RAFAEL KRÜGER TAVARES

UMA ABORDAGEM PARA BUSCA POR WEB SERVICES COM REQUISITOS DE QOS

FLORIANÓPOLIS
2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rafael Krüger Tavares

UMA ABORDAGEM PARA BUSCA POR WEB SERVICES COM REQUISITOS DE QOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

CARLOS BECKER WESTPHALL

Florianópolis, Fevereiro de 2006

UMA ABORDAGEM PARA BUSCA POR WEB SERVICES COM REQUISITOS DE QOS

Rafael Krüger Tavares

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Raul Sidnei Wazlawick

Banca Examinadora

_____

Carlos Becker Westphall

_____

Membro2

_____

Membro 3

_____

Membro 4

AGRADECIMENTOS

Gostaria de agradecer ao Professor Carlos Westphall por ter me aceito como orientando e pelas sugestões e conversas para que o trabalho fosse concluído.

Gostaria de agradecer aos membros da banca pela presença na defesa e pelos comentários colocados na defesa.

Gostaria de agradecer em geral todas as pessoas que de uma forma ou de outra contribuíram para que esta dissertação fosse feita:

Wesley, Rafael Brinhosa, Leonardo, Marcos Assunção e outros que eu não lembro agora.

Por último gostaria de agradecer á minha noiva, Lis Pavin, e à minha família pelo apoio e torcida.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

A2A: *Application to application*

API: *Application Program Interface*

B2B: *Business to Business*

CDATA: *Character Data*

CORBA: *Common Object Request Broker Architecture*

DCOM: *Distributed Component Object Model*

DOM: *Document Object Model*

DTD: *Document Type Definition*

HTML: *Hyper-Text Markup Language*

HTTP: *Hyper-Text Transfer Protocol*

IIOP: *Internet Inter-ORB Protocol*

JSP: *Java Server Pages*

PCDATA: *Parser Character Dara*

PHP: *Hypertext Preprocessor*

QoS: *Quality of Service*

RMI: *Remote Method Invocation*

SAX: *Simple API for XML*

SLA: *Service Level Agreements*

SOAP: *Simple Object Access Protocol*

UDDI: *Universal Description Discovery Integration*

URI: *Uniform Resource Identifier*

URL: *Uniform Resource Locator*

W3C: *World Wide Web Consortium*

WSDL: *Web Services Description Language*

WSLA: *Web Services Level Agreement*

WSOL: *Web Services Offer Language*

XML: *Extensible Markup Language*

XML-RPC: *XML Remote Procedure Call*

XSLT: *Extensible Stylesheet Language Transforms*

RESUMO

O objetivo dos Web Services é resolver problemas de interoperabilidade entre aplicações da Web. Um dos principais problemas recorrente aos Web Services atualmente está no fato de o cliente não conseguir a QoS (quality of service) necessária para receber ou acessar determinado serviço ou aplicação. Esta dissertação apresenta uma especificação para obtenção de QoS entre Web Services. Esta especificação utiliza um broker como módulo intermediário entre cliente, servidor e UDDI, coletando informações de QoS e tomando decisões para que o cliente consiga a qualidade de serviço desejada. Para a definição dos parâmetros de QoS forem criados Esquemas XML. São apresentados o Esquema XML e um estudo de caso para a validação da arquitetura.

ABSTRACT

The goal of Web services is to solve interoperabiliteis problems of Web's applications. One of the main problems recurrent to Web Services is the fact the customer does not obtain the necessary QoS when receiving or access determined service. This dissertation presents an architecture to achieve QoS for Web Services. This architecture uses a broker as intermediate module between client, server and UDDI, collecting QoS information and making selection decisions for clients to achieve the desired QoS. For the QoS parameters definition it was created a XML Schema. This dissertation presents the architecture proposed, the XML Schema and a study case for the proposal's validation.

1. INTRODUÇÃO

1.1 MOTIVAÇÃO

Nas últimas décadas a computação tem evoluído a níveis sem procedentes. Inovações em *software* e *hardware* proporcionaram tecnologias mais úteis e poderosas, como a programação orientada a objetos, computação distribuída, protocolos da Internet e XML (*Extensible Markup Language*). A tecnologia dos Web Services representa um dos próximos estágios em computação distribuída e tem mudado o modo de empresas de TI desenvolverem e disponibilizarem os seus serviços.

O W3C (*World Wide Web Consortium*), define Web Services como sendo uma aplicação identificada por uma URI e cujas interfaces e conexões são capazes de serem definidas, descritas e descobertas por artefatos XML, e suportam interações diretas com outras aplicações usando mensagens baseadas em XML através de protocolos baseados na Internet. [SCHLIMMER, 2002]. Um Web Service pode fornecer a cotação das ações de uma empresa, atuar como um dicionário *online*, permitir que um usuário venda ou compre mercadorias, entre outras funcionalidades.

Diferentemente das outras tecnologias para computação distribuída, os Web Services não são acessados via protocolos específicos a um modelo de objeto como DCOM, RMI ou IIOP. Ao invés disso, são acessados através de protocolos da Web, como HTTP, permitindo interoperabilidade entre aplicações. Isso fez com que os Web Services se tornassem muito populares, tanto no meio acadêmico, como no industrial [PERFECTXML, 2003].

Muitas empresas e companhias oferecem seus serviços sob a forma de Web Services hoje, e muito esforço vem sendo feito para tentar controlar ou melhorar a qualidade do serviço (QoS) final que chega ao cliente. QoS define o termo para qualidade de serviço (*quality of service*) e pode ser representada ou medida através de parâmetros como latência, disponibilidade, confiabilidade entre outros. Hoje, a única informação que um cliente recebe para acessar um determinado Web Service está contida em um arquivo WSDL (*Web Service Description Language)*. O WSDL descreve a localização (URI) e as operações que o Web Service fornece, mas não informa quais são as ofertas de QoS que o

Web Service oferece aos clientes e não permite que os clientes peçam por um determinado parâmetro de QoS para executar um serviço.

A UDDI (*Universal Description, Discovery & Integration*) proporciona um método para descoberta de Web Services. Ele armazena informações de Web Services publicadas pelas próprias companhias e possui links para os documentos WSDL dos Web Services. Esse método para descoberta de Web Services para encontrar os arquivos WSDL também é falho, e pode fazer com que o cliente perca muito tempo para encontrar o serviço que ele deseja. Vários trabalhos foram feitos nessa área, alguns substituindo a UDDI por outro sistema de depósito, outros propondo uma expansão da UDDI. Mas mesmo assim, eles ignoram a possibilidade de um cliente poder definir os seus próprios parâmetros de QoS e solicitá-los ao acessar algum Web Service.

Muito esforço tem sido feito para tentar suprir essas e outras deficiências encontradas em Web Services, como controle de QoS, gerenciamento e SLAs (Acordos de Nível de Serviço). Foram desenvolvidos três projetos por grandes corporações na tentativa de proporcionar QoS de alguma maneira: (i) pela HP o *Web Services Management Framenwork* (WSMF) [CATANIA, 2003]; (ii) pela IBM o *Web Service Level Agreemen*t, (WSLA) [LUDWIG1 2003] uma linguagem para criar e validar SLA's entre Web Services; (iii) o *Web Services Offer Language* (WSOL) [TOSIC, 2003] uma linguagem para adicionar parâmetros de QoS nos arquivos que descrevem a funcionalidade dos Web Services. Entretanto, ainda assim não é trivial conseguir uma configuração de sistema que entregue a QoS de um Web Service usando as ferramentas mencionadas acima [CHEN, 2003]. Pode-se dizer que empregar QoS em Web Services é um desafio crítico por causa da natureza dinâmica e imprevisível das aplicações e do tráfego na Internet. Aplicações de negócios com características e requerimentos muito diferentes competem entre si por recursos usados para fornecer Web Services [YUAN, 2003].

Qualidade de Serviço também é importante para diferenciar serviços que oferecem funcionalidades semelhantes. A QoS fornecida por esses Web Services pode ser usada como um critério decisivo para que o cliente escolha um serviço [TIAN, 2004]. Além disso, os requisitos de QoS para se acessar um determinado serviço podem mudar dependendo de vários fatores, como tipo do hardware que o cliente está usando, o modo

como ele deseja acessar o serviço (um serviço sendo oferecido em tempo real, ou somente entre intervalos de tempo), e o sistema e aplicações que o cliente está usando, como aplicações multimídia.

1.2 OBJETIVO GERAL

Esta dissertação tem como objetivo geral a definição e validação de uma arquitetura baseada em um *broker* para proporcionar acesso a Web Services com QoS e a definição de um Esquema XML para os parâmetros de QoS. O *broker* estende o modelo de arquitetura clássica dos padrões Web Services (*server, client, Publisher)* permitindo integração com a arquitetura clássica e o Esquema XML define a estrutura de como os parâmetros de QoS devem ser definidos pelos clientes e fornecedores.

1.3 OBJETIVOS ESPECÍFICOS

Para que o objetivo geral seja alcançado os seguintes objetivos específicos devem ser realizados:

- Proporcionar um material teórico sobre Web Services e discutir com base em trabalhos correlatos o uso, vantagens e desvantagens de QoS *brokers* para solucionar o problema de entrega e garantia de QoS em Web Services.
- Fazer uma análise sobre as vantagens e limitações que a arquitetura oferece.
- Analisar os benefícios e limitações de se expandir os parâmetros de QoS a qualquer tipo de métrica ou dado. Os parâmetros de QoS inicialmente considerados são: disponibilidade, tempo de resposta, delay e perda de pacotes.
- A partir dos conhecimentos adquiridos projetar e implementar a arquitetura.
- Desenvolver os Esquemas XML que serão usados para definir a estrutura de como os parâmetros de QoS devem ser definidos.

1.4 JUSTIFICATIVA

Esta dissertação propõe uma maneira de tornar a UDDI mais prático, permitindo ao cliente determinar os parâmetros de QoS e seus valores para executar o serviço e ao fornecedor dos serviços especificar quais parâmetros de QoS que ele oferece para os clientes. Como nem sempre uma busca do cliente por serviços na UDDI irá resultar em sucesso, e os clientes terão cada vez mais necessidade de QoS ao executar Web Services, é apropriado o emprego de parâmetros de QoS para uma melhor seleção dos serviços e para tentar fazer com que o cliente consiga o que ele procura já na primeira consulta. Também é importante manter uma monitoração depois de o serviço ter sido iniciado, para saber se algum dos valores de QoS anteriormente prometidos pelo servidor foi violado.

Nesta dissertação, é proposta uma arquitetura de Web Services para o fornecimento de QoS através de um *broker,* que irá coletar a informação de QoS e selecionar o Web Service com a melhor QoS para o cliente. O *broker* é responsável também pela decisão de qual serviço utilizar, após uma busca na UDDI, e pela negociação de QoS entre cliente e servidor. Vários trabalhos existem na área, alguns tentando adicionar mecanismos de obtenção de QoS dentro da UDDI ou propondo extensões, outros, como o que é proposto nessa dissertação, propõe uma entidade para obtenção de QoS através de um intermediador (*broker)*, que por não estar vinculado a UDDI, pode ser usado para buscas em outros tipos de diretórios para busca de Web Services. Neste trabalho é usado a UDDI por ser o serviço de diretório mais comum e como ferramenta para validação da arquitetura, mas a arquitetura pode utilizar outro tipo de diretório.

Cada Web Service pode oferecer parâmetros de QoS de acordo com a importância que esse parâmetro possui para que o serviço seja executado com qualidade. Assim, cada serviço pode ter vários parâmetros de QoS diferentes. Para lidar com diferentes tipos de parâmetros de QoS, e fazer os componentes da arquitetura entenderem o que cada parâmetro significa é proposto um Esquema XML definindo como os parâmetros de QoS devem ser especificados. Dessa forma, tanto clientes quanto fornecedores podem especificar os parâmetros mais significativos para eles na hora de oferecer/requisitar serviços com QoS.

A monitoração dos parâmetros de Qos também é importante, pois caso haja uma degradação muito grande da QoS durante a execução de um serviço, o cliente poderá saber e cancelar o serviço, podendo escolher outro serviço. Assim, a arquitetura propõe um mecanismo para a monitoração da QoS durante a execução dos serviços. Como os parâmetros de QoS poderão ser muito diferentes, de acordo com o que os clientes e os fornecedores definirem, o mecanismo de QoS monitorará somente os parâmetros de QoS mais recorrentes, e cuja monitoração faz mais sentido, como tempo de resposta e disponibilidade.

1.5 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado da seguinte forma: no capítulo 2 são apresentados os conceitos de Web Services e XML. No capítulo 3 é apresentada uma discussão sobre o uso de QoS em Web Services, os principais aspectos, benefícios, que tipos de parâmetros usar e o que a comunidade acadêmica tem feito para lidar com o problema da falta de QoS em Web Services. No capítulo 4 é apresentada a arquitetura proposta. No capítulo 5 são apresentados os Esquemas XML para a definição dos parâmetros de QoS e a implementação da arquitetura. No capítulo 6 é apresentado um estudo de caso contendo um exemplo para a validação da arquitetura. Por fim, no capítulo 7 são apresentadas as conclusões e os trabalhos futuros.

2. WEB SERVICES E XML

2.1 INTRODUÇÃO

Neste capítulo são apresentadas as principais tecnologias para o desenvolvimento de aplicações Web existentes, a linguagem XML (eX*tensible Markup Language*) [BRAY 2000] e os Web Services propriamente ditos. Ambas tecnologias surgiram com o propósito de resolver problemas de interoperabilidade entre aplicações da Web, sendo que o XML foi a linguagem precursora e que deu origem a outros padrões do W3C, como o SOAP (*Simple Object Accsses Protocol*) [GUDJIN 2003]. Na sessão 3.2 são apresentados os principais conceitos relacionados a linguagem XML e na seção 3.3 são apresentados os padrões e componentes que fazem parte da tecnologia dos Web Services.

2.2 XML

A linguagem XML foi desenvolvida inicialmente para corrigir algumas limitações da linguagem de marcação HTML e oferecer suporte a interoperabilidade a várias aplicações Web. O XML, assim como o HTML são padrões baseados em SGML, que especificam que a estrutura e o formato dos documentos sejam determinados através de elementos de marcação [ROSSET 2004].

O HTML é uma linguagem de marcação que utiliza um conjunto de marcadores chamados de *tags* que são fixos, permitindo apenas determinar a estrutura e apresentação do documento, sem a possibilidade de definir novas *tags*, o que é essencial em aplicações Web. O padrão XML permite a criação de *tags* para determinar novos tipos de dados, permitindo descrever o conteúdo de um documento. Podem ser criadas quantas *tags* forem necessárias, garantindo extensibilidade ao padrão. O XML ainda permite que seus marcadores sejam definidos em infinitos níveis de profundidade garantindo flexibilidade. Desse modo, o XML tornou-se o padrão na representação de dados em aplicações e nos Web Services, sendo uma recomendação do W3C desde Fevereiro de 1998 [FILIPPIN 2004].

Ao invés de uma arquitetura para computação distribuída baseada em objetos e que se torna incompatível com outras arquiteturas como o CORBA e o DCOM, o XML é

transmitido por protocolos de comunicação baseados em texto, o que permite compatibilidade com qualquer tipo de arquitetura, sistema e *hardware*. Isso permitiu que uma grande variedade de tecnologias, padrões, linguagens e recomendações baseadas em XML, como os sistemas de definição de tipos (DTDs) e o XML *schema*, a linguagem para apresentação dos documentos XSLT e o XML-RPC (XML *Remote Procedure Call*) que oferecem alternativas processáveis por qualquer servidor Web [COYLE 2001]. Nas próximas seções será mostrada a sintaxe dos documentos XML e como determinar novas estruturas de dados usando DTDs e XML *schema*.

2.2.1 Sintaxe XML

Para permitir a criação de vários tipos de dados em um documento XML, são usados elementos para definir esses tipos de dados. Elementos são os blocos de construção básicos de XML. O XML utiliza *tags* para definir elementos em documentos. As *tags* são delimitadas da mesma forma que no HTML, através dos sinais menor-que (<) e maior que (>), sendo que a *tag* que corresponde ao início do elemento é definida como <nome do elemento> e a *tag* que corresponde ao final do elemento é definida como </nome do elemento>. Cada elemento pode dar origem a uma cadeia de elemento, formando uma árvore de elementos do documento XML.

```
<?xml version="1.0"  encoding="UTF-8?>
<User login='rafa'>
   <nome> Rafael Tavares </nome>
   <cidade> Florianópolis</cidade>
</User>
```

Figura 2.1. Exemplo de documento XML

A Figura 2.1 ilustra um exemplo de um documento XML. O cabeçalho do documento sempre possui a versão em que o documento XML foi escrito. O elemento raiz em um documento é chamado de elemento do documento (<User>), e dá origem à árvore de elementos do documento.

Os elementos dos documentos XML podem possuir atributos que podem representar “qualidades” dos elementos. O atributo é definido logo depois do elemento ser definido e o seu valor deve ser um *string*.

Quando um documento XML for analisado por um *parser,* se for encontrada uma construção XML que não seja bem-formada (sintaticamente correta), o *parser* deve gerar uma mensagem de erro "fatal". Um documento XML é considerado bem-formado se:

- A sintaxe obedece às especificações XML;
- Os elementos formam uma árvore hierárquica, com um único nó raiz;
- Não há referências a entidades externas, a não ser que seja fornecido um DTD.

Os *parsers* XML são classificados em: voltados para o evento ou baseados em árvore. O *parser* voltado para o evento executa uma chamada para cada classe de dados XML (elemento, caracteres, instruções de processamento, comentários). O *parser* não mantém os dados depois da análise sintática, mesmo a estrutura da árvore. Isto permite que poucos recursos de sistema sejam usados na análise. Em 2000, membros da lista de e-mail XML-DEV desenvolveram uma interface padrão para análise de documentos XML, o SAX (Simple API for XML), que foi editado por David Megginson. O SAX possui implementações em diversas linguagens, como Java, Pearl e Python. Os *parsers* que seguem a abordagem baseado em árvore geralmente obedecem ao DOM (Document Object Model) da W3C. O DOM é uma plataforma (e linguagem) que permite a manipulação de documentos estruturados em árvore. O DOM constrói a árvore do documento e a armazena na memória, só então a aplicação pode acessar o DOM através de uma API [MARTIN 2001].

2.2.2 *Document Type Definition* (DTD) e XML *schema*

DTDs definem a estrutura de um documento XML, - quais elementos, atributos, etc. são permitidos no documento. O uso de DTDs para determinar a estrutura do documento é altamente recomendado pois permite que um documento XML consiga comunicar as regras sintáticas que estão definidas de forma que outro documento possa interpretar a estrutura das regras. Em um ambiente *bussiness-to-bussiness* ou em Web Services, onde a troca de documentos XML é freqüente, é essencial que ambas as partes possam interpretar a estrutura dos documentos XML trocados.

