Der goldene Schnitt

Webanwendungen mit OSGi modularisieren

Christoph v. Kuepach, Stefan Schedl

OSGi-Frameworks werden häufig auf dem Desktop eingesetzt, können aber auch auf dem Server genutzt werden. In diesem Artikel werden die Vorteile solcher OSGi-Webanwendungen beleuchtet und die unterschiedlichen Architekturen dafür erläutert. Außerdem wird dargestellt, wie ein OSGi-Framework in Webanwendungen bzw. WARs eingebunden werden kann.

OSGi auf Applikationsservern

Durch den verbreiteten Einsatz von Eclipse-RCP als Anwendungsplattform ist OSGi zu einem der Standards für Modularisierung in Desktop-Applikationen geworden, und dies zu Recht, da dieser viele Vorteile bietet, wie z. B. ein klares Abhängigkeitsmanagement und eine Vermeidung des sogenannten JAR-Hells [Wikipedia]. Auch verschiedene Versionen einer Programmbibliothek können so gleichzeitig innerhalb einer Anwendung genutzt werden.

Bei der Entwicklung von Webanwendungen ist OSGi bisher nicht verbreitet. Deshalb musste auf all diese Vorteile verzichtet werden. Aber spätestens mit Glassfish 3 wird OSGi auf Applikationsservern Einzug halten. Damit kann die Webanwendung als Teil eines OSGi-Bundles deployed werden.

Aber soll man nur für eine modulare Webanwendung gleich einen komplett neuen Applikationsserver einsetzen? Auch dafür gibt es eine Lösung: Das OSGi-Framework, eingebettet als Teil der Webanwendung.

Die wichtigsten Vorteile von OSGi auf einen Blick

Die Wartung und das Änderungsmanagement von großen und komplexen Applikationen stellen eine große Herausforderung und einen Hauptkostenfaktor im Software-Lifecycle dar. Änderungen sollen so schnell wie möglich und natürlich ohne Nebenwirkungen umgesetzt werden und trotzdem sollen einzelne Teile der Anwendung wiederverwendbar bleiben.

Mit OSGi wird es in Java erleichtert, die Anwendung in Teilbereiche (Module) aufzuteilen. Dies können sowohl externe Bibliotheken als auch selbst implementierte Module sein. So können mit OSGi dynamische Module definiert und explizite Sichtbarkeiten und Abhängigkeiten zwischen den einzelnen Modulen festgelegt werden. Außerdem definiert OSGi auch eine Art Lebenszyklus für Module. Damit können diese zur Laufzeit hinzugefügt, ausgetauscht, aktualisiert oder entfernt werden.

Für ein dynamisch verwaltetes System, in welchem die einzelnen Module zusammenarbeiten, bietet OSGi auch ein Service-Modell mit zugehöriger Infrastruktur an.

Die Vorteile von OSGi sind also:

- *Modularisierung:* Mit wohldefinierten Schnittstellen zwischen den Modulen kann die Entwicklung des Softwaresystems parallelisiert und die Komplexität bei gleichzeitig steigender Qualität reduziert werden. So können sich ein oder mehrere Entwickler auf einen Teilbereich spezialisieren.
- *Wartbarkeit:* Ein konkretes Praxisbeispiel: Für den Teilbereich einer Anwendung wurde eine neue Hibernate-Version benötigt. Da das System nicht modularisiert ist, musste für die ganze Anwendung die Hibernate-Version ausgetauscht werden. Dieser simple Austausch zog aber entsprechende Testläufe von vielen Tagen nach sich. In einer OSGi-Anwendung gibt es dieses Problem nicht. Innerhalb der Anwendung kann das gleiche Modul mit verschiedenen Versionen benutzt werden.
- *Services:* Durch den Einsatz der OSGi-Servicearchitektur können Module weiter entkoppelt und so kann die Modularisierung weiter ausgebaut werden. Die Servicearchitektur ermöglicht auch den dynamischen Austausch von Modulen während der Laufzeit [OSGi10].

