Estudo da influência de web services no desempenho de uma arquitetura orientada a serviços com QoS

Rubens Kenji Takaki Toyohara

Estudo da influência de web services no desempenho de uma arquitetura orientada a serviços com QoS

Rubens Kenji Takaki Toyohara

Orientadora: **Profa. Dra. Sarita Mazzini Bruschi**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Julho de 2011

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

T756e

Toyohara, Rubens Kenji Takaki
Estudo da influência de web services no desempenho de uma arquitetura orientada a serviços com QoS / Rubens Kenji Takaki Toyohara; orientadora Sarita Mazzini Bruschi -- São Carlos, 2011.
102 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011.

1. Web Services. 2. Avaliação de Desempenho. 3. Qualidade de Serviço. 4. Arquitetura Orientada a Serviços. I. Bruschi, Sarita Mazzini, orient. II. Título.

Agradecimentos

Agradeço primeiramente a Deus pela minha existência, pelas muitas oportunidades que surgem no meu caminho, e me conceder coragem e força para concluir mais uma etapa de minha vida. Agradeço a toda a minha família, em especial, aos meus pais, Augusto e Suely, e meus irmãos, Tsuyoshi e Pedro, por me apoiarem nesta jornada, me incentivando nos momentos difíceis. Eu tenho muito orgulho de vocês.

Quero agradecer muito à minha orientadora profa. Dra. Sarita Mazzini pela sua orientação, pela motivação, e pelo apoio em momentos de decisão.

Ao Prof. Dr. Júlio Cézar, que durante seu doutorado, idealizou a proposta deste trabalho de mestrado, e não mediu esforços para ajudar com ideias e sugestões que possibilitaram na realização deste projeto. Muito obrigado mesmo.

Agradeço à Prof. Dra. Regina Santana que me auxiliou durante as etapas de avaliação de desempenho através de seus bons olhos com números e conselhos.

Aos demais professores, tanto da UFMS, quanto do ICMC, que contribuiram com minha formação, em especial, Paulo Sérgio, Kalinka, Marcos, Monaco, Turine, Said, Debora e Sotoma.

Agradeço também aos meus amigos de república Tereré: Mario, Diogo, Gondim, Alessandro, Alex e Patrick pela amizade, pelos momentos de alegria e pela companhia em uma nova jornada. Agradeço aos demais amigos de Campo Grande, pela parceria e amizade durante a graduação, Márcio, Jucimara, Maxwell, Kishi, Lucas, Ronaldo, Rafael, Letricia. Obrigado de coração!

Muito obrigado para meus amigos do LasdPC e agregados, pelo apoio e pela hospitalidade na rep. Tibilisku nos momentos finais de conclusão da dissertação: Paulão Ipuiuna, Bruno Tardiole, Bruno Guazzelli, Daniel, João Paulo, Douglas, Luis, Ricardo, Bruno Faiçal, Priscila, Lourenço, Edwin, Roni, Pedro, e Maycon. Agradeço aos meus amigos do Seinenkai que fiz ao longo dos três anos pela descontração, e meus amigos da escalada, que proporcionaram um ótimo divertimento em diversos desafios.

Agradeço a todas as pessoas que contribuíram, de alguma forma, pela realização do projeto.

Ao CNPq, pelo apoio financeiro.

Resumo

Este projeto de mestrado tem como objetivo principal construir Web services de modo a permitir a avaliação de desempenho de uma arquitetura denominada WSARCH. A arquitetura WSARCH foi proposta de modo a prover uma infra-estrutura de Web services considerando aspectos de qualidade de serviço (QoS). Este projeto contribui diretamente com o desenvolvimento desta arquitetura, além de auxiliar na sua validação e na realização de estudos de desempenho de suas funcionalidades. Trabalhos preliminares de pesquisa foram desenvolvidos de forma que, além de auxiliar no desenvolvimento da WSARCH, também contribuíram em pesquisas relacionadas com a área de Web services. Destacam-se estudos realizados com anexos em Web services (WS-Attachments) e estudos com operações de pesquisa e publicação em repositórios UDDI. Por fim, foram realizados estudos de avaliação de desempenho com diferentes tipos de aplicação implantados em provedores que compõem a arquitetura WSARCH.

Abstract

This masters degree project has as main objective to build Web services to evaluate the performance of the WSARCH architecture. The WSARCH architecture was proposed aiming at providing Web services infrastructure implementation considering quality of services aspects. This project contributed directly with the development of this architecture, in addition to helping in validation and performance studies of WSARCHs features. Preliminary research work have been developed in order to contribute in research related to Web services besides helping in the development of WSARCH. Among them there are studies of attachments in Web services (WS-Attachments) and studies in research and publishing operations in UDDI repositories. Finally, studies of performance evaluation with different types of applications deployed on service providers that compose the architecture WSARCH.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

B2C - Business-to-Consumer

BPEL, BPEL4WS - Business Process Execution Language for Web Services

CISC - Complex Instruction-Set Computer

CORBA - Common Objetct Request Broker Architecture

DCOM - Distributed Component Object Model

DIME - Direct Internet Message Encapsulation

ESB - Enterprise Service Bus

FLOPS - Floating Point Operations per Second

HTTP - HyperText Transfer Protocol

IEEE - Institute of Electrical and Electronics Engineers

MEP - Message Exchange Patterns

MIME - Multipurpose Internet Mail Extensions

MIPS - Millions of Instructions per Second

MTOM - Message Transmission Optimization Mechanism

OASIS - Organization for the Advancement of Structured Information Standards

POP3 - Post Office Protocol

QoS - Quality of Service

RISC - Reduced Instruction-Set Computer

RMI - Remote Method Invocation

RPC - Remote Procedure Call

SLA - Service Level Agreement

SMTP	-	Simple Mail Transfer Protocol
SOA	-	Service Oriented Architecture
SOAP	-	Simple Object Access Protocol
SwA	-	SOAP with Attachments
TI	-	Tecnologia da informação
UDDI	-	Universal, Discovery, Description and Integration
W3C	-	World Wide Web Consortium
WS	-	Web Service
WS-Addressing	-	Web Service Addressing
WS-Attachment	-	Web Service Attachment
WS-Management	-	Web Service Management
WS-Reliable	-	Web Service Reliable
WS-Reliable Messaging	-	Web Service Reliable Messaging
WS-Security	-	Web Service Security
WS-Transaction	-	Web Service Transaction
WSARCH	-	Web Services Architecture
WSDL	-	Web Services Description Language
WSLA	-	Web Service Level Agreement
WSML	-	Web Service Management Language
WSOL	-	Web Service Offer Language
WWW	-	World Wide Web
XOP	-	XML-binary Optimized Packing
XML	-	eXtensible Markup Language

CAPÍTULO

1

Introdução

1.1 Considerações Iniciais

Em razão do avanço tecnológico das redes de computadores e do surgimento e popularização da Internet, são cada vez mais utilizadas aplicações distribuídas para o compartilhamento de informações. Essas informações são compartilhadas entre sistemas distintos, os quais são codificados em linguagens de programação com características diferentes e também executadas em diferentes sistemas operacionais. Inicialmente, as redes de computadores eram baseadas na comunicação cliente-servidor, depois, com o surgimento da programação orientada a objetos, novos *middlewares*[1] surgiram com o objetivo de permitir que aplicações pudessem ser escritas de maneira mais independente de hardware e sistema operacional, o que facilitaria a comunicação entre os sistemas distribuídos. Dentre os *middlewares* destacam-se o CORBA (Chiarabini, 2004), o RMI (Gray, 2004) e o DCOM (Wasznicky, 2002). Tais tecnologias permitem a comunicação entre aplicações por uma rede interna ou pela Internet, mas são limitadas no sentido de não ser possível esta interação sem o conhecimento dos recursos computacionais utilizados. Diante deste panorama, aumenta a necessidade de possibilitar a interoperabilidade entre os sistemas, o qual é possível por meio da utilização dos chamados Web Services.

Web Services é um tema bastante discutido atualmente. Sua grande popularidade, tanto na área acadêmica quanto em ambientes corporativos, é resultado da adoção de protocolos e tecnologias Web padronizados, tais como HTTP e XML, visando solucionar o problema de integração de sistemas heterogêneos presente em tecnologias similares como CORBA, DCOM e RMI. Em resumo,

[1] Termo utilizado para definir um programa mediador que interliga componentes de *software* ou aplicações em um sistema distribuído

Web Services é uma tecnologia apropriada para a comunicação entre sistemas. A comunicação entre os serviços é padronizada, o que possibilita a independência de plataforma e de linguagem de programação. Por exemplo, um sistema de desenvolvido em Java e que está sendo executado em um servidor Linux pode acessar, com transparência, um serviço feito em .Net que está sendo executado em um servidor Microsoft.

Apesar de apresentar benefícios provenientes da utilização de padrões, o uso dos Web Services sinaliza para desafios relacionados à construção de aplicações distribuídas em um ambiente dinâmico como a Internet. O arquiteto de uma aplicação distribuída necessita do conhecimento de diversos padrões envolvidos na comunicação entre os componentes de uma arquitetura orientada a serviços (cliente, provedor, UDDI e *Broker*, componentes a serem explicados posteriormente) de modo que problemas relacionados ao desempenho na integração das aplicações possam ser minimizados. Uma vez discutida a importância dos Web Services no cenário atual, o objetivo deste projeto de mestrado é a construção e avaliação de desempenho de Web Services com foco em qualidade de serviço. As aplicações desenvolvidas neste trabalho de mestrado auxiliaram na construção do protótipo de uma arquitetura orientada a serviço como foco em QoS denominada WSARCH (Estrella, 2010).

1.2 Contextualização

Os Web Services são uma solução para a integração de sistemas e na comunicação em aplicações distintas. Essa tecnologia possibilita que sistemas desenvolvidos em plataformas diferentes sejam compatíveis durante sua interação, ou então que novas aplicações sejam desenvolvidas para se comunicar com as já existentes independente da linguagem em que foi desenvolvida. Isso é possível devido à utilização de uma linguagem universal para a comunicação entre as aplicações. Os Web services permitem que as aplicações troquem informações entre si usando o formato XML. Uma aplicação com sua linguagem própria, para se comunicar com qualquer outra aplicação, traduz as informações enviadas em formato XML. A outra aplicação, por outro lado, recebe tais informações em formato XML, e os convertem para atender suas necessidades de acordo com sua linguagem e plataforma.

A tecnologia Web Service é formada por um conjunto de padrões, em geral baseados em XML. Esses padrões definem tanto os protocolos que são usados na comunicação quanto o formato das interfaces que são usadas para especificar os serviços e contratos de serviços. Dentre os padrões essenciais que fazem parte da base para o funcionamento dos Web Services, tem-se o protocolo SOAP, que encapsula os dados enviados e recebidos de uma aplicação em formato XML; e a linguagem WSDL, que é uma interface que descreve os Web services, especificando, por exemplo, o seu funcionamento e as operações e métodos disponíveis. O protocolo de transporte de dados geralmente adotado pelos Web services é o HTTP, contornando quaisquer problemas relacionados com *firewall*, por ser o padrão de protocolo utilizado pela Web. W3C (*World Wide Web Consor-*

tium) e OASIS (*Organization for the Advancement of Structured Information Standards*) são as instituições responsáveis pela padronização dos Web Services. Além disso, empresas como IBM e Microsoft, duas das maiores do setor de tecnologia, apoiam o desenvolvimento deste padrão.

As tecnologias subjacentes aos Web Services (tais como HTTP, SOAP, WSDL, UDDI, XML) são abertas, amplamente divulgadas e consensuais. No entanto, os ambientes de Web Services atuais ainda apresentam diversas limitações e não oferecem um suporte adequado à qualidade de serviço (QoS) (Yeom et al., 2009). Com a integração de Web Services como uma solução de negócios em muitas aplicações empresariais, a garantia de uma qualidade de serviço efetiva passa a ser um diferencial para que os Web Services provenham melhor atendimento aos processos de negócio, além da sua importância para propiciar alta disponibilidade e confiabilidade diante da existência da possibilidade de falha e indisponibilidade da rede de comunicação ou também dos serviços participantes (Erradi e Maheshwari, 2005).

Qualidade de serviços em aplicações que utilizam Web services é um tópico não muito explorado pelos órgãos responsáveis pela padronização da tecnologia de Web Services. Alguns trabalhos propõem possíveis soluções de arquitetura de forma isolada, dentre eles: (Ludwig, 2003), (Wu et al., 2003), (Erradi e Maheshwari, 2005), (Tsai et al., 2006), (Badidi et al., 2006), (Tavares e Westphall, 2006), (D'Mello et al., 2008), (Zhao e Tan, 2009), (Kaikai et al., 2010). Tais propostas não se preocupam em considerar a possibilidade de utilização de recursos computacionais ociosos, além de não considerar questões de avaliação de desempenho. A arquitetura WSARCH, proposta por Júlio Estrella, foi projetada para a provisão de Web services considerando atributos de qualidade de serviço, possibilitando também a avaliação de desempenho, tratando, por exemplo, de aspectos relativos às interações entre cada componente da arquitetura (Estrella, 2010).

Um dos pontos importantes a ser considerado durante a avaliação de desempenho da arquitetura WSARCH é tratar de aspectos relativos aos Web services implantados nos provedores que compõem sua estrutura, os quais podem ter diferentes características, o que pode levar a ter comportamentos distintos em termos de desempenho.

1.3 Motivação e Objetivos

Uma arquitetura orientada a serviços (SOA) é basicamente uma coleção de serviços, os quais se comunicam uns com os outros. Um serviço é uma função bem definida e que não depende do estado ou contexto de outro serviço. A comunicação entre eles pode envolver ou um simples dado ou mais serviços coordenando alguma atividade (Nickull, 2005). Como os Web Services implementam uma arquitetura orientada a serviço, o seu maior foco é fazer com que blocos funcionais sejam acessíveis sobre protocolos padrões da Internet que são independentes de plataforma e linguagem de programação.

Uma arquitetura orientada a serviço é essencialmente composta por três blocos: um provedor de serviços, um cliente que faz o acesso ao serviço e um repositório que armazena informações

sobre os serviços. A arquitetura WSARCH segue os conceitos de uma arquitetura padrão de SOA, e considera também questões de qualidade de serviço. Para isso, um novo componente denominado *broker* foi incluído na arquitetura padrão tendo como finalidade principal selecionar um serviço de forma adequada que atenda às necessidades da requisição do cliente levando em conta informações de qualidade de serviço.

Dentro do contexto de desenvolvimento de uma arquitetura orientada a serviços, os provedores de serviço podem hospedar aplicações de naturezas distintas, cada uma com suas particularidades. Tais características podem afetar negativamente o desempenho de um sistema, caso não sejam consideradas na política de escalonamento de requisições de serviço da arquitetura. Para possibilitar um melhor entendimento sobre o comportamento das aplicações, é necessário um agrupamento das aplicações em categorias de acordo com alguma característica específica. Considerando, por exemplo, a intensidade de utilização dos recursos computacionais, as aplicações podem ser classificadas em: *CPU-Bound*, *IO-Bound*, *Memory-Intensive* e *Network-Bound*. Tais informações podem ser relevantes para o desenvolvimento de algoritmos de escalonamento/roteamento de mensagens a ser utilizada pelo *Broker*, visto que alguns recursos podem sofrer sobrecarga enquanto outros permanecem ociosos, afetando, consequentemente, o desempenho do sistema.

Sendo assim, o objetivo deste projeto de mestrado é o desenvolvimento de aplicações de diferentes características considerando como base a utilização de recursos computacionais distintos. As aplicações são implementadas usando o conceito dos Web services e implantadas nos provedores de serviço disponibilizados pela arquitetura WSARCH e publicadas em um repositório de serviços para possibilitar a avaliação de desempenho da arquitetura. Além disso, como parte do escopo do projeto, o autor participa de atividades preliminares de pesquisa que são importantes para o desenvolvimento do protótipo da arquitetura. Dentre eles, estudos de desempenho relacionados com técnicas de anexos de dados em Web services (WS-Attachments) e estudos com operações de consulta e publicação em repositórios UDDI. O autor também participa ativamente no desenvolvimento do protótipo da arquitetura WSARCH e auxilia também na realização de testes de validação e de experimentos para avaliação de desempenho.

1.4 Estrutura

Esta dissertação está dividida em seis capítulos, apresentando a base conceitual necessária para o desenvolvimento deste trabalho, assim como a proposta do trabalho e experimentos realizados no decorrer deste projeto.

No segundo capítulo, são apresentados os principais conceitos relacionados a Web Services, incluindo sua arquitetura e as principais tecnologias na área, além de uma breve análise em termos de qualidade de serviço. No terceiro capítulo são abordadas as principais técnicas de avaliação de desempenho, apresentando as metodologias existentes, além de direcionar os conceitos descritos às atividades de avaliação de desempenho da arquitetura. No quarto capítulo é apresentada uma

descrição detalhada sobre a arquitetura WSARCH, assim como as aplicações implementadas e contribuições do autor dessa dissertação no desenvolvimento da arquitetura WSARCH.

No quinto capítulo são apresentados os experimentos realizados no decorrer deste projeto, os quais são descritos desde a etapa de configuração de ambiente, planejamento de experimentos, e ainda a análise de seus resultados obtidos. Nesse capítulo, são discutidas casos de testes de desempenho relacionados com anexos em Web services, operações no repositório de serviços UDDI, alguns resultados obtidos na validação da arquitetura WSARCH, e por fim a avaliação de desempenho com as aplicações desenvolvidas nesse projeto.

No sexto capítulo são apresentadas as principais conclusões obtidas no trabalho, relacionando suas vantagens e resultados. Ainda nesse capítulo é apresentada uma relação de trabalhos futuros que podem ser investigados a partir do desenvolvimento da arquitetura e seus resultados obtidos. Por fim, são relacionadas as referências bibliográficas utilizadas ao longo deste documento.

CAPÍTULO

2

Web Services e Qualidade de Serviços

2.1 Considerações Iniciais

Neste capítulo são apresentados os principais conceitos sobre Web services, incluindo sua arquitetura e as principais tecnologias da área, além de uma análise em termos de qualidade de serviço. Os conceitos apresentados neste capítulo fornecem um embasamento teórico que é de fundamental importância para o desenvolvimento do presente trabalho. Inicialmente, conceitos básicos sobre arquitetura orientada a serviços são apresentados, representando a base para os princípios da tecnologia de Web services. A seguir, diversos aspectos envolvendo Web services são considerados, desde sua definição, suas principais características, até uma descrição mais detalhada sobre os padrões fundamentais de Web services. Uma breve comparação entre Web services e tecnologias similares também é feita, discutindo as vantagens e desvantagens sobre tecnologias como CORBA e RMI. Neste capítulo também são consideradas as diferentes maneiras de se realizar troca de dados binários entre Web services através de *WS Attachments*. Esse é um típico exemplo de funcionalidade onde pode-se detectar alguns problemas de desempenho. Por fim serão considerados aspectos envolvendo Qualidade de Serviços e um exemplo prático de Web services.

2.2 Arquitetura Orientada a Serviços (SOA)

A arquitetura orientada a serviços (*Service Oriented Architecture* - SOA) é um estilo arquitetural que trata aplicações distribuídas como um conjunto de funcionalidades bem definidas em forma de serviços disponibilizados através da rede, provendo uma infra-estrutura com interfaces padronizadas. Além disso, SOA enfatiza a implementação de componentes como serviços modulares

que podem ser descobertos e usados pelos clientes (Srinivasan e Treadwell, 2005), fornecendo um repositório para permitir operações como publicação e descoberta de serviços, provendo, assim, transparência de localização.

O acoplamento fraco entre os serviços é uma das principais características presentes nos conceitos de SOA. Isto se refere ao baixo grau de dependência destes componentes sobre a lógica de negócio[1], possibilitando a manutenção dos serviços sem prejudicar a lógica do processo em si. Sendo assim, a adoção de uma arquitetura orientada a serviços facilita a adaptabilidade de sistemas, possibilitando a construção de sistemas altamente dinâmicos na medida em que os serviços podem ser substituídos (ou melhorados) de maneira transparente em tempo de execução. SOA promove também a reusabilidade de seus serviços, e permite que tais serviços sejam compostos, formando lógicas de processos mais bem estruturadas e, ao mesmo tempo, evitando dispêndio de recursos. A utilização de acoplamento fraco na arquitetura implica em vantagens como flexibilidade, escalabilidade e tolerância a falhas (Josuttis, 2008).

Considerando os aspectos de SOA discutidos anteriormente, são apresentados alguns dos princípios fundamentais que caracterizam a essência da orientação a serviços (Erl, 2004):

- Acoplamento fraco
- Reusabilidade
- Transparência de localização
- Lógica abstrata
- Autonomia de serviços
- Contrato formal de comunicação entre serviços
- Composição de serviços
- Sem estado (Propriedade *stateless*[2])

A arquitetura orientada a serviços baseia-se em interações entre entidades participantes contidas na infra-estrutura de SOA, tais como consumidores e provedores de serviços. Tais entidades estão fortemente relacionadas com conceitos fundamentais de troca de mensagens entre servidores e clientes em um sistema distribuído. Um provedor representa o papel de servidor que disponibiliza serviços (em forma de funcionalidades de negócio) na rede, enquanto consumidores são considerados clientes que utilizam os serviços disponibilizados por provedores de serviços. Uma entidade também pode assumir ambos os papéis ao mesmo tempo, possibilitando a composição de serviços. Uma interação básica entre tais entidades é ilustrada na Figura 2.1.

[1] Uma lógica de negócio é todo o processamento que se realiza dentro de uma aplicação ou serviço

[2] Os serviços devem evitar gerenciar informações de estado, pois podem ocasionar problemas de escalabilidade. (Erl, 2004)

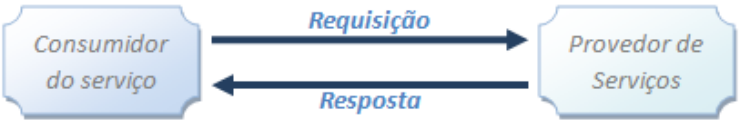

Figura 2.1: Interação Provedor-Consumidor (Bih, 2006).

Outro elemento fundamental considerado nos conceitos de SOA é o registro de serviços, definido também como um diretório central de serviços no qual, tanto o servidor, quanto o consumidor, interagem de forma que um provedor possa registrar/publicar serviços neste diretório, assim como um consumidor utiliza este registro para encontrar o serviço que ele gostaria de requisitar. Pode ser visto como uma analogia às páginas amarelas de uma lista telefônica, onde um cliente busca o telefone por tipo de serviço, e os provedores de serviço podem publicar seus serviços. Um registro também é conhecido como *broker* de serviços.

Sendo assim, em um modelo básico de SOA, podem ser definidos três tipos de entidades básicas (consumidores, provedores e registros de serviços), exercendo as operações de publicação, registro e acesso. A Figura 2.2 esquematiza este processo em SOA.

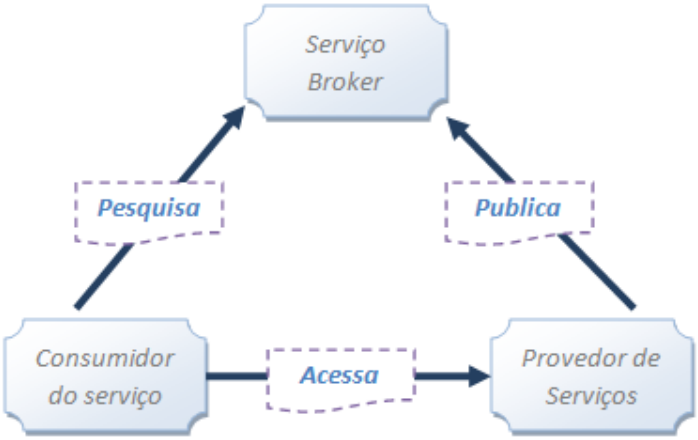

Figura 2.2: Modelo básico de SOA (Bih, 2006).

A infra-estrutura apresentada nos conceitos de SOA é geralmente chamada de **barramento de serviços corporativos** (ESB – *Enterprise Service Bus*). ESB representa uma abstração da interconexão entre diferentes sistemas, sendo responsável por envolver tarefas como: transformação de dados. roteamento, segurança, confiabilidade, monitoramento e *logging*. (Josuttis, 2008)

Existem várias tecnologias que implementam tais conceitos na prática, tais como CORBA, MQ e Tibco (Josuttis, 2008). A tecnologia Web services é a mais utilizada atualmente devido ao uso de tecnologias padronizadas baseadas em XML, que possibilitam transmitir os princípios de SOA de maneira mais concreta e eficiente em relação a outras tecnologias similares. A seção a seguir aborda os principais conceitos de Web services, assim como sua relação com a arquitetura SOA.

2.3 Conceitos de Web Services

Dentre as diferentes explicações sobre Web services presentes na literatura, o *World Wide Web Consortium* (W3C) apresenta uma definição mais completa e amplamente aceita na área de TI. A definição apresentada sofreu algumas modificações no decorrer do tempo, devido à existência de algumas inconsistências presentes em definições anteriores e para facilitar o entendimento sobre a essência do termo "Web service" e suas principais características (W3C, 2003). A mais recente definição da W3C sobre Web services até o momento é descrita a seguir:

"Um Web service é um sistema de software projetado para suportar interações máquina-para-máquina interoperáveis pela rede. Fornece uma interface descrita em um formato processável por máquina (especificamente a WSDL). Outros sistemas interagem com o Web service de maneira prescrita por sua descrição usando mensagens SOAP, tipicamente transportado utilizando o protocolo HTTP com uma serialização XML em conjunto com outros padrões relacionados com a Web" (W3C, 2004).

Sendo assim, ao contrário das aplicações Web convencionais no qual foram projetados para suportar interações aplicação-usuário (B2C), a tecnologia Web services foi desenvolvida para realizar interações aplicação-aplicação (B2B), embora também possa ser utilizada para interações com o usuário, pois operações com Web services podem ser realizadas através do navegador. Todavia, como está descrito na definição da W3C, seu foco é a interação aplicação-aplicação.

Como explicado anteriormente, o conceito de Web services engloba o uso de um conjunto de tecnologias padrão, em geral baseadas em XML. Dentre estes padrões, destacam-se SOAP, WSDL e UDDI, os quais são protocolos básicos para estabelecer uma simples conexão entre Web services.

- **WSDL** (*Web Services Description Language*): é utilizada para prover uma interface para a descrição dos Web Services. Especifica como acessá-lo e quais são os métodos (funcionalidades) disponíveis. (Thomas et al., 2003)

- **SOAP** (*Simple Object Access Protocol*): a interação entre Web services ocorre através da troca de mensagens SOAP, permitindo a comunicação de forma simples e independente de linguagem de programação e plataforma. (Papazoglou e Georgakopoulos, 2003)

- **UDDI** (*Universal Description, Discovery, and Integration*): Funciona como um serviço de diretório que contém as descrições dos Web Services, possibilitando a publicação e localização de serviços neste diretório. (Farkas e Charaf, 2003)

Os padrões citados anteriormente também são conhecidos como especificações de Web services de primeira geração, pois representam a base para a construção de uma aplicação simples orientada a serviços. Tais tecnologias serão abordadas mais adiante. A segunda geração de especificações, também conhecida como "WS-*", representa um complemento para suprir as limitações

existentes ao utilizar apenas as tecnologias básicas dos Web services e também prover novos recursos e funcionalidades (Erl, 2004). Dentre estas especificações, estão incluídas WS-Security, WS-ReliableMessaging, WS-Transaction, etc.

2.3.1 Pilha conceitual dos Web services

Para facilitar o entendimento sobre as especificações pertencentes ao conceito de Web services, estas são categorizadas em camadas de uma pilha conceitual (Silva e Rosa, 2006). Assim como os conceitos aplicados em teoria de rede, as camadas da pilha conceitual são dispostas de forma a prover serviços para camadas superiores e utilizar recursos das camadas inferiores. Com o surgimento de novos padrões, novas propostas de pilhas conceituais vão surgindo para atender tais adaptações (IBM, 2009),(Microsoft, 2009),(Innoq, 2007). Para este projeto, a pilha conceitual proposta pela IBM foi adotada por apresentar os conceitos de forma genérica e completa, englobando os padrões básicos e as principais especificações emergentes (IBM, 2009) (Figura 2.3).

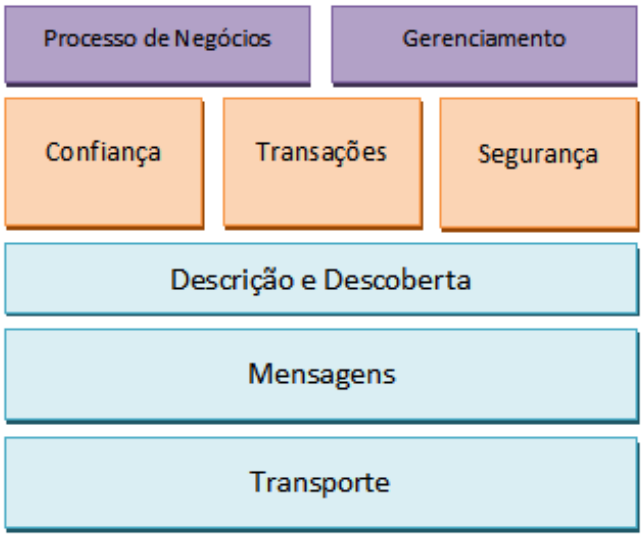

Figura 2.3: Pilha conceitual de Web services (IBM, 2009).

Camada de transporte : Consiste de protocolos responsáveis pela transmissão de dados pela rede. Ex. HTTP, SMTP, MIME, etc.

Camada de mensagens : Permite que a troca de mensagens seja feita em um ambiente descentralizado e distribuído de forma interoperável. Ex. SOAP, WS-Addressing, WS-Attachments, etc.

Camada de descrição e descoberta : Web services são significativos apenas se potenciais usuários possam encontrar informações suficientes para permitir sua execução. O foco dessas especificações e padrões é a definição de um conjunto de serviços para suportar: a descrição

e descoberta de negócios, organizações, e outros provedores de Web services; os Web services que são disponibilizados; e as interfaces técnicas para que tais serviços sejam acessados. Ex. WSDL; UDDI.

Camadas de confiabilidade, transação e segurança : Representam aspectos importantes para projetar e desenvolver aplicações distribuídas seguras e confiáveis. WS-ReliableMessaging, WS-Transaction e WS-Security são as principais tecnologias envolvidas nestas camadas.

Camadas de processo de negócios e gerenciamento : Um processo de negócios especifica a ordem em que serão executadas as operações de um conjunto de Web services, os dados compartilhados entre esses Web services, e quais são as parcerias envolvidas no processo de negócios. A linguagem BPEL (*Business Process Execution Language*) especifica processos de negócios e como se faz composição de Web services. WS-Management é responsável pelo gerenciamento dos recursos envolvidos e da configuração do ambiente.

2.3.2 SOA e Web services

Para o desenvolvimento de uma arquitetura de Web services baseada nos conceitos de orientação a serviços (SOA), devem ser levados em consideração os principais protocolos e especificações utilizadas para a tecnologia de Web services, como mostra a Figura 2.4. As interações entre clientes e serviços são realizadas basicamente por mensagens SOAP, obedecendo a interface definida pelo documento WSDL do serviço. As operações de publicação e descoberta de serviços, por outro lado, são definidas pela especificação UDDI, na forma de um repositório de descrições de serviços. Os provedores de serviço publicam seus documentos WSDL, enquanto o cliente realiza a busca pela WSDL do serviço requisitado em um registro UDDI.

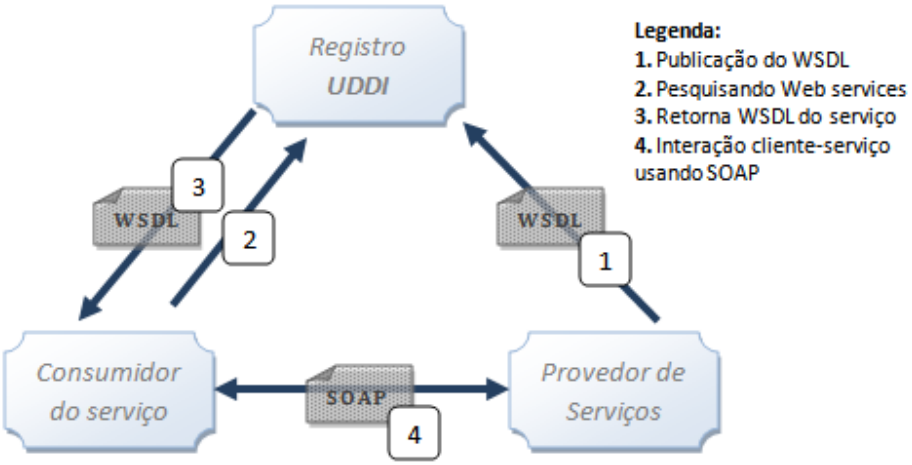

Figura 2.4: Arquitetura de Web services

2.4 Web Services e Tecnologias Similares

Os Web services foram projetados para solucionar problemas presentes em outros protocolos de programação distribuída, principalmente envolvendo integração e interoperabilidade. RMI (*Remote Method Invocation*), por exemplo, é bastante atrelado à linguagem de programação. Todas as aplicações envolvidas devem ser implementadas em Java para estabelecer uma conexão entre eles. A tecnologia CORBA (*Common Object Request Broker Architecture*), por outro lado, pode ser implementada em diferentes linguagens de programação, mas utiliza protocolos de transporte complexos para realizar comunicação entre objetos distribuídos, apresentando, assim problemas de compatibilidade com *firewalls* e *proxies* de Internet. Deste modo, são necessários mecanismos adicionais para resolver esses problemas com firewall de forma adequada (Chiarabini, 2004).

Sendo assim, os Web services solucionam tais problemas utilizando protocolos baseados no padrão XML, permitindo independência de plataforma e linguagem de programação, e também utilizando o protocolo HTTP (ou outros padrões livres como SMTP ou FTP) para realizar a transmissão de mensagens, pois não existe a interferência de firewalls neste protocolo (Sotomayor, 2005).

Entretanto, Web services também possuem algumas desvantagens sobre as outras tecnologias similares. A principal delas consiste no uso da linguagem XML para a transmissão de dados, o qual implica em perda de eficiência em relação ao uso de um código binário proprietário, pois exige-se um custo considerável para o processamento de mensagens em XML. Entretanto, essa perda de desempenho geralmente é aceitável para muitas aplicações. O que se perde em eficiência, se ganha em portabilidade (Sotomayor, 2005).

2.5 Padrões fundamentais de Web Services

Os Web services são essencialmente baseados em três padrões fundamentais: SOAP, WSDL e UDDI. Em resumo, SOAP permite que aplicações comuniquem entre si de forma interoperável. A WSDL oferece uma linguagem comum para descrever os serviços, e o UDDI fornece mecanismos necessários para publicar e localizar os serviços em um repositório. Tais protocolos serão abordados a seguir com mais detalhes.

