Middleware - Übung

Tobias Distler, Michael Gernoth, Rüdiger Kapitza

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

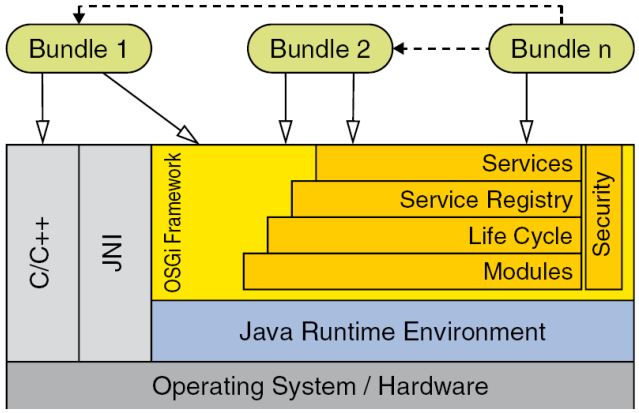
Wintersemester 2009/2010

OSGi — Überblick

- OSGi (Open Services Gateway initiative)
  - Standardisierte, komponentenorientierte Software-Plattform
  - Entspricht dem SOA-Paradigma (Service Oriented Architecture)
  - OSGi Service Platform setzt auf der Java Virtual Machine auf
- Spezifikation
  - OSGi Service Platform Release 1 (2000), aktuell Release 4.2 (2009)
  - Beschreibt lediglich API und Testcases
- OSGi-Frameworks
  - Equinox-Framework von Eclipse (`http://www.eclipse.org/equinox/`)
  - Apache Felix (ehemals Oscar) (`http://felix.apache.org/`)
  - Knopflerfish (Open Source) (`http://www.knopflerfish.org/`)
  - Diverse kommerzielle Produkte

Framework

- Universelle, sichere und zentral verwaltete Ausführungsumgebung für modulare Anwendungen (*Bundles*)
- Framework-Aufbau

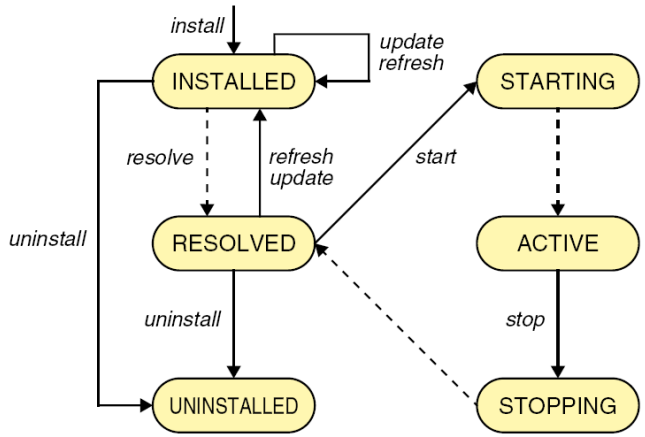


Schichten — Security Layer & Module Layer

- Security Layer
  - Bundles können aus verschiedenen Quellen stammen
  - Infrastruktur für eine kontrollierte Ausführung (controlled environment), basierend auf der Java 2 Security Architecture
  - Code-Authentifizierung durch Signaturen (Herkunft, Unterzeichner)
  - Security Layer ist optional und oft nicht implementiert
- Module Layer
  - Einheit der Modularisierung: *Bundle*
    - Format: Java Archive Datei (JAR)
    - Enthält alle für den Dienst notwendigen *Ressourcen* (inkl. weiterer JARs)
    - Eigener Class-Path und eigener Class-Loader (Policy)
    - Bundle-Beschreibung und Abhängigkeiten (z.B. Java Packages) in einer *Manifest-Datei* (`META_INF/MANIFEST.MF`)
  - Kontrolliertes Linken zwischen Bundles

Life Cycle Layer

- Verwaltung des Lebenszyklus von Bundles
- Bundles können installiert, gestartet, gestoppt und deinstalliert werden
- Bundles können außerdem zur Laufzeit aktualisiert werden



