FELIPE RODRIGUES DO PRADO
JOÃO PAULO NAKAJIMA PEREIRA

MODULARIZAÇÃO DE SOFTWARES

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE
2015

FELIPE RODRIGUES DO PRADO
JOÃO PAULO NAKAJIMA PEREIRA

MODULARIZAÇÃO DE SOFTWARES

Pesquisa para desenvolvimento do projeto apresentado à disciplina de TCC 1 do curso de Sistemas de Informação como requisito parcial para obtenção de créditos.

Orientador: Ms. Márcio Emílio Cruz Vono de Azevedo.

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE
2015

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BSD	*Berkeley Software Distribution*
CSS	*Cascading Style Sheet*
EJB	*Enterprise JavaBeans*
HTML	*HyperText Markup Language*
ICC	*Inatel Competence Center*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
JDBC	Java *Database Connectivity*
JEE	*Java Platform, Enterprise Edition.*
JMS	Java *Message Service*
JPA	Java *Persistence* API
JVM	Java *Virtual Machine*
NTT	*Nippon Telegraph and Telephone*
OSGi	*Open Services Gateway initiative*
SENAC	Serviço Nacional de Aprendizagem Comercial
SQL	*Structured Query Language*
SRA	*Systems Research and Applications Corporation*
UNIFEI	Universidade Federal de Itajubá
UNIVAS	Universidade do Vale do Sapucaí
XHTML	*Extensible HyperText Markup Language*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*

SUMÁRIO

INTRODUÇÃO

O desenvolvimento de um *software* não envolve apenas métodos de programação. Para desenvolvê-lo é necessário preocupar-se com planejamento, engenharia, metodologia e tecnologia a ser utilizada. Esses fatores influenciam na manutenção, atualização e expansão do mesmo. Dessa forma, definir tais fatores é fundamental para que o *software* seja flexível a mudanças. Pensando dessa maneira, será demonstrado um modelo de desenvolvimento modular, utilizando a especificação OSGi.

OSGi, abreviação para *Open Services Gateway Initiative*, possibilita o desenvolvimento de *softwares* em módulos, disponibilizando o gerenciamento dos mesmos e facilidades na manutenção e expansão da aplicação.

Segundo Fernandes (2009), um sistema modular possui algumas propriedades. Deve ser autocontido, os módulos podem ser incluídos, retirados, instalados ou desinstalados. Outra propriedade é ter alta coesão. Um módulo deve cumprir apenas sua finalidade, ou seja, deve fazer somente as funções que lhe foram atribuídas. O baixo acoplamento é outra propriedade muito importante. Um módulo não precisa se preocupar com implementações de outros módulos que interagem com ele, além de permitir alterá-lo sem a necessidade de atualizar os outros.

Muitos *softwares* são desenvolvidos de maneira semelhante à modularização, divididos em partes, tendo cada parte uma responsabilidade. Porém não são realmente modularizados, ou seja, não atendem ao conceito de modularização como descrito acima. Dois exemplos de projetos de *softwares* que foram desenvolvidos utilizando a especificação OSGi e que atendem a definição de modularização, são, IDE[1] Eclipse, ferramenta de desenvolvimento de softwares e o GlassFish, servidor de aplicações JEE[2].

O uso da modularização, com certeza, traz grandes benefícios para o desenvolvimento e manutenção de um software. Poder parar parte de uma aplicação para fazer uma manutenção ou poder instalar novas funcionalidades, garantindo que todas as outras partes restantes continuem funcionando normalmente, seria uma característica notável da aplicação.

Fernandes (2009) forneceu uma visão geral sobre OSGI, mostrando seus benefícios e

1 IDE – Abreviação para *Integrated Development Environment.*
2 JEE – Abreviação para Java *Platform*, *Enterprise Edition*.

salientando a importância da plataforma Java possuir um melhor suporte à modularidade, até demonstrando com um exemplo simples as premissas e vantagens do OSGi.

Mayworm (2010) demonstra a tecnologia OSGi no contexto de aplicações distribuídas, permitindo a disponibilização de seus serviços remotamente, integrando com diferentes *frameworks* de *middleware* para o desenvolvimento de aplicações empresariais.

