HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Komponentenmodelle

Vertiefung anhand von Fallbeispielen

Martin Bättig

Letzte Aktualisierung: 24. November 2022

FH Zentralschweiz

Inhalt

- Übersicht Komponentenmodelle
- Minimales Komponentenmodell in Java
- Das OSGi-Komponentenmodell
- Ein Komponentenmodell für Microservices

Lernziele

- Sie kennen die Eigenschaften eines Komponentenmodells.
- Sie können das Konzept des Komponentenmodells anhand von Beispielen erklären.
- Sie können identifizieren, ob ein Konzept ein Komponentenmodell darstellt.

Übersicht Komponentenmodelle

Nutzen eines Komponentenmodells

"Eine Software-Komponente ist ein Software-Element, das zu einem bestimmten Komponentenmodell passt und entsprechend einem Composition-Standard ohne Änderungen mit anderen Komponenten verknüpft und ausgeführt werden kann." *Councill, Heineman: Component-Based Software Engineering, Addison-Wesley, 2001*

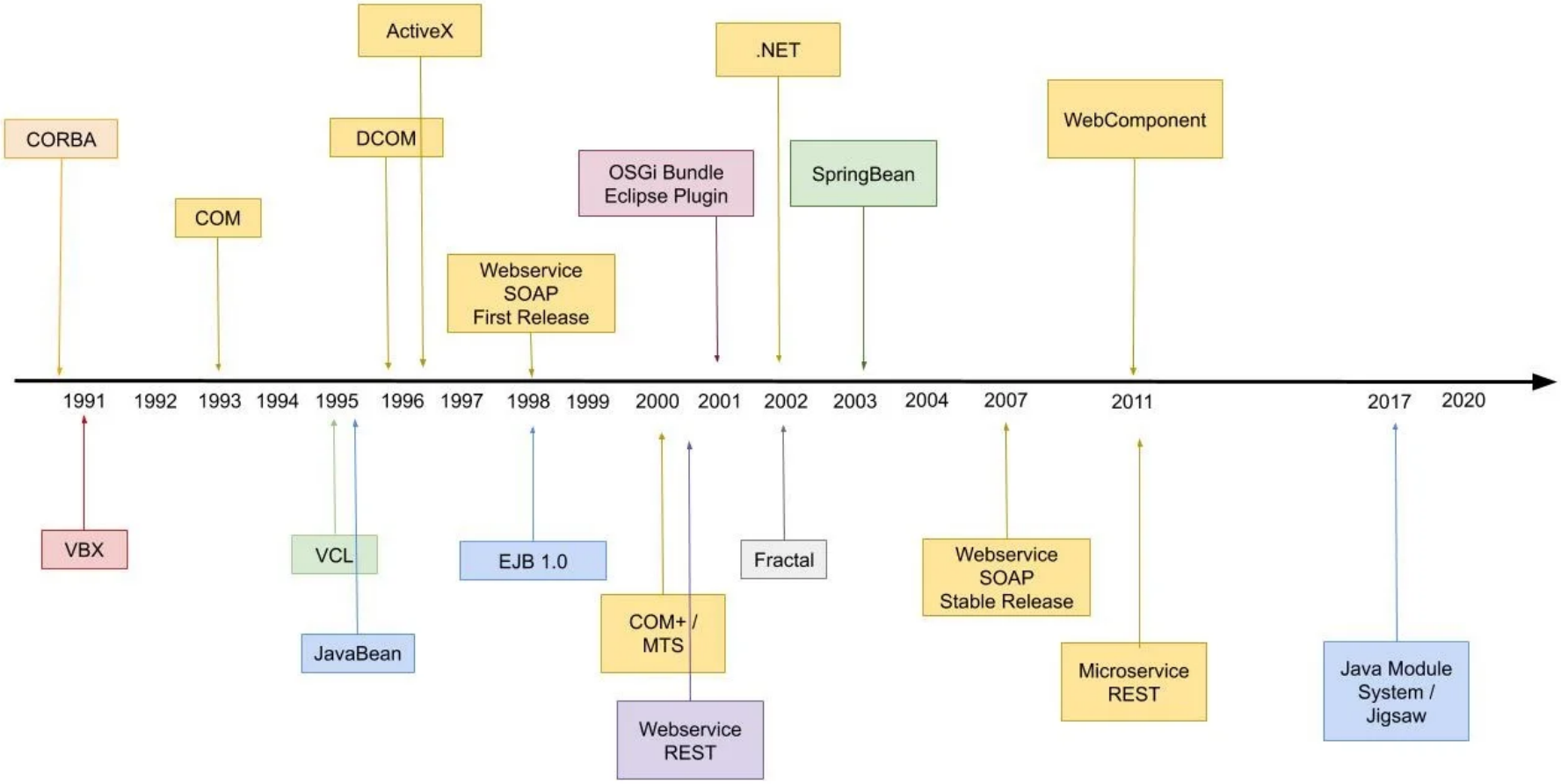
- **Komponentenmodell:**
  - Legt die grundlegenden Eigenschaften einer Komponente fest.
  - Rahmen für die Komponentenentwicklung oft in Zusammenhang mit einer Laufzeitumgebung.
  - Basis für die Interaktion: Nur kompatible Komponenten können miteinander interagieren.
  - Komponentenmodelle sind unterschiedlich spezifisch.

Grundlegende Eigenschaften eines Komponentenmodells

Eigenschaft	Inhalt
Schnittstellendefinition	Wie wird die Schnittstelle einer Komponente beschrieben?
Kommunikation und Verteilung	Wie kommuniziert die Komponente mit anderen Komponenten? Ist eine Verteilung auf mehrere Prozesse bzw. physische System möglich?
Komposition und Auffindbarkeit	Können zwei Komponenten miteinander verbunden werden? Wie wird dies gemacht? (Austauschbarkeit, Auffindbarkeit).
Deployment	Wie werden Komponenten ausgeliefert? Wie werden Komponenten gebündelt?

(Obige Liste zeigt minimale Anforderungen; teilweise spezifizieren Komponentenmodelle weitere Eigenschaften wie Sicherheit, Logging, Monitoring, etc.).

