UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

Uma ferramenta para integrar sistemas Web em repositórios OSGi

Trabalho de Graduação

Aluno: Pablo José da Silva (pjs@cin.ufpe.br)

Orientador: Nelson Souto Rosa (nsr@cin.ufpe.br)

Recife, 19 de Julho de 2010

Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

Pablo José da Silva

Uma ferramenta para integrar sistemas Web em repositórios OSGi

---

Este trabalho foi apresentado ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para a obtenção do Grau de Engenheiro da Computação.

ORIENTADOR: NELSON SOUTO ROSA

Orientador

_____

Nelson Souto Rosa

Aluno

_____

Pablo José da Silva

Recife, 19 de Julho de 2010

"Muitos dirão que sou aventureiro, e sou mesmo, só que de um tipo diferente, destes que entregam a própria pele para demonstrar suas verdades."

(Ernesto “Che" Guevara)

Agradecimentos

Agradeço a Deus por ter me dado forças para alcançar o feito de ingressar em uma universidade e aos meus pais pelo apoio e esforço para garantir a mim uma educação de qualidade. Aos meus pais dedico todo o esforço para concluir este curso, pois sempre foram e sempre serão exemplos de caráter e dedicação para que eu nunca perca o foco e a concentração nos meus objetivos.

A minha namorada e amiga Renata Bezerra pelo carinho, apoio e compreensão em todos os momentos, sempre com muito amor e sinceridade. Inicialmente como amiga e depois como namorada me agüentando e dando forças nos momentos de dificuldades com muito amor e carinho.

Aos meus amigos de turma por dividir um pouco do seu conhecimento durante as aulas, projetos e provas ao longo do curso. Gostaria de agradecer especialmente aos que se tornaram meus grandes amigos, dentro e fora da universidade, os que juntos eram chamados de 'Corujas' e os nossos agregados por passar tantas noites no nosso centro. Aos amigos que encontrei dentro da universidade, de outros centros e outros cursos, que partilharam dos momentos de tristeza e alegria dentro e fora da UFPE.

Ao professor Nelson pelo seu esforço e dedicação durante a orientação deste trabalho e pela ajuda na estruturação e revisão para torná-lo conciso e interessante para os leitores. Ao meu 'co-orientador' Fábio Nogueira de Souza que se dedicou durante estes 5 meses de trabalho me ensinando e incentivando para a conclusão deste trabalho.

Muito obrigado a todos que foram lembrados e os que não foram lembrados. Muito obrigado novamente a todos que contribuíram em qualquer detalhe pela minha passagem pela universidade. Obrigado, obrigado e obrigado.

Resumo

Devido aos avanços tecnológicos em redes de computadores e à popularização da internet surge um novo paradigma para o desenvolvimento e integração de sistemas de informação: o paradigma de orientação a serviços. Este paradigma propõe que os sistemas sejam compostos por serviços comunicantes, os quais podem fornecer e/ou consumir serviços através de interfaces que estabelecem contratos bem definidos.

Nos dias atuais existem várias plataformas que provêem suporte para a orientação a serviços, dentre as quais se destacam *Web Services* e *OSGi*. A plataforma *OSGi* foi originalmente proposta como uma plataforma centralizada na qual os serviços são instalados, modificados e removidos, de forma dinâmica, em um ambiente. Os serviços providos são publicados em um repositório para que possam ser descobertos e, eventualmente, consumidos. A plataforma *Web Services* é baseada em um modelo distribuído no qual os serviços são definidos através de interfaces *WSDL* e publicados em um repositório a partir do qual podem ser descobertos. Uma vez descobertos, consumidor e provedor de serviço se comunicam através do protocolo SOAP.

Considerando-se as vantagens de cada uma destas plataformas, torna-se interessante conceber soluções que integrem serviços de naturezas heterogêneas. Contudo, integrar diferentes arquiteturas orientadas a serviços não é uma tarefa fácil e requer que o desenvolvedor conheça bem as especificidades de cada plataforma envolvida. Diante deste cenário, surge a necessidade do desenvolvimento de um *middleware* capaz de integrar serviços heterogêneos de maneira uniforme, apresentando para o desenvolvedor uma visão unificada do ambiente.

Um componente central do *middleware* proposto é um registro (repositório) integrado de serviços heterogêneos (*OSGi* e *Web Services*). Este registro armazena um conjunto de metadados associados a cada serviço cadastrado, os quais compreendem, além da interface, aspectos ligados à semântica e *QoS (Quality of Service)* dos serviços disponíveis. Neste contexto, este trabalho tem como objetivo o desenvolvimento do componente do *middleware* que servirá como ferramenta para registro dos *Web Services* no repositório integrado.

Palavras-Chaves: Serviços, Web Services, OSGi, Middleware.

Abstract

Due to technological advances in computer networks and to the popularization of the Internet, a new paradigm for the development and integration of information systems arises: the service orientation paradigm. This paradigm proposes that the systems are composed by communicating services, which can provide and/or consume services through interfaces providing well-defined contracts.

Nowadays there are several platforms that provide support for service orientation. Among these platforms, two stand out: *OSGi* and *Web services*. The *OSGi* platform was originally proposed as a centralized platform where services are installed, modified and removed, in a dynamic way, in an environment. The provided services are published in a repository so as to be discovered and, eventually, consumed. The *Web services* platform is based on a distributed model in which services are defined through WSDL interfaces and published in a repository where they can be discovered. Once they are discovered, service consumer and provider communicate through SOAP protocol.

Considering the advantages of each of these platforms, it seems to be interesting designing solutions that integrate heterogeneous services. However, integrating different service-oriented architectures is not an easy task and requires the developer to be familiar with the specifics of each platform involved. Considering this scenario, there is the need to develop a middleware capable of integrating heterogeneous services in a uniform way, presenting a unified view of the environment to the developer.

A fundamental component of the proposed middleware is an integrated record (repository) of heterogeneous services (*OSGi* and *Web Services*). This record stores a set of metadata associated with each registered service, which include, besides the interface, aspects related to the semantic and *QoS* (Quality of Service) of the services available. In this context, this work aims at the development of the middleware component that will serve as a tool for registering the *Web Services* in the integrated repository.

Keywords: Services, Web Services, OSGi, Middleware.

Conteúdo

Lista de Figuras

1. Introdução

Devido aos avanços tecnológicos em redes de computadores e à popularização da Internet surge um novo paradigma para o desenvolvimento e integração de sistemas de informação: o paradigma de orientação a serviços [1] [2] [3]. Este paradigma propõe a integração de serviços comunicantes, os quais podem tanto fornecer quanto consumir serviços através de interfaces que estabelecem contratos bem definidos.

Atualmente existem várias plataformas que provêem suporte para a orientação a serviços, destacando-se: *OSGi* [4] e *Web Services* [5]. A plataforma *OSGi* foi originalmente proposta como uma plataforma centralizada na qual os serviços são instalados, modificados e removidos, de forma dinâmica, em um ambiente. Os serviços providos são publicados em um repositório para que possam ser descobertos e, eventualmente, consumidos. A plataforma *Web Services* é baseada em um modelo distribuído no qual os serviços são definidos através de interfaces *WSDL* [6] e publicados em um repositório no qual podem ser descobertos por outras aplicações. Com o funcionamento distribuído, a plataforma *Web Services* não suporta dinamismo no tratamento dos serviços ofertados não tendo assim, o dinamismo que a plataforma *OSGi* provê ao seu usuário.

Considerando-se as vantagens de cada uma destas plataformas, torna-se interessante conceber soluções que integrem serviços de naturezas heterogêneas. Contudo, integrar diferentes arquiteturas orientadas a serviços não é uma tarefa fácil e requer que o desenvolvedor conheça bem as especificidades de cada plataforma envolvida. Diante deste cenário, surge a necessidade do desenvolvimento de um *middleware* capaz de integrar serviços heterogêneos de maneira uniforme, apresentando para o desenvolvedor uma visão unificada do ambiente.

Diferentes abordagens podem ser utilizadas na proposição de uma plataforma unificada. Pode-se, por exemplo, propor uma nova arquitetura orientada a serviço que possua as principais características do conjunto de arquiteturas disponíveis. Os serviços disponíveis nas diferentes plataformas seriam acessados através de *Proxies*

[20] gerados pelo middleware, os quais fariam o papel de adaptadores.

