Gesellschaft für
Informations- und
Kommunikationssysteme mbH

OSGi Konzepte

Thementag OSGi

03.11.2009

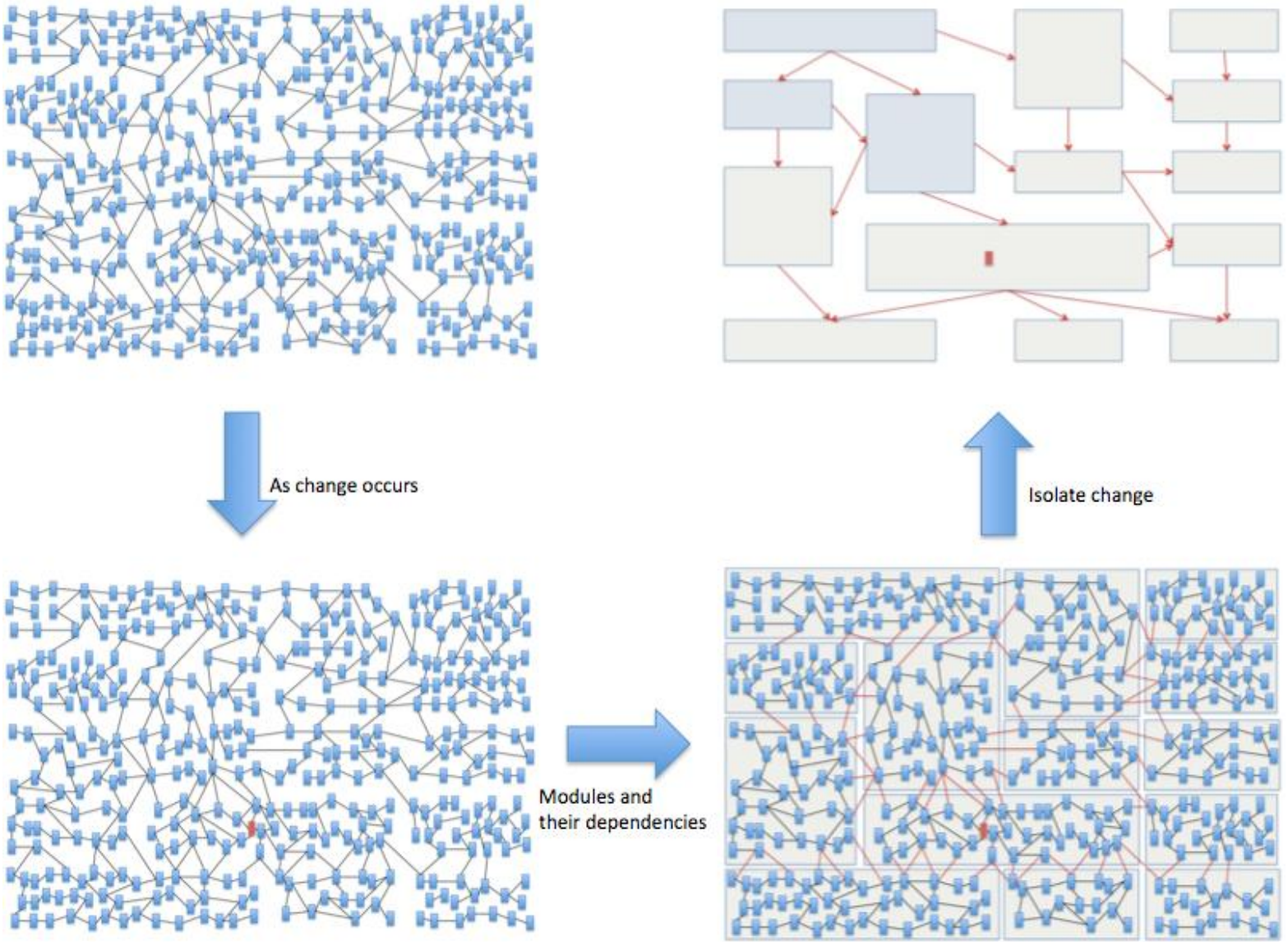
Autor:
Christoph Schmidt-Casdorff

Agenda

- **Modularisierung**
- **OSGi – ein Einstieg**
- **OSGi-Architektur**
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept
- **Zusammenfassung**

