OSGi distribuído: deployment local e execução remota

Marcelo Malcher

Monografia de Seminários de Sistemas Distribuídos

Departamento de Informática – Pontifícia Universidade Católica do Rio de

marcelom@inf.puc-rio.br

Resumo. *A plataforma OSGi fornece um framework que possibilita a instalação, atualização e desinstalação de módulos Java em tempo de execução. A especificação mais recente desta tecnologia define a forma como os serviços oferecidos por estes módulos colaboram entre si dentro de uma única JVM (Java Virtual Machine). Diversas pesquisas propõem trabalhos que ajudam a utilizar esta tecnologia em ambientes distribuídos, sendo possível acessar módulos e serviços que não estejam disponíveis localmente, seja realizando o deployment local – copiando o módulo desejado de algum nó na rede – e/ou executando o serviço através de chamada remota. O objetivo desta monografia é apresentar alguns destes trabalhos e analisar questões como transparência, descoberta de módulos disponíveis, modelo de distribuição e performance dos mesmos.*

1. Introdução

A tecnologia OSGi define um sistema de módulos dinâmicos para linguagem Java e é mantida pela OSGi Alliance – formada em 1999 por um grupo de empresas como IBM, Ericsson, Siemens, Oracle, entre outras – e especifica um *framework* para o gerenciamento de módulos Java que permite a instalação, atualização e desinstalação destes módulos em tempo de execução [1]. Este *framework* é responsável por manter a consistência dos serviços ao controlar a relação de dependência entre os módulos instalados.

O objetivo da tecnologia OSGi é possibilitar o desenvolvimento de aplicações a partir da composição de módulos colaborativos e reutilizáveis. Módulos, que no OSGi são chamados de *bundles*, são arquivos JAR – *Java Archive* – com adições em seu arquivo de manifesto, onde são incluídas informações a respeito dos serviços fornecidos/exportados e dos serviços aos quais dependem, que devem ser instanciados/importados para uma consistente execução do módulo. Todos os serviços fornecidos no OSGi implementam uma determinada interface Java e quando instalados publicam esta interface em um *service registry* local. É através deste mecanismo centralizado que os *bundles* podem verificar se os serviços aos quais depende estão disponíveis localmente, através buscas pelos nomes de suas interfaces. A Figura 1exibe o framework e a relação entre *bundles* e serviços, onde um *bundle* B utiliza um serviço fornecido pelo *bundle* A, assim, estabelecendo uma relação de dependência entre os diferentes *bundles.*

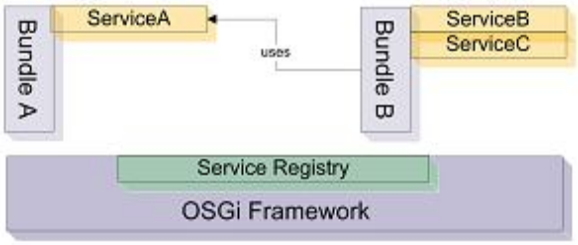

Figura 1 - Relação entre *bundles* e serviços OSGi [4]

1.1. O *framework* OSGi

A definição do *framework* é a parte principal da especificação da tecnologia OSGi, e é onde estão definidas suas principais funcionalidades [2]. A Figura 2 ilustra a divisão do *framework* em determinadas camadas.

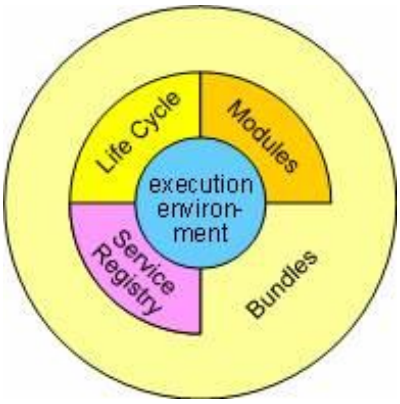

Figura 2 - Framework OSGi [2]

Ambiente de execução é especificação do ambiente de execução Java (p. ex. JSE, CDC, CLDC, MIDP, entre outros, são ambientes de execução válidos para a plataforma OSGi).

Camada dos módulos define como devem ser os módulos Java e possui regras definidas para que os diferentes pacotes Java tanto possam ser carregados e compartilhados entre *bundles*, como se tornarem invisíveis para outros determinados *bundles*.

