Apache Geronimo, Teil 1: Einsatzmöglichkeiten und Aufbau unter der Lupe

„Ich warte auf die Wende“

VON KRISTIAN KÖHLER UND CHRISTIAN DEDEK

Einer Legende der Apachen zufolge sang der letzte freie Krieger auf dem Totenlager folgende Worte „O Ha Le a“ – frei übersetzt: „Ich warte“. Manche Indianer, auch außerhalb der Apachen, interpretieren diese Worte als „ich warte auf die Wende“ [1]. Am 5. Januar 2006 endete die Wartezeit auf das Release 1.0 des J2EE 1.4-zertifizierten Application Server der Apache Software Foundation – Apache Geronimo –, dessen Einsatzmöglichkeiten und Aufbau wir unter die Lupe nehmen.

Die initiale Version dieses vollständigen Java EE-Servers mit Apache-Lizenz entstand während eines Zeitraums von circa drei Jahren durch eine konsequente Nutzung bereits vorhandener Softwarekomponenten mit Apache- oder kompatibler BSD-Lizenz. Die kostenfreie Lizenzierung der Sun Java EE-Zertifizierungstests für Projekte der Apache Software Foundation gemäß den Änderungen innerhalb des JCP garantierten den Nutzern des Geronimo bereits seit Oktober 2005 einen zertifizierten Milestone Build, der bis zum Jahreswechsel 2006 um wichtige Facetten für Produktion und Entwicklung bereichert wurde. Eine in Funktion und Optik ansprechende Webkonsole, File-basiertes und remotes Deployment von Komponenten sowie ein erstes Clustering von Webkomponenten entschädigen für das lange Warten auf den Release-Stempel. Mit der sich abzeichnenden Java EE-Zertifizierung stieg auch die Aufmerksamkeit kommerzieller Interessenten innerhalb der Geronimo-Community, wobei IBM seine Beteiligung an der Weiterentwicklung des Geronimo mit der Übernahme des Apache-Unterstützers Gluecode krönte [3]. Erwartungsgemäß steht dem geneigten Nutzer mit der WebSphere Application Server Community Edition ein erster Geronimo-Abkömmling von IBM zur Verfügung [4]. Dank der freizügigen Apache-Lizenzierung dürfte hier eine vielversprechende Zukunft für den neuen Server zu finden sein. Sie erlaubt, im Unterschied zur LGPL, die z.B. der Open-Source-Mitbewerber JBoss verwendet, den Einsatz und Abwandlung der Quellen unter geschlossenen Lizenzen. IBM möchte seinen Geronimo-Abkömmling im Moment als preiswerte Alternative zu klassischen WebSphere-Lizenzen besonders im Bereich kleiner und mittlerer Unternehmen verstanden wissen. Die Community denkt über diesen Horizont bereits hinaus und nutzt die modulare Architektur des Servers, um eine rasche Weiterentwicklung des Servers voranzutreiben [5]. Beispiele hierfür sind die an das Projekt übergebenen Quellen zum Aufbau einer eigenen CORBA-Infrastruktur durch die Firmen Trifork und IONA, welche in der Vergangenheit (IONA J2EE 1.3) bzw. Gegenwart (Trifork J2EE 1.4) im Java EE-Server-Markt mit eigenen Produkten auftraten bzw. -treten. Weitere für die Positionierung im Java EE-Server-Markt wichtige Verbesserungen vollziehen sich im Bereich Clustering und Sicherheit von Java EE-Anwendungen [6]. Noch im ersten Quartal 2006 ist daher ein Minor Release mit den Zielen

- verbesserter Support für Java EE Deployment mit Ant oder Maven,
- internes Build Management mit Maven2 und
- echte Redundanz von Webkomponenten mithilfe des Web Application Distribution Infrastructure-(WADI-)Projekts [7]

geplant. Eine Prämisse in der Weiterentwicklung stellt die zeitnahe Behebung von Sicherheits- und Stabilitätsmängeln dar, welche bisher unentdeckt in den Tiefen des ersten Release schlummern könnten. Die Wunschliste eines Major Release 2 steht bisher ganz im Zeichen der Unterstützung von Java EE 5. Dazu sollen sich ein echtes Clustering aller Java EE-Komponenten, ein unabhängiger ORB und die Nutzung von Java 5-Features wie Annotations und Generics innerhalb des Serverkerns gesellen.