Agenda

- **Modularisierung**
- **OSGi – ein Einstieg**
- **OSGi-Architektur**
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept
- **Zusammenfassung**



Quelle : http://techdistrict.kirkk.com/2009/08/13/modularity-by-example

Modularität – „Teile und Herrsche“

- **Prinzip „Teile und Herrsche“**
- **Bausteine eines Softwaresystems sind Softwareeinheiten (Module), die**
  - einen möglichst großen Grad an Isolierung/Unabhängigkeit zum Systemkontext besitzen
  - definierte Abgrenzungen zum Systemkontext und anderen Modulen besitzen
- **Kann ein Modul in Java formuliert werden?**

Modularität in Java

- **Sichtbarkeit von Klassen**
- **Package**
  - Einzige Strukturierung jenseits der Klassen
  - Hat Sichtbarkeitsregeln
  - Sichtbarkeiten sind nicht hierarchisch
    - Sichtbarkeiten sind nur innerhalb eines Packages eingeschränkt
- **JAR**
  - Ist eine auslieferbare Ansammlung von Klassen/Ressourcen
  - Keine Sichtbarkeiten auf dieser Ebene
  - Wäre Kandidat für Modul

- **Java bietet keine Unterstützung zur Modularisierung**
  - Es muss etwas getan werden

Module

- **Module**
  - Sind Softwareeinheiten (physisch)
  - Können Abhängigkeiten untereinander definieren
    - Veröffentlichung für andere Module
    - Import von Veröffentlichung anderer Module
  - Können unabhängig voneinander de-/installiert werden

- **Ein Modul kann aktiv bestimmen, welche Teile von ihm privat und welche öffentlich sind**

Komponentenmodelle

- **Komponentenmodelle sind gelebte Modularität**
  - Liefern ein(e) Verfahren / Modell / Spezifikation zur Modularisierung

- **Komponentenmodelle definieren Verträge,**
  - wie eine Komponente aufgebaut ist
  - wie Abhängigkeiten zwischen Komponenten beschrieben werden
    - welche Klassen exportiert / importiert werden
  - welche Anforderungen eine Komponente an den Systemkontext stellen kann

- **Begriffe *Modul* und *Komponente* verwenden wir synonym**
  - Module zielen mehr auf physische Struktur ab
  - Komponenten sind allgemeiner (auch auf logische Strukturen)

Komponentenframework

- **Komponentenframework**
  - Laufzeitumgebung für Module
  - Implementierung des Vertragwerks

- **Dirigent im Orchester der Module**
  - De-/Installation von Modulen
  - Auflösung der Abhängigkeit zwischen den Modulen
  - Integration der Module in das Gesamtsystem

Warum Komponenten?

- **Komponenten**
  - Komponenten sind ‚*missing link*' zwischen Architektur und OO-Design
  - Wichtige Strukturierungsebene für den Übergang
- **Management von Abhängigkeiten**
  - Unnötige Abhängigkeiten
    - Erhöhen Komplexität der Anwendung
    - Sind Haupttreiber von ‚rotten design'
- **Isolierte Entwicklung**
  - Anwendung ist eine Zusammenstellung von Komponenten
  - Geschieht erst auf Integrationsplattform
  - Idee: Anwendungsentwicklung als Baukasten

Agenda

- Modularisierung
- **OSGi – ein Einstieg**
- OSGi-Architektur
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept
- OSGi Compendium Services
- Zusammenfassung

Modularisierung und Java

- **Java besitzt keine Unterstützung zur Modularisierung**
- **Bietet aber technologische Vorrausetzungen für Modularisierung**
  - Classloader-Konzept

