MAEXO – Management Extensions for OSGi

Wir freuen uns mit MAEXO das zweite Projekt in die Hände von Google-Code geben zu können [1]. MAEXO bildet das fehlende Verbindungsstück zwischen den Technologien OSGi und JMX. Das Ansinnen dieses Projektes ist es, beliebige OSGi-Container unter der Regie von JMX überwachbar und kontrollierbar zu machen. Darüber hinaus geben wir dem Nutzer dieses Frameworks ein Programmiermodell in die Hand, mit dem er seine eigenen JMX-Komponenten nahtlos in die OSGi-Welt integrieren kann. MAEXO steht unter der Apache License, Version 2.0 [2].

Um eine hohe Verfügbarkeit von Anwendungen zu erreichen, sind Möglichkeiten des Managements zur Laufzeit unabdingbar. Dies betrifft insbesondere die Überwachung und Konfiguration von Ressourcen des Systems, z.B. Datenbankverbindungen, und die Aktivierung bzw. Deaktivierung von Tracing-Komponenten zum Aufspüren von Problemquellen.

Für das Laufzeit-Management ergeben sich oftmals folgende Anforderungen:

- Remote-Zugriff: insbesondere bei verteilten Systemen besteht oft keine physischer Zugriff auf die Laufzeitumgebung. Die Kommunikation muss daher zwangläufig über Netzwerkinfrastruktur erfolgen können.
- Authentifizierung und Autorisierung: es muss sichergestellt werden, dass nur berechtigte Personen (z. B. Administratoren) Zugriff auf die zu verwaltenden Ressourcen erlangen können. Dabei können ggf. abgestufte Rechtekonzepte von Interesse sein, die z. B. nur einen lesenden Zugriff für bestimmte Personengruppen ermöglichen.
- Standardisierte Tools: Anwendungen bestehen neben eigenen Implementierungen oft aus Komponenten unterschiedlicher Hersteller (z. B. Web-Container, Messaging-Provider).

Im ungünstigsten Fall bietet jede einzelne Komponente eine proprietäre Möglichkeit für den Management-Zugriff, z. B. durch die Bereitstellung einer spezialisierten Konsole. Dabei können die verwendeten Bedienkonzepte (Shell vs. GUI) und Technologien (Web vs. Rich Client) in Summe eine starke Inhomogenität erzeugen. Darüberhinaus werden die beiden erstgenannten Aspekte Remote-Zugriff und Authentifizierung bzw. Autorisierung oft in unterschiedlicher Qualität abgebildet. Die Verwendung standardisierter Technologien bzw. darauf basierender Werkzeuge ist daher anstrebenswert.

Management OSGi basierter Systeme

Die OSGi-Spezifikation definiert zur Verwaltung der Plattform eine Menge von Diensten (u. a. PackageAdmin, PermissionAdmin, ConfigurationAdmin). Sie lässt jedoch offen, mit welcher Technologie der Zugriff darauf erfolgen soll. Üblicherweise wird eine Kommando-Shell bereitgestellt. Die Art des Zugriffs darauf (telnet, SSH) sowie die verfügbaren Kommandos sind proprietär. Darüber hinaus existieren verschiedene Swing-GUIs und Web-Anwendungen, z. B. Knopflerfish OSGi Desktop, Knopflerfish HTTP Console oder die Apache Felix Web Console.

Derzeit ist ein starker Trend in Richtung OSGi als Container- bzw. Servicemodell für Application-Server bzw. Enterprise-Anwendungen zu beobachten. Dadurch gewinnen die beschriebenen Forderungen an das Laufzeit-Management in diesem Umfeld an Relevanz, können durch die beschriebenen Lösungen aber nur schlecht abgedeckt werden. Besonders die Frage nach standardisierten Tools wird mit steigender Anzahl installierter Komponenten im OSGi-Container dringlich.

Auf der anderen Seite schaffen die Java Management Extensions (JMX) einen Rahmen, um die genannten Anforderungen gut zu befriedigen. Der Kerngedanke liegt dabei in der Bereitstellung von MBeans durch die Anwendung. Diese MBeans repräsentieren zu verwaltende Ressourcen. Sie werden einem MBean-Server bekanntgemacht, der sie über standardisierte Adapter (z. B. RMI oder HTTP) für entfernte Management-Konsolen exportiert. Operationen und Attribute der MBeans ermöglichen dabei, den Zustand der überwachten Ressource abzufragen oder auf diesen Zustand Einfluss zu nehmen.

Insbesondere bei technischen Infrastruktur- bzw. Middleware-Komponenten (z. B. Connection-Pools, Messaging-Provider) hat JMX einen hohen Verbreitungsgrad erreicht. Es ist daher naheliegend, die Ressourcen des OSGi-Containers selbst ebenfalls per JMX zugänglich zu machen.