2.5.1 Protocolo SOAP

O protocolo SOAP (*Simple Object Access Protocol*) é um dos principais componentes da tecnologia Web services. É a especificação responsável pela troca de informações entre aplicações em um ambiente descentralizado e distribuído (SOAP, 2007). Para permitir a comunicação entre duas aplicações de forma interoperável, este protocolo estabelece um formato de mensagem padrão baseado em XML. Inicialmente, SOAP foi elaborado para dar suporte apenas ao protocolo

HTTP. Entretanto, na sua versão 1.1, diferentes tipos de protocolos, tais como SMTP e POP3, dentre outros, podem ser adotados como meios de transporte de mensagens (Brooks, 2002).

Atualmente, o protocolo SOAP, que encontra-se na versão 1.2, garante simplicidade e extensibilidade. A extensibilidade é oferecida no sentido de prover mecanismos para adicionar novas funcionalidades (tais como: confiabilidade, segurança, correlação, padrões de troca de mensagens, etc.) de maneira simples e prática (SOAP, 2007).

A troca de mensagens SOAP pode ser modelada como um simples padrão requisição-resposta, no qual as aplicações envolvidas nesta interação definem como as mensagens serão interpretadas utilizando suas próprias regras de comunicação; ou então pode ser definida como um modelo específico de uma chamada de procedimento remoto (RPC). Utilizando este modelo de comunicação, o cliente envia uma mensagem SOAP contendo a chamada e os parâmetros do procedimento requisitado, e o servidor retorna uma mensagem contendo os resultados desse procedimento. Sendo assim, ao contrário das interações mais convencionais, que oferecem maior flexibilidade, as mensagens que utilizam o estilo RPC devem seguir algumas regras pré-estabelecidas, além de algumas informações adicionais necessárias (nome do método, parâmetros, valor de retorno, etc.). Este estilo de comunicação é mais utilizado quando deseja-se implementar um certo modelo programático, similar à implementação de um procedimento tradicional. Documentos SOAP que não utilizam regras no estilo RPC, são denominados **estilo literal** (*document-style*).

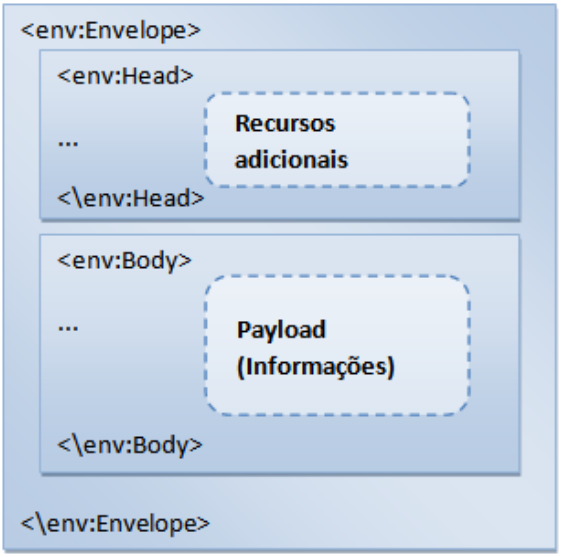

Figura 2.5: Estrutura de uma mensagem SOAP.

A estrutura de uma mensagem SOAP consiste basicamente de um recipiente externo denominado Envelope SOAP, que, por sua vez, é separado por duas áreas internas: uma representando o corpo da mensagem, e a outra representando o cabeçalho da mensagem (Figura 2.5). O Bloco de cabeçalho, representado pelo construtor Head, é uma área opcional e geralmente é utilizado para a implementação de extensões SOAP (pré-definidas ou específica de aplicação), tais como extensões introduzidas pela segunda geração de especificações; e também para prover meta informações

suplementares sobre a mensagem SOAP. A área obrigatória da mensagem é representada pelo corpo da mensagem, definida pelo construtor Body. Esta seção age como um recipiente para os dados que estão sendo enviados pela mensagem SOAP. Os dados contidos no corpo SOAP geralmente são referenciados como *payload* (carga útil). O tratamento de falhas também pode ser implementado dentro do corpo SOAP, fornecendo as informações necessárias para descrever os erros, e sobre como as excessões serão tratadas. (Erl, 2004)

2.5.2 Linguagem de Descrição WSDL

WSDL é um padrão da W3C que provê recursos para descrever Web services utilizando o formato XML. Web services precisam ser definidos de uma maneira consistente para assim serem descobertos e estabelecerem conexões com outros serviços e aplicações. Esta especificação permite estabelecer a comunicação entre os Web services provendo descrições para as localizações de serviços (*endpoints*) padronizados, e assim, fornecer as informações necessárias para promover a integração (Erl, 2004).

A especificação WSDL define uma linguagem para descrever cada funcionalidade oferecida por um serviço, assim como detalhes mais concretos de uma descrição de serviço, tais como "onde" e "como" uma funcionalidade específica é oferecida. Sendo assim, a WSDL descreve um Web service basicamente em dois estágios fundamentais: um abstrato e um concreto. Cada estágio contém um conjunto de componentes que definem a descrição de forma organizada no sentido de separar os diferentes propósitos e promover reusabilidade das descrições. (W3C, 2007)

O nível **abstrato** representa a definição da interface de serviço. Descreve quais são as operações disponíveis, quais são os parâmetros de entrada e saída que uma determinada operação utiliza para enviar e receber mensagens, e como as mensagens envolvidas são descritas. As operações e seus respectivos parâmetros são definidos pelo construtor interface e as mensagens são descritas pelo construtor message. Cada elemento representado por uma interface WSDL contém um conjunto de operações logicamente relacionadas. Este esquema é similar à estrutura de uma arquitetura orientada a componentes, no qual uma operação de uma interface equivale a um método de um componente.

O nível **concreto** define os detalhes para a implementação de um serviço. Contém dados relacionados às informações de protocolo e localizações de serviço (definindo um endereço físico), assim como permite vincular tais pontos de acesso sobre determinadas operações definidas anteriormente no documento WSDL. A descrição das informações concretas podem ser definidas pelos componentes a seguir: service, representando uma coleção de localizações de serviços; endpoint, contendo informações sobre uma localização de serviço individual; e binding, referenciando os vínculos entre *endpoints* e operações.

Existem também componentes suplementares para fornecer informações adicionais e novas funcionalidades, tais como o elemento types, que permite construir tipos de dados para dar suporte às definições de Web services; e o elemento documentation, permite incluir anotações adicionais

ao documento WSDL. Todos esses componentes citados anteriormente devem ser incluídos dentro de um componente denominado definitions, que representa as definições dos Web services em um documento WSDL. A Figura 2.6 ilustra a estrutura de um documento WSDL.

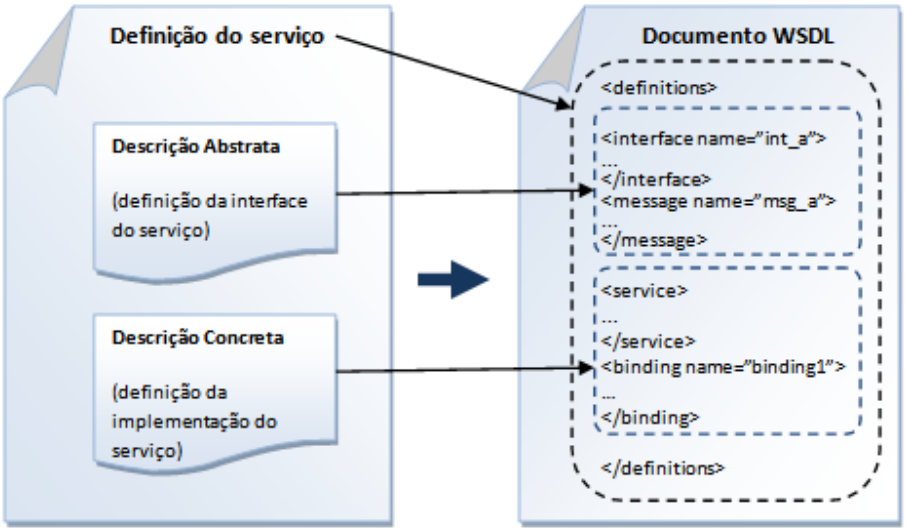

Figura 2.6: Estrutura de uma documento WSDL (Erl, 2004).

2.5.3 Repositório de Informações de Serviço UDDI

O protocolo UDDI, do acrônimo *Universal Description, Discovery, and Integration*, é um padrão especificado pela OASIS[3] (*Organization for the Advancement of Structured Information Standards*) que provê um mecanismo para publicar e descobrir provedores de serviços em uma arquitetura orientada a serviços. Quando Web services são construídos, estes serviços necessitam serem acessados em algum lugar da rede por um cliente. Uma das maneiras é fazer com que a aplicação conheça a URI[4] do serviço, caracterizando-se, assim, um modo estático para se localizar e acessar um serviço (Reckziegel, 2006). Entretanto, pode-se considerar que, à princípio, uma aplicação Web não possui a localização de um serviço na Web. Utilizando mecanismos providos pelo UDDI, serviços podem ser descobertos, antes de serem acessados, e, desta forma, provendo meios dinâmicos para se localizar um serviço.

A abordagem UDDI é baseada em um registro distribuído de descrições de serviços, implementados em um formato XML comum (geralmente WSDL). O principal componente desta especificação é um diretório de informações de serviço, também conhecido como registro UDDI, que corresponde a um documento XML utilizado para descrever informações de Web services de forma organizada. Mais especificamente, as informações providas pelo registro UDDI são organizadas em três componentes (Gomes e Gonçalves, 2004):

[3]Organização responsável pela especificação de padrões na Web
[4]Identificador de referências universal

- Páginas Brancas: possuem informações sobre nomes, endereços, números de telefone, além de outras informações sobre os fornecedores do serviço.
- Páginas Amarelas: contêm listagens comerciais baseadas nos tipos desses negócios, de maneira organizada por categoria específica ou regiões demográficas. Tais categorizações são baseadas em taxonomias.
- Páginas Verdes: são usadas para indicar os serviços oferecidos por cada negócio, incluindo todas as informações técnicas envolvidas na interação com o serviço.

Tais componentes conceituais estão, na prática, implementados no formato XML através de quatro elementos básicos (Pinto et al., 2003): (a) businessEntity, que contém informações sobre a companhia que publica o serviço; (b) businessService, com informações sobre os serviços; (c) bindingTemplate, com informação sobre o acesso aos serviços; e (d) tModel, usado para descrever especificações técnicas do serviço.

Os registros UDDI podem ser implementados de diferentes maneiras, considerando quais tipos de usuários irão consumir os serviços publicados nestes registros. Sendo assim, os registros podem ser públicos, privados ou internos, determinando restrições para grupos de usuários específicos (Erl, 2004). Um registro público representa um diretório global de descrições de serviços de negócios internacionais. Instâncias deste registro geralmente são sustentados por grandes corporações em uma série de servidores UDDI dedicados. Registros privados são repositórios de descrições de serviços hospedados dentro de uma organização. Parceiros de negócio externos também podem ser autorizados a acessar esse diretório. Um registro restrito a usuários internos pode ser referenciado como um registro interno.

A especificação UDDI também provê um conjunto de APIs programáticos agrupados em duas categorias gerais: pesquisa e publicação. Com isso, é possível gerar uma mensagem SOAP para pesquisar, por exemplo, o nome de uma companhia no repositório. (Erl, 2004)

2.6 Anexo de dados em Web Services

Assim como outros tipos de protocolos de comunicação, é possível enviar qualquer tipo de informação usando Web Services, esteja esta em qualquer formato de codificação. Para este fim, existem algumas técnicas que permitem o encapsulamento de informações em mensagens SOAP. Basicamente, existem duas maneiras de se anexar informações em documentos *SOAP/XML*:

- Armazenar estas informações na própria mensagem *SOAP*. Esta técnica é conhecida como *Pure Binary*. É necessário converter estes dados para modo texto usando o tipo xs:base64-Binary ou xs:hexBinary, tipos de dados pré-definidos do XML para referenciar codificação *base64* em modo texto e modo binário. A principal desvantagem desta abordagem é que este encapsulamento de dados no envelope *SOAP* causa sobrecarga no custo de processamento, principalmente ao decodificar a mensagem para o modo binário.

- Usar uma *URI* do tipo xs:anyURI no documento *XML* para referenciar os dados binários. As informações são anexadas fora da mensagem *SOAP*, sendo identificadas por uma *URI*. Tais dados são empacotados usando mecanismos como *DIME* ou *MIME* (os mesmos utilizados para anexos por e-mail). Neste trabalho são apresentadas duas técnicas que empacotam dados externamente à mensagem *SOAP*: *MTOM* e *SwA*.

2.6.1 Técnica MTOM

MTOM (*Message Transmission Optimization Mechanism*) (Gudgin et al., 2005) é um dos métodos existentes que envia dados binários a partir de uma mensagem SOAP. Sua base é o **XOP** (*XML-binary Optimized Packing*), um protocolo usado para transmitir dados binários na mensagem SOAP. XOP é uma alternativa de serialização [5], para se comportar de forma semelhante a um pacote MIME de múltiplas partes, sendo o documento XML o elemento raiz. Este documento difere de mensagens SOAP usando *Pure Binary* pelo fato dos dados serem substituídos por referências de partes MIME, que não são codificados em *base64*.

O processo utilizado pelo protocolo XOP (Hass, 2005) é representado por duas etapas: criação de pacote XOP e interpretação de pacote XOP. Na primeira etapa, o pacote XOP é construído a partir da serialização dos dados binários, ou seja, um arquivo binário é dividido em várias partes. Cada uma dessas partes é referenciada por uma URI, identificando cada localização correspondente. Por fim, tais referências são agrupadas em uma estrutura denominada *infoset* XML. A etapa de interpretação do pacote XOP realiza o processo inverso, desserializando os dados binários a partir das referências armazenadas no *infoset*.

MTOM especifica uma implementação concreta deste mecanismo para o envio de pacotes SOAP no protocolo HTTP, e usa o mecanismo XOP para otimizar a transmissão de mensagens SOAP com anexos. No documento XML é descrito como serializar o envelope SOAP usando o formato XOP e o empacotamento MIME Multipart/Related (Nottingham, 2004). XOP e MTOM são padrões *W3C* desde 2005, mas não são propostos como parte da especificação original de SOAP.

Para identificar MTOM na transmissão da mensagem SOAP, são necessários quatro atributos (Yang, 2007):

1. A mídia[6] application/xop+xml é obrigatória.

2. O tipo de mídia da mensagem HTTP é multipart/related.

3. O tipo de mídia para o elemento raiz do pacote MIME multipart/related é application/xop+xml

[5] Processo de transformação de objetos Java, sejam em formato binário, ou em formato de texto como o XML, para posterior armazenamento em disco, buffer de memória ou transmissão na rede de comunicação

[6] Mídia – termo utilizado para representar tipos de arquivos em XML

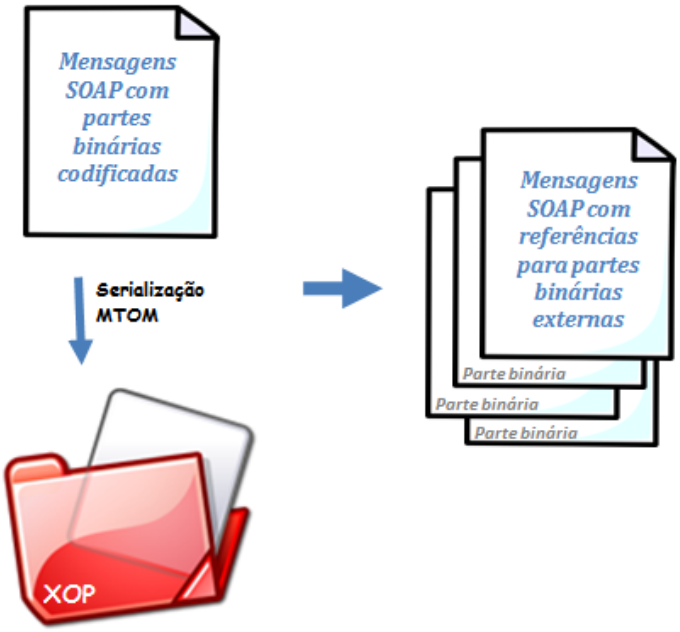

Figura 2.7: MTOM (Hass, 2005)

4. O parâmetro start-info indica o *content-type* de application/soap+xml.

Assim, um único objeto binário pode ser dividido em partes, onde cada parte possui uma referência. Tais referências são agrupadas em um *infoset XML* dentro de um documento SOAP. A Figura 2.7 ilustra uma abstração do mecanismo MTOM usando empacotamento de dados em XOP.

2.6.2 Técnica SwA

A técnica SwA (*SOAP with attachments*) define como uma mensagem SOAP pode ser composta em uma mensagem MIME do tipo multipart/related. Cada parte da mensagem é separada por um único *string* delimitador definido no começo da mensagem (Estrella, 2008). A diferença entre MTOM e SwA é que o SwA não mostra a referência do objeto como um *infoset XML*, ao contrário do que o MTOM faz. Sendo assim, a representação do conteúdo binário anexado não está em formato XML. A Figura 2.8 ilustra um envelope SOAP anexando uma imagem no formato png.

Este arquivo se encontra fora do envelope SOAP, delimitado pelo mecanismo *MIMEType*, e possui um *ID* que é referenciado pelo campo href da mensagem SOAP.

Um dos principais problemas no modelo SwA é não ser compatível com alguns conceitos relacionados aos Web Services, pelo fato de não utilizar o formato XML para tratar do anexo de dados. Por exemplo, o SwA não é compatível com o formato WS-Security, dificultando o envio de dados binários anexados de forma segura, principalmente quando se trata de troca de dados entre múltiplas organizações.

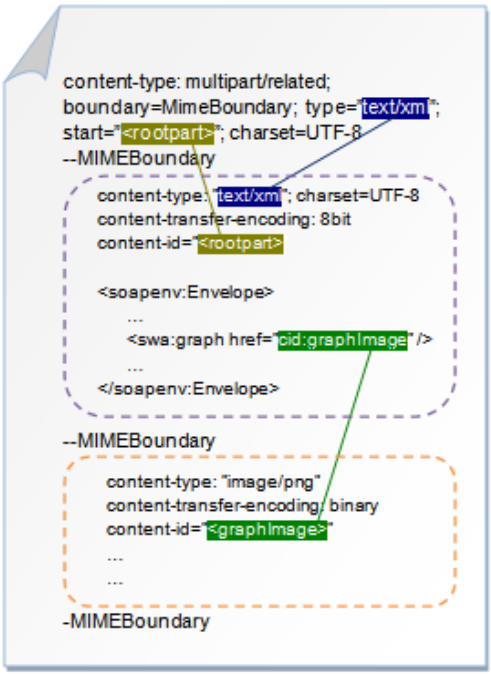

Figura 2.8: SWA (Gunarathn, 2007)

2.6.3 Comparação entre as técnicas

Como foi discutido anteriormente, existem várias formas de se enviar anexos via Web services. Uma breve comparação apresentando as vantagens e desvantagens das técnicas descritas nesta seção é discutida abaixo:

- **base64binary**: os dados anexados são enviados dentro da mensagem SOAP, aumentando-se consideravelmente o tamanho da mensagem SOAP, impactando diretamente no processamento desta.
- **SwA**: envia-se os dados anexados fora da mensagem SOAP (a mensagem SOAP contém apenas uma referência ao anexo). Isso resolve problemas de desempenho apresentadas na técnica anterior. No entanto isso dificulta na utilização de outras funcionalidades WS-*, gerando problemas de compatibilidade.
- **MTOM**: tenta resolver os problemas de compatibilidade do SwA e de desempenho do base64binary. Os anexos também são enviados fora da mensagem SOAP com uma referência, mas esses aparecem como se estivesse incorporados na mensagem, utilizando uma estrutura denominada *SOAP infoset*. Não é tão eficiente em termos de desempenho quanto a técnica SwA, mas permite, por exemplo, encriptação utilizando *WS-Security*.

Um estudo mais aprofundado em relação ao desempenho das técnicas de *attachment* é descrita na Seção 5.2.

2.7 Exemplo de Web Service

Os Web services podem ser implementados de diversas maneiras, oferecendo certa flexibilidade para o programador, de forma a permitir a escolha de um ambiente adequado, assim como a adoção de uma linguagem de programação conveniente. Existem diversas interfaces de programação de aplicações que fornecem recursos que facilitam a implementação de Web services. Uma delas é o **Apache Axis2** (Axis2, 2009), que possibilita o processo de criação e publicação de serviços de forma automatizada, tornando a tarefa mais simples para o programador. Para o exemplo apresentado nesta seção, a linguagem Java, assim como a ferramenta Axis2, foram adotadas, pois oferecem um bom suporte para a tecnologia Web services, além de serem uma das combinações de ferramentas mais utilizadas atualmente para esta finalidade.

O exemplo apresentado nesta seção é baseado na implementação disponibilizada por (Matos, 2008), que implementa uma calculadora básica que fornece as quatro operações básicas aritméticas: soma, subtração, multiplicação e divisão. Espera-se que este serviço aceite dois valores inteiros como parâmetros de entrada, e retorne para o cliente o resultado da operação requisitada. Inicialmente, é necessário criar uma aplicação que execute as quatro operações básicas. Neste exemplo, o serviço é implementado usando apenas uma classe Java com os métodos necessários para as operações, como é ilustrado na Figura 2.9.

```
public class Calculadora{
    public Integer soma(Integer op1, Integer op2)
    {
        return op1+op2;
    }
    public Integer subtracao(Integer op1, Integer op2)
    {
        return op1-op2;
    }
    public Integer multiplicacao(Integer op1, Integer op2)
    {
        return op1*op2;
    }
    public Integer divisao(Integer op1, Integer op2)
    {
        return op1/op2;
    }
}
```

Figura 2.9: Código do serviço **Calculadora**

Quando um Web service é construído utilizando-se a ferramenta Axis2, sua configuração deve ser descrita em um arquivo denominado **service.xml**, que deve estar contido no diretório META-INF dentro da estrutura de diretórios do serviço. Este arquivo de descrição contém informações

como o nome da classe que implementa o serviço, as operações disponibilizadas por este serviço, e como será a forma de passagem de parâmetros.

Com todos os arquivos e diretórios necessários criados, o serviço pode ser publicado. O processo de publicação pode ser realizado enviando-se todos os arquivos gerados em um servidor, ou então empacotando-se os documentos envolvidos em um único arquivo "*.aar" (arquivo Axis), que é uma versão compacta do serviço. Sendo assim, este arquivo pode ser enviado no lugar dos demais arquivos para o servidor, facilitando o processo de publicação do serviço. O servidor pode ser o próprio Axis2, ou então um outro tipo de servidor de *servlet*[7] como o Apache Tomcat. A ferramenta Axis2 geralmente é utilizada em conjunto com Apache Tomcat, pois esse servidor possui uma estrutura sólida, além de permitir um gerenciamento extensivo de suas características e ainda fornece suporte a várias tecnologias. A configuração do Axis2 como servidor não apresenta grande expressividade quanto à variedade de configurações e também não fornece suporte à alta disponibilidade (Matos, 2008).

Além de permitir a criação e publicação de um serviço, a ferramenta Axis2 também possui funcionalidades para auxiliar a construção de consumidores de serviços, ou seja, uma aplicação cliente que utilizará algum método do serviço publicado. Para isso, é necessária a utilização do documento **WSDL** do serviço que se deseja acessar. O **WSDL** do serviço pode ser obtido através de uma URI disponibilizada pelo serviço, ou então com ferramentas específicas que possibilitam a geração de um arquivo WSDL através do código-fonte do serviço. Os documentos WSDL também podem ser encontrados através de registros de serviços UDDI, como foi abordado anteriormente.

Os componentes responsáveis pela parte da interação entre clientes e serviços através de mensagens SOAP são denominados ***stubs***. Os *stubs* realizam toda a parte de criação de mensagens SOAP, configuração do protocolo de transporte e geram as conexões necessárias para permitir o envio e recebimento das mensagens SOAP envolvidas no acesso ao serviço. A Figura 2.10 contém o código-fonte do cliente que executa a chamada dos métodos do serviço publicado. As classes **CalculadoraStub** e as suas derivadas são automaticamente geradas pelo Axis2 através da **WSDL** do serviço, facilitando o desenvolvimento do serviço.

Com a execução da aplicação cliente, os resultados das quatro operações envolvendo os parâmetros 10 e 5 são impressos na tela. Esse é um exemplo simples de serviço, mas ilustra a utilização de ferramentas que auxiliam a implementação de Web services.

2.8 Qualidade de Serviços (QoS)

O termo "Qualidade de Serviço" (QoS) é referenciado como um conjunto de propriedades não-funcionais de Web services, tais como desempenho, confiabilidade, disponibilidade e segurança. Com a proliferação dos Web services, medidas de qualidade de serviço são utilizadas como uma

[7] *Servlets* são classes Java que podem ser utilizadas através da *Web* com passagem de parâmetros para sua execução feita via ***HTTP***

```
Import org.apache.ws.axis2.CalculadoraStub.*;
Import org.apache.ws.axis2.CalculadoraStub;
public class Calcular{
    public static void main(String[] args)
    {
        Calcular c = new Calcular();
        try{
            System.out.println(c.somar(10,5));
            System.out.println(c.dividir(10,5));
            System.out.println(c.subtrair(10,5));
            System.out.println(c.multiplicar(10,5));
        }catch(Exception e)
        {
            System.out.println("Erro");
        }

    }

    public String somar(Integer op1, Integer op2)
        throws Exception
    {
        CalculadoraStub stub = new CalculadoraStub();

        //Cria a requisição para o serviço
        CalculadoraStub.Soma request;
        request = new CalculadoraStub.Soma();
        request.setParam0(op1);
        request.setParam0(op2);

        //Invoca o serviço
        SomaResponse response;
        response = stub.soma(request);
        return String.valueOf(response.get_return());
    }

    public String dividir(Integer op1, Integer op2)
        throws Exception
    {
        CalculadoraStub stub = new CalculadoraStub();

        //Cria a requisição para o serviço
        CalculadoraStub.Divisao request;
        request = new CalculadoraStub.Divisao();
        request.setParam0(op1);
        request.setParam0(op2);

        //Invoca o serviço
        DivisaoResponse response;
        response = stub.divisao(request);
        return String.valueOf(response.get_return());
    }

    public String subtrair(Integer op1, Integer op2)
        throws Exception
    {
        CalculadoraStub stub = new CalculadoraStub();

        //Cria a requisição para o serviço
        CalculadoraStub.Subtracao request;
        request = new CalculadoraStub.Subtracao();
        request.setParam0(op1);
        request.setParam0(op2);

        //Invoca o serviço
        SubtracaoResponse response;
        response = stub.subtracao(request);
        return String.valueOf(response.get_return());
    }

    public String multiplicar(Integer op1, Integer op2)
        throws Exception
    {
        CalculadoraStub stub = new CalculadoraStub();

        //Cria a requisição para o serviço
        CalculadoraStub.Multiplicacao request;
        request = new CalculadoraStub.Multiplicacao();
        request.setParam0(op1);
        request.setParam0(op2);

        //Invoca o serviço
        MultiplicacaoResponse response;
        response = stub.multiplicacao(request);
        return String.valueOf(response.get_return());
    }
}
```

Figura 2.10: Código-fonte de um cliente para o serviço Calculadora

aferição para diferenciar serviços e seus provedores (Kalepu et al., 2003). No entanto, garantir QoS em aplicações Web é um grande desafio devido à sua heterogeneidade e natureza imprevisível (Farkas e Charaf, 2003). Dentro do contexto de Web services, QoS é composto por técnicas que visam balancear as necessidades dos requisitantes de serviço com as funcionalidades disponibilizadas pelos provedores de serviço, considerando suas limitações de rede e de recursos computacionais (Mani e Nagarajan, 2002).

A linguagem de definição da interface (WSDL) em uma arquitetura de Web services descreve apenas propriedades sintáticas de um serviço, mas não especifica parâmetros relacionados com qualidade de serviço. Web services com suporte à QoS geralmente estão associados com um **acordo de níveis de serviço** (SLA – *service level agreement*), no qual é utilizado para garantir quantificação de desempenho (Kalepu et al., 2003). SLA é um vínculo formal que define a relação entre o provedor de serviço e o consumidor, geralmente envolvendo um terceiro componente

responsável pela garantia de contrato (Sun et al., 2006). Ele consiste de descrições de parceiros de negócio envolvidos, definições de serviço, parâmetros de qualidade de serviço, e possíveis penalidades em caso de ocorrência de falhas para atender os diferentes níveis de acordo. SLAs em Web services têm sido desenvolvidos para facilitar relacionamentos complexos entre provedores e para garantir desempenho de aplicação (Kalepu et al., 2003).

Dentre alguns exemplos relacionados com condições em que um SLA pode conter para oferecer QoS em um serviço de agência de viagem, incluem (Menascé, 2002):

- O tempo de resposta médio para uma requisição ao método GetFlightAvailability, responsável pela verificação da disponibilidade de vôo, não pode passar de 0.5 segundos;
- Pelo menos 95% das requisições para o serviço Book-Flight, que disponibiliza aos usuários um conjunto de operações relacionadas à consulta de informações de vôo, devem estar concluídas em menos de dois segundos;
- O Web service responsável pela reserva de vôos deve estar disponível em pelo menos 99.9% do tempo.

Atualmente, existem algumas linguagens de especificação de QoS que oferecem suporte aos artefatos de SLA, tais como **WSML** (*Web Service Management Language*) da HP, e **WSLA** (*Web Service Level Agreement*) da IBM; e existe também uma linguagem denominada **WSOL** (*Web Service Offer Language*), que provê diferentes classes de serviço pré-definidos para distinguir os diferentes tipos de usuários. (Bianco et al., 2008)

Diferentes tipos de atributos de QoS podem ser aplicados em um serviço Web, fornecendo um certo nível de qualidade para seus usuários. Algumas das definições dessas métricas são apresentadas a seguir (Kalepu et al., 2003):

- **Disponibilidade**: Disponibilidade é um aspecto de qualidade no qual um Web service está presente ou pronto para uso imediato, representado como uma porcentagem de tempo disponível (*uptime*) de um serviço em um período de observação e está relacionada com sua confiabilidade. Para serviços freqüentemente acessados um pequeno valor de período de observação fornece uma aproximação mais precisa para sua disponibilidade.
- **Confiabilidade**: É a probabilidade na qual uma requisição é corretamente atendida dentro de um tempo máximo esperado, ou pode ser simplesmente a taxa de requisições finalizadas com sucesso. É referenciado pela garantia da troca de mensagens entre requisitantes de serviços e provedores de serviço.
- **Preço**: O valor monetário de um serviço definido pelo provedor de serviços.
- ***Throughput* (Vazão)**: *Throughput* ou vazão representa a quantidade de requisições de serviço entregues em um dado período de tempo. O tempo de resposta de um sistema aumenta

na medida em que o *throughput* aumenta. Uma importante política de decisão é promover um compromisso entre maximizar o *throughput* e minimizar o tempo de resposta.

- **Tempo de resposta**: É a quantidade de tempo entre enviar uma requisição e receber uma resposta ou então a garantia de tempo de resposta médio requerido para completar uma requisição de serviço. Também é referenciado como a duração de uma execução, no qual é computado utilizando o tempo de processamento e o tempo de transmissão das mensagens.
- **Latência**: Tempo gasto entre a chegada da requisição e o envio da resposta.
- **Desempenho**: É medido em termos de vazão e tempo de resposta. O serviço considera ter um bom desempenho quando sua vazão é alta e seu tempo de resposta é baixo. É a velocidade para se completar uma requisição de serviço.
- **Segurança**: A habilidade para garantir a não-repudiação das mensagens, confiabilidade e autenticação das entidades envolvidas.
- **Regulatoriedade**: Aspecto de qualidade que mostra se um Web service está em conformidade com as leis, regras, padronizações e acordos de níveis de serviços estabelecidos.
- **Robustez/Flexibilidade**: A habilidade de um serviço em contornar entradas inválidas, incompletas e conflitantes e gerar o resultado correto.
- **Precisão**: Taxa de erros gerados pelo serviço
- **Reputação**: É a medida de confiabilidade de um serviço calculada pelos clientes. É a avaliação do serviço fornecida pelo usuário final.

2.9 Considerações Finais

Este capítulo apresentou uma visão geral sobre os principais conceitos que envolvem a tecnologia Web services, desde a descrição de cada entidade pertencente à esta tecnologia, até a interação entre eles, apresentando uma infra-estrutura fornecida pelos fundamentos da arquitetura orientada a serviços. Também foi visto que os Web services são uma abordagem que admite a heterogeneidade de sistemas, permitindo sua integração e interoperabilidade. No entanto, a aquisição de tais propriedades pode implicar em problemas de desempenho, tornando-se um dos principais fatores que pode afetar a qualidade do serviço oferecido, visto que o desempenho é um dos principais atributos para a garantia de QoS.

Considerando ainda que os Web services são aplicações disponibilizadas em uma rede heterogênea como a Internet, garantir um bom desempenho é desafiador, tornando-se necessária a adoção de técnicas e abordagens específicas. Para comprovar se os serviços atendem as necessidades dos requisitantes em termos de qualidade de forma adequada, a adoção de técnicas adequadas para

a avaliação de desempenho é uma etapa fundamental. Sendo assim, conceitos relacionados com avaliação de desempenho serão abordados no capítulo seguinte.

CAPÍTULO

3

Avaliação de Desempenho

3.1 Considerações Iniciais

A avaliação de desempenho é considerada uma etapa fundamental para garantir melhor provisão de qualidade de serviço em um sistema computacional. Atualmente existem diversas ferramentas e metodologias para auxiliar no processo de avaliação e análise de desempenho de sistemas computacionais, tanto em sistemas tradicionais e centralizados, como em ambientes computacionais distribuídos. Algumas dessas metodologias são discutidas no decorrer desse capítulo, apresentando as principais técnicas de avaliação, além de definir alguns critérios importantes para realizar a etapa de planejamento de experimentos.

3.2 Planejamento de Experimentos

A fase de planejamento de experimentos é uma etapa fundamental da avaliação de desempenho de sistemas computacionais que consiste na definição de quais, em que quantidade, e em que condição os dados devem ser coletados durante um determinado experimento. O principal objetivo de um planejamento de experimentos apropriado é obter o máximo de informações com mínima quantidade de experimentos. Isso permite a redução considerável de esforço que seria aplicado para a realização dos experimentos. Uma análise de experimentos apropriado também auxilia na separação dos efeitos de vários fatores que podem afetar o desempenho. Além disso, é possível também determinar se um fator tem efeitos significativos sobre o desempenho, ou se a diferença observada é simplesmente devido a variações aleatórias causadas por erros de aferição ou parâmetros que não são controlados (Jain, 1991).

3.2.1 Terminologia

Os termos descritos a seguir são muito utilizados durante a etapa de projeto e análise de experimentos (Jain, 1991):

- **Variável de resposta**: A saída de um experimento é conhecido como variável de resposta. Geralmente a variável de resposta é a principal medida de desempenho de um sistema. Por exemplo, em um estudo para o projeto de uma estação de trabalho, a variável de resposta poderia ser a vazão representada por tarefas completadas por unidade de tempo, ou então poderia ser o tempo de resposta das tarefas ou qualquer outra métrica.

- **Fatores**: As variáveis que afetam a variável de resposta e que pode assumir diversas alternativas são conhecidas como fatores. Por exemplo, podem existir cinco fatores em um projeto de estação de trabalho. Os fatores são o tipo de CPU, tamanho da memória, número de discos, carga de trabalho usada e o nível educacional do usuário.

