Marcelo Mayworm
(mayworm@ieee.org): Trabalha com desenvolvimento de software, atuando na área de arquitetura de software e soluções corporativas, trabalhando desde 2008 com OSGi em aplicações SOA. Mestre pela Universidade Federal do Rio de Janeiro e committer no projeto Eclipse Communication Framework, atua como arquiteto de Software Sênior nas Nações Unidas na Alemanha.

OSGi Distribuída: Uma Visão Geral

Entenda como funciona esta extensão do modelo OSGi e o que você pode esperar dela.

Na edição 34 da revista MundoJ, foi apresentado o artigo "OSGi e os Benefícios de uma Arquitetura Modular", no qual foi abordada uma visão geral sobre framework OSGi, mostrando seus benefícios no desenvolvimento de aplicações modulares. Neste artigo, abordaremos a parte do framework OSGi conhecida como "OSGi distribuída", a qual teve sua origem a partir da RFC (Request for Comments) 119, sendo parte da especificação 4.2 early draft do framework OSGi. Esta RFC 119, atualmente incorporada na versão final da especificação 4.2, foi um empenho para a padronização de alguns aspectos como criação, publicação, localização e acesso aos serviços OSGi remotos, os quais são a base para OSGi distribuída. Um dos pontos interessantes da OSGi distribuída é que ela não vem como uma alternativa para substituir outras tecnologias, tais como EJB, RMI ou JMS, mas sim como uma extensão ao modelo de programação proposto pelo framework OSGi (ver tópico "Entendendo ciclo de vida do bundle" no artigo "OSGi e os Benefícios de uma Arquitetura Modular"). Além disso, permite definir uma forma de acessar um serviço ou carregar um bundle (como é conhecida uma unidade modular na OSGi) instalados em outras JVMs potencialmente sendo executados em um computador remoto.

O framework OSGi

Uma das razões do sucesso do framework OSGi é o desacoplamento que sua utilização proporciona entre os bundles, expondo somente as interfaces para os módulos "consumidores" (opcional, porém, considerado uma boa prática), o que proporciona aos desenvolvedores um interessante recurso para a definição e implementação de APIs, especialmente evitando que as aplicações clientes ao utilizarem essas APIs acessem diretamente as implementações concretas, proporcionando assim um baixo acoplamento. Este controle pode ser feito em dois momentos, o primeiro (o qual não é obrigatório), durante o tempo de desenvolvimento, é controlado através de tooling, ou seja, o ambiente de desenvolvimento analisa se existe o acesso a uma classe que não esteja declarada para ser acessada externamente por outro bundle. Já o segundo momento destaca-se por ocorrer no container OSGi (ambiente de execução do framework OSGi), o qual é responsável por controlar o ciclo de vida de cada bundle durante o tempo de execução, criando e gerenciando classloaders para cada um dos bundles separadamente. A independência de classloader é considerada por algumas pessoas o ponto mais forte do framework OSGi, pois isto ajuda a reduzir muito dos problemas já conhecidos e vivenciados pelos desenvolvedores que trabalham com a tecnologia Java. A declaração das interfaces a serem expostas ou "consumidas" a partir dos bundles é feita através de um arquivo chamado descritor do bundle (ver tópico "Controle Operacional" no artigo "OSGi e os Benefícios de uma Arquitetura Modular").

Para Saber Mais

OSGi e os Benefícios de uma Arquitetura Modular, MundoJ Edição 34.

O mundo enterprise

Com um modelo de programação atrativo como o proposto pelo framework OSGi, não poderia ser diferente o interesse despertado no mundo Enterprise, fazendo com que a definição de serviços distribuídos não demorasse a fazer parte deste framework. Esta proximidade com o mundo Enterprise também trouxe soluções como o framework Spring para o contexto OSGi, permitindo às aplicações desenvolvidas com Spring terem uma separação mais modular, além da possibilidade de dinamicamente adicionar, remover e atualizar os bundles durante o tempo de execução, o que se torna possível através da separação de classloader aplicada a cada bundle,

Neste artigo são apresentados alguns conceitos sobre OSGi distribuída, uma interessante extensão do framework OSGi. Veremos como são descritos e distribuídos os serviços OSGi, observando ao que este framework se propõe em termos de computação distribuída, além de entender o background de seu funcionamento.

ou seja, cada bundle tem seu próprio classloader. Um interessante acontecimento, que não podemos deixar de comentar, foi a aproximação da SpringSource e a Fundação Eclipse, quando aquela doou o SpringSource dm Server para a Fundação Eclipse, dando origem ao projeto Virgo, um projeto open source liderado pela SpringSource com o objetivo de fornecer uma plataforma de execução para o desenvolvimento de aplicações servidoras com OSGi.

