Patrik Storm

OSGi-ramverket för Java

Kandidatavhandling
Handledare: Johannes Eriksson
Institutionen för
informationsteknologi
Åbo Akademi
Åbo 2011

Referat

Den här avhandlingen kommer att gå igenom vad OSGi-ramverket är för något och vad det används till. Först beskriver jag OSGi-ramverket och dess dynamiska moduler och OSGi-alliansen. Sedan följer en genomgång av OSGi-ramverkets grundprinciper där modulsystemet, dynamiken och tjänstmodellen förklaras. Modulerna kallas för Bundler och de kommer att beskrivas i detalj. I den beskrivningen kommer strukturen, versionshanteringen och livscykeln grundligt att behandlas, eftersom modulerna är en mycket viktig del av OSGi-ramverket. Vidare kommer också OSGi-ramverkets tjänstmodell att förklaras. OSGi-tjänsterna, arkitekturen, säkerheten och olika standardtjänster behandlas. I slutet av avhandlingen kommer också olika implementationer av OSGi-ramverket att beskrivas.

Sökord: OSGi-ramverket, Bundle, OSGi-alliansen, tjänster, Eclipse.

Innehåll

1 Inledning

Programvaruprojekt blir allt mer komplexa och växer i storlek för varje år. Ett projekt kan bestå av miljontals rader programkod och gruppen som skapar programvaran kan bestå av flera hundra personer. T.ex. uppskattas det att Windows Vista innehåller 50 miljoner rader programkod [1]. Då projekten blir stora måste det finnas något sätt att få kontroll över hur hela systemet fungerar. Antalet klasser och paket blir till slut enormt och för att få en överblick av det hela måste man skapa s.k. moduler. Modulerna gör att man kan dela upp arbetet i mindre delar som grupper eller enskilda programmerare kan jobba med. Det är också mycket viktigt att alla beroenden mellan modulerna uppfylls så att systemet fungerar korrekt. Det bör också vara enkelt att installera och uppdatera programvara. Det ska också vara möjligt att sprida ny programvara på ett effektivt sätt.

För att det ska vara möjligt att skapa moduler med ovannämnda egenskaper krävs det att man har specifika krav som modulerna måste uppfylla. Det ska finnas versionshantering, vilket betyder att moduler har olika versionsnummer och att moduler kan använda sig av andra moduler med ett visst versionsomfång. Det är även bra att ha en standardiserad tjänstmodell så att man enkelt kan hålla reda på vilka tjänster modulerna registrerar och använder. Tjänsterna är ett enkelt sätt att dela ut funktionalitet till andra moduler. Modulerna bör även ha en livscykel så att man kan hålla reda på i vilket tillstånd modulerna är. En annan viktig funktion är att modulerna ska vara dynamiska. Det betyder bland annat att det ska vara möjligt att installera programvara när ett system är i gång.

OSGi-ramverket skapades av OSGi-alliansen för att göra det enklare för de som arbetar med stora projekt. Med OSGi-ramverket skapar man moduler som kallas för Bundler, eng. *Bundles*. Genom att dela upp hela systemet i dessa Bundler löser man en del av problemen som uppstår med stora projekt. OSGi-ramverket har versionshantering som gör att modulernas beroenden hanteras på rätt sätt. Versionhanteringen behandlas i kapitel 4.2. OSGi-ramverket gör det möjligt att

skapa dynamiska lösningar. Det betyder att OSGi-ramverkets Bundler kan installeras, startas, stoppas, uppdateras och avinstalleras utan att man behöver starta om systemet. OSGi-ramverkets dynamiska moduler behandlas i kapitel 2.2.

I den här avhandlingen kommer jag att gå in på OSGi-ramverkets dynamiska egenskaper och om det är möjligt att skapa modulariserade system med OSGi-ramverket. Jag kommer att behandla kärnprinciperna, Bundlerna och OSGi-tjänstplattformen i detalj. I slutet blir det även en uppräkning av några vanliga implementationer av OSGi-specifikationen. I det stora hela blir det en grundlig beskrivning av OSGi-ramverket.

2 Beskrivning av OSGi

Det här kapitlet skall handla om OSGi-ramverkets grunder. Ramverket, de dynamiska modulerna och OSGi-alliansen förklaras och dessutom utreds varför det är bra att använda sig av OSGi-ramverket.

2.1 Ramverket

Ramverket specificerar OSGi-tjänstplattformens specifikationer. OSGi-ramverket är ett Java-ramverk som stöder spridning av utbyggbara och nerladdningsbara program. Apparater som är OSGi-kompatibla kan ladda ner och installera OSGi-bundler och avinstallera dem när de inte längre behövs. Ramverket sköter om installationen och uppdateringen av Bundler. Dessutom håller ramverket reda på beroenden mellan olika Bundler och tjänster i detalj. [2]

Ramverket gör det möjligt för Bundler att växa under körning, eftersom det är möjligt att välja mellan olika implementationer som finns tillhanda i tjänstregistret. Bundler kan registrera nya tjänster, få meddelanden om förändringar om tillstånd hos tjänster eller söka efter nya tjänster för att lägga till ny funktionalitet i det existerande programmet. Ifall man behöver ny funktionalitet så är det bara att installera en ny Bundle eller modifiera en existerande Bundle. Det här går att göra under körning av programmet. Man behöver alltså inte starta om Javas virtuella maskin. [2]

2.2 Dynamiska moduler

OSGi-ramverket skapades av OSGi-alliansen för att få fram ett bra modulsystem för Java. De ville att det skulle bli enklare att skapa programvara som består av hundratusentals och ibland t.o.m. miljontals rader programkod. OSGi-ramverket är ett dynamiskt modulsystem för Java. Bundlerna kommer att behandlas i kapitel 4. Ändamålet med OSGi-ramverket är att man kan installera, uppdatera och avinstallera Bundler under körning av ett program. Det här gör det möjligt för

användaren att uppdatera sitt program utan att behöva starta om programmet. Det här är också mycket användbart i t.ex. serverapplikationer där en omstart skulle medföra komplikationer för många som använder applikationen via nätet. [1]

Modullagret i OSGi-ramverket bestämmer på vilket sätt klasserna laddas. Traditionellt har man bara en s.k. classpath i Java som bestämmer var alla klasser och resurser finns. Med OSGi-ramverket kan man begränsa synligheten till klasser inom moduler och man får också kontrollerad länkning mellan modulerna. [3]

Tyvärr är vissa lite skeptiska till OSGi-ramverkets dynamiska kapacitet, eftersom erfarenheten har visat att J2EEs EAR-moduler är begränsade och opålitliga [1]. EAR-moduler (Enterprise Archive) är också standard JAR-filer (Java Archive) med ett .ear-filtillägg [4].