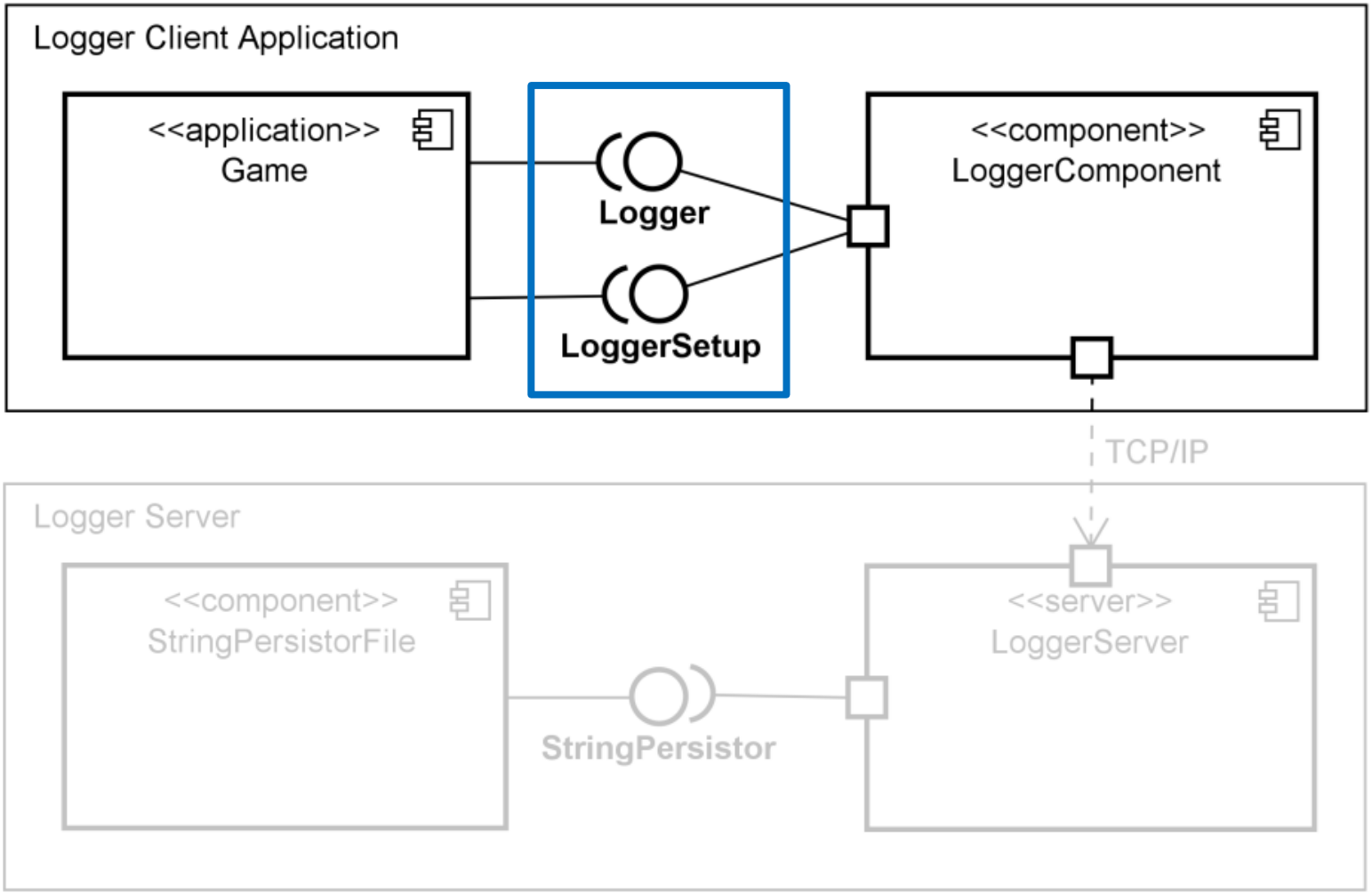
Komponentenmodelle: Eine Zeitreise...



Quelle: https://www.heise.de/hintergrund/Komponentenbasierte-Softwaretechnik-Was-ist-heute-noch-geblieben-5041855.html