Hierzu existieren bereits Implementierungen: Eclipse Equinox Resource Monitoring und Apache Felix MOSGi. Beide sind frei verfügbar und stellen sowohl MBeans als auch ein OSGi-konformes Programmiermodell zur Verfügung. Leider sind die Nebenbedingungen hier nur begrenzt akzeptabel. So weist die Eclipse-Implementierung ein sehr hohes Maß an direkten Abhängigkeiten zu Equinox bzw. spezifischen Komponenten aus dem Eclipse-Universum auf. MOSGi ist zwar plattformunabhängig, setzt jedoch die Verwendung einer proprietären Management-Konsole voraus.

Die MAEXO Architektur

Mit MAEXO stellen wir ein Framework zur Verfügung, welches eine Brücke zwischen OSGi und den Java Management Extensions baut, ohne die genannten Einschränkungen zu besitzen. Es besteht grundsätzlich aus zwei Komponenten: Die erste Komponente sind MBean-Implementierungen, welche Ressourcen der OSGi-Spezifikation abbilden, z. B. Bundles und Services. Mithilfe dieser MBeans kann beispielsweise der Status installierter Komponenten oder verfügbarer Dienste überwacht werden.

Der Export der MBeans erfolgt über die zweite Komponente von MAEXO: das Switchboard. Die Aufgabe des Switchboards ist es, Registrierungen bzw. De-Registrierungen von MBeans in der OSGi-Service-Registry zu überwachen. Aufgefundene Instanzen werden automatisch an dem oder den verfügbaren MBean-Servern registriert und damit exportiert.

So wird erreicht, dass ein Bundle, welches MBeans exportiert, sich nicht um das Auffinden eines MBean-Servers und dessen Lebenszyklus kümmern muss.

Gleichzeitig ist sichergestellt, dass MBeans spätestens dann de-registriert werden, sobald das bereitstellende Bundle deaktiviert bzw. deinstalliert wird. Abbildung 1 verdeutlicht die Zusammenhänge.

Das resultierende Programmiermodell steht nicht nur den MAEXO-Komponenten zur Verfügung. Jedes beliebige Bundle kann durch Registrierung von Instanzen unter den jeweiligen MBean-Interfaces in der OSGi-Service-Registry MBeans exportieren.

Dafür wird lediglich der gewünschte Name der MBean als Service-Property benötigt (siehe Abbildung 2).

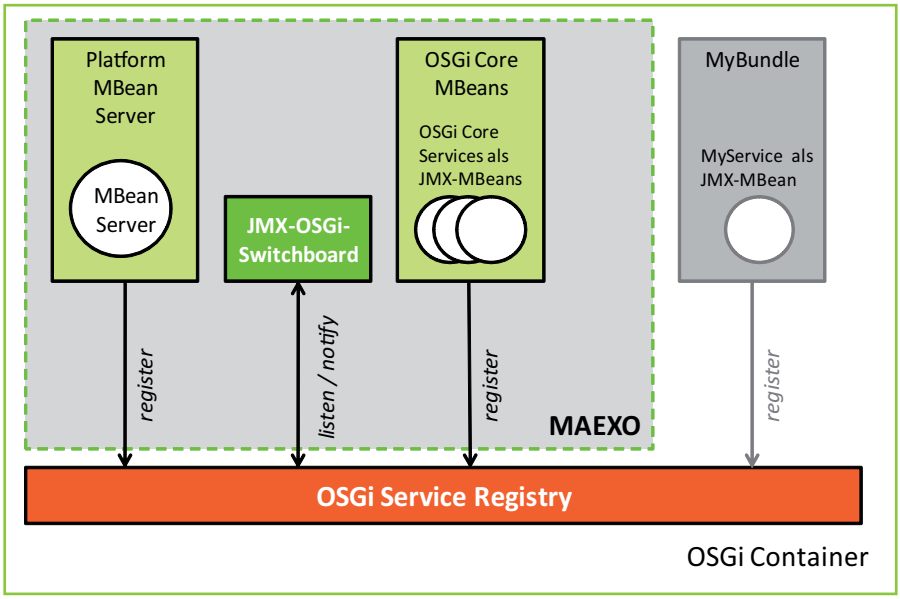

Abb. 1: Die MAEXO Architektur

```
ObjectName objectName = new ObjectName(…);

Dictionary< String, Object > properties =
        new Hashtable< String, Object >( );

properties.put( ObjectName.class.getName, objectName );

ServiceRegistration serviceRegistration =
        this.bundleContext.registerService(
                mbeanInterface.getName( ),
                mbean,
                properties );
```