2.3 OSGi-alliansen

OSGi-standarden skapades av en allians som består av ungefär 40 företag. OSGi stod förr för ”Open Systems Gateway initiative”, men nuförtiden står det inte för någonting, eftersom ramverket har expanderat mycket sedan början då det endast var menat för förmedlingsnoder i hemmen [1]. Specifikationen definierar ett antal tjänster som en OSGi-container måste implementera och ett kontrakt mellan containern och applikationen [5].

Det är OSGi-alliansen som bestämmer specifikationerna för nya versioner av plattformen och godkänner aktuella implementationer av specifikationen. Det tekniska arbetet sköts av ett antal expertgrupper Expert Group (EG), som inkluderar expertgrupper för ramverkets kärnfunktioner, mobila system, tillämpningar i fordon och större mjukvarusystem. [1]

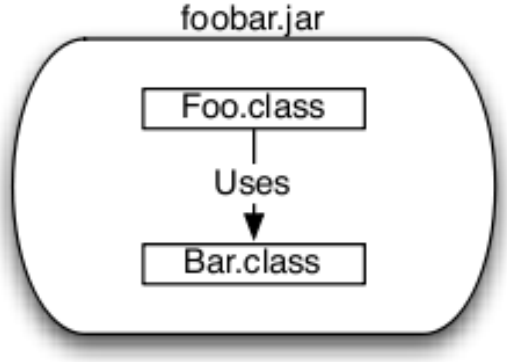
3 Kärnprinciper

Det här kapitlet kommer att beskriva de viktigaste principerna i OSGi-ramverket. Modulsystemet, dynamiken under körning och tjänstmodellen hör till de principer som karakteriserar OSGi-ramverket.

3.1 Modulsystem

Modularitet är mycket viktigt när man skapar större program. En modul skall ha hög sammanhållning och låg koppling. Hög sammanhållning betyder att modulen ska fokusera på en sak och inte göra något annat. Låg koppling innebär att modulen endast ska vara beroende av andra moduler genom stabila abstraktioner. Låg koppling gör det enkelt att byta ut moduler, eftersom de endast vet om varandra via gränssnitt. Man måste dock se till att gränssnitten hålls intakta. Hög sammanhållning gör det enklare att förstå enskilda moduler och det medför också att man börjar förstå hela systemet bättre. Testning kan också göras bättre då man kan testa modulerna åtskilt. Det är också möjligt att öka återanvändningen genom att ha ett modulariserat system. Det kan t.ex. vara möjligt att ta en modul från ett program och använda den till ett annat program. [6]

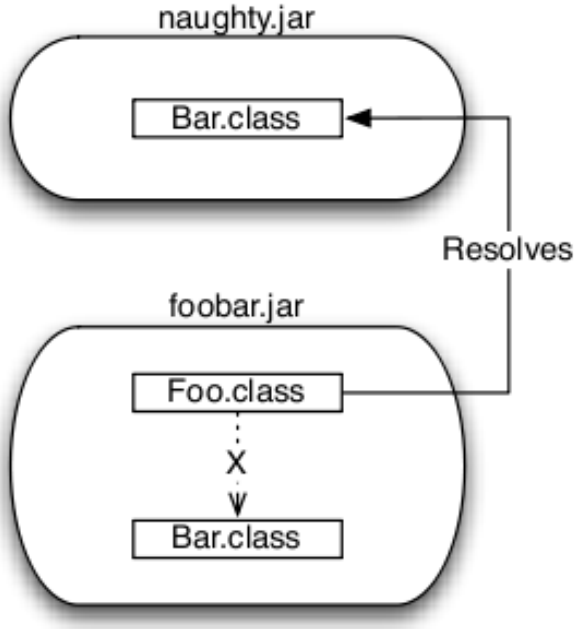
Tyvärr räcker inte Javas inbyggda funktionalitet till för att skapa modulariserade lösningar. I Java modulariseras instruktionerna till metoder som sedan modulariseras till klasser. Efter det kan klasserna även modulariseras till paket och hela programmet kan sedan spridas som en JAR-fil. Nedan följer en förklaring till att man med Javas klassladdning och JAR-filer inte kan skapa ett modulariserat system. [6]



Figur 3.1: Exempel på internt beroende i en JAR-fil.
Källa: Bartlett, Neil, OSGi In Practice, 2009