OSGi

- **OSGi**
  - Modularisierung mittels Java

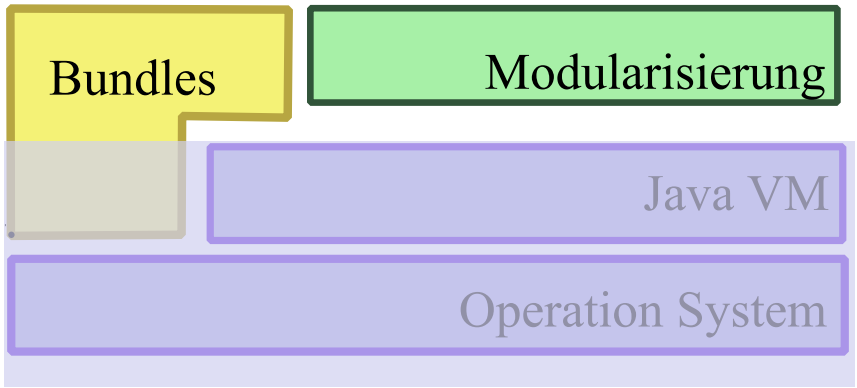
OSGi – historischer Abriss

- **Open Gateway Service initiative**
- **OSGi ist eine Spezifikation**
  - Durch OSGi Alliance organisiert
  - Gegründet 1999
- **Wurzeln von OSGi liegen in Anwendungen für *Mobile Endgeräte***
- **Mittlerweile engagiert sich OSGi im Bereich der *Java Middleware***
  - Aktuelle Version der Spezifikation ist 4.1

Agenda

- Modularisierung
- OSGi – ein Einstieg
- **OSGi-Architektur**
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept
- OSGi Compendium Services
- Zusammenfassung

OSGi-Architektur - Modularisierung



OSGi-Architektur - Modularisierung

- **Module heißen in OSGi *Bundle***
- ***Bundle* sind normale jar-Dateien**
- **Enthalten aber selbstbeschreibende Informationen**
  - Ablage als Manifest-Datei
  - Vertrag des Bundles mit dem OSGi Framework

Bundle-Selbstbeschreibung

- **Bundle-beschreibende Informationen sind beispielsweise**
  - Name, Identifikation, Version
  - Angabe der zu veröffentlichen Klassen/Ressourcen
    - Auf Ebene von java-Packages
    - Versioniert
  - Angabe von zu importierenden Klassen/Ressourcen
    - Auf Ebene von java-Packages
    - Versioniert
  - Angabe von zu importierenden Bundles
    - Versioniert

Beispiel einer Manifest-Datei

```
Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorld
Bundle-Version: 1.0
Bundle-SymbolicName:sample.HelloWorld
Bundle-ClassPath: .,
    target/dependency/junit-3.8.1.jar
Bundle-Activator:sample.internal.Activator
Export-Package:sample
Import-Package: common.commons,sample;
       version="[3.0.0, 4.0.0)",
   org.osgi.framework
Require-Bundle:com.sample.chess,
   com.sample.chess.core;version="2.0"
```

Sichtbarkeiten von Ressourcen

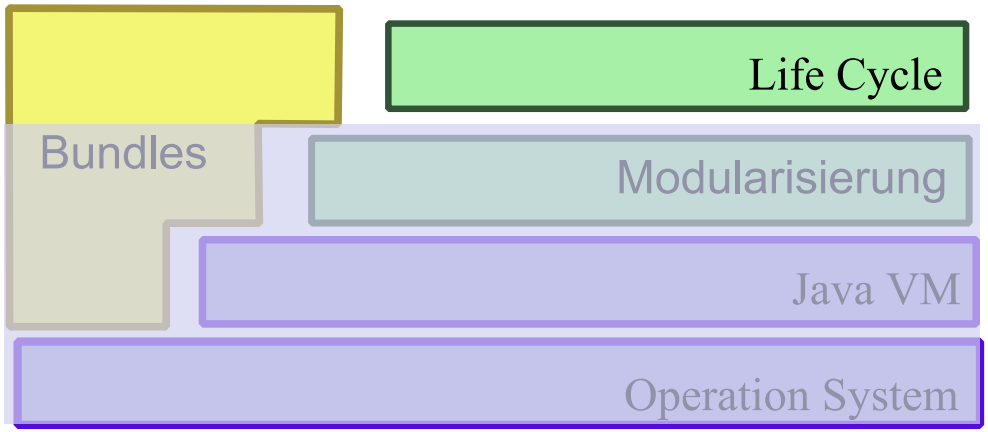
- **Die Modularisierung bedingt Sichtbarkeiten von Ressourcen**
  - Private Ressourcen müssen vor anderen Bundles versteckt werden
  - Öffentliche Ressourcen müssen anderen Bundles zugänglich gemacht werden (*Export*)
  - Angeforderte Ressourcen müssen bereitgestellt werden (*Import*)

