Universidade Federal de Pernambuco

Graduação em Ciência da Computação

Centro de Informática

2 0 1 0 . 2

Um Mecanismo de Monitoramento de Serviços na Plataforma OSGi

TRABALHO DE GRADUAÇÃO

Aluno – Fábio Almeida Melo (fam2@cin.ufpe.br)
Orientador – Nelson Souto Rosa (nsr@cin.ufpe.br)

Dezembro de 2010

Universidade Federal de Pernambuco

Graduação em Ciência da Computação

Centro de Informática

2010.2

Fábio Almeida Melo

Um Mecanismo de Monitoramento de Serviços na Plataforma OSGi

Este trabalho foi apresentado à graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Profº. Dr. Nelson Souto Rosa

Dezembro de 2010

Dedico,

Ao meu pai,

Sílvio.

Agradecimentos

Primeiramente a Deus, por me dar forças em todos os momentos difíceis que passei e pelos momentos alegres.

À minha família que sempre me apoiou, confiou e me deu forças. Em especial, ao meu pai, que infelizmente não está mais aqui conosco, mas, sei que está feliz com essa conquista onde quer que esteja e à minha mãe, exemplo de pessoa e de vida que eu amo muito.

Ao professor Nelson Rosa, pela oportunidade de trabalho e ajuda no decorrer do TG. E a Fábio Souza, pela ajuda e acompanhamento do trabalho.

Aos grandes amigos que fiz durante a faculdade e aos meus amigos de longas datas que dividiram muitos momentos de alegria e de tristeza comigo.

Muito obrigado também a todos que contribuíram pela minha passagem pela universidade.

Obrigado!

Sumário

1. Introdução 1
1.1. Objetivos 1
1.2. Estrutura 2
2. Conceitos Básicos 3
2.1. OSGi 3
2.1.1. Visão Geral do OSGi Framework 3
2.1.2. Bundles 3
2.1.3. Camada de Serviços 4
2.1.4. Camada de Ciclo de Vida 5
2.1.5. Camada Modular 5
2.1.6. Camada de Segurança 6
2.1.7. Ambiente de Execução 6
2.1.8. Interação entre Camadas 6
2.2. JMX 7
2.2.1. Arquitetura JMX 7
2.2.2. Instrumentação de Recursos 8
2.2.2.1. MBeans 8
2.2.2.1.1. Standard MBean 9
2.2.2.1.2. Dynamic MBean 9
2.2.2.1.3. Model MBean 9
2.2.2.1.4. Open Mbean 9
2.2.3. Agente JMX 10
2.2.3.1. MBeanServer 10
2.2.3.2. Agente 10
2.2.4. Serviços Distribuídos 10
2.2.4.1. Conector 11
2.2.4.2. Adaptador 11
2.3. Considerações Finais 11
3. Proposta 12
3.1. Visão Geral 12
3.2. Arquitetura 12
3.3. Implementação 14
3.3.1. Ambiente de Desenvolvimento 14

3.3.2. Diagrama de Classes 14
3.3.3. MasterAgent 16
3.3.4. Cascading Service 16
3.3.5. SubAgent 16
3.3.6. Interface Gráfica e suas funcionalidades 17
3.4. Considerações Finais 19
4. Exemplo 20
4.1. Instalando o Plugin 20
4.2. Configurando os Subdomínios 21
4.3. Domínio Master 21
4.4. Serviços 22
4.5. Considerações Finais 24
5. Conclusão 25
5.1. Trabalhos Futuros 25
Referências 26

1. Introdução

O grande número de dispositivos fixos e móveis com recursos computacionais cada vez mais potentes tem levado ao desenvolvimento de aplicações distribuídas, heterogêneas e dinâmicas. Estas aplicações têm uma característica fundamental em comum, a necessidade de se adaptarem e evoluírem em tempo de execução.

Para suprir essa necessidade, várias pesquisas têm sido feita na área para a construção, adaptação e evolução de aplicações dinâmicas dentre elas, se destaca a composição orientada a serviços.

Na literatura [1], a orientação a serviços propõe a idéia de aplicações desenvolvidas a partir de blocos de construção reutilizáveis que são os serviços. Estes são vistos como unidades funcionais comunicantes e contratualmente definidas em uma descrição de serviço contendo informações sintáticas, semânticas e comportamentais publicadas em um repositório podendo ser descobertos, consumidos e/ou consumir outros serviços dinamicamente.

Atualmente, existem diversas plataformas que possibilitam a execução de aplicações orientadas a serviços, denominadas SOAs (Service *Oriented Architectures*). Dentre elas destaca-se a OSGi (*Open Services Gateway Initiative*)[2], uma plataforma centralizada onde os serviços são instalados, alterados e removidos dinamicamente e publicados em um repositório para serem utilizados por outras aplicações da plataforma.

Devido a facilidades providas por OSGi, aplicações dinâmicas podem ser construídas com mais facilidade na plataforma. Este dinamismo das aplicações pode decorrer por diferentes razões, tais como a necessidade de inclusão de novas funcionalidades, alterações no ambiente de execução, ou mesmo substituição de componentes visando melhorias relativas à qualidade de serviço (QoS).

Então, para suprir a necessidade de melhoria da aplicação através da escolha de novos serviços, este Trabalho de Graduação propõe um mecanismo de monitoramento da qualidade de serviço.

1.1. Objetivos

Este Trabalho de Graduação tem como objetivo principal a construção de um mecanismo de monitoramento da qualidade de serviço (e.g. tempo de resposta, disponibilidade) de aplicações distribuídas na plataforma OSGi.

1.2. Estrutura

Além do Capítulo de introdução, que explana sobre o conteúdo e objetivos, este trabalho tem mais outros 4 capítulos.

O Capítulo 2 apresenta os conceitos básicos do trabalho: a especificação OSGi e a JMX.

O Capítulo 3 mostra como foi feita a implementação da ferramenta fornecendo uma visão geral, a arquitetura, partes mais importantes da implementação e uma visão das funcionalidades do plugin e sua interface gráfica.

O Capítulo 4 descreve um exemplo de uso que serve como um guia para futuros usuários. Além disso, fornece uma noção maior do uso da ferramenta.

Por último, no Capítulo 5, concluímos o trabalho apresentando, as contribuições, as limitações e os trabalhos futuros.

2. Conceitos Básicos

2.1. OSGi

A tecnologia OSGi consiste em um conjunto de especificações, desenvolvida pela OSGi Alliance, que define um sistema de componentes dinâmicos para a linguagem de programação Java. Através de uma implementação dele, é possível em tempo de execução instalar,ativar,desinstalar componentes conhecidos como *bundles*.Além disso, também pode-se registrar, desregistrar e receber notificações sobre serviços [2].

2.1.1. Visão Geral do OSGi Framework

A especificação do OSGi tem como núcleo a definição de um *framework*[2]. Nele estão contidas as principais funcionalidades do OSGi. A Figura 2-1, mostra uma visão geral do *framework* dividido-o em camadas.