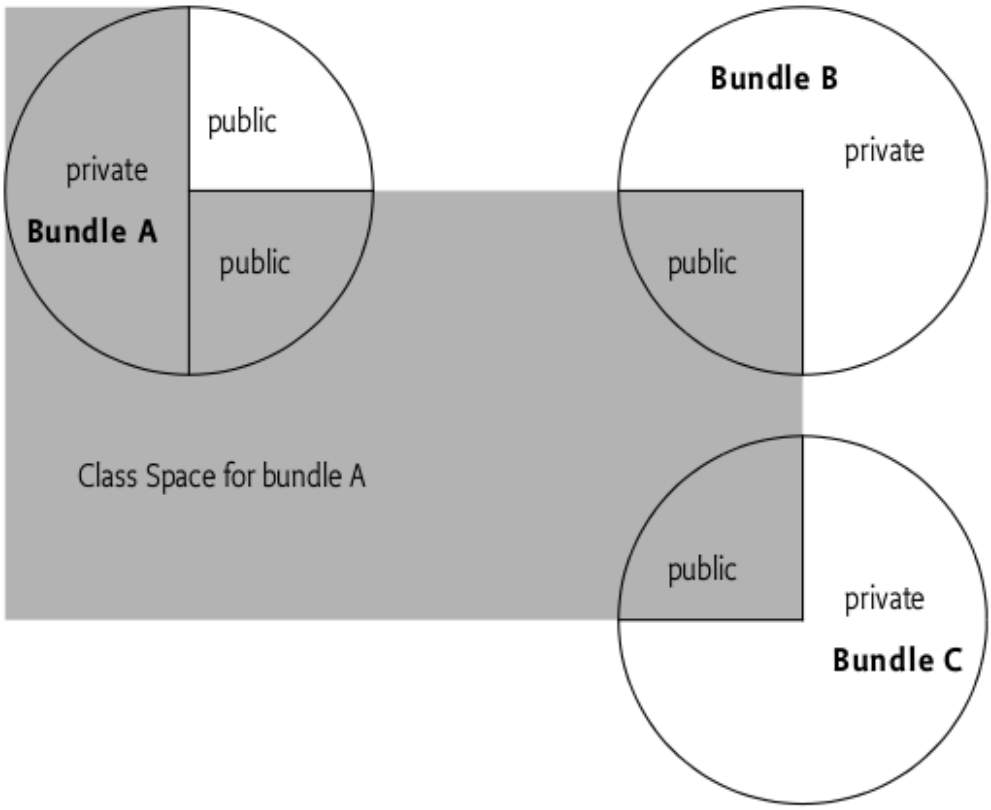
Man kan tänka sig att Javas JAR-filer är ett sätt att skapa moduler men tyvärr fungerar inte JAR-filerna så som en modul skall göra. En JAR-fil är egentligen bara ett enkelt sätt att sprida ett antal klasser, gränssnitt och andra resurser. När en JAR-fil läggs till classpathen försvinner alla gränser och all modularitet. Det här betyder att alla publika klasser i JAR-filen kan nås av alla andra klasser i klassrymden. Följden är att man med JAR-filer inte kan gömma undan modulernas interna implementationer. Med JAR-filer finns det heller inte något sätt att hantera olika versioner av moduler. [6]

I Figur 3.1 finns en JAR-fil som innehåller två klasser. Klassen Foo har skapats med antagandet att den kommer att använda sig av klassen Bar i samma JAR-fil. Då JREn (Java Runtime Environment) skapar sig en överblick av de klasser som finns i klassrymden läggs de alla i en lista. Då JREn söker efter en klass i listan så kommer den att välja den första med rätt namn. I Figur 3.2 kan vi se en annan JAR-fil som innehåller klassen Bar. Eftersom klassrymden nu innehåller två olika versioner av klassen Bar så kan det hända att klassen Foo resolverar till fel version av klassen Bar. Det här medför att systemet kanske inte kommer att fungera enligt specifikationerna. [1]



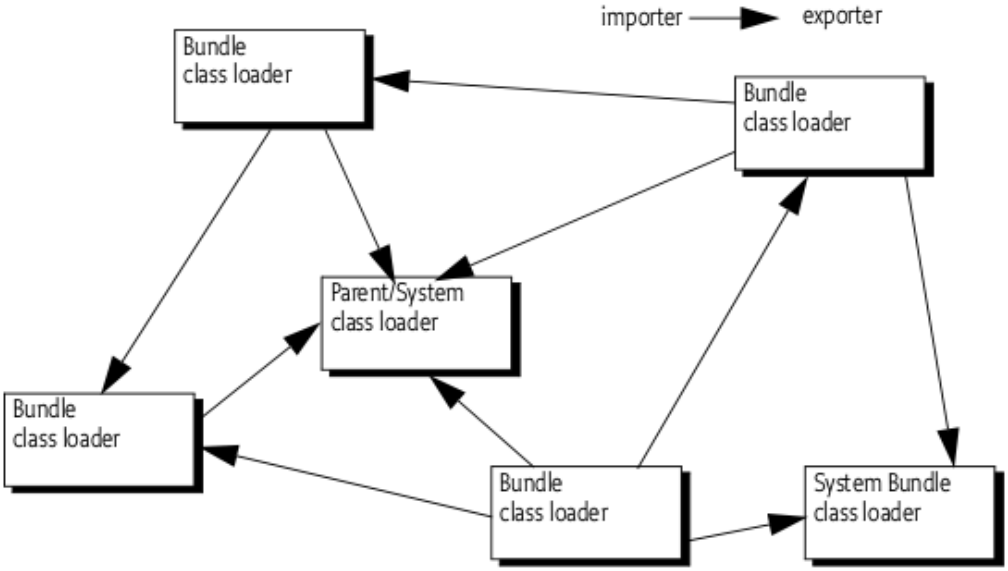
Figur 3.2: Exempel på internt beroende som inte uppfylls. Källa: Bartlett, Neil, OSGi In Practice, 2009

I OSGi-ramverket används även JAR-filer för att skapa modulerna men man lägger till några metadata som t.ex. namn, version och beroenden. OSGi-ramverkets Bundler kommer att gås igenom i detalj i kapitel 4. Bundler kan installeras, startas, uppdateras, stoppas och avinstalleras. Det här kallas för en livscykel och den kommer att gås igenom i kapitel 4.3. [6]



Figur 3.3: Exempel på en klassrymd. Källa: OSGi-alliansen.