- **Níveis**: Os valores que um fator pode assumir são conhecidos como níveis. Por exemplo, em um estudo de projeto de estação de trabalho, o tipo de CPU consiste de três níveis: Core Duo 1.5GHz, Core 2 Duo 2.0GHz, ou Pentium IV. O número de discos têm quatro níveis: 1, 2, 3, ou 4. O tamanho da memória tem três níveis: 256 Mbytes, 512 Mbytes, ou 1 Gbytes; e assim por diante.

- **Fatores primários**: Fatores que causam um grande impacto em uma variável de resposta e que devem ser considerados, e portanto, devem ser quantificados.

- **Fatores secundários**: Fatores cujo impacto na variável de resposta não é significante ou não se tem interesse em quantificar.

- **Replicação**: Repetição de todo ou de parte de um experimento.

- **Projeto**: Determina o número de experimentos a serem considerados, incluindo o número de fatores e níveis, a combinação entre os níveis e o número de replicações para cada experimento.

- **Interação**: Dois fatores interagem se o efeito de um depende do nível do outro.

3.2.2 Tipos de planejamento de experimentos

Existem vários tipos de técnicas utilizadas para determinar a etapa de planejamento de experimentos (Jain, 1991). As mais utilizadas são:

- **Planejamento Simples**: Etapas básicas dessa abordagem consistem em: iniciar com uma configuração inicial, fixar todos os valores e variar um fator por vez, e verificar se esse fator

afeta o desempenho. É fácil de ser implementado, além de ser muito utilizado. Porém não é recomendável estatisticamente, pois não permite verificar a relação entre os fatores.

- **Planejamento Fatorial Completo**: Utiliza todas as combinações considerando todos os fatores e níveis. Com isso, todos os fatores são avaliados, pode-se determinar o efeito de qualquer fator, e interações entre fatores podem ser verificadas. No entanto, o principal problema dessa abordagem é o custo da avaliação de desempenho, no qual é gerado uma grande quantidade de experimentos. Para reduzir a quantidade de experimentos gerados, pode-se reduzir o número de níveis para cada fator, reduzir o número de fatores, ou então utilizar a abordagem seguinte.

- **Planejamento Fatorial Parcial** : Algumas vezes o número de experimentos requeridos em um planejamento fatorial completo é muito grande. Isso acontece pois o número de fatores e seus níveis também é grande. Nesses casos, pode-se utilizar apenas uma fração do planejamento fatorial completo. Nessa abordagem considera-se apenas parte de todos os experimentos.

3.3 Técnicas de Avaliação de Desempenho

Quando se pretende realizar a avaliação de desempenho de um sistema complexo, um dos desafios primordiais está em encontrar uma técnica adequada para prosseguir com a avaliação. Utilizando essa técnica, espera-se obter resultados satisfatórios que avaliem corretamente o comportamento de um sistema. Entretanto, ela própria pode ser um fator degenerador do desempenho. A escolha de uma técnica adequada varia de acordo com o domínio em que se encontra uma determinada aplicação, ou seja, apesar das técnicas serem de caráter geral, elas geralmente possuem aplicações preferenciais. (Araujo, 2005)

De uma maneira genérica, as técnicas de avaliação de desempenho podem ser agrupadas em duas classes, de certa forma, complementares. A primeira relaciona aquelas técnicas que realizam experimentação no sistema e a segunda classe relaciona as técnicas que criam abstrações desse sistema, através das quais são feitas inferências sobre o seu funcionamento e de seus componentes (Araujo, 2005). As duas classes são conhecidas, respectivamente, como técnicas de aferição e de modelagem, onde o grande diferencial entre elas é ter ou não o sistema implementado (Santana et al., 1997). Para os casos em que o sistema já existe e, conseqüentemente, pode ser averiguado empiricamente, as técnicas de aferição são mais recomendadas. Em contrapartida, para os sistemas inexistentes, as técnicas de modelagem são mais utilizadas, definindo abstrações que caracterizam um sistema real, sendo que através da solução desse modelo, pode-se ter uma aproximação de como o sistema se comportaria se fosse efetivado. Entretanto, no contexto de avaliação de desempenho, esse modelo é um processo complexo e de forte teor matemático (Santana et al., 1997).

Para garantir resultados mais consistentes, uma abordagem híbrida pode ser utilizada, combinando técnicas de aferição e modelagem, de modo a utilizar dados reais obtidos pela aferição

como entrada para as técnicas de modelagem. A seguir, as técnicas de aferição e modelagem serão abordadas detalhadamente.

3.3.1 Técnicas de Aferição

Técnicas de aferição são ferramentas utilizadas para analisar o desempenho de um sistema através de sua experimentação. Essa análise pode ser efetuada utilizando *software* (como por exemplo os "*benchmarks*") ou *hardware* (como a utilização de monitores de *hardware*) (Santana et al., 1997).

Os resultados obtidos por essa abordagem, em geral, são mais precisos que as técnicas de modelagem. No entanto, ela requer a disponibilidade do sistema ou de seu projeto em fase final de desenvolvimento. Além disso, outra dificuldade encontrada é como realizar os experimentos do sistema de modo que sua influência não interfira na análise dos resultados. Outro problema ocorre também pela falta de flexibilidade de verificar diferentes alternativas de sistemas, implicando em altos custos para as alterações e teste, além da própria dificuldade de executá-la (Santana et al., 1997).

Essa classe de técnicas consiste basicamente de três tipos de técnicas: a construção de protótipos, *benchmarks* e coleta de dados. Essas abordagens serão discutidas nos próximos tópicos dessa seção.

Construção de protótipo é uma simplificação do sistema original, mantendo o mesmo comportamento funcional, o qual será levado em consideração durante a avaliação de desempenho, e descartando as funcionalidades não-relevantes, presentes somente no sistema real. Apenas as funcionalidades essenciais do sistema são abstraídos e incluídos no protótipo. A experimentação com um protótipo fornece uma boa idéia sobre o comportamento esperado do sistema a ser tratado (Santana et al., 1997).

O emprego de protótipos representa uma alternativa para se obter informações antecipadas sobre o comportamento de um futuro sistema a ser desenvolvido. A construção de todo o sistema, em contraste com a utilização de protótipos, pode ocasionar dispêndio de tempo e custo envolvidos caso o desempenho esperado não seja o atingido.

O uso de protótipos também possibilita a verificação de alternativas para o sistema. Quando o objetivo for analisar o comportamento de um sistema mediante a algumas alterações, é impraticável que essas alterações sejam feitas diretamente sobre o sistema (Santana et al., 1994).

Apesar das vantagens de menor custo e de maior facilidade de alteração em relação à construção de um sistema real, o protótipo ainda é considerado uma ferramenta cara, dependendo do nível de abstração em relação ao sistema original, e também da flexibilidade para admitir possíveis alterações no sistema. Tais fatores, além de influenciar no custo do sistema, também consomem uma quantidade considerável de tempo. Outra dificuldade é encontrada na abstração das características essenciais do sistema, principalmente quando o sistema real encontra-se em um estágio

de desenvolvimento em que diversas características relacionadas com sua configuração e operação ainda não estão bem definidas. A utilização de uma abstração mal-formulada pode inviabilizar a construção de um bom protótipo (Santana et al., 1994).

O desenvolvimento de um protótipo pode ser dividido em cinco etapas (Figura-3.1) (Santana et al., 1994):

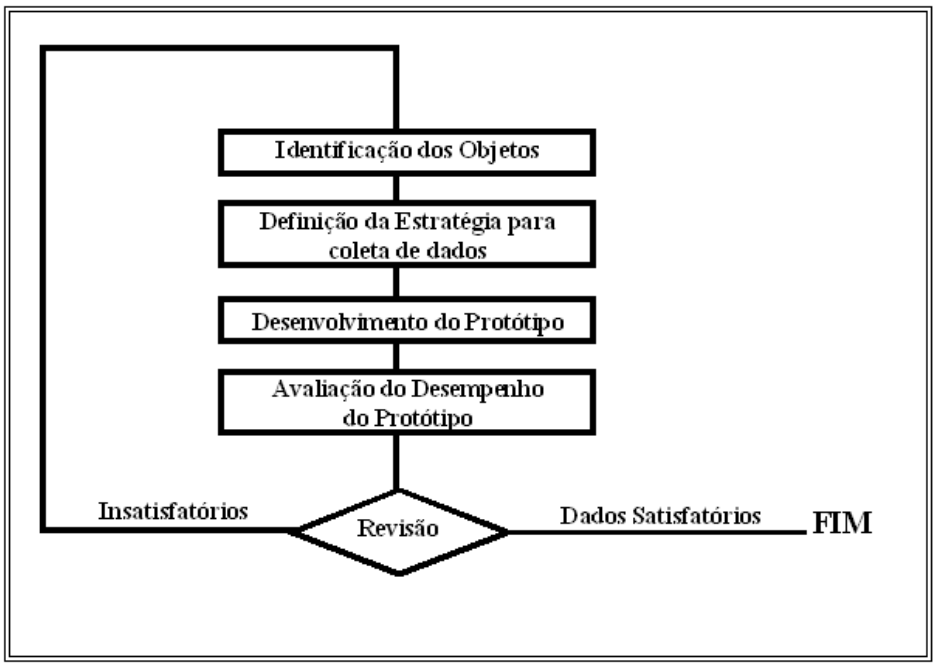

Figura 3.1: Etapas de Desenvolvimento de um Protótipo (Santana et al., 1994)

1. **Identificação dos Objetivos**: Determinam a construção do protótipo e também a definição das características essenciais do sistema em questão, com ênfase para as medidas de desempenho que deverão ser analisadas.

2. **Definição da estratégia para coleta de dados**: Essa etapa é muito importante pois informações infundadas podem determinar conclusões incorretas sobre o desempenho do sistema.

3. **Desenvolvimento do protótipo**: Nessa etapa o protótipo é construído considerando-se os requisitos definidos durante a primeira etapa.

4. **Avaliação do Desempenho do Protótipo**: Durante essa etapa o sistema é avaliado, obtendo-se as medidas de desempenho e analisando-as. A avaliação deve ser minuciosa ponderando-se a limitação do protótipo e estendendo-se os resultados para o sistema real.

5. **Revisão**: Independentemente do resultado da análise a revisão deve ser feita de modo a verificar se todos os objetivos foram atingidos corretamente e se as características essenciais foram reproduzidas no protótipo. Se os resultados obtidos não se mostraram confiáveis em relação aos requisitos estabelecidos anteriormente, é importante reiniciar todo o processo,

desde a reestruturação dos objetivos na primeira etapa. Caso contrário o desenvolvimento ou não do novo sistema é decidido baseado nas informações obtidas pela análise através do protótipo.

Benchmarks

Benchmarks são padrões de desempenhos, geralmente em forma de programas de computador, que são utilizados para testar o desempenho de um sistema de *software* ou *hardware* (Santana et al., 1997). Um programa *benckmark* pode ser executado em diferentes sistemas, obtendo-se resultados em forma de tempo de execução e, com isso, pode ser feita uma comparação entre esses sistemas, fornecendo uma boa indicação do desempenho do sistema. Programas de ordenação e de atualização de arquivos são exemplos típicos de *benchmarks* utilizados para a comparação entre sistemas.

Benchmarks são indicados para a comparação de diferentes sistemas antes da aquisição de um novo, sendo geralmente utilizados como ferramentas para a seleção de um determinado sistema, baseando-se na comparação de fatores específicos. Podem também auxiliar no desenvolvimento de um novo sistema, construindo módulos e os executando separadamente em diferentes máquinas, mas este processo é limitado no sentido de descobrir quais rotinas serão representativas do comportamento do novo sistema (Santana et al., 1994).

Além do tempo de resposta, outras medidas podem ser utilizadas para realizar a comparação de sistemas. Uma delas é a velocidade que um programa *benchmark* executa suas operações, que pode ser calculadas utilizando-se medidas como, por exemplo, MIPS (milhões de instruções por segundo) ou MFLOPS (milhões de operações de ponto flutuante por segundo). A escolha de uma unidade de medida apropriada deve ser feita de maneira cuidadosa, pois durante a comparação entre dois sistemas com tecnologias diferentes, uma certa instrução de um sistema pode equivaler a várias instruções de outro sistema como, por exemplo, a comparação entre instruções de uma processador CISC com instruções de um processador RISC, ocasionando comparações inadequadas.

Coleta de dados

Coleta de dados é a ferramenta, entre outras técnicas de desempenho abordadas, que fornece o valor mais preciso para medidas de desempenho, uma vez que nesta técnica envolvem dados reais do sistema em questão. Entretanto, coletar dados de sistemas reais durante sua execução não é uma tarefa trivial, principalmente se considerar que a aquisição de dados não pode interferir na atual operação do sistema. Isso significa que procedimentos usados para coletar dados não podem interferir nos resultados obtidos pelos usuários do sistema (Santana et al., 1997).

Essa abordagem também pode ser utilizada em combinação com técnicas de modelagem, produzindo dados para possibilitar a comparação de diferentes modelos. Sendo assim, através das informações obtidas com a coleta de dados, pode-se estimar os parâmetros de entrada requisitados

por modelos analíticos e/ou de simulação, além de possibilitar a validação de tais modelos (Santana et al., 1994). É uma opção interessante para realizar a verificação de alternativas para a avaliação do sistema, provendo resultados mais confiáveis. A coleta de dados também pode ser utilizada em conjunto com protótipos, estabelecendo uma técnica para avaliar os resultados obtidos usando o protótipo.

A coleta de dados geralmente é feita utilizando ferramentas conhecidas como **monitores de desempenho**. Um monitor de desempenho é utilizado para observar o desempenho de sistemas, capturar estatísticas de desempenho, conduzir análise de dados, e apresentar os resultados. As ferramentas de monitoração podem ser classificadas em monitores de *hardware*, de *software*, de *firmware* e híbridos (Sabetta e Koziolek, 2005).

Os **monitores de *hardware*** são um conjunto de sensores eletrônicos, porta lógicas para os sinais, contadores, *display* e unidade de armazenamento, que fazem medições das características de um determinado sistema, no qual um monitor de *hardware* pode ser conectado. Durante as fases iniciais do desenvolvimento da computação, esse era um dos métodos mais comuns para realizar medições de desempenho. Atualmente, monitoração em *hardware* é mais comumente aplicado em redes e sistemas distribuídos. Dentre suas principais vantagens, incluem a baixa perturbação das atividades do sistema e também a capacidade de monitoração com precisão relativamente alta em relação aos demais monitores. Sua desvantagem é sua falta de flexibilidade para detectar eventos e funcionalidades pertencentes a softwares (Sabetta e Koziolek, 2005).

Os **monitores de *software*** são utilizados para monitorar sistemas operacionais e aplicações em nivel de usuário. Monitores de sistemas operacionais geralmente são conhecidos como *monitores de amostragem* e são implementados por rotinas contidas no *kernel* do sistema operacional. Monitores de *software* para aplicações individuais geralmente são utilizados para verificar qual é a parte mais crítica do programa em termos de desempenho. Os monitores de *software* geralmente são versáteis, possibilitando coletar informações variados sobre o sistema monitorado. Uma desvantagem desse tipo de monitoração é que a versatilidade e diversidade das informações obtidas por monitores de software acrescentam uma sobrecarga no sistema, e assim, o processo de monitoração pode afetar nas medições de desempenho do sistema (Sabetta e Koziolek, 2005).

Monitores de *firmware* são similares em relação aos monitores de *software*, mas são implementados alterando o micro-código do processador. Geralmente são utilizados quando pontos de acesso para monitores de *hardware* são inacessíveis e demandam certa precisão de tempo que não pode ser adquirida utilizando monitores de *software*.

Monitores híbridos são uma combinação dos monitores de *software* e *hardware*. Enquanto rotinas de software são responsáveis por capturar eventos e armazenar suas informações em registros especiais de monitoração, monitores de *hardware* fornecem alta resolução para a captura das informações. Monitores híbridos consideram o melhor dos dois mundos. Entretanto, um monitor híbrido é complexo no sentido que é necessário um *hardware* especial e um projeto de *software*, formando uma arquitetura complexa (Sabetta e Koziolek, 2005).

Os monitores também podem ser classificados em dois grupos, pelo modo em que são ativados. Os **monitores baseados em eventos** são ativados se eventos[1] específicos ocorrem no sistema, enquanto os **monitores de amostragem** são ativados em intervalos de tempo específicos os quais são controlados por temporizadores externos.

3.3.2 Técnicas de Modelagem

Técnicas de modelagem consistem na construção e análise de modelos representativos do sistema a ser avaliado em termos de desempenho. As características essenciais do sistema em questão são abstraídas e em seguida são representadas em um modelo, descrito a partir de um método formal. Esse método pode envolver equações matemáticas ou então uma representação gráfica do sistema. Existem modelos cujas soluções são extremamente triviais e outros cuja complexidade limita a aplicação dessa técnica. Assim como para as técnicas de aferição, a decisão para a escolha de uma determinada modelagem varia de acordo com o domínio da aplicação envolvida, ou então do nível de conhecimento dos desenvolvedores do sistema em relação à técnica a ser escolhida (Santana et al., 1997).

O número de detalhes ou características extraídos do sistema real determinam, na maioria dos casos, o grau de dificuldade para a solução do modelo. Sendo assim, a inclusão de detalhes irrelevantes pode introduzir complicações desnecessárias, aumentando, conseqüentemente, a complexidade do modelo a ser desenvolvido. Para evitar tal situação, deve-se considerar que apenas as características essenciais do sistema devem ser refletidas no sistema. Quando um modelo se torna muito complexo, simplificações e suposições podem ser feitas durante a modelagem. Todavia, tais alterações podem trazer prejuízos à análise de desempenho, uma vez que o sistema real pode ser distorcido no modelo (Santana et al., 1994). Para minimizar tais conseqüências na análise dos resultados obtidos, os seguintes passos devem ser seguidos:

- Documentação de todas as simplificações e suposições efetuadas sobre o modelo.
- Considerar que os resultados obtidos na modelagem refletem apenas as tendências do comportamento do sistema real, e não o que realmente se passa no sistema.

A Figura 3.2 descreve a modelagem de um sistema como uma "simplificação" que descarta as características julgadas irrelevantes do sistema. Pode-se perceber que os dados de entrada e os parâmetros entre o sistema e o modelo devem estar relacionados entre si. Os dados do sistema que estão sendo analisados devem ser inferidos a partir dos dados obtidos pelo modelo (Santana et al., 1994).

As principais vantagens de se utilizar técnicas de modelagem estão na flexibilidade, pois podem ser aplicados tanto em sistemas inexistentes, quanto em sistemas existentes; e na relação custo versus grau de precisão dos resultados obtidos. As dificuldades se encontram na definição das

[1] interferências que causam mudanças de estados no sistema

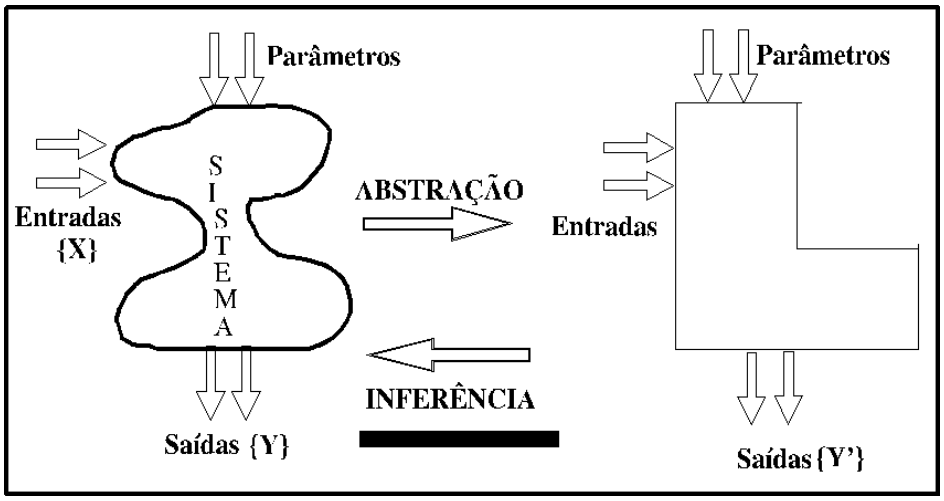

Figura 3.2: Visão da Modelagem de um Sistema (Santana et al., 1994)

características essenciais e na validação dos resultados do modelo, ou seja, assegurar que o modelo apresentado é uma representação consistente do sistema original (Santana et al., 1994).

De uma maneira genérica, o processo de modelagem pode ser definido em quatro etapas básicas, conforme apresentado na Figura 3.3. Na fase inicial da modelagem, deve-se criar uma *especificação* que descreve o sistema real de forma satisfatória, na qual devem estar contidos os componentes do sistema relevantes à avaliação, além do relacionamento entre eles. Algumas técnicas usadas para especificação são redes de filas, redes de Petri e *Statecharts*. Após a confecção do modelo com uma das técnicas citadas, deve-se parametrizá-lo com elementos que serão dados de entrada para a fase da solução. Na parametrização geralmente são utilizados taxas, tempos médios e probabilidades. A solução do modelo aplica um método matemático (estocástico), automatizado ou não, para adquirir medidas de desempenho a partir das entradas. Alguns métodos analíticos (cadeias de Markov, por exemplo) ou simulação são usados nessa fase. Por fim, a modelagem deve apresentar seus resultados através de uma maneira conveniente. Gráficos e arquivos-texto são geralmente utilizado nessa fase (Francês et al., 2006).

3.4 Protótipo e Coleta de Dados

As aplicações distribuídas (web services clientes e provedores) devem ser construídas visando a realização de testes de desempenho na arquitetura WSARCH. O procedimento para avaliação desta arquitetura é intrinsecamente relacionado com as características das aplicações que devem ser construídas neste projeto de mestrado. A construção dessas aplicações necessita considerar a diversidade de interações que podem ocorrer entre os componentes da arquitetura. Para isso, alguns parâmetros de qualidade de serviço e fatores devem ser considerados nas interações. É preciso um estudo que defina quais atributos analisar e em que situação essa análise deve ser feita. O critério para a escolha dos atributos de QoS devem ser baseado nas características das aplicações. Por isso, é importante detalhar o comportamento (isto é, identificar as necessidade de QoS) dessas

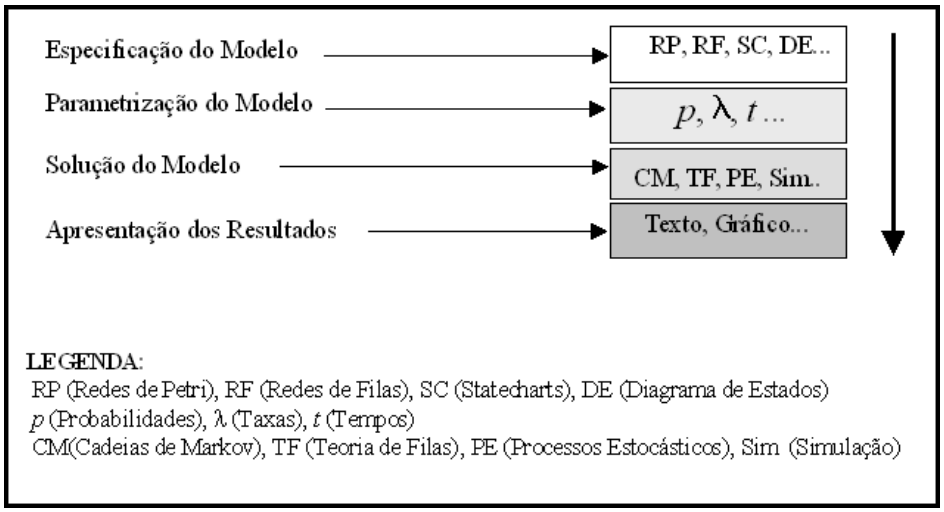

Figura 3.3: Etapas de um Processo de Modelagem (Francês et al., 2006)

aplicações e descobrir que parâmetros são mais adequados e como eles serão correlacionados. A construção dos web services deve ser pautada em decisões como:

- Implementar aplicações com características distintas, tais como:
  - Aplicações que fazem muito processamento (CPU-Bound), como por exemplo, serviços com características de soluções para aplicações científicas
  - Aplicações que fazem acesso a uma base de dados
  - Aplicações que utilizam muito a rede de comunicação (I/O-Bound), na busca de resultados de outros web services, para por exemplo compor um processo de negócios.
- Implementar aplicações que utilizem funcionalidades adicionais, como a adoção de tecnologias "WS-*" para prover segurança, confiabilidade, etc.
- Definir a parametrização dos experimentos realizados para verificar a influência de fatores específicos

Um vez determinado os critérios para a construção dos web services (cliente e provedor) é preciso detalhar a avaliação de desempenho com base, por exemplo, nos atributos de QoS:

- Considerar o tempo de resposta do provedor de serviços de acordo com a solicitação feita pelo web service cliente;
- Tempo necessário para o provedor de serviços selecionar o serviço requisitado e retornar tal informação para o broker;
- Tempo para o processamento do web service no provedor;
- Vazão do provedor de serviços;

- Carga de requisições para determinado web service;
- O impacto, em termos de desempenho, que um web service deve gerar em um provedor de serviços.

É importante ressaltar que as discussões anteriores são fundamentais para a tomada de decisão durante a implementação de um Web service e principalmente no que diz respeito à avaliação de desempenho.

3.5 Considerações Finais

O processo de avaliação de desempenho é um instrumento que possibilita um melhor entendimento sobre o comportamente de um determinado sistema computacional, utilizando uma série de métricas e ferramentais que auxiliam nesse processo. As principais metodologias existentes na área de avaliação de desempenho foram abordadas nesse capítulo, provendo um bom embasamento necessário para prosseguir com a etapa de avaliação de desempenho de aplicações com características distintas, uma das principais atividades presentes neste projeto de mestrado.

CAPÍTULO

4

Desenvolvimento de Aplicações

4.1 Considerações iniciais

Uma das principais atividades exercidas durante o desenvolvimento deste trabalho de mestrado foi contribuir com a implementação e evolução da arquitetura WSARCH, proposta na tese de doutorado de Júlio Estrella para permitir melhor provisão de qualidade de serviço durante a integração de aplicações em uma arquitetura orientada a serviços (Estrella, 2010). Sendo assim, este capítulo tem como objetivo descrever a arquitetura WSARCH, desde o modelo abstrato da arquitetura para fornecer uma visão geral do projeto, até detalhes de desenvolvimento sobre cada um dos componentes pertencentes à arquitetura. Além disso, a relação entre os módulos da WSARCH e atributos de QoS também são referenciados nesse capítulo. As principais contribuições desse projeto de mestrado em relação ao desenvolvimento da arquitetura WSARCH são abordadas na Seção 4.3.

Como parte do escopo principal do projeto, aplicações de características distintas foram definidas de acordo com critérios baseados na utilização de recursos específicos, e em seguida implantadas nos provedores de serviço, o que possibilitou os estudos de avaliação de desempenho sejam realizados na arquitetura. Detalhes sobre a implementação dessas aplicações são descritos na Seção 4.4.

4.2 Arquitetura WSARCH

Para a definição de sua estrutura, a arquitetura WSARCH considera os principais conceitos de orientação a serviços e componentes especificados em uma arquitetura orientada a serviços padrão, incluindo alguns módulos para possibilitar a provisão de qualidade de serviço em mensagens

trocadas entre os serviços envolvidos, além de prover funcionalidades para capturar informações de desempenho provenientes da interação desses componentes durante o processamento de uma requisição. Desse modo, é possível a realização de estudos com o objetivo de avaliar o desempenho da arquitetura. As informações são, inclusive, levadas em consideração para a definição de algoritmos de roteamento de mensagens que são utilizados na arquitetura para tratamento da requisição com métricas de QoS.

Um modelo conceitual de alto nível da arquitetura é ilustrado na Figura 4.1, representando as interações de seus principais componentes. Nesse modelo, são mencionados os seguintes componentes:

- **Cliente** A aplicação cliente solicita requisições de serviço ao *broker*, considerando informações de QoS.
- **Broker** Possui módulos internos para gerenciar informações de QoS sobre os serviços. Com base nas informações provindas de uma requisição do cliente, e de informações de QoS sobre os serviços disponíveis, um provedor de serviços é escolhido de acordo com as necessidades do cliente.
- **Provedor** Contém os Web services que serão requisitados pelo cliente. Possui um módulo que atualiza informações de QoS periodicamente no UDDI.
- **UDDI** São cadastrados todos os serviços disponibilizados para o cliente. Em sua forma original, apenas informações funcionais dos serviços são catalogados nesse repositório. Para fins de implementação das funcionalidades da arquitetura WSARCH, o UDDI foi alterado para armazenar informações de QoS, os quais são recebidas dos provedores.

4.2.1 Objetivo

A construção e desenvolvimento da arquitetura WSARCH foram realizados com foco em avaliação e oferecimento de desempenho, e voltada para a provisão de Web services que consideram atributos de QoS. A arquitetura proposta em (Estrella, 2010) estendeu, com adaptações e sugestões de melhorias, outros modelos arquiteturais evidenciados na literatura de Web Services, tais como os apresentados em: (Erradi e Maheshwari, 2005), (Benkner e Engelbrecht, 2006), (Sheth et al., 2002). O principal ponto da arquitetura WSARCH difere de outras arquiteturas é no sentido de fornecer recursos para inserir e estender funcionalidades relacionadas com QoS de forma modularizada, provendo em paralelo mecanismos para coletar informações de desempenho. Para isso, considera-se na arquitetura a interação entre os componentes tradicionais de SOA com um *broker* que, além de desempenhar o papel de escalonar as mensagens SOAP entre cliente, provedor e registro de serviços, é responsável por gerenciar as informações de QoS para selecionar o serviço mais adequado para o cliente. Além disso, para garantir que as funcionalidades implementadas

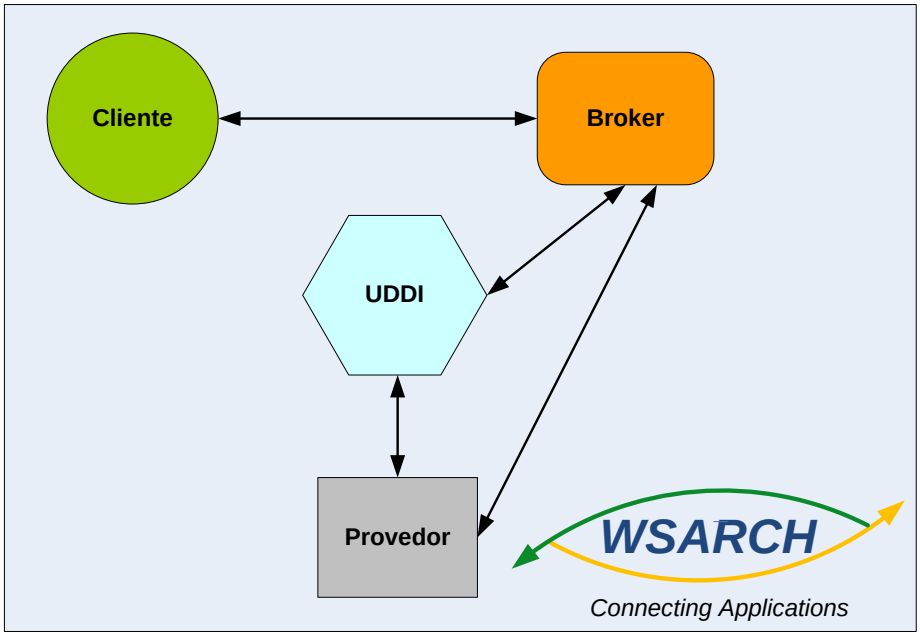

Figura 4.1: WSARCH - Modelo Abstrato - (Estrella, 2010)

da WSARCH estavam sendo aplicadas corretamente, um estudo de avaliação de desempenho foi realizado dentro do escopo do presente projeto de mestrado. Em razão da complexidade e abrangência envolvidas na implementação e outras atividades da WSARCH, o autor do presente projeto de mestrado também auxiliou na construção e avaliação de desempenho da arquitetura. As contribuições realizadas neste trabalho são descritas detalhadamente na Seção 4.3 .

4.2.2 Funcionamento

Um modelo simplificado da arquitetura WSARCH, tanto seus componentes quanto as principais interações entre eles, é apresentado na Figura 4.2.

As seguintes etapas, identificadas na Figura 4.2, são necessárias para a execução de uma requisição:

1. Cliente faz a requisição ao *broker*, o qual possui informações atualizadas do provedor do serviço (carga, tipo de serviço, classe de cliente, etc.);

2. Com base nas informações de QoS solicitadas pelo cliente do serviço, o *broker* realiza uma busca num repositório de serviços com o objetivo de encontrar o serviço mais adequado;

3. *Broker* obtém a especificação do serviço apropriado e as respectivas informações de QoS do provedor;

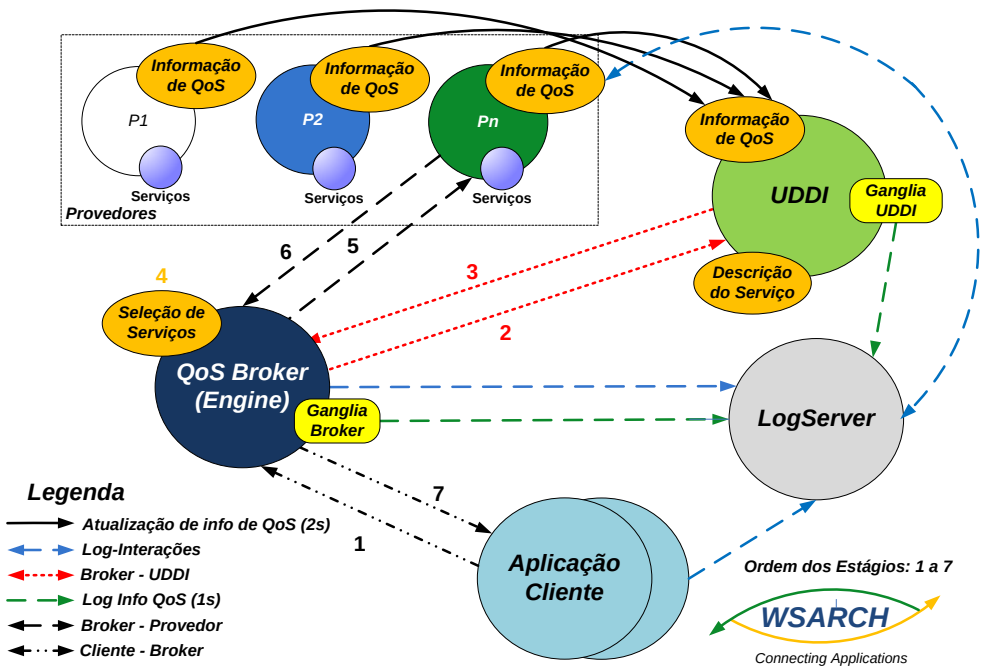

Figura 4.2: WSARCH - Componentes e interações - (Estrella, 2010)

4. Com a localização do serviço apropriado no repositório de serviços, e já de posse das informações dos provedores de serviços (essas informações de QoS são propagadas periodicamente), o *broker* escolhe o melhor provedor candidato (Seleção de Serviços)

5. Com o serviço selecionado de acordo com as informações de QoS, o *broker* realiza a requisição (invocação do Web Service) no provedor de serviços;

6. Após a requisição ser executada, a resposta é retornada diretamente ao *broker* da WSARCH;

7. Finalmente a resposta é encaminhada ao cliente que inicialmente solicitou o Web Service.

Além disso, outras etapas de interação entre os componentes da arquitetura ocorrem de forma paralela enquanto as requisições de serviço são tratados pelo *broker*. Por exemplo, provedores de serviço e UDDI interagem entre si para manter as informações de QoS referentes aos serviços atualizados. Para isso, adotou-se a utilização de uma aplicação de monitoração denominada Ganglia (Ganglia, 2003) para propagar periodicamente as informações de QoS de cada provedor de serviço para o registro de serviços. Para permitir que estudos de avaliação de desempenho possam ser realizados na WSARCH um servidor de logs foi incluído na arquitetura. Este componente é utilizado para coletar e armazenar os dados sobre o desempenho dos demais componentes e suas interações.