Malcher (2008) apresenta um modelo de componentes adaptável para se usar em ambientes distribuídos. Também analisa alguns aspectos como modelo de distribuição, transparência, descoberta de novos módulos disponíveis, desempenho e *performance* dos mesmos. Afirma ainda que para se escolher entre um modelo de distribuição – *deployment* local ou execução remota – é necessário analisar o contexto e o objetivo da aplicação, pois cada um se adapta melhor a determinadas situações e ambientes de execução.

Portanto, após despertar um grande interesse pelo modelo de desenvolvimento modular, será realizado estudos sobre a especificação OSGi, e ainda, o desenvolvimento de uma aplicação modular que demonstre tal modelo de desenvolvimento.

2 OBJETIVOS

Os objetivos desta pesquisa serão demonstrados a seguir.

2.1 Objetivo geral

- Demonstrar o modelo de desenvolvimento modular utilizando a especificação OSGi para aplicações empresariais.

2.2 Objetivos específicos

Para atingir o objetivo geral será apresentado os seguintes objetivos específicos:

- Pesquisar as melhores práticas e *frameworks* que contribuem para a produtividade no desenvolvimento modularizado;
- Desenvolver uma aplicação que exemplifique o modelo de desenvolvimento modularizado através da especificação OSGi;
- Obter através dos resultados uma conclusão sobre as vantagens do modelo de desenvolvimento modular.

3 JUSTIFICATIVA

Diante dos vários segmentos e constantes mudanças empresariais, novos *softwares* são desenvolvidos e precisam estar preparados para acompanhar as modificações que ocorrem nos processos de uma empresa. Dessa forma o desenvolvimento separado em módulos seria uma solução para atender empresas de diferentes setores em vez de criar um *software* para cada ramo empresarial. Além disso a utilização de módulos em um sistema torna mais flexível sua manutenção, pois a mesma é realizada nos módulos, não afetando o restante da aplicação.

O trabalho tem relevância na visão acadêmica diante da aprendizagem da tecnologia OSGi, além de agregar conceitos sobre o desenvolvimento de softwares de forma modularizada. E ainda, por ser pouco utilizada no mercado devido à sua complexidade de aprendizagem, este trabalho visa reunir informações e disponibilizar uma documentação com práticas de programação e vantagens que essa tecnologia pode oferecer.

Desenvolvedores de *softwares* procuram sempre a utilização de novas práticas e tecnologias produtivas. O modelo de desenvolvimento modular através da tecnologia OSGi oferece diversos benefícios que ajudam na expansão e manutenção de uma aplicação. Isto também se torna útil para as empresas, pois para novas funcionalidades seriam criados módulos, e a manutenção do sistema seria realizada somente no módulo específico, sem a necessidade de parar o restante do sistema.

Como a adoção dessa nova prática e tecnologia proporciona a reutilização de código, consequentemente não haverá a necessidade do desenvolvimento repetido de funcionalidades, economizando tempo, energia elétrica e recursos computacionais, algo que se tornou de grande valor atualmente, visando o desenvolvimento harmonioso e sustentável entre o ser humano e a natureza.

4 QUADRO TEÓRICO

Para que se possa desenvolver qualquer solução, é necessário o uso de algumas ferramentas, teorias e tecnologias. Abaixo serão apresentadas algumas delas, que serão utilizadas no desenvolvimento desta pesquisa, bem como sua utilidade.

4.1 Modularização

Falar sobre design de camadas, design modular, módulos, granularidade e mudanças.

4.2 Especificação OSGi

Grandes aplicações enfrentam um problema conhecido no seu desenvolvimento, a complexidade. Uma maneira de resolver esse problema é quebrar o sistema em partes menores, ou módulos. Devido a sua flexibilidade, a linguagem de programação Java permitiu construir sobre sua plataforma um poderoso sistema de módulos, o OSGi, que oferece suporte à construção de sistemas modulares (FERNANDES, 2009).

Segundo a OSGi Alliance (2015), a tecnologia OSGi é um conjunto de especificações, que segue um modelo de desenvolvimento em que as aplicações são dinamicamente formadas por componentes distintos e reutilizáveis.

