Symbiose: OSGi Enterprise trifft Java Enterprise

Wie passen JavaEE und OSGi zusammen?

Frank Pientka

Die Anforderungen an die Entwicklung und den Betrieb von Software wachsen ständig. Modularität und Abhängigkeitsmanagement sind Mittel, um der zunehmenden Komplexität Herr zu werden. Wir zeigen, wie das Beste aus Java EE und OSGi zusammen mit Apache Aries, Karaf und Geronimo verwendet werden kann.

Komplexität und Modularität

Die Möglichkeit, Funktionalität zu abstrahieren und Software durch Module zu strukturieren, spielt seit Langem eine große Rolle. Die Verdopplung der Größe des Softwarecodes alle sieben Jahre [Müller94] lässt sich auch bei Java-Enterprise-Anwendungen beobachten. Java bietet zur Komplexitätsreduktion bisher eine Modularisierung auf Klassen- und Paketebene an. Letztendlich landen zur Laufzeit alle Java-Klassen in einem mehr oder weniger globalen Klassenpfad. Dieser Pfad führt bei großen Installationen durch schwer vermeidbare Versionskonflikte direkt in die sogenannte JAR-Hölle.

Im Jahr 1999 wurde nicht nur die erste Java Enterprise Edition freigegeben, sondern es wurde auch, zunächst für den embedded Bereich, die OSGi Alliance [OSGI] gegründet. Dieses Industriekonsortium hatte zum Ziel, eine Ablaufumgebung und ein Modularisierungskonzept für Java zu entwickeln. Inzwischen wird OSGi darüber hinaus, auch durch die Verwendung für Eclipse-Plug-ins, immer mehr für Java-Anwendungen unterschiedlichster Art eingesetzt. Mit den gewachsenen Anforderungen wurde das OSGi Service Platform Release 4 im März 2010 in der Version 4.2 um Enterprise-Erweiterungen durch die Enterprise Expert Group (EEG) [Ward12, Alves11] ergänzt und in der Core Version 4.3 im April 2011 bzw. in der Core Version 5.0 im Juni 2012 aktualisiert.

Für die Verwendung von OSGi müssen einige Dinge, wie Versionsabhängigkeiten oder die Beschreibung der Deployment-Einheiten, explizit gestaltet werden. Zunächst benötigt man eine OSGi-Laufzeitumgebung, wie Apache Felix, das sich um die Verwaltung und den Lebenszyklus der OSGi-Anwendung kümmert. Für die Verwaltung und das Deployment hat sich Apache Karaf mit seinen GoGo-Befehlen als Alternative zu Spring DM etabliert.

Abb. 1: Enterprise OSGi

Eigene OSGi-Bundles scheinen zwar für die meisten Java-Entwickler bisher zu viel zusätzliche Arbeit gewesen zu sein. Da allerdings immer mehr Hersteller in ihren Produkten OSGi intern einsetzen, liegen inzwischen viele bekannte Bibliotheken für OSGi vor. Werkzeuge, wie bnd, SpringSource Bundlor oder Maven-Plug-ins, helfen weiter, um die für OSGi nötigen Dateien selbst zu erstellen. Trotzdem gab es gerade im Java-Enterprise-Bereich mit einigen Java-Implementierungen und den unterschiedlichsten Klassenladern der Applikationsserver Probleme. Diese werden für die JavaEE-Spezifikationen, JTA, JPA, JDBC, JNDI und JMX mit OSGi-Blueprint gelöst.

In Apache Aries werden noch mehr Spezifikationen unterstützt, als in OSGi-Blueprint vorgesehen. Neben dem gemeinsamen WAR-Format für Webanwendungen gibt es bei OSGi-Blueprint mit EAB (Enterprise Bundle Applications) und WAB (OSGi Web Application Bundles) noch eigene Deployment-Formate, die auf den bestehenden JavaEE-Formaten (WAR, RAR, EAR) aufbauen. Die kleinste Einheit bei OSGi sind Bundles, welche aus einer Manifest-Datei und einer Activator-Klasse bestehen.

