



Innovationsforum

Laufzeitmanagement von OSGi Applikationen

buschmais

Beratung . Technologie . Innovation

Tobias Israel

buschmais GbR

Inhaber

Torsten Busch, Frank Schwarz,
Dirk Mahler und Tobias Israel

tobias.israel@buschmais.com

<http://www.buschmais.de/>

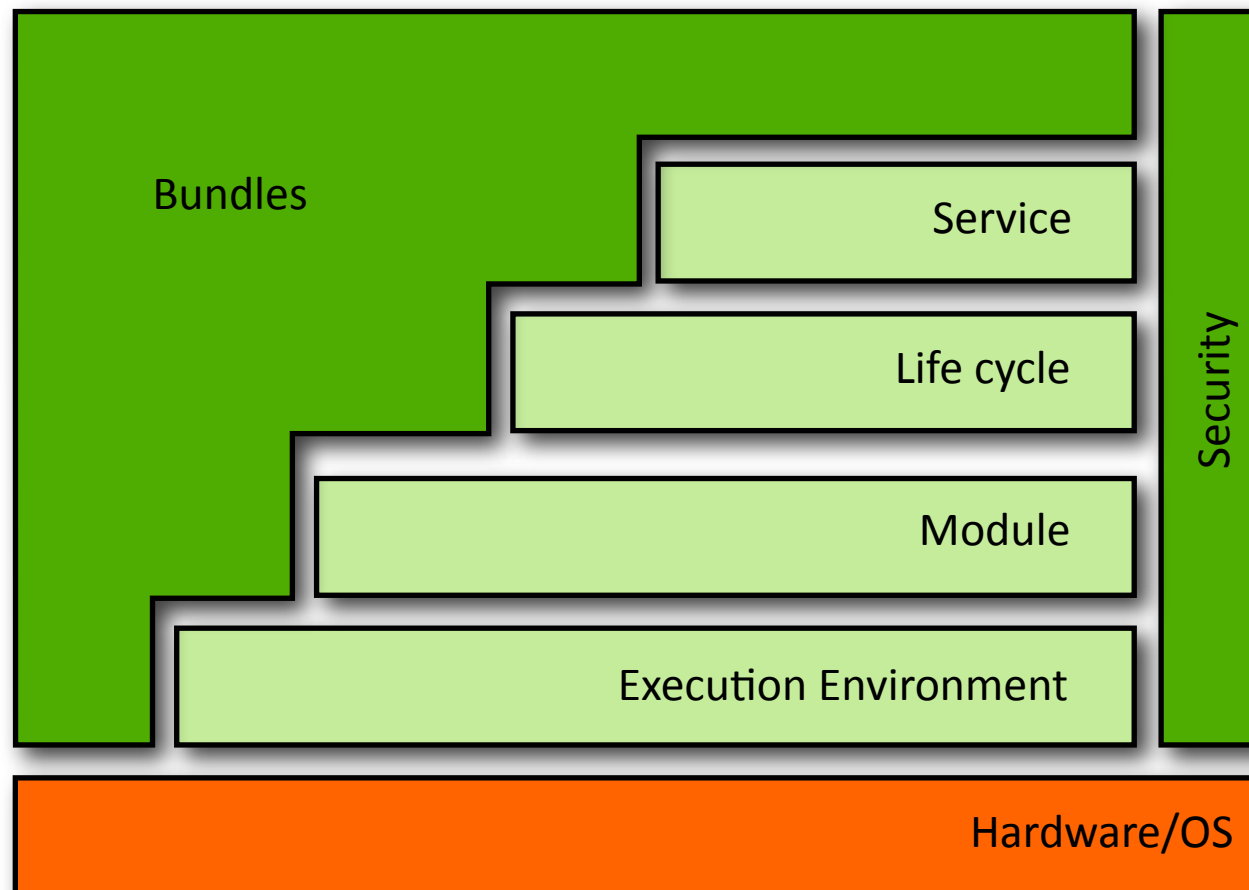
Dresden, 23.04.2010

Einstieg

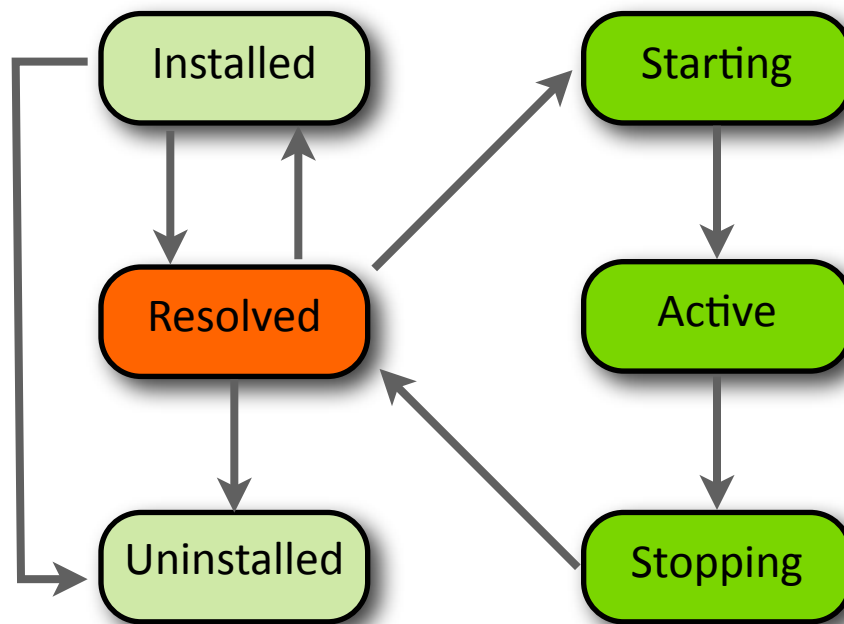
OSGi

- Ursprünglich im Kontext eingebetteter Systeme
- Definition einer dynamischen Serviceplattform auf der Basis von Java
- Begrifflichkeiten:
 - Bundle = Softwarekomponente
 - Service = Schnittstelle
 - Service-Registry = Registry für den Servicelookup;
 - Abgrenzung des Bundles nach außen, Im-/Export von Bundleartefakten
- Lifecycle-Modell (De-/Installieren, Starten, Stoppen, Aktualisieren)

□ OSGi Framework - Architektur



□ Bundle-Lifecycle-Modell:



- Starten und Stoppen von Bundles zur Laufzeit
- Dynamisches Registrieren und Entfernen von Services
- Management des Lifecycles über API
 - Bundle
 - BundleContext
 - BundleActivator
 - BundleEvent
 - FrameworkEvent
 - BundleListener
 - FrameworkListener
 - SystemBundle
 - ...

□ Das Whiteboard-Pattern

■ Problem:

- OSGi-Kontext „lebt“ von dynamischen Abhängigkeiten. Nutzer anderer Services müssen z.B. über den Lifecycle-Status anderer Services von denen sie abhängig sind informiert werden.
- Nutzung des „normalen“ Listener Pattern würde Registrierung von Listnern direkt am zu beobachtenden Service notwendig machen --> Enorme Komplexität!

