Gesellschaft für
Informations- und
Kommunikationssysteme mbH

OSGi: Toolunterstützung und Softwareentwicklungsprozess

Thementag OSGi

03.11.2009

Autor:
Thorsten Vogel

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- Wofür wird entwickelt? – Plattformen
- Wie wird getestet? – Testing unter OSGi
- Wie wird gebaut? – Build
- Wie wird deployed? – Auslieferung
- Zusammenfassung & Fazit

Herausforderungen der OSGi-Entwicklung

- **Komponenten Architektur**
  - Interfaces (public), Implementierungen (private)
  - Hollywood Prinzip: Don‘t call us, we will call you!
- **Service Architektur**
  - Publish - Subscribe Modell
  - Abhängigkeiten zu Services programmatisch oder deklarativ
- **Bei OSGi im Konzept „erzwungen"**
  - „Flache“ Classpath-Modelle funktionieren auch, erlauben dann aber nicht die wichtigsten OSGi Features wie z.B. Dynamik und lose Kopplung von Komponenten zu nutzen
  - Nicht abgegrenzter Code verursacht Umstellungsaufwand

Webanwendung ohne OSGi

- **Querschnittsfunktionalität in Libraries (JAR Dateien)**
  - Direkte Nutzung durch Auflösung im Classpath
- **Nur ein Classpath**
  - Reihenfolge entscheidend
- **Keine Dynamik**
  - Bei Austausch einer Komponente oder Library muss die gesamte Anwendung neu gestartet werden (zumeist)
- **Entwicklung, Testing, Packaging und Deployment**
  - Mit den üblichen Verdächtigen (Eclipse oder andere IDE, Maven etc.)

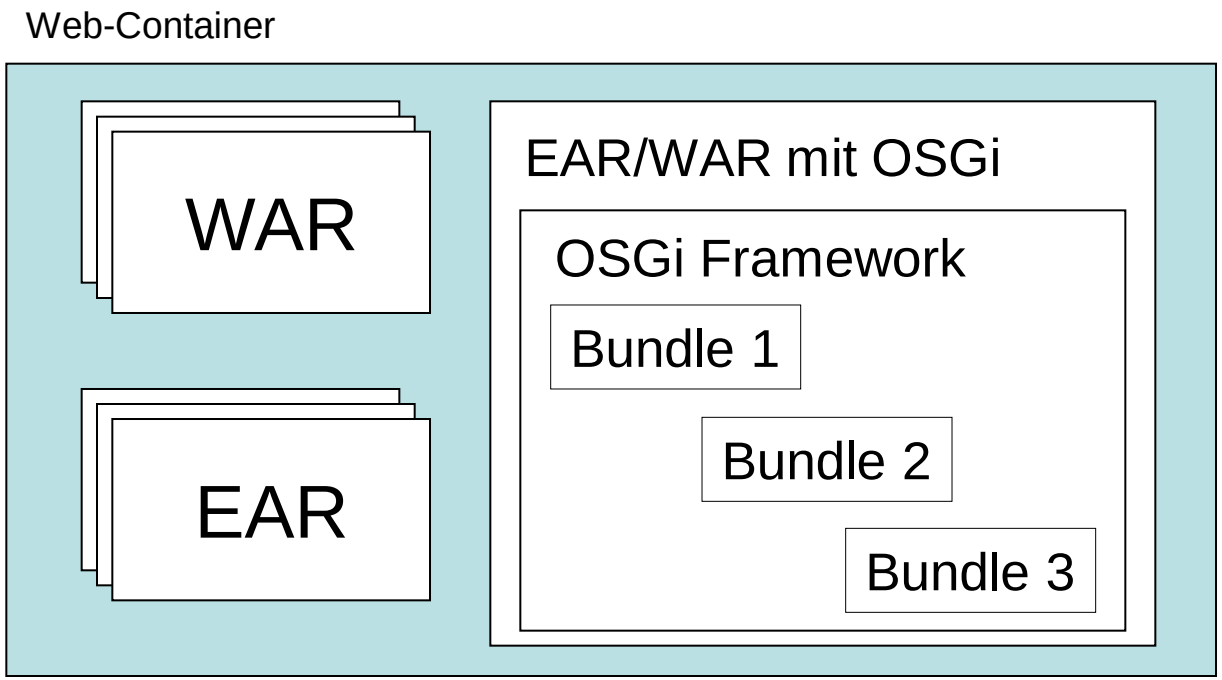
Webanwendung mit OSGi

- **Zwei Arten der Integration**

- **Embedded (In-Container)**
  - OSGi Framework wird innerhalb eines WAR/EAR ausgeliefert und gestartet
  - Nutzung nur durch die entsprechende Web-Applikation