Minimales Komponentenmodell mit Java

Komponentenübersicht des Loggersystems

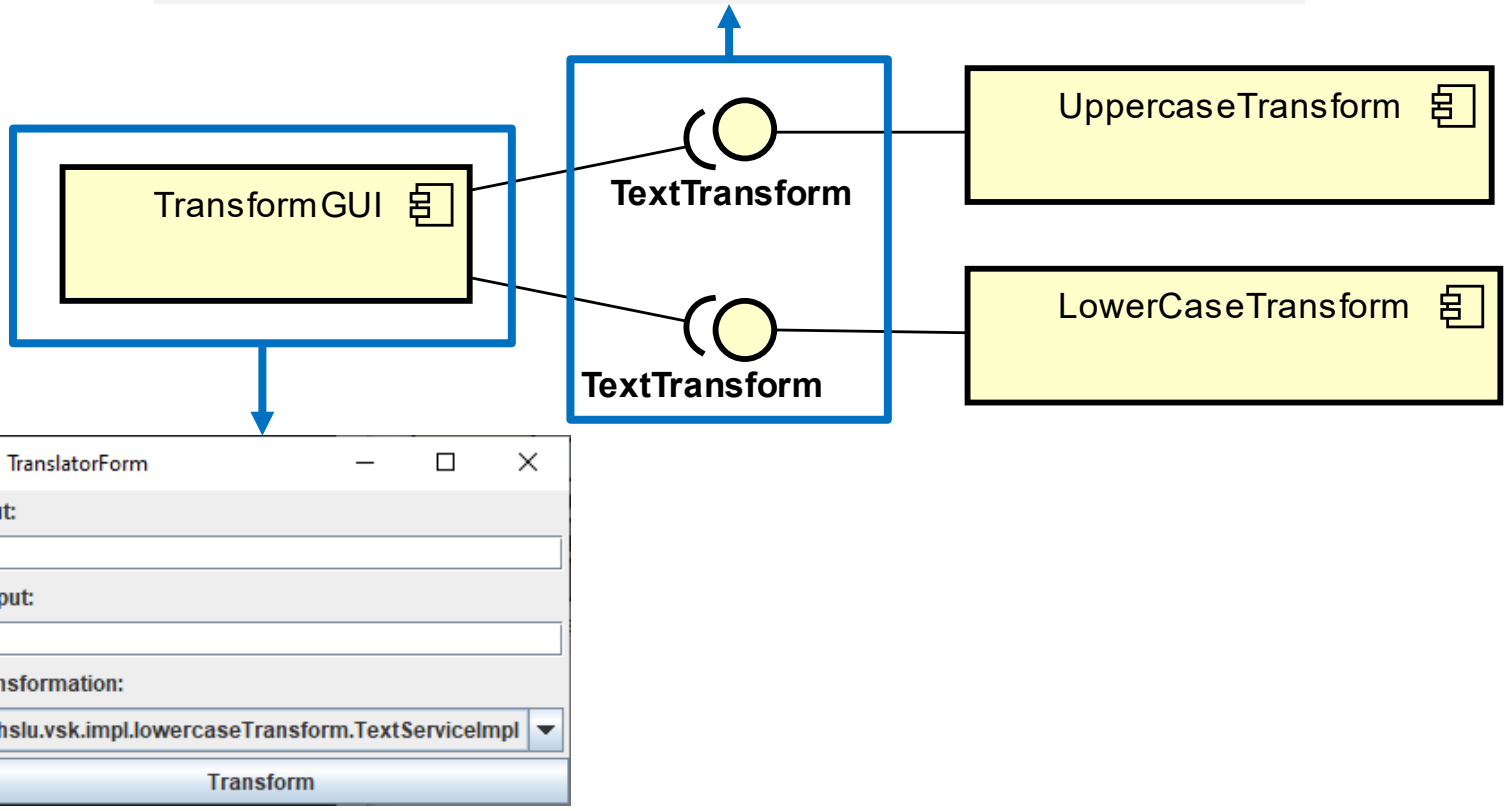


Bestimmung der vier grundlegenden Eigenschaften

Eigenschaft	Beschreibung
Schnittstellendefinition	
Kommunikation und Verteilung	
Komposition und Auffindbarkeit	
Deployment	

Kleine Demonstration mit drei Komponenten

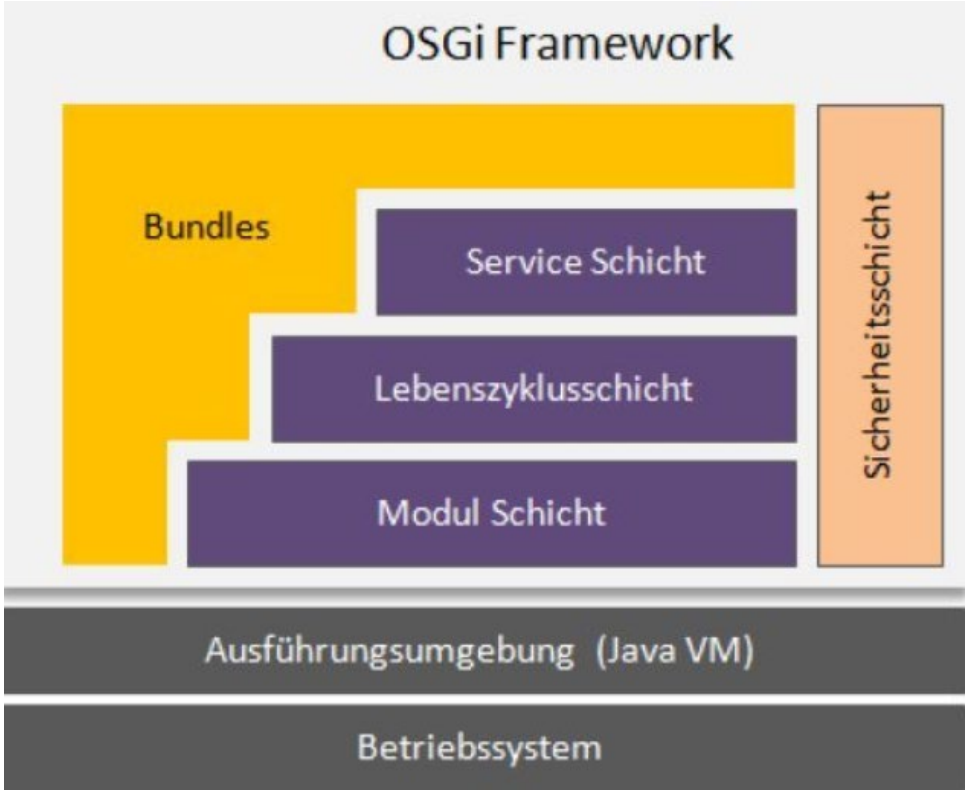
```java
public interface TextService {
    /**
     * Returns the input string in uppercase letters.
     * @param text to be converted.
     * @return text in uppercase letters.
     */
    String translate(String text);
}
```



Das OSGi-Komponentenmodell

OSGi: Übersicht

- Leichtgewichtiges Modul- und Servicesystem für Java.
- Standard: Implementationen verschiedener Hersteller verfügbar.
- Definiert ein vollständiges Komponentenmodel.



Quelle: Modularisierung mit Java 9, Guido Oelmann

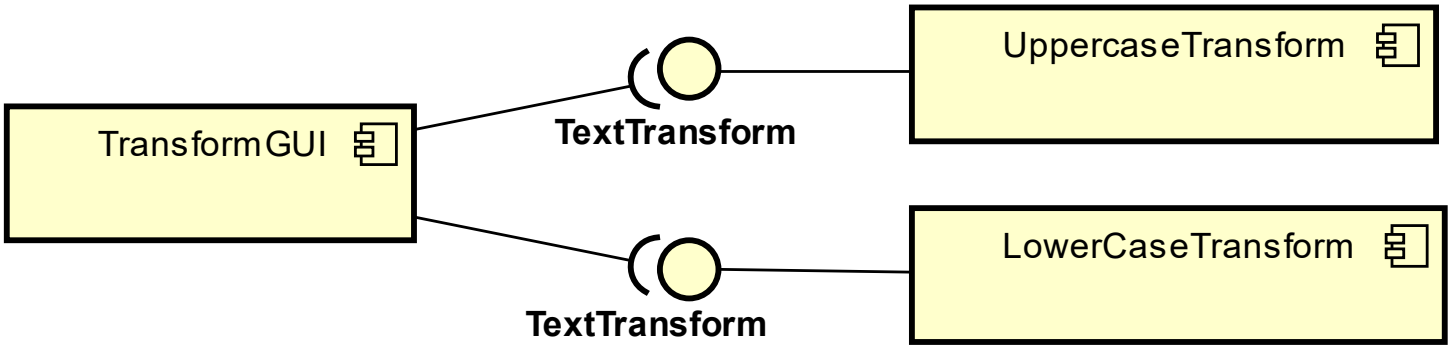
Anbieter und Anwender von OSGi

- Anbieter (Auszug):
  - Apache Felix (Basis für Apache Karaf)
  - Gemini / Equinox OSGi (Spring/Eclipse)
  - Concierge OSGi (smallfoot print OSGi for embedded devices)
  - Knopflerfish (hat ein kommerzielle Pro-Version)
  - Diverse kommerzielle Anbieter (Samsung, Hitachi, etc.)
  - Diverse Applikationsserver (WebSphere, Glashfish, WildFly, etc.)
- Einige bekannte Anwender von OSGi:
  - NetBeans
  - Confluence und JIRA
  - Eclipse
  - IntelliJ
  - usw.