Zwei mögliche Architekturvarianten

Abb. 1: Aufbau eines herkömmlichen Applikationsservers

In der OSGi-Welt gibt es keine klassischen WAR-Archive mehr (s. Abb. 1), stattdessen besteht eine Webanwendung aus OSGi-Bundles, ähnlich einer Eclipse-RCP-Anwendung. Einzelne Funktionalitäten können so in eigene Bundles verteilt werden, ebenso wie die benötigten Java-Bibliotheken, die als eigene Bundles eingebunden werden.

Grundsätzlich gibt es zwei Möglichkeiten, OSGi-Webanwendungen zu realisieren. Entweder:

- wird ein leicht-gewichtiger Servlet-Container, z. B. Jetty, in den OSGi-Container eingebettet, oder
- der OSGi-Container wird in eine Webapplikation integriert. Damit kann die Applikation wie eine ganz normale Webanwendung in einem Servlet-Container, wie z. B. Apache Tomcat, laufen.

Die Webanwendung in einem OSGi-fähigen Applikationsserver

Der Web-Container (s. Abb. 2) wird als Bundle innerhalb des OSGi-Servers deployed und gestartet. Die entwickelten Module nutzen die Services des Web-Containers und stellen die Inhalte, wie z. B. die JSPs, die Servlets oder die statische Webseiten, bereit. Als Web-Container stehen bereits verschiedene Implementierungen zur Verfügung:

- OSGi HTTP Service (Standard im OSGi Release R1),
- Equinox Jetty Integration (aus Eclipse),
- PAX Web,
- Spring DM Web Support.

Der OSGi HTTP Service wird seit dem OSGi-Release R1 als Standardservice mitgeliefert. Allerdings ist die Implementierung sehr leichtgewichtig und auf wenige Basisfeatures begrenzt. Das größte Manko ist die fehlende JSP-Unterstützung. Filter oder Listener sucht man auch vergeblich.

PAX Web erweitert den OSGi HTTP Service mit einem besserem Servlet-Support, mit Filtern, Listener, Errorpages und JSP-Unterstützung (s. Tabelle 1).

Applikationsserver, die OSGi-Bundles unterstützen, gibt es zurzeit noch sehr wenige, bzw. sie sind noch nicht beson-

	OSGi HTTP Service	PAX Web
Context name	-	✓
Context params	-	✓
Session timeout	-	✓
Servlets	✓	✓
Servlet init params	✓	✓
Servlet mapping	✓ (keine wildcards)	✓
Filters	-	✓
Filter init params	-	✓
Filter mapping	-	✓
Listener	-	✓
Error pages	-	✓
Welcome files	-	✓
Mime mapping	✓	✓

Tabelle 1: Vergleich von OSGi http Service und PAX Web [pax]

Abb. 2: Webanwendung in einem OSGi-Server

ders verbreitet. Als Beispiel kann hier Glassfish v3 genannt werden. Glassfish ist in der Lage, eine OSGi-Webanwendung direkt ohne den Umweg über eine Servlet-Bridge auszuführen. Anwendungen werden hier einfach als Bundles installiert.

OSGi-Framework eingebettet in eine Webanwendung

Ein Austausch des Applikationsservers kommt bei bestehenden Anwendungen und Systemlandschaften meistens nicht infrage. Hierfür kann das OSGi-Framework aber in eine Webanwendung eingebettet werden. Die eigentlichen Anwendungs-Bundles werden innerhalb dieser OSGi-Umgebung ausgeführt.

Mit der Eclipse-OSGi-Implementierung Equinox kann OSGi in ein WAR-Archiv (s. Abb. 3) eingebunden werden. Die Verbindung zwischen Webserver und OSGi-Framework wird hier über die Equinox-Servlet-Bridge hergestellt. Dies ist ein einzelnes Servlet, welches das OSGi-Framework startet und die eingehenden Anfragen an den OSGi HTTP Service weiterleitet.

Abb. 3: OSGi in einer bestehenden Applikation

Einen guten Startpunkt bietet die Equinox-Homepage [brigde]. Dort gibt es die Datei „bridge.war“, die eine fertige Servlet-Bridge-Webanwendung enthält. Außer dem brigde-Servlet enthält das WAR-Archiv eine Equinox-Distribution. Dies ist der Inhalt des „eclipse“-Verzeichnisses. Allerdings ist diese Equinox-Version sehr alt und sollte bei Bedarf aktualisiert werden.

