Modelos e algoritmos para composição de Web Services com qualidade de serviço

Bruno Tardiole Kuehne

Modelos e algoritmos para composição de Web Services com qualidade de serviço

Bruno Tardiole Kuehne

Orientador: ***Prof. Dr. Marcos José 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

Outubro/2009

Agradecimentos

Agradeço a Deus por ter conseguido chegar até aqui, aos meus pais, Beno e Maria de Lourdes que além do apoio emocional ainda me supriram com apoio financeiro durante o tempo de desenvolvimento deste trabalho. Agradeço a minha namorada Thaís por ter sempre me apoiado. Agradeço aos meus irmão Aldo, Henrique, Douglas e Helga por sempre terem ficado ao meu lado em tudo que faço.

Agradeço também aos meus amigos e colegas de laboratório Júlio, Jonathan, Thiago, Maycon, Pedro, Prazeres, Puiuna, Kimura, Rodolfo, Manoel, Rodrigo e Humberto tanto na ajuda técnica que foi imprescindível para o desenvolvimento do trabalho quanto nos momentos de lazer que passamos juntos durante esse período de desenvolvimento do mestrado.

O desenvolvimento deste trabalho não foi uma tarefa trivial e nem mesmo com méritos exclusivos do autor. Por isso agradeço pela contribuição para o desenvolvimento ao orientador Orientador Professor Doutor Marcos José Santana, que me ajudou sempre no que precisei, à Professora Doutora Regina Helena Carlucci Santana, que me ajudou com os experimentos, e ao aluno de doutorado Júlio Cezar Estrella, que me ajudou muito desde a elaboração do projeto até a escrita desta dissertação. Gostaria também de agradecer a todos os membros do grupo de Sistemas Distribuídos e Programação Concorrente que sempre deram ótimas sugestões para o desenvolvimento do trabalho.

Agradeço às agências de fomento CAPES e CNPq pelo financiamento deste projeto.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Símbolos

AXIS	Apache eXtensible Interaction System
ASF	Apache Software Foundation
BPM	Business Process Management
BPEL	Business Process Execution Language
BPML	Business Process Management Language
B2B	Business to Business
CGI	Common Gateway Interface
CPU	Central Processing Unit
DWSC-M	Dynamicy Web Services Composition Middleware
FTP	File Transfer Protocol
HP	Hewlett-Packard
HTTP	HyperText Transfer Protocol
IBM	International Business Machines
QoS	Quality of Service
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TI	Tecnologia da Informação
UDDI	Universal Description and Discovery Integration
URI	Uniform Resource Identifier
WSARCH	Web Services Architecture
WSML	Web Services Modeling Language
WSLA	Web Services Level Agreement
WSDL	Web service Descrition Language
WSOL	Web Services Offerings Language
XML	Extensible Markup Language
XPDL	XML Process Definition Language
YAWL	Yet Another Web Services Language

Resumo

Este trabalho de mestrado apresenta a modelagem, a prototipação e os resultados do desenvolvimento de um *middleware* para composição dinâmica de *Web Services* denominado DWSC-M (*Dynamic Web service Composition Middleware*). O objetivo principal desse *middleware* é permitir que serviços sejam compostos dinamicamente considerando aspectos de qualidade de serviço na escolha dos serviços que fazem parte de um fluxo de composição. Para complementar o funcionamento do *middleware* DWSC-M foram propostos dois algoritmos para seleção de *Web Services:* o primeiro utiliza seleção aleatória e o segundo utiliza distância Euclidiana para seleção de serviços e considera, para tal finalidade, os parâmetros de QoS enviados pela requisição do cliente do serviço.

Abstract

This work presents the modeling, the prototype and the results of the developing of a middleware for dynamic composition of *Web Services* named DWSC-M (Dynamic Web Service Composition Middleware). The main focus of this middleware is to allow services being dynamically composed, considering aspects of quality of service in the choice of services that are part of the composite flow. Complementing the operation of the DWSC-M middleware two algorithms were proposed for *Web Services* selection : the first one uses *Random* selection and the second one uses *Euclidean Distance* for the selection of services and considers for this purpose the QoS parameters sent by the client service request.

Capítulo

1

Introdução

As modificações que a Internet vem sofrendo nos últimos anos, têm propiciado muitos benefícios aos seus usuários em geral. Atualmente, a Web é amplamente utilizada para aplicações que envolvem comércio eletrônico, jogos *online*, aplicações multimídia, vídeo sob demanda, entre outras. Recentemente, com o surgimento dos *Web Services* tem sido possível que aplicações executando em diferentes plataformas e em diferentes linguagens de programação, possam se comunicar pela Web por meio de uma interface denominada WSDL (*Web Service Description Language*), toda escrita em XML (Extensible Markup Language), uma linguagem interoperável e independente de plataforma e sistema operacional (Bray et al., 2008). Nessa interface ficam descritas todas as informações necessárias para que as aplicações possam se comunicar por meio de troca de mensagens pelo protocolo SOAP (*Simple Object Access Protocol*) que também utiliza a linguagem XML para encapsular as mensagens a serem trocadas.

Com a crescente popularidade dos *Web Services*, um suporte geral à QoS (Quality of Service) (Erradi; Maheshwari, 2005) é uma questão pertinente a ser tratada visando a aumentar a satisfação dos usuários de *Web services*. Os ambientes atuais, que operam com *Web Services*, não oferecem suporte adequado à qualidade de serviço e apresentam limitações como falta de suporte a *Service Level Agreement* (SLA) e falta de garantias de confiabilidade dos provedores de serviços (Tian et al. 2003). Como os *Web Services* começam a ser aplicados em ambientes com aplicações de missão crítica e com B2B (*Business to Business*), é necessário melhorar a qualidade de serviço e que a entrega contínua de serviços torne-se uma questão crítica visando a alta disponibilidade e alta confiabilidade, apesar de possíveis falhas e indisponibilidade dos serviços participantes ou também da rede de comunicação (Erradi; Maheshwari, 2005). Nesse sentido, é importante caracterizar como vários *Web Services* interagem entre si, com o objetivo maior de prover melhor qualidade de serviços para os usuários e para as aplicações. Essa interação é denominada composição de serviços e tem

como finalidade permitir que serviços dependentes troquem informações para melhor provisão de QoS.

Segundo Menascé (2004), é importante considerar como os diferentes atributos de QoS devem ser combinados para realizar uma composição mais elaborada devido às suas características específicas. O objetivo central deste trabalho é, portanto, a aplicação de algoritmos para seleção de *Web Services* em composição de serviços incorporada em arquiteturas orientadas a serviço e, em especial, como forma de validação para a arquitetura WSARCH (Web Services Architecture), proposta por Estrella (2007).

O surgimento dos *Web Services* tem provocado mudanças de paradigma na integração de aplicações, onde serviços podem ser compartilhados para permitir a otimização de processos de negócios numa ampla estrutura de Tecnologia de Informação (TI). No estado atual de suporte, os *Web Services* são capazes de integrar aplicações executando em diferentes plataformas, habilitar informações de uma base de dados de aplicações para serem disponibilizadas para outras, além de permitir que aplicações internas se tornem disponíveis na rede mundial de computadores. *Web Services* constituem uma nova promessa para a integração de aplicações Web, sendo um tópico amplamente pesquisado pela comunidade acadêmica e por empresas como IBM, Hewlett-Packard – HP e Microsoft (Siblini; Mansour, 2005).

Os *Web Services* facilitam a interoperabilidade no nível de aplicação, oferecendo interfaces padrão para aplicações do mundo exterior. Por outro lado, processos de negócios são tipicamente formados de aplicações interagindo umas com as outras, relacionando empresas, fornecedores e clientes. Para permitir maior interação nos processos de negócios, é importante que as aplicações interajam entre si de forma flexível. Isso pode ser facilitado pela adoção de *Web Services* como padrão para aplicações que são parte de processos de negócios. O gerenciamento de processos de negócios também oferece a capacidade de incluir *Web Services* em uma arquitetura genérica orientada a serviço numa larga escala empresarial, utilizando, por exemplo, para esta finalidade, um processo de composição de serviços (Janssen; Cresswell, 2005).

Há diversos problemas que precisam ser solucionados para permitir uma melhor integração entre os *Web Services* que compõem um processo de negócios. Como se trata de uma solução que envolve muitos conceitos e aplicações de sistemas distribuídos, se faz necessária uma análise cuidadosa de problemas tais como disponibilidade, dependência e confiabilidade de *Web Services*, além da qualidade de serviço envolvida na comunicação das aplicações orientadas a serviços.

Nesse sentido, este projeto de mestrado caracteriza modelos e algoritmos de composição de serviços, levando em consideração parâmetros relativos à qualidade de serviço – QoS. Tais algoritmos foram testados em um *middleware* para composição especialmente construído para essa finalidade. O *middleware* é modular e será incorporado à arquitetura WSARCH, podendo também ser adaptado a qualquer arquitetura SOA.

A estrutura desta dissertação de mestrado envolve 8 capítulos, os quais são brevemente discutidos a seguir.

No capítulo 2 são apresentados conceitos detalhados sobre arquiteturas SOA e o que são os *Web Services*, bem como o principio de funcionamento dessa solução para sistemas distribuídos, que permite comunicação entre arquiteturas e linguagens diferentes por meio de padrões XML. No capítulo 3 são apresentados conceitos de composição de *Web Services,* o funcionamento, linguagens disponíveis para criação de serviços compostos e como qualidade de serviço deve ser tratada em composição de serviços. No capítulo 4 é apresentado de forma detalhada a arquitetura WSARCH, uma arquitetura para provisão de *Web Services* com qualidade de serviço. No capítulo 5 é discutido o *Dynamic Web Services Composition Middleware* (DWSC-M), um *middleware* proposto nesta dissertação, juntamente com o protótipo desenvolvido para tornar possível a composição de serviços com qualidade de serviço por meio de acesso dinâmico aos *Web Services*. No capítulo 6 são apresentados os algoritmos para seleção de *Web Services* para composição de serviços que foram propostos e implementados neste trabalho. No capítulo 7 são discutidos os principais resultados obtidos, o planejamento do experimento e influência de seus fatores no resultado e finalmente no capítulo 8 serão apresentados as conclusões, contribuições e trabalhos futuros.

Capítulo

2

SOA e Web Services

2.1 Considerações Iniciais

O desenvolvimento de software tem contemplado vários paradigmas, tendo destaque o da orientação a objeto. Várias linguagens de programação associadas a esse paradigma foram bem sucedidas, tais como C++, Java e Visual Basic. Porém, com o advento da Internet e o desenvolvimento de aplicações Web, as necessidades empresariais e as requisições de serviços trouxeram a demanda de uma nova arquitetura de software (Bih, 2006). Surge, então, nesse cenário o conceito de SOA (*Service-oriented Architecture*), como uma solução para esse problema, por meio da utilização dos *Web Services*.

Neste capítulo serão apresentados os conceitos, o modelo e a arquitetura de SOA e *Web Services*, tópicos de fundamental importância para o desenvolvimento deste trabalho de mestrado.

2.2 Definições de SOA

Segundo Bih (2006) SOA são serviços que podem se comunicar por meio de passagem de mensagens simples, coordenando alguma atividade que precise de meios de conexão de serviços entre si.

Para um bom entendimento de SOA é preciso ter um bom conhecimento do termo serviço. Serviço é uma função bem definida e independente do contexto ou estado de outros serviços. A arquitetura SOA básica consiste de um serviço consumidor e um provedor de serviços, onde um serviço consumidor faz uma requisição para um provedor de serviços, que responde com o resultado para o serviço consumidor; o provedor de serviços pode também ser um consumidor de serviços. As conexões de requisição e resposta são definidas de maneira que seja possível que ambos os serviços possam entendê-la.

2.2.1 Modelo SOA

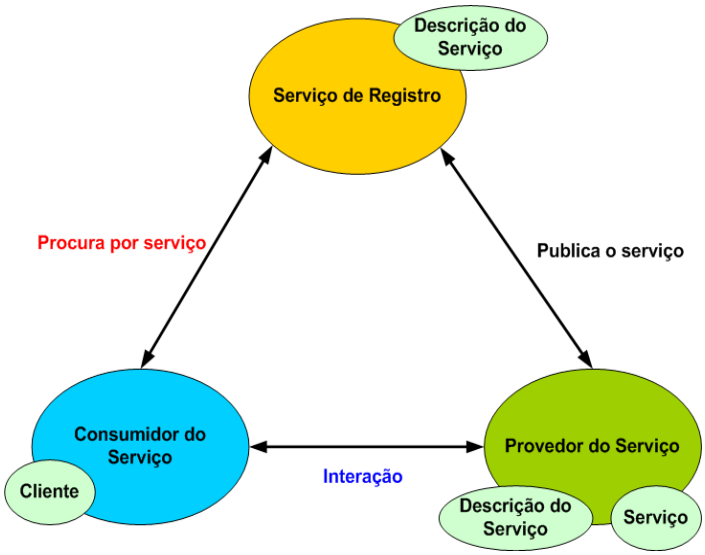

Figura 1: Arquitetura Orientada a Serviço (BIH, 2006)

O provedor de serviços fornece as interfaces de serviços para um software independente, o qual gerencia um conjunto específico de tarefas. O provedor de serviços pode representar os serviços de uma entidade de negócios ou pode representar simplesmente uma interface de serviço reutilizável de um subsistema.

O serviço consumidor é um nó na rede que descobre e invoca outros serviços para obter uma solução por meio de chamadas à operações dos serviços. Em alguns casos, o provedor pode estar no mesmo local que um consumidor de serviços e ser acessado por meio de uma rede local, ou em outros casos ele pode estar muito distante e ser acessado pela Internet. A natureza conceitual de SOA deixa a rede, os protocolos de transporte e os detalhes de segurança, para implementações especificas de SOA.

O serviço de registro atua como um repositório ou “páginas amarelas”, uma analogia às páginas amarelas de listas telefônicas, onde os provedores de serviços publicam seus serviços.

Essas três entidades de SOA interagem entre si por meio de três operações básicas: publicar, buscar e utilizar. O provedor de serviços publica serviços no serviço de registro. O consumidor de serviços busca o serviço que deseja, utilizando o serviço de registro, e então faz a utilização desse serviço.

2.3 *Web Services*

A W3C define um *Web Service* como uma aplicação identificada por uma URI (*Uniform Resource Identifier*), cujas interfaces e ligações são definidas, descritas e descobertas utilizando-se uma linguagem padrão, XML (FERRIS; FARREL, 2003).

Web Services são uma implementação de SOA (ERRADI; MAHESHWARI, 2005). Pelas definições apresentadas, pode-se observar que diferentes explicações sobre *Web Services* estão disponíveis. Uma delas menciona que as interações entre *Web Services* ocorrem tipicamente com chamadas SOAP (*Simple Object Access Protocol*), um protocolo de comunicação baseado em XML para a interação de aplicações (PAPAZOGLOU; GEORGAKOPOULOS, 2003). SOAP é apresentado como um arcabouço para uma nova geração de aplicações de computação distribuída, independente de plataforma e de linguagens de programação.

A principal função desse protocolo é encapsular as chamadas a métodos remotamente distribuídos, que serão transportados, por sua vez, por algum protocolo, tal como o HTTP (*Hyper Text Transport Protocol*), utilizado para comunicação entre um *browser* e um servidor Web (COMER, 2000). Além disso, as descrições de interfaces dos *Web Services* são expressas usando a linguagem denominada WDSL (THOMAS; THOMAS; GHINEA, 2003). Na literatura sobre Web Services é apresentado também um protocolo para serviços de diretório, que contém as descrições dos *Web Services*, denominado UDDI (*Universal Description Discovery and Integration*). Esse protocolo funciona como um registro de serviços sendo um importante componente da arquitetura orientada a serviços (FARKAS; CHARAF, 2003).

Uma arquitetura orientada a serviço é uma maneira lógica de construção de um sistema de software para prover serviços, ou para aplicações de usuários finais, ou para outros serviços distribuídos em uma rede, por meio de interfaces públicas disponíveis (nesse contexto destaca-se o WSDL) (PAPAZOGLOU, 2003). O UDDI habilita, ainda, os clientes de *Web Services* a localizarem serviços ou descobrirem seus detalhes, permitindo que registros operacionais sejam mantidos para diferentes propósitos em diferentes contextos (LEAVITT, 2004).

2.3.1 Modelo de *Web Services*

A arquitetura de *Web Services* é baseada nas interações entre três entidades que são o provedor de serviços, o registro de serviços e o consumidor de serviços. As interações

envolvem a publicação, a pesquisa e a ligação.

O provedor de serviços define uma descrição de serviços para o *Web Service* e o publica nos registros de pesquisa ou a disponibiliza diretamente para serviços consumidores. Os serviços consumidores utilizam operações de pesquisa para localizar a descrição do serviço localmente ou nos registros de pesquisa e usa a descrição do serviço para conectar-se ao provedor de serviço e invocar ou interagir com a implementação do *Web Service*. O provedor de serviços é uma construção lógica uma vez que um serviço pode ter características de provedor de serviço e de consumidor de serviço (KREGER, 2001).

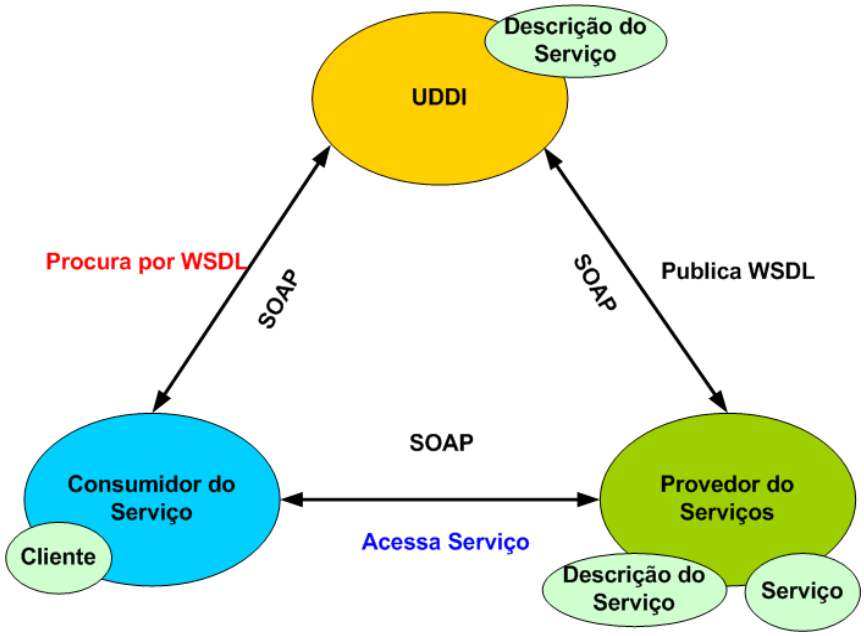

Figura 2: Modelo de Web Services (KREGER, 2001)

A figura 2 ilustra o modelo de *Web Services* composto de suas entidades e suas operações. É possível observar nessa figura que os *Web Services* são uma implementação de uma arquitetura orientada a serviço onde as entidades e operações exercem as funções discutidas na seção 2.2.1 deste capítulo.

2.3.2 Ciclo de vida de *Web Services*

O desenvolvimento do ciclo de vida dos *Web Services* inclui o requisito do projeto, publicação e tempo de execução para cada uma de suas entidades. Cada entidade possui requisitos específicos para cada elemento do desenvolvimento de seu ciclo de vida.

Segundo (KREGER, 2001) o desenvolvimento do ciclo de vida tem várias fases, as quais podem ser:

- **Construção:** Essa fase inclui o desenvolvimento e testes de um *Web Service*,

definição da descrição da interface do serviço e a definição da descrição da implementação do serviço. Os *Web Services* podem surgir a partir de novas criações, transformações de aplicações existentes e composição de *Web Services* existentes.

- **Publicação:** Fase em que ocorre a publicação da interface do serviço e definições de implementação para o consumidor ou registro de serviços e também a publicação da parte funcional do *Web service*, geralmente em um provedor de serviços.
- **Execução:** Essa é a fase onde o *Web service* fica disponível para ser acessado. Neste estágio o *Web service* já está publicado, operando e acessível pela rede por meio de um provedor de serviços. Desta forma o consumidor de serviço pode finalmente encontrá-lo e utilizá-lo.
- **Gerenciamento:** Essa fase de gerenciamento cobre o gerenciamento e a administração do *Web Service.* É importante considerar requisitos tais como segurança, disponibilidade, desempenho, qualidade do serviço e processos de negócio.

2.3.3 Arquitetura de *Web Services*

Para permitir que as três operações (publicar, pesquisar e conectar) sejam realizadas de maneira interoperável, é preciso destacar a pilha conceitual dos *Web Services* que contempla a estrutura de *Web Services,* compreendendo padrões em cada nível. A figura 3 mostra de maneira conceitual uma pilha de *Web Services*.

A base da pilha conceitual de *Web Services* (KREGER, 2001) é a rede. Os *Web Services* devem ser acessíveis pela rede para que possam ser acessados pelo consumidor de serviço. Os *Web Services* que são publicamente disponíveis na Internet, geralmente utilizam protocolos de rede bem conhecidos. O HTTP é o protocolo padrão disponível na Web para *Web Services,* apesar de ser possível utilizar outros protocolos como SMTP e FTP.

A camada de mensagens baseadas em XML representa o uso de XML como base para o protocolo de mensagem. SOAP é um protocolo baseado em XML escolhido por diversas razões:

- É o mecanismo padrão para ser o envelope para mensagens e chamadas de procedimentos utilizando XML.
- Pela simplicidade, pois é constituído basicamente de um HTTP POST com um envelope XML como carga.
- As mensagens SOAP suportam operações de publicação, pesquisa e conexão na arquitetura Web.

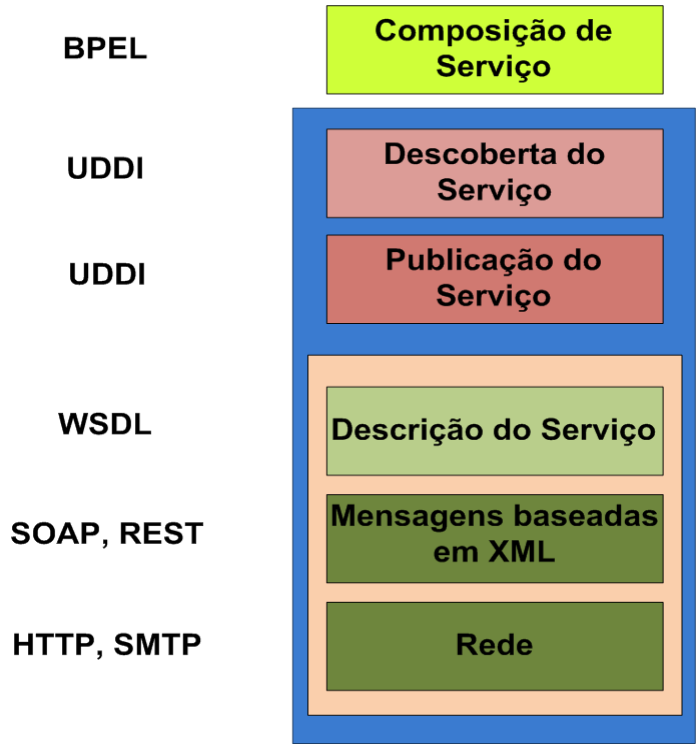

Figura 3: Pilha da estrutura de Web Services (KREGER, 2001)

A camada de descrição de serviços apresenta a WSDL como padrão, necessária para dar suporte à interoperabilidade em *Web Services*. Uma WSDL define a interface e os mecanismos de interação de serviços. A descrição é ainda necessária para especificar o contexto de negócio, QoS e relações de serviço com serviço. Documentos WSDL podem ser complementados por outros documentos de descrição de serviços para descrever os aspectos dos *Web Services*.

Uma vez que os *Web Services* são definidos para serem acessíveis pela rede via SOAP e representado por um serviço de descrição, as três primeiras camadas da pilha são necessárias para fornecer suporte ao uso de qualquer *Web Services*. A pilha mais simples deve consistir de HTTP para a camada de rede, o protocolo SOAP para a camada de mensagens baseadas em XML e WSDL para a camada de descrição.

A camada de publicação de serviços inclui a produção de descrição do serviço e sua publicação. A descrição de serviços pode ser gerada manualmente ou montada a partir de definições de interfaces de serviços existentes. Desenvolvedores podem desenvolver manualmente toda a descrição do serviço, incluindo a entrada ao UDDI. Há ferramentas como o Axis2 e jUDDI client, apresentadas no capítulo 4 para gerar as partes de WSDL e potencialmente parte da entrada UDDI a partir de meta dados de modelos de programação e publicação dos *Web Services.*

A publicação da descrição dos serviços pode ser feita utilizando uma variedade de mecanismos como, por exemplo, o jUDDI client. Esses mecanismos oferecem diferentes funcionalidades de acordo com grau de dinamicidade que a aplicação que utiliza o serviço exige. A descrição dos serviços pode ser publicada em múltiplos serviços de registro utilizando vários mecanismos diferentes. O caso mais simples é a publicação direta para o consumidor de serviço, no qual essa pode ser feita por e-mail, FTP ou até mesmo pela distribuição via CD-ROM. A publicação direta pode ocorrer após as duas partes da negociação entrarem em acordo sobre os termos de *e-business* via Web, ou após o consumidor do serviço pagar por ele. Nesse caso o consumidor do serviço poderá manter uma cópia local da descrição do serviço.

A camada de descoberta de serviços da pilha inclui a aquisição da descrição do serviço e o consumo das descrições. A aquisição pode utilizar uma variedade de mecanismos. Assim como na publicação de descrições de *Web Services* a aquisição de descrições de *Web Services* irá variar dependendo de como a descrição do serviço está publicada e quão dinâmica a aplicação deve ser. Consumidores de serviços podem encontrar *Web Services* em duas fases diferentes de um ciclo de vida da aplicação, que são o projeto e o tempo de execução. No projeto, consumidores de serviço pesquisam por descrições de *Web Services* e pelo tipo de interface que eles suportam. Em tempo de execução os serviços consumidores procuram por *Web Services* baseando-se em como eles se comunicam ou utilizando aspectos de QoS, por exemplo.

Na abordagem de publicação direta, o consumidor de serviços armazena a descrição dos serviços em tempo de projeto, para então utilizá-la em tempo de execução. A descrição do serviço pode ser estaticamente representada na lógica do programa, armazenada em um arquivo ou em um simples repositório local de descrição de serviço.

Os consumidores de serviços podem conseguir uma descrição de serviço no tempo de projeto por meio de um repositório de descrição de serviço como o UDDI. Os vários tipos de UDDI apresentam implicações no número de *Web Services* prontos para serem escolhidos. Dentre essas implicações estão a política de escolha, ou o número de prévias a serem feitas pelo consumidor antes de invocar o serviço. Aplicações UDDI de nível mais profissional exigem que prévias não sejam feitas para estabelecer confiança com o serviço. A seleção de serviço deve ser baseada no suporte da conexão, histórico de desempenho, QoS, proximidade ou balanceamento de carga.

Após um serviço ser localizado, o consumidor precisa processar a WSDL correspondente para poder, então, gerar requisições SOAP para o acesso ao serviço.

A camada de fluxo de serviços é onde os processos de negócios são definidos. Essa parte será abordada mais detalhadamente no capítulo 3.

2.4 Qualidade de Serviço em *Web Services*

Qualidade de Serviços é referenciada como um conjunto de propriedades não funcionais de *Web Services*, tais como desempenho, confiabilidade, disponibilidade e segurança. Com a difusão dos *Web Services*, medidas de qualidade de serviço são utilizadas como uma aferição para diferenciar serviços (Kalepu et al. 2003). No entanto, garantir QoS em aplicações Web não é uma tarefa trivial devido às características imprevisíveis da Web (Farkas e Charaf, 2003).

