



# Rendszerintegráció és -felügyelet laboratórium (VIMIM309)

OSGi szolgáltatások fejlesztése

Mérési segédlet

Készítette: Izsó Benedek, Ujhelyi Zoltán

Utolsó módosítás: 2012. április 5.

Verzió: 1.2

Budapesti Műszaki és Gazdaságtudományi Egyetem  
Méréstechnika és Információs Rendszerek Tanszék

## 1 Bevezető

A labor során a méréseket végző hallgató a gyakorlatban is megismerkedik a rendszerintegráció és rendszerfelügyelet során használatos módszerekkel és eszközökkel. Végigköveti egy elosztott alkalmazás megvalósításának és felügyeletének legfontosabb lépéseit, ipari környezetben használt integrációs köztes réteg (middleware) technológiák és felügyeleti eszközök használatával. A mérések a következő témakörökhöz kapcsolódnak:

Munkafolyamatok megvalósítása Java nyelven

Megbízható üzenetküldés IBM WebSphere MQ alapon

Kommunikáció JMS és JMX technológia segítségével

OSGi szolgáltatások fejlesztése

Modell alapú eszközintegráció elosztott környezetben (SDE)

Felügyeleti adatok vizuális elemzése

Rendszerfelügyelet komplexesemény-feldolgozással

A mérés során a hallgató a korábban elkészített folyamat csomópontjait OSGi szolgáltatásokként valósítja meg, majd a munkafolyamatot ezen szolgáltatások meghívásaként valósítja meg.

## 2 Szolgáltatásfejlesztés OSGi platform felett

Az OSGi platform egy Java nyelv felett futó keretrendszer, melynek célja bővíthető Java alkalmazások fejlesztésének támogatása. Ehhez egy dinamikus bővítményrendszert biztosít, kezelve a csomagok futásidejű megjelenítését illetve eltűnését, valamint lehetőséget nyújt szolgáltatások definiálására.

E tulajdonságai sok területen felhasználhatóvá teszik az OSGi keretrendszert, például integrált fejlesztőeszközökben (Eclipse, Netbeans), alkalmazásszerverekben (Eclipse Virgo, GlassFish, JBoss Application Server, Oracle Weblogic, IBM WebSphere AS) vagy akár teljesen más alkalmazásokban (DataNucleus – perzisztenciakezelés SOA környezetben, Atlassian Confluence – enterprise wiki rendszer) használják.

Az OSGi platformnak többféle implementációja létezik: az Eclipse keretrendszer alapjául szolgáló Eclipse Equinox egy szabványos implementáció, de továbbiak is elérhetőek, mint például az Apache Felix vagy a Makewave Knopflerfish. A mérés során (és az útmutatóban is) az Equinox keretrendszert használjuk, de minimális módosításokkal a többi OSGi implementációt is hasonló módon lehet igénybe venni.

Az OSGi keretrendszer a Java virtuális gép felett biztosít néhány alapvető eszközt, amelyek felhasználásával különböző szolgáltatásokat lehet építeni. Ezen eszközök közül a legfontosabbak a *kötegek* (bundle) és a *szolgáltatások* (service), amelyek felhasználásával lehet elkészíteni az OSGi alkalmazásunkat.

### 2.1 OSGi kötegek (bundle)

Az OSGi kötegek forráskódnak (illetve kapcsolódó leíróknak) egy önállóan futtatható, illetve leállítható komponensét alkotják. A Java kód mellé szükséges egy leíró fájl alkalmazása, amely tartalmazza a függőségi és megosztási információkat. Ez a leíró a projektek MANIFEST.MF fájljában található, és tartalmaznia kell a köteg nevét és verziószámát.<sup>1</sup> A leíró ezen felül tartalmazza a függőségek és az exportált csomagok listáját.

A kötegek között függőségi viszonyt kell definiálni, azaz előírni más kötegek egy halmazát, amelyek nélkül nem lehet a saját kódot futtatni. Ezen függőségi viszonyt a keretrendszer a kötegek

