Universidade Federal de Pernambuco – UFPE
Graduação em Ciência da Computação
Centro de Informática - CIn

Uma solução para mediação de serviços distribuídos no ambiente DSOA

Rafael Lima

Trabalho de Graduação

Recife, 13 de dezembro de 2011

Uma solução para mediação de serviços distribuídos no ambiente DSOA

Rafael Alberto Gomes Pereira Lima

Trabalho apresentado ao programa de 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.

Orientador: Nelson Souto Rosa

“E se me fugirem da boca as palavras, te direi com os olhos: ‘Te amo Mãe! ’”.
(Rafael Lima)

Agradecimentos

À minha família que sempre me apoiou, confiou e me deu forças. Em especial, a minha mãe, Eliane Gomes, que é a minha maior fonte de força e criatividade e me mostra sempre que eu posso encontrar tudo que preciso pra superar qualquer adversidade.

Ao professor Nelson Rosa, pela oportunidade de realizar o trabalho e pela orientação no decorrer do mesmo. E a Fábio Souza, que esteve presente em diversos momentos, pesquisando, me instruindo e lutando pelo meu sucesso como o seu próprio.

Aos incríveis amigos que ganhei no decorrer do curso e àqueles que eu já trazia de longa data e sei que sempre estarão ao meu lado, dividindo sorrisos e aflições. Ao meu incrível amigo Ênio que sempre me mostra que um vínculo fraternal é tão ou mais forte que um laço consanguíneo.

Aos colegas de trabalho de quem pude dispor da compreensão e apoio.

E também a todos aquele que contribuíram em minha caminha à universidade e durante a passagem pela mesma, meus sinceros agradecimentos.

Resumo

A escolha do paradigma de programação para o desenvolvimento de um sistema é feita focando usufruir ao máximo das facilidades geradas pelas características naturais de cada paradigma. Quando o objetivo é o desenvolvimento de aplicações de forma rápida e a baixo custo uma forte candidata é a COS (Computação Orientada a Serviços).

A COS propõe que os sistemas sejam construídos a partir de serviços comunicantes, os quais fornecem – e eventualmente consomem – um conjunto de operações. Isto é feito por meio de interfaces que representam sua abstração pública.

Entretanto, a diversidade de protocolos e padrões de comunicação, oriunda da existência de diferentes plataformas de suporte a COS, dificulta a interoperabilidade entre os serviços distribuídos utilizados em composições de serviços. Para tratar este problema, neste trabalho é desenvolvida uma solução para geração de interfaces abstratas de serviços. Usando-se estas interfaces, consumidores de serviços estarão aptos a acessá-los independentemente da tecnologia efetivamente usada pelo serviço, sendo possível inclusive, consumir serviços de interfaces distintas sem efetuar mudanças no código fonte. A solução proposta será desenvolvida no ambiente DSOA (*Dynamic Service Oriented Architecture*), utilizando a biblioteca D-OSGi (*Distributed Open Services Gateway initiative*) e a tecnologia JMX (*Java Management eXtension*).

Palavras-Chave: D-OSGi, DSOA, mapeamento.

Abstract

The choice of programming paradigm for systems development is done focusing makes the most of the facilities generated for the natural characteristics of each paradigm. When the aim is the quickly development and at low costs a good choice is SOC (Service Oriented Computing).

The SOC proposes that systems are built from communicating services, which provide – and eventually consume – a set of operations. That is done through interfaces which represent its public abstraction.

However, the diversity of standards and communication protocols, caused by the existence of different platforms to support SOC, makes harder the interoperability between distributed services used in services compositions. To resolve this problem, in this work is developed a solution for generation of abstract service interfaces. Using these interfaces, service consumers will be able to access them independently of the technology used by the service, being possible even, consume services with distinct interfaces without do any modification at source code. The proposed solution will be developed in the DSOA environment, using the D-OSGi (*Distributed Open Services Gateway initiative)* library and the JMX technology (*Java Management eXtension*).

Keywords: D-OSGi, DSOA, mapping.

Sumário

Agradecimentos ..... *iv*

Resumo ..... *v*

Abstract ..... *vi*

Sumário ..... *vii*

Índice de ilustrações ..... *ix*

1. Introdução *1*

1.1 Contextualização **1**

1.2 Problemas **2**

1.3 Objetivos **2**

1.4 Estrutura do Documento **3**

2. Conceitos Básicos *4*

2.1 Dynamic Service-Oriented Architecture **4**

2.1.1 Management Agent 5

2.1.2 Data Gathering Center 5

2.1.3 Data Processing Center 6

2.2 OSGi **6**

2.2.1 Bundles 7

2.2.2 Ambiente de Execução 7

2.2.3 Camada de Módulos 8

2.2.4 Camada de Segurança 8

2.2.5 Camada de Ciclo de Vida 8

2.2.6 Camada de Serviços 9

2.2.7 Provedor de Distribuição 9

2.3 JMX **11**

2.3.1 Nível de Instrumentação 11

2.3.2 Nível de Agentes 12

2.3.3 Nível de Serviços Distribuídos 13

3. Proposta *14*

3.1 Visão Geral **14**

3.2 Arquitetura 16

3.3 Delimitação do Escopo 16

3.4 Serviço de Publicação de Mapeamentos 17

3.4.1 Camada de Persistência 17

3.4.2 Camada de Negócios 18

3.4.3 Fachada 18

3.4.4 Camada de Distribuição 19

3.4.5 Mapas 19

3.5 Serviço de Checagem de Disponibilidade 20

3.6 Ferramenta Gráfica 20

4. Implementação 21

4.1 Ambiente de Desenvolvimento 21

4.2 Serviço de Publicação de Mapeamentos 21

4.3 Serviço de Checagem de Disponibilidade 21

4.4 Ferramenta Gráfica (Visual VM) 22

4.4.1 Aba de cadastramento de interfaces 23

4.4.2 Aba cadastramento de serviços concretos 23

4.4.3 Aba de criação de mapas 24

5. Exemplos 25

5.1 Instalação do *Plugin* no Visual VM 25

5.2 Cadastro de Interfaces Padrão 26

5.3 Cadastro de Serviços Concretos 27

5.4 Criação dos Mapas 27

6. Conclusões e Trabalhos Futuros 29

7. Referências Bibliográficas 30

Índice de ilustrações

Figura 1-1: Interações entre elementos da SOA 1
Figura 2-1: Esquematização da monitoração de qualidade [6].....4
Figura 2-2: As camadas OSGi [15].....7
Figura 2-3: Diagrama de estados de um *bundle* [15].....8
Figura 2-4: Funcionamento do D-OSGi 10
Figura 2-5: Arquitetura do JMX [18] 11
Figura 3-1: Arquitetura da solução 16
Figura 3-2: Camadas do Serviço de Publicação de Mapeamento 17
Figura 3-3: Modelo Entidade Relacionamento.....18
Figura 4-1: Estendendo a classe *DataSourceView*.....23
Figura 5-1: Instalação de *plugins* 25
Figura 5-2: Seleção dos arquivos necessários a instalação.....25
Figura 5-3: Visão inicial do Visual VM 26
Figura 5-4: Visão do *plugin*.....26
Figura 5-5: Cadastrando uma Interface 26
Figura 5-6: Cadastramento de um Serviço Concreto.....27
Figura 5-7: Aba de criação de mapas.....27
Figura 5-8: Construção de um Mapa 28

1. Introdução

Este capítulo apresentará o trabalho ao leitor, iniciando por uma contextualização e motivação do estudo, apresentando os problemas encontrados atualmente na área abordada. Em seguida serão mostrados os objetivos deste trabalho e a estrutura do restante do documento.

1.1 Contextualização

O grande número de dispositivos fixos e móveis tem motivado o desenvolvimento de aplicações distribuídas, heterogêneas e dinâmicas. A soma desta tendência ao interesse por um desenvolvimento rápido e a baixo custo tem resultado numa crescente adoção do paradigma SOC (*Service-Oriented Computing)* [1] [2] para o desenvolvimento de sistemas.

Na SOC, serviços são utilizados como estruturas básicas de programação e podem ser encarados como "peças" de software: independentes, reutilizáveis e bem definidas. Estes serviços são capazes de realizar desde operações simples, como uma calculadora, até processos de negócio complexos, como o fluxo de caixa de uma empresa. Todo serviço possui uma interface, funcionando como uma abstração de alto nível do mesmo, e uma ou mais implementações que definem diferentes maneiras de realizar as funcionalidades propostas pela interface.

Vários estudos foram realizados no intuito de projetar um ambiente que favorecesse o desenvolvimento de aplicações segundo a SOC, com isso foi criada a SOA (*Service-Oriented Architecture*), uma arquitetura que define um conjunto de papéis e interações para o desenvolvimento de aplicações orientadas a serviços [3] [4].

SOA propõe um modelo básico de interação composto de três participantes: o provedor do serviço, o consumidor do serviço e o registro de serviços. O provedor define uma descrição do serviço e a publica no registro. Esta descrição contém informações relacionadas à interface e localização do serviço. A localização contém informações que descrevem o acesso ao serviço permitindo, assim, que ele possa ser descoberto e utilizado de forma remota; já a interface refere-se à mesma mencionada anteriormente. O consumidor, com base na interface do serviço, faz uma busca no registro pela descrição de alguma implementação catalogada e, a partir desta descrição, acessa o provedor para usar o serviço. Por fim, o registro de serviços é a entidade responsável por armazenar informações sobre as instâncias existentes dos serviços. Este conjunto de interações pode ser visualizado na Figura 1-1.

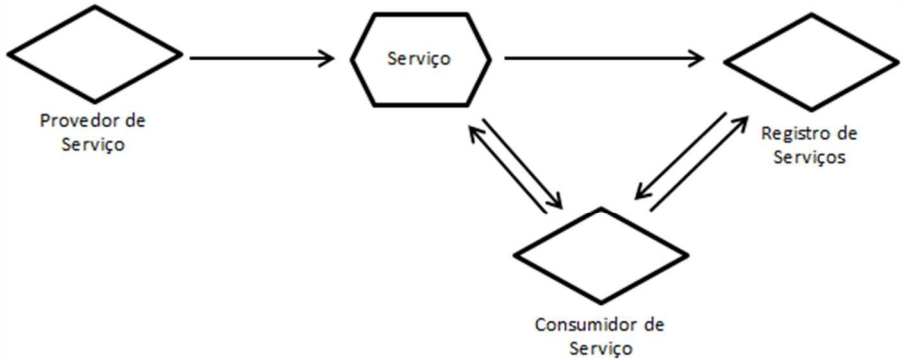

Figura 1-1: Interações entre elementos da SOA

A reusabilidade dos serviços permite a subdivisão de tarefas complexas em atividades menores que se correlacionam a fim de produzir um resultado global, esta prática é denominada composição de serviços. Diversos aspectos são importantes para o êxito na construção e execução de uma composição de serviços, dentre eles podemos destacar a monitoração de aspectos de qualidade da composição, neste contexto a qualidade é denominada QoS (*Quality of Service)* [5].

Neste cenário o DSOA (*Dynamic Service-Oriented Architecture*) modela um sistema para monitoramento contínuo de qualidade de forma dinâmica, baseada em eventos, não-intrusiva, *lightweight e* fazendo uso da arquitetura SOA.Com isso, o ambiente de execução dos serviços torna-se ciente de QoS, sem que eles precisem ser modificados [6].