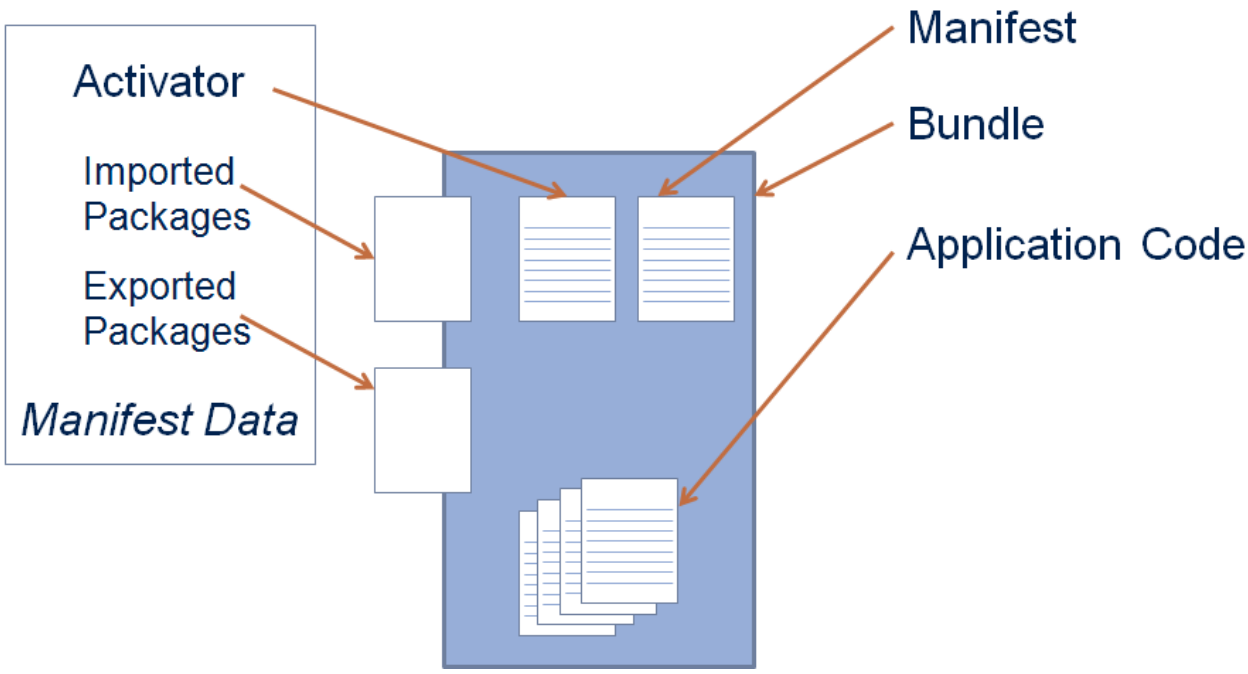

Abb. 2: OSGi-Bundle-Struktur

Als Minimum wird für OSGi-Enterprise eine um einen Blueprint-Container ergänzte OSGi-Laufzeitumgebung benötigt. Neben OSGi-Bundles, können dort als neue Installations-Formate WAB oder EBA installiert werden, um damit die JavaEE-Standards optimal verwenden zu können.

Apache Aries unterstützt die OSGi Core Specification v4.3 und mit dem Configuration Admin Service Blueprint Container, Web Applications, JNDI Services sowie JPA Services und mit dem JMX-Management-Modell Teile der Enterprise Specification v4.2. Dadurch können neben Java-Archiven (EAR, WAR und RAR) auch Enterprise Bundle Applications (EBA), OSGi Web Application Bundles (WAB) und OSGi-Bundles (JAR) installiert und verwaltet werden.

Mit dem OSGi-Standard RFC-0152 [OSGI]) können mehrere zusammengehörige Bundles und ihre Abhängigkeiten zu einem Subsystemen zusammengefaßt werden. Bei Karaf wird so ein Subsystem Feature genannt und in dem eigenen Karaf-Archiv-Format (KAR) zusammenverpackt.

Zusätzlich soll in diesem Jahr mit dem Enterprise OSGi Release 5.0 das OSGi Bundle Repository (OBR) (RFC-0112) standardisiert werden. Damit wird die einheitliche und entfernte Installation von größeren Enterprise-OSGi-Anwendungen erheblich erleichtert. Bis zur Umsetzung dieser Spezifikation in den entsprechenden Umgebungen wird leider noch etwas Zeit vergehen, sodass man hier zurzeit mit den proprietären Erweiterungen der einzelnen Implementierungen leben muss.