---

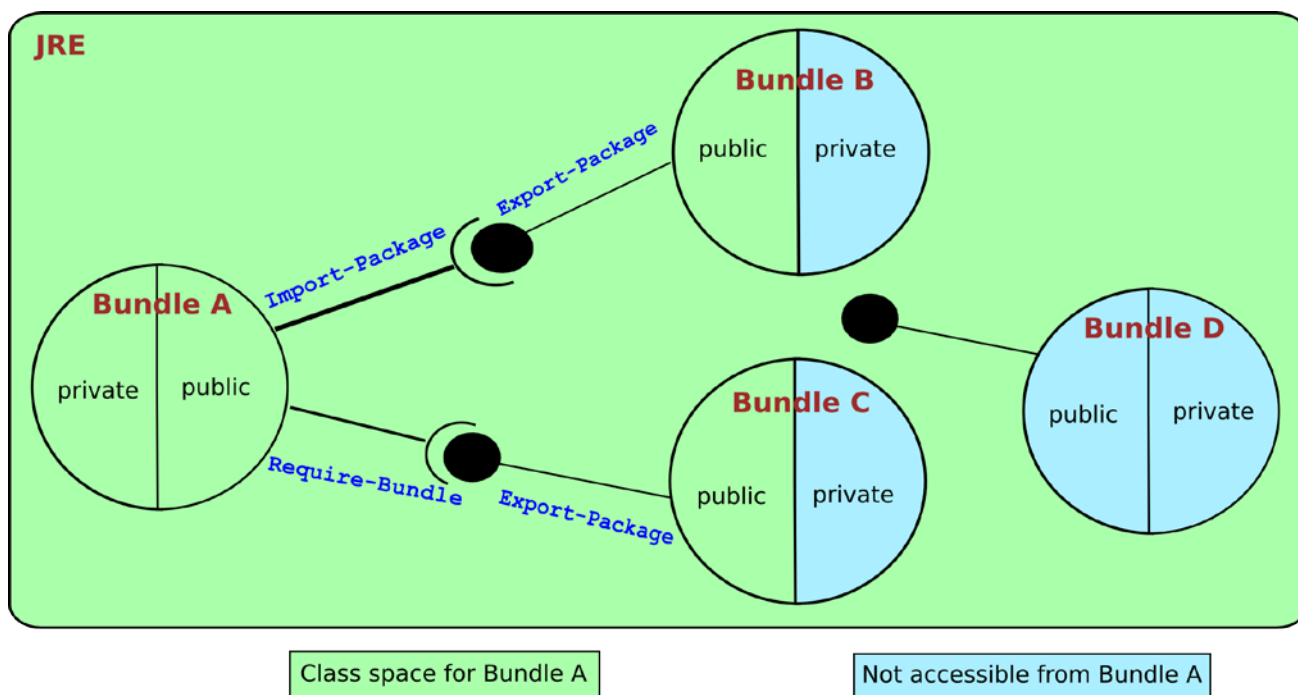
<sup>1</sup> A verziószám formátuma kötött: «major».«minor».«service».«qualifier», ahol az első három elem egy-egy szám, míg a «qualifier» egy build azonosító. Szokás szerint ezekkel a kompatibilitást jelezzük.

elérhetőségének megállapítására is felhasználja. A függőségi viszonyra épülnek különféle frissítési technikák, mint a p2 az Eclipse Equinox implementáció esetén: a függőségek megfelelő megadása esetén lehetséges egy komponenst és az összes függőségét egyszerre telepíteni.

A kötegek elősegítik az információrejtést, azaz szabályozható, hogy pontosan milyen *csomagok* (package) mely verziói legyenek elérhetőek a kötegen kívülről. Alapértelmezetten minden csomag rejtett a többi köteg számára, a nyilvános csomagokat explicit fel kell sorolni (*Export-Package*). A megoldás célja, hogy támogassa a kötegek frissítését és cseréjét azáltal, hogy a belső viselkedést megvalósító osztályokat elrejtí a többi köteg elől. Éppen ezért OSGi esetében érdemes csak a csomagfüggőséget előírni (*Import-Package*), nem pedig a kötegfüggőséget (*Require-Bundle*).<sup>2</sup>

Ezek alapján egy köteg kódjában négyféle csomag érhető el (mint azt az 1. ábrán a zöldre festett rész mutatja Bundle A szemszögéből):

- A köteg által definiált csomagok (*private*)
- A Java keretrendszer függvénykönyvtárai (*java.\**)
- *Import-Package* által kijelölt csomagok, melyek más kötegek által nyilvánosan (publikusan) megosztott csomagok közül kerülnek ki
- *Require-Bundle* által kijelölt köteg összes csomagja, amit nyilvánosan megosztott