■ Lösung:

- Alles ist ein Service
- OSGi Service Registry als zentrale Instanz bietet Plattform für Event-Dispatch und Zugriff auf Service-Referenzen

```
public final class Activator implements BundleActivator {

    /**
     * {@inheritDoc}
     */
    public void start(BundleContext bundleContext) throws Exception {
        // Bundle starten

        // Services Registrieren

        // Listener registrieren
    }

    /**
     * {@inheritDoc}
     */
    public void stop(BundleContext bundleContext) throws Exception {
        // Bundle stoppen
    }
}
```



```
ServiceListener serviceListener = new ServiceListener() {  
  
    /**  
     * {@inheritDoc}  
     */  
    public void serviceChanged(ServiceEvent serviceEvent) {  
  
        ServiceReference serviceReference = serviceEvent.getServiceReference();  
  
        switch (serviceEvent.getType()) {  
        case ServiceEvent.REGISTERED:  
            // Ein neuer Service wurde registriert  
            break;  
        case ServiceEvent.UNREGISTERING:  
            // Ein Service wurde entfernt  
            break;  
        default:  
            break;  
        }  
    }  
};  
  
bundleContext.addServiceListener(serviceListener);
```

Einstieg

Laufzeitmanagement

□ Verfügbarkeit von IT-Systemen:

Effektive Betriebszeit (t_b) = Betriebszeit - Wartungszeit

$$\text{Verfügbarkeit} = \frac{t_b - \Sigma \text{Ausfallzeit}}{t_b} \times 100\%$$