A Figura 2.2 ilustra um documento especificado por DTD. A declaração do DTD pode ser colocada no prólogo do documento, antes da declaração do elemento raiz, ou pode estar em outro documento. Existem quatro tipos de declarações diferentes usadas no DTD. As mais importantes são a construção **ELEMENT,** para declarar elementos, e a **ATTLIST,** para atributos. O DTD utiliza alguns símbolos para representar quando um elemento contém múltiplos sub-elementos. O símbolo "*" representa que o elemento pode aparecer qualquer número de vezes (zero ou mais), no caso do exemplo mostrado na figura 2.2, o catálogo pode ser composto por zero ou mais elementos CD. O símbolo "+" representa que o elemento deve aparecer pelo menos uma vez (um ou mais), e o símbolo "?" representa que o elemento é deve aparecer zero ou uma vez.. Um elemento que não possui elementos deve ser declarado como sendo PCDATA (*parsed character data*), enquanto CDATA é usado para declarar atributos que contenham somente caracteres texto.

```
<?xml version = "1.0" encoding = "UTF-8?>
<!DOCTYPE catalogo [
  <!ELEMENT catalogo (CD)*>
  <!ELEMENT CD (nome, artista)>
  <!ATTLIST CD codigo CDATA #REQUIRED
  <!ELEMENT nome (PCDATA)>
  <!ELEMENT artista (PCDATA)>
]>
```

Figura 2.2. Exemplo de um documento DTD

Os DTDs possuem algumas falhas de flexibilidade, não sendo possível definir a estrutura dos documentos XML de uma forma completa. Por exemplo, não é possível fazer buscas no documento ou transformá-lo em uma representação diferente (como HTML), pois um DTD não é um documento XML. Também não se pode colocar praticamente nenhuma restrição usando DTDs, pois ele não diz nada a respeito do texto contido dentro dos elementos e não diferencia tipos dos atributos (*integer*, *string*, etc.), sendo que todos os tipos de dados são definidos como um conjunto de caracteres (PCDATA).

O W3C reconheceu as falhas que o DTD possui e desenvolveu um padrão para corrigir as limitações do DTD, chamado de XML *Schema* [BEECH 2001]. Em 2001, várias outras companhias lançaram seus padrões para substituir os DTDs, mas o padrão recomendado pelo W3C para representar a estrutura dos documentos é o XML *schema*.

O XML *schema* é totalmente baseado na sintaxe XML, ao contrário do DTD, baseado em *Extended Backus-Naur Form* (EBNF). Assim é possível fazer buscas no documento e manipulá-lo da forma como for necessário. O XML *schema* permite que os elementos possam ser de diferentes tipos e não somente seqüências de caracteres. Assim, utilizando XML *schema* é possível a definição de tipos de dados como inteiros, datas, horas, e números de ponto flutuante.

Para fazer uma declaração de *schema* em um documento XML é necessário uma *tag* de declaração da raiz do esquema, como mostrado na linha dois da Figura 2.3. Os elementos filhos de um elemento s*chema* possuem apenas três tipos de elementos: **ElementType** para definir elementos, **AttributeType** para declarar atributos e **description** para descrição do elemento *schema***.** No exemplo abaixo, é criado um elemento complexo, chamado "pessoa", que é composta dos elementos "nome", "rua" e "telefone".

```
<?xml version = "1.0" ?>

<Schema xmlns = "urn:schemas-microsoft-com:xml-data">

        <ElementType name = "pessoa" content ="eltOnly " order = "seq"
                model = "closed">
                <element type = "nome" />
                <element type = "rua"/>
                <element type = "telefone"minOccurs = "0" maxOccurs = "*"/>
        </ElementType>

        <ElementType nome = "nome" content = "textOnly" model = "closed" />
        <ElementType rua = "rua" content = "textOnly"
                model = "closed" />
        <ElementType telefone = "telefone" content = "textOnly" model = "closed">
                <AttributeType name = "localizacao" default = "casa"/>
                <attribute type = "localizacao"/>
        <ElementType/>
</Schema>
```

Figura 2.3. Exemplo de um documento XML *Schema*

O elemento **ElementType** possui ainda atributos que podem ser especificados quando um elemento for definido. O atributo "name" é sempre obrigatório e define o nome do elemento. O atributo "content" define o tipo de dados válidos para o elemento, **empty** (elemento vazio), **eltOnly** (somente elementos), **textOnly** (somente textos) e **mixed** (elementos e texto). O atributo "model" especifica se elementos que não foram

definidos no esquema são permitidos. O atributo “dt:type” define os tipos de dados dos elementos, como *boolean, real, integer*, etc.

O elemento **element** faz referência a um elemento já definido por um **ElementType**. Isso permite que um elemento seja declarado somente uma vez, podendo ser referenciado múltiplas vezes no documento. No exemplo, os elementos “nome”, “rua” e “telefone” são referenciados como parte do elemento “pessoa”, sendo definidos logo em seguida, através do elemento **ElementType**. Os atributos “minOccurs” e “maxOccurs” definem a quantidade de vezes que um **element** pode aparecer.

Os elementos definidos podem possuir atributos, da mesma forma que nos DTDs. A forma de definir os atributos é parecida com a forma de definir elementos, **AttributeType** define um atributo, enquanto **attribute** referencia um atributo já definido.

Assim, através de um documento XML *schema* pode ser definido um conjunto de regras, uma espécie de gramática para determinar como os documentos XML que se referem ao esquema devem ser definidos. O *parser* irá analisar o documento XML e, caso o documento esteja de acordo com o esquema, o documento será validado.

2.3 *WEB SERVICES*

Como dito na introdução, os Web Services são em geral o resultado de diversos padrões abertos para comunicação via Web. Eles foram desenvolvidos com o objetivo principal de fazer uma aplicação Web poder se comunicar e trocar informações com outra, evitando falha na comunicação devido a arquiteturas ou sistemas diferentes. Eles surgiram depois da experiência da indústria com problemas de interoperabilidade, que conduziram ao desenvolvimento de padrões abertos, em um esforço para tornar viável a comunicação entre diferentes plataformas [DEITEL 2002].

Os Web Services estendem o paradigma de programação orientada a objeto. Cada Web Service é um objeto que pode ser reusado ou incorporado em outras aplicações. Desta forma, os Web Services incentivam uma programação em módulos, transformando uma rede como a Internet em uma enorme biblioteca de componentes programáveis, podendo reduzir o esforço requerido para implementar certos tipos de sistemas. Essa característica permite a construção de sistemas com acoplamento fraco, que auxilia a manter, expandir e gerar novos sistemas.

As três tecnologias principais que constituem os Web Services são o protocolo SOAP [GUDJIN 2003], a linguagem baseada em XML WSDL [CHRISTENSEN 2001], e o serviço de busca pelos serviços UDDI [BELLWOOD 2002]. Essas tecnologias foram desenvolvidas baseadas em uma Arquitetura Orientada à Serviços (SOA) [CERAMI 2002]. Neste tipo de arquitetura existem três entidades principais: o cliente, o fornecedor e um diretório (registro) que armazena descrições dos serviços. O SOAP é protocolo de comunicação entre as entidades, o WSDL é a linguagem para descrição dos serviços e a UDDI é o diretório onde os arquivos WSDL são disponibilizados para busca. A Figura 2.4 ilustra as relações entre, cliente, fornecedor do serviço e UDDI. Além das tecnologias principais, existem outras tecnologias promissoras que são derivadas dos Web Services, como os *Grid Services* [FOSTER 2002], *mobile (m)-Services* [MAAMAR 2004] e *Semantic Web enabled Web Services* [DAML 2003], o que prova que as tecnologias de Web Services tornaram-se um dos mais importantes paradigmas de computação distribuída. A seguir são colocadas algumas considerações sobre as três principais tecnologias de Web Services.

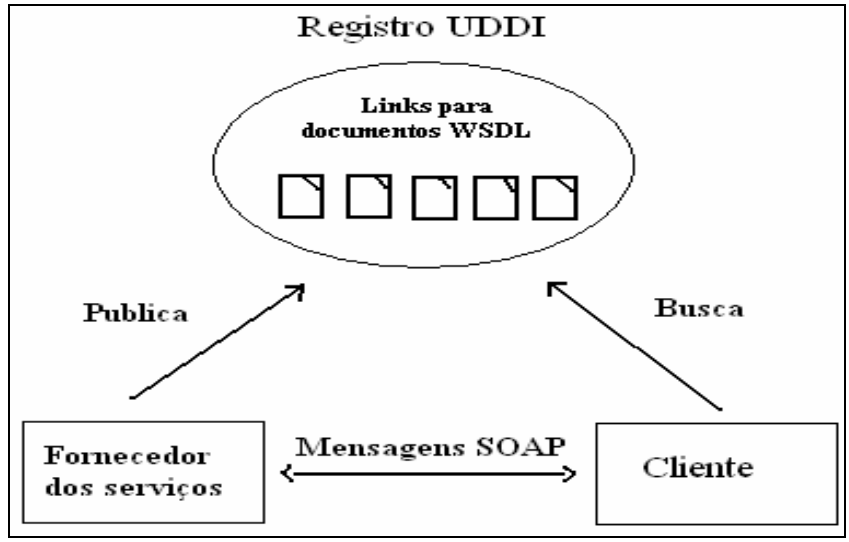

Figura 2.4. Funcionamento da publicação e busca de serviços na UDDI

2.3.1 SOAP

O SOAP é o protocolo para mensagens XML padronizado pelo W3C. Ele é um protocolo baseado em XML que permite a comunicação entre aplicações pela Internet usando documentos XML, que são chamados de mensagens SOAP. O padrão SOAP é

definido por um documento XML *schema* para troca de mensagens, encodificação de dados e algumas convenções para representar requisições e respostas para RPC (*Remote Procedure Call*).

Ele é compatível com qualquer modelo de objeto, sendo independente de plataforma e linguagem de programação. Suporta transporte através de qualquer protocolo, sendo mais usado com HTTP. Suporta qualquer método de codificação de dados. Isso faz com que aplicações baseadas em SOAP possam enviar qualquer tipo de informação.

A mensagem SOAP possui um envelope que descreve seu conteúdo, destino e requerimentos de processamento da mensagem. Ele consiste de duas partes:

* Header (opcional) – Fornece instruções de processamento para as aplicações que recebem a mensagem. Pode incorporar informações de roteamento e pode ser usado para construir outros protocolos mais complexos, estendendo a mensagem para propósitos como autenticação, gerenciamento de transação e pagamento.

* Body – Contém os dados específicos da aplicação do receptor pretendido.

A Figura 2.5 mostra um exemplo de uma mensagem SOAP requisitando um Web Service para consulta de livros através do método “getTituloLivro”. Neste exemplo a mensagem SOAP está sendo transmitida sobre HTTP, assim as primeiras quatro linhas se referem ao cabeçalho do HTTP. Esse exemplo não necessita do *header* na mensagem SOAP, pois a requisição não precisa de informações adicionais de roteamento ou processamento de transação. As cinco primeiras linhas contêm o cabeçalho POST da mensagem HTTP, sendo que a linha cinco define o conteúdo da mensagem SOAP através da declaração do SOAPAction. No resto da mensagem é definido o envelope, que é o elemento raiz. As linhas 10-14 contêm as declarações dos *namespaces* para identificar elementos na mensagem SOAP. No elemento *Body*, declarado como elemento filho, são colocadas as instruções que devem ser executadas. Cada ação a ser executada é definida como um elemento “filho” do *Body*. O elemento “getTituloLivro” é criado para acessar o método a ser invocado. Dentro do elemento são declarados os parâmetros esperados para os métodos invocados, no caso deste exemplo, é criado o elemento ISBN, que contém a identificação do livro.

```
POST /ccx/TituloLivro HTTP/1.0
Content-Length: 541
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: "capeconnect:TituloLivroService#getTituloLivro

<?xml version="1.0" encoding="UTF-8"?>

<SOAP-ENV:Envelope
        SOAP-ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">

        <SOAP-ENV:Body>

                <ns1:getTituloLivro
                        xmlns:ns1="capeconnect:TituloLivro:TituloLivroService">

                        <ISBN xsi:type="xsd:string"> 5486454536</ISBN>

                </ns1:getTituloLivro>

        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 2.5. Exemplo de Mensagem SOAP

2.3.2 WSDL

O WSDL é uma linguagem XML para descrever a funcionalidade de um Web Service. O WSDL surgiu como um método universal para a descrição de Web Services. Antes de o WSDL ser adotado como método universal para a descrição de serviços, os Web Services eram descritos por padrões proprietários, e as descrições dos serviços eram inconsistentes e incompatíveis. A Microsoft e a IBM uniram-se para criar uma linguagem padrão que descreve os serviços oferecidos pelos Web Services, e em março de 2001 submeteram a versão 1.1, que foi a usada nesta dissertação, para o W3C.

Os documentos WSDL especificam as capacidades do serviço, sua localização na Web e as instruções de como acessá-lo. Ele define a estrutura da mensagem que o Web Service envia e recebe. Sua principal utilidade é tornar um padrão comum para que os desenvolvedores de Web Services possam descrever a funcionalidade do Web Service. Assim pode-se criar depósitos de documentos WSDL na Internet, fazendo-se uma analogia às listas telefônicas. Assim, quando alguém quer procurar por algum tipo de

serviço, ele pode acessar o depósito de documento WSDL e buscar o serviço que ele deseja.

O elemento raiz para um documento WSDL é *definitions*. WSDL é um vocabulário XML e como tal deve ser estruturado de modo a ser bem-formado. Todos os outros elementos que contém a informação do Web Service são considerados "filhos". Os elementos filhos são *types, message, portType, binding* e *service*. O elemento *types* especifica os tipos de dados usados nos outros elementos. O elemento *message* representa uma mensagem. Como quando um cliente solicita um serviço ele geralmente espera uma resposta, na maioria das vezes existem dois elementos mensagem, um para representar a mensagem do cliente e outro a do servidor. O elemento *message* possui como atributo o nome da mensagem, enquanto o elemento *part* define um nome e o tipo de dado de um parâmetro da mensagem. O elemento *message* somente descreve as mensagens, sem especificar exatamente como associar as mensagens com a operação desejada. O elemento *portType* é o responsável por isso. Ele possui elementos *operation* os quais definem quem é o emissor e o receptor da mensagem através dos elementos *input* e *output*. O elemento *binding* descreve como são trocadas as mensagens entre cliente e Web Service. Ele possui outro elemento *binding* para especificar o protocolo através do qual o cliente acessa o Web Service e um elemento *operation* que define as informações de acesso do Web Service. Finalizando, o elemento *service* descreve um Web Service como uma coleção de portas, cada uma descrita por um elemento *port*. O elemento *port* possui atributos que descrevem o nome da porta e do Web Service além do endereço (URL) para acesso do Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl :definitions
                tns:TituloLivro="http://www.TítuloLivro.br/titulolivro.wsdl">

                <wsdl :message name="TituloLivro_Request">
                    <wsdl :part name = "parametro" type=" xsd:string"/>
                </wsdl :message>

                <wsdl :message name="TituloLivro_Reponse"/>
                    <wsdl :part name = "response" type=" xsd:string"/>
                </wsdl :message>

                <wsdl :portType name="TituloLivroService"/>
                    <wsdl :operation name="getTituloLivro" parameterOrder ="parametro">
                            <wsdl :input name="getTituloLivro"
                                    message="tns:TituloLivro_Requst" />
                            <wsdl :output name="getTituloLivro"
                                    message="tns: TituloLivro_Response"/>
                    </wsdl :operation>
                </wsdl :portType>

                <wsdl :binding name="TituloLivroBinding"
                    type="tns:TituloLivroService">
                     <soap :binding style="rpc"
                            transport="http://schemas.xmlsoap.org/soap/http" />
                     <wsdl: operation name="getTituloLivro">
                        <soap: operation soapAction=" " style="rpc" />
                            <wsdl :input name="getTituloLivro">
                                <soap :body use="encoded"
                                encodingStyle ="http://schemas.xmlsoap.org/soap/encoding/ "
                                namespace = "urn :TituloLivroService"   />
                            </wsdl :input>
                            <wsdl :output name="getTituloLivro">
                                <soap :body use="encoded"
                                encodingStyle=" http://schemas.xmlsoap.org/soap/encoding/ "
                                namespace="urn :TituloLivroService"   />
                             </wsdl :output>
                    </wsdl: :operation>
                </wsdl :binding>

                <service name="TituloLivro">
                    <wsdl :port name="TituloLivroService"
                    binding="tns:TituloLivroBinding">
                    <soap :address
                            location="Http://localhost:8080/soap/servlet/" />
                    </wsdl: port>
                </wsdl :service>
</wsdl :definitions>
```

Figura 2.6. Exemplo de documento WSDL

A Figura 2.6 ilustra um exemplo de um documento WSDL que descreve o Web Service “TituloLivro’, cuja função é associar o identificador de um livro, passado como

parâmetro e retornar o título do livro correspondente. O método implementado para invocar o serviço é getTituloService. O método precisa ser informado no elemento *operation* para informar o destino da mensagem de requisição (através do elemento *input*) e o emissor da mensagem de resposta (através do elemento *output*). Se mais de um método for invocado para o Web Service, então será necessário criar um elemento *operation* por método. A mensagens de entrada (*input*) para a operação é "TituloLivro_Request" e a de saída (*output*) "TituloLivro_Response". Este exemplo é uma simplificação do exemplo encontrado em [DEITEL 2002].

Como pode sernotado no exemplo, não é mostrada nenhuma informação a respeito da QoS que o Web Service oferece ao cliente. Desse modo, quando o cliente for fazer uma requisição ao serviço ele não terá idéia do que esperar a respeito de QoS.

2.3.3 UDDI

A UDDI define um método padrão para publicar e descobrir informações sobre Web Services através da Internet. Sua especificação consiste de um documento XML *schema* e uma descrição da especificação da uma API. Juntas elas formam o modelo de informação básico que fornece a habilidade de publicar informação sobre Web Services. A idéia principal da UDDI é proporcionar registros centralizados para facilitar o armazenamento, descoberta e troca de informações por empresas e seus serviços na Web.

Uma das principais justificativas para o desenvolvimento da UDDI é a necessidade de divulgação dos serviços para os clientes. Esse processo foi chamado de d*iscovery* (descoberta), que é o processo de localizar Web Services através de registros.

A UDDI é implementada e disponibilizada para acesso público na Web através do UDDI *Bussiness Registry* (UBR). A organização que hospeda um UBR é chamada de nó operador. No nó operador é que os documentos WSDL são armazenados e disponibilizados para busca. O número de UBRs no mundo é limitado, mas mesmo que o número venha a crescer, o seu modelo de publicação permite visibiliade global, ou seja, se uma companhia publicar seus serviços em um UBR, eles poderão ser procurados em todos.

A API da UDDI é dividida em duas partes, a *Inquiry* API e a *Publication* API. A *Inquiry* API permite que um programa (pode ser um UBR) encontre Web Services através de uma informação geral ou de uma forma mais refinada. A *publication* API

permite que uma companhia publique informações gerais e de seus serviços em um nó da UDDI.

A UDDI disponibilizou uma arquitetura para que um cliente possa procurar por um Web Service qualquer e selecionar o que for melhor para ele. Primeiramente, o fornecedor do serviço publica informações sobre Web Service que ele está disponibilizando. O cliente usa a UDDI para buscar os registros do tipo de serviço que ele deseja usar. São retornados para o cliente ponteiros para arquivos WSDL, que especificam a funcionalidade do Web Service. O cliente escolhe o melhor serviço de acordo com as suas necessidades e se comunica com o servidor para acessar o serviço via mensagens SOAP.

Uma das principais limitações da UDDI é que ela pode fazer o cliente perder muito tempo, ou até mesmo não encontrar o tipo de serviço que ele deseja. Como o cliente não tem conhecimento da QoS necessária para o serviço, ele pode encontrar serviços com uma QoS muito aquém do que ele deseja, e então precisará voltar para a UDDI procurar por outro serviço. Por exemplo, um cliente deseja um serviço com uma determinada largura de banda. Mas no documento WSDL que o cliente recebeu ao procurar um Web Service na UDDI não especifica qual a largura de banda que o provedor do Web Service fornece. O cliente escolhe um Web Service ao acaso, que ele não sabe se irá satisfazer o valor da largura de banda que ele deseja. Se a largura de banda for insuficiente ele terá que cancelar o serviço e procurar por outro. A UDDI não proporciona um limite de tempo para os documentos WSDL que são armazenados, assim, um determinado Web Service pode ter mudado de endereço ou ter sido desativado e mesmo assim a UDDI irá relaciona-los como resultado de uma busca para o cliente, o que pode resultar em problema para o cliente.