I Figur 3.3 ovan kan man se hur klassrymden för Bundle A kan se ut. Alla Bundler har en egen klassladdare som bestämmer vilka klasser som skall laddas. Klassladdaren kan ladda klasser och resurser från *Boot classpath*, *Framework classpath* och *Bundle space*. Bundlens klassrymd blir då alla klasser som klassladdaren kan nå. Klassrymden kan då bestå av klasser från *Boot classpathen*, importerade paket, Bundler som behövs, Bundlens *classpath* och bifogade Bundlefragment. Bundlefragment behandlas i kapitel 4.4. I Figur 3.3 kan man se hur klassrymden består av Bundle As privata och publika klasser och publika klasser från Bundle B och C. I Figur 3.4 kan man se olika Bundler och hur deras klassladdare laddar klasser från andra Bundler samt systemklassladdaren. [2]

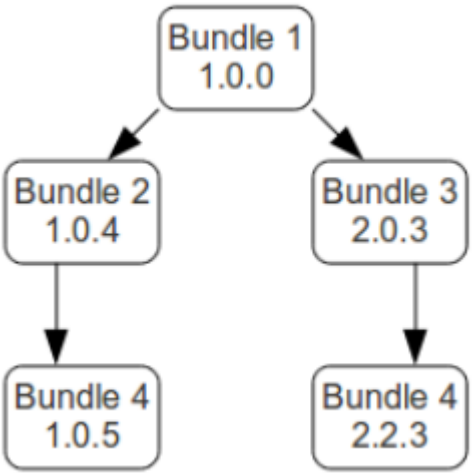


Figur 3.4: Modell över hur klassladdare delegerar. Källa: OSGi-alliansen

3.2 Dynamik under körning

En av kärnprinciperna i OSGi-ramverket är att under körning av ett program kunna installera, starta, stoppa, uppdatera och avinstallera moduler utan att behöva starta om Javas virtuella maskin (JVM, Java Virtual Machine). Då det sker en ändring i systemet blir de berörda modulerna informerade och sedan ändrar de sitt tillstånd enligt direktiven. Med OSGi-ramverket kan man alltså enkelt uppdatera programkoden på små inbyggda system, bordsdatorer och servrar. Följden är att man kan ha system i gång trots att man ändrar på programmen. [7]

Det är även möjligt att ha flera olika versioner av samma Bundle i OSGi-ramverket. Det här går eftersom alla Bundler har en egen klassrymd, vilket förklarades i kapitel 3.1. I Figur 3.5 nedan kan man se två olika versioner av Bundle 4. Utan OSGi är man tvungen att välja en version av Bundle 4 och hoppas på att den fungerar med Bundle 2 och Bundle 3. Med OSGi kan man däremot ha kvar båda versionerna av Bundle 4. Bundle 2 och Bundle 3 använder då den version av Bundle 4 som har angivits i metadata som de har. Metadata beskrivs närmare i kapitel 4.1. [6]



Figur 3.5: Flera versioner av samma Bundle.

3.3 Tjänstmodell

Tjänstmodellen i OSGi-ramverket är en s.k. publicera, hitta och förena modell [2]. Då en Bundle har aktiverats i OSGi-ramverket så finns det plötsligt ny funktionalitet som andra Bundler kan använda sig av. Problemet är att få alla andra Bundler medvetna om den nya funktionaliteten som finns till förfogande. I OSGi-ramverket förverkligas detta genom att ha ett tjänstregister. Registret länkar ihop olika Bundler dynamiskt och håller reda på deras tillstånd och beroenden. [7]

Genom tjänstregistret kan olika Bundler registrera objekt i tjänstregistret, söka efter andra objekt och motta meddelanden när tjänster blir registrerade eller avregistrerade. Objekt som har blivit registrerade i tjänstregistret kallas för tjänster. En tjänst registreras alltid med ett gränssnitt och ett antal egenskaper. Namnet på gränssnittet skall vara beskrivande så att man vet vad tjänsten har för betydelse. Med hjälp av de olika egenskaperna beskrivs tjänsten närmare för de som ämnar använda den. [7]

Tjänstregistret i OSGi-ramverket är ett s.k *in-memory registry*. De registrerade tjänsterna är dynamiska och beroende av exekveringstillståndet hos Bundlerna. Då en Bundle stoppas blir alla dess tjänster avregistrerade. På samma gång meddelas alla påverkade Bundler. Bundlerna hittar nya tjänster genom att få meddelanden om nya tjänster eller genom att aktivt söka efter tjänster med de rätta egenskaperna. [7]

Tjänstregistret gör det också möjligt att stöda program som är byggda enligt en tjänstorienterad arkitektur, SOA (Service Oriented Architecture). OSGi-ramverket är en sorts tjänstorienterad arkitektur. Med OSGi-specifikationen kommer flera kärntjänster som kan tas med i ramverket. T.ex. en HTTP-tjänst och en loggtjänst. Några standardtjänster kommer att beskrivas i kapitel 5.4. [6]

4 Beskrivning av en Bundle

I OSGi-ramverket kallas modulerna för Bundler. En Bundle består av Java-klasser och resurser. När en Bundle har startats kan den dela ut tjänster till andra Bundler installerade i OSGi-tjänstplattformen. Det här kapitlet kommer att förklara hur modulerna är uppbyggda och hur man hanterar olika versioner av modulerna. [5]

4.1 Uppbyggnad

En Bundle består av en JAR-fil men skiljer sig från vanliga JAR-filer genom att den innehåller några metadata som lagts till av OSGi-ramverket. Det är i METAINF/MANIFEST.MF-filen som det läggs till ytterliga rubriker. Vanliga Java-program kan också använda dessa moduler, eftersom de oförstådda rubrikerna då ignoreras.[1]

```
1 Manifest-Version: 1.0
2 Created-By: Patrik Storm
3 Bundle-Manifest Version: 2
4 Bundle-Name: Min första OSGi-Bundle
5 Bundle-SymbolicName: org.osgi.ramverket
6 Bundle-Version: 1.0.0
7 Bundle-Required ExecutionEnvironment: J2EE-1.3
8 Import-Package: A;version="[1,2)"
9 Export-Package: B;version=1.5.1
```

Listning 4.1: En OSGi-MANIFEST.MF-fil.