4.2.3 Módulos da WSARCH e QoS

Para melhor atender às necessidades do cliente, a diferenciação de Web services se torna uma tarefa importante quando são disponibilizados serviços semelhantes em diversos provedores dis-

tintos e, para isso, é necessário que pelo menos algum tipo de garantia de qualidade de serviço seja fornecido pelos provedores de serviço. Qualidade de serviço pode ser tratada sob diversas perspectivas, como foi discutido na Seção 2.8 e, dessa forma, cada um dos módulos envolvidos da arquitetura WSARCH e suas interações foram projetados de modo a considerar sua relação com atributos de QoS. No decorrer do desenvolvimento desse projeto, um estudo relacionando tais atributos de QoS com os módulos de uma arquitetura orientada a serviços foi realizado. Em resumo, o relacionamento entre módulos de SOA e atributos de QoS é apresentado na Tabela 4.1.

Tabela 4.1: Relação dos atributos de QoS com as módulos da WSARCH (Estrella, 2010)

	Módulos da Arquitetura			
Atributos de QoS	**Cliente**	**Provedor**	**Broker de QoS**	**UDDI**
Tempo de Resposta	X			
Vazão		X	X	X
Memória Principal	X	X	X	X
Compressão de Mensagens	X	X		
Atraso da Rede	X	X	X	X
Composição de Serviços	X	X	X	
Disponibilidade		X	X	X
Custo do Serviço			X	
Confiabilidade		X	X	X
Classe do Serviço		X	X	
Carga do Servidor		X	X	X
Latência		X	X	X
Reputação	X		X	
Segurança	X	X		X
Atomicidade		X		X
Consistência		X		X
Isolamento		X		X
Durabilidade		X		

Visto que os principais módulos da WSARCH e suas extensões possuem perspectivas distintas em relação aos atributos de QoS listados, tais módulos são descritos a seguir sob a perspectiva de qualidade de serviço.

Cliente

Um dos fatores de qualidade mais significativos sob o ponto de vista do cliente é o desempenho dos Web services que ele utiliza. Em geral, para se mensurar se este fator atende com às expectivas do cliente, verifica-se, como principal atributo de QoS, o tempo de resposta. Espera-se que o tempo de resposta seja o menor possível, e assim, é necessário identificar quais são os principais pontos que podem impactar o desempenho desejado. Como exemplo, tem-se o processamento de mensagens XML, as operações em banco de dados, e as comunicações entre módulos da arquitetura como possíveis causas de problemas de desempenho. Algumas melhorias podem ser implementa-

das com o intuito de reduzir o tempo de resposta: utilizar técnicas de compressão de mensagens, otimizar o processamento do Web service, etc.

Provedor

Para que o provedor de serviços possa disponibilizar os serviços requisitados pelo cliente de modo eficiente em termos de desempenho, algumas alternativas podem ser consideradas se a redução do tempo de resposta de processamento do serviço não for o suficiente, tais como:

- A utilização de cache de requisições, se habilitada no servidor que hospeda os serviços do provedor, pode reduzir consideravelmente o tempo de resposta percebido pelo cliente. Por exemplo, o motor de processamento de mensagens SOAP permite a configuração de cache;
- O balanceamento de carga, que tem como objetivo principal diminuir o fluxo de dados de um provedor, evitando sua sobrecarga. Para isso, pode ser utilizado um escalonador de requisições fazendo o papel de distribuir as requisições para provedores menos sobrecarregados;
- Considerar a classificação das aplicações de acordo com a natureza da carga requerida pelo serviço para fornecer um tratamento diferenciado no atendimento das requisições. Por exemplo, um serviço multimídia requer alta vazão (a quantidade de serviços que o provedor atende por segundo, por exemplo), baixos atrasos, jitter, enquanto um serviço bancário online requer segurança e qualidade de serviço transacional;
- Baseado no comportamento dos serviços em um ambiente específico, pode-se determinar estimativas do tempo de resposta final de um determinado serviço considerando diferentes tipos de carga. Por exemplo, o tempo de resposta pode ser diretamente afetado pela quantidade de acessos simultâneos ao serviço requisitado, principalmente se considerar horários de sobrecarga.
- Considerar o fluxo de dados de um provedor ao selecionar provedores adequados (vazão). Quanto maior a vazão em um provedor, maior é a sobrecarga, sendo este um dos candidatos menos adequados a ser escolhido durante o processo de seleção de serviços.

Broker

Este componente tem como papel principal intermediar as interações entre os clientes e provedores, de modo a prover melhor escalonamento dos serviços entre os clientes baseado no acordo de qualidade de serviço definido entre eles. Dentre os diversos provedores que o *broker* tem acesso, o mesmo localiza serviços semelhantes de acordo com a requisição do cliente, e em seguida seleciona o serviço mais adequado em termos de qualidade de serviço. Esse acesso é realizado através de um repositório de serviços que o *broker* interage.

Como principal componente da arquitetura, o *broker* também deve fornecer um atendimento diferenciado, no sentido de gerenciar as requisições recebidas mesmo com carga muito elevada, de modo eficiente em termos de desempenho, e além disso, também deve prover disponibilidade e confiabilidade. Vale ressaltar que o *broker* também pode ser tratado como um serviço disponibilizado por um provedor de serviço, sendo necessário que o *broker* ofereça garantia de qualidade de serviço para os clientes.

4.2.4 Implementação

Como foi mencionado no decorrer deste capítulo, a arquitetura WSARCH é composta basicamente de quatro componentes: aplicação cliente, provedor de serviços, *broker* e UDDI. Para fins de armazenamento de dados para avaliação de desempenho e testes, um servidor de logs foi incluído na arquitetura, capturando informações relevantes sobre cada um dos componentes da arquitetura e suas interações. Além disso, foi necessário também incluir um sistema de monitoração dos recursos computacionais de alguns componentes para a obtenção de informações a serem utilizadas para tomada de decisão. Para o desenvolvimento do projeto, foi adotado o sistema de monitoração Ganglia para realizar esse papel. Nesta seção são descritos como foram implementados cada um dos componentes citados e as ferramentas de suporte utilizadas. Uma visão de nível menos abstrato é apresentada na Figura 4.3.

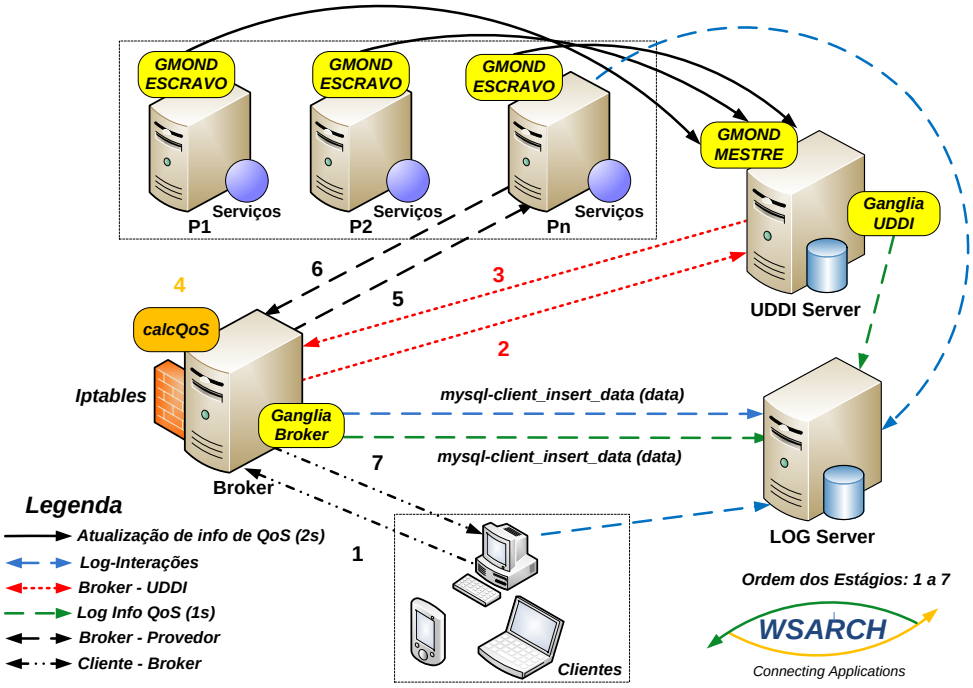

Figura 4.3: WSARCH - Informações de QoS - (Estrella, 2010)

Broker

No decorrer da Seção 4.2.2, os passos do fluxo de informações foram definidos durante a requisição de um serviço. Nota-se que as informações propagadas entre os componentes são centralizadas no *broker*, que desempenha o papel de selecionar o serviço mais adequado de acordo com os parâmetros de QoS solicitados pelo requisitante. Sendo assim, o *broker* deve fornecer as interfaces adequadas para possibilitar que tais integrações sejam realizadas.

Durante o processamento de uma requisição, após receber os parâmetros de serviço fornecidos pelo cliente, o *broker* coleta as informações dos serviços no repositório de serviços de acordo com as necessidades do cliente, em seguida seleciona e acessa o serviço no provedor de serviços escolhido e por fim, retorna os dados resultantes para o cliente. Para executar tais etapas, o *broker* fornece uma interface de entrada em formato WSDL para os clientes descrevendo o formato da mensagem SOAP de entrada e como será o formato da mensagem resultante. Além disso, o *broker* deve reconhecer a interface do serviço escolhido, e executá-lo com os parâmetros de entrada provenientes da mensagem SOAP enviada pela aplicação cliente. Sendo assim, após a etapa de seleção de serviço, uma operação de *parser* (processamento) da interface WSDL do serviço escolhido é realizada. Em resumo, as seguintes etapas são executadas no *broker* durante uma requisição:

1. Recebe a requisição do cliente;
2. Identifica o parâmetro de entrada enviada pelo cliente na mensagem SOAP. Podem ser parâmetros de entrada do serviço requisitado e informações de QoS, por exemplo;
3. Busca todos os serviços relevantes de acordo com a requisição do cliente;
4. Seleciona o melhor serviço, com base nas informações de QoS enviadas pelo cliente e das informações não-funcionais relacionadas com cada serviço;
5. Localiza o serviço escolhido e realiza um *parser* da interface WSDL deste Web service;
6. Executa o serviço;
7. Retorna os dados resultantes do serviço executado para o cliente.

Em paralelo com essas etapas, as informações de QoS são atualizadas periodicamente, sendo registradas no repositório do serviço. Na etapa de seleção de serviços (item 4), tais informações são utilizadas sendo definidas como parte de critérios de seleção em um algoritmo de escalonamento de requisições SOAP. A arquitetura WSARCH foi projetada para flexibilizar a escolha de algoritmos de seleção apropriadas com base nas informações de QoS existentes, ou até com base em outras características distintas como informações de mensagens SOAP, classes de aplicações, entre outros.

Para a concretização de um protótipo inicial da arquitetura, dois algoritmos de seleção foram implementados. O primeiro, denominado Algoritmo de classificação utiliza índices de desempenho como critérios de seleção e divide as requisições de clientes por classes de prioridade. Tais índices são compostos por indicadores de CPU e memória dos provedores de serviço, e seu cálculo foi definido a partir de uma adaptação de algoritmo proposto por (Branco, 2004). A técnica proposta por (Branco, 2004) utiliza Distância Euclidiana para calcular o índice desempenho que varia de 0 a 1. Para o algoritmo de classificação utilizado pelo WSARCH, três classes foram definidas (*Gold*, *Silver* e *Bronze*), utilizando o cálculo do índice de desempenho como base. Os clientes da classe *Gold*, no caso, têm suas requisições processadas em provedores com menor indice de desempenho (próximo a 0), o que indica provedores com menor utilização de recursos computacionais, enquanto os clientes da classe *Bronze* têm requisições processadas em provedores com maior índice de desempenho. O serviço hospedado em um provedor de serviços com índice de desempenho mais próximo da média é escolhido para a classificação dos clientes da classe *Silver*. Um problema do algoritmo de classificação é que, mantidas as mesmas proporções de chegada de requisições para clientes *Gold* e *Bronze*, o processamento das requisições dos clientes da classe Gold pode sofrer interferências. Tais observações serão discutidas na Seção 5.4.

O segundo algoritmo, denominado Algoritmo de Reserva de Recursos (RSV), é uma adaptação do primeiro algoritmo, utilizando também o índice de desempenho como principal critério. Este tem como objetivo reservar uma certa quantidade de recursos computacionais para cada classe de clientes. Por exemplo, os clientes da classe *Gold* terão mais provedores disponíveis que os clientes da classe *Silver* e estes por sua vez mais provedores disponíveis que os clientes da classe *Bronze*. A estrutura da arquitetura WSARCH possibilita a alteração e manutenção de outros tipos de algoritmos de seleção. A adoção de algoritmos eficientes requer que um estudo mais aprofundado sobre o tema seja realizado, envolvendo a relação de outros tipos de atributos de QoS ou outros tipos de informações relevantes.

Aplicação cliente

O papel da aplicação cliente na arquitetura WSARCH se resume principalmente em invocar o serviço preterido através do *broker*, passando como parâmetros de entrada as informações do serviço, assim como informações de QoS. Para o protótipo desenvolvido durante este projeto, as aplicações clientes foram implementadas tendo como objetivo a avaliação de desempenho. Os Web services invocados pelo cliente foram projetados especificamente para gerar cargas de naturezas variadas, como será abordado na Seção 4.4. Além disso, um servidor de log foi instalado para monitorar cada componente da arquitetura e, em especial, a aplicação cliente, pois é onde verifica-se o tempo total de uma requisição. Também foram desenvolvidos um conjunto de scripts para executar processos simultâneos de aplicações que invocam uma certa quantidade de requisições ao *broker* para simular um ambiente de testes com cargas específicas. As seguintes informações são enviadas ao *broker* durante uma requisição ao serviço:

- Nome do serviço: nome do serviço preterido pelo cliente;
- Parâmetros do serviço: parâmetros do serviço armazenado nos provedores;
- Informações de QoS: serão utilizados pelo algoritmo de seleção utilizado pelo *broker*;
- Informações sobre o experimento: necessários para registrar informações sobre cada requisição do cliente, como identificação da requisição e do experimento.

Como o *broker* também trata-se de um serviço disponibilizado para o cliente, para que uma requisição seja executada com sucesso, alguns pré-requisitos são necessários: o cliente deve saber a localização do broker e também como invocar e realizar uma requisição ao *broker*. Para isso, o *broker* disponibiliza um documento WSDL que descreve todas as operações disponibilizadas pelo *broker*, e assim permite ao cliente gerar sua própria estrutura para realizar a requisição.

Provedores de serviço

Os provedores de serviço foram implementados utilizando um motor de processamento de mensagens *SOAP* denominado *Axis2* (Axis2, 2009), este acoplado a um container de *servlets* chamado *Tomcat*. Diversos Web services com características distintas foram implantados nos provedores de serviço e disponibilizados para acesso ao *broker* e UDDI. Os provedores de serviços recebem as mensagens SOAP provenientes do *broker* através dos receptores de mensagens pertencentes ao Axis2, realizam o *parser* da mensagem em XML, executam o serviço e por fim retornam ao *broker* uma resposta encapsulada, que posteriormente é reencaminhada para o cliente. Assim que a aplicação é desenvolvida e implantada no provedor, e em seguida publicada no repositório de serviços UDDI, ela pode receber requisições provenientes do *broker*. Vale ressaltar que nesta etapa de implementação das aplicações provedoras foi necessário realizar uma alteração no código fonte do *Axis2* para que este coletasse informações do tempo de processamento de cada serviço implantado nos provedores. Essa alteração foi realizada nos receptores de mensagens que são parte integrantes do *Axis2*.

Registro UDDI

Um registro UDDI foi implementado na arquitetura WSARCH usando a ferramenta jUDDI, a qual simula os conceitos de repositório de informações de serviço na linguagem Java. Como foi discutido anteriormente, o registro UDDI possui como sua principal funcionalidade o armazenamento de informações funcionais sobre os Web services tais como informações sobre os provedores, implementações e metadados de serviços. Além disso, para os propósitos do projeto, o registro UDDI foi modificado para contemplar informações de QoS sobre cada serviço publicado. Tais informações podem ser atualizadas através de monitores de sistemas que enviam periodicamente informações relacionadas aos provedores monitorados para o UDDI. Para a implementação

do protótipo inicial da arquitetura, um monitor ganglia foi utilizado para coletar informações de CPU e memória de cada provedor e formar índices de desempenhos para assim serem registrados como informações de QoS no registro UDDI. O *broker*, por sua vez, coleta tais informações do registro UDDI e seleciona o serviço adequado de acordo com os critérios de seleção utilizados. Aplicações de diferentes características foram implementadas em provedores de serviços distintos e publicadas nesse registro UDDI para fins experimentais.

4.3 Contribuições no desenvolvimento da WSARCH

O autor desse trabalho de mestrado participou ativamente da implementação da arquitetura WSARCH desde sua etapa inicial, colaborando desde a implementação de cada módulo individual da arquitetura, até a integração deles. Dentre as colaborações realizadas pelo autor incluem:

- o planejamento da estrutura de implementação do *broker* em sub-módulos;
- a instalação do servidor de Log, permitindo a coleta de informações da arquitetura;
- a implementação de novas funcionalidades no *broker* para, por exemplo, permitir a manipulação de transferência de mensagens com anexos (*WS-Attachment*) na arquitetura.

A divisão em sub-módulos do serviço responsável pelas funcionalidades do broker foi necessária principalmente para possibilitar a inclusão de diferentes tipos de algoritmos de seleção, além de facilitar na manutenção do serviço, aproveitando-se das vantagens da orientação a objetos. As ferramentas de monitoração implementadas no decorrer desse projeto também foram organizados em módulos. Para a instalação do servidor de Log, uma base de dados foi estruturada e os principais elementos da arquitetura (cliente, broker e provedor) foram adaptadas de forma a permitir acesso ao banco de dados utilizando interfaces baseadas em JDBC. Isso também resultou na modificação de alguns módulos fornecidos pelo *Axis2* nos provedores, para, desta forma, possibilitar a coleta do tempo de processamento de um serviço no provedor. Além disso, o autor também auxiliou na validação e avaliação da WSARCH, construindo aplicações de diferentes características que são hospedadas nos provedores e acompanhando todo o processo de execução dos experimentos, coleta e manipulação de dados.

Devido à complexidade para a inicialização de cada execução de um experimento, e a diversidade de variáveis utilizadas para definir um experimento, o mestrando também contribuiu na implementação de alguns scripts em linguagem Shell script para automatizar algumas etapas para a inicialização e manipulação de dados dos experimentos realizados. O script run.sh para a preparação do ambiente de testes é dividido em três etapas:

1. **Reboot** Nesta etapa as máquinas provedoras, o *Broker* e o UDDI são reinicializados. Essa fase é fundamental para garantir a homogeneidade dos índices de carga e desempenho no início de cada execução dos experimentos.

2. **initFirewall** Script de inicialização do firewall no broker. Permite interligar duas redes, ou seja, os clientes que se encontram em uma rede podem acessar os serviços dos provedores que se encontram na outra rede com o firewall ativado.

3. **initWsarch** Esta etapa executa um conjunto de operações para preparar todas as máquinas (exceto clientes) para a execução de um experimento. É responsável por inicializar o servidor Tomcat de todas as máquinas, além de preparar a monitoração da arquitetura usando o Ganglia.

Após a execução do run.sh, os índices de desempenho atuais são verificados para evitar inconsistências. Em seguida um script responsável pelos clientes é executado. Sua tarefa é garantir que todos os clientes executem suas tarefas ao mesmo tempo. Para cada cliente, um arquivo de configuração do crontab é gerado contendo o horário que sua tarefa será processada. Cada cliente, por sua vez, possui um script contendo informações de execução para um experimento específico, tais como parâmetros de entrada para a invocação do broker, quantidade de processos e requisições por processo, classe do cliente, entre outros.

4.4 Categorias de Aplicações

A arquitetura WSARCH utiliza como principal funcionalidade um escalonador de serviços acoplado no broker que seleciona o serviço mais adequado para uma requisição de acordo com as informações de QoS solicitadas pelo cliente. No entanto, dependendo de como o algoritmo de seleção de serviços é implementado, este pode apresentar comportamentos distintos em termos de desempenho. Um dos fatores que pode influenciar consideravelmente o escalonador é que as aplicações disponibilizadas pelos provedores podem ser de naturezas distintas. Tais diferenças, por sua vez, podem afetar diretamente em como o roteamento/escalonamento dos serviços deve ser feito, e assim, influenciar no desempenho obtido pelo broker. Considerar apenas um tipo de aplicação pode ser considerada uma grande limitação para o sistema computacional. Essas diferenças nas aplicações podem ser agrupadas sob diferentes pontos de vista. Uma forma de classificar as aplicações é considerar quais são os recursos computacionais que estão sendo utilizados pelo serviço. As aplicações, durante sua execução, tendem a utilizar um recurso computacional específico (ou mais de um recurso) mais intensamente que os outros. Considerando tais observações, as aplicações podem ser classificadas em (Voorsluys, 2006):

CPU-bound: conhecidas também como *computation-intensive*, são aplicações que demandam muito uso de processamento do CPU. Processos gerados por esse tipo de aplicação geralmente mantém o processador ocupado a maior parte do tempo desde o momento em que foram escalonados para execução. Geralmente são compostas por operações matemáticas complexas, buscas intensivas, processamento de elementos gráficos, entre outros.

Memory-intensive: nesta categoria se enquadram aplicações que manipulam grandes conjuntos de dados e realizam operações de escrita e acesso com a memória principal. Em geral, aplicações desse tipo também podem ser consideradas *CPU-Bound*, visto que a manipulação de grandes quantidades de dados geralmente necessitam um processamento pesado desses dados. Como exemplos destacam-se: programas de data mining, modelos climáticos, reconhecimento de voz buscas em grandes bases de dados, processamento de vídeo, entre outros.

IO-bound: representam aplicações que realizam muitas chamadas a dispositivos de entrada e saída. Em geral, tais dispositivos referem-se a dispositivos de armazenamento secundário como disco rígido, pen drive, entre outros. Aplicações dessa categoria realizam operações de leitura e escrita nesses dispositivos de forma intensiva. Aplicações de geração de logs são um exemplo para essa categoria.

Network-bound: aplicações que efetuam a comunicação em rede de forma intensiva são do tipo *network-bound*. Aplicações pertencentes a essa categoria realizam muita transmissão de pacotes pela rede em ambientes distribuídos como a Internet. É geralmente um das principais problemas que afetam consideravelmente o desempenho de um sistema computacional.

Como pode ser observado, tais categorias de aplicações foram baseadas na intensidade no qual determinado recurso computacional é utilizado. É interessante destacar que essas classes de aplicações não são disjuntas, ou seja, aplicações podem pertencer a uma ou mais classes. Há aplicações que não demonstram uma tendência bem definida para serem consideradas aptas a pertencerem a uma única classe. Por exemplo, é possível que uma aplicação seja *IO-Bound* e *Memory-Intensive* ao mesmo tempo. Além disso, as aplicações podem ter comportamentos alterados durante sua execução, exibindo tendências de utilização de recursos distintas em diferentes fases de sua execução. Por exemplo, uma aplicação pode adotar o comportamento *Network-bound* durante uma ou mais fases e em outras se tornar uma aplicação do tipo *CPU-Bound*. (Voorsluys, 2006)

Outras formas de classificação também foram definidas considerando outros tipos de características comuns. As aplicações também podem ser classificadas de acordo com o grau de interação durante o tempo de execução. Segundo (Saphir et al., 1995), as aplicações podem ser divididas em *interativas* e *batch*. As aplicações interativas são aplicações que necessitam de grande interação com os usuários. Um dos fatores de qualidade mais importantes para esse tipo de aplicação é o tempo de resposta, devendo ser o mais rápido possível, evitando possíveis atrasos para o usuário. Por outro lado, as aplicações batch referem-se aos arquivos em lote. Ao contrário das aplicações interativas, os usuários de aplicações batch não esperam uma resposta imediata, possibilitando que os focos de qualidade sejam mais flexíveis em relação às aplicações interativas. Todas as classes de aplicação mencionadas anteriormente podem ser ainda do tipo interativo ou batch, desde que suas características se enquadrem nos critérios estabelecidos (Figura 4.4).

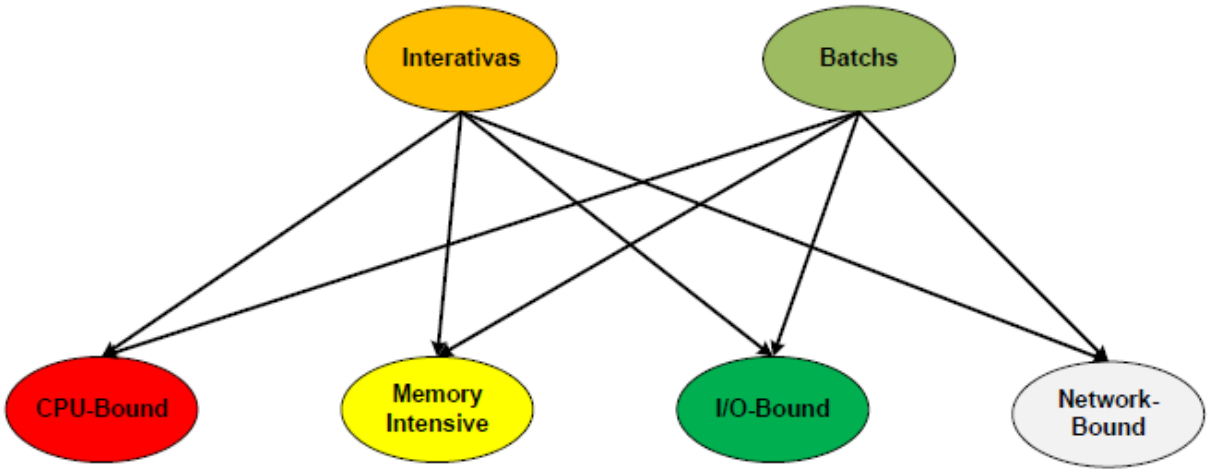

Figura 4.4: Classificação das Aplicações - (Branco, 2004)

4.4.1 Implementação das aplicações

Para concretizar o modelo da arquitetura proposta por (Estrella, 2010) descrita nesse capítulo, um protótipo inicial foi implementado. Sua validação foi realizada utilizando uma aplicação inteiramente *CPU-Bound* composta por simples operações matemáticas dentro de alguns laços aninhados na ordem de $O(n^4)$. Para os propósitos desse projeto de mestrado, aplicações de diferentes características foram implementadas e testadas. Uma delas consiste em explorar operações de leitura e escrita em disco, simulando a execução de aplicações *IO-Bound*. Para essa aplicação, o usuário pode invocar o serviço através do broker definindo como um dos parâmetros o tamanho do arquivo texto que será gerado no servidor.

Para analisar características de desempenho relacionadas às aplicações distribuídas com carga na rede (*Network-Bound*), foram implementadas aplicações de transferência de dados anexados às mensagens SOAP, utilizando as técnicas MTOM e SwA, os quais foram descritas na Seção 2.6. Para permitir que anexo de dados fosse utilizado na WSARCH, foi necessária implantar uma nova funcionalidade na arquitetura que contemplasse a correta transferência das informações entre provedor, Broker e aplicação cliente.

Por fim, uma aplicação mais robusta considerando diferentes tipos de categorias foi implementada. Nesta aplicação, as operações foram definidas de forma delimitada e sequencial para cada categoria, ou seja, após a requisição ser enviada para o provedor, uma operação de CPU-Bound é processada inicialmente, seguida de uma operação de escrita de arquivo em disco ((IO-Bound)). Por último, um arquivo binário é transferido do provedor para o cliente, realizando uma operação de *download*. Essa aplicação pode ser assimilada como uma aplicação real que executa uma tarefa que exige processamento do servidor, gera um relatório de log que é gravado no servidor e em seguida envia esse relatório para o usuário.

Assim, em resumo, quatro aplicações com escopos distintos foram implementadas durante o desenvolvimento desse projeto: *CPU-Bound*, *IO-Bound*, *Network-Bound* e aplicação mista. A

categoria *memory-intensive* não foi considerada devido às suas características em termos de desempenho serem similares aos de *CPU-Bound*. Um estudo mais detalhado de desempenho considerando tais aplicações implementadas é apresentado na Seção 5.5.

4.5 Considerações Finais

Este capítulo abordou sobre uma arquitetura proposta com a finalidade de prover qualidade de serviço em Web services, apresentando sua estrutura de maneira ampla e detalhada, os módulos e suas interações, e uma descrição sobre o desenvolvimento de um protótipo da arquitetura. Em seguida, foram discutidas as principais contribuições do autor durante o desenvolvimento da arquitetura WSARCH, os quais foram essenciais para a conclusão de um protótipo inicial da arquitetura. Além disso, a construção de um protótipo da arquitetura WSARCH foi fundamental no contexto deste projeto de mestrado, pois com isso é possível realizar experimentos relacionados com diferentes tipos de aplicações, os quais também foram apresentados neste capítulo.

O Capítulo 5 discorre sobre os experimentos de avaliação de desempenho realizados no decorrer deste projeto, apresentando alguns resultados obtidos a partir do protótipo contruído com as especificações da arquitetura WSARCH, desde alguns testes de validação até experimentos para avaliação de desempenho de aplicações de diferentes categorias, os quais foram descritas neste capítulo.

CAPÍTULO

5

Avaliação de Desempenho das Aplicações na WSARCH

5.1 Considerações iniciais

Este capítulo descreve os principais projetos experimentais de avaliação de desempenho realizados no decorrer deste projeto, apresentando inicialmente estudos sobre desempenho com *attachments* em Web services e operações em UDDI, os quais serviram como base para a construção do protótipo da arquitetura WSARCH, além de prover melhor entendimento sobre o comportamento em termos de desempenho dos componentes a serem implantados no protótipo da arquitetura. Em seguida, os primeiros projetos experimentais definidos para a validação da arquitetura WSARCH são descritos, realizando uma análise comportamental dos componentes da arquitetura e verificando se a WSARCH atinge seu objetivo como uma ferramenta para avaliação de desempenho. Por fim, são discutidos os resultados obtidos para verificar a influência de diferentes categorias de aplicações na arquitetura WSARCH, o que representa um dos principais objetivos deste trabalho de mestrado.

5.2 Estudo sobre Desempenho em WS-Attachments

Nesta seção são apresentados os primeiros experimentos realizados no decorrer deste projeto, cujos resultados foram submetidos e aceitos na **ICWS** (*International Conference on Web Services*) em fevereiro de 2009. Neste trabalho foram realizados alguns experimentos considerando o anexo

de dados binários em mensagens SOAP (*Web Services Attachments*[1]), onde as principais técnicas de *attachment* foram avaliadas e analisadas, verificando-se também a influência de diferentes redes e a influência de arquivos de tamanhos distintos. Os experimentos foram realizados em um protótipo denominado **WSATPerf**, construído para prover um ambiente de experimentação envolvendo aspectos relacionados com anexos de dados binários em mensagens SOAP.

5.2.1 Aplicação WSATPerf

A aplicação **WSATPerf** (*Web Services Attachment Performance Evaluation*) foi desenvolvida visando avaliar o desempenho das operações de envio e recepção de mensagens contento quaisquer tipo de dados, tanto em formato texto, quanto arquivos binários, encapsulados em uma mensagem SOAP. No atual estágio de desenvolvimento, esta ferramenta permite enviar (*download*) ou receber (*upload*) dados de/para um provedor de serviços usando diferentes abordagens. Dentre as funcionalidades implementadas até o momento, é possível se obter o tempo de resposta variando diversos fatores envolvendo essas operações, tais como o tipo e tamanho de arquivo, e as técnicas de *attachment*. Essa ferramenta é composta por dois módulos principais: provedor e cliente.

A *WSATPerf* foi implementada na linguagem Java e utiliza uma API para processamento de mensagens SOAP disponibilizada pelo *framework* Apache Axis2[2]. Essa API é chamada de **AXIOM** (Axis Object Model) e permite a construção em baixo nível de mensagens SOAP – provendo algumas facilidades – e a manipulação do seu conteúdo. O AXIOM accessa o fluxo de dados XML através da interface StAX[3] empacotado pela interface do construtor da mensagem. Basicamente, o construtor lê a informação do interpretador XML e constrói um modelo de objeto. O desenvolvedor pode utilizar um dos três construtores no AXIOM: StAXOMBuilder, que constrói a árvore Axiom; StAXSOAPModelBuilder, que pode ler mensagens SOAP e construir um modelo de objeto SOAP baseado no Axiom; e MTOMStAXSOAPModelBuilder, que permite ler mensagens MTOM. O Axiom também fornece uma opção para eventos StAX com ou sem a construção de um modelo de memória para acessos futuros. Este conceito, denominado cacheamento, permite o uso de memória de forma apropriada e controlada (Estrella, 2008).

Módulo Provedor de Serviços

O provedor de serviços implementa um Web service com duas operações: *upload* e *download*. O serviço cliente faz uma requisição para o provedor de serviços e seleciona a opção desejada. Cada operação encapsula os dados binários dentro de uma mensagem SOAP e envia/recebe estes dados de/para o provedor de serviços. O tempo de resposta é calculado a partir do momento em que o cliente faz a requisição até quando o cliente recebe uma resposta do provedor de serviços avisando que os dados requisitados foram armazenado no disco, tanto para operações de *download*

[1] Anexo de dados binários, seus conceitos foram apresentados na Seção 2.6
[2] http://ws.apache.org/axis2/
[3] Serviço disponibilizado pelo Axis2 para processamento de documentos XML

quanto de *upload*. Cada operação descrita no código-fonte do provedor de serviços foi implementada utilizando receptores de mensagens acoplados no *kernel* Apache Axis2.

Módulo Cliente do Serviço

Este módulo, localizado no cliente da arquitetura, é responsável por coletar todas as informações necessárias para realizar a troca de dados binários entre o cliente e o provedor de serviços. Dentre estas informações, inclui a escolha de parâmetros como tipo e tamanho de arquivo, tipo de operação (*upload* e *download*), número de threads, requisições por thread e técnicas de *attachment*. Além disso, o módulo cliente armazena os resultados em um arquivo de *log*, como pode ser observado na Figura 5.1.