2.4 RESUMO DO CAPÍTULO

Este capítulo apresentou os conceitos básicos associados às tecnologias que permitem o uso e desenvolvimento dos Web Services. Foi abordado como o XML proporcionou o crescimento das tecnologias que proporcionam interoperabilidade e que resultou em vários padrões para o uso dos Web Services. Foram apresentados e discutidos exemplos mostrando a funcionalidade dos documentos XML *schema*, SOAP e

WSDL, os quais são essenciais para o desenvolvimento dos Web Services e que serão utilizados futuramente.

Foi mostrado que as tecnologias de Web Services necessitam de mecanismos de ofertas e controle de QoS, e que a WSDL e a UDDI não proporcionam meios para a declaração da QoS oferecida pelo Web Service e o cliente não consegue expressar a QoS que ele deseja para acessar determinado Web Service.

3. ESTADO DA ARTE

Neste capítulo é analisado o estado da arte em fornecimento de QoS para Web Services. Será abordado mais a fundo a necessidade de fornecimento e gerenciamento de QoS relacionando com alguns dos principais trabalhos realizados. O capítulo está dividido em duas seções principais. Na seção 3.1 é mostrado o desenvolvimento da importância da QoS em Web Services, e como os parâmetros de QoS são abordados. Na seção 3.2 são analisados os principais trabalhos relativos ao emprego de QoS em Web Services, as vantagens e desvantagens de sua proposta e o que iria mudar se eles fossem empregados.

3.1. QOS EM WEB SERVICES

Desde o princípio da Web, a busca por QoS (*quality of service*) sempre foi objeto de pesquisa e aprimoramento. No começo um dos principais desafios era manter um simples servidor disponível durante determinado período de tempo, ou tentar aumentar a taxa de transmissão o suficiente para receber alguns dados HTML. Com o tempo foram criados parâmetros para definição de métricas de qualidade de serviço e métodos para gerenciar os recursos da rede e monitorar os parâmetros de QoS, assegurando que os elementos da rede aplicassem o tratamento correto ao tráfego.

Atualmente existem muitas ferramentas na área de gerenciamento de QoS que ligam as necessidades dos clientes com as ofertas de QoS dos fornecedores de serviços. QoS é hoje uma grande área no estudo de redes de computadores, computação em tempo real e desenvolvimento de *middlewares.*

O grande crescimento obtido pela Internet nos últimos anos e a crescente popularidade dos Web Services fez com que as aplicações disponibilizadas pelos Web Services necessitem oferecer QoS para os usuários. O grande número de pessoas que usam Web Services pode causar um *overhead* no processamento do servidor, fazendo as requisições demorarem a serem processadas e com que a resposta que será entregue ao cliente atrase. Da mesma forma, o grande número de Web Services disponíveis hoje faz com que existam muitos Web Services que possuem a mesma funcionalidade, e a

diferença de qualidade entre dois Web Services poderá ser decidida pela garantia de QoS que cada um disponibilizar. Com isso, a QoS em Web Services tornou-se um assunto que vem ganhando uma relevância científica cada vez maior, como pode ser visto em [MENASCE 2002] e [MANI 2002].

O grande problema de se empregar QoS em um ambiente Web é a forma como você irá definir e medir os parâmetros de QoS. A natureza imprevisível das aplicações e do tráfego da Internet faz com que a entrega de QoS torne-se um desafio crítico e significante para a maioria dos Web Services. Isso faz com que os parâmetros de QoS mais difundidos e que são empregados em vários tipos de redes como atraso, *jitter* (variação do atraso) e perda de pacotes não possam ser usados da mesma forma, pois é necessário saber exatamente por onde o tráfego de dados flui. Um determinado Web Service, que deveria estar fornecendo os serviços com a mesma QoS a todos os clientes, pode estar fornecendo um serviço para o cliente (A) com os parâmetros atraso e *jitter* abaixo do prometido e para o cliente (B) com os parâmetros atraso e *jitter* dentro do esperado. Se o problema for no *link* de comunicação do cliente (A) (e provavelmente será), então não há o que se possa fazer para o Web Service fornecer a QoS desejada, a não ser o cliente tentar uma conexão melhor. Outro problema para a definição de parâmetros de QoS em Web Services é a quantidade de aplicações diferentes existentes. Quanto mais diferentes as aplicações, mais diferentes podem-se tornar os parâmetros de QoS. Assim, alguns parâmetros de QoS que podem ser importantes para um Web Service não têm a menor utilidade em outro.

Para a definição de um parâmetro de QoS que um Web Service irá oferecer deve ser feito um estudo sobre a funcionalidade do Web Service e quais os serviços que ele oferecerá aos clientes. Após essa análise os parâmetros precisam ser definidos de uma forma simples, de modo que o cliente possa entendê-los. Em métricas orientadas à performance, como tempo de resposta ou atraso, é essencial definir o ponto onde essas métricas serão medidas. Considerando tempo de resposta, por exemplo, ela pode ser medida no cliente, em um ponto da rede entre o cliente e o Web Service, ou na aplicação que fornece o serviço.

A definição de um parâmetro de QoS corresponde ao estabelecimento de uma ontologia entre um fornecedor de serviços e seus clientes. Em Ciência da Computação,

ontologia significa um modelo de dados que representa um domínio e é usado para tomar decisões sobre os objetos que pertencem ao domínio. Uma ontologia pode ser definida através de dois métodos [LUDWIG2 2003]:

(a) Através de uma definição forte de termos (parâmetros) e das relações que há entre eles. Esse método foi aperfeiçoado para ser usado para Web Services pela linguagem DAML+OIL [CONNOLLY 2001]. Os parâmetros definidos resultam em um conjunto fixo de termos bem definidos.

(b) Através de ontologias construtivas. Depois de ter sido criado um conjunto de parâmetros como termos bem definidos (como em (a)), é criado um conjunto de operadores de composição e novos termos (parâmetros) podem ser definidos através da composição de dois ou mais parâmetros que foram definidos anteriormente. Um exemplo seria a definição da métrica tempo de resposta médio, que poderia ser definido aplicando-se o operador *média* para a *seleção* dos tempos de resposta de todas as requisições na última hora. Neste exemplo, o termo bem definido é tempo de resposta, enquanto os operadores de composição são *média* e *seleção*.

As soluções para resolver o problema da definição de parâmetros de QoS devem levar em consideração a heterogeneidade das implementações e a natureza dos Web Services de serem interoperáveis e voltados para interações B2B (*bussiness-to-bussiness*) e A2A (*application-to-application*). Assim, a melhor forma para descrever os parâmetros de QoS é através do XML, que mantém as características dos Web Services.

Hoje, com os clientes requisitando serviços através da Web, tem surgido uma nova perspectiva para oferecer QoS que deve ser levada em consideração por fornecedores e usuários de Web Services: os clientes também desejam definir parâmetros de QoS quando solicitam um determinado Web Service, e não só definir parâmetros, mas também receber garantias de estes parâmetros estão sendo cumpridos. Com usuários requisitando serviços pela Web, ele pode requisitar um serviço de uma empresa desconhecida, com um servidor em uma rede que ele não sabe como funciona e

se a qualidade do serviço poderá suprir as suas necessidades. Considerando esse cenário de fornecimento de QoS em Web Services, onde um cliente pode pedir por determinado parâmetro de QoS e seu valor, forma-se um ambiente onde o cliente pode não conseguir entender os parâmetros que o fornecedor oferece, e que o fornecedor não entende os requisitos de QoS do cliente. É preciso utilizar um meio comum para a definição dos parâmetros de QoS de forma que tanto o cliente quanto o fornecedor do serviço possam definir parâmetros de QoS de uma forma que possa ser entendida por ambos.

Um outro conceito interessante e que foi utilizado para a definição de múltiplos níveis de QoS para um Web Service neste trabalho é o de classes de serviço. Quando um Web Service atende várias requisições em paralelo ele fornece diferentes níveis de QoS devido às características e necessidades dos clientes, que geralmente são diferentes. Uma maneira de fazer essa diferenciação entre os valores dos parâmetros de QoS que podem ser desejados é permitindo ao cliente escolher entre diversas classes de serviço diferentes. Classes de serviço distintas apresentam pequenas variações do serviço e da QoS fornecidos por um Web Service [TOSIC 2004]. Dessa forma, um Web Service pode oferecer os seus serviços com garantias de QoS diferentes, dependendo da classe de serviço que o cliente escolher. Classes de serviço se referem ao mesmo arquivo WSDL, mas diferem em restrições (requisitos e garantias de QoS). Escolhendo fornecer algumas classes de serviço, um Web Service pode aumentar o número de clientes, já que com diferentes classes de serviço ele poderá atingir clientes com diferentes necessidades de QoS. Uma boa analogia é através dos pacotes de TV a cabo oferecidos. Você pode escolher diversos pacotes diferentes nos quais você pode escolher os canais que você realmente gosta e pagar apenas por eles, sem a necessidade de pagar por canais pelos quais você não se interessa.

Utilizando diferentes classes de serviços, os fornecedores poderão aumentar as possibilidades de preencher os requisitos de um determinado cliente sem prejudicar os requisitos de QoS de outro. Considerando um Web Service cujas ofertas de QoS possuam sempre os mesmos valores e métricas. Se um novo cliente requisitar um Web Service, e ele já estiver sobrecarregado, não será possível manter a QoS que o Web Service oferece, e haverá uma degradação geral de performance, podendo inclusive, ocorrer uma degradação para uma qualidade abaixo do desejado pelo cliente. É interessante notar que

não importa que as necessidades dos clientes sejam diferentes, pois se a QoS que o Web Service oferece satisfazer os desejos dos clientes, eles receberão a QoS que o Web Service oferece, e que pode ser muito maior que o desejo do cliente. Caso haja uma degradação, ela não se dará nos clientes que podem ter degradação, e sim distribuída ente todos os clientes utilizando o Web Service. Utilizando classes de serviço, os clientes poderão receber a QoS de acordo com a classe que expresse melhor os seus desejos. Os clientes que desejarem uma QoS baixa, poderão se enquadrar em uma classe de serviço que oferece uma QoS correspondente. Dessa forma, os recursos de um fornecedor irão ser utilizados de um modo que faz as chances de eles se esgotarem devido a um grande número de clientes utilizando o serviço diminuam.

Neste trabalho, através dos esquemas XML que serão apresentados, será possível definir qualquer parâmetro de QoS de uma maneira simples e de fácil entendimento para o cliente. O cliente poderá procurar por Web Services com os parâmetros de QoS definidos por ele, e entenderá o que significa determinado parâmetro de QoS que um Web Service oferece, pois todos os parâmetros serão definidos através do XML, mantendo o objetivo, a natureza e as características dos Web Services.

3.2. TRABALHOS CORRELATOS

Como discutido na seção anterior, prover um Web Service com a capacidade de fornecer QoS é um processo muito complicado e que depende de um grande estudo. Desde que foram padronizadas as tecnologias para desenvolvimento de Web Services o assunto tem sido discutido. Existem diversas formas para tornar um Web Service apto a fornecer QoS para os clientes e para que Web Services possam negociar a QoS com os clientes. Os trabalhos podem ser classificados em quatro áreas diferentes: arquiteturas baseadas em *brokers*, arquitetura de agentes e Web Semântica, extensões de diretórios de registros – como UDDI –, e linguagens baseadas em XML. A seguir serão mostrados os principais trabalhos em cada área.

3.2.1 QoS *Broker* para descoberta de Web Services

A proposta de utilizar um *broker* consiste de empregar um intermediador para realizar a procura por Web Services. O *broker* realiza a busca por Web Services com a

QoS requisitada pelo cliente em algum diretório de registros. Ele parte do princípio que não é necessária uma etapa para negociação de QoS para estabelecer a QoS que o cliente irá receber. Se um cliente requisitar um Web Service cujo fornecedor oferece uma QoS que satisfaz os seus desejos, então a QoS que o cliente irá receber deverá ser a que o fornecedor oferece. Esse método é uma maneira simples de adquirir um Web Service com QoS, já que uma etapa de negociação entre cliente e fornecedor na maioria das vezes não é proveitosa. O tempo gasto em uma etapa de negociação pode se tornar muito alto, o que torna o acesso ao Web Service mais demorado, mesmo que seja com a QoS ideal. Não havendo negociação também força o provedor do serviço a ter um controle maior dos recursos, já que os Web Services disponíveis devem sempre ser acessados com a mesma QoS. Outra grande vantagem de *brokers* é que eles não estão sujeitos a um tipo específico de diretórios de armazenamento de registros. O diretório mais difundido e popular é a UDDI; no entanto ela não é um padrão reconhecido e um *broker* possui a flexibilidade para ser usado com qualquer diretório.

Em [TIAN, 2004] é proposto um *broker* que procura em registros UDDI por Web Services com o objetivo requisitado pelo cliente e escolhe dentre os encontrados o que possui a QoS solicitada ao *broker*, proporcionando integração de QoS em Web Services. O *broker* proposto funciona de uma forma muito simples e nenhum detalhe é dado sobre a sua arquitetura. A definição de parâmetros de QoS e estratégias para seleção de serviços não são abordadas, sendo o foco dirigido para a definição de ontologias para o mapeamento de parâmetros de QoS e provar a praticidade do protótipo. A principal contribuição deste trabalho é a idéia de utilizar documentos Esquemas XML para definição de parâmetros de QoS, e foi o trabalho que motivou a desenvolver um *broker* com parâmetros de QoS integrados entre cliente e fornecedores com estratégias para a seleção de serviço bem definidas.

Já o *broker* apresentado em [CHEN, 2003] implementa três parâmetros de QoS (tempo de resposta, custo do serviço e largura de banda) que podem ser solicitados pelo cliente. Uma demonstração da performance também é apresentada, sendo provado que a entrega de serviço com QoS é ideal para Web Services multimídia. A principal desvantagem é que a arquitetura não permite a definição de outros parâmetros de QoS,

estando o cliente restrito apenas aos quatro parâmetros definidos pela arquitetura. O grupo de trabalho deste projeto, da Universidade da Califórnia, continuou o desenvolvimento do trabalho estudando o impacto de dois algoritmos de alocação de recursos analisando a performance e a instabilidade de ambos algoritmos [YU 2004]. Nota-se que existe uma preocupação apenas com o desempenho da arquitetura, sendo que o cliente poderá escolher somente as métricas de quatro parâmetros de QoS quando for utilizar o *broker*.

Em [ZENG 2004] é apresentada uma plataforma de *middleware* composta por três entidades, entre elas um *broker*. O trabalho desenvolvido concentra-se na seleção de serviços para Web Service compostos. Um módulo da plataforma apresentada é responsável por acessar o *broker* toda vez que um Web Service composto é iniciado. A função do *broker* então é fazer a busca no diretório para procurar por Web Services candidatos. A partir dos Web Services que forem selecionados como candidatos é feita a estratégia para seleção de serviço. Entretanto, o trabalho não possui uma extensão para parâmetros de QoS, i.e., quando um plano para a seleção de serviço for feita, serão analisados sempre os mesmos parâmetros de QoS para definir quais os serviços serão utilizados para realizar o serviço composto.

3.2.2 Arquitetura de agentes e Web Semântica

Agentes e Web semântica têm sido empregados com o objetivo de prover a Web de inteligência, mesmo que pequena, para resolver problemas de descoberta de Web Services com suporte a QoS. Muitos trabalhos propõem estratégias de negociação de QoS, onde agentes representando o cliente e o servidor tentam chegar a um acordo da QoS que será fornecida ao cliente depois de analisar os recursos disponíveis no servidor no momento da requisição. O trabalho de [YUAN, 2003] é um trabalho usando arquitetura de agentes. Ele propõe o uso de agentes sem utilizar a Web Semântica, pois segundo o autor, ainda vai levar algum tempo até que os padrões definidos pela Web Semântica sejam aprovados e utilizados. O trabalho propõe o uso de agentes dotados de uma inteligência simples nos servidores dos Web Services, que seriam chamados de WISE Web Servers. Os agentes podem ser empregados somente no servidor, somente no cliente ou em ambos, o que fornece flexibilidade, pois mesmo que um servidor possua os

agentes instalados ele pode receber requisições de clientes normais, o mesmo acontecendo com os clientes que possuam agentes instalados. Como o objetivo era prover uma inteligência simples, foram considerados os parâmetros de QoS mais populares e simples de entender e medir.

Em [SYCARA 2004] é proposta uma arquitetura de *broker* para descoberta de Web Services semânticos sem suporte a QoS. É usada a OWL-S (*Web Services Language Ontology*), uma linguagem para descrever ontologias para Web Services semânticos, com o objetivo de descrever o protocolo de interação entre cliente-fornecedor. Como na Web Semântica supõe-se que exista um certo nível de inteligência, o cliente é desenvolvido como um agente de software, permitindo assim, a qualquer hardware acessar um Web Service. O *broker* executa o protocolo, fazendo a intermediação entre cliente e fornecedor de uma forma inteligente, tentando completar algumas informações que não foram passadas pelo cliente quando ele fez a requisição do serviço e traduzindo diferentes formas sintáticas de requisições e respostas para uma forma comum que todos os parceiros possam entender. O mecanismo de descoberta do *broker,* apesar de não possuir um mecanismo para expressar parâmetros de QoS, permite que ele associe alguns requisitos do cliente com as propagandas dos fornecedores, e através de uma inteligência implementada um OWL-S, escolha o melhor serviço.

Em [PARK 2003] assim como em [ZENG 2004], também é construído um *middleware* para descoberta de Web Services. Só que no trabalho de [PARK 2003] o *middleware* é construído através de agentes, e o processo de descoberta é baseado na Web Semântica e não em padrões XML. Nesse *middleware* também não é possível expressar desejos de QoS em acessar determinado Web Service. A arquitetura apresentada não é flexível o suficiente, pois é compatível somente com a UDDI, sendo que existem outros tipos de diretórios. Os agentes são empregados somente com o objetivo de fornecer uma descrição melhor do serviço, não possuindo uma estratégia de seleção de serviços, ou um mecanismo de inteligência que justificasse seu uso.

Em [DAY 2004] são usados modelos semânticos baseados nas linguagens RDF (*Resource Description* Framework) e OWL (*Web Ontology Language*) para construir um cliente automatizado. O trabalho cria uma espécie de UDDI com informações de QoS que são chamadas de “QoS Fórum”. O QoS Fórum contém informações sobre a QoS que os

clientes receberam ao invocar os serviços. Assim, quando o cliente vai requisitar um serviço ele procura no QoS Fórum por um serviço que satisfaça suas necessidades. Com os modelos semânticos utilizados é possível ter um processo automatizado de seleção do melhor serviço. O modelo de procura de Web Services do cliente é testado com dois modelos, uma através de seleção de serviço dinâmica e outro através de classificação e seleção com Bayes nativo.

Fica claro então, que arquiteturas baseadas em agentes e Web Semântica mantém outro foco. Apesar de serem trabalhos relevantes e que lidam com uma tecnologia que ainda não é usada, e que ainda causa muitas discussões sobre a implantação e uso que é a Web Semântica, esses trabalhos têm um foco diferente em relação aos parâmetros de QoS do que este trabalho. As arquiteturas não se importam muito com fornecimento de QoS através de parâmetros que podem ser declarados, requisitados e medidos. As arquiteturas baseadas em agentes concentram-se em algoritmos e modelos de inteligência voltados para descoberta e seleção de serviço.