1. ábra Egy köteg által elérhető csomagok

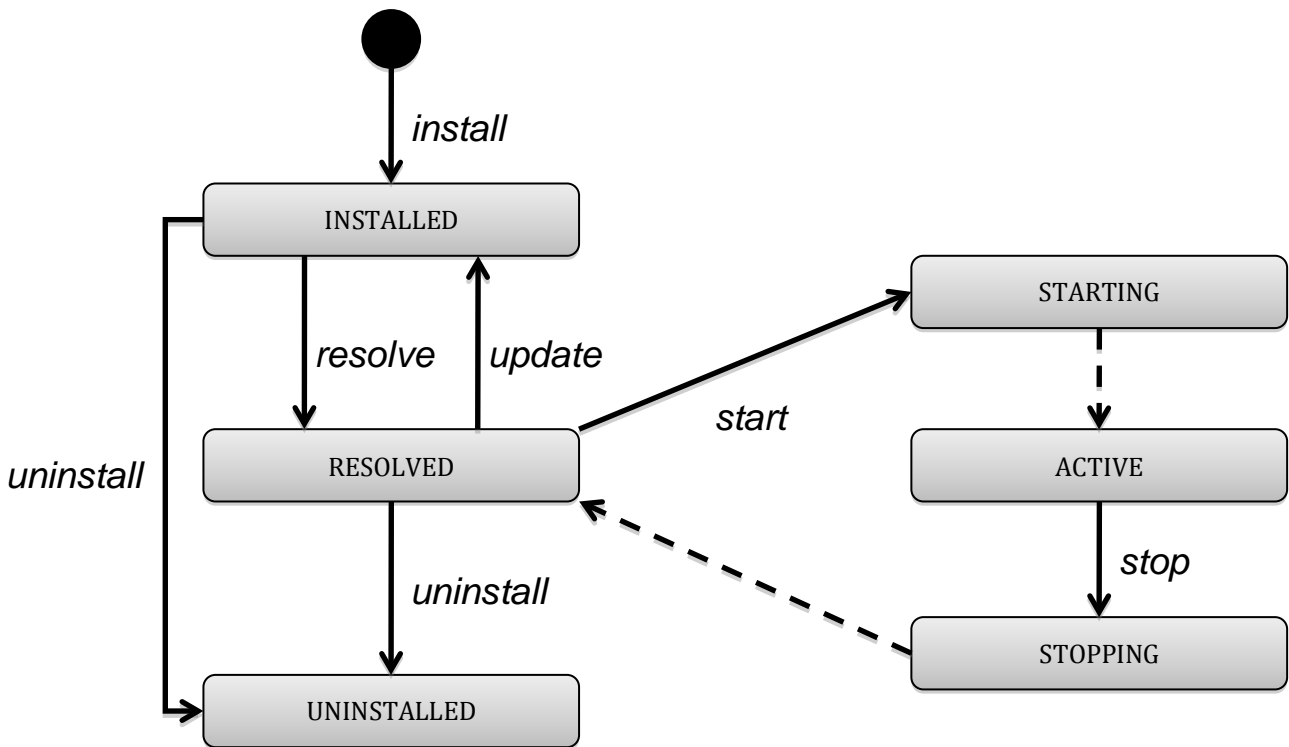
A kötegek életciklusa, ahogy a 2. ábrán látható, két részből áll: egy köteget először *telepíteni* (install) kell a keretrendszerbe, amely ezután megpróbálja *feloldani* (resolve) a függőségeit. Ha ez sikerrel jár, utána a köteget el lehet *indítani* (start) és használni, amíg *leállításra* (stop) nem kerül. Végül, ha már egyáltalán nincsen szükség a kötegre, *el lehet távolítani* (uninstall) a rendszerből.

A függő kötegek kiszámítása futási időben történik úgy, hogy minden egyes csomag importhoz keres a keretrendszer egy másik köteg által felajánlott nyilvános csomagot, amely megfelel az opcionális verziókényszereknek is. Mellesleg így lehetőség van egy Java függvénykönyvtár több verziójának betöltésére is, ilyenkor minden köteg csak azt a verziót látja, amivel együtt tud működni.

Az OSGi lehetővé teszi egy különleges, ún. *Activator* osztály regisztrációját. Ez az osztály callback metódusokat definiál: a *start* és *stop* metódusát meghívja az OSGi keretrendszer a köteg elindulásakor,

<sup>2</sup> Ez a futásidejű csomagfeloldás (*Import-Package*) nagyon rugalmas megoldást biztosít, amit fel lehet használni, ugyanakkor a statikus hibakeresésnek nem kedvez. Emiatt az Eclipse plug-inek (speciális OSGi kötegek) fejlesztésekor csomagok helyett a teljes plug-int szokás importálni (*Require-Bundle*); OSGi esetén viszont ez ellenjavallott.

illetve leállásakor. Ezen metódusok biztosítják az egyetlen garantált be- illetve kilépési pontot a megfelelő kötegből.



