WEBTRANSACT-EM: UM MODELO PARA A EXECUÇÃO DINÂMICA DE SERVIÇOS WEB SEMANTICAMENTE EQUIVALENTES

Valdino de Azevedo Junior

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Paulo de Figueiredo Pires, D.Sc.

Profª. Maria Luiza Machado Campos, Ph.D.

Prof. Alberto Henrique Frade Laender, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2003

AZEVEDO JUNIOR, VALDINO DE

WEBTRANSACT-EM: Um Modelo para a Execução Dinâmica de Serviços Web Semanticamente Equivalentes [Rio de Janeiro] 2003

VII, 117 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2003)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Serviços Web
2. Qualidade de Serviço
3. Execução Dinâmica

I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

WEBTRANSACT-EM: UM MODELO PARA A EXECUÇÃO DINÂMICA DE SERVIÇOS WEB SEMANTICAMENTE EQUIVALENTES.

Valdino de Azevedo Junior

Março/2003

Orientadores: Marta Lima de Queirós Mattoso
Paulo de Figueiredo Pires

Programa: Engenharia de Sistemas e Computação

A tecnologia de serviços Web oferece os mecanismos necessários de forma a permitir a comunicação interoperável entre aplicações e serviços dentro do ambiente Web. Cada vez mais, serviços oferecidos por diferentes organizações, porém, possuindo funcionalidades equivalentes, são disponibilizados na Web, facilitando a construção de novas aplicações através da composição de serviços Web. Entretanto, essa escolha de qual desses serviços semanticamente equivalentes executar não é uma tarefa simples, dada essa grande oferta de serviços equivalentes e a ausência na literatura de modelos de execução específicos para serviços Web que auxiliem nessa escolha. De forma a oferecer a aplicações clientes uma visão homogeneizada desses serviços equivalentes, estes precisam ser localizados e agrupados em classes, sendo posteriormente escalonados para execução de acordo com seus parâmetros de custo e critérios de qualidade. Nesta dissertação, propomos o *WebTransact-EM*, um modelo que representa critérios de qualidade e parâmetros de custo aplicáveis a serviços Web e suas operações e que realiza a escolha de serviços semanticamente equivalentes dentro de uma classe para serem executados, baseado nesses critérios. Esse modelo foi avaliado através da implementação da arquitetura *WebTransact*, que provê uma visão homogeneizada de serviços semanticamente equivalentes através do uso da tecnologia de mediadores. Exemplos de uso do modelo *WebTransact-EM* que exploram características de custo e qualidade de serviços Web foram realizados, resultando em escalonamentos de serviços que refletem de forma consistente o que foi solicitado por uma aplicação cliente, tornando o processo de execução dinâmica de serviços flexível e transparente.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

WEBTRANSACT-EM: A MODEL FOR DYNAMIC EXECUTION OF SEMANTICALLY EQUIVALENT WEB SERVICES

Valdino de Azevedo Junior

March/2003

Advisors: Marta Lima de Queirós Mattoso
Paulo de Figueiredo Pires

Department: Systems Engineering and Computer Science

The Web Services technology provides the necessary mechanisms to enable communication interoperability between applications and services in the Web environment. An increasing number of on-line services provided by different organizations, but having the same semantic functionality are available in the Web, facilitating the building process of new applications through Web Services compositions. However, the choice of which of these semantically equivalent services to execute is not an easy task, given this great offer of equivalent services and the lack of execution models specifically for web services in the literature that helps on this choice. In order to provide to client applications a homogenized view of these equivalent services, they must be discovered and grouped in service classes, being further scheduled for execution according to their cost parameters and quality criteria. In this dissertation, we propose the *WebTransact-EM*, a model that represents quality criteria and cost parameters applied to web services and its operations and that chooses semantically equivalent services inside a classe to be executed, based on these criteria. This model was evaluated through the implementation of the WebTransact framework, which provides a homogenized view of semantically equivalent web services through the usage of the mediator technology. Examples using the WebTransact-EM model, exploring cost and quality characteristics of services were made, resulting in services schedules that shows in a solid way what was asked by a client application, making the process of dynamic execution of services more clear and flexible.