Für Karaf können Sie sich die bereits verfügbaren Funktionalitäten mit karaf@root>features:list anzeigen lassen oder sich z. B. mit features:install webconsole die Verwaltungskonsole (s. Abb. 3) mit dazugehörigem Webserver installieren. Diese können Sie mit http://localhost:8181/system/console und dem Benutzer karaf sowie dem gleichnamigen Passwort aufrufen.

Apache Karaf Web Console
Bundles

Admin | Bundles | Configuration Status | Dienste | Events | Features | Gogo | Konfiguration | Lizenzen | Log Service | OSGi Repository | Shell | System Information

Bundle information: 83 Bundles total, 82 aktive Bundles, 1 aktive Fragmente, 0 aufgelöste Bundles, 0 installierte Bundles.

Filter anwenden | Filter auf Header anwenden | Aktualisieren | Installieren/Aktualisieren... | Packages aktualisieren

Id	Name	Version	Kategorie	Status	Aktionen
0	System Bundle *(org.apache.felix.framework)*	3.0.9		Aktiv	
9	Apache Aries Blueprint Bundle *(org.apache.aries.blueprint;blueprint.graceperiod:=false)*	0.3.1		Aktiv	
18	Apache Aries JMX Blueprint Bundle *(org.apache.aries.jmx.blueprint)*	0.3.0		Aktiv	
25	Apache Aries JMX Bundle *(org.apache.aries.jmx)*	0.3.0		Aktiv	
7	Apache Aries Proxy Bundle *(org.apache.aries.proxy)*	0.3.0		Aktiv	
8	Apache Aries Util *(org.apache.aries.util)*	0.3.0		Aktiv	
50	Apache Felix Bundle Repository *(org.apache.felix.bundlerepository)*	1.6.4		Aktiv	
5	Apache Felix Configuration Admin Service *(org.apache.felix.configadmin)*	1.2.8	osgi	Aktiv	
6	Apache Felix File Install *(org.apache.felix.fileinstall)*	3.1.10		Aktiv	
76	Apache Felix Metatype Service *(org.apache.felix.metatype)*	1.0.4	osgi	Aktiv	
82	Apache Felix Web Console Event Plugin *(org.apache.felix.webconsole.plugins.event)*	1.0.2		Aktiv	
36	Apache Karaf :: Admin :: Command *(org.apache.karaf.admin.command)*	2.2.4		Aktiv	
19	Apache Karaf :: Admin :: Core *(org.apache.karaf.admin.core)*	2.2.4		Aktiv	
38	Apache Karaf :: Admin :: Management *(org.apache.karaf.admin.management)*	2.2.4		Aktiv	
20	Apache Karaf :: Deployer :: Blueprint *(org.apache.karaf.deployer.blueprint;blueprint.graceperiod:=false)*	2.2.4		Aktiv	
37	Apache Karaf :: Deployer :: Features *(org.apache.karaf.deployer.features;blueprint.graceperiod:=false)*	2.2.4		Aktiv	
40	Apache Karaf :: Deployer :: Karaf Archive (.kar) *(org.apache.karaf.deployer.kar;blueprint.graceperiod:=false)*	2.2.4		Aktiv	
11	Apache Karaf :: Deployer :: Spring *(org.apache.karaf.deployer.spring;blueprint.graceperiod:=false)*	2.2.4		Aktiv	
32	Apache Karaf :: Deployer :: Wrap Non OSGi Jar *(org.apache.karaf.deployer.wrap)*	2.2.4		Aktiv	

Abb. 3: Karaf-Webkonsole

Wenn Sie zusätzliche Funktionalitäten verfügbar machen wollen, so müssen Sie dazu die jeweilige Feature-Repository-URL registrieren, um dann von dort die benötigten Funktionalitäten remote zu installieren. Die bereits existierenden Repositories können Sie mit features:listrepositories anzeigen. Möchten Sie z. B. CXF, Camel- oder ActiveMQ-Funktionen nutzen, so können Sie diese mit

```
features:addurl mvn:org.apache.cxf.karaf/apache-cxf/2.6.1/xml/features
features:addurl mvn:org.apache.camel.karaf/apache-camel/2.10.1/xml/features
features:addurl mvn:org.apache.activemq/activemq-karaf/5.6.0/xml/features
```

über das Maven-Protokoll aus einem entfernten Maven-Repository hinzufügen oder auch zunächst hinzufügen und dann mit

```
features:install cxf
features:install camel-core
features:install camel-cxf
features:install activemq
```

installieren.