Embedded OSGi Integration

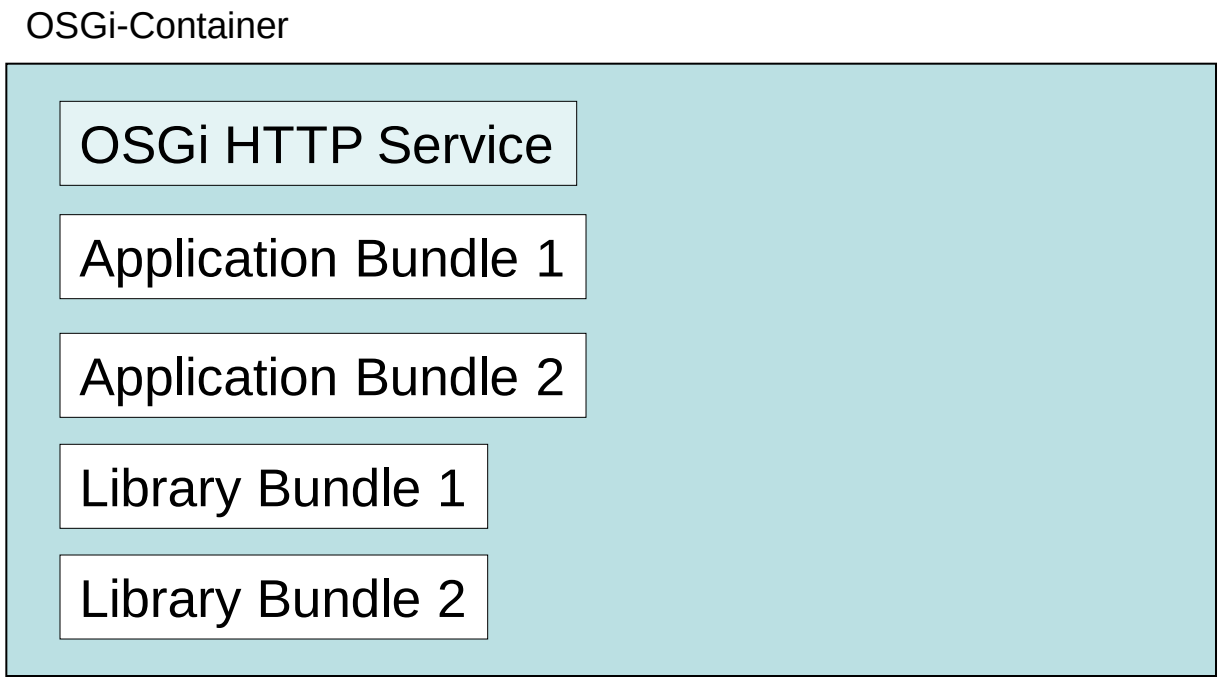
- **(In-Container)**



Reine OSGi Web-Applikation

- OSGi Framework stellt Web-Container zur Verfügung und ist gleichzeitig Ausführungsumgebung der Anwendung
- Konsequente Nutzung aller OSGi Features
- OSGi-embedded Web-Container (Jetty, Tomcat, …)
- Springsource tc/dm Server

Reine OSGi Web-Applikation



Webanwendung mit OSGi

- **Querschnittsfunktionalität als OSGi Services**
  - Bundles stellen Funktionalität bereit, sind aber nicht durch einen statischen Classpath festgeschrieben
  - JAR-Manifest bestimmt Sichtbarkeiten
- **Dynamik**
  - Classpath wird durch das OSGi Framework verwaltet
  - Austausch von Komponenten zur Laufzeit ohne Neustart der Anwendung (zumeist)
- **Entwicklung, Testing, Packaging und Deployment**
  - mit den üblichen Verdächtigen (Eclipse oder andere IDE, Maven etc.) sowie entsprechenden Plugins und Erweiterungen und anderen Tools

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- **Was wird entwickelt? - Bundles**
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- Wofür wird entwickelt? – Plattformen
- Wie wird getestet? – Testing unter OSGi
- Wie wird gebaut? – Build
- Wie wird deployed? – Auslieferung
- Zusammenfassung & Fazit

Was wird entwickelt? – Bundles!