Entendendo OSGi distribuída

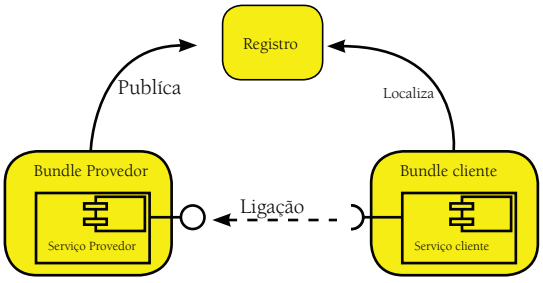

Figura 1. O bundle provedor publica o serviço no registro onde o consumidor pode descobrir quais serviços estão disponíveis para uso.

Como mostrado no artigo "OSGi e os Benefícios de uma Arquitetura Modular" o framework OSGi permite que as dependências de um bundle sejam especificadas de forma declarativa no arquivo descritor. O bundle, por sua vez, define o que importar, mas não tipicamente de onde importar. Sendo assim, é possível obter um baixo acoplamento entre os bundles através do modelo de serviços do framework OSGi. Desta forma, um bundle que deseja prover um serviço pode publicar a implementação da especificação de um serviço (exemplo: interface Java que define o contrato do serviço, ou seja, métodos públicos a serem expostos para acesso) no registro de serviços do framework (Service Registry), um mecanismo que permite aos bundles detectarem a existência de serviços, ou mesmo serem notificados quando um serviço não está mais presente. Para os bundles consumidores, este mecanismo de registro funciona como uma forma de auxiliar a descoberta de serviços, e a partir disto, ocorre a ligação entre o provedor e o consumidor do serviço.Com esta abordagem, fica claro que os bundles clientes somente conhecem o tipo da interface do serviço que está sendo consumido, e não conhecem sobre a implementação do serviço. É importante notar que o framework OSGi gerencia a consistência dos pacotes exportados, ou seja, verifica, por exemplo, se o conjunto de dependências estão consistentes em relação as versões requeridas, caso contrário, o container OSGi notificará os bundles clientes através do lançamento da exceção ClassNotFoundException que a requerida classe não está presente.

Como foi originalmente desenvolvido para rodar em uma única JVM, começamos a notar que algumas características do framework OSGi passam a não estar disponíveis em contextos de computação distribuída, especialmente em um mundo Enterprise onde podemos visualizar facilmente o cenário de diversos servidores, cada um rodando um conjunto de serviços, e estes, por sua vez, trocando informações via diferentes protocolos de comunicação para atender aplicações corporativas. Desta forma, novos "requisitos" aparecem, como, por exemplo, a localização e utilização de um serviço registrado em um diferente computador ou mesmo JVM. Neste cenário, a OSGi distribuída aparece como uma solução com intenção de permitir que um conjunto mínimo de funcionalidades existentes no framework OSGi possam ser mantidas, como, por exemplo, a localização e utilização de serviços, porém não mais somente serviços locais como também remotos. Uma das preocupações desta solução é garantir que desen-

volvedores acostumados a construir aplicações utilizando o framework OSGi, possam implementar ou utilizar serviços remotos de uma forma natural e direta como os serviços locais, sem a necessidade de aprendizado de novas APIs. Do ponto de vista do bundle, a transparência deve ser mantida, sendo indiferente rodar em um único container OSGi ou rodar em diversos containers distribuídos.

De uma forma geral, a OSGi distribuída aplica as seguintes extensões ao framework OSGi para permitir acesso remoto aos serviços:

- fazer com que bundles localizados em diferentes JVMs ou computadores possam publicar e consumir serviços através de variados protocolos de comunicação, tais como: HTTP, SOAP, XMPP, JMS, entre outros;
- permitir que um serviço OSGi possa ser publicado como um Web Service, e assim ser consumido por um cliente não-OSGi através de um "endereço" específico.

A OSGi distribuída não define nenhum novo protocolo de comunicação, formato de dados ou mesmo política de acesso, simplesmente cria uma extensão ao framework OSGi e metadados que definem como acessar e carregar módulos a partir de protocolos de comunicação já existentes. A figura 2 ilustra um cenário simples do papel da OSGi distribuída.

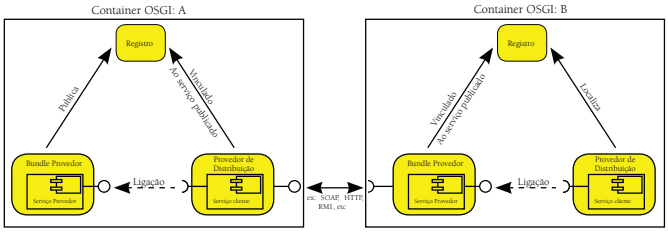

Figura 2. Um serviço OSGi cliente consumindo um serviço OSGi remoto publicado através de um provedor.

