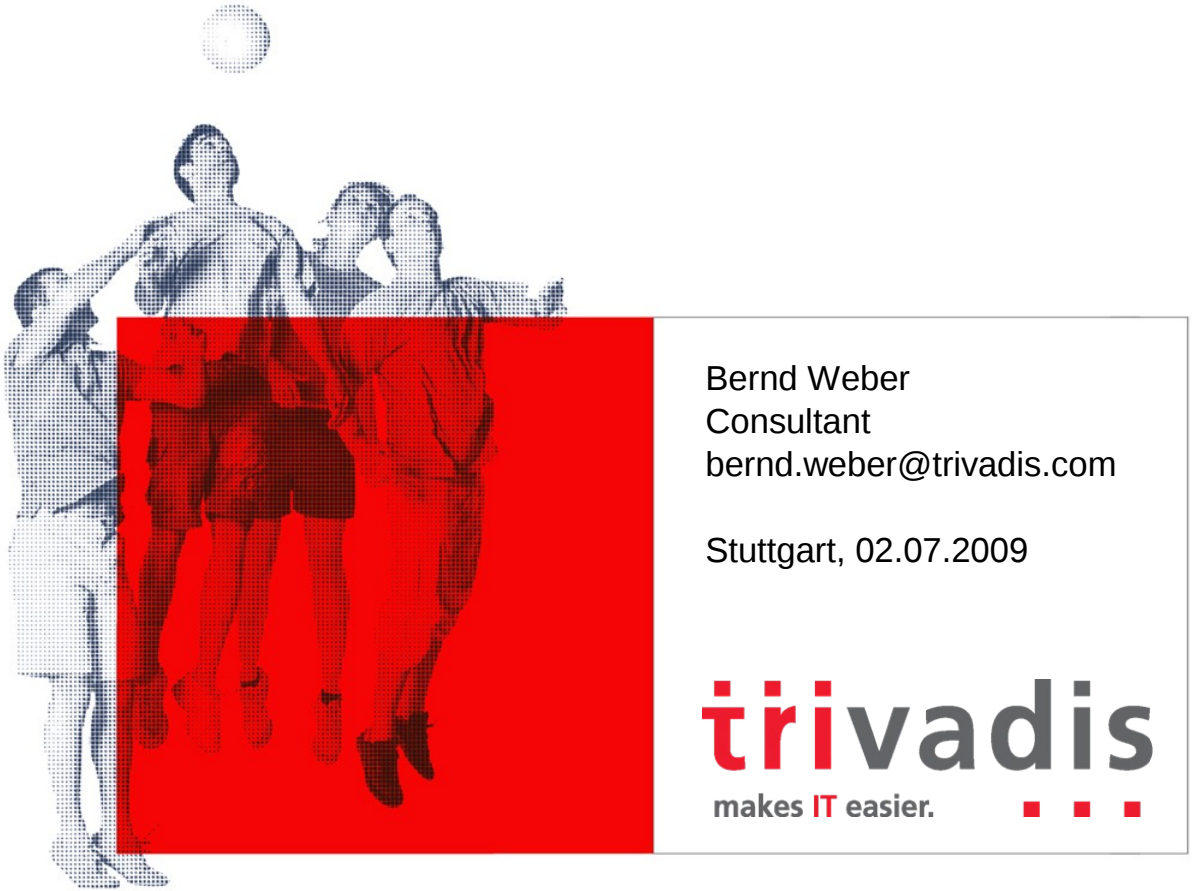
OSGi in der Praxis oder „Wie fange ich an?“



Bernd Weber
Consultant
bernd.weber@trivadis.com

Stuttgart, 02.07.2009

trivadis
makes IT easier.