- **Bundles**
  - JAR Dateien (Java-Classes & Resources)

- **Manifest**

```
Export-Package: com.foo.bar
Import-Package: com.baz
etc ...
```

- **Services**
  - Public API Bundle: wird exportiert und kann somit von anderen Bundles genutzt werden
  - (Zumeist) Private Implementation Bundle: kann dynamisch ausgetauscht werden ohne das abhängige Dienste davon etwas mitbekommen
  - Bundles können Services anbieten und selber nutzen

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- **Wie und womit wird entwickelt? - Entwicklungswerkzeuge**
- Wofür wird entwickelt? – Plattformen
- Wie wird getestet? – Testing unter OSGi
- Wie wird gebaut? – Build
- Wie wird deployed? – Auslieferung
- Zusammenfassung & Fazit

Wie wird entwickelt? – Entwicklungswerkzeuge

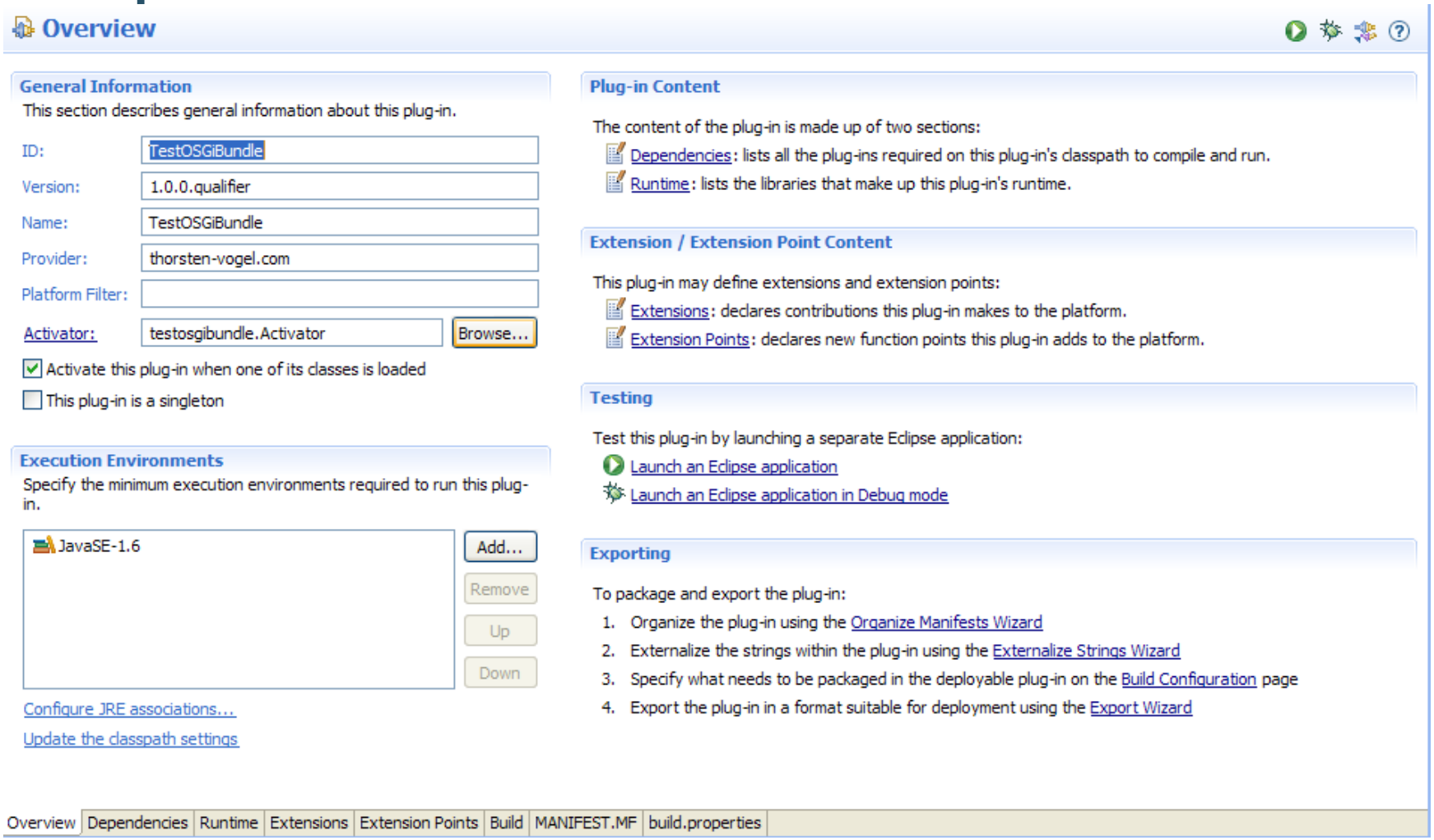
- **BND**
  - Author Peter Kriens (aCute)
  - Erzeugung von OSGi-Bundles
  - Lauffähiges Standalone oder integriert mit Eclipse, Maven und Ant
  - Arbeitet mit sogenannten Specifications (bnd files):
    - `Export-Package: some.service.*`
    - `Private-Package: some.service.impl.*`
    - `Include-Resource: some.resource.*`