Índice

Índice de Figuras

1. Introdução

1.1 - Motivação

Há poucos anos atrás, o uso da *World Wide Web* era limitado às interações entre clientes, utilizando seus navegadores (*browsers*), e servidores Web, que ofereciam documentos e informações a serem acessados. Com a evolução das tecnologias ligadas à Internet, especialmente após o surgimento de padrões baseados em XML (BRAY *et al.*, 2000), tornou-se possível a comunicação entre serviços e aplicações desenvolvidos em plataformas distintas, dando um grande impulso à comunicação distribuída e à troca de informações entre organizações. Essa comunicação entre aplicações e serviços atualmente é bastante facilitada devido à tecnologia de serviços Web.

De uma forma simplificada, um serviço Web pode ser descrito como um processo de negócio, uma aplicação ou uma funcionalidade que se encontra disponível para ser utilizada, tanto internamente, dentro de uma organização, ou externamente, para aplicações clientes utilizando a Internet e seus principais protocolos.

Uma das principais características da tecnologia de serviços Web é o uso de protocolos e linguagens baseadas em XML para a especificação de interfaces, descrição de serviços e protocolos de comunicação. Como principais protocolos e linguages desta tecnologia, temos a WSDL (CHRISTENSEN *et al.*, 2001), linguagem para descrição de serviços, o SOAP (BOX *et al.*, 2001), protocolo de comunicação, e o UDDI (UDDI.org, 2000), especificação para registro e localização de serviços, funcionando como um repositório de serviços.

A principal vantagem obtida com a utilização de serviços Web é a interoperabilidade. Serviços desenvolvidos em diferentes linguagens de programação e executando sob sistemas operacionais distintos podem comunicar-se e trocar informações de uma forma facilitada. Isso é possível devido à utilização das linguagens e protocolos padrão, baseados em XML. Devido ao grau de interoperabilidade alcançado através da tecnologia de serviços Web, espera-se um menor custo no desenvolvimento de novas aplicações, dada a maior possibilidade de

reuso de sistemas legados desenvolvidos em diferentes linguagens e hospedados em qualquer máquina que possa ser acessada através da Internet.

Com a grande oferta de serviços existente atualmente, é bastante comum encontrarmos serviços desenvolvidos por diferentes organizações, com interfaces distintas, porém, oferecendo a mesma funcionalidade ou desempenhando uma mesma tarefa (ex: serviços de validação de cartão de crédito, reservas de hotéis e carros, compras de livros, entre outros). Esses serviços são ditos *semanticamente equivalentes*, dada a similaridade semântica presente em suas funcionalidades.

De forma a facilitar a utilização de serviços Web por parte de uma aplicação cliente, esta poderia informar apenas a classe de serviços que gostaria de executar. Esta classe agruparia serviços de funcionalidade equivalente, podendo ser visualizada como um *serviço virtual*, que oferece uma interface única para as aplicações clientes e delega sua execução para um ou mais dos serviços remotos por ela agregados, utilizando como base dessa escolha as características não funcionais desses serviços. Além disso, a utilização dessas classes tornaria mais flexível o processo de composição de serviços, uma vez que ao invés de inserirmos um serviço específico em uma composição, apenas indicaríamos a classe de serviços que gostaríamos de executar (bem como as restrições de qualidade que um serviço dessa classe deve atender), com o serviço a ser efetivamente executado sendo apenas decidido em tempo de execução. Como exemplo de utilização de serviços de funcionalidade equivalente, podemos citar o trabalho de (ROCCO e CRITCHLOW, 2002), onde aplicações legadas de bioinformática de mesma funcionalidade são integradas, através de um mediador com uma interface única. Esse mediador, então, exporta essa interface como um serviço Web, de forma que essa visão integrada da aplicação possa ser acessada remotamente através da Internet. Apesar de realizar a integração de aplicações legadas através de uma interface de um serviço Web, nenhuma consideração é feita sobre como funciona o processo de seleção de aplicações a serem executadas, a partir de uma solicitação utilizando a interface desse serviço.

Para a definição dessas classes de serviços semanticamentes equivalentes, duas questões devem ser tratadas: a descoberta de serviços Web de funcionalidade equivalente e a escolha de quais serviços pertencentes a uma classe efetivamente executar em um determinado instante de tempo.