A figura 2 mostra um bundle instalado rodando em um container OSGi (container A) que publica um serviço que é consumido em outro container OSGi (neste caso, o container B). É importante ressaltar que ambos os serviços cliente e servidor não necessariamente devem ser codificados para serem distribuídos, por exemplo, implementando interfaces especificas, eles serão codificados da mesma forma como os serviços locais. Entretanto, em algumas situações, existe a possibilidade de codificar os serviços cliente e/ou servidor utilizando recursos da API da OSGi distribuída, permitindo que façam um melhor proveito do ambiente distribuído, como, por exemplo, acessar informações existentes no protocolo de comunicação utilizado. Estes recursos geralmente estão presentes no componente "Provedor de Distribuição" (Distribution Provider), que também é o responsável por publicar através de um endpoint (endereçamento único) o serviço a ser consumido remotamente. No lado cliente (neste caso, o container OSGi B), o componente "Provedor de Distribuição" fornece o acesso ao serviço publicado com um endpoint como uma referência local (criando um proxy) capaz de acessar o serviço remoto. Quando o cliente aciona o serviço remoto o "Provedor de Distribuição", com base no protocolo de comunicação utilizado, sabe como executar os métodos presentes no serviço remoto. Existem implementações da OSGi distribuída, como, por exemplo o Eclipse Communication Framework, que oferecem recursos de chamadas síncronas e/ou assíncronas para os serviços remotos.

Serviço de descoberta

Em situações do mundo real existe a possibilidade de diferentes containers OSGi rodarem simultaneamente, onde cada um pode publicar e/ou consumir diversos serviços remotos. Nesse tipo de ambiente, podemos facilmente nos deparar com o desconhecimento sobre a localização de determinados serviços remotos. Como vimos anteriormente, o componente "Provedor de Distribuição" sabe como fazer um serviço OSGi ficar disponível na rede e ser acessado remotamente, porém não é responsabilidade deste saber como descobrir os serviços existentes em outros containers OSGi espalhados na rede. Com isso, um componente especial na OSGi distribuída vem à tona, o "Serviço de Descoberta" (Discovery Service). Este componente tem como foco localizar serviços existentes na rede utilizando protocolos específicos de localização. O "Provedor de Distribuição" e o "Serviço de Descoberta" atuam como parceiros, a interação entre eles acontece registrando e descobrindo serviços que estejam disponíveis na rede. Apesar de prover uma tarefa importante, o componente "Serviço de Descoberta" é opcional na OSGi distribuída, pois uma vez que se conheça o endpoint, ou seja, o endereço do serviço que se queira acessar, o "Provedor de Distribuição" sabe como obter uma referência para acesso. Entretanto, existem momentos onde queremos obter um serviço, ou lista de serviços, que atendam a um critério específico de busca, utilizando para isso os metadados. Uma interessante característica do "Serviço de Descoberta" é o mecanismo assíncrono de notificação, o qual informa aos bundles clientes interessados, através de callback sobre a existência de um serviço remoto específico, tão logo esteja disponível na rede.

O "Serviço de Descoberta" pode prover vários protocolos de localização, através de diversas implementações, dentre os quais, os já conhecidos Apache ZooKeeper, UDDI (Universal Description Discovery and Integration), SLP (Service Location Protocol) e Bonjour/Zeroconf, ou mesmo protocolos proprietários. Provavelmente, na maioria dos casos de aplicações corporativas, a utilização se dará por meio dos serviços já conhecidos, entretanto existem situações em aplicações distribuídas, por exemplo, aplicações de computação pervasiva, onde dispositivos clientes podem não conhecer a real localização dos serviços a serem consumidos, somente os metadados que representam os serviços, ou então serviços clientes querem poder estar aptos a lidar com a situação de localizar uma implementação alternativa de um especifico serviço no caso de um problema repentino de disponibilidade. Nestes casos, a existência do “Serviço de Descoberta” torna-se importante. A figura 3 mostra o modelo da OSGi distribuída estendido com a adição do "Serviço de Descoberta", podemos ver que uma vez que o serviço existente no bundle provedor é publicado, ele torna-se remoto e disponível para ser localizado através da interação do componente "Provedor de Distribuição", com o "Serviço de Descoberta". No momento da notificação ao componente "Serviço de Descoberta", informações adicionais como metadados do serviço remoto podem ser registrados, permitindo assim ao bundle cliente, através

do "Serviço de Descoberta", localizar ou receber notificação da existência de um ou mais serviços remotos. É interessante notarmos como um mesmo serviço remoto pode estar disponível para ser localizado através de diferentes protocolos de localização, pois pode existir mais de uma implementação do componente "Serviços de Descoberta".