Camada de ciclo de vida define o modelo de ciclo de vida para os *bundles*, determinando quando estes poderão ser iniciados ou parados, e quando estes deverão ser instalados, atualizados e desinstalados.

Camada de serviços define o modelo de programação dinâmico e conciso que simplifica o desenvolvimento de serviços ao desacoplar suas interfaces de suas implementações. Assim, desenvolvedores podem criar as relações entre serviços através apenas das especificações de suas interfaces.

Através do *service registry*, *bundles* selecionam implementações de serviços disponíveis em tempo de execução a partir de suas interfaces, e podem registrar novos serviços, receber notificações sobre os estados dos serviços atuais ou procurar por novos serviços para melhor se adequar às capacidades do dispositivo atual. De acordo com [1], isto faz

com que os *bundles* possam ser extensíveis após a instalação dos mesmos, visto que estes podem ser atualizados sem a necessidade de reiniciar a aplicação. A Figura 3 exibe a interação entre as diferentes camadas do OSGi e um *bundle*.

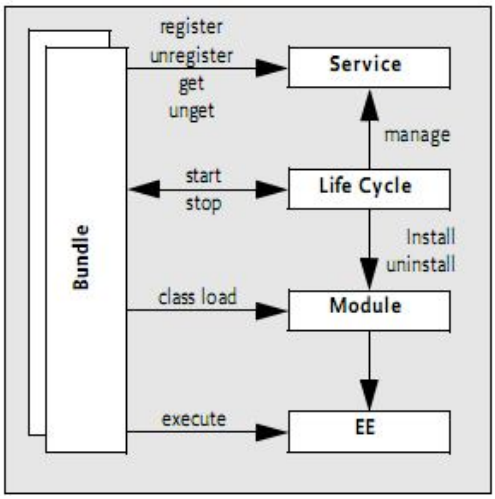

Figura 3 - Interação entre camadas [1]

Exemplos de implementações do *framework* OSGi especificado são o Apache Felix [10], Knopflerfish [11] e OSCAR [13].

2. OSGi Distribuído

Inicialmente a tecnologia OSGi foi desenvolvida para uso em dispositivos embarcados, onde o *framework* executa em um ambiente isolado, sem necessidade de comunicação e interação com outros dispositivos. Com o avanço da tecnologia e suas especificações, o OSGi acabou se tornando padrão para modularização e gerenciamento de quaisquer tipos de aplicações Java, desde ambientes de desenvolvimento a servidores de aplicação [2], visto sua facilidade na atualização dinâmica de *bundles* (componentes) e resolução das dependências em tempo de execução.

A especificação mais recente da tecnologia determina como os serviços disponíveis colaboram entre si em uma única JVM (Java Virtual Machine), determinando que todos os *bundles* devem estar presentes localmente para que possam executar corretamente. Contudo, o modelo de gerenciamento e execução de *bundles* providos pelo *framework*, onde já estão definidos os tratamentos de resolução de dependências e os eventos relacionados ao carregamento e falha de *bundles*, possibilita a pesquisa de diversos trabalhos que buscam tornar o OSGi um ambiente distribuído, no qual os *bundles* possam estar espalhados pela rede, podendo serem descobertos, copiados de outros nós e carregados em tempo de execução, e/ou executados de forma remota.

Diversas pesquisas estão sendo realizados para que isto seja possível, e estes trabalhos variam da forma como distribuem a tecnologia OSGi. Para avaliar os mesmos, são definidos os seguintes critérios:

- **Transparência** é o quanto os recursos OSGi já criados devem ser alterados para que possam ser utilizados de forma distribuída;

- **Mecanismo de descoberta** é a maneira como os *bundles* e serviços requisitados são descobertos na rede;
- **Modelo de distribuição** representa como é dado o acesso ao *bundle* remoto, se é realizado o *deployment* local ou se o serviço é executado através de chamadas remotas;
- **Performance** busca verificar o esforço necessário para acessar um serviço de um *bundle* remoto – desde o tempo de pesquisa até o retorno da execução do serviço.

A seguir são apresentados os trabalhos pesquisados que buscam distribuir a tecnologia OSGi.