Um Webservices verwenden zu können, sollten Sie die *nicht* OSGi-fizierten Standard-JRE-WebService- und XML-Klassen deaktivieren, indem Sie ihrer jeweiligen Paket-Namen # javax.jws und # javax.xml in der Karaf-Konfigurationsdatei etc/jre.properties auskommentieren. Um JavaEE- oder Blueprint-Anwendungen installieren zu können, müssen Sie mit folgenden Befehlen die Voraussetzungen dafür schaffen:

```
features:install war
features:install jpa
features:install application-without-isolation
```

Jetzt können Sie einfach eine Anwendung im neuen Installationsformat Enterprise Bundle Applications (EBA), OSGi Web Application Bundles (WAB) und OSGi-Bundles (JAR) installieren, indem Sie dies bei einer mit karaf.[bat|sh] gestarteten Umgebung in das deploy-Verzeichnis kopieren. Wenn Sie hier org.apache.aries.samples.blog.jdbc.eba-0.3.eba oder org.apache.aries.samples.ariestrader.jdbc-0.3.eba kopieren [ARIES], stehen die Beispielwebanwendungen mit http://localhost:8181/ariestrader oder http://localhost:8181/blog zur Verfügung.

Neben dem Karaf-Hauptprojekt gibt es noch das Cluster-Unterprojekt Cellar, das auf dem verteilten Cache Hazelcast basiert, und Cave, das einen auf Subsystemen beruhenden OBR implementiert. Mit KarafEE (EE = Enterprise Edition) gibt es einen Vorschlag, ähnlich wie bei TomEE, eine vorkonfigurierte und getestete Variante im OpenEJB-Projekt zu veröffentlichen, der dann OpenJPA, Aries JPA, Aries JNDI, Pax Web und Aries Transaction Manager enthält.

Eine Aries-Blueprint-Anwendung besteht neben mehreren bundleN.jar aus mindestens einem Eintrag in der Datei META-INF/APPLICATION.MF. Ein OSGi-Bundle ist selbstbeschreibend durch die Datei META-INF/MANIFEST.MF dargestellt. Für JPA muss zusätzlich noch die Datei META-INF\persistence.xml und der durch Blueprint zur Verfügung gestellte Dienst in der Datei OSGI-INF/blueprint/blueprint.xml beschrieben werden.

Das klingt nach mehr Arbeit. Als Best-Practicies hat es sich bewährt, die für das Deployment benötigten OSGi-Beschreibungsdateien am besten von den entsprechenden Maven-Plug-ins oder dem Bndtools-Pluug-in für Eclipse erzeugen zu lassen. Alternativ kann man auch entsprechende OSGi-Plug-ins für Eclipse verwenden. Durch die explizite Beschreibung der Abhängigkeiten werden nicht erfüllte Abhängigkeiten entweder beim Deployen oder spätestens beim Starten der Anwendung und nicht erst zur Laufzeit, wie bei JavaEE, aufgelöst. Da die Versionen der Bundles deklariert sind, können in einem OSGi-Container mehrere unterschiedliche Versionen eines Bundles parallel existieren, was für die Aktualisierung der Anwendung hilfreich ist und Bibliothekskonflikte der berüchtigten JAR-Hölle vermeidet.