Agenda

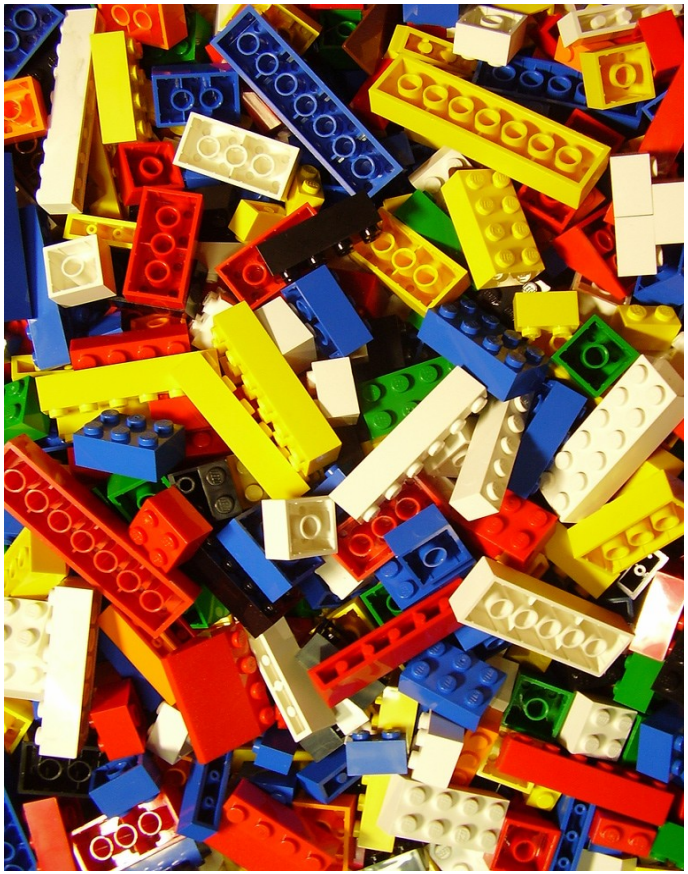
- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

Agenda

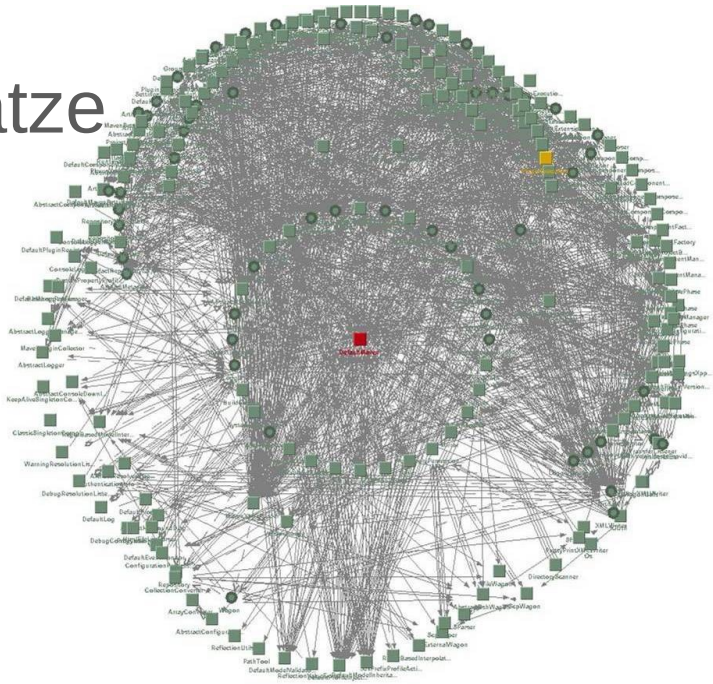
- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