Em decorrência da heterogeneidade de protocolos e padrões de comunicação existentes, grande parte do sucesso de uma plataforma de *middleware* é resultado de sua versatilidade e abrangência. Com o intuito de diminuir a curva de aprendizagem necessária para a construção de uma composição de serviços complexa, um *middleware* pode prover um mecanismo de mediação de invocações. Este mecanismo é responsável por traduzir as invocações, realizadas através de uma interface padrão, em formatos aceitos por diferentes instâncias de serviço, tornando transparente, ao desenvolvedor, a tecnologia utilizada em cada serviço da composição.

1.2 Problemas

A crescente prática de composições de serviços faz com que seja comum a existência de serviços equivalentes, isto é, serviços que proveem a mesma funcionalidade, mas cujas interfaces possuem pequenas diferenças (no nome da operação, ou na ordem ou tipo dos parâmetros). A existência destas interfaces redundantes ocorre muitas vezes em virtude da diversidade de padrões e protocolos de comunicação. Que pode acarretar, por exemplo, na criação de um serviço *Web Service* [7], um *RESTFul* [8] e um Corba [9] para a mesma demanda.

Em uma composição de serviços, a interface utilizada para cada serviço é normalmente escolhida durante o desenvolvimento. Por conta disso o serviço composto não tem ciência da existência de serviços similares como possíveis alternativas para o caso de um dos serviços membros se encontrar indisponível no momento da chamada. Desta forma, a indisponibilidade de um único serviço pode levar a não execução da composição até que todos os serviços utilizados nela estejam disponíveis.

Outro problema ocorre quando um serviço viola um SLA (*Service Level Agreement)* [10], o que pode implicar na substituição do mesmo e na necessidade de uma reformulação do código da composição para se adequar ao novo serviço. Interrompendo momentaneamente o funcionamento da mesma.

1.3 Objetivos

Para lidar com os problemas supracitados, podem ser utilizadas técnicas de mediação de serviços. Estas técnicas possibilitam que a invocação de uma única interface seja traduzida em chamadas a diferentes interfaces, com nome de operação e parâmetros diferentes, mas capazes de realizar uma operação equivalente à

desejada, possibilitando assim, que a interface utilizada na composição seja determinada em tempo de execução, minimizando o risco de a composição falhar por indisponibilidade de um serviço membro.

O presente trabalho tem como objetivo conceber e desenvolver uma solução de mediação de serviços para compor um dos módulos da infraestrutura da DSOA, o que garantirá a solução informações de qualidade para a tomada de decisão durante a mediação, garantindo uma melhora de qualidade global na composição e tornando a DSOA mais versátil, poderosa e agregando valor à plataforma como um todo.

A solução projetada proverá suporte para o cadastramento de interfaces de serviços, as quais servirão de referência durante o desenvolvimento da composição e também, a partir destas, serão possíveis as mediações, que ocorrerão em tempo de execução, para um serviço concreto em uma tecnologia de distribuição qualquer de forma transparente ao desenvolvedor.

Por fim, esta solução será concebida visando uma integração com ferramentas de gerenciamento já existentes tal como a VisualVM [11].Logo a arquitetura deve projetada para facilitar a integração com esta e outras ferramentas de gerenciamento.

1.4 Estrutura do Documento

Feitas as considerações iniciais sobre o campo de estudo e objetivo deste trabalho, o restante deste texto está organizado como segue:

- Capítulo 2 – Introdução de padrões, tecnologias e ferramentas necessários à compreensão da proposta.
- Capítulo 3 – Apresentação da proposta, justificativas e inovações.
- Capítulo 4 – Concretização do projeto, decisões de alto nível mostrando a viabilidade da proposta.
- Capítulo 5 – Exemplos de uso da ferramenta elaborada.
- Capítulo 6 – Considerações finais e sugestões de trabalhos futuros.

2. Conceitos Básicos

Esta sessão faz uma breve revisão sobre conceitos e tecnologias utilizadas no desenvolvimento da solução.

2.1 Dynamic Service-Oriented Architecture

A partir das descrições dos serviços utilizados em composições de serviços é possível extrair: uma interface de acesso e um conjunto de atributos não funcionais. De acordo com o tipo dos atributos, existem diferentes abordagens para analisa-los para então medir a qualidade da composição. Atributos que não variam com o tempo como preço, podem ser inferidos diretamente do contrato. No entanto, atributos como tempo de resposta, vazão, disponibilidade e taxa de falhas podem variar com o tempo e precisam de um mecanismo de monitoramento capaz de detectar violações no SLA [6].

Apesar de esta medição ser de grande importância para garantir a qualidade dos serviços, sua implementação é complexa, dada a grande quantidade de atributos que pode abranger. Além disto, sua execução pode impactar negativamente no desempenho da composição como um todo. O DSOA é uma especificação de sistema para monitoramento contínuo de qualidade: dinâmico, baseado em eventos, não-intrusivo e *lightweight* desenvolvido para resolver o problema supracitado [6].

No DSOA os parâmetros da monitoração (serviço alvo, atributo de qualidade medido, prazo de validade da monitoração) podem ser alterados sem que qualquer serviço da composição seja interrompido. Ao mesmo tempo em que os serviços não precisam de qualquer modificação em seu código para tornarem-se elegíveis ao monitoramento, pois tudo é feito de forma transparente tanto ao provedor, como ao consumidor do serviço.

Uma representação de alto nível do fluxo da monitoração pode ser vista na Figura 2-1.

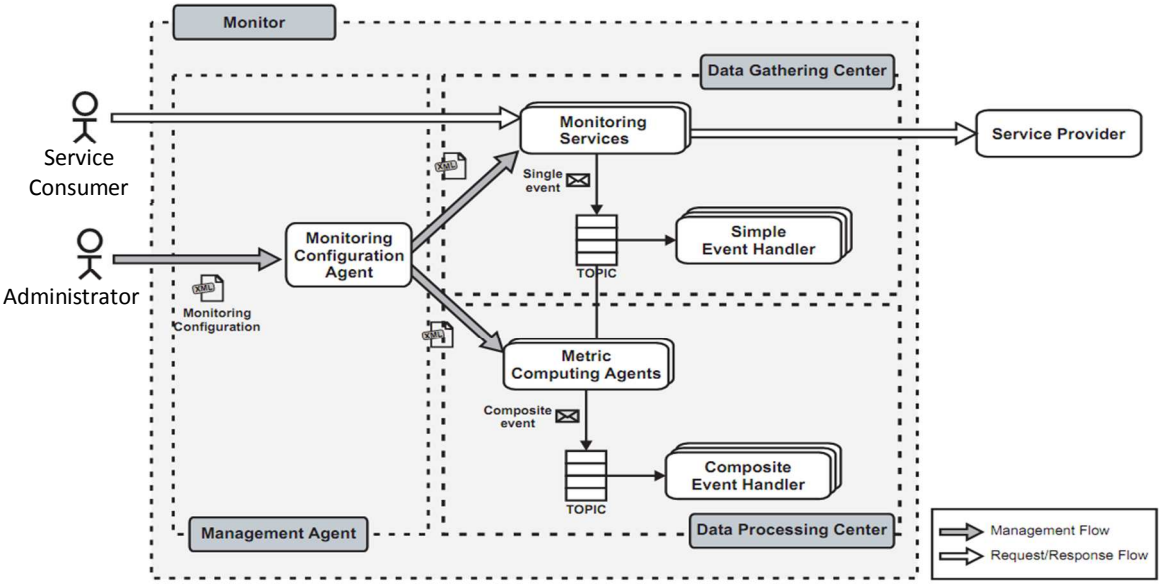

Figura 2-1: Esquematização da monitoração de qualidade [6]

2.1.1 *Management Agent*

O M*anagement Agent* é responsável pelo gerenciamento do ambiente como um todo, ele fornece interfaces de acesso local ou remoto que disponibilizam funções para a interrupção momentânea da monitoração, bem como criação de configurações de monitoração. Tais configurações podem ser vistas como perfis que determinam os parâmetros efetivos da monitoração: serviço alvo, atributo de qualidade a ser medido, intervalo das verificações, período de validade da monitoração e métricas a serem avaliadas (por exemplo, média, máximo e variância).

Para cada configuração de monitoração ativa, são selecionados serviços capazes de realizar a medição do atributo selecionado. Feito isto, o M*anagement Agent* notifica o *Data Processing Center* e o *Data Gathering Center* para iniciar a monitoração. As configurações que fazem referência a uma característica para a qual não existe um serviço de monitoração disponível são colocadas numa lista de espera, de forma que para cada novo serviço de monitoração registrado o M*anagement Agent* verifica esta lista para saber se o mesmo atende a alguma das configurações.

2.1.2 *Data Gathering Center*

O *Data Gathering Center* é o mecanismo responsável por coletar, continuamente, métricas de qualidade dos serviços. Ele faz uso dos padrões de projeto conhecidos como *Invocation Interceptor* e *Observer* [12] [13] tornando possível coletar as informações desejadas e empacotá-las em mensagens de evento que podem ser enviadas assincronamente para o *Data Processing Center*. Desta forma, desacoplam-se as responsabilidades de medir e analisar os atributos de qualidade, reduzindo o impacto da monitoração como um todo no desempenho do serviço, pois os cálculos podem ser realizados em paralelo e remotamente.

Para ter acesso em tempo de execução à cadeia de interceptadores, tornando possível montá-la dinamicamente, o *Data Gathering Center* faz uso de um encadeador customizado que é inserido estaticamente no topo da cadeia de interceptadores de cada serviço, ou seja, existe uma modificação estrutural no *middleware* original para contemplar esta situação. No entanto, esta modificação objetiva tornar as demais fases do processo transparentes ao desenvolvedor. Com isto, o encadeador torna-se responsável por adicionar os interceptadores, fornecidos pelos respectivos serviços de monitoração. Serviços estes, selecionados a partir das configurações de monitoramento, que serão descobertos apenas em tempo de execução e possibilitarão a medição dos atributos de qualidade para os quais foram concebidos.

Além do encadeador, o DSOA faz uso de outro interceptador inserido estaticamente na pilha de cada serviço, o publicador, responsável por repassar as informações coletadas por cada serviço de monitoração para tópicos onde estas poderão ser acessadas e interpretadas assincronamente seguindo o padrão *Observer.*

2.1.2.1 Serviços de Monitoração

Um serviço de monitoração é um serviço comum, descrito através de uma interface, que é desenvolvido com a finalidade de produzir um interceptador para coletar informação a cerca de um atributo de qualidade específico (por exemplo, tempo de resposta). Sua modelagem torna o ambiente DSOA extensível visto que se

não foi previsto a monitoração de um atributo de qualidade, esta pode ser facilmente adicionada apenas desenvolvendo-se um novo serviço capaz de medir este atributo.

2.1.2.2 Manipuladores de Evento

Um manipulador de eventos é um serviço simples que lê em um tópico e dá alguma finalidade a informação ali contida, como persistir em um banco de dados, por exemplo. O DSOA possui uma implementação de manipulador padrão que encaminha as informações dos tópicos para o *Data Processing Center.*

2.1.3 *Data Processing Center*

Para analisar os atributos de qualidade e computar métricas estatísticas a partir dos dados coletados dos serviços, uma abordagem intuitiva é armazenar tais dados em um banco de dados e processá-los a posteriori. Entretanto, em virtude do volume de eventos gerados contendo informações de qualidade, essa abordagem geraria um grande número de operações de entrada/saída e uma grande área de armazenamento.