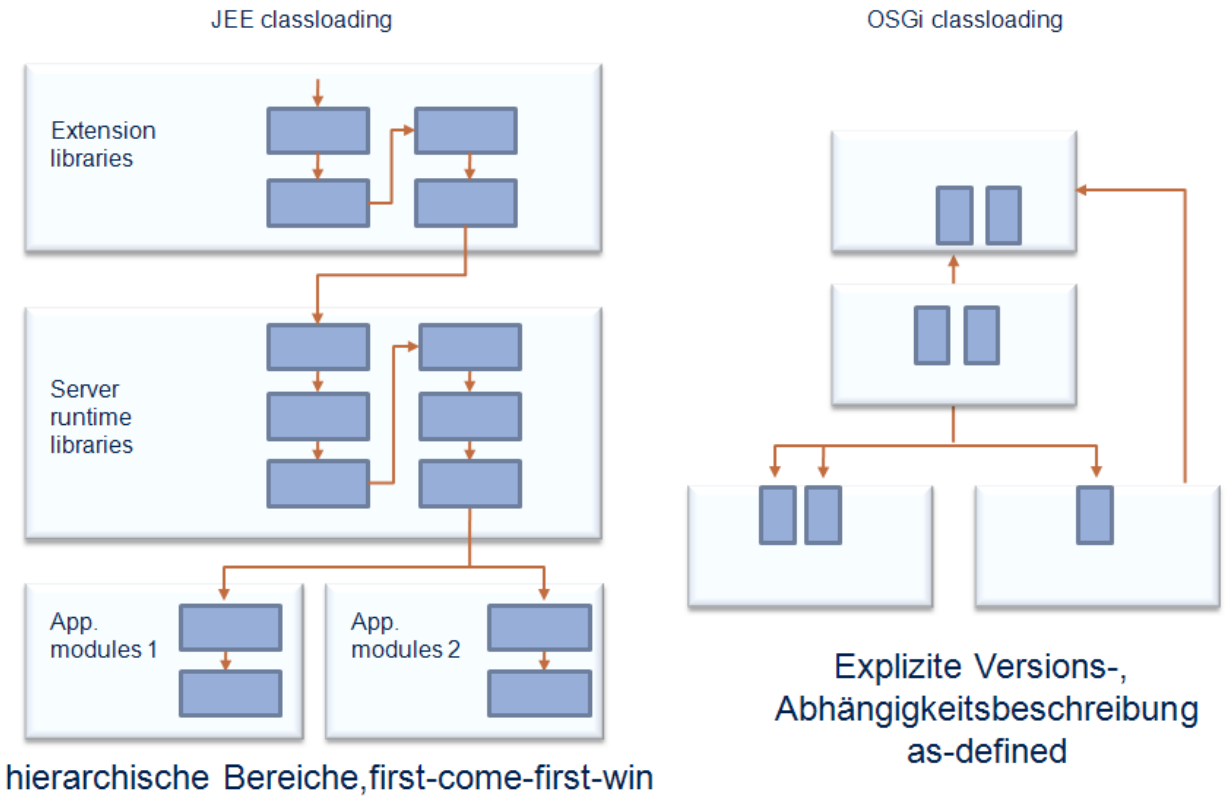

Abb. 4: Hierarchisches oder vernetztes Klassenladen

JavaEE trifft OSGi

Anwendungsserver haben sich als hochskalierbare Ablaufumgebung für eigene Anwendungen und Zusatzprodukte wie Portale, Prozesse oder Messaging-Systeme bewährt. Durch die erfolgte Marktkonsolidierung im Bereich Java-Anbieter spielt Open Source hier eine immer größere Rolle. Mit OSGi wird es leichter, einzelne Funktionen auszutauschen oder nach Bedarf dynamisch zu erweitern.

Apache Geronimo ist schon länger als Anwendungsserver mit einem eigenen GBean-Modulkonzept für J2EE 1.4 und JavaEE 5 zertifiziert. Die aktuelle Version Geronimo 3.0 [GERO30] wird in sechs Varianten ausgeliefert. Die Variante mit Tomcat, Axis und Wink ist vollständig für Java EE 6 und das Web Profile zertifiziert. Die Variante mit Jetty und CXF wird erst später die Zertifizierung erreichen.

Durch die Umstellung der Architektur von den proprietären GBeans auf eine standardisierte OSGi-Umgebung können sowohl bestehende JavaEE-Anwendungen und neue OSGi-Anwendungen nebeneinander verwendet werden. Wie bisher können Deployment-Abhängigkeiten in einem Geronimo-Deployment-Plan beschrieben werden. Zusätzlich sind die OSGi-Bundles jedoch selbstbeschreibend, sodass diese ihre Abhängigkeitsdefinitionen mitbringen oder gegenüber einem OBR auflösen können.