Atualmente, a descoberta de serviços Web disponíveis é feita através de consultas efetuadas sobre o registro UDDI. Dada a dificuldade de se expressar semanticamente o que buscamos em uma consulta a esse registro, fica difícil localizarmos serviços que tenham uma mesma funcionalidade. De forma a resolver esse problema, diversos trabalhos vêm sendo desenvolvidos, tendo em comum a adição de uma maior capacidade semântica ao registro UDDI, permitindo a realização de consultas semanticamente mais complexas (ANKOLEKAR *et al.*, 2001, PAOLUCCI *et al.*, 2002a, MCILRAITH *et al.*, 2001). Todos esses trabalhos utilizam conceitos da Web Semântica (BERNERS-LEE, *et al.* 2001, HENDLER *et al.*, 2001), que busca através da associação de significados a cada dado presente na Web, facilitar processos de busca automatizados a encontrar determinada informação. Mecanismos para busca e extração de dados na Web (CALADO *et al.*, 2002, LAENDER *et al.*, 2002, MOURA *et al.*, 1998) também poderiam ser utilizados para auxiliar no trabalho de localização desses serviços. Esses processos de busca seriam úteis também no sentido de localizarem um serviço através de sua funcionalidade.

Dado um conjunto de serviços Web pertencentes a uma mesma classe de serviço, a decisão de qual destes serviços semanticamente equivalentes executar não é uma tarefa simples, devido a grande oferta de serviços no ambiente Web. Essa decisão deve ser baseada nas características não funcionais destes serviços, como seus custos de execução (custo monetário para sua utilização e tempo de resposta), seu suporte transacional e demais critérios de qualidade (mecanismos de segurança apresentados pelos serviços, entre outros). Atualmente, nenhuma plataforma com suporte a utilização de serviços Web oferece esse tipo de seleção de serviços, dada a dificuldade no agrupamento de serviços semanticamente equivalentes e a ausência de modelos para a definição dessas características não funcionais de um dado serviço Web.

1.2 – Objetivos

Mesmo considerando resolvido o problema da descoberta de serviços Web semanticamente equivalentes, identificamos ainda três questões a serem tratadas no processo de execução dinâmica desses serviços: Como agrupar esses serviços semanticamente equivalentes em classes? Como fazer para escolher um desses serviços para ser executado e tornar essa escolha um processo transparente para uma aplicação cliente? Como operar esses serviços semanticamente equivalentes?

Para o agrupamento de serviços em classes, pode ser utilizada a tecnologia de mediadores e tradutores (*wrappers*) (WIEDERHOLD, 1995). Mediadores são responsáveis por oferecer uma visão homogeneizada da classe de serviços equivalentes através de uma interface única, enquanto os tradutores tratam das dissimilaridades semânticas existentes entre as interfaces dos serviços e a interface do mediador.

Para a escolha de um desses serviços para ser executado, precisamos determinar estratégias de execução que levem em consideração critérios de custo e de qualidade de serviço que auxiliem nessa decisão. Além disso, alguns controles precisam ser providenciados para que as várias estratégias de execução sejam avaliadas, de forma que possamos escolher a mais apropriada para uma execução específica.

Finalmente, de forma a transformar essa escolha num processo transparente para uma aplicação cliente, as estratégias de execução devem estar implementadas numa plataforma que suporte a execução de serviços Web. Diversas plataformas comerciais foram desenvolvidas para darem suporte à utilização e criação de serviços Web. Como exemplo, podemos citar os seguintes produtos: Microsoft .NET (MICROSOFT CORP., 2000), IBM WebSphere (IBM CORP.,2000) e Sun One (SUN MICROSYSTEMS, 2002), entre outros. Apesar de permitirem e facilitarem a criação de novos serviços e a utilização de serviços já existentes disponíveis na Internet, essas plataformas não tratam de outros aspectos importantes na tecnologia de serviços Web, como: composições, um modelo de gerenciamento de transações específico e modelos de segurança, todos aplicáveis a serviços e suas operações. Estes não podem ser considerados problemas triviais, pois envolvem áreas distintas, como a definição de linguagens para a composição de serviços, a definição de critérios de qualidade para a escolha de serviços baseada nesses critérios e a especificação de um modelo transacional específico para serviços Web (que também pode ser considerada uma tarefa complexa, dado o caráter autônomo e heterogêneo dos serviços).