Web Servcies com suporte à QoS geralmente devem ser associados a um acordo de níveis de serviços SLA, no qual é acordado formalmente níveis de serviços entre provedores e clientes. Esse acordo geralmente envolve um terceiro componente responsável pela garantia de contrato (Sun et al., 2006). SLAs em Web services têm sido desenvolvidos para facilitar relacionamentos complexos entre provedores e clientes (Kalepu et al., 2003).

A linguagem de definição de interface WSDL não oferece suporte a especificação de parâmetros de QoS, mas existem outras linguagens que fazem essa especificação e oferecem suporte à SLA como WSML (*Web Service Modeling Language*), WSLA (*Web Service Level Agreement*) e WSOL (*Web Service Offer Language*) que oferecem diferentes classes de serviço pré-definidos para distinguir os diferentes tipos de usuários. (Zhou e Niemela, 2006)

A seguir serão apresentados alguns parâmetros de qualidade de serviço de acordo com Kalepu (2003):

- **Disponibilidade:** Disponibilidade é um aspecto de qualidade de serviço no qual um *Web Service* está presente ou pronto para uso imediato, representado como uma porcentagem de tempo disponível de um serviço em um período de observação e está relacionado com sua confiabilidade. Para serviços freqüentemente acessados um pequeno valor de período de observação fornece uma aproximação mais precisa para sua disponibilidade.
- **Confiabilidade:** É a probabilidade que um serviço irá responder no período de tempo esperado pelo consumidor e probabilidade do serviço executar sua funcionalidade de maneira correta. Este atributo caracteriza a confiança que um consumidor de serviços

pode assumir no provedor de serviços.

- **Custo:** O Valor monetário cobrado pelo provedor de serviços pelo acesso feito ao serviço.
- ***Throughput*:** Representa o número de requisições que são finalizadas por determinado período de tempo. O aumento do *throughput* está diretamente relacionado ao tempo de resposta do sistema, pois quanto mais clientes ele conseguir atender em menos tempo, menor será a sobrecarga para receber novas requisições.
- **Tempo de resposta:** É o tempo que uma requisição demora a partir do momento que ela é iniciada para ser atendida até o momento em que o requisitante recebe a resposta dessa requisição. Este tempo inclui toda a sobrecarga e possíveis atrasos que ocorram na rede.
- **Latência:** Tempo gasto entre a chegada da requisição e o envio da resposta.
- **Desempenho:** É a medida feita utilizando métricas como Tempo de Respota e *Throughput.* O desempenho é considerado bom quando o *throughput* é alto e o tempo de resposta é baixo.
- **Segurança:** A habilidade para garantir o não-repúdio das mensagens. Este atributo é importante para conseguir confiabilidade dos provedores de serviços.
- **Regulatoriedade:** Aspecto de qualidade de serviço que mostra se um *Web Service* está em conformidade com as leis, regras, padronizações e acordos de níveis de serviços estabelecidos.
- **Robustez/Flexibilidade:** A habilidade de um serviço em contornar entradas inválidas, incompletas e conflitantes e gerar o resultado correto.
- **Precisão:** Avalia a probabilidade do serviço gerar erro.
- **Reputação:** É medido pela avaliação que os clientes do serviço atribuem a ele. No final de cada acesso isso pode ser coletado por meio de uma enquete sobre satisfação do usuário.

2.5 Considerações Finais

Este capítulo apresentou uma visão geral do que é SOA e uma abordagem mais detalhada do conteúdo de *Web Services*. Foi apresentada a pilha da estrutura dos *Web Services* e descritas todas as suas camadas para a permitir a construção de aplicações distribuídas com *Web Services*.

No próximo capítulo serão abordados fundamentos sobre composição de serviços, complementando o assunto apresentado no presente capítulo com aspectos de QoS relacionados à composição de serviços.

Capítulo

3

Composição de *Web Services* com Qualidade de Serviço

3.1 Considerações Iniciais

A composição de serviços é um tópico importante na literatura de *Web Services*, visto que um único *Web Service* tem suas funcionalidades limitadas em relação a um processo de negócio complexo.

A arquitetura de *Web Services* mostrada no capitulo 2 é satisfatória para interações cliente servidor, mas se for necessário combinar aspectos de vários serviços para gerar um novo serviço complexo, o ideal seria a utilização de composição de serviços para definir a execução dessas interações.

Um processo de negócio geralmente envolve mais de uma organização que possui serviços independentes e que precisam ser integrados para então fornecer ao cliente um resultado desejado. Nesse sentido percebe-se a importância de pesquisar composição de *Web service* com QoS, pois assim é possível fornecer aos clientes um resultado otimizado para suas necessidades e condições de aquisição de serviço.

Neste capítulo serão abordados tópicos importantes sobre composição de serviços com QoS. Dessa forma, será apresentada uma introdução ao assunto, seguida de uma seção sobre QoS aplicada à composição de serviços. A linguagem que se pretende utilizar para o desenvolvimento deste trabalho será objeto de discussão e, por fim, serão apresentadas ferramentas de implementação que serão utilizadas para auxiliar no desenvolvimento do trabalho.

3.2 Motivação para composição de serviços

Um dos principais conceitos utilizados para a integração de processos dentro de BPM (*Business Process Management*) é a composição de serviços, na qual as linguagens BPEL

(*Business Process Execution Language*) e YAWL (*Yet Another Workflow Language*) (ADAMS et al., 2008) se baseiam. Um aspecto valioso do modelo de serviços é que novos serviços podem ser criados a partir da existência de outros, sem desconsiderar o paradigma da arquitetura orientada a serviços (WEERAWARANA et al., 2005).

Um exemplo clássico na literatura sobre o assunto é o sistema para reservas de viagens. Nesse sistema é possível, a partir de *Web Services* para reserva de hotel, reserva de passagens e aluguel de veículos, fazer a combinação desses serviços gerando um serviço composto mais complexo onde o usuário, por meio da chamada de um único serviço, poderá conseguir um pacote de viagem.

Como o processo de composição de serviços não é um tópico trivial na literatura da área, alguns critérios devem ser considerados para que *Web Services* sejam compostos de forma adequada (CHANDRASEKARAN, 2002).

3.2.1 Tipos de composição

Na composição de serviços existem dois grandes domínios: a orquestração e coreografia de serviços. A orquestração descreve o fluxo de execução que deve ser coordenado por um *engine,* que pode ser centralizado ou distribuído para fazer a integração dos serviços. A principal linguagem utilizada para se fazer orquestração é denominada BPEL. Um aspecto valioso do modelo de serviços é que podem ser criados novos serviços a partir da existência de outros, sem abandonar o paradigma de computação orientada a serviços.

A coreografia descreve a colaboração que cada participante irá obter, definindo de um ponto de vista global, seu comportamento por meio de troca de informações para que entrem em consenso sobre as regras para a integração entre entidades (W3C 2004). Para a colaboração funcionar com sucesso, as regras para interação entre todos os participantes devem ser definidas. Atualmente essas regras são descritas de forma discursiva. Por isso surgiu a necessidade de um padrão que descreva essas interações para que não houvesse mais ambiguidade na documentação e responsabilidade dos participantes. A especificação WS-CDL (Web Services Choreography Description Language) tem como objetivo principal descrever a colaboração entre qualquer tipo de participante de acordo com a plataforma suportada ou modelo de programação para implementação do seu ambiente. A coreografia facilita a orquestração da execução de serviços existentes [Weerawarana et al, 2005].

Na literatura, é comum confundir sobre os tipos de composição de *Web Services* (manual ou automática; estática ou dinâmica). Os trabalhos de Rao and Su (2005), Dustdar

and Schreiner (2005) e Alamri, Eid and Saddik (2006) comentam sobre esse tipo de classificação e a partir dessas classificações e experiências com implementação de composição de serviço foi possível chegar a conclusão de forma sucinta que composição manual e automática são duas grandes classificações onde na composição manual um grafo contendo a descrição de execução de fluxo e a referência lógica a *Web Services* é planejado por um desenvolvedor. Na composição automática o fluxo é definido automaticamente, por meio de técnicas como inteligência artificial e semântica, usando, como referência para definir o fluxo, apenas as entradas e saídas fornecidas ao processo pelo usuário. A composição dinâmica e estática são classificações que podem ser aplicadas dentro de composição automática e manual. Composição estática se refere à associação de cada implementação de serviço em tempo de projeto. Na composição dinâmica os serviços são selecionados para serem executados em tempo de execução. O desenvolvimento deste projeto de mestrado utiliza o conceito de composição de serviços manual dinâmica.

3.2.2 Fases da composição

As fases da composição de um *Web Service* são: planejamento, definição, análise da execução, análise da eficiência e componentes do tempo de execução. O planejamento é a fase em que se discute qual tipo de composição e quais técnicas de busca pelos serviços são as mais apropriadas. Na fase de definição é feita uma representação da estrutura do processo de composição, em uma linguagem adequada para a resolução do problema. Para isso, é necessária a análise da execução que é responsável pela interpretação da especificação da composição, pela avaliação de desempenho e pelas técnicas de execução. Entre as técnicas de execução duas são destacadas como as mais importantes: a execução distribuída, em que a responsabilidade de coordenar um processo de composição é distribuída entre os diversos provedores de serviços; e a execução centralizada, em que a composição tem um único responsável por gerenciá-la (CHANDRASEKARAN, 2002).

A fase de análise da eficiência é feita por meio de parâmetros de QoS, tais como o tempo de serviço. Também é analisada a habilidade para lidar com sobrecarga, os métodos matemáticos para avaliar a QoS de um *Web Service*, e uma simulação da composição para análise da eficiência. Na fase de componentes do tempo de execução são analisados: o tempo que um serviço gasta para realizar somente a execução da tarefa: o tempo de transmissão das mensagens SOAP: e o tempo de espera gerado em razão da sobrecarga no sistema.

3.2.3 Técnicas

Avaliar os serviços envolvidos em um processo de composição deve contemplar: a descoberta do *Web service* envolvido no processo de composição que causa sobrecarga no sistema: o *Web service* que tem alta sobrecarga de comunicação gerada pelo processamento das mensagens SOAP: e a capacidade de carga individual dos serviços envolvidos no processo, ou seja, a carga que cada serviço pode suportar efetivamente (CHANDRASEKARAN, 2002).

3.3 QoS em composição de serviços

Outros trabalhos também discutem a necessidade de caracterizar parâmetros de QoS para o processo de composição (GOUSCOS; KALIKAKIS; GEORGIADIS, 2004), (TOSIC et al., 2002). Para os autores é preciso relacionar quais são os parâmetros de QoS que permitem a escolha do melhor serviço em determinada ocasião (quais métricas, quais técnicas). Encontrar um *Web Service* adequado (em um grande conjunto que oferece funcionalidades semelhantes) pode ser uma tarefa vagarosa e o sucesso da escolha nem sempre estará garantido. Determinar as estimativas para as propriedades de QoS de um *Web Service* é uma tarefa desafiadora e envolve (SHETH et al., 2002):

- Uma combinação de estimativas de projetistas da aplicação.
- Estimativas de QoS são paramétricas:
  - Exemplo: O tempo de resposta de um serviço que obtém um documento XML como entrada, dependerá do tamanho do documento.
  - Estimativas para *Web Services* compostos podem ser desenvolvidas de dois modos:
    - Atomicamente (para cada serviço).
    - Conjuntamente.

Ainda com relação à caracterização de QoS é interessante a especificação do comportamento de execução de um *Web Service*, que pode ser feita utilizando-se de duas classes:

- **Básica:** Associa a cada dimensão de QoS do *Web Service* um valor mínimo, médio e máximo.
  - Exemplo: A dimensão do custo corresponde a um valor mínimo, médio e máximo associada à a execução de uma tarefa. Os valores especificados nessa classe são

empregados por métodos matemáticos para computar métricas de QoS.

- **Distribucional**: Corresponde à especificação de uma constante ou de uma função de distribuição (exponencial, lognormal, normal, etc) que estatisticamente descreve o comportamento da tarefa em tempo de execução. Os valores são apresentados por meio de simulação para computar o fluxo de trabalho de QoS.

Há diferentes categorias de QoS que definem requisitos para a seleção entre os serviços disponíveis para composição destacando-se (Jaeger; Lader, 2005):

- **Tempo de Execução**: o tempo de execução é o tempo necessário para que o serviço seja executado mais a comunicação.
- **Custo**: o custo representa o montante de recursos necessários para o uso de um serviço, para o cálculo do custo na composição de serviços é necessário considerar que recursos são gastos a cada vez que um serviço é utilizado.
- **Reputação**: o conceito de reputação representa a qualificação do serviço fornecida pelos usuários aos serviços por eles utilizados, como por exemplo, o *skype* (programa para ligação por voz sobre IP) pede aos usuários notas sobre a qualidade da ligação. Assim como a reputação é a média dos *rankings* dos usuários, na composição de serviços é utilizada a reputação de cada serviço individualmente.
- **Disponibilidade**: a disponibilidade de um serviço implica na probabilidade de que a invocação do serviço seja feita com sucesso e que o resultado seja entregue com as demais categorias de QoS atendidas.

Avaliar cada categoria de QoS de algum serviço é importante, mas é preciso um método para calcular a QoS de uma composição como um todo. Para isso é necessário um modelo para a composição de serviços. Jaeger e Ladner (2005) explicam um modelo que facilita essa tarefa. Nesse modelo são apresentadas sete estruturas básicas:

- **Seqüência de execução de serviço**: onde a seqüência que os serviços serão executados é pré-definida.
- **Laço**: a execução de um serviço será repetida algumas vezes.
- **Separação AND seguida por uma junção AND**: ‘n’ serviços são executados paralelamente e todos os serviços precisam terminar a execução com sucesso para que haja sincronização.
- **Separação AND seguida por uma junção *m-out-of-n***: ‘n’ serviços são executados em paralelo, mas é preciso que $m \leq n$ tarefas terminem para que haja sincronização.
- **Separação XOR seguida por uma junção XOR**: para ‘n’ serviços prontos serem executados em paralelo apenas um é iniciado. A operação de sincronização considera

apenas as tarefas que começaram a ser executadas.

- **Separação OR seguida por uma junção OR**: para ‘n’ serviços prontos serem executados em paralelo mais que um, mas não todos os serviços são executados juntos. A sincronização precisa que todas as tarefas iniciadas juntas sejam executadas com sucesso.
- **Separação OR seguida por uma junção *m-out-of-n***: para ‘n’ serviços prontos serem executados em paralelo, mais de um serviço são iniciados juntos, mas não todos, e ‘m’ ≤ ‘n’ precisam ser finalizados com sucesso para que a sincronização seja feita.

Utilizando essas estruturas é possível agregar valores de QoS para cada um dos padrões descritos e, então, calcular uma QoS total para uma composição completa. Na figura 4 é possível ver um exemplo simples deste caso, onde as tarefas 3 e 4 são agregadas e então recebem um valor de QoS e se tornam semelhantes às outras tarefas, que também são agregadas formando uma única tarefa, recebendo assim, um valor final de QoS.

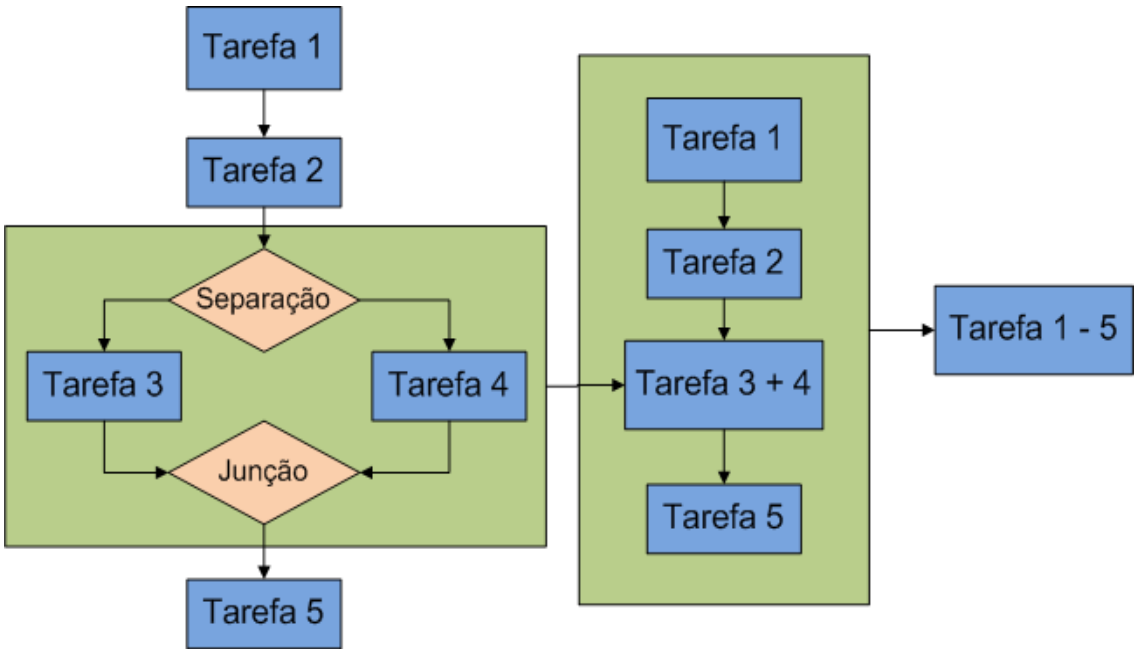

Figura 4: Grafo de Agregação (JAEGER; LADNER, 2005)

É importante destacar que analisar o desempenho de uma composição de *Web Services* na forma de execução não é uma técnica adequada, uma vez que tal processo requer que haja o controle sobre os *Web Services* e os *hosts* envolvidos, e isso não pode ser alcançado em um ambiente como a Web, dada sua natureza dinâmica. Uma falha em um serviço, por exemplo, pode afetar todo o ambiente de avaliação, a menos que tal falha seja caracterizada no processo de composição e que seu comportamento também seja avaliado. Testar e simular *Web*

Services para avaliar o desempenho são duas novas áreas com poucos trabalhos abordados atualmente. Vale destacar que a qualidade de serviço resulta da interação entre a demanda do usuário, o comportamento do sistema sobre essa demanda e os recursos que esse comportamento requer (WOODSIDE; MENASCÉ, 2004). Caso isso não seja considerado (tais fatores), a tendência é o mal uso de recursos e uma QoS não controlada. Com um grande número de serviços, consumidores tendem a querer distinguir entre os bons e os maus provedores e as técnicas de QoS são uma maneira de se fazer essa distinção. Para atingir tal finalidade faz-se necessário mapear o que o usuário deseja, em relação ao que o provedor de serviços tem para oferecer (THAER et al., 2005).

O ponto principal deve ser o conhecimento do comportamento das aplicações e suas demandas por recursos. Outro problema é a dificuldade de descobrir esse comportamento. Apesar das dificuldades envolvidas, um estudo sobre composição de *Web Services* é importante no cenário atual da Web, onde empresas, instituições financeiras e universidades trocam informações e necessitam a cada dia de algumas garantias que permitam às suas aplicações a correta realização de suas tarefas. Em particular, o desenvolvimento deste projeto de mestrado deve ajudar os desenvolvedores a entenderem o comportamento das atividades de um processo de composição de aplicações distribuídas, além de ajudar na avaliação de desempenho da arquitetura WSARCH, discutida no quarto capítulo.

3.4 BPEL

BPEL (Business Process Execution Language) é uma linguagem para descrever o comportamento de processos de negócios baseado em *Web Services*. A notação BPEL inclui o fluxo de controle, as variáveis, a execução concorrente, entradas e saídas, a transação escopo/compensação e o manipulador de erro (ACTIVE ENDPOINTS, 2009).

Processos freqüentemente invocam *Web Services* para executar tarefas funcionais, podendo um processo ser tanto abstrato quanto executável. Processos abstratos são parecidos com bibliotecas: eles descrevem o que o processo pode fazer e suas entradas e saídas, mas não descrevem como isso é feito. Processos abstratos são úteis para descrever processos de negócio para outros que desejem utilizar o processo. Processos executáveis fazem o trabalho "pesado" e contém todos os passos da execução que representa a unidade coesiva do trabalho.

Um processo de negócio de *Web Services* consiste de atividades conectadas. Um processo algumas vezes contém apenas uma atividade, mas, geralmente um processo

apresenta um conjunto de atividades. O caminho feito por meio das atividades é determinado por muitos parâmetros, incluindo os valores de variáveis e a avaliação de expressões.

Há ainda outras linguagens para definição de fluxo de trabalho, tais como XPDL (XML Process Definition Language), BPML (Business Process Modeling Language), YAWL,etc., mas, apenas BPEL está sendo citada neste trabalho, pois, todas as ferramentas escolhidas para servirem como apoio a este projeto de mestrado trabalham com a linguagem BPEL.

3.5 Considerações Finais

Para a escrita deste capítulo foi feito um levantamento bibliográfico sobre composição de W*eb Services*, assunto central deste projeto de pesquisa. Os tipos, fases e técnicas de composição foram investigados para conseguir um entendimento claro de como é o processo para montar um fluxo de serviços.

Foram apresentados trabalhos relacionados que investigam QoS em composição de serviços e algumas técnicas para conseguir estipular QoS para um fluxo de serviço como um todo a partir de informações de QoS de serviços de forma individual. Este é o ponto crucial deste projeto, que pretende conseguir composição de serviços utilizando QoS.

Foi apresentada, também, uma breve introdução à uma linguagem para definição de fluxos de serviços e ferramentas de apoio para desenvolvimento, publicação e execução de serviços compostos. E por fim um exemplo inicial para um primeiro contato com composição de serviços.

No próximo capítulo será apresentada a arquitetura WSARCH, uma arquitetura criada para provisão de *Web Services* com qualidade de serviço. Como este trabalho de mestrado é parte integrante desta arquitetura, será mostrado como deverá ser feita a integração do presente trabalho de mestrado na arquitetura WSARCH.

Capítulo

4

WSARCH – Web Services Architecture

4.1 Considerações Iniciais

Neste capítulo será apresentada uma arquitetura para provisão de *Web Services* (ESTRELLA, 2007), na qual se pretende introduzir os algoritmos para composição de serviços desenvolvidos nesta dissertação de mestrado.

Será apresentada uma visão geral da arquitetura WSARCH, seus componentes principais e a contextualização de composição de serviços aplicada à arquitetura WSARCH, no qual será abordado em que parte da arquitetura será introduzida a composição de serviços.

Uma vez que a WSARCH é uma arquitetura que fará provimento de W*eb Services* com QoS, é importante essa integração, pois assim ela também contribuirá com a composição fornecendo informações de QoS de cada serviço de forma individual para que então seja possível obter QoS para a composição de serviços como um todo.

4.2 A Arquitetura WSARCH

A arquitetura WSARCH é uma proposta de arquitetura para a provisão de W*eb Services* com qualidade de serviço. A arquitetura tem como objetivo melhorar a provisão dos W*eb Services* utilizando-se de critérios de QoS, alguns já definidos e avaliados. A figura 5, mostra de forma geral, como é o funcionamento da arquitetura WSARCH:

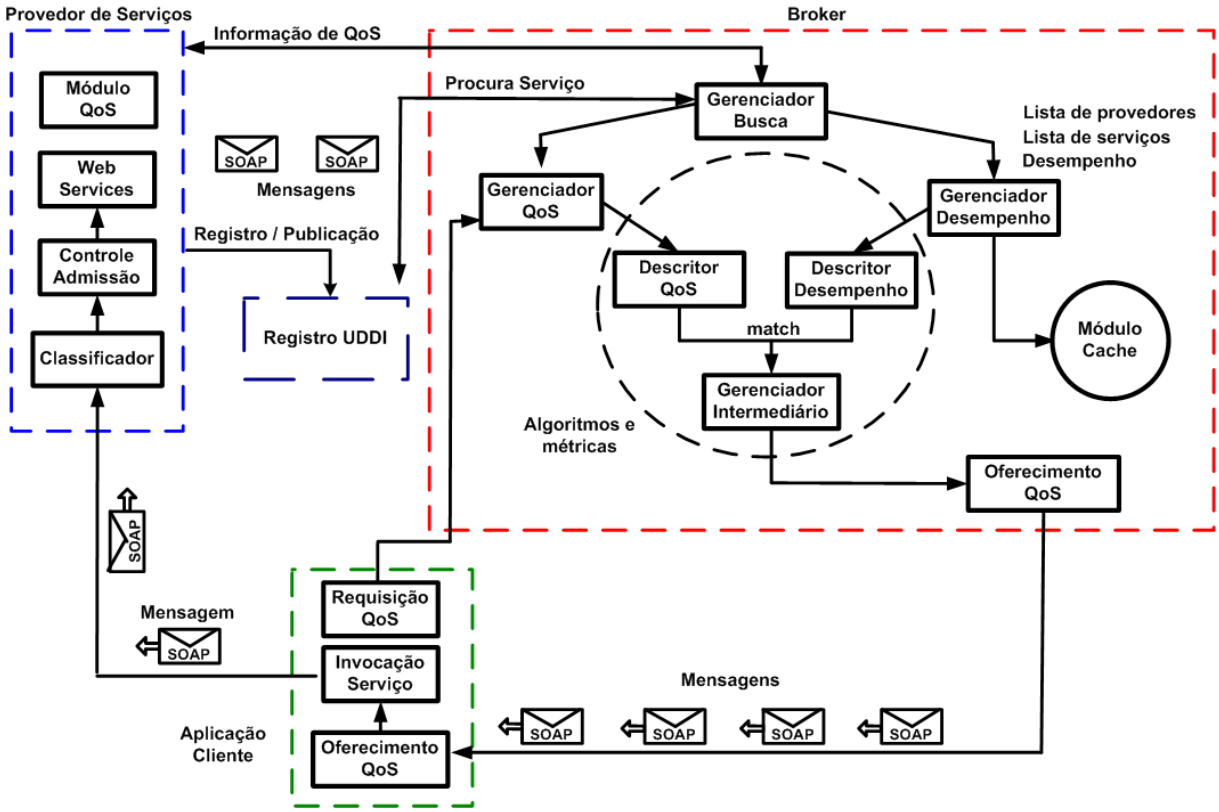

Figura 5: A arquitetura WSARCH e seus componentes (ESTRELLA, 2007)

Dentre os componentes dessa arquitetura se destacam:

- **Provedor de serviços**: Disponibiliza um serviço para um cliente W*eb Service*. A localização desse serviço deve ser registrada num repositório denominado UDDI. Além disso, o provedor de serviços também disponibiliza ao *broker* informações sobre a qualidade do serviço oferecido aos clientes W*eb Services*, tais como uso de CPU, quantidade de clientes atendidos por unidade de tempo no sistema, etc.

- ***Broker***: Responsável por realizar a busca de um serviço de acordo com as necessidades do W*eb Service* cliente. Deve também gerenciar informações de QoS vindas do provedor de serviços. Outro sub-componente importante é a negociação. O *broker* tentará obter o melhor serviço de acordo com as informações disponíveis sobre o provedor. Caso contrário, uma decisão deve ser tomada para que ou o serviço seja atendido com critérios de QoS relaxados ou não seja mais atendido. Destaca-se também uma base de dados para o armazenamento de mensagens sobre a qualidade de serviço.

- **Cliente**: Este faz o papel de solicitante de um determinado serviço.

Primeiramente o cliente deve solicitar ao *broker* que deseja acessar um serviço e para isso troca informações de QoS com ele. Após encontrar o serviço desejado, o *broker* repassa informações para o cliente (tais como: localização do W*eb Service*, qual sua descrição e como invocá-lo). Desse ponto em diante, o cliente invoca o serviço diretamente do provedor de serviços.