Komponenten und Schnittstellen in OSGi

- Bundle ist die Bezeichnung von Komponenten bzw. Modul in OSGi.
- Sowohl Komponenten und Schnittstellen als Bundle geliefert.
- Bundle: JAR-Datei mit einem Manifest META-INF/MANIFEST.MF

Beispiel mit drei Komponenten und einer Schnittstelle:



⇒ vier Bundles

Beispiel eines OSGi-Bundle-Manifests

```
Manifest-Version: 1.0
Bnd-LastModified: 1616424844077
Build-Jdk-Spec: 11
Bundle-Activator: ch.hslu.vsk.textservice.ActivatorBundle
ManifestVersion: 2
Bundle-Name: Transform API (OSGi Variant)
Bundle-SymbolicName: ch.hslu.vsk.TransformApiBundle-Version:
1.0.0.SNAPSHOT
Created-By: Apache Maven Bundle Plugin
Export-Package:
ch.hslu.vsk.textservice;uses:="org.osgi.framework";version="1.0
.0"
Import-Package: org.osgi.framework;version="[1.4,2)"
Require-Capability:
osgi.ee;filter:="(&(osgi.ee=JavaSE)(version=11))"
Tool: Bnd-5.1.1.202006162103
```

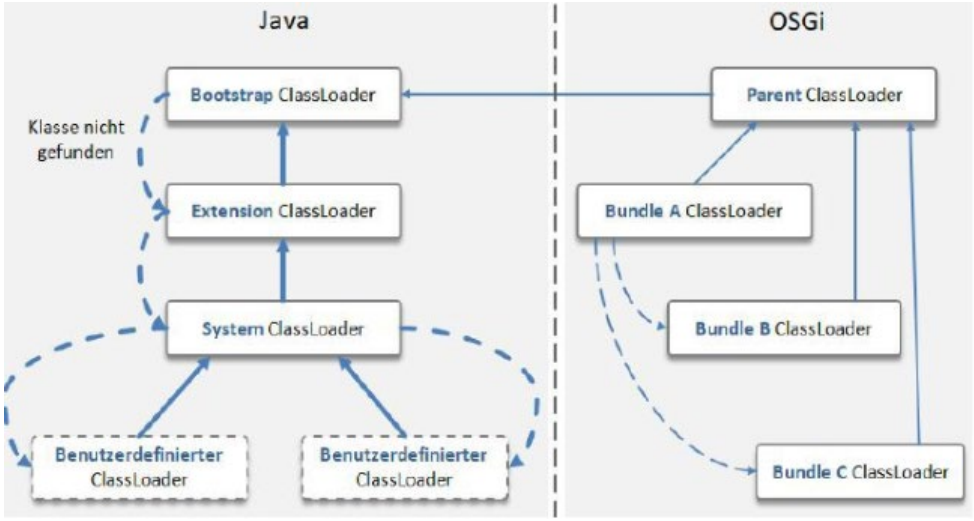
Schnittstellendefinition

- Reguläre Java-Interfaces.
- Als separates Bundle (JAR).

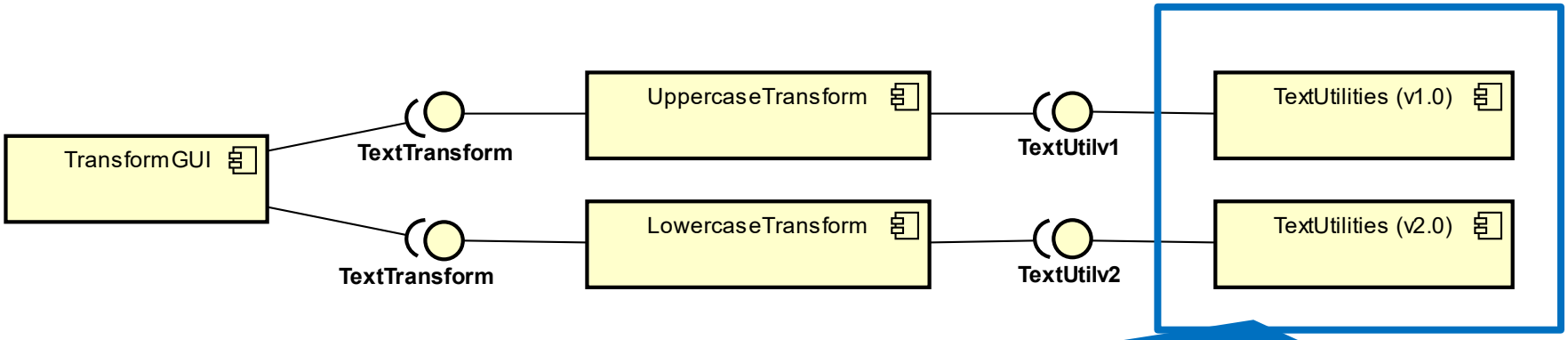
```
package ch.hslu.vsk.textservice;

public interface TextService {
    /**
     * Transforms the input string and returns the transformation.
     * @param text to be transformed.
     * @return transformed text.
     */
    String translate(String text);
}
```

Lösung der Versionierungsproblematik durch Isolation

- Nur exportierte Packages sind von anderen Bundles sichtbar.
- Ansatz: Verwendung von verschiedenen ClassLoader (automatisch).