Apesar do grande número de trabalhos e projetos de pesquisa que tratam da gerencia de serviços Web, não identificamos trabalhos que enderecem essas três questões identificadas no problema de escolha de serviços Web. O principal objetivo desta dissertação é, então, a elaboração de um modelo para a execução dinâmica de serviços Web, baseado em parâmetros de custo e critérios de qualidade de um

serviço, de forma que este determine qual serviço executar em um determinado instante de tempo. Para o desenvolvimento de tal modelo, foram primeiramente identificados critérios de qualidade aplicáveis a serviços Web e suas operações, de forma que nosso processo de seleção possa se basear nesses critérios. Ainda, uma linguagem que padroniza como uma aplicação cliente deve oferecer esses critérios de qualidade a aplicações clientes, conhecida como WSQD (*Web Service Quality Definitions*) também é proposta nessa dissertação.

Para aplicarmos nosso modelo de execução, precisamos que serviços Web semanticamente equivalentes estejam agrupados de alguma forma. Para realizarmos esse agrupamento, utilizamos a tecnologia de mediadores, capaz de oferecer uma visão homogeneizada desses serviços equivalentes e tratar as diferenças existentes entre eles. Assim, utilizamos nesta dissertação a plataforma para execução de serviços Web conhecida como *WebTransact*, definida em (PIRES, 2002, PIRES *et al.* 2002, PIRES *et al.*, 2003) e baseada na tecnologia de mediadores. Utilizamos ainda o modelo de transações para serviços Web presente na *WebTransact*, de forma a controlarmos a execução em paralelo de diversos serviços semanticamente equivalentes. Além da implementação dos módulos básicos dessa plataforma, foram construídos módulos que utilizam o modelo de custo e execução apresentados nesta dissertação, que por estar implementado nessa plataforma, é conhecido como *WebTransact-EM* (*WebTransact Execution Model*).

Foram realizados exemplos de uso do modelo *WebTransact-EM* que exploram características de custo e qualidade de serviços Web semanticamente equivalentes. Através da variação das restrições feitas em cima dos critérios de qualidade de cada serviço, avaliamos o comportamento do modelo no processo de escolha e execução de serviços, obtendo como resultados escalonamentos de serviços que refletem de forma consistente o que foi solicitado por uma aplicação cliente, tornando o processo de execução dinâmica de serviços flexível e transparente.

1.3 – Trabalhos Relacionados

Não foram encontrados na literatura trabalhos que propusessem um modelo para a seleção e execução de serviços Web semanticamente equivalentes. Entretanto, alguns trabalhos abordam a questão da execução dinâmica de serviços Web, bem como da definição de critérios de custo e qualidade para serviços.

Em (KEIDL *et al.*, 2002a) e (KEIDL *et al.*, 2002b) é proposta uma plataforma conhecida como *ServiceGlobe* que além da construção e utilização de serviços, permite a alocação de serviços Web em computadores espalhados pela rede. A seleção dinâmica de serviços também é possibilitada por esta plataforma, embora apenas para serviços com a mesma interface e participantes da comunidade *ServiceGlobe*. Esta seleção é baseada em restrições feitas pelo cliente do serviço, estando armazenadas na forma de metadados em arquivos XML. O *ServiceGlobe* pode ser visto como uma extensão para serviços Web da plataforma *ObjectGlobe* (BRAUMANDL *et al.*, 2001), um processador de consultas distribuído para fontes de dados na Web. Ambos necessitam que um módulo do sistema seja instalado em cada máquina participante do *ServiceGlobe* ou do *ObjectGlobe*, o que limita um pouco a sua utilização. Ao contrário do *ServiceGlobe*, a *WebTransact* permite que a seleção não se restrinja apenas a serviços com mesma interface. Além disso, a camada de mediação também não precisa estar instalada numa máquina cliente participante, podendo ser utilizada de forma mais flexível, não sendo uma solução proprietária.