Statt der bisherigen Geronimo-Shell (GShell) wird als Verwaltungsumgebung Apache Karaf verwendet. Da Karaf von einigen anderen Apache-Projekten, wie ServiceMix, ActiveMQ, CXF und Camel, unterstützt wird, sind die Integration und das Arbeiten mit diesen verschiedenen Umgebungen einheitlich. Nach Starten des Geronimo-Servers mit geronimo.[bat|sh] run gelangen Sie durch Tab in die Karaf-Umgebung und können sich mit help dessen GoGo-Befehle anzeigen lassen.

In der Karaf-Konfigurationsdatei etc/config.properties können Sie statt des OSGi-Containers equinox den ebenfalls mitgelieferten Apache Felix verwenden, indem Sie karaf.framework=equinox auskommentieren und die Kommentarzeichen aus der Zeile mit karaf.framework=felix entfernen. Offiziell werden Java 6 und 7 als Laufzeitumgebung sowohl von Oracle als auch von IBM unterstützt.

Mit der Geronimo 3.0-Version wird sowohl Karaf als auch Aries ausgeliefert, so dass damit direkt die

OSGi-Enterprise-Funktionen genutzt werden können. Eine Anwendung kann lokal mit einer schlanken Karaf-Umgebung entwickelt werden, um identisch auf der ausgereiften und skalierbaren Ablaufumgebung des Geronimo-Anwendungsservers nahtlos abzulaufen. Damit lassen sich die Vorteile der beiden Ansätze OSGi und Java für Enterprise-Anwendungen optimal nutzen. Wer möchte, kann bei Partner-Unternehmen, wie IBM für die Geronimo-Variante WebSphere Community Edition oder bei Talend oder RedHat für die auf Karaf basierende Camel oder CXF-Bundles offizielle Support-Leistungen erwerben.