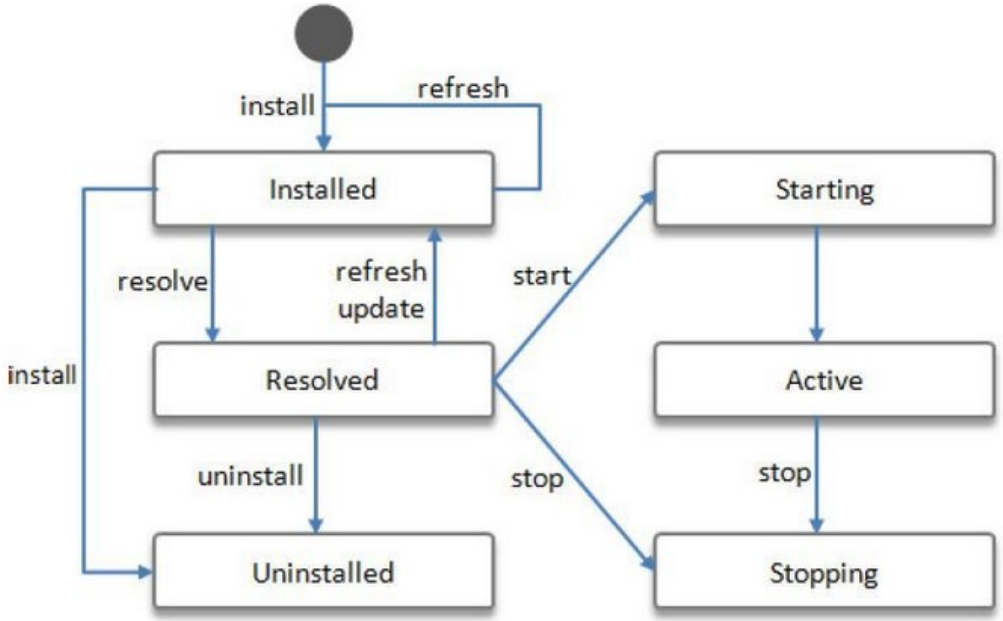
Beispiel:



Verwenden beide Klasse: `ch.hslu.vsk.impl.util.StringUtil`

Lebenszyklus von OSGi-Modulen

- Module werden dynamisch geladen, benutzt und entladen.
- Ideal für Plugins (Kein Neustart), aber keine garantierte Verfügbarkeit (analog zu Netzwerkdiensten).
- Bundle wird via BundleActivator (festgelegt in Bundle-Manifest) benachrichtigt, sobald es aktiviert wird.



Quelle: Modularisierung mit Java 9, Guido Oelmann

Beispiel eines Bundle-Activators

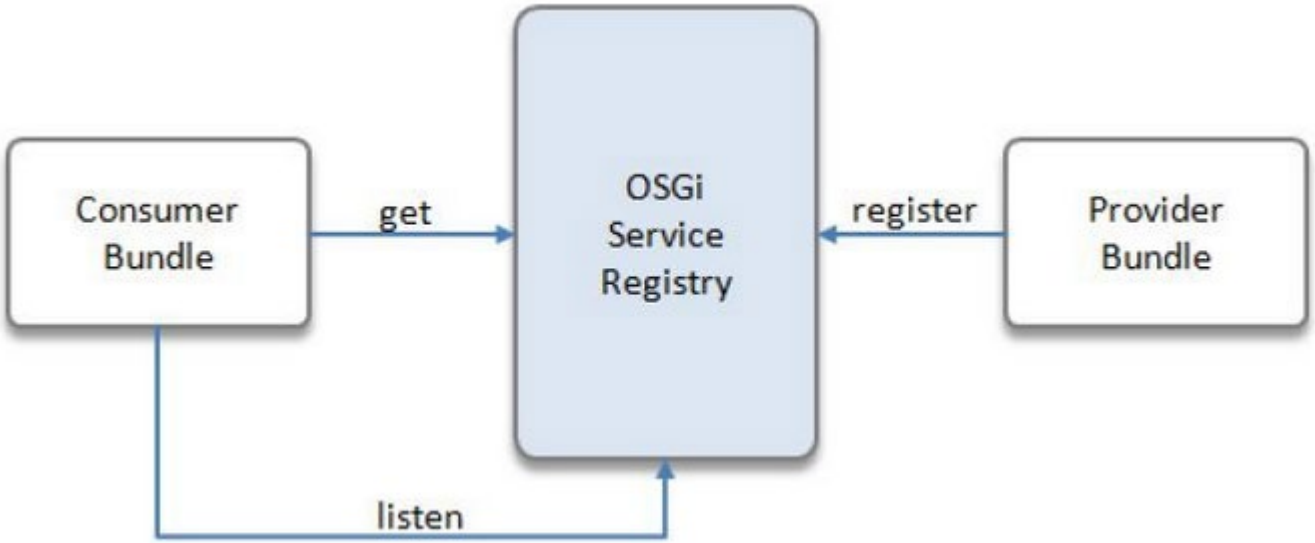
```
public class Activator implements BundleActivator {
    private JFrame main;

    public void start(BundleContext context) {
        main = TransformerClient.createForm(context);
    }

    public void stop(BundleContext context) {
        main.dispose();
    }
}
```

`start()`: Aufgerufen, wenn bundle aktiviert wird.

`stop()`: Aufgerufen, wenn bundle gestoppt wird.

OSGi Services und Service Registry

- Service-Registry: Zentrales Verzeichnis über alle Services.
- Service: Objektinstanz, welche ein bestimmtes API implementiert und in der Service-Registry publiziert wird.



Quelle: Modularisierung mit Java 9, Guido Oelmann

Beispiel: Registrieren und Auffinden von Services

- Registrieren einer Service-Instanz mit Zusatzinformationen:

```
Hashtable<String, String> properties = new Hashtable<>();
properties.put("name", "Lowercase Transform");
context.registerService(TextService.class.getName(),
                                        new TextServiceImpl(), properties);
```

- Finden einer Service-Instanz:

```
// Abrufen der Services, die das TextService API implementieren
ServiceReference[] refs = context.getServiceReferences(
                                        TextService.class.getName(), null);

// Service pinnen
TextService textService = (TextService) context.getService(refs[0]);

// Service verwenden
output = textService.translate(text);

// Service unpinnen
context.ungetService(refs[0]);
```

OSGi als Komponentenmodell

Eigenschaft	Beschreibung
Schnittstellendefinition	▪ Java-Interfaces.
Kommunikation (Verteilung)	▪ Method-Calls.
Komposition (Austauschbarkeit, Auffindbarkeit):	▪ Import/Export von Interfaces. ▪ Service-Registry. ▪ Isolation der nicht-exportierten Packages.
Deployment	▪ Einzelne JARs (Bundles). ▪ Bundle-JAR.