A especificação teve início em março de 1999 pela própria OSGi Alliance. Seus principais desafios na época não era desenvolver uma solução para a execução de diferentes versões de um mesmo projeto na mesma aplicação, mas sim de elaborar uma maneira que diferentes componentes que não se conhecem possam ser agrupados dinamicamente sob o mesmo projeto (OSGI ALLIANCE, 2015).

Segundo Knoernschild (2012), para que uma aplicação seja modularizada, seus

módulos devem ser instaláveis, gerenciáveis (parar, reiniciar e ser desinstalado sem interromper o restante da aplicação), reutilizáveis (utilizar em outros sistemas), combináveis (combinar com outros módulos), não guardar estado e oferecer uma interface clara.

OSGi Alliance (2015) diz que a API[3] disponibilizada permite o fácil gerenciamento (instalação, inicialização, parada e atualização) dos pacotes, bem como possibilita enumerar os pacotes e utilizar seus serviços.

De acordo com OSGi Alliance (2015), essa tecnologia tem como benefício a redução da complexidade do desenvolvimento de modo em geral, como por exemplo, aumento da reutilização de módulos, incremento da facilidade de codificação e teste, gerenciamento total dos módulos, sem a necessidade de reiniciar a aplicação, aumento do gerenciamento da implantação e detecção antecipada de *bugs*.

Os *bundles* – como os módulos são chamados no contexto da OSGi – consomem ou disponibilizam serviços. Eles estão organizados de uma maneira que formam a tríade consumidor, fornecedor e registro de serviços, na qual pode-se consumir serviços ou disponibilizá-los. Para a liberação de um novo serviço, deve-se registrá-lo em um catálogo, onde este teria visibilidade por parte de *bundles* externos que consumiriam esses serviços disponibilizados (OSGI ALLIANCE, 2015).

A especificação OSGi define uma camada chamada *Life Cycle* que gerencia o ciclo de vida e provê uma API que permite o desenvolvedor instalar, desinstalar, iniciar, parar e atualizar os *bundles* (BOSSCHAERT, 2012).

Bartlett (2009) afirma que um *bundle* pode passar por vários estados em seu ciclo de vida. Na Figura 1, a seguir, é demonstrado cada um desses estados e suas funções.