Serviços declarativos

Um dos mecanismos existentes no OSGi e já incorporado na maioria das implementações de OSGi existentes, é o mecanismo de declaração de serviços. Introduzido na especificação 4 do framework OSGi, ele traz melhorias para o quesito modularidade e flexibilidade na "arena" OSGi. Pois ao invés de trabalharmos com o modelo de serviços da OSGi de forma "programática", ou seja, registrando os serviços através de API, isto pode ser feito de forma declarativa através de arquivos XML armazenados dentro do bundle e processados em tempo de execução pelo OSGi's Service Component Runtime (SCR). A utilização dos serviços declarativos fornece vantagens, tais como: consumo de memória otimizado e melhor iniciação da aplicação (pois utiliza o abordagem de iniciação lazy, não carregando os bundles antes do momento que realmente seja necessário para utilização).

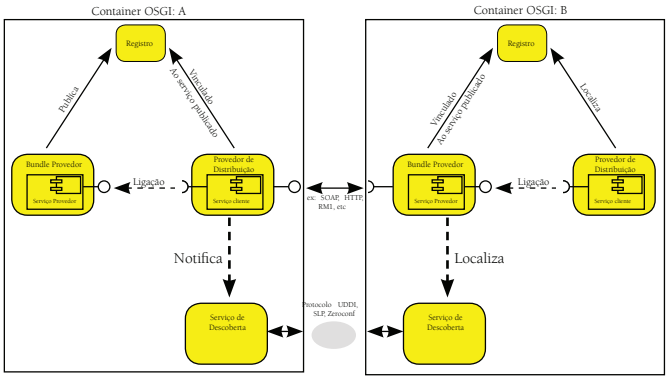

Figura 3. Modelo estendido de serviços da OSGi distribuída com a adição do "Serviço de Descoberta".

Publicando e consumindo um serviço remoto

O processo de publicar e/ou consumir um serviço remoto requer que alguns passos sejam seguidos para atender à especificação da OSGi. Mesmo sendo o componente "Serviço de Descoberta" opcional, é interessante ilustrar a sequência de passos que constituem estas operações. A seguir, veremos a figura 4 mostrando como um serviço "X" é registrado no framework OSGi e criado a partir disto, um acesso remoto com a possibilidade de ser descoberto em uma rede. Um ponto interessante desta figura é o momento em que se obtêm informações adicionais ao serviço, ou seja, os metadados ou as informações sobre o "contrato" do serviço (interface Java usada para definir o serviço OSGi), permitindo ao "Provedor de Distribuição" realizar a ligação entre a interface Java utilizada para publicar o serviço e a implementação fornecida pelo bundle provedor. A partir disto, o "Serviço de Descoberta" poderá utilizar diferentes protocolos com base nos metadados, para publicar o serviço, tornando-o disponível à localização, por exemplo, no caso de um serviço com informações específicas para UDDI, este será o protocolo de localização utilizado. Os metadados do serviço são obtidos a partir do bundle provedor e as propriedades do serviço que são parte da "inscrição" do serviço no registro.O consumo de um serviço no lado cliente é mostrado na figura 5. Um serviço remoto OSGi pode ser consumido por um cliente não-OSGi, como no caso do serviço ter sido publicado através de um Web Service (com o respectivo documento WSDL, por exemplo). Algumas implementações da OSGi distribuída fornecem um mecanismo para publicar um serviço remoto OSGi através de Web Service, possibilitando a integração de serviços OSGi com diversas outras plataformas. Porém, neste caso, assumindo que o cliente é um bundle rodando em um container OSGi diferente do provedor do serviço, veremos como esta operação acontece (figura 5).

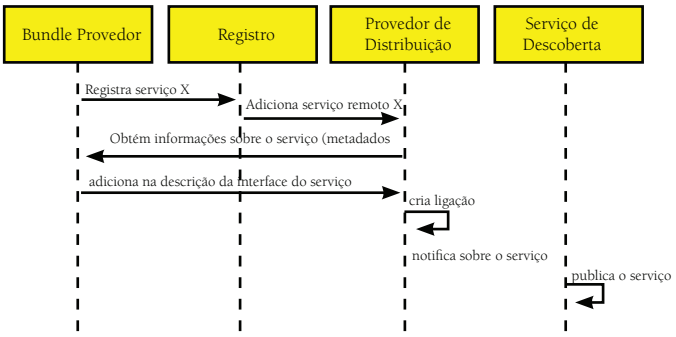

Figura 4. Expondo um serviço remoto.

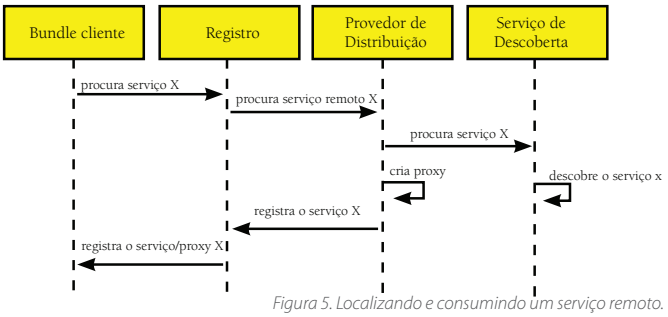

Figura 5. Localizando e consumindo um serviço remoto.

