Caracterização de cargas de trabalho para
avaliação de desempenho em Web services

Thiago Caproni Tavares

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 16 de junho de 2009

Assinatura: _____

Caracterização de cargas de trabalho para avaliação de desempenho em Web services

Thiago Caproni Tavares

Orientadora: *Prof. Dra. Regina Helena Carlucci Santana*

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.

**USP - São Carlos
Junho/2009**

*“Ainda que eu falasse as línguas
dos homens e dos anjos, e não tivesse Amor,
seria como o metal que soa
ou como o sino que tine.
(1 Coríntios 13)*

Dedicatória

*Aos meus pais Aluísio e Ida,
por terem me dado a vida
e a alegria de viver.
Amo vocês . . .*

Agradecimentos

A Deus, em primeiro lugar, por me conceder, saúde, paz, e coragem para vencer os desafios da vida.

Aos meus pais e ao meu irmão, por todo o incentivo e amor a mim dedicados durante toda a minha vida. Obrigado pelo amor incondicional.

A Profa. Regina, a quem serei eternamente grato, pela orientação e suporte ao longo de todo o desenvolvimento do trabalho. Ao Prof. Marcos pelas lições e ensinamentos da vida profissional e pessoal.

A minha namorada Keninha, pelo amor, carinho e amizade durante todo esse período de mestrado. Ela quem esteve sempre presente em todos os momentos difíceis e felizes.

Ao Júlio César, pela ajuda na idealização e no desenvolvimento deste trabalho. Obrigado pelas dicas e sugestões que permitiram a realização do projeto.

A família Swerts, pela amizade e carinho a mim atribuídos durante todos esses anos. A eles, que nos momentos de descontração, inundaram meu coração de felicidade a qual me fazia seguir sempre em frente.

Aos meus amigos machadenses em especial Renato, Ney, Thiago, Eduardo (Seco), Bruninho e Márcio, pela presença constante em minha vida.

Aos amigos de república Jonathan, Bruno e Douglas. Obrigado pelas conversas (e discussões) e pela companhia ao longo desses anos.

Aos amigos do LaSDPC Maycon, Felipe, Pedro, Edwin, Kenji, Mário, Lourenço, Luiz César, Fabiano, Oswaldo, Michelle, Priscila, Paulo, Bruno (Cascão), Bruno (Nardone), Luiz, pelo convívio alegre e saudável.

Aos meus amigos dos mais variados estados brasileiros Cássio, Bruno Kimura, Ana Luiza, Roberto Rigolin, Gabriel e Fernando, pelos momentos de descontração e trocas de experiências.

Aos professores do grupo de Sistemas Distribuídos e Programação Concorrente, que contribuíram com sugestões e discussões sobre o trabalho.

Aos demais funcionários do ICMC, pessoal da secretaria da Pós-Graduação, pessoal da cantina, vigilantes, faxineiras e todos aqueles que de alguma forma contribuíram para o sucesso deste trabalho.

A FAPESP pelo suporte financeiro.

Resumo

Esta dissertação de mestrado aborda um estudo sobre a caracterização de cargas de trabalho para *Web services* por meio da análise de Documentos *WSDL* (*Web Service Description Language*). Esses documentos, que representam a interface para os serviços, foram obtidos na *Web* e seus conteúdos analisados estatisticamente. A metodologia utilizada para alcançar a caracterização desejada, constitui-se da coleta e análise dos dados de 1346 arquivos *WSDL* válidos. O resultado do estudo proposto nesta dissertação contribui para a avaliação de desempenho no que diz respeito a caracterizar os diversos aspectos das cargas de trabalho de *Web services*. Dentre os dados obtidos ressaltam-se as porcentagens da ocorrência de algumas características tais como o número de operações, tipos de *Binding*, quantidade de parâmetros de entrada e saída e tipos de dados mais utilizados. Para auxiliar na busca e avaliação das características das *WSDLs* uma nova ferramenta denominada *WSDLANalyzer* é proposta e implementada.

Abstract

This MSc dissertation approaches a study on workload characterization for web services by means of the analysis of WSDL (Web Service Description Language) documents. These documents, that represent the interface to the services, were all obtained in the web and their contents were statistically analyzed. The methodology used to reach the desired characterization comprises the collection and analysis of 1346 WSDL valid files. The result of the study proposed in this dissertation contributes to the performance evaluation of web services in terms of characterizing the several aspects of the workloads. Among the data obtained, the percentage of occurrences of some features, such as the number of operations, the types of binding, the amount of input and output parameters and the more used data types, are highlighted. To support the search and evaluation of the WSDL features a new tool, named WSDLANalyzer, is proposed and implemented.

Sumário

1	Introdução	1
1.1	Contextualização	2
1.2	Motivação e Objetivos	3
1.3	Estrutura da Dissertação	4
2	SOA e Web Services	5
2.1	Arquitetura Orientada a Serviço	6
2.1.1	Conceitos Básicos	8
2.2	Web Services	10
2.2.1	Visão Geral dos Web Services	11
2.2.2	Padrões de Web Services	14
2.3	Arquitetura dos Web Services	14
2.3.1	Tecnologias Base	14
2.3.2	Camada de Descrição - WSDL	15
2.3.3	Camada de Descoberta	18
2.3.4	Camada de Mensagem	19
2.3.5	Demais Camadas	23
2.4	Considerações Finais	24
3	Avaliação de desempenho de WS: Cargas de Trabalho	25
3.1	Avaliação de Web Services	26
3.2	Benchmarks para Web Services	29
3.3	Cargas de Trabalho	32
3.3.1	Caracterização de Cargas de Trabalho	32
3.3.2	Modelo de Carga de Trabalho	33
3.3.3	Busca por Invariantes	33
3.3.4	Fonte de Dados	34

3.3.5	Filtragem dos dados	35
3.4	Considerações Finais	35
4	Caracterização de Carga de Trabalho de WS	37
4.1	Modelo de Carga de Trabalho para WS	37
4.1.1	Invariantes na Carga de Trabalho para WS	40
4.1.2	Fonte de dados	40
4.1.3	Filtragem dos Dados	41
4.1.4	Modelagem da Carga de Trabalho	41
4.2	Considerações Finais	43
5	WSDLAnalyzer	45
5.1	Módulo de Busca	46
5.1.1	Google SOAP Search API	46
5.1.2	Biblioteca HTML Parser	48
5.1.3	Filtragem dos dados	49
5.2	Módulo de Análise	51
5.2.1	Gráficos	54
5.3	Modelagem da WSDLAnalyzer	58
5.3.1	Diagrama de Classes	59
5.3.2	Diagramas de Sequência	61
5.4	Interface Gráfica	63
5.4.1	Janela do Módulo de Análise	63
5.4.2	Janelas do Módulo de Busca	65
5.5	Considerações Finais	66
6	Análise e Avaliação de WSDLs	69
6.1	Coleta de Dados	70
6.2	Binding - SOAP 1.1	71
6.2.1	Tamanhos dos Documentos WSDL	72
6.2.2	Quantidade de Operações	73
6.2.3	Requisições no Estilo Document	75
6.2.4	Requisições no Estilo RPC	77
6.2.5	Codificação do Tipo Literal	78
6.2.6	Codificação do Tipo Encoded	80
6.2.7	Parâmetros de Entrada	82
6.2.8	Parâmetros de Saída	84
6.2.9	Resumo das Distribuições Estatísticas para o Binding SOAP 1.1 .	85
6.2.10	Resumo das Variáveis Estatísticas para o Binding SOAP 1.1 . . .	86
6.3	Binding SOAP 1.2	86

6.3.1	Tamanhos dos documentos WSDL	87
6.3.2	Quantidade de Operações	88
6.3.3	Requisições no Estilo Document	89
6.3.4	Requisições no Estilo RPC	90
6.3.5	Codificação do Tipo Literal	91
6.3.6	Codificação do Tipo Encoded	92
6.3.7	Parâmetros de Entrada	92
6.3.8	Parâmetros de Saída	94
6.3.9	Resumo das Distribuições Estatísticas para o Binding SOAP 1.2 .	95
6.3.10	Resumo das Variáveis Estatísticas para o Binding SOAP 1.2 . . .	96
6.4	Binding SOAP 1.1 Vs Binding SOAP 1.2	96
6.5	Binding HTTP	99
6.5.1	Tamanhos dos documentos WSDL	99
6.5.2	Quantidade de Operações	100
6.5.3	Métodos GET e POST	101
6.5.4	Parâmetros de Entrada	102
6.5.5	Parâmetros de Saída	103
6.5.6	Resumo das Distribuições Estatísticas para o Binding HTTP . . .	104
6.5.7	Resumo das Variáveis Estatísticas para o Binding HTTP	105
6.6	SOAP 1.1 vs SOAP 1.2 vs HTTP	105
6.7	Considerações Finais	106
7	Conclusão	109
7.1	Resultados e Contribuições	110
7.2	Trabalhos Futuros	112
A	Gráficos Complementares	115
A.1	Binding SOAP 1.1 - Gráficos dos Tipos de Dados	115
A.1.1	Distribuições Estatísticas para Parâmetros de Entrada	115
A.1.2	Distribuições Estatísticas para Parâmetros de Saída	116
A.1.3	Comparação Entre as Distribuições	118
A.2	Binding SOAP 1.2 - Gráficos	119
A.2.1	Distribuição Estatística para o Número de Operações	119
A.2.2	Distribuição Estatística para Parâmetros de Entrada	119
A.2.3	Distribuição Estatística para Parâmetros de Saída	121
A.2.4	Comparação Entre as Distribuições	122
B	Funções de Distribuição de Probabilidade	123
B.1	Beta-Generalizada	123
B.1.1	Parâmetros e Domínio	123

B.1.2	Função Acumulativa	123
B.2	Weibull	124
B.2.1	Parâmetros e Domínio	124
B.2.2	Função Acumulativa	124
B.3	Lognormal	124
B.3.1	Parâmetros e Domínio	124
B.3.2	Função Acumulativa	124
B.4	Pearson	125
B.4.1	Parâmetros e Domínio	125
B.4.2	Função Acumulativa	125
B.5	Geométrica	125
B.5.1	Parâmetros e Domínio	125
B.5.2	Função Acumulativa	125
B.6	Gaussiana Inversa	125
B.6.1	Parâmetros e Domínio	125
B.6.2	Função Acumulativa	126

Lista de Figuras

2.1	Elementos da SOA - Adaptado de Papazoglou [2003].	8
2.2	Visão Empresarial da Arquitetura Orientada a Serviço - Adaptado de Erl [2005]	9
2.3	Composição de Web Services - Adaptado de [Coulouris et al., 2005]. . . .	12
2.4	Arquitetura dos Web Services - Adaptado de [W3C, 2004a].	15
2.5	Parte Abstrata de um documento WSDL	16
2.6	Parte Concreta de um documento WSDL	17
2.7	Interações da Arquitetura de Web Services - Adaptado de [Yu et al., 2008].	19
2.8	Exemplo de um envelope SOAP de requisição.	20
2.9	Exemplo de um envelope SOAP de resposta.	21
4.1	Lacuna na avaliação de desempenho de Web services.	38
4.2	Variedade de Funcionalidades dos Web Services [Li et al., 2007].	39
5.1	Processo de Busca.	46
5.2	Algoritmo de Busca em Portais de Web Site.	48
5.3	Processo de Análise.	52
5.4	Tipos de Dados - XML Schema [W3C, 2004c]	53
5.5	Exemplo de Gráfico Barras	56
5.6	Exemplo de Gráfico de Linhas	56
5.7	Exemplo de Gráfico de Pizza	57
5.8	Exemplo de Gráfico de Dispersão	58
5.9	Diagrama de Classes Conceitual.	59
5.10	Diagrama de Sequência do Processo de Análise.	62
5.11	Diagramas de Sequência do Processo de Busca.	63
5.12	Janela Principal da Ferramenta WSDLAnalyzer.	64
5.13	Janela da Busca no Google.	65

5.14 Janela da Busca em Web Sites.	66
6.1 SOAP 1.1: Gráfico de Pizza do Tamanho dos Documentos WSDL.	72
6.2 SOAP 1.1: Gráfico de Pizza do Número de Operações.	74
6.3 SOAP 1.1: Gráfico de Dispersão - Tamanho X Número de Operações . .	74
6.4 SOAP 1.1: Ajuste de Distribuição para o Número de Operações.	75
6.5 SOAP 1.1: Gráfico de Pizza de Operações no Estilo Document.	76
6.6 SOAP 1.1: Ajuste de Distribuição para o Número de Requisições no Estilo Document.	76
6.7 SOAP 1.1: Gráfico de Pizza de Operações no Estilo RPC.	77
6.8 SOAP 1.1: Ajuste de Distribuição para o Número de Requisições no Estilo RPC.	78
6.9 SOAP 1.1: Gráfico de Pizza da Utilização do Tipo de Codificação Literal. .	79
6.10 SOAP 1.1: Gráfico de Dispersão - Document X Literal	79
6.11 SOAP 1.1: Ajuste de Distribuição para o Número de Codificação do Tipo Literal.	80
6.12 SOAP 1.1: Gráfico de Pizza da Utilização do Tipo de Codificação Encoded. .	81
6.13 SOAP 1.1: Gráfico de Dispersão - RPC X Encoded	81
6.14 SOAP 1.1: Ajuste de Distribuição para o Número de Codificação do Tipo Encoded.	82
6.15 SOAP 1.1: Gráfico de Pizza - Número Médio de Parâmetros de Entrada por Operação.	82
6.16 SOAP 1.1: Ajuste de Distribuição para o Número Parâmetros de Entrada. .	83
6.17 SOAP 1.1: Gráfico de Pizza - Número Médio de Parâmetros de Saída por Operação.	84
6.18 SOAP 1.1: Ajuste de Distribuição para o Número Parâmetros de Saída. .	85
6.19 SOAP 1.2: Gráfico de Pizza do Tamanho dos Documentos WSDL.	87
6.20 SOAP 1.2: Gráfico de Pizza do Número de Operações.	88
6.21 SOAP 1.2: Gráfico de Dispersão - Tamanho X Número de Operações . .	89
6.22 SOAP 1.2: Gráfico de Pizza de Operações no Estilo Document.	90
6.23 SOAP 1.2: Ajuste de Distribuição para o Número de Requisições no Estilo Document.	90
6.24 SOAP 1.2: Ajuste de Distribuição para o Número de Codificação do Tipo Literal.	91
6.25 SOAP 1.2: Gráfico de Pizza - Número Médio de Parâmetros de Entrada por Operação.	93
6.26 SOAP 1.2: Ajuste de Distribuição para o Número Parâmetros de Entrada. .	93
6.27 SOAP 1.2: Gráfico de Pizza - Número Médio de Parâmetros de Saída por Operação.	94

6.28	SOAP 1.2: Ajuste de Distribuição para o Número Parâmetros de Saída. .	95
6.29	HTTP: Gráfico de Pizza do Tamanho dos Documentos WSDL.	99
6.30	HTTP: Gráfico de Pizza do Número de Operações.	100
6.31	HTTP: Ajuste de Distribuição para o Número de Operações.	100
6.32	HTTP: Gráfico de Linha para os Métodos GET e POST.	101
6.33	HTTP: Ajuste de Distribuição para o Método GET.	102
6.34	HTTP: Ajuste de Distribuição para o Número de Parâmetros de Entrada. .	103
6.35	HTTP: Ajuste de Distribuição para o Número de Parâmetros de Saída. .	103
A.1	SOAP1.1: Ajuste de Distribuição para o Número de ComplexType.	116
A.2	SOAP1.1: Ajuste de Distribuição para o Número de SimpleType.	116
A.3	SOAP1.1: Ajuste de Distribuição para o Número de String.	117
A.4	SOAP1.1: Ajuste de Distribuição para o Número de ComplexType.	117
A.5	SOAP1.1: Ajuste de Distribuição para o Número de SimpleType.	118
A.6	SOAP1.1: Ajuste de Distribuição para o Número de String.	118
A.7	SOAP 1.2: Ajuste de Distribuição para o Número de Operações.	119
A.8	SOAP 1.2: Ajuste de Distribuição para o Número de ComplexType. . . .	120
A.9	SOAP 1.2: Ajuste de Distribuição para o Número de String.	120
A.10	SOAP 1.2: Ajuste de Distribuição para o Número de ComplexType. . . .	121
A.11	Ajuste de Distribuição para o Número de String.	121

Lista de Tabelas

3.1 Invariantes dos Servidores Web.	34
4.1 Invariantes dos Web Services	40
6.1 Chaves de Pesquisa do Dicionário de Busca.	71
6.2 Resumo das Distribuições Estatísticas para o Binding SOAP 1.1	85
6.3 SOAP 1.1: Resumo Total das Características Requisição/Codificação. . .	86
6.4 SOAP 1.1: Resumo Parcial das Características Requisição/Codificação. .	86
6.5 SOAP 1.2: Resumo das Distribuições Estatísticas para o Binding SOAP 1.1	95
6.6 SOAP 1.2: Resumo Total das Características Requisição/Codificação. . .	96
6.7 SOAP 1.2: Resumo Parcial das Características Requisição/Codificação. .	96
6.8 Tabela Comparativa de Distribuições de Ajuste	97
6.9 Tabela Comparativa de Variáveis Estatísticas	98
6.10 HTTP: Resumo das Distribuições Estatísticas para o Binding HTTP . . .	104
6.11 HTTP: Resumo Total das Variáveis Estatísticas para o Binding HTTP. . .	104
6.12 HTTP: Resumo Parcial das Variáveis Estatísticas para o Binding HTTP. .	104
6.13 Tabela Comparativa de Distribuições de Ajuste	106
6.14 Tabela Comparativa de Variáveis Estatísticas	106
A.1 Comparação das Distribuições Estatísticas dos Tipos de Dados de En- trada e Saída	119
A.2 Comparação das Distribuições Estatísticas dos Tipos de Dados de En- trada e Saída	122

Lista de Abreviaturas

API: *Application Programming Interface*
B2B: *Business-to-Business*
BPEL: *Business Process Execution Language*
C2B: *Costumer-to-Business*
CCM: *CORBA Component Model*
CORBA: *Common Object Request Broker Architecture*
DTD: *Document Type Definition*
DUNS: *Data Universal Numbering System*
FTP: *File Transfer Protocol*
HTML: *HyperText Markup Language*
HTTP: *HyperText Transfer Protocol*
IDL: *Interface Description Language*
J2EE: *Java 2 Platform, Enterprise Edition*
JAVA2WSDL: *Java To WSDL*
LAN: *Local Area Network*
MS-DCOM: *Microsoft's Distributed Component Object Model*
OASIS: *Advancing Open Standards for the Information Society*
QoS: *Quality of Service*
RDF: *Resource Description Framework*
RMI: *Remote Method Invocation*
RPC: *Remote Procedure Call*
SMTP: *Simple Mail Transfer Protocol*
SOA: *Service Oriented Architecture*
SOAP: *Simple Object Access Protocol*
TCP/IP: *Transmission Control Protocol/Internet Protocol*
TPC: *Transaction Processing Performance Council*
UDDI: *Universal Description, Discovery, and Integration*
UML: *Unified Modeling Language*
URI: *Uniform Resource Identifier*
URL: *Uniform Resource Locator*
URN: *Uniform Resource Name*

W3C: *Wide World Web Consortium*
WAN: *Wide Area Network*
WS: *Web Services*
WSCE: *Web Services Composition Environment*
WSDL: *Web Service Description Language*
WSDL: *Web Services Description Language*
WSDL4J: *Web Services Description Language for Java*
WSS: *Web Services Security*
XML Infoset: *XML Information Set*
XML: *Extensible Markup Language*
XSD: *XML Schema Definition*

Introdução

O contexto empresarial mundial tem sofrido inúmeras mudanças nos últimos anos. Muitas dessas mudanças ocorreram devido à ascensão da *Web* como ferramenta de negócio. No final da década de noventa, diretores executivos de muitas empresas investiram muito dinheiro em *TI* (Tecnologia de informação) com a crença de que o retorno do investimento seria garantido. No entanto, quando esse retorno não se materializou na proporção que se esperava percebeu-se que a área de *TI* deveria ser gerida mantendo-se os custos reduzidos e sem expectativa de que ela traz vantagens de negócio, pois a maioria das empresas a utiliza [Carter, 2007].

Nesse contexto, as arquiteturas orientadas a serviço (*SOA - Service Oriented Architecture*) foram propostas como ponto comum entre negócios e *TI*. Essa nova abordagem surge como uma promessa para que as grandes corporações consigam acompanhar o dinamismo do mercado. A decomposição dos sistemas em serviços bem definidos que se comunicam utilizando um conjunto de padrões compõem o principal apelo desse novo tipo de arquitetura. Essa decomposição permite que vários serviços sejam recombinaados em novos processos de negócios aumentando a capacidade de uma empresa se adaptar a novos contextos de mercado.

Os conceitos de *SOA* podem ser aplicados para a construção de serviços publicados na *Web*: os chamados *Web services*. De forma abrangente, os *Web services* são uma maneira de expor um serviço através da *Web*, ou seja, eles são uma instanciação de uma arquitetura orientada a serviço. A popularidade dos *Web services* como uma alternativa para a construção de aplicações “*business to business*”, de forma interoperável e flexível, fez com que as definições sobre a arquitetura e a instanciação da arquitetura se sobrepusessem.

1.1 Contextualização

Os *Web services* têm sido foco de atenção de muitos pesquisadores ao redor do mundo. Essa tecnologia tornou-se uma importante abordagem na construção de sistemas distribuídos devido às características como interoperabilidade e fraco acoplamento (flexibilidade). Tais características são providas através de um conjunto de protocolos de transporte *text-based*. Esses protocolos permitem que diferentes aplicações interajam umas com as outras sobre ambientes heterogêneos. Em contrapartida, os protocolos aumentam a sobrecarga de comunicação em relação a outras tecnologias como *RMI (Remote Method Invocation)* e *RPC (Remote Procedure Call)*, que trocam dados na forma binária.

Os *Web services* são atualmente a principal instanciação de uma arquitetura orientada a serviços. A *SOA* tem como principal objetivo a resolução de problemas de negócios dentro do âmbito empresarial. As empresas têm buscado utilizar essa nova abordagem nas suas soluções de TI. Recentemente, em matéria apresentada na *ComputerWorld*, a *Gartner* (empresa importante na área de consultoria em TI) relatou que a procura pelas arquiteturas orientadas a serviços é crescente. Esse crescimento, em alguns casos, pode trazer expectativas em torno da *SOA* que podem levar a conclusão de que essa arquitetura é cara e não traz consigo os resultados esperados, diz Paolo Malinverno vice-presidente de pesquisas da *Gartner* [ComputerWorld, 2009]. A utilização das arquiteturas orientadas a serviço é recente e pode implicar em um custeio excessivo para a sua implantação por falta de “know-how” sobre o assunto. Além disso, as empresas devem entender quais as vantagens e desvantagens de adotar essa proposta de arquitetura. Dessa forma, pesquisadores em todo o mundo têm buscado encontrar soluções que minimizem custos e maximizem as vantagens de se utilizar arquiteturas orientadas a serviço. Pesquisas para o desenvolvimento de ferramentas, técnicas de composição e seleção de serviços, segurança, avaliação de desempenho são exemplos de linhas de pesquisa bastante discutidas.

Vários trabalhos encontrados na literatura têm focado a avaliação e modelagem de desempenho dos *Web services*. Essa é uma tarefa complexa devido ao número de camadas envolvidas na pilha da arquitetura dos *Web services* - *UDDI (Universal Description Discovery and Integration)*, *brokers*, consumidores e provedores de serviço. Diversos trabalhos têm avaliado o desempenho de *Web services* considerando os problemas relativos à interação entre consumidores e provedores de serviço [Cai et al., 2002], [Chen et al., 2006] e [Chen et al., 2007]. Na maioria dos casos, trabalhos que seguem essa proposta realizam a avaliação de desempenho utilizando medições de forma direta no sistema. Essas medições são realizadas através de monitores ou *benchmarks*, avaliando partes específicas, como a influência da variação de tamanhos das mensagens de requisição, das linguagens de programação ou dos protocolos de

transporte. Outros trabalhos, ainda, indicam as melhores maneiras, ferramentas e padrões que visam provisão de desempenho dos *Web services* [Machado and Ferraz, 2005] e [Estrella et al., 2008].

Dentro do contexto empresarial, a avaliação de desempenho tem um papel importante no conhecimento de quão bem um determinado sistema computacional atende as necessidades de uma organização. Essa avaliação pode ser utilizada principalmente para realizar um planejamento da capacidade do sistema analisado. No caso dos *Web services*, muitos aspectos influenciam o desempenho de um sistema que utiliza essa proposta de implementação de sistemas distribuídos.

1.2 Motivação e Objetivos

A caracterização de cargas de trabalho é parte importante na avaliação de desempenho de sistemas computacionais [Menascé and Almeida, 2002]. Dentro do contexto dos *Web services*, este trabalho caracteriza a composição das mensagens que são utilizadas na interação consumidor/provedor de *Web services*. Essas mensagens podem ser consideradas como o componente básico da carga de trabalho, ou seja, elas são as unidades genéricas de carga enviadas e recebidas pelo provedor, e também, pelo consumidor do serviço. Os *Web services* possuem algumas características particulares sobre as cargas de trabalho. O processo de requisição de serviço é custoso tanto para o consumidor quanto para o provedor do serviço. O consumidor ao requisitar um *Web service* deve serializar os dados pertinentes a requisição do serviço em uma mensagem que geralmente é definida no formato *SOAP* (*Simple Object Access Protocol*). Por sua vez, o servidor, ao receber a mensagem de requisição, deve deserializar essa mensagem, processá-la sob uma determinada regra de negócio, e então, serializar os resultados em uma mensagem de resposta que deve ser enviada ao consumidor. Ao receber a mensagem de resposta, o consumidor deserializa a mensagem, e então, manipula os resultados da requisição da maneira que lhe for mais conveniente.

Diversos estudos encontrados na literatura tratam de fatores específicos de todo esse processo de requisição de um *Web service*. Alguns trabalhos avaliam o custo de se transformar objetos de uma determinada linguagem em uma mensagem *SOAP* [Zilora and Ketha, 2008]. Outros estudos avaliam o desempenho dos *toolkits* de processamento das mensagens *SOAP* disponibilizados pela indústria [Davis and Parashar, 2002] e [Head et al., 2005]. Há também trabalhos que propõem modelos de desempenho onde são considerados aspectos que influenciam a latência entre enviar uma requisição e receber uma resposta através do mapeamento de diversos fatores [Chen et al., 2007]. Enfim, existe uma variedade de trabalhos tratando diversos aspectos que apresentam impacto no desempenho dos *Web services*. Entretanto,

um estudo sobre a composição das mensagens (componente básico) utilizadas nesses trabalhos não é encontrado. Isso sinaliza uma lacuna nos trabalhos de avaliação de desempenho em *Web services*.

O trabalho de mestrado apresentado nesta dissertação contribui para o preenchimento dessa lacuna encontrada propondo uma caracterização de cargas de trabalho de *Web services* baseada em documentos *WSDL* obtidos na *Web*. A linguagem *WSDL* é um formato *XML* que descreve a maneira como um serviço é acessado através da especificação das operações disponíveis pelo serviço, estilo de requisição, tipos de codificação e parâmetros de entrada e saída utilizados no acesso as operações. Assim, os documentos *WSDL* são uma fonte para analisar como é composto o componente básico da carga de trabalho dos *Web services*. Dessa forma, uma proposta de caracterização de cargas de trabalho estática é considerada neste trabalho. O nome estático para essa caracterização é dado pelo fato de que o estudo se baseia nas informações contidas nos documentos *WSDL*. Adicionalmente, o trabalho apresentado nesta dissertação apresenta uma implementação de uma ferramenta chamada *WSDLANalyzer* que dá suporte a obtenção e análise de documentos *WSDL*.

1.3 Estrutura da Dissertação

O restante deste documento está organizado como segue:

- O Capítulo 2 apresenta a fundamentação teórica que embasa o trabalho, reunindo de forma sucinta os principais conceitos utilizados no seu desenvolvimento e que são importantes para o seu entendimento.
- O Capítulo 3 discorre sobre os aspectos relacionados a avaliação de desempenho em *Web services*. Além disso, são discutidas as maneiras de se definir cargas de trabalho para que um sistema computacional possa ser avaliado.
- O Capítulo 4 aborda a metodologia utilizada para a caracterização das cargas de trabalho para *Web services*.
- O Capítulo 5 apresenta os detalhes da implementação da ferramenta *WSDLANalyzer*. Essa ferramenta é utilizada para a busca e análise de documentos *WSDL*.
- O Capítulo 6 discorre sobre os resultados obtidos com a realização deste trabalho. Esse capítulo apresenta um conjunto de análises que são demonstradas através de distribuições de probabilidade, gráficos e variáveis estatísticas.
- O Capítulo 7 resume os principais resultados obtidos, as contribuições do projeto e os trabalhos futuros sugeridos.

SOA e Web Services

Os projetos de aplicações distribuídas e sistemas de informação apresentaram grandes mudanças desde o advento dos *middlewares*. O primeiro *middleware* a ser utilizado foi o de objetos distribuídos, que tornava possível a construção de aplicações distribuídas através de objetos remotos como se eles estivessem sendo acessados localmente. Isso ocorreu em meados dos anos 90 quando esse paradigma tornou-se popular na indústria de software. O *CORBA* (*Common Object Request Broker Architecture*) foi o primeiro *framework* de suporte à computação distribuída através de objetos [Martin-Flatin and Löwe, 2007]. A necessidade de reduzir custos e aumentar a reutilização de código fez com que se criasse o conceito de componentes de software. Coletivamente, os componentes de software cooperam na execução de uma determinada tarefa através de uma interface que os descreva. Outros exemplos de *frameworks* que oferecem esse conceito são o *J2EE* (*Java 2 Enterprise Edition*), *CCM* (*CORBA Component Model*) e o *COM/COM+*. O grande problema desses *frameworks* é com relação ao forte acoplamento. A complexidade de se garantir interoperabilidade e abertura entre sistemas diferentes, fez com que eles fossem indicados àqueles nichos de mercado onde a necessidade de desempenho superasse a flexibilidade [Martin-Flatin and Löwe, 2007].

No final dos anos noventa, com o sucesso da *Web*, o *W3C* (*World Wide Web Consortium*) introduziu o conceito de *Web services*, sob o paradigma da arquitetura orientada a serviços. A proposta foi criar uma arquitetura em que fosse possível construir sistemas fracamente acoplados, visando principalmente, os sistemas *e-business*.

2.1 Arquitetura Orientada a Serviço

A SOA deve ser vista e entendida como um modelo arquitetural independente de qualquer plataforma tecnológica. Atualmente, os *Web services* são a principal plataforma tecnológica que promove a instanciação de uma arquitetura orientada a serviço [Erl, 2007]. A seção 2.2 aborda os principais conceitos e aspectos dos *Web services*.

Uma analogia interessante para o entendimento de SOA é apresentada em [Colan, 2004]. No ano de 221 a.C. o imperador Qin Shi Huang Di unificou vários estados em conflito dentro de um novo país que hoje chamamos de China. Isso se deu através da introdução de padrões que consolidaram a cultura e facilitaram as atividades de comércio. Um padrão interessante diz respeito às dimensões que as carroças deveriam possuir para permitir que todas elas conseguissem trafegar, de forma eficiente, nas estradas, também padronizadas, construídas por Qin. Outro padrão diz respeito à unificação da escrita que permitiu que todos os habitantes pudessem trocar mensagem. Não obstante, o imperador ainda iniciou a construção da muralha da China. De modo semelhante aos sistemas de informação, o imperador Qin desenvolveu modelos padronizados de transporte, troca de mensagens e segurança (como um *firewall*). Da mesma forma, a integração de sistemas em tempos modernos tem se beneficiado de padrões que aplicações disponíveis sobre uma variedade de ambientes heterogêneos se comuniquem de forma eficaz (interoperabilidade).