4.3 Composição de Serviços aplicada à arquitetura WSARCH

Para que o desenvolvimento deste projeto seja adequado às propostas da arquitetura descrita anteriormente, primeiramente é apresentado na figura 6 o modelo da arquitetura em modo simplificado (sem a caracterização da composição) e também as possíveis interações entre seus componentes:

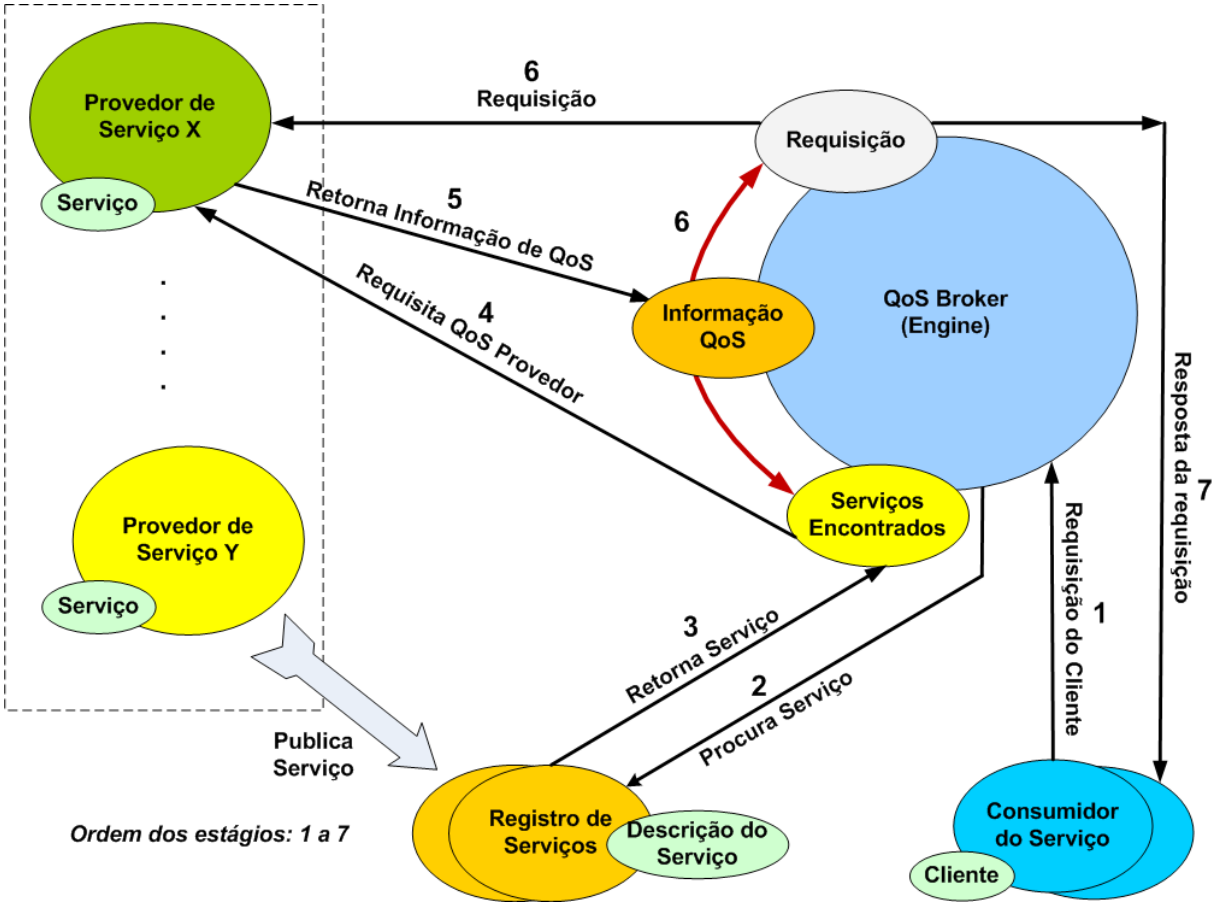

Figura 6: WSARCH em modo simplificado – Sem composição (ESTRELLA, 2007)

Descrição das fases da arquitetura WSARCH sem composição de serviços:

1. Cliente faz requisição de serviço com especificação de QoS para o *broker*, este

contendo informações atualizadas do provedor do serviço (carga, etc);

2. Com base nas informações de QoS solicitadas pelo cliente do serviço, o *broker* realiza uma busca num repositório de serviços com o objetivo de encontrar o serviço mais adequado;
3. *Broker* retorna especificação do serviço apropriado;
4. Com a localização do serviço apropriado no repositório de serviços, o *broker* solicita ao provedor de serviços qual sua capacidade de atender a uma requisição (levando em consideração os parâmetros de QoS feitos pelo cliente). Se positivo, as informações do provedor são retornadas ao *broker*;
5. Nesta fase o *broker* deve cruzar as informações solicitadas pelo cliente com aquelas disponíveis pelo provedor de serviços;
6. Se tais informações coincidirem, é feito o acesso ao serviço exigindo cumprimento com os parâmetros de QoS solicitados;
7. O resultado do acesso ao serviço com QoS é então retornado ao cliente. Esse oferecimento, no entanto, pode ser aquele que o usuário solicitou ou uma QoS de "melhor esforço".

A figura 7 mostra a arquitetura WSARCH considerando composição de serviços e a inserção de um *middleware* (DWSC-M) incorporado ao *broker*. Tais fases são apresentadas na figura a seguir:

1. Cliente faz requisição de serviço com especificação de QoS para o *broker*.
2. *Broker* encontra no registro de serviços o serviço e descobre que a requisição feita é para um serviço composto. O processo de composição no modelo inicial deve ser feito no provedor de serviço composto. Isso deve envolver:
    i. Análise do tempo para o processo de composição;
    ii. Busca pelos serviços no repositório;
    iii. Algoritmos para selecionar os melhores serviços;
    iv. Técnicas que podem ser consideras segundo a literatura:
        1. O W*eb Service* envolvido no processo de composição que causa sobrecarga no sistema;

2. O W*eb Service* que gera uma alta sobrecarga de comunicação devido o processamento das mensagens SOAP;

3. Capacidade de carga individual dos serviços envolvidos no processo, ou seja, a carga que cada serviço pode suportar efetivamente.

3. O b*roker* recebe do registro de serviços todos os serviços candidatos a fazerem parte do serviço composto.

4. O *broker* busca informações de QoS dos serviços candidatos a fazerem parte do serviço composto junto a seus respectivos provedores de serviço.

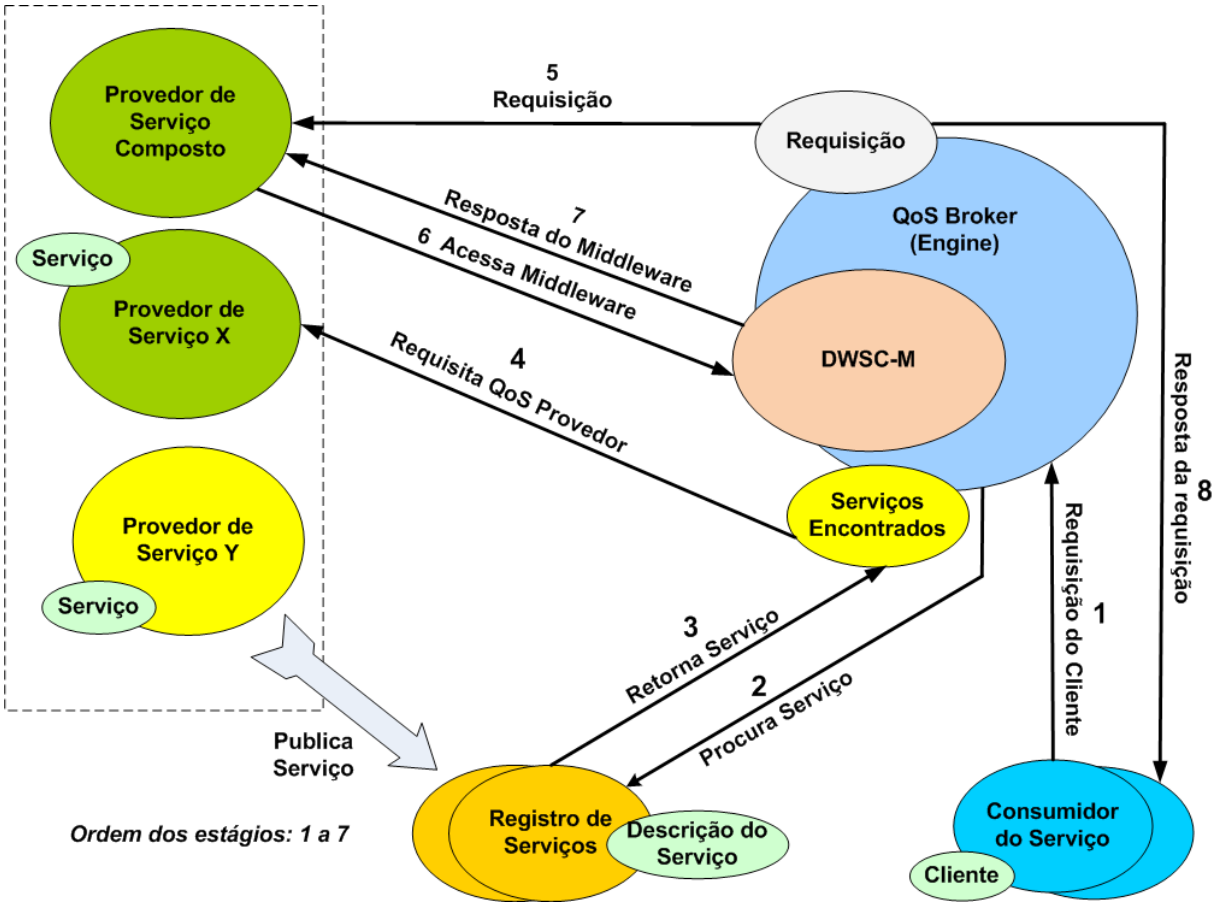

Figura 7: WSARCH em modo simplificado – Com composição (ESTRELLA, 2007)

5. O processo de composição no modelo inicial deve ser feito no provedor de serviço composto, por isso a requisição de serviço composto é submetida a ele. Isso deve envolver:

   v. Análise do tempo para o processo de composição;

   vi. Algoritmos para selecionar os melhores serviços;

6. Durante a execução do serviço composto, o provedor de serviço composto dispara requisições ao DWSC-M, parte integrante do *broker*, que será o responsável por acessar os serviços que poderão atender a requisição com QoS do cliente.
7. O DWSC-M envia as respostas dos serviços ao provedor de serviço composto. Os passos 6 e 7 são feitos para cada serviço componente do serviço composto.
8. O resultado do processo de composição será então computado no provedor de serviço composto, que retorna ao cliente o resultado do acesso ao serviço composto (adequada à solicitação de QoS).

4.4 Consideração Finais

A arquitetura apresentada neste capítulo abordou um trabalho no qual há um esforço no sentido de desenvolver uma arquitetura completa de *Web Services,* que considere qualidade de serviço na interação entre as entidades participantes (cliente, *broker*, provedor, UDDI). Isto constitui um trabalho muito importante no contexto deste projeto de mestrado, pois por meio dele vai ser possível realizar experimentos de composição de *Web Services* em um ambiente que irá tratar diversas situações que envolvem *Web Services* (troca de mensagens, composição de serviços, registro de serviços, etc.). O próximo capítulo destaca a construção de um módulo independente de composição de serviços a ser aplicado na arquitetura WSARCH e também em qualquer arquitetura SOA existente.

O próximo capítulo irá apresentar a ferramenta DWSC-M que foi desenvolvida durante este trabalho de mestrado. Tal ferramenta possibilitou a criação de *Web Services* compostos com qualidade de serviço.

Capítulo

5

Dynamic Web Service Composition Middleware

5.1 Considerações iniciais

Composição de serviços é uma solução emergente e muita pesquisa ainda é feita para a obtenção de composição dinâmica, já que esta ainda está restrita a protótipos e não há atualmente nenhuma ferramenta disponível para utilização desse tipo de composição para desenvolvedores. A composição dinâmica é importante para o desenvolvimento deste trabalho, pois, é necessário considerar a característica dinâmica da Web quando se trata de questões como qualidade de serviço. Sendo assim, são necessários mecanismos para o acesso a serviços equivalentes mas hospedados em provedores de serviços diferentes.

Este capítulo apresenta um *middleware* denominado DWSC-M (*Dynamic Web Service Composition Middleware*), desenvolvido visando o objtivo deste trabalho de mestrado, ou seja, realizar composição de *Web Services* com qualidade de serviço (QoS). Este *Middleware* é composto de um mecanismo capaz de encontrar *Web Services* similares por meio de um registro UDDI, de modo que inicialmente um algoritmo de seleção de serviços seja aplicado e posteriormente seja utilizado um segundo mecanismo para fazer o acesso dinâmico ao serviço encontrado.

5.2 Dynamic Web Service Composition Middleware

O *middleware* apresentado neste capítulo apresenta dois objetivos principais: o primeiro é ser um ambiente capaz de compor serviços dinamicamente e o segundo é a capacidade de dar suporte aos algoritmos com provisão de QoS para seleção de *Web Services* que serão parte de um processo de composição.

Para utilizar o *middleware* é necessário realizar algumas alterações no modo de construir um serviço composto em relação ao modo estático. Essa alteração é apenas na

maneira de como se constrói o fluxo por meio da linguagem BPEL. Vale ressaltar que nenhuma alteração no *engine* de composição é necessária devido ao fato do *middleware* ser um *Web Service.* Acredita-se que qualquer linguagem ou *engine* de composição de serviço possa trabalhar com o *middleware* pelo fato dele ser um *Web Service.* No entanto os testes realizados com o *middleware* foram feitos utilizando apenas a linguagem BPEL e o *engine* ActiveBPEL.

Para escrever um serviço composto dinâmico com o DWSC-M, é preciso considerar um passo adicional que denominamos de operação **QoSCalc.** Essa operação recebe como parâmetro quais os serviços farão parte do processo composto, o que permite ao DWSC-M obter uma visão geral do serviço composto e então estabelecer a QoS absoluta para ele por meio de um algoritmo de seleção incorporado ao *middleware*. O DWSC-M ainda considera aspectos importantes para aplicação de QoS em composição de *Web Services,* tais como*:* fluxos paralelos e *loops* de serviços. Ele identifica quais trechos do fluxo são sequenciais, paralelos e *loops* por meio da gramática apresentada na tabela 1.

Outra mudança para criação de um serviço composto utilizando o *middleware* é em relação à maneira de como os serviços participantes do fluxo devem ser acessados. Ao invés de adicionar a referência direta ao serviço participante é necessário adicionar a referência à operação **dynamicCall** do *middleware* passando como parâmetro o nome real do serviço a ser utilizado (o nome precisa ser exatamente o cadastrado no registro UDDI, pois esta ferramenta ainda não contempla conceitos de Ontologia), qual operação do serviço deverá ser acessada e também os parâmetros da operação do serviço. O *middleware* foi implementado utilizando as ferramentas de código aberto Apache Tomcat, jUDDI Server, ActiveBPEL e Apache Axis2.

Tabela 1: WS-Flow Grammar – Uma gramática para definição de fluxo para Web Services

flow := service+
service := ‘$’ ident name	‘&’ num name	name
num := digit +
ident := num ‘.’ num
name := alpha +
digit := [0-9]
alpha := [‘a’-‘z’] +	[‘A’-‘Z’] +	digit +

Há três tipos de serviços que podem ser identificados pela gramática. Esses serviços são representados por **('$' ident name)**, **('&' num name)** e **(name)**, os quais são definidos a

seguir:

- **('$' ident name)** identifica quais serviços irão executar paralelamente por meio do identificador **('$')** e o primeiro **(num)** de **(ident)** identifica em qual fluxo paralelo está o serviço. O segundo **(num)** de **(ident)** identifica dentro do fluxo paralelo se o serviço é sequencial a outro.
- **('&' num name)** identifica um trecho do fluxo onde ocorre laço em que o símbolo **'&'** identifica o laço, **num** identifica o número de iterações do laço e *name* identifica o nome do serviço que está no laço.
- **(name)** identifica que é um serviço simples executado de forma sequencial no fluxo.

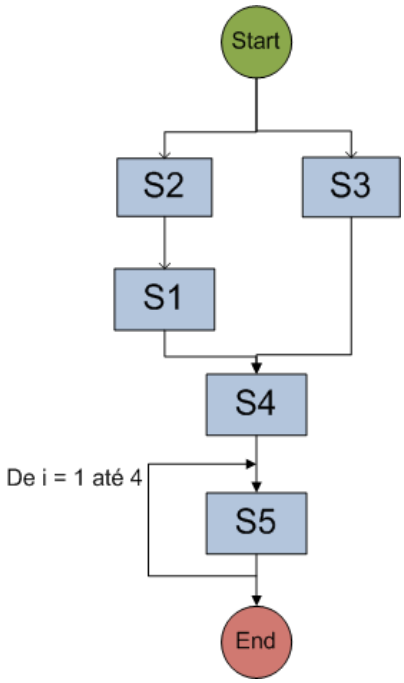

Figura 8: Tipos de fluxos em composição de Web Services

Utilizando a figura 8 como exemplo e serviços enviados como parâmetros para operação **QoSCalc** ($1.1S_2$ $1.1S_1$ $1.0S_3$ S_4 &$4S_5$) onde S2 e S1 são sequenciais entre eles mas paralelos em relação ao serviço S3. O serviço S4 é um simples *Web Service* sequencial e S5 é um serviço que será executado quatro vezes em um laço.

5.2.1 Apache Tomcat

Atualmente a Web não possui apenas páginas estáticas, mas também inúmeras paginás dinâmicas, as quais, em sua maioria, contêm conteúdos que são gerados de acordo com a interação do usuário com o site. *Common Gateway Interface* (CGI) é o mecanismo original que tornou possível a dinamicidade existente em páginas Web. Posteriormente a Microsoft

lançou o *Active Server Pages* (ASP) que permitia a execução de *scripts* dentro de código HTML. O ASP teve enorme sucesso e foi o catalisador para a linguagem Java para Web, surgindo inicialmente na Web por meio dos *applets (* aplicações que são executadas no *browser* do cliente) (Moodie, 2007). A grande contribuição de Java para Web começou com o surgimento dos *Servlets*.

Servlet é uma solução que oferece ao programador Java uma interface para criação de aplicações que possam ser disponibilizadas em servidores Web.

O Tomcat tem sua origem nos desenvolvimentos mais recentes para *servlets*. A Sun Microsystems criou o primeiro contêiner de *servlet*, Java Web Server, para demonstrar sua solução, mas este servidor não era muito robusto. Ao mesmo tempo, a *Apache Software Foundation* (ASF) criou o JServ, um *engine* para *servlets* integrada aos servidor Web Apache.

Em 1999, a Sun Microsystems doou o código do Java Web Server para Apache e os dois projetos foram fundidos dando origem ao Tomcat. A versão 3.x foi a primeira da série Tomcat e foi descendente direta do código original provido pela Sun Microsystems para a Apache.

5.2.2 Apache Axis2

O Apache Axis2 (Apache eXtensible Interaction System) é um *engine* para processamento de mensagens SOAP que possui implementações para Java e C++. Diferentemente do Axis 1.0, o Axis2 apresenta diversas melhorias, principalmente em relação ao desempenho e também em relação à modularidade. A arquitetura do Axis2 é separada em componentes (módulos), e subdivide-se em componentes do núcleo e componentes não pertencentes ao núcleo (JAYASINGHE., 2008), (FOUNDATION, 2009).

O Axis2 é disponibilizado por meio de um servidor independente que é pouco robusto ou como Web Archive (WAR). WAR é uma forma de distribuição de aplicações Web, onde a aplicação é empacotada em um arquivo mantendo um padrão de pastas para que a aplicação possa ser instalada em qualquer contêiner de *servlet*. Isso permite instalar o *engine* axis2 em servidores de aplicação mais robustos como o Apache Tomcat ou outros.

5.2.3 ActiveBPEL

O ActiveBPEL é uma ferramenta de código aberto, o que possibilita acrescentar novas funcionalidades à ferramenta. Isso não foi preciso durante o desenvolvimento deste projeto pelo fato do DWSC-M ter sido desenvolvido para suportar qualquer *engine* para composição de *Web Services* (ACTIVE ENDPOINTS, 2008).

Os mecanismos do ActiveBPEL são uma implementação aberta das especificações da linguagem BPEL, escrito em Java. Ele lê as definições de processos BPEL e cria representações de processos BPEL. Quando uma mensagem recebida dispara um início de atividade, o mecanismo cria uma nova instância de processo e começa a executar. O mecanismo cuida da persistência, das filas, dos alarmes e dos muitos outros detalhes da execução. O ActiveBPEL *engine* pode ser instalado, por exemplo, em um contêiner de *servlet* como o Apache Tomcat.

A caracterização de um processo BPEL sem uma ferramenta de auxílio é mais propensa ao erro e torna o processo de desenvolvimento tedioso. Para isso, é utilizado o ActiveBPEL Designer, uma ferramenta que auxilia no desenvolvimento de serviço composto por meio de uma interface gráfica de arrastar e soltar componentes para criação do serviço composto.

5.2.4 jUDDI Server

Na literatura sobre *Web Services* é apresentado também um protocolo para serviços de diretório que contém as descrições dos *Web Services*, denominado UDDI (*Universal Description Discovery and Integration*). Esse protocolo funciona como um registro de serviços sendo um importante componente da arquitetura orientada a serviços (Farkas, 2003).

O UDDI permite ainda aos clientes do sistema localizar serviços ou descobrir seus detalhes e permite que registros operacionais sejam mantidos para diferentes propósitos em diferentes contextos. O jUDDI é uma implementação Java *open source* da especificação do UDDI para *Web Services*.

5.3 DWSC-M – Ponto de vista do usuário

A arquitetura do DWSC-M sob o ponto de vista do usuário do *middleware* é apresentada na figura 9.

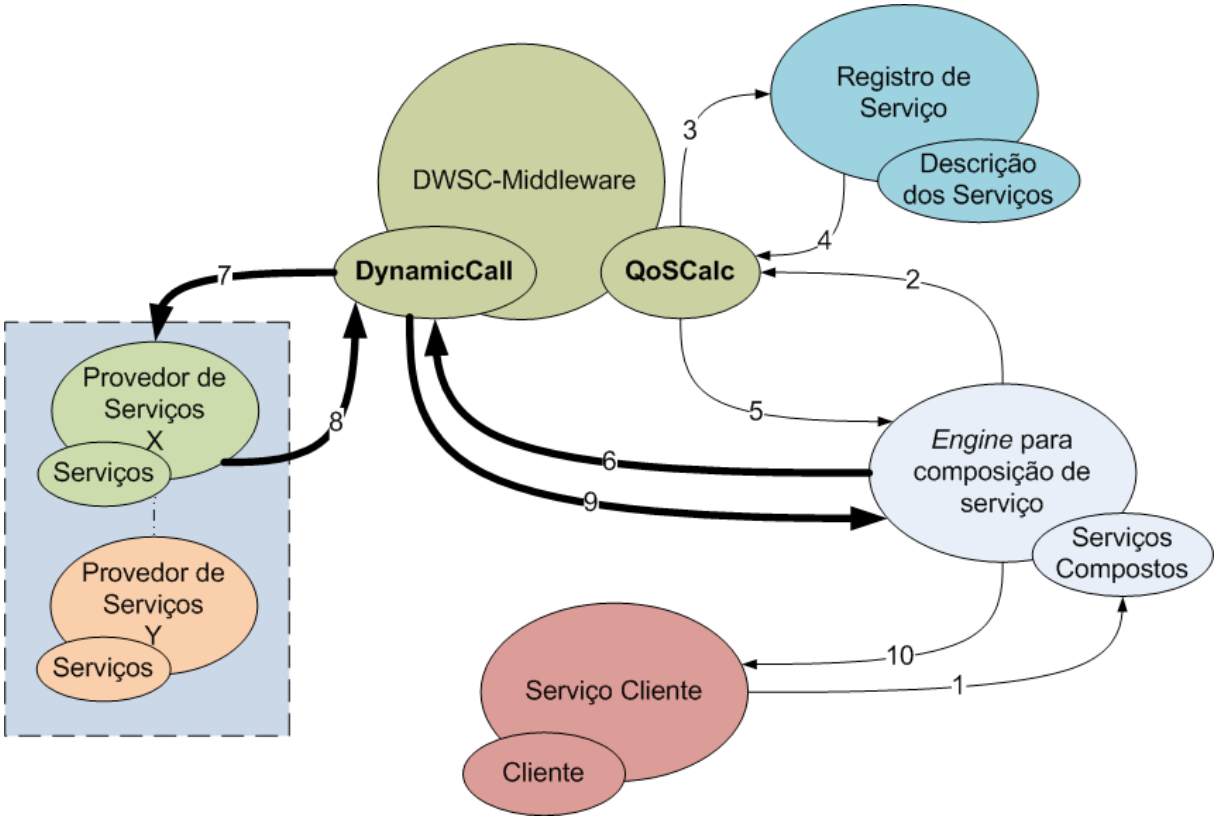

Figura 9: Middleware para composição dinâmica de Web Services – Visão do usuário

Os passos enumerados na figura 9 apresentam as seguintes chamadas:

1. O serviço cliente faz requisição de serviço composto com critérios de QoS ao engine de composição de serviços.
2. No serviço composto o primeiro passo é fazer um acesso à operação **QoSCalc** do **DWSC-M** informando os serviços do fluxo a serem executados.
3. A operação **QoSCalc** faz buscas no registro de serviços para encontrar os serviços candidatos a serem parte do fluxo composto.
4. O registro de serviços envia à operação **QoSCalc** informações de QoS e como acessar os serviços candidatos a composição
5. A operação **QoSCalc** define qual implementação de cada serviço deverá ser executada de acordo com os parâmetros de QoS solicitados pelo usuário por meio do algoritmo

de seleção de serviços, gera um identificador para os serviços escolhidos e o envia para o *engine* de composição de serviços.

6. O *engine* para composição de serviços acessa a operação **dynamicCall** passando como parâmetro para ela o identificador recebido no passo 5, o nome do serviço a ser acessado, o nome da operação a ser acessada e os parâmetros a serem passados para a operação a ser acessada.
7. A operação **dynamicCall** encontra a WSDL do serviço a ser acessado pelo identificador recebido como parâmetro e por meio de interpretação automática das informações da WSDL e as demais informações recebidas como parâmetros a operação **dynamicCall** gera automaticamente a mensagem SOAP para acessar a devida implementação do serviço e faz o acesso a ele.
8. Serviço é executado e envia o resultado de sua execução à operação **dynamicCall.**
9. A operação **dynamicCall** envia ao *engine* de composição de serviço a resposta do acesso à implementação do serviço. O *engine* de composição então repete os passos 6, 7, 8 e 9 para todos os serviços componentes do fluxo até terminar sua execução.
10. O *engine* de composição de serviços envia ao usuário o resultado da execução do serviço composto por ele requisitado.

5.4 DWSC-M – Visão do sistema

Para detalhar o funcionamento interno do *middleware*, é preciso destacar duas operações principais: a operação **QoSCalc** e a operação **dynamicCall**.

A operação **QoSCalc** recebe como parâmetro todos os *Web Services* lógicos (apenas o nome do serviço e não o endereço da implementação) que o serviço composto irá acessar, identificados por símbolos que irão classificá-los quanto à disposição do fluxo: paralelo, sequencial ou laço. A gramática apresentada na tabela 1 é utilizada para fazer o *parser* (processamento) dos parâmetros e identificar qual é a classificação e a disposição de cada *Web Service* no fluxo.