- **Bundle Inhalt**
  - `META-INF/MANIFEST.MF`
  - `some/service/Service.class`
  - `some/service/impl/ServiceImplementation.class`
  - `some/resource/Foo.gif`

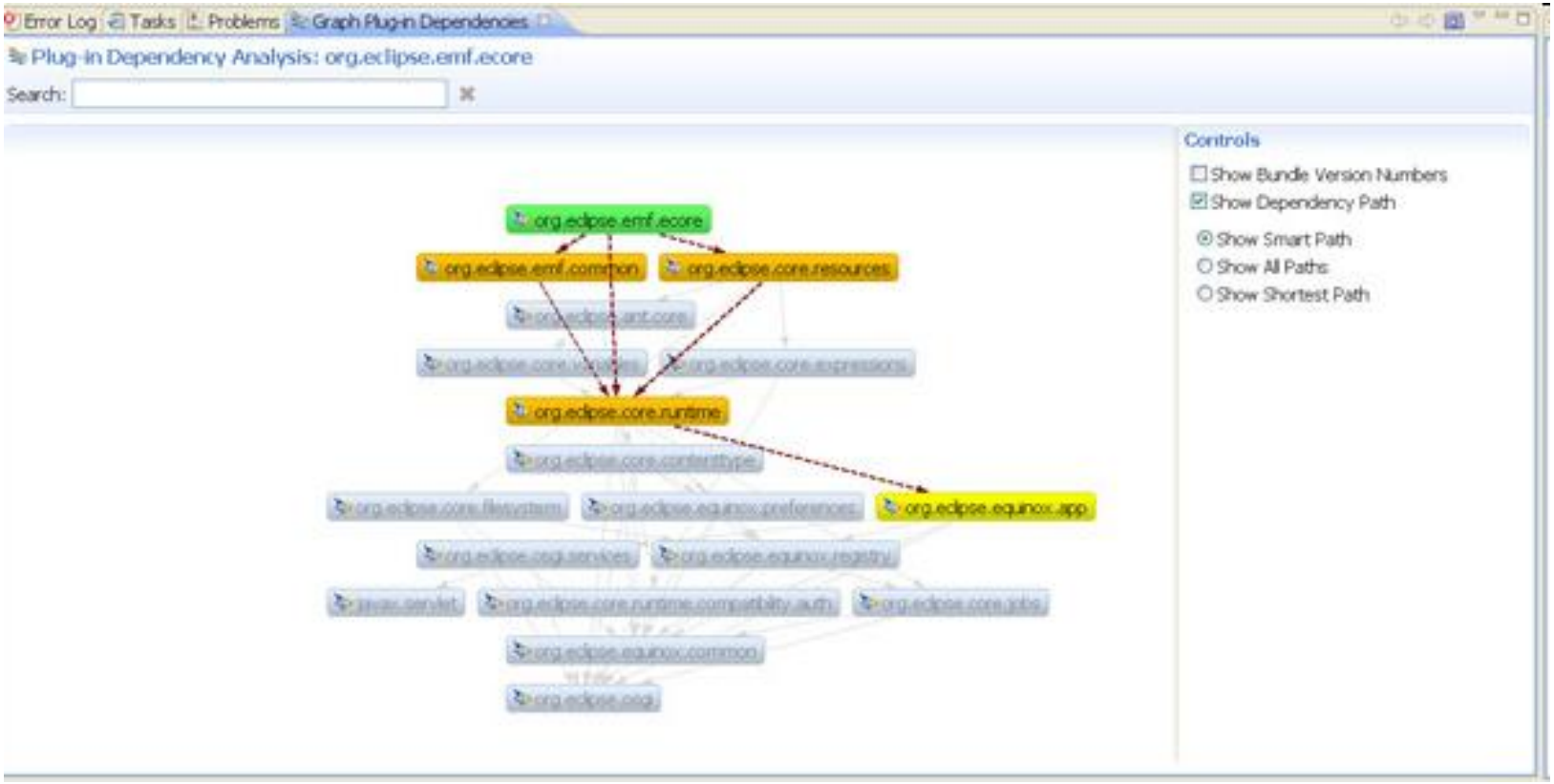
Eclipse PDE

- **Übersicht**
  - Plugin == Bundle (Namensgebung hat historische Gründe)
  - Equinox-Spezifika, aber auch reine OSGi-compliant Bundles möglich
  - Wizard-getriebene Erstellung von Plugin-Projekten, reiche Auswahl an Templates für Eclipse-Erweiterungen
  - Run-Configurations zur Ausführung von Code
- **Bearbeitung der MANIFEST Dateien**
  - Volle Konfiguration über eine schicke Oberfläche
  - Bezieht Informationen aus dem Project-Classpath und den Project-Dependencies
- **Erzeugung von Bundles**
  - Build-Prozess ist integriert und konfigurierbar
  - Deployment ist Eclipse-spezifisch, d.h. kein automatisches Deployment in 3rd Party Container (hierzu sind Erweiterungen notwendig)

Eclipse Manifest Editor



Dependency Visualization (Incubator)



Integration von Laufzeitumgebungen

- **Integration von Laufzeitumgebungen in die IDE**
  - Develop-Deploy-Test-Cycle wird vereinfacht
  - Direktes Deployment in den Container der Wahl
  - (Remote-) Debugging wird ermöglicht
- **…über spezifische Plugins**
  - Z.B. Apache Felix Eclipse Plugin
  - Springsource Tools Suite
- **…oder über Pax Cursor / Pax Runner**