Na figura 5, o "Provedor de Distribuição" está usando o componente "Serviço de Descoberta" para localizar na rede uma implementação do serviço "X", a qual satisfaça os critérios de busca especificados pelo cliente no momento de pesquisa. Uma vez localizado um serviço, o "Provedor de Distribuição" gera um proxy a partir do código da interface do serviço desejado em tempo de execução com base no protocolo de comunicação utilizado entre o cliente e provedor, este proxy é então adicionado ao registro e retornado ao cliente através de um callback. O proxy age exatamente como um serviço local no lado do cliente, e caso exista algum problema de comunicação com o serviço remoto, uma exceção de tempo de execução é lançada notificando o cliente. Uma vez que o serviço remoto referenciado pelo proxy não esteja mais disponível por qualquer razão, o "Serviço de Descoberta" deverá notificar o cliente através da desativação do proxy, tornando este não mais acessível como um serviço local registrado no cliente. Para a comunicação do bundle cliente com o serviço remoto é necessário que o cliente tenha acesso a interface Java que define o serviço. Desta forma, uma boa prática é ter um bundle separado que contém

a interface do serviço e outro (localizado remotamente) com a implementação do serviço, mantendo, desta forma, o fraco acoplamento entre os "módulos". Com isto, vimos que este bundle que contém somente a interface do serviço estará presente no lado cliente e no lado do provedor do serviço. Este bundle que contém somente a interface do serviço será declarado como uma dependência nos bundles cliente e provedor

Implementações existentes

Atualmente existem algumas implementações de OSGi distribuída, open source ou comerciais. Entre as principais estão o Apache CXF e o Eclipse Communication Framework.

O Apache CXF é um projeto da Fundação Apache, o qual implementa OSGi distribuída e também fornece API para suporte a Web Service. Além disso, implementa as especificações JAX-WS e JAX-RS, permitindo a utilização dos serviços OSGi remotos através dos protocolos http e JMS. Outro ponto forte deste framework é a facilidade de publicar os serviços OSGi como Web Services. O Eclipse Communication Framework está sob o "guarda-chuva" da Fundação Eclipse. Este framework implementa um mecanismo genérico de "provedor", fornecendo assim uma API que permite a implementação de diferentes suportes para os diversos tipos de protocolos de comunicação. Por este motivo, protocolos como HTTP/HTTPS, SOAP, JMS, XMPP, JGroups, IRC, Skype e outros, podem ser usados para suportar os serviços remotos através da implementação de OSGi distribuída deste framework. Este framework também fornece mecanismo de balanceamento de carga através do mecanismo de OSGi distribuída.

Criando um serviço remoto

Como vimos, existem especificações as quais definem o framework OSGi e também seus serviços remotos. Desta forma, para criar aplicações modulares usando OSGi e implementar os serviços OSGi remotos, é necessário usar um framework que implemente o core da especificação OSGi e também uma implementação da OSGi distribuída. Existem diferentes implementações de OSGi disponíveis, tais como Equinox, Knopilerfish, Felix, entre outras (mais detalhes ver tópico "Implementações" no artigo "OSGi e os Benefícios de uma Arquitetura Modular"). Para o exemplo deste artigo será utilizado o Equinox como a implementação do framework OSGi e o ECF (Eclipse Communication Framework) para fornecer os serviços OSGi remotos.

A seguir, será apresentado um exemplo simples, publicando e consumindo um serviço remoto. Este serviço provê a funcionalidade de um dicionário a partir de uma palavra enviada como parâmetro, em que um conjunto de textos contendo possíveis significados da palavra serão retornados. Existem várias formas de implementar e/ou organizar os serviços OSGi em bundles, para efeito deste exemplo criamos três bundles conforme apresentados na figura 6. O bundle servicosremotos.dicionario contém o serviço OSGi, ou seja, a interface Java IDicionario e a classe Java DicionarioImpl que implementa o serviço. Neste cenário, o bundle servicoremotos.dicionario.cliente consumirá o serviço implementado pela classe DicionarioImpl e publicado através da interface IDicionario pelo bundle servicosremotos.dicionario.provedor.

Figura 6. Bundles e a relação de dependência entre eles.

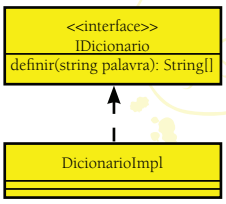

Figura 7. Modelo gráfico mostrando o serviço Dicionário.