Este trabalho, contudo, se enquadra no contexto de outra arquitetura de *middleware* que propõe a extensão da plataforma OSGi permitindo a integração de de serviços remotos e heterogêneos. Neste middleware particular, os serviços são selecionados dinamicamente com base em um conceito amplo de contrato, que compreende não só a interface sintática e semântica dos serviços consumidos, mas também requisitos relacionados à Qualidade de Serviço.

Dentre os critérios que orientaram a escolha de OSGi como plataforma de base, destaca-se o suporte ao dinamismo através de um mecanismo de notificação capaz de informar às aplicações em execução quando do aparecimento, desaparecimento ou alteração nos serviços utilizados. Tal mecanismo permite o suporte ao desenvolvimento de sistemas adaptativos robustos, os quais suportam, de modo automático, modificações em seu ambiente de execução.

Para compor a camada de distribuição, o middleware propõe a utilização das tecnologias relacionadas à *Web Services*. Esta camada é transparente para os desenvolvedores uma vez que os serviços remotos são utilizados através de *Proxies*. Desta forma, para que os desenvolvedores possam utilizar o *middleware* proposto, é suficiente que estes estejam familiarizados com a plataforma OSGi.

1.1 Objetivos

Um componente central do *middleware* proposto é um repositório integrado de serviços heterogêneos (*OSGi* e *Web Services*). Este repositório armazena um conjunto de metadados associados a cada serviço cadastrado, os quais compreendem, além da interface, aspectos ligados à semântica e *QoS (Quality of Service)* dos serviços disponíveis. Tais metadados são utilizados na geração de proxies e na etapa de descoberta dinâmica de serviços. Nesse contexto, este trabalho tem como objetivo o desenvolvimento de uma ferramenta capaz de suportar o registro de *Web Services* no repositório integrado de serviços.

1.2 Estrutura do Documento

O documento está organizado na seguinte forma:

- **Capítulo 2** apresenta os principais conceitos relacionados à Arquitetura orientada a serviços.
- **Capítulo 3** faz o resumo da descrição e do funcionamento das tecnologias utilizadas na plataforma de serviços *Web Services.*
- **Capítulo 4** faz o resumo da descrição e do funcionamento das tecnologias utilizadas na plataforma centralizada de serviços *OSGi*: OSGi, *DOSGi* (Apache CXF) e *ZooKeeper*.
- **Capítulo 5** mostra as etapas de desenvolvimento do componente e o seu funcionamento.
- **Capítulo 6** encerra o texto apresentando as conclusões do trabalho e as direções para trabalhos futuros.

2. Service Oriented Architecture

Este capítulo apresenta os conceitos básicos e uma visão geral sobre o paradigma de orientação a serviços.

2.1 Conceitos Básicos

A Arquitetura Orientada a Serviço (SOA) é um paradigma para organização e utilização de competências distribuídas que estão sob controle de diferentes domínios proprietários [1].

Normalmente, entidades (pessoas e organizações) criam competências para solucionar problemas que encontram nos seus negócios. Contudo essas competências criadas podem ser necessidades para outras entidades. SOA visa oferecer um poderoso *framework* para compatibilizar necessidades e competências.

O principal conceito de SOA é o de serviço. O nome "serviço" é definido no dicionário como "O desempenho de trabalho (uma função) por alguém para outro". Contudo, serviço, como o termo é geralmente entendido, também combina as idéias relacionadas com:

A competência de executar o trabalho para outro.

- A especificação do trabalho oferecido para outro.
- A oferta para executar trabalho para outro.

No SOA, os serviços são o mecanismo pelo qual as necessidades e as competências são colocadas juntas.

Em geral, as entidades (pessoas e organizações) oferecem competências e atuam como provedores de serviço. Aqueles com necessidades que fazem uso dos serviços são referenciados como consumidores de serviço.

2.1.1 Os Benefícios da Arquitetura Orientada a Serviço

Segundo [1] os benefícios do uso de SOA são:

- Facilitar o gerenciamento do crescimento dos sistemas corporativos de larga escala.
- Facilitar o provisionamento da escalabilidade da Internet para uso por serviços.
- Reduzir custos nas organizações para cooperação das organizações.

2.2 Serviço

Um serviço é um mecanismo para habilitar o acesso a um conjunto de uma ou mais competências, onde o acesso é provido usando uma interface que descrita e é consistentemente exercitada com restrições e políticas que são especificados pela descrição de serviço.

Um serviço é acessado por meio de uma interface de serviço onde a interface inclui as especificidades do conhecimento para acessar as competências subjacentes. Não há restrições no que constitui as competências subjacentes ou como o acesso é implementado pelo provedor. Então, o serviço pode ser executado como descrito na funcionalidade através de um ou mais processos automáticos e/ou manuais que ele próprio invocar a outros serviços disponíveis.

Figura 2-1: Estrutura associada ao serviço

A figura 2-1 representa a estrutura associada ao serviço.

Seguindo os conceitos de [1] vamos detalhar cada estrutura para melhor compreender a arquitetura SOA.

2.2.1 Descrição do Serviço

A descrição do serviço representa a informação necessária de forma a usar um serviço. Na maioria dos casos, não há uma descrição "certa", mas ao invés disto, os elementos da descrição requerida dependem do contexto e das necessidades das partes que usam a entidade associada.

O propósito da descrição é facilitar a interação e a visibilidade, particularmente quando os participantes estão em domínios proprietários diferentes, entre participantes na interação de serviço. Pelo oferecimento da descrição, ela torna possível a potenciais participantes construir sistemas que usam serviços e até mesmo oferece serviços compatíveis.

Por exemplo, as descrições permitem que os participantes discriminem entre possíveis escolhas para interação de serviço; como se o serviço oferecido requer competências, como acessar o serviço, e negociar sobre as funcionalidades específicas de serviço. Ainda mais, as descrições podem ser usadas para suportar e gerenciar serviços, ambos sob a perspectiva do provedor de serviço e sob a perspectiva do consumidor de serviço.

2.2.2 Políticas e Contratos

Uma política representa alguma restrição ou condição sobre o uso, distribuição ou descrição de uma entidade própria com definida por qualquer participante. Um contrato, por outro lado, representa um acordo de duas ou mais partes. Assim como as políticas, acordos são também sobre as condições de uso de um serviço; ele pode também restringir os efeitos esperados no mundo real ao usar o serviço.

2.2.3 Contexto de Execução

O contexto de execução de uma interação de serviço é o conjunto de elementos de infraestrutura, entidades processo, afirmações e acordos de políticas que são identificados como parte de uma interação de serviço instanciado, e então forma um caminho entre aqueles com necessidades e aqueles com competências.

2.3 As Dinâmicas dos Serviços

Seguindo a definição de [1] existem três conceitos fundamentais para entender o que está envolvido na dinâmica dos serviços:

- A visibilidade entre provedores de serviços e consumidores;
- A interação entre eles;
- Os efeitos no mundo real da interação com um serviço;

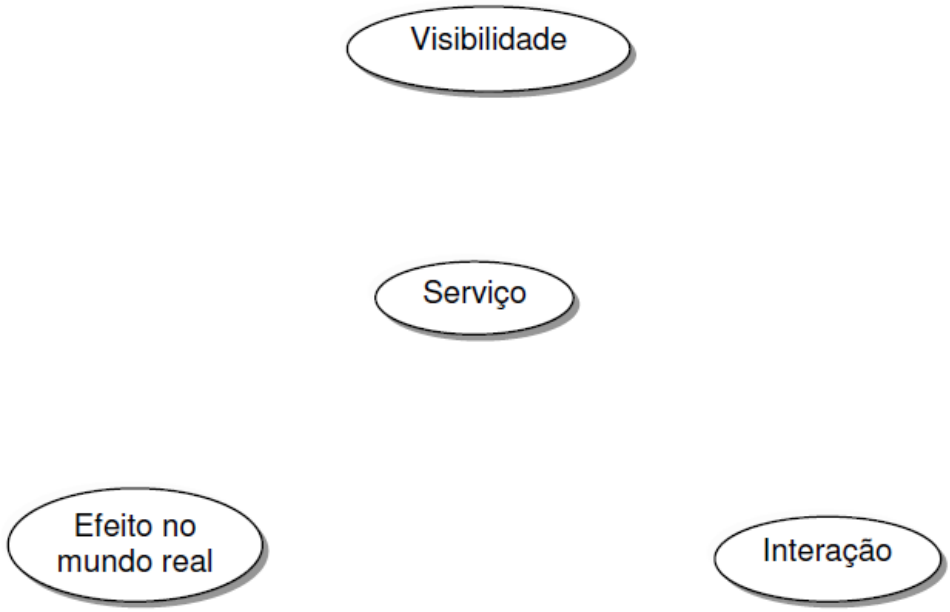


Figura 2-2: Conceito sobre as dinâmicas do serviço

A figura 2-2 mostra os conceitos sobre as dinâmicas do serviço.