Com o intuito de viabilizar um processamento em tempo real das informações, permitindo que violações de contrato sejam detectadas rapidamente e minimizar o impacto que a abordagem descrita anteriormente agregaria, o DSOA faz uso de um *Event Stream Processing* (ESP) [14] para processar continuamente o fluxo de eventos gerado pelo *Data Gathering Center.*

As métricas utilizadas no processamento dos dados são determinadas dinamicamente e ficam armazenadas nas configurações de monitoramento, como visto anteriormente, isto é feito para tornar este sistema dinamicamente extensível permitindo que novas métricas sejam adicionadas sem impactar na execução dos serviços. As métricas são computadas através de *queries* enviadas ao ESP por meio de *Metric Computing Agents* que são serviços, descobertos em tempo de execução, registrados para gerar uma *query* referente a uma métrica específica.

Por fim, os dados processados são empacotados em *Composite Events* e enviados a um tópico onde assinantes podem interpretar as informações e executar operações cabíveis, desde o envio de um e-mail notificando a violação de um SLA até mesmo uma reconfiguração complexa do ambiente da composição.

2.2 OSGi

OSGi (*Open Services Gateway initiative)* [15] é uma especificação aberta de uma plataforma de *middleware, lightweight,* desenvolvida pela OSGi Alliance. Um *container* OSGi (e.g., *Equinox* e *Felix*) provê um ambiente seguro e gerenciável para o desenvolvimento de aplicações dinâmicas e extensíveis em Java, permitindo que estas sejam instaladas, iniciadas, paradas, atualizadas e desinstaladas remotamente e facilitando a subdivisão da aplicação em módulos cujas dependências vão ser administradas pelo próprio mecanismo do *container*. Uma visão geral da arquitetura OSGi pode ser encontrada na Figura 2-2.

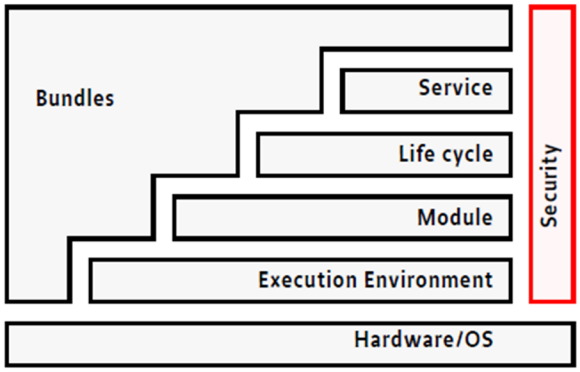

Figura 2-2: As camadas OSGi [15]

2.2.1 Bundles

Num ambiente OSGi as aplicações são empacotadas em *bundles.* Um *bundle* é um arquivo que contem classes Java e outros recursos que juntos podem prover funcionalidades para o usuário final. Estes arquivos são construídos da mesma maneira que arquivos JAR, entretanto em seu manifesto há um conjunto de informações permitindo ao *container* identificá-lo como um *bundle* OSGi válido e fornecendo a este os dados referentes a: resolução de dependências, componentes públicos e privados, bem como o programa principal do pacote (o *bundle activator*). O programa 2-1 mostra um exemplo de manifesto, nele vemos alguns dos meta-dados utilizados para caracterizar um *bundle*: linhas 1 e 2 a nomeação do *bundle*, na linha 6 a definição da classe ativadora, nas linhas 7 e 8 a explicitação dos pacotes exportados e importados pelo *bundle,* respectivamente.

```
1| Bundle-Name: Example Bundle
2| Bundle-SymbolicName: org.helloworld
3| Bundle-Description: An Sample bundle
4| Bundle-ManifestVersion: 2
5| Bundle-Version: 1.0.0
6| Bundle-Activator: org.Activator
7| Export-Package: org.helloworld;version="1.0.0"
8| Import-Package: org.osgi.framework;version="1.3.0"
```

Programa 2-1: O arquivo MANIFEST.MF

O OSGi permite, em tempo de execução, através do registro de serviços, que *bundles* registrem serviços, recebam notificações sobre o estado de serviços e procurem por serviços, fazendo uso inclusive, de filtros sobre meta-dados dos serviços cadastrados.

2.2.2 Ambiente de Execução

O ambiente de execução OSGi define como *containers* podem informar aos *bundles* sobre os ambientes de execução Java disponíveis (por exemplo, *Java to Standard Edition* 1.5, *Android*, *Java to Standard Edition* 1.7) com o propósito de definir variações no espaço de nomes *java.**. Um ambiente de execução pode ainda incluir pacotes em outros *namespaces* e deve definir um nome para identificá-lo de modo

que um *bundle* possa requisitar ao *container* um ambiente de execução específico para ser resolvido.

2.2.3 Camada de Módulos

A máquina virtual Java provê um suporte limitado de encapsulamento, entrega e validação para as aplicações ou componentes. Em decorrência disto, o OSGi implementa seu próprio mecanismo de modularização. Esta camada contem c*lass loaders* especializados para realizar as atividades anteriormente mencionadas. Com isto, é possível, por exemplo, existir uma classe pública dentro de um *bundle* OSGi que é invisível para qualquer *bundle* que o importa.

2.2.4 Camada de Segurança

A camada de segurança do OSGi é baseada na *Java 2 Security Architecture* [15] e é responsável por fornecer infraestrutura para a gerenciamento e implantação de aplicações executadas no ambiente OSGi. Além disso, é uma camada opcional do *container* e fica subjacente à camada de serviços.

2.2.5 Camada de Ciclo de Vida

A camada de ciclo de vida implementa as operações necessárias para lidar com todas as possíveis mudanças de estados de um *bundle.* Figura 2-3 mostra as mudanças de estados permitidas a um *bundle.*

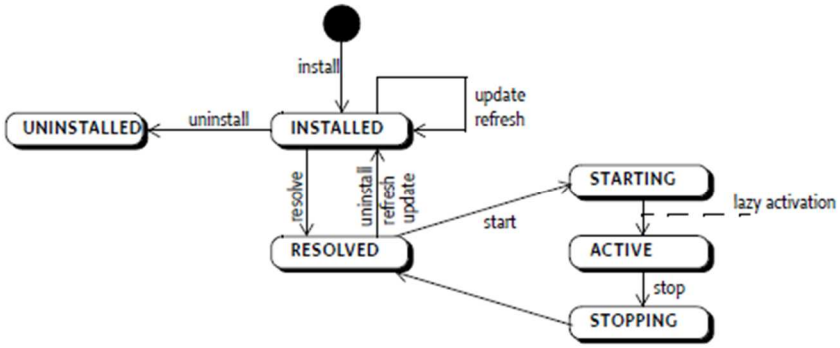

Figura 2-3: Diagrama de estados de um *bundle* [15]

- **INSTALLED** – O *bundle* está instalado. Quando um *bundle* é instalado ele é persistido pelo *runtime* até ser explicitamente removido. Cada mudança sobre o estado do *bundle* é refletida no seu registro persistido permitindo que seja restaurado quando o *runtime* reinicia.
- **RESOLVED** – Todas as dependências explicitadas no manifesto do *bundle* podem ser resolvidas pelo *runtime,* o *bundle* está pronto para inicializar ou parar.
- **STARTING** – O *bundle* está sendo iniciado, o método *start* da classe ativadora, indicada no manifesto, começou a ser executado, mas ainda não foi finalizado. Um *bundle* pode utilizar um mecanismo de *lazy*

activation prendendo-o ao estado de STARTING até um evento de ativação ser invocado.

- **ACTIVE** – O método de ativação do *bundle* foi concluído com sucesso e este se encontra em execução.
- **STOPPING** – O *bundle* está sendo parado, uma execução do método *stop* na classe ativadora foi iniciada, mas ainda não foi completada.
- **UNINSTALLED** – O *bundle* foi removido, e não pode mais ser movido para outro estado.

2.2.6 Camada de Serviços

A camada de serviços OSGi modela um sistema de publicação, busca e vinculação de serviços de forma dinâmica, colaborativa, segura, reflexiva, com controle de versão e visando aspectos de escalabilidade, pois o OSGi foi concebido para ser executado em uma grande variedade de hardwares com diferentes restrições [15]. O uso padrão de OSGi provê um repositório local para publicação e busca de serviços, estando ambos (provedor e consumidor) no mesmo *runtime*.

Um serviço é um objeto Java comum que implementa uma ou mais interfaces, definidas ou não pelo usuário, empacotadas em *bundles* e publicadas no registro de serviços. Além dos serviços dinâmicos desenvolvidos pelo usuário, o *container* trabalha com alguns serviços especiais denominados *hooks,* estes fazem parte do mecanismo de extensão do OSGi e são responsáveis por implementar o modelo de serviços descrito anteriormente.

Uma busca por um serviço é feita com base na interface desejada, não sendo possível determinar qual instância atualmente ativa será retornada. No entanto, é possível adicionar meta-dados no formado: chave/valor para restringir a busca e permitir que o resultado esteja mais específico.

2.2.7 Provedor de Distribuição

Como foi dito na seção 2.2.6, originalmente o OSGi contempla apenas um registro local de serviços, mas sua especificação prevê a possibilidade de extensão através de um provedor de distribuição.

Um provedor de distribuição se aproveita do baixo acoplamento entre *bundles* para exportar um serviço através da criação de um *endpoint*. Ele pode também criar um *proxy* para acessar um *endpoint* previamente criado fora do *runtime*. Um *runtime* pode prover diversos mecanismos de distribuição simultaneamente de forma independente, importando ou exportando serviços.

Visando implementar um provedor de serviços OSGi conforme descrito em [15], foi criado o D-OSGi *(Distributed Open Services)* [16], um conjunto de serviços, empacotados em *bundles* e construídos para atender a especificação de um provedor de distribuição descrita em [6]. O D-OSGi age recebendo as notificações emitidas a cada novo serviço registrado e verificando os meta-dados em busca de parâmetros para configurar sua atuação. Estes parâmetros se referem à localização do serviço, ao mecanismo de exportação, a interface exportada e outros (alguns dos meta-dados aceitos pelo D-OSGi podem ser vistos na Tabela 2-1). De posse destas informações, o D-OSGi adiciona um registro em um servidor *Zookeeper* [17], independentemente do

registro local OSGi, e cria um *endpoint* para atender as invocações feitas ao serviço. Um processo semelhante ocorre quando um serviço tenta consumir um serviço remoto. Neste caso, o D-OSGi faz uma busca pelo mesmo através do *Zookeeper* e caso encontre-o devolve ao consumidor um *proxy* para que este acesse o serviço. Uma visão simplificada do funcionamento do D-OSGi pode ser vista na Figura 2-4.

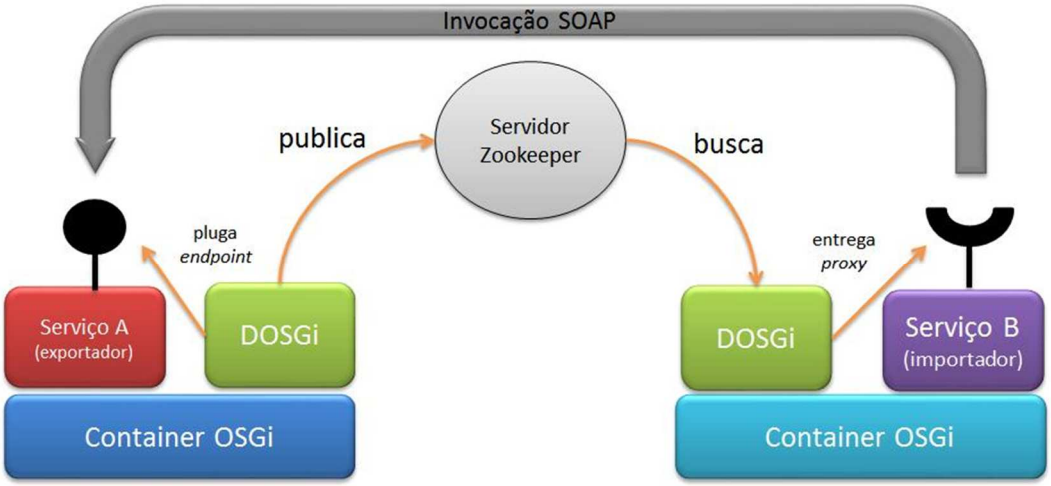

Figura 2-4: Funcionamento do D-OSGi

É importante notar que a atuação do D-OSGi é complementar a do OSGi, isto é, caso um serviço tente importar um *Web Service* [7] e este se encontre hospedado no mesmo *runtime* do primeiro, o *runtime* irá entregar a referência direta ao serviço e não o *proxy* como poderia ser esperado. Isto ocorre porque o D-OSGi começa a agir a partir do momento que o registro local não encontra um determinado serviço.