- **OSGi löst Sichtbarkeiten über eine komplexe Classloader-Hierarchie**
  - Jedes Bundle hat seinen eigenen Classloader
  - Ein Bundle wird durch die von ihm angeforderten Bundles (transitiv) erweitert

Sichtbarkeiten von Ressourcen

- **Bundles können in unterschiedlichen Versionen koexistieren**
- **Die Konsistenz eines Bundles beeinflusst den Zustand anderer Bundles**
  - Export / Import

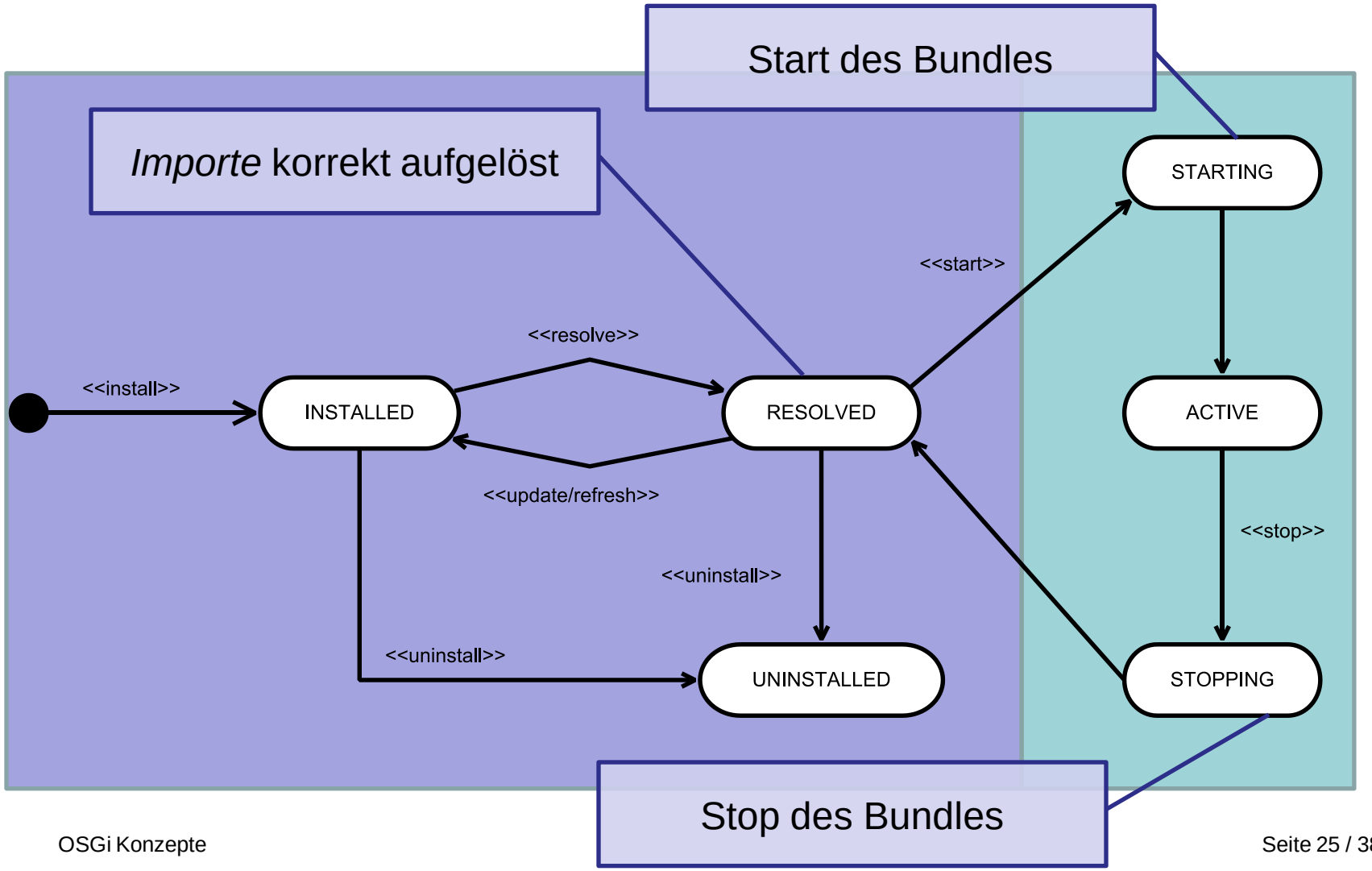
OSGi-Architektur – Life Cycle



Life Cycle Layer - Dynamik von Bundles

- **Offene Fragen**
  - Ist in OSGi die Abhängigkeit zwischen Bundles statisch? **NEIN**
  - Welche Folgen hat die De-/ Installation eines Bundles ?
    - Imports sind womöglich nicht mehr aufzulösen
