

Relatos ocasionais sobre tecnologia

novembro 24, 2008

Uma visão geral sobre a plataforma OSGi

Posted by kievagama under [Java](#) | Tags: [Java](#), [OSGi](#) |

[Deixe um comentário](#)

Conversando há algumas semanas com um colega de equipe, que desenvolve um modelo de componentes para OSGi, ele me perguntou se no Brasil as pessoas desenvolviam aplicativos em cima do OSGi.

Não sei se me equivoquei, mas respondi: “acho que não pois o uso de OSGi é mais voltado para infraestrutura de software (middlewares e frameworks) e o mercado brasileiro é bem focado no desenvolvimento de Sistemas de Informação. Indiretamente usa-se software criado em cima do OSGi”.

Pesquisando no Google não deu pra achar muito material de OSGi em português. A maior parte composta de definições superficiais ou relatos de pessoas testando o OSGi. Isso me motivou a criar este blog com o intuito de esclarecer mais sobre esta tecnologia para outros lusófonos, sejam brazucas ou não, além de ter um espaço para compartilhar outros relatos tecnológicos.

A sigla

Nunca mais digam Open Services Gateway Initiative. Isso é uma “gafe” no mundo OSGi. Este termo é obsoleto segundo a OSGi Alliance, que controla a especificação. Agora é só OSGi, ou OSGi Service Platform. Mas muita gente também chama de OSGi framework.

Visão Geral

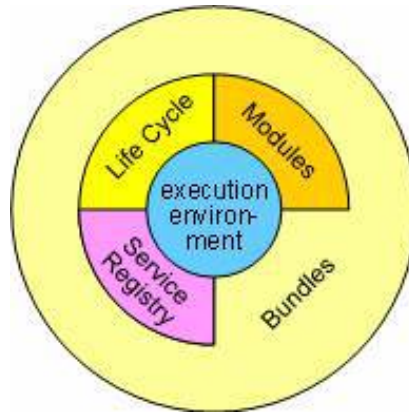
Trata-se de um conjunto de especificações que define um sistema de componentes dinâmicos e orientado à serviços, construído sobre a plataforma Java. Até então, Java não tinha bem definido o conceito de módulo (a [JSR-277](#) é meio que uma tentativa da Sun tomar de voltas as rédeas da modularização em Java, usando e estendendo conceitos do OSGi).

Um das grandes vantagens é que o OSGi fornece um ambiente de execução que resolve problemas de deployment e versionamento de módulos/componentes. Tentando sintetizar um pouco, os módulos no OSGi podem:

- exprimir suas dependências (o que ele precisa para ser executado, resumindo em Java: a lista dos pacotes que ele usa),
- serem instalados e atualizados dinamicamente
- coexistir com módulos de diferentes versões numa mesma VM

Um pouco de história

Quem iniciou a especificação OSGi foi a Sun Microsystems na JSR-8 em 1999. Logo, OSGi não é nada novo. A idéia era voltada para “gateways” em SmallOffices/HomeOffices (SOHO). Um exemplo comum de um gateway é um modem ADSL que nos conecta à Internet, mas pode integrar vários dispositivos wireless do nosso escritório ou casa. Serviços de diferentes fabricantes de equipamentos poderiam ser “deployados” no mesmo software de administração do gateway (i.e. na mesma JVM) de forma simples, isolada e segura. Aparentemente a Sun não tinha tanto interesse e doou a spec que deu origem à OSGi Alliance.



Visão do OSGi em camadas, conforme esquema da OSGi Alliance

Bundles e ciclo de vida

A plataforma OSGi fornece um ambiente para o deployment de módulos (ou componentes, se preferir), chamados de bundles na terminologia OSGi. Os bundles são arquivos jar comuns que contêm classes e arquivos de recursos. A diferença é que eles trazem em seu manifest vários atributos específicos à plataforma OSGi, que definem, entre outros metadados, as dependências do componente e os pacotes que ele fornece.

Estas dependências são um nível de visibilidade num grão maior que o OSGi introduz no Java: visibilidade de pacotes (O .NET fornece mais controle ainda no nível da linguagem com o modificador internal) fora do jar. Bundles fornecem explicitamente pacotes através do atributo Export-Package, e da mesma forma pode importar via Import-Package.

Bundles possuem um ciclo de vida (simplificando: install, uninstall, start, stop, update) dinâmico que permite a atualização de componentes em tempo de execução. Naturalmente funciona como um ambiente de plugins com hot-deploy. Se o ambiente de execução não satisfaz as dependências de um componente, ele fica apenas como Installed.