Die selbst entwickelten Bundles werden in das Verzeichnis „plugins“ importiert und zusätzlich in der Datei „config.ini“ registriert:

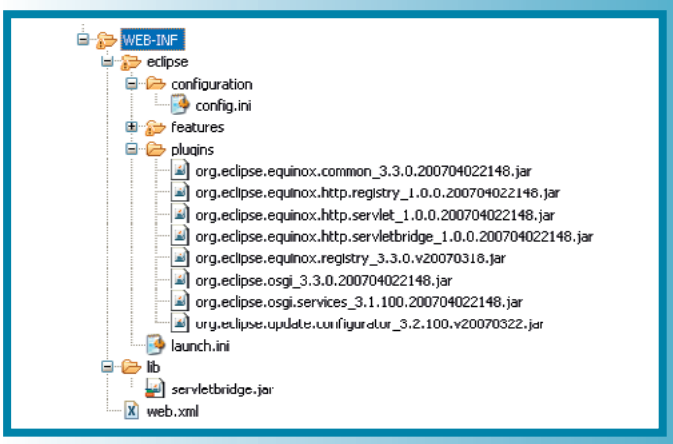

Abb. 4: Inhalt der brigde.war

```
#Eclipse Runtime Configuration File
osgi.bundles=org.eclipse.equinox.common@2:start,
             org.eclipse.update.configurator@start,
             org.eclipse.equinox.http.servletbridge@start,
             org.eclipse.equinox.http.registry@start,
             org.rsp.example.resource.css@start
sgi.bundles.defaultStartLevel=4
```

Damit werden die eigenen Bundles beim Applikationsstart automatisch aktiviert.

Im Gegensatz zur traditionellen Webentwicklung wird in der Datei „web.xml“ nur das in dem Java-Archiv „servletbrigde.jar“ enthaltene Equinox-Bridge-Servlet angegeben. Die Equinox-Instanz wird über `init`-Parameter konfiguriert:

```
<servlet id="bridge">
  <servlet-name>equinoxbridgeservlet</servlet-name>
  <display-name>Equinox Bridge Servlet</display-name>
  <description>Equinox Bridge Servlet</description>
  <servlet-class>org.eclipse.equinox.servletbridge.BridgeServlet
  </servlet-class>
  <init-param>
    <param-name>commandline</param-name>
    <param-value>-console</param-value>
  </init-param>
```

```
osgi> ss
Framework is launched.

id      State       Bundle
0       ACTIVE      org.eclipse.osgi_3.4.2.R34x_v20080806-1600
                    Fragments=21
1       ACTIVE      org.eclipse.equinox.common_3.4.0.v20080421-2006
2       ACTIVE      org.eclipse.update.configurator_3.2.100.v20070322
3       ACTIVE      org.eclipse.equinox.http.servletbridge_1.0.0.200704022148
4       ACTIVE      org.eclipse.equinox.http.registry_1.0.100.v20080427-0830
5       ACTIVE      com.mycompany.module1_1.0.1
6       ACTIVE      com.mycompany.module2_1.0.0
7       RESOLVED    javax.servlet.jsp_2.0.0.v200806031607
8       RESOLVED    javax.servlet_2.4.0.200811191445
9       RESOLVED    org.apache.commons.el_1.0.0.v200806031608
10      RESOLVED    org.apache.commons.logging_1.0.4.v20080605-1930
11      RESOLVED    org.apache.jasper_5.5.17.v200806031609
13      ACTIVE      org.eclipse.equinox.http.helper_1.0.0.200811191116
16      ACTIVE      org.eclipse.equinox.http.servlet_1.0.100.v20080427-0830
17      <<LAZY>>    org.eclipse.equinox.jsp.jasper.registry_1.0.0.v20080427-0830
18      ACTIVE      org.eclipse.equinox.jsp.jasper_1.0.100.v20080427-0830
19      RESOLVED    org.eclipse.equinox.jsp.jstl_1.0.0
20      ACTIVE      org.eclipse.equinox.registry_3.4.0.v20080516-0950
21      RESOLVED    org.eclipse.equinox.servletbridge.extensionbundle_1.0.0
                    Master=0
22      RESOLVED    org.eclipse.osgi.services_3.1.200.v20071203
25      ACTIVE      org.rsp.example.http.jsp2_1.1.0
26      ACTIVE      org.rsp.example.http.jsp_1.1.0
27      ACTIVE      org.rsp.example.http.servlet_1.0.0
28      ACTIVE      org.rsp.example.resource.css_1.0.0
30      ACTIVE      org.rsp.example.resource_1.0.0
31      ACTIVE      org.rsp.http_1.0.0
32      ACTIVE      org.rsp.resource_1.0.0
33      ACTIVE      com.mycompany.myapp.root_1.0.0

osgi> install file:D:/workspaces/osgi-web/org.rsp.example.resource.css_2.0.0.jar
Bundle id is 41

osgi>
```