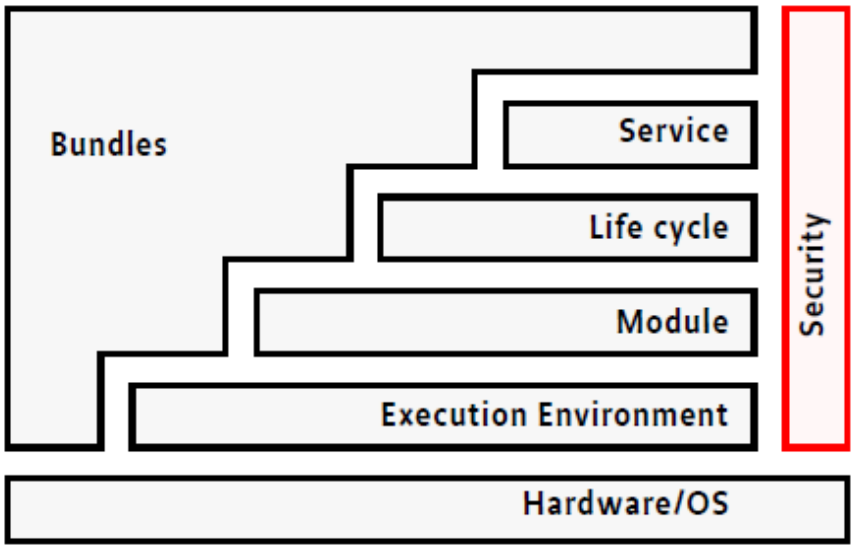

Figura 2-1: Camadas OSGi

2.1.2. Bundles

O *framework* OSGi define de forma padronizada e genérica uma modularização para Java que tem como unidade básica o *bundle*. Ele é composto por classes Java, um arquivo

MANIFEST.MF (que contém metadados) e outros recursos opcionais. Através dele, é possível exportar e importar pacotes Java com o Import-package e o Export-package declarados no manifesto[2]. A Figura 2-2 abaixo mostra um exemplo de um MANIFEST.MF.

```
Manifest-Version: 1.0
Private-Package: teste
Bundle-Version: 0
Tool: Bnd-0.0.384
Bnd-LastModified: 1289333184726
Bundle-Name: helloworld
Bundle-ManifestVersion: 2
Created-By: 1.6.0_22 (Sun Microsystems Inc.)
Bundle-Activator: teste.HelloWorldActivator
Import-Package: com.sun.jdmk.remote.cascading,javax.management,javax.m
 anagement.remote,org.osgi.framework;version="1.5"
Bundle-SymbolicName: helloworld
```

Figura 2-2: Exemplo de Manifest.MF

Um *bundle* é implantado como um arquivo JAR (Java ARchieve File) armazenando o conteúdo em um formato ZIP (ZIP Format File) definido por[6]. Este JAR é composto por[2]:

- Um MANIFEST contendo informações sobre o *bundle*, por exemplo: quais pacotes são necessários para executar o *bundle* e/ou quais pacotes ele exporta e informações para que o *framework* possa o instalar e ativar o *bundle* de forma correta.

- Recursos para fornecer funcionalidades como: classes Java, outros JARs, arquivos HTML,XML, entre outros.

- Informações opcionais no diretório OSGI-OPT do arquivo JAR ou em um dos seus subdiretórios .

2.1.3. Camada de Serviços

Primeiramente, um serviço OSGi é definido como um objeto Java definido sobre uma ou mais interfaces Java registrados através do *service register*. Ele é executado no *bundle* e através dele é possível registrar serviços, procura-los ou receber notificações sobre a mudança de estado[2].

Já a camada de serviços provê a comunicação entre os *bundles* e é definida como um modelo colaborativo e dinâmico altamente integrado com a camada do ciclo de vida. Ela é dinâmica pois, é capaz de lidar com mudanças exteriores e estruturas subjacentes e colaborativa porque é possível publicar,procurar e consumir serviços. Além disto, ela também facilita o desenvolvimento, uma vez que separa as interfaces dos serviços das implementações dele.

2.1.4. Camada de Ciclo de Vida

No *framework* OSGi, a camada de ciclo de vida é responsável pelos controle da segurança e pelos estados dos *bundles*[2].

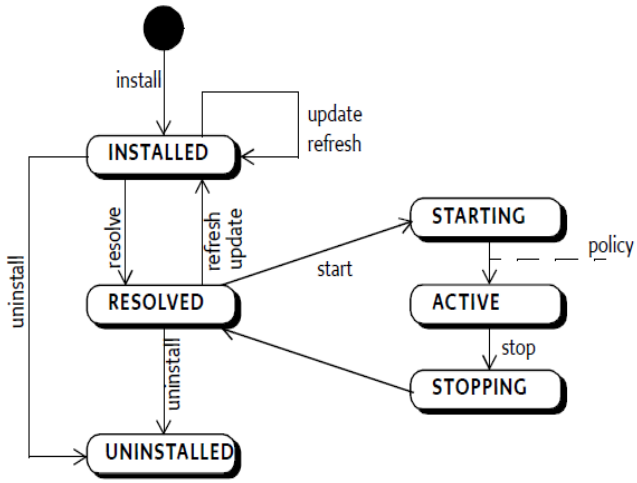

Figura 2-3: Cilco de Vida de um *Bundle*

- *Installed* – Quando o *bundle* é instalado com sucesso.
- *Resolved* – Esse estado indica que o *bundle* está pronto para ser iniciado ou parado. Neste estado, todas as classes Java utilizadas pelo *bundle* estão disponíveis.
- *Starting* – O *bundle* foi iniciado e o método *start* do *Activator* foi chamado, mas, ainda não retornou.
- *Active* – O *bundle* está em execução. Neste estado, o método *start* foi chamado e retornou.
- *Stopping* – O *bundle* está parando. Neste estado, o método *stop* foi chamado, mas, ainda não retornou.
- *Uninstalled* – O *bundle* foi desinstalado.

2.1.5. Camada Modular

A camada modular do *OSGi framework* define como são os módulos Java. Como dito anteriormente, a unidade de modularização é o *bundle*. Esta camada determina a forma

como os pacotes e as classes Java são compartilhados e carregados através do MANIFEST.MF e do Activator.

2.1.6. Camada de Segurança

A camada de segurança do OSGi é baseada na Java 2 Security Architeture[7]. Ela é a responsável por fornecer infra-estrutura para a gerenciamento e implantação de aplicações executadas no ambiente OSGi. Além disso, ela é uma camada opcional do *framework* e que fica subjacente a camada de serviços.

2.1.7. Ambiente de Execução

No ambiente de execução OSGi é onde os *bundles* são implantados. Este é o da especificação do ambiente de execução Java como, por exemplo, o J2SE.

2.1.8. Interação entre Camadas

A intereção entre as camadas do OSGi dão ao *framework* uma grande flexibilidade, pois, as camadas e os *bundles* interagem dinamicamente[2]. É possível, por exemplo, registrar novos serviços, procurar por outros para se adaptar a QoS(Qualidade de Serviço, do inglês, *Quality of Service*) requeridas ou capacidades de um dispositivo, também pode-se instalar novos *bundles* em tempo de execução e até mesmo modificar os existentes sem resetar o sistema.