2. ábra: Kötegek életciklusa

## 2.2 Szolgáltatások megadása

A kötegek mellett a másik fontos OSGi által nyújtott eszköz az OSGi szolgáltatások. A szolgáltatások lehetőséget nyújtanak arra, hogy adott funkcionalitást biztosító kódrészleteket tegyünk elérhetővé explicit függőségek használata nélkül is.

A szolgáltatások eléréséhez egy *szolgáltatás tároló* (service registry) komponenst biztosít az OSGi. A felkínált szolgáltatásokat ide kell beregisztrálni, és innen lehet lekérni az egyedi szolgáltatásokat. Egy OSGi framework egy ilyen szolgáltatás tárolót használ. A szolgáltatásokat egy szöveges azonosító azonosítja, ami tipikusan a szolgáltatást nyújtó interfész típusa.

A szolgáltatásokat egy interfész és egy implementációs osztály segítségével adjuk meg, ahol az interfész egy nyilvános csomagban helyezkedik el, míg a megvalósító osztály tipikusan rejtett a jobb adatretetés érdekében. Fontos, hogy az interfész és az implementációs osztály nem kell, hogy azonos köteggben legyen (gyakran nem is így történik), mindössze az kell, hogy elérhető legyen az interfészt definiáló csomag.

Az így definiált szolgáltatások nagy előnye, hogy nincsenek kötegghez kötve: ha egy szolgáltatást több köteg is megvalósít, akkor lehetőség van akár az összes implementáció használatára, vagy pedig választásra tetszőleges logika szerint.<sup>3</sup>

Az OSGi szolgáltatások kezelését a *ServiceTracker* és a *ServiceRegistration* komponens végzi: az előbbi lehetővé teszi szolgáltatások elérését, míg az utóbbi szolgáltatások regisztrálására és eltávolítására szolgál. Mindkét komponenst a köteg Activator osztályán keresztül lehet elérni.

### 2.2.1 Szolgáltatások definiálása

Az alábbiakban egy egyszerű szolgáltatás definícióját mutatjuk be, amely egyetlen metódust biztosít. A szolgáltatást az *IHelloWorld* interfész jellemzi:

<sup>3</sup> A választás egy lehetséges módja a megfelelő szolgáltatást nyújtó kötegek kézi elindítása, ill. leállítása (programozottan ez ugyanakkor nem javasolt).

```
public interface IHelloWorldService {
    public void sayHello();
}
```

A szolgáltatás egy egyszerű implementációját a HelloWorldServiceImpl osztály tartalmazza:

```
public class HelloWorldServiceImpl implements IHelloWorldService {

    @Override
    public void sayHello() {
        System.out.println("Hello, world");
    }

}
```

Ezután a szolgáltatást elérhetővé tehetjük a keretrendszer számára. Ez két időpontban történhet: vagy a köteg indulásakor, vagy pedig futási időben (egyéb feltételektől függően, pl. más szolgáltatások elérése). Mindkét esetben az Activator osztályban hozzá kell férnünk a köteghez tartozó *ServiceRegistration* komponenshez, és ezen keresztül lehet hozzáadni, illetve eltávolítani a komponenst. Erre a következő kódrészlet mutat egy példát:

```
public class Activator implements BundleActivator {

    private ServiceRegistration registration;

    public void start(BundleContext context) throws Exception {
        IHelloWorldService bookManagerService = new HelloWorldServiceImpl();
        registration = context.registerService(IHelloWorldService.class.getName(),
            bookManagerService, null);
        System.out.println("IHelloWorld service is registered!");
    }

    public void stop(BundleContext context) throws Exception {
        registration.unregister();
        System.out.println("IHelloWorld service is unregistered!");
    }

}
```

Fontos, hogy a szolgáltatásainkat mindig állítsuk le (*unregister*) a köteg leállításakor, hogy a szolgáltatásunk felhasználói értesüljenek a változásról.