Modularisierung

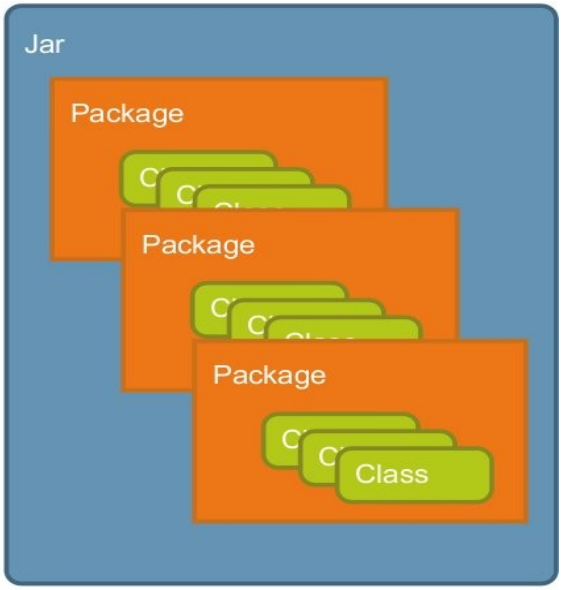
- Recycling über dreißig Jahre alter Ideen:
  - Hohe Kohäsion
  - Schwache Kopplung
- Modularisierung minimiert die Komplexität durch die Festlegung klarer Grenzen zwischen den Bestandteilen eines Systems
- Vernünftig modularisierte Systeme sind leichter wart- und erweiterbar
- Änderungen sind besser lokalisierbar und haben weniger Seiteneffekte auf das Gesamtsystem



Modularisierung – bisherige Ansätze



- OO fokussiert auf Kapselung von Instanzvariablen – zu kleine Granularität
- OO Systeme verflechten schnell - das Programm kennt seine eigene Struktur
- Entwurfsmuster wie SOA, Factories, DI minimieren u.a. die OO-Nachteile
- Java bietet Quasi-Modularität mit Sichtbarkeitsregeln bis zum Package
- Java Archive haben keine modularisierenden Eigenschaften
- CP: Wer zuerst kommt, mahlt zuerst
- Kein standardisiertes Kollaborations- und Erweiterungsmodell

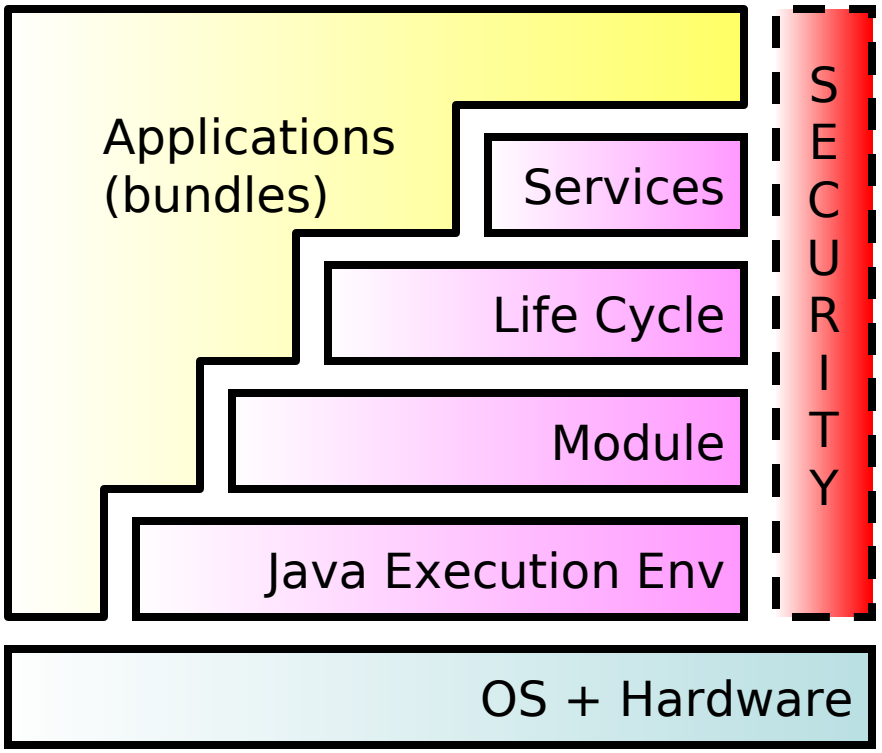


Agenda

- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

Architektur des OSGi-Frameworks - Überblick

- Die OSGi Service Platform spezifiziert eine modulare Architektur für dynamische, komponentenbasierte Systeme
  - Execution Environment
  - Module Layer
  - Life Cycle Layer
  - Service Layer
  - Security
- Führt den Begriff *Bundles* für Module ein