  - Wo wird denn die eigentliche Logik angestoßen?
    - Wo geht eine OSGi-Anwendung los?
- **Mit diesen Fragen beschäftigt sich der *Life Cycle Layer***

Laufzeitzustände von Bundles

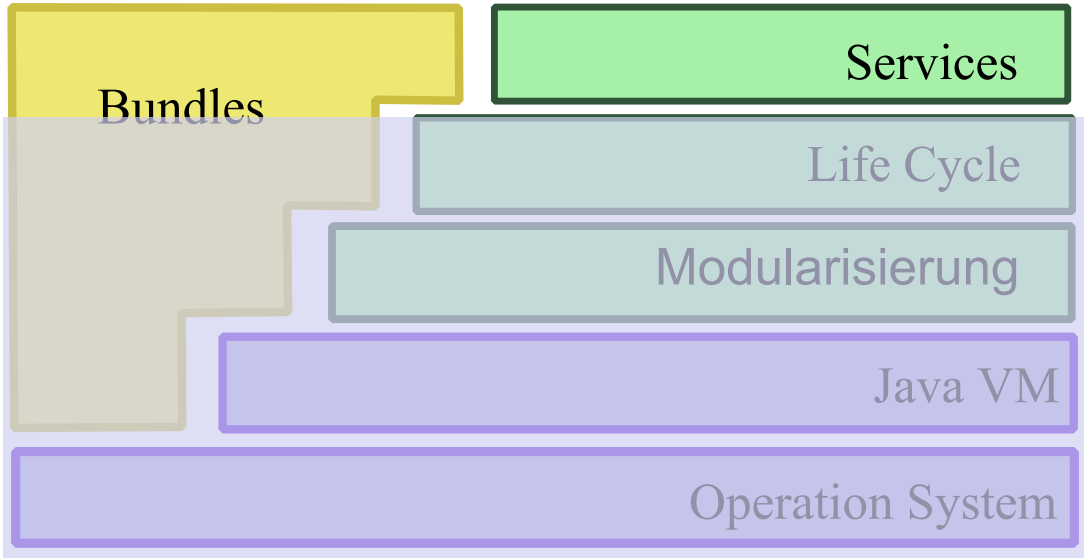


Dynamik von Bundles

- **OSGi Framework**
  - Ist für die Zustände der Bundles verantwortlich
    - Genauer gesagt : *management agent*
  - Prüft die Zustände dynamisch
    - Selbstheilung des OSGi Frameworks

- **Bundle kann spezifischen Code ausführen**
  - Während der Aktivierungsphase (start)
  - Während der Deaktivierungsphase (stop)
  - Methoden des *Bundles* werden durch OSGi Framework gerufen
  - Methoden *Bundle Activator* bereitgestellt