  - Erlaubt, unterstützt OSGi Plattformen in Eclipse zu integrieren
  - Deploy, Run & Debug von Bundles
  - Momentan Felix, Equinox, Knopflerfish, Concierge

Integration von Laufzeitumgebungen #2

- **Apache Felix Eclipse Integration**
  - Felix wird in Eclipse ausgeführt
  - Konsole steht zur Verfügung (install/start/stop etc.)

- **Springsource Tool Suite (STS)**
  - Springsource dm und tc Integration
  - Basiert auf Equinox, Container sind jedoch weitestgehend austauschbar
  - Automatische OSGi MANIFEST Erzeugung und Validierung
  - Dependency Graph

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- **Wofür wird entwickelt? – Plattformen**
- Wie wird getestet? – Testing unter OSGi
- Wie wird gebaut? – Build
- Wie wird deployed? – Auslieferung
- Zusammenfassung & Fazit

Wofür wird entwickelt? – Plattformen

- **Apache Felix**
  - Gestartet 2006, nicht-kommerzielles Open-Source Projekt
  - Ziel ist die vollständige Einhaltung der OSGi R4.x Spezifikation
  - Sehr aktive Community
  - Wird mit Sun‘s GlassFish ausgeliefert

- **Knopflerfish**
  - Gestartet 1999, Contributor zu den ersten OSGi Spezifikationen
  - Auch Open-Source (BSD), kommerzieller Hintergrund (Makewave)
  - Pro-Variante der Plattform im Angebot

- **Equinox**
  - Gestartet 2003
  - Ist Eclipse-Basis (seit Version 3), dadurch sehr weit verbreitet
  - Viele nicht-standardisierte Erweiterungen