Listagem 1. Interface IDicionario. Esta interface define o serviço publicado.

```
package servicosremotos.dicionario;

public interface IDicionario {

   /**
    * Metodo publicado para os bundles cliente
    * @param palavra
    * @return
    */
   public String[] definir(String palavra);
}
```

O bundle servicosremotos.dicionario contém o serviço OSGi, ou seja, a interface Java IDicionario e a classe Java DicionarioImpl que implementa o serviço. Neste cenário o bundle servicoremotos.dicionario.cliente consumirá o serviço implementado pela classe DicionarioImpl e publicado através da interface IDicionario pelo bundle servicosremotos.dicionario.provedor.

A interface IDicionario e a classe DicionarioImpl são exportadas pelo bundle serviçoremotos.dicionario. Enquanto o bundle servicosremotos.dicionario.cliente somente importa a interface IDicionario, o bundle servicosremotos.dicionario.provedor importa tanto a interface IDicionario quanto a implementação DicionarioImpl, pois o provedor é o responsável por publicar a definição do serviço (interface Java) e a implementação propriamente dita. Neste caso, a classe DicionarioImpl. O bundle servicosremotos.dicionario basicamente conterá só a interface IDicionario e a classe DicionarioImpl, como mostradas nas Listagens 1 e 2.

Ainda no bundle servicosremotos.dicionario, podemos ver o arquivo descritor do bundle (MANIFEST.MF) na Listagem 3 exportando os pacotes que contêm a interface e a classe Java do serviço dicionário.

A publicação do serviço, de forma que este seja acessado remotamente pelo bundle cliente, será de responsabilidade do bundle servicosremotos.dicionario.provedor. Este bundle simplesmente irá registrar o serviço definido pela interface IDicionario através do mecanismo da OSGi distribuída, tornando o simples POJO DicionarioImpl ser acessado remotamente por outros bundles espalhados em diferentes containers OSGi. Na Listagem 4 é apresentado o arquivo descritor

deste bundle, o qual mostra a dependência existente entre o bundle servicosremotos.dicionario e o bundle servicosremotos.dicionario.provedor através dos pacotes importados.

Por "trás das cenas", o ECF cuida de todo o mecanismo de publicação e gerenciamento do serviço publicado para ser acessado remotamente. Na Listagem 5 veremos o trecho de código o qual publica o serviço.

O bundle cliente servicosremotos.dicionario.cliente irá acessar o serviço remoto especificado através da interface IDicionario. Este bundle somente "conhece" esta interface, evitando ter acesso à implementação "concreta" do serviço. Na Listagem 6 é apresentado o arquivo descritor deste bundle cliente, onde somente existe a dependência declarada para o pacote que contém a interface IDicionario. Existem diferentes formas de acessar os serviços OSGi remotos. Neste exemplo está sendo abordado o mecanismo utilizando no lado cliente o ServiceTrackerCustomizer. Este mecanismo permite o bundle cliente ser notificado quando um serviço remoto é registrado pelo bundle provedor, também sendo acionado quando o serviço é alterado ou removido. Na Listagem 7 é apresentada a classe no bundle cliente que recebe esta notificação.

Neste exemplo, vimos os "trechos" de códigos necessários para publicar e consumir os serviços OSGi remotos, além dos arquivos descritores dos bundles. O código-fonte completo, contendo as implementações dos três bundles e todas as dependências necessárias para serem executados na plataforma Eclipse usando o Equinox e ECF, pode ser baixado no site da revista.

Listagem 2. Classe DicionarioImpl. Esta classe contém a implementação do serviço publicado.

```
package servicosremotos.dicionario.impl;

import java.util.HashMap;
import java.util.Map;

import servicosremotos.dicionario.IDicionario;

public class DicionarioImpl implements IDicionario {

    /*
     * uma simples forma de armazenar algumas definicoes
     */
  Map<String, String[]> definicoes = new HashMap<String, String[]>();

  /**
   * Instancia algumas definicoes
   */
  public DicionarioImpl() {
    // a nível de exemplo, somente uma única palavra na lista de definições
    String defsVotar[] = new String[] { "Dar o voto em favor de.",
    "Fazer voto de, prometer solenemente.",
    "Decretar, outorgar, conferir, deferir.",
    "Consagrar, dedicar, destinar." };

    definicoes.put("votar", defsVotar);
  }

  public String[] definir(String palavra) {
    return definicoes.get(palavra.toLowerCase());
  }

}
```