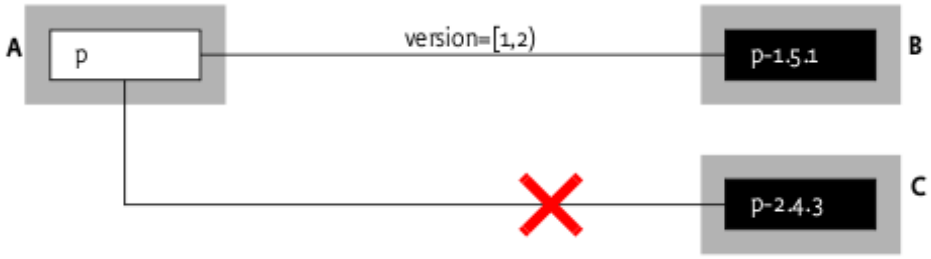
I programkodslistningen 4.1 finns de vanligaste attributen som används i OSGi-ramverket. De två kursiverade raderna är de enda som hör till MANIFEST.MF-filen i JAR-filer och det är endast attributet ”Manifest-Version” som är obligatoriskt i JAR-filernas MANIFEST.MF-fil. Det går att lägga till flera valfria attribut i JAR-filens MANIFEST.MF-fil enligt JAR-specifikationen men

utomstående ramverk som OSGi får lägga till egna attribut. I programkodslistningen 4.1 hör alla andra attributen till OSGi-ramverket men de flesta är valbara. Det är endast obligatoriskt att ha med attributet ”Bundle-SymbolicName”. [1]

4.2 Versionshantering

OSGi-ramverket sköter beroenden mellan moduler genom att ha namn på modulerna och genom att ha ett versionsnummer i modulerna. Versionsnummer gör det enklare att hålla reda på olika publikationer av modulerna så att det inte uppstår komplikationer. [1]

OSGi-ramverket klarar av att använda flera olika versioner av samma moduler, vilket är unikt. Export-paketen får ett versionsnummer och import-paketen ett omfång där man definierar vilka versioner som modulen kan använda. [8]



Figur 4.1 Exempel på versionshantering. Källa: OSGi-alliansen.

I Figur 4.1 ovan ser man export-paketet B som har versionsnumret 1.5.1 och import-paketet A som har omfånget [1,2). Export-paketet C har versionsnumret 2.4.3 och faller därför inte inom paket As omfång. Alla dessa beroenden måste uppfyllas för att modulen skall kunna installeras. Det här gör att det inte installeras moduler som inte kommer att fungera p.g.a. av att versionerna inte stämmer överens.

Versionsnumren delas upp i fyra delar: *major*, *minor*, *micro* och *qualifier*, t.ex. 1.2.3.beta4. [8]

1. *Major* – Då två paket har två olika nummer som *major*-del så är de helt inkompatibla.

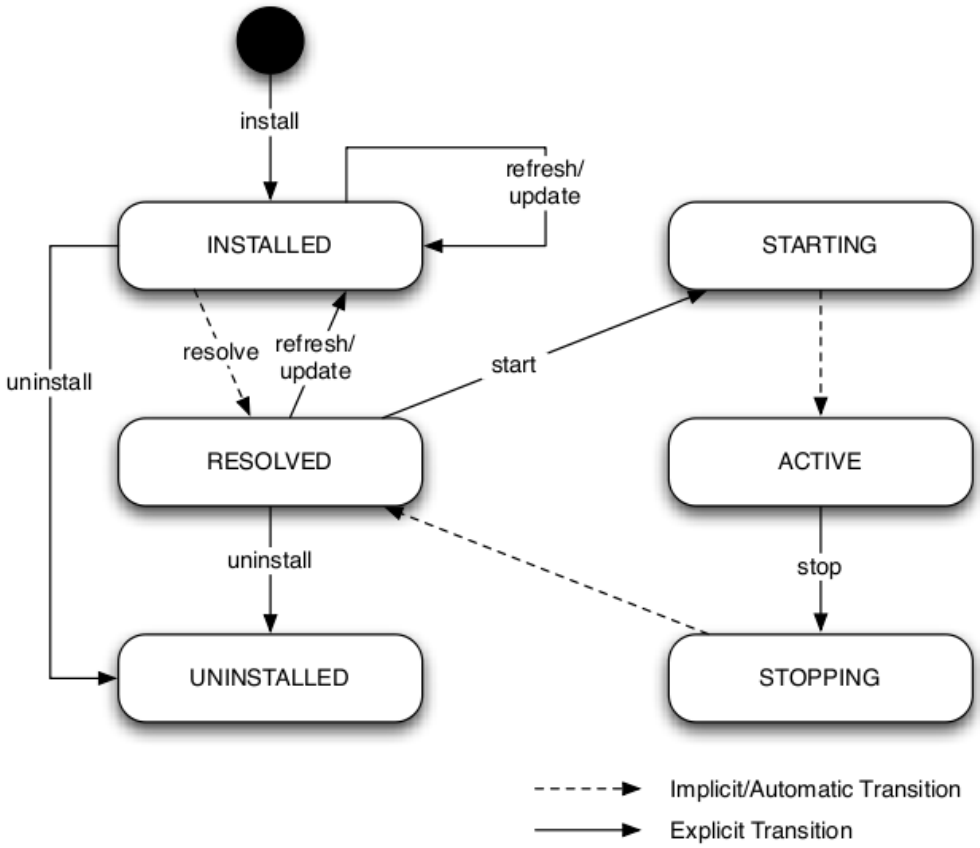
2. *Minor* – De som använder APIn är kompatibla med import-paket som har samma *major*-del och en *minor*-del som är större än eller lika med. De som förmedlar APIn är kompatibla med import-paket som har samma *major*-del och *minor*-del.

3. *Micro* – Ändring i den här delen introducerar inte inkompatibla paket. Den här delen används bara för att fixa buggar som inte inverkar på APIn.

4. *Qualifier* – Den här delen anger i vilket stadium publikationen är, t.ex. i beta-stadium.

4.3 Livscykel för en Bundle

En Bundle har olika tillstånd beroende på om den t.ex. nyligen har blivit installerad eller är i körning. En Bundle kan ha tillstånden *Installed*, *Resolved*, *Starting*, *Active*, *Stopping* och *Uninstalled*. Om man använder sig av Equinox-implementationen av OSGi-ramverket så kan man se i vilket tillstånd en Bundle är genom att ge kommandot `ss` [6]. Tillstånden *Starting* och *Stopping* är s.k. övergångstillstånd som kan vara svåra att se, eftersom en Bundle har dessa tillstånd i en mycket kort tid.