SOA é uma proposta arquitetural de tecnologia da informação orientada a negócios. Ela suporta a integração de negócios através de serviços e assegura que os sistemas computacionais possam se adaptar às mudanças nas necessidades dos negócios de forma rápida, fácil e econômica. Em suma, a SOA é uma forma de reorganização de aplicações distribuídas. Isso é alcançado por meio de uma infra-estrutura que engloba um conjunto de serviços que são acessados através de interfaces padronizadas. Uma vez que todos os elementos da arquitetura estão alocados, as aplicações podem acessá-las, utilizando protocolos de transporte padrão, sem a necessidade de envolver soluções ponto a ponto sobre uma infra-estrutura de protocolos proprietários. Particularmente, essa abordagem de arquitetura é utilizada quando múltiplas aplicações precisam se comunicar sobre uma variedade de tecnologias e plataformas. Assim, empresas podem reunir serviços no intuito de executar transações de negócios com um esforço de programação reduzido [Papazoglou, 2003].

A SOA é uma forma lógica de desenvolvimento de sistemas de software para fornecer serviços para outras aplicações ou outros serviços distribuídos em uma rede através de publicação e descoberta de interfaces. SOA, na sua essência, segue na contramão do desenvolvimento de aplicações monolíticas. A idéia é que sistemas computacionais sejam desenvolvidos como um conjunto de serviços orquestrados por

um processo de negócio [Carey, 2008]. Os princípios básicos de SOA são orientação a componentes e acoplamento fraco. Entretanto, esses dois requisitos são abstratos na prática, pois não definem exatamente como um sistema de software baseado em SOA deve ser construído. Em [Hau et al., 2008] são apresentadas algumas regras necessárias para se construir arquiteturas de software compatíveis com a SOA:

- **Orientação a Interface:** refere-se à propriedade de encapsular as funcionalidades oferecidas pelo serviço (assim como no paradigma de Orientação a Objetos). A maneira como a funcionalidade é executada não é importante para o usuário. Na prática, isso é alcançado através de uma especificação bem definida que descreva a interface, a capacidade e a disponibilidade do serviço.
- **Padronização:** os componentes da SOA devem ser capazes de integrar diferentes elementos para a realização de novas funcionalidades. Um componente de software pode trocar dados com um outro utilizando a mesma interface através de uma forma padronizada de interação.
- **Autonomia e modularidade:** autonomia, no contexto de SOA, significa que o serviço é uma entidade que pode executar uma determinada tarefa sem interagir com outro serviço. Em outras palavras, um serviço não deve depender de um outro serviço na realização de uma tarefa. Adicionalmente, a comunicação entre serviços deve ser fracamente acoplada (baseada em mensagens e *stateless*).
- **Orientação a negócios:** consiste na tradução da orientação a objetos para o mundo dos sistemas empresariais. A idéia central da orientação a objetos é representar programaticamente coisas encontradas no mundo real na forma de objetos. Nesta mesma linha de raciocínio, um serviço deve representar uma abstração de algum processo de negócio encontrado no mundo real.

A construção de aplicações distribuídas utilizando os conceitos da SOA traz uma série de vantagens que justificam a adoção desse tipo de arquitetura. Dentre elas, destacam-se [Hau et al., 2008]:

- **Agilidade:** empresas tornam-se ágeis e se adaptam com maior rapidez a novos requisitos. Por exemplo, um processo de negócio pode utilizar serviços disponíveis ordenados de forma diferente para se ajustar a novas regulamentações de mercado. Consequentemente, as mudanças no código das aplicações são reduzidas.
- **Complexidade:** a infra-estrutura de TI torna-se menos complexa. Ela é manipulada de forma mais simples, pois cada componente pode ser considerado individualmente. Os componentes que têm seu código fonte modificado não afetam seus usuários.

- **Transparência:** Os usuários conhecem apenas a interface que descreve o serviço. Eles não necessitam conhecer a maneira como o serviço é implementado.
- **Reuso:** componentes de software podem ser utilizados em vários contextos sem que eles sejam implementados várias vezes.
- **Interoperabilidade:** considerado o maior benefício da SOA devido à facilidade dos componentes de software interagirem e trocarem dados, uma vez que apenas as interface e *middleware* da aplicação devem ser conhecidos ao invés de toda a implementação.

2.1.1 Conceitos Básicos

A base da arquitetura orientada a serviço define uma interação entre agentes de software como uma troca de mensagens entre clientes e provedores de serviços. Clientes são agentes de software que requisitam a execução de um serviço, enquanto provedores são agentes que fornecem o serviço e ambos podem ser simultaneamente clientes e provedores. Os provedores são responsáveis pela publicação e descrição dos serviços fornecidos e os clientes devem estar aptos a encontrar as descrições e acessar os serviços.

Três tipos de elementos são partes básicas da SOA: o provedor do serviço, o registro do serviço e o consumidor do serviço. As interações envolvem a publicação, a busca e o acesso aos serviços como ilustrado na Figura 2.1.

Figura 2.1: Elementos da SOA - Adaptado de Papazoglou [2003].

De forma simplificada, um provedor de serviço hospeda um módulo de software disponível (uma dada implementação de serviço) em uma rede, e então, define uma descrição desse serviço e a publica para o consumidor ou para o componente de descoberta. Um serviço é um processo de negócio implementado em software,

encapsulado com uma interface documentada formalmente que deve ser conhecida não apenas pelos agentes que o projetaram, mas também pelos agentes que não possuem o conhecimento da maneira como o serviço foi projetado [Papazoglou, 2003].

Figura 2.2: Visão Empresarial da Arquitetura Orientada a Serviço - Adaptado de Erl [2005] .

O cliente utiliza uma operação de busca para receber a descrição do serviço do componente de descoberta que pode ser um registro ou um repositório como o *UDDI* [Martin-Flatin and Löwe, 2007] e através dessa descrição invocar ou interagir com a implementação do serviço. Provedores e clientes são construções lógicas e podem exibir características de ambos os componentes.

Fundamentalmente, a visão lógica de um serviço é que ele possui uma camada de descrição e uma de aplicação. A camada de descrição detém as interfaces que descrevem os serviços e ela é associada ao elemento de registro apresentado na Figura 2.1. Por sua vez, a camada de aplicação possui a implementação das aplicações que são disponibilizadas como serviços. Essas aplicações podem estar localizadas em diferentes provedores. Há ainda uma terceira camada, onde as empresas que utilizam a arquitetura orientada a serviço definem seus processos de negócios como ilustrado na Figura 2.2. Nessa camada são criados serviços ou uma composição deles a partir das necessidades de cada uma das áreas de atuação das empresas [Erl, 2005]. As interfaces são utilizadas para descrever o serviço e esconder detalhes

referentes à maneira com que as aplicações foram implementadas (independência de linguagens de programação), fornecendo assim a principal característica da SOA, a interoperabilidade entre as aplicações. As aplicações consumidoras podem ser organizadas de forma a agrupar serviços para se formar um determinado processo de negócio (Figura 2.2), ou ainda de forma simples como apresentado na Figura 2.1.

A arquitetura orientada a serviço é a base dos *Web services*. Sua característica é prover uma arquitetura de software que reorganiza as funcionalidades de uma aplicação através de serviços que são acessados por meio de interfaces, utilizando para tal finalidade o paradigma de requisição/resposta. Eles podem ser caracterizados como um sistema distribuído heterogêneo e fracamente acoplado. A arquitetura orientada a serviço é um estilo arquitetural de software que tem como foco alcançar acoplamento fraco entre agentes de software que interagem [Martin-Flatin and Löwe, 2007]. Provedores de serviços e consumidores são implementados via agentes de software em que um serviço é uma unidade de trabalho feita por um provedor de forma a alcançar um resultado final para o consumidor. Cada serviço implementa uma função de negócio específica que é disponibilizada aos consumidores sem que eles tenham o conhecimento da forma com que essa implementação foi realizada [Martin-Flatin and Löwe, 2007]. As principais características da arquitetura orientada a serviços são:

- **Neutralidade de tecnologia:** a arquitetura deve ser implementada através de protocolos padrões, descrições e mecanismos de descoberta.
- **Acoplamento fraco:** não há a necessidade do conhecimento de quaisquer convenções ou estruturas internas no cliente ou no lado do serviço.
- **Transparência de localização:** os serviços devem ter suas definições e informação de localização armazenadas em repositórios como um *UDDI* e serem acessados por uma variedade de clientes que podem localizar e invocar os serviços não importando a sua localização.

2.2 Web Services

Os *Web services* utilizam o conceito de Arquiteturas Orientadas a Serviços (SOA), para disponibilizar serviços a clientes na *Web*, e têm apresentado rápida adoção e suporte de companhias como *SUN Microsystems*, *IBM*, *Microsoft*, *BEA*, *Oracle* e *Borland* [Yu et al., 2007] [Martin-Flatin and Löwe, 2007]. Os *Web services* podem ser vistos como um componente de software onde os detalhes da implementação são escondidos por uma interface. Ferramentas como *java2WSDL* e *Apache Axis* fornecem formas de se obter essas interfaces a partir de sistemas orientados a objetos

já existentes [Yu et al., 2007] [Apache, 2009b]. De modo geral, a interface de um *Web service* possui um conjunto de operações que podem ser usadas por um cliente na *Web* [Coulouris et al., 2005]. As próximas seções contemplam as principais características dos *Web services* abordando as descrições dos seus protocolos de comunicação e os componentes que pertencem a sua arquitetura.

2.2.1 Visão Geral dos Web Services

Os serviços oferecidos pelos *Web services* podem ser acessados por diferentes clientes. Para viabilizar esse acesso é necessário adotar uma sintaxe comum e extensível para descrevê-los e uma forma de comunicação entre os serviços. O W3C - *Web Services Architecture Working Group* destaca-se como o órgão que investiga essa atividade. Para a descrição dos *Web services*, vem sendo adotado o padrão XML [W3C, 2009] que trata-se de um vocabulário de comunicação padrão entre os serviços.

Um próximo ponto a ser considerado é a definição de um protocolo para comunicação entre os *Web services* e clientes. O SOAP [W3C, 2007a] é um protocolo amplamente utilizado pelos *Web services* para esse fim. A principal característica de grande parte dos *Web services* é a possibilidade que eles possuem de processar mensagens SOAP formatadas em XML [Coulouris et al., 2005]. Mais detalhes sobre esses protocolos são discutidos posteriormente.

Servidores de grandes empresas da *Web* como *Amazon*, *Yahoo*, *Google*, oferecem interfaces de serviços permitindo aos clientes manipularem seus recursos *Web*. A *Amazon.com*, por exemplo, oferece um *Web service* que permite operações onde os clientes podem requisitar informações de produtos, adicionar itens no carrinho de compras ou então verificar o status de uma compra.

Composição de Web Services

O processo de composição e execução de *Web services* (WSCE - *Web Service Composition and Execution*) consiste na criação de um fluxo de trabalho, a partir de *Web services* já disponíveis, que possibilita a funcionalidade de um novo serviço [Agarwal et al., 2008].

A disponibilidade da interface de um *Web service* permite que as operações providas por ele sejam combinadas com as operações de outros *Web services* fornecendo assim uma nova funcionalidade. As vantagens de compor serviços podem ser ilustradas através do seguinte exemplo: muitas pessoas utilizam seus navegadores *Web* para fazer reservas de passagens aéreas, hotéis e aluguel de carro em várias páginas diferentes. Contudo, se cada um desses *Web sites* oferecesse uma interface de serviço padrão, todas essas operações poderiam ser combinadas por um único serviço de agente de viagens conforme ilustrado na Figura 2.3 [Coulouris et al., 2005].

Figura 2.3: Composição de Web Services - Adaptado de [Coulouris et al., 2005].

Padrões de Comunicação

De forma geral, os *Web services* utilizam protocolos de requisição e resposta síncrona com seus clientes ou se comunicam através de mensagens assíncronas. Este último estilo de comunicação pode ser utilizado mesmo quando as requisições exigem respostas, ou seja, um cliente envia uma requisição e recebe uma resposta posteriormente de maneira assíncrona. O serviço de gerente de viagens ilustra de forma adequada os dois padrões de comunicação disponíveis nos *Web services*:

- A troca assíncrona de documentos pode ser exemplificada através da troca de informações sobre os detalhes de datas e destinos das viagens que são retornadas, ao consumidor do serviço, de tempos em tempos. Neste caso não há a preocupação com desempenho.
- A troca síncrona deve ocorrer, por exemplo, no momento da verificação do cartão de crédito e as interações com o cliente devem ser feitas com um protocolo de requisição e resposta.

Um terceiro padrão de comunicação considera a ocorrência de eventos. Os clientes realizam o registro nos eventos de interesses e são notificados da ocorrência desses eventos [Coulouris et al., 2005].

Modelo de Programação

Os *Web services* são projetados para serem utilizados na *Web*, onde há uma enorme variedade de linguagens de programação. Eles podem ser implementados em qualquer linguagem que ofereça mecanismos para a sua construção (bibliotecas e *APIs* - *Application Programming Interface*), pois eles utilizam protocolos de comunicação, protocolos de acesso e linguagens de descrição não proprietários e abertos [Martin-Flatin and Löwe, 2007]. Dessa forma, é possível criar uma forma comum e interoperável de comunicação. Por exemplo, a representação do protocolo *SOAP* e dos dados é realizada utilizando-se o padrão *XML*¹.

Referência do Serviço

A *Web* utiliza o *URI* (*Uniform Resource Identifiers*) como uma maneira simples de referenciar documentos. Os *URIs* são encontrados de duas formas: (1) na forma de um *URL* (*Uniform Resource Locator*) que fornece uma referência dependente de localização para acesso a um documento e (2) através de um *URN* (*Uniform Resource Names*) que é uma referência persistente e independente de localização [Tanenbaum and van Steen, 2006].

Assim, cada serviço na *Web* tem um *URI* que é utilizado como referência pelos clientes do serviço. O *URL* é a forma mais comum de *URI* utilizada e contém o nome de domínio de um computador onde os serviços estão hospedados. No entanto, há a possibilidade de que o *URL* seja mudado freqüentemente, e devido a essas modificações haverá a necessidade de se obter o novo *URL* a partir de um serviço de pesquisa em um *URN*.

Transparência

Uma característica a ser considerada é afastar o programador dos detalhes como representação e encapsulamento dos dados fazendo com que as invocações remotas sejam parecidas com as execuções locais. Essa característica não é garantida pela infra-estrutura ou pelos *middlewares* dos *Web services*. Em linguagens de programação como *Java*, *Perl*, *Python* ou *C++*, existem *API's* que fornecem os meios necessários para a geração automática dos procedimentos de empacotamento de *XML*, que podem ser utilizados para auxiliar o desenvolvedor na implementação de *Web services* [Coulouris et al., 2005].

¹O padrão textual do *XML* tem a desvantagem de ocupar maior espaço do que um formato binário, provocando um maior custo de processamento. Todavia, o formato *XML* é bastante legível para os seres humanos e facilita a construção e depuração das mensagens.

2.2.2 Padrões de Web Services

Os *Web services* são definidos através de um número de padrões da indústria. Eles podem ser identificados dentro de duas gerações. Cada uma dessas gerações é associada a um conjunto de padrões e especificações [Erl, 2007]:

- **Primeira geração:** geração onde foi definido o núcleo (*core*) das tecnologias e especificações dos *Web services*. Nessa geração, especifica-se a descrição dos *Web services* através de documentos *WSDL* e *XSD (XML Schema Definition)*. O protocolo para troca de mensagem é especificado por meio do *SOAP (Simple Object Access Protocol)*. E por fim, os métodos para publicação e descoberta de serviço são definidos através de *UDDI*. Tais especificações têm sido adotadas pela indústria há algum tempo. No entanto, essa opção apresenta lacunas no que diz respeito a qualidade de serviço e a funcionalidade dos serviços em um nível empresarial.
- **Segunda geração:** algumas das lacunas relacionadas à qualidade de serviço englobam segurança no nível de mensagem, transações entre serviços e confiabilidade na comunicação. Essas lacunas vêm sendo preenchidas com a segunda geração dos *Web services*. O preenchimento dessas lacunas é alcançado através de várias extensões que são chamadas de *WS-**. Um exemplo é o *WS-Policy (W3C Recommendation)* que representa um conjunto de especificações que definem, dentre outras características, restrições de segurança.

2.3 Arquitetura dos Web Services

A arquitetura dos *Web services* envolve muitas camadas e tecnologias. Nesta seção serão abordadas as ferramentas e tecnologias utilizadas no projeto, na construção e na publicação dos *Web services*. A Figura 2.4 ilustra algumas dessas famílias de tecnologias que podem ser utilizadas no desenvolvimento dos *Web services* [W3C, 2004a] e [IBM, 2009]. Por sua vez, a Figura 2.7 mostra a associação dessas tecnologias com suas respectivas camadas.

2.3.1 Tecnologias Base

A pilha arquitetural dos serviços *Web* possui algumas tecnologias base. Dentre elas podem ser destacadas:

- **eXtensible Markup Language:** O *XML* foi desenvolvido pelo *XML Working Group* (originalmente conhecido como *SGML Editorial Review Board*) formado sob a supervisão do W3C em 1996. O *XML* é uma tecnologia chave utilizada em muitos

Figura 2.4: Arquitetura dos Web Services - Adaptado de [W3C, 2004a].

contextos. Ela oferece um formato de dados estruturado, extensível, flexível e padronizado. O *XML* é utilizado como linguagem base no núcleo das tecnologias utilizadas na pilha da arquitetura dos *Web services*.

- **Document Type Definition:** o *DTD* define a construção legal dos blocos de um documento *XML*, ou seja, ele define quais elementos e atributos um documento *XML* deve possuir e como eles devem estar dispostos no documento.
- **XML Schema:** O *XML-Schema* é uma alternativa ao *DTD*, baseada em *XML*. Ele é comumente referenciado como *XSD* (*XML Schema Definition*). Na prática, o *XSD* possui a mesma função que o *DTD*. No entanto, o *XSD* é mais poderoso que o *DTD* devido ao suporte a tipos de dados, a utilização da sintaxe da linguagem *XML* e ao fato de ser extensível.

2.3.2 Camada de Descrição - WSDL

A camada de descrição contém a semântica formal para representar as mensagens que os *Web services* podem entender, descrevendo as restrições de dados dentro das mensagens. O *RDF* (*Resource Description Framework*) é utilizado na criação de ontologias, as quais categorizam mensagens e serviços. Além disso, essa camada tem como responsabilidade definir a maneira como os *Web services* são acessados. O estudo apresentado nesta dissertação baseia-se nessa camada para obter informações

de carga de trabalho dos *Web services*. Portanto, ela será abordada com mais detalhes, visando principalmente o entendimento dos documentos *WSDL*.

```
<definition>
  ...
  <message name="BookInfoIn">
    <part name="otherParameters" element="BookInfoComplexType"/>
    <part name="BookName" type="xs:string"/>
  </message>

  <message name="BookInfoOut">
    ...
  </message>

  <interface name="Catalog">
    <operation name="GetBook">
      <input name="BookInfoRequest" message="BookInfoIn"/>
      <output name="BookInfoResponse" message="BookInfoOut"/>
    </operation>
  </interface>
</definitions>
```

Figura 2.5: Parte Abstrata de um documento WSDL

O *WSDL* é análogo à interface Java, arquivos *headers* do C++ ou a linguagem de definição de interface (*IDL - Interface Description Language*) do *CORBA*. Ele define a gramática que descreve os *endpoints*, entradas e saídas de um *Web service*. Como os protocolos de comunicação e formatos de mensagens são padronizados pela comunidade *Web*, o *WSDL* tornou-se um importante mecanismo capaz de descrever as comunicações de um modo estruturado. Atualmente existem duas versões do padrão *WSDL*, a versão 1.1 e a versão 2.0 (especificada em junho de 2007). No entanto, todos os documentos *WSDL* encontrados para a realização deste trabalho tratavam-se de documentos na versão 1.1. O *WSDL* é um formato *XML* inicialmente desenvolvido pela *IBM* e *Microsoft* que fornece um meio de descrever *Web services*. Ele permite separar o serviço de duas formas: abstrata e concreta [Erl, 2005]; [W3C, 2004a].

Antes de acessar o serviço é necessário conhecer o que esse serviço faz. A parte abstrata do documento *WSDL* oferece essa informação referindo-se ao conteúdo que é independente dos detalhes de implementação. Esses detalhes são definidos dentro de dois elementos: *interface/portType*² e *message*. O elemento *interface* define um grupo de operações disponíveis no serviço. As operações são representadas utilizando o elemento *operation* que possui os elementos *input* e *output* como elementos filhos. Esses elementos filhos referenciam mensagens. As mensagens são representadas pelo elemento *message* que é filho do elemento *definition*. O elemento *message* possui uma ou mais partes lógicas. Tais partes são representadas

²A versão 2.0 mudou o nome desse elemento de *portType* para *interface* [W3C, 2007b].

pelo elemento `part` com um tipo de algum sistema de tipos usando um atributo de `message-typing`. Existem dois atributos `message-typing`: `element` e `type`. O atributo `element` pode referenciar um elemento *XSD QName (Qualified Name)*. O atributo `type`, por sua vez, referencia tipos simples ou complexo (*Simple/Complex type*). A Figura 2.5 mostra um exemplo da parte abstrata de um documento WSDL.

```
<definition>
...
<binding name="CatalogSOAP" type="tns:Catalog">
  <soap:binding style="rpc"/>
  <transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetBook">
    <input name="BookInfoRequest">
      <soap:body use="literal"/>
    </input>
    <output name="BookInfoResponse">
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

<service name="CatalogBookService">
  <endpoint name="Endpoint1" binding="CatalogSOAP">
    <soap:address location="http://localhost:8080/CatalogBookService">
    </soap:address>
  </endpoint>
</service>
</definitions>
```

Figura 2.6: Parte Concreta de um documento WSDL

A parte concreta refere-se a informações de localização e implementação do serviço no que diz respeito a detalhes de acesso. Os detalhes concretos são especificados com os elementos `binding`, `service` e `port/endpoint`³. O elemento `binding` especifica o formato da mensagem e os protocolos de transporte utilizados na comunicação para uma ou mais interfaces (`portType/Interface`). O elemento `binding` tem dois elementos filhos principais: `soap:binding` e `http:binding`. O elemento `soap:binding` possui dois atributos: `style` e `transport`. Os valores do atributo `style` podem ser `rpc` ou `document`. O atributo `transport` especifica o protocolo de transporte que será usado na comunicação. O elemento `http:binding`, definido na especificação do *WSDL 1.1*, informa que a comunicação será realizada com os métodos *HTTP GET* e *HTTP POST*. Assim como na parte abstrata, a parte concreta possui um elemento `operation` que é filho do elemento `binding`. Nele é definido como será a construção do corpo da mensagem SOAP (`literal` ou `encoded`). O elemento `endpoint` associa um endereço de rede com um `binding`. Os endpoints, pelos quais os *Web services* podem ser acessados, são definidos dentro do elemento `service`. A

³A versão 2.0 mudou o nome deste elemento de `port` para `endpoint` [W3C, 2007b].

Figura 2.6 mostra um exemplo da parte concreta de um documento WSDL.

O suporte a tipos de dados oferecido pelo WSDL é provido dentro do elemento `types`. Ele define os tipos de dados que são relevantes na execução de troca de mensagens. A linguagem WSDL usa *XML Schema* como sistema de tipos. O XSD é utilizado no elemento `part` para definir os tipos de dados que serão transmitidos na comunicação. Esses tipos de dados podem ser primitivos, derivados ou extensíveis. Tipos de dados primitivos são `xs:string`, `xs:double`, `xs:float`, `xs:time` e etc. Tipos derivados são `xs:normalizedString`, `xs:integer`, `xs:Name` e etc. Tipos extensíveis são semelhantes a uma *struct* da linguagem C. Ele permite que sejam criados novos tipos de dados.

2.3.3 Camada de Descoberta

Os *Web services* são úteis apenas se potenciais usuários conseguirem encontrar informações que permitam acessá-los. A camada de descoberta oferece processos e mecanismos pelos quais os *Web services* podem ser encontrados. Lu et al. [2007] apresenta um trabalho avaliando qual é o mecanismo de busca de *Web service* preferido pelos desenvolvedores. O autor constatou que a maioria dos desenvolvedores prefere fazer uma busca manual através de *engines* de busca e portais especializados.

Contudo, existe um protocolo Universal de Integração, Descoberta e Descrição (UDDI) que é um padrão aprovado pela OASIS (*Advancing Open Standards for the Information Society*). Tal protocolo é membro chave da pilha da arquitetura dos *Web services*, pois define um método padrão para publicação e descoberta de componentes de software baseados na arquitetura orientada a serviços [OASIS, 2006].

No contexto dos *Web services*, o UDDI tem o propósito de representar dados e metadados sobre os *Web services*. Um registro pode servir a uma rede pública ou infra-estrutura interna de alguma organização. O UDDI prove um mecanismo baseado em padrões para classificar, catalogar e gerenciar *Web services*. Assim, eles podem ser descobertos e acessados por outros serviços. O UDDI utiliza vários padrões da indústria como *HTTP*, *XML*, *XSD*, *SOAP* e *WSDL*.

Conceitualmente, três tipos de informações podem ser armazenados no UDDI. Apesar da especificação não referenciar formalmente essas informações, elas oferecem uma visão geral sobre os dados que podem ser armazenados:

- **Páginas brancas:** provê identificadores e informações de contato básico sobre uma determinada empresa. Essas informações incluem nome do negócio, endereço, contato e números *DUNS* (*Data Universal Numbering System*). Essas informações permitem que usuários consigam encontrar *Web services* através de identificadores de negócio.

Figura 2.7: Interações da Arquitetura de Web Services - Adaptado de [Yu et al., 2008].

- **Páginas amarelas:** oferece informações que descrevem os *Web services* através de categorias ou taxonomias. Tais informações permitem que usuários descubram serviços por meio de categorias, por exemplo, serviços de reserva de hotel, venda de produtos esportivos e etc.
- **Páginas verdes:** descreve informações técnicas sobre as funcionalidades dos *Web services*. Elas incluem ponteiros para a localização do serviço.

2.3.4 Camada de Mensagem

A camada de mensagem é responsável por oferecer mecanismos de transmissão de dados pela rede. O principal protocolo utilizado, dentro do mundo dos *Web services*, é o *SOAP* [W3C, 2004a]. O protocolo *SOAP* oferece um mecanismo baseado em *XML* que pode ser utilizado para a troca estruturada de informações entre nós dentro de um ambiente descentralizado e distribuído [W3C, 2007a].

A especificação do protocolo *SOAP* é expressa como um *XML Infoset*⁴. Isso significa que o emissor da mensagem *SOAP* deve criar um *Infoset* que o receptor possa

⁴*XML Infoset* fornece um conjunto consistente de informações para uso em outras especificações que precisam referenciar informações em um documento *XML* bem formado W3C [2004b].

reconstruir. A maneira padrão para realizar essa tarefa é através da serialização utilizando a sintaxe da linguagem *XML*. No entanto, a especificação permite que outras representações de dados sejam utilizadas.

Principais Elementos

A mensagem *SOAP* é composta por um envelope *XML* que apresenta dois elementos principais chamados *Header* e *Body* que são elementos filhos do elemento envelope (elemento raiz). O conteúdo desses elementos é definido pela aplicação que utiliza o envelope *SOAP* e não por sua especificação. No entanto, a especificação define como deve ser a manipulação desses elementos [W3C, 2007a].

O *SOAP Header* é um mecanismo de extensão para a passagem de informações que não pertencem ao escopo da aplicação. Geralmente, informações contextuais ou diretivas para o processamento da mensagem são passadas como dados de controle dentro do elemento *Header* da mensagem. Isso permite que desenvolvedores criem suas próprias mensagens de acordo com as necessidades das suas aplicações. Os elementos filhos do elemento *Header* são chamados de blocos *Header* e representam um agrupamento lógico de dados que podem ser processados por nós intermediários durante o caminho da mensagem até seu receptor final (*Ultimate Receiver*). O *Header* é um elemento opcional no envelope *SOAP*.

Figura 2.8: Exemplo de um envelope SOAP de requisição.

O *Body* é um elemento obrigatório dentro da mensagem *SOAP*. Ele armazena o *payload* da mensagem. Esse *payload* contém as informações que são transmitidas pelo emissor ao último receptor da mensagem *SOAP*. O *Body* é formado pelos dados necessários para a execução do serviço pelo último receptor da mensagem [Weerawarana et al., 2005]. Esses dados são específicos para cada aplicação de acordo

com as regras de negócio que os serviços implementam.

As Figuras 2.8 e 2.9 apresentam um exemplo de interação entre um cliente e um *Web service* utilizando o *HTTP* como protocolo de transporte. No exemplo, uma requisição é enviada ao serviço. A requisição é representada pela Figura 2.8. Essa figura apresenta um envelope *SOAP* requisitando o serviço *GetPrice* que possui o parâmetro *Item* com o valor *Apples*. Por sua vez, a Figura 2.9 apresenta a resposta correspondente a requisição demonstrada na Figura 2.8. A construção *GetPriceResponse*, da mensagem de resposta, indica que o valor retornado pelo serviço (construção *Price*), diz respeito a execução do serviço *GetPrice*.

Figura 2.9: Exemplo de um envelope SOAP de resposta.

Tipos de Codificação

Um documento *WSDL* define todas as informações que são necessárias a um consumidor de serviço realizar uma requisição de um serviço. Como comentado na seção 2.3.2, o elemento *binding* define os detalhes concretos de um *portType*/*interface* associada a um protocolo de transporte. O elemento *binding* é responsável por apresentar o estilo de requisição que será utilizado no protocolo *SOAP*. Adicionalmente, ele define o tipo de codificação que será utilizado para compor o corpo da mensagem *SOAP*. O estilo de requisição pode ser *RPC* ou *Document*. O estilo *Document* indica que o corpo da mensagem contém um documento *XML* e que as partes das mensagens especificam os elementos *XML* que serão colocados nesse corpo. O estilo *RPC* indica que o corpo da mensagem contém uma representação *XML* da chamada do método e que as partes da mensagem representam os parâmetros desse método.

Com relação a tipos de codificação, existem dois tipos principais: *Encoded* e *Literal*. No tipo *Encoded*, as definições abstratas são traduzidas para um formato concreto utilizando as regras de codificação *SOAP*. No caso da codificação *Literal*, as definições de tipos abstratas tornam-se concretas por elas mesmas, ou seja, os tipos de dados definidos pelo *XSD* são colocados “literalmente” dentro do corpo da mensagem.

O protocolo *SOAP* oferece três estilos principais de composição estilo/codificação: *RPC/Encoded*, *RPC/Literal* e *Document/Literal*. Existem outros estilos de codificação que não são amplamente utilizados devido a divergências com os padrões da pilha arquitetural dos *Web services* [Cohen, 2003]. A Microsoft, por exemplo, criou o *DIME* (*Direct Internet Message Exchange*) para codificar dados binários enquanto a opção amplamente utilizada é o protocolo *SOAP* com *attachments* (*SwA* e *MTOM*).

O *RPC/Encoded* é o estilo de codificação que oferece a maior simplicidade. A idéia central é chamar um objeto remoto passando todos os parâmetros necessários para a execução de um determinado serviço. A pilha do protocolo *SOAP* serializa os parâmetros em linguagem *XML* e move os dados para o destino utilizando algum protocolo de transporte como o *HTTP* ou *SMTP*. Por sua vez, o receptor deserializa a mensagem em um objeto, e então, é realizado o processamento. Finalmente, o receptor envia uma mensagem contendo a resposta do processamento realizado pelo *Web service*. O estilo *SOAP RPC* define um conjunto de regras que devem ser seguidas na realização de uma chamada do tipo requisição-resposta. Ele exige que o elemento *Body*, da mensagem *SOAP*, contenha um elemento com o nome do método que será invocado seguido de elementos que contenham as entradas para cada parâmetro ou o valor de retorno, caso seja uma mensagem de resposta [Cohen, 2003].