Life Cycle Layer – Entitäten

- `Bundle`: Repräsentiert ein installiertes Bundle
- `Bundle Context`: Ausführungskontext eines Bundle
  - Zugriff auf Informationen des Framework und die Service-Registry
  - Installieren anderer Bundles
  - Wird beim Starten/Stoppen an den Bundle Activator weitergereicht
- `Bundle Activator`: Interface implementiert durch eine Klasse des Bundle
  - Starten des Bundle
  - Stoppen des Bundle
- `Bundle Event`: Signalisiert Zustandsänderungen des Lebenszyklus
- `Framework Event`: Signalisiert Zustandsänderungen des Framework
- `Bundle/Framework Listener`: Listener für Events
- `Bundle Exception`: Ausnahme für Framework-Operationen
- `System Bundle`: Bundle-Repräsentation des Frameworks

Service Layer

- *Service Registry*
  - Registrierung von Diensten
  - Finden von Diensten
  - Binden von Diensten
- Dienste
  - Java-Objekte (*Service Object*)
  - Registriert mit ihrem/ihren Interface(s) (*Service Interface*)
- Bundles können Dienste registrieren, suchen und ihren Zustand abfragen
- Mit dem Stoppen des Bundle werden alle registrierten Dienste entfernt

Service Layer – Entitäten

- `Service`: Registrierter Dienst in der Service Registry, das Service Object gehört zu und läuft in einem Bundle
- `Service Registry`: Verwaltet die Dienste
- `Service Reference`: Referenz auf einen Dienst

  Zugriff auf Diensteigenschaften (auf Service Object über Bundle Context)
- `Service Registration`: Rückgabe der Registrierung, erlaubt Aktualisieren und Entfernen des Diensts
- `Service Factory`: Ermöglicht es dem Anbieter, das Service Object auf den Nutzer anzupassen
- `Service Event`: Informationen über Registrierung, Änderung und Deregistrierung von Service Objects
- `Service Listener`: Listener für Service Events
- `Filter`: Objekt für die Auswahl über Diensteigenschaften

Felix OSGi Framework

- Apache Top-Level-Projekt
  - URL: `http://felix.apache.org`
  - Installation: Download und entpacken
- Felix starten und testen

```
user@faui48a:/local/felix> java -jar bin/felix.jar

Welcome to Felix.
=================

Enter profile name: integration_test

DEBUG: WIRE: 31.0 -> org.osgi.service.packageadmin -> 0
[...]
DEBUG: WIRE: 33.0 -> org.apache.felix.shell -> 31.0

->
```

Felix OSGi Framework

Felix Shell

```
-> help
bundlelevel <level> <id> ... | <id> - set or get bundle start level.
cd [<base-URL>]                     - change or display base URL.
headers [<id> ...]                  - display bundle header properties.
help                                - display impl commands.
install <URL> [<URL> ...]           - install bundle(s).
obr help                            - OSGi bundle repository.
packages [<id> ...]                 - list exported packages.
ps [-l | -s | -u]                   - list installed bundles.
refresh [<id> ...]                  - refresh packages.
resolve [<id> ...]                  - attempt to resolve the specified bundles.
services [-u] [-a] [<id> ...]       - list registered or used services.
shutdown                            - shutdown framework.
start <id> [<id> <URL> ...]         - start bundle(s).
startlevel [<level>]                - get or set framework start level.
stop <id> [<id> ...]                - stop bundle(s).
uninstall <id> [<id> ...]           - uninstall bundle(s).
update <id> [<URL>]                 - update bundle.
version                             - display version of framework.
```

Felix OSGi Framework

- Konfigurationsdateien von Felix
  - `conf/system.properties` und `conf/config.properties`
  - Bundles können über `config.properties` konfiguriert werden
  - Zugriff über `BundleContext.getProperty()`
  - Alternative Standorte festlegen

```
java -Dfelix.system.properties=file:/local/felix/conf/system.properties
```

bzw.

```
java -Dfelix.config.properties=file:/local/felix/conf/config.properties
```

- Parameter (Beispiele)
  - `felix.auto.install`: Automatisch zu installierende Bundles
  - `felix.auto.start`: Automatisch zu startende Bundles
  - `felix.cache.dir`: Verzeichnis des Bundle-Cache (default ∼`/.felix/`)
  - `felix.cache.profile`: Name des Profils (innerhalb des Bundle-Cache)
- Integration in Eclipse