Ein Komponentenmodell für Microservices

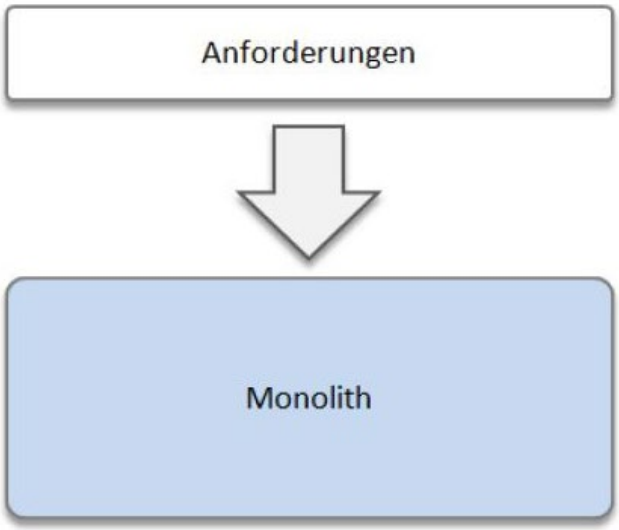
Was sind Microservices?

"In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies." (James Lewis and Martin Fowler, 2014)

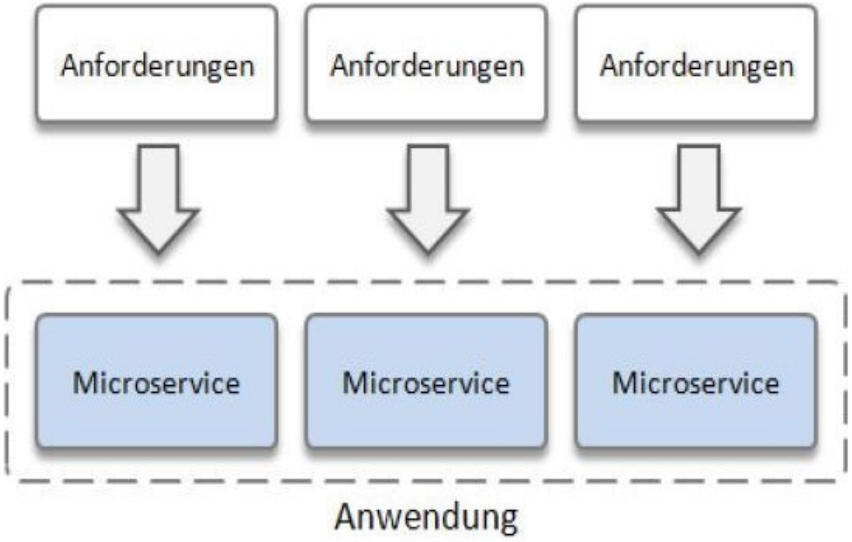
Hinweis: Nur Einordnung von Microservice als Komponente. Einführung der Microservices-Architektur erfolgt in Modul SWDA.

Aufteilung nach "Business Capabilities"

- **Ziel:** Bessere Aufteilung auf verschiedene Entwicklungsteams: Konzentration auf Fachdomänen und ggf. separate Auslieferung.



Ohne Microservices

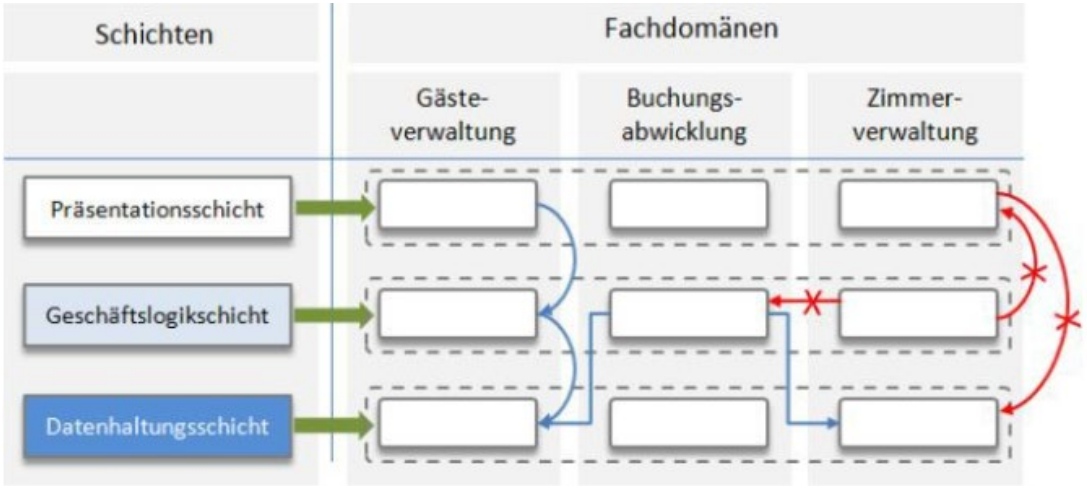


Mit Microservices

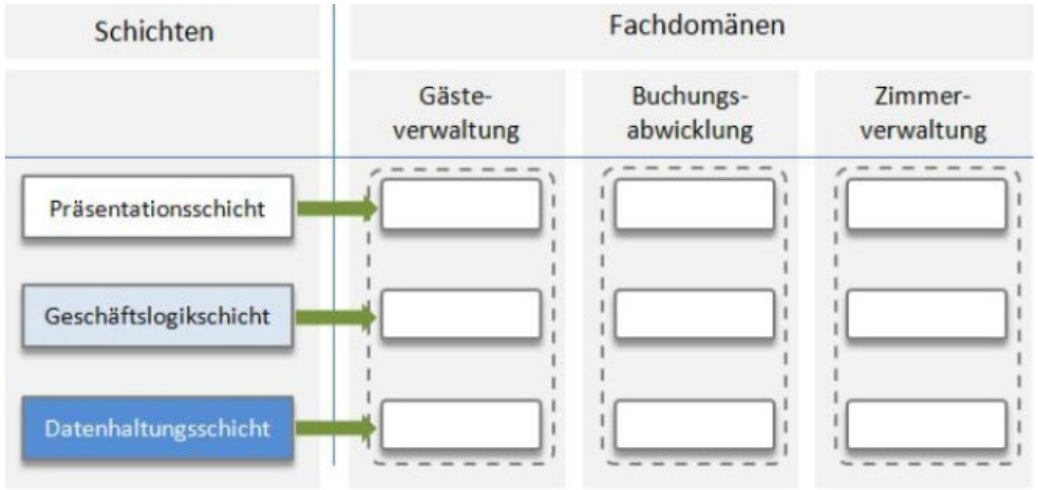
Bildquellen: Modularisierung mit Java 9, Guido Oelmann

Beispiel: Hotelbuchungssystem

Aufteilung nach Schichten:



Aufteilung nach Fachdomänen:



Bildquellen: Modularisierung mit Java 9, Guido Oelmann

Welche Aufteilung ist besser?