3 Abreviação para *Application Program ming Interface*.

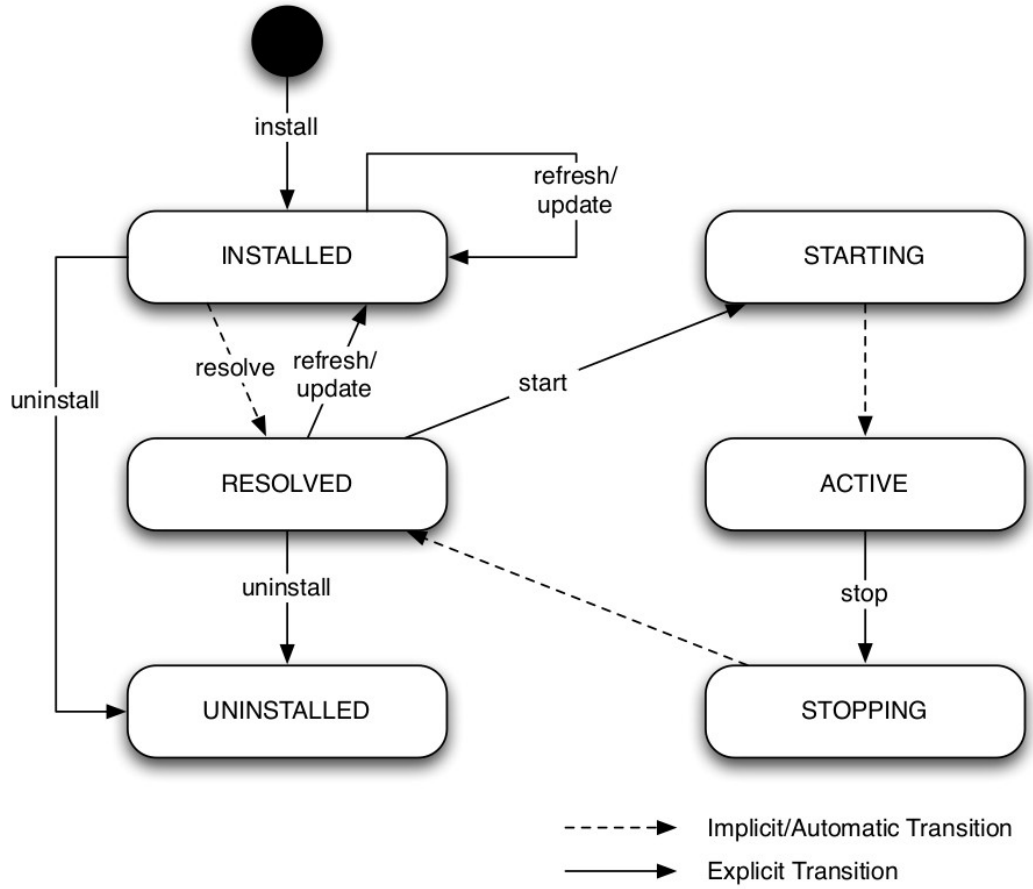

Figura 1 – Estados dos *bundles* dentro de um contêiner. Fonte: OSGi In Practice (2009)

Esses estados e funções podem ser assim explicados:

- **Installed:** o *bundle* é instalado com êxito;
- **Resolved:** avalia se o *bundle* está pronto para ser iniciado ou parado, validando informações dos *bundles*, seus pacotes e versões, e ainda a versão do Java necessária para executar o *bundle*;
- **Starting:** estado apenas de transição, neste estado o método BundleActivator.start() do *bundle* é invocado para que o mesmo seja iniciado. Por ser um estado de transição, nenhum *bundle* fica parado nesse estado;
- **Active:** nesse estado o *bundle* está validado e ativo, somente esperando alguma requisição para ser utilizado;
- **Stopping:** também um estado de transição, parecido com o estado *Starting*, porém

nesse estado o método BundleActivator.stop() é chamado para parar o *bundle*, com isso o mesmo é transferido para o estado *Resolved* novamente;

- **Uninstalled:** quando ocorre a desinstalação do *bundle*, não se pode transitá-lo para nenhum outro estado e este não é mais representado no *framework*. Se o *bundle* for reinstalado, ele assume o papel de um novo *bundle*.

Segundo Fernandes (2009), após a instalação do *bundle*, o mesmo fica armazenado em um meio persistente do *framework* até ser desinstalado explicitamente.

Um aspecto importante é que um *bundle* pode ser executado em qualquer implementação OSGi. Ou seja, um *bundle* desenvolvido e testado em uma implementação pode ser executado em qualquer outra implementação. (LUCENA, 2010).

O *framework* é o centro da especificação da plataforma OSGi, definindo um modelo para o gerenciamento do ciclo de vida da aplicação, registro dos serviços e ambiente de execução (FERNANDES, 2009).

BOSSCHAERT, 2012

A OSGi Alliance (2015), define que o *framework* OSGi utiliza uma arquitetura de camadas em sua implementação, conforme demonstrado na Figura 2 a seguir.