Em (AIELLO *et al.*,2002) e (PAPAZOGLOU *et al.*,2002) é apresentada uma linguagem para a requisição de serviços Web conhecida como XSRL (*XML Service Request Language*). Com um formato semelhante ao utilizado pela linguagem XQuery (BOAG *et al.*, 2002), uma linguagem para consultas em documentos XML, a XSRL possibilita a execução de um ou mais serviços Web de forma a se executar uma determinada tarefa (que poderia ser vista como uma composição de serviços). Cada domínio de aplicação deve ser traduzido em um esquema XML, de forma que consultas XSRL possam ser feitas em cima desse esquema. Entretanto, a seleção de serviços é feita tendo por base elementos de um arquivo XML, específico para cada domínio de aplicação. Aspectos de qualidade e parâmetros de custo, ao contrário de nossa proposta, não são considerados nestes trabalhos.

Propostas para a elaboração de linguagens que representem aspectos de custo e critérios de qualidade também foram feitas. Em (FERGUSON, 2001), é apresentada a idéia da linguagem WSEL (Web Services Endpoint Language), cujo objetivo é descrever aspectos de qualidade associados a uma implementação de um serviço Web. Entretanto, até o momento, a especificação dessa linguagem ainda não está disponível. Já em (TOSIC et al.,2002), é proposta a linguagem WSOL (*Web Services Offerings Language*), que também descreve aspectos de qualidade associados a serviços Web. Ela se baseia na definição de contratos (SLAs – *Service Level Agreements*) entre provedor e cliente de serviços Web, originando classes de

serviços que seriam formadas dependendo do que esteja definido nestes contratos. Uma vez disponível uma implementação dessas linguagens, esta poderia se beneficiar do modelo de custo e execução de serviços apresentado nessa dissertação.

A questão da composição de serviços Web também tem sido bastante pesquisada, resultando na criação de plataformas que auxiliem num processo de composição de serviços Web de forma dinâmica e confiável. Além da plataforma *ServiceGlobe*, discutida anteriormente, podemos citar a arquitetura DAMSC (*Dynamically Adaptable and Manageable Service Compositions*), proposta em (TOSIC *et al.*, 2001), que se baseia na criação de classes de serviços com diferentes critérios de qualidade e a plataforma SELF-SERV (SHENG, *et al.*, 2002), onde serviços são compostos através de uma linguagem declarativa e executados de forma dinâmica e distribuída. A plataforma WebTransact (PIRES, 2002, PIRES *et al.* 2002, PIRES *et al.*, 2003) também possibilita a composição de serviços e o agrupamento destes em classes, garantindo ainda a execução de forma confiável através do gerenciamento de transações entre os serviços participantes.

A vantagem da WebTransact em relação às demais plataformas é sua capacidade de tratar serviços semanticamente equivalentes, porém de interfaces diferentes, agrupando-os em classes de serviços representadas por mediadores. Dissimilaridades semânticas e transacionais entre os serviços pertencentes a uma classe também são tratadas, resultando numa plataforma confiável para a composição de serviços (livrando uma aplicação cliente de ter de gerenciar transações entre serviços participantes) e que não necessita da instalação de nenhum programa em uma máquina cliente que queira utilizá-la. Entretanto, em sua especificação (PIRES, 2002), a WebTransact não levava em consideração, em seu modelo de execução, critérios de qualidade e parâmetros de custo de um serviço, que auxiliassem no processo de seleção e execução de serviços. Assim, utilizamos essa especificação como base para a implementação e validação do modelo de execução discutido nesta dissertação, o *WebTransact-EM*.

1.4 – Organização do Texto

O restante dessa dissertação está organizado da seguinte forma: no Capítulo 2 é feita uma revisão da tecnologia de serviços Web, mostrando seus principais

protocolos (WSDL, SOAP, UDDI) e propostas de linguagens para composição de serviços Web.

No Capítulo 3, é apresentada a plataforma *WebTransact*, que possibilita a execução e composição de serviços de forma segura através da definição de serviços, grupos de serviços e aplicações especificados numa linguagem conhecida como WSTL (*Web Services Transaction Language*).

No Capítulo 4, são discutidas as estratégias para a execução dinâmica de serviços Web. São apresentados os critérios de qualidade e parâmetros de custo aplicáveis a serviços e suas operações, bem como uma linguagem para a representação desses parâmetros (conhecida como WSQD – *Web Services Quality Definitions*). Além disso, é mostrado o modelo de custo utilizado para o cálculo do tempo de resposta de um serviço e as estratégias do modelo de execução dinâmica a serem utilizadas dependendo dos critérios de qualidade selecionados por uma aplicação cliente.

