an. Diese sind Grundlage der Themen Struts und JSF.

19. In Praxisprojekten werden kaum mehr rudimentäre JSPs und Servlets geschrieben, wie sie in Kapitel 18 eingeführt wurden. Vielmehr wird das Struts-Framework verwendet.
20. Struts ist jedoch nicht Teil von J2EE wie beispielsweise JSF. Zudem bietet JSF viel mehr Fähigkeiten an, als dies von Struts geboten wird. Daher wird in Kapitel 20 in das Komponentenmodell JSF eingeführt.
21. Verteilte Applikationen können nachrichtenorientiert aufgebaut werden. In Kapitel 13 wurden hierzu die MDBs vorgeführt. Wer jedoch nicht unbedingt MDBs schreiben möchte, der kann auch zum rudimentären JMS greifen. Die Grundlagen von JMS werden in Kapitel 21 thematisiert.
22. JMS besitzt verschiedene Nachrichtentypen. Diese sind Thema des 22. Kapitels.
23. In Kapitel 23 werden Reliability und Performance von JMS Applikationen betrachtet
24. Nach der Erläuterung aller JMS Komponenten wird in Kapitel 24 die in Kapitel 15 erstellte EJB Applikation um eine JMS Schnittstelle erweitert.
25. Schließlich muss erläutert werden, wie der Message Broker unter JBoss konfiguriert werden kann, sodass alle Möglichkeiten von JMS genutzt werden können. Dies ist Aufgabe des 25. und letzten Kapitels.
26. Der Appendix führt in die Thematik der EJB 3.0 ein, die zum Zeitpunkt der Niederschrift dieses Buches zum einen noch nicht den finalen Status erreicht haben und voraussichtlich erst im Jahr 2007 im Zusammenhang mit J2EE 1.5 aktuell sind.

Und jetzt viel Spaß beim Durcharbeiten dieses Buches!

Mit den besten Empfehlungen

Daniel Reiberg, Köln.

Torsten Langner, Aachen.