O *RPC/Literal* oferece um modo de serialização para dados que já se encontram no formato *XML*. Os dados são serializados como um campo simples e enviados ao *Web service*. Como não existe uma hierarquia nesse campo, a pilha do protocolo *SOAP* serializa apenas um valor. Além disso, o *SOAP* negocia com o protocolo de transporte para associar a requisição a um *Web service* e manipula a resposta enviada por esse serviço [Cohen, 2003].

Por último, no *Document/Literal*, o protocolo *SOAP* envia um documento *XML* inteiro ao *Web service* sem sequer exigir um valor de retorno. Ele não define regras para a composição do conteúdo dentro do elemento *Body* da mensagem. Nesse caso, a mensagem pode conter qualquer tipo de dados *XML* que seja apropriado à execução do serviço. Além disso, os consumidores e provedores de serviço podem validar a estrutura do corpo da mensagem utilizando um *XSD*. No *Document/Literal*, o desenvolvedor da aplicação é responsável por manipular os protocolos de transporte (*HTTP*, *SMTP* e etc), serializar e deserializar o corpo do envelope *SOAP* (<*Body*>) e a extrair (*parsing*) os dados necessários no documento *XML* [Cohen, 2003].

Em termos de facilidades para o desenvolvedor, o *RPC/Encoded* oferece mais vantagens que trazem consigo algumas penalidades com relação à escalabilidade e desempenho. No estilo de codificação *SOAP RPC/Literal* o desenvolvedor fica mais envolvido com a manipulação dos arquivos *XML*, no entanto, o protocolo *SOAP* é sobrecarregado na negociação com os protocolos de transporte. O *Document/Literal* é o tipo de codificação com a maior complexidade para os desenvolvedores, mas que

possui a menor sobrecarga do protocolo SOAP [Cohen, 2003].

Diferenças entre Versões

O protocolo SOAP possui duas versões - as versões 1.1 e 1.2. A versão 1.2 realiza todas as funções oferecidas pela versão 1.1, além de adicionar outras. O SOAP 1.2 é baseado no Conjunto de Informações XML (XML Infoset)). O XML Infoset é utilizado para descrever um arquivo XML através de um XML Schema. Contudo, o tipo de serialização utilizada pelo Infoset não é necessariamente baseada no XML 1.0 como acontece com a versão 1.1. Essa forma de descrever arquivos XML pode utilizar outros formatos de serialização, como um formato binário. Por exemplo, pode-se utilizar um formato binário para compactar uma mensagem e excluir algumas *tags* que não sejam necessárias [W3C, 2003].

Outra característica importante do SOAP 1.2 é o fornecimento de um modelo de processamento claro que evita ambigüidades. Essas ambigüidades trazem consigo problemas de interoperabilidade. O XML Working Group encontrou mais de 400 questões não contempladas pelo SOAP 1.1 que traziam consigo problemas de interoperabilidade.

2.3.5 Demais Camadas

O trabalho apresentado nesta dissertação de mestrado tem como foco principal as camadas de descrição e mensagem. Todavia, as demais camadas da pilha de protocolos dos Web services, apresentada na Figura 2.4, são brevemente discutidas a seguir:

- **Camada de Confiabilidade:** a camada de confiabilidade é responsável por garantir que um processo de negócio seja completamente executado pelos seus participantes. Para cumprir essa tarefa é necessário um mecanismo que possibilite uma entrega confiável das mensagens trocadas entre clientes e provedores de serviço. A principal tecnologia envolvida nessa camada é o *WS-ReliableMessaging*.
- **Camada de Transações:** Transações é um conceito fundamental na construção de aplicações distribuídas confiáveis. Os Web services requerem um comportamento coordenado fornecido através de um mecanismo tradicional no controle das operações, e conseqüentemente, da aplicação. *WS-Coordination* em conjunto com suas extensões compõem a principal tecnologia dessa camada.
- **Camada de Segurança:** camada em que são definidas as especificações necessárias para oferecer mecanismos de comunicação confiáveis. Tais especificações

são apresentadas dentro dos padrões do WSS (*Web Service Security*) [OASIS, 2004].

- **Camada de Processo de Negócio:** especifica uma ordem de execução de um conjunto de *Web services*. Os *Web services* podem ser ordenados de diferentes formas para cumprir diferentes regras de negócio. Adicionalmente, essa camada é responsável por definir os dados compartilhados entre os serviços, os parceiros que estão envolvidos no processo e as relações existentes entre eles. A linguagem *BPEL (Business Process Execution Language)* é utilizada na especificação desses processos de negócio e as suas relações com os *Web services*.
- **Camada de Gerenciamento:** camada responsável por gerenciar os recursos e as configurações dos *Web services*. Ela gerencia o desempenho, a taxa de utilização, a disponibilidade e as configurações dos serviços dentro da arquitetura dos *Web services*. O *WS-Management* é responsável por oferecer os mecanismos de gerenciamento especificados nessa camada.
- **Camada de transporte:** especifica os protocolos de rede utilizados na troca de mensagens entre clientes e provedores de serviço. Dentre os principais protocolos destacam-se o *HTTP*, *SMTP*, *FTP*, entre outros.

2.4 Considerações Finais

Este capítulo apresentou os conceitos básicos relacionados à arquitetura orientada a serviços e aos *Web services*, mostrando seus padrões, técnicas e protocolos utilizados na construção dessa recente abordagem de desenvolvimento de sistemas distribuídos. Atualmente, os *Web services* têm sido foco de inúmeras pesquisas realizadas pela academia e pela indústria, a qual tem investido esforços na sua evolução apoiando as padronizações junto ao W3C. No entanto, há inúmeras questões que representam lacunas dentro dessa solução distribuída como a caracterização de carga de trabalho, segurança e qualidade de serviço.

O objetivo deste trabalho é contribuir para a caracterização da carga de trabalho em *Web services*. Nesse sentido, os capítulos 3 e 4 apresentam respectivamente uma revisão sobre modelos de cargas de trabalho em sistemas computacionais e as diferenças de se caracterizar cargas para *Web services*.

Avaliação de desempenho de WS: Cargas de Trabalho

Diversos trabalhos encontrados na *web* se propõem a avaliar o desempenho de *Web services*. Alguns desses trabalhos avaliam partes específicas de *Web service* [Cai et al., 2002], [Suzumura et al., 2005] e [Gómez-Martínez and Merseguer, 2006], enquanto outros provêm uma visão geral do comportamento dos *Web services* [Chen et al., 2006], [Chen et al., 2007] e [Estrella et al., 2008]. Nos casos em que se tem experimentação de modelos ou monitoração de *Web services*, observa-se que as cargas utilizadas na experimentação são definidas de forma arbitrária ou avaliam fatores específicos. Por exemplo, custos de serialização e deserialização das mensagens SOAP pelos *toolkits* de processamento disponibilizados pela indústria. Esses trabalhos não se baseiam em informações reais sobre os tipos de *Binding*, estilo de requisição/codificação das mensagens, tipos de dados mais utilizados e número de operações disponíveis pelo serviço. Essas características podem impactar no desempenho dos *Web services* do ponto de vista do consumidor ou do provedor de serviço, visto que ambos os componentes manipulam, de forma direta, as mensagens.

Na próxima seção são apresentados alguns trabalhos relacionados a avaliação de desempenho de *Web services* e, quando pertinente, comentado como é definida a carga de trabalho. Considerando que um *benchmark* pode ser uma carga de trabalho a ser imposta ao sistema em avaliação, a seção 3.2 apresenta alguns trabalhos relacionados à utilização de *benchmarks* na avaliação de desempenho para *Web services*.

A seção 3.3 discute, em termos gerais, como se definir carga de trabalho para um

sistema computacional ser avaliado e na seção 3.4 são apresentadas as considerações finais deste capítulo.

3.1 Avaliação de Web Services

Os *Web services* consideram um conjunto significativo de tecnologias, protocolos, padrões e linguagens que possibilitam o oferecimento de serviços acessíveis por meio da *internet*. Devido a diversificação das ferramentas empregadas, a tarefa de avaliar o desempenho de um *Web service* e oferecer um serviço de qualidade não é trivial. Diversos trabalhos têm tratado desse problema. Nesta seção, alguns deles são discutidos.

Em [Chen et al., 2006] é apresentada uma avaliação e modelagem de desempenho dos *Web services*. O objetivo principal do trabalho é prover uma visão geral do comportamento dos *Web services* no que diz respeito à desempenho. Além disso, o estudo oferece um modelo para estimar a latência do processo de requisição, por um consumidor, para o provedor de serviço. Esse trabalho avaliou quatro implementações de *Web services* (*toolkits* de processamento SOAP) verificando questões de desempenho do processo de serialização das mensagens. Essa avaliação ainda foi combinada com outros três fatores: tamanho das mensagens, estilos de requisição/codificação (*RPC-Document/Literal-Encoding*) e ambientes de teste (*LAN* e *WAN*). Outra avaliação realizada por esse estudo foi com relação a tecnologias alternativas ao protocolo SOAP. Foram avaliados os protocolos *MS-DCOM*, *MS .NET Remoting* e *MS Message Queue*. Esse trabalho concluiu que o protocolo SOAP é mais sensível em desempenho com o aumento ou diminuição dos tamanhos das mensagens. O SOAP é um protocolo interessante com relação as alternativas binárias para mensagens pequenas. Contudo, ele possui uma grande degradação de desempenho para mensagens grandes. Outra constatação importante foi com relação a estilo de requisição. O estilo *Document/Literal* possui melhor desempenho para todos os tamanhos de mensagens quando comparado com *Document/Encoded* e *RPC/Encoded*. Entretanto, os resultados apresentados são relativos a latência da entrega da mensagem. Os autores não deixam claro se foi considerado o processamento da mensagem SOAP para entregar os dados do serviço na forma de objeto em uma determinada linguagem de programação. Os testes realizados por Chen et al. [2006] foram implementados sem a definição de um critério mais elaborado para a construção das mensagens. A quantidade de parâmetros e tipos de dados pode influenciar no desempenho do processamento das mensagens.

Em Zilora and Ketha [2008] são apresentados vários aspectos da avaliação de desempenho em *Web services* através da caracterização dos fatores que influenciam o seu cenário atual de implementação. Esses fatores têm impacto no desempenho

final da troca de informações entre consumidores e provedores de serviço. Os fatores analisados são:

- Construção da requisição em linguagem nativa do consumidor
- Serialização da requisição de linguagem nativa para *XML* seguindo o formato do protocolo *SOAP* (influência das linguagens de programação).
- Transmissão da mensagem ao provedor de serviço utilizando o protocolo *TCP/IP* (influência do tamanho das mensagens).
- Recebimento da mensagem pelo provedor de serviço (influências das implementações de servidores de aplicação).
- Deserialização da mensagem *XML* para uma linguagem nativa pelo provedor de serviço (influência das linguagens de programação).
- Processamento da mensagem e envio da resposta. O tempo de processamento da requisição tem relação com linguagens de programação, poder computacional e lógica de negócio do serviço.

Uma das conclusões apresentadas nesse trabalho é que o custo da transmissão sobre o protocolo *TCP/IP* é pequeno quando comparado com os custos de serialização e deserialização das mensagens *SOAP*. Os autores constataam ainda que embora os *Web services* sejam apropriados para transmissão de mensagens pequenas, eles possuem melhor desempenho para mensagens grandes. Isso não quer dizer que as mensagens maiores possuem processamento mais rápido, e sim que os tempos de processamento dessas mensagens não possuem um aumento significativo se comparados ao processamento de várias mensagens pequenas. Esses resultados divergem dos apresentados por Chen et al. [2006]. No entanto, diferentes fatores foram avaliados nos dois trabalhos. Assim como no trabalho apresentado em [Chen et al., 2006], o trabalho de Zilora and Ketha [2008] realizou os testes de avaliação utilizando mensagens (cargas) criadas de forma arbitrária (variando os tamanhos das mensagens). Os autores não consideraram ainda a influência dos tipos de requisição (apresentados em [Chen et al., 2006]) nas requisições dos serviços.

Em [Li et al., 2007] é apresentado um estudo exploratório dos *Web services* na internet, onde foram investigados quatro aspectos principais. Dentre esses aspectos destacam-se: número, complexidade, qualidade de descrição e diversidade de funcionalidades. O objetivo do trabalho foi inspecionar o estado atual dos *Web services* através da coleta de dados. Foram utilizados vários sites de publicação disponíveis na *Web* para a obtenção dos arquivos *WSDL*, e também, a *Google Search SOAP API*. O trabalho apresentou resultados que julgaram a complexidade dos

Web services através de análises dos tamanhos dos arquivos *WSDL*, do número de operações oferecidas, da qualidade de descrição, da diversidade de funcionalidades, dentre outras características. O trabalho concluiu que a maioria dos *Web services* na internet não são complexos, a qualidade de descrição é inadequada para a maioria dos *Web services* e os domínios das aplicações oferecidas possuem um escopo limitado. Uma importante contribuição desse trabalho, além de apresentar o estado atual dos *Web services*, foi a forma de investigação dos arquivos *WSDL* utilizando a *Google SOAP Search API*. Contudo, a proposta de avaliação apresentada por Li et al. [2007] não considerou os detalhes relativos a construção das mensagens definidas pelo elemento *binding* do documento *WSDL*. Adicionalmente, aspectos relativos a ocorrência dos tipos de dados nos parâmetros de entrada e saída, também não foram considerados.

Em [Estrella et al., 2008] são apresentados algumas diretrizes para a avaliação de desempenho em *Web services*. O trabalho mapeia alguns pontos importantes que devem ser considerados no processo de avaliação de desempenho especificamente para o contexto dos *Web services*. Os autores mostram que problemas com as especificações dos padrões utilizados pelos *Web services* influenciam negativamente na provisão de serviços com qualidade. A linguagem *WSDL*, por exemplo, define um serviço visando apenas os aspectos funcionais sem considerar questões sobre critérios de qualidade de serviço. Por sua vez, o protocolo *SOAP* (utilizado para acessar os serviços) é representado em formato *XML*, e portando, carrega consigo todos os problemas relativos a custos da extração das informações e falta de otimização. Outro problema apresentado é com relação ao canal de comunicação. Os *Web services*, por utilizar a internet como canal de comunicação, devem considerar também questões sobre latência, congestionamento e garantias de entrega. Além disso, esse trabalho apresenta uma arquitetura que tem como objetivo uma melhor provisão de qualidade de serviço entre as mensagens trocadas pelas aplicações *Web*. Um dos componentes principais dessa arquitetura é o *Broker*. Esse componente tem como funcionalidade avaliar diversos critérios de *QoS* que permitem indicar qual *Web service* possui o melhor atendimento para uma determinada requisição de um consumidor de serviço. Um desses critérios é a carga de trabalho imposta ao provedor do serviço. Esse trabalho mostra, dentre outros fatores, a importância de se conhecer as cargas de trabalho para a provisão de *Web services* com desempenho elevado.

Como pode ser observado, um ponto a ser investigado e que não é tratado com profundidade nos trabalhos citados é como definir a carga de trabalho para *Web services*. Uma forma de verificar como essa carga pode ser considerada é analisando como os *benchmarks* para *Web services* trabalham, uma vez que, em geral, esses *benchmarks* são utilizados para submeter uma carga a um provedor de serviços.

3.2 Benchmarks para Web Services

Um dos *benchmarks* mais completos para *Web services* é o *@Bench* que possui como principal objetivo a verificação de quão fácil é para os servidores de aplicação e ferramentas correspondentes a exposição da lógica de negócio oferecida pelo *Web service* [García et al., 2006] e [Doculabs, 2003]. Além disso, esse *benchmark* ainda verifica as capacidades de execução e escalabilidade do *Web service* frente a um grande número de requisições de clientes de forma concorrente. Esse aspecto é verificado através de testes de desempenho e escalabilidade. O *@Bench* provê três serviços: *GetOrderDetails*, *GetCostumer*, *NewCostumer*. Os usuários requisitam os três serviços com a mesma probabilidade. O tempo entre duas requisições seguidas é definido por um valor aleatório que varia entre 2 a 8 segundos. A funcionalidade principal desse *benchmark* é modelar as requisições entre usuários e servidores de uma mesma companhia, configurando um modelo *C2B* (*Costumer to Business*) [Doculabs, 2003]. Uma deficiência deste *benchmark* é não possibilitar a modelagem de interações entre servidores de diferentes companhias, impedindo a configuração de uma aplicação *B2B* (*Business to Business*). Outro ponto negativo, com relação à caracterização da carga de trabalho, é a falta de uma metodologia que implemente uma submissão de uma carga mais específica. Além disso, o *benchmark @Bench* é regido com relação a construção das mensagens que são enviadas ao provedor de serviço. Ele não oferece uma flexibilidade para a realização de testes com diferentes composições de mensagens.

Em [García et al., 2006] é apresentado uma avaliação da plataforma *J2EE* e o *framework .NET* utilizando um *benchmark* chamado *TPC-App* definido pelo *Transaction Processing Performance Council*, uma entidade responsável por definir *benchmarks* que avaliam processos transacionais e banco de dados com intuito de liberar resultados confiáveis para a indústria [TPC, 2009]. O *TPC-App* emula as atividades de um servidor de aplicação transacional *B2B* com o intuito de obter uma indicação da capacidade de desempenho do sistema servidor [García et al., 2006]. A arquitetura do *benchmark* apresenta três elementos principais: o *SUT* (*System Under Test*), o *RBE* (*Remote Emulator Business*) e o emulador externo. O elemento *SUT* expõe sete métodos remotos para que o *RBE* emule um processo de negócio e oferece métodos como *CreateOrder*, *NewCostumer* e *ChangePaymentMethod*. O emulador de processo remoto, por sua vez, considera um processo com várias linhas de execução (*multithreaded*), onde cada uma das *threads* realiza requisições aos *Web services* emulando um processo de negócio. O trabalho de García et al. [2006] apresenta duas implementações de *Web services*, uma para o *.NET* e outra para o *J2EE*, sendo que ambas as implementações contêm o mesmo projeto básico seguindo uma arquitetura de três camadas. A carga de trabalho utilizada nos testes estressa o servidor de

aplicação por intermédio de chamadas aos *Web services* descritos anteriormente. A principal métrica fornecida pelo *benchmark TCP-App* é a vazão (*throughput*) do servidor de aplicação que é medida em interações por segundo. Cada um dos *Web services* possui uma determinada probabilidade de ser requisitado sendo o `CreateOrder` com a maior possibilidade de ser executado (probabilidade 50%). Segundo esse trabalho, o *framework .NET* possui uma vantagem sobre a plataforma J2EE. Esse trabalho mostrou de forma prática a utilização do *TCP-App*, no entanto, não foram apresentadas maneiras mais realistas de submissões de cargas de trabalho. A carga imposta pelo *benchmark* oferece apenas um teste de estresse ao servidor e a sua especificação não apresenta nenhum estudo mais detalhado que defina o comportamento das requisições realizadas ao *Web service*, ou seja, apenas define para cada *Web service* uma probabilidade de ser requisitado.

Anterior ao trabalho de García et al. [2006], no ano de 2004, houve uma divergência entre a *SUN Microsystems* e a *Microsoft* com relação a análise de desempenho das plataforma J2EE e *framework .NET*. Pesquisadores da *SUN* publicaram um artigo denominado “*Web Service Performance: Comparing Java 2 Enterprise Edition and .NET Framework*” que discute vários casos de testes utilizando o *benchmark WSTEST* [WSTest, 2009]. Esse *benchmark* é desenvolvido pela *SUN* para medir o desempenho de vários tipos de chamadas de *Web services* [Sun, 2004]. Foram realizados os seguintes tipos de chamada:

- `echoVoid` - Envia e recebe uma mensagem vazia. Essa chamada, segundo a *SUN*, verifica a infra-estrutura dos *Web services*.
- `echoStruct` - Envia e recebe um vetor de 20 elementos onde cada um dos elementos são compostos por um valor `Integer`, um `Float` e uma `String`.
- `echoList` - Envia e recebe uma lista encadeada de 20 elementos onde cada um dos elementos possui uma estrutura que foi definida no `echoStruct`.
- `echoSynthetic` - Envia e recebe múltiplos parâmetros de diferente tipos como `String`, `Struct` e vetor de *bytes* de um 1K.

O *WSTEST* simula um programa servidor *multi-threaded* que realiza múltiplas chamadas aos *Web services* de forma paralela. A simulação mede a vazão do sistema (média do número de *Web services* executados por segundo) e o tempo de resposta (tempo médio levado para processar a requisição). Os testes executados pela *SUN* apresentaram um melhor desempenho da plataforma J2EE em relação ao *framework .NET*. No entanto, um mês após da publicação do artigo pela *SUN*, a *Microsoft* publicou um outro artigo rebatendo os resultados obtidos pelos pesquisadores da *SUN*. O artigo denominado “*Web Service Performance: Comparing Java 2 Enterprise Edition*

and .NET Framework - A Response to SUN Microsystem's benchmark" alegou que a SUN não disponibilizou os códigos fontes para que outras empresas pudessem verificar, por elas mesmas, os resultados a partir da recriação dos testes e apresentou uma outra metodologia de testes. A nova metodologia de testes utilizada pela *Microsoft* propôs refazer os testes realizados pela SUN e criou alguns outros testes, mais complexos, que segundo a *Microsoft* representava uma forma mais realista de analisar o desempenho dos *Web services* [Microsoft, 2004]. Desta vez, os resultados favoreceram o *framework .NET* que teve resultados de duas a três vezes melhores que os da plataforma *J2EE*. Com relação à caracterização das cargas de trabalho submetidas nos testes apresentados pela SUN e *Microsoft*, pode-se dizer que não houve um estudo mais profundo com relação a este aspecto. Ambas as empresas não realizaram um estudo estatístico (um modelo) que representasse de forma realista as submissões realizadas nos testes. Uma outra deficiência é em relação às construções das mensagens considerando os estilos de requisição e tipos de codificação utilizados. Além disso, trabalhos como o apresentado por [Head et al., 2005] medem a influência dos tipos de dados nos custos de empacotamento e desempacotamento das mensagens.

Um *benchmark* interessante é apresentado em [Head et al., 2005]. Nesse trabalho, são avaliados vários *toolkits SOAP* (*AxisJava 1.2*, *XSUL*, *GSOAP* e *BSOAP*), que são utilizados para empacotar as mensagens que são trocadas entre um cliente e o seu provedor de serviço. O trabalho enfatiza o desempenho desses *toolkits* em processar o empacotamento dos tipos de dados em um envelope SOAP. Além disso, são discutidas algumas técnicas das quais os *toolkits* poderiam fazer uso para efetuar otimizações para determinadas estruturas de dados, casos de uso e padrões de comunicação. No trabalho foram realizados testes que contabilizavam os custos dos processos de empacotamento e desempacotamento de alguns tipos de dados dentro do envelope SOAP. Nesse contexto, comparou-se o desempenho e a eficácia de cada um das implementações. Contudo, alguns dos *toolkits* analisados já possuem versões atualizadas cujo desempenho pode ter sido elevado, como é o caso do *Apache Axis2* [Apache, 2008]. De maneira complementar, os resultados apresentados nesta dissertação definem a composição das mensagens SOAP considerando, dentre outras características, quais os tipos de dados mais utilizados em aplicações reais. Assim sabendo-se o custo de empacotamento e desempacotamento dos tipos de dados aliados a probabilidade deles ocorrerem é possível uma análise mais fidedigna das características das cargas de trabalho dos *Web services*.

Os *benchmarks* analisados nesta seção submetem uma carga de trabalho que, em geral, tem o objetivo de sobrecarregar os provedores de serviço. Como pode ser observado, a maior parte dos exemplos tratados utiliza uma carga definida de forma arbitrária ou avaliam fatores específicos como poder de atendimento de

um determinada regra de negócio ou capacidade de um determinada linguagem de programação implementar serviço com maior desempenho. A próxima seção apresenta, em termos gerais, como uma carga de trabalho deve ser definida.

3.3 Cargas de Trabalho

A caracterização de cargas de trabalho de sistemas computacionais é bastante discutida na literatura especializada. Muitos desses trabalhos têm como objetivo principal a caracterização de cargas de trabalho para servidores *Web* através da utilização de *traces* [Arlitt and Williamson, 1996] [Weber and Hariharan, 2003] [Williams et al., 2005] [Gill et al., 2007]. Um exemplo de trabalho seguindo essa linha de pesquisa é o trabalho de mestrado apresentado por Silva [2006]. O pesquisador mostrou uma caracterização de cargas de trabalho para testes de modelos de servidores *Web*. Diversas categorias de páginas *Web* foram analisadas visando aspectos como intervalo de chegada de requisições, código de resposta, tipo e tamanho de objetos. A partir da análise realizada, modelos estatísticos que representassem a carga de trabalho em estudo puderam ser criados.

3.3.1 Caracterização de Cargas de Trabalho

A primeira medida a ser considerada na caracterização de uma carga de trabalho é a identificação do componente básico que é determinado pela natureza do sistema [Menascé and Almeida, 2002]. Um componente básico pode ser um *job*, uma transação, um processo ou uma requisição *HTTP*. No caso dos *Web services*, o componente básico pode ser definido pelas mensagens trocadas entre um cliente e o provedor do serviço. O processo de caracterização de carga de trabalho deve responder algumas questões importantes:

1. Como se caracteriza a carga de trabalho do objeto em questão? A análise que deve ser feita nesta questão envolve definir a carga de trabalho que é imposta ao objeto analisado. Por exemplo, a carga submetida a um microcomputador cliente de um sistema distribuído constitui uma seqüência de “cliques” realizados por um usuário. Do lado do servidor, a carga de trabalho é definida por um conjunto de requisições que é recebida por um período determinado de monitoração. Uma outra análise poderia ser com relação à rede onde a caracterização depende de aspectos relacionados ao comportamento de tráfego, distribuição do tamanho e o tempo de chegada entre os pacotes [Menascé and Almeida, 2002].
2. Qual o nível da descrição da carga de trabalho? A descrição da carga pode ser dividida em três níveis distintos:

- **Nível comercial:** caracteriza a carga de trabalho do ponto de vista do usuário.
- **Nível funcional:** caracteriza a carga de acordo com as aplicações e protocolos que são usados na composição do sistema.
- **Nível físico:** caracteriza a carga considerando o consumo dos recursos do sistema.

As cargas de trabalho, após serem modeladas, podem ser utilizadas em diversas atividades como seleção de sistemas computacionais, ajuste de desempenho e planejamento de capacidade [Menascé and Almeida, 2002].

3.3.2 Modelo de Carga de Trabalho

O desempenho de sistemas computacionais depende das características das cargas de trabalho a eles submetidas. Eles devem se adaptar e auto-configurar com as mudanças das intensidades das cargas de trabalho [Menascé et al., 2007]. A avaliação desses sistemas depende da representatividade dessas cargas para que se consiga produzir resultados confiáveis. Isso pode ser alcançado pela coleta de dados reais que permitem a criação de modelos estatísticos que representam as principais características do sistema analisado [Feitelson, 2002]. O principal objetivo da modelagem de cargas de trabalho é fazer com que o modelo imite a carga de trabalho real. Após a modelagem, um conjunto de programas, com o objetivo de testar artificialmente um sistema dentro de um ambiente controlado, pode ser implementado baseando-se no modelo criado [Menascé and Almeida, 2002].

3.3.3 Busca por Invariantes

Em [Arlitt and Williamson, 1996] e [Williams et al., 2005] é definido o termo invariante como sendo uma verdade universal, que se aplica à caracterização de carga de trabalho imposta a um sistema. Os estudos apresentados por esses autores referem-se à caracterização de cargas de trabalho para servidores *Web*. Diversos fatores são analisados nesse trabalho como tipos de arquivos, distribuição do tamanho e tipos de comportamento das referências para determinados tipos de documentos.

O trabalho apresentado em [Arlitt and Williamson, 1996] foi repetido em [Williams et al., 2005] onde foram verificadas se as invariantes encontradas em 1996 ainda faziam sentido após 10 anos. O resultado mostrou que apesar das mudanças nas tecnologias *Web* como novos protocolos, linguagens de *script*, uso de *caches* nos servidores, a maioria das invariantes encontradas no trabalho anterior continuavam verdadeiras em 2005. A Tabela 3.1 apresenta um quadro que mostra os dados obtidos

por Williams et al. [2005] e o estado das invariantes em relação ao trabalho realizado por Arlitt and Williamson [1996].

Tabela 3.1: Invariantes dos Servidores Web.

Invariantes	[Williams et al., 2005]	Estado
Taxa de Sucesso	65–70% das requisições ao servidor Web tiveram sucesso na transferência de um documento	Menor
Tipos de arquivos	Arquivos HTML e de imagens representam 70-85% dos documentos transferidos pelo servidores Web	Menor
Tamanho das transferências	Tamanho médio da transferência é pequeno ($\leq 5\text{KB}$)	Igual
Requisições distintas	Uma pequena fração menor, aproximadamente 1% das requisições foi para documentos distintos.	Igual
Referências únicas	Uma porcentagem significativa dos arquivos (15-26%) e bytes (6-21%) foram acessados uma única vez no log.	Igual
Distribuição de tamanhos	As distribuições de tamanhos de arquivos e de transferência são de cauda pesada (heavy-tailed). Seguindo o modelo de Pareto com $\alpha \approx 1$	Igual
Concentração	10% dos arquivos acessados correspondem a 80-90% das requisições ao servidor e 80-90% dos bytes transferidos	Igual
Tempos entre referências	Os tempos entre sucessivas referências ao mesmo arquivo são distribuídos exponencialmente e independentemente	Igual
Requisições remotas	Sites remotos representam mais de 70% ou mais dos acessos ao servidor e mais de 80% ou mais dos bytes transferidos	Igual
Uso da WAN	Não houve análise, mas em 1996 10% dos domínios deram origem a mais de 75% dos acessos	Não estudado

3.3.4 Fonte de Dados

A obtenção de dados para serem utilizados em uma modelagem de cargas de trabalho pode ser feita de duas formas: usar dados disponíveis em um determinado lugar ou coletá-los especificamente para a construção do modelo [Feitelson, 2002].

Uma das formas de se obter dados para realizar uma modelagem é através de *traces*. Eles são mantidos por alguns sistemas para permitir a realização de auditorias e manter formas de monitoração de comportamento. Em servidores *Web*, por exemplo, são mantidos *traces* com informações sobre todas as requisições recebidas por ele

em um determinado intervalo de tempo [Menascé and Almeida, 2002]. Há na literatura diversos trabalhos que fizeram uso de *traces* para o estudo de sistemas computacionais como os apresentados por [Gill et al., 2007] [Silva, 2006], [Feitelson and Nitzberg, 1995]. As análises de *traces* podem fornecer distribuições estatísticas de vários componentes de um sistema possibilitando, por exemplo, encontrar padrões de chegada de requisições de um servidor *Web*, tamanho de *jobs* submetidos a um *cluster*, e no caso de *Web services*, fornecer o tamanho dos documentos *XML* trocados entre as aplicações. No entanto, os *traces* nem sempre possuem níveis de detalhes suficientes para a caracterização de uma carga, ou seja, eles podem fornecer, por exemplo, informações sobre as requisições recebidas por um servidor *Web* e suprimir informações sobre o nível dos pacotes. Em [Yoo et al., 2006] é apresentado um trabalho interessante envolvendo a modelagem de cargas de trabalho através de redes neurais artificiais. Os autores utilizaram *traces* comerciais de *Web services* para validar a utilização das redes neurais no processo de caracterização de carga. Todavia, a caracterização foi realizada para um caso de teste específico. A carga de trabalho não foi generalizada para outros casos como a caracterização de carga de trabalho para servidores *Web* apresentada por Williams et al. [2005].