Cada *log* gerado pela ferramenta corresponde apenas a uma largura de banda específica utilizada durante os experimentos. O arquivo de log armazena o tipo de técnica, o tamanho de arquivo, os tempos de resposta mínimo, médio e máximo, desvio padrão e intervalo de confiança do tempo de resposta, e a percentagem de requisições que falharam em relação ao total de requisições do experimento. Os resultados obtidos através da avaliação de diferentes técnicas usando diferentes larguras de banda de rede foram coletados utilizando o arquivo de *log*.

5.2.2 Configuração do Ambiente

O principal objetivo da ferramenta *WSATPerf* foi avaliar o desempenho das técnicas de transferência de arquivos em *Web Services* sobre a influência de diferentes redes. Assim, dentre os elementos que compõem o ambiente de testes, encontram-se duas máquinas: uma representando um cliente e outra um servidor (provedor de *Web Services*). Uma terceira máquina se comporta como um *gateway*, responsável pelo controle de banda da transmissão de arquivos entre o cliente e o servidor. No ambiente definido, o cliente e o servidor encontram-se em redes distintas e, para permitir a comunicação entre eles, o *gateway* utiliza duas interfaces de rede (duas placas de rede *gigabit*) e um *firewall*. A Figura 5.2 ilustra a interação entre o cliente, *gateway*, e servidor.

As máquinas utilizadas são homogêneas, ou seja, possuem a mesma configuração de *hardware*, como mostra a Tabela 5.1.

Tabela 5.1: Informações de hardware para a configuração do ambiente de testes.

Configuração de Hardware	
Sistema Operacional	Ubuntu Linux 8.04
CPU	Intel Core 2 Quad - 2.4 GHz
Memória	2GB
Tananho do Disco	250GB
Switch	3Com Gigabit Ethernet
Interface de Rede	Realtek Gigabit Ethernet

Atts	Size	Average	Min	Max	ConfLOW	ConfHIGH	Deviation	Failures	Success
swa	file10KB	1,8328	1,7634	1,8847	1,7559	1,9097	0,0393	0	144
swa	file10KB	1,8179	1,7463	1,8777	1,7249	1,9108	0,0474	0	144
swa	file10KB	1,8291	1,7454	1,8831	1,7475	1,9108	0,0417	0	144
swa	file10KB	1,8207	1,7460	1,8816	1,7377	1,9038	0,0424	0	144
				...					
swa	file100KB	4,0434	3,2718	4,3691	3,1684	4,9184	0,4464	0	144
swa	file100KB	4,1834	4,0533	4,2878	4,0039	4,3629	0,0916	0	144
swa	file100KB	4,2129	4,0539	4,3170	4,0074	4,4185	0,1049	0	144
				...					
swa	file5MB	45,7467	16,1590	61,8850	15,3766	76,1168	15,4949	5	139
swa	file5MB	31,2846	13,4929	46,1452	12,0447	50,5246	9,8163	4	140
swa	file5MB	39,0804	25,5328	61,8955	16,8407	61,3201	11,3468	5	139
				...					
mtom	file10KB	1,8578	1,7831	1,9240	1,7586	1,9570	0,0506	0	144
mtom	file10KB	1,8507	1,7805	1,9252	1,7524	1,9490	0,0502	0	144
				...					
mtom	file10MB	47,0701	17,3096	77,2769	9,5502	84,5901	19,1428	5	139
mtom	file10MB	52,5231	17,9993	74,0719	12,0731	92,9730	20,6377	6	138
mtom	file10MB	41,2639	14,0074	63,2028	15,2959	67,2319	13,2490	7	137
				...					
binary	file5MB	52,3386	12,0349	85,5313	1,8144	102,8628	25,7776	16	128
binary	file5MB	49,2648	20,9323	74,3623	13,2986	85,2310	18,3501	17	127
binary	file5MB	48,5597	12,0776	68,5310	7,6664	89,4530	20,8639	23	121
				...					

Figura 5.1: Arquivo de log para o módulo cliente

Além disso, algumas ferramentas e tecnologias foram utilizadas para o desenvolvimento do ambiente de experimentos, dentre eles:

- **Apache Tomcat**: Responsável por disponibilizar os serviços no provedor de serviços.
- **Apache Axis2**: Motor de processamento de mensagens *SOAP*.
- **Iptables**: Usado neste contexto para permitir que duas redes conversem entre si de forma transparente via *NAT* (*Network Address Translation*).
- **CBQ/iproute**: Responsável pelo controle de banda no *gateway*. *CBQ* (Floyd, 1995) é um algoritmo de gerenciamento de tráfego desenvolvido pelo grupo de pesquisa de redes de computadores do *LBNL* (*Lawrence Berkeley National Laboratory*) (Floyd et al., 2009).
- **GenFile**: Um gerador para criar os arquivos XML utilizados nos experimentos. Este gerador é capaz de gerar arquivos de tamanhos especí ficos escolhidos pelo usuário (Estrella, 2008).

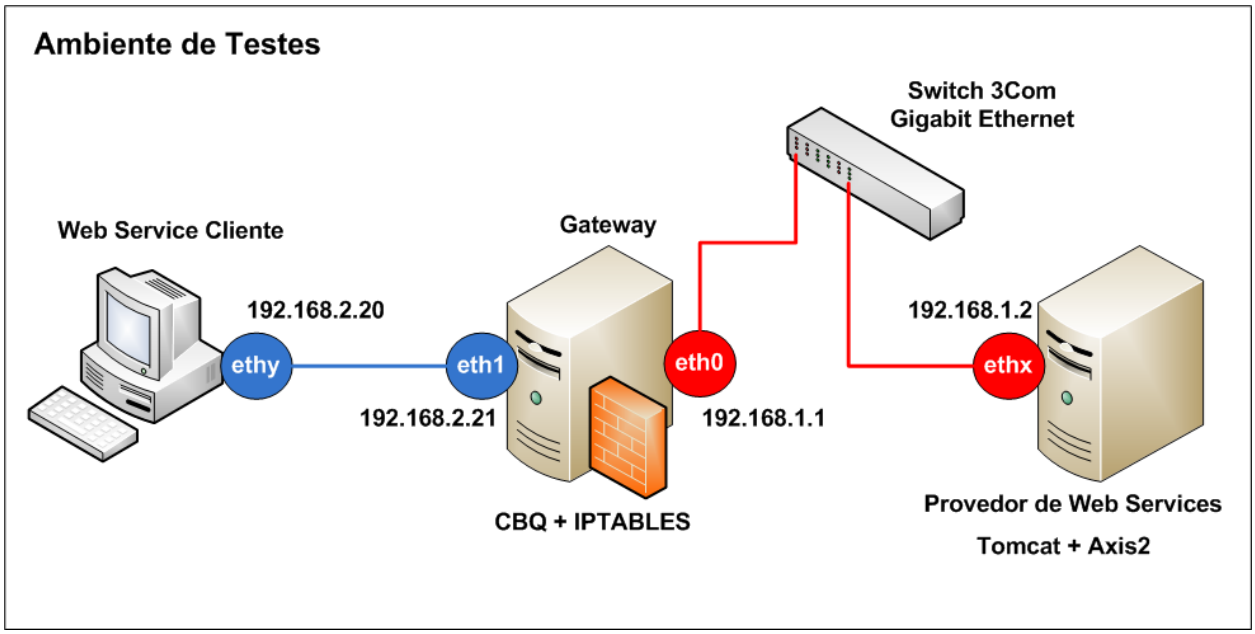

Figura 5.2: Ambiente de testes

O processo de execução dos experimentos foi realizado utilizando a ferramenta *WSATPerf* implementada especificamente para avaliar dados anexados em mensagens *SOAP* sob diversas perspectivas. Tal ferramenta é parametrizável, de forma a permitir a configuração de vários fatores como mostra a Tabela 5.2.

Tabela 5.2: Fatores customizáveis da ferramenta de avaliação.

Parâmetros Customizáveis	
Número de Threads	1 a 12
N. de interações por thread	1 a 12
Tipo de operação	download, upload
Técnicas de Attachment	pure binary, MTOM, SwA
Tipo de arquivo	texto, binário (image, audio, video)
Tamanho de arquivo	(10, 100) KB, (1, 5, 10) MB
Largura de banda	100Kbps, (1, 10, 100)Mbps
Cache	sim ou não

5.2.3 Planejamento de experimentos

A execução dos experimentos foi realizada variando três fatores, de forma a permitir a verificação da influência em relação às técnicas de *attachments*, rede e tamanho de arquivos. A definição da parametrização destes fatores pode ser observada na Tabela 5.3.

Outro ponto fundamental a ser considerado na fase de planejamento de experimentos é a definição dos fatores fixos que devem ser considerados na parametrização da aplicação utilizada. Tais fatores são evidenciados na Tabela 5.4.

Tabela 5.3: Parametrização dos fatores utilizados nos experimentos.

Parametrização dos Experimentos	
Técnicas de Attachment	pure binary, MTOM,SwA
Tamanho de Arquivo	(10, 100) KB, (1, 5, 10) MB
Largura de banda	100Kbps, (1, 10, 100)Mbps

Tabela 5.4: Valores fixos associados a fatores utilizados nos experimentos.

Parâmetros Fixos	
Operação	upload
Número de threads	12
Número de iterações por thread	12
Timeout de Conexão HTTP	4 minutos
Espaço de Pilha Java	1024 MB
Tipo de Arquivo	texto

Sendo assim, para cada experimento, o cliente dispara 12 *threads*, no qual cada *thread* envia um arquivo por vez (no total de 12 arquivos) para o provedor de serviços. Para evitar excesso de perda de requisições, foram definidos um *timeout* de 4 minutos e um espaço de pilha *Java* de 1GB, pois em algumas técnicas a influência da memória é bastante evidente. Estes parâmetros foram definidos tanto no lado do cliente quanto no lado do provedor. Os arquivos de tamanhos específicos foram gerados de forma aleatória em formato texto. A análise de resultados foi feita baseada no tempo médio de resposta de cada *thread*. Somente são considerados os tempos de resposta de requisições bem sucedidas, ou seja, resultados obtidos por meio de requisições não-completadas não são consideradas no cálculo do valor médio do tempo de resposta. Entretanto, estas requisições mal sucedidas são consideradas como variáveis de resposta para gráficos que demonstram o comportamento das *threads*.

De maneira mais detalhada, para cada *thread*, é calculado o tempo médio de resposta de cada iteração com requisição aceita. Deste modo, são obtidos os tempos de resposta médio de cada *thread*. Assim, a variável de resposta é obtida calculando-se a média dos tempos de resposta médio obtido por cada *thread*, resultando também o desvio padrão, a variância, tempos de resposta máximo e mínimo, e o intervalo de confiança, com 95% de grau de confiança. Para a apresentação dos resultados, 10 execuções foram feitas para cada experimento. Isso permite maior confiabilidade para a análise dos resultados.

5.2.4 Análise de Resultados

Nesta seção é realizada uma análise sobre a influência da rede e das técnicas de *attachments* com base nos resultados obtidos durante os experimentos. A influência da rede foi verificada através de gráficos de intervalos de confiança, onde cada coluna caracteriza um nível de largura de banda da rede. Também foi fixado um nível de tamanho de arquivo e uma técnica específica

para cada gráfico. A influência das técnicas de attachment foi verificada de forma semelhante, mas usando técnicas representando cada coluna, em vez da largura de banda da rede. O tempo de resposta médio é ilustrado por cada uma das caixas do gráfico, onde a linha divisória da caixa representa o tempo médio em si e o limite superior e inferior desta caixa representam o intervalo de confiança. A Figura 5.3 mostra uma legenda que resume a forma como os resultados apresentados nos gráficos das figuras 5.4 e 5.5 devem ser interpretados. Nota-se também a existência de uma legenda para a representação de cada uma das redes em termos de largura de banda.

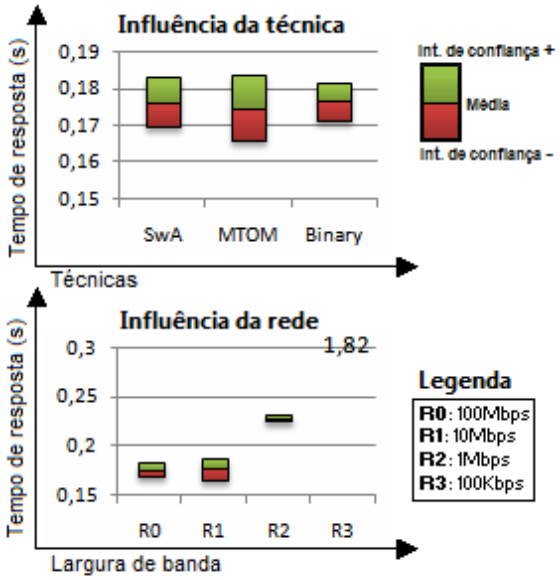

Figura 5.3: Legenda usada para a interpretação dos gráficos

Influência da rede

Na Figura 5.4 os diferentes tipos de redes considerados, e os diferentes tamanhos de arquivos foram analisados. Os resultados mostram que a variação para redes de menor largura de banda (100Kbps - R3) é mais evidente. Além disso, para uma rede de maior largura de banda, não há diferença perceptível no tempos médios de resposta entre as técnicas. Assim, devido à alta velocidade de transmissão na rede, este fator não é um fator limitante, sendo a variável de resposta mais influenciada em relação a outros fatores como o processamento das mensagens SOAP.

Há também algumas diferenças entre as redes de 1Mbps (R2) e 10Mbps (R3). Tais diferenças tornam-se um pouco mais definidas quando o tamanho do arquivo cresce. Isto mostra que ainda há uma certa influência sobre a rede no tempo de resposta das mensagens, mas menor do que em redes de largura de banda 100Kbps (R0) e 1Mbps (R1). Além disso, nota-se uma nítida influência do tamanho do arquivo nos experimentos. Outro ponto que é importante considerar é o comportamento

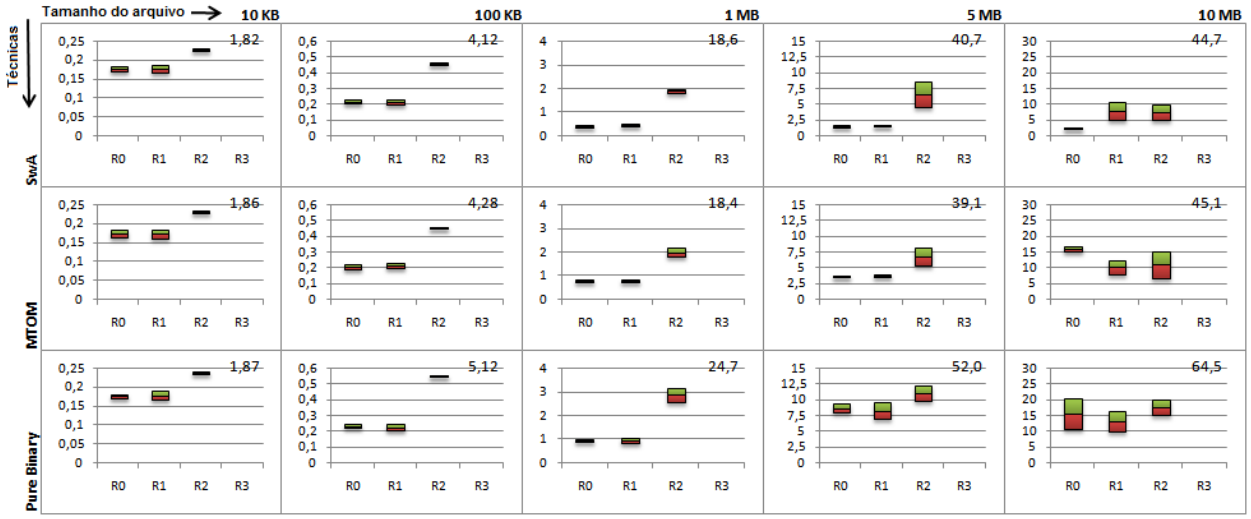

Figura 5.4: Influência da rede no tempo de resposta

anormal de arquivos de 10MB nos experimentos realizados. Isso acontece devido à influência de outros fatores como o uso excessivo de memória. Este fato é melhor explicado a seguir.

Influência das técnicas

As principais características que podem afetar o desempenho das técnicas avaliadas são o tempo de processamento das mensagens *SOAP* para identificar o conteúdo anexado, e a memória utilizada para armazenamento de requisições em espera que aguardam a liberação do processamento. Tais fatores podem ser percebidos pela maneira em que uma determinada técnica pode anexar seu arquivo na mensagem *SOAP*. Por exemplo, enquanto o *MTOM* e *SwA* utilizam referências usando *URI* para anexar as informações, o *Pure Binary* anexa as informações no corpo <SOAPBody> da mensagem *SOAP*. O processo de conversão *base64* utilizado em *Pure Binary* para incluir o conteúdo dentro da mensagem *SOAP*, e converter de volta para o formato original, envolve um custo maior de processamento em relação às outras duas técnicas.

Além disso, pode-se notar diferenças menos perceptíveis mas evidentes em relação às técnicas *MTOM* e *SwA*, apesar de ambas anexarem as informações fora da mensagem *SOAP*. *MTOM* é uma técnica criada para resolver problemas de incompatibilidade em relação ao *SwA*, incluindo as referências de arquivos anexados em um *infoset XML*. Porém, este processo também pode representar um certo custo de desempenho, tanto de processamento, quanto de memória utilizada. Para arquivos grandes, o *MTOM* divide o arquivo em porções menores, referenciando cada porção no *infoset XML*. Entretanto, tais porções do arquivo podem não chegar na mesma ordem em que foram referenciadas, o que obriga a utilização de um *buffer*. O processo de serializar partes do arquivo implica em custo de processamento e também a alta utilização de memória devido à necessidade de *buffers* (Evdemon, 2005). Este comportamento é ilustrado nos gráficos da Figura 5.5, em que cada linha representa uma largura de banda de rede. Isto é mais evidente em arquivos maiores, onde a técnica *Pure Binary* obteve o pior desempenho. Os experimentos com *SwA* apresentaram

os menores tempos médio de resposta. Em arquivos de tamanho menor, não houve variações de tempo médio de resposta, pois exigem menos tempo de processamento em relação aos arquivos maiores.

Nos gráficos apresentados na Figura 5.5 é possível verificar uma grande variação no intervalo de confiança para a rede de 100Kbps. Isto ocorre devido a uma grande rajada de requisições em uma rede de baixa capacidade de transmissão, o que torna menos visível a diferença entre as técnicas. Neste caso, a velocidade da rede é mais influente que o tempo de processamento de mensagens SOAP.

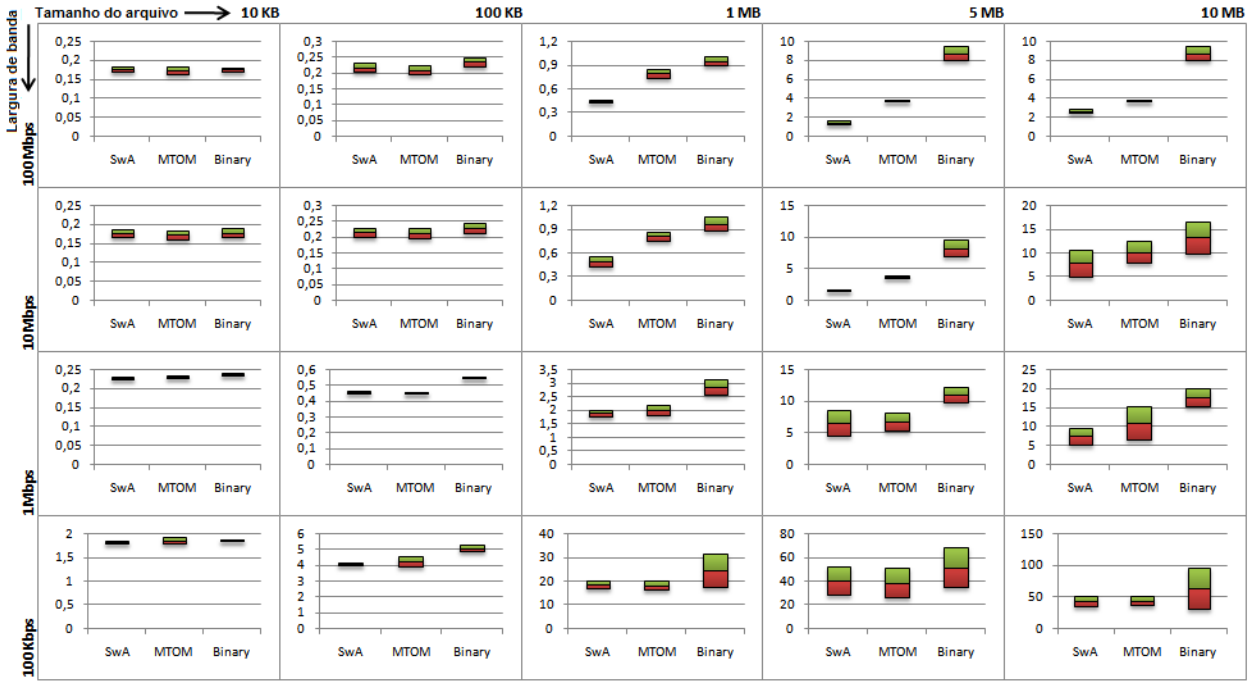

Figura 5.5: Influência das técnicas no tempo de resposta (s)

A Tabela 5.5 indica a percentagem média de requisições que falharam para cada experimento realizado. Estes valores são calculados utilizando o número de perda de pacotes entre as 144 requisições. Observa-se que as falhas são mais concentradas na rede de alta velocidade e com arquivos maiores. Isto acontece devido ao aumento do fluxo de dados a partir da alta taxa de transferência de arquivos. Consequentemente, o provedor de serviços necessita de mais memória para alocar as requisições em espera para a liberação de recursos para o processamento. Este problema é mais evidente com técnicas de *attachments* que requerem mais tempo de processamento de mensagens, principalmente a técnica *pure binary*. Nesta técnica, os dados são encapsulados dentro da mensagem e pela parametrização do experimento, o uso desta técnica utiliza muito recurso de CPU e rede, e assim, a tendência era apresentar mais erros em função disso. Algumas falhas também podem ocorrer quando a rede apresenta baixa velocidade de transmissão. A grande quantidade de requisições somada ao baixo fluxo de dados pode causar perdas de pacotes durante a transmissão dos arquivos.

Tabela 5.5: Erros/Acertos

Tamanho de arquivo	100Mbps			10Mbps		
	BIN	MTOM	SwA	BIN	MTOM	SwA
10KB	0%	0%	0%	0%	0%	0%
100KB	0%	0%	0%	0%	0%	0%
1MB	0%	0%	0%	0%	0%	0%
5MB	2,84%	0%	0%	4,16%	0%	0%
10MB	81,11%	51,80%	0%	61,59%	2,43%	1,66%
	1Mbps			100Kbps		
	BIN	MTOM	SwA	BIN	MTOM	SwA
10KB	0%	0%	0%	0%	0%	0%
100KB	0%	0%	0%	0%	0%	0%
1MB	0%	0%	0%	1,66%	0%	0,06%
5MB	1,80%	0,06%	0,13%	10,27%	2,36%	3,19%
10MB	24,37%	2,15%	2,15%	26,73%	4,65%	5,06%

5.3 Estudo sobre Desempenho em Operações do Registro UDDI

Outra atividade exercida durante o desenvolvimento desse projeto de mestrado foi realizar um estudo de avaliação de desempenho considerando interações de registros de serviço em uma arquitetura orientada a serviços. Os registros de serviços são componentes importantes que manipulam informações essenciais de Web services, como sua localização e como utilizá-los. No entanto, fazem parte de um ambiente distribuído, no qual podem interagir com outros componentes da arquitetura, podendo contribuir negativamente no tempo final de resposta, e assim, consequentemente, afetar o desempenho da arquitetura. Sendo assim, projetos experimentais foram realizados considerando diferentes tipos de carga de trabalho e larguras de banda em operações de consulta e publicação em um registro UDDI para fins de obtenção de resultados de desempenho. Para isso, uma ferramenta foi construída em um ambiente isolado considerando máquinas homogêneas, larguras de banda distintas e diferentes operações suportadas pelo servidor jUDDI. Os estudos e implementações realizados nessa etapa serviram como base para a construção do protótipo da arquitetura WSARCH.

5.3.1 Ferramenta WS-UDDIPerf

A Ferramenta **WS-UDDIPerf** (Web Services UDDI Performance Evaluation) foi projetada com o intuito de avaliar o desempenho de duas operações suportadas pelo servidor jUDDI[4]. No estágio atual, a ferramenta é capaz de realizar requisições de consulta para obter informações do serviço no servidor jUDDI, e também possibilita publicar dados sobre Web services implantados em provedores de serviço independentes. Requisições de consulta podem ser simples ou completas. Isso significa que o registro pode retornar uma informação específica ou detalhes completos sobre o serviço requisitado. A Figura 5.6 ilustra o funcionamento de duas operações: de publicação

[4]Implementação Java da especificação UDDI, versão 2.0

e de consulta. Tais operações utilizam bibliotecas proporcionadas pelo jUDDI no módulo *front-end* (cliente ou provedor) para possibilitar sua comunicação com o registro UDDI. A operação de publicação também inclui um estágio adicional para realizar o processamento do arquivo WSDL obtido para coletar informações sobre o Web Service, e assim, tais informações são enviadas ao servidor jUDDI. O banco de dados *MySQL* foi adotado para armazenar informações do servidor jUDDI devido a sua ótima compatibilidade com a ferramenta.

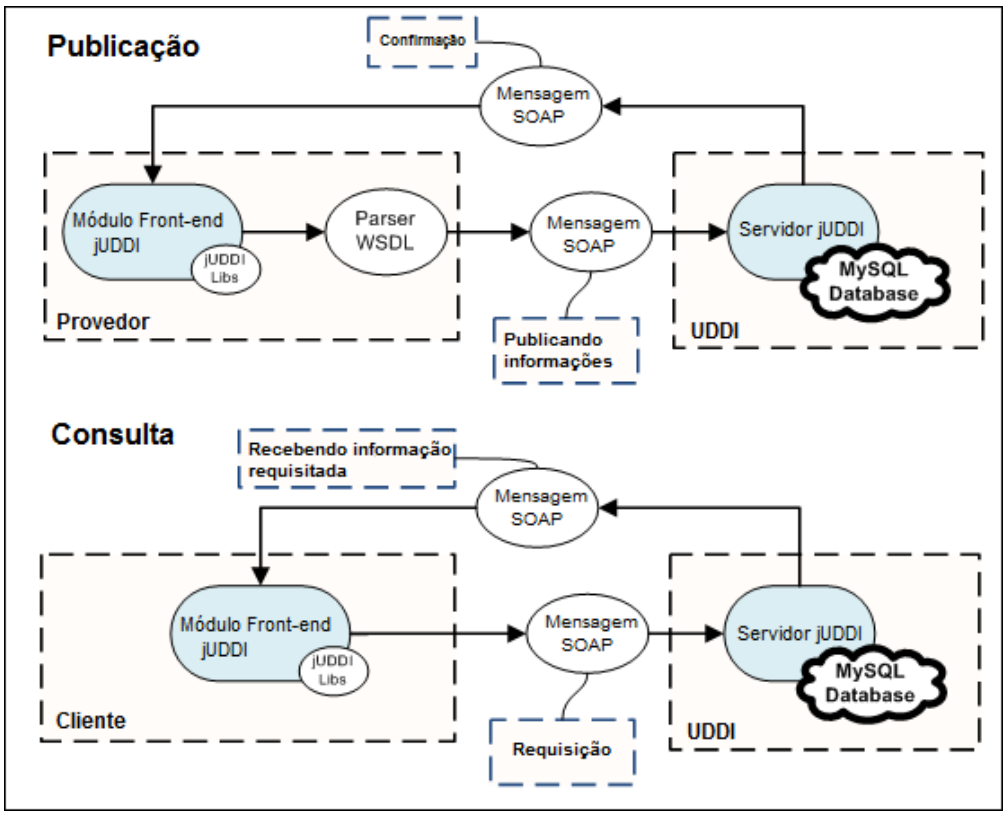

Figura 5.6: Operações de Consulta (Inquiry) e Publicação (Publish)

A ferramenta *WS-UDDIPerf* foi implementada na linguagem Java e utiliza uma API jUDDI para criar módulos cliente e de provedores de serviço. Cada módulo é responsável por realizar operações de consulta e publicação, respectivamente. Ambos os módulos são *multithread*, permitindo que requisições sejam realizadas no servidor jUDDI concorrentemente. Também é possível definir parâmetros como o tipo de operação, número de *threads* e número de iterações por *thread* para a execução desses módulos na ferramenta *WS-UDDIPerf* como apresentado na Tabela 5.6. Tais módulos são responsáveis por coletar dados referentes às operações realizadas no servidor jUDDI. Além disso, os resultados obtidos pela execução dos módulos são armazenados em um arquivo de *log* como ilustrado na Figura 5.7.

Cada log gerado pela ferramenta corresponde a uma única largura de banda utilizada durante os experimentos. O arquivo de log armazena o tipo de operação, tempos de resposta mínimo, médio e máximo, desvio padrão e intervalo de confiança, além da percentagem de falhas das requisições.

Tabela 5.6: Parâmetros customizáveis na ferramenta WS-UDDIPerf

Parâmetros customizáveis	
Número de *threads*	Mais de 1
Número de iterações por *thread*	Mais de 1
Operação	Publicação ou Consulta
Complexidade do WSDL	WSDL utilizado para obter informações sobre o serviço
Largura de banda	Maior que 100Kbps
Número de registros (banco de dados)	Mais de 1

A partir dos resultados de cada log, um novo arquivo é gerado representando a média de tempo de resposta de cada experimento.

Operation	Nthreads	Average	Min	Max	ConfLO	ConfHI	Dev
SimpleInq	10	0,8437	0,7983	0,8673	0,8030	0,8843	0,0207
SimpleInq	10	0,8446	0,8338	0,8505	0,8319	0,8574	0,0065
SimpleInq	10	0,8414	0,8282	0,8529	0,8251	0,8577	0,0083
...							
SimpleInq	15	0,9942	0,4701	1,1267	0,4777	1,5108	0,2635
SimpleInq	15	1,2727	1,2599	1,2797	1,2614	1,2841	0,0058
SimpleInq	15	1,2749	1,2601	1,2880	1,2604	1,2895	0,0074
...							
SimpleInq	20	1,3504	0,4838	1,5712	0,6251	2,0757	0,3700
SimpleInq	20	1,6803	1,6501	1,7243	1,6265	1,7341	0,0274
SimpleInq	20	1,6802	1,6330	1,7243	1,6227	1,7377	0,0293
...							
CompleteI	10	4,1023	3,9958	4,1441	3,9778	4,2267	0,0635
CompleteI	10	4,3226	4,2827	4,3496	4,2817	4,3635	0,0209
CompleteI	10	4,3250	4,2640	4,3604	4,2662	4,3837	0,0300
...							
CompleteI	15	6,5045	6,4566	6,5534	6,4491	6,5598	0,0283
CompleteI	15	6,4820	6,4147	6,5164	6,4120	6,5520	0,0357
CompleteI	15	6,4924	6,4428	6,5276	6,4390	6,5458	0,0272
...							
CompleteI	20	8,6515	8,5162	8,7660	8,5510	8,7521	0,0513
CompleteI	20	8,5271	8,3704	8,6461	8,3446	8,7097	0,0931
CompleteI	20	8,5943	8,3029	8,7358	8,4193	8,7694	0,0893

a) Consulta (Cliente)

Operation	Nthreads	Average	Min	Max	ConfLO	ConfHI	Dev
Publish	10	10,7087	10,2730	11,0472	10,2378	11,1797	0,2403
Publish	10	10,6523	10,5195	10,8896	10,4376	10,8670	0,1095
Publish	10	10,6165	10,3676	10,7454	10,3783	10,8547	0,1215
...							
Publish	15	15,4579	4,3188	16,6066	14,0784	16,8374	0,7038
Publish	15	15,5334	14,9443	16,2546	14,6450	16,4217	0,4533
Publish	15	15,7880	15,3106	16,1151	15,2553	16,3207	0,2718
...							
Publish	20	20,2825	18,5589	21,7569	18,4007	22,1644	0,9601
Publish	20	20,9440	19,9310	21,7008	19,8770	22,0109	0,5444
Publish	20	20,5868	19,0762	21,9549	19,1390	22,0346	0,7387

b) Publicação (Provedor)

Figura 5.7: Logs dos módulos provedor e cliente

Módulo Provedor de Serviços

Neste módulo foi implementado um programa que captura informações sobre um arquivo WSDL específico implantado (*deploy*) em um provedor de serviços e em seguida envia essa informação ao servidor jUDDI. Para obter informações sobre tais serviços implantados no provedor de serviços, é necessário realizar um *parser* (análise sintática) do WSDL do serviço e encapsular todas as informações coletadas na mensagem SOAP a ser enviada na base de dados do servidor jUDDI. O tempo de resposta da operação de publicação é mensurado quando o provedor realiza uma requisição enviando informações sobre o Web service e finaliza quando este recebe a resposta do servidor jUDDI retornando que todas as informações sobre o Web service foram armazenados na base de dados do jUDDI.

Módulo Cliente

No módulo cliente foi implementado um programa que obtém informações sobre um arquivo WSDL específico publicado no servidor jUDDI. Para adquirir tais informações sobre esse arquivo WSDL, o módulo cliente realiza uma requisição que encapsula uma mensagem SOAP contendo informação especificada do cliente, como por exemplo o nome da empresa responsável ou todos os serviços utilizados pela empresa. O critério de consulta feito pelo módulo cliente pode ser simples ou completo. A operação de consulta pode ser simples quando uma requisição carrega um único parâmetro, como por exemplo obter apenas o nome da empresa associada ao arquivo WSDL específico. Em geral, a resposta retornada pelo servidor jUDDI também é um parâmetro simples contendo o nome pesquisado. Por outro lado, a operação de consulta é completa quando uma requisição carrega vários parâmetros com o intuito de obter todas as informações associadas ao WSDL específico no servidor jUDDI.

5.3.2 Configuração do Ambiente

Um dos principais objetivos deste estudo é a avaliação do desempenho de um sistema de serviços que permite realizar a publicação e os mecanismos de descoberta usando um registro UDDI. Para isso, quatro máquinas foram utilizadas: a primeira representa um cliente que faz operações de consulta no registro UDDI. A segunda é um *gateway* que controla a largura de banda entre cliente, provedor e registro UDDI. A terceira máquina é um provedor de serviços, e por fim, a quarta máquina representa um registro UDDI. Para permitir o controle de largura de banda, é necessário que o registro UDDI esteja em uma rede separada em relação aos outros componentes, e ainda utilizar um *gateway* para interconectar essas redes. No *gateway* são utilizados duas interfaces de rede (duas placas de rede *gigabit*) e um *firewall* que permite comunicação entre essas redes. O ambiente de testes construído é ilustrado na Figura 5.8. As máquinas são homogêneas, ou seja, possuem a mesma configuração de *hardware* como pode ser observado na Tabela 5.7.