2.3.1 Visibilidade

A Visibilidade é o relacionamento entre os consumidores e provedores de serviço que é satisfeito quando eles estão aptos a interagirem entre si. As precondições para visibilidade são consciência, concordância e acessibilidade. O iniciante numa interação de serviço precisa ter a percepção dos outros parceiros, os participantes precisam estar predispostos para uma interação, e os participantes precisam estar aptos a interagir [1].

A figura 2-3 mostra os conceitos ao redor de Visibilidade, são eles: Consciência, concordância e acessibilidade.

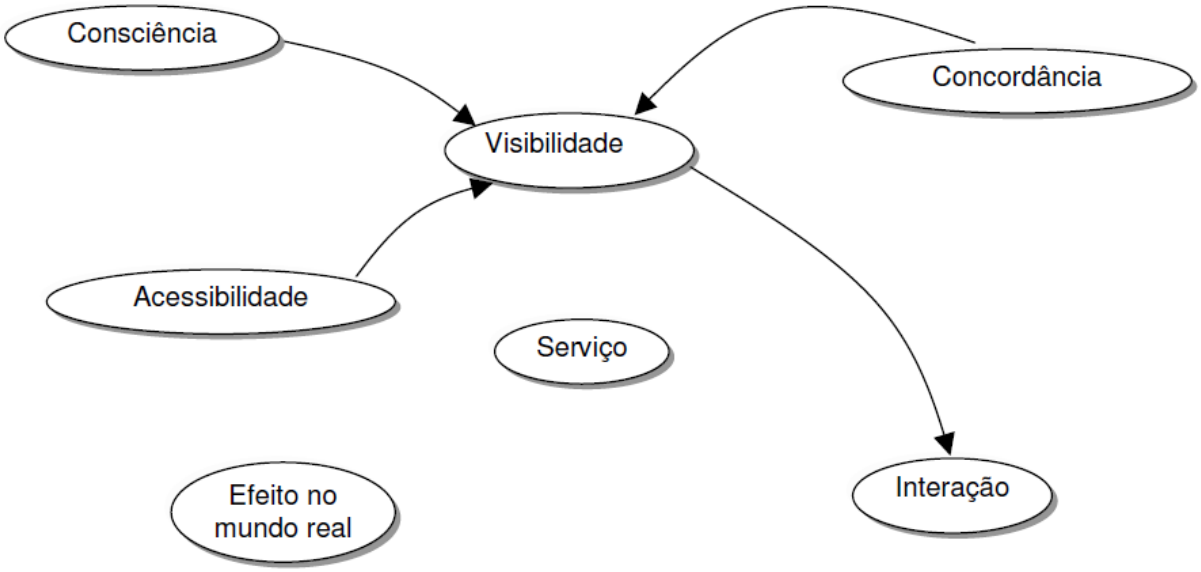


Figura 2-3: Conceitos ao redor de Visibilidade

- **Consciência:**

  Um estado por meio do qual uma parte tem conhecimento da existência de outra parte.

- **Concordância:**

  A predisposição dos provedores e consumidores de serviço para interagirem.

- **Acessibilidade:**

  A habilidade de um consumidor de serviço e um provedor de serviço interagir.

2.3.2 Interação

A interação com o serviço envolve a execução de ações sobre o serviço. Em muitos casos, isto é realizado pelo envio e recebimento de mensagens, mas há outros modos possíveis que não envolvem explicitamente a transmissão de mensagens [1].

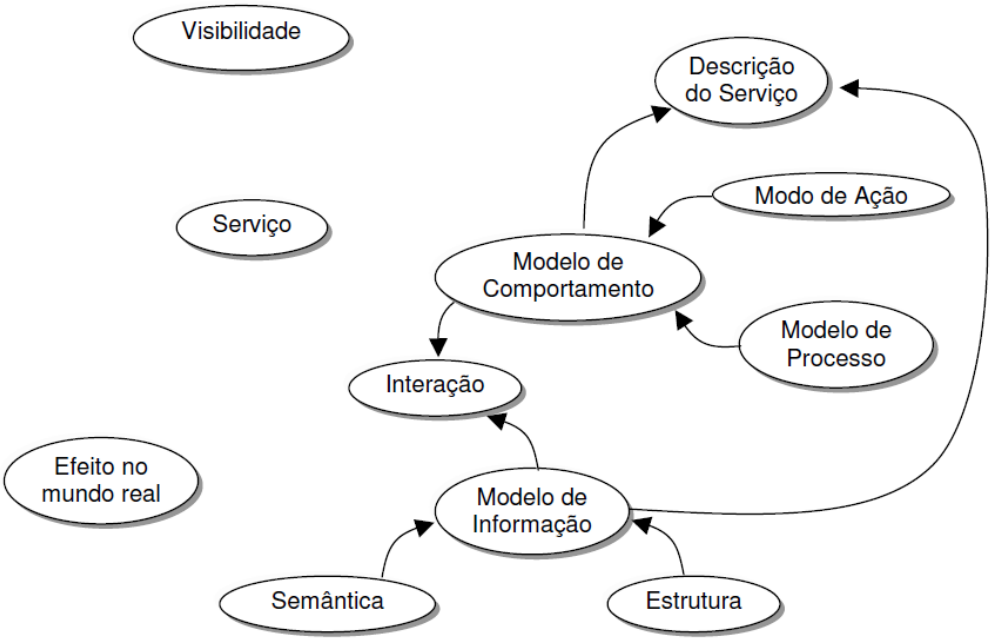


Figura 2-4: Conceitos na interação de serviço

A figura 2-4 ilustra os conceitos chaves que são importantes no entendimento do que está envolvido numa interação com serviços; estes conceitos estão relacionados à descrição do serviço – que referencia um modelo de informação e um modelo de comportamento [1].

- **Modelo de informação:**

  O modelo de informação de um serviço é uma caracterização da informação que pode ser trocada com o serviço. O modelo de informação pode ser bipartido em Semântica e Estrutura.

- **Modelo de Comportamento:**

  É o conhecimento das ações envolvidas no serviço e no processo ou aspectos temporais de uma interação com o serviço. Isto é caracterizado como conhecimento das ações sobre as respostas, e as dependências temporais entre ações sobre o serviço [1].

2.3.3 Efeito no mundo real

O resultado real do uso de um serviço, ao invés de apenas a competência oferecida pelo provedor de serviços [1].

3. Web Services

Segundo [5] e [6] vamos mostrar os principais conceitos sobre *Web Services* para maior entendimento da ferramenta que será apresentada no capítulo 5.

3.1 Definição

Um *Web service* é um software desenhado para suportar a interoperabilidade entre computadores em uma *network*. O *web service* tem uma interface descrita em uma linguagem chamada WSDL. Outros sistemas se comunicam com *web service* utilizando o protocolo SOAP (Simple Object Access Protocol) tipicamente serializado em XML veiculado utilizando HTTP [21].

3.1.1 Agentes e Serviços

Um *Web service* é uma abstração que deve ser implementada por um agente concreto. O agente é uma parte concreta de software ou hardware que manda e recebe mensagens enquanto o serviço é o recurso caracterizado pelo conjunto de funcionalidades que são providas.

3.1.2 Solicitantes e Provedores

A entidade provedora é a pessoa ou organização que provê um agente apropriado para a implementação de um serviço particular.