3.2.3 Extensões de diretórios de registros

Como o próprio nome diz, extensões de diretórios de registros se propõem a melhorar um diretório, adicionando capacidades de uma busca inteligente, ou fornecimento de QoS. Como, apesar de a UDDI ser o diretório mais conhecido, ainda não existe um modelo de diretório padrão, uma extensão de diretório fica restrita somente a ele, e caso um outro diretório venha definitivamente a ser padronizado, as extensões feitas se perdem. Mesmo assim, existem dois trabalhos desenvolvidos para a UDDI que poderiam ser aplicados, possibilitando uma melhora imediata nas procuras por Web Services. Em [SHAIKHALI 2003] é proposta a incorporação de *Blue pages* na UDDI. As *Blue pages* informam ao cliente os recursos necessários para acessar determinado Web Service, e fornecem um mecanismo automático para atualização dos documentos na UDDI quando os serviços forem modificados. Esse trabalho disponibiliza os parâmetros de QoS que o Web Service oferece na UDDI, porém não faz a seleção de serviço e nem verifica se o servidor irá conseguir suprir os requisitos do cliente quando invocar o serviço. Já em [ZHOU 2004] não são feitas modificações na API da UDDI. A extensão da UDDI é feita através de servidores chamados UX (UDDI *eXtension*). Esses servidores

estão localizados em redes com domínios diferentes e conectados através de conexões de alta velocidade. O principal objetivo é receber *feedbacks* da QoS que o cliente recebeu ao acessar determinado Web Service, assim, quando um outro cliente for acessar o serviço, os servidores UX podem prever a QoS que o cliente poderá receber. A previsão ainda analisa o modelo da rede do cliente para ter uma previsão melhor de QoS. Apesar de possuir uma maneira confiável de medir os parâmetros de QoS que são oferecidos pelos UX *servers*, eles são restritos, e muitas vezes podem não satisfazer os requisitos do cliente. O modelo de descoberta de Web Services torna-se muito próximo de um *broker*, sendo que os UX *servers* podem funcionar como se fossem um só servidor.

Os trabalhos que se destinam a estender a UDDI podem disponibilizar boas alternativas para contornar os problemas da UDDI. A idéia de fornecer informações sobre a QoS de um serviço na UDDI é útil, mas o cliente ainda perderá tempo para escolher o serviço que ele irá acessar. O ideal seria adicionar um mecanismo de seleção para escolher os melhores serviços para o cliente, mas ainda assim a UDDI não teria como monitorar a QoS durante o recebimento do serviço.

3.2.4 Linguagens baseadas em XML

As linguagens baseadas em XML desenvolvidas servem como ferramentas para criar novas tecnologias em Web Services. Depois da padronização do XML *schema*, do WSDL e do SOAP, surgiram muitas outras linguagens derivadas que não foram padronizadas e que podem ajudar a fornecer e gerenciar a QoS em Web Services.

Em [LUDWIG1 2003] é descrito o WSLA *Framework* da IBM. Ele permite que se faça uma especificação de SLA flexível e um *framework* para monitoração com o foco em Web Services. É baseado em um Esquema XML e permite a parceiros (cliente e fornecedor) definirem garantias de QoS em serviços eletrônicos e processos para monitorá-los. As informações que um SLA feito pela linguagem WSLA contém são os parceiros que estão envolvidos, a descrição do serviço e os parâmetros observáveis e as obrigações que devem ser impostas nos parâmetros do SLA. É uma ferramenta muito útil, visto que SLAs entre organizações são usados em todas as áreas de serviços de tecnologias, desde que os parceiros tenham como trocar informação sobre os parâmetros de QoS.

Já a linguagem WSOL [TOSIC 2003] apresenta uma especificação formal para classes de serviço, vários tipos de restrições (QoS, direitos de acesso) e gerenciamento. A linguagem é uma extensão, ou complemento da WSDL, sendo que a WSOL é compatível com WSDL. Através da WSOL é possível informar as ofertas de QoS que o fornecedor dispõe ao cliente e definir várias classes de ofertas, ou seja, o mesmo serviço sendo oferecido com diferentes níveis de QoS. Para a verificação dos documentos WSOL foi criado um *parser*. A linguagem WSOL fornece um poderoso complemento ao WSDL. Com a possibilidade de fornecer informações sobre a QoS do serviço no mesmo local onde é realizada a procura por Web Services, ganha-se uma ferramenta muito útil para desenvolver parâmetros de QoS e arquiteturas que auxiliam a descoberta de Web Services.

Hoje, tanto empresas quanto o meio acadêmico se preocupam em melhorar ou adicionar QoS em Web Services. Isto fica claro através dos *frameworks* e padrões que as empresas estão lançando e dos trabalhos anteriores, que mostram diferentes preocupações envolvendo Web Services e QoS. O presente trabalho se diferencia por definir uma arquitetura simples e sem o emprego de agentes para a negociação de QoS. Além disso, também é apresentado um modelo de documento XML para padronizar os diferentes parâmetros de QoS que o cliente pode desejar e que o fornecedor se comprometa a entregar, de modo que ambos entendam o que cada parâmetro significa.

3.3. RESUMO DO CAPÍTULO

Neste capítulo foram mostrados os rumos que a adição de QoS em Web Service está tomando. Foi discutido como QoS tem se tornado importante em Web Services, como os parâmetros são definidos, e a dificuldade que se tem em definir diversos tipos de parâmetros de QoS para os clientes. Além disso foram mostrados e discutidos alguns dos principais trabalhos desenvolvidos na área de descoberta e fornecimento de QoS em Web Services.

4. A ARQUITETURA PROPOSTA

Neste capítulo é apresentada a arquitetura para fornecer QoS em Web Services. Nas próximas subseções serão apresentadas as entidades da arquitetura. Na seção 4.1 será apresentada uma visão geral da arquitetura. Na seção 4.2 será apresentado o cliente. A arquitetura adiciona uma entidade a mais na arquitetura dos Web Services, o *broker* que é descrito detalhadamente na seção 4.3. O servidor, que é a entidade responsável por disponibilizar os serviços para o cliente, é descrito na seção 4.4. Algumas outras tecnologias desenvolvidas por terceiros foram usadas na arquitetura, e são descritas na seção 4.5. As conclusões do capítulo são apresentadas na seção 4.6.

4.1. VISÃO GERAL

Como citado no capítulo 2, a UDDI possui algumas limitações. Entre as principais limitações está a falta de qualquer informação sobre a QoS dos Web Services e o fato que o cliente não pode expressar seus desejos de QoS para escolher um determinado serviço. O objetivo geral da arquitetura proposta nesta dissertação é resolver essas limitações acrescentando informações sobre a QoS que o provedor do Web Service disponibiliza. Além de informações sobre a QoS, através da arquitetura o cliente poderá escolher os parâmetros de QoS e seus valores para acessar um Web Service. O cliente ainda poderá monitorar em tempo real os parâmetros de QoS, durante a execução de um Web Service. A arquitetura adiciona um *broker*, que serve como módulo intermediário entre o cliente, a UDDI e o provedor do Web Service. O *broker* é a entidade responsável por armazenar as informações de QoS dos Web Services e fazer a seleção de serviço entre os Web Services encontrados. A figura 4.1 ilustra a arquitetura, os componentes são descritos a seguir.

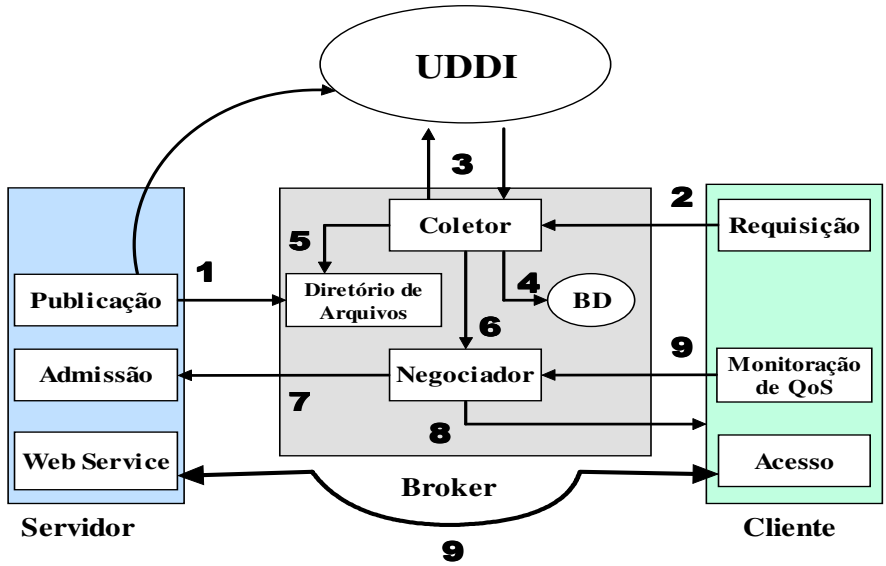

Figura 4.1. Visão geral da arquitetura

4.2. SERVIDOR

O Servidor é a entidade que possui os serviços disponibilizados para a Web, geralmente chamada de provedor do serviço. Cada Web Service disponibilizado pelo Servidor deve possuir um documento XML que descreve os parâmetros de QoS que ele fornece ao cliente.

O Servidor utiliza uma forma de publicação similar à da UDDI. Assim, para os parâmetros de QoS se tornarem de conhecimento público eles devem ser publicados no *broker*. Quando a publicação é feita, o *broker* recebe o documento XML contendo a descrição dos parâmetros de QoS, o nome do arquivo XML, o nome do Web Service e a URI. O arquivo XML é armazenado no Diretório de Arquivos do *broker* e os outros dados são armazenados no banco de dados do *broker*. A Figura 4.2 mostra o Diagrama de Seqüência UML para a publicação dos documentos XML.

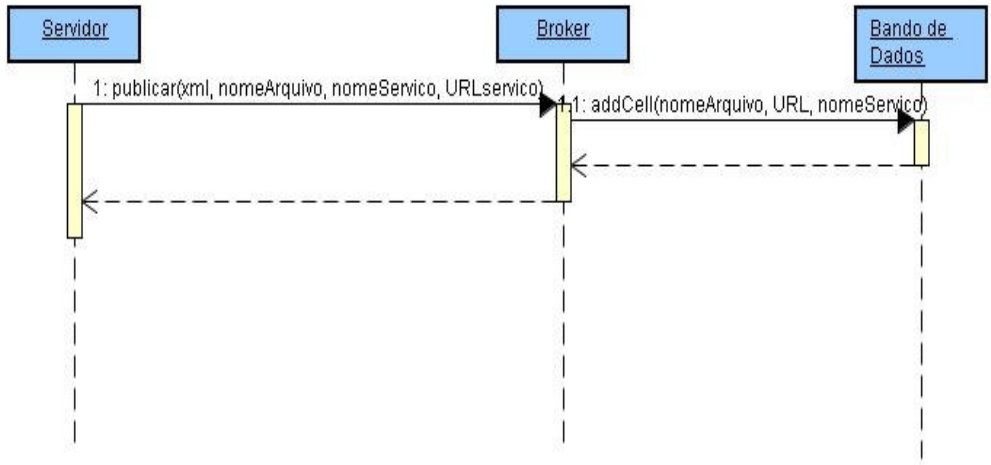

Figura 4.2. Diagrama de Seqüência UML para a operação publicar do Servidor

Esse método de divulgação das informações de QoS foi utilizado pois apesar de ele fazer com que o mesmo serviço seja publicado duas vezes (uma na UDDI e uma na arquitetura) ele pode ter um desempenho melhor do que se fosse empregado um método de divulgação de parâmetros em tempo real, como o feito em [CHEN, 2003]. Nesse método, para cada Web Service que for retornado da busca na UDDI, o provedor do serviço é consultado para informar os parâmetros de QoS ao *broker*. No método apresentado no presente trabalho o acesso aos parâmetros de QoS é feito de forma local, o que permite um melhor desempenho caso muitos Web Services sejam retornados da UDDI.

Outra maneira de acessar a informação de QoS dos Web Services seria criar uma extensão da UDDI como em [SHAIKHALI 2003]. Dessa forma a informação de QoS seria publicada quando o Web Service fosse publicar o serviço na UDDI. A UDDI é utilizada nesta dissertação apenas como um *publisher* comum, podendo ser substituída por outra tecnologia. Outro motivo que levou a não informar os parâmetros de QoS na publicação na UDDI é que fazendo uma extensão da UDDI fica-se preso às restrições que a API da UDDI impõe. Colocar o modelo de informação de QoS no *broker* torna mais flexível o desenvolvimento dos documentos XML para a descrição dos parâmetros de QoS.

Após receber a requisição do negociador, o servidor checa se ele pode garantir a QoS que o cliente está solicitando. Isso é feito a partir o número de serviços que ele está

fornecendo no momento. Após subtrair o que cada serviço consome pelo total de recursos disponíveis, pode ser conhecido o total de recursos livres. Se o servidor puder garantir a QoS, ele envia a notificação para o negociador, que enviará ao cliente e irá fazer a reserva dos recursos necessários para que o cliente receba o serviço. O cliente tem acesso somente aos recursos previamente reservados pelo servidor. Em seguida o cliente invocará o serviço através de uma mensagem SOAP. Caso o servidor não consiga garantir a QoS que o cliente deseja, o negociador receberá uma mensagem de recusa e tentará encontrar outro Web Service.

A dissertação usa a definição de classes de serviços definida em [TOSIC 2004]. Cada Web Service pode possuir uma classe de serviço diferente. Um exemplo seria um serviço A dividido em 3 classes de serviços, Bronze, Prata e Ouro, sendo que cada uma tem um preço diferente. Quando o coletor acessa o Diretório de Arquivos para acessar a informação de QoS do Web Service ele armazena as informações no banco de dados, e o negociador acessa as informações do cliente. Se o cliente tiver colocado preço como um dos parâmetros de QoS, o negociador verificará as diferentes classes que o Web Service oferece em busca da classe de serviço compatível com o desejo do cliente. Se o preço solicitado pelo cliente for maior que o preço estipulado pelo servidor para o serviço classe Ouro, e o os outros parâmetros de QoS atenderem aos outros requisitos do cliente, este receberá uma notificação para usar a classe Ouro do serviço A, caso contrário, o cliente receberá a notificação para usar a classe de serviço correspondente aos preço que o cliente concorda em pagar.

4.3. CLIENTE

O cliente foi projetado apenas como sendo uma entidade que deseja acessar algum Web Service (podendo ser uma pessoa através de uma GUI ou um outro Web Service) e que através de uma mensagem XML informe o *broker* sobre o tipo de serviço e os parâmetros de QoS que ele deseja. Como o *broker* é um Web Service, o cliente enviará uma mensagem SOAP para o *broker* encapsulando os dados de QoS, que seguem o modelo definido por um Esquema XML. O Esquema XML é apresentado no capítulo 6. Para que o cliente possa definir os parâmetros de QoS que ele deseja de uma forma rápida e eficiente, ele possui um editor de parâmetros de QoS. Esse editor permite que apenas os

valores dos parâmetros padrões sejam colocados, e fornece todos os campos necessários para a criação de um novo parâmetro. Quando isso ocorre, o editor adiciona ao documento XML as *tags* referentes aos dados do cliente. Para mais detalhes, veja o capítulo 5.

Após definir os parâmetros de QoS e seus valores, o cliente envia uma requisição para o *broker* contendo as palavras-chaves para o tipo de Web Service e os requisitos de QoS que ele deseja receber. Os parâmetros de QoS podem ser escolhidos de acordo com o que o usuário achar necessário, e podem variar dependendo das características do Web Service desejado. A figura 4.3 mostra o diagrama de seqüência em UML para o cliente fazer uma consulta ao *broker*.

O *broker*, após receber a mensagem do cliente faz uma análise para verificar se existe algum Web Service que combine com as descrições fornecidas. O cliente então pode receber dois tipos de mensagens; uma mensagem informando os resultados encontrados, ou uma mensagem avisando que nenhum Web Service pôde ser encontrado. Caso o cliente receba a mensagem com os Web Services encontrados, ele poderá selecionar o serviço que ele deseja. Para o caso de vários Web Services serem listados na resposta, o *broker* calcula qual Web Service oferece a melhor QoS e o recomenda ao cliente, economizando tempo para o cliente na hora da escolha. Apesar do cálculo de melhor serviço, a escolha do cliente é livre, sendo que ele pode escolher o Web Service preferido, em detrimento do melhor. Após a escolha o cliente envia uma mensagem SOAP para o servidor para obter acesso ao serviço.

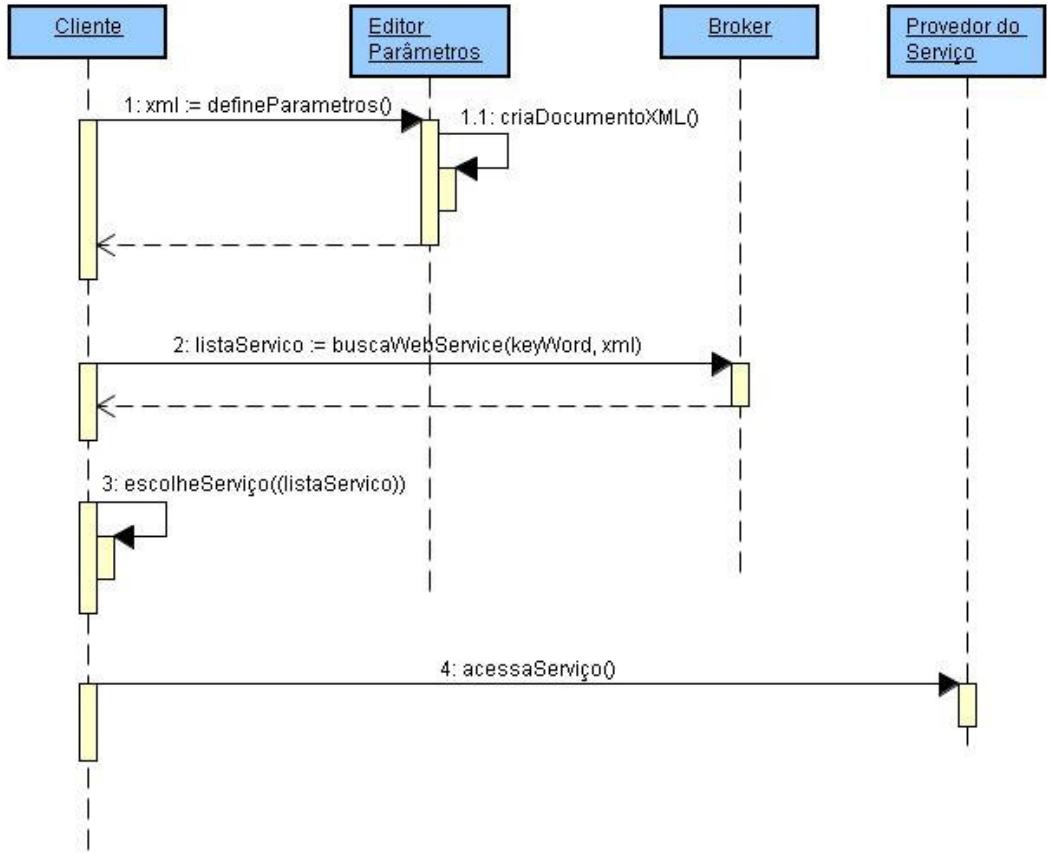

Figura 4.3 Diagrama de Seqüência UML para a operação de busca do cliente.