Anschleichen – Installation

Der erste Kontakt mit dem Server gestaltet sich, wie man es bei Java-Erzeugnissen der Apache Software Foundation gewöhnt ist, relativ unproblematisch. Zum Betrieb des Geronimo als vollständig in Java implementierte Software ist lediglich die lokale Verfügbarkeit einer Sun JRE 1.4 Voraussetzung.

Apache Geronimo – die Serie

- **Teil 1: Einsatz und Aufbau**
- Teil 2: Erstellen eigener Geronimo-Distributionen und Servererweiterungen

Lesetipp

Bitte beachten Sie auch das im Mai 2006 im Software & Support Verlag erscheinende Buch der beiden Artikel-Autoren: **Geronimo – Apache Geronimo im Einsatz.**

Abb. 1: Geronimo Installer

Durch die Kopplung an den Sun Java 1.4 ORB ist der Einsatz einer beliebigen J2SE 1.4-Laufzeit-Plattform zurzeit nicht empfehlenswert. Leider entfällt aus diesem Grund auch die Möglichkeit des Einsatzes von Java 5. Die vom Server genutzte JVM wird über die Umgebungsvariablen *JRE_HOME* bzw. *JAVA_HOME* gesteuert. Ein JDK ist nur bei Nutzung des Debug-Modus nötig.

Für den Bezug der Software stehen unter geronimo.apache.org/downloads.html zwei plattformunabhängige Distributionen, welche gängigen Archivierungswegen verschiedener Plattformen folgend verpackt wurden (*tar.gz/zip*), als Download zur Wahl. Die zwei Distributionen erlauben eine Wahl des Webcontainers zwischen Jetty und Apaches hauseigenem Container Tomcat. Von einer ursprünglichen Präferenz der Entwicklergemeinde für Jetty ist hier nichts mehr zu bemerken. Die Archive müssen ausgepackt und mit den entsprechenden Zugriffsrechten für den ausführenden User des Servers konfiguriert werden. Zum Zeitpunkt des Erscheinens dieses Artikels sollte zusätzlich ein Java-basiertes Installationsprogramm zum Download bereitstehen. Dieses wird als selbstaktivierendes *.jar*-Archiv ausgeliefert, das in grafischen Oberflächen per Mausklick oder auf Shell-Ebene mit der Zeile *java -jar geronimo-installer.jar* gestartet werden kann (Abb. 1).

Nach dem Entpacken (bzw. der Installation) in das Verzeichnis *GERONIMO_HOME* stehen im Verzeichnis *GERONIMO_HOME/bin* *.*bat* und *.*sh* Skripte zum Start und Stopp des Servers sowie zum Deployment bereit. Durch Anpassung dieser Shell-Skripte lassen sich auch Speicherbedarf, Profiling oder Verhalten der Garbage Collection auf Ebene der JVM konfigurieren. Alternativ zu den Skripten besteht zusätzlich die Möglichkeit, die selbstaktivierenden *.jar*-Archive *server.jar, shutdown.jar* und *deployer.jar* direkt zu nutzen.

Beim Starten des Servers kann man Art und Umfang der Konsolennutzung des Serverprozesses über Kommandozeilenoptionen bestimmen (*Geronimo Help* listet alle Optionen auf). Das Logging des Servers basiert auf Log4J und kann mit der Datei *GERONIMO_HOME/var/logserver-log4j.properties* konfiguriert werden. Die Standardeinstellungen schreiben ausführliche Logging-Informationen in die Datei *GERONIMO_HOME/var/log/geronimo.log* sowie Meldungen ab dem Log-Level *INFO* auf die Serverkonsole.

Der Server besitzt bereits in der initialen Installation ein Security-Management. Die Authentifizierung basiert dabei auf so genannten Security Realms. Deshalb müssen z.B. für den Shut-down ein gültiger Benutzername und ein Passwort übergeben werden. Die administrativen Komponenten des Geronimo werden über einen gemeinsamen Realm (Console Realm) verwaltet, der sich aus Property-Dateien im Verzeichnis *GERONIMO_HOME/var/security* zusammensetzt. Gültige Benutzernamen und Passworte lassen sich in der Datei *users.properties* anlegen. Eine Zuordnung eines Benutzers zu einer Gruppe erfolgt durch Eintrag des Benutzers in die *groups.properties*. Ein Benutzer *system* mit dem Passwort *manager* und der Zuordnung zur Gruppe *admin*, welche Rechte zur Ausführung sämtlicher administrativer Komponenten besitzt, ist die Default-Einstellung der Distribution. Das Selbstinstallationsprogramm fragt während des Installationsvorgangs Benutzernamen und Passwort des einzurichtenden administrativen Benutzers ab. Viele administrative Komponenten lassen sich über eine Weboberfläche einfach ansteuern. Diese Portlet-basierte Konsole kann nach der Installation und einem fehlerfreien Start des Servers unter *http://localhost:8080/console* geöffnet werden (Abb. 2). Zu den Tätigkeiten, die über die Konsole durchgeführt werden können, gehören unter anderem:

- Stopp des Servers
- Auswertung und Konfiguration von Logfiles
- Monitoring der Performance von Serverkomponenten
- Verwalten von Benutzern und Gruppen
- Anlagen und Überwachung JCA Connection Pools
- Installation und Kontrolle von Java EE- und Serverkomponenten

Ein besonderer Leckerbissen für erfahrene Java EE-Serverbenutzer ist die einfache Konfigurationsmöglichkeit der vom Server benutzen Ports. In der Datei *GERONIMO_HOME/var/config/config.xml* können die folgenden Dienste und Ports festgelegt werden:

- RMIRegistry: 1099
- EJB Remoting: 4201
- JAAS Remoting: 4242
- CORBA Naming: 1050
- Active MQ: 61616
- Derby Database: 1527
- Webcontainer: 8080
- Apache Connecto: 8009
- Webcontainer SSL: 8443
- Mailserver: 25

Die genaue Bedeutung der Konfigurationsdatei und ihre Funktionsweise im Zusammenspiel mit der Architektur des Servers werden weiter unten behandelt.

Ausschwärmen – Deployment

Die Verteilung und Installation von Java EE-Applikationen und Komponenten in

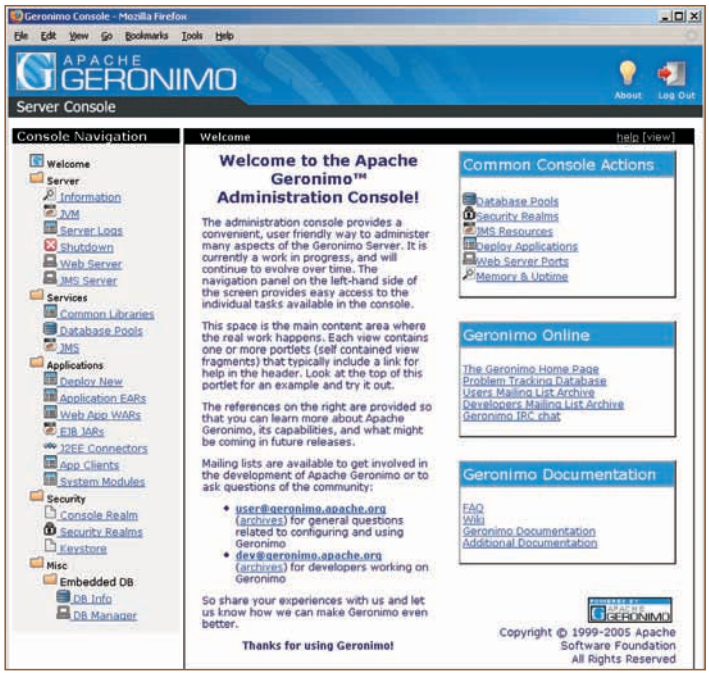

Abb. 2: Geronimo Console

den Server stützt sich grundsätzlich auf das J2EE Deployment Management API (JSR 88). Der entsprechende Treiber liegt unter *GERONIMO_HOME/repository/geronimo/jars/geronimo-deploy-jsr88-1.0.jar* der Distribution bei. Leider unterstützt Geronimo im Moment nicht die so genannten Konfigurationsaspekte, mit deren Hilfe Deployment-Deskriptoren über das API zur Laufzeit erzeugt werden können. Im Standardumfang werden drei unterschiedliche Werkzeuge angeboten, die auf dieses API aufsetzen:

- Administrationskonsole
- dateibasiertes Hot Deployment
- Kommandozeilenwerkzeug

Das Deployment ist nicht auf diese Werkzeuge beschränkt. Prinzipiell kann jedes JSR 88-kompatible Tool eingesetzt werden. Mit dem Geronimo Development Tools-Unterprojekt [8] steht z.B. eine Eclipse Web Tools Project-(WTP-)basierte Lösung zur Verfügung.