2.1. OSGi Bundle Repository – OBR

O OSGi Bundle Repository – OBR é um repositório central de *bundles* OSGi em um servidor, que fornece informações acerca dos *bundles* disponíveis para download em um arquivo XML [3]. Os *frameworks* OSGi podem ler/parsear arquivo XML e sempre atualizar seus *bundles* quando uma nova versão estiver disponível. As dependências do *bundle* também estão contidas no arquivo e podem ser resolvidas a fim de garantir a consistência dos serviços.

É possível também pesquisar por *bundles* explicitamente através de uma interface web. O resultado é listado para o usuário, que pode selecionar diversos *bundles*, conferir suas descrições e verificar as dependências existentes para os mesmos. Depois de verificadas as dependências, o usuário realiza o download e instala os *bundles* localmente em seu *framework*.

A idéia de um repositório central de *bunldes* foi definida primeiramente pelo OSCAR Bundle Repository [12]. OSCAR é uma implementação open-source da tecnologia OSGi [13].

2.3. Newton Project

O projeto Newton [8] apresenta um modelo de componentes distribuídos que se baseia na tecnologia OSGi para modularização de componentes, na tecnologia Jini [7] para infra-estrutura de comunicação distribuída, e no SCA (Service Component Architecture) [15] para descrever as composições de serviços, chamadas no Newton de *composites*. Um *composite* é um conjunto de *bundles* e serviços OSGi que provêm e dependem de múltiplos serviços (ver Figura 4).

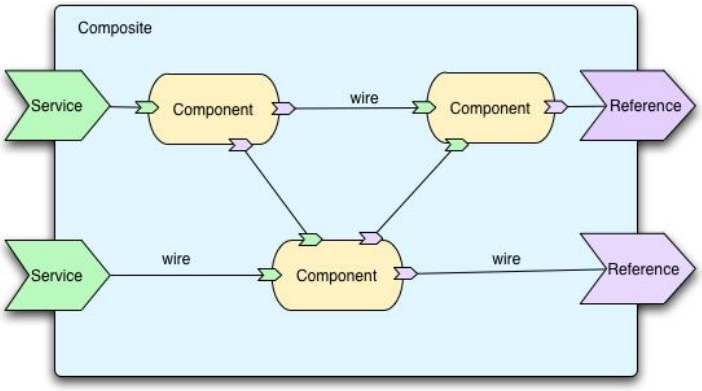

Figura 4 - Um *composite* SCA no Newton [8]

Estes *composites* são formados por outro *bundle* OSGi conhecido como *factory bundle*. Os *factory bundles* contêm ou importam todas as classes e recursos necessários para instanciar um *composite*. O Newton monitora o ciclo-de-vida de um *composite*, verificando as ligações entre seus *bundles*, que podem ser locais através do *framework* OSGi utilizado, ou via Jini, quando os *bundles* estão em diferentes dispositivos. Quando uma dessas ligações é perdida, devido falha na conexão ou desinstalação de um *bundle*, o Newton tenta recriar esta ligação, alterando-a para um serviço alternativo se necessário.

Bundles e outros recursos necessários para a composição de *composites* ficam disponíveis para o Newton quando adicionados ao *Content Distribution System* (CDS). O CDS se divide em duas zonas: *boot zone* e *remote zone*. A primeira se refere a quando o Newton está sendo iniciado, e os recursos para isto estão presentes localmente. Após este estágio, o Newton passa a se comunicar com vários outros dispositivos utilizando o Jini, passando a compartilhar a mesma zona remota com outros dispositivos, e assim podendo copiar sob demanda os *bundles* necessários para formação dos *composites*. O CDS mantém um *OBR* (vide 2.1) para armazenar e disponibilizar os *bundles*.

A Figura 5 ilustra o processo de instalação de um componente no sistema Newton, onde somente o descritor do *composite* é necessário. Durante a instalação, o instalador identifica o *factory bundle* do *composite* a partir do descritor, e delega a instalação para o mesmo. Todos os *bundles* necessários para execução do *composite* são verificados e caso algum não esteja presente localmente, o Newton faz uma busca no CDS, e realiza o *deployment* local.