3.3.5 Filtragem dos dados

Após a coleta dos dados é importante filtrá-los antes da criação do modelo da carga de trabalho. Isso deve ser feito porque os dados podem conter eventos incomuns que não fazem sentido para a caracterização da carga [Feitelson, 2002]. A decisão de incluir essa característica incomum como parte importante da caracterização é subjetiva porque de fato ela existe. No entanto, há uma dificuldade de se explicar tais características, e deixá-las na carga de trabalho implica correr o risco de incluir uma característica que ocorre com certa atipicidade [Feitelson, 2002].

3.4 Considerações Finais

Entender as cargas de trabalhos é uma tarefa importante em qualquer estudo de avaliação de desempenho. Muitos trabalhos encontrados na literatura realizam avaliação de desempenho considerando um conjunto de fatores tais como custo de serialização e deserialização, linguagens de programação, protocolos de transporte, servidores de aplicação, *toolkits* de processamento *SOAP*, latência de rede e tamanhos de mensagens *SOAP*. Além dessas características, alguns *benchmarks* oferecem ainda a funcionalidade para a execução de processos de negócios. Apesar desses processos serem muito específicos ao tipo de aplicação provida pelo *Web service*, alguns desses *benchmarks* fornecem um processo de negócio padrão que realiza interações com bancos de dados e acesso a arquivos e outros serviços. Contudo, os trabalhos

apresentados não consideram questões que são relacionadas ao tipo de carga de trabalho no que diz respeito a construção das mensagens trocadas na interação entre consumidores e provedores de serviço. O Capítulo 4 desta dissertação apresenta um estudo sobre os problemas relacionados às cargas de trabalho para *Web services*.

Caracterização de Carga de Trabalho de WS

O Capítulo 3 desta dissertação apresenta uma metodologia para caracterização de cargas de trabalho utilizando, como exemplo, cargas para servidores *Web*. A maioria dos casos estudados baseia-se em traces para realizar esse estudo de caracterização. Contudo, metodologias para a caracterização de cargas de trabalho baseadas em traces não puderam ser aplicadas neste trabalho. A dificuldade de encontrar tais *traces*, dentro do contexto dos *Web services*, impossibilitou a sua utilização. Portanto, uma proposta alternativa de caracterização de carga trabalho é considerada. Neste trabalho, a caracterização da carga de trabalho tem como foco principal caracterizar as mensagens trocadas entre clientes e provedores de serviço através de dados coletados na *Web*. O nível funcional da carga de trabalho é abordado definindo as características dos *Web services* através de uma análise de documentos *WSDL*, explicados na seção 2.3.2 desta dissertação.

4.1 Modelo de Carga de Trabalho para WS

Em se tratando de *Web services*, os trabalhos de caracterização de cargas são complexos devido à maneira como a sua arquitetura é composta. Muitos fatores influenciam as cargas, tais como:

- Linguagens de programação;
- *Toolkits* de processamento *SOAP*;

- Protocolos de transporte;
- Latência da rede;
- Custos de empacotamento e desempacotamento das mensagens;
- *Parsing* e interpretação dos dados *XML* em objetos de uma determinada linguagem e vice-versa;

Diversos trabalhos abordando esses fatores são encontrados na literatura [Zilora and Ketha, 2008], [Chen et al., 2007], [Chen et al., 2006], [Gómez-Martínez and Merseguer, 2006] e [Suzumura et al., 2005]. Entretanto, existe uma lacuna no que se refere a utilizar os documentos *WSDL* como fonte de informações para caracterizar carga de trabalho. O objetivo dessa caracterização é encontrar informações baseadas nas peculiaridades da construção das mensagens utilizadas na execução do serviço. Além disso, tais documentos ainda podem prover informações adicionais que são apresentadas na seção 4.1.1. A Figura 4.1 mostra em qual contexto este trabalho está inserido. Os resultados dessa caracterização, apresentados no Capítulo 6, podem ser utilizados para a construção de serviços e consumidores de serviço com um enfoque na avaliação de desempenho.

Figura 4.1: Lacuna na avaliação de desempenho de Web services.

Alguns trabalhos de avaliação de desempenho em *Web services* abstraem a execução do serviço. Isso acontece devido à especificidade dos fluxos de negócio

de cada serviço. Por exemplo, a execução de um serviço de reserva de passagens aéreas pode ser diferente, em consumo de recursos computacionais, que um serviço de cartão de crédito. Um serviço pode ter diferentes comportamentos de entrada e saída, acesso a banco de dados e trocas de mensagens com outros serviços. Todas essas variáveis dependem de como foi definida a regra de negócio que o serviço implementa. Nesses trabalhos são avaliados, principalmente, os custo de processamento para serializar e deserializar mensagens, bem como a sobrecarga dos protocolos de transporte. Por outro lado, há trabalhos que consideram uma execução padrão, utilizando *benchmarks* que possuem uma determinada regra de negócio implementada [García et al., 2006]. O problema com esses trabalhos é com relação à generalização das funcionalidades do serviço. Os *Web services* possuem uma variedade de funcionalidades, apresentada na Figura 4.2, que acabam não sendo contempladas por esses *benchmarks*. Este trabalho visa à provisão de informações para avaliação de desempenho de *Web services* sem considerar aspectos de consumo de recursos computacionais. A análise de documentos *WSDL* não possibilita extrair informações que permitam inferir a maneira como o serviço está implementado, mas sim a forma como é feita a interação entre consumidores e provedores de serviço.

Figura 4.2: Variedade de Funcionalidades dos Web Services [Li et al., 2007].

O trabalho apresentado por Chen et al. [2006] mostra uma avaliação e modelagem de desempenho dos *Web services*. Nesse trabalho, são realizados testes comparando diferentes tecnologias para avaliar o desempenho dos *Web services*. Essa avaliação é realizada através da combinação de várias implementações de *Web services* com diversos tamanhos de mensagens que são trocadas na execução do serviço. Os autores do trabalho constataram que o tamanho, bem como a complexidade das mensagens, influencia no desempenho do provedor que fornece o serviço. Entretanto, a construção dessas mensagens é definida de maneira arbitrária, sem apresentar um

Tabela 4.1: Invariantes dos Web Services

Invariantes	Descrição
Número de Operações	Número de Operações disponíveis pelo <i>Web service</i> . Esse número pode indicar informações de quantidade de requisições recebidas pelo provedor do serviço.
Tipos de Binding	Basicamente, existem três tipos mais utilizados de <i>Binding</i> para acessar o <i>Web service</i> . Existem duas versões de Binding SOAP (1.1 e 1.2) e o <i>Binding HTTP</i> que pode ser definido com os métodos <i>GET</i> ou <i>POST</i> .
Tipos de Codificação	Para o <i>Binding SOAP</i> existem três combinações <i>style/use</i> de requisição (<i>RPC/Encoded</i> , <i>RPC/Literal</i> e <i>Document/Literal</i>). Essas combinações especificam como será a construção da mensagem SOAP no que diz respeito a estilo de requisição e uso de codificação.
Tipos de Dados	Oferecem informações sobre a ocorrência de cada um dos tipos de dados especificados pelo <i>XML Schema</i> [W3C, 2004c].

estudo mais aprofundado de como essas mensagens deveriam ser compostas. O foco principal deste trabalho é apresentar um estudo de caracterização das principais invariantes que influenciam o comportamento das cargas de trabalho dos *Web services* visando principalmente a construção dessas mensagens.

4.1.1 Invariantes na Carga de Trabalho para WS

As invariantes encontradas por Arlitt and Williamson [1996], dentro do contexto dos servidores *Web*, não se aplicam aos *Web services*. A diferença principal é que essas invariantes foram definidas a partir de *traces* de alguns servidores *Web* de algumas universidades. Como dito anteriormente, a dificuldade de se encontrar *traces* de provedores de *Web services* impossibilitou utilizar essa metodologia. Portanto, a metodologia utilizada para a realização deste trabalho focou em uma caracterização estática a partir de documentos *WSDL* encontrados na *Web*. Os documentos *WSDL* oferecem informações de acesso e de construção das mensagens que são utilizadas na execução do serviço. Assim, foram encontradas algumas invariantes que estão descritas na Tabela 4.1.

4.1.2 Fonte de dados

A dificuldade de se encontrar *logs*, além da sua desvantagem de supressão de informações, provocou a utilização de uma metodologia de análise estática, baseada em documentos *WSDL*. O trabalho apresentado nesta dissertação utilizou a *Web* como fonte de dados na coleta de documentos *WSDL*. Em Li et al. [2007] é apresentado um estudo exploratório do status dos *Web services* na *Web*, utilizando arquivos *WSDL*, para analisar aspectos como número, complexidade, qualidade de descrição e diversidade funcional dos *Web services*. Contudo, o presente trabalho utiliza os documentos *WSDL* com um outro enfoque. A idéia central gira em torno da utilização

das informações obtidas nesses documentos para caracterizar, de maneira estática, as mensagens trocadas na interação entre consumidores e provedores de serviço.

4.1.3 Filtragem dos Dados

A coleta de dados através da *Web* tem como empecilho a obtenção de dados replicados e dados que não correspondem àqueles que se espera. No caso da coleta de documentos *WSDL*, foram utilizados dois mecanismos de busca para obter a maior quantidade possível desses documentos. Esses mecanismos basearam-se na obtenção de documentos *WSDL* através de *engines* de busca como o *Google* e por sites especializados em prover *Web services*. Os principais detalhes sobre tais mecanismos de busca são discutidos no Capítulo 5. Em suma, um mecanismo pode obter um arquivo que fora baixado anteriormente por outro mecanismo, criando assim réplicas dos documentos. Outro problema é com relação à obtenção de arquivos *WSDL* que na verdade não possuem o conteúdo correspondente. Além disso, os documentos devem ser validados para o processo de análise. Todos esses problemas devem ser solucionados através do processo de filtragem para que análise seja realizada de forma correta e coerente.

4.1.4 Modelagem da Carga de Trabalho

O principal objetivo desse trabalho consiste na caracterização das informações contidas no documento *WSDL* que têm influência na carga de trabalho. Como comentado anteriormente, muitos trabalhos utilizam *traces* para a realização de trabalhos com foco em cargas de trabalho. Esses *traces* podem ser utilizados de duas formas distintas: (1) usar uma carga de trabalho obtida de um *trace* para ser submetida diretamente a uma simulação [Menascé and Almeida, 2002] e [Feitelson, 2002], (2) criar um modelo de carga através de um *trace* para ser utilizado em uma análise ou em uma simulação. Contudo, como não foi possível a utilização de *traces* no contexto dos *Web services* foi proposto um modelo de carga baseado em documentos *WSDL*. Esse modelo consiste de um sumário estatístico das informações extraídas dos documentos *WSDL*. Há várias vantagens em se criar um modelo de cargas de trabalho:

- Possibilidade de mudar os parâmetros com intuito de analisá-los individualmente e analisar a influência de cada um, enquanto alguns são mantidos constantes. Isso permite medir diretamente a sensibilidade do sistema com diferentes valores de parâmetros.
- Com o uso do modelo é possível repetir os experimentos sobre condições estatísticas semelhantes, mas não idênticas. Por exemplo, uma simulação pode

ser executada várias vezes com diferentes sementes, geradas aleatoriamente.

- A modelagem aumenta o entendimento das cargas de trabalho e pode conduzir novos projetos de análise de desempenho baseados nesse entendimento.

O principal problema com os modelos é a falta de representatividade, ou seja, o nível da modelagem representa o que sistema encontrará na prática? A resposta depende do nível de detalhes que é incluído no modelo. Por exemplo, um *Web service* é composto por várias instruções e elas podem interagir com o computador em vários níveis. Uma opção seria modelar esses níveis explicitamente, criando uma hierarquia. O grande problema dessa proposta é que quanto mais níveis são adicionados mais complexidade é incluída no modelo.

A questão sobre o que é necessário modelar e qual o nível de detalhes que a modelagem deve possuir é muito difícil. Podem-se caracterizar completamente todos os atributos importantes, ou então, criar um modelo mais econômico e fácil de gerenciar com um número reduzido de parâmetros. Na modelagem de cargas de trabalho, alguns aspectos devem ser considerados com bastante atenção para que a modelo possa representar, fielmente, a carga real [Feitelson, 2002], [Menascé and Almeida, 2002]:

- **Ajuste de distribuição:** o objetivo de se construir um modelo é criar uma carga de trabalho sintética que imite a carga real. O modelo da carga de trabalho é composto por uma distribuição ou uma coleção de distribuições dos vários atributos da carga. Assim, através de amostras das distribuições é possível criar o modelo da carga. Uma distribuição adequada pode ser obtida através de momentos, pelo significado e pela variância das amostras dos dados. Por exemplo, as amostras de documentos *WSDL* podem apresentar uma grande dispersão do tamanho das mensagens trocadas entre as aplicações, conduzindo assim a carga a um modelo exponencial.
- **Correlações:** a modelagem de cada atributo da carga de trabalho de forma isolada não é suficiente. É importante correlacionar diferentes atributos, pois essa correlação pode apresentar algum comportamento importante do sistema. Por exemplo, no escalonamento de *jobs* em uma máquina com vários processadores o que se conhece é a quantidade necessária de processadores para executar aquele determinado *job*, mas o tempo de execução é desconhecido. Encontrar uma correlação entre tamanho dos *jobs* e tempo de execução pode conduzir a escolha de uma política de escalonamento que se ajuste melhor ao comportamento encontrado pela correlação. Dentro do contexto dos *Web services*, pode ser realizada uma investigação, por exemplo, entre os tipos do

estilo de requisição de uma operação com o tipo de codificação utilizado para compor a mensagem SOAP.

- **Correlação não fixa:** a correlação não fixa é relacionada com o tempo. O comportamento da carga de trabalho muda de acordo com a hora do dia ou com o passar do tempo. Isso pode ocorrer devido a vários fatores: aumento da popularidade do sistema (mais usuários acessando o sistema) e o comportamento dos usuários que submetem diferentes cargas de acordo com o horário do dia, etc. A correlação não fixa é utilizada quando se tem *traces* com informações de tempo. No caso deste trabalho, ela não foi utilizada por se tratar de uma abordagem diferente das caracterizações de cargas tradicionais.

4.2 Considerações Finais

A modelagem de cargas de trabalho não representa uma atividade trivial devido ao número de fatores que devem ser considerados. Encontrar relações entre atributos da carga de trabalho, ajustar as distribuições dos atributos no modelo e identificar o componente básico da carga são tarefas que devem ser desenvolvidas com muito cuidado, pois uma caracterização ruim pode levar a uma análise de desempenho não condizente com a carga de trabalho real. A proposta inicial deste projeto de mestrado visa principalmente a provisão das características das cargas de trabalho de *Web services* para a avaliação de desempenho desse tipo de sistema. O Capítulo 5 discute uma ferramenta para análise de documentos WSDL e as suas funcionalidades que permitem a construção de um modelo estatístico.

WSDLAnalyzer

A *WSDLAnalyzer* é uma ferramenta para a busca e análise de documentos *WSDL*, que apresenta, como principal funcionalidade, mecanismos para extração dos principais aspectos desses documentos. Os documentos *WSDL* podem ser utilizados para caracterizar, dentre outras informações, a construção das mensagens trocadas entre os componentes (consumidor e provedor) da arquitetura dos *Web services*. Essa caracterização visa suprir algumas das necessidades da avaliação de desempenho dos *Web services* considerando a construção dessas mensagens baseada em informações reais. Além disso, a ferramenta possibilita obter diferentes tipos de informações tais como número de operações oferecidas pelos serviços que podem indicar a quantidade de requisições recebidas por um provedor de serviço, os tipos de Requisição/Codificação e tipos de dados utilizados nos parâmetros de entrada e saída.

A ferramenta é dividida em dois módulos: busca e análise. O primeiro módulo provê um mecanismo para a busca de serviços na *Web*. Esse módulo foi baseado no trabalho apresentado por Li et al. [2007] e apresenta duas abordagens de busca. Uma abordagem consiste na busca por *hyperlinks* que endereçam documentos *WSDL* em *Web sites*. Existem na *Web* vários portais especializados em publicar listas de *Web services*. A outra abordagem utiliza uma *API* chamada de *Google SOAP Search* que permite acessar a *engine* de busca oferecida pelo *Google*. O segundo módulo realiza a análise de documentos *WSDL* e é capaz de extrair um conjunto de informações presentes nos documentos *WSDL*.

Figura 5.1: Processo de Busca.

5.1 Módulo de Busca

A literatura apresenta vários trabalhos para automatizar a descoberta e composição de serviços [Petrie et al., 2008]. Alguns desses estudos têm utilizado métodos formais e técnicas de inteligência artificial tais como Ontologias para realizar essa tarefa através da implementação de protótipos [Yu et al., 2008]. Contudo, a maioria dos desenvolvedores de *Web services* preferem usar métodos manuais [Lu et al., 2007]. Os principais métodos manuais utilizados são a busca em *Web sites* especializados e em sistemas de busca como o *Google*.

O principal objetivo do módulo de busca implementado na ferramenta apresentada neste trabalho é encontrar documentos *WSDL* na *Web* através da automatização dos principais métodos de procura utilizados por desenvolvedores [Lu et al., 2007]. Como comentado no Capítulo 4, os documentos *WSDL* permitem a realização de uma análise que provê informações sobre a construção das mensagens trocadas entre os elementos da arquitetura dos *Web services*. Assim, utilizando-se o módulo de busca da ferramenta é possível encontrar uma quantidade significativa de documentos para que seja viabilizada uma análise baseada em dados reais. A Figura 5.1 mostra a metodologia utilizada para a obtenção desses documentos. As próximas seções discorrem sobre os dois métodos implementados no módulo de busca, detalhando as principais características de duas *APIs* utilizadas nessa implementação.

5.1.1 Google SOAP Search API

A *Google SOAP Search API* é uma ferramenta disponível para as linguagens *Java*, *Perl*, *Ruby* e *C#* [Google, 2001]. Ela foi criada com intuito de oferecer, a pesquisadores e desenvolvedores de software, um mecanismo para acessar a *engine* de busca oferecida pelo *Google*. A *API* utiliza o protocolo *SOAP* para acessar um *Web service* especialmente criado para processar requisições de busca por um determinado critério ou chave. Após esse processamento, uma estrutura de dados contendo as respostas é retornada ao programa cliente que solicitou a pesquisa. Essa estrutura

retorna resultados extraídos de bilhões de páginas indexadas pelo *Google* [Google, 2001].

A *Google SOAP Search API* é limitada a aplicações não comerciais. Os programas que utilizam a API possuem uma limitação de 1000 consultas por dia. Essas consultas são realizadas com uma chave obtida através da criação de uma conta. Desde dezembro de 2006, o *Google* não oferece suporte para essa API e as chaves que permitem acesso ao serviço já não estão disponíveis. A empresa alega que os desenvolvedores da API decidiram concentrar esforços em outra ferramenta chamada de *AJAX Search API*. Todavia, essa nova API não permite o acesso programático a *engine* de busca, ou seja, ela impede que sejam desenvolvidos programas que acessem de forma direta os recursos do *Google*. A descontinuidade da *Google SOAP Search API* não afetou os antigos usuários, pois a empresa manteve as antigas contas garantindo o uso da API por tempo indeterminado. O módulo de busca, especificamente na parte que utiliza os recursos de pesquisa oferecidos pelo *Google*, contou com a colaboração pesquisador Yan Li, da Universidade de Pequim.

A ferramenta *WSDLAnalyzer* utiliza a *Google SOAP Search API* para implementar o método que realiza o acesso a *engine* de busca do *Google* para obter documentos WSDL. Essa parte do módulo é dividida em duas funcionalidades. A primeira delas é responsável por realizar uma busca através de um dicionário de palavras. Essas palavras são utilizadas como parâmetros de busca. O objetivo principal do dicionário de palavras é automatizar a execução de várias buscas. Dessa forma, o usuário não é obrigado a inserir critérios de busca de forma manual. Para se obter um bom resultado na busca, os usuários devem construir dicionários de palavras que permitam categorizar os *Web services*. Uma alternativa pode ser a construção de um dicionário com palavras que fazem parte da sintaxe dos documentos WSDL. Por exemplo, um dicionário com palavras da sintaxe da WSDL pode ser composto por *schema*, *portType*, *message* e *binding*. Um dicionário de palavras com intuito de categorizar os *Web services* pode ser composto das palavras *entertainment*, *e-business*, *commerce*, *data Lookup*, dentre outras. A segunda funcionalidade é responsável por realizar uma busca onde o usuário pode inserir, de forma isolada, uma única palavra como chave de busca. Vale ressaltar que a construção do dicionário de busca bem como a definição das palavras que o compõem é realizado de forma empírica. A *Google SOAP Search API* e a ferramenta implementada não oferecem garantias de que uma consulta, por exemplo, pela categoria de *E-Business*, retornará documentos WSDL que de fato referenciam *Web services* pertencentes a tal categoria. Essa questão é um tema de pesquisa bastante discutido na literatura. Muitos pesquisadores tem enfrentado desafios para incluir questões de *Web* semântica na descoberta e composição de serviços [Petrie et al., 2008].

5.1.2 Biblioteca HTML Parser

A biblioteca *HTML Parser* é uma ferramenta responsável por oferecer um conjunto de filtros para extrair e modificar o conteúdo de documentos *HTML*. Ela permite extrair quaisquer informações desses documentos como *links*, imagens, áudio, vídeo e até mesmo códigos de *scripts*. Os desenvolvedores da biblioteca tiveram um cuidado especial em disponibilizar um conjunto de exemplos tornando fácil a sua utilização. Além disso, há ainda os *Javadocs* que possibilitam a realização de consultas às funcionalidades dos métodos oferecidos pelas classes da biblioteca. Por esses motivos, a biblioteca *HTML Parser* é uma boa opção, para a utilização neste trabalho, devido as facilidades de projeto, rapidez e capacidade de manipular páginas *HTML* [HTMLParser, 2006].

O módulo de busca oferece dois métodos para encontrar arquivos *WSDL* na *Web*. O primeiro método é implementado utilizando a *Google SOAP Search API*. O segundo método é responsável por extrair *links* que referenciam documentos *WSDL* dentro de páginas *HTML*. A biblioteca *HTML Parser* foi utilizada na implementação de um algoritmo recursivo para a realização dessa tarefa. Resumidamente, o algoritmo processa páginas *HTML* extraindo todos os *links* pertencentes a página em questão. Se o *link* referenciar um documento *WSDL*, uma conexão é realizada com o servidor que o hospeda, e assim, é efetuado o *download* do arquivo. A Figura 5.2 apresenta um fluxograma demonstrando o comportamento desse algoritmo.

Figura 5.2: Algoritmo de Busca em Portais de Web Site.

O primeiro passo do algoritmo consiste na passagem manual de uma página *HTML* (*URL* para um *Web site*) como parâmetro de entrada. Para a realização de uma busca com maiores chances de sucesso é recomendável a passagem de *Web sites* especializados em oferecer formas de acessar *Web services*. Diversos desses sites são

destacados por Li et al. [2007] e Lu et al. [2007]:

- www.xmethods.com
- www.webservicelist.com
- www.strikiron.com
- www.trynt.com
- www.remotemethods.com
- www.webservicex.net
- www.esynaps.com

A segunda etapa resume-se em extrair cada um dos *links* pertencentes aquele *Web site*. Na etapa seguinte é verificado se o *link* faz referência a um documento *WSDL*. Se a resposta for positiva, acontece o armazenamento do documento para um diretório do sistema de arquivos definido pelo usuário. Se a resposta for negativa, o algoritmo segue verificando se existem mais *links* a serem visitados. Se ainda houver, efetua-se a checagem de profundidade da busca. Isso é necessário para assegurar que o algoritmo tenha um ponto de parada evitando a sua execução em um laço infinito. Além disso, é possível configurar se o algoritmo percorrerá por entre páginas de diferentes domínios. Por exemplo, se o usuário passar como parâmetro o portal “www.webservicex.net” e definir que o algoritmo deve percorrer apenas páginas de mesmo domínio, o algoritmo irá acessar apenas as páginas dentro do domínio “webservicex.net”. Se o nível de profundidade não for alcançado o algoritmo segue acessando os demais *links* que ainda não foram percorridos. O algoritmo termina quando se esgotam os *links* ou quando a profundidade da busca é alcançada.

5.1.3 Filtragem dos dados

Como mencionado na seção 4.1.3 desta dissertação, o processo de obtenção de documentos *WSDL* na *Web* possui dois problemas principais: (1) muitos dos documentos com extensão *.wsdl* não possuem conteúdos correspondentes a documentos *WSDL* e (2) os dois métodos de pesquisa implementados no módulo de busca podem, eventualmente, realizar o *download* de arquivos replicados. Dessa forma, a ferramenta *WSDLANalyzer* deve prover meios para evitar esses problemas. O processo de filtragem de dados é realizado em duas etapas:

1. A primeira etapa corresponde à tarefa de filtrar arquivos *WSDL* que não correspondem à especificação da linguagem *WSDL*. Essa etapa engloba a filtragem dos

arquivos com outros conteúdos (ex. conteúdo *HTML*), e também, documentos *WSDL* mal formados e não válidos. Essa etapa contou com ajuda de uma API chamada *WSDL4J* [WSDL4J, 2006]. Essa API é uma implementação da IBM como referência a especificação da *JSR-110 (Java Specification Request 110)* [JCP, 2006]. Além de prover uma validação, a API *WSDL4J* oferece diversas funcionalidades para criar e manipular documentos *WSDL*. Essa etapa é realizada da seguinte forma: os métodos de pesquisa implementados no módulo de busca realizam uma verificação, em memória, avaliando se o documento referenciado pelo *link* obtidos por eles corresponde, de fato, a um documento *WSDL* bem formado (escritos na linguagem *WSDL*) e válido. Se essas restrições forem confirmadas ocorre, então, o armazenamento do documento para o sistema de arquivos.

2. A segunda etapa realiza a filtragem dos arquivos replicados. Para realizar esse procedimento de eliminação foi utilizada a classe `MessageDigest`, do pacote `java.security` da linguagem *Java*, que oferece implementações dos algoritmos *MD5 (Message-Digest Algorithm 5)* e *SHA (Secure Hash Algorithm)* [Sun, 2008]. Esses algoritmos convertem dados de tamanhos arbitrários em valores *hash* de tamanhos fixos. No caso deste trabalho foi utilizado o algoritmo *MD5*. O usuário da ferramenta, ao realizar uma busca, deve escolher o diretório onde são armazenados os documentos *WSDL* encontrados. Inicialmente, o procedimento de filtragem efetua o cálculo de todos os eventuais arquivos que já estejam presentes nesse diretório. Sem essa etapa de leitura do diretório de armazenamento, o usuário correria o risco de armazenar dados replicados em buscas realizadas em momentos distintos. Por exemplo, o usuário realiza uma busca e salva todos os resultados no diretório ‘Minhas_WSDLs’. Após essa busca, ele finaliza a execução da ferramenta, que por sua vez perde os valores *hash* que estavam armazenados em memória. Ao reabrir a ferramenta e executar uma nova busca, a ferramenta cria uma tabela *hash* inicial de todos os documentos *WSDL* do diretório de armazenamento definido pelo usuário. Ou seja, para o exemplo em questão o usuário deverá escolher o diretório “Minhas_WSDLs” caso seja necessária a garantia de que não existam arquivos replicados em diferentes execuções da busca. Para a criação da tabela *hash* utilizou-se um objeto da classe `HashSet`. Esse objeto tem a capacidade de detectar a presença de um determinado valor na tabela. Ao executar o método `HashSet.add()` ocorre o retorno de um valor `boolean`. Se a inserção for realizada com sucesso o valor retornado é `true`, senão é `false`.

Uma terceira filtragem corresponde à seleção de tamanho dos documentos *WSDL* que serão analisados. O usuário tem a possibilidade de selecionar um intervalo

de tamanho de documentos para a execução de uma análise mais específica. Por exemplo, arquivos *WSDL* muito grandes são raros (acima de um 1MB) de ser encontrados, e por isso, podem influenciar negativamente no resultado da análise. Esse filtro permite ao usuário da ferramenta incluir ou excluir esses dados que ocorrem eventualmente dentro de uma determinada amostra caso essa seja uma atividade necessária. A ferramenta oferece três tipos de granularidade para esse tipo de filtragem. O usuário pode escolher se a filtragem de tamanho de arquivos será realizada em *bytes*, *KB* (Kilobytes) ou *MB* (Megabytes).

5.2 Módulo de Análise

O processo de análise dos documentos *WSDL*, obtidos através do módulo de busca, utiliza a *API WSDL4J* assim como na primeira etapa de filtragem apresentada na seção 5.1.3. Dentro do contexto de análise, a *API* foi utilizada para extração de informações, e não apenas com o objetivo de realizar uma validação. O módulo de análise permite realizar a extração de várias informações dos documentos *WSDL* que podem ser utilizadas para o desenvolvimento de um estudo de caracterização de carga de trabalho. A Figura 5.3 apresenta um esquema de todas as possíveis informações quem podem ser obtidas pelo módulo de análise. O módulo inicia a análise extraindo as informações de *Binding*. O *Binding* é responsável por fornecer dados concretos que são associados a parte abstrata do documento *WSDL*. Esses dados concretos definem o estilo de requisição do serviço e o tipo de codificação utilizada na construção da mensagem.

As informações de *Binding* são divididas em três categorias: *SOAP 1.1*, *SOAP 1.2* e *HTTP*. Cada documento *WSDL* pode conter qualquer combinação dessas categorias ou simplesmente uma delas. O protocolo *SOAP*, explicado na seção 2.3.4, produz diferentes tipos de mensagens de acordo com a especificação do *Web service* que é definida pelo documento *WSDL*. Essas mensagens podem conter diferentes *payloads* dependendo da combinação de diferentes tipos de requisição/codificação (*RPC-Document/Encoded-Literal*). Por sua vez, o *Binding HTTP* é responsável por oferecer uma maneira de interação entre um *browser* e um *Web service* [W3C, 2001]. No entanto, esse tipo de *Binding* é limitado a serviços menos complexos em termos dos tipos de dados. Essa complexidade minimizada refere-se ao fato de que os parâmetros passados ao serviço utilizam os métodos *GET* e *POST* do protocolo *HTTP* de forma direta. Nesses casos, o protocolo *SOAP* não é utilizado na interação entre consumidor e provedor de serviço e os parâmetros são passados na requisição *HTTP*. No caso do método *GET*, os parâmetros são passados no *URI*. Por sua vez, no caso do método *POST*, os parâmetros são incluídos dentro do corpo da mensagem *HTTP*.