Figur 4.2: Tillståndsdiagram för en Bundle. Källa: Bartlett, Neil, OSGi In Practice, 2009

Tillståndsdiagrammet för en Bundle finns i Figur 4.2 ovan. Livscykeln för en Bundle startar från den svarta pricken med kommandot `install`. Bundlen hamnar då i tillståndet *Installed*, men ifall alla beroenden för en Bundle kan uppfyllas så övergår Bundlen direkt till tillståndet *Resolved*. Om Bundlen stannar i tillståndet *Installed* så måste det läggas till de Bundler eller Java-paket som fattas för att Bundlen skall kunna övergå till tillståndet *Resolved*. För att resolvera en Bundle kan det krävas att flera andra Bundler måste resolveras. Beroenden kan vara indirekta eller cykliska [7]. Från tillståndet *Installed* kan en Bundle också avinstalleras med kommandot `uninstall`. Då hamnar Bundlen i tillståndet *Uninstalled*. Man kan dock aldrig se att en Bundle är i tillståndet *Uinstalled*, eftersom en Bundle genast tas bort från ramverket när man utför kommandot `uninstall`. [6]

En installerad Bundle får kontroll genom att man instantierar en klass från Bundlen. Den här klassen måste implementera *BundleActivator*-gränssnittet där `start` och `stop` metoderna finns. [7]

Ifall man ger kommandot `start` så kommer tillståndet för en Bundle att byta till *Active*, men endast om Bundlen är i tillståndet *Resolved*. Då en Bundle är i tillståndet *Resolved* så är alla beroenden uppfyllda. Ifall man ger kommandot `start` till en Bundle som är i tillståndet *Installed* så försöker ramverket först resolvera Bundlen före ramverket försöker starta Bundlen. Före tillståndet *Active* så är Bundlen i tillståndet *Starting* under en kort tid då Bundlen startas. Det som sker i tillståndet *Active* beror på vad som kördes igång under tillståndet *Starting*. [1]

Då man get kommandot `stop` övergår Bundlen till tillståndet *Stopping* och då avslutas Bundlens körning. Under *Stopping*-tillståndet rensas också sånt bort som skapats under *Starting*-tillståndet. Efter tillståndet *Stopping* övergår Bundlen igen till tillståndet *Resolved*. [1]

Det är också möjligt att avinstallera en Bundle fast den är i tillståndet *Active*. Man kan inte se det här från tillståndsdiagrammet men då övergår Bundlen på samma sätt först till tillståndet *Stopping* och vidare till *Resolved* och till slut till tillståndet *Uninstalled*. [6]

När man avinstallerar en Bundle kan beroenden till en annan Bundle bli ouppfyllda. Det här skapar dock aldrig en situation där Bundlen direkt blir påverkad, eftersom paketen den avinstallerade Bundlen exporterade lämnar kvar så länge som det finns andra resolverade Bundler som har beroenden till dem. Det är dock möjligt att enheten *Management Agent* kan utföra kommandot `refresh` och då stoppas alla Bundler som påverkats. Då tas beroendena bort och alla Bundler som stoppats resolveras igen och startas om. En Bundle stoppas alltid före något av dess beroenden ändras. [7]

Det är också möjligt att spara information om vilka Bundler som har varit startade, ifall OSGi-ramverket av nån anledning stängs av. Bundler har också olika startnivåer, vilket medför att en Bundle kanske inte startas direkt igen efter en återställning. Ramverket kan byta startnivåer under körningen. [7]

4.4 Bundlefragment

Bundlefragment introducerades i version 4 av OSGi. Ett bundlefragment är en Bundle som inte är helt klar och kan därför inte göra någonting själv. Men det är möjligt att koppla ett bundlefragment till en annan Bundle som agerar som värd. Värdbundlen får inte vara ett bundlefragment utan måste vara färdigimplementerad. När bundlefragmentet är kopplat till värdbundlen kan fragmentet lägga till klasser och resurser till värdbundlen. Bundlefragmentets klasser läggs till värdbundlens classpath under körning. [1]

Bundlefragment används när man inte vill skapa fullt fungerande Bundler för att skapa modularitet. T.ex. när man endast vill tillföra en del resurser till en Bundle. Med hjälp av bundlefragment kan man spara diskutrymme och nerladdningstid, eftersom storleken på bundlefragment är mindre än hos fullt implementerade Bundler. [1]

5 OSGi-tjänstplattformen

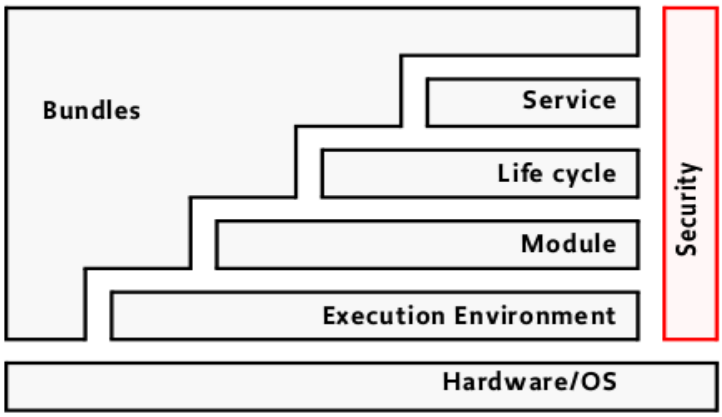
5.1 OSGi-tjänster

Då en OSGi-tjänst skapas så implementeras den som ett tjänstobjekt. Tjänstobjektet ägs och körs inom en Bundle, den här Bundlen måste registrera tjänsten i tjänstregistret. När tjänsten är registrerad finns den tillgänglig för alla andra Bundler under kontroll av ramverket. Då en tjänst registreras skapas ett gränssnitt som specificerar tjänstens publika metoder. Efter det kan andra Bundler förvärva tjänster genom namnet på gränssnitten och komma åt metoder genom gränssnittet. [2]

Tjänsterna har också en PID (Persistent Identifier). En sådan behövs för att det skall vara möjligt att identifiera tjänster efter att ramverket har startats om. En PID är unik för varje tjänst och ändras inte fast ramverket startas om. Det är viktigt att varje tjänst alltid använder samma PID. En PID ska inte ändras om Bundlen stoppas och startas om. [2]

För att andra Bundler skall ha möjlighet att hitta tjänster som de behöver så har tjänsterna även en del egenskaper som skapas när tjänsten registreras. Det finns ett filterspråk som Bundler kan använda för att hitta de tjänster som behövs. Om det inte finns en tjänst till förfogande när Bundlen aktiveras kan den lämna och vänta på att en tjänst med de rätta egenskaperna framträder. [9]

5.2 Arkitektur



Figur 5.1: OSGI-arkitekturen. Källa: OSGI-alliansen.