A entidade solicitadora é a pessoa ou organização que deseja utilizar um serviço provido por uma entidade provedora.

3.1.3 Descrição do Serviço

A mecânica de troca de mensagens é documentada no *Web Service Description* (WSD). O WSD é a especificação da interface do *Web service*, essa especificação é escrita em WSDL. No WSD é definido o formado das mensagens, os

tipos de dados, os protocolos de transporte e o formato de serialização de transporte que deverá ser utilizado entre o agente solicitante e o agente provedor. Também pode ser especificado no WSD em quais *networks* cada agente provedor pode ser invocado.

O WSD representa o acordo que governa o mecanismo de interação com o serviço.

3.1.4 Semântica

A semântica de um *Web service* é o compartilhamento da expectativa sobre o comportamento do serviço, em particular a resposta das mensagens que são enviadas para o serviço. A semântica é o contrato entre a entidade solicitante e a entidade provedora guardando a finalidade e as conseqüências da interação.

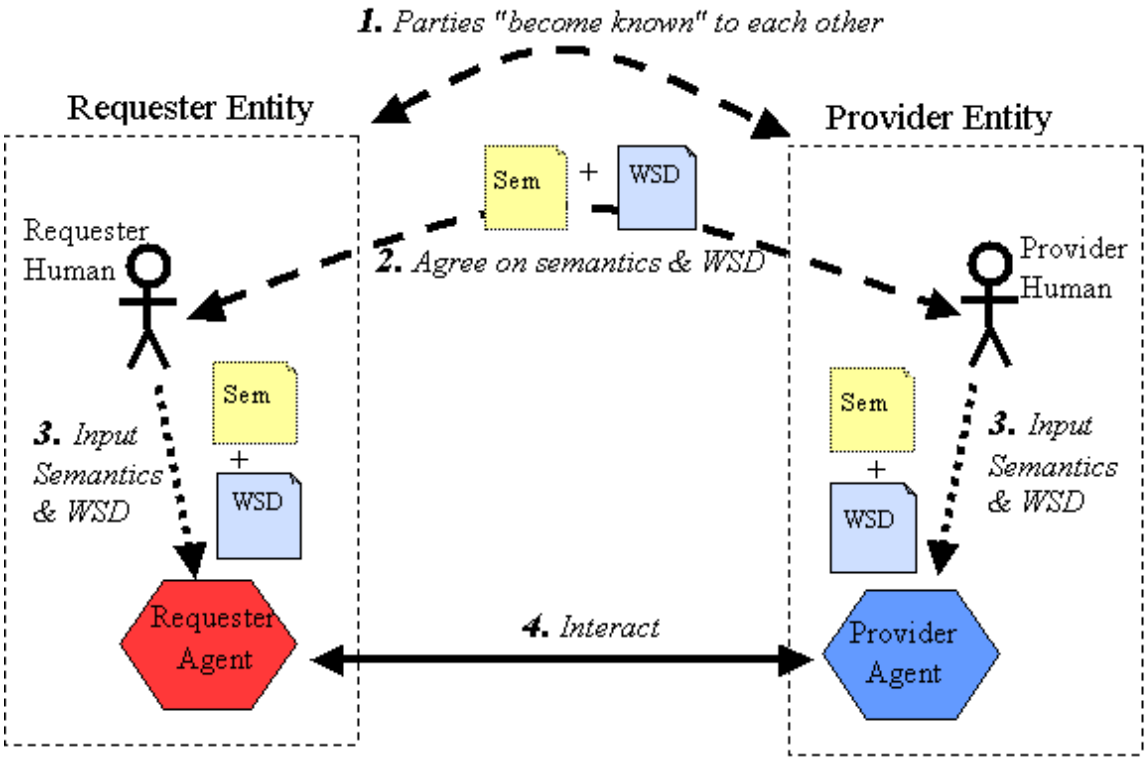
Enquanto o WSD representa o contrato que governa o mecanismo de interação de um particular serviço, a semântica representa o contrato que governa o significado e os fins da interação.

3.1.5 Esquemático

Existem várias maneiras de uma entidade solicitadora utilizar um *Web service*. Usualmente os passos básicos requeridos são mostrados na figura 3-1.

1. As entidades solicitadoras e provedoras conhecem a existência uma das outras.
2. As entidades solicitadoras e provedoras aceitam o contrato WSD e semântica que irão governar a interação entre os agentes solicitadores e provedores.
3. O WSD e a semântica são realizados pelos agentes solicitadores e provedores.

4. Os agentes solicitantes e provedores trocam mensagens.



3-1:Esquemático de web services

4. OSGi

Neste capítulo abordaremos as definições e os conceitos clássicos para o entendimento da plataforma OSGi baseando-se nas idéias de [4] e [9].

4.1 Definição

A tecnologia OSGi é definida como um sistema de módulos dinâmicos para linguagem Java. OSGi foi desenvolvida pela OSGi Alliance, fundada em 1999, e tem como objetivo a especificação de um *framework* para o gerenciamento de módulos Java que permite a instalação, atualização e desinstalação destes módulos em tempo de execução [4]. Este *framework* é responsável por manter a consistência dos serviços ao controlar a relação de dependência entre os módulos instalados.

4.2 OSGi Framework

A definição de framework é o core principal da especificação da tecnologia OSGi, *OSGi Service Platform Specifications*. Nesta especificação estão definidas as suas principais funcionalidades [9]. A figura 4-1 exemplifica como a funcionalidade do framework é definida em camadas.

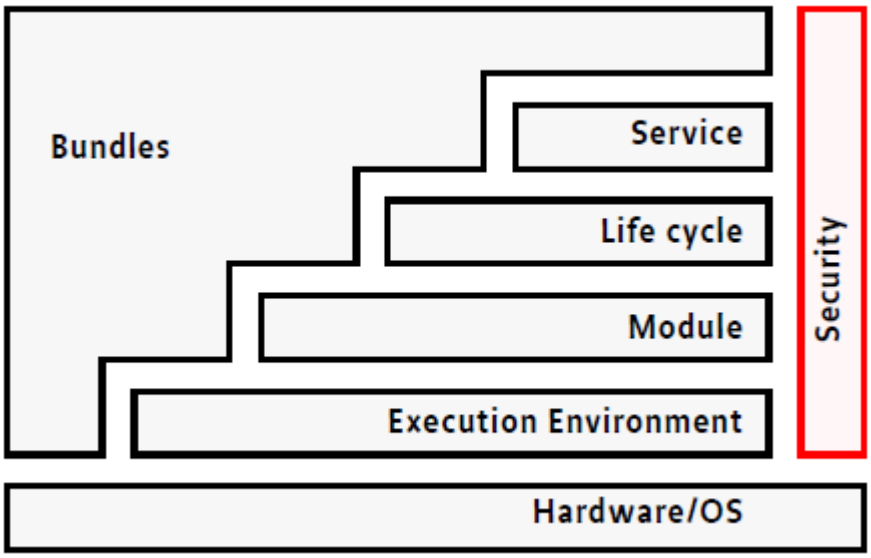


Figura 4-1: Camadas OSGi

4.2.1 Bundles

O framework OSGi define uma unidade de modularização chamada bundle. Um bundle é composto por classes Java e outros recursos, que juntos podem prover funcionalidades para os usuários finais. Bundles podem compartilhar Java packages através de um *exporter-bundle* e um *importer-bundles* de uma maneira bem definida [4].

Um bundle é implantado como um Java ARchive (JAR) file. Arquivos JAR são utilizados para guardar aplicações e seus recursos em um formato ZIP-bases file. Este formado é definido por [12] Zip File Format [4].

Bundle é um arquivo JAR que:

- Contém recursos necessários para prover alguma funcionalidade.
- Contém um arquivo Manifest que descreve o conteúdo do arquivo JAR e provê informação sobre o bundle. No manifest existem cabeçalhos que especificam o que o framework necessita para instalar e ativar corretamente o bundle.
- Pode conter uma documentação opcional no diretório OSGI-OPT do arquivo JAR ou em um dos seus subdiretórios. Qualquer informação nesses diretórios é opcional.

4.2.2 Ambiente de execução

O ambiente de execução é a especificação do ambiente de execução Java. No ambiente de execução os bundles são implantados.