A Figura 2-4 mostra como é feita esta interação:

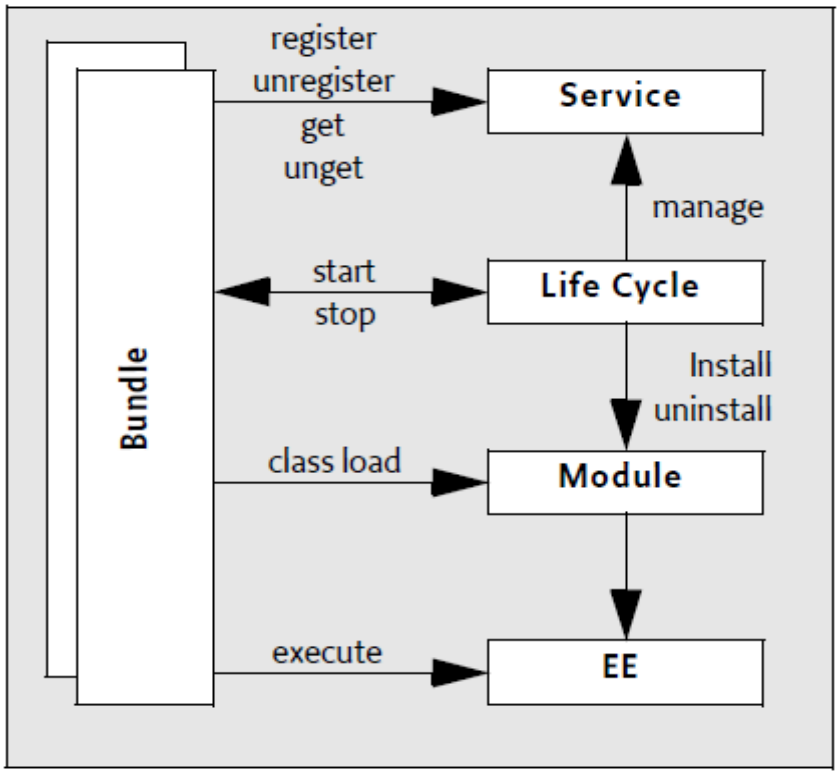

Figura 2-4: Interação entre Camadas OSGi

2.2. JMX

A tecnologia JMX (*Java Management Extensions*) provê de uma maneira dinâmica, simples e padronizada o gerenciamento dos recursos de aplicações, serviços, dispositivos ou até mesmo a JVM (*Java Virtual Machine*)[11]. Ela foi adicionada a plataforma *Java 2 Plataform*, *Standard Edition* (J2SE) 5.0 [12].

Com JMX, cada recurso é instrumentado por um ou mais *MBeans*(*Maneged Beans*) registrados em um *MBeanServer* que atua como um agente de gerenciamento. Além do *MBeanServer*, ainda há uma coleção de serviços para a manipulação direta dos recursos e para disponibiliza-los para os aplicativos dos recursos remotos definidos pela especificação da JMX[11].

Para que os recursos disponibilizados sejam acessados remotamente, a tecnologia JMX define conectores padrões. Eles usam protocolos diferentes que proveêm a mesma interface de gerenciamento.

2.2.1. Arquitetura JMX

A arquitetura da JMX está dividida em três níveis: Instrumentação, Agente e Serviços Distribuídos[9]. A Figura 2-5 a seguir dá uma idéia dessa divisão.

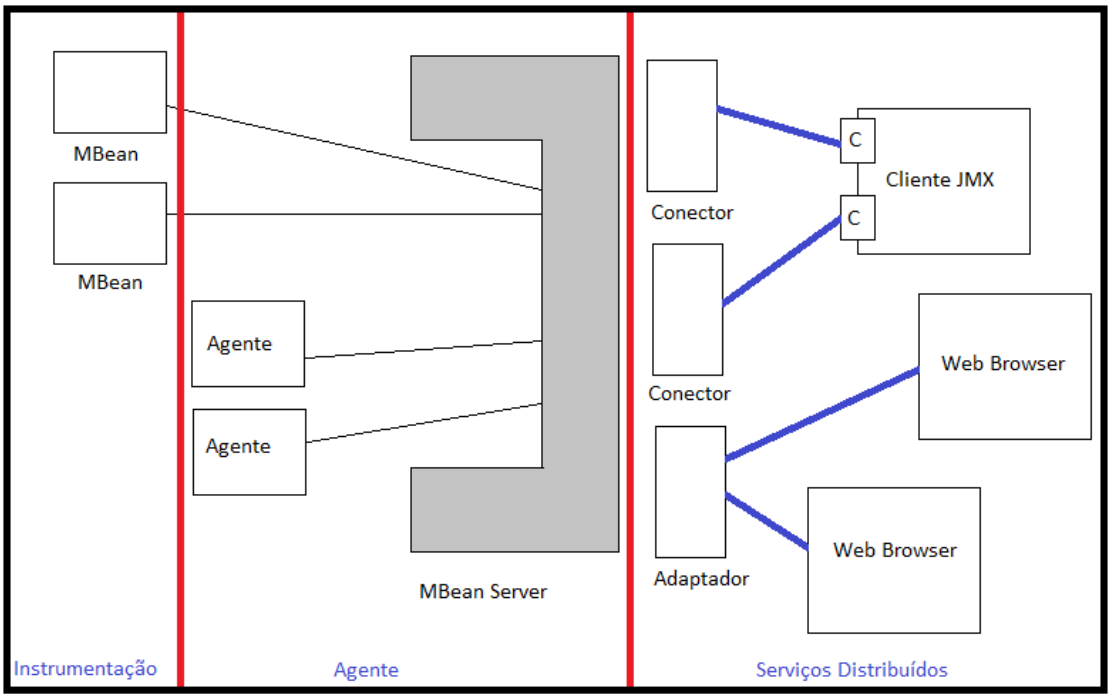

Figura 2-5: Arquitetura JMX

2.2.2. Instrumentação de Recursos

Ao nível de intrumentação é onde ocorre a coleta de informação a respeito dos recursos gerencíaveis. Isto, é feita através dos *MBeans*.

2.2.2.1. *MBeans*

Os *MBeans* são objetos Java que implementam uma interface e segue o padrão de projeto Listener e Emitter para informa o *MBeanServer* as mudanças de registro e as mudanças internas respectivamente. Ele segue quatro regras básicas[9]:

- O estado do recurso deve ser completamente descrito através de métodos de *get* e *set.*

- Deve seguir um dos padrões existentes*: standard, dynamic, model* ou *open*
- O *MBean* deve ter ao menos um construtor público.
- Ele não pode ser declarado como *abstract.*

2.2.2.1.1. *Standard MBean*

O *Standard MBean* é o mais simples de todos. Ele fornece uma interface estática composta por propriedades e operações. As propriedades são atributos da classe expostas através de métodos *getters* e *setters*, enquanto as operações são os outros métodos[13].

2.2.2.1.2. Dynamic MBean

O *MBean* dinâmico expõe suas propriedades e operações apenas em tempo de execução. A interface que ele implementa deve conter o método *getMBeanInfo* retornando um objeto do tipo *MBeanInfo.* Este objeto contém uma lista de atributos, operações e outras informações de gerenciamento[13].

2.2.2.1.3. Model MBean

O *model* é um *MBean* genérico e configurável usado para aparelhar qualquer recurso dinamicamente, tornando-o gerenciável em tempo de execução[13]. Para isto, a aplicação de gerenciamento deve fornecer uma interface de gerenciamento de acordo com o *model MBean* para expor e especificar um objeto que implementa o recurso.

2.2.2.1.4. Open Mbean

O *open MBean* é um tipo de *MBean* dinâmico com restrições de tipos de dados. Este *MBean* descobre objetos gerencíaveis em tempo de execução e permite que aplicações e administradores entendam e usem[9].

2.2.3. Agente JMX

A camada de agentes da especificação JMX é baseada em duas partes: o *MBean Server* e os agentes de serviços JMX. Enquanto, este último fornece informações adicionais obrigatórias da especificação como escalonamento e carregamento dinâmico, o primeiro, funciona como um registro de *MBeans* e um *broker* de comunicação entre as aplicações e os agentes JMX[9].

2.2.3.1. *MBeanServer*

Como dito anteriormente, o *MBeanServer* funciona como registro de *MBeans* para isto, é utilizada a classe *ObjectName* do JMX que serve como identificação única dos *MBeans*. Uma vez que, os *MBeans* são registrados, os agentes JMX fazem consultas através do *MBeanServer* passando o nome do objeto. Então, é feito um *lookup* no registro através do nome fornecido e retorna uma referência ao *MBean* procurado.