Tabela 2-1: Meta-dados do D-OSGi

Propriedade	Descrição	Exemplo
service.exported.interfaces	Define a interface que será exportada remotamente.	`org.examplo.Cotacao`
service.exported.configs	Determina como é feita a configuração do serviço exportado, os valores possíveis são: • *org.apache.cxf.ws* • *wsdl_configuration*	`org.apache.cxf.ws`
org.apache.cxf.ws.address	Para o caso do serviçoser exportado como *Web Service* esta propriedade indica o endereço do *endpoint.*	http://localhost:99/WSsample
org.apache.cxf.rs.address	Para o caso do serviçoser exportado como *RESTful JAXRS*esta propriedade indica o endereço do *endpoint.*	http://localhost:98/RSsample

2.3 JMX

O JMX (*Java Management Extensions*) [18] define uma arquitetura, padrões de projeto e APIs para aplicações, gerenciadores e monitores de rede, através da linguagem de programação Java. O JMX fornece ao desenvolvedor as ferramentas necessárias para criar agentes Java inteligentes, implementar gerenciadores para sistemas de middleware distribuídos e integrar de forma coesa, dinâmica, simples e padronizada estas soluções em sistemas de gerenciamento e monitoramento existentes. Ela foi adicionada a partir da plataforma *Java 2StandardEdition*(J2SE) 5.0.

Com JMX, cada recurso é instrumentado por um ou mais *MBeans* (*Managed 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 disponibilizá-los para os aplicativos dos recursos remotos definidos pela especificação da JMX. Para que os recursos disponibilizados sejam acessados remotamente, o JMX define conectores padrões, estes usam protocolos diferentes, mas proveem a mesma interface de gerenciamento.

A Figura 2-5mostra as interações entre os principais componentes da arquitetura do JMX, estes estão dispostos em três níveis que serão mais bem descritos a seguir.

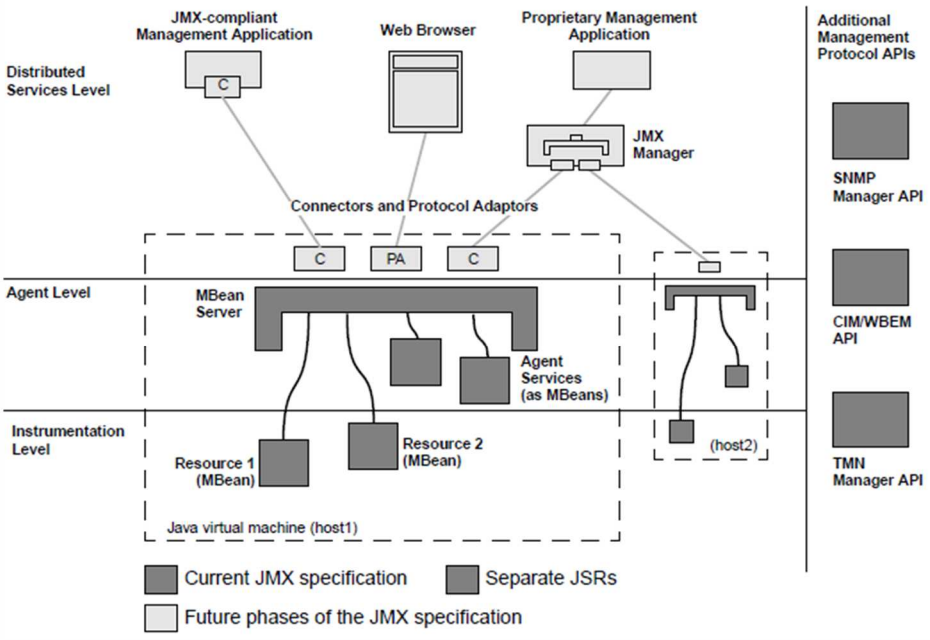

Figura 2-5: Arquitetura do JMX [18]

2.3.1 Nível de Instrumentação

O nível de instrumentação do JMX especifica como os recursos devem ser estruturados para que possam ser gerenciados. Esses recursos podem ser aplicações, módulos ou dispositivos e não precisam ser codificados em Java, desde que tenham ao menos um *wrapper* [13] em Java.

Neste nível é definido também um mecanismo de notificações que permite a qualquer um se cadastrar para receber eventos emitidos por um recurso, estes eventos são enviados em *broadcast* e cabe, no momento da subscrição, ao ouvinte determinar os filtros seletores dos eventos que deseja receber.

Os recursos desenvolvidos segundo esta especificação são denominados *managed beans* ou *MBeans* [18] e são padronizados para garantir que o acesso aos mesmos se dará da forma adotada. O público alvo desta etapa da especificação são programadores cujo objetivo é garantir capacidade de gerenciamento para suas aplicações, gerenciamento este, que pode ser dado numa maior ou menor granularidade dependendo do escopo dos objetos definidos pelos *MBeans*.

Um *MBean* é uma classe que implementa sua própria interface (ou a *DynamicMBean,* tornando-o um *MBean* dinâmico) e segue algumas regras básicas:

1. Não pode ser declarado como *abstract*
2. Deve implementar alguma interface própria, ou a *DynamicMBean.*
3. O estado do recurso deve ser completamente descrito através de métodos de *get* e *set.*
4. O *MBean* deve ter ao menos um construtor público.

De acordo com sua construção os *MBeans* podem ser classificados como S*tandard MBean, DynamicMBean, Open MBean ou ModelMBean.* Sendo usados principalmente os *Standard e DynamicMBeans.*

2.3.1.1 *Standard MBean*

Este pode ser considerado o *MBean* mais simples, implementa uma interface qualquer, definida pelo usuário ou não, composta por propriedades e operações. As propriedades referem-se ao estado do *MBean* e devem ser implementadas através de métodos de *get* e *set*. As operações representam as tarefas executáveis pelo *MBean* e são implementadas através de métodos quaisquer.

2.3.1.2 *DynamicMBean*

Esta tipo de *MBean* é mais versátil sendo capaz de definir suas propriedades e operações em tempo de execução. Deve obrigatoriamente implementar a interface *DynamicMBean* que contem o método *getMBeanInfo* responsável por informar as propriedades e operações do *MBean* em questão através de um objeto do tipo *MBeanInfo.* Este objeto contem informações como: nome da classe, lista de construtores, lista de propriedades e lista de operações.

2.3.2 Nível de Agentes

Esta camada providencia agentes capazes de disponibilizar os recursos para monitoramento remoto. Os agentes ficam acima dos recursos da camada de instrumentação e fazem uso deles, mas não precisam estar necessariamente hospedados na mesma máquina. É o que ocorre, por exemplo, quando o recurso gerenciado oferece um ambiente proprietário não Java, neste caso esta camada pode ser instanciada em um elemento concentrador da arquitetura.

2.3.2.1 MBeanServer

O *MBeanServer* é um registro de *MBeans* e todas as operações de acesso a estes passam por ele. Um *MBean* pode ser registrado através de um agente ou através de

outro *MBean*, no momento do registro é utilizado um objeto do tipo *ObjectName* servindo como um identificador único para cada *MBean*. Após o registro, o *MBeanServer* é quem gerencia as requisições por *MBeans,* consulta de propriedades e invocações destes, bem como identificação da interface da qual fazem parte e ainda cadastros para recebimento de notificações emitidas por um *MBean*.

2.3.2.2 Agente

Um agente *JMX* é uma entidade composta de um *MBeanServer*, um conjunto de recursos na forma de *MBeans,* ao menos um serviço para utilizar os recursos e ao menos um *protocole adaptor* ou *conector.* O padrão definido para os agentes torna-os aptos a manipular quaisquer recursos desenvolvidos por esta especificação, isto é, o agente não precisa ter ciência de qual recurso está manipulando.

2.3.3 Nível de Serviços Distribuídos

Este nível fornece interfaces e componentes capazes de fazer uso dos agentes ou hierarquizar os mesmos. Através destes componentes é possível interagir de forma transparente com um agente e seus recursos, encaminhar uma informação de gerenciamento de um nível mais alto da plataforma para diversos agentes, centralizar e analisar informações anteriormente distribuídas em diversos agentes e implementar a segurança.

3. Proposta

Este capítulo apresenta o estudo central do trabalho, iniciando por uma visão global do problema onde o mesmo se insere, mostrando sua inovação e delimitando o foco de atuação da solução aqui detalhada.

3.1 Visão Geral

Conforme descrito nas Seções 1.2 e 1.3 deste documento, este trabalho se insere no contexto de composição de serviço focando na área de monitoramento de qualidade. Em particular, este estudo concentra-se no estabelecimento de estratégias e técnicas que possam coletar informações a partir de eventos produzidos pelos serviços que estão sendo disponibilizados, tornando-os melhor administrados. Este monitoramento permite, por exemplo, a substituição de serviços que estejam apresentando pontos de falha, a identificação de serviços que violaram o SLA, a geração de relatórios estatísticos sobre QoS [19].

Como visto na Seção 2.1, o DSOA modela uma infraestrutura de suporte à composição de serviços, tornando esta ciente de qualidade sem impactar em nenhuma etapa do processo de desenvolvimento do software. Visto isso, pretende-se integrar ao DSOA um mecanismo de seleção de serviços baseado em qualidade capaz de intermediar a invocação de diferentes serviços em diferentes interfaces e/ou tecnologias de distribuição. Desta forma, a solução proposta provê ao consumidor o serviço que se apresenta mais bem cotado para aquela operação, tornando a composição mais dinâmica e permitindo que violações de SLA sejam contornadas mais facilmente.

A princípio será criado um módulo de mapeamento responsável por fornecer uma interface para a criação e persistência dos mapas. Estes mapas armazenam a lógica da intermediação, isto é, as informações que permitirão ao *proxy* direcionar a invocação de uma operação em uma interface para diferentes operações em diferentes serviços, fazendo as conversões necessárias. Estes mapas terão como referência uma interface padrão, descrita em Java, construída pelos gerentes do sistema, e serão vinculados a serviços reais com operações logicamente equivalentes, isto é operações que resultam em valores iguais quando acionadas com os mesmos parâmetros. Para isso, o gerente do sistema deverá definir para cada método da interface padrão a operação equivalente no serviço concreto, fazendo os devidos ajustes quanto aos parâmetros destas chamadas.

Fazendo uso das vantagens de um ambiente de distribuição de serviços, este módulo de mapeamento será fornecido como um serviço remoto, doravante denominado Serviço de Publicação de Mapeamentos, escrito em Java e distribuído através do D-OSGi na forma de um *Web Service*. Para aproveitar a existência de ferramentas já consolidadas na área de administração e gerenciamento, a interface de acesso ao serviço de mapeamento será construída segundo a especificação JMX (ver Seção 2.3), o que tornará a integração com a VisualVM simples [13].