Tabela 5.7: Configuração de *Hardware*

Configuração de Hardware	
Sistema Operacional	Ubuntu Linux 8.04
CPU	Intel Core 2 Quad - 2.4 GHz
Memória	2GB
Disco rígido	250GB
Switch	3Com Gigabit Ethernet
Interface de rede	Realtek Gigabit Ethernet

As seguintes tecnologias foram utilizadas durante o desenvolvimento dos experimentos relacionados com operações no registro UDDI:

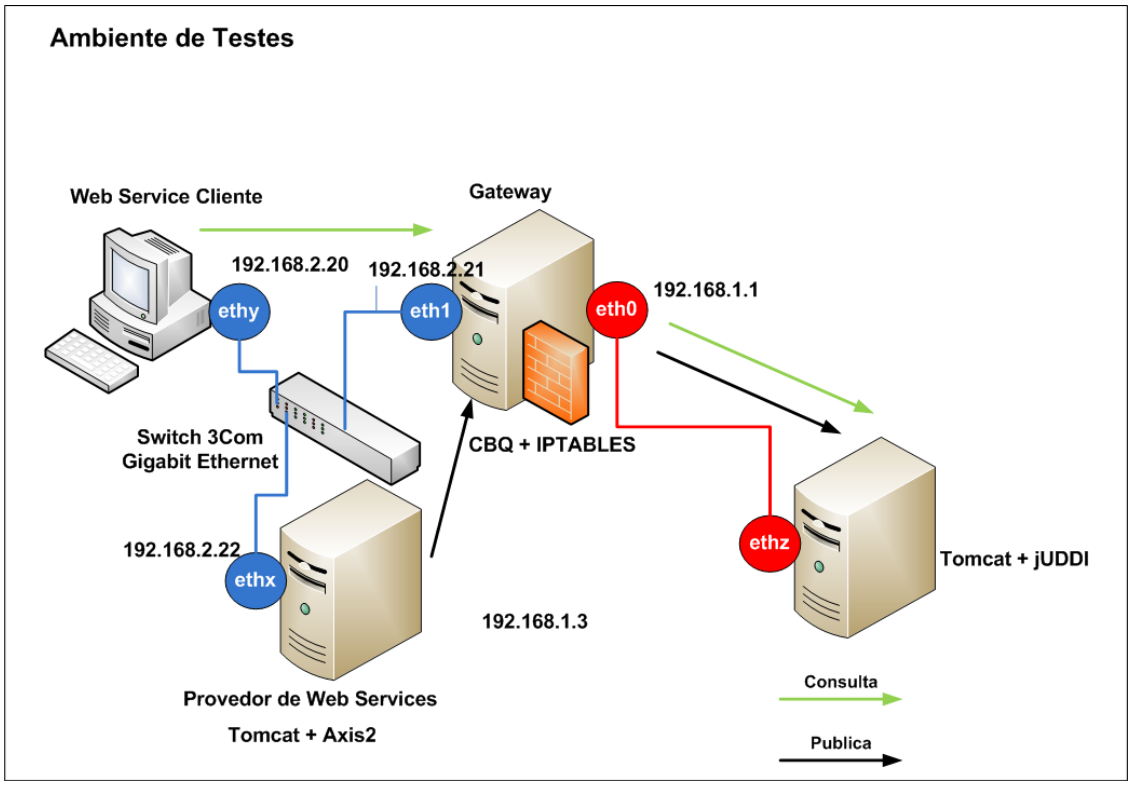

Figura 5.8: Ambiente de testes para experimentos com registro jUDDI

- **WS-UDDIPerf**: Ferramenta implementada para simular o cliente e o provedor de serviço. Além disso, WS-UDDIPerf coleta todos os dados necessários para realizar análise de resultados.
- **Apache jUDDI**: Implementação Java para especificações UDDI.
- **Apache Tomcat**: Container Web utilizado para implantar (*deploy*) os Web services em um provedor de serviços.
- **Iptables**: É utilizado para permitir que duas redes se comuniquem entre si de forma transparente através do NAT (*Network Address Translation*)
- **CBQ / Iproute**: É responsável pelo controle de largura de banda entre cliente e o registro UDDI, e entre provedor de serviços e registro UDDI. CBQ (Floyd, 1995) é um algoritmo para gerenciamento de tráfego desenvolvido pelo grupo de pesquisas de redes de computação da LBNL (*Lawrence Berkley National Laboratory*) (Floyd et al., 2009).

A implementação dos experimentos foi realizada utilizando aplicações desenvolvidas na linguagem Java para possibilitar a execução de operações de publicação e consulta em um ambiente *multithread*, ou seja, simular múltiplas requisições no registro UDDI.

5.3.3 Planejamento de Experimentos

Os experimentos foram feitos variando três fatores com o intuito de verificar a influência da quantidade de *threads* e da comunicação na rede para operações de consulta e inserção de informações no registro UDDI. A parametrização desses fatores é definida na Tabela 5.8.

Outro ponto chave a ser enfatizado é em relação aos parâmetros fixos usados nesses experimentos. Nesse caso, a quantidade de iterações foi definida como 20 requisições por *thread*. Sendo assim, para cada *thread*, as requisições são enviadas uma após a outra na sequência até a *thread* completar a tarefa de executar a quantidade máxima de requisições. Além disso, as *threads* competem entre si por recursos computacionais por serem executadas de forma concorrente.

Tabela 5.8: Parametrização

Parametrização	
Operação	Consulta; Publicação
Número de *threads*	10; 15; 20
Largura de Banda	100Kbps; 300Kbps; 500Kbps; 1Mbps

A análise de resultados foi baseada no tempo de resposta de cada *thread*. Para cada *thread*, é calculado o tempo médio de resposta de cada iteração para cada requisição concluída. Além disso, a variável de resposta é obtida calculando-se a média do tempo de resposta obtido para cada *thread*, obtendo-se também o desvio padrão, a variância, tempos de resposta mínimo e máximo, e o intervalo de confiança, este último calculado com 95% de nível de confiança. Para os resultados, 30 execuções foram realizadas para cada experimento. Isso possibilita maior confiabilidade na análise de resultados.

5.3.4 Análise de Resultados

Uma das tarefas mais importantes da análise de experimentos é verificar a influência dos fatores envolvidos. Uma das influências é a da rede durante a execução das operações de pesquisa e publicação, as quais podem ser visualizadas nos gráficos da Figura 5.9. Sendo assim, pode-se notar que a rede é um dos fatores que mais causam impacto no tempo de resposta. Para uma rede de 100Kbps, o tempo de resposta é maior que o tempo de resposta da rede de 1Mbps para operações de consulta no registro UDDI. O número de *threads* é também um fator que influencia o tempo médio de resposta, assim como o quanto a quantidade de requisições concorrentes aumenta, maior é a sobrecarga imposta no registro UDDI. Esse fator caracteriza um grande número de consultas/publicações no registro UDDI e a concorrência por recursos computacionais, tais como rede ou processamento. Os gráficos utilizados para ilustrar os resultados desses experimentos foram representados de forma diferente do conjunto de experimentos relacionados com *WS-attachments*. Foram representados usando gráfico de barras para identificar a média do tempo de resposta incluindo o intervalo de confiança.

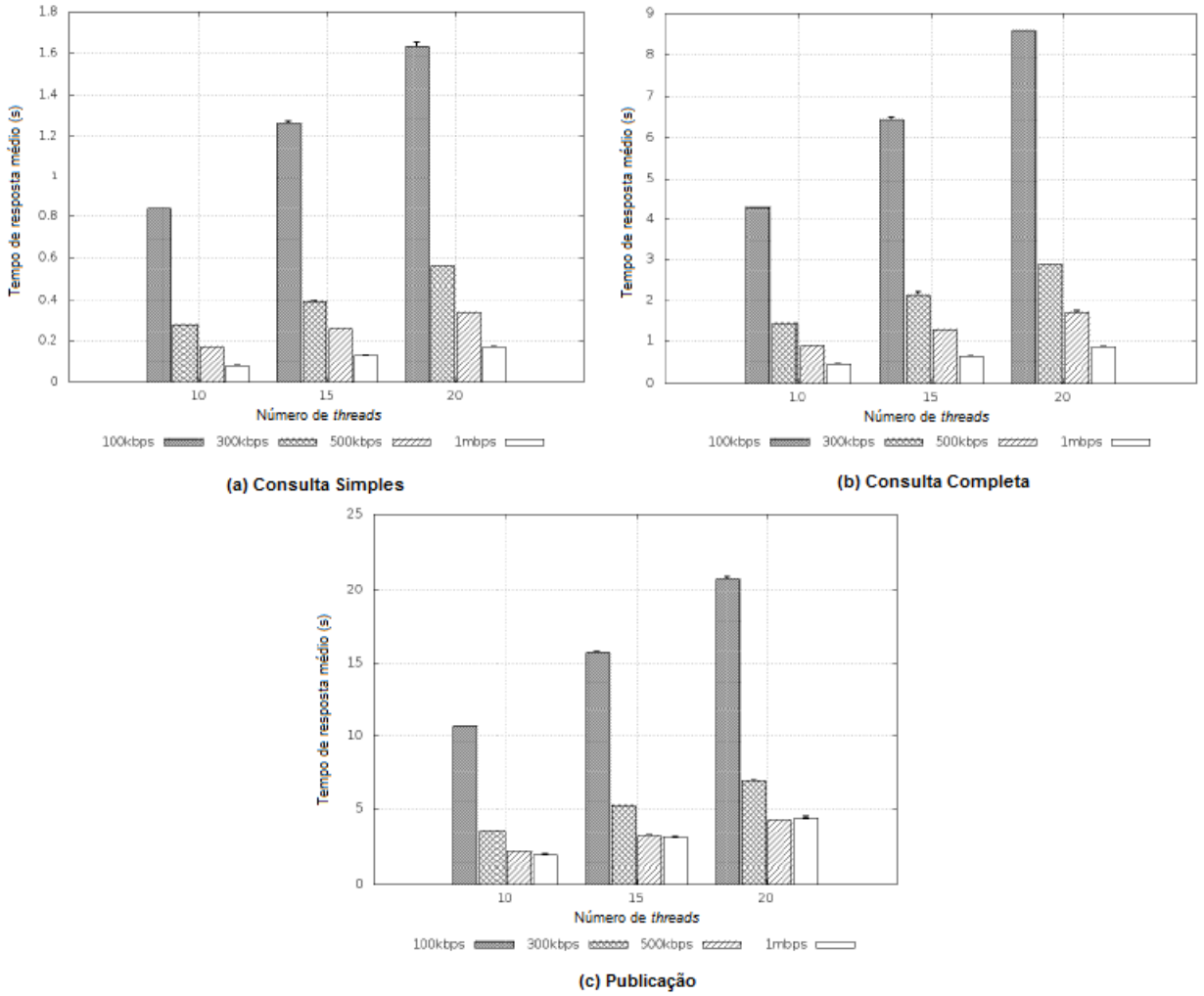

Figura 5.9: Influência da rede em relação a cada uma das operações

A influência das operações no tempo médio de resposta pode ser melhor visualizada nos gráficos da Figura 5.10, os quais comparam as operações de pesquisa e publicação em certas redes para um conjunto específico de *threads*. Para a operação de consulta, o que difere uma simples consulta de uma consulta completa é a quantidade de informações retornada para o cliente do Web service. Se por um lado, a pesquisa simples apenas retorna informações básicas sobre o serviço em participar (por exemplo, em um serviço para compras de passagens, a operação retorna apenas o nome do provedor de serviço que dispõe de passagens mais baratas para o destino solicitado), por outro lado, a consulta completa retorna todas as informações sobre o provedor de serviços (nome do provedor, *URL*, detalhes do serviço disponibilizado pelo provedor de serviços).

Em relação à operação de publicação, a diferença de desempenho em relação aos testes realizados com operações de consulta acontece devido à existência de uma tarefa adicional para realizar o processamento da interface WSDL, e assim, obter os dados armazenados no registro UDDI. Isso pode contribuir na degradação do tempo de resposta médio percebido pelo cliente do Web service. Para os experimentos realizados com operações em UDDI, não é analisada a influência do tamanho do documento WSDL, muito menos sua complexidade. Da mesma forma, para garantir a consis-

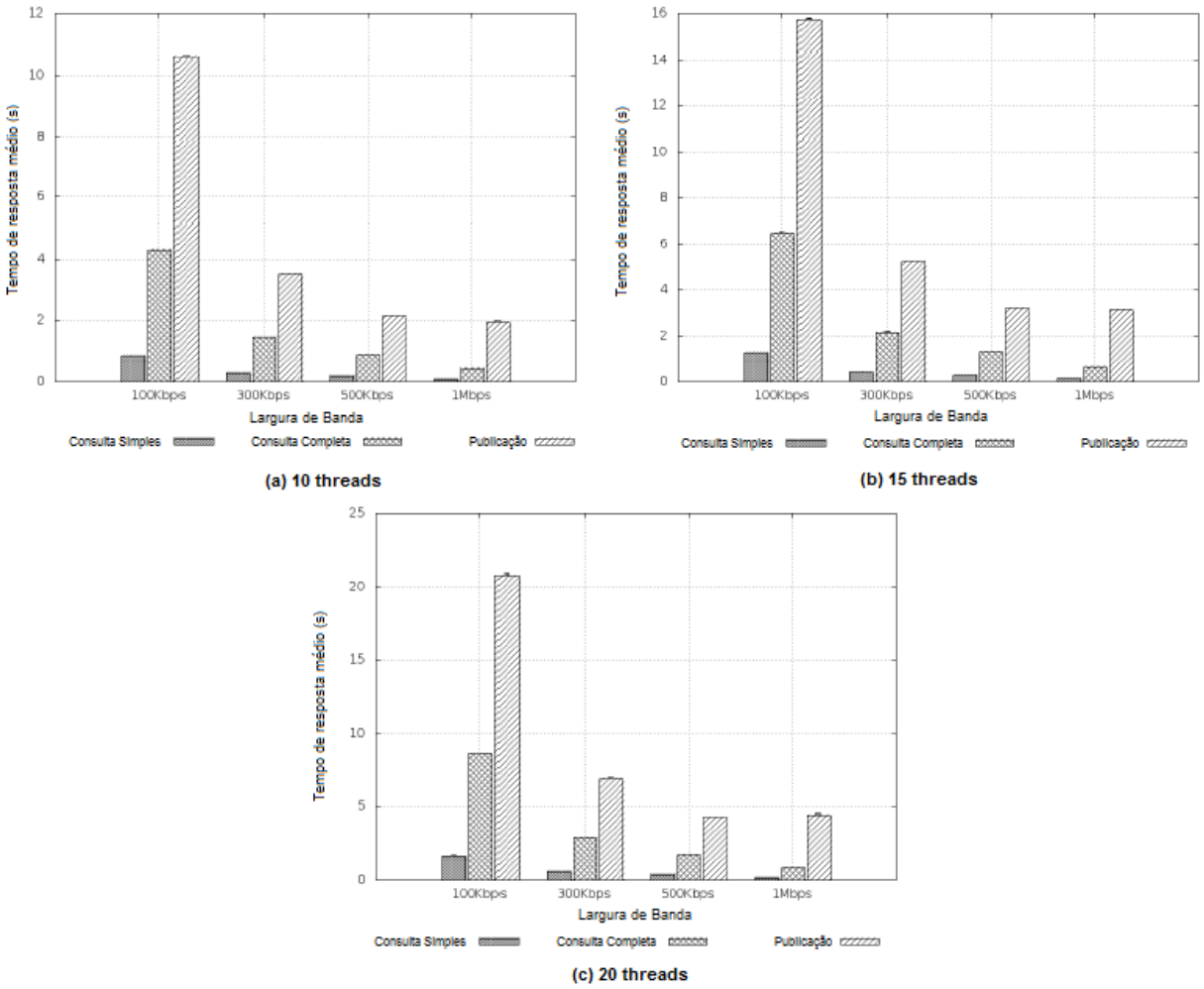

Figura 5.10: Influência das operações em relação ao tipo de rede

tência dos experimentos realizados com operações de consulta, apenas um serviço foi registrado no banco de dados do registro UDDI. Tal consideração é necessária pois a quantidade de dados armazenados no banco de dados pode afetar as operações em termos de desempenho, e assim, uma análise mais detalhada sobre isso deve ser realizada. Essa análise não foi considerada como parte do escopo neste trabalho.

5.4 Validação inicial da arquitetura WSARCH

Após a implementação de um protótipo para a arquitetura WSARCH, alguns projetos experimentais foram realizados para validar as funcionalidades da arquitetura. Tais experimentos foram definidos sob dois enfoques (Estrella, 2010). O primeiro verifica se a arquitetura está funcionando da forma projetada, considerando uma análise comportamental da arquitetura e o impacto da inclusão dos componentes *broker* e *logserver* nos resultados da execução das requisições. O segundo enfoque mostra como a arquitetura proposta atinge seu objetivo de ser uma ferramenta de avaliação de desempenho de *Web services* com qualidade de serviço. É importante ressaltar que

os experimentos realizados nesta etapa foi realizado em conjunto com Júlio Estrella, o autor da proposta da arquitetura WSARCH. Os resultados deste estudo também foram apresentados em sua tese (Estrella, 2010). Esta seção descreve como foram realizados tais experimentos e uma análise dos resultados obtidos.

5.4.1 Configuração do ambiente

Tabela 5.9: Elementos de *Hardware* (Estrella, 2010)

Componente	Quantidade	Tipo	Descrição
Provedor de serviços	7 unidades	Intel QuadCore Q6600 2GB de RAM, HD de 120GB, 7200 RPM	Hospedar os serviços implementados
UDDI	1 unidade	Intel QuadCore Q6600 2GB de RAM, HD de 120GB, 7200 RPM	Repositório de informações de serviços
Broker	1 unidade	Intel QuadCore Q9400 8GB de RAM, HD de 500GB, 7200 RPM	Roteamento / Escalonamento de mensagens com base em algoritmos de seleção de serviços e índice de carga dos provedores de serviços
Clientes	8 unidades	4 unidades (Intel QuadCore Q9400 4GB de RAM, HD de 750GB, 7200 RPM) e 4 unidades (Intel QuadCore Q6600 4GB de RAM, HD de 320GB, 7200 RPM)	Clientes que acessam os serviços hospedados nos provedores de serviços. São três tipos (classes de clientes) cada uma das quais contendo X requisições.
Servidor de logs	1 unidade	Intel QuadCore Q9400 4GB de RAM, HD de 500GB, 7200 RPM	Armazenar informações em tempo de execução dos componentes da arquitetura (clientes, provedores, Broker, UDDI)

Para uma correta avaliação de desempenho de um sistema computacional é imprescindível detalhar todos os elementos de hardware e software utilizados e verificar como tais elementos afetam de forma positiva ou negativa o desempenho da arquitetura (Estrella, 2010). A Tabela 5.9 apresenta a infra-estrutura computacional utilizada, a qual é composta por um ambiente de testes contendo sete unidades de processamento que funcionam como provedores de serviços, uma unidade de processamento que atua como um registro de informações de serviços, oito unidades atuando como clientes e uma unidade funcionando como o *Broker* da arquitetura. Há também uma unidade de processamento que corresponde ao servidor de Log (LogServer) que armazena informações dos experimentos (em tempo de execução) dos componentes da arquitetura.

Além dos componentes de *hardware* descritos na Tabela 5.9, a Tabela 5.10 representa os elementos de *software* que foram utilizados no desenvolvimento da arquitetura. Na Tabela 5.10, são

Tabela 5.10: Elementos de *Software* (Estrella, 2010)

Nome	Descrição	Utilização
Apache Axis2 #	Motor de processamento de mensagens SOAP	Clientes, Provedores, Broker, UDDI
Apache Tomcat #	Container de Servlets	Provedores
JUDDI #	Implementação Java da especificação UDDI da OASIS	UDDI
Java Virtual Machine	Máquina Virtual Java	Clientes, Provedores, Broker, UDDI
Ganglia	Sistema de monitoração distribuído e escalável	UDDI, Broker, Provedores
MySQL Server	Sistema Gerenciador de Bancos de Dados (SGBD), utilizado pelo jUDDI e também pelo Broker da WSARCH	UDDI, LogServer
WSARCH-Broker *	Broker da arquitetura WSARCH	Broker
WSARCH-LoadInfoColector *	Aplicação desenvolvida para padronizar as informações obtidas do daemon Gmetad do Ganglia	UDDI, Broker, Provedores
WSARCH-GeneralLoadIndex *	Aplicação desenvolvida para compor um índice de carga para ser utilizado pelo Broker na seleção de serviços	UDDI, Broker
WSARCH-Client *	Aplicações clientes desenvolvidas para testes da arquitetura	Clientes
CPU-BoundApp *	Aplicação servidora orientada ao consumo de CPU	Provedores

enfatizados elementos de *software* utilizados como ferramentas de suporte e ferramentas desenvolvidas para que a arquitetura fosse concebida em sua totalidade. Os elementos assinalados com um (*) foram desenvolvidos para a WSARCH enquanto os assinalados com um (#) foram adaptados.

5.4.2 Planejamento de experimentos

Uma das etapas mais importantes para uma avaliação de desempenho bem sucedida é a etapa de planejamento de experimentos. No entanto, esta não é uma tarefa trivial, considerando-se a variedade de fatores que podem influenciar um sistema computacional grande e complexo. Outro ponto importante a ser considerado é o fato de tratar-se de uma avaliação de desempenho em um sistema computacional real envolvendo a interação entre vários componentes em um sistema distribuído, exigindo certos cuidados para a preparação dos experimentos e obter o máximo de informações com mínima quantidade de elementos. Sendo assim, a escolha dos fatores a serem considerados nos experimentos foi definida de forma a validar o funcionamento dos componentes do protótipo da arquitetura, e em seguida, verificar a influência de alguns dos fatores mais pertinentes.

Considerando tais observações, seis aspectos relacionados ao protótipo da WSARCH foram considerados durante os experimentos: validação da arquitetura, sobrecarga gerada pela utilização do *LogServer*, sobrecarga do Broker, carga de trabalho, o tempo de processamento nos provedores de serviços e os algoritmos de diferenciação de serviços (Estrella, 2010). Para cada um desses aspectos foram definidos de forma isolada quais seriam os fatores a serem variados, a maioria considerando planejamento fatorial simples, visto que não houve preocupação nesta etapa de avaliação de desempenho em determinar a interação entre os fatores identificados. Os fatores e seus respectivos níveis são sumarizados na Tabela 5.11.

Tabela 5.11: Fatores e níveis relativos aos experimentos (Estrella, 2010)

Fatores	Níveis
Log	On/Off
Carga de trabalho - Tempo entre requisições	0s - 3s
Carga de trabalho - Número de processos	1 - 6
Algoritmo de diferenciação de serviços	Classificação RSV (4 Gold/3 Bronze) RSV (5 Gold/2 Bronze)
Tempo de execução do serviço	0,001s - 6s

A Tabela 5.12 apresenta a definição dos principais fatores utilizados no estudo de avaliação de desempenho dos algoritmos de escalonamento de mensagens implementados para a validação do protótipo da arquitetura. Tais algoritmos são descritos em detalhes na Seção 4.2.4. Com isso, três experimentos foram realizados para esse estudo, utilizando o algoritmo de classificação e o de reserva de recursos (RSV). Este último, por sua vez, foi realizado determinando a quantidade de recursos reservados para clientes *gold* e *bronze* em dois experimentos. Vale ressaltar que os resultados obtidos nesse estudo geraram uma publicação internacional em *Proceedings of the 2010 6th World Congress on Services* (Estrella et al., 2010).

Para todos os experimentos foram realizadas 5 execuções utilizadas para determinar a média, o desvio padrão e o intervalo de confiança de 95% de acordo com a tabela *T-Student*. O número de execuções foi definido considerando-se a necessidade de obter diferença significativa dentre os intervalos de confiança dos experimentos para comparação dos resultados.

5.4.3 Análise dos resultados

Esta seção apresenta um resumo dos resultados obtidos com a execução dos primeiros experimentos relativos ao protótipo da arquitetura WSARCH. Tais experimentos foram conduzidos considerando os seis aspectos citados anteriormente na Seção 5.4.2. É importante ressaltar que essa seção tem como objetivo apresentar de forma sucinta as conclusões para cada um desses aspectos. Uma visão mais concreta e mais detalhada sobre cada experimento pode ser conferida em (Estrella, 2010). Os resultados relacionados a cada aspecto serão abordados de forma isolada a seguir.

Tabela 5.12: Experimentos para comparação de algoritmos de roteamento (Estrella et al., 2010)

	Exp001	**Exp002**	**Exp003**
Número de clientes	8	8	8
Classes do cliente	50% *bronze* 50% *gold*	50% *bronze* 50% *gold*	50% *bronze* 50% *gold*
Número de processos	6	6	6
Número de requisições por processo	200	200	200
Parâmetro de entrada	200 (3s)	200 (3s)	200 (3s)
Tipo de serviço	CPU-Bound	CPU-Bound	CPU-Bound
Tempo entre requisições	0-1segs	0-1segs	0-1segs
Algoritmo (Broker)	Classificação	RSV(4G,3B)	RSV(5G,2B)

1. Validação da arquitetura/Análise comportamental

Nesta etapa, um único experimento foi realizado de modo a verificar se as informações estavam sendo propagadas de forma correta e se a integração entre os componentes da arquitetura estava consistente de modo a possibilitar a troca de mensagens SOAP corretamente. Não houve preocupação em avaliar o desempenho do sistema implementado, nem há propósitos comparativos. Dentre os principais fatores considerados para esse experimento estão a escolha do algoritmo de escalonamento de mensagens (algoritmo de classificação), para verificar como tal roteamento se comporta nos provedores de serviço; e o processamento do serviço no provedor, que possui uma carga considerável, permitindo que o serviço leve em média de 6 segundos para ser processado no provedor, garantindo assim que esse provedor fique ocupado por certo período de tempo. Além disso, este experimento considerou metade dos clientes envolvidos como classe *gold* e outra metade como classe *bronze*.

Os resultados obtidos foram satisfatórios, visto que a distribuição de requisições para cada provedor foi balanceada tanto para requisições de clientes *gold*, quanto para de clientes *bronze*, o que foi esperado como comportamento para o Algoritmo de Classificação nas condições proporcionadas pela parametrização. O Algoritmo de Classificação não considera nenhuma reserva de recursos e apenas distribui as requisições de acordo com o índice de desempenho. Dois gráficos de distribuição de requisições - um para cliente *gold* e um para *bronze* - são apresentados na Figura 5.11 ilustrando o comportamento dos provedores durante a execução do experimento com o Algoritmo de Classificação.

2. Sobrecarga no Broker

Esse estudo verifica o impacto imposto na inclusão de um *Broker* na arquitetura WSARCH. Essa verificação é uma etapa importante da avaliação da arquitetura devido ao Broker ser um componente centralizado realizando interações com todos os outros componentes da arquitetura, e assim, existem possibilidades de se tornar o gargalo durante o processamento de

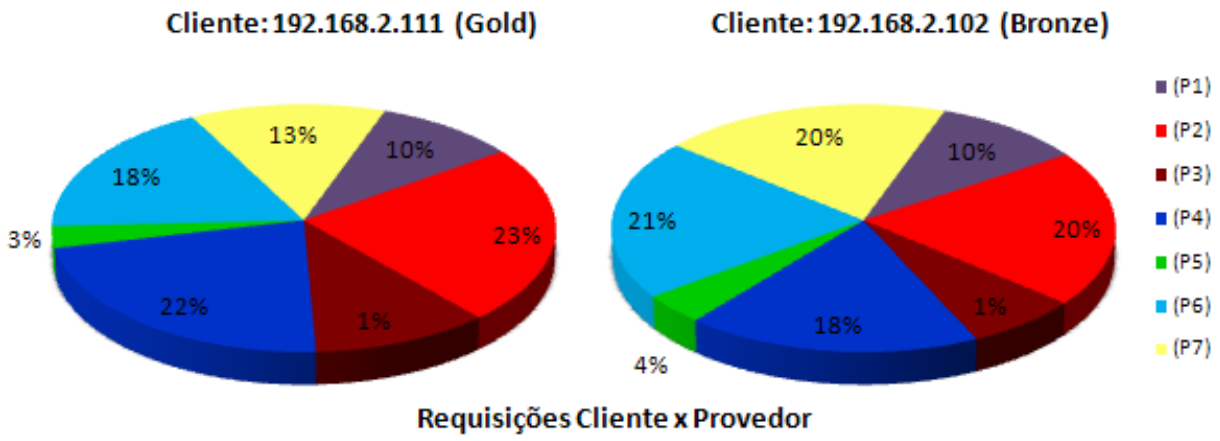

Figura 5.11: Distribuição das requisições - Cliente x Provedor (validação da arquitetura) (Estrella, 2010)

uma requisição na arquitetura. Antes de realizar os experimentos, foi necessário identificar quais os fatores que influenciam no fluxo das requisições no Broker. Sendo assim, o aumento da quantidade de processos por cliente e a diminuição do tempo entre o envio das requisições fizeram parte do procedimento para aumentar a quantidade de requisições enviadas ao Broker em menos tempo. Além disso, do lado do provedor, o tempo de processamento do serviço também influencia no fluxo das requisições no Broker, visto que as requisições são síncronas, ou seja, o processo cliente após lançar uma requisição ao Broker, espera esta ser processada para então enviar a próxima requisição. Com isso, três experimentos foram definidos considerando o fluxo de informações no Broker:

- **ExpA**: Processamento de serviço lento (média de 6s); 1 processo por cliente; tempo entre requisições de 3 segundos
- **ExpB**: Processamento de serviço rápido (média de 0,001s); 1 processo por cliente; tempo entre requisições de 3 segundos
- **ExpC**: Processamento de serviço rápido (média de 0,001s); 6 processo por cliente; tempo entre requisições de menos de 1 segundo

A Tabela 5.13 apresenta informações sobre o comportamento do Broker nesses três experimentos. Seu comportamento é representado através da média e desvio padrão dos índices de carga (CPU e memória) e desempenho durante a execução de 200 requisições. Vale ressaltar que quanto mais próximo de zero estiver o índice melhor. O cálculo da média e desvio padrão dos índices obtidos foi feito desconsiderando a faixa inicial de requisições (*warmup* inicial do experimento). Nota-se que existe um aumento pouco significativo na média dos índices de desempenho ao aumentar o carga de trabalho no Broker. Também pode ser verificado que baseado em tais resultados,

o Broker da arquitetura não sofre sobrecarga em termos de CPU e memória, o que sinaliza um índice de desempenho próximo de zero.

Tabela 5.13: Média e Desvio Padrão do comportamento do Broker (Estrella, 2010)

	ExpA		ExpB		ExpC	
	Média	**D.Padrão**	**Média**	**D.Padrão**	**Média**	**D.Padrão**
Índice CPU	0,0185	0,0111	0,02531	0,01229	0,03490	0,02962
Índice Memória	0,0885	0,0021	0,14740	0,01980	0,16572	0,02610
Índice Desempenho	0,0535	0,0057	0,08635	0,01249	0,10031	0,02246

3. Sobrecarga devido ao LogServer

Algumas informações de desempenho relacionadas aos componentes da arquitetura WSARCH, incluindo tempo de processamento nos provedores, tempo de resposta das requisições dos clientes, tempo de busca, tempo de seleção de serviços, tempo de *parser*, dentre outros, são registradas em um servidor de Log (Estrella, 2010). É importante salientar que o *LogServer* não é um mecanismo obrigatório para o funcionamento da arquitetura. No entanto, para possibilitar que estudos com informações detalhadas de avaliação de desempenho da arquitetura possam ser realizados, a inclusão de um servidor de *Logs* é essencial de modo a registrar informações como tempo, processamento, vazão, entre outros, sobre cada componente da arquitetura e suas interações. Entretanto, pode existir um custo adicional no tempo de resposta gerado pela inclusão do *LogServer* na arquitetura, aspecto também investigado durante os estudos de validação da arquitetura WSARCH.

Tabela 5.14: Média e Desvio Padrão do comportamento do Broker em relação à inclusão do LogServer (Estrella, 2010)

	LogServer Ativado		LogServer Desativado		% de Aumento
	Média	**D.Padrão**	**Média**	**D.Padrão**	
192.168.2.102 (Gold)	0,5827	0,0152	0,4769	0,00142	22,19
192.168.2.103 (Gold)	0,5818	0,159	0,4733	0,0130	22,92
192.168.2.104 (Gold)	0,5809	0,0112	0,4803	0,0104	20,93
192.168.2.105 (Gold)	0,6470	0,0135	0,5373	0,0050	20,41
192.168.2.108 (Bronze)	0,7035	0,0158	0,5963	0,0058	17,98
192.168.2.112 (Bronze)	0,7179	0,0012	0,6156	0,0074	16,60
192.168.2.114 (Bronze)	0,6712	0,0142	0,5652	0,0129	18,74
192.168.2.115 (Bronze)	0,6922	0,0077	0,5940	0,0077	16,53

A Tabela 5.14 sumariza a porcentagem de aumento na sobrecarga da WSARCH, e destaca que a influência máxima do LogServer no tempo de sobrecarga na arquitetura não ultrapassa os 23%. A sobrecarga do LogServer é gerada em função dos vários acessos para escrita de informações em uma base de dados relacional que armazena todos os dados dos experimentos. Vale ressaltar que o tempo de processamento do serviço nesse estudo é desprezado, sendo considerado apenas a sobrecarga imposta pelas funcionalidades da arquitetura. Um outro estudo foi realizado apenas

considerando o processamento de serviço, mas foi verificado que não existe diferença estatística com *LogServer* ativado ou desativado durante essa etapa da arquitetura (Estrella, 2010).

4. Influência da Carga de Trabalho

Para verificar a influência da variação da carga de trabalho no comportamento da arquitetura em termos de desempenho, dois experimentos foram realizados de modo semelhante aos experimentos realizados para analisar a sobrecarga do Broker. O primeiro experimento (*EXP_A*) consiste em um único processo ativo por cliente com tempo entre requisições seguindo uma distribuição uniforme entre 0 e 3 segundos, enquanto o segundo (*EXP_B*) utiliza seis processos concorrentes por cliente com tempo entre requisições de apenas no máximo 1 segundo.

A análise da influência da carga de trabalho foi feita verificando a sobrecarga imposta à arquitetura e o tempo médio de resposta na WSARCH, como ilustrado na Figura 5.12. Nota-se que o aumento da carga de trabalho ocasionou um impacto negativo considerável no tempo de resposta da requisição, principalmente considerando a sobrecarga imposta à arquitetura.