O processo de análise é realizado através da passagem do diretório do sistema

Figura 5.3: Processo de Análise.

de arquivos onde estão armazenados os documentos *WSDL*. Após a definição desse diretório, o usuário da ferramenta *WSDLANalyzer* deve informar quais dados serão analisados. Assim, o módulo executa uma análise de cada um dos arquivos *WSDL* contidos dentro do diretório escolhido pelo usuário. O processo de análise é iniciado extraíndo todos os elementos *binding* do documento *WSDL*. Esses elementos são responsáveis por fornecer informações concretas de como serão construídas as mensagens que serão trocadas entre consumidores e provedores de serviço. Essas informações estipulam os protocolos de transporte, estilo de requisição e tipo de codificação que serão utilizados na construção da mensagem. No caso do *HTTP Binding*, são extraídos os métodos *HTTP* utilizados. Um documento *WSDL* pode ter um ou mais elementos *binding* e cada um desses elementos são associados a um elemento *portType/interface* (parte abstrata do documento *WSDL*). A próxima etapa analisa isoladamente cada operação definida pelo elemento *binding* e compreende a extração de informações sobre os parâmetros de entrada e saída das operações. Nesse momento, o módulo de análise contabiliza os tipos de dados utilizados como parâmetros de entrada e saída. Ainda dentro desse contexto, o módulo calcula as médias do número de ocorrência de cada um dos tipos de dados definidos pelo *XML Schema*, apresentados na Figura 5.4. Adicionalmente, o processo de análise

Figura 5.4: Tipos de Dados - XML Schema [W3C, 2004c]

permite que o usuário defina filtros para tamanho de documentos *WSDL*. De acordo com Li et al. [2007] o tamanho dos documentos *WSDL* podem indicar, na maioria das vezes, a complexidade do *Web services*. Dessa forma, o usuário da ferramenta tem a possibilidade de relacionar o tamanho dos documentos *WSDL* com outras características. Ao finalizar a análise dos documentos, o módulo de análise apresenta os resultados de duas formas:

1. **Tabela de dados brutos:** Todas as informações extraídas são armazenadas em uma tabela que possui dados brutos. Cada linha dessa tabela possui informações de cada um dos documentos *WSDL* analisados. As colunas da tabela são compostas pelas informações selecionadas pelo usuário. As informações de tipos de dados são apresentadas pela média por operação de cada um dos tipos de dados escolhido pelo usuário. O número de operações, tamanho dos arquivos *WSDL* e detalhes de *Binding* (estilo de requisição e codificação) são apresentados pela quantidade encontrada dentro do documento, ou seja, não ocorre o cálculo da média para esses valores. Ao término da análise é apresentada uma janela

com os resultados que possui um botão que permite a visualização dessa tabela de dados brutos. Além disso, o usuário pode exportar os dados dessa tabela para um padrão *CSV (Comma-Separated Values)* que podem ser facilmente editados por softwares tais como *Microsoft Excel* e *OpenOffice Calc*.

2. **Tabela Estatística:** A janela principal dos resultados apresenta uma tabela que contém dados estatísticos sobre as informações que estão contidas na tabela de dados brutos. Essa tabela possui nas suas linhas as estatísticas sobre as informações selecionadas pelo usuário. Por sua vez, as colunas são compostas pelo número máximo e mínimo, média, variância, e desvio padrão de cada um dos dados analisados. As informações estatísticas, providas pela ferramenta *WSDLAnalyzer*, são calculadas considerando todos os documentos *WSDL* pertencentes ao diretório informado pelo usuário. Os números máximo e mínimo informam o maior e o menor número encontrado para cada característica analisada. A coluna da média provê a média aritmética das características. Por sua vez, a variância e o desvio padrão apresentam a dispersão dos dados em relação a média. Essas variáveis de dispersão podem apresentar a ocorrência de algum padrão nos dados. Por exemplo, se a média de número de operações por serviço for 2,2 e o desvio padrão for 1,66, o usuário poderá verificar que na maioria dos casos os *Web services* disponibilizam entre uma e três operações.

5.2.1 Gráficos

O módulo de análise oferece uma opção para visualizar uma combinação de características, escolhidas pelo usuário, em forma de gráficos. A parte do módulo de análise responsável pela criação de gráficos foi implementada utilizando a biblioteca *JFreeChart*. Essa biblioteca foi projetada para ser utilizada em aplicações, *Applets*, *Servlets* e *JSP (JavaServer Pages)*. A *JFreeChart* é distribuída sob os termos da *GNU Lesser General Public Licence* e pode ser utilizada tanto em aplicações proprietárias quanto livres [JFreeChart, 2009]. Uma característica relevante dessa biblioteca é que ela permite que os gráficos sejam exportados para dois tipos de formatos de imagens: *Portable Network Graphics (PNG)* ou *Joint Photographic Experts Group (JPEG)*. Adicionalmente, ela ainda oferece ao usuário a realização da ação de *zoom* que é útil em alguns casos onde se deseja destacar alguns valores.

O módulo de análise provê quatro tipos de gráficos: barra, linha, pizza e dispersão. O gráfico de barras pode ser utilizado para comparar diversas características levando em consideração um conjunto de categorias. Esses gráficos podem apresentar quantidades absolutas ou relativas (porcentagens) para dois ou mais valores no eixo vertical. No caso da ferramenta *WSDLAnalyzer*, foi utilizado o gráfico de barras com porcentagens. O gráfico de barras é interessante para variáveis discretas. Para

valores contínuos, a ferramenta oferece a criação de gráficos de linha seguindo o mesmo esquema de criação do gráfico de barras. O gráfico de linha oferece uma visualização mais clara dos dados. O gráfico de barras tem a desvantagem de se tornar poluído na comparação de muitas características. O gráfico de pizza tem como função a visualização de uma determinada característica de forma isolada. Por último, o gráfico de dispersão tem o objetivo de confrontar duas características. Utilizando o gráfico de dispersão, o usuário da ferramenta tem a possibilidade de encontrar relações entre características. Nesses gráficos são apresentadas as proporções relativas para uma determinada característica escolhida pelo usuário. As próximas seções mostram exemplos de utilização dos quatro tipos de gráficos que a ferramenta *WSDLAnalyzer* oferece.

Gráfico de Barra e Linha

O gráfico de barras permite que o usuário compare vários dados da análise. O eixo X apresenta as categorias onde são enquadradas uma ou mais características. Cada uma dessas categorias possui um conjunto de barras que representam as características selecionadas pelo usuário. A criação do gráfico é realizada selecionando uma ou mais linhas da tabela estatística da janela de resultados. O módulo de análise utiliza as informações da tabela de dados brutos para a geração dos gráficos. O eixo Y apresenta as frequências relativas (porcentagens) de cada característica analisada dentro das categorias definidas pelo usuário. Ao requisitar a criação do gráfico, o usuário informa a quantidade e o tamanho das categorias desejadas. Uma observação importante, dentro do processo de criação, é que os valores fracionados na tabela de dados brutos precisam ser arredondados, uma vez que algumas características contêm valores médios, por exemplo, tipos de dados por operação.

A ferramenta *WSDLAnalyzer* ainda permite a criação de gráficos de linha que podem ser mais úteis em ocasiões onde a comparação de muitas características se faz necessária. Como comentado anteriormente, o gráfico de barras pode se tornar pouco legível em situações de comparação de muitas características.

A Figura 5.5 apresenta um exemplo de gráfico de barras. Nesse caso, o usuário selecionou duas características: quantidade de operações utilizando o estilo de requisição *Document* e codificação *Literal*. O gráfico de barras do exemplo, mostra que o usuário escolheu cinco categorias de tamanho 1 apresentando a proporção de documentos *WSDL* que possuem essas características para cada categoria. A categoria 0 (zero), por exemplo, indica que 27,5% dos documentos *WSDL* não possuem nenhuma requisição no estilo *Document* e que aproximadamente 25% não possuem nenhuma requisição utilizando codificação *Literal*.

A Figura 5.6 apresenta um exemplo de gráfico de linhas. Nesse exemplo, o usuário selecionou quatro características: requisições com estilo *RPC*, estilo de codificação

Figura 5.5: Exemplo de Gráfico Barras **Figura 5.6:** Exemplo de Gráfico de Linhas

Literal, quantidade de tipos complexos e tipos *String*. O gráfico de linhas possui o mesmo propósito que o gráfico de barras. No entanto, ele permite analisar a disposição dos dados quando comparado com o gráfico de barras de forma mais clara. O gráfico de linha segue o mesmo método de interpretação dos dados. Para a característica de tipos complexos, por exemplo, o usuário pode fazer a leitura de que aproximadamente 25% dos documentos *WSDL* não possuem tipos complexos, 20% possuem apenas um tipo e assim sucessivamente.

Figura 5.7: Exemplo de Gráfico de Pizza

Gráfico de Pizza

O gráfico de pizza oferecido pelo módulo de análise possui o objetivo de analisar características de forma isolada. O processo de criação desse gráfico é semelhante ao gráfico de linhas. O usuário deve informar, ao módulo de análise, a quantidade e o tamanho das categorias que serão apresentadas no gráfico. A ferramenta *WSDLAnalyzer* acrescenta uma categoria extra representando os dados que se encontram acima da última categoria requerida. Por exemplo, se o usuário configurar a criação do gráfico de pizza contendo duas categorias, a ferramenta realiza a inserção de uma categoria a mais com dados que excedem a segunda categoria configurada. A Figura 5.7 mostra um exemplo do gráfico de pizza que foi construído para analisar tamanhos de documentos *WSDL*. O exemplo mostra uma configuração onde o usuário selecionou cinco categorias com 10240 bytes (10KB) de tamanho (o tamanho da categoria, no caso de análise de arquivos *WSDL*, é dado em bytes). Automaticamente, o módulo insere uma sexta categoria indicando a proporção de documentos com tamanhos superiores a 50KB.

Gráfico de Dispersão

A função principal do gráfico de dispersão é apresentar o relacionamento entre duas características analisadas. Cada ponto do gráfico representa um valor arredondado de duas características da tabela de dados brutos, sem qualquer cálculo estatístico. Por exemplo, um determinado documento *WSDL* possui em média, por operação, 3,25 tipos *String* e 4,32 tipos *Int*. Nesse caso, o módulo de análise

arredonda os valores para 3 e 4 respectivamente. Se o eixo X representar tipos `String` e o eixo Y representar os tipos `Int`, um ponto será criado na posição $X=3$ e $Y=4$. O gráfico de dispersão apresenta os valores das características analisadas em escala logarítmica. Essa definição de escala é necessária, pois diversos documentos *WSDL* possuem valores muito elevados para determinadas características. Dessa forma, a visualização dos dados torna-se mais clara. A Figura 5.8 mostra um exemplo de gráfico de dispersão criado pelo módulo de análise.

Figura 5.8: Exemplo de Gráfico de Dispersão

5.3 Modelagem da WSDLANalyzer

A modelagem de sistemas computacionais é parte importante dentro do desenvolvimento de um software. Ela é interessante no sentido de simplificar a realidade facilitando a compreensão dos problemas que devem ser solucionados pelo software [Fowler and Scott, 2000]. A modelagem, dentro do contexto deste trabalho, tem como intuito oferecer uma forma simplificada de entendimento da estrutura de implementação da ferramenta *WSDLANalyzer*. Além disso, a modelagem é essencial para a continuação do desenvolvimento e manutenção da ferramenta. Para realizar a modelagem foi utilizada a *UML (Unified Modeling Language)*. A *UML* é um formalismo para análise e projeto de software. Ela apresenta um conjunto de diagramas para a definição da estrutura e comportamento de um sistema computacional [Simmonds and Bastarrica, 2005]. No entanto, não foram utilizados todos os recursos fornecidos pela *UML*. O principal objetivo desta seção é apresentar, da forma mais simplificada possível, os componentes implementados e os seus relacionamentos.

5.3.1 Diagrama de Classes

O diagrama de classes permite a modelagem, em um modo declarativo, da estrutura que compõe o domínio de aplicação, em termos de conceitos e relacionamentos. O uso de um diagrama de classe é válido para demonstrar um sistema sob um foco conceitual onde os detalhes de implementação não são considerados [Berardi et al., 2005], embora ele permita também uma especificação sob uma perspectiva da implementação [Fowler and Scott, 2000]. De acordo com Fowler and Scott [2000], existem três perspectivas de utilização dos diagramas de classe: conceitual, especificação e implementação. No caso deste trabalho, foi utilizado o diagrama de classes sob a perspectiva conceitual que representa os conceitos do domínio da ferramenta *WSDLAnalyzer*. Cada conceito possui uma classe que irá representá-lo. A idéia é que o diagrama forneça informações sem considerar o software que irá implementá-lo, ou seja, o diagrama deve prover uma visão que independe de linguagem de programação.

Figura 5.9: Diagrama de Classes Conceitual.

A Figura 5.9 apresenta um diagrama de classes conceitual que determina a estrutura da ferramenta *WSDLAnalyzer*. Embora o diagrama seja conceitual, foi colocado um legenda relacionando cada uma das classes especificadas com os seus respectivos pacotes de implementação. O objetivo dessa legenda é nortear os

conceitos da ferramenta com os pacotes que, de fato, os implementam. É importante ressaltar que diversas classes e pacotes implementados para compor a ferramenta *WSDLANalyzer* não estão demonstradas na Figura 5.9. O principal motivo para a supressão dessas classes e pacotes é apresentar um diagrama intuitivo e claro. Basicamente, a ferramenta está dividida em seis pacotes principais:

- `wsdlanalyzer.layout.frames` - pacote que contém as classes que implementam as principais janelas da aplicação. A classe `MainFrame` implementa a janela principal onde podem ser instanciados os módulos de busca e análise apresentados nas seções 5.1 e 5.2 respectivamente.
- `wsdlanalyzer.layout.internalframes` - pacote que compreende as classe que implementam as janelas internas que oferecem uma interface gráfica para os módulos de busca e análise. Essas janelas internas, quando instanciadas, são mostradas dentro da janela `MainFrame`. As principais classes desse pacote são `AnalysisInternalFrame`, `GoogleSearchInternalFrame` e `WebSiteInternalFrame`.
- `Wsdlanalyzer.wsdl.search.websearch` - pacote que inclui as classes que implementam o módulo de busca. Os dois métodos oferecidos pelo módulo de busca, explicados nas seções 5.1.1 e 5.1.2, são implementados pelas classes `WebGoogleSearch` e `WebSiteSearch`. Por sua vez, a classe `FileDownloadURL` é responsável por implementar o processo de filtragem explicado na seção 5.1.3. Ela faz o *download* dos arquivos *WSDL* encontrados pelas classes `WebGoogleSearch` e `WebSiteSearch` verificando se esses arquivos já existem no sistema de arquivos. Além disso, a classe verifica se os arquivos baixados correspondem, de fato, a documentos *WSDL*.
- `wsdlanalyzer.wsdl.search.threads` - a ferramenta *WSDLANalyzer* é projetada para suportar a execução dos módulos de busca e análise de forma paralela. Ou seja, os módulos são independentes e podem ser executados paralelamente. Para permitir essa execução paralela são utilizados *threads*. As classes `WebGoogleSearchProcessThread` e `WebSiteSearchProcessThread` implementam *threads* que instanciam objetos das classes `WebGoogleSearch` e `WebSiteSearch`. Dessa forma, o usuário da aplicação tem a possibilidade de realizar pesquisas nos dois métodos providos pelo módulo de busca simultaneamente.
- `wsdlanalyzer.wsdl.analysis.statistic` - pacote que contém as classes responsáveis pela extração das informações estatísticas dos documentos *WSDL* através da manipulação dos dados brutos. A principal classe desse pacote é a

`StatisticDataTableProcess` que manipula os dados brutos através do suporte de um pacote estatístico, provido pelo grupo Apache [Apache, 2009a], chamado `Math`.

- `wsdlanalyzer.wsdl.analysis.threads` - pacote que possui a classe `RawDataTableProcessThread`. Essa classe implementa uma *thread* que tem como funcionalidade o preenchimento da tabela de dados brutos. A análise efetiva dos documentos *WSDL* é realizada pelo pacote `wsdlanalyzer.wsdl.analysis.parser`. No entanto, a classe preenche a tabela de dados brutos com as informações obtidas pelo pacote `parser`. Após esse preenchimento, ocorre uma instanciização da classe `StatisticDataTableProcess` que realiza a manipulação dos dados brutos transformando-os em informações estatísticas.
- `wsdlanalyzer.wsdl.analysis.parser` - pacote que contém as classes responsáveis por analisar e extrair as informações, definidas pelo usuário da ferramenta, dos documentos *WSDL*. A classe que concentra todo o gerenciamento dessa análise é a `WSDLAnalysis`. A partir dessa classe são criadas instâncias das classes `WSDLBindingAnalysis` e `InputOutputAnalysis` que são responsáveis por obter, respectivamente, informações de *Binding* e de parâmetros de entrada e saída. As classes `PrimitiveOrDerivedTypesXS` e `ExtensibilityNodeTypeHandler` auxiliam a classe `InputOutputAnalysis` no processo de extração da informações de entrada e saída. A classe `PrimitiveOrDerivedTypesXS` tem a função de contabilizar a utilização dos tipos de dados, apresentados na Figura 5.4, para as entradas e saídas. Por sua vez, a classe `ExtensibilityNodeTypeHandler` é responsável por fazer a leitura do *XMLSchema* que define os tipos de dados utilizados nas operações disponibilizadas pelo *Web service*.

5.3.2 Diagramas de Sequência

O diagrama de sequência é utilizado para demonstrar a relação entre os diversos tipos de objetos através de uma sequência temporal de chamadas de métodos [Page-Jones, 2000]. No contexto deste trabalho, os diagramas de sequência são utilizados para apresentar os detalhes de interação entre os objetos que efetuam os processos de busca e análise dos documentos *WSDL*.

O diagrama de sequência apresentado na Figura 5.10 mostra as relações entre os objetos das classes que implementam o processo de análise dos documentos *WSDL*. O processo é iniciado através do objeto da classe `AnalysisInternalFrame` que executa o método `start()` do objeto `rawTableThread` (classe `RawDataTableProcessThread`). Posteriormente, o objeto `rawTableThread` executa o método `performAnalysis()`

do objeto `wsdlAnalysis`. Esse método é responsável por iniciar a extração das informações dos documentos WSDL. Por sua vez, o objeto `wsdlAnalysis` utiliza o objeto `wsdlBindAnalysis` (classe `WSDLBindingAnalysis`) para extrair informações dos tipos de *Binding* que podem estar definidos no documento. Após a obtenção dos tipos de *Binding*, o objeto `wsdlAnalysis` obtém as informações sobre os tipos de dados utilizados nos parâmetros de entrada e saída das operações. Essas informações de dados são extraídas através do objeto `inOutAnalysis` da classe `InputOutputAnalysis`. A definição dos tipos de dados, explicada na seção 2.3.2, são manipulados pelo objeto da classe `ExtensibilityNodeTypeHandler` que manipula o *XMLSchema* definido dentro do documento WSDL.

Figura 5.10: Diagrama de Sequência do Processo de Análise.

Os diagramas de sequência apresentados na Figura 5.11 mostram as relações entre os objetos que realizam o processo de busca. O processo de busca realizado através da *engine* de busca do *Google*, apresentado na Figura 5.11a, mostra o objeto `googleInternalFrame` iniciando o objeto `googleThread` através do método `start()`. Assim, o objeto `googleThread` executa a busca por meio do objeto `googleSearch` (classe `WebGoogleSearch`). A execução dessa busca é efetivada através da chamada do método `performSearch()` que possui como parâmetro de retorno um `ArrayList` contendo todas as *URLs* encontradas referenciando documentos WSDL. Por sua vez, o processo de busca em *Web sites* (Páginas HTML) é apresentado na 5.11b. Nesse caso, a busca é iniciada através do objeto `siteThread` que executa o método `navigaSite()` do objeto `siteSearch` (classe `WebSiteSearch`). Esse método implementa a navegação por entre os *links* das páginas HTML dos *Web sites* definidos pelo usuário. O *download* dos arquivos, para a busca em *Web sites*, são efetuados pelo próprio objeto `siteSearch`, diferentemente da busca pelo *Google* onde os *downloads* são realizados

pelo objeto `googleThread`.

Figura 5.11: Diagramas de Sequência do Processo de Busca.

5.4 Interface Gráfica

A ferramenta *WSDLAnalyzer* é composta por uma interface gráfica amigável que permite ao usuário configurar, de forma intuitiva, a realização da busca e análise dos documentos *WSDL*. A finalidade dessa interface está em facilitar a utilização dos módulos da ferramenta. A composição da interface gráfica da ferramenta utiliza, principalmente, dois componentes oferecidos pela *API Swing*¹ da linguagem *Java*: *JDesktopPane* e *JInternalFrame*. A ferramenta *WSDLAnalyzer* utiliza um modelo de interface gráfica chamada de *MID* (*Multiple-Document Interface*). Em resumo, a interface é composta por uma janela principal denominada “janela pai” (implementada utilizando o componente *JDesktopPane*) e janelas filhas (implementadas utilizando o componente *JInternalFrame*) que provêm a utilização dos módulos de busca e análise [Deitel, 2009]. A Figura 5.12 mostra a janela principal da ferramenta com uma janela filha de análise instanciada.

5.4.1 Janela do Módulo de Análise

A janela do módulo de análise possui três painéis de configuração. O primeiro painel permite ao usuário escolher quais tipos de dados serão analisados. Os dados estão divididos em tipos gerais: *String*, *Date*, Numéricos e Miscelânea. O segundo painel tem a função prover configurações relativas às informações dos arquivos *WSDL*

¹Swing é uma API de suporte a criação de interfaces gráfica na linguagem *Java*.

Figura 5.12: Janela Principal da Ferramenta WSDLAnalyzer.

(nome e tamanho), informações de *Binding* (*SOAP 1.1*, *SOAP 1.2* e *HTTP*), e também, número de operações. Por último, o painel de filtragem por tamanho de arquivos é responsável por prover as configurações para a filtragem dos dados, explicada na seção 5.1.3, considerando o tamanho dos arquivos.

5.4.2 Janelas do Módulo de Busca

A Figura 5.13 mostra a janela responsável por oferecer uma interface para de configuração da busca através da *engine* do *Google*. O usuário pode realizar uma pesquisa simples através de uma chave de busca (palavra) apenas, ou então, efetuar uma busca por um dicionário de palavras. Esse dicionário é um arquivo texto que deve possuir, em cada uma de suas linhas, as palavras chave que serão utilizadas na pesquisa. Ainda nessa janela, é informada a localização do sistema de arquivos onde serão armazenados os documentos *WSDL* encontrados.

Figura 5.13: Janela da Busca no Google.

A janela de busca para a pesquisa em *Web sites* é apresentada na Figura 5.14. Nessa janela, o usuário da ferramenta tem a possibilidade de inserir um conjunto de *Web sites* onde serão efetuadas as pesquisas por documentos *WSDL*. A inserção desses *Web sites* é realizada através da definição das *URLs* (protocolo://máquina/caminho/recurso). Além da definição das *URLs*, o usuário deve configurar a profundidade da busca que impede a execução cíclica do algoritmo (ver seção 5.1.2). Assim como na janela de busca pelo *Google*, a janela de busca em *Web sites* possui um campo de configuração da localização onde são armazenados os

Figura 5.14: Janela da Busca em Web Sites.

documentos *WSDL*.

Ambas as janelas do módulo de busca possuem um painel que contém um console de apresentação das atividades executadas pelos métodos de pesquisa oferecidos. Além disso, para a busca na *engine* do *Google* é mostrada uma barra informando o progresso da pesquisa. Essa barra de progresso não é implementada para o método de busca em *Web site*, uma vez que não é possível prever o número de *links* que serão encontradas dentro de um determinado *Web site*.

5.5 Considerações Finais

Este capítulo apresentou a especificação da *WSDLANalyzer*, uma ferramenta para a busca e análise de documentos *WSDL*. Uma das principais características da ferramenta é facilitar o processo de busca de documentos *WSDL*. O usuário da *WSDLANalyzer* tem a possibilidade de obter, de forma automatizada e simples, um conjunto de documentos *WSDL* através da *Web*. Além disso, ela provê um mecanismo que obtém um conjunto de informações sobre quaisquer conjuntos de documentos *WSDL*. Essas informações podem ser utilizadas para a construção de *Web services* com perfis semelhantes aos encontrados na *Web*, para a avaliação de desempenho através das informações sobre a construção das mensagens utilizadas na interação consumidor/provedor, e também, para verificar aspectos sobre protocolos e padrões mais utilizados.

O início do capítulo descreveu os módulos que implementam esses mecanismos

bem como a descrição das *APIs* utilizadas. O módulo de busca oferece métodos para a realização de pesquisas em *Web sites*, e também, na *engine* de busca do *Google*. Adicionalmente, um método de filtragem de dados para a seleção de documentos *WSDL* válidos é discutido. O módulo de análise, apresentado na seção 5.2, descreve quais informações podem ser extraídas, e também, a maneira como essa extração é implementada. A seção 5.3 mostra a modelagem da ferramenta *WSDLANalyzer*. Essa modelagem é realizada utilizando diagramas de classe e seqüência da *UML*. Ainda nessa seção é apresentada a descrição da interface gráfica que permite utilização da ferramenta.

O Capítulo 6 desta dissertação apresenta um conjunto de resultados obtidos através da utilização da ferramenta *WSDLANalyzer*. Esse capítulo apresenta as principais informações, obtidas através da coleta de documentos *WSDL* na *Web*, para a caracterização das mensagens trocadas entre consumidores e provedores de serviço.

Análise e Avaliação de WSDLs

O estudo de cargas de trabalho em sistemas computacionais visa a provisão de modelos que preservem as características mais relevantes da carga de trabalho real [Menascé and Almeida, 2002]. Como comentado no Capítulo 4, as metodologias comumente aplicadas a estudos de caracterização de cargas de trabalho em servidores *Web* não puderam ser aplicadas dentro do contexto dos *Web services*. A principal dificuldade de se aplicar essas metodologias é a impossibilidade de se encontrar, na indústria, *traces* que mapeiem a execução de *Web service*.

Assim, o presente trabalho caracteriza a carga de trabalho sob a ótica da construção das mensagens trocadas entre consumidores e provedores de serviço. Essa caracterização baseia-se em arquivos *WSDL* encontrados na *Web*. Essas mensagens contêm as invariantes que são as características aplicadas a todo o conjunto de dados estudado, ou seja, comuns a todos os sistemas baseados em *Web services*. Essas mensagens representam o conjunto de informações recebidas por um provedor de serviço para atendimento de requisições dos consumidores do serviço. As próximas seções deste capítulo discutem a coleta e as análises dos documentos *WSDL* que foram obtidos e analisados através da ferramenta *WSDLANalyzer*. Para viabilizar a caracterização da carga de trabalho para os *Web services* optou-se por avaliar o que esses serviços oferecem através de informações obtidas nos documentos *WSDL*, e por meio dessas informações, determinar qual carga pode estar sendo imposta a eles.

Os *Web services* possuem algumas peculiaridades em relação as cargas de trabalho. Uma delas diz respeito ao processo de negócio que é executado pelo serviço. Esse processo de negócio é muito específico e depende da área de atuação da empresa que disponibiliza o serviço. Por exemplo, uma aplicação da bolsa de valores

de São Paulo pode expor um *Web service* para a realização da cotação de moedas de diferentes países. Em contrapartida, uma empresa de cartões de crédito pode utilizar um *Web service* para realizar uma transação de compra. Cada um desses serviços exemplificados possuem diferentes implementações, e computacionalmente, podem variar, por exemplo, em custo de processamento. O primeiro exemplo pode simplesmente realizar o cálculo das cotações através da computação de variáveis disponibilizadas pelo mercado. Por outro lado, um *Web service* que oferece transações financeiras necessita consultar bancos de dados, e talvez, acessar *Web services* de outras empresas para concretizar uma operação financeira.

Por esses motivos, diversos trabalhos de avaliação de desempenho em *Web services* abstraem a implementação do serviço considerando-o como uma caixa preta que serializa e deserializa mensagens [Chen et al., 2006], [Chen et al., 2007] e [Zilora and Ketha, 2008]. Outros trabalhos utilizam *benchmarks* que emulam a execução de um serviço padrão como é caso do estudo apresentado por García et al. [2006], detalhado no Capítulo 3.

6.1 Coleta de Dados

O processo de coleta de dados resultou na obtenção de 1346 arquivos *WSDL*. Essa coleta utilizou os dois métodos de busca oferecidos pela ferramenta *WSDLAnalyzer*. A coleta através do método de busca que utiliza a *engine* de busca do *Google* foi realizada configurando um dicionário de busca na forma como foi explicado na seção 5.1.1. Esse dicionário foi dividido em duas partes: palavras reservadas de documentos *WSDL* e categorias de serviços. A Tabela 6.1 apresenta as chaves utilizadas para compor as duas partes do dicionário de busca.

A coleta de dados utilizando o método de busca em *Web sites* foi realizada configurando o módulo de busca com os endereços definidos na seção 5.1.2. Esse módulo de busca exige que uma profundidade de busca seja definida para que o algoritmo de pesquisa tenha um ponto de parada. A definição do tamanho da profundidade é realizada de forma empírica, ou seja, não há um formalismo que indique qual o melhor valor para esse atributo. Assim, o melhor valor de ajuste encontrado para obtenção do maior número de documentos *WSDL*, na busca realizada neste trabalho, foi a profundidade de grau três.

Os dois métodos de busca, explicados na seção 5.1, efetuam o processo de filtragem durante a execução da pesquisa. Os arquivos replicados, inválidos e não correspondentes a documentos *WSDL* são eliminados antes mesmo de serem armazenados no sistema de arquivos. Dessa forma, após a execução desses métodos obteve-se um conjunto de arquivos *WSDL* que proporcionaram a realização das análises estatísticas que são discutidas nas próximas seções deste capítulo.

Tabela 6.1: Chaves de Pesquisa do Dicionário de Busca.

Chaves de Busca	Descrição
Palavras Reservadas	definitions, schema, message, service, operation, portType, interface, type, complextype, simpletype, part, input, output, body, document, literal, rpc, encoded, soap, port, binding, attribute, element .
Categorias	business, commerce, entertainment, communication, development, data conversion, computation, data lookup, graphic, multimedia, information management.

A realização das análises sobre os documentos *WSDL* encontrados pelos métodos de busca utilizaram o módulo de análise provido pela ferramenta *WSDLAnalyzer*. Outro software utilizado para o desenvolvimento da análise foi o *OpenOffice Calc* (software análogo ao *Microsoft Excel*). Esse software permitiu a realização de análises adicionais que não são providas automaticamente pela ferramenta *WSDLAnalyzer*. Adicionalmente, um software, chamado *BestFit*, é utilizado para a criação de um modelo estatístico da análise. Esse software dispõe como principal funcionalidade a provisão de ajuste de distribuições de probabilidade para um determinado conjunto de dados [Jankauskas and McLafferty, 1996] e [Palisade, 2009]. Os gráficos de ajuste de distribuição apresentados neste capítulo possuem no eixo das abscissas as quantidades das características analisadas, enquanto que o eixo das ordenadas define a frequência acumulada. A análise dos dados é dividida da seguinte forma:

- Análise dos tipos de Binding (*SOAP 1.1*, *SOAP 1.2* e *HTTP*).
- Análise dos tamanhos dos documentos *WSDL*.
- Análise da quantidade de operações providas para cada um dos tipos de *Binding*.
- Análise dos tipos de dados utilizados como parâmetros de entrada e saída.

O final de cada análise de *Binding* apresenta um conjunto de tabelas que resume as distribuições ajustadas nos dados e as informações estatísticas obtidas para cada característica examinada.

6.2 Binding - SOAP 1.1

O uso do *Binding* através do protocolo *SOAP* na versão 1.1 corresponde a aproximadamente 97% (1311 arquivos) dos documentos *WSDL* encontrados. O W3C encontrou mais de 400 problemas com relação a ambigüidade no modelo de processamento para a versão 1.1 do protocolo *SOAP*. Essas ambigüidades são os principais fatores que dificultam os aspectos de interoperabilidade entre os *Web services*. Portanto, a maioria dos dados analisados remete a serviços que podem

conter muitos problemas de interoperabilidade quando acessados via protocolo SOAP 1.1.

As próximas seções deste capítulo utilizam os conceitos sobre tipos de Requisição/Codificação introduzidos na seção 2.3.4. Essas características definem qual o estilo de requisição e tipos de codificação utilizados nas operações para criar as mensagens SOAP que são trocadas entre consumidores e provedores de serviço. Sumariamente, os estilos de requisição *Document* e *RPC* definem que as mensagens SOAP devem ser trocadas na forma de um documento e como uma chamada de procedimento respectivamente. Na forma de documento, o *payload* da mensagem é composto por um documento XML. No caso do *RPC* há uma representação em XML da chamada do método. Para a codificação *Literal*, os tipos de dados que compõem o corpo da mensagem segue “literalmente” o que está definido no XML Schema do documento WSDL. Por sua vez, a codificação *Encoded* segue as regras de codificação do protocolo SOAP. Adicionalmente, uma análise dos tamanhos dos documentos WSDL e número médio de operações por Web service é apresentada. Por último, são discutidos a quantidade e o tipos de parâmetros utilizados na definição das operações.