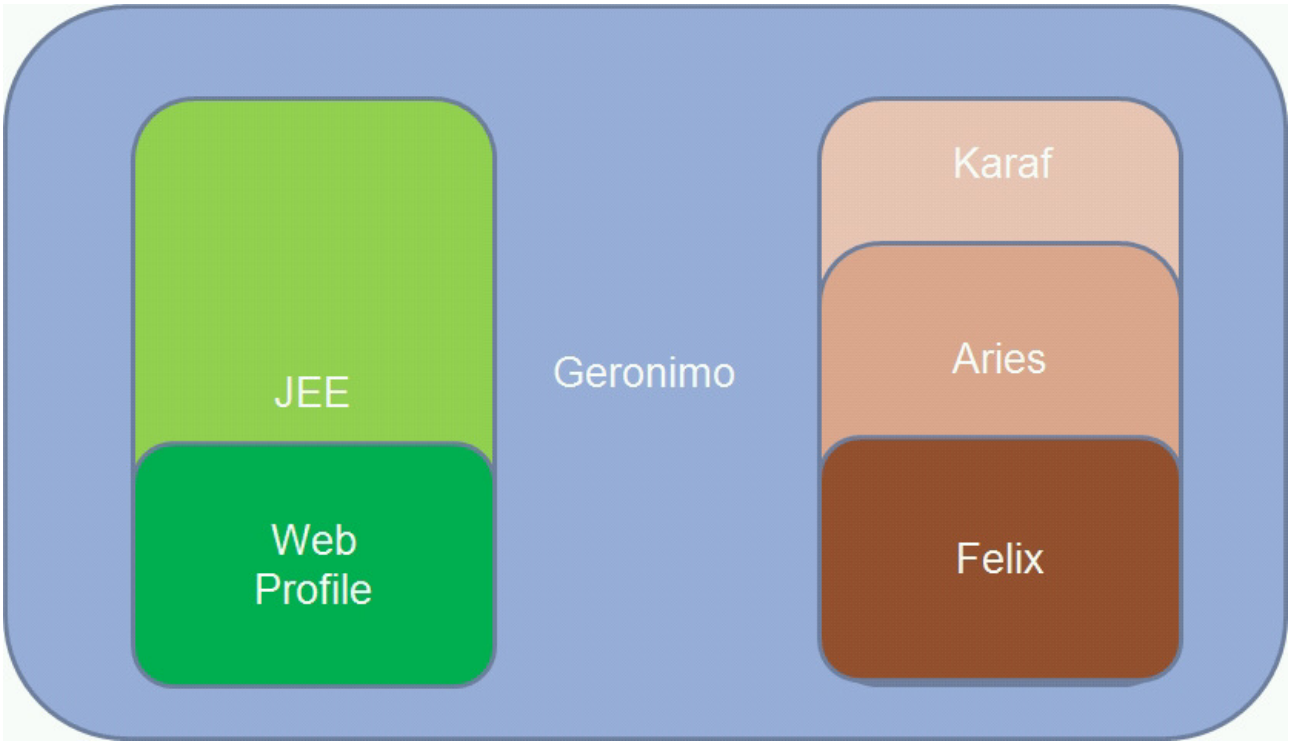

Abb. 5: OSGi-JavaEE in Geronimo

Eine erste Bloganwendung

Um das schon genannte Blog-Beispiel [Aries] in Geronimo zu installieren, müssen zunächst die verwendete Datenquelle und die dazugehörigen Tabellen mit

```
deploy deploy tranql-connector-derby-embed-xa-1.7.rar
  aries-datasource.xml
```

angelegt werden. Das Enterprise-Blueprint-Bundle wird entweder ins deploy-Verzeichnis oder mit

```
deploy deploy org.apache.aries.samples.blog.jdbc.eba-0.3.eba
```

installiert. Dann kann die Bloganwendung mit http://localhost:8080/blog aufgerufen werden. Die Aufteilung der Anwendung in abhängige Bundles mit den dazugehörigen Beschreibungsdateien ist in Abbildung 6 visualisiert.

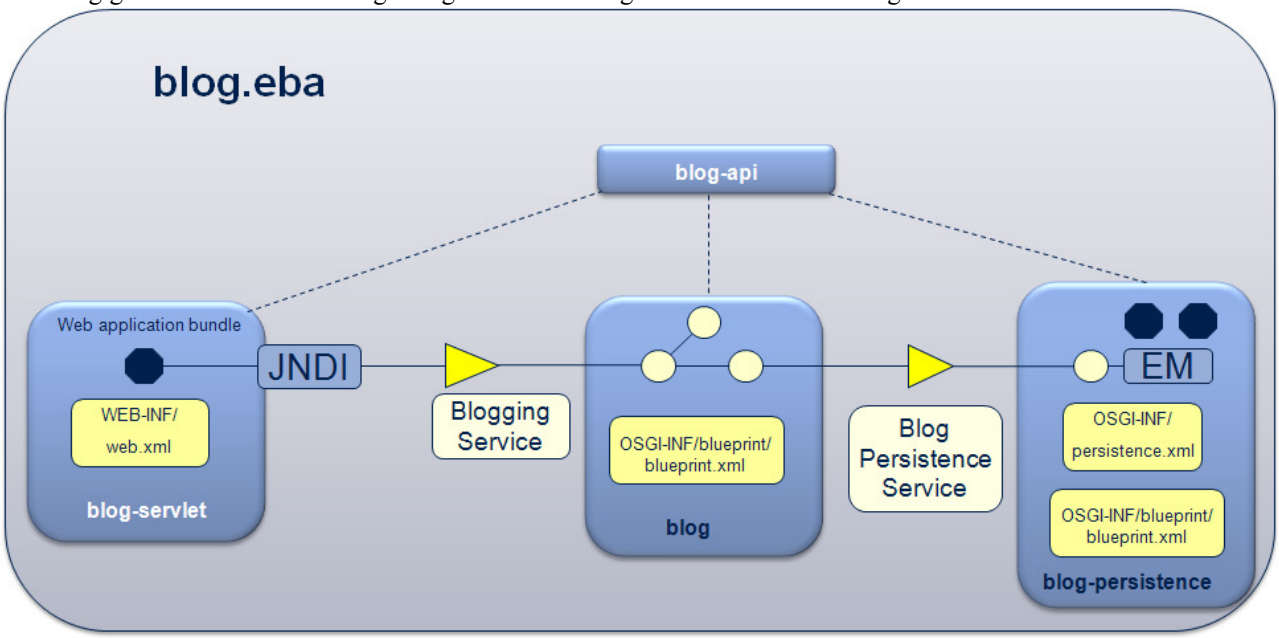

Abb. 6: Anwendungsaufbau: Blueprint-Blog-Beispiel mit JNDI und JPA

Ausblick und Fazit

OSGi hat sich inzwischen in vielen Bereichen etabliert und lässt sich mit den Blueprint-Erweiterungen gut mit Java verwenden. Inzwischen sind einige Java-Bibliotheken auch als OSGi-Bundles verfügbar und einige aktuelle Anwendungsserver nutzen OSGi nicht nur intern, sondern ermöglichen auch das Deployment von OSGi-Artefakten. Gerade die in 2012 erschienenen OSGi Release 5.0 Enterprise-Spezifikation enthält für das Provisioning und Monitoring wichtige Erweiterungen, mit denen sich schneller und flexibler größere Anwendungen realisieren und betreiben können.