4.2.3 Camada de segurança

A camada de segurança do OSGi é uma camada opcional que é subjacente a camada de Serviços. Esta camada é baseada em *Java 2 Security*

Architecture [11] e provê toda a infra-estrutura para implantar e gerenciar aplicações que devem ter como finalidade o controle das aplicações que estão sendo executadas no ambiente OSGi [4].

4.2.4 Camada de módulos

O framework OSGi provê uma solução genérica e padronizada para a modularização Java. A camada de módulos define como devem ser os módulos Java e possui regras para que os diferentes pacotes Java tanto possam ser carregados e compartilhados.

4.2.5 Camada de ciclo de vida

Define o modelo de ciclo de vida para os *bundles*, determinando quando estes poderão ser iniciados ou parados, e quando estes deverão ser instalados, atualizados e desinstalados. A camada de ciclo de vida é baseada nas camadas de módulos e segurança.

Os bundles seguem os seguintes estados no ambiente de execução:

- **Installed**: O bundle foi instalado, porém não foram verificadas as suas dependências.
- **Resolved**: Bundle instalado e com suas dependências verificadas. Pronto para ser inicializado.
- **Uninstalled**: Após o comando 'uninstall' o bundle é desinstalado e o estado vai para uninstalled.
- **Starting**: Bundle inicializado, após o comando 'start' o bundle vai para o estado active.
- **Active**: Bundle ativo, em funcionamento.
- **Stopping**: Bundle parado, após o comando 'stop' o bundle volta ao estado resolved.

A figura 4-2 demonstra o ciclo de vida de um bundle no ambiente de execução OSGi [10].

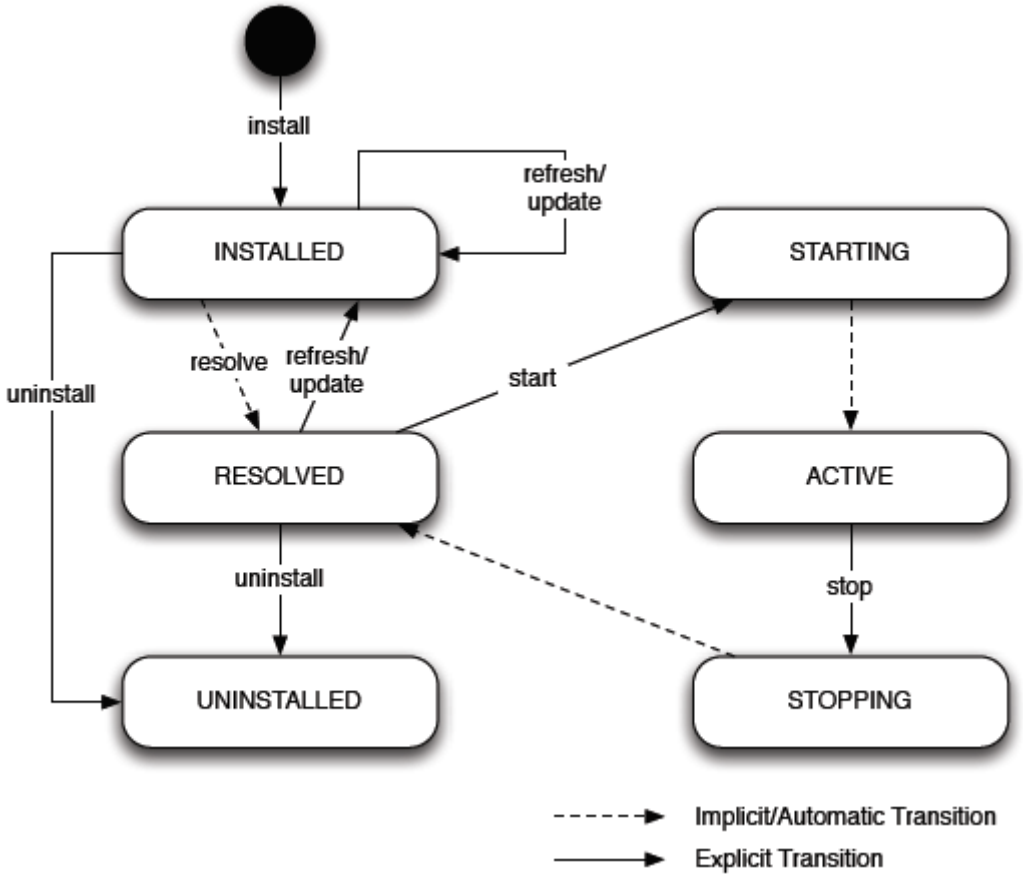


Figura 4-2: Ciclo de vida de um bundle

4.2.6 Camada de serviços

A camada de serviços OSGi define o modelo de programação dinâmico colaborativo que simplifica o desenvolvimento de serviços ao desacoplar suas interfaces de suas implementações. A camada de serviços é estritamente ligada à camada de ciclo de vida.

Um serviço é um objeto Java que é registrado a uma ou mais interfaces Java com o *service registry*. Através do *service registry*, os *bundles* selecionam serviços disponíveis a partir de suas interfaces podendo registrar novos serviços e receber notificações sobre os estados dos serviços atuais, assim

como procurar novos serviços que melhor se adaptem aos dispositivos atuais. A figura 4-3 mostra a relação entre bundles e serviços.

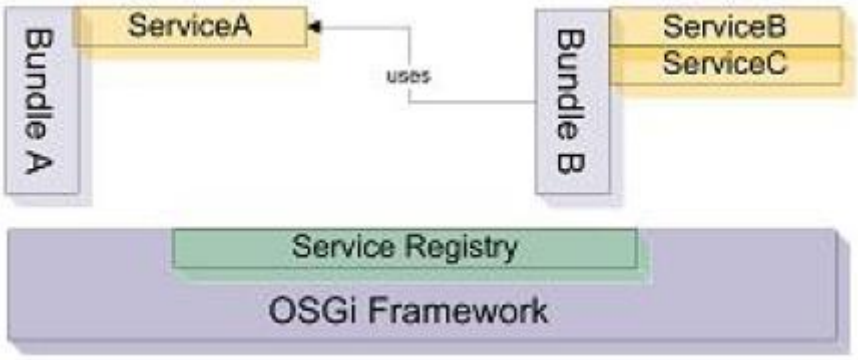


Figura 4-3: Bundles e Serviços

4.2.7 Comunicação entre as camadas

A figura 4-4 ilustra a interação entre as camadas do OSGi e um bundle.

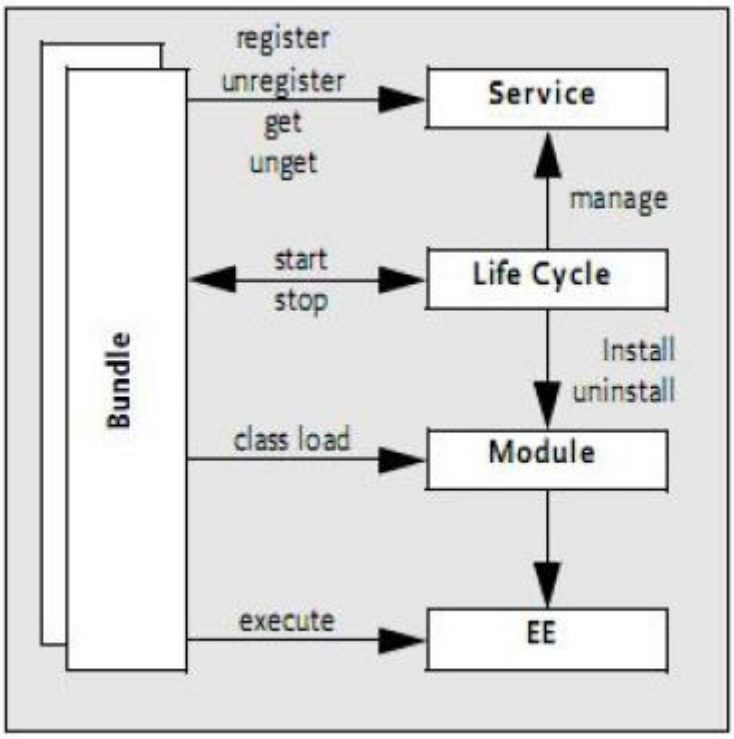


Figura 4-4: Comunicação entre camadas

4.3 Distributed OSGi

Inicialmente a tecnologia OSGi foi desenvolvida para uso em dispositivos embarcados, onde o *framework* executa em um ambiente isolado[9], porém surgiu a necessidade de integração entre serviços distribuídos então surgiu a idéia do Distributed OSGi (DOSGi).