`http://felix.apache.org/site/integrating-felix-with-eclipse.html`

Felix OSGi Framework

- Life Cycle: Manuelles Installieren und Starten eines Bundle

```
-> install file:bundle/simple.jar
Bundle ID: 10
-> ps
START LEVEL 1
   ID   State          Level  Name
[   0] [Active     ] [    0] System Bundle (1.1.0.SNAPSHOT)
[...]
[  10] [Installed  ] [    1] Simple Bundle (1.0.0)
-> start 10
Simple bundle 10 has started.
From native: Hello!
```

- Stoppen, Aktualisieren und Deinstallieren von Bundles analog

  Achtung: Bei Abhängigkeiten wird evtl. ein `refresh` notwendig

Manifest

- Beschreibt das Bundle und seine Abhängigkeiten
- Notwendige Parameter
  - `Manifest-Version`: Manifest Spezifikation (1.0)
  - `Bundle-Name`: Name des Bundles
  - `Import-Package`: Von anderen Bundles importierte Pakete (zwingend notwendig ist `org.osgi.framework`)
- Wichtige Parameter
  - `Bundle-Activator`: Klasse die das Interface `BundleActivator` implementiert, andernfalls dient das Bundle nur als Bibliothek
  - `Bundle-SymbolicName`: Eindeutiger Name (meist in Domain-Form)
  - `Export-Package`: Pakete die von diesem Bundle exportiert werden
  - `Bundle-Version`: Version
  - `Bundle-Classpath`: Intra-Bundle Classpath für eingebettete JARs
  - `Require-Bundle`: Importiert alle Pakete der angegebenen Bundles

Manifest

- Aktivator-Klasse wird beim Start des Bundle ausgeführt (vgl. `main()`)
- Jedes Bundle hat eigenen Class-Loader und Class-Space
  Classpath ergibt sich aus: RTE + Framework + Bundle-Classpath
- Beispiel (`META-INF/manifest.mf`)

```
Manifest-Version: 1.0
Bundle-Name: simplebundle
Bundle-SymbolicName: gruppe0.osgi.simplebundle
Bundle-Version: 1.0.0
Bundle-Description: Demo Bundle
Bundle-Activator: gruppe0.osgi.simplebundle.impl.Activator
Import-Package: org.osgi.framework
```

- Auf korrekte Formatierung achten!
  - `Bezeicher: Wert`
  - Listen kommasepariert
  - Umbruch mit Leerzeichen einrücken
  - Zeilenumbruch am Ende der Datei

Build File

- Bauen eines OSGi-Bundle
  - Übersetzen der Quellen
  - Packen aller Ressourcen und des Manifests zu einem JAR
- Beispiel für ein Ant `build.xml`:

```
<?xml version="1.0"?>
<project name="simplebundle" default="all">
    <target name="all" depends="compile,jar"/>
    <target name="compile">
        <javac destdir = "./classes" srcdir = "./src"></javac>
    </target>
    <target name="jar">
        <jar basedir = "./classes"
            jarfile = "./build/simplebundle.jar"
            includes = "**/*"
            manifest = "./meta-inf/MANIFEST.MF"
        />
    </target>
</project>
```

Auflösen der Abhängigkeiten

- Auflösen der Abhängigkeiten (*Resolving*) zwischen Bundles
  Verdrahten von Importeuren und Exporteuren anhand von Bedingungen
- Bedingungen werden statisch definiert durch
  - importierte und exportierte Pakete
  - benötigte Bundles (import aller Pakete)
  - Fragmente (Teil eines größeren logischen Bundles)
- Ein Bundle kann aufgelöst werden wenn
  - alle Importe verdrahtet sind und
  - alle benötigten Pakete verfügbar und ihre Exporte verdrahtet sind
- Nach dem Auflösen kann ein Bundle geladen und ausgeführt werden

Bundle-Aktivator

`org.osgi.framework.BundleActivator`

- Aktivator-Klasse implementiert das Interface `BundleActivator`
- Beispiel

```
public class Activator implements BundleActivator {
    public static BundleContext bc;

    public void start(BundleContext bc) throws Exception {
        System.out.println("starting...");
        Activator.bc = bc;
    }

    public void stop(BundleContext bc) throws Exception {
        System.out.println("stopping...");
        Activator.bc = null;
    }
}
```

- Methoden `start()` und `stop()` müssen implementiert werden
  - `BundleContext` sollte beim Starten/Stoppen gesetzt/gelöscht werden
  - In `start()` wird das Bundle hochgefahren (z.B. Threads gestartet)
  - In `stop()` muss alles wieder aufgeräumt werden