Eigenschaften von Microservices

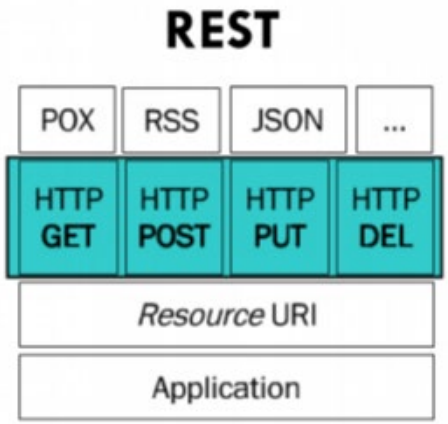
- **Aufteilung in Services:** Anwendung in kleine Services unterteilt, welche als Komponenten verwendet werden.
- **Schichten übergreifend:** enthält alle Schichten einer Funktionalität / Domäne (Frontend / Businesslogik / Persistenz).
- **Unabhängiges Deployment** (der kleinen Services).
- **Kommunikation über Netzwerk:** REST, MQ, SOAP, etc.
- **Entkopplung:** Microservice nur über Schnittstellen bekannt. Ausführung als eigener Prozess.
- **Technologische Entkopplung:** pro Microservices kann Plattform, Datenbank, Programmiersprache individuell festgelegt werden.

Komponentenmodell für Microservices

Eigenschaft	Beschreibung
Schnittstellendefinition	???
Kommunikation (Verteilung)	???
Komposition (Austauschbarkeit, Auffindbarkeit):	???
Deployment	???

Kommunikation

- Muss eine Interprozess-Kommunikation sein.
- Fowler und Lewis: "often an HTTP resource API".



URL	Method	POST Body	Result
http://example.com/entries	GET	empty	Returns all entries
http://example.com/entries	POST	JSON String	New entry created (ID)
http://example.com/entries/<id>	GET	Empty	Returns single entry
http://example.com/entries/<id>	PUT	JSON string	Updates existing entry
http://example.com/entries/<id>	DELETE	empty	Deletes existing entry

Komponentenmodell für Microservices

Eigenschaft	Beschreibung
Schnittstellendefinition	???
Kommunikation (Verteilung)	▪ HTTP REST API ▪ Verteilung über virtuelle und physische Systeme möglich.
Komposition (Austauschbarkeit, Auffindbarkeit):	???
Deployment	???

Schnittstellendefinition

- Populär ist aktuell OpenAPI (https://www.openapis.org).

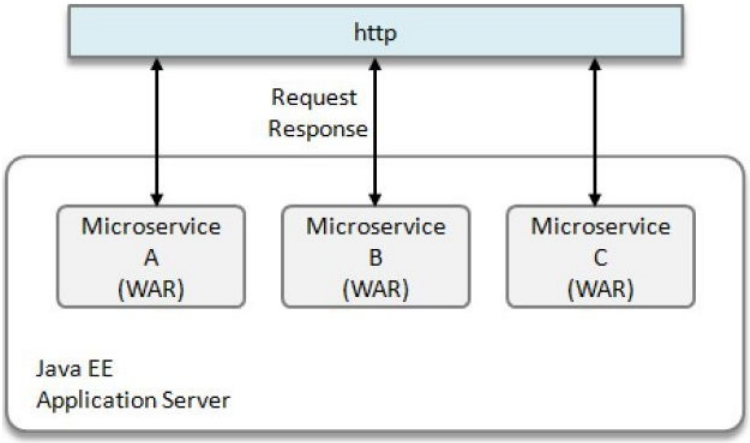
```yaml
openapi: "3.0.2"
info:
  title: "TransformApi"
  version: '1.0'
  description: "Interface for text transformations."
  license:
    name: "Apache License 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0"
paths:
  /transform:
    get:
      summary: "Applies transformation to input"
      description: "Applies transformation to input"
      parameters:
      - name: text
        in: query
        description: "Contains text to transform"
        schema:
          type: string
... and more ...
```

Komponentenmodell für Microservices

Eigenschaft	Beschreibung
Schnittstellendefinition	▪ OpenAPI
Kommunikation (Verteilung)	▪ HTTP REST API ▪ Verteilung über virtuelle und physische Systeme möglich.
Komposition (Austauschbarkeit, Auffindbarkeit):	???
Deployment	???

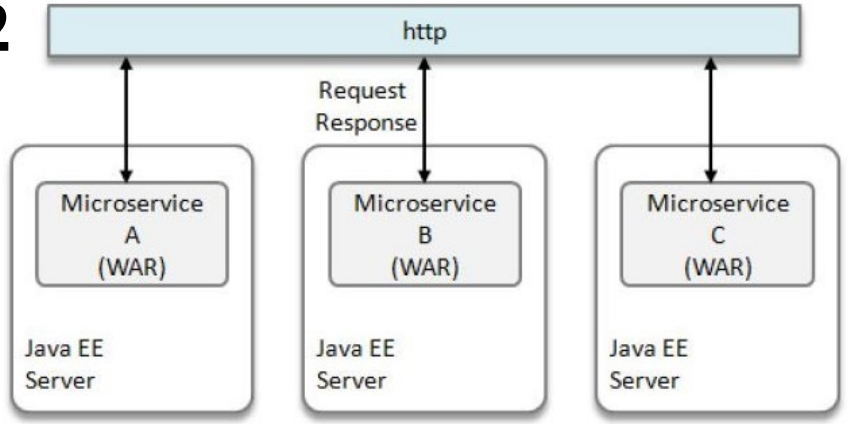
Varianten des Deployments

1



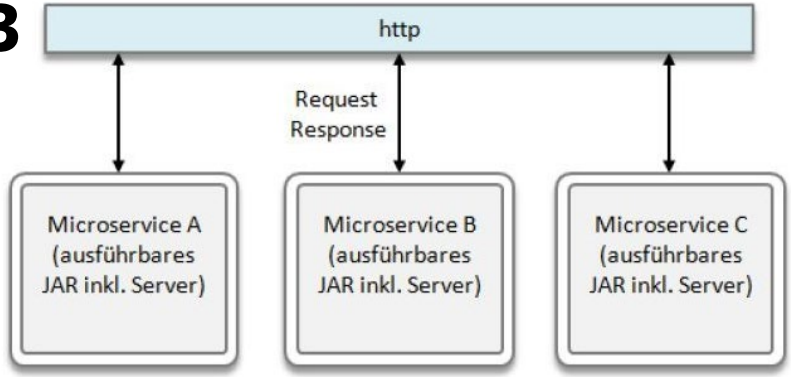
Kein eigener Prozess: Normale Webservices.