A idéia principal do DOSGi é a criação de um repositório de serviços centralizado onde cada que seja publicado nos repositórios locais, e que sejam marcados como serviços remotos, tenham uma cópia no repositório global.

No nosso projeto utilizaremos a implementação do DOSGi Apache CXF[7] que utiliza como repositório global o serviço para configuração e monitoramento de informações de serviços distribuídos ZooKeeper[14].

A figura 4-5 mostra a arquitetura do DOSGi.

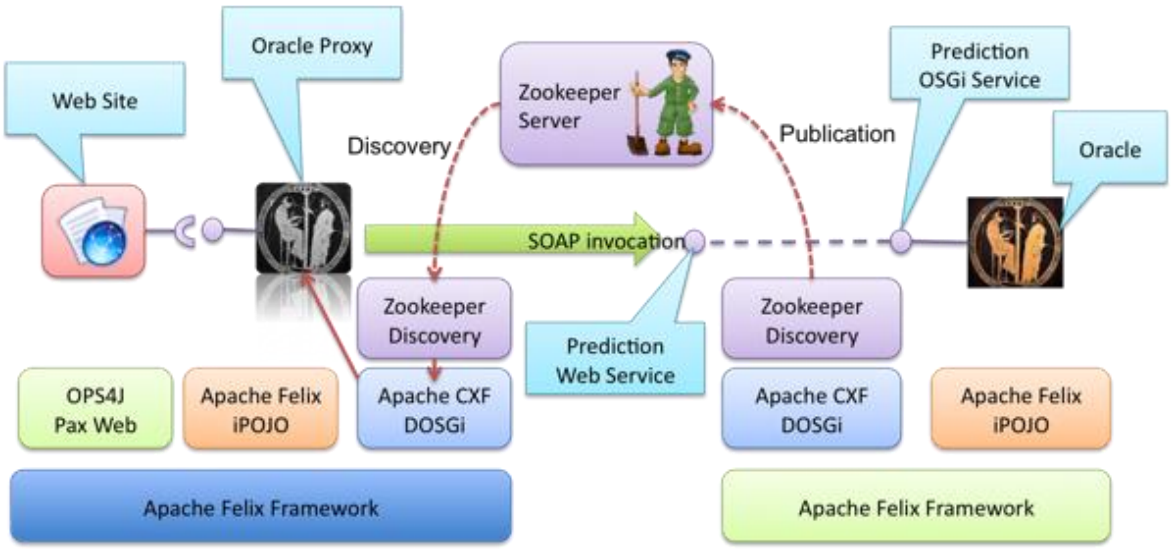


Figura 4-5: Arquitetura DOSGi

5. Componente

Este capítulo aborda o desenvolvimento do componente do *middleware* que implementa uma ferramenta para registro de *Web Services* no repositório integrado de serviços. O Componente do *middleware* pode ser dividido em duas partes: O repositório e O gerador.

5.1 Arquitetura

Conforme mencionado no capítulo 1, este trabalho foi desenvolvido dentro do contexto da concepção de um *middleware* capaz de integrar serviços disponíveis em plataformas SOA heterogêneas. O *middleware* em questão foi projetado para suportar o desenvolvimento de aplicações adaptáveis, concebidas como composições dinâmicas de serviços, as quais podem ser modificadas em tempo de execução através da troca de serviços utilizados por outros considerados "compatíveis". A seleção dos serviços que compõem uma aplicação é realizada (e, possivelmente modificada) dinamicamente, utilizando como critério, aspectos relacionados à qualidade desses serviços.

A figura 5-1 mostra a arquitetura do *middleware* proposto. O componente central deste middleware é o barramento de serviços que é fornecido pela plataforma OSGi, a qual foi escolhida devido às qualidades apresentadas no capítulo 4 desse trabalho. Os serviços ligados ao barramento são catalogados em um repositório integrado, tornando-se visíveis para as aplicações instaladas na plataforma. Cada serviço catalogado é descrito através de uma interface, que define a forma de acesso à funcionalidade correspondente, e de um conjunto extensível de metadados que determinam características semânticas e parâmetros de qualidade, os quais são interpretados e monitorados pelo middleware.

Conforme ressaltado na figura 5-1, os serviços implementados em outras plataformas são acessados através de *proxies* que encapsulam o mecanismo utilizado na comunicação (normalmente SOAP/HTTP) fornecendo transparência de localização.

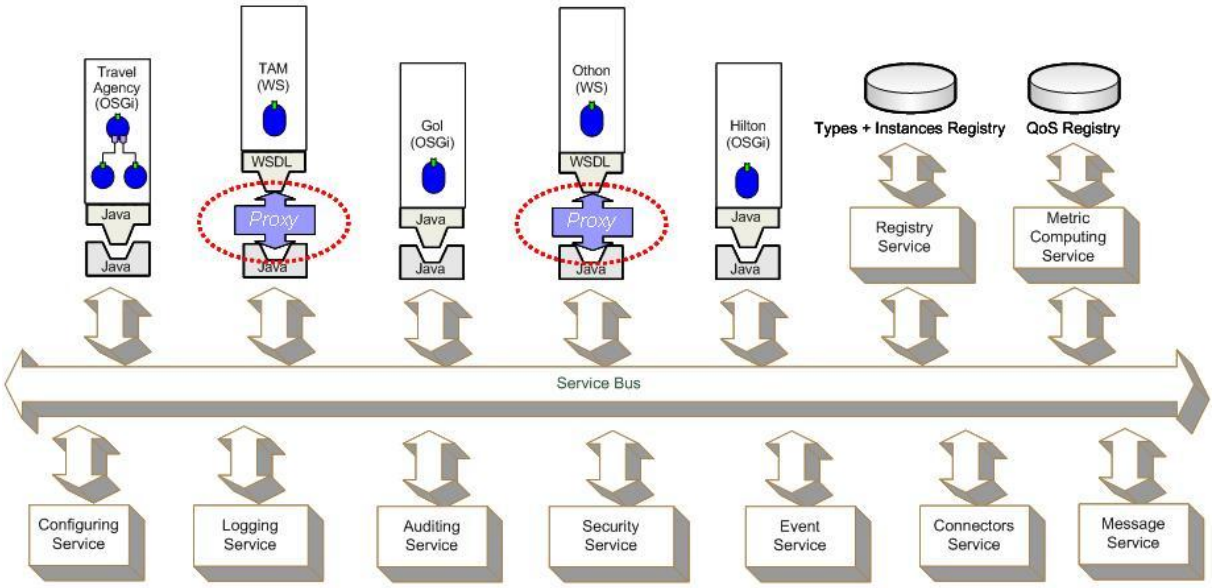
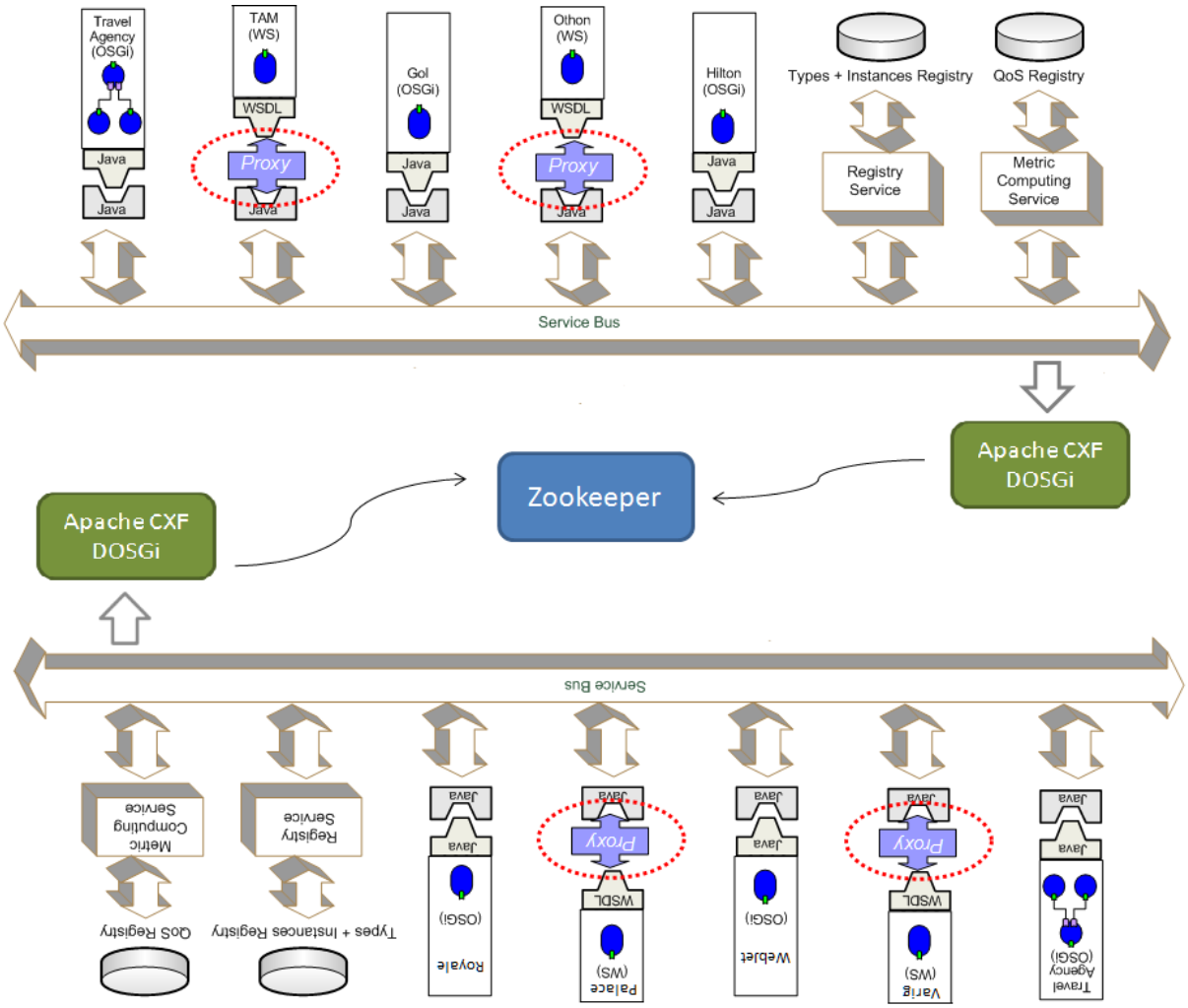