Uma vez construídos, os mapas serão publicados junto à descrição do serviço em um servidor de nomes, sendo informada como interface de acesso ao serviço a interface padrão escrita em Java e adicionado aos meta-dados da descrição um conjunto de informações tornando possível a mediação. Será então, necessário desenvolver uma entidade capaz de interpretar estes meta-dados e controlar, em

tempo de execução, as invocações realizadas pelo consumidor direcionando-as para o serviço concreto. Esta entidade é o *Mediator Proxy* desenvolvido como uma especialização do *proxy* padrão fornecido pelo D-OSGi. Caberá ao *Mediator Proxy* viabilizar a comunicação com os diversos mecanismos de distribuição existentes seja *Web Service, JAXWS* ou outros.

Para possibilitar que o *Mediator Proxy* encaminhe sempre a invocação para o serviço mais bem cotado dentro de um mapa, será criado um motor de seleção implementando algoritmos para a tomada desta decisão. Estes algoritmos farão uso tanto das informações contidas no mapa, quanto das informações coletadas durante o monitoramento e analisadas pelo ESP [14].

Entretanto, a natureza remota dos serviços cria a necessidade de uma checagem contínua sobre sua disponibilidade assegurando que o *Mediator Proxy* não gere um erro tentando acessar um serviço que foi momentaneamente interrompido. Para isso, propomos a criação de um Serviço de Checagem de Disponibilidade que fará verificações regulares em cada serviço concreto com mapeamento ativo, atualizando o estado do mesmo no mapeamento e evitando o problema mencionado.

A solução proposta consiste dos seguintes elementos:

Tabela 3-1: Componentes da Solução

Componente	Descrição	Origem
OSGi	*Container* de execução da solução.	Apache Felix
D-OSGi	Provedor de distribuição para o OSGi.	Apache CXF D-OSGi
Agente de Gerenciamento	Componente responsável pela publicação das *monitoring configuration.*	DSOA
Centro de Coleta de Dados	Componente responsável pela coleta de dados de qualidade dos serviços monitorados.	DSOA
Centro de Processamento de Dados	Componente responsável pela análise dos dados de qualidade coletados.	DSOA
Motor de seleção de serviços ótimos	Componente que fornece a informação sobre os serviços monitorados mais bem cotados.	A ser desenvolvido
Serviço de Publicação de Mapeamentos	Serviço para publicação dos mapas	Desenvolvido neste projeto
Mediador Proxy	*Proxy* capaz de mediar as invocações das operações.	A ser desenvolvido
Serviço de Checagem de Disponibilidade	Serviço para verificação do estado de disponibilidade dos serviços remotos.	Desenvolvido neste projeto
Serviço de Nomes	Armazena as informações sobre as instâncias dos serviços.	Apache Zookeeper
Tela de criação das configurações de monitoramento	Interface de acesso ao Agente de Gerenciamento	DSOA
Tela para criação de mapeamentos para a mediação	Interface de acesso ao Serviço de Publicação de Mapeamentos	Desenvolvido neste projeto

3.2 Arquitetura

A arquitetura da solução proposta é baseada em conceitos SOA e estende o DSOA incorporando os elementos necessários para viabilizar o que foi sugerido.

Em nossa solução o *Zookeeper* fará o trabalho de um servidor de nomes, o Felix será utilizado como *Framework OSGi* e o Apache CXF D-OSGi será o provedor de distribuição acoplado ao Felix. Uma visão geral da organização da solução pode ser encontrada na Figura 3-1. A descrição e origem dos elementos da Figura 3-1 podem ser observadas na Tabela 3-1

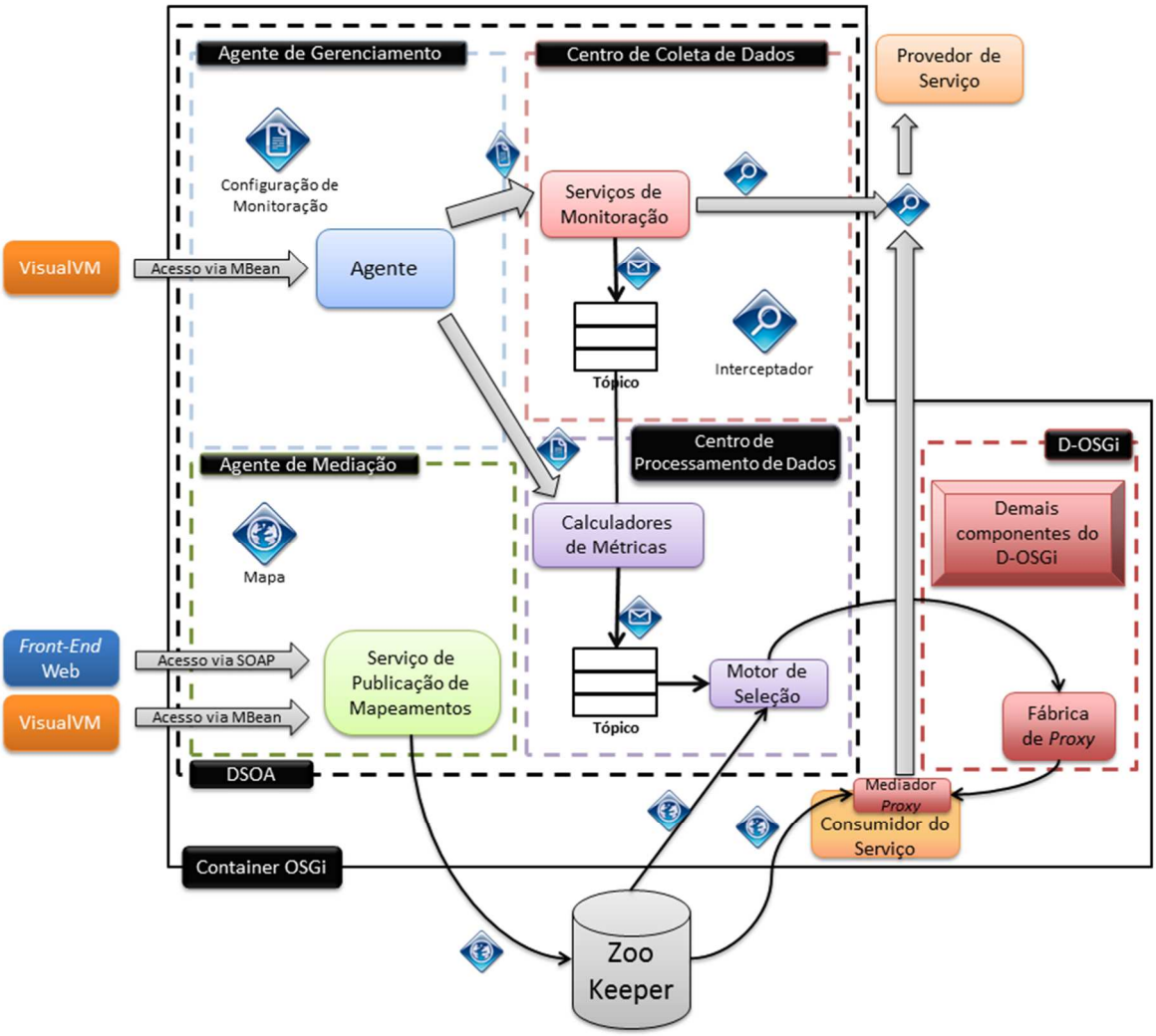

Figura 3-1: Arquitetura da solução

3.3 Delimitação do Escopo

Com base nas restrições temporais as quais está sujeito este trabalho, foi feita uma delimitação do escopo total apresentado na Seção 3.1 de modo a viabilizar sua concretização no prazo e com o padrão de qualidade estipulado.

Por conta disto, este trabalho concentrou-se na modelagem e implementação do Serviço de Publicação de Mapeamentos, bem como a interface gráfica apta a utilizá-lo, a qual possui como pré-requisito integração com a ferramenta VisualVM.

Adicionalmente, foi desenvolvido o Serviço de Checagem de Disponibilidade, que atua de forma auxiliar ao mapeamento.

As seções seguintes deste documento mostrarão as etapas de realização do projeto e farão referências ao mesmo, considerando o escopo agora definido.

3.4 Serviço de Publicação de Mapeamentos

Desenvolvido segundo o modelo de serviços do OSGi (ver Seção 2.2), o serviço de publicação tem como principal objetivo fornecer uma interface para criação dos mapas utilizados na mediação.

Sua arquitetura foi dividida em quatro camadas, seguindo conceitos de boas práticas de programação, delegando a cada camada uma tarefa mais específica. A Figura 3-2 mostra como foi feita esta divisão.

Figura 3-2: Camadas do Serviço de Publicação de Mapeamento

3.4.1 Camada de Persistência

Esta camada tem como principal responsabilidade abstrair das camadas superiores a comunicação com o banco de dados. Sendo também, encarregada de lidar com a conversão de tipos entre o Serviço de Publicação de Mapeamentos e o banco, lidar com as heterogeneidades oriundas dos diferentes bancos de dados, fornecer um mecanismo simples para a execução de consultas ao banco e gerenciar o ciclo de vida das entidades persistentes.

3.4.1.1 Modelo Entidade Relacionamento

Para representar fidedignamente o nosso domínio, foi efetuado um estudo que findou com a modelagem de um diagrama entidade relacionamento, a partir do qual foram construídas as tabelas para povoar o banco de dados. Este diagrama pode ser visto na Figura 3-3.

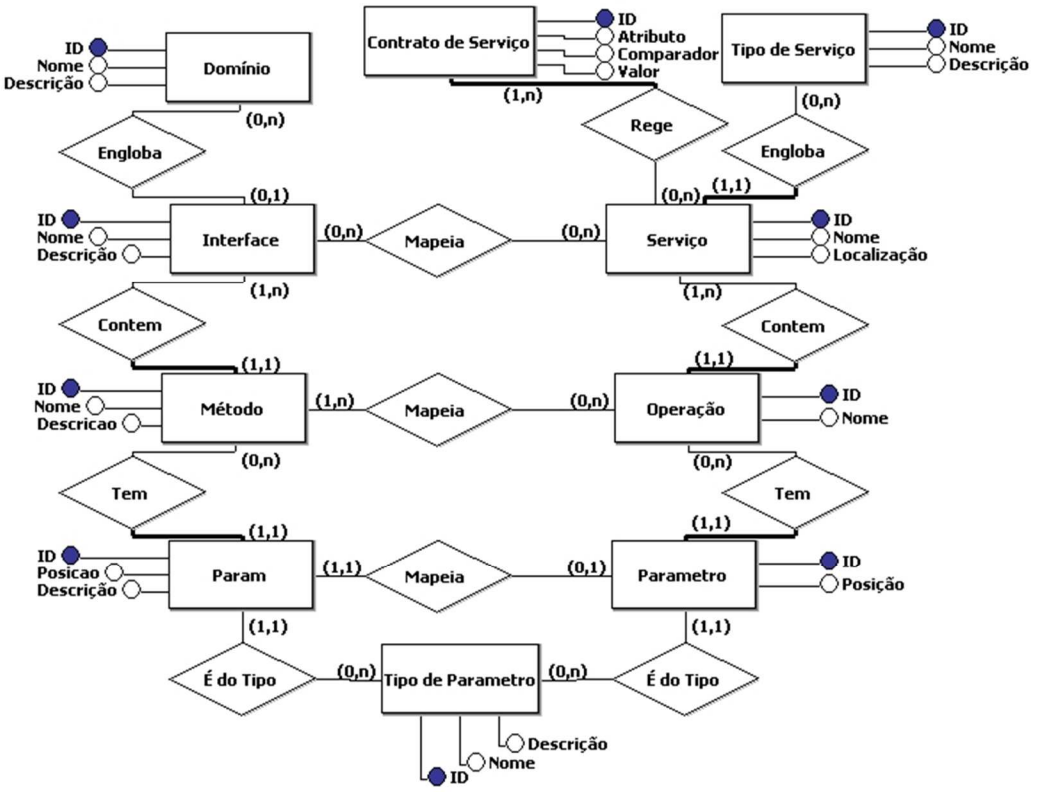

Figura 3-3: Modelo Entidade Relacionamento

A modelagem foi feita visando garantir ao máximo a simplicidade do esquema. Na Figura 3-3 as entidades mostradas à esquerda são as que estão relacionadas com a interface padrão a ser utilizada para mediação. Estas entidades representam a interface de mediação propriamente, seus métodos e os parâmetros destes, bem como, o conjunto de relacionamentos necessário para modelar este cenário. Do lado direito da figura, são vistas as entidades e relacionamentos referentes aos serviços concretos que serão alvo das invocações. Este conjunto deve prover as informações necessárias para viabilizar a invocação das operações destes serviços. Por fim, o *Contrato de Serviço* foi utilizado para representar o SLA ao qual o serviço está vinculado.