O cliente possui um módulo para monitorar a QoS em tempo real durante o fornecimento do serviço para detectar alguma degradação da QoS. O principal objetivo da monitoração é alertar o cliente caso o Web Service sofra uma degradação de QoS abaixo dos valores estabelecidos pelo cliente. Se a QoS fornecida for abaixo do desejado, o cliente poderá solicitar o cancelamento do serviço. Quando o cancelamento for solicitado, se o valor de algum parâmetro de QoS estiver abaixo do esperado, o cliente avisa o *broker* que o provedor do Web Service não cumpriu com os valores de QoS acordados. O *broker* então, seleciona o Web Service com a melhor QoS ente os outros Web Services da lista que foi enviada ao cliente. A Figura 4.4 mostra o Diagrama de Seqüência em UML para a monitoração do serviço

Com base na monitoração de QoS o cliente irá qualificar o Web Services depois de utilizá-lo. Caso o cliente tenha cancelado o fornecimento do serviço, ele enviará ao *broker* uma mensagem de qualificação negativa, mas caso o cliente tiver recebido o

serviço dentro dos valores de QoS estipulados ele enviará uma classificação positiva. Deste modo, estabelece-se uma classificação por reputação para os Web Services. Essa classificação será armazenada no banco de dados do *broker* e indicará a reputação que os Web Services possuem. Assim, caso algum Web Service possua uma classificação negativa, na próxima vez que ele for listado em uma busca do cliente, o *broker* irá avisar o cliente que o serviço não correspondeu aos requisitos pedidos pelo cliente em um acesso anterior, mas a opção de o cliente acessar o serviço será mantida.

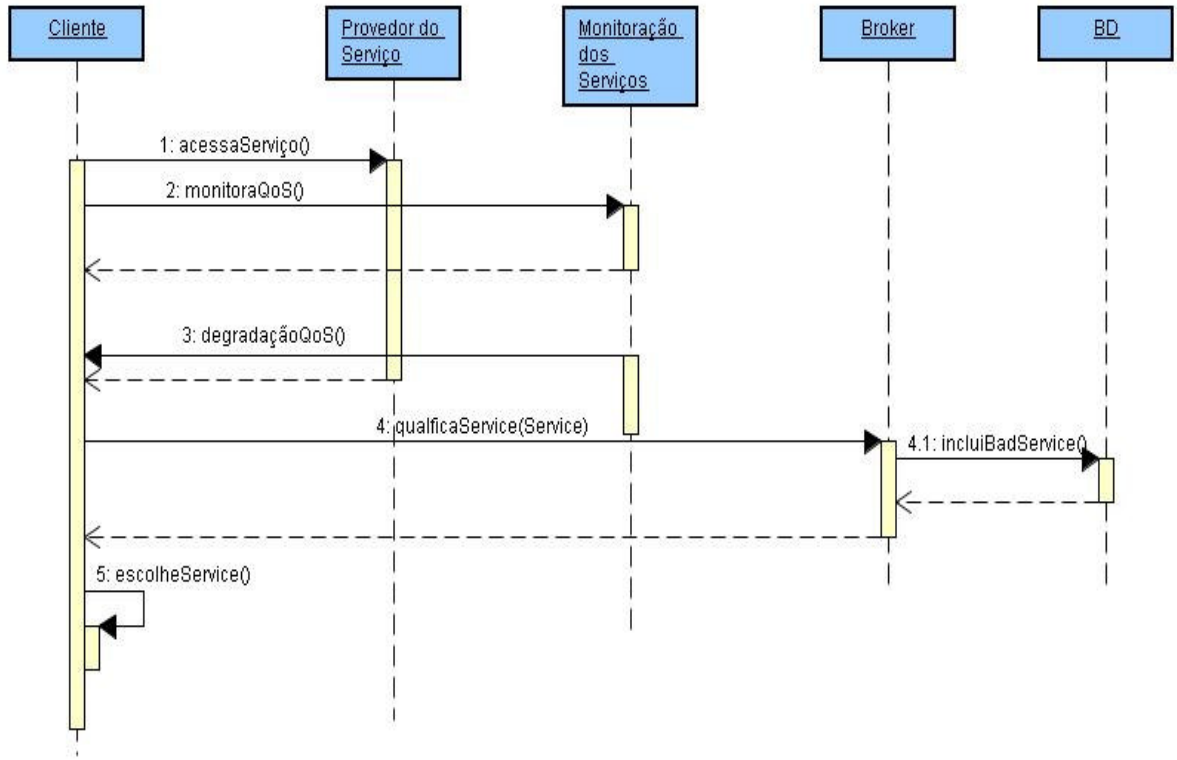

Figura 4.4. Diagrama de Seqüência UML para a monitoração do serviço

4.4. *BROKER*

O *broker* foi construído como um Web Service, assim, ele se comunica com o cliente, UDDI e servidor através de mensagens SOAP. O *broker* é responsável por receber e armazenar os arquivos XML que definem a QoS dos Web Services e pela seleção do melhor serviço para o cliente com base nos parâmetros de QoS que ele definiu. Ele possui três módulos: o coletor, o negociador, e o banco de dados. Além desses módulos o *broker* possui um Diretório de Arquivos onde são armazenados os

arquivos XML publicados pelo provedor dos Web Services. As próximas seções especificam essas entidades detalhadamente.

4.4.1. Coletor

O Coletor é a entidade responsável por armazenar e receber os dados de entrada e saída do *broker*. Ele que recebe os arquivos XML publicados pelo provedor do Web Service e armazena-os no Diretório de Arquivos, recebe as requisições do cliente com os parâmetros do cliente, acessa e recebe a resposta da UDDI e do banco de dados. Pode-se considerar que o Coletor serve como interface entre as outras entidades da arquitetura (cliente, Servidor e UDDI) e o *broker*.

Quando um Web Service publica sua informação de QoS, o Coletor recebe os dados do Web Service e armazena o arquivo XML no Diretório de Arquivos. As outras informações passadas pelo provedor do Web Service são armazenadas no banco de dados. Junto com esses dados é armazenada uma referência para o arquivo XML.

Quando o Coletor recebe a mensagem XML, ele a analisa para obter as palavras-chave e os parâmetros de QoS do serviço que o cliente deseja. Logo em seguida ele faz a operação de busca na UDDI usando as palavras-chave que o cliente enviou. Quando o coletor encontra algum serviço que tenha ligação com as palavras-chaves, ele acessa informações como URI, nomes dos métodos e tipos dos parâmetros necessários para acessar o Web Service. Esses dados são obtidos através do documento WSDL que é encontrado na UDDI, e são armazenados no banco de dados fazendo uma associação com o Web Service que foi publicado anteriormente. Desse modo, o *broker* sabe o nome do arquivo XML que está armazenado no Diretório de Arquivos e pode ter acesso a informação de QoS do Web Service. A figura 4.5 mostra o diagrama de seqüência para as funções do Coletor.

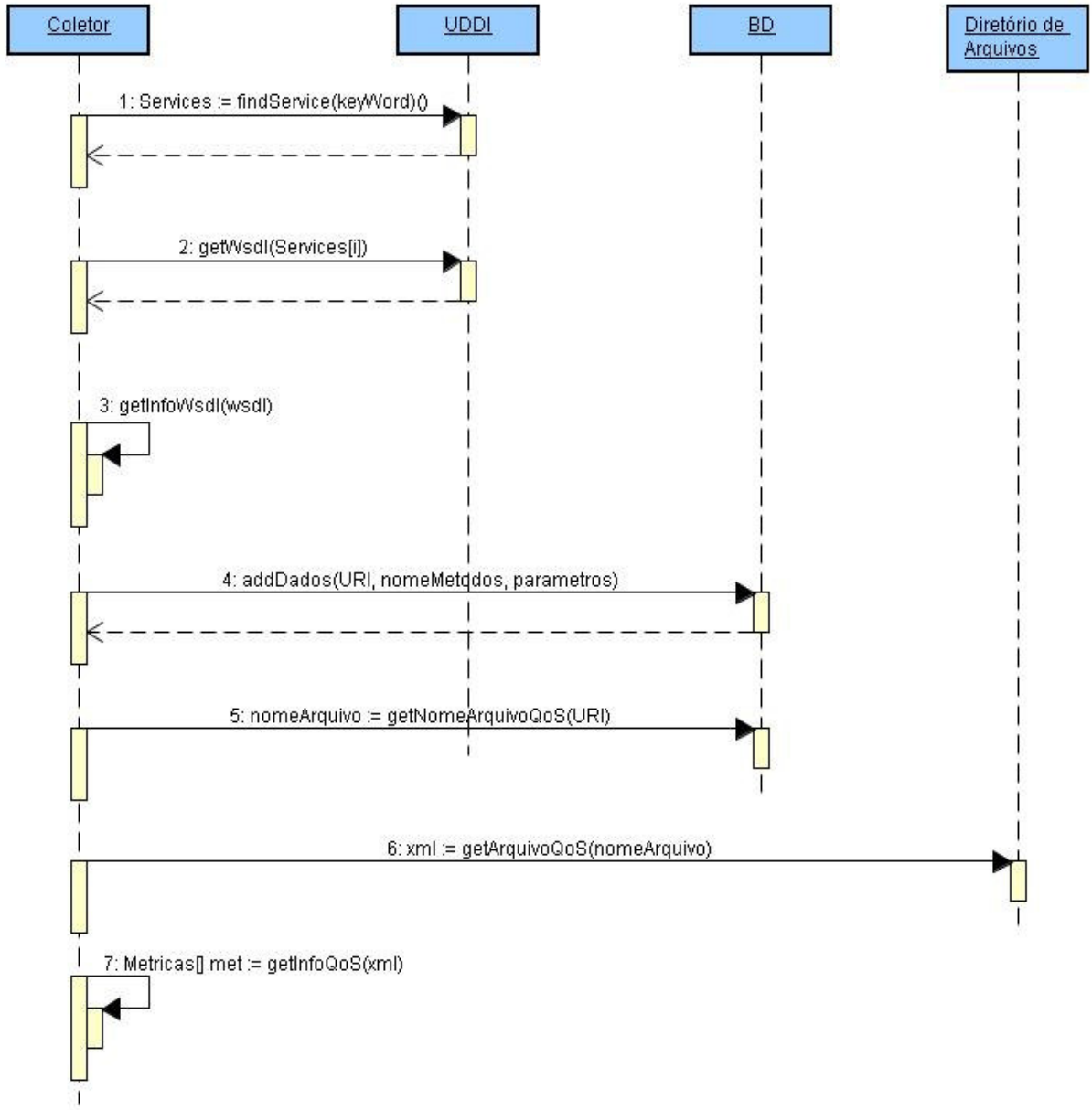

Figura 4.5. Diagrama de Sequência UML para as operações do Coletor

4.4.2. Negociador

O negociador é a parte principal do *broker*, responsável pela seleção do melhor serviço e pela verificação dos parâmetros de QoS com o Servidor. Ele irá receber do Coletor as informações dos parâmetros de QoS que o provedor do Web Service oferece, e as informações sobre a requisição do cliente. Ele então, envia uma mensagem para o servidor requisitando a QoS que o cliente pediu. Se o servidor puder cumprir os

requisitos do cliente, então ele envia uma mensagem de confirmação, caso contrário, uma mensagem de recusa é enviada. O negociador faz esse processo para cada Web Service que o coletor achou durante a busca na UDDI e reúne os resultados de cada um deles e calcula o serviço com a melhor QoS de acordo com os parâmetros enviados pelo cliente. Sabendo quais são os Web Services que cumprem os requisitos, o Negociador envia uma mensagem para o cliente retornando os Web Services encontrados. O Web Service que possuir a melhor QoS de acordo com os requisitos do cliente é recomendado pelo *broker* para o cliente, que poderá escolher qualquer Web Service dentre os que foram selecionados. A figura 4.6 mostra o diagrama de seqüência UML para as operações do Negociador.

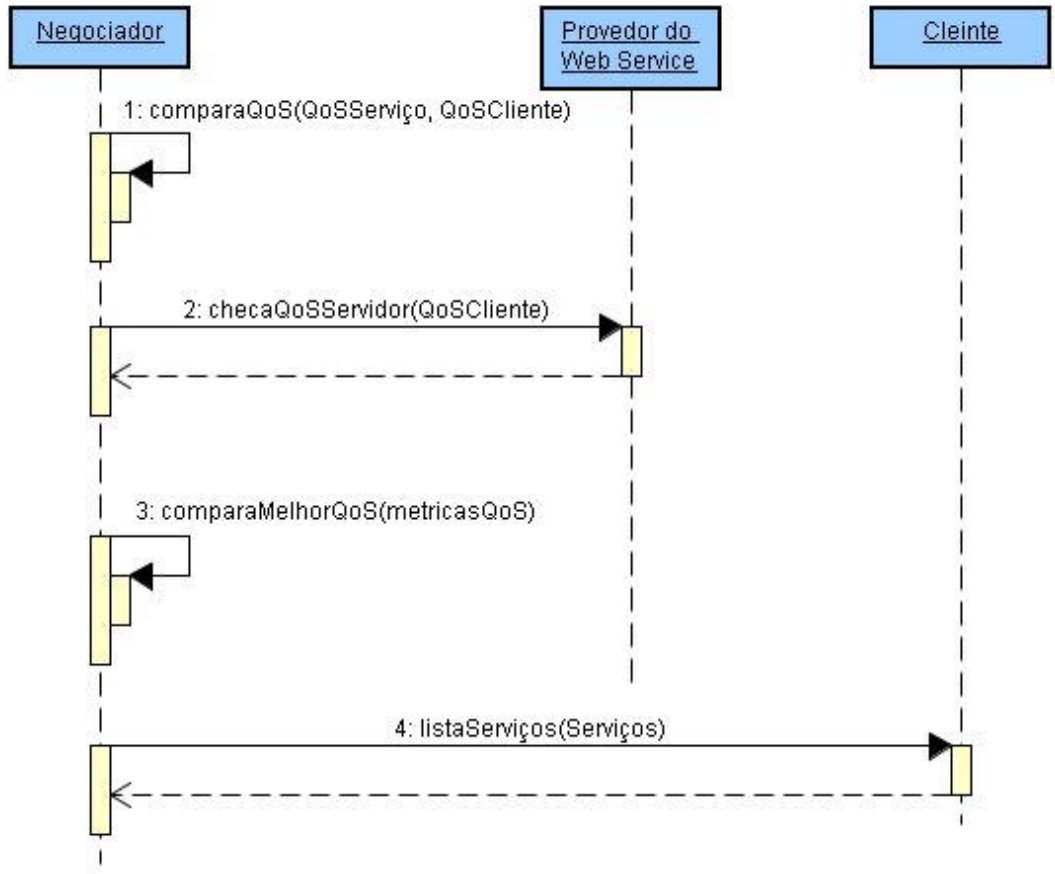

Figura 4.6. Diagrama de Seqüência UML para as operações do Negociador

Se durante o recebimento do serviço, o cliente notar que um ou mais parâmetros de QoS tiveram degradação dos valores abaixo do prometido, o *broker* irá adicionar uma

qualificação negativa para Web Service, e oferecerá outro serviço para o cliente caso tenha sido encontrado mais de um Web Service na busca anterior. Se o Web Service já tiver recebido uma classificação negativa, ele irá receber uma classificação pior, e quando for selecionado pelo *broker* para algum cliente, o *broker* irá mostrar ao cliente que o Web Service já descumpriu os parâmetros acordados.

4.4.3 Banco de Dados

O banco de dados armazena as informações de todos os serviços que foram publicados pelo provedor do Web Service. Quando o provedor publica o Web Service é criada uma nova linha na tabela 'Web Services' e são preenchidos os seguintes campos:

- nomeWebService - Nome do Web Service
- URI – *Universal Resource Identifier* do Web Service
- NomeArquivo – Nome do arquivo XML que descreve as informações de QoS.

Alguns outros campos são deixados em branco e são preenchidos à medida que os Web Services forem sendo encontrados na UDDI ou selecionados para o cliente. Quando o Web Service for retornado de uma busca na UDDI, através da URI contida no WSDL é possível associar o Web Service com seu respectivo arquivo de QoS. De posse do arquivo WSDL, são obtidos os métodos do Web Service e os parâmetros para cada método. Esses dados também são armazenados no banco de dados e serão mostrados ao cliente caso o Web Service seja relacionado para o cliente.

A arquitetura fornece um sistema de qualificação por reputação, onde o cliente pode manchar a reputação do Web Service caso ele não cumpra com os requisitos de QoS que foram acordados. Essa qualificação é armazenada em um campo no banco de dados cada vez que o Web Service degradar a QoS abaixo do esperado. Desse modo, sempre que um Web Service for selecionado pelo *broker*, o *broker* terá acesso à reputação do Web Service e irá mostrar ao cliente a reputação.

4.5 TECNOLOGIAS DE TERCEIROS

4.5.1 Apache Axis

O Apache Axis é uma implementação Open Source cliente/servidor do protocolo SOAP. Foi desenvolvido pelo projeto Jacarta e roda em cima do servidor Tomcat. A versão 1.2 do Apache Axis possui as seguintes características:

- Suporte ao protocolo SOAP 1.1/1.2
- Usa o SAX (*Simple API for XML*) como *parser* de documentos XML.
- É completamente independente de protocolo. Emissores e receptores podem usar SMTP, HTTP, FTP e *middlewares*.
- Gera documentos WSDL automaticamente dos Web Services desenvolvidos em Java. Também possui a ferramenta wsdl2java a qual gera *proxies* e *skeletons* em Java a partir do documento WSDL.
- Suporta o desenvolvimento de EJBs (*Enterprise Java Beans*) como Web Services
- Fornece suporte para serviços SOAP baseados em mensagem e RMI.

Como o cliente precisa se comunicar com os Web Services e com o *broker* através do protocolo SOAP, é necessário uma ferramenta que crie e processa os envelopes SOAP. O Apache Axis foi escolhido por ser Open Source, permitir o uso da linguagem de programação Java para criar Web Services e ser compatível com o Tomcat, que foi o servidor escolhido para hospedar o *broker*.

4.5.2 jUDDI

O jUDDI é uma implementação Java Open Source da especificação UDDI para Web Services. Sendo o jUDDI uma aplicação Web, o Java pode ser utilizado com qualquer servidor de aplicação ou mecanismo *servlet* que suporte a versão 2.1 ou superior do *servlet* API. Com o jUDDI foi criado um UDDI privado para publicar os Web Services. A UDDI privada foi implementada para uso no estudo de caso, que é apresentado no Capítulo 6. Foi optado em implementar uma UDDI privada ao invés de usar o registro público da UDDI por uma simples questão de bom senso, mantendo os

serviços usados no estudo de caso e suas informações dentro de um domínio privado. Detalhes da implementação da UDDI podem ser encontrados no Capítulo 5.

4.6 RESUMO DO CAPÍTULO

Neste capítulo foi apresentada a arquitetura para fornecimento de QoS para Web Services. Foi apresentada inicialmente uma visão geral da arquitetura. Em seguida foram apresentadas todas as entidades que compõe a arquitetura e suas operações e como uma entidade se relaciona com a outra através de diagramas de seqüência UML. Por fim foram apresentadas uma justificativa e uma explicação para as tecnologias de terceiros que são usadas pela arquitetura.

5. PROJETO E IMPLEMENTAÇÃO

Neste capítulo será apresentada a implementação da arquitetura proposta no capítulo anterior. Na seção 5.1 é apresentado o Esquema XML que a arquitetura utiliza para definir os parâmetros de QoS. A seção 5.2 descreve a implementação das entidades da arquitetura.