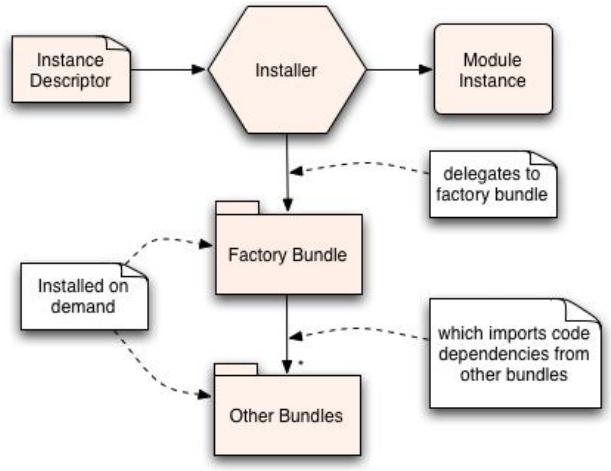

Figura 5 - Processo de instalação de um componente no Newton [8]

2.3. R-OSGi

O R-OSGi (Remote OSGi) permite que aplicações OSGi centralizadas tornem-se distribuídas de forma transparente, onde as chamadas a serviços remotos sejam tratadas como chamadas a serviços de *bundles* locais [4]. A Figura 6 ilustra um exemplo com dois nós conectados através do R-OSGi.

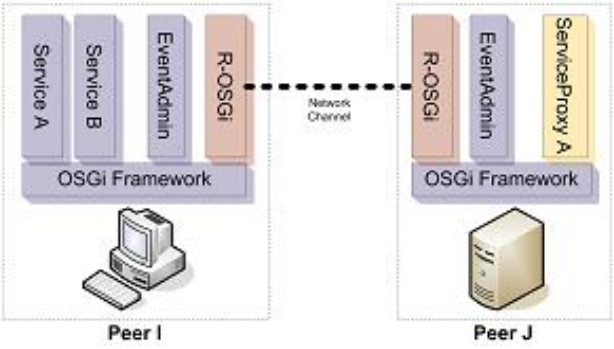

Figura 6 - Arquitetura do R-OSGi [4]

Para que o acesso a *bundles* distribuídos fique transparente ao usuário – ou seja, o usuário não saiba que o serviço acessado é um serviço instanciado em outro dispositivo –, o R-OSGi utiliza de quatro técnicas: geração dinâmica de *proxies*, registro de serviços distribuídos, distribuição transparente de serviços, e injeção de tipos para resolver dependências de tipos distribuídos.

Geração dinâmica de *proxies* é a técnica utilizada pelo R-OSGi para criar de forma transparente nos dispositivos clientes os *proxies* para serviços remotos em tempo de execução. Estes *proxies* atuam como serviços locais e redirecionam todas as chamadas de método para o serviço original presente no dispositivo remoto, e o resultado é enviado de volta para o *proxy* cliente. Os *proxies* são gerados a partir do código da interface do serviço desejado em tempo de execução.

Registro de serviços distribuídos é necessário para localizar serviços em dispositivos remotos, pois a especificação do OSGi prevê um registro de serviços centralizado, que funciona para a procura de serviços presentes localmente. É utilizado um protocolo de descoberta de serviços reativo, que verifica a demanda de *bundles* por determinados serviços. Enquanto isso, cada nó anuncia seus serviços que serão oferecidos de forma remota. Quando o R-OSGi verifica a demanda por determinado um serviço, aliada ao anúncio do mesmo por um nó remoto, este cria um *proxy* para o serviço remoto e o registra este *proxy* no registro local de serviços (*service registry*).

Distribuição transparente de serviços é atingida ao aproveitar as próprias características da tecnologia OSGi. Quando se faz uso de chamadas remotas, problemas como latência de comunicação e de perdas mensagens são básicos em sistemas distribuídos [14]. O R-OSGi se aproveita da forma como são desenvolvidos os *bundles*, onde os desenvolvedores já estão preparados para eventuais falhas relacionadas à dependência entre *bundles*. As falhas de comunicação são apenas mapeadas como falhas decorridas de eventos relacionados ao carregamento ou ausência de um *bundle*. O serviço original também pode disparar exceções – não relacionadas a questões de comunicação – e estas exceções são enviadas para o cliente de forma a reproduzir efeito igual caso o *bundle* estivesse presente localmente.