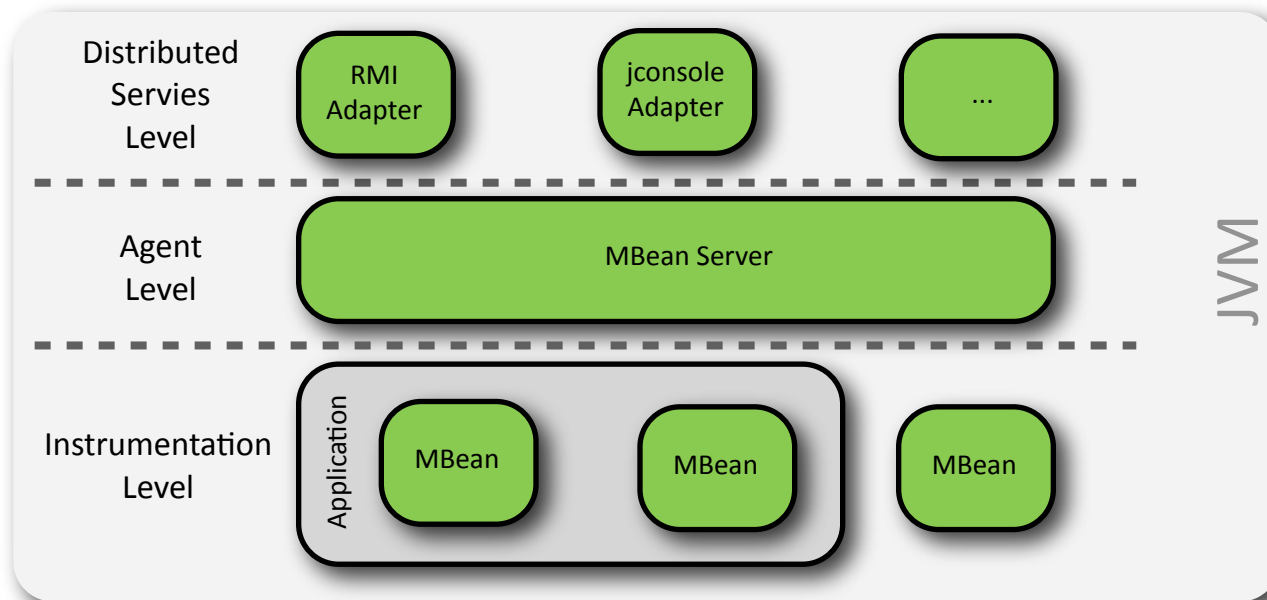
- Schnelle Reaktion auf Abweichungen des Systemverhaltens vom Erwartungswert notwendig
- „Alles was im laufenden Betrieb erledigt werden kann ist gut“

- Einige Rahmenbedingungen für den Betrieb von IT-Systemen
 - Überwachung von Systemressourcen
 - Schnelles Aktivieren-/Deaktivieren spezieller Tracing-Komponenten
 - Gute Zugänglichkeit des Systems
 - Ortsunabhängigkeit
 - Einheitliches (standardisiertes) Toolset
 - Schutz des Systems und dessen Ressourcen vor unautorisiertem Zugriff

Einstieg

Java Management Extension

- Bestandteil von Java SE/EE
- Standardisierte Schnittstelle für Laufzeitmanagement
- Einheitliches Programmiermodell
- Berücksichtigung von Sicherheitsaspekten



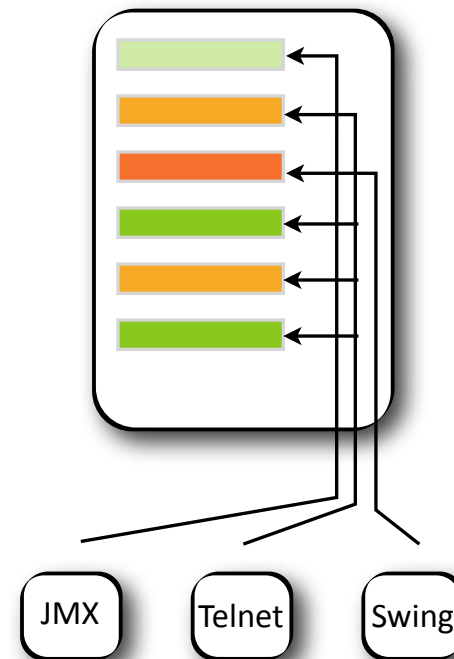
- Definition verschiedener MBean-Typen:
 - Standard MBeans
 - Dynamic MBeans
 - Open MBeans
 - ...
- MBean-Server als Bindeglied zwischen MBeans und der Managementanwendung
 - Registrieren von MBeans an einer MBean-Serverinstanz
 - *ObjectName* als eindeutiger Name für registrierte MBeans

```
com.buschmais.maexo.mbean:description=  
MaexoMBean,type=MaexoMBean
```

Betrachtung

OSGi + Management ?

- Keine standardisierten Schnittstellen nach außen vorgegeben
- Zahlreiche Insellösungen
 - Spezielle Shells
 - telnet
 - HTTP-Konsolen
 - Swing-Anwendungen
 - Spezielle Bundles
 - ...