Figura 5-1: Arquitetura do Middleware

Um segundo cenário é descrito na figura 5-2. No qual existe uma rede de computadores e em cada computador existe uma instancia do *middleware* descrito acima. Nesse modelo os serviços tornam-se visíveis para as aplicações instaladas em qualquer plataforma da rede.

Para que isso aconteça entram em prática o *DOSGi* e o *Zookeeper*. O *Zookeeper* tem o papel de organizar o repositório integrado entre as plataformas que estão inseridas na rede e o *DOSGi* tem como principal característica a criação dos *proxies* entre os serviços de plataformas distintas.

Os *Web services* são devidamente registrados no repositório integrado entre as plataformas regido pelo *Zookeeper*. A partir desse repositório central os Web Services se tornam visivéis para ambas as plataformas como se fosse um serviço não-OGSi registrado localmente no *Types/Instances Registry*.



5-2: Arquitetura Remota

O Componente desenvolvido no trabalho descrito a seguir tem como principal funcionalidade o registro de *Web Services* no repositório integrado entre as plataformas distintas para que seja possível a composição entre *Web Services* e Serviços *OSGi*.

Além do cadastro do *Web Service* o componente proverá todo o suporte de comunicação com o *Web Service* gerando as interfaces Java que são utilizadas para requisição de tarefas.

As interfaces serão disponibilizadas em um repositório para que sejam utilizadas pelos clientes que necessitam acessar os *Web Services* cadastrados na plataforma.

5.2 Versões

Nesta seção são citadas quais são as versões de desenvolvimento que foram utilizadas como base para a construção do componente.

- **OSGi Framework**: Apache Felix 2.0.1 [8];
- **DOSGi**: Apache CXF 1.2 [7];
- **ZooKeeper**: Apache Zookeeper Server 3.2.1 [14];
- **IDE'S**: Eclipse Helios [17] NetBeans 6.8 [18];

5.3 Configuração

O Componente é facilmente configurado pelo administrador. Existem dois arquivos de configuração: zookeeper.conf e wsdl2java.conf

- **zookeeper.conf**: Tem a finalidade de conter o endereço e a porta do Servidor ZooKeeper. Exemplo: "172.0.0.1:2801"
- **wsdl2java.conf:** Contém o endereço dos pacotes necessários para o Web Service Axis funcionar corretamente.

5.4 Repositório

A primeira parte do desenvolvimento do componente do middleware foi feita a partir da inclusão de um repositório de Web Services na ferramenta. Pois temos que ter controle sobre quais Web Services estão sendo disponibilizados ao repositório OSGi.

A partir desse repositório é vetada a inclusão de *Web Services* duplicados de forma rápida e transparente. Assim, evita-se o *overhead* de chamada para checagem do *Web Service* no repositório do middleware.

```
public class Service {

    private String name;
    private String wsdl;
    private String type;


    public Service(String newName, String newWsdl, String newType) {
        this.setName(newName);
        this.setWsdl(newWsdl);
        this.setType(newType);
    }
```

Figura 5-3: Objeto *Service*

A figura 5-3 mostra o código do objeto *Service* que será instanciado e inserido no repositório da aplicação.

- **name**: Nome do *Web Service* adicionado.
- **wsdl**: url do wsdl a ser adicionado.
- **type**: Tipo de *Web Service* adicionado.

Os tipos de *Web Service* a serem adicionados são escolhidos a partir das interfaces que os *Web Services* constituem. Nesta versão os tipos de *Web Service* a serem adicionados são pré-definidos no sistema. São eles: AgenciaNoticias, AgenciaBancaria e AgenciaViagens.

5.5 Geração

A geração pode ser divida em 3 momentos: Dados Pré-Conexão, Conexão e Geração de Interfaces.

5.5.1 Dados Pré-Conexão

Após a inserção do *Web Service* no repositório do componente, um novo objeto será criado a partir dos dados recebidos (*WSDL*, nome e *type* do *Web Service*). Este objeto é chamada de ZooKeeperPackage. A figura 5-4 ilustra o código de descrição do ZooKeeperPackage.

```
public class ZooKeeperPackage {


    private String node;
    private String path;


    public ZooKeeperPackage(Service newService) {
        this.setNode(this.makeNode(newService));
        this.setPath(this.makePath(newService));

    }
```

Figura 5-4: Objeto ZooKeeperPackage

O objeto ZooKeeperPackage recebe como parâmetro para sua criação um Service (Service este que foi devidamente registrado no repositório da aplicação). A partir dos dados obtidos do objeto Service o objeto ZooKeeperPackage é criado com os dois atributos descritos abaixo:

- **node**: Arquivo em formato *XML*[15] que contém os dados nos quais a plataforma *OSGi* utiliza para identificar os serviços disponíveis no seu repositório global.
- **path**: Caminho no qual será inserido o *node* no repositório *ZooKeeper*.