Entre as entidades da esquerda e da direita há os relacionamentos que definem o mapa propriamente. A utilização da entidade: Tipo de Parâmetro foi uma escolha de projeto com o intuito de possibilitar a conversão entre os diversos tipos de dados de um mesmo contexto (por exemplo, *int* para *short*).

3.4.2 Camada de Negócios

Nesta etapa do processo é feito o controle de transações, verificações de integridade, checagem de permissões e outros procedimentos com o objetivo de garantir a coerência, consistência e corretude das operações.

3.4.3 Fachada

Para viabilizar a utilização simultânea de variadas interfaces de acesso ao sistema (*Web Service* e *MBean,* por exemplo) permitindo também a adição futura de mecanismos de acesso não pensados em tempo de projeto, foi utilizado o padrão de

projeto *Facade* que disponibiliza uma interface única para um conjunto de interfaces distintas dentro do mesmo sistema [13].

Sendo assim, esta camada é responsável por simplificar a construção das camadas superiores fazendo as devidas manipulações e conversões de tipos entre os dados que trafegam destas para as camadas inferiores com o intuito de evitar o retrabalho.

3.4.4 Camada de Distribuição

Esta camada é responsável por tornar possível o acesso do Serviço de Publicação de Interface por outras aplicações. Os mecanismos de acesso planejados durante o projeto foram:

- *Web Service:* Aproveitando a facilidade de construção de um serviço deste tipo dentro do nosso ambiente, o *Web Service* se mostra uma alternativa bastante interessante por permitir inclusive acesso por outras máquinas em rede e a construção de uma ferramenta administrativa acessível via Web.
- *JMX:* Este mecanismo faz uso de *MBeans* para promover o acesso do sistema através da interface planejada para o VisualVM. Estes *MBeans* são registrados no *MBeanServer* da máquina hospedeira e têm como característica a utilização apenas de tipos primitivos Java para os parâmetros das operações.

No momento da inicialização do serviço, a camada de distribuição se encarrega de alocar os recursos necessários para tornar o serviço acessível externamente, registrando o mesmo nos devidos locais e criando os respectivos *endpoints.*

Esta camada não está restrita a uma única interface, sendo assim, cada tecnologia de distribuição utilizada aqui disponibilizará sua própria interface. Esta decisão foi tomada para não perder as facilidades inerentes a cada tecnologia em detrimento de um acesso padrão. A padronização dos tipos e acessos é feita pela Fachada que aqui é vista como um *wrapper* [13].

3.4.5 Mapas

É a principal estrutura de dados da solução, representa a relação existente entre uma interface e um serviço concreto. Nele é incorporada toda informação sobre os vínculos de métodos e operações juntamente com os respectivos parâmetros. Em decorrência disso, não é um tipo de dados de simples manuseio, sua construção e manipulação exigem acesso a vários outros tipos de dados, desde interfaces abstratas até parâmetros concretos de operações em serviços.

Pode-se considerar que um mapa é uma função que leva de uma interface a um serviço concreto, e por isso, existem algumas regras que precisam ser seguidas para a construção de um mapa válido.

1. Todos os métodos da interface devem possuir uma operação alvo no serviço.

2. Não pode haver métodos apontando para duas ou mais operações
3. Toda operação de um serviço concreto que possua mapeamento em algum método de uma interface, deve possuir obrigatoriamente uma imagem para cada um de seus parâmetros, refletindo uma constante ou um parâmetro do método que possui mapeamento.
4. Todos os parâmetros de um método mapeado a uma operação devem apontar para parâmetros desta operação ou para vazio.

3.5 Serviço de Checagem de Disponibilidade

Um grande problema na utilização de sistemas com componentes distribuídos ocorre em situações de indisponibilidade temporária destes componentes. Na maioria das vezes, este fato está fora do controle dos consumidores do serviço, entretanto a ciência destes eventos de indisponibilidade é deveras importante, pois permite a adoção de medidas de contingencia.

A forma mais simples de testar a conectividade entre dois sistemas em rede é enviando uma mensagem ICMP (*Internet Control Message Protocol*) *echo*, também conhecido como *ping* [20]. No entanto, dada a natureza complexa de uma invocação SOAP (*Simple Object Access Protocole*) [21], este artifício não é suficiente, pois o *host* pode responder ao *ping* e estar sem o serviço desejado ativo. Para melhor atender esta demanda foi concebido o Serviço de Checagem de Disponibilidade.

Este serviço permite ao usuário o cadastramento de itens de verificação. Estes itens correspondem às operações de um serviço remoto, sendo preciso para o cadastro a passagem de informações sobre localização, tipo de serviço, parâmetros e frequência da verificação. Estes dados são armazenados numa agenda e nos intervalos pré-definidos é iniciada uma *thread* para cada item de verificação agendando.

Com isso, o Serviço de Checagem de Disponibilidade se encarrega de atualizar o estado do serviço monitorado junto ao serviço de nomes de modo a permitir ou não a utilização do mesmo nos mapas cadastrados. É considerado êxito da verificação quando a chamada remota responde dentro de um tempo pré-estimado (de forma análoga ao *ping*) não é feita nenhuma verificação quanto à corretude desta resposta.

Este serviço foi concebido para ser tão simples quanto possível de modo a não causar uma sobrecarga desnecessária ao sistema ou congestionar a rede.

3.6 Ferramenta Gráfica

A construção da ferramenta gráfica para publicação dos mapeamentos é feita a partir da implementação de um cliente para o Serviço de Publicação de Mapeamentos. Este cliente é comandado por uma GUI (*Graphic User Interface)* e neste projeto, será integrada na forma de uma aba da ferramenta de gerenciamento e monitoramento padrão Java, a Visual VM. A ferramenta gráfica opera possibilitando a conexão aos vários aplicativos Java em execução numa dada máquina dos quais são extraídas informações de gerenciamento e monitoração.

Apesar da escolha da Visual VM como plataforma para criação de uma ferramenta gráfica para nossa solução, o projeto foi concebido de modo a possibilitar a construção de GUI de diversas outras naturezas.

4. Implementação

Este capítulo mostrará os aspectos gerais do desenvolvimento da solução proposta no Capítulo 3. Não serão abordados as minúcias da implementação, sendo o objetivo deste capítulo mostrar o roteiro que foi seguido para a concretização da proposta, mostrando a viabilidade técnica da mesma.

4.1 Ambiente de Desenvolvimento

Para o desenvolvimento deste trabalho foram utilizadas as seguintes ferramentas e tecnologias:

- IDE:
  - Eclipse 3.6
  - NetBeans 7.0
- Banco de dados
  - Apache Derby 10.8.1.2
- *Framework* de persistência de dados
  - Hibernate 4.0
- Java
  - Java SE 1.6.0.23
  - VisualVM 1.3.2

4.2 Serviço de Publicação de Mapeamentos

Este serviço, desenvolvido segundo a especificação JMX, fornece um conjunto de operações permitindo a inserção, remoção, busca e alteração de todas as estruturas mapeadas no diagrama entidade relacionamento visto na Figura 3-3. Em decorrência disso, sua interface possui uma extensa variedade de métodos. Estes métodos foram especificados com uso de tipos primitivos apenas de modo a possibilitara criação de um *MBean* para o serviço.

O seu programa principal é responsável por criar o *endpoint* de acesso via *Web Service* e por registrar um *MBean* para este.

Por ser um *framework* amplamente difundido e atender as necessidades da solução (ver Seção 3.4.1), o Hibernate [22] foi escolhido para compor sua camada de persistência, sendo utilizado em sua versão 4.0. O mapeamento das entidades persistentes foi feito com uso de JPA 2.0 (*Java Persistence API)* [23].

A camada de negócios implementa o modelo lógico do sistema, faz controle de transação, concorrência e lança as devidas exceções para sinalizar operações ilegais.

A fachada funciona como um *wrapper* [13] mediando e padronizando a passagem dos dados entre um conjunto de interfaces mais externas e a camada de negócios.

4.3 Serviço de Checagem de Disponibilidade

Este serviço, desenvolvido de acordo com a especificação JMX, fornece um conjunto mínimo de operações para checagem de disponibilidade dos serviços

remotos e permite o cadastro, a remoção e a verificação instantânea de itens agendados.

Foi desenvolvido com uso da API Cron4j, sendo esta a responsável por administrar a agenda de tarefas acionando eventos no horário indicado para as verificações. Sua escolha se baseou em sua simplicidade e na compatibilidade de formato com a ferramenta de agenda de tarefas *cron* [24] do sistema operacional Linux.

Neste serviço, também é utilizada a API de comunicação padrão do *Zookeeper*, permitindo que este modifique os metadados do XML (*eXtensible Markutp Language*) contendo as interfaces de serviço publicadas.

Visando obter uma maior simplicidade, este serviço foi desenvolvido sem a utilização de persistência para os dados. Com isso, sua agenda precisa ser carregada a cada inicialização da solução.

O seu programa principal é responsável por criar o *endpoint* de acesso via *Web Service* e por registrar um *MBean* para este.

4.4 Ferramenta Gráfica (Visual VM)

A Visual VM permite a conexão aos vários aplicativos Java em execução numa dada máquina. Ela possui um mecanismo de extensão que permite a incorporação de novas funcionalidades a partir de *plugins* compilados no formato NBM (*Net Beans Module*).

A Visual VM define alguns pontos de entrada onde podem ser acoplados os *plug-ins*. Deste modo, o desenvolvedor precisa identificar qual destes melhor se adequa ao seu cenário. Nossa solução foi incorporada na forma de uma aba, ficando hierarquicamente no mesmo nível das abas padrões do Visual VM.

Para isso, a nossa solução fornece uma classe que estende a classe *DataSourceViewProvider<Application>*, sendo este ponto de entrada, ativado no momento que a Visual VM se conecta a uma nova aplicação. Para garantir que a aplicação em questão dispõe de um serviço de publicação para ser utilizado por nossa ferramenta, é efetuada uma busca no *MBeanServer* da mesma por um *MBean* com o identificador equivalente ao *ObjectName* com o qual o *MBean* do Serviço de Publicação de Mapeamentos foi registrado. Em caso afirmativo a aba é exibida e cada operação do usuário é enviada ao *MBean* que executará a devida ação e retornará um resultado a ser exibido para o usuário. Caso o *ObjectName* não seja encontrado, a aba se oculta impedindo que o usuário tente invocar uma ação indisponível.