- Teilweise auch Ansätze zur Nutzung von JMX
 - Bsp.: Eclipse Equinox
 - sehr kalorienreicher Ansatz durch hohe Anzahl von Abhängigkeiten
 - Bsp.: Apache Felix: MOSGi
 - Nutzt Notification-basierten MBean Ansatz zur Anbindung der hauseigenen Managementanwendung
- OSGi Version R4.2
 - Bewegung in Richtung Enterprise Java
 - Explizit auch Bezug zu JMX (RFC139)
 - Kein Programmiermodell zur dynamischen Registrierung/
Deregistrierung von MBeans

Framework

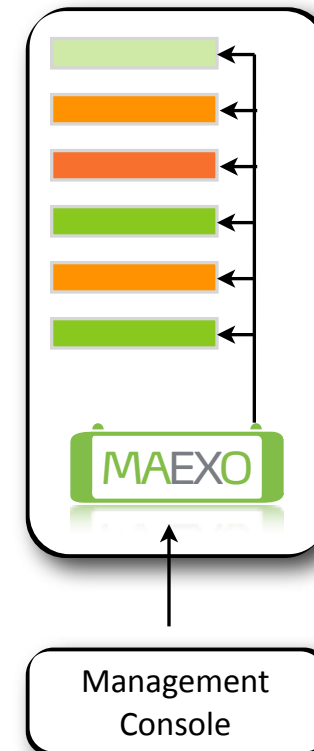
OSGi + JMX = MAEXO



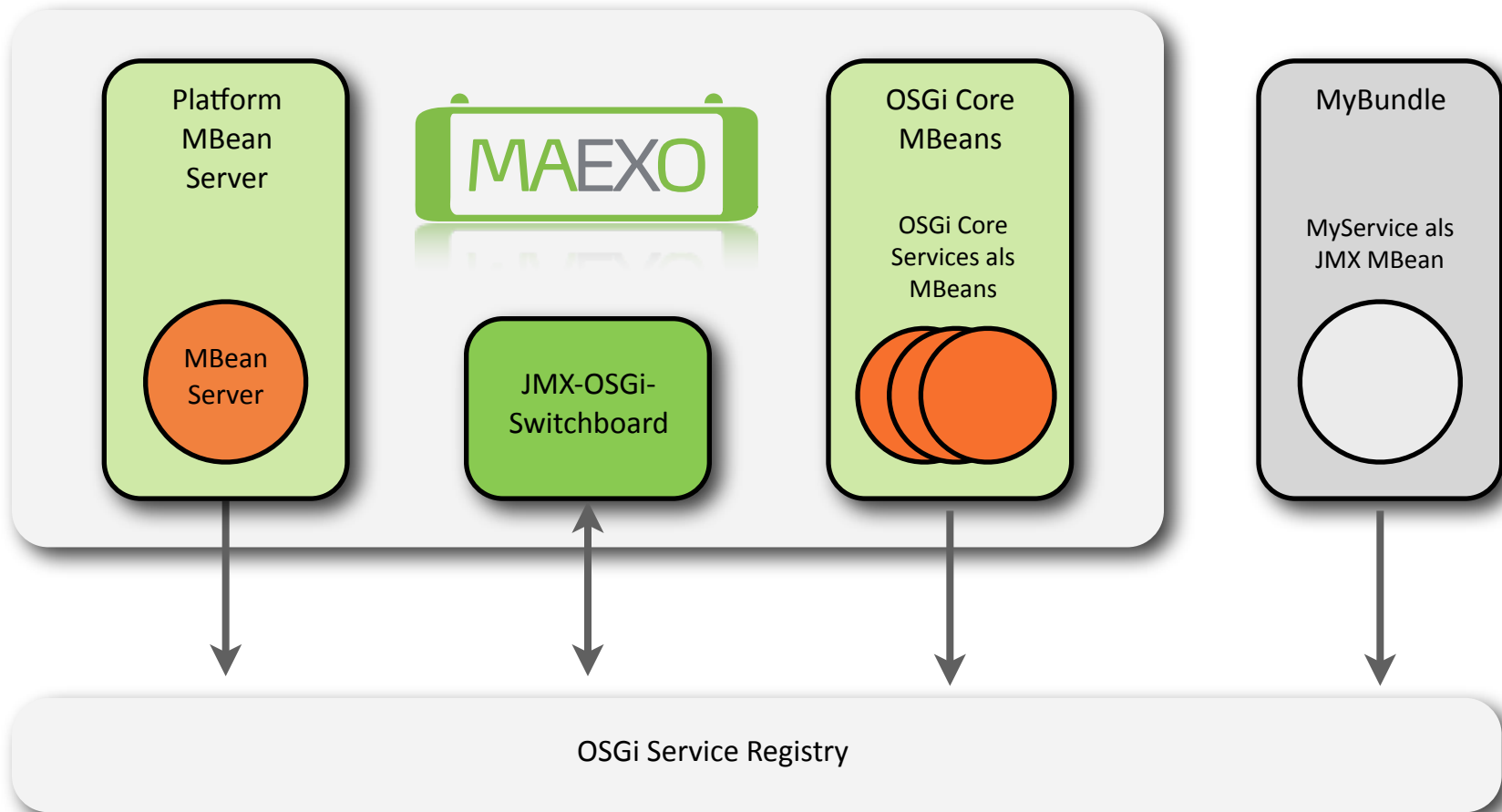
- ❑ MAEXO = Management Extensions for OSGi
- ❑ Projektstart 2008
- ❑ Open Source (Apache License)
- ❑ April 2009: Veröffentlichung Version 1.0
- ❑ aktuell: Version 1.1