Os passos da operação **QoSCalc** apresentados na figura 10 são:

- **Passo 1:** Encontrar serviços participantes do processo de composição por meio de uma

consulta ao UDDI

- **Passo 2:** O UDDI retorna à operação **QoSCalc** as WSDLs e valores de QoS das opções de implementações dos serviços que serão parte do serviço composto. A operação **QoSCalc** irá armazenar estes valores de QoS em uma Matriz
- **Passo 3:** Algorítimo de seleção define quais as implementações do serviço deverão ser utilizadas
- **Passo 4:** Armazena em uma base de dados o melhor fluxo encontrado pelo algoritmo de seleção e gera um identificador que deverá ser enviado ao fluxo como resposta da chamada à operação **QoSCalc**.

A operação **dynamicCall** recebe como parâmetros o nome do serviço a ser acessado, a operação que o serviço deverá acessar e os parâmetros que deverão ser passados para operação do serviço. Além disso, a operação **dynamicCall** recebe também o identificador dos serviços selecionados para compor o serviço pela operação **QoSCalc**.

- **Passo 1: dynamicCall** recebe a WSDL da implementação do serviço por meio de uma pesquisa na base de dados onde estão armazenados os serviços selecionados pela operação **QoSCalc**. A pesquisa é feita pelo nome do serviço e pelo código recebido da operação **QoSCalc**.
- **Passo 2:** A operação **dynamicCall** cria a mensagem SOAP para acessar a implementação do serviço por meio da WSDL do serviço recebida no **passo 1** e o nome da operação e parâmetros para acessar o serviço que foram recebidos na chamada da operação **dynamicCall** como parâmetros.

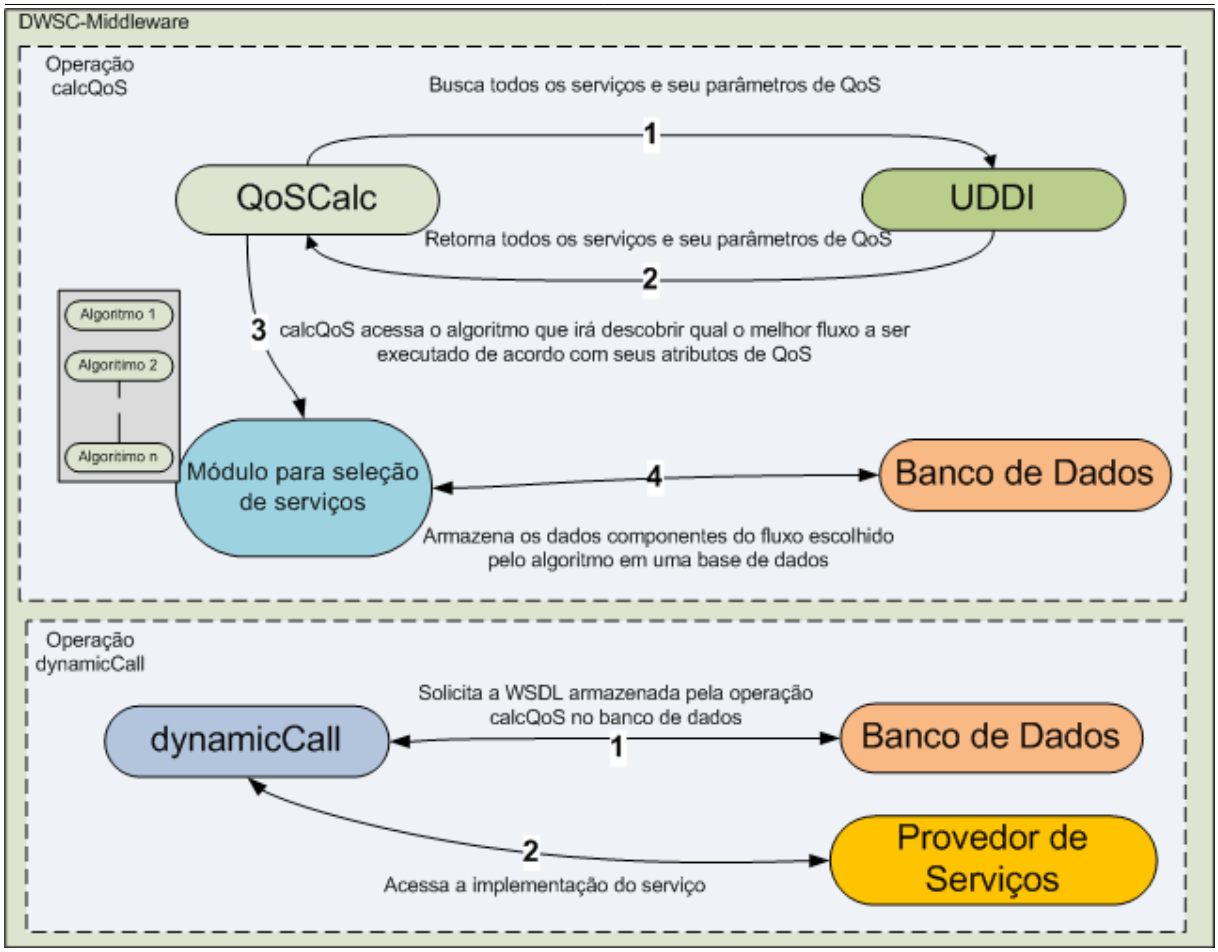

Figura 10: Middleware para composição dinâmica Web Services – Visão do Sistema

5.5 Trabalhos relacionados

Os trabalhos relacionados apresentados nesta seção são sobre ferramentas que dão suporte à adaptação dinâmica em composição de serviços.

5.5.1 VieDAME

Em Moser (2007) o autor descreve um sistema chamado VieDAME (*Vienna Dynamic Adaptation and Monitoring Environment for WS-BPEL*) que tem como objetivo monitorar o estado dos *Web Services* e fazer adaptação dinâmica de acordo com critérios estabelecidos por contratos de QoS. Nesse sistema é utilizado um monitor que identifica parâmetros de QoS de um *Web Service*. Se ele está abaixo do esperado, o sistema VieDAME intercepta a requisição do serviço e redireciona o acesso a outro Web Service equivalente em termos de entrada e saída, mas com parâmetros de QoS que atendam ao estabelecido na requisição do serviço composto. Essa abordagem é interessante, mas como ela não prevê o conhecimento do serviço

composto como um todo ela não permite definir a QoS absoluta para o Web Service Composto, impossibilitando assim a aplicação algoritmos mais sofisticados para seleção de serviço.

5.5.2 RobustBpel2

Em Ezenwoye (2007) é apresentada a ferramenta RobustBPEL 2.0, que utiliza um *proxy* para realizar o acesso aos *Web Services*. O funcionamento ocorre do seguinte modo: todo acesso ao *Web Service* feito pelo *Web Service* composto passa por um *proxy* que quando detecta que um *Web Service* não está respondendo procura dinamicamente por meio do UDDI (*Universal Description, Discovery and Integration*) um novo *Web Service* correspondente ao que falhou para que o serviço seja executado de maneira confiável. A versão do RobustBPEL 2.0 se diferencia da versão 1.0 pelo falto de fazer a escolha de um novo serviço de maneira dinâmica, pois na versão 1.0 era mantida dentro do código fonte do RobustBPEL uma lista de serviços que ele poderia utilizar ao invés de realizar uma busca por um novo serviço em algum registro de serviço como o UDDI. Essa solução apesar de conseguir manter o *Web Service* composto confiável, não provê nem um tipo de QoS para o usuário.

5.5.3 Dynamo

Em Baresi (2007) é apresentada uma solução para auto adaptação de processos BPEL. A proposta apresentada nesse trabalho é baseada no Dynamo, um *framework* desenvolvido com técnicas de programação orientada a aspecto feito como uma extensão do ActiveBPEL. Esse trabalho não apresenta de forma explicita como é solucionado o problema de seleção de serviços alternativos e também não faz menção sobre qualidade de serviço.

5.6 Considerações Finais

Este capítulo apresentou o *middleware* denominado DWSC-M capaz de dar suporte à adaptação dinâmica e QoS de serviços compostos por meio de algoritmos que podem ser trocados de acordo com a preferência do usuário da ferramenta. Detalhes técnicos e ferramentas utilizadas para o desenvolvimento também foram apresentados de forma a ilustrar como é o funcionamento do *middleware* DWSC-M. Alguns trabalhos relacionados também

foram apresentados, com o diferencial de que os trabalhos já implementados que suportam adaptação dinâmica, estão mais relacionados com tolerância à falha e não levam em consideração fatores de qualidade de serviço, este o maior motivador para o desenvolvimento do DWSC-M. O próximo capítulo irá apresentar dois algoritmos para composição de serviços propostos neste trabalho. Tais algoritmos foram utilizados para avaliar o desempenho do DWSC-M, e seus resultados serão apresentados no capítulo 7.

Capítulo

6

Algoritmos para Composição de Serviços

6.1 Considerações iniciais

Na composição de serviços com QoS é necessária a utilização de algoritmos que possam selecionar quais as melhores implementações para cada serviço que compõem um fluxo, de acordo com seus atributos de QoS. Nesse sentido, este capítulo apresenta algoritmos de seleção de serviços para um processo de composição. Dois deles foram implementados para testar a ferramenta apresentada no capítulo 5. O algoritmo **Random** seleciona de forma aleatória os serviços participantes do fluxo e o algoritmo de **Distância Euclidiana** que escolhe quais os serviços são capazes de prover a QoS relativa mais próxima da requisição do cliente.

6.2 Algoritmo para seleção aleatória de Web Services

O algoritmo de seleção aleatória (Algoritmo **Random)** faz a escolha dos serviços participantes de forma aleatória utilizando o método *random* da classe *Math* da linguagem Java. O algoritmo faz a busca no registro de serviços para determinado serviço e seleciona algum deles de forma aleatória. O código 1 apresenta o pseudocódigo do funcionamento do algoritmo de seleção aleatória de *Web Services* para serviços compostos.

Código 1: Algoritmo Random

```
Entrada ()
1. enquanto seleciona serviços do jUDDI
2.     fluxo.adiciona(random(service))
3. id = armazena fluxo
Saida (id)
```

O algoritmo Random foi proposto e implementado para servir de parâmetro na avaliação do algoritmo de Distância Euclidiana que foi proposto para seleção de *Web Services* com QoS.

6.3 Algoritmo de Distância Euclidiana

Na proposta original apresentada em (TAHER, 2005) a seleção é feita para apenas um *Web Service*. No presente projeto de mestrado, a extensão do algoritmo foi feita no sentido de adaptar a técnica para conseguir realizar a combinação dos atributos de QoS para assim definir os atributos de QoS possíveis para o serviço composto.

$$\sqrt{(x_j - x_k)^2 + (y_j - y_k)^2} \qquad (1)$$

A distância Euclidiana é calculada por meio da equação 1. O objetivo do algoritmo é encontrar o valor de QoS do serviço composto mais próximo possível do valor requisitado pelo cliente. Os passos apresentados no código 2 são:

Código 2: Algoritmo de Distância Euclidiana

Entrada (QoS_(r))

1. constrói a matriz de serviços disponíveis
2. constrói uma matriz tridimensional
3. para i=1 até y faça
4. define valores dos atributos de QoS do fluxo(i)
5. normaliza os valores de atributos de QoS
6. calcula a distrância euclidiana dos atribututos de QoS requisitado em relação ao do fluxo(i)
7. escolhe o fluxo(i) com valores dos atributos de QoS com menor Distância Euclidiana em relação aos atributos de QoS requisitados.

Saída fluxo_(i)

- **Passo 1**: Construção de uma matriz de *Web Services* disponíveis com seus atributos de QoS. A matriz apresentada na tabela 2 apresenta os serviços candidatos para compor o fluxo com seus atributos de QoS onde n, j e k são o número de implementações de

cada *Web Service* e m é o número de *Web Services* acessados pelo serviço composto.

Tabela 2: Atributos de QoS no registro de serviço

$S_{1.1}$	$S_{1.1}$ rt	$S_{1.1}$ cost
$S_{1.n}$	$S_{1.n}$ rt	$S_{1.n}$ cost
$S_{2.1}$	$S_{2.1}$ rt	$S_{2.1}$ cost
$S_{2.j}$	$S_{2.j}$ rt	$S_{2.j}$ cost
$S_{m.1}$	$S_{m.1}$ rt	$S_{m.1}$cost
$S_{m.k}$	$S_{m.k}$ rt	$S_{m.k}$ cost

- **Passo 2:** Construção de uma matriz tridimensional contendo a combinação dos *Web Service*s listados na tabela 1 em todos os fluxos possíveis que o *Web Service* composto pode acessar. A matriz tridimensional ainda contém informações dos atributos de QoS de cada serviço candidato a fazer parte do serviço composto no terceiro eixo da matriz tridimensional. A figura 11 apresenta um exemplo de matriz tridimensional contendo custo e tempo de resposta como atributos de QoS. Muitos outros atributos de QoS são apresentados em (Kalepu, 2003). Neste exemplo foram escolhidos apenas dois atributos de QoS (custo e tempo de resposta) de modo a facilitar a visualização da construção da matriz tridimensional.

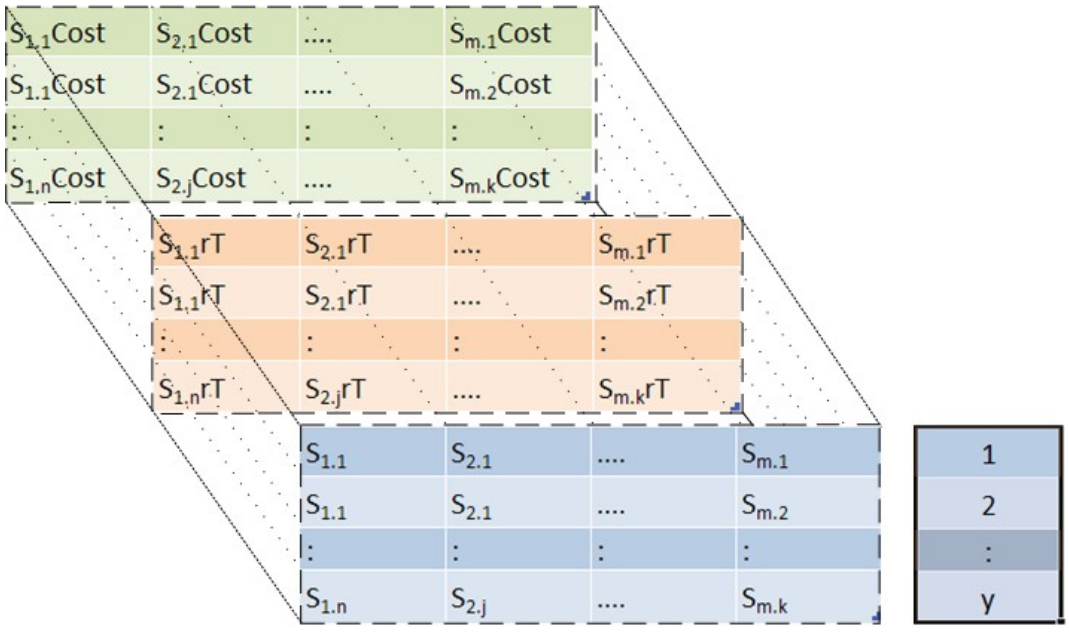

Figura 11: Representação do algoritmo de Distância Euclidiana por meio da construção de uma matriz tridimensional

- **Passo 3:** Laço de 1 até y, onde y é o número de linhas da matriz tridimensional
- **Passo 4:** É realizado o cálculo para definir os valores dos atributos de QoS para cada fluxo encontrado (linha da matriz).
- **Passo 5:** Neste passo, é realizada uma normalização dos valores dos atributos de QoS obtidos no passo 4, essa normalização é feita por meio do cálculo da porcentagem da diferença entre os atributos de QoS dos serviços candidatos e dos atributos de QoS exigidos pelo usuário. Com todos os atributos de QoS normalizados, o peso de cada atributo que o algoritmo de distância Euclidiana utilizará para tomar a decisão de qual fluxo é o melhor para atender a requisição do cliente torna-se justa.
- **Passo 6:** Calcula a distância Euclidiana entre os atributos de QoS de *flow(i)* e a dos atributos de QoS que o cliente requisitou por meio da equação X.
- **Passo 7:** Escolhe o *flow(i)* com os atributos de QoS que tenha a menor distância Euclidiana em relação aos atributos de QoS solicitados pelo usuário, armazena-o no banco de dados e retorna o identificador do fluxo ao *engine* de composição.

6.3.1 Estudo de Caso com algoritmo de seleção Euclidiana

A figura 12 apresenta um exemplo de serviço composto para teste do DWSC-M. Cada serviço lógico S1, S2 e S3 possuem suas próprias implementações (serviços candidatos), por exemplo S1 possui S1.1 e S1.2, S2 possui S2.1 e S2.2 e S3 possuiu S3.1 e S3.3 como está descrito na tabela 3.

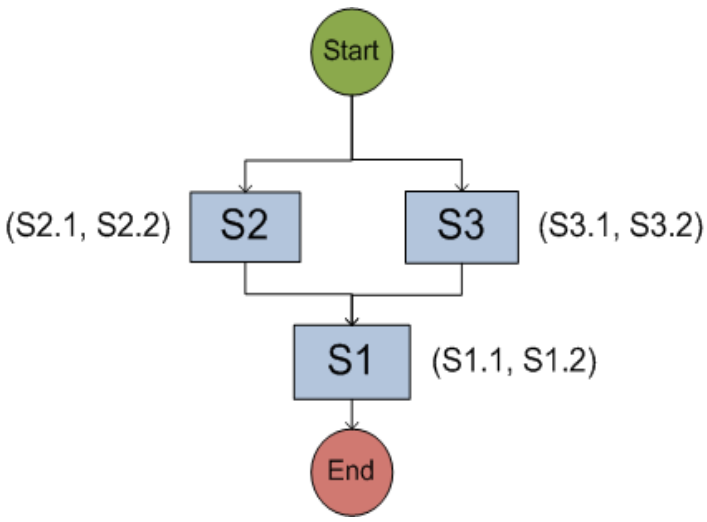

Figura 12: Exemplo de fluxo

Tabela 3: Valores de atributos de QoS no registro de serviços

Nome do Serviço	Tempo de Resposta	Custo
S1.1	4s	$0,30
S1.2	6s	$0,20
S2.1	2s	$0,30
S2.2	3s	$0,20
S3.1	3s	$0,30
S3.2	5s	$0,20

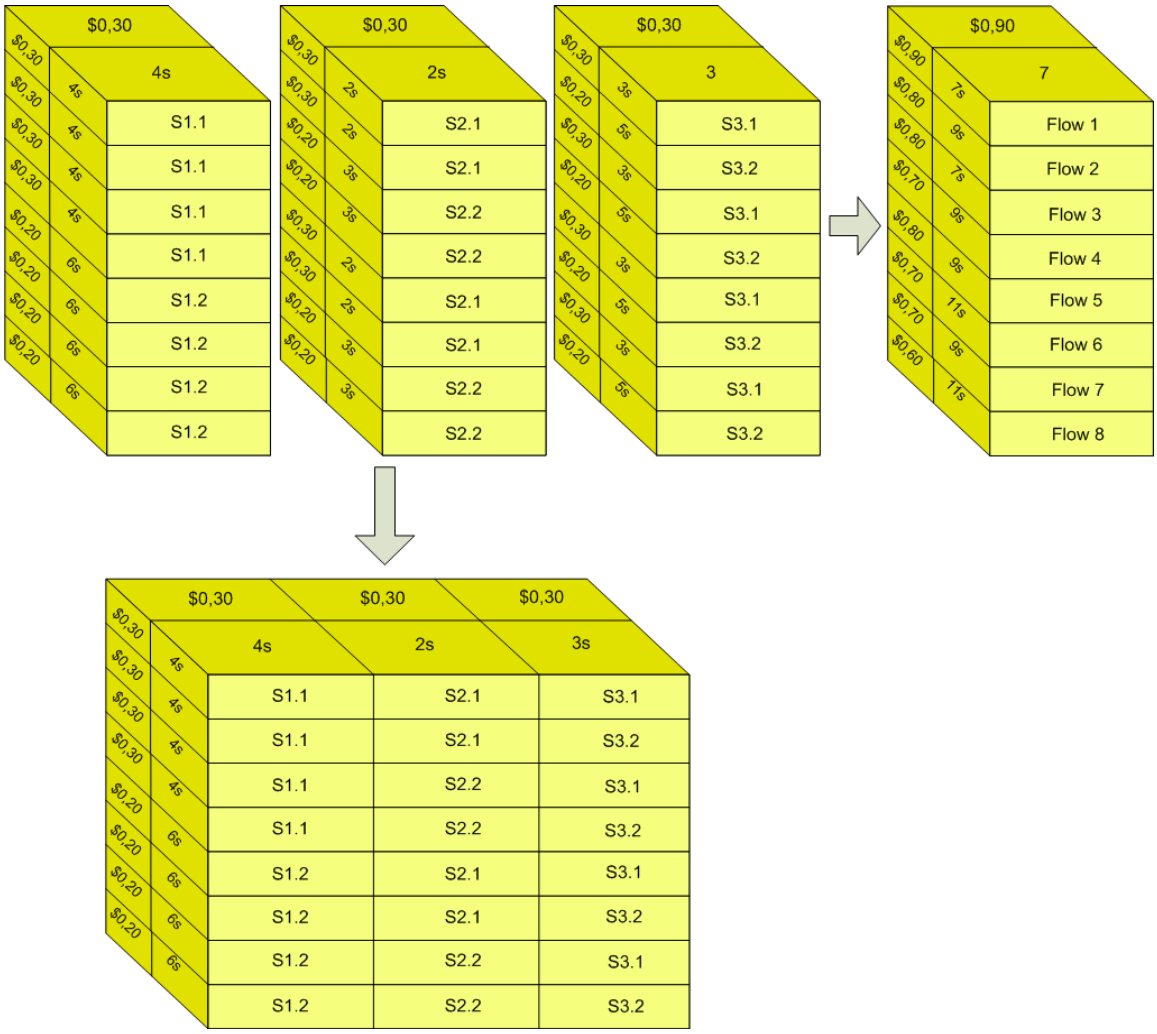

Figura 13: Matriz tridimensional com serviços candidatos e seu atributos de QoS

A matriz tridimensional apresentado na figura 13 é construído por meio da combinação das informações sobre os serviços contidos na tabela 3. Cada linha da matriz tridimensional resulta em um fluxo de serviços candidatos ao serviço composto. O algoritmo de distância Euclidiana faz uma busca linha a linha procurando pelo fluxo que apresente os atributos de QoS mais próximos possíveis dos solicitados pelo cliente. Por exemplo, se um

cliente faz uma requisição ao serviço composto mostrado na figura 12 com atributo de QoS custo = $0,80 e tempo de resposta = 9s, o algoritmo irá encontrar a segunda e quinta linha como melhores casos para atender a requisição, pois a distância Euclidiana em relação a elas será 0, como pode ser observado comparando com a matriz tridimensional da figura 13. Nesta requisição é possível verificar que o algoritmo sempre tenta encontrar o fluxo com valores de QoS mais próximos possíveis dos requisitados. Mesmo que seja possível encontrar um fluxo de serviços com melhor desempenho, o algoritmo o descarta por não ser o mais próximo do que foi solicitado.

Neste exemplo deve ser observado que os fluxos 4 e 7 poderiam apresentar o tempo de resposta igual a 9s, no entanto, com um custo inferior, cujo valor é de $0,70.

6.4 Trabalhos Relacionados

Há outros trabalhos na literatura com o propósito de oferecer serviços compostos de acordo com exigências de qualidade de serviços, alguns utilizando algoritmos de seleção baseados em composição manual dinâmica e outros utilizando composição automática dinâmica. Nesta seção serão apresentadas algumas destas soluções.

6.4.1 Seleção de serviços por classes de usuário

O trabalho apresentado em (Cardellini, 2007) relata um modelo de seleção de serviços que utiliza classes de usuários e de acordo com essas classes os serviços são escolhidos passo a passo. As classes são ouro, prata e bronze. Na classe ouro os melhores serviços são escolhidos para fazer parte da composição do serviço, na classe prata é feito uma parte da seleção com os melhores serviços e outra com os piores serviços. No entanto, na classe bronze, os piores serviços são selecionados. Essa solução é interessante, uma vez que sempre prioriza clientes que paguem mais pelo serviço. No entanto, essa solução, por não considerar a visão do fluxo como um todo, pode prejudicar os clientes em situações que ocorrem em fluxo paralelo. Neste caso, o cliente ouro poderia acessar serviços com menor custo e com tempo de resposta que não afetariam o tempo de resposta total do fluxo. A seleção do fluxo é feita por um *broker*, mas não são apresentados detalhes de implementação ou quais ferramentas foram utilizadas.

6.4.2 Pattern-wise Selection

No trabalho apresentado em (GRONMO e JAEGER, 2005) o autor apresenta uma abordagem para seleção de serviços um fluxo composto, em que é considerada a visão do fluxo como um todo e não serviços de forma isolada. A idéia apresentada para seleção de serviços consiste em construir todas as combinações de serviços possíveis para o fluxo, fazer a normalização dos valores para permitir uma comparação mais justa, criar pesos para os atributos de QoS e então criar uma nota para cada uma das combinações. Também são levados em conta nessa nota fatores que quanto menores melhores (tempo de resposta), e também fatores que quanto maiores são melhores (*throughput).* A proposta apresentada nesse trabalho é aplicada na escolha dos melhores serviços, sendo possível pelo fato de considerar pesos aos atributos de QoS mais importantes. Entretanto, nem sempre todos clientes irão compartilhar a idéia de que aquele atributo que foi considerado mais importante é de fato o mais importante. Isso pode causar insatisfação no cliente pelo fato de o atendimento não ser realizado de acordo com suas preferências. Nesse contexto o Algoritmo de seleção Euclidiana pode sinalizar para uma maior satisfação ao usuário, pelo fato do usuário decidir os critérios de quais combinações de atributos de QoS irão ser mais adequados. No entanto é necessário realizar testes comparativos entre as abordagens e concluir qual delas é mais eficiente. Ainda em relação a abordagem de seleção de serviços para um fluxo composto, outros importantes trabalhos que propõe solução para problemas de composição de *Web Services* com QoS devem ser mencionados, (Nguyen, Kowalczyk e Phan, 2006) e (Wang et. al., 2006) que utilizam lógica Fuzzy para seleção de serviços e o trabalho apresentado em (De Paoli, Lulli e Maurino,2006) que também utiliza modelos matemáticos para solução do problema de seleção de serviços. Como trabalhos futuros prentende-se implementar tais abordagens, uma vez que algumas delas são apenas modelos.

6.5 Considerações Finais

Este capítulo apresentou dois algoritmos de seleção de serviços para composição de *Web Services*. O algoritmo de seleção que utiliza distância Euclidiana para atender requisições considerando aspectos de QoS e o algoritmo de seleção aleatória, proposto para servir de parâmetro para comparação quando a composição de *Web Services* não considera parâmetros

de QoS. Foram apresentados também alguns trabalhos relacionados que deverão ser utilizados em trabalhos futuros para avaliação de desempenho com diferentes algoritmos para composição de *Web Services* com QoS.

Capítulo

7

Análise de Resultados

7.1 Considerações iniciais

Este capítulo irá abordar a avaliação de desempenho do DWSC-M. Essa avaliação será feita utilizando os algoritmos de seleção aleatória e de distância Euclidiana. O objetivo deste capítulo é considerar o comportamento da ferramenta quando submetida a requisições simultâneas e também destacar a eficácia dos algoritmos de seleção.