Architektur des OSGi-Frameworks - Eigenschaften

- Bundle-spezifisches Laden von Klassen
  - Beseitigt viele Probleme bei aufgeteilten (split) Packages
  - Klassen werden in großen Systemen schneller geladen
- Erlaubt verschiedene Versionen derselben Klasse in einer VM
  - Class spaces
- Bundles können Packages exportieren oder für privat erklären
- Services bieten ein Modell der Zusammenarbeit in einer VM analog zu einer SOA („in-VM SOA“)
  - Kein Aufruf-Overhead
  - Full Life Cycle Model

Dynamische Eigenschaften in OSGi-Applikationen

- Auslieferungs-Einheit:
  - Bundle = JAR + Manifest-Einträge
- Unterstützt dynamische Szenarien
  - Aktualisierung
  - Installation/Deinstallation
  - Start/Stop
- Bundles können Abhängigkeiten z.B. zu Packages oder Diensten haben
- Diese Abhängigkeiten müssen hinsichtlich der möglichen Dynamik berücksichtigt werden!

Vorteile von OSGi

- Anforderungen können in Bundles aufgeteilt weden
- Die lose Kopplung über Services ermöglicht die Wiederverwendung von Standardkomponenten
- Bundles sind „selbständiger“ und daher besser statisch testbar
- Die Ausrichtung auf Dienste vereinfacht das Testen zusätzlich
- OSGi zwingt zur Disziplin ;-)

Agenda

- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

OSGi Implementierungen und Hersteller

Gatespace Telematics

SIEMENS

SAMSUNG

IBM

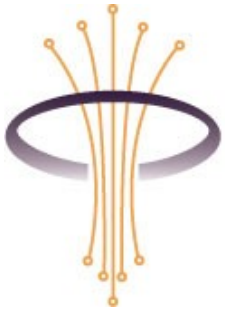
espial

ProSyst

APACHE felix

HITACHI

eclipse



OSGi™ Alliance

makewave

Open Source Implementierungen:

- Freie Apache-Implementierung der OSGi R4 Service Platform
  - Enthält Kern des OSGi-Frameworks und Standard-Services
  - Derzeit sind große Teile der OSGi R4 Spezifikation implementiert
  - An den noch fehlenden Teilen wird intensiv gearbeitet
  - Framework-Funktionalität von Felix ist sehr stabil
- Bietet weitere interessante OSGi-bezogene Technologien, z.B.
  - iPOJO – DI Framework mit vielen Schnittstellen
- Wird zunehmend als Basis anderer OS-Projekte verwendet:
  - Apache ServiceMix (Enterprise Service Bus, Kernel based on Felix)
  - Apache Tuscany (Open Source implementation of SCA)
  - Application servers GlassFish, OW2 JonAS
  - PAX-Runner

Open Source Implementierungen: Equinox

- Ziel: Erstklassige OSGi Community, die die Vision von Eclipse als einem Zusammenspiel von OSGi-Bundles fördert
  - OSGi Framework-Implementierung für alle Bereiche von Eclipse
  - Implementierung aller Aspekte der OSGi Spezifikationen (incl. Mobile, Home und Vehicle)
- Optionale Erweiterungen außerhalb des OSGi-Standards
  - Buddy-Class-Loading, Extension Points
- Populärste OSGi-Implementierung
- Projekte und Produkte auf Basis von Equinox
  - Eclipse und alle seine Derivate
  - Open-Xchange
  - Produkte von Iona, Innoopract, ProSyst, ...