## 2.2.2 Szolgáltatások elérése

A szolgáltatások az OSGi környezetben a *ServiceTracker* komponens segítségével érhetőek el, amihez –a *ServiceRegistration* komponenshez hasonlóan– a köteg Activator osztályában férhetünk hozzá.<sup>4</sup>

A *ServiceTracker* nem közvetlen elérést biztosít egy szolgáltatáshoz, hanem lehetőséget nyújt ahhoz, hogy egy saját, eseménykezelőhöz hasonló osztály segítségével reagáljunk a megfelelő szolgáltatás megjelenésére, ill. eltűnésére. Ennek az eseménykezelőnek meg kell valósítania a *ServiceTrackerCustomizer* interfészt. Ez az interfész felhasználható arra, hogy építsünk egy aktuális referenciát a megfelelő szolgáltatást megvalósító komponensekből.

Egy ilyen eseménykezelő lehet a következő:

```
public class HelloWorldServiceTrackerCustomizer implements
    ServiceTrackerCustomizer {
    private final BundleContext context;
    private List<IHelloWorldService> services;
    public HelloWorldServiceTrackerCustomizer(BundleContext context) {
        this.context = context;
    }
}
```

<sup>4</sup> A szolgáltatások ennél egyszerűbben is elérhetőek a *context.getServiceReference()* hívás használatával, de ez erősen ellenjavallott az OSGi dinamikus természete miatt – ez nem kezeli, ha hirtelen leáll egy szolgáltatás a rendszerben.

```

        services = new ArrayList<IHelloWorldService>();
    }
    @Override
    public Object addingService(ServiceReference reference) {
        IHelloWorldService service = (IHelloWorldService) context
            .getService(reference);
        services.add(service);
        return service;
    }
    @Override
    public void modifiedService(ServiceReference reference, Object service) {
    }
    @Override
    public void removedService(ServiceReference reference, Object service) {
        services.remove(service);
    }
}

```

Az így elkészült eseménykezelőt pedig átadhatjuk a kapcsolódó *ServiceTracker* objektumnak, aki ezután az eseménykezelőn keresztül értesíti a kódunkat, ha változik a megfelelő szolgáltatások listája. Ezt a beállítást a felhasználó köteg *Activator* osztályában lehet felhasználni.

```

public class Activator implements BundleActivator {

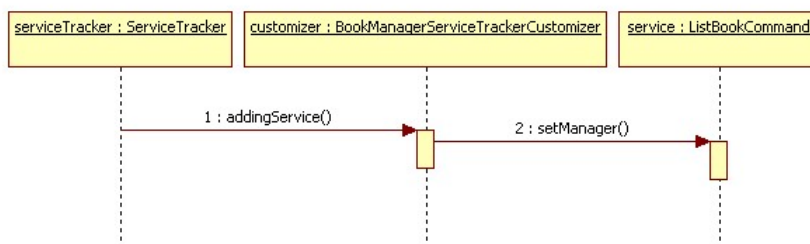
    ServiceTracker serviceTracker;
    @Override
    public void start(BundleContext context) throws Exception {
        ServiceTrackerCustomizer customizer =
            new HelloWorldServiceTrackerCustomizer(context);
        serviceTracker = new ServiceTracker(context,
            IHelloWorldService.class.getName(), customizer);
        serviceTracker.open();
    }
    @Override
    public void stop(BundleContext context) throws Exception {
        serviceTracker.close();
    }
}

```

Az eseménykezelő ezután már mindig friss referenciákkal rendelkezik a szolgáltatásokhoz, így megbízható módon felhasználható az alkalmazás többi részében.

A *ServiceTracker* leállítása szintén fontos a köteg leállításakor, így elkerülve, a szükségtelen (és helytelen) értesítések küldését.

Összegzésként a 3. ábrán látható szekvenciadiagram ismerteti, hogyan értesíthetjük a saját kódunkat a felhasznált szolgáltatások változásáról.



3. ábra: Áttekintő a *ServiceTracker* működéséről

## 2.3 Deklaratív szolgáltatások

Mint láhattuk, OSGi szolgáltatások esetén viszonylag sok, monoton ismétlődő kódot kell írni. Az OSGi 4.2-es verziójától kezdve lehetőség van a kézi szolgáltatás-regisztráció helyett egy deklaratív, XML-alapú *komponens definíció* felhasználásával regisztrálni és elérni a szolgáltatásainkat (függetlenül attól, hogy az deklaratív vagy hagyományos módon lett regisztrálva).

Ezeket a leírókat egy opcionális köteg (Equinox esetén *org.eclipse.equinox.ds*) olvassa be, és végzi el a regisztrációjukat.