Wofür wird entwickelt? – Plattformen #2

- **Concierge**
  - Hoch-Optimierte R3 Implementierung
  - 80kb Runtime, optimal für Kleingeräte
- **Prosyst**
  - Kommerzieller Anbieter
  - mBedded OSGi Server, auch für Android
- **Oscar**
  - Gestartet 2005
  - Seitdem keine Releases mehr
  - Proof-of-Concept Implementierung der heutigen Spec-Leads
- **und viele weitere…**

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- Wofür wird entwickelt? – Plattformen
- **Wie wird getestet? – Testing unter OSGi**
- Wie wird gebaut? – Build
- Wie wird deployed? – Auslieferung
- Zusammenfassung & Fazit

Wie wird getestet? - OSGi Testing

- **Unit-Testing auf Bundle-Ebene**
  - Wie gehabt: JUnit et al.
  - Mocking von Services (Mockito et al.)
- **Integrationstests mit dem Container**
  - Notwendig, um Abhängigkeiten und Kommunikation von Bundles zu prüfen
  - Werden alle Verträge eingehalten und gibt es Seiteneffekte?
- **Begriff: Provisioning**
  - Bereitstellung von Bundles in eine Laufzeitumgebung (OSGi Container)
- **PAX Exam / PAX Runner**
  - Viele verschiedene vorgegebene Laufzeitumgebungen

Integrationstests unter OSGi

- **Programmierung der Tests im Prinzip wie Unit-Tests**
  - `org.ops4j.pax.exam.junit.JUnit4TestRunner`
  - `felix().version("1.8.1");`
  - `provision(bundle(„file://bundle.jar“));`
- **Annotation-getriebene Vorgehensweise**
  - `@Configuration` – Definiert eine Methode als Konfiguration
  - `@Inject` – Injiziert Abhängigkeiten
- **Beispiel:**

Beispiel für einen OSGi Integrationstest

```
@RunWith(JUnit4TestRunner.class)
public class MyJUnitTest {

   @Inject
   BundleContext bundleContext;

   @Configuration
   public static Option[] configuration() {
      return options(frameworks(
                                          felix(),
                                          equinox(),
                                          knopflerfish()));
   }

   @Test
   public void testMethod() throws Exception {
      System.out.println(bundleContext
            .getProperty(Constants.FRAMEWORK_VENDOR));
   }
}
```

Integrationstests unter OSGi

- **Ablauf eines Integrationstests mit PAX Exam**
  - 1. Start des OSGi Containers
  - 2. Provisioning der Bundles
  - 3. Dynamische Erstellung eines Test-Bundles
  - 4. Deployment in den Container
  - 5. Ausführen der Testmethoden

- **Volle Debugging Möglichkeiten mit Eclipse Bordmitteln**

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- Wofür wird entwickelt? – Plattformen
- Wie wird getestet? – Testing unter OSGi
- **Wie wird gebaut? – Build**
- Wie wird deployed? – Auslieferung
- Zusammenfassung & Fazit

Wie wird gebaut? – OSGi Buildprozess

- **Automatisierung des Softwarebuilds**
  - Außerhalb der IDE mit
  - Maven
  - Gradle
  - et al.
- **Continuous Integration**
  - Fortlaufender Build bei Änderungen am Source-Code, stetige Rückmeldung von evtl. Problemen
- **Automatisierung der Unit- und Integrationstests**
  - Maven Plugins z.B. von Apache Felix
  - `maven-bundle-plugin` (nutzt bnd tool)

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- Wofür wird entwickelt? – Plattformen
- Wie wird getestet? – Testing unter OSGi
- Wie wird gebaut? – Build
- **Wie wird deployed? – Auslieferung**
- Zusammenfassung & Fazit

Wie wird deployed? – OSGi Deployment

- **OBR – OSGi Bundle Repositories**
  - Halten Bundles in maschinenlesbarer Struktur vor
  - Können durch automatisierte Buildprozesse befüllt werden
  - Siehe maven-bundle-plugin deploy goal
  - Open Source OBRs existieren, z.B. von Springsource und Codehaus
- **Synchronisation der Entwicklung im Team**
  - Über OBR werden beim lokalen Build Abhängigkeiten automatisch gezogen
- **Verwaltung von Bundle-Versionen**
  - Jede Version eines Bundles hat ein definiertes Ablageverzeichnis

