Rendszerintegráció és -felügyelet laboratórium (VIMIM309)

OSGi szolgáltatások fejlesztése

Mérési segédlet

Készítette: Ujhelyi Zoltán

Utolsó módosítás: 2011. március 25.

Verzió: 1.0

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)

Szabályalapú üzleti logika

Üzleti folyamatok felügyelete

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 implementáció felett is hasonló módon lehet megalkotni.

2.1 OSGi kötegek

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.

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.[1] A leíró ezen felül tartalmazza a függőségek és az exportált csomagok listáját.

[1] 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.

A komponensek között függőségi viszonyt lehet definiálni, azaz előírni komponensek egy halmazát, amelyek nélkül nem lehet a saját kódot futtatni. Ez a függőségi viszony megadása kötelező, ugyanakkor a keretrendszer felhasználja ezt az információt a kötegek elérhetőségének megállapítására.[2]

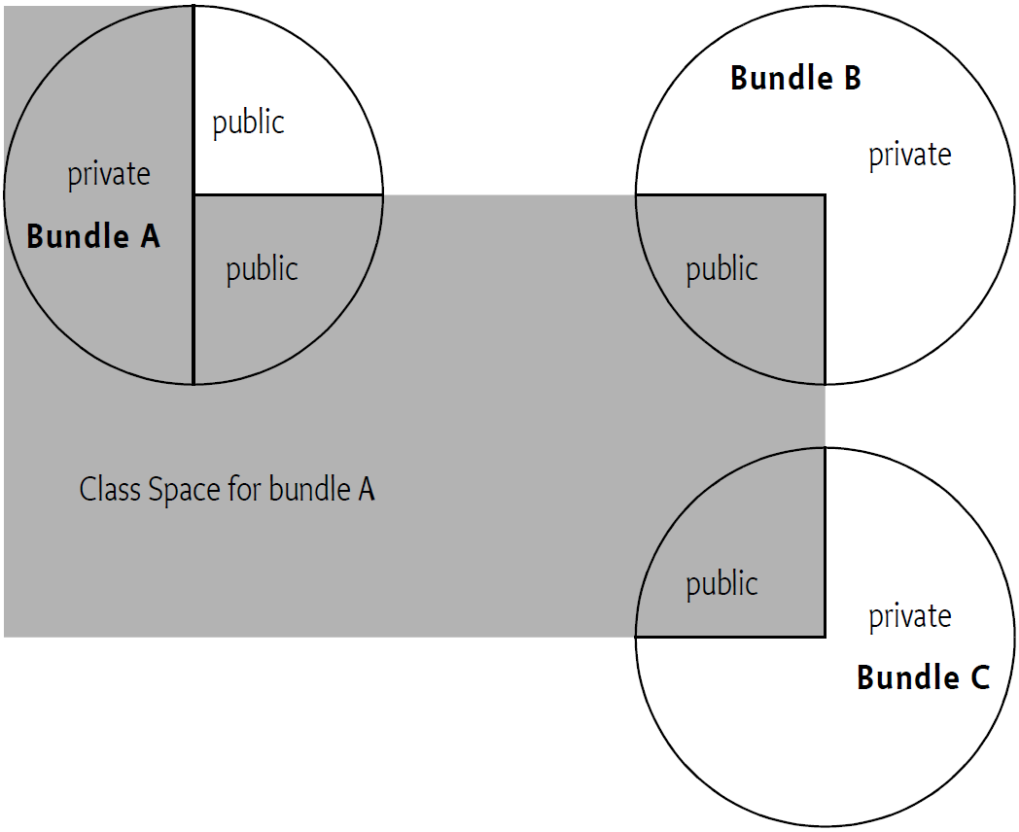
A kötegek elősegítik az információrejtést, azaz szabályozható, hogy pontosan milyen *csomagok* (package) mely verziói érhetőek el a kötegen kívülről. Alapértelmezetten minden csomag rejtett a többi köteg számára, explicit fel kell sorolni a nyilvános csomagokat. A megoldás célja, hogy támogassa a kötegek frissítését és cseréjét azáltal, hogy a belső viselkedést megvalósító osztályokat elrejtse a többi köteg elől.[3]

Ezek alapján egy köteg kódjában háromféle csomag érhető el:

A köteg által definiált csomagok

A Java keretrendszer függvénykönyvtára

A függő kötegek nyilvános (megosztott) csomagjai



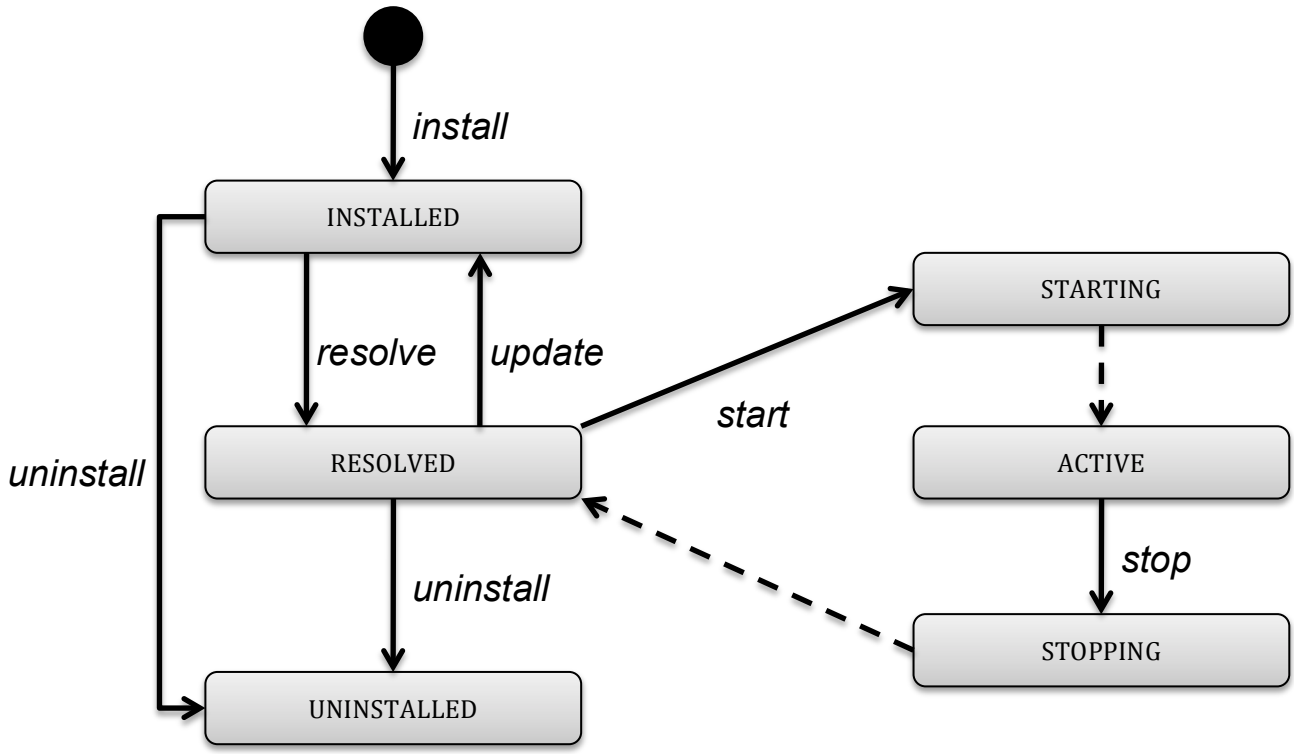
1. ábra Egy köteg által elérhető csomagok

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 – minden egyes csomag importhoz keres a keretrendszer egy másik köteg által felajánlott nyilvános csomagot, amely megfelel az opcionális verziókényszereknek is.[4]

[2] 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.
[3] Mellesleg így lehetőség van egy Java függvénykönyvtár több verziójának betöltésére is – minden köteg csak azt a verziót látja, amivel együtt tud működni.
[4] Ez a futásidejű csomagfeloldás 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; OSGi esetén viszont ez ellenjavallott.

Az OSGi lehetővé teszi egy különleges, ún. *Activator* osztály regisztációját. Ez az osztály callback metódusokat definiál: a *start* és *stop* metódusát meghívja az OSGi keretrendszer a köteg elindulásakor, illetve leállásakor. Ezen metódusok biztosítják az egyetlen garantált be- illetve kilépési pontot a megfelelő kötegből, ennek megfelelően gyakran használatosak.



2. ábra: Kötegek életciklusa

2.2 Szolgáltatások megadása

A kötegek mellett a másik fontos OSGi által nyújtott eszköz az OSGi szolgáltatások használata. 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 szolgáltatások. Egy OSGi framework egy ilyen szolgáltatás tárolót használ. A szolgáltatások azonosítására egy szöveges azonosító tartozik, ami a szolgáltatást nyújtó interfész típusával szokás megadni.

Szolgáltatások tipikusan 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 adatrejtés érdekében. Fontos, hogy az interfész és az implementációs osztály nem kell, hogy azonos kötegben legyen (gyakran nem is így történik), mindössze az kell, hogy elérhető legyen az interfészt definiáló csomag.

Az így definiált szolgáltatások nagy előnye, hogy nincsenek köteghez kötve: ha egy szolgáltatást több köteg is megvalósít, akkor lehetőség van akár az összes implementáció használatára, vagy pedig választásra tetszőleges logika szerint.[5]

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

[5] A választás egy lehetséges módja a megfelelő szolgáltatást nyújtó kötegek kézi elindítása, ill. leállítása (programozottan ez ugyanakkor nem javasolt).