A komponens definíciókat a szolgáltatás implementálóihoz és igénybe vevőikhez egyaránt definiálni kell. Előbbi esetben csak a szolgáltatást implementáló osztályt, illetve a biztosított interfészt kell megadni. Az interfészt, mint szöveget használja a rendszer a szolgáltatás azonosítására –hasonlóan az előző szakaszban leírt módon, csak ott kézzel kellett megadni a `registerService` első paramétereiként.

Ahhoz, hogy egy már létező szolgáltatásra hivatkozhatunk, meg kell adnunk az interfészt amivel hivatkozhatunk rá, a szolgáltatás megvalósítóinak számosságát (cardinality) valamint a szolgáltatás statikus vagy dinamikus mivoltát. A számosság segítségével kifejezhető, hogy elvárjuk-e a megfelelő szolgáltatás létezését, valamint azt, hogy tudunk-e több szolgáltatáspéldányt használni. Statikus szolgáltatás esetén a hívott szolgáltatás megszűnése esetén a keretrendszer az igénybe vevő szolgáltatást újraindítja, míg dinamikus esetben az igénybe vevő szolgáltatásnak kezelni kell az igénybe vett szolgáltatás megszűnéseit, megjelenéseit, anélkül, hogy újraindulna.

Ezen túl pedig meg kell adni még két metódust, amelyet szolgáltatás megjelenése (*bind*), illetve megszűnése (*unbind*) esetén hív meg a keretrendszer. Ezek segítségével folyamatosan nyomon tudjuk követni az igénybe vehető, éppen futó szolgáltatásokat.

A következő XML részlet publikálja az előző szakaszban ismertetett szolgáltatást, miközben felhasználja hozzá az OSGi Service Compendium gyűjteményben definiált naplózó szolgáltatást.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
    name="hu.bme.mit.riflab.osgi.impl.assigndeveloper">
  <implementation class="hu.bme.mit.riflab.osgi.nodes.impl.HelloWorldServiceImpl"/>
  <service>
    <provide interface="hu.bme.mit.riflab.osgi.interfaces.IHelloWorldService"/>
  </service>
  <reference interface="org.osgi.service.log.LogService"
    cardinality="1..1" policy="static"
    bind="setLog" unbind="unsetLog" name="LogService"/>
</scr:component>
```

Egy OSGi szolgáltatás definíciójának leírására a grafikus component editor használható, amivel tipikusan a projekten belül az OSGI-INF mappába érdemes létrehozni a szolgáltatást leíró xml fájlokat. A platform a deklaratív szolgáltatásokat az őket regisztráló köteg indulásakor kísérli meg elindítani. Amennyiben ilyenkor egy kötelező szolgáltatás nem érhető el, akkor a felhasználó köteg sem indul el.

Amennyiben a varázsló nem hivatkozná be szolgáltatás definíciókat, akkor induláskor semmi sem történne (fail silent). Az összes leíró hivatkozhatjuk a MANIFEST.MF következő sorával:

**Service-Component:** OSGI-INF/\*.xml

## 2.4 Tippek, futtatás, parancssor

### 2.4.1 Minta alkalmazás

Az OSGi deklaratív szolgáltatásokra példát is láthatunk, ha az Eclipse-ben új plug-in projektet hozunk létre:

New | Other | Plug-in project

Nevet kitöltjük, és a Target Platformnak bejelöljük az „an OSGi framework: Equinox” lehetőséget.



Next | Next | Create a plug-in using one of the templates: OSGi Declarative Services Example.

Figyeljük meg, hogy épül fel plugin. A MANIFEST.MF utolsó sorában szerepel a Service-Component sor. Az OSGI-INF mappa alatt a dictionary.xml írja le, hogy a DictionaryImpl osztály valósítja meg a Dictionary nevű szolgáltatást.

Az Activator.java tartalmazza azt a kódot, ami a DictionaryServiceImpl osztályt ajánlja ki, mint DictionaryService szolgáltatást. Ez azonban nem deklaratív leírás, pont ezt a kódolást váltja ki az xml fájlok szerkesztése, a mérés során ilyen kód írására nincs szükség.

A component.xml definiálja, hogy a ServiceComponent osztály igénybeveszi a DictionaryService szolgáltatást; illetve implementálja a CommandProvider szolgáltatást. Utóbbi arra jó, hogy az osztályban „\_” (underscore) karakterrel kezdődő metódusokat meghívja a rendszer, ha azt az OSGi console felületen kiadjuk. A függvények egyetlen paramétere egy CommandInterpreter objektum, amiből ki lehet nyerni a parancssorban megadott paramétereket. Az osztályban megvalósított getHelp() metódus által visszaadott String írja le az általunk megvalósított parancsok működését.

## 2.4.2 Futtatás

Az OSGi alkalmazások futtatásának egyik lehetséges módja az Eclipse futtatási beállítások használatával történhet. A Run | Run Configurations alatt új futtatási beállítást kell definiálni az OSGi Framework típusból.

- Nem kell semmiképp az összes köteget betölteni, ezért válasszuk a Deselect All menüpontot.
- A Workspace kötegek közül válasszuk a saját OSGi kötegeinket.
- Ne felejtjük al az org.eclipse.equinox.ds köteget is bejelölni, mivel ez kezeli a deklaratív szolgáltatásokat, e nélkül fail silent hiba jelentkezik.
- A „Validate Bundles” gombra kattintva valószínűleg hibát ír ki. Az „Add Required Bundles” viszont hozzáadja a függőségeket, ami után már hiba nélkül kell lefusszon az ellenőrzés.
- Run gombra kattintva lehet futtatni az OSGi alkalmazást.

## 2.4.3 Parancsok definiálása az Eclipse GoGo parancssor segítségével (Eclipse 4.2-től)

Az parancsot feldolgozó osztály sokkal egyszerűbb lesz, a paramétereket nem a CommandInterpreter objektumban kapjuk meg, hanem a rendszer feldolgozza. A help parancsonként adható meg (nem osztályonként mint eddig) annotációval, ehhez már függeni kell az org.apache.felix.gogo.runtime kötegtől. Egy ilyen parancsot megvalósító osztályt mutat a következő kód:

```
public class SimpleCommand {
    @Descriptor(value="A simple command that demonstrates OSGi felix console.")
    public void simpleCommand(
        @Descriptor(value="A string parameter to be echoed.")String message,
        @Descriptor(value="An integer parameter to be processed.")Integer value) {
        System.out.println(message);
        System.out.println(value+3);
    }
}
```

Az osztályt szolgáltatásként ki kell ajánlani, illetve egy scope-ot meg kell határozni, ami a parancs „névtere” lesz. Utána pedig a parancsokat kell újsorral elválasztott listában felsorolni. (A property nyitó és záró tagek között, nem pedig a value attribútumban, amit a rendszer egy darab parancs esetén ajánlana fel!)

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="osgiCommandTest">
  <implementation class="osgicommandtest.SimpleCommand"/>
  <property name="osgi.command.scope" type="String" value="simp"/>
  <property name="osgi.command.function" type="String">
    simpleCommand
  </property>
```



```
<service>
  <provide interface="org.osgi.commandtest.SimpleCommand"/>
</service>
</scr:component>
```

Parancs futtatása az OSGi parancssorból:

```
osgi> simp:simpleCommand "Hello world" 123
Hello world
126
```

A futtatáshoz Eclipse 4.2 szükséges, illetve futás közben az org.eclipse.equinox.ds mellett a következő kötegeket kell a futtatási konfigurációba bevenni:

- org.apache.felix.gogo.command
- org.apache.felix.gogo.runtime
- org.apache.felix.gogo.shell
- org.eclipse.equinox.console

#### 2.4.4 Futási szintek

A mérés során valószínűleg nem lesz rá szükség, de érdemes pár szót ejteni az OSGi futási szintekről. Minden köteghez egy nem negatív szám rendelhető (kivéve org.eclipse.osgi, amihez -1 van rendelve). Az OSGi környezet indításakor a nagyobb futási szinten lévő köteget később tölti be a kisebb futási szinten lévőnél. Ennek az a lényege, hogy ha valamilyen oknál fogva egy köteg más kötegek betöltöttségére számít, akkor így lehet késleltetni betöltődését. A legtöbb köteghez a 0 szint van rendelve, ami az alapértelmezett (default start level), amit az OSGi Framework Run Configuration kötet beállítások között ha megnézünk, 4. A rendszer induláskor a kis futási szintről egyesével az egyre magasabb futási szinten lévő kötegeket tölti be, míg el nem éri az osgi.startLevel értéket, ami Equinox esetében alapértelmezetten 6. (Ekkor egy 7-es futási szintre beállított köteg már nem töltődik be). A futási szintekkel a kötetek fejlesztésekor nem kell foglalkozni, hiszen különböző telepítések esetén különböző gépeken a köteg más és más futási szinthez tartozhat.