Para o maior entendimento do arquivo *node* é necessário uma explicação sobre seu conteúdo. A figura 5-5 mostra o conteúdo do *node* criado para inserção de um *Web Service* no repositório *OSGi*.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint-descriptions xmlns="http://www.osgi.org/xmlns/rsa/v1.0.0">
  <endpoint-description>
    <property name="endpoint.framework.uuid" value="0" />
    <property name="endpoint.id" value="http://localhost:37455/AgenciaNoticias/A
mfAgenciaEJB" />
    <property name="endpoint.package.version.br.ufpe.cin.amf.agencia.AgenciaNoti
cias" value="0.0.0" />
    <property name="endpoint.service.id" value-type="Long" value="0" />
    <property name="interval" value-type="Integer" value="7000" />
    <property name="name" value="AmfAgenciaEJB" />
    <property name="objectClass">
      <array>
        <value>br.ufpe.cin.amf.agencia.AgenciaNoticias</value>
      </array>
    </property>
    <property name="org.apache.cxf.ws.address" value="http://localhost:37455/Age
nciaNoticias/AmfAgenciaEJB" />
    <property name="response.time" value="1" />
    <property name="service.id" value-type="Long" value="0" />
    <property name="service.imported" value="true" />
    <property name="service.imported.configs">
      <array>
        <value>org.apache.cxf.ws</value>
      </array>
    </property>
    <property name="service.intents">
      <array>
        <value>SOAP.1_1</value>
        <value>HTTP</value>
        <value>SOAP</value>
      </array>
    </property>
    <property name="service.pid" value="AmfAgenciaEJB" />
  </endpoint-description>
</endpoint-descriptions>

cZxid = 484
ctime = Wed Jun 30 21:12:24 GMT-03:00 2010
mZxid = 484
mtime = Wed Jun 30 21:12:24 GMT-03:00 2010
pZxid = 484
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0
dataLength = 1412
numChildren = 0
```

5-5: Conteúdo do arquivo node

O preenchimento dos campos que formam o conteúdo do *node* foram indicados pela implementação do Apache CXF DOSGi [7] via inspeção do código fonte. No conteúdo do arquivo vamos destacar alguns itens:

- **endpoint.service.id**: Setado como '0' ou pode ser deixado em branco para serviços que não são OSGi.
- **endpoint.framework.uuid**: Setado como '0' para serviços não OSGi.
- **endpoint.package.version**: Setado como '0.0.0' pois não trataremos versionamento de classe e interfaces no momento.
- **endpoint.id**: Setado como o endereço para o Web Service correspondente, esse campo nos dará a chave única de identificação.

Os demais campos são determinados pelo url do *WSDL*, nome e *type* definidos

pelo *Web Service* que está sendo publicado na plataforma.

5.5.2 Conexão

Após a criação dos dados pré-conexão é iniciada a fase da conexão. Nesse momento o módulo inicializa uma conexão com o repositório global *OSGi* (*ZooKeeper Server)*. A partir desse momento é inserido no servidor no caminho determinador pelo "path" o conteúdo do "node" que habilita o *Web Service* a ser disponibilizado no repositório global *OSGi*.

```
ls /osgi/service_registry/br
[ufpe]
ls /osgi/service_registry/br/ufpe
[cin]
ls /osgi/service_registry/br/ufpe/cin
[amf]
ls /osgi/service_registry/br/ufpe/cin/amf
[agencia]
ls /osgi/service_registry/br/ufpe/cin/amf/agencia
[AgenciaNoticias]
ls /osgi/service_registry/br/ufpe/cin/amf/agencia/AgenciaNoticias
[]
ls /osgi/service_registry/br/ufpe/cin/amf/agencia/AgenciaNoticias
[localhost#37455##AmfAgenciaEJB]
```

5-6: Node criado no ZooKeeper Server

A Figura 5-6 indica a criação do node no ZooKeeper Server. Logo em seguida o *Web Service* poderá ser utilizado por qualquer composição de serviço no middleware desenvolvido.

5.5.3 Geração de Interfaces

Ao concluir as fases de Dados Pré-Conexão e Conexão a próxima fase a ser concluída é a Geração de Interfaces. Nessa fase o componente disponibiliza as classes Java que serão utilizadas para se conectar ao *Web service*.

Utilizando o *Web Service Axis*[16] e a url *WSDL* do *Web Service* que foi inserido no repositório do *middleware* a interface Java que é necessária a conexão com o *Web Service* é gerada e disponibilizada em um repositório a parte.

O Componente tem a funcionalidade de parar um *Web Service* que está

devidamente cadastrado e inserido no repositório OSGi.

5.6 Considerações Finais

Ao longo deste capítulo foi explicado o processo de desenvolvimento do módulo *Web Service Connector* para conectar *Web Services* ao *middleware* com o intuito de disponibilizar serviços de natureza heterogêneas na plataforma desenvolvida.

Através do fluxo da aplicação e de alguns códigos fontes pode-se entender a estratégia utilizada para garantir a conexão de forma confiável sem comprometer o dinamismo utilizado em serviços homogêneos *OSGi*.

6. Conclusão e Trabalhos Futuros

Este trabalho abordou o processo de desenvolvimento de uma ferramenta para simplificar o registro de *Web Services* em repositórios OSGi com o objetivo de tornar possível a utilização de serviços de natureza heterogênea em um middleware baseado em OSGi.

Para definir a estratégia traçada foi dirigido um estudo sobre as tecnologias OSGi, *Web Services* e DOSGi que nos motivou a solução devidamente integrada ao repositório global, *ZooKeeper*.

O uso do módulo de registro de *Web Services* facilita a composição de serviços dentro do *middleware* por incluir mais serviços ao repositório e aumentar a gama de serviços disponíveis na dinâmica OSGi.

Como melhorias futuras para o trabalho existem algumas possibilidades que serão descritas abaixo:

A primeira possibilidade seria a extensão do módulo para cadastro de Web Services registrados em repositórios *Universal Description, Discovery and Integration (*UDDI) [19] para que ao receber apenas a url do repositório UDDI o módulo seria capaz de registrar todos os *Web Services* cadastrados no repositório UDDI.

A Segunda possibilidade seria acoplar ao módulo um serviço de monitoramento dos *Web Services* cadastrados para que se um *Web Service* cadastrado parar de responder ou sair do ar a ferramenta seria informada pelo serviço de monitoramento e removeria o *Web Service* do repositório OSGi.

Por último temos a possibilidade de desenvolver um repositório baseado em QoS para que além de cadastrar o Web Service informações como tempo de resposta, segurança, entre outras sejam disponibilizadas para suprir necessidades dos clientes que utilizarão o serviço oferecido.

Referências

[1] OASIS. *Service Oriented Architecture Reference Model*. 2006. http://docs.

oasis-open.org/soa-rm/v1.0/. [Online; acessado em 04-Setembro-2009].

[2] Papazoglu M. P., Geogakopoulos D. *Service-Oriented Computing.* Communications of the ACM, 46 (10):25-34, October 2003

[3] Erl, T. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, 2005.

[4] OSGi Alliance. OSGi Service Platform: Core Specification, Release 4, Version 4.1.

Technical report, 2007.

[5] W3C. Web Services Architecture. 2004. http://www.w3.org/TR/ws-arch/. [Online; acessado em 29-Junho-2010].

[6] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. 2007. http://www.w3.org/TR/wsdl20/. [Online; acessado em 29-Junho-2010].

[7] Apache CXF: An Open Source Service Framework; http://cxf.apache.org/. [Online; acessado em 23-Março-2010].

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

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

[10] Bartlett Neil, OSGi in Pratice, January 2009.

[11] *Java 2 Security Architecture* Version 1.2, Sun Microsystems, March 2002.

[12] *Zip File Format* O arquivo de formato Zip é definido por java.util.zip package.

[13] OSGi Alliance. OSGi Service Platform: Core Specification, Release 4, Version 4.2.

Technical report, 2007.

[14] Apache ZooKeeper; http://hadoop.apache.org/zookeeper/ [Online; acessado em 29-Junho-2010].

[15] Extensible Markup Language (XML) 1.0 (Fifth Edition); http://www.w3.org/TR/REC-xml/. [Online; acessado em 23-Março-2010].

[16] Web Service Axis; http://ws.apache.org/axis/java/. [Online; acessado em 23-Março-2010].

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

[18] NetBeans IDE; http://netbeans.org/. [Online; acessado em 23-Março-2010].

[19] *Universal Description, Discovery and Integration* (UDDI); http://uddi.xml.org/. [Online; acessado em 23-Março-2010].

[20] Proxy; http://pt.wikipedia.org/wiki/Proxy [Online; acessado em 23-Março-2010].

[21] HTTP - Hypertext Transfer Protocol; http://www.w3.org/Protocols/ [Online; acessado em 23-Março-2010].