Figura 6.1: SOAP 1.1: Gráfico de Pizza do Tamanho dos Documentos WSDL.

6.2.1 Tamanhos dos Documentos WSDL

A Figura 6.1 mostra a proporção dos tamanhos dos arquivos em um gráfico de pizza. O gráfico é dividido em 10 categorias de 10KB e mais uma categoria compreendendo os documentos WSDL com tamanhos superiores a 100KB. Os resultados obtidos para o tipo *Binding* SOAP 1.1 mostram que 47% dos arquivos possuem até 10KB e 77% são menores que 30KB. A quantidade de documentos que

possuem tamanhos de até 100KB representa 93% dos dados obtidos. Com relação a documentos grandes, com tamanhos superiores a 1MB, apenas 9 arquivos foram encontrados representando aproximadamente 0,61% do total de arquivos. De acordo com Li et al. [2007], os tamanhos dos documentos *WSDL* podem dar indicativos sobre a complexidade dos *Web services*. Ou seja, os tamanhos dos arquivos *WSDL* se elevam a medida com que a complexidade dos *Web services* cresce. Vale ressaltar que a análise dos tamanhos dos arquivos foi efetuada também no estudo apresentado em [Li et al., 2007]. No entanto, alguns números obtidos por este trabalho diferem em alguns aspectos dos apresentados por Li et al. [2007]. Uma explicação possível para essa diferença pode ser atribuída ao ano em que os estudos foram realizados. O trabalho apresentando por esta dissertação realizou a coleta dos dados em julho de 2008, e posteriormente, em fevereiro de 2009. O trabalho de Li et al. [2007] realizou as buscas dos dados em dezembro de 2006. A dinamicidade da *Web* aliada a maturidade gradativa da tecnologia dos *Web services* com o decorrer dos anos pode ter influenciado os resultados obtidos pelos dois estudos. Além disso, uma outra diferença é que a análise dos tamanhos dos documentos *WSDL* realizada neste trabalho é dividida por tipos de *Binding*.

6.2.2 Quantidade de Operações

A Figura 6.2 apresenta um gráfico de pizza com as proporções do número de operações encontrados nos documentos *WSDL* que possuem *Binding SOAP 1.1*. Os resultados demonstram que 69% dos documentos disponibilizam menos de 6 operações. O número de operações em 79% dos documentos analisados não excede a quantidade de 20 operações. A média da quantidade de operações para esses 79% é de 2,66. No entanto, a média de operações para todos os documentos *WSDL* é 15,67. O aumento da média na análise de todos os documentos obtidos é explicado devido a existência de 3 documentos grandes com 1.778, 2.147 e 3.830 operações. Assim como o tamanho, a quantidade de operações definidas nos documentos *WSDL* pode servir de parâmetro na indicação da complexidade dos *Web services* [Li et al., 2007]. Diferentemente da análise dos tamanhos dos documentos *WSDL*, os valores estatísticos encontrados sobre as quantidades de operações se assemelham com os valores apresentados por [Li et al., 2007].

Partindo do pressuposto que um provedor de serviços torna-se mais sobrecarregado a medida que ele prove um maior número de serviços, os documentos *WSDL* analisados indicam que um *Web service*, na maioria dos casos, não oferece muitas implementações de sistemas complexos.

Figura 6.2: SOAP 1.1: Gráfico de Pizza do Número de Operações.

Figura 6.3: SOAP 1.1: Gráfico de Dispersão - Tamanho X Número de Operações

Tamanhos de Documentos WSDL X Número de Operações

O gráfico da Figura 6.3 mostra que o aumento da quantidade de operações está relacionado, em muitos casos, com o aumento do tamanho do arquivo. Isso parece óbvio, mas a figura mostra que essa relação não é linear. A explicação para esse fato é a inexistência de uma relação proporcional para o crescimento do tamanho dos documentos *WSDL* com o aumento do número de operações. Outros fatores influenciam para aumento dos documentos *WSDL* como: utilização de *tags* de documentação, complexidade dos tipos de dados (aumento do *XSD*) e quantidade de

tipos de *Binding*.

Ajuste de Distribuição para o Número de Operações

A distribuição que melhor se ajusta as quantidades de operações encontradas nos documentos WSDL é a Geométrica com o parâmetro $p = 0,12910$. A Figura 6.4 apresenta um gráfico que mostra a distribuição de ajuste comparada aos dados coletados. O gráfico mostra as frequências acumuladas para a ocorrência das quantidades de operações por documento WSDL.

Figura 6.4: SOAP 1.1: Ajuste de Distribuição para o Número de Operações.

6.2.3 Requisições no Estilo Document

A Figura 6.5 mostra um gráfico de pizza com as proporções das quantidades de operações no estilo *Document* encontradas nos documentos WSDL. O setor do gráfico representado pela categoria 0 (zero) revela que 27% dos documentos analisados não possuem operações no estilo de requisição *Document*. Esse valor informa, de maneira inversa, que 73% (955 arquivos) dos documentos WSDL contém operações com requisições do tipo *Document*, pois no *SOAP Binding* uma operação deve possuir obrigatoriamente o estilo *Document* ou *RPC*. O número de operações que utilizam o estilo de requisição *Document* não ultrapassa, em 59% do total dos documentos analisados (*RPC* e *Document*), a quantidade de 10 incluindo os documentos que não possuem operações com esse estilo. Considerando apenas os documentos WSDL que possuem esse estilo de requisição, o valor sobe de 59% para aproximadamente 80%. O número médio encontrado é de aproximadamente 6 operações com estilo *Document*, considerando todos os documentos analisados. Excluindo-se os arquivos que não possuem operações com o estilo *Document* esse valor sobe de 6 para aproximadamente

9 operações por documento WSDL.

Ajuste de Distribuição para o Número de Requisições no Estilo Document

As operações que definem requisições no estilo *Document* possuem a *Beta-Generalizada* como distribuição de ajuste. Os parâmetros para essa distribuição são: $\alpha_1 = 0,25410$, $\alpha_2 = 2,7763$, $min = 0,0$ e $max = 56,402$. A Figura 6.6 mostra o gráfico com as curvas da distribuição de ajuste e as frequências acumuladas dos dados coletados para essa característica.

Figura 6.5: SOAP 1.1: Gráfico de Pizza de Operações no Estilo Document.

Figura 6.6: SOAP 1.1: Ajuste de Distribuição para o Número de Requisições no Estilo Document.

6.2.4 Requisições no Estilo RPC

A quantidade de documentos *WSDL* que fazem uso do estilo *RPC* nas operações providas pelo Web service representa 27% (356 arquivos) dos dados analisados, ou seja, 73% não possuem nenhuma operação no estilo *RPC*. A Figura 6.7 mostra as proporções das quantidades encontradas na análise para essa característica. Dentre os 356 arquivos que possuem operações *RPC*, 74% possuem até 10 operações por documento. A quantidade máxima de operações com o estilo *RPC* encontrada foi de 3830 operações. No entanto, apenas três documentos *WSDL* apresentam número de operações superior a 300. A média considerando todos os arquivos *WSDL* é de aproximadamente 9 operações por documento. Considerando apenas os documentos que possuem operações *RPC* e excluindo os três documentos que possuem mais de 300 operações nesse estilo, o valor médio sobe de 9 para 11 operações por documento *WSDL*.

Figura 6.7: SOAP 1.1: Gráfico de Pizza de Operações no Estilo RPC.

Ajuste de Distribuição para o Número de Requisições no Estilo RPC

A distribuição que melhor se ajusta aos dados do número de requisições no estilo *RPC* é a *Beta-Generalizada* com os parâmetros $\alpha_1 = 0,12005$, $\alpha_2 = 1,6070$, $min = 0,0$ e $max = 10,526$. A Figura 6.8 mostra o gráfico de ajuste. A probabilidade de não ocorrer operações que utilizam o estilo de requisição *RPC* é alta. O gráfico mostra a probabilidade da categoria 0 (zero) em aproximadamente 80%. Ou seja, quase 80% dos documentos *WSDL* analisados não apresentam operações que utilizam o estilo de requisição *RPC*. Essa baixa ocorrência de operações que utilizam o *RPC* faz com que

Figura 6.8: SOAP 1.1: Ajuste de Distribuição para o Número de Requisições no Estilo RPC.

se tenha um ajuste ruim aos dados coletados. É importante observar no gráfico da Figura 6.8 que o início do distribuição de ajuste se dá, aproximadamente, no valor 0,5 do eixo das frequências acumuladas enquanto que a curva para os dados reais inicia em aproximadamente 0,8.

6.2.5 Codificação do Tipo Literal

Os valores encontrados para o tipo de codificação *Literal* assemelham-se aos valores encontrados para as operações no estilo *Document*. A Figura 6.9 mostra um gráfico de pizza com diferenças imperceptíveis em relação ao gráfico apresentado na Figura 6.5. No entanto, alguns valores divergem. Por exemplo, 29% dos documentos WSDL possuem até 2 operações no estilo *Document*, e 31% desses mesmos documentos utilizam o tipo de codificação *Literal*. De acordo com Butek [2005], o tipo de Requisição/Codificação mais indicado para construir *Web services* com um maior grau de interoperabilidade, considerando o tipo de Requisição/Codificação, é o tipo *Document/Literal*.

Os dados obtidos mostram que a maioria dos *Web services* utiliza esse tipo de configuração. Os dados para o estilo *Document* e para a codificação *Literal* não são exatamente iguais devido a existência de alguns documentos WSDL que utilizam *RPC/Literal* nas suas definições. A Figura 6.10 mostra um gráfico de dispersão entre as características *Document* e *Literal*. A relação entre essas duas características é linear para a maioria dos documentos analisados. Os resultados obtidos para o tipo *Document/Literal* retrata o esforço dos desenvolvedores de *Web services* na provisão de serviços com um maior grau de interoperabilidade, embora, esse seja o tipo mais complexo de Requisição/Codificação para se manipular.

Figura 6.9: SOAP 1.1: Gráfico de Pizza da Utilização do Tipo de Codificação Literal.

Figura 6.10: SOAP 1.1: Gráfico de Dispersão - Document X Literal

Ajuste de Distribuição para o Número de Codificação do Tipo Literal

Os dados para o número de operações que utilizam o tipo de codificação *Literal* obtiveram a *Beta-Generalizada* como distribuição de ajuste. Como comentado anteriormente, os dados para o tipo de codificação *Literal* assemelham-se se aos dados do estilo de requisição *Document*. Por esse motivo, as distribuições de ajuste para o estilo de requisição *Document* (Figura 6.11) e para o tipo de codificação *Literal* também são semelhantes. O gráfico apresentado na Figura 6.11 apresenta os seguintes parâmetros para a distribuição de ajuste: $\alpha_1 = 0,27349$, $\alpha_2 = 2,9879$, $min = 0,0$ e

$max = 58,019$.

Figura 6.11: SOAP 1.1: Ajuste de Distribuição para o Número de Codificação do Tipo Literal.

6.2.6 Codificação do Tipo Encoded

Os valores obtidos para o tipo de codificação *Encoded*, por sua vez, assemelham-se com os valores obtidos para as operações no estilo *RPC*. A Figura 6.12 apresenta o gráfico de pizza para essa característica. Assim como para o tipo *Document/Literal* (Requisição/Codificação), os gráficos apresentados nas Figuras 6.7 e 6.12 possuem diferenças sutis nas proporções para o tipo de requisição *RPC/Encoded*. A quantidade de documentos *WSDL* que utilizam a codificação *Encoded* para a construção das mensagens *SOAP* corresponde a 25% dos dados analisados contra 27% dos documentos que utilizam o estilo *RPC*. Novamente, essa diferença é explicada devido a existência de alguns documentos que possuem tipos de requisições *RPC/Literal*. A Figura 6.13 mostra um gráfico de dispersão comparando as duas características. A maioria das operações nos documentos que possuem requisições no estilo *RPC* utiliza proporcionalmente a codificação *Encoded*.

Ajuste de Distribuição para o Número de Codificação do Tipo Encoded

Assim como para as operações que utilizam o estilo de requisição *RPC*, os dados para o tipo de codificação *Encoded* obtêm a *Beta-Generalizada* como distribuição de ajuste. Contudo os parâmetros são um pouco diferentes: $\alpha_1 = 0,11826$, $\alpha_2 = 1,8446$, $min = 0,0$ e $max = 10,920$. A Figura 6.14 apresenta o gráfico da distribuição de ajuste para esse caso. Para 93% dos casos, o número de operações com esse estilo não excede 10 operações por documento. Assim, para se obter uma distribuição com um

Figura 6.12: SOAP 1.1: Gráfico de Pizza da Utilização do Tipo de Codificação Encoded.

melhor ajuste, optou-se por excluir os documentos com mais de 10 operações com o tipo de codificação *Encoded*. O mesmo problema de ajuste discutido para o estilo de requisição *RPC* acontece para o tipo de codificação *Encoded*. A baixa ocorrência desse tipo de codificação prejudica o ajuste da distribuição.

Figura 6.13: SOAP 1.1: Gráfico de Dispersão - RPC X Encoded

Figura 6.14: SOAP 1.1: Ajuste de Distribuição para o Número de Codificação do Tipo Encoded.

6.2.7 Parâmetros de Entrada

O número médio de parâmetros de entrada para o *Binding SOAP 1.1* é próximo a 4 parâmetros por operação. Aproximadamente 76% das operações não possuem quantidades superiores a 5 parâmetros. As operações que contêm mais de 20 parâmetros correspondem a 8% do total das operações encontradas nos documentos WSDL. A Figura 6.15 apresenta um gráfico de pizza mostrando as proporções das quantidades médias de parâmetros de entrada.

Figura 6.15: SOAP 1.1: Gráfico de Pizza - Número Médio de Parâmetros de Entrada por Operação.

Figura 6.16: SOAP 1.1: Ajuste de Distribuição para o Número Parâmetros de Entrada.

O principal tipo de dado utilizado nos parâmetros de entrada é o tipo *String*. Aproximadamente, 71% dos parâmetros correspondem a tipos de dados *String*. O segundo tipo mais utilizado é o tipo *Int* com 9% dos parâmetros. Os tipos *Boolean* e *Double* compreendem cada um deles, aproximadamente 3% dos parâmetros de entrada. Tipos *Float*, *Long*, *DateTime* e *Base64Binary* possuem 1% dos parâmetros cada um.

O valor médio para a quantidade de tipos complexos (*ComplexType*) é de aproximadamente 1 tipo por operação. Vale ressaltar que um tipo complexo pode ser composto por vários outros tipos complexos ou simples. O valor médio para tipos simples (*SimpleType*) é, também, de aproximadamente 1 tipo por operação. Um tipo simples pode ser *String*, *Float*, *Double*, etc. Essas médias são calculadas considerando todos os documentos *WSDL* que possuem o *Binding SOAP 1.1* sem a realização de filtros. Em [Head et al., 2005] é mostrado, dentre outras análises, a influência dos tipos de dados nos processos de serialização e deserialização. Nesse trabalho é demonstrado o custo para realizar a tradução dos tipos de dados recebidos em XML para uma estrutura de dados específica de uma determinada linguagem.

Ajuste de Distribuição para o Número de Parâmetros de Entrada

A distribuição que melhor se ajusta ao número de parâmetros de entrada é a *Weibull* com os parâmetros $\alpha = 1,2937$ e $\beta = 2,9991$. A Figura 6.16 apresenta o gráfico da distribuição para esse caso. Adicionalmente, o ajuste de distribuição é realizado para alguns tipos de dados. Entretanto, esse ajuste, para a maioria dos tipos, não foi possível devido a sua baixa ocorrência nos parâmetros de entrada, exceto para os tipos simples, complexos e *String*. O Apêndice A.1.1 apresenta os gráficos para os

tipos de dados.

6.2.8 Parâmetros de Saída

A quantidade média de parâmetros de saída por operação é de aproximadamente 6 parâmetros. Essas quantidades não excedem o valor de 10 parâmetros para 77% dos documentos analisados. As operações com mais de 20 parâmetros de saída correspondem a 14% dos dados. A Figura 6.17 apresenta um gráfico de pizza mostrando as proporções das quantidades médias de parâmetros de saída por operação.

Assim como nos parâmetros de entrada, o principal tipo utilizado como parâmetro de entrada é o tipo `String`. No entanto, a proporção desse tipo de parâmetro é menor correspondendo a 63% do total de tipos de dados utilizados. Os tipos de dados `Int` e `Double` possuem respectivamente 9% e 6%. Tipos `Float`, `Long`, `DateTime` e `Base64Binary` possuem cada um 2% total de parâmetros. Tipos `Boolean` e `Decimal` possuem cada um 5% e 1% respectivamente.

Figura 6.17: SOAP 1.1: Gráfico de Pizza - Número Médio de Parâmetros de Saída por Operação.

A quantidade média de tipos complexos é de aproximadamente 2 por operação considerando todos os documentos que definem o *Binding SOAP 1.1*. A utilização de tipos simples utilizados como parâmetros de saída é de aproximadamente 1 parâmetro por operação. Essa análise mostra que os tipos de dados de saída possuem uma complexidade maior, ou seja, a estrutura dos dados de retorno das operações, em geral, é mais complexa.

Figura 6.18: SOAP 1.1: Ajuste de Distribuição para o Número Parâmetros de Saída.

Ajuste de Distribuição para o Número de Parâmetros de Saída

A Figura 6.18 apresenta o gráfico de distribuição de ajuste para essa característica. A distribuição *Lognormal*, com os parâmetros $\mu = 5,3525$ e $\sigma = 9,7205$, possui o melhor ajuste para os parâmetros de saída. Assim como para os parâmetros de entrada, o ajuste de distribuição também é realizado para apenas alguns tipos de dados pelos mesmos motivos comentados na seção 6.2.7. O Apêndice A.1.2 mostra os gráficos de ajuste de distribuição para os tipos de dados utilizados nos parâmetros de saída.

6.2.9 Resumo das Distribuições Estatísticas para o Binding SOAP 1.1

A Tabela 6.2 mostra um resumo das distribuições estatísticas ajustadas nas características do Binding SOAP 1.1. Quatro tipos de distribuições são utilizadas: *Geométrica*, *Beta-Generalizada*, *Weibull* e *Lognormal*. O Apêndice B apresenta as equações e os domínios para o cálculo dessas distribuições.

Tabela 6.2: Resumo das Distribuições Estatísticas para o Binding SOAP 1.1

Característica	Nome da Distribuição	Parâmetros
Nro. Operações	Geométrica	$p = 0,12910$
Document	Beta-Generalizada	$\alpha1 = 0,25410, \alpha2 = 2,7763, \min = 0,0$ e $\max = 56,402$
RPC	Beta-Generalizada	$\alpha1 = 0,12005, \alpha2 = 1,6070, \min = 0,0$ e $\max = 10,526$
Literal	Beta-Generalizada	$\alpha1 = 0,27349, \alpha2 = 2,9879, \min = 0,0$ e $\max = 58,019$
Encoded	Beta-Generalizada	$\alpha1 = 0,11826, \alpha2 = 1,8446, \min = 0,0$ e $\max = 10,920$
Parâmetros Entrada	Weibull	$\alpha = 1,2937$ e $\beta = 2,9991$
Parâmetros Saída	Lognormal	$\mu = 5,3525$ e $\sigma = 9,7205$

Tabela 6.3: SOAP 1.1: Resumo Total das Características Requisição/Codificação.

Características	Max	Min	Média	Desvio Padrão
Nro. Operações	3830	1	15,68	132,23
Document	214	0	6,36	15,5
RPC	3830	0	9,05	131,45
Literal	252	0	6,68	16,93
Encoded	3830	0	8,99	131,60
Parâmetros Entrada	254	0	3,93	8,37
Parâmetros Saída	321	0	6,11	14,67

Tabela 6.4: SOAP 1.1: Resumo Parcial das Características Requisição/Codificação.

Características	Max	Min	Média	Desvio Padrão
Nro. Operações	20	1	4,67	4,33
Document	20	1	4,57	4,18
RPC	20	1	4,95	4,73
Literal	20	1	4,57	4,59
Encoded	20	1	4,91	4,64
Parâmetros Entrada	20	1	3,53	3,27
Parâmetros Saída	20	1	4,25	4,26

6.2.10 Resumo das Variáveis Estatísticas para o Binding SOAP 1.1

As Tabelas 6.3 e 6.4 mostram um resumo dos valores obtidos para cada uma das características do *Binding SOAP 1.1*. A Tabela 6.3 mostra as variáveis estatísticas considerando todos os documentos WSDL analisados. Por sua vez, a Tabela 6.4, apresenta as variáveis estatísticas filtrando os valores dos documentos que não possuem a característica em questão, e também, valores superiores a 20 unidades. Por exemplo, 27% dos documentos WSDL não possuem estilo de requisição *Document* e apenas 8% possui mais de 20 operações com esse estilo. Para esse caso, o cálculo das variáveis estatísticas é realizado para aqueles arquivos WSDL que possuem entre 1 e 20 operações com o estilo *Document*. A quantidade de qualquer característica analisada neste trabalho, na maioria dos documentos WSDL, não possui valores superiores a 20 unidades. Por isso, optou-se por apresentar uma tabela que exclui as quantidades superiores a esse valor com a finalidade de se obter variáveis estatísticas condizentes com os documentos WSDL mais comumente encontrados.

6.3 Binding SOAP 1.2

O uso do *Binding* através do protocolo SOAP na versão 1.2 está presente em aproximadamente 14% (185 arquivos) dos documentos WSDL encontrados. Assim como na seção 6.2 onde são discutidas as características do *Binding SOAP 1.1*,

esta seção discorre sobre as mesmas características analisadas anteriormente, no entanto, sob a ótica do *Binding SOAP 1.2*. Esta seção discorre sobre os tamanhos dos documentos WSDL, número de operações, tipos de Requisição/Codificação, quantidade e tipos de parâmetros de entrada e saída. Em geral, os documentos WSDL que provêm o tipo de *Binding SOAP 1.2* também provêm o *Binding SOAP 1.1*. A explicação para esse fato pode ser atribuída à manutenção da acessibilidade dos consumidores implementados para utilizar o protocolo SOAP na versão 1.1.

6.3.1 Tamanhos dos documentos WSDL

Os resultados obtidos para os tamanhos de arquivos do tipo *Binding SOAP 1.2* mostram que 46% dos arquivos possuem até 10KB e 70% deles são menores que 30KB. A quantidade de documentos que possuem tamanhos de até 100KB representa 95% dos dados obtidos. Com relação a documentos grandes, não foram encontradas ocorrências de arquivos com tamanhos superiores a 1MB. A Figura 6.19 mostra a proporção dos tamanhos dos arquivos em um gráfico de pizza. O gráfico é dividido em 10 categorias de 10KB e mais uma categoria compreendendo os documentos WSDL com tamanhos superiores a 100KB. Os valores encontrados para tamanho de arquivos para o Binding SOAP 1.2 assemelham-se aos valores encontrados no *Binding SOAP 1.1*. As Figuras 6.1 e 6.19 mostram que os gráficos de pizza, para ambos os tipos de *Binding*, possuem pequenas diferenças. A partir dessa observação pode-se concluir que o tamanho de um documento WSDL, um dos índices para avaliar a complexidade de um *Web service*, não está relacionado com o tipo de *Binding* utilizado para acessar o serviço.

Figura 6.19: SOAP 1.2: Gráfico de Pizza do Tamanho dos Documentos WSDL.

6.3.2 Quantidade de Operações

As quantidades de operações disponíveis nos documentos que possuem o *Binding SOAP 1.2* também são semelhantes aos valores obtidos para o *Binding SOAP 1.1*. A Figura 6.20 apresenta um gráfico de pizza com as proporções do número de operações encontrados nos documentos WSDL que possuem o *Binding SOAP 1.2*. Os resultados demonstram que 68% dos documentos disponibilizam menos de 6 operações. O número de operações em 91% dos documentos analisados não excede a quantidade de 20 unidades. A média da quantidade de operações para esses 91% é de 2,27 operações por documento WSDL. No entanto, a média de todos os documentos WSDL é de 8,09 operações. Alguns documentos possuem valores superiores a 50 operações determinando o aumento da média.

A mesma análise sobre as quantidades de operações para os documentos com *SOAP Binding 1.1* vale para o *SOAP Binding 1.2*. Sob a ótica da quantidade de operações, a maioria dos *Web services*, de forma isolada, não implementam sistemas complexos.

Figura 6.20: SOAP 1.2: Gráfico de Pizza do Número de Operações.

Tamanhos de Documentos WSDL X Número de Operações

A Figura 6.21 apresenta um gráfico mostrando a relação entre tamanhos de documentos WSDL e número de operações. A mesma relação é mostrada na análise do *Binding SOAP 1.1* e novamente percebe-se que a quantidade de operações está relacionada com o tamanho do arquivo WSDL. Embora a relação não seja linear, o gráfico mostra que há uma relação entre as duas características.

Figura 6.21: SOAP 1.2: Gráfico de Dispersão - Tamanho X Número de Operações

A distribuição que melhor se ajusta a quantidade de operações é a Geométrica assim como no ajuste de distribuição para essa característica no *Binding SOAP 1.2*. No entanto, o parâmetro da função possui valor distinto. O gráfico com o ajuste de distribuição é mostrado no Apêndice A.2.1.

6.3.3 Requisições no Estilo Document

Os valores encontrados para as operações que utilizam o estilo *Document*, por sua vez, possuem diferenças significativas em relação a análise dessa mesma característica no *Binding SOAP 1.1*. A Figura 6.22 mostra um gráfico de pizza com as proporções das quantidade de operações no estilo *Document* encontradas. Diferentemente do *Binding SOAP 1.1*, o setor do gráfico representado pela categoria 0 (zero) mostra que apenas 4% dos documentos analisados não possuem operações no estilo de requisição *Document*. Esse valor informa, de maneira inversa, que 96% (176 arquivos) dos documentos *WSDL* possuem operações com requisições do tipo *Document*. O número de operações que utilizam o estilo de requisição *Document* não ultrapassa, em 80% do total dos documentos analisados (*RPC* e *Document*), a quantidade de 10 operações incluindo os documentos que não possuem operações com esse estilo. O número médio de operações no estilo *Document* é de aproximadamente 8 operações, considerando todos os documentos analisados.

Ajuste de Distribuição para o Número de Requisições no Estilo Document

A distribuição de ajuste para as operações que definem requisições no estilo *Document*, para o *Binding SOAP 1.2*, é a *Gaussiana Inversa*, diferentemente do *Binding*

Figura 6.22: SOAP 1.2: Gráfico de Pizza de Operações no Estilo Document.

SOAP 1.1 que possui a distribuição Beta-Generalizada. Os parâmetros para essa distribuição são: $\mu = 6,6350$ e $\lambda = 3,6363$. A Figura 6.23 mostra o gráfico com as curvas da distribuição de ajuste e as frequências acumuladas dos dados coletados para essa característica.

Figura 6.23: SOAP 1.2: Ajuste de Distribuição para o Número de Requisições no Estilo Document.

6.3.4 Requisições no Estilo RPC

As operações que utilizam o estilo de requisição RPC, no *Binding SOAP 1.2*, não são frequentemente encontradas. Apenas 4% (7 arquivos) dos 185 documentos WSDL

encontrados para o *Binding SOAP 1.2* possuem operações com esse estilo. Esse fato pode ser explicado devido às recomendações de uma organização da indústria chamada *Web Service Interoperability (WS-I)*. Essa organização é responsável por estabelecer boas práticas para a manutenção da interoperabilidade dos *Web services* [WS-I, 2009]. Por ser um protocolo mais recente, é compreensível que a utilização do protocolo *SOAP 1.2* seja feita seguindo as recomendações para a construção de *Web services* com maior interoperabilidade. Os dados coletados mostram justamente isso, grande parte dos documentos WSDL disponibiliza somente operações que utilizam o estilo *Document*. Uma distribuição de frequência para a ocorrência de operações no estilo *RPC* não pôde ser ajustada. A baixa ocorrência dessa característica impossibilitou o ajuste de uma curva que representasse o comportamento dos dados.

6.3.5 Codificação do Tipo Literal

Os valores encontrados para o tipo de codificação *Literal* no *Binding SOAP 1.2* assemelham-se aos valores encontrados para as operações no estilo *Document*, assim como se assemelham os valores para as características *Document/Literal* do *Binding SOAP 1.1*. O fato é que praticamente todos os documentos encontrados possuem em suas operações o estilo *Document* combinado com o tipo de codificação *Literal*. Contudo a ocorrência do tipo de codificação *Literal* é superior em apenas uma unidade, pois existe um documento WSDL, dentro dos arquivos coletados, utilizando o tipo de requisição *RPC/Literal*. Por esse motivo, os valores discutidos para o estilo de requisição *Document* também são válidos para o tipo de codificação *Literal*.

Figura 6.24: SOAP 1.2: Ajuste de Distribuição para o Número de Codificação do Tipo Literal.

Ajuste de Distribuição para o Número de Codificação do Tipo Literal

A distribuição *Gaussiana Inversa* é a distribuição que melhor se ajusta aos dados para o tipo de codificação *Literal*, assim como para as operações no estilo *Document* (Figura 6.23). A Figura 6.24 apresenta o gráfico com as curvas dos dados coletados e a distribuição de ajuste para o tipo de codificação *Literal*. Os valores dos parâmetros da distribuição *Gaussiana Inversa* para esse caso são: $\mu = 6,6632$ e $\lambda = 3,7068$.

6.3.6 Codificação do Tipo Encoded

O tipo de codificação *Encoded* é utilizado em poucos documentos *WSDL*, aproximadamente 3% (6 arquivos) dos documentos coletados. Como comentado anteriormente, o tipo de codificação *Encoded* geralmente é associado as operações com estilo de requisição *RPC*. Como esse estilo é pouco utilizado no *Binding SOAP 1.2*, o tipo de codificação *Encoded* também possui baixa ocorrência. O ajuste de uma distribuição de probabilidade não foi possível ser realizada para essa característica visto que grande parte dos documentos analisados não define esse tipo de codificação para a construção das mensagens *SOAP*.

6.3.7 Parâmetros de Entrada

Com relação aos parâmetros de entrada, o *Binding SOAP 1.2* possui valores similares aos valores obtidos para o *Binding SOAP 1.1*. O número médio de parâmetros de entrada para o *Binding SOAP 1.2* é próximo a 4 parâmetros por operação. Aproximadamente 82% das operações não possuem quantidades superiores a 5 parâmetros. As operações que contém mais de 20 parâmetros correspondem a 5% do total das operações. A Figura 6.15 apresenta um gráfico de pizza mostrando as proporções das quantidades médias de parâmetros de entrada por operação.

O principal tipo de dado utilizado nos parâmetros de entrada é o tipo *String*. Aproximadamente, 78% dos parâmetros correspondem a tipos de dados *String*. O segundo tipo mais utilizado, assim como para o *Binding SOAP 1.1*, é o tipo *Int* com 9% dos parâmetros. Tipos *Boolean*, *Double*, *Float*, *DateTime* e *Base64Binary* definem 1% dos parâmetros cada um.

O valor médio para a quantidade de tipos complexos (*ComplexType*) é de aproximadamente 1 tipo por operação. O valor médio para tipos simples (*SimpleType*), no entanto, é de 0,21. Ou seja, no *Binding SOAP 1.2* a utilização de tipos simples ocorre com baixa frequência. Essas médias são calculadas considerando todos os documentos *WSDL* que possuem o *Binding SOAP 1.2* sem a realização de filtros.