2.2.3.2. *Agente*

Um *JMX agent* é um agente de gerenciamento padrão que controla diretamente os recursos e os fazem disponíveis remotamente para as aplicações de gerenciamento. Um agente inclui um *MBeanServer*, um conector para permitir acesso pela aplicação e um conjunto de serviços para gerenciar os *MBeans*.

2.2.4. Serviços Distribuídos

A tecnologia JMX permite o gerenciamento através do uso de interfaces disponibilizadas através dos conectores ou adaptadores. Estes podem usar qualquer tipo

existente de protocolo de comunicação como o SNMP (*Simple Netowork Management Protocol*) ou através de algum protocolo proprietário.

2.2.4.1. *Conector*

O conector consiste em duas partes: o conector cliente e o conector servidor. No lado do servidor, o conector é anexado ao *MBeanServer* e funciona com um *listener* de requisições do cliente. Já o no lado do cliente, é o responsável por se conectar ao servidor. Um conector servidor pode estabelecer várias conexões e com vários clientes, enquanto o conector cliente só se conecta a um servidor[9].

2.2.4.2. *Adaptador*

Um adaptador, diferentemente de um conector, não possui um componente no cliente. Ele é executado no servidor e torna o estado do *MBeanServer* apropriado para o reconhecimento pelo cliente[9].

2.3. Considerações Finais

Este capítulo apresentou as duas principais tecnologias usadas no trabalho. Foram explicados as arquiteturas da plataforma OSGi e da JMX. Além disso, também apresentamos os conceitos de *MBeans*, *bundles*, serviços, conectores e agentes que serão fundamentais para o entendimento do próximo capítulo.

3. Proposta

Como dito no Capítulo 1, este trabalho visou a construção de um mecanismo de gerenciamento de serviços na plataforma OSGi. Para isso, foi desenvolvido um plugin para a Visual VM.

A Visual VM é uma ferramenta que vem junto ao Java e fornece uma interface gráfica para a visualização detalhada de informações sobre aplicações baseadas na tecnologia Java[14]. Além disso, este trabalho, teve como base a implementação já existente de um *plugin* de gerenciamento de *bundles* na plataforma OSGi[15].

3.1. Visão Geral

Em termos gerais, o plugin desenvolvido acessa as informações através de um único domínio. Este, denominado *Master*, se conecta a diversos outros domínios, através de JMX, de maneira que seja possível acessar vários *runtimes* OSGi, remotos ou não, com uma única interface de gerenciamento. Assim, estende-se a idéia utilizada em [15] onde era possível acessar apenas um *runtime* por vez.

Através da interface, é possível se conectar ou desconectar-se dinamicamente de qualquer *runtime* OSGi. Além disso, também é possível instalar,desinstalar, parar ou inciar um *bundle* em um ambiente OSGi conectado ao domínio *Master*, bem como, ter acesso aos metadados do *bundle* e aos serviços fornecidos por ele.

3.2. Arquitetura

Baseado nas necessidades do desenvolvimento de uma aplicação capaz de se conectar a diversos *runtimes* OSGi foi elaborada uma arquitetura em três camadas: interface, domínio Master e subdommínios. A arquitetura faz com que a interface se comunique apenas com a camada do domínio Master que serve com intermedio para os subdomínios.

A camada dos subdomínios é onde se encontram os ambientes OSGi em execução. Estes ambientes podem está funcionando localmente ou remotamente. Para que este ambiente seja gerencíavel pelo *plugin* é necessário que ele possua o *bundle* SubAgent

ativado. Este *bundle* é o responsável por criar um conector JMX para o domínio Master se conectar e ter acesso aos *bundles* e serviços.

A segunda camada, a do domínio Master, faz o intermedio entre a interface de administração e os domínios gerenciáveis. Esta camada é responsável por se conectar aos subdomínios e expor os *MBeans* para gerenciamento da ferramenta.

A camada de interface tem como função permitir gerenciamento dos ambientes de execução de maneira simples. Através dela, o usuário pode:

- se conectar e desconectar a *runtimes* OSGi;
- gerenciar *bundles* (instalar,ativar,parar e desinstalar);
- ter acesso aos serviços e metadados fornecidos pelos *bundles*;
- ver as informações gerais dos *runtimes* OSGi.

A Figura 3-1 a seguir mostra uma visão lógica da arquitetura.

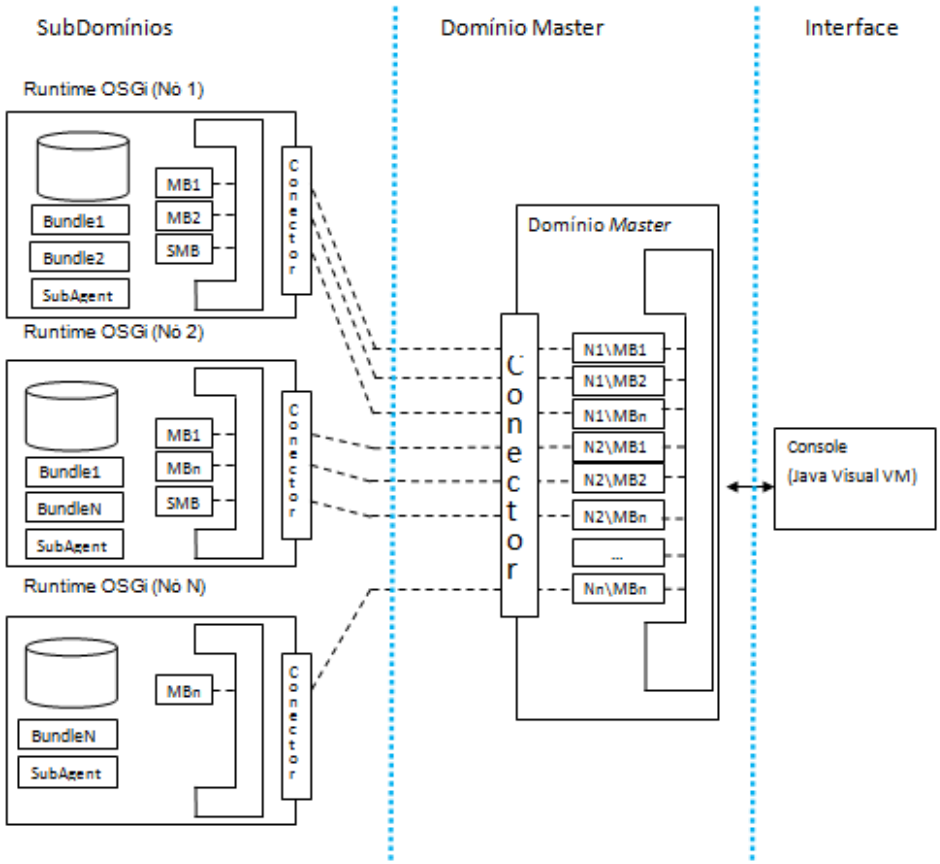

Figura 3-1: Arquitetura da Ferramenta

3.3. Implementação

Nesta seção, mostraremos como a ferramenta foi desenvolvida ao longo do trabalho. Na subseção seguinte, mostraremos as tecnologias utilizadas para o desenvolvimento. E em seguida, será mostrado um diagrama de classes e explicado as partes fundamentais da implementação.

3.3.1. Ambiente de Desenvolvimento

Para o desenvolvimento deste trabalho foram utilizadas diversas ferramentas. Para o desenvolvimento do *plugin* foi utilizada a IDE NetBeans [16]. Já para o desenvolvimento dos *bundles* SubAgents foi o usado o IDE Eclipse [17] com o plugin BndTools[18]. E finalmente, como *framework* OSGi foi o utilizado o Felix Apache [19] .