Open Source Implementierungen:

- Ziel: Einfach verwendbarer OS Code, Build tools und Anwendungen mit Bezug zur OSGi Technologie
  - Vollständige Umsetzung aller Pflichtteile von OSGi R4 (Core, Service Compendium)
  - Bei den optionalen Teilen fehlen noch ein paar Sicherheitsaspekte
  - Makewave stellt einige Entwickler für Pflege und Weiterentwicklung
- Viele Werkzeuge und Komponenten über den Standard hinaus
  - Desktop zur Verwaltung eines laufenden OSGi-Frameworks
  - Verschiedene Konsolen (TTY, TCP, Log and Framework commands, ...)
- Erste OpenSource OSGi Framework Implementierung
- Knopflerfish Pro mit kommerziellem Support und weiteren Tools

Agenda

- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

Migration - Legacy Code

... legacy code is a challenge. Many developers say things like: “My code is very modular”, or “My code doesn’t depend on very much”, or “No one uses any of my classes except from the Foo package”.

Unless they are already using OSGi, they are wrong. Until modularity is enforced, ***it is not there****.*

John Wells, BEA

Migration – Bibliotheken zu Bundles

- Analyse
  - Abhängigkeiten zwischen den Jars Ihrer Applikationen
  - Enthaltene Jar-Doubletten
  - Erweiterungsmechanismen mit dynamischem Laden von Klassen

- OSGi Bundles brauchen Manifest-Einträge
  - Exportierte Packages
  - Importierte Packages
  - Optionalität
  - Version
  - Bundle Identität
  - ...

Migration - Bibliotheken zu Bundles

- Verschiedene OS Projekte bieten OSGi-Metadaten:
  - Apache (Derby, Struts, Felix, etc.)
  - Alle Eclipse-Plugins
  - Codehaus Groovy
- Es gibt immer mehr OSGi-Repositories:
  - OSGi Bundle Repository (OBR)
  - Apache Felix Commons
  - Eclipse Orbit
  - SpringSource Enterprise Bundle Repository
  - OSGi-Möglichkeiten in Maven (z.B. Maven Bundle plugin)
- Falls alles nicht hilft: bnd Utility
  - OSGi Bundle Analyse- und Erstellungswerkzeug ...

Migration: Eigenschaften eines Bundles

- Ein Bundle ist eine auslieferungsfähige „Applikation“
  - Ähnlich einer Windows EXE
  - Manifestation als erweiterte JAR-Datei
- Ein Bundle benennt seine Abhängigkeiten vollständig
- Ein Bundle registriert 0-n Dienste
  - Ein Dienst wird durch ein Java-Interface spezifiziert und kann in verschiedenen Bundles implementiert werden
  - Ein Dienst ist mit dem Lebenszyklus des umgebenden Bundles verknüpft
- Über Abfragemechanismen können von anderen Bundles registrierte Dienste gefunden und genutzt werden

Migration - Bibliotheken in Bundles auslagern

- Neues Projekt mit allen Applikations-JARs und Resourcen
  - Bibliotheken im Classpath berücksichtigen
- BundleActivator erstellen, der die main() aufruft
- Alle Bibliotheken in ein einziges Bundle packen
  - Wird ziemlich groß...
  - Verwenden Sie bnd
- Ein funktionierendes Bundle (egal wie groß) ist eine gute Basis zum schrittweisen Ersetzen der enthaltenen Jars durch Bundles
  - Sollte nach jeder Änderung funktionieren
  - Abwägung zwischen Einbindung und Referenzierung
  - Bereits vorhandene Bundles verwenden

Migration - Dynamisches Laden von Klassen

- Viele Anwendungen haben ihr eigenes Plugin-Konzept
  - Konfiguration durch Strings
  - Class.forName zur Laufzeit
- Modularisierung wird ignoriert...
  - Klassennamen haben keine Version
- Daher:
  - Prüfen, ob dynamisches Klassenladen wirklich notwendig ist
  - Umwandeln des Plugin-Mechanismus in Services, die mit Versionen und Kompatibilitätsaspekten umgehen können
  - Wenn es gar nicht anders geht, ClassLoader.loadClass verwenden statt Class.forName (Bug!)
  - Am Besten: Weg mit allen Class loaders - OSGi kann es bestimmt besser! :-)