Injeção de tipos é utilizada pelo R-OSGi para garantir a consistência de tipos para a interface dos serviços remotos. É feito uma análise no código do serviço remoto registrado em busca de todos os tipos utilizados que estejam contidos no *bundle* do serviço remoto, e que pertençam a um pacote Java que seja exportado pelo mesmo. Estas classes são adicionadas a uma lista de injeção. Quando o *proxy* para determinado serviço é gerado, estas injeções são materializadas e armazenadas no *bundle* do *proxy*. Os pacotes de todas as outras classes não referenciadas na lista de injeção, são declarados como importados do *bundle* do *proxy*. Os pacotes das classes geradas pela lista de injeção são declarados como exportados do *bundle* do proxy para garantir a consistência do *framework*.

Sobre a implementação, o mecanismo de registro distribuído R-OSGi utiliza o jSLP, uma implementação Java do Service Location Protocol (SLP) [16], e a comunicação formada entre dois *frameworks* é realizada através de mensagens via conexões TCP.

Após a realização de diversos testes, a performance do R-OSGi para chamadas de serviços remotos se mostrou superior aos resultados obtidos utilizando Java RMI.

2.4. *uOSGi*

O *uOSGi* tem como objetivo a descoberta dinâmica de *bundles* e serviços de forma centralizada e descentralizada, a seleção automática do *bundle* a partir de requisitos funcionais e não-funcionais e o suporte para descoberta, seleção e *deployment* de *bundles* de forma transparente [5]. Para que isto seja possível, o *uOSGi* estabelece uma comunicação *peer-to-peer* com os nós vizinhos utilizando JXTA [9].

Segundo os requisitos do *uOSGi*, os recursos OSGi devem ser descobertos automaticamente de acordo com a descrição definida do *bundle* desejado. Após *bundles* apropriados serem descobertos, um desses é selecionado e copiado para o *framework*

OSGi local. As dependências com outros *bundles* OSGi devem ser automaticamente resolvidas, e caso elas não sejam satisfeitas, um mecanismo de *backtracking* é acionado e um outro *bundle* é selecionado. A Figura 7 exibe a seqüência de passos a serem realizados para encontrar um *bundle* adequado e que possa ser utilizado localmente. Este processo deve ser realizado de forma automática em tempo de execução, sem a necessidade de intervenção do usuário. Para que o *bundle* apropriado seja carregado dinamicamente, os *bundles* devem conter metadados descrevendo suas propriedades, requisitos e dependências [6].

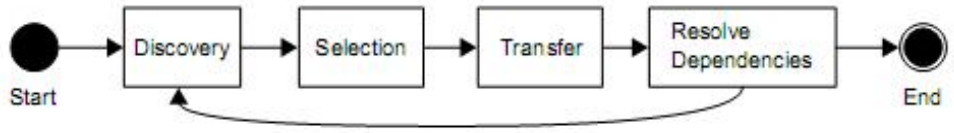

Figura 7 - Requisitos do *uOSGi* para seleção do *bundle* apropriado [5]

A arquitetura do *uOSGi* é dividida em duas partes: os serviços P2P que formam a infra-estrutura de comunicação, utilizada para descoberta eficiente de *bundles*, onde o *code sharing service* é responsável pelo carregamento e compartilhamento de recursos e o *code discovery service* suporta a descoberta e publicação de metadados dos recursos; e os serviços *uOSGi* que fornecem a interface OSGi para a plataforma, onde o *loading service* provê seleção e carregamento automático de recursos OSGi, e o *repository service* que gerencia e fornece os recursos locais já carregados e disponíveis. Com a ajuda dos outros dois, o *resolver service* é responsável por controlar automaticamente a dependência entre *bundles* e serviços OSGi.

O *loading service* espera como parâmetro o nome da interface ou do *bundle* a ser carregado. A Figura 8 exibe todo o processo de carregamento de um recurso OSGi a partir de requisitos obrigatórios e opcionais. A diferença é que requisitos obrigatórios influem no processo de seleção de recursos OSGi, enquanto requisitos opcionais influem na avaliação de qual recurso é mais adequado para determinada situação.