Listagem 3. Arquivo descritor do bundle.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Dicionario
Bundle-SymbolicName: servicosremotos.dicionario
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: servicosremotos.dicionario.Activator
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Import-Package: org.osgi.framework;version="1.3.0"
Export-Package: servicosremotos.dicionario, servicosremotos.dicionario.impl
```

Listagem 4. Arquivo descritor do bundle servicosremotos.dicionario.provedor.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Provedor
Bundle-SymbolicName: servicosremotos.dicionario.provedor;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: servicosremotos.dicionario.provedor.Activator
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Import-Package: org.eclipse.ecf.core;version="3.0.0",
 org.eclipse.ecf.core.identity;version="3.0.0",
 org.eclipse.ecf.osgi.services.distribution;version="1.0.0",
 org.eclipse.equinox.app;version="1.0.0",
 org.osgi.framework,
 org.osgi.util.tracker;version="1.4.2",
 servicosremotos.dicionario,
 servicosremotos.dicionario.impl
Require-Bundle: org.eclipse.equinox.common
Export-Package: servicosremotos.dicionario.provedor
```

Listagem 5. Trecho de código para publicação do serviço OSGi remoto.

```
bundleContext = Activator.getContext();

// Processa Argumentos
processarArgs(appContext);

// Configura as propriedades para o serviço remoto
Properties props = new Properties();
// adiciona a OSGi propriedade indicando que todas as interfaces
// passadas para o registerService devem ser possíveis de acessar
// remotamente (wildcard)
props.put(IDistributionConstants.SERVICE_EXPORTED_INTERFACES,
      IDistributionConstants.SERVICE_EXPORTED_INTERFACES_WILDCARD);
//...
// registra o serviço remoto
ServiceRegistration dicionarioRegistration = bundleContext.
registerService(IDicionario.class.getName(), new DicionarioImpl(), props);
```

Cloud Computing, OSGi e OSGi distribuída

Cloud Computing é um assunto que vem a cada dia atraindo mais desenvolvedores. Muito tem sido feito e discutido ao redor deste assunto, especialmente no que diz respeito a como se desenvolver aplicações para cloud. Cloud computing e OSGi são conceitos diferentes, entre-

tanto estão "próximos" um do outro. OSGi é um framework que fornece uma plataforma para o desenvolvimento de aplicações modulares e orientadas a serviços, enquanto cloud computing pode ser visto através de diferentes níveis que vão desde o processamento de informações a data centers remotos. Os serviços de cloud computing existentes passam por ambientes de servidores virtuais como o Amazon EC2 a plataformas de mash-up para serviços externos como o Yahoo Pipes. Geralmente o desenvolvimento de aplicação para cloud computing ainda "sofre" por não ser simples para projetar, desenvolver, testar, realizar o deployment e gerenciar, onde para se ter total vantagem dos benefícios oferecidos pela cloud computing ainda existem várias implicações arquiteturais que precisam ser observadas. Por exemplo, o desafio inicial pode ser dividir a aplicação em unidades para serem instaladas em um cenário distribuído.

Frequentemente isto é feito usando a abordagem de camadas, como nas aplicações convencionais de multicamadas. Na realidade, cada camada é também uma aplicação distribuída, por exemplo, um servidor de aplicação, e infelizmente a gama de ferramentas para auxílio em projetar tais aplicações ainda não é vasta. Porém o mundo de cloud computing está em constante e rápida evolução, e muito tem sido prometido para a próxima geração de ferramentas de desenvolvimento, especialmente no que diz respeito à orquestração de serviços e provisioning.

No lado da OSGi, diversas pesquisas e desenvolvimentos estão sendo realizados para o sucesso desta proximidade, especialmente para ajudar em alguns pontos que ainda não são triviais na cloud computing, como o deployment distribuído/remoto e manutenção dos serviços de software. Neste cenário de cloud computing, o conceito de módulos (bundles OSGi) e serviços (local e remoto) existentes na OSGi traz interessantes benefícios. A separação de código dentro de módulos encoraja os desenvolvedores a definirem interfaces e, entretanto, criarem um acoplamento e coesão do código. Coesão significa que todas as funções de um módulo devem estar de alguma forma relacionadas. Sendo assim, o módulo somente conterá funcionalidades em comum, significando alta coesão. As funcionalidades que serão publicadas como serviços são exportadas para outros módulos como interfaces para evitar um forte acoplamento. Fraco acoplamento significa que um módulo não depende das implementações internas de um segundo módulo, mas somente das interfaces exportadas.

Como em cloud computing é muito importante poder reutilizar pedaços de software como serviços. Então, caso o baixo acoplamento e alta coesão possam ser atingidos, a base de código pode ser tratada como uma coleção de módulos independentes através de interfaces de serviços com a comunicação sendo gerenciada por um framework OSGi. Para o framework OSGi qualquer POJO pode ser um serviço quando ele é publicado através de uma interface. Os clientes por sua vez podem consumir o serviço publicado utilizando para a localização deste o nome da interface, ou opcionalmente filtrar a partir dos metadados do serviço.

O modelo OSGi é bastante avançado na forma de lidar com módulos. Entretanto, uma característica do projeto complica ou minimiza a utilização da OSGi neste ambiente de cloud, é que o framework OSGi somente lida com serviços rodando em uma mesma JVM. Neste momento, a OSGi distribuída "entra em cena" provendo para os módulos cliente os "proxies" para acessar os serviços remotos. Desta forma, estes clientes podem transparentemente conversar com os serviços remotos publicados em outros containers e rodando em outros nós do cloud.