O método A, a seguir, implementa o mecanismo descrito anteriormente e retorna um booleano indicando se nossa aba deve ou não ser exibida. Existe também um método B que retorna um objeto do tipo *DataSourceView* organizando a estrutura visual que desejamos exibir ao usuário. Os métodos A e B citados no texto podem ser vistos no programa 4-1.

```
72      //Método A
73      protected boolean supportsViewFor(Application app) {
74          boolean ret = false;
75          try {
76
77              ret = JmxModelFactory.getJmxModelFor(app)
78                      .getMBeanServerConnection()
79                      .isRegistered(new ObjectName(MBEAN_NAME));
80          } catch (Exception ex) {
81              Exceptions.printStackTrace(ex);
82
83          }
84
85          return ret;
86      }
87
88      //Método B
89      protected DataSourceView createView(Application app) {
90          try {
91              mbean = JMX.newMXBeanProxy(JmxModelFactory.getJmxModelFor(app)
92                      .getMBeanServerConnection(), new ObjectName(MBEAN_NAME),
93                      IPublishMappingService.class);
94          } catch (Exception ex) {
95              Exceptions.printStackTrace(ex);
96          }
97
98          return new PrincipalView(app);
99      }
```

Figura 4-1: Estendendo a classe *DataSourceView*

Com base no modelo entidade relacionamento e visando construir uma interface limpa, intuitiva e fácil de usar, nossa aba foi dividida em três sub abas que correspondem ao: cadastramento de interfaces padrão, cadastramento de serviços concretos e criação dos mapas.

4.4.1 Aba de cadastramento de interfaces

Nesta aba o administrador cadastra as interfaces padrão que serão utilizadas para o desenvolvimento dos sistemas e serão mapeadas em serviços concretos. A fim de tornar mais ágil este processo e evitar o retrabalho, esta parte do cadastro é feita através da indicação de um arquivo JAR (*Java ARchive*) contendo as interfaces que se deseja publicar. Este arquivo é lido e então os dados das interfaces ali contidas são exibidos para confirmação antes do cadastramento efetivo (notem que qualquer outro tipo de dado também incluído no JAR será desconsiderado).

4.4.2 Aba cadastramento de serviços concretos

Nesta aba o administrador cadastra as instâncias concretas dos serviços remotos que serão o alvo final das invocações. Na versão atual da ferramenta, há suporte apenas para o cadastramento de serviços do tipo *Web Services*. O cadastramento de um *Web Service* é feito através da indicação de um arquivo WSDL (*Web Service Description Language*) [25] sobre o qual é feito um *parsing*, com auxílio da biblioteca *Membrane SOA Model*, objetivando extrair os dados do referido serviço. Feito isto, os dados são exibidos para confirmação antes do cadastramento efetivo.

Efetuada a publicação, a ferramenta automaticamente se conecta ao Serviço de Checagem de Disponibilidade para cadastrar uma checagem para o serviço publicado, pois é esta que valida os serviços para uso. Fazendo uso das informações contidas no WSDL a ferramenta busca uma operação sem parâmetros para cadastrar junto à

checagem, no caso da inexistência de tal operação os parâmetros precisam ser informados manualmente.

A partir deste momento o serviço terá seu estado aferido periodicamente e poderá ser utilizado na construção dos mapas.

4.4.3 Aba de criação de mapas

Nesta aba o administrador define os mapas entre as interfaces e serviços cadastrados. Todo o processo é manual e para facilitá-lo foi desenvolvida uma interface intuitiva e organizada. Os mapas vão sendo montados e exibidos numa estrutura hierárquica e ao final deste processo são publicados. Para isso, é acionado o *MBean* do Serviço de Publicação de Mapeamentos que recebe como parâmetro as associações recém criadas na interface.

Para a criação dos mapas são necessárias sucessivas invocações ao Serviço de Publicação de Mapeamentos, pois como dito anteriormente, a criação do mapa lida com uma complexa manipulação de dados para representar adequadamente o cenário criado pelo administrador do sistema.

5. Exemplos

Neste capítulo serão mostrados cenários de uso da ferramenta desenvolvida, servindo como um guia aos usuários. Os requisitos de software para acompanhar este tutorial podem ser encontrados na Seção 4.1

5.1 Instalação do *Plugin* no Visual VM

A distribuição padrão do Visual VM, incluída no JDK (*Java Development Kit*), pode ser obtida em: http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u1-download-513651.html, depois de instalado, o Visual VM pode ser executado a partir do diretório: *%JAVA_HOME%*/bin/jvisualvm.exe (onde *%JAVA_HOME%* é o diretório de instalação do JDK.

Nosso *plugin* foi compilado separadamente em dois arquivos, o "Ferramenta Visual VM.nbm" contém o conjunto de funcionalidades aqui especificado, já o "Dependencias.nbm" encapsula o conjunto das bibliotecas utilizadas pelo *plugin*.

Para a instalação do *plugin* deve ser acionado o menu "*tools*" em seguida clicado sobre o item "*Plugins*" como mostra a figura.

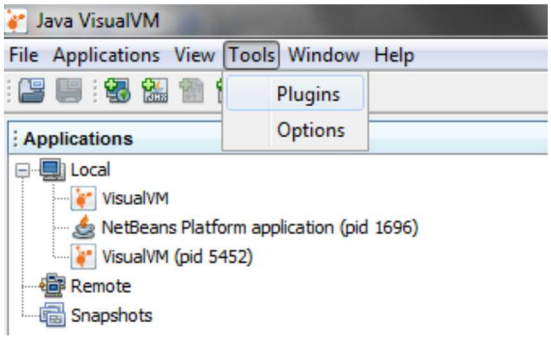

Figura 5-1: Instalação de *plugins*

Na tela exibida, deve ser selecionada a aba "*Downloaded*" e clicado no botão "*add plugin*" da mesma. Neste momento devem ser selecionados os dois arquivos referentes ao *plugin.*

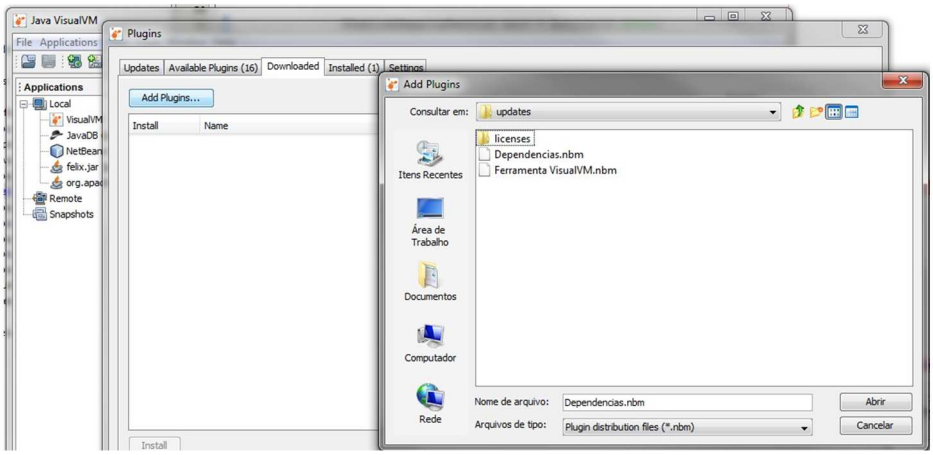

Figura 5-2: Seleção dos arquivos necessários a instalação

Por fim, confirme a intenção de instalá-lo clicando no botão "*install*". A partir deste momento o *plugin* estará instalado e pronto para uso.

5.2 Cadastro de Interfaces Padrão

Com o *plugin* devidamente instalado, vamos fazer o cadastro de um conjunto de interfaces. Inicialmente vê-se a tela da Visual VM ainda sem a exibição de nossa aba (Figura 5-3), pois como foi dito, esta apenas é exibida quando é detectado um Serviço de Publicação de Mapeamentos (Figura 5-4).

Figura 5-3: Visão inicial do Visual VM

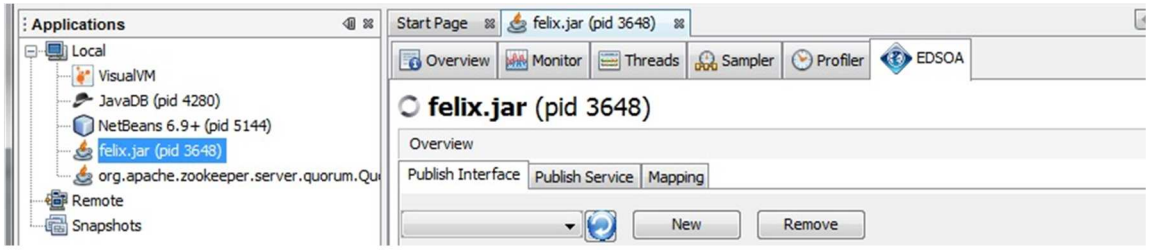

Figura 5-4: Visão do *plugin*

Dentro da sub aba "*Publish Interface*" podemos selecionar ou criar um domínio para englobar o contexto das interfaces que desejamos cadastrar e então indicar o arquivo JAR onde estas estão contidas. O exemplo aqui mostrado carregará interfaces para acesso a redes sociais, vinculando-as ao domínio "*Social Networks*". Para cada interface e método é possível adicionar uma descrição com alguma informação complementar. Ao fim deste processo é deve-se clicar no botão "*submit*" que encaminhará as informações ao Serviço de Publicação de Mapeamentos de modo a persistir os dados. A Figura 5-5 exemplifica este processo.

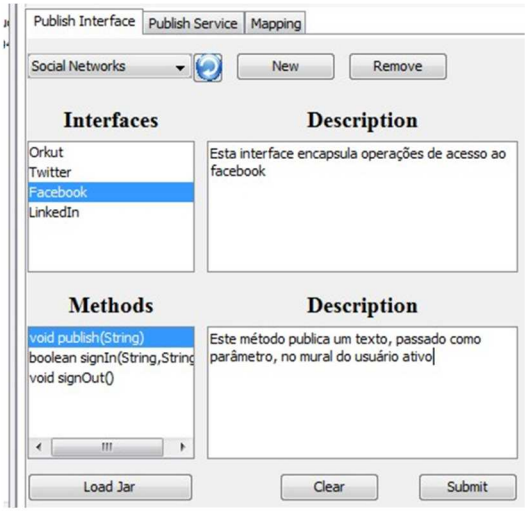

Figura 5-5: Cadastrando uma Interface

5.3 Cadastro de Serviços Concretos

Esta aba possui um visual semelhante a anterior, entretanto, difere quanto ao tipo do arquivo que é indicado para o carregamento automático das informações. Neste caso deve ser informado um arquivo contendo uma WSDL. Feito isto, as informações serão lidas e estarão listadas para o acréscimo de descrições. A Figura 5-6 resume este processo.

Ao clicar no botão "*submit*" o usuário é solicitado a informar a periodicidade da checagem de disponibilidade antes que o processo seja concluído.