Um bundle pode ser ativado se ele fornecer uma implementação de `org.osgi.framework.Activator`, que define os métodos `start` e `stop`. Cada um desses métodos tem como parâmetro um objeto `BundleContext` que dá ao bundle acesso ao framework. Através do `BundleContext` o bundle (o módulo/componente/jar) pode, por exemplo, acessar o `Service Registry` do OSGi e registrar ou obter serviços.

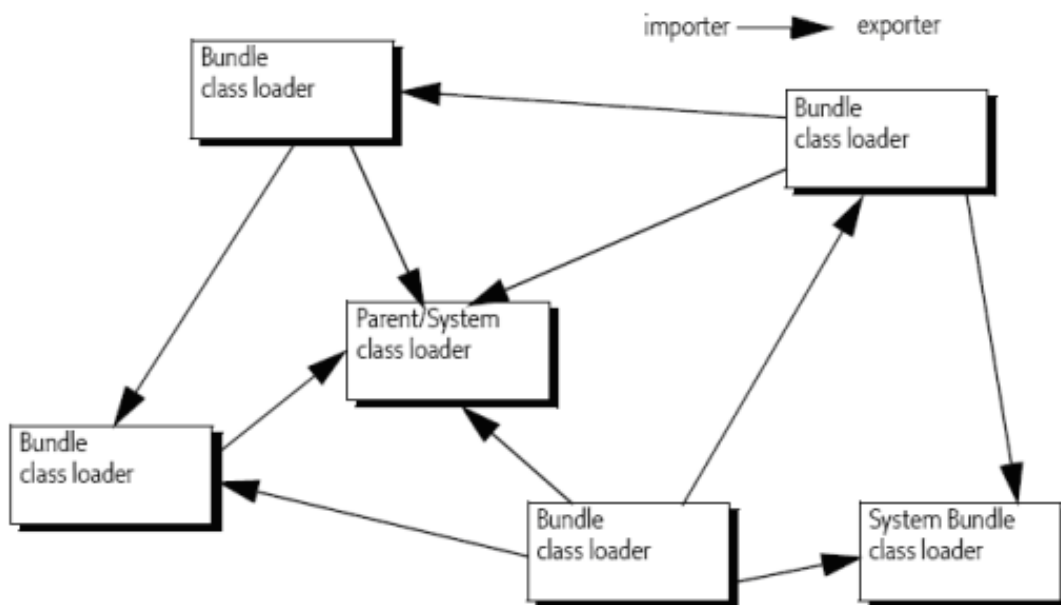
Fim do “Classpath hell”

Ao invés de usar uma linha de comando tipo: `“java -cp xyz.jar;lib2.jar;meuxmlparser.jar;minhalib.jar -jar minhaapp.jar”` as dependências são resolvidas em runtime. Você precisa apenas executar algo do gênero: `java -jar minhaplataformaosgi.jar`, e à

medida que for adicionando novos bundles/componentes as dependências de tipos são resolvidas baseado no que está declarado explicitamente no manifesto do bundle. Um bundle cujas dependências não estão disponíveis é instalado, mas não é ativado.

Classloading

Muitos dizem que OSGi é composto de Classloaders em esteróides, ou Classloaders++. Realmente um dos grandes trunfos é seu mecanismo de Classloaders. Cada bundle no OSGi recebe sua própria instância de class loader. O pseudo-isolamento que existe é consequência de um complexo mecanismo de class loading que se baseia nos atributos dos manifestos dos bundles para efetuar o import/export de classes. O mecanismo normal do carregamento de classes em Java é hierárquico, e as classes que não estão disponíveis em determinado classloader são solicitadas ao ancestral imediato, que repete o processo até encontrar a classe ou chegar no class loader raiz. O mecanismo do OSGi também pergunta aos classloaders “irmãos”, inclusive especificando a versão desejada. Esse mecanismo de vários classloaders e de visibilidade controlada entre “irmãos” permite a coexistência de classes de mesmo nome em bundles separados (que no fim das contas são classloaders diferentes), e ainda, versões diferentes.



Source: OSGi Spec. R 4.1

Esquema de Classloaders no OSGi

Approach Orientado a Serviços

Como dito anteriormente, os bundles podem, publicar ou consumir serviços. Encontramos no OSGi a tradicional tríade de **consumidor, fornecedor e catálogo de serviços**, que seriam um bundle consumidor de serviços, um bundle que fornece o serviço e o ServiceRegistry, respectivamente. Serviços são objetos Java, sem nada especial. Você pode publicar um serviço `java.lang.Long`, um `foo.bar.MeuInterfaceServico` e por aí vai. Cabe ao conjunto de bundles presentes (i.e. a aplicação) usar os serviços publicados para executar algum objetivo.