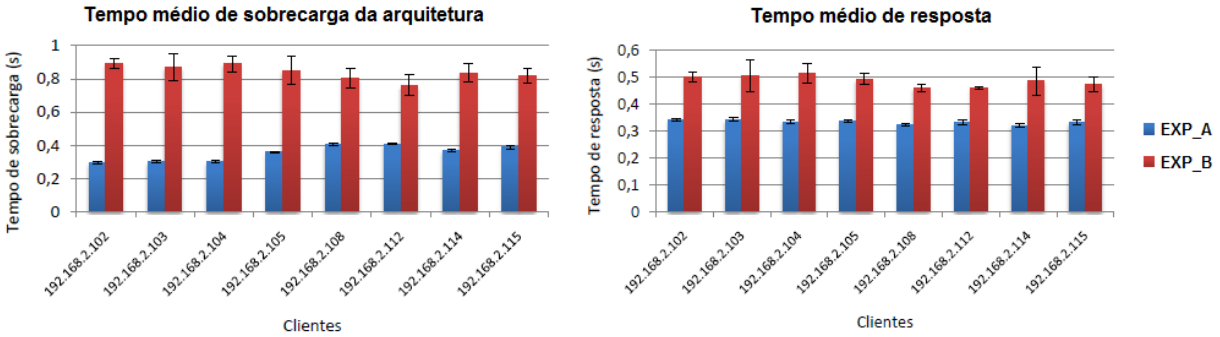

Figura 5.12: Tempo médio de resposta para os experimentos da influência da carga de trabalho (Estrella, 2010)

5. Influência do Processamento do Serviço nos Provedores

Na etapa seguinte foi realizada uma análise da variação do processamento de serviço por parte do provedor para verificar se tem impacto no tempo gasto pelas funcionalidades da arquitetura WSARCH. Para isso, dois experimentos foram executados, ambos consistindo de uma aplicação *CPU-Bound* que recebe um parâmetro que sinaliza uma quantificação para a utilização do CPU. Isso determina se a aplicação deve demorar ou não a ser executada pelo provedor de serviços. Esse parâmetro de entrada é enviado pelo cliente como parte da mensagem SOAP, sendo desencapsulada no provedor, e, enfim, utilizado pelo serviço que será executado. No primeiro experimento (*Exp_A*), o tempo gasto para a execução do serviço no provedor foi em média de 0,001 segundos, enquanto no segundo experimento *Exp_B*, a carga gerada pelo serviço foi maior, levando em torno de 3 segundos para a aplicação ser executada no provedor.

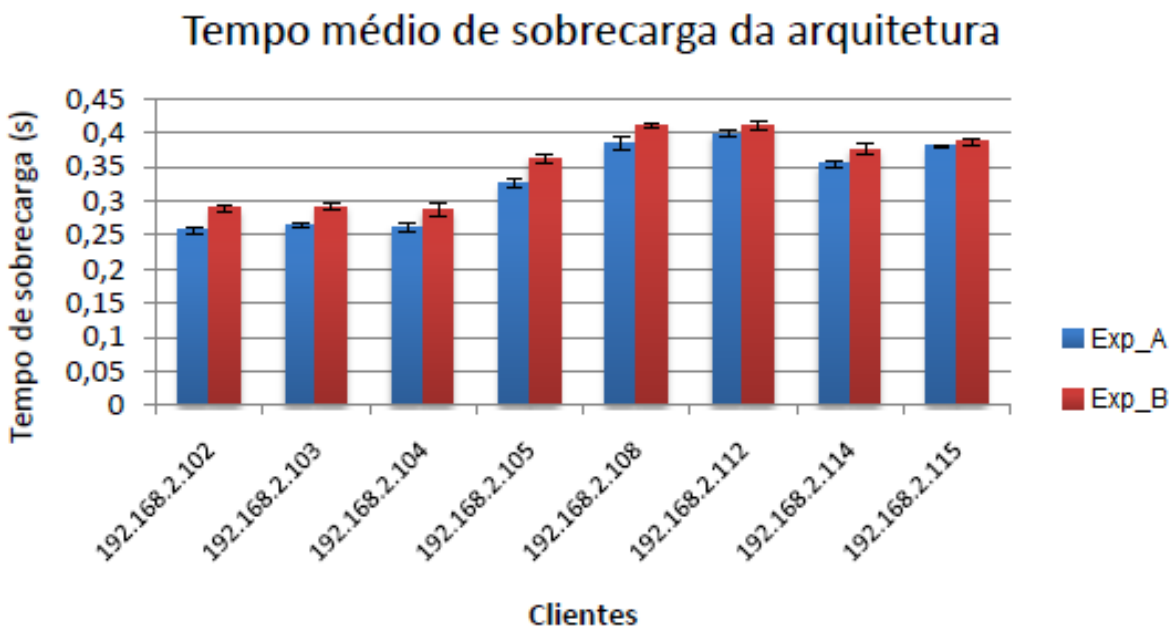

Figura 5.13: Sobrecarga da arquitetura com variação do tempo de processamento nos provedores de serviços (Estrella, 2010)

Os resultados obtidos durante a execução dos experimentos indicam que houve pouca influência imposta nas funcionalidades da arquitetura com o aumento da carga de processamento no servidor, como pode ser observado na Figura 5.13. Isso acontece devido a capacidade do Broker da arquitetura ser maior em termos de recursos computacionais (CPU e Memória) presentes nele, conforme apresentado nos experimentos do aspecto sobre a sobrecarga do Broker. Em uma análise mais detalhada, verificou-se que com o aumento do tempo de processamento, a sobrecarga no Broker aumentou até menos de 12,8%, mostrando que a sobrecarga para execução dos serviços tem pouca influência no tempo de processamento da requisição.

6. Impacto dos algoritmos de roteamento

Com seus resultados publicados em uma conferência internacional denominada *SERVICES '2010* (*Sixth World Congress on Services*), a análise de diferentes algoritmos de roteamento/escalonamento de mensagens foi um dos principais testes conduzidos para verificar o correto funcionamento da arquitetura com foco em estudos de análise de desempenho, visto que teve como principal objeto de estudo o roteamento de mensagens através do Broker. Os dois algoritmos implementados para este fim são relativamente simples quanto à diferenciação de serviços, mas cumprem seu papel em termos de escalonamento de mensagens de acordo com os critérios pré-estabelecidos, como definido na Seção 4.2.4. Devido às características modulares presentes na arquitetura WSARCH, é possível outros tipos de algoritmos de escalonamento considerando, por exemplo, o uso de técnicas de inteligência artificial, Web semântica, entre outros.

A primeira bateria de experimentos corresponde aos resultados obtidos com a utilização do algoritmo de Classificação considerando oito clientes nos quais metade pertencem à classe *Gold*

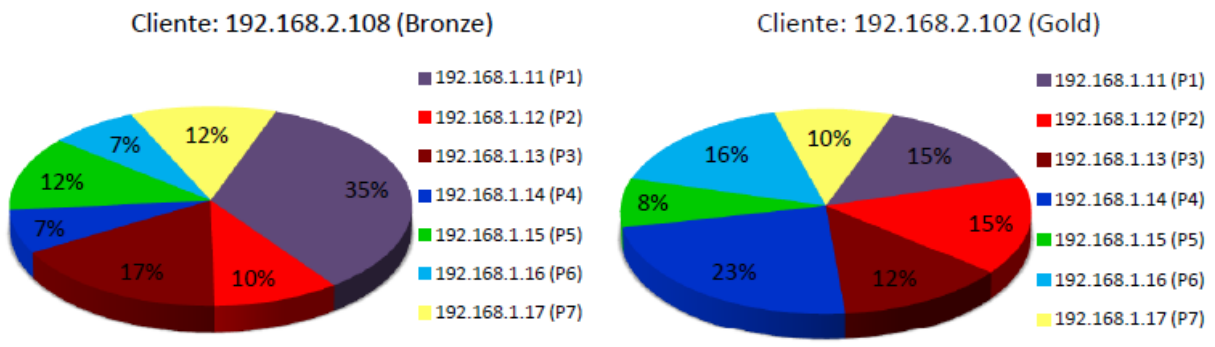

Figura 5.14: Distribuição de requisições para algoritmo de classificação (Estrella et al., 2010)

e o restante na classe *Bronze*. A distribuição das requisições para cada provedor é representada pela Figura 5.14. Nota-se que apenas uma amostra de cada classe é apresentada neste gráfico, visto que clientes de uma mesma classe se comportaram de modo semelhante entre eles. Na Figura 5.14, observa-se que existe uma concentração maior de requisições em apenas um dos provedores (P1) na amostra representada pela classe *Bronze* de clientes. Isso acontece devido a um comportamento esperado do algoritmo de classificação, onde clientes da classe *Bronze* selecionam o pior dos provedores da lista em termos de utilização de recursos, ou seja, o provedor *P1* teve a tendência de ser o mais utilizado da lista de provedores disponibilizado pelo UDDI. Como este era o provedor mais utilizado, a melhoria do índice de desempenho deste provedor seria menos provável em relação aos demais provedores, atraindo assim, mais requisições da classe *Bronze*. Por outro lado, a distribuição de requisições para clientes da classe *Gold* foi de forma mais uniforme. Também foi verificado que o tempo médio de resposta é maior em clientes da classe *Bronze* em relação aos da classe *Gold*, como ilustrado na Figura 5.15.

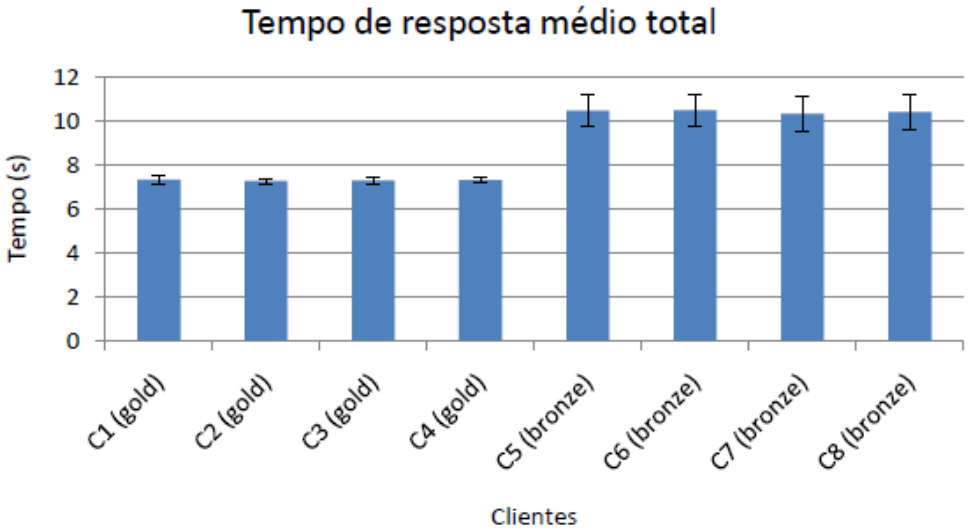

Figura 5.15: Tempo médio de resposta para algoritmo de classificação (Estrella et al., 2010)

A segunda rodada de experimentos é relacionada com dois experimentos que utilizam como base o Algoritmo de Reserva de Recursos (*RSV*), representados pelos experimentos *ExpB* e *ExpC* definidos anteriormente na Tabela 5.12. Tais experimentos foram diferenciadas pela variação do fator quantidade de recursos reservados por classe de cliente. Enquanto o experimento *ExpB* foram reservados apenas quatro dos sete provedores de serviço disponíveis para a classe *Gold*, o experimento *ExpC* reservou cinco provedores para a classe *Gold*, restando apenas dois para a classe *Bronze*.

As Figuras 5.16 e 5.17 representam gráficos de distribuição de requisições para os experimentos *ExpB* e *ExpC*, respectivamente, os quais respeitam os recursos disponibilizados para cada classe de clientes. Vale ressaltar que assim como a distribuição de requisições feita para o algoritmo de classificação, estes experimentos também consideram apenas uma única amostra para as classes *Gold* e as classes *Bronze*, respectivamente. Observa-se que a distribuição de requisições ao utilizar o algoritmo *RSV* é realizada de forma mais balanceada, o que mostra que ocorre um equilíbrio ao selecionar um serviço considerando o índice de desempenho. Enquanto um provedor está processando uma ou mais requisições, é natural que o índice de desempenho seja um pouco maior, e assim é menor a possibilidade dele ser escolhido pelo algoritmo de escalonamento.

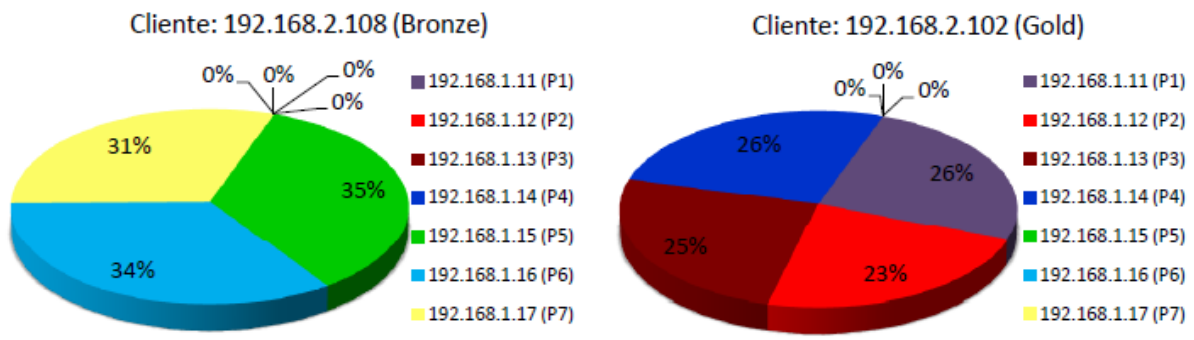

Figura 5.16: Distribuição de requisições para algoritmo RSV (4 Gold, 3 Bronze) (Estrella et al., 2010)

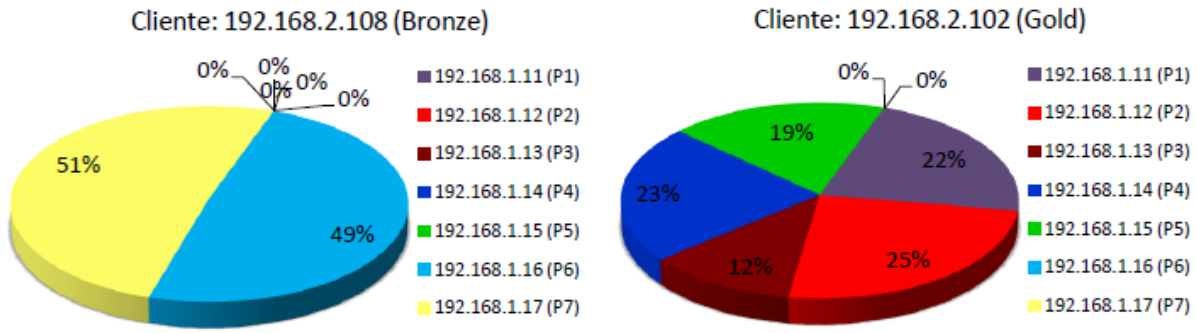

Figura 5.17: Distribuição de requisições para algoritmo RSV (5 Gold, 2 Bronze) (Estrella et al., 2010)

Em relação ao tempo de resposta, considerando a configuração definida para o experimento *ExpB*, com quatro recursos alocados para a classe *Gold* e três para a classe *Bronze*, não é possível afirmar estatisticamente que houve diferenciação de serviços, como pode-se notar na Figura 5.18. A pequena diferença de recursos alocados para cada classe não foi o suficiente para verificar a existência de alguma diferenciação de serviço entre as classes (Estrella, 2010). No entanto, essa diferença de serviços proporcionada pelo algoritmo de reserva de recursos começa a ser perceptível ao aumentar a quantidade de recursos disponíveis para clientes da classe *Gold*, como demonstrado na Figura 5.17. O gráfico da Figura 5.19 ilustra visualmente uma pequena diferença no tempo de resposta médio entre clientes das classes *Gold* e *Bronze*, mas comprovando que a diferenciação de serviços nesse caso foi realizada de forma efetiva.

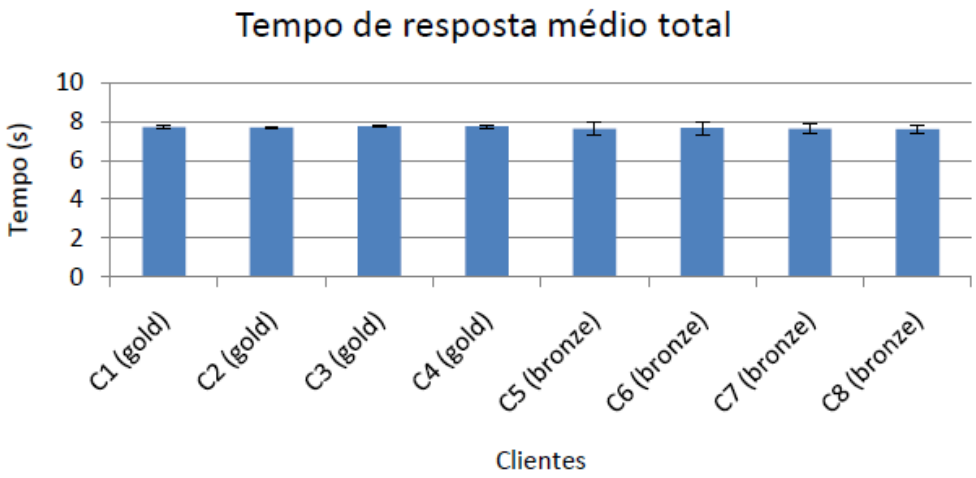

Figura 5.18: Tempo de resposta médio total para algoritmo RSV (4 Gold, 3 Bronze) (Estrella et al., 2010)

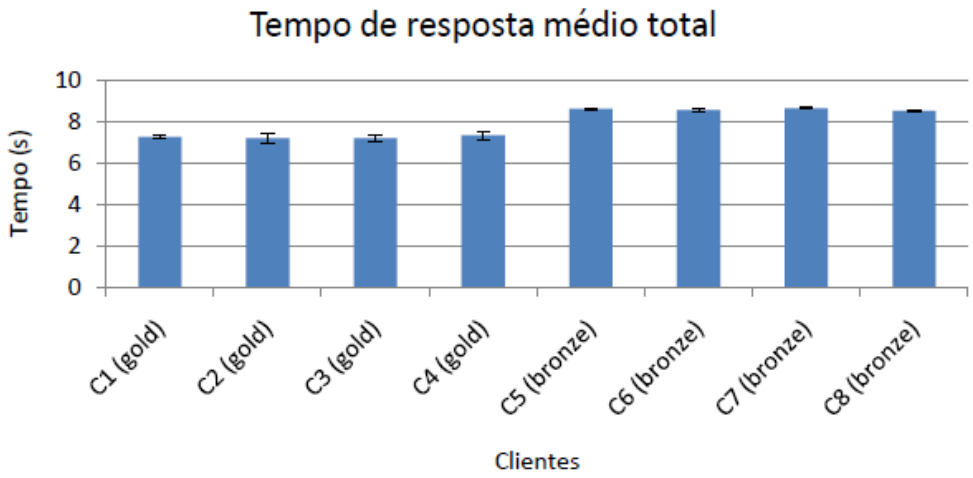

Figura 5.19: Tempo de resposta médio total para algoritmo RSV (5 Gold, 2 Bronze) (Estrella et al., 2010)

5.5 Estudo sobre o desempenho de diferentes categorias de aplicações

Nesta seção serão descritos os projetos experimentais realizados para um estudo de avaliação de desempenho considerando diferentes categorias de aplicações, os quais foram abordados na Seção 4.4. O estudo de desempenho com diferentes tipos de aplicações é uma etapa importante dentro do contexto de desenvolvimento da arquitetura WSARCH, visto que espera-se que a arquitetura possibilite a provisão de qualidade de serviço para quaisquer aplicações implantadas nos provedores de serviço, independente dos recursos computacionais utilizados. O principal objetivo deste estudo de avaliação de desempenho é verificar o comportamento de categorias distintas de aplicações diante da variação da carga de trabalho representada pela quantidade de requisições enviadas pelos clientes. Assim como outros estudos de desempenho apresentados neste capítulo, essa seção é composta pela descrição das fases de configuração de ambiente e planejamento de experimentos, e por fim a análise dos resultados obtidos.

5.5.1 Configuração do ambiente

Os experimentos realizados neste estudo de desempenho utiliza a mesma infra-estrutura computacional definida para os primeiros testes de validação do protótipo construído para a arquitetura WSARCH, a qual é especificada na Tabela 5.9. Além disso, novos elementos de *software* foram implementados, complementando os que foram especificados na Tabela 5.10. Tais elementos referem-se a cada aplicação implementada representada por uma categoria distinta de aplicação, e também a uma adaptação no broker para possibilitar a execução de aplicações que provêm suporte à *attachments*, utilizada para ilustrar o comportamento de aplicações *network-bound*. Outro ponto importante a ser comentado é em relação aos clientes, os quais foram adaptados individualmente para cada aplicação, de modo a enviar parâmetros específicos solicitados por cada tipo de aplicação. A Tabela 5.15 apresenta um resumo dos principais elementos de *softwares* utilizados para realizar os experimentos. Consideram-se elementos adaptados para atender solicitações de aplicações de diferentes categorias como o *broker* e o cliente, novas aplicações de categorias distintas, e a mesma aplicação *CPU-Bound* utilizada para a validação inicial da arquitetura.

5.5.2 Planejamento de experimentos

Para a realização dos projetos experimentais definidos para esse estudo, foram consideradas quatro aplicações implementadas com base nas categorias de aplicações descritas na Seção 4.4. Três delas se tratam de categorias específicas de aplicação de forma isolada: *CPU-Bound*, *IO-Bound* e *Network-Bound*. Tais aplicações são descritas detalhadamente na Seção 4.4.1.

Tabela 5.15: Principais elementos de *Software* utilizados nos testes de desempenho com diferentes tipos de aplicações

Nome	Descrição	Utilização
WSARCH-Broker	Broker da arquitetura WSARCH	Broker
WSARCH-Client	Aplicações clientes desenvolvidas para testes da arquitetura	Clientes
CPU-BoundApp	Aplicação servidora orientada ao consumo de CPU	Provedores
IO-BoundApp	Aplicação servidora orientada à escrita em disco	Provedores
Network-BoundApp	Aplicação servidora orientada ao consumo de rede	Provedores
Mixed-App	Aplicação servidora orientada ao consumo de CPU, rede e escrita em disco	Provedores

Em paralelo com os tipos de aplicação a serem executados nos testes de desempenho, também são verificadas a influência da carga de trabalho e a variação de parâmetros que definem o processamento do serviço na arquitetura. Sendo assim, levando em conta tais considerações, os níveis e fatores para a definição dos experimentos foram definidos, sendo especificados de acordo com a Tabela 5.16. Os demais fatores não presentes na Tabela 5.16 foram considerados como parâmetros fixos, alguns definidos com valores padrão, como pode ser visualizado na Tabela 5.17.

Tabela 5.16: Fatores e níveis para experimentos de influência das categorias de aplicações

	Aplicações	Parâmetros	Nº de processos
Exp1	CPU-Bound	150	1 processo
Exp2	CPU-Bound	250	1 processo
Exp3	CPU-Bound	150	4 processos
Exp4	CPU-Bound	250	4 processos
Exp5	IO-Bound	100KB	1 processo
Exp6	IO-Bound	1MB	1 processo
Exp7	IO-Bound	100KB	4 processos
Exp8	IO-Bound	1MB	4 processos
Exp9	Network-Bound	100KB	1 processo
Exp10	Network-Bound	1MB	1 processo
Exp11	Network-Bound	100KB	4 processos
Exp12	Network-Bound	1MB	4 processos

Tabela 5.17: Parâmetros fixos para experimentos de influência das categorias de aplicações

Parâmetros fixos	
Número de clientes	8
Classes do cliente	50% bronze 50% gold
Número de requisições por processo	200
Tempo entre requisições	0-1segs
Algoritmo (Broker)	RSV(5G,2B)

Considerando aplicações com categorias individuais, foram definidos 12 experimentos, variando três fatores em um planejamento fatorial completo. O fator "Número de processos" representa a carga de trabalho imposta pelos clientes, definindo consequentemente a quantidade de requisições consecutivas sendo recebidas pelo broker. O fator "Parâmetros" consiste nas informações de entrada solicitadas por uma aplicação. Tais parâmetros definem o nível de processamento dos recursos computacionais utilizados pelo serviço, e por esse motivo, diferem entre cada tipo de aplicação. Em aplicações *CPU-Bound*, por exemplo, o parâmetro de entrada indica o quanto de processamento de CPU seria utilizado para a execução de uma aplicação de complexidade $O(n^4)$. No caso de aplicações *IO-Bound*, este parâmetro representa o tamanho de cada arquivo a ser escrito na máquina que hospeda o provedor de serviços. Da mesma forma, o parâmetro de entrada também representa o tamanho de arquivo em aplicações *Network-Bound*, mas para esse arquivo é realizado uma operação de *download*, enviando uma cópia desse arquivo à máquina do cliente que enviou a solicitação.

A variável de resposta escolhida para análise de resultados foi o tempo de resposta, que posteriormente é subdividida em partes para realizar uma análise mais detalhada sobre os resultados obtidos. Para todos os experimentos foram realizadas 5 execuções utilizadas para determinar a média, o desvio padrão e o intervalo de confiança de 95% de acordo com a tabela *T-Student*. O número de execuções foi definido considerando-se a necessidade de obter diferença significativa dentre os intervalos de confiança dos experimentos que implica em comparação dos resultados.

Para ilustrar melhor o comportamento dos experimentos realizados, o tempo de resposta foi dividido em três etapas, considerando separadamente cada um dos principais componentes da arquitetura WSARCH e suas interações, durante o processo de execução de uma requisição. Para cada uma dessas etapas foi definida uma variável de resposta, representada pela média de tempo gasto para processar as operações pertencentes a uma determinada etapa. Tais etapas são descritas a seguir e ilustradas na Figura 5.20.

1. Tempo de interação Broker-Cliente: Consiste no tempo gasto para enviar a requisição ao broker, somado ao tempo de receber a resposta da requisição do broker logo após as operações de processamento realizado pelo broker. Também é incluído o tempo de processamento da mensagem SOAP recebida na aplicação cliente, especialmente se considerar serviços com operação de download.

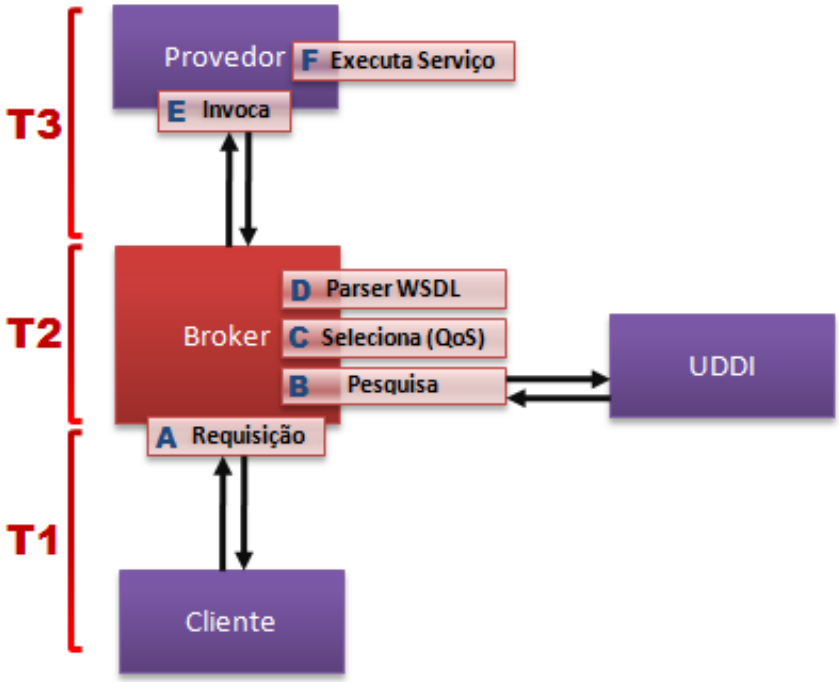

Figura 5.20: Processo de execução de uma requisição na WSARCH

2. Tempo de processamento no Broker: Representa o tempo de todas as operações da WSARCH que são processadas diretamente pelo broker, desde a chegada da requisição no broker até antes de invocar o serviço a ser executado pelo provedor. Consiste em três tarefas: acesso e busca no UDDI de todos os serviços existentes relacionados com a requisição do cliente; seleção de um serviço a partir de informações de QoS enviadas pelo cliente; e análise da WSDL do serviço escolhido.

3. Tempo de processamento do serviço no Provedor: Representa o tempo total de execução do serviço mais o tempo de propagação da requisição de serviço do Broker para o servidor e o retorno da resposta do serviço após sua execução.

5.5.3 Análise de Resultados

Aplicações CPU-Bound

Os gráficos da Figura 5.21 apresentam diferentes perspectivas sobre os dados obtidos com os experimentos realizados para Aplicações *CPU-Bound*. Tais gráficos foram organizados de modo a ilustrar o comportamento de requisições de cada cliente envolvido de acordo com as etapas de execução definidas na Figura 5.20. Nota-se a princípio que o tempo final de resposta das requisições está diretamente relacionado com o tempo gasto para processamento nos provedores, visto que o tempo utilizado na etapa T3 representa uma grande percentagem do tempo total em relação às outras etapas, se considerar experimentos com muito uso de processamento de CPU, como mostra a Figura 5.22.

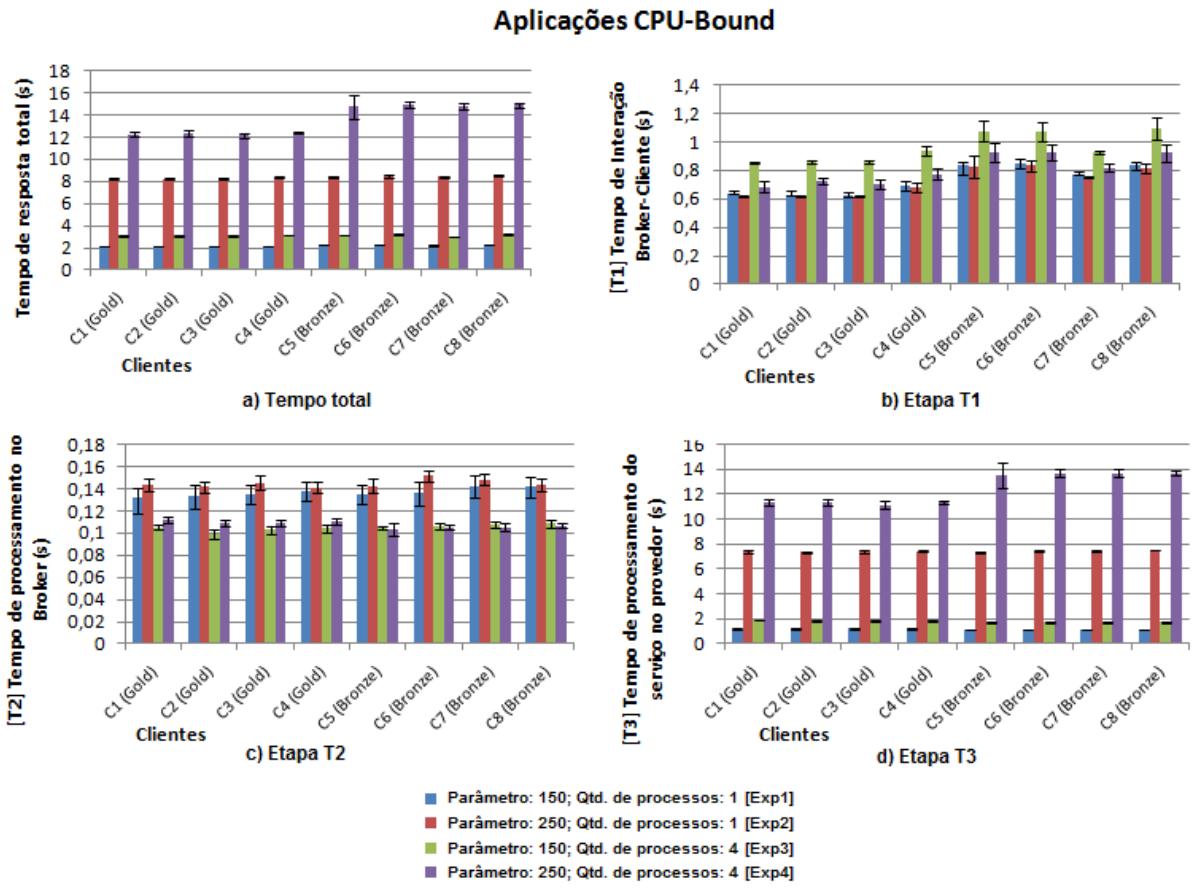

Figura 5.21: Tempo de resposta para aplicações CPU-Bound

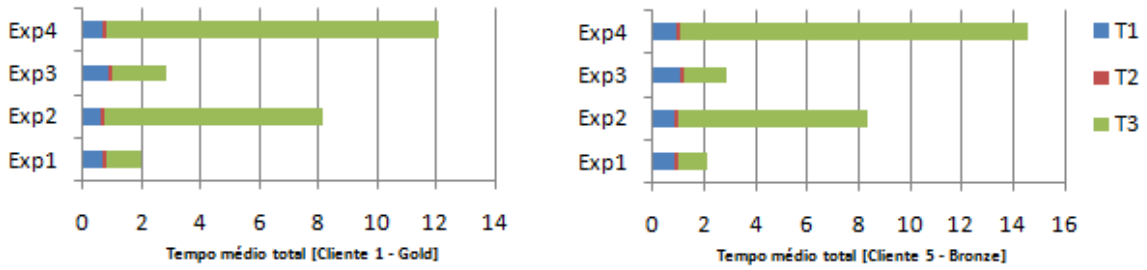

Figura 5.22: Fases do tempo de execução para aplicações CPU-Bound

Outro ponto a ser considerado, tanto no tempo de resposta total, quanto na etapa T3, está na diferenciação de serviços proporcionada pelo algoritmo de seleção instalado no *broker*, a qual só pôde ser percebida no experimento Exp4, pois os níveis definidos para os três primeiros experimentos não foram suficientes no sentido de manter os provedores ocupados. Um único processo por máquina representa um total de oito requisições simultâneas, o que não é suficiente para manter sete provedores suficientemente ocupados para possibilitar a existência de diferenciação de serviços entre as classes, como pode ser observado nos experimentos Exp1 e Exp2. Por outro lado, o curto tempo de processamento nos provedores proporcionado por um parâmetro de valor 150 para

a aplicação *CPU-Bound* também não foi suficiente para manter os provedores ocupados, mesmo com uma quantidade maior de requisições, comportamentos evidenciados nos experimentos Exp1 e Exp3.

Na etapa T1, representada pelas interações entre *broker* e cliente, para experimentos com apenas 1 processo por cliente, não houveram diferenças significativas no tempo gasto, independente do tempo de processamento nos provedores, mas com a variação da quantidade de processos, tais diferenças puderam ser visualizadas, mas de forma pouco significativa. Para experimentos com quatro processos, no entanto, o primeiro comportamento inesperado foi evidenciado: o tempo gasto pela etapa T1 é menor quando o tempo de processamento nos provedores é maior. Uma das prováveis causas para esse comportamento é pelo fato das requisições serem síncronas, ou seja, o cliente deve esperar pela requisição ser processada para enviar a próxima ao *broker*. Deste modo, o cliente espera a resposta a ser processada pelo provedor, e em razão do longo tempo de processamento no provedor, o cliente aguarda um tempo maior pela resposta da requisição, o que aumenta sua ociosidade computacional, refletindo, assim, na diminuição da carga na rede entre cliente e *broker*.