7.2 Ambiente de testes

O experimento foi planejado com base na tabela 4, onde o fator Quantidade Clientes Simultâneos é referente a quantidade de clientes que fazem acesso concorrente ao serviço composto de modo a gerar uma sobrecarga no sistema. O algoritmo de Seleção é aquele utilizado para selecionar os serviços componentes do fluxo de serviço composto e o tamanho do fluxo é o número de serviços que compõem o fluxo.

A metodologia escolhida para a realização do experimentos foi fatorial completo, com cada execução de experimento sendo repetida 25 vezes, de modo a obter o intervalo de confiança de 95% das variáveis de respostas, resultando em 12 experimentos (300 execuções).

Tabela 4: Fatores e Níveis relacionados ao Planejamento de Experimentos

Fatores	**Níveis**
Quantidade de Clientes Simultâneos	• 3 clientes • 19 clientes • 38 clientes
Algoritmo de Seleção	• Distância Euclidiana • *Random*
Tamanho do Fluxo	• 5 serviços • 8 serviços

O fluxo de execução criado para os serviços compostos por 5 e 8 serviços são apresentados na figura 14. Os valores de QoS para cada implementação de serviço é apresentados na tabela 5. Os valores de QoS de cada serviço varia por provedor, ou seja o serviço S1 e S2 do mesmo provedor de serviço tem os mesmos atributos de QoS. O atributo de QoS Reputação com valor 1 é a melhor reputação que um serviço consegue obter e com valor 3 é a pior reputação que um serviço consegue obter.

Tabela 5: Valores dos Atributos de QoS oferecidos por cada provedor de serviço

Provedor	Tempo de Resposta	Reputação	Custo
1	1s	1	R$ 0,30
2	3s	2	R$ 0,25
3	5s	2	R$ 0,20
4	7s	3	R$ 0,15
5	9s	3	R$ 0,10
6	10s	3	R$ 0,05

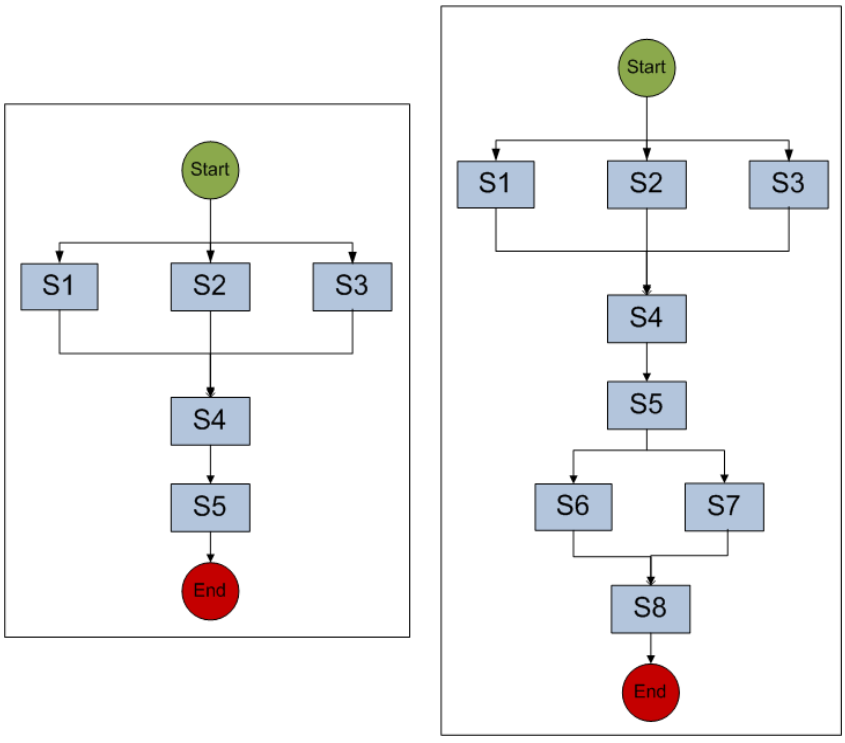

Figura 14: Fluxos de execução para serviço composto

O ambiente utilizado para a realização dos experimentos é apresentado na figura 15. O ambiente é composto por unidades de processamento que funcionam como clientes, uma

unidade de processamento configurada com o *engine* para composição de *Web Services* e o *middleware* DWSC-M, uma unidade de processamento representando o registro de serviços UDDI e seis provedores de *Web Services*, cada um oferecendo atributos de QoS diferentes.

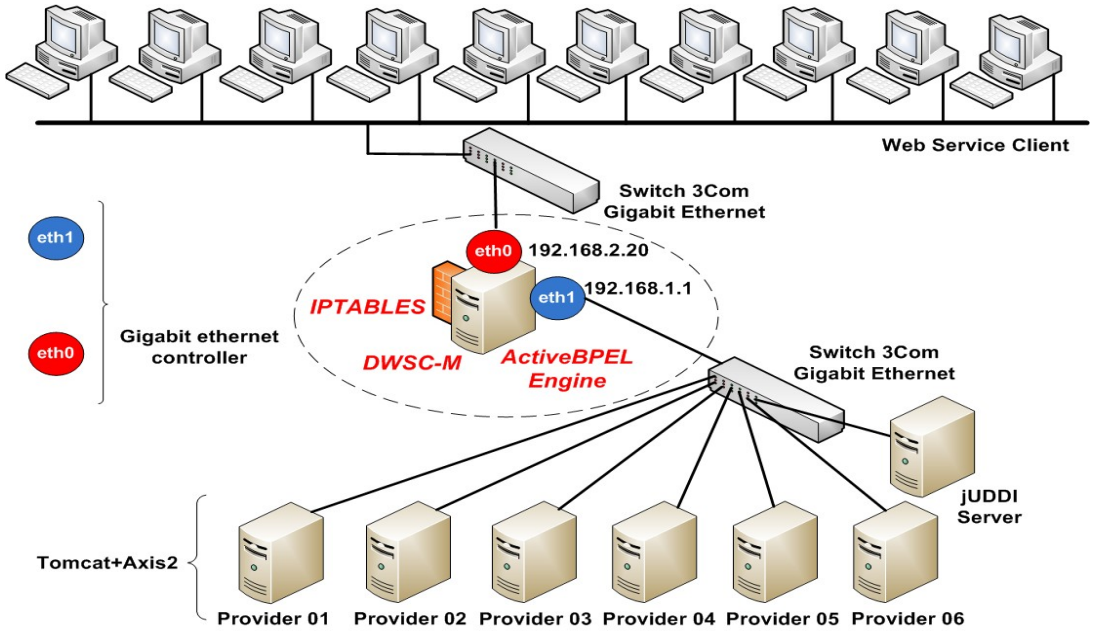

Figura 15: Ambiente utilizado para fazer experimentos

7.3 Avaliação Comportamental

Nesta seção serão apresentados os resultados dos experimentos discutidos na seção 7.2. A apresentação será dividida em 6 subseções, cada uma contendo a comparação entre os algoritmos de distância Euclidiana e o algoritmo de seleção aleatória variando os níveis dos seguintes fatores: Quantidade de Clientes Simultâneos e Tamanho do fluxo.

Cada subseção irá apresentar um gráfico mostrando o tempo de resposta requisitado pelo usuário, o tempo de resposta real da execução utilizando serviços selecionados para os algoritmos de seleção Euclidiana e seleção aleatória e também os tempos de resposta que os algoritmos de seleção calcularam para a execução do fluxo. Um gráfico com a reputação requisitada pelo usuário e a reputação com as quais os algoritmos de seleção responderam à requisição e um gráfico com o custo monetário para o acesso ao serviço composto calculado pelos algoritmos de seleção, também são apresentados. E por fim, serão discutidos os resultados contendo a satisfação do cliente, que mede o quanto cada cliente ficou satisfeito com o atendimento de sua requisição. O cálculo da satisfação do usuário foi feito por meio da

porcentagem de satisfação de cada cliente, onde cada atributo de QoS que é atendido seguindo a exigência da requisição do cliente, ganha até 100% de satisfação. Atributos de QoS que são parcialmente atendidos recebem sua porcentagem de satisfação descontada de acordo com o quão ruim foi o atendimento. Por exemplo, se a requisição de tempo de resposta for igual a 1s e o tempo de resposta obtido for 2s, esta requisição recebe 0% de satisfação, mas se a mesma requisição de 1s for atendida em 1,5 segundos a requisição receberá 50% de satisfação. Depois de calculado separadamente a porcentagem de cada atributo de QoS é realizado a média dos três atributos para obter o valor final de satisfação para aquela requisição.

7.3.1 Teste com 3 clientes simultâneos e fluxo composto por 5 serviços

O gráfico apresentado na figura 16 contém os tempos de resposta para um fluxo de serviço composto por 5 *Web Services*. Neste gráfico é possível observar que o algoritmo de seleção que utiliza distância Euclidiana é bastante eficiente quando apenas 3 clientes utilizam o serviço ao simultaneamente. Apenas o cliente 2 não conseguiu obter um tempo de resposta de acordo com o requisitado. O algoritmo escolheu uma combinação de serviços que resultou em tempo de resposta igual a 16 segundos ao invés dos 15 segundos requisitados. A recusa no atendimento pode ter ocorrido porque a estratégia do algoritmo é atender a requisição visando o equilíbrio entre todos os atributos de QoS solicitados pelo cliente para conseguir manter maior possível, uma satisfação total da requisição. Outro fato possível de observar com os resultados deste experimento é que a sobrecarga gerada no tempo de resposta pelo algoritmo de seleção utilizando o algoritmo de seleção Euclidiana foi baixo, tanto que o tempo de resposta médio com intervalo de confiança, neste teste, ficou abaixo do calculado pelo algoritmo para os três clientes.

O algoritmo de seleção aleatório, por sua vez, manteve a mesma média de tempo de resposta para os três clientes, isso ocorreu por que ele não leva em consideração os atributos de QoS solicitados pelo cliente. Esse comportamento do algoritmo resultou em um tempo de resposta ruim para o primeiro cliente que é bastante exigente em relação ao tempo de resposta. Para o segundo cliente, que solicitou um tempo de resposta mediano, sua requisição não foi perfeitamente atendida pelo algoritmo, no entanto manteve um grau de satisfação adequado no atendimento. Por fim, o terceiro usuário, o menos exigente em relação ao tempo de resposta, foi mais bem atendido, até mesmo em relação ao algoritmo de seleção Euclidiana.

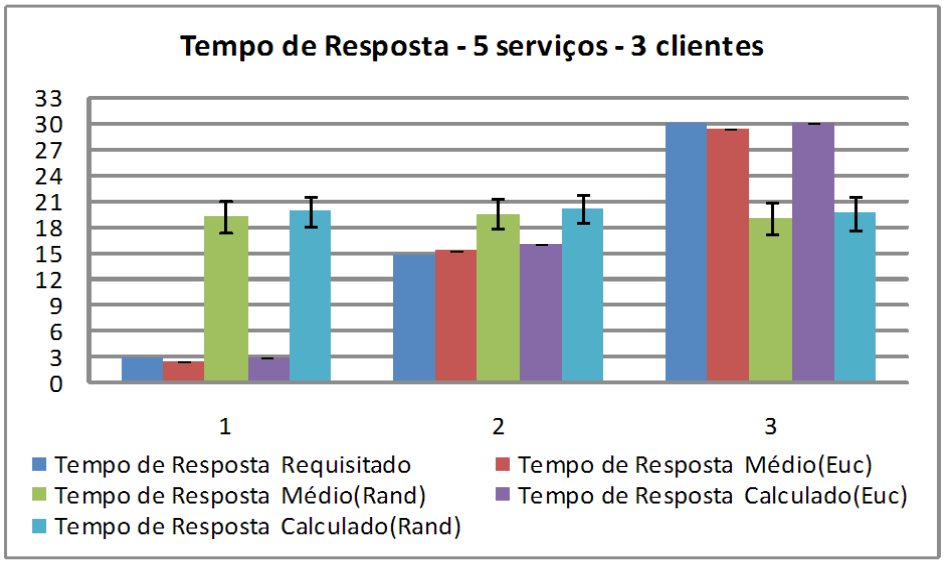

Figura 16: Gráfico de tempo de resposta – fluxo com 5 serviços e 3 clientes simultâneos

A figura 17 apresenta o gráfico que contém a reputação requisitada pelo cliente e a que os algoritmos de seleção Euclidiana e aleatória encontraram para cada cliente. A melhor reputação é a de valor 1 e a pior é a de valor 3. O algoritmo de seleção aleatório não foi totalmente satisfatório apenas no cliente 2. Assim como no tempo de resposta, esse não atendimento completo deste atributo pode ter ocorrido em função de um melhor equilíbrio entre todos os atributos de QoS requisitados pelo cliente. Nos demais casos o algoritmo conseguiu atender o que foi requisitado pelo cliente.

O algoritmo aleatório manteve o comportamento apresentado no tempo de resposta, mantendo sempre a mesma média para todos os clientes. Isso ocorre pois ele não considera os atributos de QoS requisitados pelo cliente. O algoritmo de seleção aleatório atendeu de forma satisfatória o cliente 3, o menos exigente, atendeu razoavelmente o cliente 2 que tem exigência média e não conseguiu atender de forma satisfatória o cliente 1, o mais exigente.

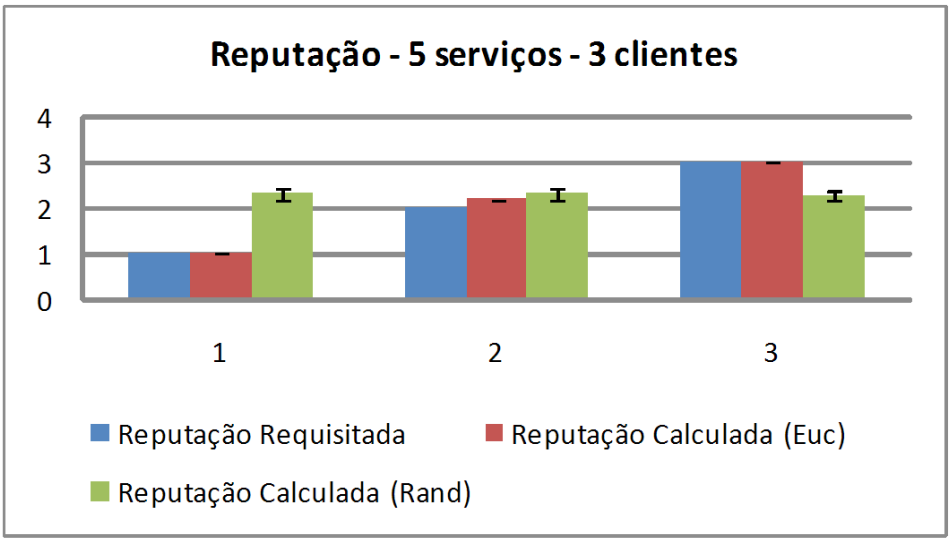

Figura 17: Gráfico de reputação – fluxo com 5 serviços e 3 clientes simultâneos

A figura 18 apresenta o gráfico que contém o custo requisitado pelo cliente e os custos encontrados pelos algoritmos de seleção Euclidiana e aleatório. Para o custo existe uma inversão do cliente mais exigente, sendo que o cliente 1 é o menos exigente em relação ao custo e o cliente 3 é o mais exigente em relação ao custo. O cliente 2, por sua vez, se mantém com exigência mediana. O algoritmo de distância Euclidiana conseguiu atender todos os 3 clientes de maneira satisfatória. O algoritmo de seleção aleatório manteve o mesmo comportamento para os demais atributos de QoS. O atendimento ao cliente menos exigente foi satisfatório, por outro lado, o algoritmo atendeu de maneira razoável o cliente de exigência mediana, e de maneira insatisfatória o cliente mais exigente.

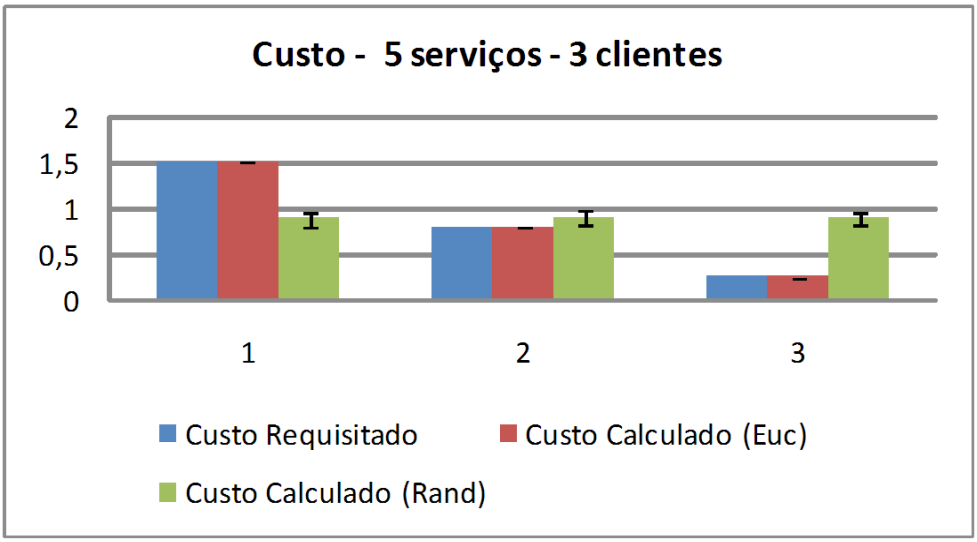

Figura 18: Gráfico de reputação – fluxo com 5 serviços e 3 clientes simultâneos

A figura 19 apresenta a satisfação do usuário calculada por meio da junção dos dados

dos gráficos apresentados nas figuras 16, 17, e 18.

O gráfico da figura 19 apresenta o algoritmo de seleção Euclidiana com melhor satisfação do usuário para os 3 clientes, com 100 % de satisfação para os clientes 1 e 3 e 95% para o cliente 2. Essa penalização na satisfação do cliente para o algoritmo de distância Euclidiana no atendimento à requisição do cliente 2 ocorreu devido ao fato de ele não ter conseguido atender de forma completamente satisfatória aos atributos de tempo de resposta e reputação.

O algoritmo aleatório apresentou a melhor satisfação para o cliente 2. Isso se explica pelo fato do algoritmo aleatório ter um comportamento que escolhe valores medianos para o atendimento das requisições. Por essa razão, o cliente 2 determina para os três atributos de QoS uma exigência mediana e acaba sendo melhor atendido pelo algoritmo aleatório em relação aos cliente 1 e 3 que apresentam grande exigência para os atributos tempo de resposta e custo respectivamente.

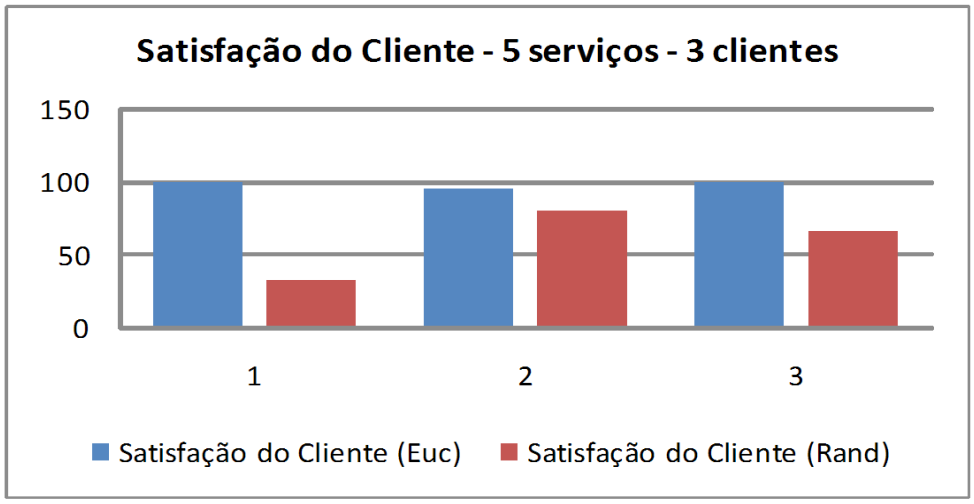

Figura 19: Gráfico de Satisfação do Cliente – fluxo com 5 serviços e 3 clientes simultâneos

7.3.2 Teste com 19 clientes simultâneos e fluxo composto por 5 serviços

Na figura 20 é apresentado o gráfico com o tempo de resposta para o fluxo composto de 5 *Web Services* e com 19 clientes acessando-o simultaneamente. Neste gráfico é possível observar que com a ocorrência de um maior número de clientes acessando simultaneamente o *middleware* DWSC-M há uma sobrecarga considerável na ferramenta, principalmente em relação aos clientes mais exigentes. A sobrecarga é identificada por meio da diferença entre o tempo de resposta obtido e o tempo de resposta calculado pelo algoritmo. Se o tempo de

resposta que foi calculado pelo algoritmo de distância Euclidiana for analisado, observa-se que o algoritmo sempre procura atender de forma adequada à requisição. Isso somente não é possível porque o tempo para o cálculo do fluxo com a menor distância Euclidiana em relação aos atributos de QoS requisitados fica alto quando muitos acessos simultâneos são realizados e quando aumentam as opções de serviços a serem escolhidos para compor o fluxo.

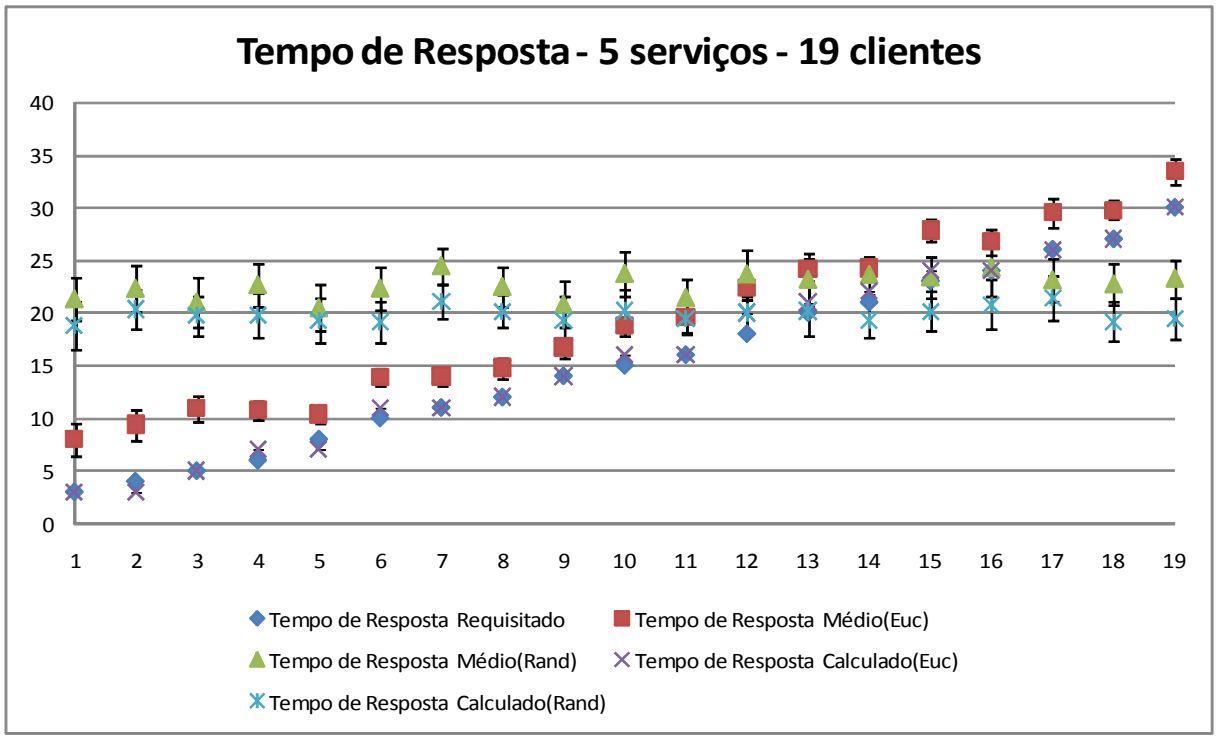

Figura 20: Gráfico de tempo de resposta – fluxo com 5 serviços e 19 clientes simultâneos

Apesar da sobrecarga, o algoritmo de seleção Euclidiana atende na média, de forma razoável as requisições dos clientes, enquanto o algoritmo de seleção aleatório, por ser trivial, e não sobrecarregar o sistema mantém um comportamento semelhante ao teste da subseção 7.3.1.

O gráfico da Figura 21 apresenta a reputação dos serviços selecionados pelos algoritmos de seleção Euclidiana e aleatório. Para este atributo de QoS é possível perceber que o algoritmo de distância Euclidiana é melhor que o algoritmo aleatório, visto que no caso menos exigente, apesar do algoritmo de seleção aleatória ser melhor, o algoritmo de distância Euclidiana ainda satisfaz o cliente. Para o caso de clientes de exigência mediana os dois algoritmos apresentam valores estatisticamente iguais e ambos não conseguem apresentar satisfação total para os clientes. Para clientes mais exigentes o algoritmo de seleção Euclidiana consegue reputação melhor que o algoritmo aleatório, apesar de também não conseguir satisfação total para todos os clientes mais exigentes.

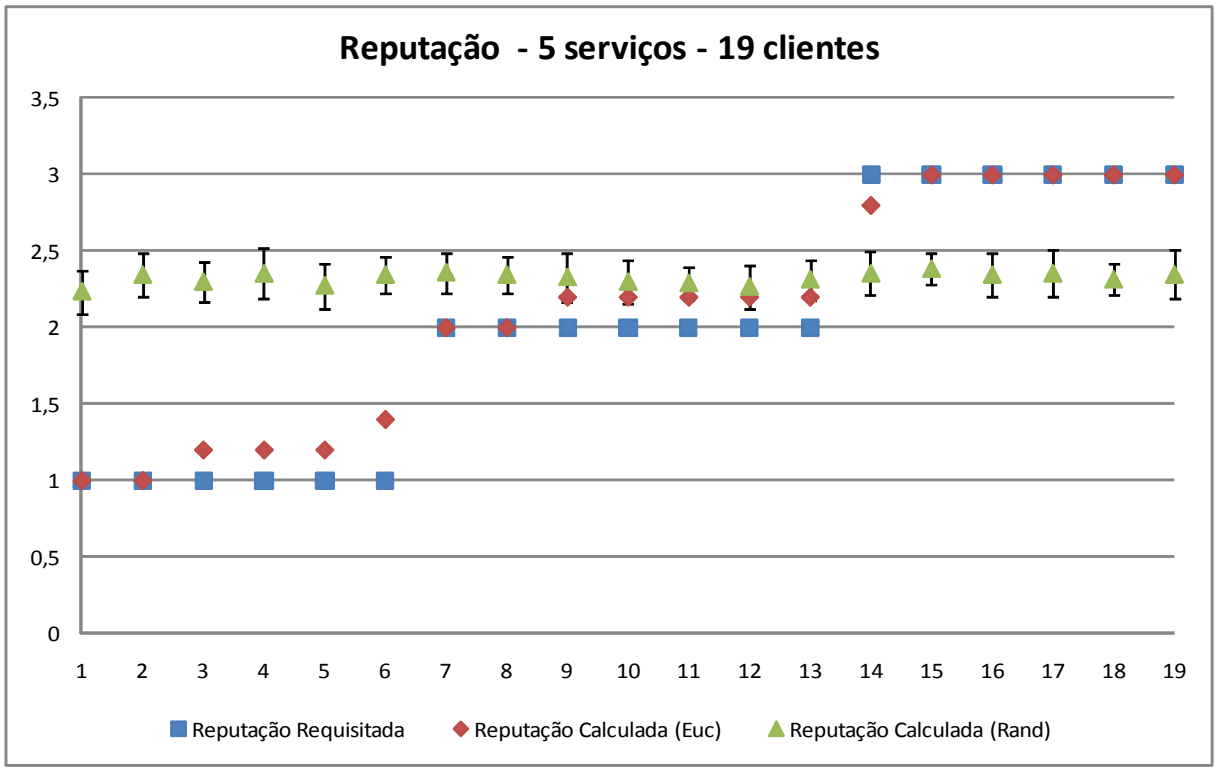

Figura 21: Gráfico de reputação – fluxo com 5 serviços e 19 clientes simultâneos

O gráfico da Figura 22 mostra o custo por acesso ao serviço composto. Neste caso, é possível perceber a inversão de exigência dos clientes em relação aos resultados apresentados nas figuras 20 e 21, como ocorreu no teste da subseção 7.3.1. Novamente o algoritmo de seleção Euclidiana conseguiu melhor satisfação do cliente nos casos mais exigentes, no entanto piorou a satisfação para os clientes menos exigentes. O algoritmo de seleção Euclidiana conseguiu atender apenas alguns dos clientes com pouca exigência de forma satisfatória. Os clientes de 3 a 6 não foram atendidos de forma satisfatória, diferentemente dos clientes de exigência média e alta. Apenas os clientes de maior exigência foram atendidos com satisfação total. No algoritmo de seleção aleatória, com seu padrão de seleção de serviços medianos, foi possível atender de forma satisfatória os clientes menos exigentes e os com exigência média, e de forma pouco satisfatória os clientes que apresentam alta exigência em relação à satisfação.