3.3.2. Diagrama de Classes

A partir da arquitetura apresentada, foram desenvolvidos dois diagramas de classes simplificado apresentados na Figura 3-2 e na 3-3. O primeiro diagrama contém as camadas de Domínio Master e Interface. Já o segundo possui as classes utilizadas para a gereção do *bundle* que é executado nos subdomínios.

A classe *BundlesView* é a classe que contém a maior parte das classes usadas na construção da interface, porém, essas classes não foram colocadas no diagrama para simplifica-lo e por não ser de fundamental importância para o entendimento da ferramenta.

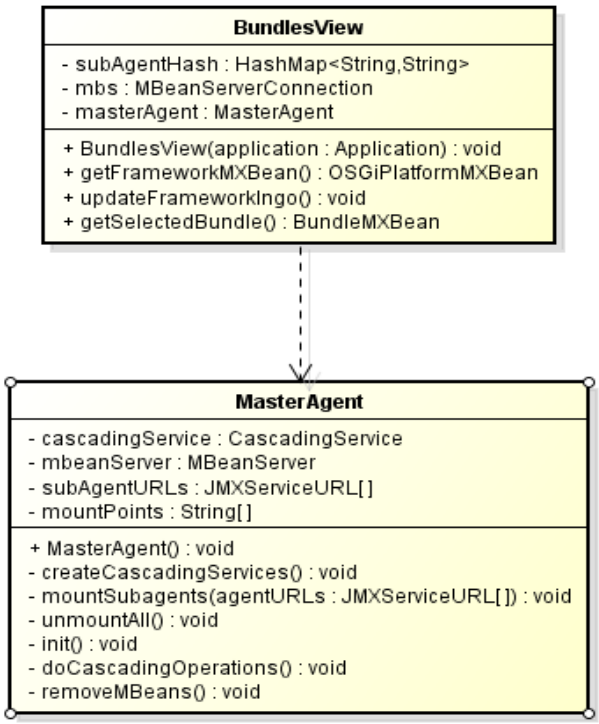

Figura 3-2: Diagrama de Classes do *Plugin*

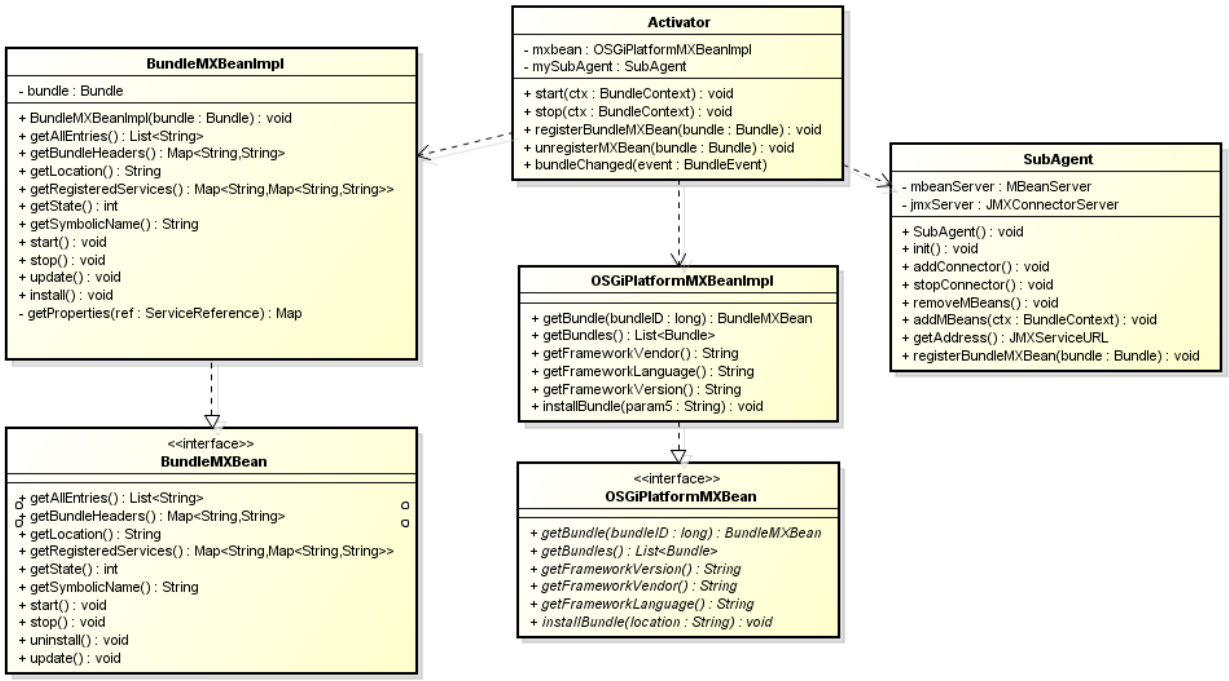

Figura 3-3: Diagrama de Classes do *Bundle* SubAgent

3.3.3. MasterAgent

O MasterAgent é o agente responsável pela conexão com os subdomínios. Ele possui um *MBeanServer* onde os *MBeans* dos subdomínios são montados e podem ser acessados para gerenciamento. Para que um *MasterAgent* se conecte a um subdomínio é preciso conhecer o endereço do SubAgent e possuir o mesmo conector dele que no caso do plugin foi utilizado o JMX *Message Protocol* (JMXMP). Além disso, um *Cascading Service* é registrado no *MBeanService* usado no acesso aos *MBeans* gerenciados.

3.3.4. Cascading Service

Para que fosse possível acessar os *MBeans* localizados em um *runtime* OSGi remoto através do *MBeanServer* do domínio master foi utilizado o *Cascading Service*. Através dele,foi possível montar vários *MBeansServers* em um único *MBeanServer* que fica no domínio *Master*[8]. A figura a seguir mostra como criar um *CascadingService*.

```
CascadingService service = new CascadingService();
ObjectInstance cascadingInstance =
        mbeanServer.registerMBean(service, null);
```

Figura 3-4: Criando o *CascadingService*

O *Cascading Service* vem disponível na JDMK[8] e é implementado como um *MBean* e pode ser gerenciado dinamicamante. Logo, é possível adicionar ou remover *runtimes* OSGi em tempo de execução através dos *subAgents*. Como mostrado no código,o registro dele no *MBeanServer* ocorre igual aos outros *MBeans*.

Para cada *SubAgent* que é montado no *MasterAgent* o *Cascading Service* cria um *ProxyCascadingAgent*. Através dos *proxies* todo gerenciamento é realizado de maneira transparente. Para realizar a operação de montagem é só utilizar um método do *CascadingService* passando a URL de conexão.

3.3.5. SubAgent

O *SubAgent* é o agente cujo o *Cascading Service* é conectado em relação ao seu *MasterAgent*. Nele é criado um conector do mesmo tipo encontrado no Master para que essa conexão possa ser feita.

Este agente fica no subdomínio e foi implementado como um *bundle*. Para que o domínio *Master* possa se conectar a ele é fornecida uma URL no momento de sua ativação.

Como esse *bundle* usa classes não fornecidas pelo OSGi, ele necessita importar classes de outro *bundle*. Então, o *bundle* JDMK foi criado com essa de função de importar pacotes.

3.3.6. Interface Gráfica e suas funcionalidades

A seguir serão explicados todas as funcionalidades que fazem parte do *plugin* que foi desenvolvido. É importante ressaltar que este trabalho teve como objetivo a construção de uma ferramentea de gerenciamento. A busca por pontos de gerenciamento é feito de forma não automática, ou seja, o usuário deve saber de antemão a URL do *runtime* a ser gerenciado, pois, não há nenhum servidor de nomes. A figura a seguir, mostra a interface inicial da ferramenta na VisualVM.