Das bei der Installation genannte Security-Management greift ebenfalls beim Aufruf über das JSR 88 API, sodass die Angabe valider Benutzerinformationen notwendig ist. Hierzu kann der Default-Benutzer verwendet werden. Ein zentrales, aus dem JSR 88 übernommenes Konzept ist die Nutzung eines Deployment-Plans. Hierbei handelt es sich im Fall von Geronimo um eine XML-Datei, welche die Deployment-Konfiguration einer Komponente beschreibt. Sie ist vergleichbar mit einem containerspezifischen Deployment-Deskriptor aus der Java EE-Welt. Jedem Java EE-Archiv, das Enterprise-Services des Containers nutzen möchte, ist zur Durchführung eines Deployments eine solche Datei zuzuordnen. Nach Bereitstellung des Plans gestaltet sich das Deployment des Archivs recht einfach. Das mitgelieferte Kommandozeilenwerkzeug lässt sich folgendermaßen aufrufen:

```
./deploy.bat deploy application.ear application-plan.xml
```

Alternativ kann das Deployment über die Administrationskonsole unter dem Menüeintrag Deploy New durchgeführt werden (Abb. 3).

Java EE-Anwendungen, welche keine speziellen Anforderungen an die Laufzeitumgebung stellen, können ohne Plan installiert werden. In diesem Fall wird die Anwendung in einer Default-Umgebung ausgeführt. Neben der Möglichkeit, Deployment Plans erst zum Zeitpunkt der Installation anzugeben, besteht die Option, die Datei in das entsprechende Java EE-Archiv aufzunehmen. Ähnlich zu Java EE-Deployment-Deskriptoren stehen zur Ablage der Konfigurationsdateien innerhalb des

Abb. 3: Deployment über die Administrationskonsole

Archivs verschiedene Varianten zur Verfügung. Durch die Aufnahme des Plans in das Archiv kann ein Hot Deployment durch Kopieren in das *GERONIMO_HOME/deploy*-Verzeichnis stattfinden.

Die größte Hürde zur Inbetriebnahme einer bestehenden Java EE-Anwendung ist im Moment die Erstellung eines entsprechenden Deployment-Plans. Eine automatisierte Konvertierung containerspezifischer Deployment-Deskriptoren wäre an dieser Stelle wünschenswert.

Attacke – Architektur

Ziel des Geronimo-Projektes war von Anfang an, einen Java EE-konformen Application Server zur Verfügung zu stellen. Basis des Projektes sollte ein „Best of Breed"-Ansatz sein. In der aktuellen Version werden wahlweise Tomcat oder Jetty als Webcontainer angeboten. Die JMS-Unterstützung kommt von ActiveMQ, einem performanten JMS Message Broker. Statt einer Neuimplementierung des kompletten Java EE Stacks sollten bestehende Java EE-konforme Komponenten als Serverbestandteile integriert werden. Dieser Integrationscharakter spiegelt sich in der Architektur des Servers wider. Die einzelnen Serverbestandteile sollten nicht starr und statisch, sondern relativ lose miteinander verbunden sein.

Um diese Anforderung abdecken zu können, wurde die Geronimo Bean oder GBean-Architektur geschaffen, bei der es sich um ein Dependency-Injection-basiertes Konfigurations- und Managementsystem handelt. Trotz des Ziels, einen Java EE-konformen Application Server zu erstellen, ist die GBean-Architektur nicht auf die Java EE-Welt beschränkt. Der Geronimo-Kern kann vielmehr als Server-Framework bezeichnet werden, das auch als Basis für eigene Projekte eingesetzt werden kann.

Zentraler Bestandteil der Architektur ist der Geronimo-Kernel, an dem sämtliche Komponenten über Geronimo Beans, kurz GBeans, angemeldet werden (Abb. 4). Entweder handelt es sich bei der Komponente selbst um ein GBean oder die Komponente bietet ein GBean an, über das sie an den Kernel angemeldet wird. Dies bietet den Vorteil, dass alle angemeldeten Komponenten über eine einheitliche Management- und Konfigurationsmöglichkeit verfügen sowie einem einheitlichen Lebenszyklus unterworfen sind. Innerhalb des Servers werden sämtliche installierten Komponenten, seien es Serverbestandteile wie der Transaktionsmanager oder Java EE-konforme Anwendungen wie Webanwendungen, als GBeans angemeldet.

Mehrere GBeans können zu Konfigurationen zusammengefasst und über den Geronimo-Kernel einzeln verwaltet werden. Jede Konfiguration besitzt hierbei eine eindeutige ID, über die sie angesprochen werden kann. Konfigurationen können über drei unterschiedliche Wege in den Server gelangen:

- zum Übersetzungszeitpunkt des Servers,
- über einen Module Builder/Deployer oder
- programmtechnisch.