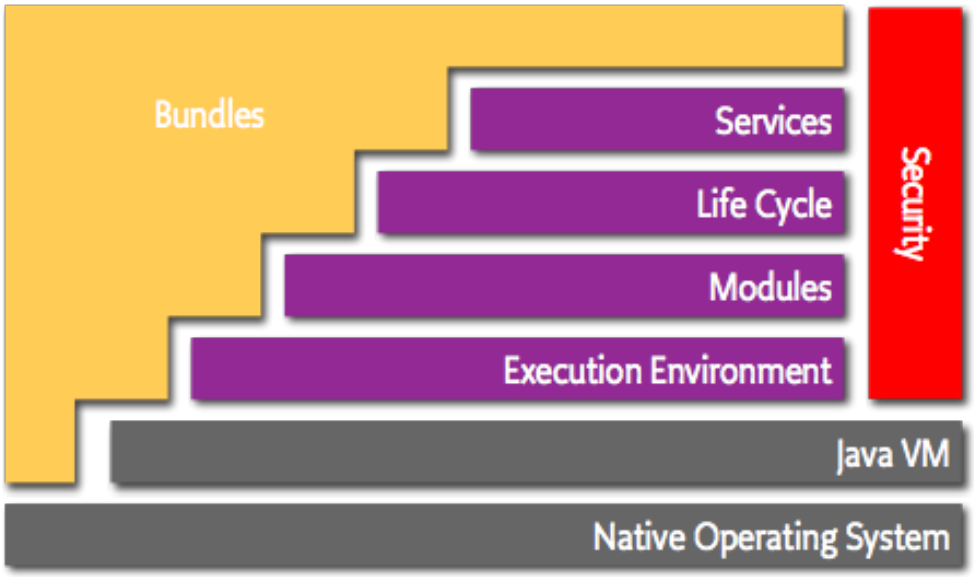

Figura 2 – Estrutura de camadas. Fonte: OSGi Alliance (2015)

As camadas utilizadas pelo *framework* OSGi podem ser assim explicadas:

- **Bundles:** são os componentes (módulos) criados pelos desenvolvedores de softwares;
- **Services:** camada responsável por conectar os *bundles* de forma dinâmica oferecendo um modelo *publish-find-bind*. Esta camada permite registrar e obter os serviços dos módulos;
- **Life Cycle:** camada que provê uma API para instalar, iniciar, parar, atualizar, e desinstalar os *bundles*;
- **Modules:** camada que administra a importação e exportação de códigos de um *bundle*.
- **Execution Environment:** camada que define quais os métodos e classes estão disponíveis em uma plataforma específica.

Segundo Bartlett (2009), a OSGi Alliance apenas define uma especificação do *framework*, por isso existem várias implementações do mesmo nos dias atuais, mas as quatro a seguir merecem destaque por serem as principais e terem seu código livre.

- **Equinox:** é o *framework* mais usado atualmente, sua grande popularidade provém de fazer parte do gerenciador de *plug-ins* do Eclipse. Pode ser encontrado em muitos outros *softwares*, como os da IBM Lotus Notes e Websphere Application Server. Encontra-se sob a licença Eclipse Public License (EPL) e implementa a versão 4.1 das especificações OSGi1.
- **Knopflerfish:** é uma implementação bem madura e popular da versão 4.1 da especificação. É desenvolvido e mantido pela empresa Sueca Makewave AB, a qual oferece seu produto tanto em uma versão gratuita sob a licença BSD quanto uma versão comercial com suporte oferecido pela empresa.
- **Felix:** é a implementação mantida pela Apache Software Foundation, se encontra na versão 4.1 e possui foco na compactação e facilidade de incorporação do *bundle* na aplicação. Está sob a licença Apache 2.0.
- **Concierge:** é uma implementação bem compacta e altamente otimizada da especificação versão 3. É mais usado para plataformas com recursos limitados, como por exemplo, aplicações móveis e embarcadas. Se encontra sob a licença BSD.

O OSGi especifica um conceito de versionamento de componentes, em que os

módulos podem trabalhar com versões diferentes, seguindo uma consistente estrutura de numeração, utilizando três segmentos de números e um segmento alfanumérico, são eles, *major*, *minor*, *micro* e *qualifier*, respectivamente. Exemplo, "1.2.3.beta_1". Esses segmentos também podem ser omitidos, formando por exemplo a versão "1.2". Isso se torna necessário para resolver o problema de incompatibilidade de versões entre módulos (FERNANDES, 2009).

A OSGi Alliance, responsável pela plataforma OSGi trabalha na especificação de uma infraestrutura, que possa distribuir e realizar a comunicação de aplicações e serviços baseados em componentes de forma transparente.

4.3 Java

4.4 Spring Framework

4.5 Hibernate

4.6 PostgreSQL

4.7 HyperText Markup Language (HTML)

4.8 JavaScript

4.9 Cascading Style Sheet (CSS)

4.10 Test-Driven Development (TDD)

5 QUADRO METODOLÓGICO

REFERÊNCIAS