OSGi-ramverkets funktionalitet är uppdelad i olika lager. I Figur 5.1 kan man se de olika lagren i arkitekturen. Säkerhetslagret baserar sig på säkerheten i Java 2 men i det här lagret läggs det till en del som inte hör till standard Java. Säkerhetslagret gör det möjligt att sprida och hantera program som måste köras i en kontrollerad omgivning. Det här lagret är valfritt. [2]

Modullagret beskriver den modulariserade modellen för Java. Det här lagret bestämmer hur Bundler delar paket mellan varandra och hur de begränsar synligheten till interna paket. Det här lagret kan användas utan livscykellagret och tjänstlagret. [2]

Livscykellagret ger en livscykel-API åt Bundler. Den här APIn beskriver en modell för hur Bundler i modullagret hanteras under körning. Den beskriver hur Bundler startas och stoppas samt hur Bundler installeras, uppdateras och avinstalleras. Livscykellagret kräver modullagret men säkerhetslagret är valfritt. [2]

Tjänstlagret anger hur man skapar dynamiska lösningar och hjälper till med att skapa och sprida tjänster genom att man skapar ett gränssnitt till tjänsten så att man får låg koppling. Det här lagret gör det möjligt att använda tjänster endast genom att veta om deras gränssnitt. Tjänstlagret förser Bundler med en kommunikationsmodell. [2]

5.3 Säkerhet

Ett av målen med OSGi-ramverket är att det skall vara möjligt att köra program från många olika källor under strikt kontroll. Det är därför nödvändigt att ha en säkerhetsmodell. Säkerhetsmodellen består av Java 2 programkodssäkerhet, kontroll av vad Bundler har synligt och kontrollerad kommunikation mellan Bundler. Java 2s programkodssäkerhet gör att man måste ha olika tillstånd för att få använda olika resurser. Alla Bundler har ett antal krav på tillstånd som kan ändras under körning. [7]

Java kontrollerar att man inte kan komma åt programkod som skyddas genom att begränsa synlighet av metoder eller klasser eller programkod som endast är synligt inom samma paket. Med OSGi-ramverket får man också möjligheten att begränsa synligheten av paket så att de endast syns inom en Bundle. De klasserna är synliga för klasser inom den Bundlen men inte för klasser i andra Bundler. OSGi-specifikationen gör det också möjligt för Bundler att begränsa export och import av paket till pålitliga Bundler. Bundler måste också ha ett tillstånd för att ha möjlighet att registrera eller använda tjänster från tjänstregistret. [7]

5.4 Olika standardtjänster

Det finns tjänster i olika kategorier: ramverkstjänster, systemtjänster, protokolltjänster, blandade tjänster och hjälptjänster. De här tjänsterna är valfria och kan användas för att bygga ihop ett önskat system. [7]

Ramverkstjänsterna används för att styra ramverket. Exempel på sådana tjänster är t.ex. *Package Admin* och *Start Level*. *Package Admin* användas för att beräkna beroenden mellan Bundler och för att få information om vilka paket som är utdelade. *Start Level* anger när en Bundle skall startas. Om man använder den tjänsten kan man kontrollera så att vissa Bundler startas före andra om det krävs. [7]

Systemtjänster står för vanlig funktionalitet som ofta behövs i system. Exempel på sådana tjänster är t.ex. en *Log Service* och *Device Access*. *Log Service* används för att håller reda på information, varningar, debuginformation och information om fel som inträffar. Bundler som använder den här tjänsten får ovannämnda

information. *Device Access* används när man lägger till en ny enhet som behöver en drivrutin. Tjänsten hittar då en drivrutin och laddar ner en Bundle som implementerar den drivrutinen. [7]

Protokolltjänster har skapats av OSGi-alliansen för att koppla några utomstående protokoll till OSGi-tjänster. Exepel på sådana tjänster är t.ex. *Http Service* och *UPnP* . Http-tjänsten gör det möjligt för Bundler att tillhandahålla servlets. P.g.a. OSGi-ramverkets dynamiska natur är det här ett mycket attraktivt alternativ. *UPnP* (Universal Plug and Play) används för att registrera enheter från ett *UPnP*-nätverk till tjänstregistret. Den kan också registrera OSGi-tjänster till *UpnP*-nätverket. [7]

Till de blandade tjänsterna hör *Wire Admin* och *XML Parser*. *Wire Admin* kopplar ihop olika tjänster enligt en konfiguration. *XML Parser* gör det möjligt för en Bundle att hitta en parser med de rätta egenskaperna och kompatibilitet med JAXP (Java API for XML Processing). [7]

Hjälptjänsterna finns till för att hjälpa programmerarna, eftersom OSGi-ramverkets dynamiska egenskaper gör programmen komplexare. Till hjälptjänsterna hör *Service Tracker* och *Declarative Services*. *Service Tracker* håller reda på vilka tjänster en applikation använder. *Declarative Services* sköter om att en Bundle initialiseras endast när det finns ett behov för en tjänst från den Bundlen. [7]

6 Olika implementationer av OSGi

Det här kapitlet kommer att handla om några vanliga implementationer av OSGi-ramverket. Det kommer att vara korta beskrivningar om versionerna Equinox, Knopflerfish, Felix och Concierge.