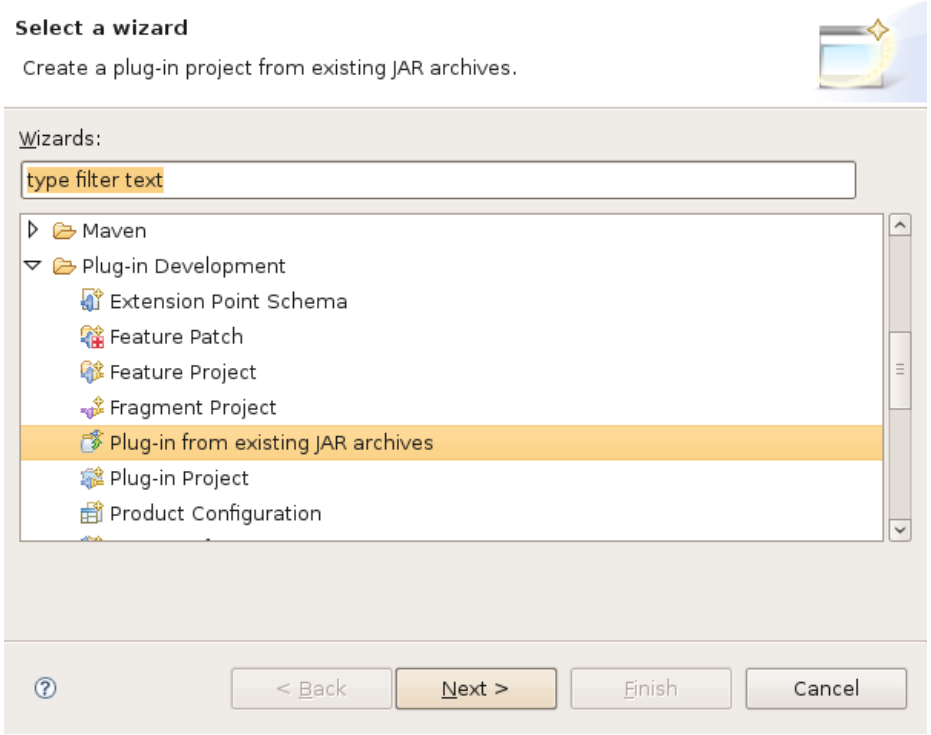
Migration – Beispiel mit bnd-Steuerdatei

- hibernate3.jar wird zum Bundle mittels org.hibernate.bnd:

```
-classpath: hibernate3.jar, \
    lib/antlr-2.7.6.jar, \
    lib/asm-attrs.jar, \
    lib/asm.jar, \
    lib/cglib-2.1.3.jar, \
    lib/commons-collections-2.1.1.jar, \
    lib/commons-logging-1.0.4.jar, \
    lib/dom4j-1.6.1.jar, \
    lib/log4j-1.2.11.jar, \
    lib/jta.jar
Private-Package: *
Import-Package: javax.xml.*, javax.sql.*, \
                *;resolution:=optional
Export-Package: javax.transaction.*,
                org.hibernate.*
```

Migration – Verwendung von Eclipse PDE

- Neues PDE-Projekt auf Basis vorhandener JAR-Datei
- Bibliotheken hinzufügen
- Header-Einträge des OSGi-Bundles definieren (mindestens BSN+Version)
- Mittels Export-Wizard als Bundle exportieren



Agenda

- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

Build-Prozess - Grundsätze

- Eclipse ist eine exzellente IDE für die OSGi-Entwicklung
  - PDE - Plugin Development Environment, Pax-Runner, JDT + Bnd
- Bundles können aber auch mit anderen IDEs wie Netbeans entwickelt werden
- Ein automatisiertes Build-System funktioniert allerdings anders
  - Headless PDE ist einigermaßen schwierig aufzusetzen
- Alternative Build-Systeme
  - Maven + Bundle Plugin (from Apache Felix, based on bnd)
  - Ant + bnd
- Gute Unterstützung für neue Bundles durch Pax-Construct

Build: Beispiel-Apps mit Pax-Construct

- Neues OSGi Projekt erstellen
  - > pax-create-project -g examples -a test
- Beispiel-Bundle im neuen Projekt erstellen
  - > cd test
  - > pax-create-bundle -p org.example.pkg -n test.bundle
- Bauen, deployen und mittels Felix starten
  - > mvn clean install pax:provision
- Eclipse-Projekt erzeugen
  - mvn pax:eclipse
- Projekt in Eclipse importieren
- Spring-DM Demo analog...