5.1. ESQUEMA XML

Cada serviço disponibilizado por um fornecedor possui um documento XML contendo as definições de métricas de QoS, que segue o modelo definido pelo Esquema XML. A QoS do Web Service é especificada através de um elemento do tipo *eQoS*. Se esse Web Service possuir diferentes classes de serviço, cada classe de serviço será especificada como um Web Service diferente, sendo criado um elemento do tipo *eQoS* para cada classe do Web Service. Todos os documentos que especificam a QoS dos Web Services são armazenados no Diretório de Arquivos do *broker*.

Mesmo com o cliente e o provedor do Web Service estando livres para criar os parâmetros de QoS que eles desejam, alguns parâmetros de QoS são definidos pelo Esquema XML, estando já declarados para uso na arquitetura. Isso torna o uso da arquitetura mais fácil, especialmente para clientes que não possuem experiência sobre parâmetros de QoS e que não consigam se aproveitar da extensibilidade para definir os parâmetros que seriam mais úteis para acessar determinado serviço. Os parâmetros definidos foram:

- Disponibilidade: A probabilidade de o serviço estar disponível em determinado período de tempo.
- Tempo de Resposta: O total do tempo gasto pelo servidor, desde o momento em que ele recebeu a requisição, até o momento em que ele envia a resposta.
- Perda de Pacotes: Porcentagem dos pacotes que foram perdidos durante o fornecimento do serviço.
- Delay: O tempo que o pacote demora para ir do servidor ao cliente.
- Preço: O preço pago para acessar o serviço.

Tanto o cliente como o fornecedor do Web Service estão livres para criarem os parâmetros que eles desejam, podendo ser declarados através de um elemento *eDeclaracao* que especifica as definições de parâmetros de QoS e protocolos de referência. Isso permitirá ao cliente especificar como parâmetro de QoS, por exemplo, uma determinada faixa de preço para um automóvel, entre os Web Services que oferecem vendas de automóveis, ou determinado gênero de filme para comprar um DVD e ainda utilizar os parâmetros clássicos de QoS para obter Web Services com melhor desempenho.

O atributo nome do elemento *eQoS* define o nome do Web Service sobre o qual será feita a definição das ofertas de serviço. Este elemento irá fazer parte do elemento *ofertas* quando armazenado no servidor, e parte do elemento *requisição* quando enviado pelo cliente, contendo os requerimentos para o cliente utilizar determinado serviço.

A Figura 5.1 ilustra a estrutura do elemento *eQoS*. O elemento *eDeclaracao* armazena as declarações de todas as métricas de QoS do serviço. Essa entidade é explicada a seguir. A entidade *eMonitoracao* é responsável por definir protocolos e métodos necessários para o monitoramento e gerenciamento dos parâmetros de QoS, já que o cliente precisa saber como monitorar a QoS durante o recebimento de um serviço. Ela é uma entidade opcional, mas por outro lado, podem existir vários meios para fazer o monitoramento. A entidade preço define o valor e a moeda corrente a ser pago pelo fornecimento do serviço, podendo ser considerado um parâmetro de QoS independente, e não sendo necessário incluí-lo junto com os outros parâmetros na entidade *eDeclaracao.* A entidade preço é um dos parâmetros de QoS que já são declarados previamente, pois como é um parâmetro muito usado, disponibilizá-lo previamente irá poupar tempo significativo para o cliente e o fornecedor dos serviços. A entidade extensão é reservada para futuras extensões.

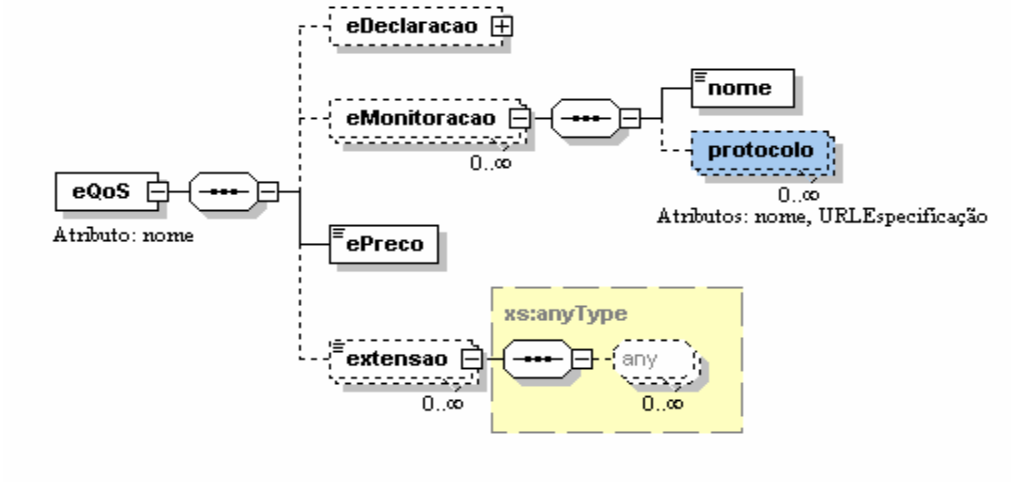

Figura 5.1. Estrutura do elemento *eQoS*

O elemento *eDeclaracao* é o responsável pelos campos para a definição dos parâmetros de QoS. Ele é dividido em *eServerMetrics,* que considera os parâmetros relacionados somente ao desempenho do servidor, e *eTransportMetrics*, que define as métricas relacionadas ao transporte dos dados na rede. A Figura 5.2. ilustra a estrutura do elemento *eDeclaracao.*

O elemento *metrica* representa os diferentes parâmetros de QoS que o usuário e o servidor têm a opção de definir. O campo extensão fornece extensibilidade, podendo ser aplicado inclusive, para parâmetros de segurança, que não são abordados neste trabalho. Cada elemento *metrica* possui vários atributos que são responsáveis por definir as características dos parâmetros, sendo responsável por especificar como os parâmetros devem ser definidos. Os atributos do elemento *metrica* são: nome, a unidade de medida, o intervalo de tempo em segundos, direção de como ele é medido, além de uma descrição breve. Desse modo, tanto o cliente como o fornecedor de serviços podem definir os parâmetros de QoS de uma forma que quando o cliente fizer uma requisição, o *broker* entenderá os parâmetros que o cliente está solicitando e poderá comparar com os parâmetros oferecidos pelo servidor

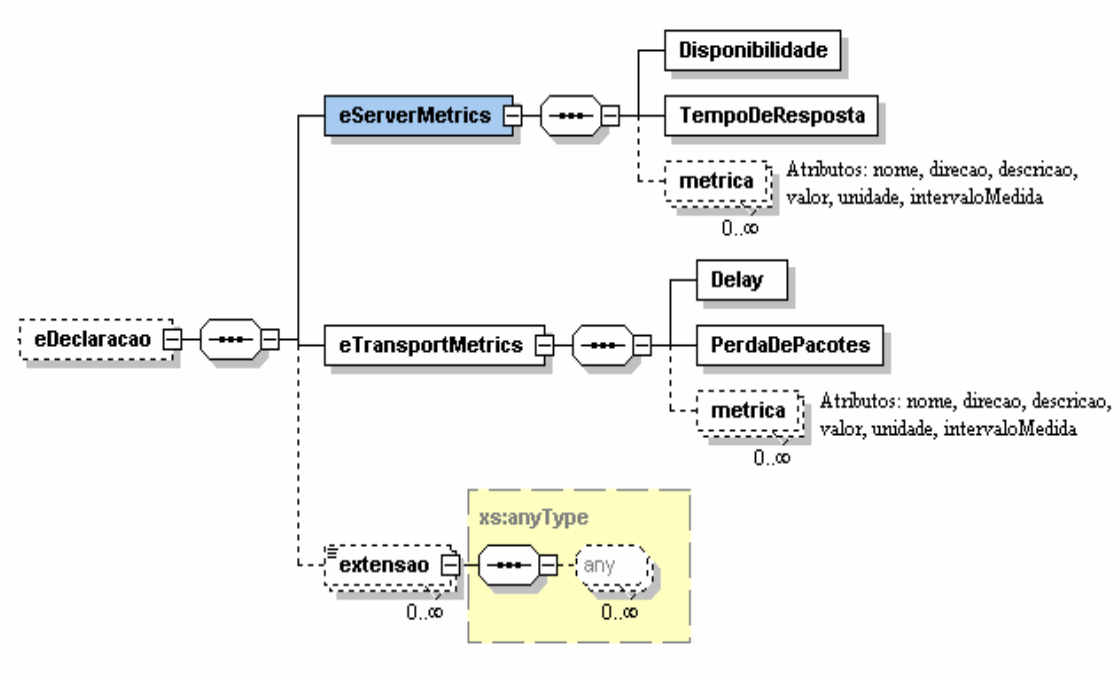

Figura 5.2. Estrutura do elemento *eDeclaracao*

5.2. IMPLEMENTAÇÃO DA ARQUITETURA

O protótipo foi desenvolvido utilizando a ferramenta Axis [Axis], descrita no capítulo anterior. Todas as mensagens são trocadas pelas entidades da arquitetura através do Axis. A seguir são descritas as implementações de cada entidade da arquitetura.

5.2.1. Servidor

O servidor publica a informação de QoS que os Web Services que ele fornece e suas classes de serviço possuem. Isso é feito invocando o método *publishClient* do *broker*. Como o *broker* é implementado usando o Axis, para que o método possa ser usado remotamente foi criado um cliente Axis para acessar o método. A figura 5.3. mostra a captura de tela do cliente desenvolvido.

O método *publishClient* possui como parâmetros, além do arquivo XML contendo os parâmetros de QoS, que é armazenado no Diretório de Arquivos, o nome do Web Service, seu URI e o nome do documento XML que descreve as QoS.

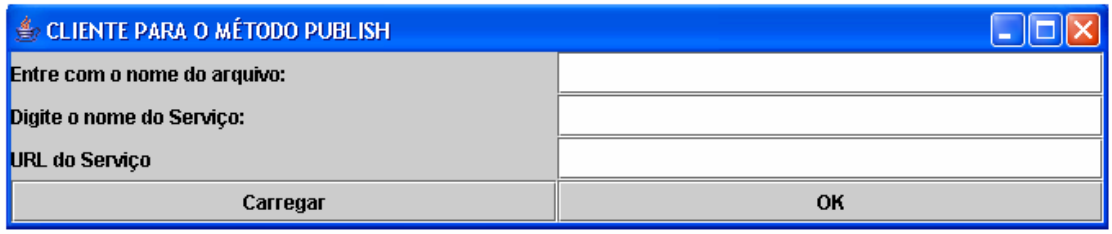

Figura 5.3. Captura de tela do cliente do método Publish

5.2.2. Cliente

Para que o cliente possa fazer a busca, ele precisa acessar o método *consultClient*, oferecido pelo *broker*. Como o *broker* é implementado usando o Axis, para que o método possa ser usado remotamente foi criado um cliente Axis para acessar o método.

O cliente pode ter dificuldades em desenvolver seus próprios parâmetros de QoS em XML. Para amenizar esse problema foi desenvolvida para o cliente uma interface que cria o documento XML contendo os parâmetros de QoS do cliente. Essa interface permite criar o documento sem erros sintáticos e de uma forma mais rápida e intuitiva para o cliente.

Quando a aplicação se inicializa é criado um documento *template*, que possui toda a estrutura do documento XML. Depois, para cada parâmetro adicionado pelo cliente são criadas *tags* adicionais no *template*. Os parâmetros de QoS disponibilizados na arquitetura já estão declarados no *template*, de modo que o cliente só precisa adicionar os valores desejados. A figura 5.4 mostra a captura de tela do cliente Axis desenvolvido para acessar o método *consultClient*.

A monitoração dos parâmetros em tempo real é disponibilizada para os parâmetros que foram definidos pela arquitetura. Para coletar a informação de QoS em tempo real foram usados coletores para extrair informações gerenciais nos computadores remotos. Já para os parâmetros que forem definidos pelo cliente ou pelo servidor, o Esquema XML define a entidade *eMonitoracao* para definir protocolos e métodos necessários para o monitoramento dos parâmetros de QoS.

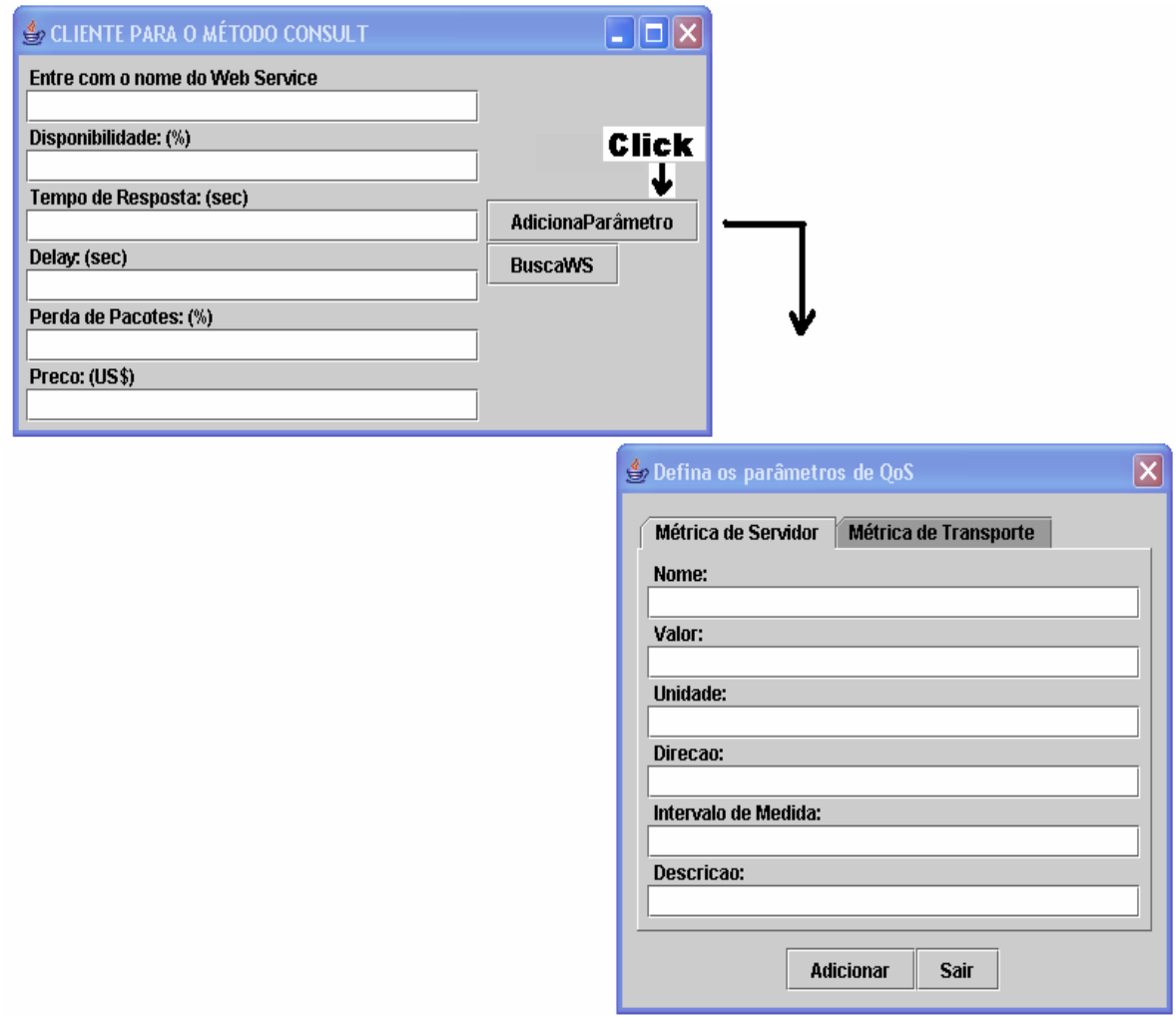

Figura 5.4 – Captura de tela do cliente do método Consult

5.2.3. *Broker*

O *broker* foi desenvolvido como um Web Service, para que possa ser acessado através da Web. Para isso, seus métodos foram disponibilizados no servidor Tomcat que o Axis fornece. Foram desenvolvidos dois métodos principais, um para acesso do provedor do serviço, e outro para acesso do cliente. Para cada um desses métodos foram desenvolvidos clientes Axis mostrados anteriormente..

- *publishClient* - Método para o provedor dos serviços informar quais os Web Services e os parâmetros de QoS que o provedor oferece. Quando esse método é acessado o provedor informa qual é o nome do Web Service, o nome do arquivo XML, a URL onde o Web Service é fornecido e passa o documento XML

descrevendo os parâmetros de QoS e os respectivos valores. O arquivo XML é armazenado no Diretório de Arquivos do *broker* e os outros dados são armazenados no banco de dados.

- *consultClient* - Método que o cliente usa para fazer a busca por Web Services. O cliente passará como parâmetro a palavra-chave para a busca na UDDI e o arquivo XML com os parâmetros de QoS desejados pelo cliente.

Além desses métodos foi criado o método *findService*. Esse método é responsável por fazer a consulta na UDDI com a palavra-chave passada pelo cliente através do método *consultClient* e receber a lista dos Web Services retornados pela UDDI. Esse método retorna as seguintes informações: o nome do Web Service, a URI para acessa-lo, e o link para o documento WSDL. De posse do documento WSDL, ele é analisado e as informações dos nomes dos métodos e seus parâmetros são obtidas. Pela URI do Web Service é possível saber qual é o Web Service e o documento XML que especifica seus parâmetros de QoS. Os dados obtidos pelo documento WSDL também são armazenados no banco de dados.

Quando o *broker* tem o método *consultClient* acessado, significa que um cliente deseja buscar por um Web Service. Depois de ter os Web Services retornados da UDDI, o *broker* acessa o banco de dados para obter o documento XML com os parâmetros de QoS do Web Service. Logo em seguida, é criado um objeto *Service* e ele acessa o método *xmlToMetrica*, que lê os documentos XML e transforma os parâmetros de QoS definidos nele em objetos *Métrica*. O método *xmlToMetrica* também é chamado para transformar os parâmetros de QoS do documento XML do cliente em objetos *Métrica*.

Para definir se um determinado parâmetro de QoS cumpre as requisições do cliente, é feita uma análise do tipo da métrica. Se for um valor descendente (como preço), cujo melhor é o menor valor, e se a oferta do Web Service possuir um valor menor, ele será relacionado. Caso o valor da métrica for ascendente (como disponibilidade), é feita a comparação e se a oferta do Web Service possuir um valor maior, ele será retornado. Para selecionar o melhor Web Service, é feita a mesma comparação entre todos os Web Services que passaram na etapa anterior, e o que tiver o melhor valor, entre as métricas relacionadas pelo cliente será escolhido. A figura 5.5 mostra o Diagrama de Classes UML da arquitetura.