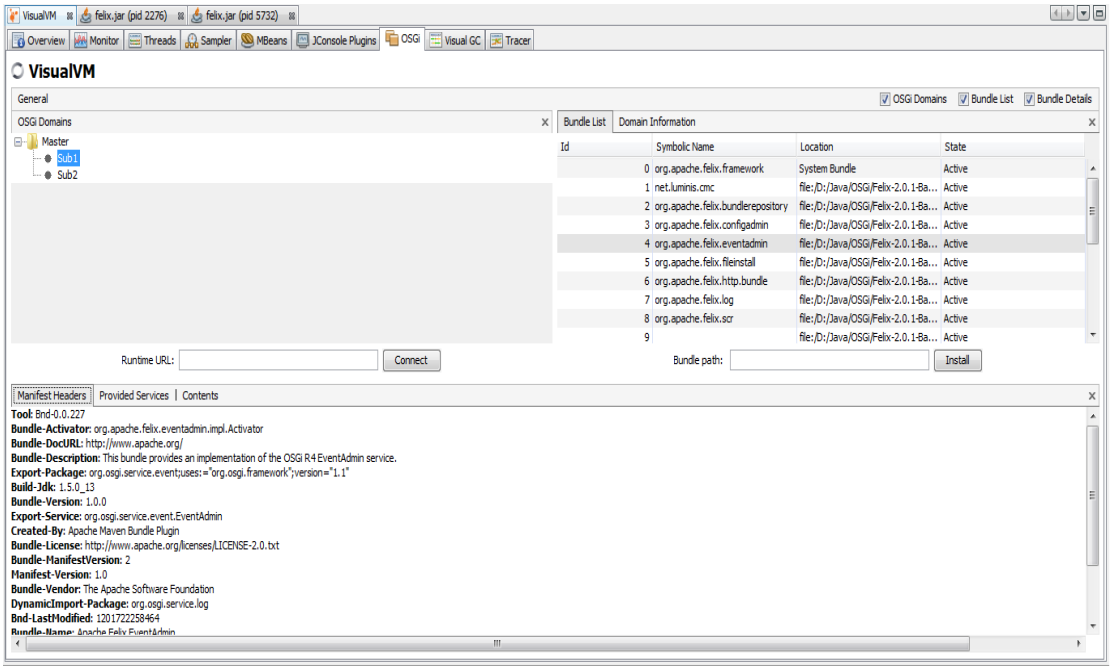

Figura 3-7: Interface da Ferramenta

A interface do *plugin* é subdividida em três regiões. A parte localizada no canto superior esquerdo, chamada de OSGi *Domains*, exibe os domínios que estão sendo gerenciados em forma de árvore. Nessa parte é possível conectar novos domínios digitando a URL do domínio desejado no campo *Runtime* URL. Além disso, também é possível selecionar um nó da árvore para removê-lo ou para visualizar informações desse nó.

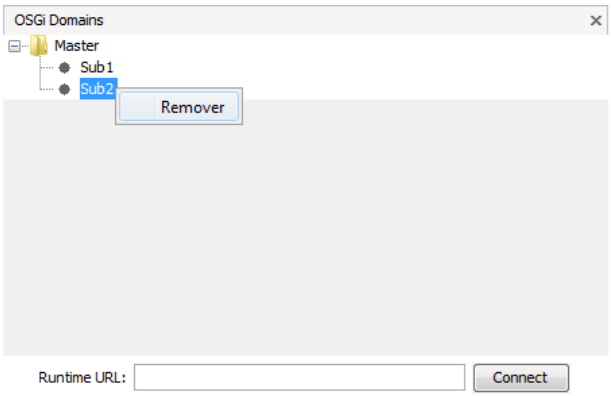

Figura 3-8: Árvore de Domínios

Na região localizada na parte superior à direita é onde se visualiza as informações gerais do nó selecionado na árvore. Através da aba *Domain Information* são exibidas informações como: o número total de *bundles* no domínio, a versão e o desenvolvedor do *framework*. Ainda na mesma região, porém, na aba *Bundle List* é possível visualizar todos os *bundles* do domínio selecionado através de uma tabela que informa o a identificação, o nome, a localização e o estado atual do *bundle*. Ao selecionar um *bundle* na tabela é possível executar quatro ações: inicia-lo, para-lo, atualiza-lo ou desinstala-lo.

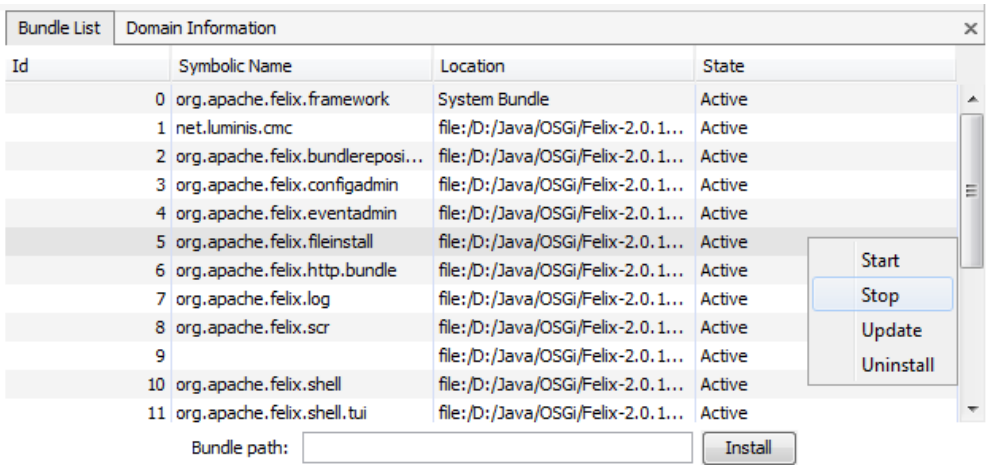

Figura 3-9: *Bundle List*

No canto inferior da ferramenta existe três abas: *Manifest Headers*, *Provided Service*s e *Contents*. Por meio da aba *Manifest Headers* as informações do MANIFEST.MF do *bundle* selecionado são exibidas. Já na aba *Provided Services* são vistos os serviços fornecidos pelo *bundle* e seus respectivos valores. E por úlimo, na aba *Contents* são exibidas as classes Java do *bundle* escolhido.

3.4. Considerações Finais

Ao longo deste Capítulo foi explicado o processo de desenvolvimento da ferramenta. Por meio da visão geral e da arquitetura foi possível ter um entendimento geral de como a ferramenta se comporta da arquitetura e como as partes envolvidas se comunicam.

Através do diagrama de classes e da explicação dos componentes utilizados para o desenvolvimento foi possível entender como o *plugin* foi desenvolvido.

4. Exemplo

Neste capítulo será apresentado um exemplo de uso da ferramenta desenvolvida que servirá como um guia do usuário.

Foi elaborado um cenário, onde existem dois *runtimes* OSGi em execução com *bundles* de agência que oferecem serviços de cotações de moedas e informações sobre as caracteristicas desses serviços. Para mostrar tal cenário será mostrada desde a instalação ao uso da ferramenta.

4.1. Instalando o Plugin

Inicialmente, é preciso ter a VisualVM que encontra-se disponível em [14] ou na própria JDK do Java 6. Após, executar a VisualVM deve ser feita a instalação do plugin desenvolvido através do menu como mostra a figura abaixo.