Dienste anbieten

- Erweiterung des Bundles um einen Dienst
  - Bundle erhält ein Interface (Service Interface) für den Dienst
  - Das Manifest wird erweitert
- Manifest Erweiterung

```
Export-Package: gruppe0.osgi.simpleservice; version="1.0.0"
```

- Dienst-Interface

```
package gruppe0.osgi.simpleservice;
import java.util.Date;

public interface SimpleService {
    public String getFormattedDate(Date date);
}
```

- Dienst-Implementierung (*Service Implementation*) analog

Dienste anbieten

- Aktivator registriert den Dienst
  - `BundleContext.registerService();`
  - `ServiceRegistration` zur Verwaltung der Registrierung
- Beispiel

```
public class Activator implements BundleActivator {
    public static BundleContext bc;
    private ServiceRegistration registration;

    public void start(BundleContext bc) throws Exception {
        Activator.bc = bc;
        SimpleService service = new SimpleServiceImpl();
        registration = bc.registerService(
                        SimpleService.class.getName(), service,
                        new Hashtable());
    }

    public void stop(BundleContext bc) throws Exception {
        registration.unregister();
        Activator.bc = null;
    }
}
```

Dienst nutzen

- Manifest für die Nutzung eines Dienstes

```
Manifest-Version: 1.0
Bundle-Name: serviceuser
Bundle-SymbolicName: gruppe0.osgi.serviceuser
Bundle-Version: 1.0.0
Bundle-Description: Demo Bundle
Bundle-Activator: gruppe0.osgi.serviceuser.impl.Activator
Import-Package: org.osgi.framework, gruppe0.osgi.simpleservice
```

- Anforderung eines Dienstes
  - `BundleContext.getServiceReference(ServiceInterface);`
  - `BundleContext.getService(ServiceReference);`
- Framework erlaubt dynamisches Installieren, Aktualisieren & Löschen
  - `ServiceReference` muss daher immer überprüft werden
  - `ServiceReference` nach Verwendung freigeben

Dienst nutzen — Beispiel

```
import gruppe0.osgi.SimpleService;

public class Activator implements BundleActivator {
   public static BundleContext bc;

   public void start(BundleContext bc) throws Exception {
       Activator.bc = bc;
       ServiceReference r = bc.getServiceReference(
                              SimpleService.class.getName());
       if(r != null) {
           SimpleService s = (SimpleService) bc.getService(r);
           System.out.println(s.getFormattedDate(new Date()));
           bc.ungetService(r);
       } else {
           System.err.println("Service unavailable!");
       }
   }

   public void stop(BundleContext bc) throws Exception {
       Activator.bc = null;
   }
}
```

Dienst nutzen

- Dynamisches Binden von Diensten
  - Problem mit erstem Ansatz falls Dienst beim Start nicht verfügbar
  - Lösung: Regelmäßig prüfen oder `ServiceListener`
- `ServiceEvent` und Filter nutzen
  - Ein Filter ist ein LDAP ähnlicher String über den Events gefiltert werden
  - Um den Filter erweiterte `start()`-Methode des Aktivators:

```
public void start(BundleContext bc) throws Exception {
    Activator.bc = bc;
    String filter =
        "(objectclass=" + SimpleService.class.getName() + ")";
    bc.addServiceListener(this, filter);

    ServiceReference refs[] =
                      bc.getServiceReferences(null, filter);
    for(int i = 0; (refs != null) && (i < refs.length); i++) {
        this.serviceChanged(
           new ServiceEvent(ServiceEvent.REGISTERED,refs[i]));
    }
}
```

Dienste nutzen

- Der Aktivator erhält das Interface `ServiceListener`
  - Zusätzliche Methode `serviceChanged(ServiceEvent event)`
  - Hilfsmethoden (z.B. `stopUsingService()` und `startUsingService()`)
- Beispiel

```
public void serviceChanged(ServiceEvent event) {
  switch (event.getType()) {
  case ServiceEvent.REGISTERED:
       this.service = (SimpleService) Activator.bc.getService(
                                 event.getServiceReference());
       this.startUsingService();
       break;
  case ServiceEvent.MODIFIED:
       this.stopUsingService();
       this.service = (SimpleService) Activator.bc.getService(
                                 event.getServiceReference());
       this.startUsingService();
       break;
  case ServiceEvent.UNREGISTERING:
       this.stopUsingService();
       break;
} }
```