OSGi-Architektur - Services



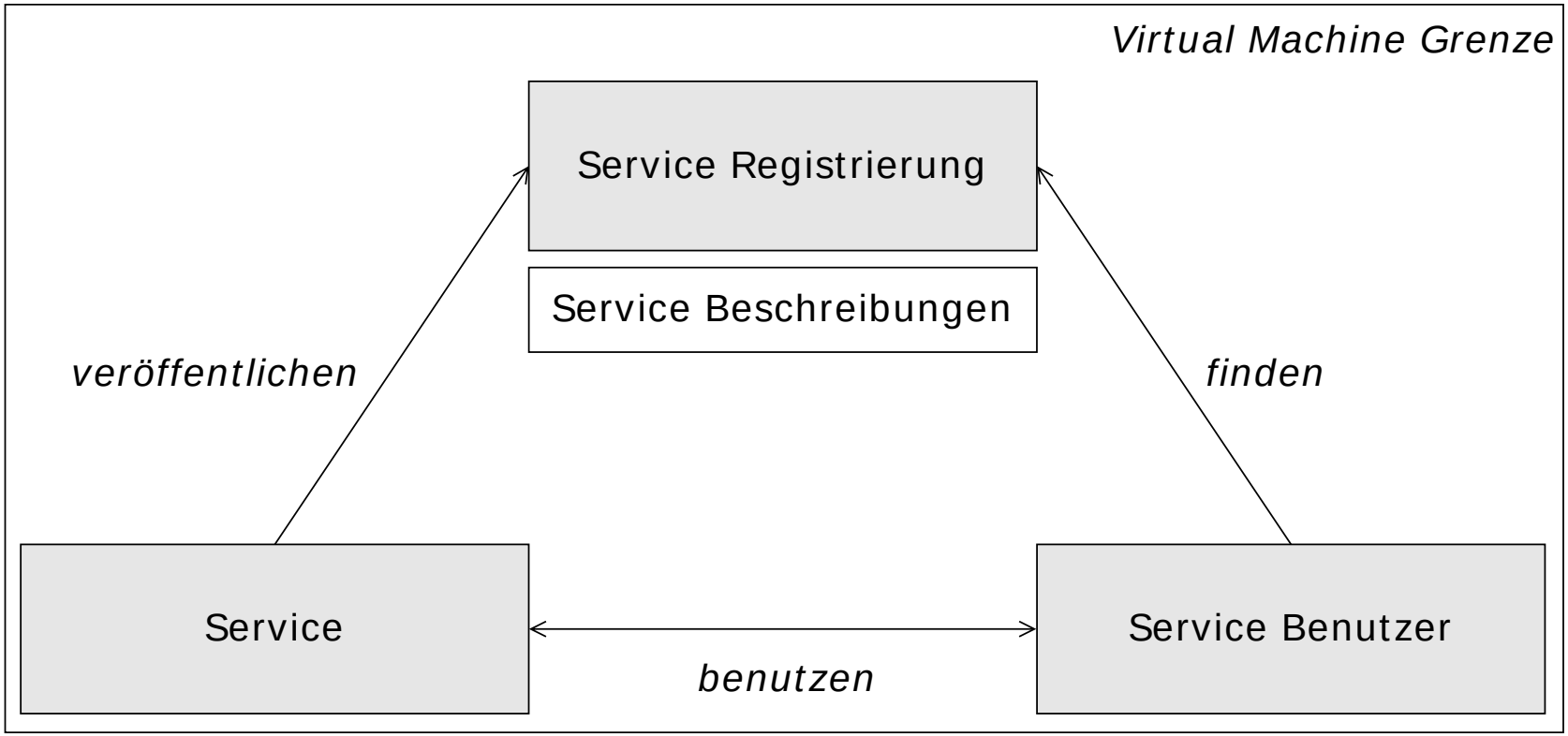
OSGi Services

- **Services**
  - Sind Java Objekte
  - Bieten Dienste an, die für andere Bundles interessant sein könnten
  - Werden i.d.R. beim Start eines Bundles erzeugt

- **OSGi bietet eine zentrale Service-Registratur an, bei der**
  - sich Services eintragen können
    - Mit Kriterien, unter denen sie gefunden werden
    - Qualitative Beschreibung des Service
  - Servicenutzer suchen nach bestimmten Kriterien Services

- **Keine direkte Beziehung zwischen Serviceanbieter und –Nutzer**
  - Implementierungen können wechseln
  - Mehrere Implementierungen können pro Service angeboten werden

OSGi Service



Dynamik von Services

- **Anbieter können ihre Services beliebig an- und abmelden**
  - Situation für den Servicenutzer kann sich jederzeit ändern
  - Dynamik von Services

- **OSGi bietet Bordmittel auf unterschiedlichen Leveln an**
  - low level: Nutzung von Verwaltungsmechanismen des Frameworks
    - Sehr komplex und sehr aufwendig
  - medium level: Sogenannte ***Service Tracker***, die im Hintergrund die Situation für den Nutzer aktuell halten
    - Komplex und aufwendig
  - high level: (*Service Component*) Modelle, die diese Dynamik im Hintergrund abhandeln
    - Spring DM, OSGi Declarative Services, iPOJO ….
    - Neue Programmiermodelle