Die Modularisierung der Systeme ist auch heute nicht einfach zu haben, aber nötig und mit OSGi möglich. Die Investition in Flexibilität und Anpassbarkeit mit OSGi lohnt sich, trotz des initial höheren Aufwandes langfristig.

Mit der neuen Java Enterprise Edition 6 und dem darin enthaltenen Web Profile werden die Anwendungsserver deutlich schlanker und die Entwicklung und Installation neuer Anwendungen deutlich schneller. In der aktuellen Geronimo-Version ist das Beste aus den beiden Welten OSGi und Java EE integriert und verfügbar. Durch das gute Zusammenspiel mit vielen weit verbreiteten und bereits OSGi-fizierten Apache-Komponenten steht mit Geronimo 3.0 oder Karaf 3.0 eine ausgereifte Umgebung zu Verfügung. Dadurch, dass OSGi die bestehende Entwicklungs- als auch Betriebsinfrastruktur wie bei der neuen hybriden Geronimo-Version mitverwenden kann, steigen die Chancen auf eine größere Verbreitung. Gleichzeitig können ältere JavaEE-Anwendungen relativ leicht auf die neue Version gebracht werden. Damit bestehen interessante Alternativen für zukünftige Entwicklungen.

Wie sich die beiden Welten OSGi und JavaEE weiter aufeinander zubewegen, wird sich in den nächsten Jahren zeigen. Dadurch, dass sich sowohl die Modulkonzept auf Java 9 verschoben hat, ist es wichtig mit OSGi ein etabliertes Modularisierungskonzept zu haben, das inzwischen die wichtigsten JavaEE-Standards unterstützt werden und einige aktuelle Applikationsserver das Deployment von OSGi-Anwendungen erlauben. So kann man mit der bewährten Infrastruktur sowohl JavaEE-Anwendungen und OSGi-Anwendungen betreiben. Somit können bestehende Anwendung auch für den Betrieb schrittweise modularisiert werden, wodurch sich eine höhere Flexibilisierung und Dynamisierung des Deployments ergibt.

Jetzt ist ein guter Zeitpunkt sich mit den bereits existierenden bewährten Konzepten und deren Umsetzung in Produkte zu beschäftigen, um für die Anforderungen der Zukunft gerüstet zu sein.

Literatur und Links

[Alves11] A. Alves, OSGi in Depth, Manning, 2011

[ARIES] Apache Aries-Beispiele, https://cwiki.apache.org/GMOxDEV/apache-aries-samples-running-in-geronimo-30.html, http://aries.apache.org/modules/samples/ariestrader.html, http://aries.apache.org/modules/samples/blog-sample.html

[GERO30] Geronimo 3.0, https://geronimo.apache.org/2011/11/16/apache-geronimo-v30-beta-1-released.html

[KARAF] http://karaf.apache.org

[Müller94] H. A. Müller, K. Wong, S. R. Tilley, Understanding Software Systems Using Reverse Engineering Technology, 1994, 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences Proceedings, s. a.: http://www.rigi.cs.uvic.ca/downloads/pdf/ussuret.pdf

[OSGI] OSGi-Spezifikationen, OSGi Enterprise Expert Group (EEG): http://www.osgi.org/EEG/HomePage und OSGi Release 5 Core und Enterprise Spezifikationen: http://www.osgi.org/Specifications/HomePage,

[Ward12] H. Cummins, T. Ward, Enterprise OSGi in Action, Manning, 2012

Frank Pientka ist Senior Software Architect bei der Materna GmbH in Dortmund. Er ist zertifizierter SCJP und Gründungsmitglied des iSAQB. Als Autor eines Buches zu Apache Geronimo beschäftigt er sich intensiv mit dem Einsatz von Java-Open-Source-Software. E-Mail: frank.pientka@gmx.de