Das Maven-basierte Build-System des Geronimo-Servers erzeugt zum Übersetzungszeitpunkt aus Deployment Plans einzelne Konfigurationen. Diese werden in Configuration Archives (CAR), einem Geronimo-eigenen Format, abgelegt und in einem zweiten Schritt zum eigentlichen Server zusammengestellt. Die Deployment Plans enthalten als XML serialisierte, gruppierte und vorkonfigurierte GBeans. Geronimo besteht auf diese Weise in der Auslieferung bereits aus 22 Konfigurationen, mit Serverbestandteilen sowie Beispielanwendungen. Alternativ hierzu führt jedes Deployment einer Java EE-Anwendung ebenfalls zu einer neuen Konfiguration. Ein Module Builder legt hierbei für die Anwendung entsprechende GBeans an. Diese werden am Kernel registriert und ebenfalls als Konfiguration von ihm verwaltet.

Die dritte Variante der programmtechnischen Anlage einer Konfiguration ist prinzipiell möglich, wird allerdings im Normalfall nicht praktiziert. Da der Kernel nicht zwischen Serverbestandteil oder Anwendung unterscheidet, es handelt sich in beiden Fällen um GBeans, ist es ebenfalls möglich, Serverkomponenten, wie Connection Pools, mit einer Anwendung zu gruppieren und als eigenständige Konfiguration zu verwalten. Eine Anwendung kann auf diese Weise direkt mit der entsprechenden Datenbankanbindung deployt werden. Diese hängt nicht von zwangsweise im Vorfeld konfigurierten Serverbestandteilen ab.

Innerhalb des Servers werden alle zur Verfügung stehenden Konfigurationen in einem *ConfigStore* gespeichert. Dieser kann neben den unterschiedlichen Konfigurationen auch eine Konfiguration in mehreren Versionen enthalten. Enthaltene Konfigurationen müssen nicht zwangsläufig ausgeführt werden. Die Standard-*ConfigStore*-Implementierung speichert ihre Daten im *config-store*-Verzeichnis unterhalb der Serverinstallation. Jede Konfiguration wird hierbei in einem separaten Unterverzeichnis abgelegt. Der Inhalt dieser Verzeichnisse entspricht dem eines entpackten CAR-Archivs. In der Datei *index.properties* wird zusätzlich Buch darüber geführt, welche Konfiguration in welchem Unterverzeichnis abgelegt wurde. Die zur Verfügung stehenden Verwaltungsoperationen für Konfigurationen entsprechen den im Java EE Deployment Management API angegebenen Methoden:

- *distribute*: Lädt eine Konfiguration in den *ConfigStore*. Sie wird nicht automatisch gestartet.
- *start/stopp:* Startet bzw. stoppt eine Konfiguration aus dem *ConfigStore*.
- *deploy/undeploy/redeploy*: Lädt bzw. löscht eine Konfiguration im *Config-*

Store und startet bzw. stoppt diese automatisch.

Diese Kommandos können mit den mitgelieferten Werkzeugen abgesetzt werden. Folgendes hält die Willkommensseite des Tomcat Bundle im laufenden Betrieb an:

```
./deploy.bat --user system --password manager stop
                    geronimo/welcome-tomcat/1.0/car
```

Das verwendete Deploy-Tool befindet sich im *bin*-Verzeichnis des Servers. Im vorherigen Beispiel wird der Server angewiesen, die Konfiguration mit der Config-ID *geronimo/welcome-tomcat/1.0/car* zu stoppen. Mit dem Kommando *start* lässt diese sich entsprechend wieder starten. Die Information, welche Konfigurationen ausgeführt werden sollen, bleibt über einen Neustart des Servers hinweg erhalten.

In der Standard-*ConfigStore*-Implementierung führt ein Redeployment nicht zum Austausch der vorhandenen Konfiguration, sondern zu einer neuen Konfiguration im *ConfigStore* und einer entsprechenden Anpassung in der *index.properties*-Datei. Dies hat zur Folge, dass bei vielen Redeployments das Dateisystem unnötig gefüllt wird. Allerdings lassen sich im Gegenzug ältere Versionen recht einfach wieder herstellen. Eine Option zum Löschen vorhandener Konfigurationen wäre hilfreich.