## 3 A mérés elvégzése

### 3.1 Mérési feladatok

A mérés során el kell készíteni a munkafolyamat *csomópontjait* OSGi *deklaratív szolgáltatásokként*. Fontos, hogy ezen szolgáltatások implementációja legyen teljesen független egymástól (leszámítva természetesen a paraméterek típusának egyeztetését)! A szolgáltatások állapotának befolyásolhatósága érdekében szükséges, hogy ezek a csomópontok *ne egy közös bundle* belsejében legyenek megvalósítva, hanem legalább kettő vagy esetleg három kötegbe legyenek szétosztva. A munkafolyamatok átlapolódásának kezelése nem feladat.

Biztosítsunk kézi vezérlést a munkafolyamathoz! A vezérlést kétféle módon lehet biztosítani:

- A meglévő grafikus felületet átalakításával, hogy a szolgáltatásokat használják.
- A munkafolyamat OSGi parancsok sorozataként való megvalósításával.

(Az OSGi LogService használata mindkét esetben pozitív elbírálásban részesül).

Figyeljük meg, hogy a futó bundle leállítása hogyan befolyásolja a munkafolyamat elérhetőségét. Ehhez változtassuk meg a folyamat csomópontjainak kardinalitását is. Bemutatáskor meg kell mutatni, hogyan kezeli a munkafolyamatot megvalósító alkalmazás, ha egy bundle leáll, és így a kapcsolódó szolgáltatások nem elérhetőek.

## 3.2 A mérés értékelése

A mérés után szükséges, hogy bemutassuk a munkafolyamatot futás közben, valamint egy mérési jegyzőkönyvet is kell készíteni.

A mérési jegyzőkönyvnek tartalmaznia kell:

- a munkafolyamat leírását (képpel)
- megvalósításának magas szintű, architekturális leírását
- valamint a lényeges implementációs részeket kiemelve.

A forráskódot exportált Eclipse projekteként kell mellékelni, a jegyzőkönyvben legfeljebb különösen érdekes részeket kell kiemelni.

Az értékelés részben a kódminőség, részben a jegyzőkönyv alapján történik. A kódminőséghez hozzátartozik a megfelelő hibakezelés (a kivételek kiírása a normál kimenetre nem számít hibakezelésnek), a forráskód olvashatóság (beleértve a tördelést is, ehhez ajánlott az automatikus kódtördelés használata).

## 4 További segédanyagok

Az anyag mélyebb megértéséhez, illetve a felmerülő kérdések megválaszolásához további források is elérhetőek.

*Erősen ajánlott* az Equinox Console, illetve a deklaratív szolgáltatásokról szóló leírások alaposabb áttanulmányozása a mérés előtt.

- OSGi Service Platform Core Specification 4.2 (2009. június) és OSGi Service Platform Service Compendium 4.2 (2009. augusztus): <http://www.osgi.org/Specifications/HomePage>
- [OSGi](#) – Segédanyag az Eclipse alapú technológiák tárgyhoz
- How to get Started with OSGi: <http://www.osgi.org/About/HowOSGi>
- Neil Bartlett: OSGi and How It Got That Way <http://njbartlett.name/2010/03/07/osgi-and-how-it-got-that-way.html>
- Sarath Chandra: Using Equinox CommandProvider to make OSGi console interactive <http://blog.sarathonline.com/2008/12/using-equinox-commandprovider-to-make.html>
- Chris Aniszczyk: Explore Eclipse's OSGi Console <http://www.ibm.com/developerworks/library/os-ecl-osgiconsole/index.html>
- Chris Aniszczyk: [OSGi Declarative Services](#)

## 5 Ellenőrző kérdések

1. Milyen csomagok elérhetők el egy OSGi bundle kódjának írásakor?
2. Miért előnyös az importált és exportált csomagok kézi felsorolása OSGi használatakor?
3. Melyik feladatra használandó az Activator osztály: szolgáltatások regisztrációja vagy szolgáltatásokhoz példány megszerzése? Miért?
4. Mi a ServiceTracker osztály feladata?
5. Mivel azonosíthatjuk a szolgáltatásokat a deklaratív szolgáltatások igénybevételekor?
6. Mire használjuk a *bind* és *unbind* metódusokat deklaratív szolgáltatások definiálásakor és/vagy felhasználásakor?