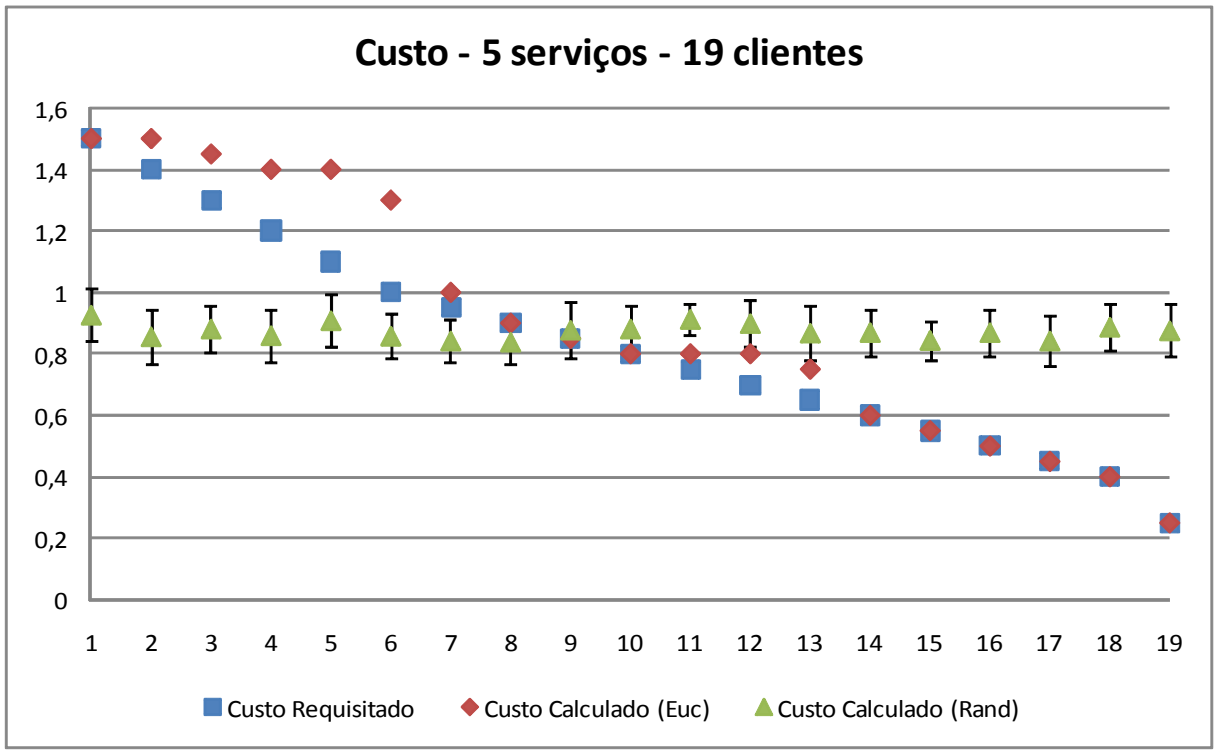

Figura 22: Gráfico de custo – fluxo com 5 serviços e 19 clientes simultâneos

O gráfico apresentado na figura 23 é referente à satisfação do cliente para o teste da subseção 7.3.2. A satisfação do cliente é melhor para os serviços escolhidos pelo algoritmo de seleção Euclidiana, onde há casos em que o algoritmo de seleção aleatória apresenta satisfação proporcional ao algoritmo de seleção Euclidiana. É evidente também que em relação ao teste da subseção 7.3.1 há um desgaste muito grande na satisfação do usuário em relação ao algoritmo de seleção Euclidiana, se comparado o cliente 1 com cliente 1, cliente 2 com cliente 10 e cliente 3 com cliente 19 nos testes das subseções 7.3.1 e 7.3.2 respectivamente que são clientes com as mesmas exigências de QoS. Nota-se que no primeiro caso, há uma perda considerável na satisfação do cliente usando o algoritmo Euclidiano e o algoritmo de seleção aleatória permanece equivalente. No segundo caso, o algoritmo de seleção Euclidiana e aleatório permanecem parecidos, com uma pequena perda em relação ao teste da subseção 7.3.2. No caso 3, ambos os algoritmos permaneceram parecidos com pequena perda em relação ao teste da subseção 7.3.2. A perda de desempenho presente neste teste se explica pelo fato da sobrecarga gerada pelos algoritmos de seleção penalizar os clientes mais exigentes no atributo tempo de resposta. No caso do cliente 1 a sobrecarga média foi de 4,9s. Se ele requisitou tempo de resposta igual a 3 segundos, obteve um tempo de resposta médio de 7,9s, resultando em um tempo de resposta final maior que o dobro do requisitado. Isso faz com que sua satisfação em termos de tempo de resposta seja 0% e não

mais 100% como no teste apresentado na subseção 7.3.1, o que torna significativa a queda na satisfação final do cliente. Por outro lado, no caso do cliente 19 deste teste onde o tempo de resposta requisitado foi igual a 30s e sobrecarga média igual a 3,4s o cliente obteve um tempo de resposta total de 33,4s em média, não atingindo uma satisfação total. Como o atraso é relativamente baixo, o resultado é uma satisfação de aproximadamente 90% para o tempo de resposta e não mais 100% como no teste da subseção 7.3.1. Os demais atributos de QoS por não sofrer influência da execução permanecem iguais, o que não influencia na alteração da satisfação do cliente em testes em que vários clientes acessam simultaneamente o DWSC-M.

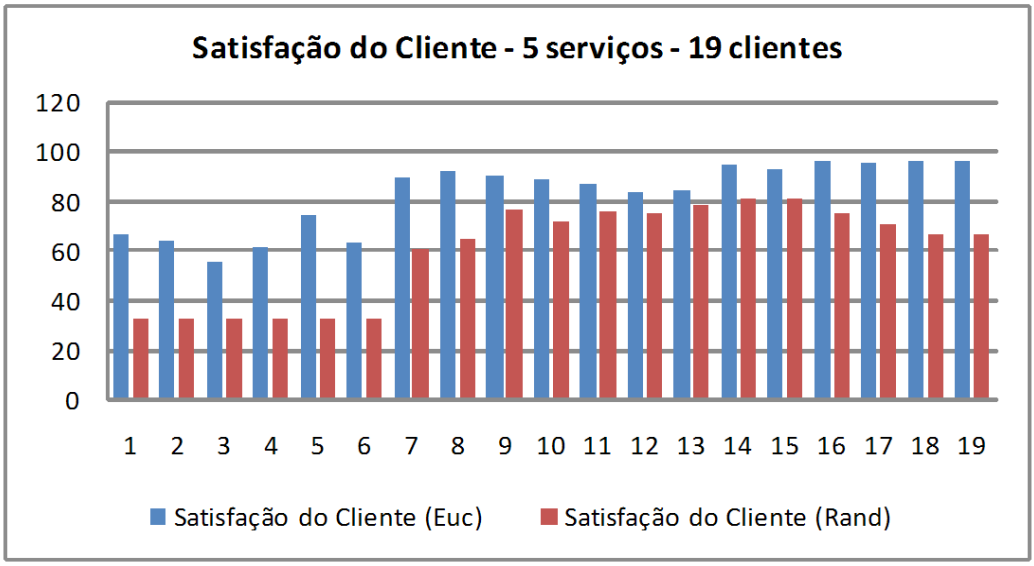

Figura 23: Gráfico de satisfação do cliente – fluxo com 5 serviços e 19 clientes simultâneos

7.3.3 Teste com 38 clientes simultâneos e fluxo composto por 5 serviços

Neste teste os mesmos clientes utilizados no teste apresentado na subseção 7.3.2 foram utilizados, mas replicados para a gerar um total de 38 clientes. A diferença nos resultados deste teste para os demais, com fluxo composto de 5 *Web Services,* foi em relação à sobrecarga no tempo de resposta e a satisfação do cliente.

A figura 24 apresenta os tempos de resposta por cliente para os algoritmos de seleção Euclidiana e de seleção aleatória. Para o algoritmo de seleção Euclidiana é possível observar que os serviços selecionados para atender a requisição deveriam realizar o atendimento de forma satisfatória em relação ao tempo de resposta exigido por cada cliente. No entanto, devido a alta sobrecarga gerada pela quantidade de clientes requisitando acessos ao serviço de forma concorrente, nenhum cliente conseguiu obter o tempo de resposta requisitado. O algoritmo de seleção aleatória, por sua vez, apresentou sobrecarga, no entanto, menor que a

do algoritmo Euclidiano e conseguiu atender em casos em que o atributo tempo de resposta tem exigência menor para determinado cliente.

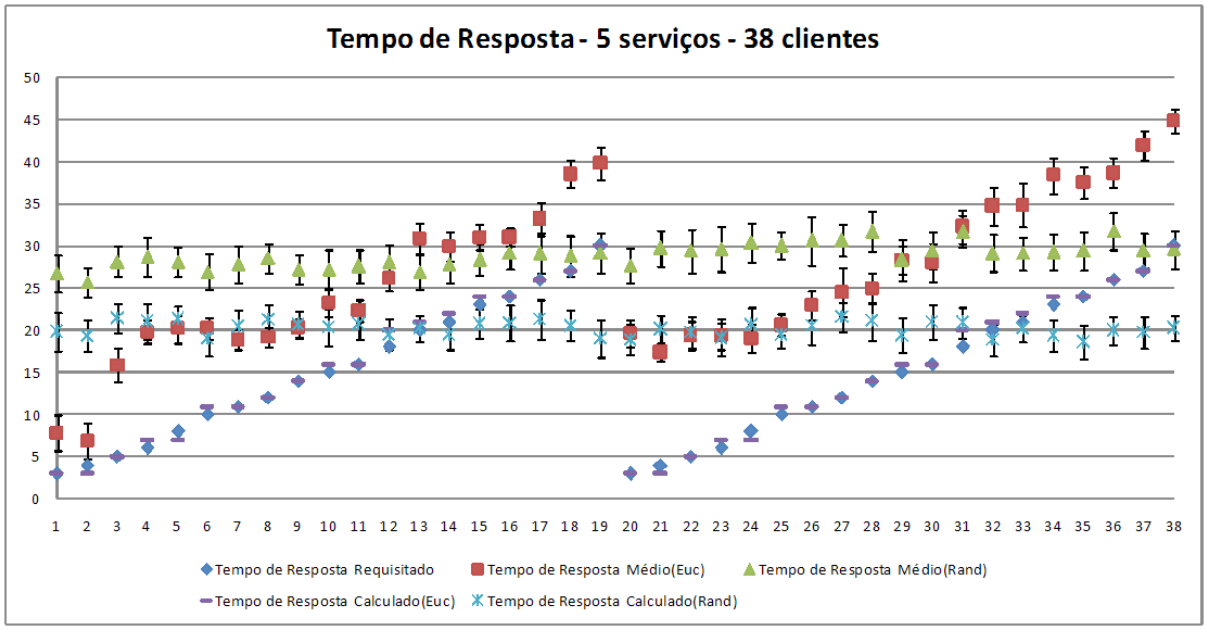

Figura 24: Gráfico de tempo de resposta – fluxo com 5 serviços e 38 clientes simultâneos

Para os atributos de QoS reputação e custo apresentados nas Figuras 25 e 26 respectivamente o comportamento de ambos os algoritmos se mantêm semelhantes aos dos testes já apresentados.

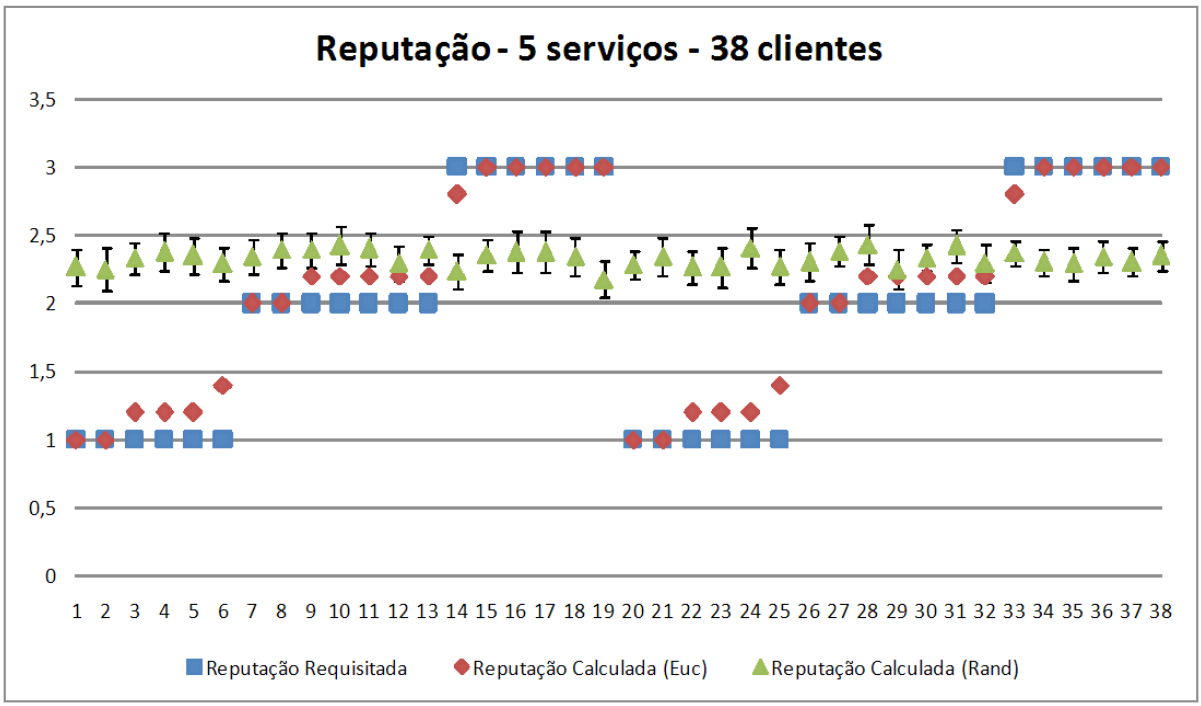

Figura 25: Gráfico de reputação – fluxo com 5 serviços e 38 clientes simultâneos

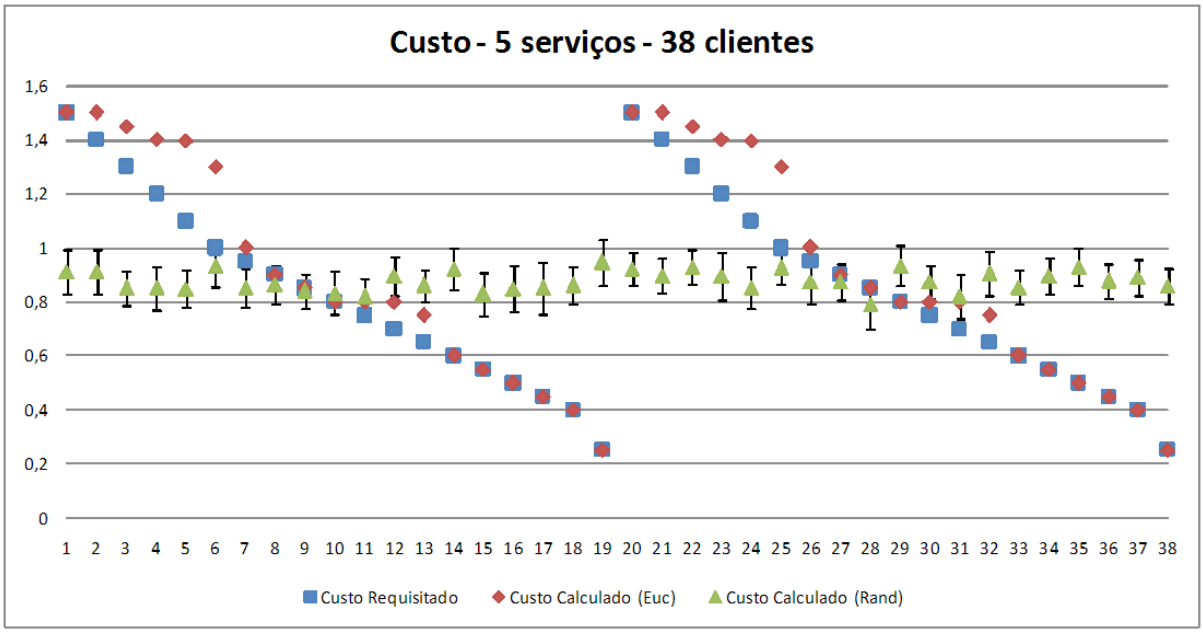

Figura 26: Gráfico de custo – fluxo com 5 serviços e 38 clientes simultâneos

A figura 27 apresenta a satisfação do usuário. Neste gráfico é possível observar ainda uma melhor satisfação do usuário de maneira geral para usuários do algoritmo de seleção Euclidiana. Apesar de o tempo de resposta gerar satisfação ruim ao utilizar este algoritmo, ele ainda consegue boa satisfação para o cliente ao contabilizar os demais atributos de QoS pelo fato de eles não sofrerem influência do tempo de execução. O algoritmo de seleção aleatório manteve a satisfação do cliente semelhante ao do teste apresentado na subseção 7.3.2, já que a sobrecarga causada por ele é baixa.

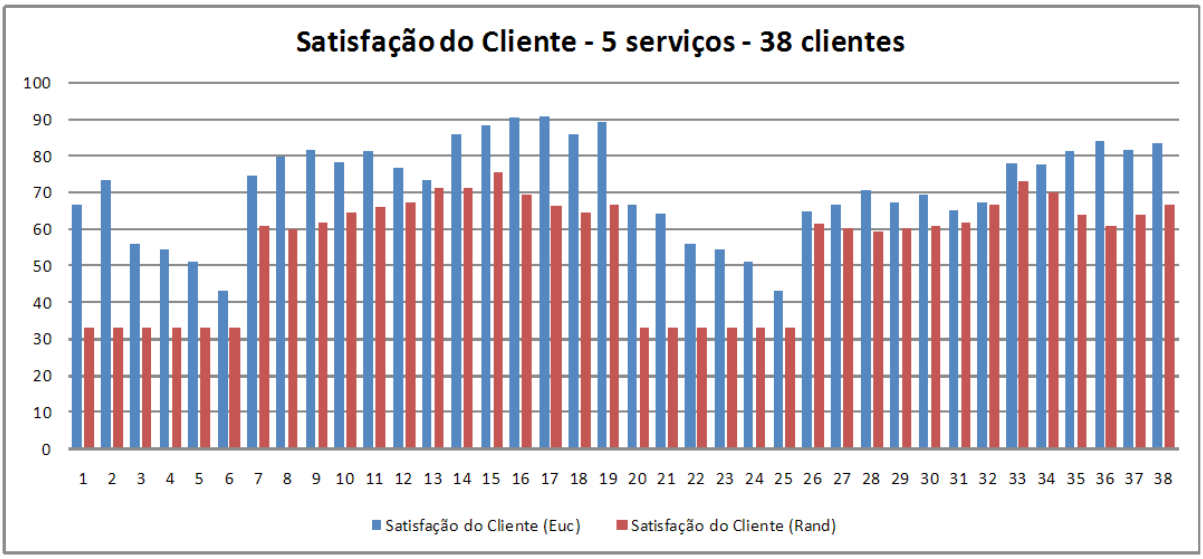

Figura 27: Gráfico de satisfação do cliente – fluxo com 5 serviços e 38 clientes simultâneos

7.3.4 Teste com 3 clientes simultâneos e fluxo composto por 8 serviços

O gráfico apresentado na figura 28 é referente aos valores de tempo de resposta do teste com 3 clientes simultâneos e fluxo composto de 8 serviços. No gráfico é possível observar a ocorrência de atraso na execução devido à sobrecarga no algoritmo de seleção Euclidiana com poucos clientes simultâneos. O algoritmo consegue calcular o fluxo que deveria atender a requisição para os 3 clientes, no entanto as três requisições têm um atraso médio de quase 1s. Isso ocorreu pelo fato deste fluxo ser maior, o que torna mais complexa a tarefa de encontrar o fluxo com menor distância Euclidiana. O algoritmo de seleção aleatória, por ser um algoritmo trivial, mantém o mesmo comportamento em relação aos outros testes, uma vez que sua complexidade não muda com o aumento de serviços no fluxo.

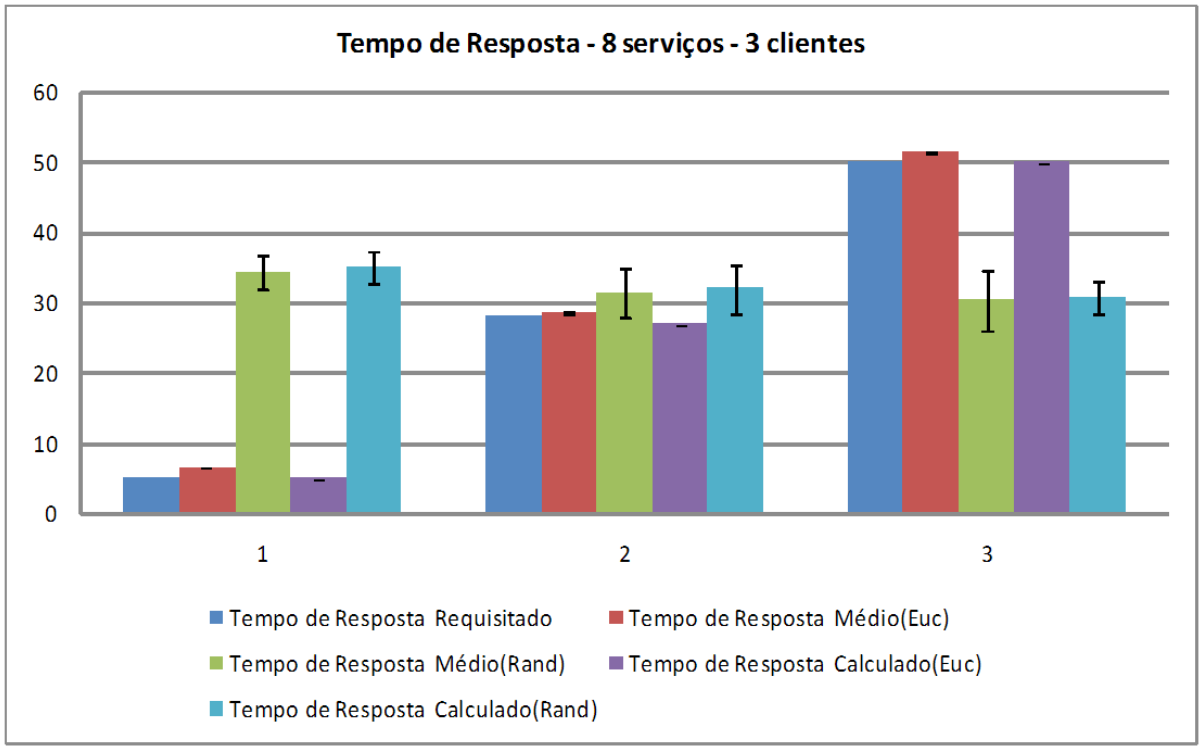

Figura 28: Gráfico de tempo de resposta – fluxo com 8 serviços e 3 clientes simultâneos

A figura 29 apresenta o gráfico de reputação requisitada pelo cliente que foi calculada utilizando os dois algoritmos de seleção discutidos neste projeto de mestrado. Neste caso, é possível observar que o algoritmo de distância Euclidiana consegue melhorar o tempo de resposta para os clientes 1 e 2. O cliente 3 obteve melhor tempo de resposta com o algoritmo de seleção aleatória, no entanto, na média o algoritmo de seleção Euclidiana conseguiu melhor atendimento que o de seleção aleatória.

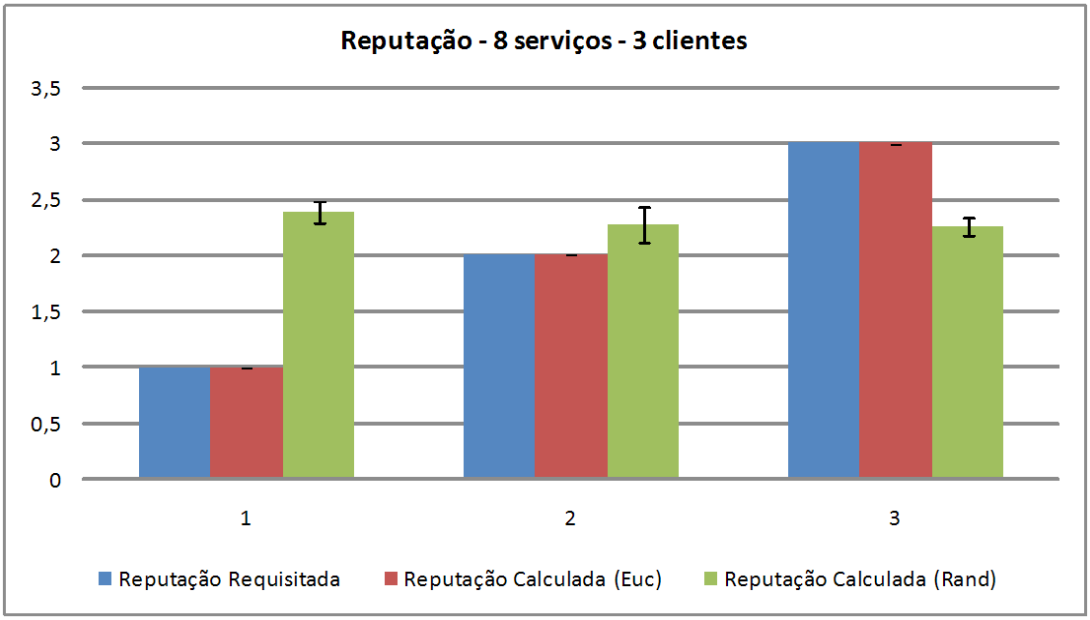

Figura 29: Gráfico de reputação – fluxo com 8 serviços e 3 clientes simultâneos

A figura 30 apresenta o gráfico que contém o custo requisitado pelo cliente e os custos encontrados pelos algoritmos de seleção Euclidiana e aleatório. O algoritmo de distância Euclidiana conseguiu atender todos os 3 clientes de maneira satisfatória, não conseguindo 100% de satisfação apenas para o cliente 2. O algoritmo de seleção aleatória manteve o mesmo comportamento que teve para os demais atributos de QoS. O uso do algoritmo aleatório sinalizou com bom atendimento aos clientes menos exigentes, um atendimento razoável para o cliente de exigência mediana e um atendimento insatisfatório para o cliente mais exigente.