Build – Steuerungsinformationen

- Verwendung von bnd-Dateien (s. Hibernate-Beispiel)
  - Unmittelbare Verwendung des Basis-Werkzeugs
  - Viele Konfigurationsmöglichkeiten incl. Macros/Variablen
  - Hierarchische Strukturierung möglich
- Verwendung von pom.xml
  - Einbindung des Maven-Bundle-Plugin (basiert auf bnd)
  - Viele vernünftige Standard-Einstellungen
  - Einbindung von bnd-Dateien möglich
- Vorhandene Manifest-Datei kann eingebunden werden
  - Ergebnis aus Migration per Eclipse-PDE nutzbar

Build – Maven und Eclipse PDE

- Maven als Build-Tool der Wahl
  - Unzählige Plugins
  - Bestens für Continuous Integration geeignet
  - Ist aber zunächst nur ein Kommandozeilen-Werkzeug
- maven-eclipse-plugin kann PDE-Projektnatur erzeugen

```
<plugin>
    <artifactId>maven-eclipse-plugin</artifactId>
    <configuration>
        <pde>true</pde>
```

- Anpassung des maven-bundle-plugins wegen Manifest-Datei

```
<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <extensions>true</extensions>
    <configuration>
        <!-- Hier erwartet Eclipse das Plugin-Manifest für die PDE-Umgebung -->
        <manifestLocation>META-INF</manifestLocation>
        <instructions>
```

Build – Maven und Eclipse PDE

- Noch eine Anpassung des maven-bundle-plugins erzeugt schließlich die Manifest-Datei

```
<!-- PDE-Manifest erzeugen in "process-classes"-Phase -->
<executions>
    <execution>
        <id>bundle-manifest</id>
        <phase>process-classes</phase>
        <goals>...
            <goal>manifest</goal>
        </goals>...
    </execution>
</executions>
</plugin>
```

- Bug: Leerzeile zwischen Manifesteinträgen führt zu PDE-Fehler

Agenda

- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

Deklarative Ansätze - Kernaussagen

- Reine OSGi APIs sind mächtig, aber nicht immer einfach zu verwenden
- Am Besten schreibt man POJOs (Plain Old Java Objects) ohne Bezug zu einem Framework
- Vorsicht bei extrem Performance-kritischen Anwendungen
- Es gibt einige deklarative Applikationsmodelle für OSGi:
  - OSGi Declarative Services
  - Spring-DM (Dynamic Modules)
  - Apache iPOJO
- Suchen Sie sich das für Sie am besten geeignete Modell aus, NACHDEM Ihre bestehende Anwendung als Bundle läuft

Deklarative Ansätze – OSGi Declarative Services

- Teil des OSGi Standards im Service Compendium
- Ausgereifte Technologie ähnlich zu Spring-DI
- Konfiguration einzelner Komponenten durch jeweilige XML-Dateien unter OSGI-INF-Verzeichnis
- Bundle-Header Service-Component enthält die XML-Dateien
- Setzen/Entfernen von Referenzen mit entsprechenden Methoden
- Angabe von Kardinalitäten, Konfiguration etc.
- Leichtgewichtige Umsetzung der Inversion of Control

Deklarative Ansätze - Spring-DM

- Konfiguration wie in Spring üblich mit XML-Dateien, die den Application-Context beschreiben
- Verweis auf Spring-Konfiguration in speziellem Bundle-Header Spring-Context oder Dateien in META-INF/spring

```
Spring-Context: config/osgi-*.xml;wait-for-dependencies:=false
```

- Services werden via <osgi:service ... /> publiziert (publish) und
  - via <osgi:reference ... /> verwendet (consume)
- Mittel der Wahl, wenn Spring-Umfeld gesetzt ist
- Benötigt diverse Bundles (Spring-Core, Spring-DM, ...)
- Aufpassen bei Abhängigkeiten zwischen XML-Dateien!