Sammeln – Konfiguration

Zum Startzeitpunkt des Servers werden alle zu startenden Konfigurationen ermittelt und entsprechend geladen. Hierzu dient die Datei *config.xml* im *GERONIMO_HOME/var/config*-Verzeichnis. Sie enthält neben der Information, welche Konfigurationen gestartet werden sollen, auch Angaben zum Überschreiben einzelner Konfigurationsattribute. Folgendes Element innerhalb der Datei setzt z.B. den Standardport des integrierten Tomcat auf 8080:

```
...
<configuration name="geronimo/tomcat/1.0/car">
  ...
  <gbean name="TomcatWebConnector">
   <attribute name="host">0.0.0.0</attribute>
   <attribute name="port">8080</attribute>
   <attribute name="redirectPort">8443</attribute>
  </gbean>
  ...
</configuration>
...
```

Dies hat den Vorteil, dass ein *ConfigStore* bzw. die darin enthaltenen Konfigurationen von mehreren Serverinstanzen verwendet werden können und ausschließlich die serverabhängigen Werte, wie Portnummern, über die *config.xml*-Datei angepasst werden können. Angedacht sind zentrale *ConfigStore*-Implementierungen, die ihre Daten beispielsweise in einer Datenbank oder LDAP-System ablegen und diese mehreren Serverinstanzen gleichzeitig zur Verfügung stellen. Auf diese Weise ließen sich in einer großen Installation viele identische Geronimo-Server starten. Eine zentrale Konfigurationsänderung würde sich auf alle Instanzen auswirken.

Die Anpassung der *config.xml*-Datei kann zum einen bei angehaltenem Geronimo direkt mittels XML-Editor erfolgen, zum anderen besteht die Möglichkeit, über die mitgelieferte Administrationskonsole diese Attribute zur Laufzeit anzupassen. Änderungen, die über die Konsole vorgenommen werden, werden in die *config.xml*-Datei übernommen und stehen somit auch nach einem Neustart des Servers wieder zur Verfügung. Neben der Anpassung bestimmter Konfigurationsattribute und dem Anwendungs-Deployment lassen sich ebenfalls Serverbestandteile wie z.B. Connection Pools anlegen und konfigurieren.

Der Funktionsumfang der Administrationskonsole umfasst in Version 1.0 noch nicht sämtliche Konfigurationsmöglichkeiten des Servers. Somit ist eine Bearbeitung der *config.xml*-Datei an manchen Stellen, wie zum Beispiel die Konfiguration der Thread-Anzahl des Tomcat HTTP Connector, nötig. Leider sind die Deployment Plans, mit denen die Serverkonfiguration erstellt wurde und die alle Konfigurationsoptionen aufzeigen, nicht in der Binary-Distribution enthalten und müssen separat aus dem Source-Download extrahiert werden.

Die Beute des Raubzuges

Die Architektur sowie die Lizenz des Servers machen einen Einsatz als Basis

Abb. 4: Geronimo Kernel mit GBeans

für eigene, kommerzielle Produkte reizvoll. Mit dem zur Verfügung gestelltem Konfigurationsmechanismus wird die Zusammenstellung eines eigenen Servers mit angepasstem Funktionsumfang erleichtert. Sollten bestimmte Bestandteile die eigenen Anforderungen nicht erfüllen, so können diese problemlos ausgetauscht werden. Möchte man etwa sämtliche Konfigurationen im *ConfigStore* in einem verschlüsselten Format ablegen oder eigene Archive-Typen unterstützen, ließen sich diese Erweiterungen in den Server integrieren. Die nächste Folge dieser Artikelreihe behandelt die Erstellung einer eigenen Geronimo-Distribution sowie die Erweiterung des Servers.


Christian Dedek und **Kristian Köhler** beschäftigen sich bei Orientation in Objects mit der Architektur, Entwicklung und Beratung von Java EE-Applikationen. Kontakt: *geronimo@oio.de*.

Links & Literatur

[1] de.wikipedia.org/wiki/Geronimo

[2] de.wikipedia.org/wiki/Chiricahua

[3] www.computerwoche.de/index.cfm?pid=254&pk=556430

[4] www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/was-ce

[5] www.eweek.com/article2/0,1759,1728802,00.asp

[6] www.chariotsolutions.com/slides/geronimo-pad-006.pdf;jsessionid=C6FD91456646DF276279287E42E78B85

[7] incubator.apache.org/wadi/

[8] geronimo.apache.org/devtools.html

[9] Lars Röwekamp, Jens Schumann: Geronimo hat Geburtstag. Ein Jahr Apache Geronimo, in *Java Magazin* 2.2005