Abb. 2: Registrierung eigener MBeans

```
<!-- declare the mbean instance -->
<bean id="myMBean"
      class="com.buschmais.my.DynamicMBeanImpl.class"/>

<!-- export the mbean using the DynamicBean interface -->
<!-- and the object name -->
<osgi:service interface="javax.management.DynamicMBean"
              ref="myMBean">
  <osgi:service-properties>
    <entry key="objectName"
           value="com.buschmais.osgi.maexo:id=myMbean"/>
  </osgi:service-properties>
</osgi:service>
```

Abb. 3: Registrierung eigener MBeans mit Spring-DM

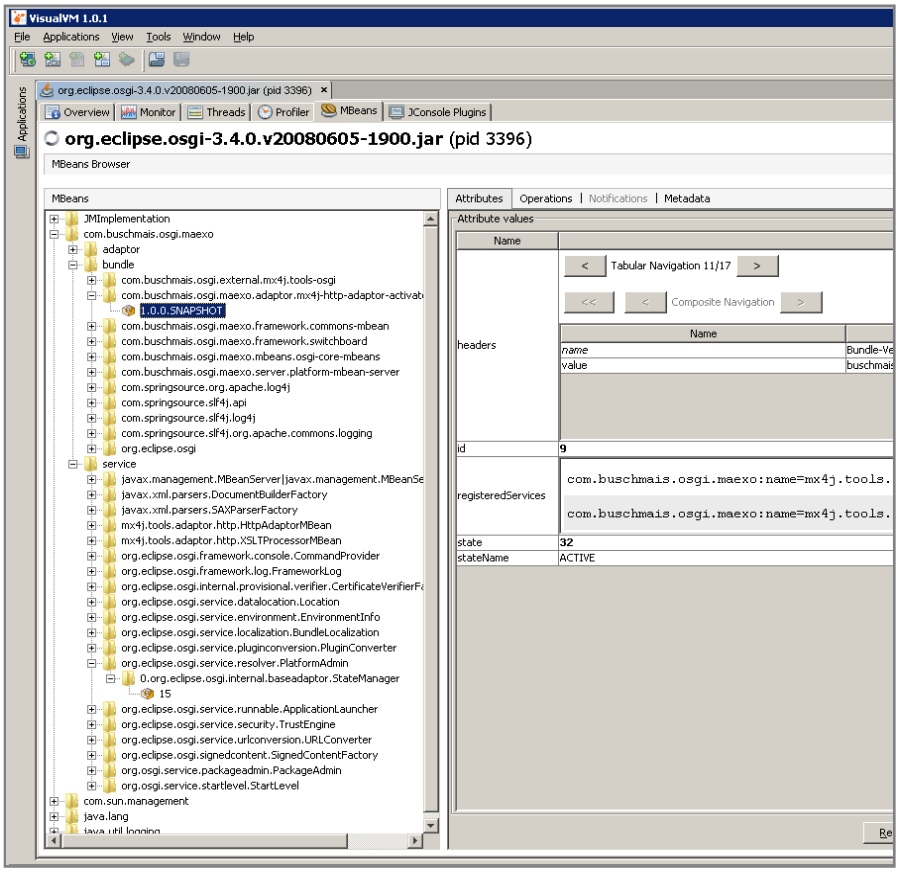

Abb. 4: MAEXO in der VisualVM Konsole

Selbstverständlich können auch deklarative Ansätze verwendet werden. Im Falle von Spring-DM sieht dies wie in Abbildung 3 aus.

MAEXO weist keine Kopplung an eine konkrete OSGi-Plattform auf und besitzt nur eine externe Abhängigkeit zum SLF4j Logging-Framework. Der Installationsaufwand ist somit minimal.

Da sich MAEXO streng an die Spielregeln von JMX hält, können die bereitgestellten MBeans der Anwendung als auch des OSGi-Containers mit gängigen Management-Konsolen, wie JConsole oder VisualVM, angesprochen werden.

Ohne Aufwand steht somit eine vollständige JMX basierte Management-Infrastruktur für OSGi-Container zur Verfügung.

Ausblick

Abbildung 4 zeigt einen Screenshot der Java6-Management-Konsole VisualVM mit Blick auf einen Equinox-OSGi-Container. In diesem Screenshot sieht man bereits die Fülle von OSGi-MBeans, die durch MAEXO erstellt und exportiert werden.

Dieser Artikel sollte einen Vorgeschmack auf unser MAEXO-Projekt geben. Wir haben noch viele Ideen, die wir darin verwirklichen wollen. Seien Sie auch gespannt auf den nächsten Artikel, in dem es um die Installation und die ersten Schritte mit MAEXO gehen wird.

TEXT: DIRK MAHLER

Referenzen

[1] http://code.google.com/p/maexo/
MAEXO bei Google Code

[2] http://www.apache.org/licenses/LICENSE-2.0.html
Apache License, Version 2.0

kontakt

buschmais GbR
Leipziger Straße 93
01127 Dresden

Tel +49 (0)351 3209230
Fax +49 (0)351 32092329
info@buschmais.de
www.buschmais.de