Deklarative Ansätze – Apache iPOJO

- Modernster Ansatz
- Basiert auf Annotations

```
@Component
@Provides
public class MyServiceImplementation implements MyService {
        ...
}
```

- Bietet als Framework viele Erweiterungen z.B. zur Nutzung durch JMX oder zur Persistierung
- Komponenten können deklarativ per OSGi ConfigurationAdmin konfiguriert werden und auf OSGi-Events reagieren
- Wird durch ein einziges Bundle von 205 kB ermöglicht
  - Keine Abhängigkeit von weiteren Bibliotheken/Bundles
- Viele Tools (Maven- und Eclipse-Plugin, ...), gut erweiterbar
- Sauschnell!

Agenda

- Modularisierung
- Architektur des OSGi-Frameworks
- OSGi Implementierungen
- Migration bestehender Applikationen
- Der Build-Prozess
- Deklarative Ansätze
- OSGi Application Server

OSGi Application Server – SpringSource dm Server

- Vollständig OSGi-basierter JEE-Server
  - Laut Hersteller „neues Maß an Flexibilität“
- Eclipse-Plugins in Community Edition
  - Eigene IDE auf Eclipse-Basis in kommerzieller Version
- Basiert auf Eclipse Equinox und Apache Tomcat-Bundle
- Veröffentlichung im Oktober 2008
- Erweiterung des OSGi-Standards um Abhängigkeitsdefinitionen zu Bibliotheksmengen (Import-Library & Import-Bundle)
- Basis ist das SpringSource Enterprise Bundle Repository

OSGi Application Server – Weitere Ansätze

- Jboss 5
- IBM Websphere
- Oracle Weblogic Event Server
- SUN Glassfish
- OW2 JonAS
- SAP plant OSGi als Basis für die nächste NetWeaver-Plattform
  (http://www.heise.de/open/Open-Service-Gateway-10-Jahre-OSGi-Alliance--/news/meldung/132605)

OSGi - Zukünftige Entwicklung

- Die OSGi Alliance hat eine “Enterprise Expert Group” berufen. Deren Themen sind u.a.:
  - Distribution / Verteilte Systeme
  - Service Registry Extensions
  - Gemeinsames Java-Komponentenmodell (JSR 294) in Java 7 (?)
- Bundle-Bee: OSGi-basiertes Grid-Computing
- Zunehmende Unterstützung, mehr Implementierungen und Werkzeuge

Quellen / Links

- Vorträge
  - Peter Kriens auf OOP 2007, „kriens Mi6-4 WR.pdf“
  - Peter Kriens auf OOP 2008, „Kriens Do1-1 WRe 2up.pdf“
  - Peter Kriens auf OOP 2009, „Kriens Do8-3 2FS Large Applications.pdf“
  - Kai Toedter et al. auf OOP 2009, „Wuetherich Lippert Toedter a4 OSGi Applications.pdf“

- Links

http://www.osgi.org	Die OSGi-Homepage
http://www.aqute.biz	Peter Kriens, OSGi Cheftechnologe
http://felix.apache.org	Apache Felix
http://www.eclipse.org/equinox	Eclipse Equinox
http://www.knopflerfish.org	Knopflerfish
http://static.springframework.org/osgi	Spring-DM API, Doku, Downloads
http://www.ops4j.org	PAX Runner uvm.

- OSGi Demos
  - http://max-server.myftp.org/trac/pm Dynamische Spring-Demo
  - http://svn.apache.org/repos/asf/felix/trunk/examples/servicebased.* Felix GUI Demo

Vielen Dank!