Agenda

- Modularisierung
- OSGi – ein Einstieg
- OSGi-Architektur
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept
- **OSGi Compendium Services**
- Zusammenfassung

OSGi Compendium Services

- **OSGi standardisiert eine Reihe von Querschnittservices**
  - Logging
  - Eventhandling
  - Integration von Fremdanwendungen
  - Monitoring
  - Konfiguration
  - . . .
- **OSGi Compendium Services**
  - Sind (i.d.R.) normale Services
  - Setzen auf OSGi auf
- **Jede standardkonforme Implementierung läuft auf jedem OSGi Framework**

Agenda

- **Modularisierung**
- **OSGi – ein Einstieg**
- **OSGi-Architektur**
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept
- **OSGi Compendium Services**
- **Zusammenfassung**

Architektur reloaded

- **Modularisierung**
  - Definiert Vertrag zwischen Bundles und ihrer Umgebung
  - Definiert, wie dieser Vertrag umzusetzen ist
  - Definiert, welche Aufgaben das OSGi Framework zu übernehmen hat

- **Life Cycle Layer**
  - Definiert Zustände von Bundles und deren Übergängen
  - Bildet Modularisierung auf Zustände ab
  - Bietet den Einstiegspunkt zu eigener Programmlogik

- **Service Layer**
  - Definiert das Servicekonzept von OSGi
  - Beschreibt die Dynamik von Services und deren Konsequenzen
  - Beschreibt den Umgang mit dynamischen Services

OSGi Rekapitulation

- **Modularisierung**
  - Statische Sicht auf die Auslieferungseinheiten
  - Reduziert Abhängigkeiten der Module
    - Ausufernde Abhängigkeiten sind auschlaggebend für ‚*rotten design*‘
- **Life Cycle Layer**
  - Module können zur Laufzeit de-/installiert werden
  - Dynamische Funktionalität
  - Selbstheilung des Systems
- **Services**
  - Korrespondenz der Funktionalität via Service-Registry
  - SOA light

Zusammenfassung

- **Modularisierung und Komponentenmodelle**
- **Ansatz von OSGi**
- **Architektur von OSGi**
  - Modularisierung
  - Dynamik von Bundles
  - Servicekonzept

Referenzen

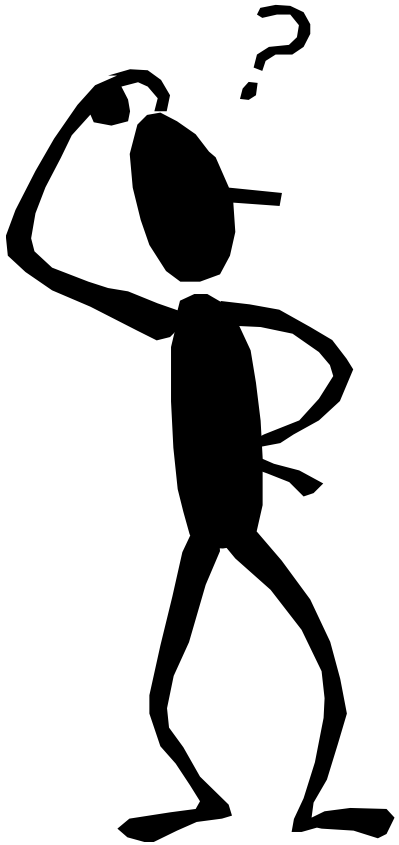
[CBSD]
http://en.wikibooks.org/wiki/Computer_programming/Component_based_software_development

Weiterführende Literatur

- Gerd Wütherich, Nils Hartmann, Bernd Kolb , Matthias Lübken
  Die OSGI Service Platform - Eine Einführung mit Eclipse Equinox ,
  dpunkt Verlag; Auflage: 1 (18. April 2008) ISBN-13: 978-3898644570

- OSGi In Practice von Neil Bartlett http://neilbartlett.name/blog/osgibook/

Fragen?



Gesellschaft für
Informations- und
Kommunikationssysteme mbH

www.iks-gmbh.com