2.2.1 Szolgáltatások definiálása

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

```
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 érhető el, amihez a *ServiceRegistration* komponenshez hasonlóan a köteg Activator osztályában férhetünk hozzá.[6]

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 {
```

[6] 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.

```
	private final BundleContext context;
	private List<IHelloWorldService> services;
	public HelloworldServiceTrackerCustomizer(BundleContext context) {
		this.context = context;
		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.

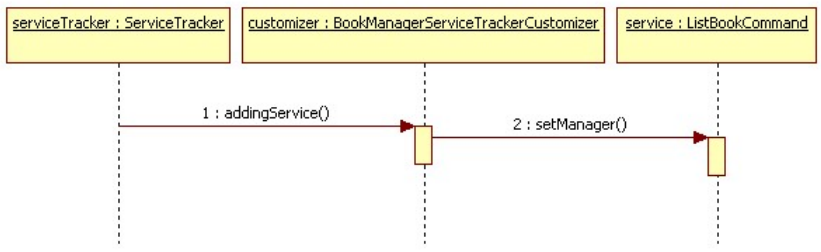
```
public class Activator implements BundleActivator {
	ServiceTracker serviceTracker;
	@Override
	public void start(BundleContext context) throws Exception {
		ServiceTrackerCustomizer customizer = new
HelloworldServiceTrackerCustomizer(context);
		serviceTracker = new ServiceTracker(context, IHelloWorldService.class
				.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, hogy 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 láthattuk, OSGi szolgáltatások esetén viszonylag sok, monoton ismétlődő kódot kell írni. Az OSGi 4.2-es verziójától kezdve lehetőség van a kézi szolgáltatásregisztráció helyett egy deklaratív, XML-alapú *komponens definíció* felhasználásával regisztrálni és elérni a szolgáltatásainkat (függetlenül attól, hogy az deklaratív vagy nem 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 felhasználóihoz egyaránt definiálni lehet. Előbbi esetben csak a szolgáltatásra jellemző interfészt kell megadni (ebből származtatja a szolgáltatás azonosítóját – hasonló módon, ahogy az előző szakaszban kézzel kellett), míg utóbbi esetben több adat megadása is szükséges.

Ahhoz, hogy egy már létező szolgáltatásra hivatkozhassunk, 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 feltételezheti, hogy belőle mindig egy példányt használnak (amíg létezik), így állapottal rendelkező szolgáltatás esetén mindig ilyet használjunk; dinamikus esetben a keretrendszer készíthet újabb példányokat a szolgáltatásból menet közben.

Ezen túl pedig megadhatunk még két metódust, amely szolgáltatás megjelenése (*bind*), illetve megszűnése (*unbind*) esetén kell meghívnia a szolgáltatás során.

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. Ez gyakran elfogadható viselkedés, de ha a konkrét esetben nem, akkor a szolgáltatások kézi regisztrációja szükséges.

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"/>
   <reference bind="setLog" cardinality="1..1"
interface="org.osgi.service.log.LogService" name="LogService" policy="static"/>
   <service>
      <provide interface="hu.bme.mit.riflab.osgi.interfaces.IHelloWorldService"/>
   </service>
</scr:component>
```

3 A mérés elvégzése

3.1 Mérési feladatok

A mérés során el kell készíteni a munkafolyamat csomópontjait OSGi 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.

Készítsünk egy konzolos parancsot, amely a folyamat összes csomópontján automatikusan végigmegy, és ez követhető a szöveges kimenetben (az OSGi LogService használata pozitív elbírálásban részesül). Itt nem szükséges, hogy kezeljük több munkafolyamat átlapolódását.

Figyeljük meg, hogy a futó bundle leállítása hogyan befolyásolja a munkafolyamat elérhetőségét. Ehhez változtassuk meg a folyamat csomópontjainak kardinalitását is.

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

Alakítsuk át úgy a meglevő grafikus felületet, hogy a szolgáltatásokat használják.

Valósítsuk meg a munkafolyamatot OSGi parancsok sorozataként.

Az alkalmazás mindkét esetben kezelje, ha egy bundle leáll, és így a kapcsolódó szolgáltatás nem elérhető.

3.2 A mérés kiértékelése

A mérés után szükséges, hogy bemutassuk a munkafolyamatot futás közben, valamint egy mérési jegyzőkönyv készítése.

A mérési jegyzőkönyvnek tartalmaznia kell a munkafolyamat leírását (képpel), megvalósításának magas szintű, architekturális leírását, valamint a lényeges implementációs részeket kiemelve. A forráskódot exportált Eclipse projektekként 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 tördelést is – ajánlott az automatikus kódtördelés használata).

4 További segédanyagok

Az anyag mélyebb megértéséhez, illetve a felmerülő kérdések megválaszolásához a 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?