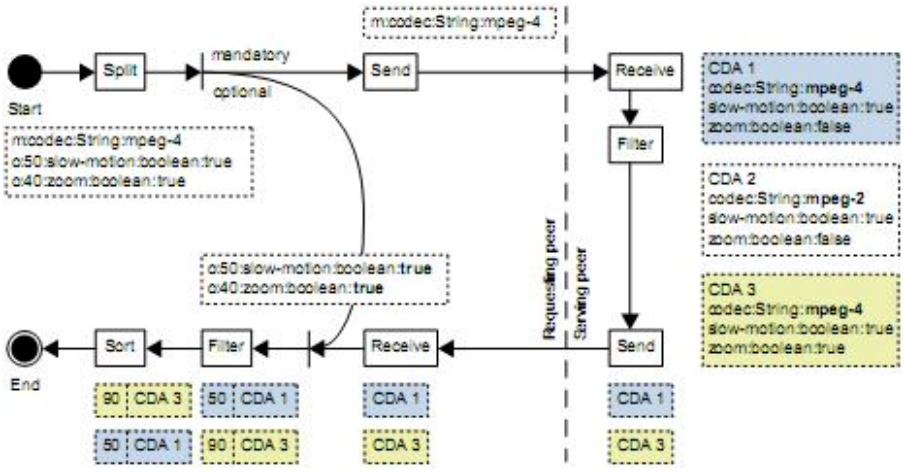

Figura 8 - Processo de carregamento de um recurso OSGi [5]

O *repository service* gerencia os recursos já carregados localmente. Assim, primeiramente as aplicações procuram por *bundles* e serviços no repositório local, e só depois o *loading service* é usado para encontrar *bundles* remotos utilizando JXTA.

O *resolver service* é utilizado para verificar e realizar o *deployment* automático das dependências dos *bundles* selecionados. Para resolver estas dependências, primeira é verificado no repositório local, e só após verifica-se se estão disponíveis remotamente. Estas dependências são incluídas nos arquivos de manifesto dos *bundles*, facilitando assim o trabalho do *resolver service*.

No *uOSGi* existem duas formas de iniciar o processo de carregamento de um novo *bundle* (*loading service*): de forma manual ou automática. Um usuário pode utilizar o console de gerenciamento do *framework* OSGI, e utilizando comando específicos do *uOSGi* pode explicitamente requisitar a resolução das dependências de um *bundle* já instalado ou requisitar a instalação de um novo *bundle* resolvendo automaticamente suas dependências.

Da forma automática, o *uOSGi* provê o *uOSGi Service Tracker*. Um *service tracker* padrão OSGi controla os *bundles* monitorando diretamente se os serviços requisitados estão disponíveis durante o tempo de execução, verificando continuamente a inclusão e remoção de serviços no *service registry* local. O *uOSGi Service Tracker* é uma extensão deste *service tracker* padrão para que automaticamente realize o *deployment* de *bundles* que implementem serviços requisitados de acordo com requisitos não-funcionais. O *uOSGi service tracker* adota a seguinte política para *deployment* de *bunldes* requisitados: verifica primeiramente no repositório local (*service registry*), procura por *bundles* remotos que atendam aos requisitos especificados, utiliza R-OSGi (ver 2.3) para execução remota.

Apesar de se relacionarem quanto a questão da distribuição do OSGi, com o *uOSGi* é possível tanto realizar o *deployment* local de *bundles* remotos como acessá-los remotamente via R-OSGi. Porém, deve-se criar uma forma de definir quando utilizar um ou outro mecanismo de distribuição.

Para avaliar o *uOSGi*, foram realizados testes comparando-o ao OBR (ver 2.1) e ao próprio R-OSGi. Em relação ao OBR, foram feitas avaliações com diferentes tamanhos e constatou-se que a tecnologia JXTA apresenta problemas quando os *bundles* tem tamanho maior que 1MB, e como esperado, o OBR obteve melhores resultados. Contudo, *bundles* raramente tem tamanho superior a 1MB, portanto, os tempos obtidos pelo *uOSGi* podem ser considerados satisfatórios.

Quando comparado ao R-OSGi, fica evidente a diferença nas duas abordagens, e como podem se complementar. Enquanto que o R-OSGi é utilizado sempre de forma remota, o processo de descoberta e *deployment* local do *uOSGi* é obviamente a tarefa mais custosa. Contudo, após o *bundle* instalado localmente, sempre uma chamada local será mais rápida e eficiente que uma chamada remota.