2



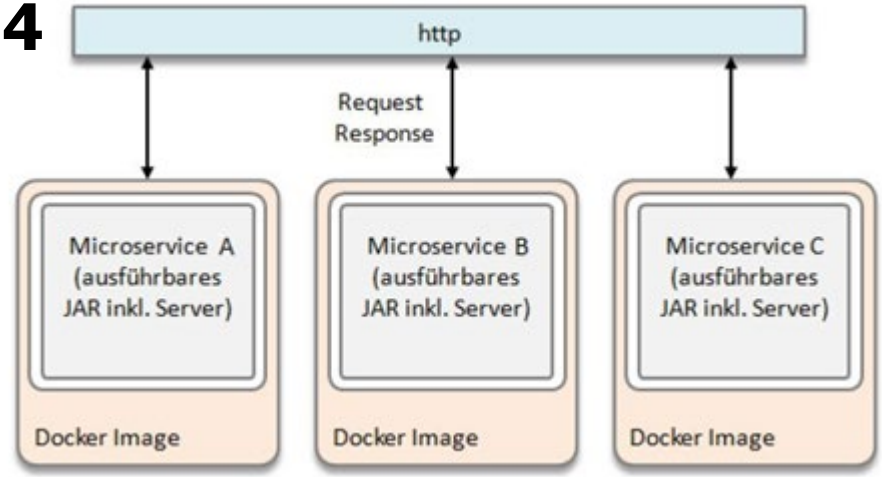
Bündeln mit vollwertigem Applikationsserver: Komplex (Konfiguration) / viele Ressourcen.

3



JAR -> Technologie neutral?

4



Container-Images mit allen Abhängigkeiten.

Bildquellen: Modularisierung mit Java 9, Guido Oelmann

Komponentenmodell für Microservices

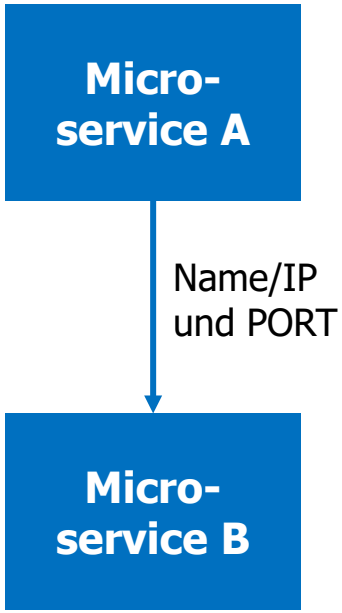
Eigenschaft	Beschreibung
Schnittstellendefinition	▪ OpenAPI
Kommunikation (Verteilung)	▪ HTTP REST API ▪ Verteilung über virtuelle und physische Systeme möglich.
Komposition (Austauschbarkeit, Auffindbarkeit):	???
Deployment	• Gebündelt mit HTTP-Server und allen Abhängigkeiten als Container-Image. • Abgelegt in einem binär Repository.

Verbinden von Services (Import / Export)

- Regulär via Hostname/IP und Port, jedoch sehr variable Umgebung: Viele Services, unabhängiges Deployment, Skalierung, Virtualisierung, Load-Balancing, Fault-Tolerance, usw.
- Statische Zuweisung von einem Microservice zum anderen z.B. via properties-File möglich, aber nicht praktikabel.

Mögliche Variante:

- **Export:** Container-Laufzeitumgebungen vergeben oft automatisch Namen (DNS) an Container.
- **Import:** Beim Start eines Containers können vergebene Namen in einer Systemumgebungs-Variable oder als Parameter automatisch an Konsumenten übergeben werden.



Komponentenmodell für Microservices

Eigenschaft	Beschreibung
Schnittstellendefinition	▪ OpenAPI
Kommunikation (Verteilung)	▪ HTTP REST API. ▪ Verteilung über virtuelle und physische Systeme möglich.

Komposition (Austauschbarkeit, Auffindbarkeit):	▪ Orchestrierung:
	  ▪ Automatische Vergabe von Name/IP und Port an Service-Anbieter.
	  ▪ Automatische Übergabe von Name/IP und Port an Konsument z.B. mit Umgebungsvar.
	▪ Isolation durch unabhängige Prozesse.

Deployment	• Gebündelt mit HTTP-Server und allen Abhängigkeiten als Container-Image. • Abgelegt in einem binär Repository.

Zusammenfassung

- Komponentenmodelle bilden einen Rahmen innerhalb dessen zwei oder mehr Komponenten direkt zusammengesteckt werden können.
- Ein vollständiges Komponentenmodell legt mindestens folgende Eigenschaften fest: Schnittstellendefinition, Kommunikation, Komposition, Deployment.
- Der OSGi-Standard definiert ein vollständiges Komponentenmodell auf Basis von Java, welches ein dynamisches Laden von Modulen (Komponenten) innerhalb einer JVM sowie die Definition von Services ermöglicht.
- Microservices ist eine Softwarearchitektur, aber kein Komponentenmodell. Wir können jedoch ein Komponentenmodell für Microservices herleiten.

Fragen?

Literatur

- Component-Based Software Engineering von Bill Councill und George T. Heineman, Addison-Wesley, 2001.
- Modularisierung mit Java 9 von Guido Oelmann, dpunkt.verlag GmbH, 2017.
- Microservices - a definition of this new architectural term von James Lewis and Martin Fowler, www.martinfowler.com, 2014.
- OpenAPI Specification Version 3.1.0, OpenAPI Initiative, a Linux Foundation Collaborative Project.