No gráfico da figura 31 o algoritmo de distância Euclidiana contribuiu para penalizar a satisfação do cliente devido à sobrecarga gerada pelo o cálculo do fluxo, mas ainda obteve melhor satisfação que o algoritmo de seleção aleatória (ao menos 90% de satisfação total para os 3 clientes). O algoritmo de seleção aleatória não teve seu comportamento alterado se comparado com os demais testes do experimento.

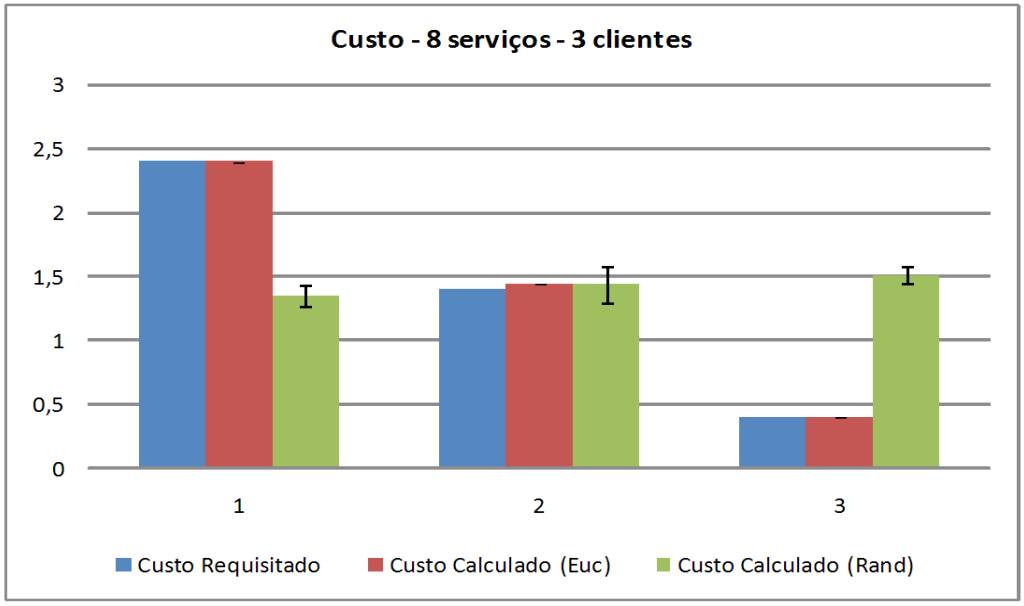

Figura 30: Gráfico de custo – fluxo com 8 serviços e 3 clientes simultâneos

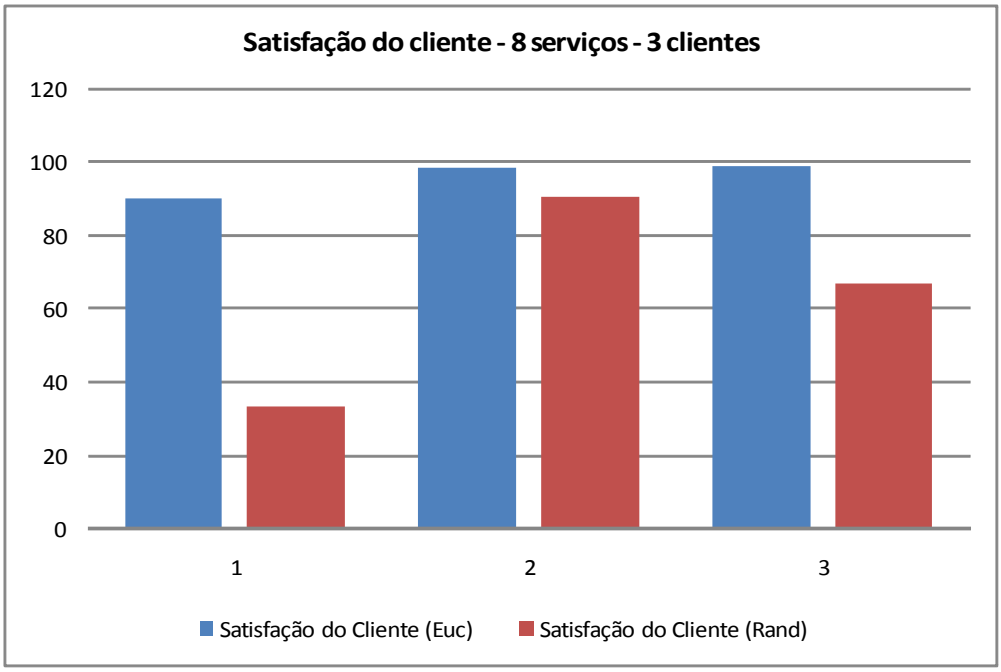

Figura 31: Gráfico de satisfação do cliente – fluxo com 8 serviços e 3 clientes simultâneos

7.3.5 Teste com 19 clientes simultâneos e fluxo composto por 8 serviços

O gráfico apresentado na figura 32 é referente ao tempo de resposta do fluxo composto de 8 *Web Services* e 19 clientes simultâneos. Neste gráfico fica visível a sobrecarga gerada pelo algoritmo de seleção Euclidiana, onde o tempo de resposta que foi calculado pelo algoritmo deveria atender de forma bastante satisfatória a requisição do cliente. Devido à

sobrecarga gerada, o algoritmo atende com muito atraso em relação ao tempo de resposta requisitado. No entanto, o algoritmo de seleção Euclidiana atende de forma melhor que o algoritmo de seleção aleatória os clientes mais exigentes em relação a tempo de resposta. No caso de clientes menos exigentes o algoritmo de seleção Euclidiana é bastante ruim em relação ao algoritmo de seleção aleatória que consegue 100% de satisfação nos casos dos clientes 14 a 19.

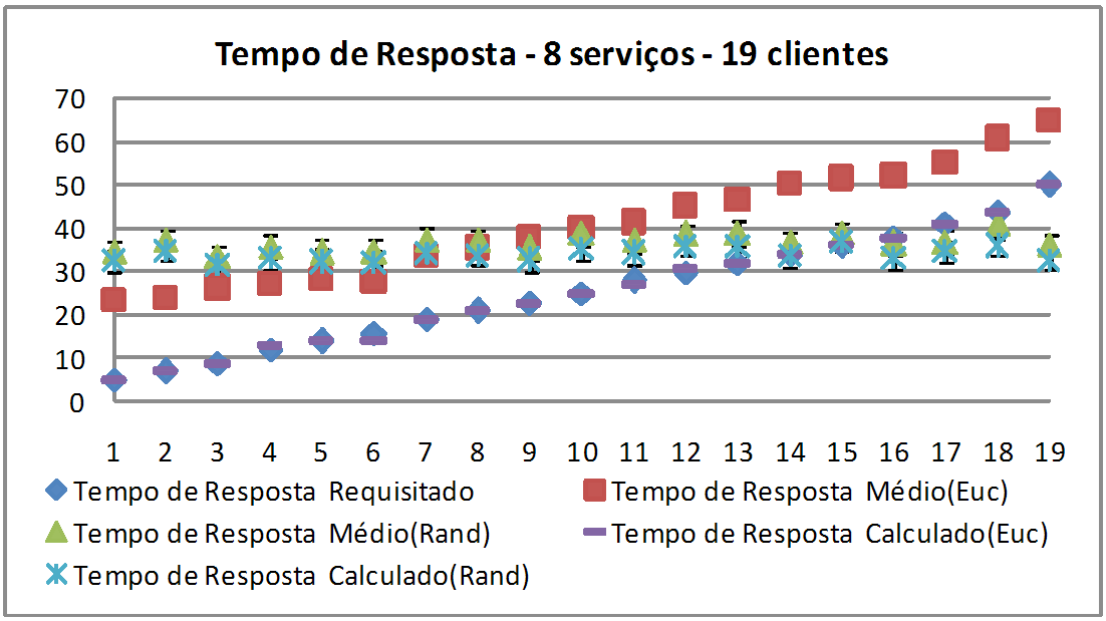

Figura 32: Gráfico de tempo de resposta – fluxo com 8 serviços e 19 clientes simultâneos

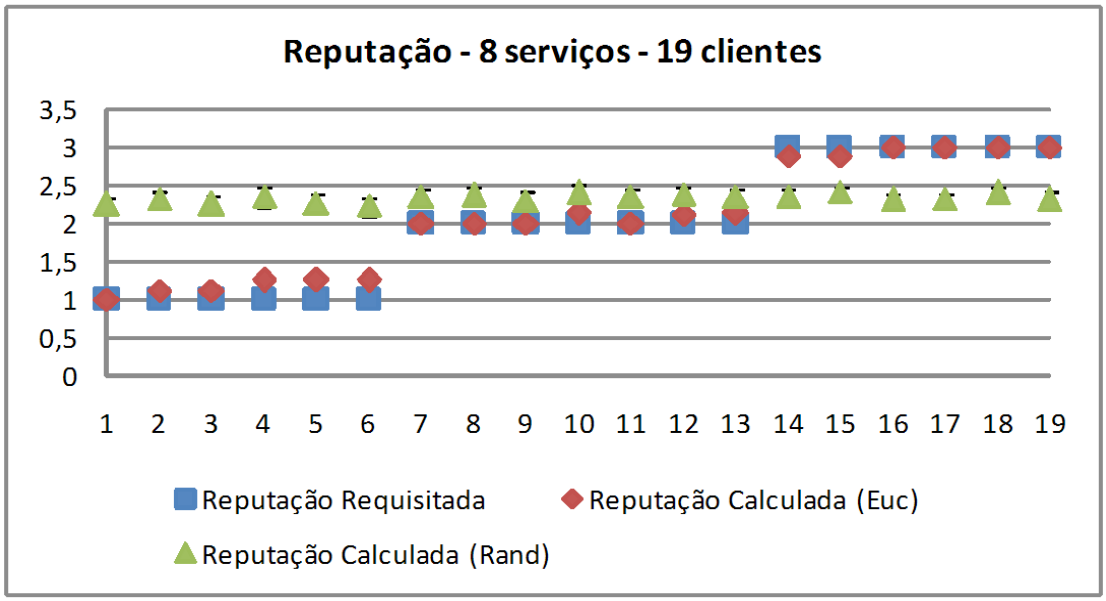

Figura 33: Gráfico de reputação – fluxo com 8 serviços e 19 clientes simultâneos

Para os atributos de QoS reputação e custo apresentados nas figuras 33 e 34 respectivamente é mantido o comportamento de ambos os algoritmos semelhantes ao dos testes já apresentados.

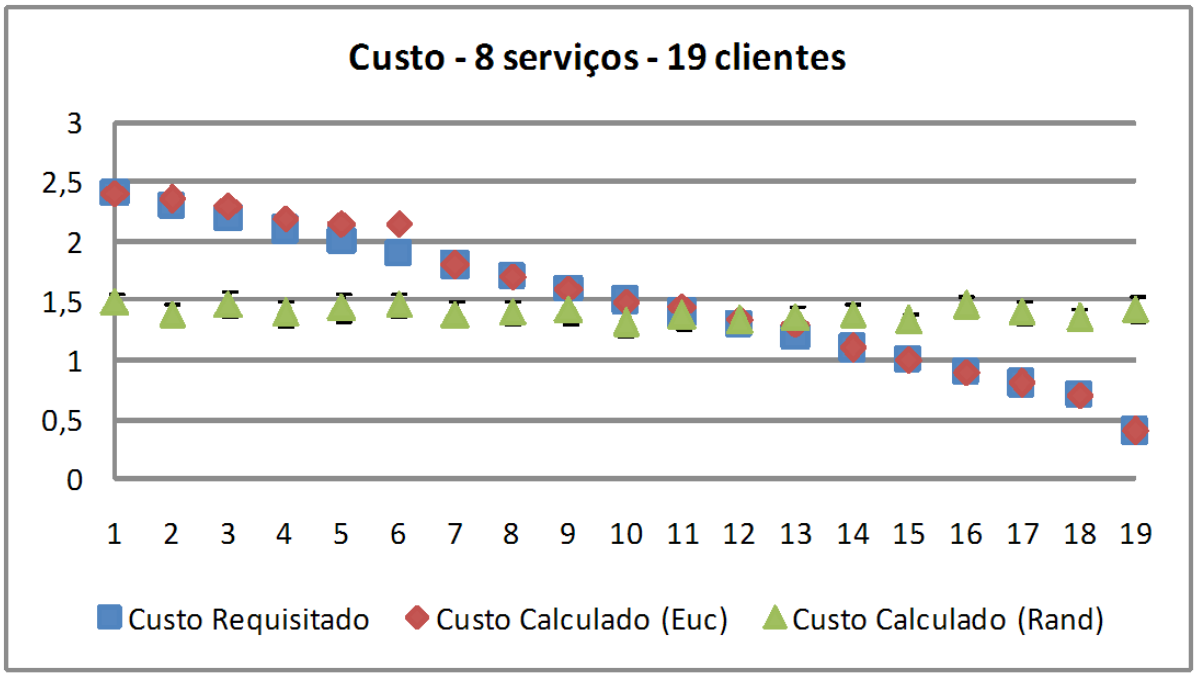

Figura 34: Gráfico de custo – fluxo com 8 serviços e 19 clientes simultâneos

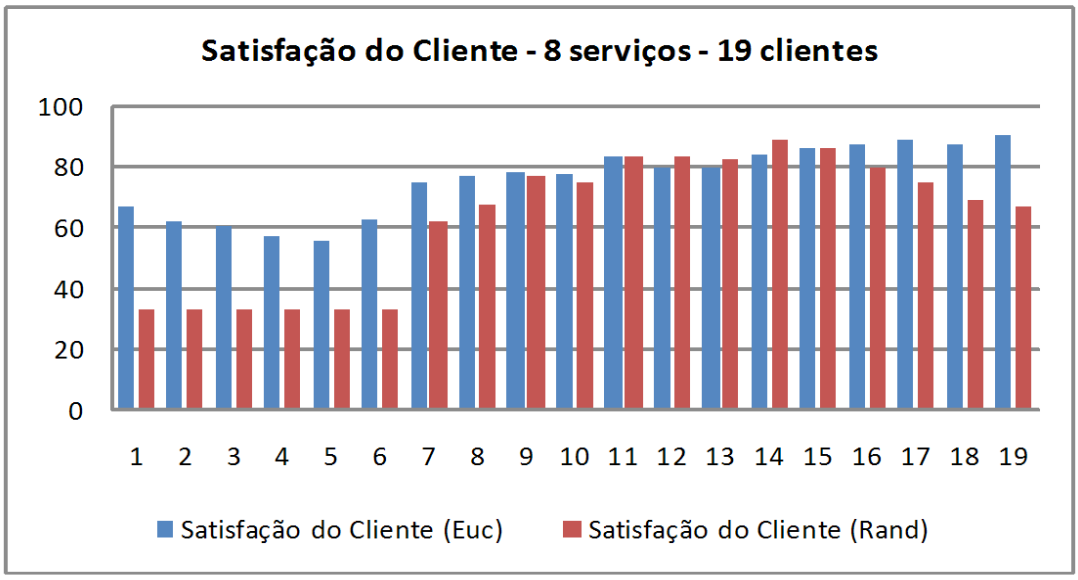

Figura 35: Gráfico de satisfação do cliente – fluxo com 8 serviços e 19 clientes simultâneos

O gráfico da figura 35 apresenta os primeiros clientes que obtiveram satisfação maior utilizando o algoritmo de seleção aleatória. No entanto, na média a satisfação gerada pelo algoritmo de seleção Euclidiana ainda foi maior que a satisfação gerada pelo algoritmo de seleção aleatória. Neste teste, ocorreu uma satisfação maior para alguns clientes atendidos pelo algoritmo de seleção aleatória pelo fato de o algoritmo de seleção Euclidiana ter piorado.

Isso se deve à carga de trabalho submetida ao DWSC-M ser grande o suficiente para aumentar o tempo de resposta médio quando um grande número de serviços acessam a ferramenta simultaneamente.

7.3.6 Teste com 38 clientes simultâneos e fluxo composto por 8 serviços

Neste teste é possível observar no tempo de resposta apresentado pela figura 36, que a sobrecarga gerada pelo algoritmo de seleção Euclidiana é tão alta que nos melhores casos o tempo de resposta fica igual ao tempo de resposta do algoritmo de seleção aleatório. Neste teste até o algoritmo de seleção aleatória apresenta grande sobrecarga, embora em alguns casos onde a exigência em tempo de resposta do cliente é muito baixa, o algoritmo de seleção aleatória consegue atendê-la com 100% de satisfação.

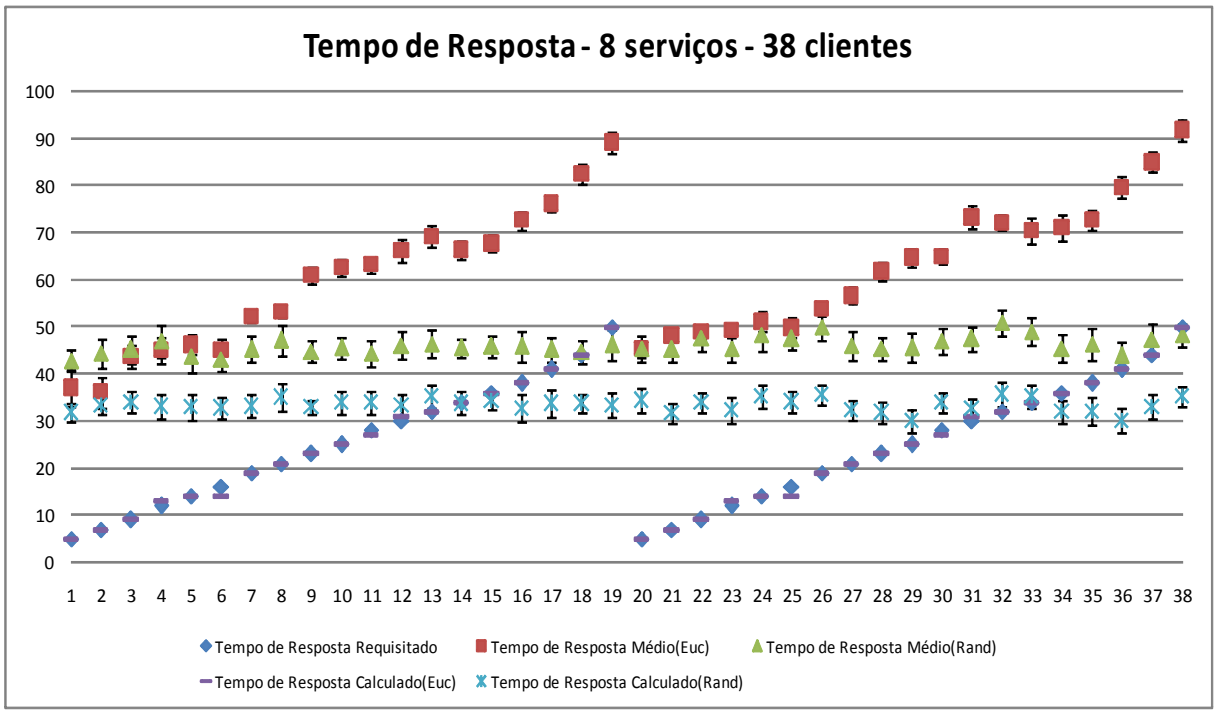

Figura 36: Gráfico de tempo de resposta – fluxo com 8 serviços e 38 clientes simultâneos

Para os atributos de QoS reputação e custo não houve alterações sobre o comportamento de seleção quando aumentado o número de clientes acessando simultaneamente o DWSC-M. Os valores para reputação e custo estão apresentados nas figuras 37 e 38 respectivamente.

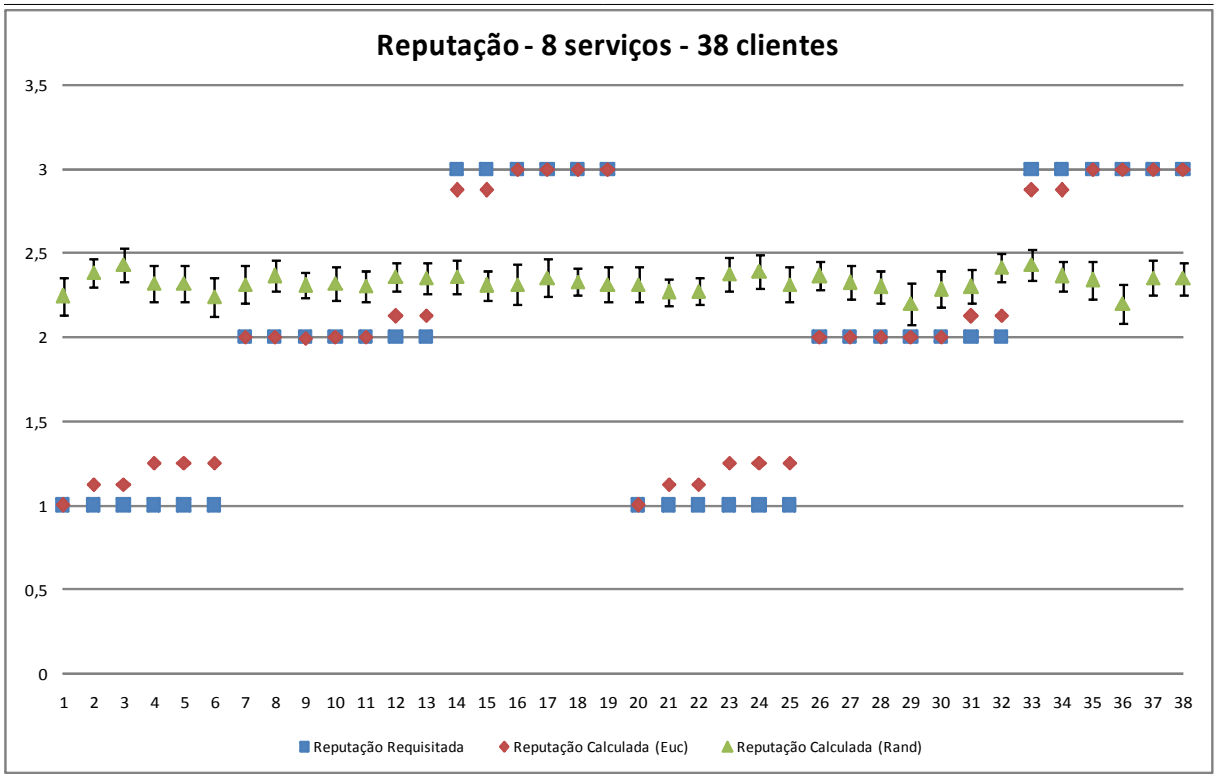

Figura 37: Gráfico de reputação – fluxo com 8 serviços e 38 clientes simultâneos

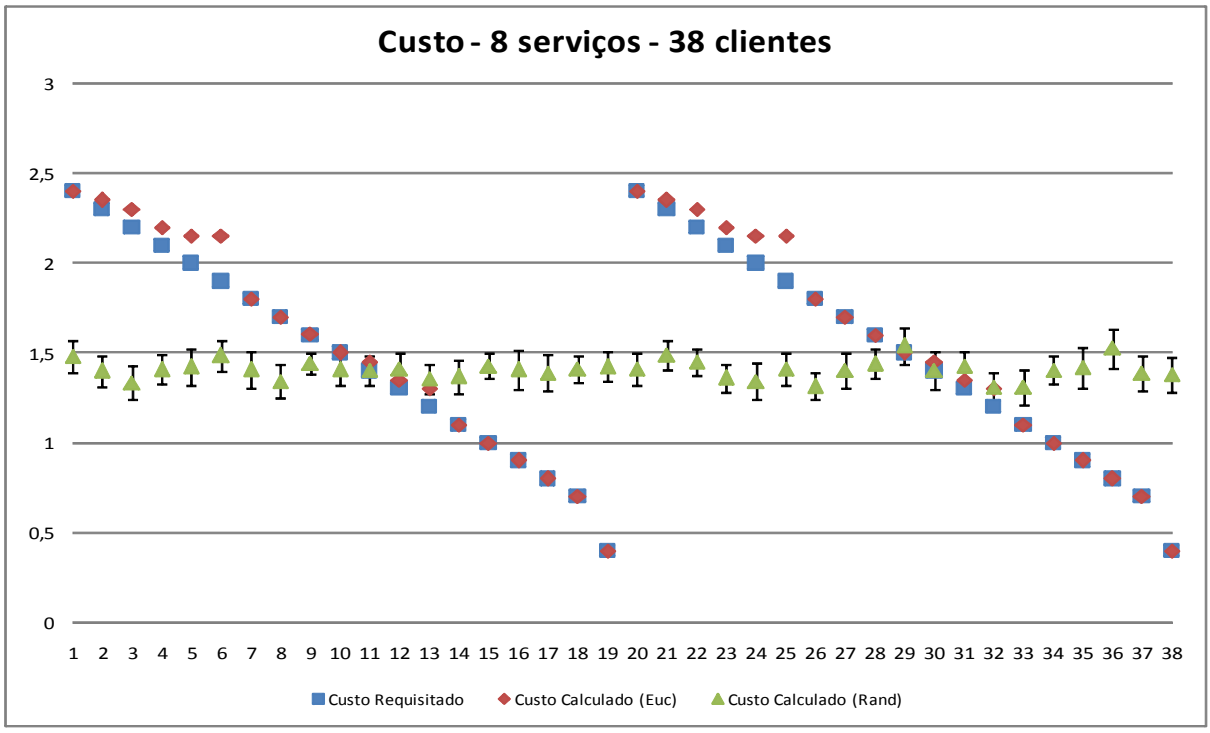

Figura 38: Gráfico de custo – fluxo com 8 serviços e 38 clientes simultâneos

Considerando a figura 39, a satisfação do cliente com o algoritmo de seleção Euclidiana caiu consideravelmente, mantendo-se por volta de 60%, enquanto a satisfação conseguida pelo algoritmo de seleção aleatória se manteve constante em relação aos outros

testes. Isso ocorre porque o algoritmo de seleção aleatório gera pouca sobrecarga na composição de serviços. O atributo de QoS tempo de resposta foi o responsável por essa mudança na satisfação do cliente no algoritmo de seleção Euclidiana. O fato ocorreu devido à sobrecarga que ele gera na composição na situação em que muitos clientes fazem acessos simultâneos ao serviço composto, considerando aspectos de qualidade de serviço.