Abb. 5: Equinox-Konsole

Ebenfalls in den `init`-Tags ist der Parameter `-console` angegeben. Mit diesem Aufruf wird zusätzlich die Equinox-Konsole (s. Abb. 5) gestartet. Als Test kann „ss“ eingegeben werden. Die Status der Bundles, die Teil der Applikation sind, werden angezeigt. Wie bereits erwähnt, können im laufenden Betrieb Bundles installiert oder ausgetauscht werden. Mit `install file:<Pfad >` kann ein neues Bundle installiert werden. Weitere Schritte, wie z. B. eine Downtime des Systems oder gar ein Neustarten des Applikationsservers, sind nicht notwendig. Nicht einmal die Anwendung selbst muss neu gestartet bzw. undeployed/deployed werden.

Eine gute Anleitung, die sowohl erklärt, wie die Entwicklungsumgebung aufzusetzen ist, als auch einen tieferen Einstieg in die Thematik bietet, kann unter [Geh09] gefunden werden.

Fazit

Die Vorteile von OSGi, wie die Modularisierung, können endlich auch in Webanwendungen genutzt werden. Wenn das OSGi-Framework in das WAR-Archiv eingebunden wird, kann die Webanwendung in bestehenden Infrastrukturen weiter betrieben werden.

Die mögliche Modularisierung entsteht aber nicht automatisch durch den Einsatz von OSGi. Die Aufteilung in die Module muss vom Architekten/Entwickler immer noch sorgfältig geplant werden. Für eine gute Struktur sollten sinnvolle Bundles mit teilweisen Sichtbarkeiten der Pakete definiert werden. Ein guter Ansatzpunkt ist die Trennung der Darstellung von der Logik.

Allerdings muss auf umfangreiche Dokumentation zu OSGi-Webanwendungen noch verzichtet werden.

Links

[brigde] Web Application Archive für Servlet-Bridge-Webanwendung,
`http://www.eclipse.org/equinox/server/downloads/bridge.war`

[Geh09] W. Gehner, Modularizing existing web applications with OSGi, Javalobby, 2009,
`http://java.dzone.com/articles/modularizing-existing-web`

[OSGi10] Benfits of Using OSGi, OSGi Alliance,
`http://www.osgi.org/About/WhyOSGi`

[pax] Pax Web – OPS4J,
`http://wiki.ops4j.org/display/paxweb/Pax+Web`

[Wikipedia] Java Classloader: JAR hell,
`http://en.wikipedia.org/wiki/JAR_hell#JAR_hell`

Christoph v. Kuepach ist als Technical Consultant bei der Cirquent GmbH im Bereich Innovation & Product Lifecycle Management tätig. Dort ist er neben der Projektarbeit u. a. mit der Aufgabe des Softwarearchitekten für eine PLM-Monitoring-Applikation betraut.
E-Mail: christoph.vonKuepach@cirquent.de.

Stefan Schedl arbeitet für die Acando GmbH als Consultant. Sein Aufgabengebiet stellt die Betreuung von Kundenprojekten mit Schwerpunkten Java/JEE5 sowie Rich-Client-Architekturen auf Basis von Eclipse-RCP dar.
E-Mail: stefan.schedl@acando.de.