Figura 6.25: SOAP 1.2: Gráfico de Pizza - Número Médio de Parâmetros de Entrada por Operação.

Ajuste de Distribuição para o Número de Parâmetros de Entrada

A distribuição que melhor se ajusta ao número de parâmetros de entrada é a *Weibull* com os parâmetros $\alpha = 1,5496$ e $\beta = 3,1854$. A Figura 6.26 apresenta o gráfico da distribuição. Assim como na seção 6.2.7, o ajuste de distribuição foi realizado para alguns tipos de dados. Essas distribuições são apresentadas no Apêndice A.2.2. O ajuste para a maioria dos tipos dados não foi possível devido a sua baixa ocorrência nas definições dos parâmetros de entrada, exceto para os tipos complexos (*ComplexType*) e *String*.

Figura 6.26: SOAP 1.2: Ajuste de Distribuição para o Número Parâmetros de Entrada.

6.3.8 Parâmetros de Saída

O número médio de parâmetros de saída por operação é de aproximadamente 5 parâmetros. As quantidades de parâmetros não excedem em 10 unidades para 81% dos documentos analisados. As operações com mais de 20 parâmetros de saída correspondem a 15% dos dados analisados. A Figura 6.27 apresenta um gráfico de pizza mostrando as proporções das quantidades médias de parâmetros de saída por operação. Vale ressaltar que os valores obtidos para os parâmetros de saída, mais uma vez, são semelhantes aos valores encontrados para o *Binding SOAP 1.1*.

Assim como nos parâmetros de entrada, o principal tipo utilizado como parâmetro de entrada é o tipo `String`. No entanto, a proporção desse tipo de parâmetro é menor correspondendo a 65% do total de tipos de dados utilizados para definir os parâmetros de saída. Os tipos de dados `Int`, `Boolean` e `Double` definem respectivamente 8%, 6% e 4% do total de parâmetros de saída. Tipos `Float` e `DateTime` definem cada um 2% do total de parâmetros. Demais tipos como `Long`, `Decimal` e `Base64Binary` definem 1% dos parâmetros cada um.

A quantidade média de tipos complexos é de aproximadamente 2 unidades por operação, considerando todos os documentos analisados. A utilização de tipos simples como parâmetros de saída é de aproximadamente 0,14, ou seja, possui baixa ocorrência. Essa análise mostra, assim como nos parâmetros de entrada, que tipos de dados complexos têm maior ocorrência para os parâmetros de saída.

Figura 6.27: SOAP 1.2: Gráfico de Pizza - Número Médio de Parâmetros de Saída por Operação.

Figura 6.28: SOAP 1.2: Ajuste de Distribuição para o Número Parâmetros de Saída.

Ajuste de Distribuição para o Número de Parâmetros de Saída

A distribuição de ajuste que melhor se ajusta ao número de parâmetros de saída é a *Beta-Generalizada* com os parâmetros $\alpha_1 = 0,48223$, $\alpha_2 = 1,2088$, $min = 0,0$ e $max = 10,109$. A Figura 6.28 apresenta o gráfico da distribuição de ajuste. As distribuições de ajuste para os tipos de dados relativos aos parâmetros de saída são mostradas nos gráficos do Apêndice A.2.2.

6.3.9 Resumo das Distribuições Estatísticas para o Binding SOAP 1.2

A Tabela 6.5 mostra um resumo das distribuições estatísticas ajustadas nas características do *Binding SOAP 1.1*. Quatro tipos de distribuições são utilizadas: *Geométrica*, *Gaussiana Inversa*, *Weibull* e *Beta-Generalizada*. O Anexo B apresenta as equações e os domínios para o cálculo dessas distribuições. O caractere “-” indica a impossibilidade de se ajustar uma distribuição para a característica.

Tabela 6.5: SOAP 1.2: Resumo das Distribuições Estatísticas para o Binding SOAP 1.1

Característica	Nome da Distribuição	Parâmetros
Nro. Operações	Geométrica	$p = 0,13660$
Document	Gaussiana Inversa	$\mu = 6,6350$ e $\lambda = 3,6363$
RPC	-	-
Líteral	Gaussiana Inversa	$\mu = 6,6632$ e $\lambda = 3,7068$
Encoded	-	-
Parâmetros Entrada	Weibull	$\alpha = 1,5496$ e $\beta = 3,1854$
Parâmetros Saída	Beta-Generalizada	$\alpha_1 = 0,48223$, $\alpha_2 = 1,2088$, $min = 0,0$ e $max = 10,109$

Tabela 6.6: SOAP 1.2: Resumo Total das Características Requisição/Codificação.

Características	Max	Min	Média	Desvio Padrão
Nro. Operações	150	1	8,09	15,27
Document	150	0	7,75	15,34
RPC	12	0	0,34	1,81
Literal	150	0	6,68	16,93
Encoded	12	0	0,3	1,8
Parâmetros Entrada	39,9	0	4,08	5,45
Parâmetros Saída	63,4	0	5,22	9,01

6.3.10 Resumo das Variáveis Estatísticas para o Binding SOAP 1.2

As Tabelas 6.6 e 6.7 mostram um resumo das valores obtidos para cada uma das características do *Binding SOAP 1.2*. A Tabela 6.6 mostra as variáveis estatísticas considerando todos os documentos WSDL analisados. Por sua vez, a Tabela 6.7, apresenta as variáveis estatísticas filtrando os valores dos documentos que não possuem a característica em questão, e também, valores superiores a 20 unidades.

Tabela 6.7: SOAP 1.2: Resumo Parcial das Características Requisição/Codificação.

Características	Max	Min	Média	Desvio Padrão
Nro. Operações	20	1	4,78	4,87
Document	20	1	4,6	4,85
RPC	20	1	8,86	3,67
Literal	20	1	4,63	4,85
Encoded	20	1	8,83	4,02
Parâmetros Entrada	20	1	3,58	3,43
Parâmetros Saída	20	1	3,78	3,37

6.4 Binding SOAP 1.1 Vs Binding SOAP 1.2

As seções 6.2 e 6.3 apresentaram um conjunto de distribuições de ajuste para as características do *Binding SOAP* nas versões 1.1 e 1.2. Adicionalmente, as duas seções mostram algumas variáveis estatísticas como média e desvio padrão para cada característica. Esta seção visa realizar uma análise comparativa entre os dois tipos de *Binding SOAP*.

A Tabela 6.8 mostra a definição das distribuições de ajuste para cada uma das características analisadas em ambos os tipos de *Binding*. O *Binding SOAP 1.1* possui, para as características do estilo de requisição *Document/RPC* e tipo de codificação *Literal/Encoded*, a distribuição *Beta-Generalizada*. O *Binding SOAP 1.2* não possui uma distribuição de ajuste para o estilo *RPC* e para o tipo de codificação *Encoded*. Contudo, para o estilo *Document* e tipo codificação *Literal* a distribuição *Gaussiana*

Inversa obteve o melhor ajuste nos dados. As distribuições estatísticas evidenciam que o estilo de requisição está intimamente relacionado com o tipo de codificação. Isso é observado através dos parâmetros das distribuições. Por exemplo, analisando-se separadamente os *Binding SOAP 1.1* e *1.2* o estilo de requisição *Document* possui as mesmas distribuições com diferenças mínimas nos parâmetros quando comparadas com o tipo de codificação *Literal* do mesmo tipo de *Binding*. No entanto, não há um relacionamento entre as distribuições dos tipos *Binding SOAP 1.1* e *1.2*. Embora os valores médios e de desvios padrão sejam próximos. Esse fato pode ser explicado devido a quantidade de dados coletados para ambos os *Bindings*. A ocorrência do *Binding SOAP 1.1* é observada em 1312 documentos WSDL contra 186 documentos para o *Binding SOAP 1.2*. O ajuste de uma distribuição estatística sobre um conjunto de dados aumenta na medida em que se disponibiliza um maior número de dados.

Tabela 6.8: Tabela Comparativa de Distribuições de Ajuste

Características	Binding SOAP 1.1	Binding SOAP 1.2
Nro. Operações	Geométrica $p = 0,12910$	Geométrica $p = 0,13660$
Document	Beta-Generalizada $\alpha_1 = 0,25410$, $\alpha_2 = 2,7763$ $min = 0,0$ e $max = 56,402$	Gaussiana Inversa $\mu = 6,6350$ e $\lambda = 3,6363$
RPC	Beta-Generalizada $\alpha_1 = 0,12005$, $\alpha_2 = 1,6070$ $min = 0,0$ e $max = 10,526$	-
Literal	Beta-Generalizada $\alpha_1 = 0,27349$, $\alpha_2 = 2,9879$ $min = 0,0$ e $max = 58,019$	Gaussiana Inversa $\mu = 6,6632$ e $\lambda = 3,7068$
Encoded	Beta-Generalizada $\alpha_1 = 0,11826$, $\alpha_2 = 1,8446$ $min = 0,0$ e $max = 10,920$	-
Parâmetros Entrada	Weibull $\alpha = 1,2937$ e $\beta = 2,9991$	Weibull $\alpha = 1,5496$ e $\beta = 3,1854$
Parâmetros Saída	Lognormal $\mu = 5,3525$ e $\sigma = 9,7205$	Beta-Generalizada $\alpha_1 = 0,48223$, $\alpha_2 = 1,2088$ $min = 0,0$ e $max = 10,109$

A Tabela 6.9 mostra um resumo das variáveis estatísticas encontradas para ambos os *Binding SOAP*. Cada célula da tabela contém a média (M), o desvio padrão (DV) e os intervalos confiança (IC) para todas as características do *Binding*. Os cálculos realizados para a construção dessa tabela utilizam a eliminação dos documentos que não possuem a característica em questão, e também, os documentos que possuem mais de 20 unidades para cada característica.

Para o número de operações, as variáveis mostram que a maioria dos *web services* não provê mais de 6 operações considerando os valores dos intervalos de confiança. O desvio padrão para ambos os *Binding* são similares indicando, praticamente, a mesma variação para essa característica. Para as características

Tabela 6.9: Tabela Comparativa de Variáveis Estatísticas

Características	Binding SOAP 1.1	Binding SOAP 1.2
Nro. Operações	$M = 4,67$ e $DP = 4,33$ $IC = 4,38 - 4,97$	$M = 4,78$ e $DP = 4,87$ $IC = 3,91 - 5,66$
Document	$M = 4,57$ e $DP = 4,18$ $IC = 4,24 - 4,91$	$M = 4,6$ e $DP = 4,85$ $IC = 3,74 - 5,49$
RPC	$M = 4,95$ e $DP = 4,73$ $IC = 4,33 - 5,58$	$M = 8,86$ e $DP = 3,67$ $IC = 5,37 - 12,34$
Literal	$M = 4,57$ e $DP = 4,59$ $IC = 4,21 - 4,93$	$M = 4,63$ e $DP = 4,85$ $IC = 3,74 - 5,51$
Encoded	$M = 4,91$ e $DP = 4,64$ $IC = 4,18 - 5,65$	$M = 8,83$ e $DP = 4,02$ $IC = 4,65 - 13,02$
Parâmetros Entrada	$M = 3,53$ e $DP = 3,27$ $IC = 3,31 - 3,75$	$M = 3,58$ e $DP = 3,43$ $IC = 2,97 - 4,18$
Parâmetros Saída	$M = 4,25$ e $DP = 4,26$ $IC = 3,95 - 4,55$	$M = 3,78$ e $DP = 3,37$ $IC = 3,16 - 4,41$

Document/Literal os números médios por operação não ultrapassa, na maioria dos casos, a quantidade de 6 características por documento WSDL. Contudo para as características *RPC/Encoded*, no *Binding SOAP 1.2*, a média é de aproximadamente 9 unidades. Como comentado anteriormente, poucos documentos que definem o *Binding SOAP 1.2* utilizam esse tipo de Requisição/Codificação (7 arquivos). Por sua vez, a quantidade de parâmetros de entrada para ambos os tipos de *Binding* ficam em um intervalo aproximado de 3 a 4 parâmetros por operação. Os desvios padrão para essas características seguem o comportamento das demais características indicando que em geral esses valores também não excedem 5 unidades. Os intervalos de confiança, para esse caso, mostram que a quantidade de parâmetros de entrada em ambos os *Binding* são iguais. As médias para a quantidade de parâmetros de saída são ligeiramente superiores as médias para os parâmetros de entrada. Para o *Binding SOAP 1.1* é possível afirmar, estatisticamente por meio dos intervalos de confiança, que as quantidades de parâmetros de saída são superiores ao parâmetros de entrada. Contudo, essa afirmação não pode ser feita para o *Binding SOAP 1.2*, pois os intervalos de confiança para os parâmetros de entrada e saída estão sobrepostos. Vale ressaltar que os cálculos das variáveis apresentadas na Tabela 6.9 são realizados sobre dados filtrados desconsiderando os documentos que não possuem a característica em questão, e também, documentos que possuem valores superiores a 20 unidades. Essas variáveis quando calculadas considerando todo o conjunto de documentos WSDL possuem médias e desvios padrão muito superiores. Contudo, a maioria dos *web services* possuem quantidades relativamente baixas para as características analisadas.

6.5 Binding HTTP

Como comentado anteriormente na seção 5.2, o *Binding HTTP* é utilizado para que um *browser Web* acesse um serviço através da utilização dos métodos *GET* e *POST* providos pelo protocolo *HTTP*. Nesse tipo de *Binding* não ocorre o uso da mensagem *SOAP* na requisição de um serviço. Os parâmetros dos métodos são passados na *URL* da requisição, no caso do método *GET*, e no corpo da mensagem da requisição, no caso do método *POST*. A resposta da requisição do serviço pode ser retornada em forma de uma mensagem *SOAP* ou não. Geralmente, em testes realizados em *Web services* comerciais o retorno das requisições de serviços com o *Binding HTTP* é realizado através de um documento no formato *XML*. O *Binding HTTP* é utilizado em aproximadamente 28% (379 arquivos) dos documentos *WSDL* encontrados.

Figura 6.29: HTTP: Gráfico de Pizza do Tamanho dos Documentos WSDL.

6.5.1 Tamanhos dos documentos WSDL

Os resultados obtidos para o tipo *Binding HTTP* mostram que 37% dos arquivos possuem até 10KB e 68% são menores que 30KB. A quantidade de documentos que possuem tamanhos de até 100KB representa 91% dos dados obtidos. Nenhum documento, com tamanho superior a 1MB, foi encontrado para o *Binding HTTP*. A Figura 6.29 mostra a proporção dos tamanhos dos arquivos em um gráfico de pizza. O gráfico é dividido em 10 categorias de 10KB e mais uma categoria compreendendo os documentos *WSDL* com tamanhos superiores a 100KB. Novamente, os resultados mostram que os arquivos *WSDL*, em sua maioria, não possuem tamanhos superiores a 30KB para todos os tipos de *Binding*.

6.5.2 Quantidade de Operações

A Figura 6.30 apresenta um gráfico de pizza com as proporções do número de operações encontrados nos documentos WSDL que definem o *Binding HTTP*. Os resultados demonstram que 50% dos documentos disponibilizam menos de 6 operações. O número de operações em 76% dos documentos analisados não excede a quantidade de 20 operações.

Figura 6.30: HTTP: Gráfico de Pizza do Número de Operações.

Figura 6.31: HTTP: Ajuste de Distribuição para o Número de Operações.

Ajuste de Distribuições para o Número de Operações

A distribuição que melhor se ajusta as quantidades de operações encontradas nos documentos *WSDL* é a Geométrica com o parâmetro $p = 0,089451$. A Figura 6.31 apresenta um gráfico que mostra as curvas dos dados coletados e da distribuição de ajuste para as freqüências acumuladas das quantidades de operações. Uma observação importante encontrada nos dados é que a quantidade de operações para todos os tipos de *Binding* obtém a mesma distribuição de ajuste.

Figura 6.32: HTTP: Gráfico de Linha para os Métodos GET e POST.

6.5.3 Métodos GET e POST

Em geral, os documentos *WSDL* que utilizam o *Binding HTTP* definem o acesso ao serviço utilizando os métodos *GET* e *POST*. Ou seja, as operações providas pelo serviço que utiliza esse tipo de *Binding* podem ser acessadas através desses dois métodos do protocolo *HTTP*. Isso é comprovado analisando os dois métodos através do gráfico de linha apresentado na Figura 6.32. As linhas representadas no gráfico estão praticamente sobrepostas. Isso indica a semelhança das quantidades de operações que utilizam os dois métodos *HTTP* simultaneamente. A maioria dos documentos possui a definição de ambos os métodos. A quantidade de operações que utilizam esses métodos na provisão do serviço para 84% dos documentos analisados não excede 20 unidades. A quantidade de documentos que possuem até 4 operações representa 54% do total analisado. A média de operações que utilizam os métodos *GET* e *POST* é de aproximadamente 7 operações por documento *WSDL*.

Ajuste de Distribuições para os Métodos GET e POST

A distribuição que melhor se ajusta as operações que utilizam o método *GET* é a *Gaussiana Inversa* com os parâmetros $\mu = 7.0819$ e $\lambda = 3.6261$. A Figura 6.33 apresenta um gráfico que mostra as curvas das freqüências para o método *GET*. O gráfico de ajuste de distribuição para o método *POST* é praticamente igual a distribuição de ajuste para o método *GET*. Uma pequena diferença pode ser observada no valor dos parâmetros que para o método *POST* é $\mu = 6.6350$ e $\lambda = 3.6363$.

Figura 6.33: HTTP: Ajuste de Distribuição para o Método GET.

6.5.4 Parâmetros de Entrada

O número médio de parâmetros de entrada é de aproximadamente 3 parâmetros por operação. Aproximadamente 55% dos documentos analisados possuem 2 parâmetros e 85% não excedem 4 parâmetros por operação. Menos de 1% dos documentos analisados possuem uma quantidade de parâmetros superior a 20 unidades. A distribuição de ajuste para os parâmetros de entrada é a *Weibull* com os parâmetros $\alpha = 1,7203$ e $\beta = 2,7834$. A Figura 6.34 mostra o gráfico de ajuste da distribuição para essa característica.

Uma vez mais, o principal tipo de dado utilizado como parâmetro de entrada é o tipo *String*. Aproximadamente 93% dos parâmetros de entrada para o *Binding HTTP* são definidos por tipos *String*. Os demais tipos dividem a definição dos 7% restantes. A análise dos tipos de dados apresenta uma informação importante sobre a complexidade dos dados. A média de utilização de tipos complexos encontrada nos documentos que definem o *Binding HTTP* é de 0.09 parâmetros *ComplexType* por operação. Em contrapartida, a quantidade média de tipos simples é de 2.5 por operação. Esses dados mostram que a utilização do *Binding HTTP* está relacionada

a serviços que não provêem uma complexidade no que diz respeito a estrutura dos dados de entrada.

Figura 6.34: HTTP: Ajuste de Distribuição para o Número de Parâmetros de Entrada.

6.5.5 Parâmetros de Saída

O número médio de parâmetros de saída por operação é de aproximadamente 7 parâmetros. As quantidades de parâmetros não excedem 10 parâmetros para 73% dos documentos analisados. As operações com mais de 20 parâmetros de saída correspondem a 6% dos dados analisados. A distribuição de ajuste para os parâmetros de saída é a *Pearson Tipo 5* com os parâmetros $\alpha = 1,1843$ e $\beta = 2,1245$. A Figura 6.35 mostra o gráfico de ajuste da distribuição.

Figura 6.35: HTTP: Ajuste de Distribuição para o Número de Parâmetros de Saída.

Assim como nos parâmetros de entrada, o principal tipo utilizado como parâmetro

de saída é o tipo `String`. No entanto, a proporção desse tipo de parâmetro é menor que a proporção encontrada para os parâmetros de entrada correspondendo a 64% do total de tipos de dados utilizados nos parâmetros de saída (dados semelhantes aos encontrados no *Binding SOAP* 1.1 e 1.2). Os tipos de dado `Double` e `DateTime` definem respectivamente 11% e 5% do total de parâmetros de saída. Tipos `Boolean` e `Int` definem cada um 3% dos parâmetros. Tipos `Float` e `Long` definem 2% e os tipos `Decimal` e `Base64Binary` correspondem a 1% dos dados cada um. Os demais tipos de dados dividem o restante da definição dos parâmetros.

6.5.6 Resumo das Distribuições Estatísticas para o Binding HTTP

A Tabela 6.10 mostra um resumo das distribuições estatísticas ajustadas nas características do *Binding HTTP*. Quatro tipos de distribuições são utilizados: *Geométrica*, *Gaussiana Inversa*, *Weibull* e *PearsonV*. O Anexo B apresenta as equações e os domínios para o cálculo dessas distribuições.

Tabela 6.10: HTTP: Resumo das Distribuições Estatísticas para o Binding HTTP

Característica	Nome da Distribuição	Parâmetros
Nro. Operações	Geométrica	$p = 0,089451$
GET	Gaussiana Inversa	$\mu = 7,0819$ e $\lambda = 3,6261$
POST	Gaussiana Inversa	$\mu = 6,6350$ e $\lambda = 3,6363$
Parâmetros Entrada	Weibull	$\alpha = 1,7203$ e $\beta = 2,7834$
Parâmetros Saída	PearsonV	$\alpha = 1,1843$ e $\beta = 2,1245$

Tabela 6.11: HTTP: Resumo Total das Variáveis Estatísticas para o Binding HTTP.

Características	Max	Min	Média	Desvio Padrão
Nro. Operações	118	0	14,28	20,20
GET	59	0	7,18	10,25
POST	59	0	7,08	10,09
Parâmetros Entrada	38	0	2,62	2,85
Parâmetros Saída	66	0	7,55	10,92

Tabela 6.12: HTTP: Resumo Parcial das Variáveis Estatísticas para o Binding HTTP.

Características	Max	Min	Média	Desvio Padrão
Nro. Operações	20	1	6,48	5,25
GET	20	1	4,84	4,66
POST	20	1	4,8	4,66
Parâmetros Entrada	20	1	2,65	2,16
Parâmetros Saída	20	1	5,18	5,49

6.5.7 Resumo das Variáveis Estatísticas para o Binding HTTP

As Tabelas 6.11 e 6.12 mostram um resumo dos valores obtidos para cada uma das características do *Binding HTTP*. A Tabela 6.12 mostra as variáveis estatísticas considerando todos os documentos WSDL analisados. Por sua vez, a Tabela 6.11, apresenta as variáveis estatísticas filtrando os valores dos documentos que não possuem a característica em questão, e também, valores superiores a 20 unidades.

6.6 SOAP 1.1 vs SOAP 1.2 vs HTTP

Esta seção tem o objetivo de apresentar um resumo das distribuições de ajuste e das variáveis estatísticas para todos os tipos de *Binding*. As Tabelas 6.13 e 6.14 mostram quadros comparativos para a análise das características que são comuns entre os três tipos de *Binding*. As características consideradas são: número de operações, parâmetros de entrada e saída.

A Tabela 6.13 mostra a definição das distribuições de ajuste para as características comuns aos três tipos de *Binding*. No caso do número de operações, a distribuição Geométrica obteve o melhor ajuste para todos os tipos *Binding* considerados. A diferença entre elas é o valor do parâmetro p . Os parâmetros de entrada também possuem a mesma distribuição de ajuste: a Weibull. No entanto, o comportamento da ocorrência da quantidade de parâmetros de saída para todos os *Bindings* é diferente. Cada um dos tipos de *Binding* obtiveram a sua própria distribuição de ajuste.

Para o número de operações, as variáveis mostram que a maioria dos *web services* não provê mais de 8 operações. Para os tipos *Binding SOAP* esse valor não ultrapassa 6 operações considerando os intervalos de confiança. As variáveis mostram que as quantidades de operações para os tipos de *Binding SOAP*, estatisticamente, são iguais. Contudo, o *Binding HTTP* possui maiores quantidades de operação por documento WSDL. Os parâmetros de entrada para os *Binding SOAP*, como comentado na seção 6.4 possuem quantidades semelhantes com valores entre 3 e 4. Entretanto, a quantidade de parâmetros de entrada para o tipo de *Binding HTTP* é menor que ambos os *Binding SOAP*. O *Binding HTTP* possui os maiores valores para os parâmetros de saída, enquanto os *Binding SOAP* possuem valores estatisticamente iguais e menores comparados aos parâmetros de saída do *Binding HTTP*.

É importante destacar que os cálculos das variáveis apresentadas na Tabela 6.14 são realizados sobre dados filtrados desconsiderando os documentos que não possuem a característica em questão, e também, documentos que possuem valores superiores a 20 unidades. Essas variáveis quando calculadas considerando todo o conjunto de documentos WSDL possuem médias e desvios padrão muito superiores. Contudo, a maioria dos *web service* possuem quantidades relativamente baixas para

as características analisadas.

Tabela 6.13: Tabela Comparativa de Distribuições de Ajuste

Características	Binding SOAP 1.1	Binding SOAP 1.2	Binding HTTP
Nro. Operações	Geométrica $p = 0,12910$	Geométrica $p = 0,13660$	Geométrica $p = 0,089451$
Parâmetros Entrada	Weibull $\alpha = 1,2937$ e $\beta = 2,9991$	Weibull $\alpha = 1,5496$ e $\beta = 3,1854$	Weibull $\alpha = 1,7203$ e $\beta = 2,7834$
Parâmetros Saída	Lognormal $\mu = 5,3525$ e $\sigma = 9,7205$	Beta-Generalizada $\alpha_1 = 0,48223$, $\alpha_2 = 1,2088$ $min = 0,0$ e $max = 10,109$	PearsonV $\alpha = 1,1843$ e $\beta = 2,1245$

Tabela 6.14: Tabela Comparativa de Variáveis Estatísticas

Características	Binding SOAP 1.1	Binding SOAP 1.2	Binding HTTP
Nro. Operações	$M = 4,67$ e $DP = 4,33$ $IC = 4,38 - 4,97$	$M = 4,78$ e $DP = 4,87$ $IC = 3,91 - 5,66$	$M = 6,48$ e $DP = 5,25$ $IC = 5,76 - 7,2$
Parâmetros Entrada	$M = 3,53$ e $DP = 3,27$ $IC = 3,31 - 3,75$	$M = 3,58$ e $DP = 3,43$ $IC = 2,97 - 4,18$	$M = 2,65$ e $DP = 2,16$ $IC = 2,39 - 2,92$
Parâmetros Saída	$M = 4,25$ e $DP = 4,26$ $IC = 3,95 - 4,55$	$M = 3,78$ e $DP = 3,37$ $IC = 3,16 - 4,41$	$M = 5,18$ e $DP = 5,49$ $IC = 4,45 - 5,91$

6.7 Considerações Finais

Este capítulo apresentou as análises estatísticas para o conjunto de documentos WSDL obtidos através do módulo de busca da ferramenta *WSDLAnalyzer*. Essas análises foram realizadas utilizando o *OpenOffice Calc*, o software *BestFit* e o módulo de análise da ferramenta *WSDLAnalyzer*. O módulo de análise foi responsável por extrair todas as características dos documentos WSDL disponíveis e prover uma análise estatística inicial de cada uma dessas informações. O *OpenOffice Calc* foi utilizado para realizar análises estatísticas complementares que puderam filtrar características com pouca ocorrência. Por sua vez, o software *BestFit* foi utilizado para ajustar distribuições de probabilidade sobre as características analisadas.

Através da análise apresentada neste capítulo, pode-se avaliar o número médio de operações providas pelos *Web services* descritos nos documentos WSDL. Além disso, informações sobre o estilo de requisição e tipos de codificações utilizados para acessar os *Web services* foram discutidas para cada um dos tipos de *Binding SOAP*. No caso do *Binding HTTP*, foram realizadas análises sobre os métodos *GET* e *POST*. Por fim, os parâmetros de entrada e saída para cada um dos tipos de *Binding* foram analisados.

As análises apresentadas neste capítulo mostram que os *Web services* encontrados na Web não são complexos, pois o tamanho médio dos documentos WSDL em todos os tipos de *Binding* não ultrapassam, na maioria dos arquivos analisados, o

tamanho 30KB. No que diz respeito a quantidade de operações providas, os *Web services* também não podem ser considerados complexos, pois a quantidade média de operações não excede a quantia de 8 operações por serviço. O tipo de dado mais utilizado na definição dos parâmetros de entrada e saída é o tipo *String*. A quantidade parâmetros de saída para os tipos *Binding 1.1* e *Binding HTTP* é maior que a quantidade de parâmetros de entrada. No *Binding 1.2* não houve comprovação estatística para essa relação de parâmetros de entrada/saída. Na maioria das vezes, os estilos de requisição *Document* e *RPC* são associados com os tipos de codificação *Literal* e *Encoded* respectivamente.

O Capítulo 7 apresenta as conclusões obtidas com o desenvolvimento deste trabalho, e também, as sugestões de trabalhos futuros.

Conclusão

*E*sta dissertação de mestrado contribui para a caracterização de cargas de trabalho para *Web services*, por meio da análise de documentos *WSDL*. Essa caracterização de carga, dentro do contexto dos *Web Services*, tem sido pouco explorada na literatura devido a dois fatores: (1) falta de uma quantidade de *traces* de acesso a esse tipo de serviço que garanta uma caracterização confiável e (2) a grande diversidade de funcionalidades envolvidas com *Web services*.

A abordagem utilizada nesta dissertação é a análise dos documentos *WSDL*, que descrevem o modo como os serviços são acessados. Por meio dessa análise pode-se caracterizar os serviços requisitados em termos de complexidade, quantidade de parâmetros de entrada/saída, dentre outras características. Com o intuito de viabilizar a obtenção e análise das *WSDLs*, propõe-se uma ferramenta (*WSDLANalyzer*) que automatiza essa tarefa. A *WSDLANalyzer* permite não apenas analisar as *WSDLs* obtidas para este trabalho mas também futuras investigações envolvendo tanto pesquisas genéricas quanto com enfoque particularizado para alguma característica. Um exemplo de um enfoque particular pode ser atribuído a *WSDLs* relacionadas ao comércio eletrônico ou a qualquer outra forma de utilização. A *WSDLANalyzer* possibilitou a obtenção e análise de 1346 documentos, que resultou em um estudo sobre as características dos serviços oferecidos.

Dessa forma, o objetivo principal deste trabalho é alcançado uma vez que define-se uma metodologia para a caracterização de cargas de trabalho para os *Web services* e demonstra-se as informações que podem ser avaliadas a partir da análise dos resultados obtidos. A próxima seção enumera as contribuições alcançadas durante o desenvolvimento deste trabalho.

7.1 Resultados e Contribuições

O projeto de mestrado apresentado neste trabalho contribui para área de avaliação de desempenho no que se refere a caracterização de cargas de trabalho de *Web services*. Dentre elas, destacam-se:

1. Uma proposta para caracterização de cargas de trabalho para *Web services*. Metodologias para caracterização de cargas de trabalho baseadas em *traces* são bastante difundidas em trabalhos de avaliação de desempenho em servidores *Web*. O grupo de pesquisa em Sistemas Distribuídos e Programação Concorrente (GSDPC-ICMC/USP) tem expertise em trabalhos com esse enfoque. A aplicação de uma metodologia baseada em *traces*, para a caracterização de cargas de trabalho em *Web services*, seria interessante, se alguns obstáculos não impossibilitassem seu uso. Esse tipo de metodologia não pôde ser aplicado para o contexto dos *Web services* devido a dificuldade de se encontrar uma quantidade significativa de *traces*, na indústria ou em instituições acadêmicas, que permitam o desenvolvimento deste trabalho. Alternativamente, uma abordagem para a caracterização de cargas de trabalho baseadas em documentos *WSDL* é sugerida.
2. Construção de uma ferramenta denominada *WSDLANalyzer* para a busca e análise de documentos *WSDL*. A principal funcionalidade dessa ferramenta é a busca e manipulação de documentos *WSDL* com o objetivo de extrair informações que indiquem o modo como as mensagens trocadas entre consumidores e provedores de serviço são construídas. A ferramenta *WSDLANalyzer* é dividida em dois módulos: busca e análise. O módulo de busca implementa um mecanismo para a obtenção de arquivos *WSDL* na *Web*. Esse mecanismo é composto por duas abordagens. A primeira abordagem obtém dados através da *engine* de busca do *Google*, utilizando uma *API* chamada de *Google SOAP Search*. A segunda abordagem é responsável por extrair *hyperlinks* com referências para arquivos *WSDL* em *Web sites* especializados em prover listas de *Web services*. Por sua vez, o módulo de análise é responsável por extrair as informações dos documentos *WSDL*, obtidos pelo módulo de busca, que podem ser utilizadas para a caracterização dos *WSDLs* e consequentemente dos serviços requisitados. O módulo de análise extrai informações de cada um dos tipos de *Binding* encontrados nos documentos *WSDL*. Esses tipos de *Binding* são responsáveis por definir o modo de acesso ao serviço. Para cada tipo de *Binding* podem ser analisadas as seguintes características:

- A complexidade do serviço através do tamanho dos documentos *WSDL*.