Um breve exemplo de publicação de serviços:


```
package foo;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import java.util.Hashtable;

public class FooBundle implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        Hashtable props = new Hashtable();
        props.put("cor", "azul");
        context.registerService("foo.MeuServico", new foo.impl.MeuServi
    }

}
```



e um breve exemplo de um outro bundle que consome o serviço acima:

```
package bar;


import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import foo.MeuServico;

public class BarBundle implements BundleActivator {

    public void start(BundleContext context) throws Exception {

        //utilizando filtro para trazer o serviço com os atributos de:
        ServiceReference[] sr = context.getServiceReferences("foo.MeuSe
        MeuServico foo = null;
        if (sr != null) {
            foo = (MeuServico)context.getService(sr[0]);
        }
        /*sem utilizar filtro. Caso haja mais de um serviço o framewor
        retornará o serviço mais requisitado, se utilizar o mecanis
        ServiceReference ref = context.getServiceReference("foo.MeuSei
        if (ref != null) {
            foo = (MeuServico)context.getService(ref);
        }
    }

}
```



Cada activator acima pertence a bundles distintos. Para que o código do exemplo funcione, o tipo `foo.MeuServico` deve estar visível para o segundo bundle:

1. Algum bundle precisa exportar o pacote foo (Export-Package: foo). No caso acima o próprio bundle que contém o activator FooBundle o exporta.
2. O segundo bundle (o que define o activator BarBundle) precisa da diretiva Import-Package: foo descrita em seu manifesto.

Eventos

Sendo uma plataforma dinâmica, todas as partes envolvidas precisam saber o que acontece: bundles que chegam, que partem, serviços instalados, desinstalados, etc. O framework OSGi notifica estes eventos aos diversos tipos de listeners (principalmente BundleListener e ServiceListener) que foram registrados no framework pelos bundles. Um ponto crucial no desenvolvimento de aplicações dinâmicas utilizando OSGi.

O mecanismo tradicional de binding de serviços é bem passível de falhas, e um ponto delicado para quem está aprendendo e não lida corretamente com os eventos que notificam a desinstalação/parada de bundles e serviços. Alguns mecanismos como Declarative Services e iPOJO, foram desenvolvidos para, entre outras coisas, gerar estas dependências minimizando a quantidade de código e de erros.

Onde encontrar

Como trata-se de uma especificação, existem diferentes implementações. As mais conhecidas do mercado são gratuitas e open-source:

- [Eclipse Equinox](#)
- [Apache Felix](#) (Antigo [OSCAR](#) da ObjectWeb)
- [Makewave Knopflerfish](#) (Anteriormente GATESpace Telematics Knopflerfish)

Existem variações das implementações acima, como:

- [Concierge](#), baseado no OSCAR e que tem como alvo sistemas com menos recursos
- [ProSyst](#), empresa que fornece implementações “turbinadas” do Equinox

Teoricamente, os bundles que você desenvolver devem funcionar em qualquer uma das implementações.

Tendência Industrial

O Eclipse usa OSGi desde a versão 3.0 com sua própria implementação (Equinox). Todo plugin eclipse é um bundle. Entretanto, o Eclipse continuou usando seu mecanismo de Extension Points e quase não usa a camada de serviços do OSGi.

O JonAS foi o primeiro servidor JEE rodando sobre o OSGi. O Spring framework também investiu pesado nesta tecnologia. Pra não ficar repetindo as listas que têm por aí, pode-se consultar a lista do [wikipedia](#) e a da [OSGi Alliance](#).

Vantagens

As vantagens que me vêm à cabeça:

- Hot deploy de verdade, simples e direto
- Versionamento de componentes

- Isolamento de código
- Approach orientado a serviços
- Dinamismo

Desvantagens

- Curva de aprendizado
- Modelo bastante passível de falhas de programação
- Especificação fechada e controlada por empresas, diferente do JCP

Eu devo usá-lo também?

O OSGi fornece uma estrutura mais utilizada em middlewares e frameworks. Existe uma curva de aprendizado e “pitfalls” que não valem a pena e nem fazem sentido se você está desenvolvendo aplicações que não precisam das vantagens do OSGi.

Tutoriais

Por enquanto, fico só na tentativa do esclarecimento sobre a tecnologia. Há um tutorial em português [neste link](#) que serve pra começar. Em breve (o papo furado de um bocado de blogs...) tentarei postar algo fora do padrão “alô mundo”, mas sem fugir do nível básico.

Espero que o breve texto tenha esclarecido algo.

About these ads

[Blog no WordPress.com.](#) — [O tema Connections.](#)

© Seguir

Seguir “Relatos ocasionais sobre tecnologia”

Crie um site com WordPress.com