6.1 Equinox

Equinox-implementationen av OSGi-ramverket är den mest använda av dessa fyra. Det beror mycket på att Eclipse använder sig av den. Equinox används också av Lotus Notes [10] och IBM WebSphere Application Server [11]. Equinox är licenserat under Eclipse Public License (EPL). [1]

Equinox är en implementation av *OSGi R4 core framework specification*. Equinox implementerar också en del valfria OSGi-tjänster och annan infrastruktur som behövs för att köra OSGi-baserade system. Målet med Equinox-projektet är att det ska vara ett första klassens OSGi-community och bana väg för Eclipse som ett landskap av Bundler. Equinox-projektet ansvarar också för att utveckla och leverera implementationen av OSGi-ramverket för Eclipse. [12]

6.2 Knopflerfish

Knopflerfish är en populär och mogen implementation av OSGi. Knopflerfish version 3 implementerar hela OSGi-ramverket enligt OSGi R4 v4.2. Knopflerfish är ett öppet källkodsprojekt som utvecklas och underhålls av Makewave AB som är ett svenskt företag. Makewave erbjuder också Knopflerfish Pro som är en kommersiell produkt. Med den får man support och ett antal extra komponenter som inte finns i den öppna Knopflerfishimplementationen. [13]

6.3 Felix

Felix är en implementation av OSGi R4 och är utvecklad av ett community. Felix implementerar också en del andra intressanta OSGi-relaterade teknologier. Felix är licenserat under Apache licens version 2.0. [14] Felix är designat för att vara så kompakt som möjligt och är också den minsta implementationen av OSGi R4 om man ser på JAR-storleken. [1]

6.4 Concierge

Concierge är en optimerad implementation av OSGI R3. Den lämpar sig bra till mobila system och inbyggda datorsystem. Vid designen av Concierge har man beaktat att plattformerna som Concierge kommer att köras på har VMs som är kompakta och inte optimerade. Concierge använder resurser sparsamt och prestandan i resursbegränsade miljöer är bra. Concierge är också mycket kompatibelt med andra implementationer av OSGi-ramverket. [15]

7 Slutsatser

Programvaruprojekt blir allt större i antal rader programkod och människor som är involverade i projektet. Det är helt klart att det behövs något sätt att kontrollera den enorma mängden information som ingår i ett stort projekt. Det måste finnas något sätt att dela upp arbetet på och på samma gång ha en fungerande lösning till problemen som uppstår när man skapar modulariserade lösningar. Det måste vara lätt att integrera alla gruppers enskilda moduler till det slutliga systemet. Då en grupp har skapat en modul ska en annan grupp ha möjligheten att enkelt använda den tjänsten som modulen exporterar. Därför behöver man en standardiserad tjänstmodell.

Många system behöver i dagens läge också möjligheten att på ett enkelt sätt installera och uppdatera programvaran utan att behöva starta om systemet. Det ska också vara enkelt att sprida ny programvara till en stor mängd apparater. Som exempel kan man ta smarta telefoner som nuförtiden kan köra en hel del olika program och då krävs det förstås mycket uppdateringar. Då kan man med OSGi-ramverket på ett enkelt sätt sprida dessa uppdateringar.

OSGi-ramverket erbjuder mycket bra lösningar till en stor del av problemen. OSGi-ramverkets Bundler är mycket dynamiska och gör det möjligt att skapa modulariserade dynamiska lösningar. Bundlerna skapar också ett nytt sätt att begränsa synligheten av klasser inom Bundlerna, vilket skapar pålitligare system. Bundlernas versionshantering gör också systemen pålitligare. Genom att ha en tjänstplattform kan man enkelt koppla ihop dessa Bundler till ett stort system. OSGi-ramverket skapar löst kopplade system med hjälp av Bundlerna och tjänsterna. Systemet kan också anpassa sig till omgivningen under körning. T.ex. kan en bil känna av en telefon med bluetooth och anpassa sig därefter.

Man måste dock tänka på att systemen inte blir modulariserade endast för att man börjar använda sig av OSGi-ramverket. Man måste fortsättningsvis tänka på att använda sig av sådan design som skapar modulariserade lösningar från grunden. Man måste t.ex. lägga klasser och gränssnitt i paket enligt vilken funktionalitet de har och även komma ihåg att skapa låg koppling genom att ha funktionaliteten bakom gränssnitt. Man kan också behöva använda sig av injektion av beroende med t.ex. Spring-ramverket för att byta ut implementationsklasser. Användningen av OSGi-ramverket gör det enklare att skapa modulariserade lösningar, eftersom det är enkelt att skapa väldefinierade moduler.

Källor

[1] Bartlett, Neil, OSGi In Practice, 2009
[2] OSGi Alliance, OSGi Service Platform Core Specification, 2009
[3] OSGi™ Alliance, OSGi Technology, Februari, 2011, http://www.osgi.org/About/Technology
[4] Oracle, The Java EE 5 Tutorial, , 2010, http://download.oracle.com/javaee/5/tutorial/doc/bnaby.html
[5] Bosschaert David; Diesler Thomas, JBossOSGi - User Guide, 2011
[6] Walls, Craig, Modular Java - Creating Flexible Applications with OSGi and Spring, 2009
[7] OSGi Alliance, About the OSGi Service Platform, 2007
[8] OSGi Alliance, Semantic Versioning, 2010
[9] OSGi Alliance, The OSGi Architecture, , 2011, http://www.osgi.org/About/WhatIsOSGi
[10] IBM, Lotus Notes, March, 2011, http://www-01.ibm.com/software/lotus/products/notes/
[11] IBM, WebSphere Application Server, March, 2011, http://www-01.ibm.com/software/webservers/appserv/was/
[12] The Eclipse Foundation, Mission Statement, March, 2011, http://www.eclipse.org/equinox/
[13] Makewave AB, Produkt: Knopflerfish Pro, March, 2011, http://www.makewave.com/site.sv/products/knopflerf
[14] The Apache Software Foundation, Apache Felix, March, 2011, http://felix.apache.org/site/index.html
[15] ETH Zurich, Overview, March, 2011, http://concierge.sourceforge.net/