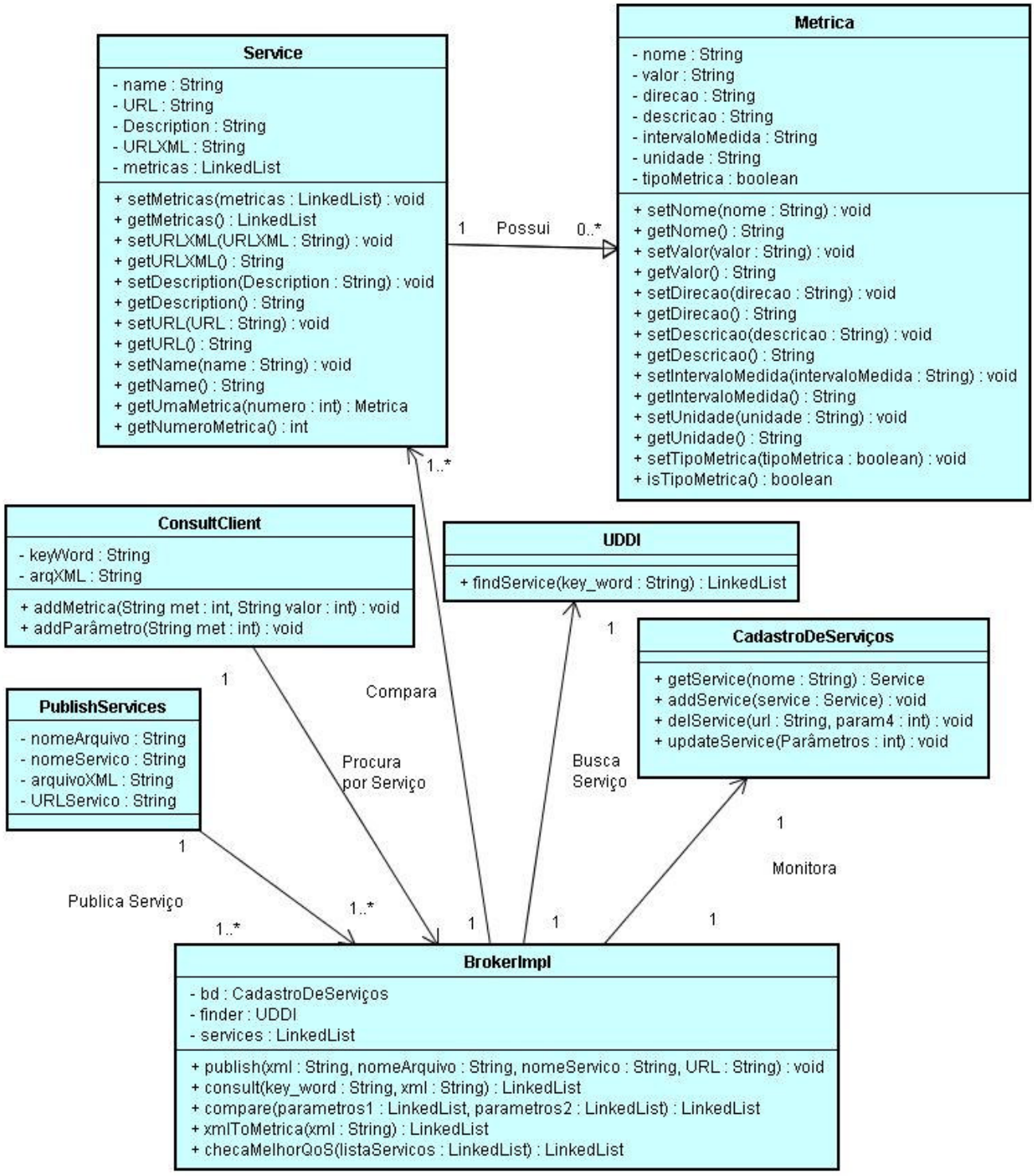

Figura 5.5. Diagrama de Classe UML da arquitetura implementada

5.2.4. UDDI

Foi necessário implementar um registro UDDI para o estudo de caso, já que alguns Web Services deveriam estar publicados no UDDI. Optou-se por uma implementação de um UDDI privado, já que garante um maior controle na publicação dos Web Services, não ocupa o UDDI público com informações sobre Web Services que não são verdadeiros e oferece o mesmo resultado e padrão para publicação do que o UDDI público, já que a API usada para acessar os métodos da UDDI é a mesma tanto para registros privados como públicos.

O registro UDDI foi implementado utilizando a versão 0.9 do jUDDI (Java UDDI) [JUDDI] em uma máquina remota, e foram publicados nela alguns Web Services para o estudo de caso. Os dados publicados no jUDDI são armazenados em um banco de dados. O banco de dados utilizado foi o MySQL. Os dados referentes aos Web Services que são publicados através de métodos da API, são então armazenados em tabelas SQL criadas pelo jUDDI. Para que um Web Service possa ser publicado, o jUDDI precisa estar disponível em um servidor para que possa ser acessado remotamente. O servidor que é compatível com o jUDDI é o Tomcat disponibilizado pelo JWSDP (*Java Web Services Development Pack*) [JWSDP]. Foram instalados o JWSDP e o Tomcat e foi feita a configuração para o UDDI ficar disponível remotamente.

O *broker* acessa a UDDI através da API UDDI4j (*UDDI for Java*) [UDDI4J]. A UDDI4j fornece classes em Java para acessar os modelos de informações do UDDI. A partir dos métodos dessas classes são retornados objetos que contém as informações publicadas no UDDI. Quando o *broker* executa o método *findService* ele chama os métodos disponibilizados pela UDDI4j e faz a busca na UDDI privada, que foi implementada pelo jUDDI. A figura 5.6 mostra as relações entre os módulos da UDDI privada. O JWSDP é um *toolkit* para o desenvolvimento de Web Services. Ele contém o Tomcat configurado para o jUDDI, que disponibiliza o acesso remoto. O Tomcat encapsula o Axis para dar suporte ás mensagens SOAP, que é o protocolo de comunicação entre o *Publisher* e o UDDI. O UDDI privado é implementado através do jUDDI e as operações de busca podem ser executadas através da UDDI4j. Todos os dados publicados pelo jUDDI são armazenados em um banco de dados MySQL, e através do *pulg-in MySQL-Connector* consultas SQL podem ser feitas através de classes Java.

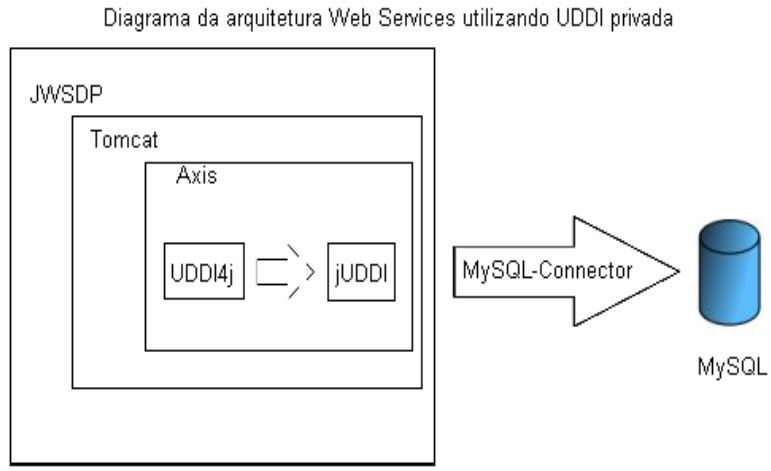

Figura 5.6. Relações entre os módulos do UDDI privado

Acessando o UDDI, o *broker* recebe as informações do nome e da URL, a descrição do Web Service, e o documento WSDL. Através do documento WSDL é possível acessar as informações dos métodos que o Web Service disponibiliza e os parâmetros dos métodos. Esses dados serão armazenados no Banco de Dados do *broker*. Note que o servidor Tomcat onde o *broker* é hospedado é diferente do Tomcat onde está o UDDI, assim como os banco de dados.

5.2.5. Banco de Dados

O banco de dados do *broker* é necessário para fazer a associação entre o Web Service que foi retornado da UDDI com o Web Service do provedor quando o *broker* acessar a informação de QoS do Web Service, além de manter informações sobre os métodos e parâmetros dos Web Services. O banco de dados utilizado foi o MySQL, que através do *plug-in MySQL-Connector* fornece suporte para Java. O banco de dados possui três tabelas, sendo que a tabela que possui os principais dados dos Web Services armazena os seguintes dados:

- Nome do Web Service
- Nome do arquivo XML que descreve os parâmetros de QoS
- URI do Web Service (chave primária)
- Endereço do arquivo WSDL.

Para cada Web Service são criados duas tabelas, uma com os métodos que ele disponibiliza e seus respectivos argumentos e outra para armazenar a reputação do cliente.

5.3. RESUMO DO CAPÍTULO

Neste capítulo foi apresentado o Esquema XML para a definição de parâmetros de QoS e os detalhes da implementação da arquitetura. Foram apresentados os modelos referentes ao Esquema XML e os elementos que o compõem. A implementação da arquitetura foi explicada detalhadamente, apresentando todas as entidades, mostrando captura de telas e um diagrama de classe UML com todas as entidades.

6. ESTUDO DE CASO

Para que a arquitetura e o Esquema XML proposto pudessem ser validados, é necessário um estudo de caso que comprove que a arquitetura e os Esquemas XML cumpriram os seus propósitos. Esta seção apresenta um estudo de caso descrevendo como um cliente poderia utilizar a arquitetura para conseguir um Web Service com os parâmetros de QoS que ele deseja. O UDDI foi disponibilizado para acesso na máquina A. Os Web Services desenvolvidos conforme a seção 6.1 foram hospedados na máquina B. O *broker* está hospedado no Tomcat instalado na máquina C, e o cliente faz a requisição para o *broker* através da máquina D. A seção 6.1. descreve os Web Services que foram implementados para o estudo de caso. A seção 6.2. mostra como os Web Services foram publicados e a seção 6.3. mostra o acesso de um cliente ao *broker* para fazer a busca por Web Services com QoS.

6.1. DESCRIÇÃO DOS WEB SERVICES

Para o estudo de caso foram implementados cinco Web Services. Como a principal função do estudo de caso não é mostrar Web Services robustos funcionando, e sim comprovar a eficiência da busca por Web Services através do *broker*, os Web Services desenvolvidos são triviais, e não foram desenvolvidos clientes e interfaces Web para acessá-los. Os parâmetros de QoS podem não se justificar para serem usados em Web Services com essas funcionalidades, mas implementar um Web Service robusto, estudar os parâmetros de QoS e seus valores e definir um método para garanti-los está fora do escopo desse trabalho.

Para cada Web Service criado foram criadas três cópias, que foram colocados em URI's diferentes. Isso foi feito com o objetivo de criar Web Services com a mesma funcionalidade, o que irá fazer com que seja retornado mais de um Web Service em uma busca na UDDI. Os Web Services desenvolvidos são os seguintes:

- Calculator, Calculator1, Calculator2, Calculator3: Disponibiliza os quatro métodos básicos de uma calculadora, adição, subtração, multiplicação e divisão, para somente dois operandos, que são passados por parâmetros.

Links de acesso: http://blue:8080/test3/Calculator.jws, http://blue:8080/test3/Calculator1.jws, etc.

- FarenheintToCelsus, FarenheintToCelsus1, FarenheintToCelsus2, FarenheintToCelsus3: Converte graus na escala Farenheint para a escala Celsus. Fornece apenas o método para a conversão de escalas, com o valor a ser convertido como argumento do método. Links de acesso: http://blue:8080/test3/FarenheintToCelsus.jws, http://blue:8080/test3/FarenheintToCelsus1.jws, etc.
- BookService, BookService1, BookService2, BookService3: Serviço que simula uma livraria *on-line*. São disponibilizados os seguintes métodos para os usuários: *findBook*, *getNumberOfBooks*, *getAuthors, getNpages*. Links de acesso: http://blue:8080/test3/BookManager.jws, http://blue:8080/test3/BookManager1.jws, etc.
- CDService, CDService1, CDService2, CDService3: Serviço análogo ao BookService, efetuando os mesmos métodos, mas simulando uma loja de Cds. Links de acesso: http://blue:8080/test3/CDManager.jws, http://blue:8080/test3/CDManager1.jws
- DVDService, DVDService1, DVDService2, DVDService3: Serviço de vendas de DVD *on-line* que utiliza os mesmos métodos definidos para os serviços CDService e BookService. Links de acesso: http://blue:8080/test3/DVDManager.jws, http://blue:8080/test3/DVDManager1.jws, etc.
- StockQuoteService, StockQuoteService1, StockQuoteService2, StockQuoteService3: Retorna a cotação das ações de http://services.xmethods.net/axis/getQuote em formato XML. O Web Service faz uma requisição pedindo a cotação da ação, lê o documento XML e devolve o valor. Links de acesso: http://blue:8080/test3/StockQuoteService.jws, http://blue:8080/test3/StockQuoteService1.jws, etc.

O Axis usa extensão de arquivos .jws (Java Web Service) para acessar um Web Service sem interface Web (JSP, php ou HTML) disponível. Dessa forma pode se acessar um Web Service remotamente por um *browser* como se fosse uma linha de comando.

O Axis possui uma ferramenta que gera automaticamente o documento WSDL de um Web Service. Essa ferramenta foi utilizada para gerar os documentos WSDL dos Web Services. A figura 6.1. mostra o documento WSDL gerado para o Broker. No Apêndice A são mostrados os WSDL dos Web Services desenvolvidos para a validação.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/test3/Calculator.jws"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/test3/Calculator.jws"
xmlns:intf="http://localhost:8080/test3/Calculator.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!--WSDL created by Apache Axis version: 1.2on May 03, 2005 (02:20:24 EDT)-->
        <wsdl:message name="multiplieResponse">
                <wsdl:part name="multiplieReturn" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="subtractRequest">
                <wsdl:part name="i1" type="xsd:int"/>
                <wsdl:part name="i2" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="subtractResponse">
                <wsdl:part name="subtractReturn" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="addRequest">
                <wsdl:part name="i1" type="xsd:int"/>
                <wsdl:part name="i2" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="addResponse">
                <wsdl:part name="addReturn" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="divideResponse">
                <wsdl:part name="divideReturn" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="multiplieRequest">
                <wsdl:part name="i1" type="xsd:int"/>
                <wsdl:part name="i2" type="xsd:int"/>
        </wsdl:message>
        <wsdl:message name="divideRequest">
                <wsdl:part name="i1" type="xsd:int"/>
                <wsdl:part name="i2" type="xsd:int"/>
        </wsdl:message>
        <wsdl:portType name="Calculator">
                <wsdl:operation name="add" parameterOrder="i1 i2">
                        <wsdl:input message="impl:addRequest" name="addRequest"/>
                        <wsdl:output message="impl:addResponse" name="addResponse"/>
                </wsdl:operation>
                <wsdl:operation name="divide" parameterOrder="i1 i2">
                        <wsdl:input message="impl:divideRequest" name="divideRequest"/>
                        <wsdl:output message="impl:divideResponse" name="divideResponse"/>
                </wsdl:operation>
                <wsdl:operation name="subtract" parameterOrder="i1 i2">
                        <wsdl:input message="impl:subtractRequest" name="subtractRequest"/>
                        <wsdl:output message="impl:subtractResponse" name="subtractResponse"/>
                </wsdl:operation>
                <wsdl:operation name="multiplie" parameterOrder="i1 i2">
                        <wsdl:input message="impl:multiplieRequest" name="multiplieRequest"/>
                        <wsdl:output message="impl:multiplieResponse" name="multiplieResponse"/>
                </wsdl:operation>
        </wsdl:portType>
```

```
        <wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
                <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
                <wsdl:operation name="add">
                        <wsdlsoap:operation soapAction=""/>
                        <wsdl:input name="addRequest">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
                        </wsdl:input>
                        <wsdl:output name="addResponse">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/test3/Calculator.jws" use="encoded"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="divide">
                        <wsdlsoap:operation soapAction=""/>
                        <wsdl:input name="divideRequest">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
                        </wsdl:input>
                        <wsdl:output name="divideResponse">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/test3/Calculator.jws" use="encoded"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="subtract">
                        <wsdlsoap:operation soapAction=""/>
                        <wsdl:input name="subtractRequest">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
                        </wsdl:input>
                        <wsdl:output name="subtractResponse">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/test3/Calculator.jws" use="encoded"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="multiplie">
                        <wsdlsoap:operation soapAction=""/>
                        <wsdl:input name="multiplieRequest">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
                        </wsdl:input>
                        <wsdl:output name="multiplieResponse">
                                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/test3/Calculator.jws" use="encoded"/>
                        </wsdl:output>
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="CalculatorService">
                <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
                        <wsdlsoap:address location="http://localhost:8080/test3/Calculator.jws"/>
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

Figura 6.1. Documento WSDL do Web Service Calculator

6.2. PUBLICAÇÃO DOS WEB SERVICES

Os Web Services implementados foram publicados no UDDI privado implementado através do jUDDI. Como mostrado na figura 5.6, o jUDDI é incorporado pelo servidor Tomcat. Assim, o jUDDI disponibiliza uma interface para acesso Web. A configuração do Tomcat disponibilizou o seguinte link para acessar a interface Web do jUDDI: http://localhost:8080/juddi/console/

A interface Web do jUDDI disponibiliza um console para publicação de Web Services. Quando um Web Service é publicado através do console do jUDDI, uma mensagem SOAP é enviada para o jUDDI que analisa os dados e faz a publicação no banco de dados do UDDI, que no caso do jUDDI, é implementado em My-SQL, conforme a figura 5.6.

Foram publicados os seguintes dados de cada Web Service:

- nome do Web Service;
- endereço URI;
- descrição breve;
- endereço do documento WSDL

Depois que os Web Services são publicados no UDDI foram criados os documentos que descrevem os parâmetros de QoS fornecidos pelo Web Services. A figura 6.2. mostra o documento XML criado para o Web Service Calculator. Cada cópia do Web Service tem os valores dos parâmetros de QoS diferentes, sendo que a seleção será feita sempre pelo Web Service que tiver os melhores parâmetros de QoS de acordo com os desejos do cliente.

Cada Web Service invocou o cliente desenvolvido para o método *publish* do *broker* e passou suas informações para o *broker*. As informações que o provedor do Web Service passa para o *broker* nessa etapa são:

- Nome do Web Service: informação trivial
- URI do Web Service: Chave primária. A URI é utilizada para descobrir se o banco de dados possui informações sobre o Web Service.
- Nome do documento XML: necessário para indexar o documento no Diretório de Arquivos. O nome fica armazenado no banco de dados, para que o *broker* saiba qual é o arquivo correspondente ao Web Service.

- Documento XML

Os arquivos XML foram validados pelo *broker* e armazenados no Diretório de Arquivos.

6.3. PROCURA E ACESSO DOS WEB SERVICES POR UM CLIENTE

A busca por Web Services é feita através do cliente do Axis desenvolvido para acessar o método *consultClient* do *broker*. Quando o cliente é inicializado um documento *template* é carregado e é preenchido de acordo com os parâmetros de QoS estipulado. Foram feitas requisições fazendo mudança de palavra-chave e alterando os parâmetros de QoS. A seguir é descrito o resultado da busca para o Web Service Calculator.

Com os Web Services publicados na UDDI e tendo os parâmetros de QoS definidos, o cliente pode fazer a requisição ao *broker.* Foram executados os seguintes passos:

Passo 1: O cliente define a palavra-chave para acessar o Web Service Calculator, e os parâmetros de Qos que ele deseja., e acessa o *broker* através do método *consult*.

Passo 2: O *broker* acessa a UDDI e descobre que há quatro Web Services que correspondem a uma calculadora. O *broker* obtém o documento WSDL, os métodos e parâmetros de cada Web Service, assim como sua URL, e os armazena no banco de dados.

Passo 3: Através da URL de cada Web Service, o *broker* descobre qual o arquivo XML que descreve as ofertas de QoS de cada Web Service e obtém os dados de QoS de cada Web Service Calculator. O *broker* acessa os requisitos de QoS do cliente e compara os valores dos parâmetros de QoS definidos. Dos quatro Web Services encontrados na UDDI, apenas três cumprem os requisitos de QoS do cliente.