Ao analisar a etapa T2 (responsável pelas interações no *broker*), verifica-se que não ocorreu diferença estatística no tempo observado em relação à variação do tempo de processamento nos provedores. No entanto, este tempo gasto foi menor em experimentos com 4 processos por cliente do que em experimentos com um único processo por cliente. Para solucionar tal problema, os resultados foram reajustados desconsiderando as requisições de *warm-up* e além disso, a quantidade de requisições para realizar os cálculos estatísticos foi igualada entre os experimentos que variam a quantidade de processos. O objetivo foi recalcular a média do conjunto de requisições de apenas um dos quatro processos envolvidos nos experimentos com quatro processos. Entretanto, os resultados obtidos apresentaram comportamentos semelhantes aos observados inicialmente. Esse comportamento pode ter sido ocasionado pela influência do *cache* dos provedores nos resultados. Para comprovar essa suspeita, uma opção que pode ser analisada em um futuro próximo é a reconstrução dos cenários de testes em um ambiente virtualizado.

Aplicações IO-Bound

As aplicações *IO-Bound* apresentaram alguns comportamentos similares em relação aos experimentos realizados para aplicações *CPU-Bound*, principalmente nos tempos obtidos durante as etapas T1 e T2, conforme pode ser observado na Figura 5.23. Além disso, também foi verificado que os resultados obtidos na etapa T3 das aplicações *IO-Bound* apresentaram tempos relativamente menores em maior parte dos experimentos (Exp5, Exp6 e Exp7), e ainda no Exp8 esse tempo foi bem maior, o que significa que o tempo de processamento nos provedores não afetou de forma significativa o tempo gasto pela interação entre cliente e *broker*, e o tempo de processamento no *broker*. Esse comportamento é melhor evidenciado nos gráficos da Figura 5.24, os quais dividem o tempo total para cada uma das três etapas envolvidas em clientes *Gold* e *Bronze*.

Nota-se ainda que o experimento Exp8 apresentou um comportamento semelhante ao Exp4 das aplicações *CPU-Bound* em termos de diferenciação de serviços para clientes *Gold* e *Bronze*. O motivo para tal comportamento também é o mesmo: apenas um único processo por cliente não foi suficiente para manter os provedores de serviços ocupados, e ainda, mesmo com quatro processos, o tempo gasto para a escrita em disco de arquivos de 100KB foi pequeno demais para possibilitar diferenciação de serviço.

Em geral, as características observadas nas aplicações *IO-Bound* foram similares aos de *CPU-Bound* devido aos recursos computacionais utilizados por cada aplicação estarem localizados nos mesmos componentes da arquitetura, os provedores de serviço.

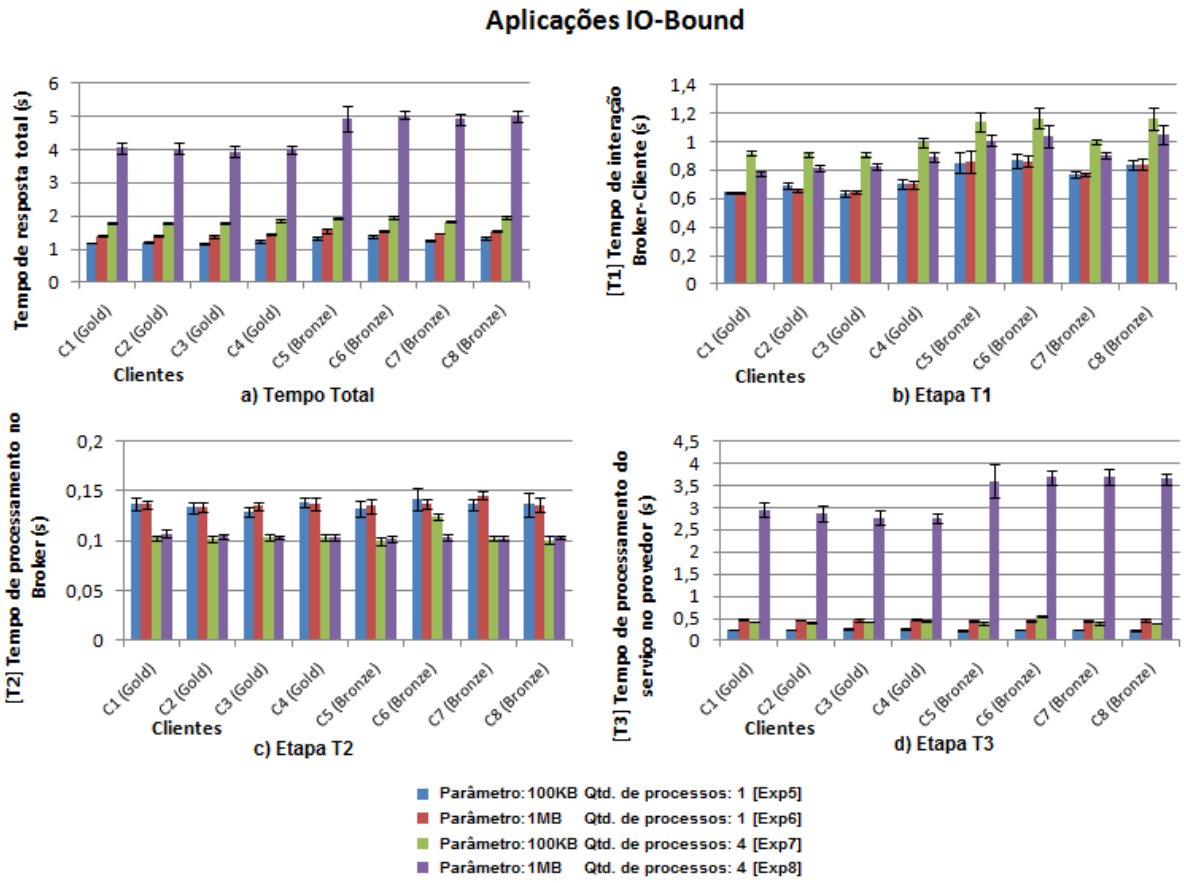

Figura 5.23: Tempo de resposta para aplicações IO-Bound

Aplicações Network-Bound

Para possibilitar a execução de aplicações *Network-Bound* na arquitetura WSARCH, foi implantada em sua estrutura uma nova funcionalidade para contemplar a transferência de informações entre provedor, *broker* e aplicação cliente via anexo de dados (*attachments*) em mensagens SOAP. Um estudo detalhado sobre *attachments* em Web services foi realizado no decorrer deste projeto, e é abordado na Seção 5.2 considerando três técnicas distintas. Para os experimentos com aplica-

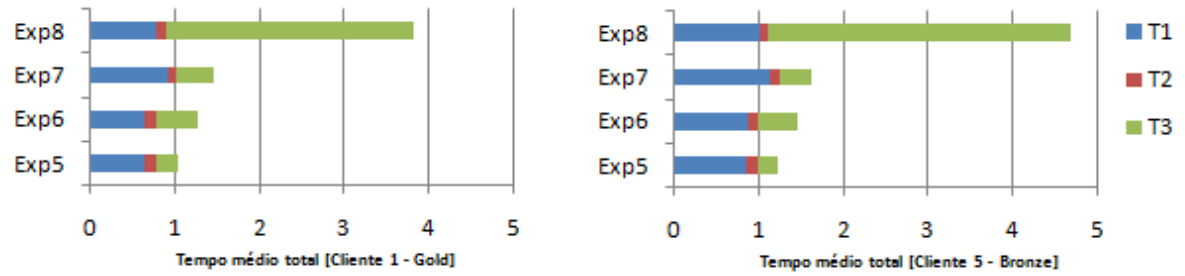

Figura 5.24: Fases do tempo de execução para aplicações IO-Bound

ções *Network-Bound*, foram consideradas a técnica *MTOM* e a operação de *download* durante a implementação da aplicação envolvida. Os gráficos da Figura 5.25 apresentam os resultados dos testes de desempenho realizados com aplicações *Network-Bound*.

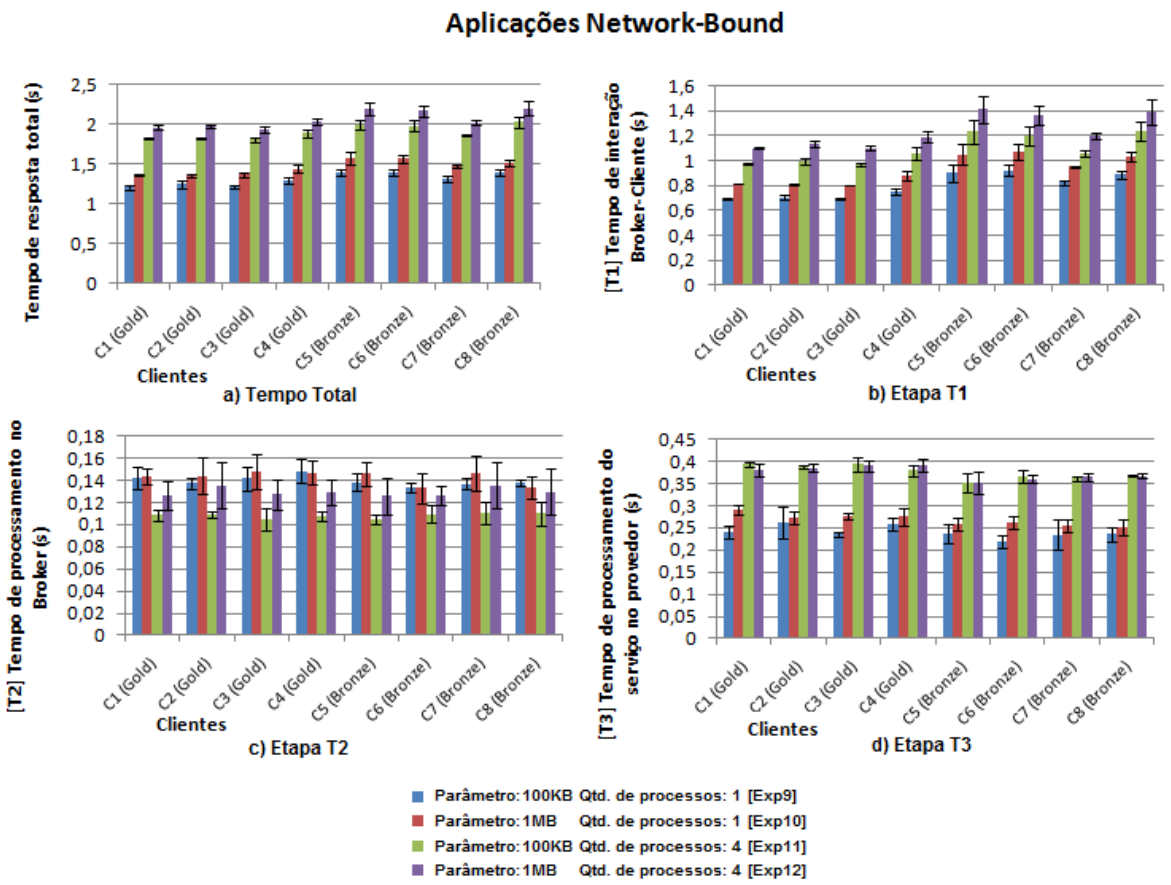

Figura 5.25: Tempo de resposta para aplicações Network-Bound

Nota-se que, ao contrário das aplicações *CPU-Bound*, a fase de execução de maior parcela de tempo de resposta é a etapa T1 (responsável pela interação entre cliente e *broker*), apresentando, dessa forma, valores mais baixos para o tempo de processamento em provedores, como pode ser evidenciado na Figura 5.26. Isso significa que as aplicações *Network-Bound* implementadas levam um tempo menor para realizar o processamento da requisição de serviço no lado do provedor em

comparação com o tempo gasto para interações entre cliente e *broker* e para o processamento da requisição no cliente.

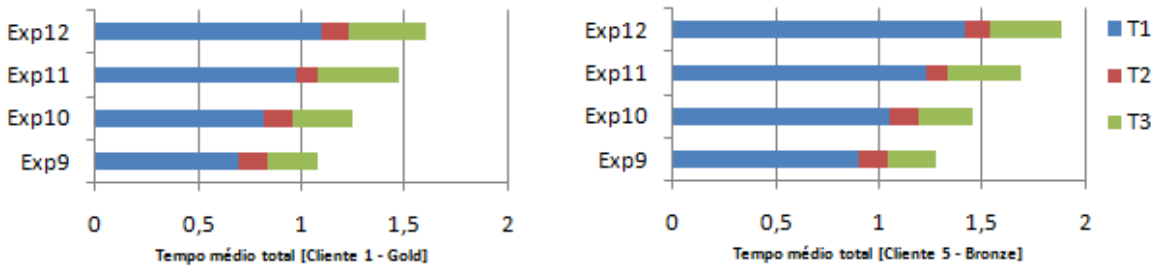

Figura 5.26: Fases do tempo de execução para aplicações Network-Bound

Analisando a etapa T1 dos gráficos da Figura 5.25, verifica-se a influência, tanto da quantidade de processos, quanto do tamanho do arquivo recebido pelo cliente foi melhor evidenciada nos resultados desta etapa. O processamento da mensagem SOAP com o parseamento (decodificação) do anexo recebido no lado do cliente, assim como maior utilização da rede entre cliente e o *broker* devido ao aumento do fluxo de dados, foram responsáveis por este comportamento. Além disso, na etapa T3 também houve um aumento significativo entre os experimentos relacionados com o aumento do número de processos, pelo mesmo motivo da utilização de rede. No entanto, o aumento do tamanho de arquivo não influenciou significativamente no tempo gasto pela etapa T3.

5.6 Considerações finais

Este capítulo apresentou as análises de desempenho e comportamentais relacionadas aos projetos experimentais realizados no decorrer deste projeto de mestrado. Para cada um dos estudos realizados, foram discutidos a configuração do ambiente de testes e o planejamento de experimentos e por fim, os resultados obtidos foram analisados. Os estudos de desempenho relacionados com técnicas de *attachment* de dados em *Web services* e com operações no repositório UDDI serviram como base para auxiliar na implementação do protótipo funcional da arquitetura WSARCH. Após a implementação deste protótipo, foi realizada uma bateria de testes para a validação da WSARCH. Uma análise comportamental foi realizada, comprovando-se também que a WSARCH atinge seu objetivo de ser uma ferramenta que possibilita avaliar o desempenho de *Web Services* com qualidade de serviço. Por fim, um estudo de desempenho com as aplicações implementadas neste projeto foi realizado, verificando-se a existência de comportamentos distintos para cada uma das aplicações ao analisar os resultados sob diferentes perspectivas.

O próximo capítulo tem como objetivo apresentar as principais conclusões obtidas com a implementação das aplicações e os estudos de avaliação de desempenho realizados neste projeto. Também são discutidas as principais contribuições do autor no desenvolvimento da arquitetura

WSARCH e do trabalho em relação ao estado da arte em arquiteturas orientadas a serviço com foco em desempenho.

CAPÍTULO

6

Conclusões

6.1 Conclusão geral

As atividades exercidas durante este projeto de mestrado foram de fundamental importância para o desenvolvimento de uma arquitetura orientada a serviços com foco em qualidade de serviços denominada WSARCH. Uma das principais atividades realizadas foi a implementação de aplicações *Web services* de diferentes características, as quais se tornaram peças importantes para o desenvolvimento e validação de um protótipo funcional da arquitetura WSARCH. Desta forma, as aplicações implementadas foram incorporadas na estrutura desenvolvida para a arquitetura, definindo-se estudos de avaliação de desempenho que possibilitaram a análise de comportamentos distintos observados durante os testes realizados para cada uma das aplicações.

Para alcançar os objetivos propostos de forma efetiva, o autor desta dissertação participou ativamente no processo de desenvolvimento do protótipo da arquitetura WSARCH desde sua etapa inicial, auxiliando na preparação da infraestrutura, assim como na configuração do ambiente e na implementação dos módulos da arquitetura, além de colaborar com os testes de validação da WSARCH. Tais etapas foram essenciais de modo a prover uma infraestrutura adequada para a realização de testes de desempenho com as aplicações implementadas neste projeto. Além disso, o autor também auxiliou na elaboração e desenvolvimento de projetos experimentais relacionados com componentes individuais de uma arquitetura orientada a serviços, tais como testes de desempenho com *attachments* e operações de consulta e publicação para UDDI, os quais foram posteriormente integrados na arquitetura WSARCH.

Uma das maiores preocupações durante a execução de testes de desempenho foi garantir que o ambiente estava sendo configurado corretamente, e ainda, certificar de que as informações corretas

estavam sendo coletadas, diante da complexidade provida pela arquitetura. Sendo assim, surgiu a necessidade de automatizar a execução dos experimentos, e como parte das atividades deste projeto, foram elaborados *scripts* de automatização e, dessa forma, facilitou a configuração e execução dos experimentos.

O desenvolvimento deste projeto de mestrado resultou na implementação de três aplicações com características distintas para fins experimentais, com base nos tipos de recursos computacionais utilizados: *CPU-Bound*, *IO-Bound*, e *Network-Bound*. Tais aplicações foram implantadas nos provedores de serviço da arquitetura e com isso, uma bateria de testes foi elaborada e executada. Além disso, como parte do planejamento de experimentos, o tempo de resposta final das requisições foi dividido em três partes, possibilitando, desta forma, analisar de forma mais ampla o comportamento de cada uma das aplicações. A carga de trabalho também foi considerada dentro do escopo de execução dos experimentos, a fim de verificar sua influência diante as diferentes categorias de aplicações.

Analisando os resultados obtidos, verificou-se a existência de comportamentos distintos para cada uma das aplicações, o que pôde ser visualizado apenas com a divisão do tempo de resposta em três etapas. Em aplicações *CPU-Bound*, destaca-se maior variação no tempo de processamento nos provedores, enquanto em aplicações *Network-bound*, verifica-se um impacto maior no tempo de interação entre cliente e *broker*. As aplicações *IO-Bound* apresentaram resultados similares aos de *CPU-Bound* principalmente quando a carga de trabalho é maior.

Sendo assim, este estudo apresentou resultados interessantes a respeito de cada uma das aplicações na arquitetura WSARCH, os quais podem ser utilizados como forma de encontrar meios de otimização das aplicações. Conhecer melhor o comportamento das aplicações envolvidas possibilita a construção de mecanismos para garantir melhor atendimento ao clientes que acessam tais aplicações, principalmente quando a infra-estrutura é limitada. É necessário conhecer melhor seu comportamento atual para investigar meios de otimização da aplicação, e com isso, possibilitar a garantia de qualidade de serviço. Este trabalho também contribuiu com pesquisas de avaliação de desempenho na área de Web services, comparando-se técnicas de anexo de arquivos em um protótipo construído em ambientes reais, servindo como base para implementação do protótipo da arquitetura WSARCH.

6.2 Contribuições

A principal contribuição deste trabalho foi a implementação e avaliação de desempenho de *Web services* de diferentes características aplicadas em uma arquitetura orientada a serviços com foco em qualidade de serviços. Os resultados obtidos neste estudo apresentaram uma análise aprofundada sobre o desempenho de cada uma das aplicações, os quais demonstraram comportamentos distintos. Este estudo pode ser aplicado como uma forma de investigar possíveis problemas de de-

sempenho na arquitetura, e assim, facilitar no processo de otimização do sistema, provendo melhor qualidade de serviço ao usuário final.

Além disso, o autor deste trabalho de mestrado também contribuiu em diversos aspectos para o desenvolvimento da arquitetura WSARCH, auxiliando na construção de um protótipo funcional, assim como em testes de validação e avaliação de desempenho da arquitetura. Durante a implementação do protótipo da arquitetura WSARCH, o autor auxiliou na inclusão de artefatos para a coleta de dados relacionados ao desempenho da arquitetura, tais como a implantação do *Log Server* e a alteração do motor de processamento de mensagens SOAP presente nos provedores. Para o desenvolvimento de testes de avaliação de desempenho, o autor participou das decisões de planejamento de experimentos e, para facilitar a execução dos experimentos, implementou *scripts* na linguagem *Shell Script* para automatizar a configuração do ambiente de testes.

Outra contribuição importante foi participar da realização de estudos de desempenho de alguns componentes da tecnologia *Web service* que serviram como base para o desenvolvimento da arquitetura WSARCH. O primeiro deles foi a avaliação de desempenho de três técnicas de *attachments* de dados em *Web services*, considerando a influência da rede e o tamanho de arquivo. Este estudo resultou em publicação de artigo em um congresso internacional (ICWS), o que representa uma contribuição importante para a área. O segundo trata-se da avaliação de desempenho das operações de busca e inserção de informações funcionais sobre os Web Services, essas presentes no registro UDDI. Neste estudo foram consideradas WSDLs de diferentes tamanhos, no qual foram avaliados o impacto em relação ao tamanho da WSDL no tempo de resposta final, uma vez que uma operação de inserção de dados no UDDI envolve: recepção da mensagem, *parser* dos dados da WSDL, inserção dos dados em uma base de dados relacional e mensagem de resposta retornada ao cliente.

6.2.1 Produção Científica

Este projeto de mestrado resultou na publicação de alguns artigos em eventos científicos que são listados a seguir:

ESTRELLA, J. C.; ENDO, A. T.; TOYOHARA, R. K. T.; SANTANA, M. J.; SANTANA, R. H. C.; BRUSCHI, S. M. A Performance Evaluation for Web Services Attachments. In: *Proceedings of the IEEE International Conference on Web Services*,ICWS '09,Washington, DC, USA: IEEE Computer Society, 2009, p. 799-806.

ESTRELLA, J. C.; TOYOHARA, R. K. T.; KUEHNE, B. T.; TAVARES, T. C.; SANTANA, R. H. C.; SANTANA, M. J.; BRUSCHI, S. M. A performance evaluation for a qos-aware service oriented architecture. In: *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, Washington, DC, USA: IEEE Computer Society, 2010, p. 260-267.

ESTRELLA, J. C.; TOYOHARA, R. K. T.; SANTANA, M. J.; SANTANA, R. H. C.; BRUSCHI, S. M. *Uma avaliação de desempenho para web services attachments*. Relatório técnico, ICMC-USP, São Carlos, SP, 2009.

6.3 Trabalhos futuros

Os estudos realizados neste projeto de mestrado resultaram em uma grande contribuição o desenvolvimento e evolução da arquitetura WSARCH. No entanto, diversas melhorias podem ser realizadas para a obtenção de novos resultados. Sendo assim, as principais sugestões para trabalhos futuros são descritas a seguir:

- Desenvolvimento de aplicações do mundo real para teste na arquitetura WSARCH, tais como: aplicações complexas que envolvem serviços compostos; mapeamento de aplicações que envolvem grande troca de informações entre cooporações; e aplicações para troca de dados entre experimentos em centro de pesquisa.
- Virtualização do ambiente de testes de desempenho para a obtenção de resultados mais precisos.
- Testes com diferentes cargas de trabalho, as quais simulam comportamentos mais próximos de aplicações reais.
- Utilização de outras métricas para avaliar o resultado dos projetos experimentais, visando a garantia de qualidade de software, tais como o *throughput* (vazão) e SLA (cumprimento de acordo de níveis de serviço).

Referências

ARAUJO, R. M. *Smd - uma ferramenta inteligente para monitoração de desempenho de redes de rádio e predicao de tráfego*. Dissertação de mestrado, UFPA, Belém, PA, 2005.

AXIS2 Apache axis2. Disponível em: http://ws.apache.org/axis2., 2009.

BADIDI, E.; ESMAHI, L.; SERHANI, M. A.; ELKOUTBI, M. Ws-qosm: A broker-based architecture for web services qos management. In: *Innovations in Information Technology*, 2006, p. 1 –5.

BENKNER, S.; ENGELBRECHT, G. A generic qos infrastructure for grid web services. In: *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and International Conference on Services/Advanced*, 2006, p. 141 – 141.

BIANCO, P.; LEWIS, G. A.; MERSON, P. Service level agreements in service-oriented architecture environments. Disponível em: http://www.scribd.com/doc/12863121/Service-Level-Agreements-in-ServiceOriented-Architecture-Environments. Último acesso: 17/02/2011, 2008.

BIH, J. Service oriented architecture (soa) a new paradigm to implement dynamic e-business solutions. *Ubiquity*, v. 7, n. 30, p. 1–1, 2006.

BRANCO, K. R. L. J. C. *Avaliação de carga e desempenho em ambientes paralelos/distribuídos, modelagem e métricas*. Doutorado em ciências de computação, Instituto de Ciências Matemáticas e de Computação, ICMC - USP - São Carlos, 2004.

BROOKS, C. A. An introduction to web services. *Journal*, 2002.

CHIARABINI, L. Corba vs. web services. Disponível em: http://www.mathematik.uni-muenchen.de/chiarabi/middleware.pdf. Último acesso: 23/01/2009, 2004.

D'MELLO, D. A.; ANANTHANARAYANA, V. S.; THILAGAM, S. A qos broker based architecture for dynamic web service selection. In: *AMS '08: Proceedings of the 2008 Second Asia International Conference on Modelling & Simulation (AMS)*, Washington, DC, USA: IEEE Computer Society, 2008, p. 101–106.

ERL, T. *Service-oriented architecture - a field guide to integrating xml and web services*. 1st. ed. Prentice Hall, 2004.

ERRADI, A.; MAHESHWARI, O. A broker-based approach for improving web services reliability. In: *IEEE International Conference on Web Services (ICWS'05)*, 2005.

ESTRELLA, J. C. Real time compression of soap messages in a soa environment. *ZIGDOC '08: Proceedings of the 26th Annual ACM International Conference on Design of Communication*, p. 163–168, 2008.

ESTRELLA, J. C. *Wsarch: Uma arquitetura para a provisão de web services com qualidade de serviço*. Tese de doutorado, Universidade de São Paulo, São Carlos, SP, 2010.

ESTRELLA, J. C.; TOYOHARA, R. K. T.; KUEHNE, B. T.; TAVARES, T. C.; SANTANA, R. H. C.; SANTANA, M. J.; BRUSCHI, S. M. A performance evaluation for a qos-aware service oriented architecture. In: *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, Washington, DC, USA: IEEE Computer Society, 2010, p. 260–267 (*SERVICES '10*,). Disponível em: http://dx.doi.org/10.1109/SERVICES.2010.70

EVDEMON, J. More thoughts on mtom. Disponivel em: http://blogs.msdn.com/jevdemon/archive/2005/05/07/415321.aspx. Ultimo acesso em: 22 nov 2008., 2005.

FARKAS, P.; CHARAF, H. Web services planning concepts. In: *WSCG'2003 – Journal of WSCG*, Plzen, Czech Republic: UNION Agency – Science Press., 2003.

FLOYD, S. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, v. 3, p. 365–386, 1995.

FLOYD, S.; LERES, C.; PAXSON, V.; JACOBSON, V.; FALL, K.; MCCANNE, S. Network research group of lawrence berkeley national laborator. Http://www-nrg.ee.lbl.gov/, 2009.

FRANCÊS, C. R. L.; SANTANA, M. J.; SANTANA, R. H. C.; VIJAYKUMAR, N. L.; CARVALHO, S. V. Queuing statecharts: Uma proposta de especificaÃ§Ã£o para sistemas de filas baseada em statecharts. Disponível em: http://www.lac.inpe.br/~vijay/download/Papers/Sbpo2006.pdf. Último acesso: 27/02/2009, 2006.

GANGLIA Ganglia monitoring system. Disponível em: http://ganglia.sourceforge.net/. Último acesso: 20/12/2010, 2003.

GOMES, M.; GONÇALVES, A. L. Web services: Uma abordagem teórica. *ICPC*, v. 2, n. 5, p. 41–47, 2004.

GRAY, N. A. B. Comparison of web services, java-rmi, and corba service implementation. In: *Fifth Australasian Workshop on Software and System Architectures*, 2004.

GUDGIN, M.; MENDELSOHN, N.; NOTTINGHAM, M.; RUELLAN, H. Soap message transmission optimization mechanism. Disponivel em: http://www.w3.org/TR/soap12-mtom/. Ultimo acesso em: 22 jan 2011., 2005.

GUNARATHN, T. Using soap with attachments in apache axis2. Http://wso2.org/library/1148, 2007.

HASS, H. Foundations and future directions of web services. Http://www.w3.org/2005/Talks/1114-hh-ecows/, 2005.

IBM Standards and web services. Disponível em: http://www-128.ibm.com/developerworks/webservices/standards/. Último acesso: 21/01/2009, 2009.

INNOQ Web services standards overview. Disponível em: http://www.innoq.com/resources/ws-standards-poster/. Último acesso: 21/01/2009, 2007.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley, 1991.

JOSUTTIS, N. M. *SOA na prática - A Arte da Modelagem de Sistemas Distribuídos*. 1st. ed. Oreill'y, 2008.

KAIKAI, G.; WENBIN, Z.; XINTONG, Z. Service-oriented personalized potal platform with qos guarantee. In: *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, 2010, p. V6–205 –V6–209.

KALEPU, S.; KRISHNASWAMY, S.; LOKE, S. W. Verity: a qos metric for selecting web services and providers. In: *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on*, 2003, p. 131–139.
Disponível em: http://dx.doi.org/10.1109/WISEW.2003.1286795

LUDWIG, H. Web services qos: external slas and internal policies or: how do we deliver what we promise? In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops, 2003*, 2003, p. 115 – 120.

MANI, A.; NAGARAJAN, A. Understanding quality of service for web services. Ultimo acesso em: 21 fev 2009, 2002.
Disponível em: http://www.ibm.com/developerworks/library/ws-quality.html

MATOS, J. *Distribuição de carga flexível e dinâmica para provedores de web services*. Qualificação de mestrado, Universidade de São Paulo, São Carlos, SP, 2008.

MENASCÉ, D. A. Qos issues in web services. *IEEE Internet Computing*, v. 6, n. 6, p. 72–75, 2002.

MICROSOFT Web services specification. Disponível em: http://msdn.microsoft.com/en-us/library/ms951274.aspx. Último acesso: 21/01/2009, 2009.

NICKULL, D. Service oriented architecture - whitepaper. 2005.
Disponível em: http://www.adobe.com/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf (Acessado em 05/04/2010)

NOTTINGHAM, M. Xop and mtom. Disponivel em: http://www.mnot.net/blog/2004/02/14/xop. Ultimo acesso em: 12 dez 2008., 2004.

PAPAZOGLOU, M. P.; GEORGAKOPOULOS, D. Service-oriented computing. *Commun. ACM*, v. 46, n. 10, p. 24–28, 2003.

PINTO, H.; BOAS, N. V.; JOSE, R. Utilização do uddi no suporte à descoberta de serviços baseados na localização. *XML Aplicações e Tecnologias Associadas*, 2003.

RECKZIEGEL, M. Descrevendo, descobrindo e integrando web services - uddi. Disponível em: http://imasters.uol.com.br/artigo/4474/webservices/descrevendo_descobrindo_e_integrando_web_services_-_uddi/. Último acesso: 02/02/2009, 2006.

SABETTA, A.; KOZIOLEK, H. Measuring performance metrics: Techniques and tools. In: EUSGELD, I.; FREILING, F. C.; REUSSNER, R., eds. *Dependability Metrics*, Springer, 2005, p. 226–232 (*Lecture Notes in Computer Science*, v.4909).
Disponível em: http://dblp.uni-trier.de/db/conf/dagstuhl/dm2005.html#SabettaK05a

SANTANA, M. J.; SANTANA, R. H. C.; FRANCÊS, C. R. L.; ORLANDI, R. C. Methodologies for performance evaluation of distributed systems - a comparison study. In: *The Proceedings of the: Summer Computer Simulation Conference*, Arlington, Virginia, USA, 1997, p. 124–28.

SANTANA, R. H. C.; SANTANA, M. J.; ORLANDI, R. C. G. S.; SPOLON, R.; JÚNIOR, N. C. Técnicas para avaliação de desempenho de sistemas computacionais. Notas didáticas do ICMC-USP, 1994.

SAPHIR, W.; TANNER, L. A.; TRAVERSAT, B. Job management requirements for nas parallel systems and clusters. In: *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK: Springer-Verlag, 1995, p. 319–336.

SHETH, A.; CARDOSO, J.; MILLER, J.; KOCHUT, K.; KANG, M. QoS for service-oriented middleware. 2002.
Disponível em: http://lsdis.cs.uga.edu (Acessado em 20/12/2010)

SILVA, F. O.; ROSA, P. F. The quest for the web services stack: a fast trip. In: *ICWE '06: Proceedings of the 6th international conference on Web engineering*, New York, NY, USA: ACM, 2006, p. 93–94.

SOAP Soap specifications. Disponível em: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. Último acesso: 23/01/2009, 2007.

SOTOMAYOR, B. A short introduction to web services. Disponível em: http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s02.html. Último acesso: 23/01/2009, 2005.

SRINIVASAN, L.; TREADWELL, J. An overview of service-oriented architecture, web services and grid computing. Disponível em: http://devresource.hp.com/drc/technical_papers/grid_soa/index.jsp. Último acesso: 06/02/2009, 2005.

SUN, W.; ZHANG, J.; LIU, F. Ws-sla: A framework for web services oriented service level agreements. *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, p. 1–4, 2006.

TAVARES, R.; WESTPHALL, C. An architecture to provide qos in web services. In: *ICC '06 - IEEE International Conference on Communications*, 2006, p. 921 –925.

THOMAS, J. P.; THOMAS, M.; GHINEA, G. Modeling of web services flow. *E-Commerce Technology, IEEE International Conference on*, v. 0, p. 391, 2003.

TSAI, W. T.; LEE, Y.-H.; CAO, Z.; CHEN, Y.; XIAO, B. Rtsoa: Real-time service-oriented architecture. In: *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering*, Washington, DC, USA: IEEE Computer Society, 2006, p. 49–56.

VOORSLUYS, W. *Avaliação deÍndices de carga de memória em sistemas computacionais distribuídos*. Mestrado em ciências de computação, Instituto de Ciências Matemáticas e de Computação, ICMC - USP - São Carlos, 2006.

W3C Web services architecture. Disponível em: http://www.w3.org/TR/2003/WD-ws-arch-20030514/. Último acesso: 21/01/2009, 2003.

W3C Web services architecture. Disponível em: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. Último acesso: 21/01/2009, 2004.

W3C Web services description language (wsdl) version 2.0. Disponível em: http://www.w3.org/TR/2007/REC-wsdl20-20070626. Último acesso: 23/01/2009, 2007.

WASZNICKY, M. Using web services instead of dcom. Disponível em: http://msdn.microsoft.com/en-us/library/aa302336.aspx. Último acesso: 02/02/2011, 2002.

WU, C.-H.; SU, D.-C.; CHANG, J.; WEI, C.-C.; LIN, K.-J.; HO, J.-M. The design and implementation of intelligent transportation web services. In: *The design and implementation of intelligent transportation Web services*, 2003, p. 49 – 52.

YANG, Y. Faster data transport means faster web services with mtom/xop. Disponivel em: http://www.devx.com/xml/Article/34797. Ultimo acesso em: 12 dez 2008., 2007.

YEOM, G.; TSAI, W.-T.; BAI, X.; MIN, D. Design of a contract-based web services qos management system. In: *Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09. 29th IEEE International Conference on*, 2009, p. 306 –311.

ZHAO, Q.; TAN, Y. A load balancing based model for dynamic web service selection. *International Symposium on Computational Intelligence and Design*, v. 1, p. 501–505, 2009.