□ MAEXO bietet ...

- JMX zum Management von OSGi-Applikationen
- Einfaches, schlankes Framework mit wenig Abhängigkeiten zu Drittbibliotheken
- Anschluss generischer Managementkonsolen an OSGi-basierte Applikationen ermöglichen
- Bereitstellen eines einfachen Programmiermodells
- Lösen des (aus JMX-Sicht) bestehenden Dynamik-Dilemmas im OSGi-Kontext



□ MAEXO - Architektur



- MAEXO MBean-Server Bundles
 - Bundle für MBean Plattformserver
 - Bundle für Thirdparty MBean Server (z.B. MX4J)
- Das MAEXO Switchboard
 - Dynamisches Auffinden und Registrieren von MBean-Serverinstanzen
 - Dynamisches Registrieren und Entfernen von MBeans
 - **Das Konzept:** MBeans werden nicht direkt am MBean-Server registriert sondern als Service an der OSGi Service Registry

- Das (wirklich einfache) Programmiermodell
 - Exportieren der MBean-Interfaces als OSGi-Service
 - Definition des ObjectName einer MBean als Service-Property

```
/**
 * The bundle activator.
 */
public class Activator implements BundleActivator {

    public final void start(BundleContext bundleContext) throws Exception {

        // register MBean as service

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

        publisherProperties.put("objectName", "com.buschmais.maexo.sample:type=ServicePublisher");

        this.myMBeanServiceRef = bundleContext.registerService(
            ServicePublisherMBean.class.getName(),
            new ServicePublisher(bundleContext),
            publisherProperties)
    }
}
```

...

□ Nutzung des Programmiermodells mit Spring Dynamic Modules

```
<!-- Declare the instance of the MBean implementation. -->
<bean class="com.buschmais.maexo.samples.spring.mbean.Sample" />

<osgi:service interface="com.buschmais.maexo.samples.spring.mbean.SampleMBean">
  <osgi:service-properties>
    <entry key="objectName" value="com.buschmais.maexo.sample:type=SampleMBean" />
  </osgi:service-properties>
</osgi:service>
```

□ Nutzung des Programmiermodells mit OSGi Declarative Services

```
<implementation class="com.buschmais.maexo.samples.ds.mbean.Sample" />
  <property name="objectName" value="com.buschmais.maexo.sample:type=SampleMBean" />
  <service>
    <provide interface="com.buschmais.maexo.samples.ds.mbean.SampleMBean" />
  </service>
```

- Was ist MAEXO noch?
 - Bereitstellen der OSGi Core Services als MBean
 - BundleMBean
 - FrameworkMBean
 - PackageAdminMBean
 - ServiceMBean
 - StartLevelMBean
- Out-of-the-Box MX4J HTTP-Managementkonsole
- MAEXO Commons MBean Support

Demo

MAEXO im Einsatz

Ausblick

The next MAEXO

- Importieren von MBeans über remote MBean-Server Connection
- Weitere OSGi Services als MBean
 - Preferences
 - OSGi Bundle Repository
 - ...
- OSGi R4.2 konforme Implementierung definierter MBeans

Loslegen

Infos zum Durchstarten

- MAEXO bei GoogleCode mit Doku, Screencast, Maven-Repository, ...
 - <http://maexo.googlecode.com>

- OSGi Alliance mit Spezifikationstexten, News, etc.
 - <http://www.osgi.org>

- Java Management Extension bei Oracle
 - <http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/>

Das war's. Vielen Dank!