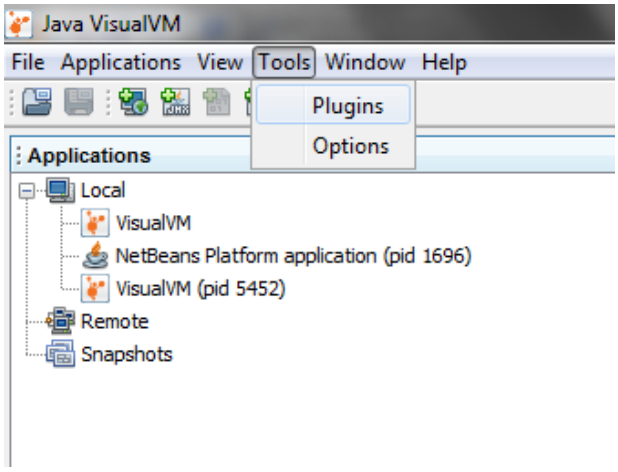

Figura 4-1: Menu da VisualVM

Em seguida, deve-se selecionar a aba downloaded da janela plugin que foi aberta e clicar em “add plugin...” e selecionar o plugin a ser instalado.

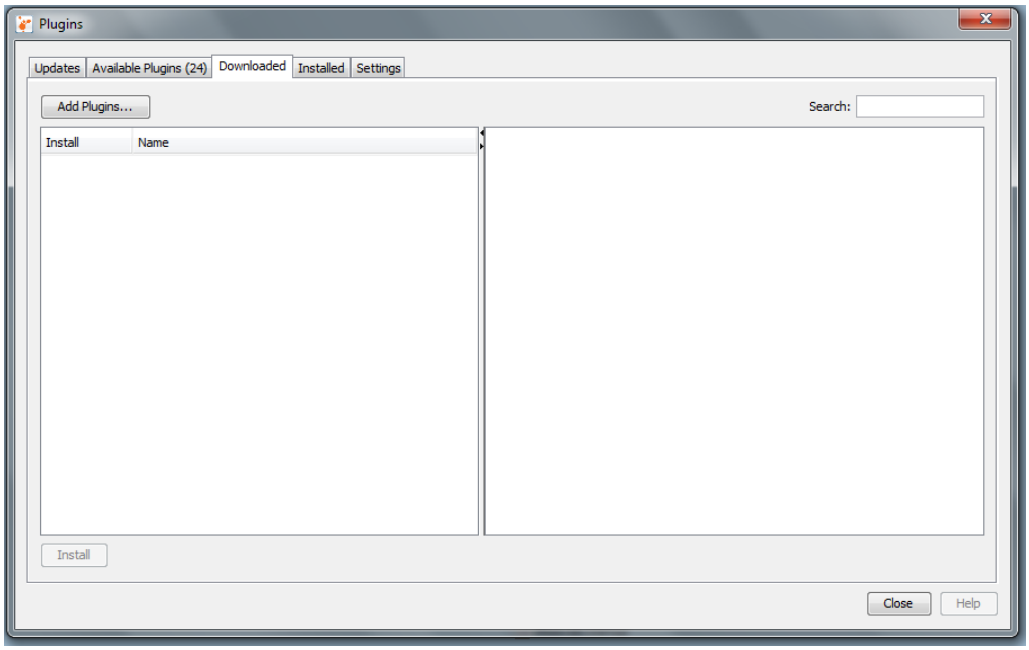

Figura 4-2: Tela de Instalação de *Plugins*

No caso, são dois arquivos Encapsulador.nbm e Plugin.nbm. O primeiro contém as bibliotecas utilizadas pelo segundo que contém o *plugin* de fato. Depois, é só confirmar a instalação selecionando os arquivos e clicando no botão "Install". Pronto, instalação concluída.

4.2. Configurando os Subdomínios

Como dito anteriormente, é necessário ter o *bundle SubAgent* ativo em cada *runtime* OSGi será administrado. Além disso, também é necessário a instalação e ativação do *bundle* JDMK, pois, esse importa classes utilizadas pelo *bundle SubAgent*.

No exemplo em questão, foi utlizado o Apache Felix[19]. Foram criados dois *runtimes* OSGi e instalados e ativados os *bundles* JDMK e SubAgent. Ao ativar o *bundle* SubAgent é gerada uma URL pelo conector do *bundle*. Essa URL deverá ser usada pela ferramenta para se conectar a esse domínio.

4.3. Domínio Master

O domínio *Master* não precisa de nenhuma configuração, ele funciona normamelnte junto com o *plugin*. Porém, para ser conectar aos demais domínios é preciso inserir a URL obtida na ativação do *bundle SubAgent* no campo *runtime* URL e clicar em "Connect". A partir daí, o domínio *Master* já está apto a administrar os subdomínios que

foram conectados a ele basta seleciona-lo na árvore de domínios. No exemplo, foram conectados os dois subdomínios Sub1 e Sub2.

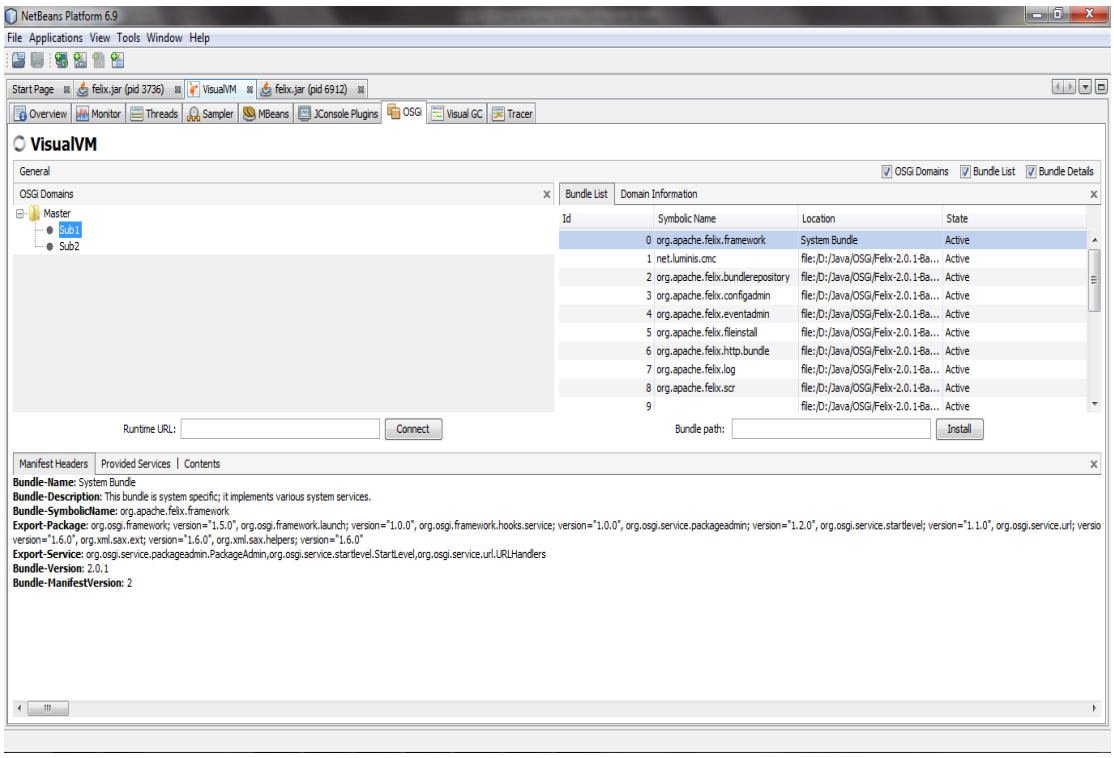

Figura 4-3: Subdomínios Configurados

4.4. Serviços

No exemplo em questão, o subdomínio Sub1 possui vinte e um *bundles* (informação que pode ser vista na aba *Domain Information*) e o subdomínio Sub2 possui vinte e dois *bundles* ambos em um *runtime* Felix Apache[19] como dito anteriormente.