Recentemente um grupo de trabalho liderado pela OSGi Alliance foi criado para definir uma especificação apropriada para a extensão do framework OSGi em cloud computing. Em um primeiro momento este grupo está focando nas seguintes áreas: Provisioning e configuração, ambiente de gerenciamento, eficiência, debugging, mecanismo de descoberta, migração e modelo de aplicação.

Considerações finais

Com um modelo de desenvolvimento modular e flexível, o framework OSGi está cada vez mais presente no mundo Enterprise. Neste artigo, apresentamos uma visão geral sobre OSGi distribuída, a qual aplica uma extensão ao modelo OSGi, disponibilizando de forma natural e simples a criação de serviços OSGi remotos. Isto traz o framework OSGi para o contexto de aplicações distribuídas, permitindo a integração com diferentes frameworks de middleware.

Não podemos deixar de citar também que aspectos importantes e já conhecidos do mundo Java EE, como persistência, transações, suporte a JMX, WAR e outros, estão a caminho para serem integrados ao framework OSGi•

Referências

Sites:
[1] - http://www.osgi.org/WG/HomePage – Página da OSGi Alliance
[2] - http://www.osgi.org/Specifications – Especificações OSGi
[3] - http://www.eclipse.org/ecf – Eclipse Communication Framework
[4] - http://cxf.apache.org – Apache CXF
[5] - http://www.eclipse.org/equinox – Projeto Equinox, implementação da OSGi
[6] - http://felix.apache.org – Projeto Felix, implementação da OSGi
Livros:
[7] – Jeff McAffer, Paul Vanderlei, Simon Archer. OSGi and Equinox: Creating Highly Modular Java Systems.

Listagem 6. Arquivo descritor do bundle servicos-remotos.dicionario.cliente.

```
Manifest-Version: 1.0

Bundle-ManifestVersion: 2

Bundle-Name: Cliente

Bundle-SymbolicName: servicosremotos.dicionario.cliente

Bundle-Version: 1.0.0.qualifier

Bundle-Activator: servicosremotos.dicionario.cliente.Activator

Bundle-ActivationPolicy: lazy

Bundle-RequiredExecutionEnvironment: J2SE-1.5

Import-Package: org.eclipse.ecf.core;version="3.0.0",

 org.eclipse.ecf.core.identity;version="3.0.0",

 org.eclipse.ecf.osgi.services.distribution;version="1.0.0",

 org.eclipse.ecf.remoteservice,

 org.eclipse.ecf.remoteservice.events,

 org.eclipse.equinox.app;version="1.0.0",

 org.eclipse.equinox.concurrent.future;version="1.0.0",

 org.osgi.framework,

 org.osgi.util.tracker;version="1.4.2",

 servicosremotos.dicionario

Require-Bundle: org.eclipse.equinox.common

Export-Package: servicosremotos.dicionario.cliente
```

Listagem 7. Trecho de código da classe ClienteApp.

```
public class ClienteApp implements IApplication, IDistributionConstants,
ServiceTrackerCustomizer {
   //.....
   public Object start(IApplicationContext appContext) throws Exception {

      // configura o bundle context (para uso com o service trackers)
      bundleContext = Activator.getContext();
      processarArgs(appContext);

      //cria um container ECF para lidar com o serviço remoto
      getContainerFactory().createContainer(containerType);

      // cria um service tracker para gerenciar o serviço IDicionario
      Filter filter = bundleContext.createFilter("(&("
            + org.osgi.framework.Constants.OBJECTCLASS + "="
            + IDicionario.class.getName() + ")(" +
            IDistributionConstants.SERVICE_IMPORTED + "=*))");

      dicionarioServiceTracker = new ServiceTracker(bundleContext,filter, this);
      dicionarioServiceTracker.open();

      iniciaLocalServicoDescobertaCasoExista();

      waitForDone();

      return IApplication.EXIT_OK;
   }

   /**
    * Método é acionado quando uma instância do serviço remoto IDicionario é
    * registrada no container OSGi. Este método está definido na interface
    * ServiceTrackerCustomizer, sendo utilizado pelo framework OSGi. Caso
    * o ambiente de execução seja Java 5, pode ser retornado
    * IDicionario, através de retorno covariante, entretanto as implementa
    * ções do framework OSGi se baseiam no ambiente mínimo Java 1.4.
    */
   public Object addingService(ServiceReference reference) {
      // Since this reference is for a remote service,
      // O serviço retornado é um proxy para a implementação da interface
      // IDicionario
      IDicionario dicionario = (IDicionario) bundleContext.getService(reference);
      return dicionario;
   }
   //.....

}
```