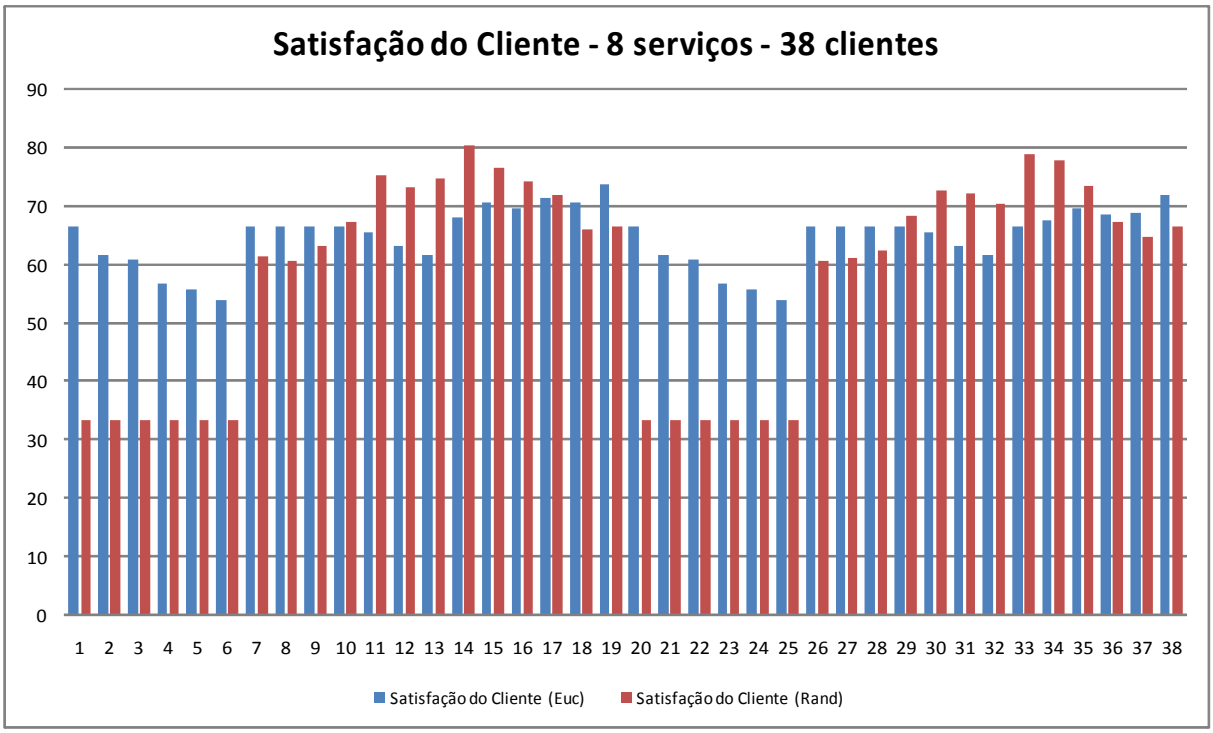

Figura 39: Gráfico de satisfação do cliente – fluxo com 8 serviços e 38 clientes simultâneos

7.4 Estudo da Influência dos Fatores na Satisfação do Cliente

Nesta seção é apresentado o estudo da influência dos fatores na satisfação total do cliente através do planejamento do experimento apresentado na seção 7.2. As tabelas 5, 6, 7 e 8 contêm em cada linha a influência de cada fator ou a interação entre eles. Os fatores representados por A, B e C são caracterizados como:

- **A:** Clientes que realizam acessos simultâneos ao ambiente de experimentação.
- **B:** Algoritmo de Seleção utilizado no experimento.
- **C:** Quantidade de *Web Services* que compõem o fluxo de serviços.

As colunas da tabela 6, 7, 8 e 9 contêm os valores das influências na satisfação do cliente em porcentagem, situação em que os níveis do fator A são isolados (Clientes Simultâneos) resultando em três combinações de dois níveis em cada análise.

Tabela 6: Influência dos Fatores na Satisfação do Usuário

Satisfação	03 e 19 Clientes Simultâneos	03 e 38 Clientes Simultâneos	19 e 38 Clientes Simultâneos
A (Clientes Simultâneos)	9,24%	34,47%	31,19%
B (Algoritmo de Seleção)	79,71%	46,68%	55,73%
C (Serviços por Fluxo)	0,12%	0,07%	0,05%
AB	8,84%	18,14%	7,48%
AC	0,18%	0,04%	1,23%
BC	1,80%	0,52%	3,35%
ABC	0,08%	0,06%	0,93%

Na tabela 6 é possível observar no caso com 03 e 19 clientes que o fator B (algoritmo de seleção) tem grande influência no experimento. Isso se deve ao fato do algoritmo de distância Euclidiana conseguir satisfação do usuário elevada em relação ao algoritmo de seleção aleatória. O fator A (clientes simultâneos) também tem influência considerável no experimento, tanto isoladamente, como combinado com o fator B. Tal influência ocorre pelo fato da execução com 19 clientes simultâneos gerar atraso considerável no tempo de resposta médio em relação a 3 clientes simultâneos.

No experimento com 03 e 38 clientes simultâneos é possível notar que o fator A passa a ser de grande influência no experimento. Isso ocorre porque o atendimento a 38 clientes simultâneos gera um alto atraso em relação ao atendimento a 3 clientes simultâneos. Nesse caso o algoritmo de seleção ainda continua apresentar maior influência na satisfação do cliente, uma vez que a satisfação do cliente é determinada pelo tempo de resposta, custo e reputação. Os dois últimos atributos não são afetados pela sobrecarga no sistema, gerada pela excessiva quantidade de requisições simultâneas.

No experimento com 19 e 38 clientes o comportamento na influência dos fatores permaneceu semelhante ao do experimento com 03 e 38 clientes. O fator C não apresentou influência significativa nos experimentos.

7.4.1 Estudo da Influência dos fatores na satisfação do cliente considerando Tempo de Resposta

Nesta subseção é apresenta a influência dos fatores em relação à satisfação do cliente, isolando o atributo tempo de resposta, para efeito de análise.

Tabela 7: Influência dos Fatores na Satisfação do Usuário (Tempo de Resposta)

Satisfação (Tempo de Resposta)	03 e 19 Clientes Simultâneos	03 e 38 Clientes Simultâneos	19 e 38 Clientes Simultâneos
A (Clientes Simultâneos)	48,63%	73,89%	52,54%
B (Algoritmo de Seleção)	20,24%	1,90%	5,67%
C (Serviços por Fluxo)	4,06%	1,51%	15,66%
AB	16,53%	18,36%	8,42%
AC	2,04%	0,70%	0,03%
BC	8,00%	3,42%	17,65%
ABC	0,47%	0,19%	0,01%

A tabela 7 apresenta a influência dos fatores e suas interações na satisfação do cliente em relação ao tempo de resposta.

Nos experimentos apresentados nesta tabela é possível notar que de forma geral o fator A é o que influencia na satisfação do tempo de resposta de forma mais significativa. Isso se explica, pois, variar a quantidade de cliente gera atraso na resposta, o que acarreta na variação da satisfação do cliente para o tempo de resposta.

No experimento com 3 e 19 clientes simultâneos foi possível observar que o algoritmo de seleção apresentou influência significativa. Isso se justifica, pois, embora tenha havido aumento no número de clientes simultâneos (3 para 19), o algoritmo de seleção Euclidiano foi satisfatório em tentar satisfazer os clientes. Por outro lado, o algoritmo de seleção aleatória não leva em consideração o atendimento da exigência do cliente. Para os demais cenários, como a variação do atraso foi muito alta, o fator B passou a apresentar influência baixa quando isolado, e ser mais significativo se combinado com o fator A.

O fator C apresentou-se de forma significativa nos experimentos. No algoritmo de seleção Euclidiana quando há sobrecarga no sistema, ele contribui para o atraso no tempo de resposta. Este fato ocorre porque o número de serviço aumenta a complexidade para determinar o fluxo a ser executado, fazendo com que o tempo de resposta do algoritmo de seleção também seja elevado.

7.4.2 Estudo da Influência dos fatores na satisfação do cliente considerando Reputação

Nesta subseção é apresenta a influência dos fatores em relação à satisfação do cliente com o atributo de QoS reputação isolado.

Tabela 8: Influência dos Fatores na Satisfação do Usuário (Reputação)

Satisfação (Reputação)	03 e 19 Clientes Simultâneos	03 e 38 Clientes Simultâneos	19 e 38 Clientes Simultâneos
A (Clientes Simultâneos)	0,55%	0,53%	0,00%
B (Algoritmo de Seleção)	98,38%	98,46%	99,79%
C (Serviços por Fluxo)	0,00%	0,22%	0,07%
AB	0,73%	0,66%	0,00%
AC	0,05%	0,00%	0,02%
BC	0,15%	0,09%	0,08%
ABC	0,00%	0,01%	0,01%

Na tabela 8 é apresentada a satisfação dos clientes em relação ao atributo de QoS reputação. Nesta tabela fica visível que o fator A foi o que mais influenciou nesse experimento, isso porque o algoritmo de seleção é o responsável pela satisfação do cliente. Como explicado anteriormente, o algoritmo de seleção Euclidiana sempre procura satisfazer o cliente, enquanto o algoritmo de seleção aleatória não possui essa característica em particular.

7.4.3 Estudo da Influência dos fatores na satisfação do cliente considerando Custo

Nesta subseção é apresentada a influência dos fatores em relação à satisfação do cliente, isolando para tal finalidade o atributo de QoS denominado custo..

Tabela 9: Influência dos Fatores na Satisfação do Usuário (Custo)

Satisfação (Custo)	03 e 19 Clientes Simultâneos	03 e 38 Clientes Simultâneos	19 e 38 Clientes Simultâneos
A (Clientes Simultâneos)	1,16%	1,04%	0,00%
B (Algoritmo de Seleção)	89,77%	90,67%	91,81%
C (Serviços por Fluxo)	1,56%	1,33%	7,67%
AB	6,24%	5,94%	0,00%
AC	0,76%	0,61%	0,02%
BC	0,46%	0,35%	0,47%
ABC	0,00%	0,03%	0,01%

Na tabela 9 é apresentada a satisfação dos clientes em relação ao custo. Neste experimento o comportamento da influência é bastante semelhante ao mostrado na subseção 7.4.2. As diferenças que ocorrem são que a interação dos fatores A e B foram levemente significantes nos experimento com 3 e 19; e 3 e 38 clientes simultâneos. Isso pode ter ocorrido pelo fato de haver grande diversidade de requisições exigindo atributos de qualidade de serviço, fazendo com que houvesse variação na média da satisfação do cliente. Nem todos os clientes conseguiram apresentar a mesma satisfação, ocasião que pode ter ocorrido no experimento com 19 e 38 clientes, no entanto, afetando o fator C.

7.4.4 Estudo da influência da quantidade clientes simultâneos na satisfação do cliente

Nesta subseção foi realizado o replanejamento do experimento com os dados já apresentados anteriormente, cujo objetivo é a verificação de apenas dois fatores: A (Algoritmo de Seleção) e B (Tamanho do Fluxo). Os níveis de cada fator estão relacionados na tabela 10. Com o replanejamento do experimento é possível observar a influência que o algoritmo de seleção e a quantidade de serviços por fluxo determina na satisfação do cliente.

Tabela 10: Fatores e Níveis relacionados ao Planejamento de Experimentos

Fatores	**Níveis**
Algoritmo de Seleção	• Distância Euclidiana • *Random*
Tamanho do Fluxo	• 5 serviços • 8 serviços

Na tabela 11 é possível observar que para esta análise de experimento o algoritmo de seleção é o fator principal que influencia na satisfação do cliente, e que o fator B passa a influenciar mais, na medida em que aumenta a quantidade de clientes realizando acessos simultâneos.

Tabela 11: Influência dos Fatores na Satisfação do Usuário (Custo)

Satisfação	**03 Clientes Simultâneos**	**19 Clientes Simultâneos**	**38 Clientes Simultâneos**
A (Algoritmo de Seleção)	99,20%	91,50%	83,43%
B (Serviços por Fluxo)	0,00%	1,61%	2,02%
AB	0,78%	6,88%	14,53%

7.5 Considerações Finais

Este capítulo apresentou de forma detalhada o ambiente de experimentos utilizado para os testes de dois algoritmos de seleção de *Web Services* implantados no *middleware* DWSC-M. Os resultados obtidos com a execução dos experimentos e a discussão desses também foram apresentados. Também foi objeto de estudo a discussão sobre as características de cada algoritmo e como eles se comportaram em diversas situações, sejam elas em relação ao aumento do número de cliente acessando o middleware e também em relação à quantidade de serviços componentes de um determinado fluxo.

Foi apresentado detalhadamente o planejamento dos experimentos e discutido o estudo da influência dos fatores na satisfação do cliente de um modo geral: a satisfação do cliente por tempo de resposta, por reputação e por custo. Destaque para o estudo da influência da quantidade de clientes simultâneos na satisfação do cliente. Outros estudos de como os fatores influenciam nas variáveis de respostas podem ser analisados. Para o processo de avaliação de desempenho do *middleware*, foi considerada como técnica a *aferição* através da construção de um protótipo com posterior coleta de dados. O planejamento de experimento foi do tipo Fatorial Completo em que foram considerados 3 fatores com no máximo 3 níveis. No entanto, para efeito de análise dos resultados, os níveis foram agrupados de forma pareada. Os fatores analisados foram os algoritmos de seleção de serviços, a quantidade de clientes simultâneos e a quantidade de serviço por fluxo.

Em diversas situações da análise dos resultados, pode ser verificado como os fatores foram combinados para identificar a influência de cada um na variável de resposta analisada. A combinação desses fatores é de suma importância no processo de avaliação dos dados resultantes da avaliação de desempenho, e não seria possível afirmar somente pela análise dos gráficos que determinado algoritmo é bom ou ruim ou se o tempo de resposta e a reputação são ou não satisfatórios. Na realidade, a análise comportamental dos resultados obtidos serviu de base para que uma análise mais criteriosa fosse efetuada.

Em todas as análises realizadas, sempre foram considerados até três fatores e no máximo três níveis. Essa escolha se deve à facilidade de interpretação dos resultados quando estes são analisados de forma pareada. Certamente, uma outra análise envolvendo N fatores e N níveis também poderia ser feita. No entanto, um cuidado maior deveria ser tomado ao escolher a técnica de análise e os métodos adequados a serem utilizados. Essa possibilidade

não está descartada na próxima etapa para a continuidade deste trabalho, em que devem ser analisados outros fatores e outros níveis com uso de por exemplo Análise de Variância com Múltiplos Fatores (ANOVA N-WAY) e técnicas de regressão. Por se tratar de uma análise mais completa que a mostrada neste trabalho e também por esta não ser o escopo do trabalho, decidiu-se que ela análise deva ser feita em trabalhos futuros que englobe o uso do *middleware* DWSC-M e a arquitetura orientada a serviço da qual o *middleware* funciona como um módulo.

O próximo capítulo irá apresentar as conclusões obtidas com o desenvolvimento deste trabalho, as contribuições obtidas e trabalhos futuros que podem ser considerados para melhorar o *middleware* de composição de serviços.

Capítulo

8

Conclusões e Trabalhos Futuros

8.1 Conclusões

O objetivo deste projeto foi o desenvolvimento e implementação de um *middleware* de composição de *Web Services* considerando aspectos de qualidade de serviço e a proposição de dois algoritmos para testar o *middleware*. Para alcançar esse objetivo foi preciso lidar com a questão da natureza estática das ferramentas disponíveis para criação de composição de *Web Services*. Isso faz com que os serviços integrantes de um fluxo de serviços sejam sempre aqueles escolhidos durante a elaboração do serviço composto, tornando impraticável o fornecimento de QoS, o que inviabiliza a opção de utilizar serviços alternativos com diferentes fornecimentos de QoS.

Para resolver esse problema foi proposto o *middleware* DWSC-M que é composto por dois módulos. Um módulo responsável pela execução de algoritmos para seleção de *Web Services* e outro para realizar o acesso dinâmico aos serviços que melhor atendem às requisições feitas pelos clientes. O protótipo do DWSC-M possibilitou a criação de composição de serviços de forma dinâmica, e foi avaliado conforme mostrado no capítulo 7 desta dissertação, apresentando desempenho satisfatório para os testes realizados. A forma de desenvolvimento da ferramenta considerou a possibilidade de funcionamento com qualquer *engine* de composição de *Web Services*. Isso foi alcançado pelo fato do DWSC-M ter sido desenvolvido como um *Web Service,* o que viabiliza o acessado por qualquer *engine* que considere composição de *Web Service.*

Outro desafio para atingir o objetivo deste trabalho foi a implementação de algoritmos para seleção. Para isso foi proposto e implementado o algoritmo de seleção por distância Euclidiana. Esse algoritmo foi avaliado e comparado com um outro algoritmo que não leva em consideração atributos de QoS e seleciona de forma aleatória os serviços componentes de um fluxo. O algoritmo de seleção por distância Euclidiana se mostrou eficiente em casos onde

a complexidade do fluxo é baixa e em casos onde não existiam muitos clientes fazendo acessos simultâneos ao serviço composto.

O trabalho desenvolvido neste projeto de mestrado é uma primeira versão de um protótipo que ainda carece de evolução para poder entrar em funcionamento de forma estável em ambientes corporativos. Na fase atual do *middleware*, ele deve passar por novas melhorias para ser integrado como um módulo da arquitetura WSARCH, proposta para prover *Web Services* considerando aspectos de QoS. A literatura raramente apresenta trabalhos em que existe a implementação de um ambiente que realize composição de serviços capaz de oferecer qualidade de serviço, e em que exista suporte à inserção de algoritmos mais complexos que possam receber a visão do fluxo como um todo, para poder escolher serviços que atendam à requisição de forma otimizada.

8.2 Contribuições

As principais contribuições realizadas como o desenvolvimento deste trabalho de mestrado são listadas a seguir:

- **Um protótipo de ferramenta capaz de fornecer serviço composto considerando qualidade de serviço, por meio da composição dinâmica**: O DWSC-M pode ser considerado a maior contribuição deste trabalho, uma vez que por meio dele os algoritmos propostos foram implementados e testados. O desenvolvimento do *middleware* deverá se estender em um projeto de doutorado e, como forma de contribuir para pesquisas na área, há a pretensão de disponibilizar o código fonte como software livre.
- **Algoritmo para seleção de Web Services para composição de serviços com QoS**: Outra contribuição relevante obtida com o desenvolvimento deste trabalho foi a implementação do algoritmo para seleção de *Web Services* por distância Euclidiana. Esse algoritmo, cuja idéia extendeu um trabalho da literatura da área, se mostrou uma abordagem interessante para seleção de serviços para composição, uma vez que os resultados se mostraram adequados diante da solicitação dos clientes, especialmente quando o fluxo composto não é muito complexo e também em casos em que a sobrecarga do sistema não é alta.

- **Estudo de caso em composição de serviços com QoS:** Outra contribuição que pode ser destacada durante o desenvolvimento deste projeto de mestrado foi a definição do experimentos, ocasião em que foi implementado um ambiente SOA com a participação de vários provedores de serviços, um registro de serviço e a ferramenta para composição de serviços com QoS. Por meio dos experimentos foi possível observar em um ambiente implementado, como um algoritmo de seleção complexo, o de distância Euclidiana, por exemplo, pode sofrer quando a complexidade do fluxo e a sobrecarga no sistema aumentam.

8.3 Trabalhos Futuros

Como discutido na seção 8.2, diversas melhorias em torno do desenvolvimento do *middleware* podem ser vislumbradas, sendo algumas destacadas aqui como trabalhos futuros:

- **Estudos de como os fatores influenciam nas variáveis de respostas podem ser analisados:** Foram apresentados no capítulo 7 desta monografia o planejamento de experimentos e o estudo de como os fatores influenciaram nos resultados de tais experimentos. Esse estudo deverá ocorrer de maneira mais ampla e cobrir mais detalhes da influência desses fatores nos experimentos.
- **Expansão do *middleware* DWSC-M: A** expansão da ferramenta deverá ser feita no sentido de conseguir dar suporte a mais tipos de conexões SOAP. Atualmente ela não permite acesso a qualquer tipo de *Web Service*. Além disso pretende-se criar um novo módulo na ferramenta que suporte ontologias, cujo enfoque é a descoberta de serviços para composição de *Web Services.*
- **Implementação e avaliação de algoritmos para composição de *Web Services*:** Outros modelos de algoritmos propostos em trabalhos da literatura deverão ser implementados para avaliar seu comportamento por meio de testes comparativos.
- **Criação de SLAs para composição de serviços:** A idéia é adaptar o DWSC-M para suportar SLAs e tornar mais eficiente a forma de como as requisições dos clientes são atendidas. Assim, não será necessário executar o algoritmo de seleção de serviços a cada requisição, o que deve sinalizar para uma diminuição na sobrecarga pela execução do serviço composto.

Referências Bibliográficas

ACTIVE ENDPOINTS. ***ActiveBPEL Open Source Engine, BPEL Standard - Active Endpoints.*** Disponível em: <**http://www.activevos.com/community-open-source.php**> (Último acesso em 20/10/2009)

ADAMS, Michael et al. **Yet Another Workflow Language**. Disponível em: <http://www.yawl-system.com>. (Último acesso em: 20/10/2009)

ALAMRI, A., EID, M., and SADDIK, A. E. 2006. **Classification of the state-of-the-art dynamic web services composition techniques.** *Int. J. Web Grid Serv.* 2, 2 (Sep. 2006), 148-166. DOI= http://dx.doi.org/10.1504/IJWGS.2006.010805

BIH, Joseph. ***Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions***. Ubiquity, New York, p.1-1, 08 ago. 2006.

BRAY, Tim et al. ***Extensible Markup Language (XML) 1.0*** (Fourth Edition). Disponível em: <http://www.w3.org/TR/2004/REC-xml11-20040204/>. Acesso em: 25 fev. 2008.

CHANDRASEKARAN, Senthilanand. ***Composition, Performance Analysis and Simulation of Web Services***. 2002. 39 f. Thesis (Master) - University Of Georgia, Georgia, 2002.

CARDELLINI, V., CASALICCHIO, E., GRASSI, V., Lo PRESTI, F.: ***Flow-Based Service Selection for Web Service Composition Supporting Multiple QoS Classes.*** In: Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, Utah, July 2007, pp. 743–750 (2007)

COMER, D. E.. ***Internetworking with TCP/IP: Principles, Protocols, and Architectures***. 4. ed. : Prentice Hall, 2000. 750 p.

DE PAOLI, F., LULLI, G., MAURINO, A.: **Design of quality-based composite web services.** In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 153–164. Springer, Heidelberg (2006)

DUSTDAR, S. and SCHREINER, W. 2005. **A survey on web services composition**. *Int. J. Web Grid Serv.* 1, 1 (Aug. 2005), 1-30.

ERRADI, Abdelkarim; MAHESHWARI, Piyush. ***A Broker-Based Approach for Improving Web Services Reliability***. Proceedings Of The Ieee International Conference On Web Services (icws'05), Washington Dc., p.355-362, 2005.

ESTRELLA, J. C. ***WSARCH: Uma arquitetura para provisão de Web Services com Grids e Qualidade de Serviço (QoS)***. 06/09/2007. 67 f. Dissertação (Qualificação de Doutorado) - ICMC - USP, São Carlos, 2007.

EZENWOYE, O. and SADJADI M., Robustbpel2: ***Transparent autonomization in business processes through dynamic proxies, Autonomous Decentralized Systems***, 2007. ISADS '07. Eighth International Symposium on, March 2007, pp. 17– 24.

FARKAS, P.; CHARAF, H. ***Web Services Planning Concepts***. Journal Of .net Technologies, p.9-12, 2003.

FERRIS, C.; FARREL, J. ***What are Web Services?*** Communications Of The Acm, New York, n. , p.31-31, jun. 2003.

FOUNDATION, Apache Software. WebServices - Axis. Disponível em: <http://ws.apache.org/axis/>. Acesso em: 10 out. 2009.

GOUSCOS, D.; KALIKAKIS, M.; GEORGIADIS, P. ***An Approach to Modeling Web Service QoS and Provision Price***. Proceedings Of The Fourth International Conference On Web Information Systems Engineering Workshops (wisew'03), 2004.

GRONMO, R.; JAEGER, M. ***Model-Driven Methodology for Building QoS-Optimised Web Service Compositions.*** International Federation for Information Proceeding.(IFIP 2005) p.68-82, 2005

JANSSEN, M.; CRESSWELL, A. M.. ***An enterprise application integration methodology for e-government***. Journal Of Enterprise Information Management, p. 531-547, 2005

JAEGER, M. C.; LADNER, H. ***Improving the QoS of WS Compositions based on Redundant Services***. Proceedings Of The International Conference On Next Generation Web Services Practices (nwesp'05), 2005.

JAYASINGHE , D. **Quickstart Apache Axis2: A practical guide to creating quality web services** . Birmingham: Packt Publishing, 2008. 186 p.

KALEPU, S., KRISHNASWAMY, S., and LOKE, S.W. (2003). **Verity: A qos metric for selecting web services and providers**. In Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW´03). IEEE CS Press.

KREGER, H. et al. ***Web Services Conceptual Architecture (WSCA 1.0)***. Disponível em: <http://www.cs.uoi.gr/~zarras/mdw-ws/WebServicesConceptualArchitectu2.pdf>. Acesso em: 24 fev. 2008. maior 2001.

LEAVITT, N. ***Are Web Services Finally Ready to Deliver?*** Computer, p.14-18, 2004.

MENASCÉ, D. A.. ***Composing Web Services: A QoS View***. Ieee Internet Computing, p. 88-90, dez. 2004.

MOODIE, M. ***Pro Apache Tomcat 6***. New York: Apress, 2007. 325 p.

MOSER O., ROSENBERG F., and DUSTDAR S. ***Non-intrusive monitoring and service adaptation for ws-bpel***, WWW '08: Proceeding of the 17th international conference on World Wide Web (New York, NY, USA), ACM, 2008, pp. 815–824.

NGUYEN, X. T.; KOWALCZYK, R.; PHAN, M. T.; **Modelling and Solving QoS Composition Problem Using Fuzzy DisCSP.** Web Services, 2006. ICWS '06. International Conference on 18-22 Sept. 2006 Page(s):55 - 62

PAPAZOGLOU, Mike P.. ***Service -Oriented Computing: Concepts, Characteristics and Directions***. Proceedings Of The Fourth International Conference On Web Information Systems Engineering (wise'03), 2003.

PAPAZOGLOU, M. P.; GEORGAKOPOULOS, D.. ***Service-Oriented Computing***. Communications Of The Acm, p.25-28, out. 2001.

Ping Wang, Kuo-Ming Chao, Chi-Chun Lo, Chun-Lung Huang, Yinsheng Li, **"A Fuzzy Model for Selection of QoS-Aware Web Services,"** icebe, pp.585-593, IEEE International Conference on e-Business Engineering (ICEBE'06), 2006

RAO J. and SU X.. **"A Survey of Automated Web Service Composition Methods"**. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, July 6th, 2004.

SHETH, Amit et al. ***QoS for Service-oriented Middleware***. Proceedings Of The Conference On Systemics, Cybernetics And Informatics, Orlando, 2002.

SIBLINI, R.; MANSOUR, N. ***Testing Web services***. The 3rd Acs/ieee International Conference On Computer Systems And Applications, 2005.

TAHER, L.; BASHA, R.; KHATIB, H. E. ***Establishing Association between QoS Properties in Service Oriented Architecture***. International Conference On Next Generation Web Services Practices (nwesp'05), 2005.

THOMAS, J. P.; THOMAS, M.; GHINEA, G. ***Modeling of Web Services Flow.*** Proceedings Of The Ieee International Conference On E-commerce (cec'03), 2003.

TIAN, M. et al. ***A concept for QoS integration in Web services***. Fourth International Conference On Web Information Systems Engineering Workshops, 2003. Proceedings., 2003.

TOSIC, V. et al. ***Management of Compositions of E-and-M Business Web Services with Multiple Classes of Services***. Network Operations And Management Symposium, 2002. Noms 2002. 2002 Ieee/ifip, 2002.

W3C, KAVANTZAS N., BURDETT D., RITZINGER G., **Web Services Choreography Description Language Version 1.0** Disponível em: http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/, Abril 2004 (Último Acesso em: 20/10/2009)

WEERAWARANA, S. ***Web Services Platform Architecture***. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. 1. ed.: Prentice Hall PTR, 2005.

WOODSIDE, M.; MENASCÉ, D.. ***Application-Level QoS***. IEEE Computer Society, 2004.