Então, selecionamos os *bundles* DSAamfAgenciaBBC do Sub1 e podemos verificar que ele possui um tempo de resposta igual a dez.Além disso, pode-se observar outras propriedades com um pooling de intervalo igual mil, nome da agência BBC, a id do serviço que é setenta, entre outras.

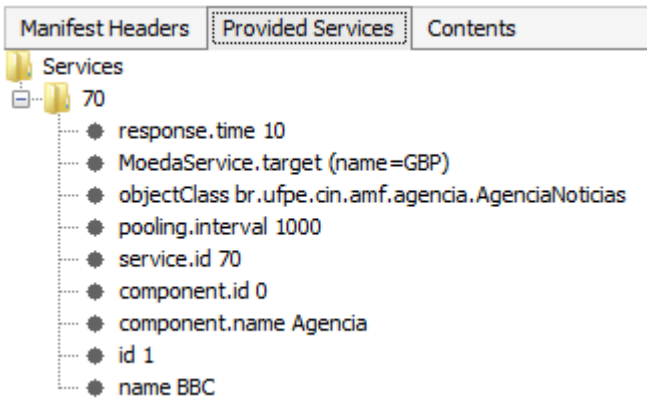

Figura 4-4: Serviços de DSAamfAgenciaBBC

Já no subdomínio Sub2 no *bundle* AgenciaBloomberg verificamos dois serviços fornecidos o cinquenta e nove e o cinquenta e oito.

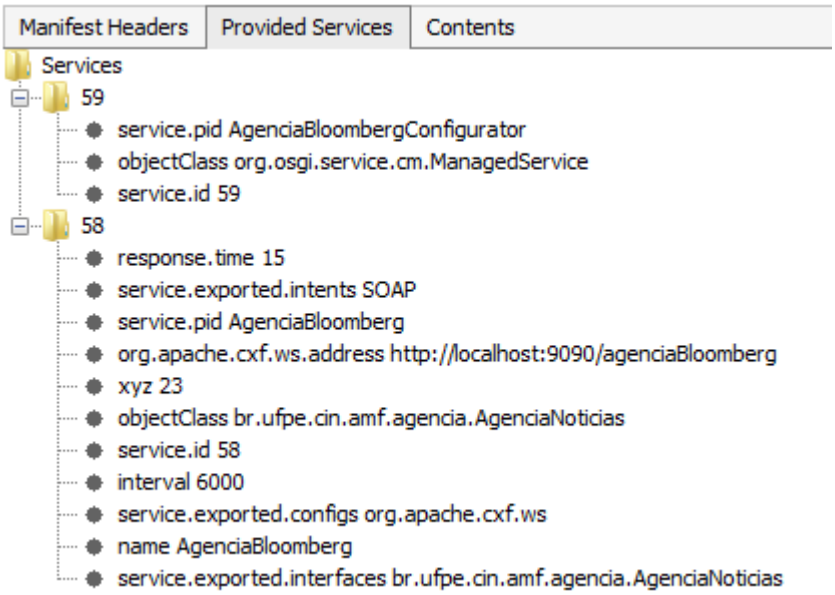

Figura 4-5: Serviços de AgenciaBloomberg

Além dos serviços fornecidos, também é possível ver informações do manifesto e do conteúdo do *bundle* através das abas Manifest Headers e Contents.

4.5. Considerações Finais

Este capítulo apresentou um exemplo de uso da ferramenta, nele foi explicado desde como fazer a instalaçaõ do *plugin* na VisualVM à como configurar o ambiente para o gerenciamento. O próximo capítulo apresentará as conclusões do trabalho e as possibilidades de trabalhos futuros

5. Conclusão

Neste trabalho apresentamos o processo de desenvolvimento de uma ferramenta para o gerenciamento de serviços e *bundles* na plataforma OSGi.

Para tal, um estudo foi realizado para enteder as tecnologias utilizadas durante o processo de construção da ferramenta. Assim, foram apresentadas as características e os conceitos fundamentais da especificação OSGi e da JMX.

Em seguida, foram explicadas as partes principais da implementação deste trabaho e as funcionalidades fornecidas pelo *plugin.* Depois, foi mostrado um exemplo de uso que serve com um guia para usuários da ferramenta.

Enfim, a uso da ferramenta possibilita ao usuário facilidades na tarefa de gerenciamento da plataforma OSGi através da interface. Além disso, ela também fornece a possibilidade de gerenciar mais de uma plataforma OSGi o que é uma grande benefício para administração de serviços distribuídos que é um dos principais usos do OSGi.

5.1. Trabalhos Futuros

Como melhorias futuras para o trabalho existe a possibilidade de adicionar novas funcionalidades ao *plugin.* Pode-se então, realizar um estudo para incluir novas funções para a parte de serviços como,por exemplo, alterar, remover, atualizar e incluir novos serviços.

Outra melhora poderia ser a exibição de gráficos relacionados a características dos serviços devido a sua dinamicidade. Poderiam ser criados gráficos para mostrar características dos serviços que mudam com o tempo, por exemplo, gráficos relacionados ao tempo de resposta do serviço.

Referências

[1] H. Cervantes and R.S. Hall, “Autonomous adaptation to dynamic availability using a service-oriented component model,” published in the proceedings of the International Conference on Software Engineering, 2004,pp 614-623.

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

[3] Bartlett Neil, OSGi in Pratice, Janeiro 2009.

[4] OSGi Alliance. http://www.osgi.org. [Online; acesso em 29-Junho-2010].

[5] OSGi Alliance. About the OSGi Service Paltaform, Revision 4.1, Technical Whitepaper. 2007.

[6] ZIP. http://en.wikipedia.org/wiki/ZIP_(file_format) [Online; acesso em 07-Novembro-2010].

[7] Java 2 Security Architecture Version 1.2, Sun Microsystems, Março de 2002.

[8] Java Dynamic Management Kit 5.1Tutorial. Sun Microsytems. Junho de 2004

[9] Perry, J. Steven. Java Management Extensions. Junho de 2002

[10] Juha Lindfors, Marc Fleury e o JBoss Group. JMX: Managing J2EE with Java Management Extensions, SAMS, 2002.

[11] Trail: Java Management Extensions (JMX). http://download.oracle.com/jav ase /tutorIal/jmx/index.html. [Online; acesso em 20-Novembro-2010].

[12] Java; http://java.com. [Online; acesso em 02-Dezembro-2010].

[13] WebNMS Agent Toolkit java Edition 6; http://www.webnms.com/javaagent/help/mp _agent/index.html#jmx/j_jmx_agent_archi.html. [Online; acesso em 29-Novembro-2010].

[14] VisualVM; http://visualvm.dev.java.net/. [Online; acesso em 20-Agosto-2010].

[15] Just An Ordinary Java Blog; http://ordinaryjava.blogspot.com/2009/06/visualvm-osgi-plugin.html. [Online; acesso em 02-Outubro-2010].

[16] NetBeans IDE; http://netbeans.org/. [Online; acesso em 23-Novembro-2010].

[17] Eclipse IDE; http://www.eclipse.org. [Online; acesso em 23-Novembro-2010].

[18] BndTools; http://www.aqute.biz/Code/Bnd [Online; acesso em 02-Dezembro-2010].

[19] Apache Felix; http://felix.apache.org/. [Online; acessado em 29-Agosto-2010].