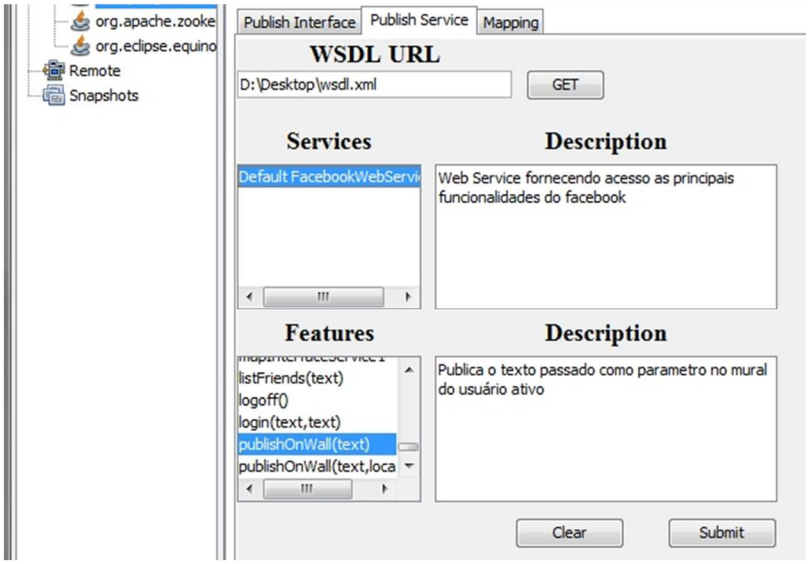

Figura 5-6: Cadastramento de um Serviço Concreto

5.4 Criação dos Mapas

Nesta aba o administrador verá três painéis: o mais a esquerda contem os dados referentes às interfaces cadastradas, o mais a direita contem dados dos serviços concretos cadastrados (notem que neste momento são considerados todos os serviços, independentemente da sua situação junto ao Serviço de Checagem de Disponibilidade) e o painel central contem os mapas utilizando uma notação (este cenário é observado na Figura 5-7).

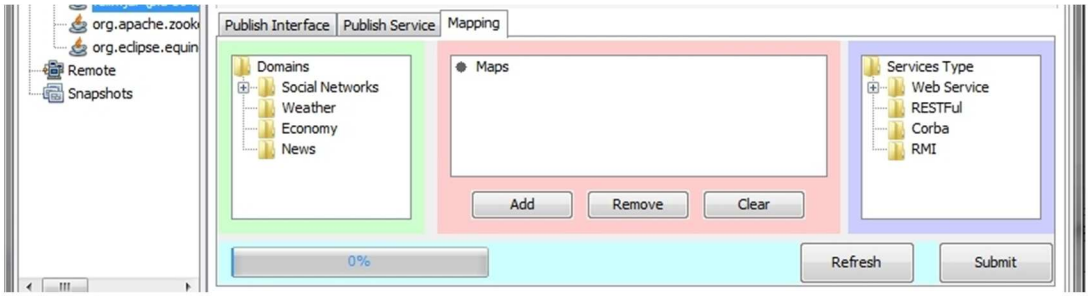

Figura 5-7: Aba de criação de mapas

A partir daqui o administrador pode navegar pelas várias interfaces e serviços cadastrados construindo os mapas. Isso é feito selecionando-se um item no painel esquerdo, outro item no painel direito e clicando no botão do painel central "*add*" que

vinculará os dois e exibirá esta relação no painel central (Figura 5-8). Quando a construção do mapa estiver completa deve ser clicado no botão *"submit"* que enviará as informações ao Serviço de Publicação de Mapeamentos.

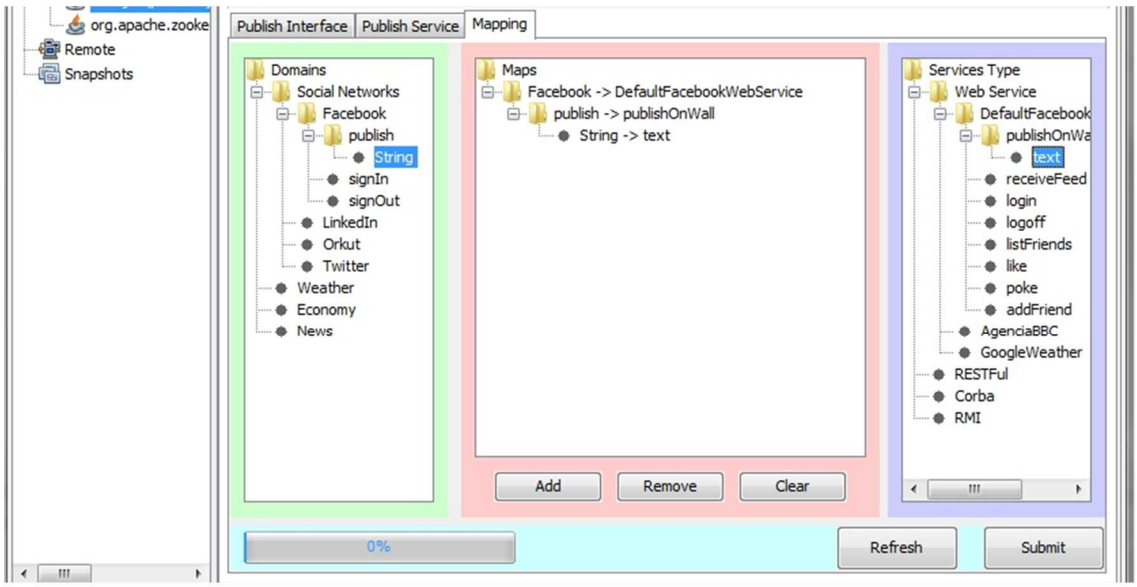

Figura 5-8: Construção de um Mapa

A construção dos mapas é o passo inicial para o funcionamento do *Mediator Proxy* e para a mediação dos serviços como um todo.

6. Conclusões e Trabalhos Futuros

Este trabalho mostrou o processo de desenvolvimento de uma solução para mediação de serviços distribuídos, capaz de tornar uma composição de serviços mais dinâmica, agregando ao DSOA a capacidade de utilização de serviços redundantes de interface, localização e tecnologia diferentes, através de uma única descrição cujas invocações são intermediadas para serem mapeadas em seus serviços alvo.

A implementação desta solução foi precedida de um estudo sobre as tecnologias OSGi e D-OSGi, uma compreensão detalhada da plataforma DSOA, reuniões e debates com participação da equipe criadora do DSOA, tudo isso visando garantir a adequação e a qualidade da proposta como um todo.

A finalização deste trabalho não conclui o escopo global do projeto elaborado, como espaço para trabalhos e estudos futuros ficam abertas as seguintes possibilidades:

- Implementação do *Mediador Proxy*, figura imprescindível para a completude da solução desenvolvida e que em detrimento das restrições temporais impostas para a conclusão deste trabalho não teve sua implementação contemplada no projeto. Responsável por ler as informações dos mapeamentos e realizar a mediação para a invocação concreta abstraindo as diferenças de interface ao consumidor do serviço.
- Criação de um gestor de QoS, um mecanismo responsável por disponibilizar aos consumidores de serviços uma instância, dentre as disponíveis, que se adeque aos parâmetros de qualidade requisitados pelos mesmos cruzando os dados da SLA informada pelo provedor, com os da monitoração. Este mecanismo deve contemplar algoritmos seleção que tornem viável a seleção dos serviços em tempo de execução, sem prejudicar o desempenho das composições.
- Extensão da ferramenta para suportar mediações compostas, isto é, quando a invocação de um método da interface A corresponde a invocação encadeada de dois métodos de B.

7. Referências Bibliográficas

[1] Papazoglou, M. P. (2003), Service-Oriented Computing: Concepts, Characteristics and Directions, *in* 'WISE', IEEE Computer Society, , pp. 3--12.

[2] Papazoglou, M. P. & Georgakopoulos, D. (2003), 'Introduction: Service-Oriented Computing', *Communications of the ACM* **46**(10), 24--28.

[3] Chung, J.-Y. & Chao, K.-M. (2007), 'A view on service-oriented architecture', .

[4] Pahl, C. & Barrett, R. (2010), 'Pattern-based software architecture for service-oriented software systems', .

[5] Frolund, S. & Koistinen, J. (1998), 'Quality-of-Service Specification in Distributed Object Systems'(HPL-98-159), Technical report, Hewlett Packard Laboratories, 39.

[6] Souza, F.; Lopes, D.; Gama, K.; Rosa, N. S. & Lima, R. (2011), Dynamic Event-Based Monitoring in a SOA Environment, *in* Robert Meersman; Tharam S. Dillon; Pilar Herrero; Akhil Kumar; Manfred Reichert; Li Qing; Beng Chin Ooi; Ernesto Damiani; Douglas C. Schmidt; Jules White; Manfred Hauswirth; Pascal Hitzler & Mukesh K. Mohania, ed., 'OTM Conferences (2)', Springer, , pp. 498--506.

[7] Booth, D.; Haas, H.; McCabe, F.; Newcomer, E.; Champion, M.; Ferris, C. & Orchard, D. (2011), 'Web Services Architecture', World Wide Web Consortium.

[8] Richardson, L. & Ruby, S. (2007), *RESTful web services - web services for the real world.*, O'Reilly.

[9] OMG (2008), 'Common Object Request Broker Architecture (CORBA/IIOP).v3.1', Technical report, Object Management Group.

[10] Ul Haq, I. & Schikuta, E. (2010), Aggregation patterns of service level agreements, *in* 'Proceedings of the 8th International Conference on Frontiers of Information Technology', ACM, New York, NY, USA, pp. 40:1--40:6.

[11] (visualvm.net), 'VisualVM is a visual tool integrating several commandline JDK tools and lightweight profiling capabilities.'.

[12] Schmidt, D. C.; Rohnert, H.; Stal, M. & Schultz, D. (2000), *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, John Wiley & Sons, Inc., New York, NY, USA.

[13] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1995), *Design Patterns*, Addison-Wesley, Boston, MA.

[14] Li, M.; Mani, M.; Rundensteiner, E. A.; Wang, D. & Lin, T. (2009), Interval event stream processing, *in* 'Proceedings of the Third ACM International Conference on Distributed Event-Based Systems', ACM, New York, NY, USA, pp. 35:1--35:2.

[15] The OSGi Alliance (2011), 'OSGi Service Platform Core Specification, Release 4.3', http://www.osgi.org/Specifications.

[16] Domaschka, J.; Schmidt, H.; Hauck, F. J.; Kapitza, R. & Reiser, H. P. (2009), DOSGi: An Architecture for Instant Replication, *in* 'Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009), Supplemental Volume'.

[17] Apache Software Foundation (2011), 'Zookeeper', letzter Zugriff: 11. Oktober 2011.

[18] Kreger, H.; Harold, W. & Williamson, L. (2003), *Java and JMX: Building Manageable Systems*, Addison-Wesley, Boston, MA.

[19] Papazoglou, M. P.; Traverso, P.; Dustdar, S. & Leymann, F. (2007), 'Service-Oriented Computing: State of the Art and Research Challenges', *Computer* **40**(11), 38--45.

[20] Postel, J. (1981), 'Internet Control Message Protocol'(792), Internet Engineering Task Force, IETF, RFC 792 (Standard), Updated by RFCs 950, 4884

[21] Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.-J. & Nielsen, H. F. (2003), 'SOAP Version 1.2 Part 1: Messaging Framework', W3C Recommendation.

[22] Bauer, C. & King, G. (2006), *Java Persistence with Hibernate*, Manning Publications Co., Greenwich, CT, USA.

[23] Yang, D. & Phd, D. (2010), *Java Persistence with Jpa*, Outskirts Press.

[24] Keller, M. S. (1999), 'Take Command: cron: Job Scheduler', *Linux J.* **1999**.

[25] Chinnici, R.; Moreau, J.-J.; Ryman, A. & Weerawarana, S. (2007), 'Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language', World Wide Web Consortium, Recommendation REC-wsdl20-20070626.