2.5. Análise

O OBR, o único especificado pela OSGi Alliance, apresenta uma forma simples de distribuir a plataforma OSGi, mas representa um ponto central de falha no ambiente distribuído, ou seja, se por algum motivo o servidor não esteja disponível para acesso, as atualizações e buscas por novos *bundles* fica impossibilitada. Da mesma forma o

Newton, que utiliza um OBR para realizar o *deployment* de *bundles* necessários para a formação de *composites*, possui a mesma desvantagem citada. Ambos os trabalhos também tem a desvantagem de serem invasivos, ou seja, os *bundles* devem ser criados para utilizar especificamente suas tecnologias.

Dentre os quatro trabalhos apresentados, apenas o R-OSGi funciona de forma totalmente transparente, mesmo para os *bundles* OSGi já desenvolvidos. O *uOSGi* realiza o *deployment* local de forma transparente, e necessita apenas de inclusões nos arquivos de manifesto para auxiliar seus serviços de descoberta e carregamento de *bundles* remotos. Estes trabalhos apresentam formas automáticas e ortogonais para descoberta e acesso a *bundles* e serviços distribuídos. Portanto, como já previsto e feito em [5], podem ser integrados para uma solução distribuída mais completa, deixando em aberto a questão de qual forma ser a mais adequada em determinados momentos, quando é mais vantajoso realizar o *deployment* local ao invés de realizar chamadas remotas.

3. Conclusão

Esta monografia apresentou o OSGi, um sistema de módulos dinâmicos para Java, e os trabalhos realizados com o intuito de distribuir esta tecnologia, que apresenta um modelo de componentes totalmente adaptável para um ambiente distribuído. As formas de distribuir esta tecnologia vêm se aperfeiçoando com o decorrer do tempo e já existem trabalhos na própria OSGi Alliance para que na próxima especificação da tecnologia este assunto já venha documentado (vide RFC 119 – Distributed OSGi [sem referência oficial]).

Provavelmente o futuro do OSGi distribuído estará relacionado com a integração entre o R-OSGi e o *uOSGi*. Os diferentes modelos de distribuição – *deployment* local e execução remota – são adequados para determinadas situações e ambientes de execução, contudo, deve ser pesquisado como definir quando um modelo é mais adequado que o outro. Esta questão deixada em aberto deve estar no centro das próximas pesquisas acerca da distribuição da tecnologia OSGi.

Referências

[1] OSGi Alliance. OSGi Service Platform: Core Specification, Release 4, Version 4.1. Technical report, 2007.

[2] OSGi Alliance. http://www.osgi.org (2008)

[3] OSGi Alliance. RFC-0112 Bundle Repository, February 2006.

[4] J. S. Rellermeyer, G. Alonso, and T. Roscoe. R-OSGi: Distributed Applications Through Software Modularization. In Middleware ’07, volume 4834 of LNCS, 2007.

[5] *uOSGi: An Infrastructure for Ubiquitous OSGi Services with Dynamic Code Deployment

[6] H. Schmidt, J. H. Yip, F. J. Hauck, and R. Kapitza. Decentralised Dynamic Code Management for OSGi. In 6th MiNEMA workshop. ACM, 2008

[7] Waldo, J.: The Jini architecture for network-centric computing. Communications of the ACM 42(7) (1999) 76–82.

[8] Paremus: The Newton Project. http://newton.codecauldron.org (2006).

[9] L. Gong. JXTA: A Networking Programming Environment. IEEE Internet Computing, 5(3), 2001.

[10] Apache Foundation: Apache Felix. http://felix.apache.org (2008)

[11] Gatespace Telematics SA: Knopflerfish OSGi. www.knopflerfish.org (2003)

[12] Oscar Bundle Repository. http://oscar-osgi.sourceforge.net (2008)

[13] OSCAR – OSGi Framework. http://oscar-osgi.sourceforge.net/ (2008)

[14] Waldo, J.,Wyant, G.,Wollrath, A., Kendall, S.: A Note on Distributed Computing. Technical Report SMLI TR-94-29, Sun Microsystems Labs (1994)

[15] Service Component Architecture – Open SOA Collaboration http://www.osoa.org/display/Main/Service+Component+Architecture+Home (2008)

[16] Guttman, E., Perkins, C., Veizades, J.: RFC 2608: Service Location Protocol v2. IETF. (1999)