- O número de operações disponíveis pelo serviço.
 - Os estilos de requisição utilizados nas operações.
 - Os tipos de codificação utilizados na construção das mensagens.
 - As quantidades e os tipos de dados utilizados como parâmetros de entrada e saída nas operações.
3. Resultados da análise do conjunto de dados obtidos. A procura por documentos WSDL na Web, utilizando o módulo de busca da ferramenta *WSDLAnalyzer*, resultou na obtenção de 1346 documentos WSDL. Esses documentos, após serem examinados pelo módulo de análise da ferramenta, foram convertidos em um grande conjunto de informações quantitativas sobre as características que influenciam a composição das mensagens de requisição/resposta. Para cada documento WSDL, foram extraídas 57 informações dentre tipos de dados utilizados nas operações, tipos de *Binding*, número de operações e tamanhos de arquivo. As planilhas de dados brutos (sem um processo de análise estatística) são compostas de 1346 x 57 informações passíveis de serem manipuladas. A grande quantidade de dados exige um esforço considerável para a realização de análises estatísticas complementares às realizadas automaticamente pela ferramenta *WSDLAnalyzer*. Como resultado dessas análises pode-se chegar a diversas conclusões, dentre elas cabe destacar:
- A maioria dos *Web services* disponíveis atualmente não implementam sistemas complexos. Aproximadamente 90% dos documentos encontrados não possuem tamanhos superiores a 100KB.
 - O número médio de operações providas pelos os *Web services*, em todos os tipos de *Binding*, não ultrapassa a quantia de 6 operações por documento WSDL.
 - O tipo mais utilizado de Requisição/Codificação, para a construção das mensagens SOAP, é o tipo *Document/Literal*. Isso mostra os esforços da comunidade de desenvolvedores em *Web services* em utilizar o estilo de Requisição/Codificação que possui o maior nível de interoperabilidade.
 - Os documentos WSDL que definem o tipo de *Binding HTTP*, em geral, implementam os métodos *GET* e *POST*, de forma conjunta, para acessar o serviço.
 - O protocolo SOAP 1.2 ainda é pouco utilizado sendo definido em apenas 14% dos documentos WSDL analisados.
 - A quantidade de parâmetros de saída, na maioria das vezes, é maior que a quantidade de dados de entrada. As análises mostram que para os tipos

de *Binding SOAP 1.1* e *Binding HTTP* essa afirmação é estatisticamente comprovada. No entanto, para o tipo de *Binding SOAP 1.2* não houve essa comprovação estatística avaliando-se os intervalos de confiança.

- O tipo de dado mais utilizado na definição dos parâmetros de entrada e saída é o tipo `String`. Em [Head et al., 2005] é apresentado, dentre outras análises, um estudo sobre a influência de alguns tipos de dados no desempenho final dos *Web services*.
- Tipos de dados complexos (`ComplexType`) são mais utilizados nos tipos de *Binding SOAP 1.1* e *1.2*. O *Binding HTTP* utiliza, em sua maioria, dados simples (`SimpleType`).
- Os altos desvios padrão encontrados para as características analisadas indicam uma variação muito grande da quantidade das características analisadas nos documentos analisados.
- As mesmas distribuições de ajuste definem, para todos os tipos de *Binding*, os comportamentos sobre a ocorrência das quantidades de operações e números de parâmetros de entrada. As distribuições de ajuste para os parâmetros de saída são diferentes.

4. Artigos publicados em eventos científicos:

- TAVARES, T. C.; SANTANA, Regina Helena Carlucci; ESTRELLA, J. C.; SANTANA, Marcos José. Caracterização dos Dados Trocados Entre Serviços Web e Provedores de Serviço. In: *XIV Simpósio Brasileiro de Sistemas Multimídia e Web*, 2008, Vila Velha - ES. Anais do XIV Simpósio Brasileiro de Sistemas Multimídia e Web, 2008. v. 2.
- ESTRELLA, J. C.; SANTANA, Regina Helena Carlucci; SANTANA, Marcos José; TAVARES, T. C.; TARDIOLE, B.; SILVA, J. C. F; LEONARDO,. Diretrizes para Avaliação de Desempenho de Web Services. In: *XXVIII Brazilian Computing Society Congress*, 2008, Belém - PA. Workshop on Performance of Computational and Communication Systems, 2008. p. 111-126.

7.2 Trabalhos Futuros

A caracterização de carga de trabalho apresentada neste trabalho, por se tratar de um estudo inicial, pode ser ampliada. As seguintes sugestões para trabalhos futuros são apresentadas:

- Extensão da ferramenta *WSDLAnalyzer* para a provisão de ajuste de distribuições estatísticas sobre os dados analisados. Um trabalho nesse sentido seria

interessante para que o usuário da ferramenta não tenha a preocupação em utilizar softwares complementares para a execução dessa tarefa.

- Estudo sobre a aplicação dos conceitos da *Web* semântica e técnicas de utilização de ontologias para a categorização dos documentos *WSDL*. O trabalho apresentado nesta dissertação considerou os *Web services* de uma maneira geral. A divisão de tipos (funcionalidades) de *Web services* pode trazer resultados interessantes em relação a caracterização de cargas de trabalho. Além disso, pode-se avaliar diferentes tipos de ontologias como *OWL-S* [Martin et al., 2007], *WSMO* [Fensel et al., 2006], *SWSF* [Battle et al., 2005] e *SAWSDL* [Farrell and Lausen, 2007] e verificar qual delas é melhor aplicada no contexto de caracterização de cargas.
- Aplicação de técnicas de mineração. A grande quantidade de dados dificulta encontrar relações entre as características analisadas. Assim, as técnicas de mineração de dados podem auxiliar na verificação de padrões eventualmente presentes nas características analisadas. Esse tipo de técnica permite a extração automatizada de conhecimento sobre grandes conjuntos de dados.
- Geração automática de serviços. Os resultados obtidos definem, através de um sumário de distribuições estatísticas, as probabilidades de uma determinada quantidade de características ocorrer nos documentos *WSDL*. Dessa forma, é possível implementar um software de geração automática de serviço através da utilização de ferramentas como a *WSDL2Java*. Essa ferramenta é capaz de transformar um documento *WSDL* em um *Web service* na linguagem Java. O processo de criação do serviço seria feito da seguinte forma: (1) cria-se um documento *WSDL* com as características providas pelo sumário de distribuição, e então, traduz-se esse documento para um *Web service* na linguagem Java utilizando a ferramenta *WSDL2Java*. Os serviços gerados poderiam ser utilizados em trabalhos de avaliação de desempenho em *Web services*.

Gráficos Complementares

A.1 Binding SOAP 1.1 - Gráficos dos Tipos de Dados

A.1.1 Distribuições Estatísticas para Parâmetros de Entrada

Este apêndice apresenta as distribuições estatísticas para o Binding SOAP 1.1 que melhor se ajustam aos tipos de dados de entrada analisados. Essas distribuições são ajustadas para três tipos de dados: `ComplexType`, `SimpleType` e `String`. O ajuste para os demais tipos não foi possível devido a pouca ocorrência nos documentos WSDL. Os valores dos tipos de dados nos gráficos apresentados são a média da ocorrência dos tipos por operação.

Complex Type

A Distribuição que se ajustou aos tipos de dados complexos foi a Gaussiana Inversa com os parâmetros $\mu = 1.8066$ e $\lambda = 6.0031$. A Figura A.1 mostra o gráfico de ajuste de distribuição para os dados coletados.

Simple Type

A Beta-Generalizada é a distribuição que melhor se ajusta aos valores obtidos para os tipos de dados simples. Os parâmetros da função são: $\alpha_1 = 0.15338$, $\alpha_2 = 1.9424$, $min = 0.0$ e $max = 9.6595$. A Figura A.2 mostra o gráfico de ajuste de distribuição para os dados coletados.

Figura A.1: SOAP1.1: Ajuste de Distribuição para o Número de ComplexType.

Figura A.2: SOAP1.1: Ajuste de Distribuição para o Número de SimpleType.

Tipos String

A distribuição que melhor se ajustou aos tipos de dados string é a Lognormal com os parâmetros $\mu = 2.8083$ e $\sigma = 2.0150$. A Figura A.3 mostra o gráfico de ajuste de distribuição para os dados coletados.

A.1.2 Distribuições Estatísticas para Parâmetros de Saída

Assim como para os parâmetros de entrada, as distribuições estatísticas para os tipos de dados de saída são ajustadas para ComplexType, SimpleType e String. Nenhuma distribuição se ajusta de maneira satisfatória aos demais tipos de dados.

Figura A.3: SOAP1.1: Ajuste de Distribuição para o Número de String.

Complex Type

A Distribuição que se ajustou aos tipos de dados complexos foi a Gaussiana Inversa com os parâmetros $\mu = 2.8891$ e $\lambda = 7.4321$. A Figura A.4 mostra o gráfico de ajuste de distribuição para os dados coletados.

Figura A.4: SOAP1.1: Ajuste de Distribuição para o Número de ComplexType.

Simple Type

A Logística é a distribuição que melhor se ajusta aos valores obtidos para os tipos de dados simples. Os parâmetros da função são: $\alpha = 0.24601$, $\beta = 0.32626$. A Figura A.5 mostra o gráfico de ajuste de distribuição para os dados coletados.

Figura A.5: SOAP1.1: Ajuste de Distribuição para o Número de SimpleType.

Tipos String

A distribuição que melhor se ajustou aos tipos de dados string é a Lognormal com os parâmetros $\mu = 2.6176$ e $\sigma = 5.3799$. A Figura A.6 mostra o gráfico de ajuste de distribuição para os dados coletados.

Figura A.6: SOAP1.1: Ajuste de Distribuição para o Número de String.

A.1.3 Comparação Entre as Distribuições

A Tabela A.1 apresenta uma comparação entre as distribuições dos tipos de dados de entrada e saída. Essa tabela apresenta as distribuições para os tipos ComplexType, SimpleType e String. As principais distribuições de ajuste utilizadas para os parâmetros de entrada e saída são: Gaussiana Inversa, Beta-Generalizada,

Logística e Lognormal.

Tabela A.1: Comparação das Distribuições Estatísticas dos Tipos de Dados de Entrada e Saída

Tipos de Dados	Distribuições de Entrada	Distribuições de Saída
ComplexType	Gaussiana Inversa - $\mu = 1.8066$ e $\lambda = 6.0031$	Gaussiana Inversa - $\mu = 2.8891$ e $\lambda = 7.4321$
SimpleType	Beta-Generalizada - $\alpha_1 = 0.15338$, $\alpha_2 = 1.9424$, $min = 0.0$ e $max = 9.6595$	Logística - $\alpha = 0.24601$, $\beta = 0.32626$
String	Lognormal - $\mu = 2.8083$ e $\sigma = 2.0150$	Lognormal - $\mu = 2.6176$ e $\sigma = 5.3799$

A.2 Binding SOAP 1.2 - Gráficos

Este apêndice apresenta as distribuições estatísticas para algumas das características do Binding SOAP 1.2. A distribuição para o número de operações é dada na seção A.2.1. Por sua vez, as distribuições de ajuste para os tipos de dados é apresentada na seção A.2.2.

A.2.1 Distribuição Estatística para o Número de Operações

A distribuição estatística que melhor se ajusta ao número de operações por documento WSDL é a Geométrica com o parâmetro $p = 0,13660$. A Figura A.7 mostra o gráfico com as curvas para os dados coletados e para a distribuição ajustada.

Figura A.7: SOAP 1.2: Ajuste de Distribuição para o Número de Operações.

A.2.2 Distribuição Estatística para Parâmetros de Entrada

A maioria dos tipos de dados utilizados para compor os parâmetros de entrada e saída não possuíam um número de ocorrência suficiente para o ajuste de uma

distribuição estatística. No caso do Binding SOAP 1.1 o ajuste de uma distribuição foi realizado apenas para os tipos complexos e String.

Complex Type

A Distribuição que se ajustou aos tipos de dados complexos foi a Gaussiana Inversa com os parâmetros $\mu = 2.0843$ e $\lambda = 26.4061$. A Figura A.8 mostra o gráfico de ajuste de distribuição para os dados coletados.

Figura A.8: SOAP 1.2: Ajuste de Distribuição para o Número de ComplexType.

Figura A.9: SOAP 1.2: Ajuste de Distribuição para o Número de String.

Tipos String

A distribuição que melhor se ajustou aos tipos de dados string é a Lognormal com os parâmetros $\mu = 2.8757$ e $\sigma = 1.6968$. A Figura A.9 mostra o gráfico de ajuste de distribuição para os dados coletados.

A.2.3 Distribuição Estatística para Parâmetros de Saída

Complex Type

A Distribuição que se ajustou aos tipos de dados complexos foi a Gaussiana Inversa com os parâmetros $\mu = 3.1315$ e $\lambda = 22.9177$. A Figura A.10 mostra o gráfico de ajuste de distribuição para os dados coletados.

Figura A.10: SOAP 1.2: Ajuste de Distribuição para o Número de ComplexType.

Figura A.11: Ajuste de Distribuição para o Número de String.

Tipos String

A distribuição que melhor se ajustou aos tipos de dados string é a Lognormal com os parâmetros $\mu = 2.4067$ e $\sigma = 0.37771$. A Figura A.11 mostra o gráfico de ajuste de distribuição para os dados coletados.

A.2.4 Comparação Entre as Distribuições

A Tabela A.2 apresentada uma comparação entre as distribuições dos tipos de dados de entrada e saída. Essa tabela apresenta as distribuições para os tipos ComplexType e String. As principais distribuições de ajuste utilizadas para os parâmetros de entrada e saída são: Gaussiana Inversa e Lognormal.

Tabela A.2: Comparação das Distribuições Estatísticas dos Tipos de Dados de Entrada e Saída

Tipos de Dados	Distribuições de Entrada	Distribuições de Saída
ComplexType	Gaussiana Inversa - $\mu = 2.0843$ e $\lambda = 26.4061$	Gaussiana Inversa - $\mu = 3.1315$ e $\lambda = 22.9177$
String	Lognormal - $\mu = 2.8757$ e $\sigma = 1.6968$	Lognormal - $\mu = 2.4067$ e $\sigma = 0.37771$

Funções de Distribuição de Probabilidade

B.1 Beta-Generalizada

B.1.1 Parâmetros e Domínio

Parâmetros

$$\alpha_1 > 0, \alpha_2 > 0 \text{ e } \min < \max$$

Domínio

$$\min \leq x \leq \max$$

B.1.2 Função Acumulativa

$$F(x) = \frac{B_z(\alpha_1, \alpha_2)}{B(\alpha_1, \alpha_2)} \equiv I_z(\alpha_1, \alpha_2) \quad (\text{B.1})$$

$$\text{with } z \equiv \frac{x - \min}{\max - \min} \quad (\text{B.2})$$

B.2 Weibull

B.2.1 Parâmetros e Domínio

Parâmetros

$$\alpha > 0 \text{ e } \beta > 0$$

Domínio

$$0 \leq x < +\infty$$

B.2.2 Função Acumulativa

$$F(x) = 1 - e^{-(x/\beta)^\alpha} \quad (\text{B.3})$$

B.3 Lognormal

B.3.1 Parâmetros e Domínio

Parâmetros

$$\mu > 0 \text{ e } \sigma > 0$$

Domínio

$$0 < x < +\infty$$

B.3.2 Função Acumulativa

$$F(x) = \Phi \left(\frac{\ln(x) - \mu'}{\sigma'} \right) \quad (\text{B.4})$$

$$\text{with } \mu' \equiv \ln \left[\frac{\mu^2}{\sqrt{\sigma^2 + \mu^2}} \right] \quad (\text{B.5})$$

$$\text{and } \sigma' \equiv \sqrt{\ln \left[1 + \left(\frac{\sigma}{\mu} \right)^2 \right]} \quad (\text{B.6})$$

B.4 Pearson

B.4.1 Parâmetros e Domínio

Parâmetros

$$\alpha > 0 \text{ e } \beta > 0$$

Domínio

$$0 \leq x < +\infty$$

B.4.2 Função Acumulativa

$$f(x) = \frac{1}{\beta \Gamma(\alpha)} \cdot \frac{e^{-\beta/x}}{(x/\beta)^{\alpha+1}} \quad (\text{B.7})$$

B.5 Geométrica

B.5.1 Parâmetros e Domínio

Parâmetros

$$0 < p \leq 1$$

Domínio

$$0 \leq x < +\infty$$

B.5.2 Função Acumulativa

$$F(x) = 1 - (1 - p)^{x+1} \quad (\text{B.8})$$

B.6 Gaussiana Inversa

B.6.1 Parâmetros e Domínio

Parâmetros

$$\mu > 0 \text{ e } \sigma > 0$$

Domínio

$$x > 0$$

B.6.2 Função Acumulativa

$$F(x) = \Phi \left[\sqrt{\frac{\lambda}{X}} \left(\frac{X}{\mu} - 1 \right) \right] + e^{2\lambda/\mu} \cdot \Phi \left[-\sqrt{\frac{\lambda}{X}} \left(\frac{X}{\mu} + 1 \right) \right] \quad (\text{B.9})$$

Referências Bibliográficas

- Agarwal, V., Chafle, G., Mittal, S., and Srivastava, B. (2008). Understanding approaches for web service composition and execution. In *Compute '08: Proceedings of the 1st Bangalore annual Compute conference*, pages 1–8, New York, NY, USA. ACM.
- Apache (2008). Apache axis2. Disponível em: <http://ws.apache.org/axis2/>. Último acesso: 24/05/2009.
- Apache (2009a). Apache commons - commons math. Disponível em: <http://commons.apache.org/math/userguide/stat.html>. Último acesso: 29/05/2009.
- Apache (2009b). Web services - axis. Disponível em: <http://ws.apache.org/axis/>. Último acesso: 10/04/2009.
- Arlitt, M. F. and Williamson, C. L. (1996). Web server workload characterization: the search for invariants. *SIGMETRICS Perform. Eval. Rev.*, 24(1):126–137.
- Battle, S., Bernstein, A., Boley, H., Grosz, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005). Semantic web services framework (swsf) overview – w3c member submission. Disponível em: <http://www.w3.org/Submission/SWSF/>. Último acesso: 14/05/2009.
- Berardi, D., Calvanese, D., and Giacomo, G. D. (2005). Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1-2):70 – 118.
- Butek, R. (2005). Which style of wsdl should i use? Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/>. Último acesso: 07/06/2009.
- Cai, M., Ghandeharizadeh, S., Schmidt, R. R., and Song, S. (2002). A comparison of alternative encoding mechanisms for web services. In *DEXA '02: Proceedings of the*

- 13th International Conference on Database and Expert Systems Applications*, pages 93–102, London, UK. Springer-Verlag.
- Carey, M. (2008). Soa what? *Computer*, 41(3):92–94.
- Carter, S. (2007). *The New Language of Business: SOA & Web 2.0*. IBM Press.
- Chen, S., Yan, B., Zic, J., Liu, R., and Ng, A. (2006). Evaluation and modeling of web services performance. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 437–444, Washington, DC, USA. IEEE Computer Society.
- Chen, S., Zic, J., Tang, K., and Levy, D. (2007). Performance evaluation and modeling of web services security. In *ICWS'07: Proceedings of the IEEE International Conference on Web Services*, pages 431–438.
- Cohen, F. (2003). Discover soap encoding's impact on web service performance. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-soapenc/>. Último acesso: 23/04/2009.
- Colan, M. (2004). Service-oriented architecture expands the vision of web service. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-soaintro.html>. Último acesso: 10/04/2009.
- ComputerWorld (2009). Gartner: arquitetura orientada a serviços é super valorizada. Disponível em: <http://computerworld.uol.com.br/gestao/2009/05/25/soa-empresas-criam-expectativas-exageradas-diz-gartner/>. Último acesso: 05/06/2009.
- Coulouris, G., Dollimore, J., and Kindberg, T. (2005). *Distributed Systems: Concepts and Design*. Addison Wesley.
- Davis, D. and Parashar, M. P. (2002). Latency performance of soap implementations. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 407, Washington, DC, USA. IEEE Computer Society.
- Deitel (2009). Multiple document interfaces with jdesktoppane and jinternal-frame. Disponível em: http://www.deitel.com/articles/java_tutorials/20060505/. Último acesso: 25/05/2009.
- Doculabs (2003). @bench report web services: Performance benchmark study. Disponível em: <http://www.formsite.com/doculabs/atbench/index.html>. Último acesso: 16/04/2009.

- Erl, T. (2005). *Service-Oriented Architecture : Concepts, Technology, and Design*. Prentice Hall PTR.
- Erl, T. (2007). *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR.
- Estrella, J. C., Santana, R. H. C., Santana, M. J., Tavares, T. C., Kuehne, B. T., Silva, J. C. F., and Paccanaro, L. C. (2008). Diretrizes para avaliação de desempenho de web services. In *Anais do XXVIII Congresso da Sociedade Brasileira de Computação - WPerformance*, pages 111–126.
- Farrell, J. and Lausen, H. (2007). Semantic annotations for wsdl and xml schema. Disponível em: <http://www.w3.org/TR/sawSDL/>.. Último acesso: 09/06/2009.
- Feitelson, D. G. (2002). Workload modeling for performance evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 114–141, London, UK. Springer-Verlag.
- Feitelson, D. G. and Nitzberg, B. (1995). Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860. In *Job Scheduling Strategies for Parallel Processing*, volume 949/1995, pages 337–360.
- Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., and Domingue, J. (2006). *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Fowler, M. and Scott, K. (2000). *UML Essencial: um breve guia para a linguagem padrão de modelagem de objetos*. Bookman.
- García, D. F., García, J., García, M., Peteira, I., García, R., and García, R. (2006). Benchmarking of web services platforms: An evaluation with tpc-app benchmark. In *WEBIST'06: International Conference on Web Information Systems and Technologies, 2006*, pages 1–6.
- Gill, P., Arlitt, M., Li, Z., and Mahanti, A. (2007). Youtube traffic characterization: a view from the edge. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 15–28, New York, NY, USA. ACM.
- Gómez-Martínez, E. and Merseguer, J. (2006). Impact of soap implementations in the performance of a web service-based application. In *Lecture Notes in Computer Science. Frontiers of High Performance Computing and Networking -ISPA 2006 Workshops.*, volume 4331, pages 884–896. Springer-Verlag.

- Gómez-Martínez, E. and Merseguer, J. (2006). Impact of soap implementations in the performance of a web service-based application. In *Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops*, volume 4331/2006, pages 884–896, Berlin, GE. Springer Berlin / Heidelberg.
- Google (2001). Google soap search api/. Disponível em: <http://code.google.com/apis/soapsearch/>. Último acesso: 18/05/2009.
- Hau, T., Ebert, N., Hochstein, A., and Brenner, W. (2008). Where to start with soa: Criteria for selecting soa projects. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, pages 314–314.
- Head, M. R., Govindaraju, M., Slominski, A., Liu, P., Abu-Ghazaleh, N., van Engelen, R., Chiu, K., and Lewis, M. J. (2005). A benchmark suite for soap-based communication in grid web services. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 19, Washington, DC, USA. IEEE Computer Society.
- HTMLParser (2006). Html parser library. Disponível em: <http://htmlparser.sourceforge.net/>. Último acesso: 19/05/2009.
- IBM (2009). Standards and web services. Disponível em: <http://www-128.ibm.com/developerworks/webservices/standards/>. Último acesso: 21/01/2009.
- Jankauskas, L. and McLafferty, S. (1996). Bestfit, distribution-fitting software by palisade corporation. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 551–555, Washington, DC, USA. IEEE Computer Society.
- JCP (2006). Java community process - java specification requests. Disponível em: <http://jcp.org/en/jsr/detail?id=110>. Último acesso: 19/05/2009.
- JFreeChart (2009). Charts for the java(tm) platform. Disponível em: <http://www.jfree.org/jfreechart/>. Último acesso: 25/05/2009.
- Li, Y., Liu, Y., Zhang, L., Li, G., Xie, B., and Sun, J. (2007). An exploratory study of web services on the internet. In *ICWS'07: Proceedings of the IEEE International Conference on Web Services*, pages 380–387, Salt Lake City, UT, USA. IEEE Computer Society.
- Lu, J., Yu, Y., Roy, D., and Saha, D. (2007). Web service composition: A reality check. In *WISE 2007: Web Information Systems Engineering, 2007*, volume 4831/2007, pages 523–532. Springer Berlin / Heidelberg.

- Machado, A. C. C. and Ferraz, C. A. G. (2005). Guidelines for performance evaluation of web services. In *WebMedia '05: Proceedings of the 11th Brazilian Symposium on Multimedia and the web*, pages 1–10, New York, NY, USA. ACM.
- Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D. L., Sirin, E., and Srinivasan, N. (2007). Bringing semantics to web services with owl-s. *World Wide Web*, 10(3):243–277.
- Martin-Flatin, J. P. and Löwe, W. (2007). Special issue on recent advances in web services. *World Wide Web*, 10(3):205–209.
- Menascé, D. A. and Almeida, V. A. F. (2002). *Planejamento de Capacidade para Serviços na Web: Métricas, Modelos e Métodos*. Editora Campus.
- Menascé, D. A., Ruan, H., and Gomaa, H. (2007). Qos management in service-oriented architectures. *Perform. Eval.*, 64(7-8):646–663.
- Microsoft (2004). Web services performance: Comparing javatm 2 enterprise edition (j2eetm platform) and .net framework: A response to sun microsystem's benchmark. Disponível em: [HTTP://download.microsoft.com/download/1/9/b/19bc8aa7-05fa-4e86-a612-c2cc181e4ee6/sun_ws_benchmark_response.pdf](http://download.microsoft.com/download/1/9/b/19bc8aa7-05fa-4e86-a612-c2cc181e4ee6/sun_ws_benchmark_response.pdf). Último acesso: 10/04/2009.
- OASIS (2004). Web services security (wss). Disponível em: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss. Último acesso: 25/04/2009.
- OASIS (2006). Uddi - (universal description, discovery, and integration). Disponível em: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec. Último acesso: 16/04/2009.
- Page-Jones, M. (2000). *Fundamentals of Object-Oriented Design in UML*. Makron Books.
- Palisade (2009). Bestfit developer kit. Disponível em: <http://www.palisade-br.com/devkits/bdk.asp>. Último acesso: 03/05/2009.
- Papazoglou, M. P. (2003). Service -oriented computing: Concepts, characteristics and directions. In *WISE '03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, page 3, Washington, DC, USA. IEEE Computer Society.
- Petrie, C., Margaria, T., Lausen, H., and Zaremba, M. (2008). *Semantic Web Services Challenge: Results from the First Year*. Springer Publishing Company, Incorporated.

- Silva, L. H. C. (2006). *Caracterização de Cargas de Trabalho para Testes de Modelos de Servidores Web*. Dissertação de mestrado, ICMC-USP, São Carlos, SP - Brasil.
- Simmonds, J. and Bastarrica, M. C. (2005). A tool for automatic uml model consistency checking. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 431–432, New York, NY, USA. ACM.
- Sun (2004). Web services performance: Comparing javatm 2 enterprise edition (j2eetm platform) and .net framework. Disponível em: http://java.sun.com/performance/reference/whitepapers/WS_Test-1_0.pdf. Último acesso: 27/05/2009.
- Sun (2008). Class message digest. Disponível em: <http://java.sun.com/javase/6/docs/api/java/security/MessageDigest.html>. Último acesso: 20/05/2009.
- Suzumura, T., Takase, T., and Tatsubori, M. (2005). Optimizing web services performance by differential deserialization. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 185–192, Washington, DC, USA. IEEE Computer Society.
- Tanenbaum, A. S. and van Steen, M. (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall.
- TPC (2009). Transaction processing performance. Disponível em: http://www.tpc.org/tpc_app/default.asp. Último acesso: 01/06/2009.
- W3C (2001). Web services description language (wsdl) 1.1. Disponível em: <http://www.w3.org/TR/wsdl>. Último acesso: 23/05/2009.
- W3C (2003). From soap/1.1 to soap version 1.2 in 9 points. Disponível em: <http://www.w3.org/2003/06/soap11-soap12.html>. Último acesso: 08/06/2009.
- W3C (2004a). Web services architecture. Disponível em: <http://www.w3.org/TR/ws-arch/>. Último acesso: 10/04/2009.
- W3C (2004b). Xml information set. Disponível em: <http://www.w3.org/TR/xml-infoset/>. Último acesso: 22/04/2009.
- W3C (2004c). Xml schema part 2: Datatypes second edition. Disponível em: <http://www.w3.org/TR/xmlschema-2/>. Último acesso: 04/05/2009.
- W3C (2007a). Latest soap versions. Disponível em: <http://www.w3.org/TR/soap/>. Último acesso: 10/04/2009.

- W3C (2007b). Web services description language (wsdl) version 2.0. Disponível em: <http://www.w3.org/TR/wsdl20/>. Último acesso: 10/04/2009.
- W3C (2009). Extensible markup language. Disponível em: <http://www.w3.org/XML/>. Último acesso: 10/04/2009.
- Weber, S. and Hariharan, R. (2003). A new synthetic web server trace generation methodology. In *ISPASS '03: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 80–90, Washington, DC, USA. IEEE Computer Society.
- Weerawarana, S., Curbera, F., Leymann, F., Storey, T., and Ferguson, D. F. (2005). *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR.
- Williams, A., Arlitt, M., Williamson, C., and Barker, K. (2005). Web workload characterization: Ten years later. In *Web Content Delivery*, volume 2, pages 3–21. Springer US.
- WS-I (2009). Establishing best practices for web services standards. Disponível em: <http://www.ws-i.org/>. Último acesso: 09/06/2009.
- WSDL4J (2006). Web services description language for java - <http://sourceforge.net/projects/wsdl4j>. Disponível em: <http://sourceforge.net/projects/wsdl4j>. Último acesso: 19/05/2009.
- WSTest (2009). Benchmark and test for wsdl based web services performance. Disponível em: <https://wstest.dev.java.net/>. Último acesso: 07/04/2009.
- Yoo, R., Lee, H., Chow, K., and Lee, H.-H. (2006). Constructing a non-linear model with neural networks for workload characterization. In *Proceedings of the IEEE International Symposium on Workload Characterization*, pages 150–159.
- Yu, Q., Liu, X., Bouguettaya, A., and Medjahed, B. (2008). Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, 17(3):537–572.
- Yu, Y., Lu, J., Fernandez-Ramil, J., and Yuan, P. (2007). Comparing web services with other software components. In *ICWS'07: Proceedings of the IEEE International Conference on Web Services*, volume 0, pages 388–397, Los Alamitos, CA, USA. IEEE Computer Society.
- Zilora, S. and Ketha, S. (2008). Think inside the box! optimizing web services performance today [web services in telecommunications, part ii]. *Communications Magazine, IEEE*, 46(3):112–117.