Wie wird deployed? – OSGi Deployment #2

- **Embedded OSGi**
  - Z.B. Eclipse RCP
  - Springsource tc Server
  - Oder beliebige andere Applikationserver

- **OSGi Container**
  - Z.B. Springsource dm Server, Apache Felix, etc.
  - Über Konsole oder OSGi Management Agent Implementierungen mit GUI
  - Springsource dm: eigenes Packaging Format: PAR (Platform Archive)
  - Beinhaltet Kollektion von Bundles und Resources (vgl. WAR/EAR)
  - Austausch von einzelnen aktualisierten Bundles zur Laufzeit des Servers

Agenda

- Herausforderungen der OSGi Entwicklung: Beispiel Webanwendung
- Was wird entwickelt? - Bundles
- Wie und womit wird entwickelt? - Entwicklungswerkzeuge
- Wofür wird entwickelt? – Plattformen
- Wie wird getestet? – Testing unter OSGi
- Wie wird gebaut? – Build
- Wie wird deployed? – Auslieferung
- **Zusammenfassung & Fazit**

Zusammenfassung

- **Es werden Komponenten entwickelt**
  - Weitreichende Auswirkungen auf Architekten und Entwickler
- **OSGi Toolunterstützung durch weitverbreitete IDEs**
  - Durch Plugins
  - Eclipse PDE
  - Springsource ToolSuite
- **Test-Unterstützung (und mehr) durch PAX Toolchain**
- **Build-Unterstützung durch Maven-Plugin**
  - Aus dem Apache Felix Projekt

Fazit

- **OSGi-Konzepte müssen klar verstanden und umgesetzt werden**
- **Relativ steile Lernkurve**
- **Einige neue Plugins und Tools müssen gelernt werden**
- **Starke Veränderungen an Tools, teilweise wöchentlich neue Releases**
- **Einarbeitungszeit und Aktualisierungsaufwand ist signifikant und sollte eingeplant werden**

Weiterführende Literatur

- **Tim O'Brien, Stuart McCulloch, Maven Handbook**
  http://www.sonatype.com/books/mcookbook/reference/index.html
- **Springsource Tool Suite**
  http://www.springsource.com/products/sts
- **Apache Felix**
  http://felix.apache.org
- **Knopflerfish**
  http://knopflerfish.org

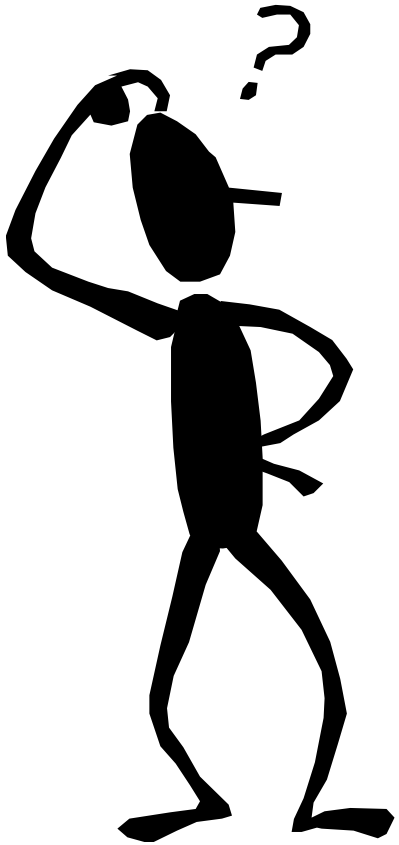
Weiterführende Literatur

- **BND Tool**
  http://www.aqute.biz/Code/Bnd
- **PDE Dependency Visualization**
  http://www.eclipse.org/pde/incubator/dependency-visualization/index.php
- **Maven**
  http://maven.apache.org/
- **Gradle**
  http://www.gradle.org/

Weiterführende Literatur

- **PAX Tools**
  http://www.ops4j.org
- **PAX Runner**
  http://paxrunner.ops4j.org/space/Pax+Runner
- **PAX Exam**
  http://paxrunner.ops4j.org/display/paxexam

Fragen?



Gesellschaft für
Informations- und
Kommunikationssysteme mbH

www.iks-gmbh.com