Passo 4: O *broker* contata os servidores dos Web Services para confirmar se a requisição do cliente pode ser aceita. Um dos servidores responde que não pode garantir os valores de QoS. Os outros dois servidores notificam o *broker* positivamente.

```
<?xml version="1.0"?>
<QoS xmlns = "ParameterQoSSchema.xml">
<elementQoS>
<name>Calculator</name>
<eDeclaracao>
  <eServerMetrics>
    <metrica>
          <name> Tempo de Resposta </name>
           <eOntology>
                  <unidade> sec </unidade>
                  <intervaloMedida> 60 </intervaloMedida>
                  <valorOferta> 0.05 </valorOferta>
                  <direcao> Server to client </direcao>
                    <Descricao> Tempo gasto pelo servidor para processar a requisicao e enviar a resposta
                  </Descricao>
           </eOntology>
    </metrica>
    <metrica>
          <name> Disponibilidade </name>
          <eOntology>
                  <unidade> porcentagem </unidade>
                  <intervaloMedida> 60 </intervaloMedida>
                  <valorOferta> 95 </valorOferta>
                    <Descricao> Porcentagem em que o servidor do Web Service permanece disponivel
                  </Descricao>
          </eOntology>
    </metrica>
  </eServerMetrics>
  <eTranportMetrics>
    <metrica>
          <name> Largura de Banda </name>
          <eOntology>
                  <unidade> Hz </unidade>
                  <intervaloMedida> 60 </intervaloMedida>
                  <valorOferta> 86 </valorOferta>
                  <direcao> Server to client </direcao>
                  <Descricao> Banda necessaria para receber o servico </Descricao>
          </eOntology>
    </metrica>
    <metrica>
          <name> Delay </name>
          <eOntology>
                  <unidade> sec </unidade>
                  <intervaloMedida> 60 </intervaloMedida>
                  <valorOferta> 0,2 </valorOferta>
                  <direcao> Client </direcao>
                  <Descricao> Tempo que a resposta demora para transitar sobre a rede ate o cliente </Descricao>
          </eOntology>
    </metrica>
  </eTranportMetrics>
</eDeclaracao>
<ePreco> R$0,25 </ePreco>
</elementQoS>
</QoS>
```

Figura 6.2. XML para a descrição dos parâmetros de QoS do Web Service Calculator

Passo 5: Dos dois Web Services que foram selecionados, é calculado o que possui a melhor QoS. As informações dos dois Web Services são então retornadas para o cliente que faz a escolha entre os Web Services. As informações do Web Service com a melhor QoS são colocadas no topo da lista. A figura 6.3 mostra as informações retornadas pelo *Broker* ao cliente.

Passo 6: O cliente acessa o Web Service e faz a monitoração dos parâmetros de QoS em tempo real.

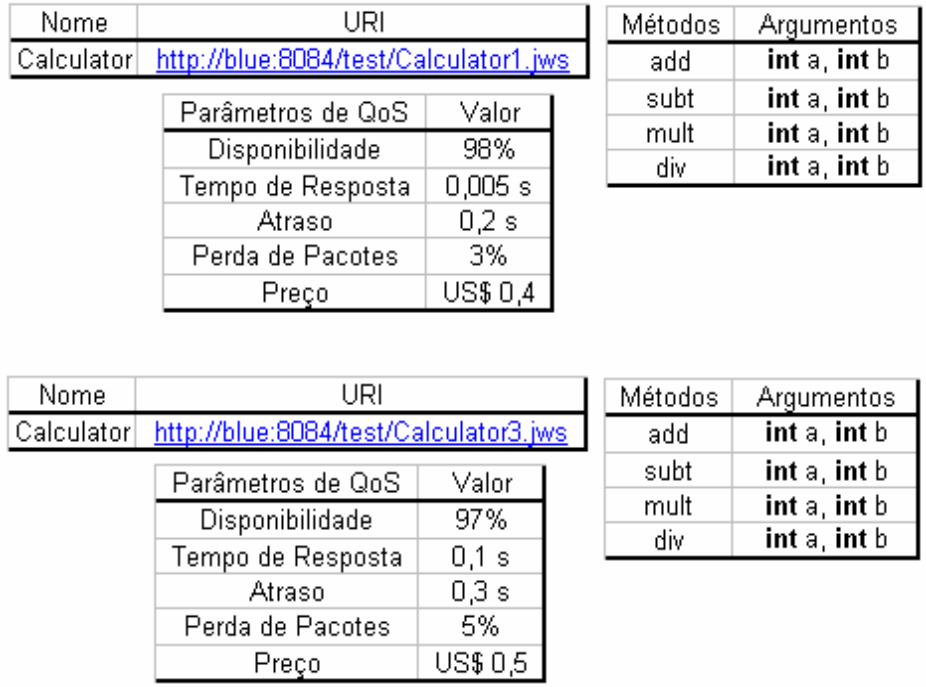

Figura 6.3 Informação dos Web Services retornados pelo Broker

Na mensagem que o *broker* retorna ao cliente é mostrada qual é o nome, URI, métodos, e parâmetros de QoS e seus valores. O cliente pode então acessar o Web Service. Quando o cliente acessar o Web Service ele pode inicializar a interface para a monitoração dos parâmetros de QoS. A interface mostra o desempenho dos parâmetros que foram definidos na arquitetura, sendo que para os parâmetros definidos pelo cliente e pelo servidor, o Esquema XML define a entidade *eMonitoracao* para definir protocolos e métodos necessários para o monitoramento dos parâmetros de QoS.

6.4. RESUMO DO CAPÍTULO

Neste capítulo foi apresentado um Estudo de Caso para a validação da implementação da arquitetura proposta nesta dissertação. O Estudo de Caso comprova que a arquitetura consegue encontrar Web Services automaticamente, de uma forma mais rápida e precisa do que a busca manual na UDDI. Com a disponibilização de informações sobre a QoS dos Web Services, o cliente pode encontrar o Web Service mais apropriado para satisfazer as suas necessidades.

O Estudo de Caso mostrou todas as etapas da validação da arquitetura. Primeiramente, foi mostrado como os Web Services e seus documentos WSDL foram criados. A seguir foram criados os documentos XML para a descrição dos parâmetros de QoS de cada Web Service, e foi feita a publicação deles na UDDI e no *broker*. Por último, foi feito um teste para cada tipo de Web Service publicado através do cliente. O teste comprovou que a arquitetura consegue encontrar o melhor Web Service para o cliente de acordo com os valores de QoS definidos pelo cliente na busca.

7. CONCLUSÕES

Neste trabalho foi especificada, implementada e validada uma arquitetura para fornecer QoS em Web Services. Os Web Services estão se tornando cada vez mais populares e a evolução dos serviços tem mostrado demanda por QoS, o que é difícil de se empregar em ambientes Web. Através da arquitetura proposta, o cliente poderá encontrar o serviço que deseja de uma forma rápida e precisa, sem a perda de tempo e dificuldades que ocorrem fazendo a busca direta no UDDI. O *broker* (intermediador) facilita o processo de busca e seleção dos serviços e a negociação de parâmetros de QoS entre o cliente e o servidor.

Para adicionar informações sobre QoS para os Web Services pode ser necessário que uma outra entidade seja adicionada à arquitetura original dos Web Services. A Seção 3.2. mostrou diversos caminhos que a comunidade científica vem explorando para adicionar informação de QoS nos Web Services. O uso de um *broker* dá flexibilidade à arquitetura, uma vez que os métodos e funções do *broker* podem ser construídos livremente, sem o engessamento que uma extensão da UDDI causaria. O *broker* também pode ser responsável por várias outras funções, como selecionar serviço, fazer o controle de admissão, armazenar informações sobre a QoS que os clientes obtiveram ao acessar determinado serviço, etc.

O uso de um Esquema XML para a definição de parâmetros de QoS possibilita a interoperabilidade dos parâmetros de QoS em Web Services. Com isso, podem ser definidos parâmetros de QoS para definir o nível da QoS que o cliente requisita e que o servidor oferece. Estabelece-se assim um cenário extensível, onde qualquer métrica pode ser definida.

7.1. PRINCIPAIS CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

a) Definição de uma arquitetura para Web Services disponibilizarem QoS
b) Definição de um Esquema XML para descrever os parâmetros de Qos
c) Integração entre o *broker* da arquitetura e o UDDI
d) Interoperabilidade da arquitetura, sendo desenvolvida como um Web Service

e) Mecanismo de seleção de serviço na arquitetura, para selecionar o melhor serviço retornado com base nos requisitos do cliente
f) Definição de um método para medir a QoS recebida por um cliente durante o acesso a um Web Service.
g) Definição de um método para qualificação dos Web Services por reputação.

7.2. VISÃO GERAL DO TRABALHO

Este trabalho abordou os conceitos básicos de XML e alguns padrões de definição de documentos (XML *schema).* Também apresentou noções de W*eb Services* e seus respectivos padrões de comunicação (SOAP), descrição (WSDL) e publicação (UDDI).

O trabalho mostrou a necessidade e a importância de se adicionar QoS em Web Services e quais são as principais dificuldades encontradas para que aconteça. Foram comentadas as várias áreas de atuação de pesquisa dos Web Services, mostrando sempre como a adição de QoS é importante, além de relacionar este trabalho com os outros trabalhos desenvolvidos.

Foi descrita uma arquitetura para busca e seleção de serviços para Web Services que utiliza um *broker* para fazer as interações cliente-UDDI-provedor. A arquitetura proporciona a utilização de parâmetros de QoS. Foi descrito também o Esquema XML criado para definir os parâmetros de QoS.

Finalmente o trabalho apresentou a implementação do protótipo, que proporciona a visualização de como ocorre o processo de publicação dos parâmetros de QoS e da requisição de um Web Service por parte de um cliente.

7.3. TRABALHOS FUTUROS

O mecanismo de reputação possui várias falhas, e pode ser violado por alguém que queria prejudicar a reputação de um Web Service. Uma extensão deste trabalho poderia ser o aperfeiçoamento do mecanismo de qualificação dos Web Services utilizando protocolos de reputação e métodos que garantem que a reputação do Web Service é verdadeira.

Existem muitos protocolos de seleção de serviço diferentes. Seria interessante analisar a questão de desempenho entre alguns desses protocolos.

Os Web Services compostos também estão em evidência, e fornecer QoS para Web Services composto constitui-se de um desafio ainda mais crítico. Assim, uma possível continuação ser estudada para adaptar a arquitetura para medir a QoS entre Web Services compostos, com a definição de algoritmos de seleção de serviço e análise de desempenho para Web Services compostos.

REFERÊNCIAS

[AXIS] Apache Axis **"Axis User´s Guide"**. Acessado em 23 de Setembro de 2005. Disponível em: http://ws.apache.org/axis/java/user-guide.html, 2005.

[BEECH 2001] BEECH David; MALONEY Murray; MENDELSOHN Noah et al. (2001) "**XML Schema Part 1: Structures"**. Recomendação W3C em Maio de 2001. Disponível em http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/ .

[BELLWOOD 2002] BELLWOOD Tom; CLÉMENT Luc; RIEGEN Claus von (2002) "**UDDI Version 3.0"**. UDDI Spec Technical Committee Specification, 19 July 2002. Disponível em http://uddi.org/pubs/uddi_v3.htm .

[GUDJIN 2003] GUDJIN Martin; HADLEY Marc; MENDELSOHN Noah et al. (2003) "**Simple Object Access Protocol (SOAP)1.2"**, Recomendação W3C em 24 de Junho de 2003. Disponível em http://www.w3.org/TR/soap12-part1/

[BRAY 2000] BRAY Tim; PAOLI Jean; MALER Eve et al. **"Extensible Markup Language (XML) 1.0 (Second Edition)"**. World Wide Web Consortium (W3C) Recommendation, October 2000. Disponível em http://www.w3c.org/TR/REC-xml .

[CATANIA 2003] CATANIA N., KUMAR P., MURRAY B., POURHEDARI H., VAMBENEPE W., WURSTER K. (2003) **"Web Services Management Framework"**, Version 2.0, Hewlett-Packard, http://devresource.hp.com/drc/specifications/wsmf/WSMF-Overview.pdf

[CERAMI 2002] CERAMI, E.: **Web Services Essentials.** 1st edition. O'Reilly & Associates (2002)

[CHEN, 2003] CHEN H., YU T., LIN K. (2003) **"QCWS: An Implementation of QoS-Capable Multimedia Web Services"**. Proc. of IEEE Fifth International Symposium on Multimedia Software Engineering (ISMSE`03), 10-12 Dec. 2003. Page(s): 38 - 45.

[CHRISTENSEN 2001] CHRISTENSEN Erik; CURBERA Francisco, MEREDITH Greg et al. **Web Services Description Language(WSDL) Version 1.1**. W3C Note 15, March 2001, http://www.w3.org/TR/wsdl.

[CONNOLLY 2001] CONNOLLY D., VAN HARMELEN F., HORROCKS I., et al. (2001) **"DAML+OIL (March 2001) Reference Description"**, W3C, http://www.w3.org/TR/daml+oil-reference, December 18, 2001.

[COYLE 2001] COYLE F. P. (2001) **Breathing life into legacy.** IT Professional Volume 3, Issue 5, Sept.-Oct. 2001. Page(s): 17 – 24.

[DAY 2004] DAY J., DETERS R., (2004). **"Selecting the Best Web Service"**. In Proceedings of the 14th Annual IBM Centers for Advanced Studies Conference (CASCON), pp. 293-307. 2004

[DAML 2003] The DAML Services Coalition: DAML-S: Semantic Markup for Web Services. "**WWW resource for DAML-S version 0.9"**. May 5, 2003. Disponível em: http://www.daml.org/services/daml-s/0.9/daml-s.html (2003)

[DEITEL 2002] DEITEL H.M , DEITEL P.J, GADZIC J. P. et al.(2002) "**Java Web Services For Experienced Programers"**. Printed by Prentice Hall, Uper Saddle River , New Jersei 07458, 2003.

[DOTNET 2005]. Microsoft .NET. **Basics of .NET.** Acessado em 15 de março de 2005. http://www.microsoft.com/Net/Basics.aspx.

[FILIPPIN 2004] FILIPPIN Cleber V.(2004) "**Proposta de Arquitetura para Distribuição e Gerenciamento de Licenças em sistemas DRM".** Dissertação de Mestrado. Programa de Pós Graduação em Ciências da Computação, Universidade Federal de Santa Catarina (PPGCC-UFSC). Florianópolis. Fevereiro, 2004.

[FOSTER 2002] Foster, I., Keselman, C., Nick, J. M., Tuecke, S.: **"Grid Services for Distributed Systems Integration"**. Computer, Vol. 35, No. 6 (June 2002). IEEE-CS (2002) 37-46

[JUDDI] jUDDI. **"jUDDI User's Guide"**. Acessado em 20 de Agosto de 2005. Disponível em http://ws.apache.org/juddi/usersguide.html.

[JWSDP] JWSDP. **"Java Web Services Development Pack".** Acessado em 10 de Agosto de 2005. Disponível em http://java.sun.com/webservices/jwsdp/index.jsp

[LUDWIG1 2003] LUDWIG H., KELLER A., DAN A., KING R., FRANCK R. (2003) "**Web Service Level Agreement (WSLA) Language Specification"**, Version 1.0, IBM Corporation, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[LUDWIG2 2003] LUDWIG H. (2003) "**Web services QoS: external SLAs and internal policies or: how do we deliver what we promise?"** Fourth International Conference on Web Information Systems Engineering Workshops, 2003. Proceedings. 13 Dec. 2003 Page(s):115 – 120.

[MAAMAR 2004] MAAMAR, Z., SHENG, Q.Z., BENATALLAH B. (2004) **"On Composite Web Services Provisioning in an Environment of Fixed and Mobile Computing Resources**". Information Technology and Management, Vol. 5, No. 3. Kluwer Academic Publishers (2004) 251-270

[MANI 2002] MANI, Anbazhagan; NAGARAJAN, Arun (2002) "**Understanding quality of service for Web services".** IBM Technical Report, Jan. 2002. Disponível em: "www6.software.ibm.com/software/developer/library/ws-quality.pdf"

[MARTIN 2001] MARTIN Didier; BIRBECK Mark; KAY Mark et al. (2001) **"Professional XML".** Rio de Janeiro: Editora Ciência Moderna Ltda., 2001.

[MENASCE 2002] MENASCE, Daniel A., (2002) **"QoS Issues in Web Services"**. IEEE Internet Computing, November-December, pp. 72-75, 2002.

[PERFECTXML, 2003]. PerfectXML. "**Web Services Introduction".** Acessado em 16 de março de 2005. Disponível em: http://www.perfectxml.com/WebSvc1.asp

[PARK, 2003] PARK, Noh-sam; LEE, Gil-haeng; (2003) "**Agent-based Web services middleware".** Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE Volume 6, 1-5 Dec. 2003 Page(s):3186 - 3190 vol.6.

[ROSSET, 2004] ROSSET, Valério (2004). "**Modelo de Arquitetura de Autorização e Distribuição de Direitos Sobre Conteúdos Digitais"**. Dissertação de Mestrado. Programa de Pós Graduação em Ciências da Computação, Universidade Federal de Santa Catarina (PPGCC-UFSC). Florianópolis, 2004.

[SYCARA 2004] SYCARA K., PAOLUCCI, M., SOUDRY J., SRINIVASAN N., (2004) "**Dynamic discovery and coordination of agent-based semantic Web services"** Internet Computing, IEEE Volume 8, Issue 3, May-Jun 2004 Page(s):66 - 73

[SCHLIMMER, 2003] SCHLIMMER, J.C. (ed.): **Web Services Description Requirements**. World Wide Web Consortium (W3C) working draft. Oct. 28, 2002. Acessado em 16 de março de 2005. http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028/

[SHAIKHALI 2003] SHAIKHALI A., RANA O. F., AL-ALI R., WALKER D. W., (2003). **"UDDIe: An Extended Registry for Web Services".** Proceedings of the

IEEE Symposium on Applications and the Internet Workshops, 27-31 January 2003. Page(s): 85 – 89.

[TIAN, 2004] TIAN M., GRAMM A., NAUMOWICZ T., RITTER H., SCHILER J. (2004) **"A Concept for QOS Integration in Web Services"**. Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03), p p. 149-155.

[TOSIC, 2003] TOSIC V., PAGUREK B., PATEL K. (2003) **"WSOL – A Language for the Formal Specification of Classes of Service for Web Services"**. Proc. of ICWS'03 (The 2003 International Conference on Web Services), Las Vegas, USA, June 23-26, 2003, CSREA Press, pp. 375-381.

[TOSIC 2004] TOSIC V. (2004) **"Service Offerings for XML Web Services and Their Management Applications"** Doctor of Philosophy (Ph. D.) in Electrical Engineering. Department of Systems and Computer Engineering, Carleton University. Ottawa, Canada, 2004.

[UDDI4J] UDDI4j **"UDDI for Java Project"**. Acessado em 15 de Agosto de 2005. Disponível em http://uddi4j.sourceforge.net/index.html.

[WEBSPHERE, 2004] Web sphere. **IBM Web Sphere Software.** Acessado em 15 de março de 2005. Disponível em: http://www-306.ibm.com/software/websphere/

[YUAN, 2003] YUAN S., LIN K. (2003) **"WISE – Building Simple Intelligence into Web Services"**. Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03), 13-17 October 2003. Page(s): 26 – 32.

[ZENG 2004] ZENG L., BENATALLAH B., NGU A. H. H., et al. (2004) **"QoS-Awre Middleware for Web Services Composition".** IEEE Transactions on Software Engineering, Vol. 30, No. 5, May 2004.

[ZHOU 2004] ZHOU C., CHIA Liang-Tien, LEE Bu-Sung (2004). **"QoS-Aware and Federated Enhancement for UDDI"**. International Journal of Web Services Research, Volume 1, Number 2, April-June 2004. Pages 58-85.