Dienste nutzen

- Dienste können mit Eigenschaften registriert werden
  - Zusätzliche Beschreibung des Diensts
  - Ermöglicht „best fit"-Auswahl bei mehreren Dienstanbietern
- Beispiel
  - Dienstanbieter

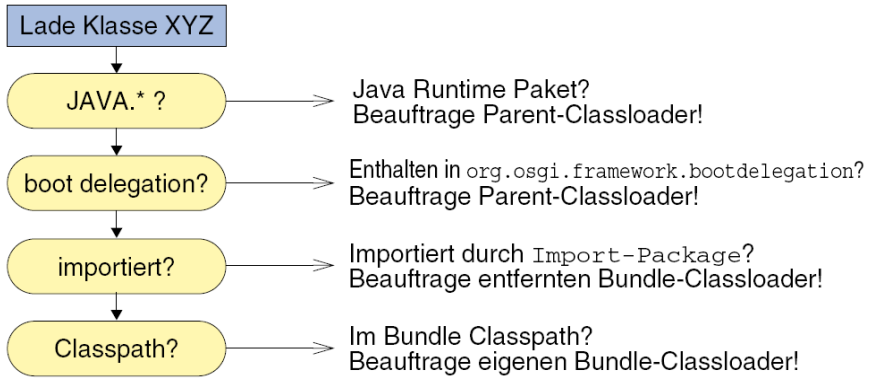
```
Hashtable properties = new Hashtable();
properties.put("color", Boolean.FALSE);
properties.put("A3paper", Boolean.TRUE);
bundleContext.registerService(
              "inf4.LaserJet", new Laserjet(), properties);
```

  - Beispiel: Dienstnutzer

```
ServiceReference[] srefs = bc.getServiceReferences(
           "inf4.LaserJet", "(A3paper=true),(color=false)");
```

Class-Loading

- Jedes Bundle besitzt einen eigenen `Class-Loader`
  - Dadurch verfügt jedes Bundle über eigenen `Namespace`
  - `Bundle-ClassPath` (Manifest) beschreibt Intra-Bundle-Classpath
- Suchreihenfolge



Class-Loading

- Kann zu nicht offensichtlichen Effekten führen
  - System-Class-Loader kann Klasse laden, wird aber ggf. wegen Regelwerk nicht ,,weitergereicht"
  - Framework-Konfiguration bestimmt welche Klassen exportiert werden
- Problematisches Beispiel
  - Klasse `XYZ` wird durch zwei Class-Loader `C1` und `C2` geladen
  - $\texttt{XYZ}_{C1}$ ist nicht gleich $\texttt{XYZ}_{C2}$

→ Kein Cast möglich: unterschiedliche Typen

Stale Reference

- In OSGi kann der Besitzer eines Objektes ,,verschwinden"
- ,,Abgestandene" Referenzen: Referenz auf ein Objekt
  - dessen Class-Loader bzw. Bundle gestoppt und
  - dessen Dienst deregistriert wurde
- Beispiel
  - Bundle `A` nutzt Service von Bundle `B`
  - `A` hält eine Referenz auf Objekt `O` der Klasse `K`, geladen durch den Class-Loader von `B`
  - Wird Bundle `B` gestoppt, ist Referenz auf `O` eine "*Stale Reference*"
- Problematik
  - Klassen eines gestoppten Bundles können nicht vom Garbage-Collector eingesammelt werden
  - Probleme beim Aktualisieren

→ Auftreten von Stale References verhindern

Aufgabe 7

- Bibliotheks-Server aus Aufgabe 3 als OSGi-Bundle
- Modularisierung von Server und Datenbank
  - Server-Bundle
  - Database-Bundle
- Vorgaben
  - Felix OSGi Framework: `/local/felix`
  - JAR-File für OSGi-Klassen: `/local/felix/bin/felix.jar`
  - Konfiguration: `/proj/i4mw/pub/aufgabe7`
- Ziel: dynamisches Starten und Stoppen von Datenbank und Server
  - Erst den Bibliotheks-Server starten, dann die Datenbank
  - Datenbank während des Betriebs stoppen und neu starten
  - ...