No Capítulo 5, são mostrados a especificação e o projeto de implementação do modelo de execução *WebTransact-EM*, como por exemplo: o diagrama de classes utilizado na criação da plataforma *WebTransact* e os algoritmos utilizados para execução dinâmica de serviços. No Capítulo 6, mostramos exemplos de serviços com características de qualidade distintas sendo executados usando o *WebTransact-EM*.

Finalmente no Capítulo 7, são apresentadas as conclusões obtidas com esta dissertação, bem como sugestões para trabalhos futuros.

2 – Serviços WEB

2.1 – Introdução

Há poucos anos atrás, o uso da *World Wide Web* (WWW) era limitado às interações entre clientes, usando seus navegadores (*browsers*) e servidores Web. Inicialmente, apenas páginas estáticas eram construídas: o cliente solicitava uma página a um servidor Web e este retornava a página HTML para o cliente, que era então interpretada e exibida pelo navegador. Com o passar do tempo, iniciou-se o desenvolvimento de páginas dinâmicas: o cliente fazia a requisição de uma página; essa requisição era então processada pelo servidor, gerando um código HTML dinâmico que era retornado ao solicitante.

Com a evolução da Internet, tornou-se possível a programação de aplicações Web, que a partir de informações fornecidas pelo cliente, realizam um processamento no servidor, acessando informações em bancos de dados e tomando decisões a partir dessas informações. Hoje em dia, é comum podermos realizar várias ações pela Internet, como pagamento de contas em bancos, reservas de passagens aéreas, entre outras.

Apesar de todas essas mudanças, a comunicação entre clientes e servidores ainda ficava limitada ao uso de navegadores. Isso tornava difícil a troca de informações e serviços entre organizações; o que se desejava era que uma aplicação, produzida por uma empresa, pudesse ser utilizada por outra aplicação, desenvolvida por uma outra organização. A forma tradicional de comunicação via Internet, utilizando navegadores que na maioria dos casos apenas interpretavam documentos HTML era um fator limitante a essa comunicação entre aplicações.

Sendo assim, passaram a surgir algumas novas tecnologias, numa tentativa de melhor identificar e resolver esse problema. Uma dessas definições foi o conceito de um serviço eletrônico, mais conhecido como *E-Service*.

Segundo (MECELLA e PERNICI, 2001), um serviço eletrônico é uma aplicação, produzida por uma organização, que pode ser facilmente reutilizada num ambiente distribuído como a Internet. Para ser considerada um serviço eletrônico porém, essa aplicação deveria ser:

- Aberta, isto é, independente de plataformas o máximo possível ;
- Desenvolvida principalmente para ser utilizada entre organizações e não somente para ser utilizada dentro da própria empresa que a produziu ;
- Facilmente Adaptável a outras aplicações, isto é, sua integração com outras aplicações não tornaria preciso o uso de adaptadores complexos.

Algumas iniciativas da indústria, principalmente aquelas baseadas no desenvolvimento de componentes, como o modelo COM+ (*Component Object Model*) (MICROSOFT CORP. e DIGITAL EQUIPMENT CORP.,1995) e o EJB (*Enterprise Java Beans*) (SUN MICROSYSTEMS, 2001), tiveram grande sucesso no desenvolvimento de aplicações distribuídas e interoperáveis dentro de uma organização. Entretanto, pelo fato de não serem totalmente independentes de plataformas e tecnologias, utilizá-las para a comunicação entre aplicações de diferentes organizações não é uma tarefa simples.

Finalmente, com o surgimento de padrões baseados em XML para a troca de dados e comunicação entre computadores, tornou-se possível o desenvolvimento de serviços facilmente integráveis e independentes de plataforma, dado o alto grau de interoperabilidade obtido através do uso desses padrões. Criou-se então, o conceito de serviços Web, ou *Web Services*.

2.2 – Definição de Serviços Web

Utilizaremos como definição de serviços Web a definição proposta pelo W3C (*World Wide Web Consortium*): "Um serviço Web é uma aplicação, identificada por um URI (*Universal Resource Identifer*), cujas interfaces e implementações são capazes de serem definidas, descritas e localizadas utilizando-se a linguagem XML e suas ferramentas. Um serviço Web deve ser capaz de interagir com outras aplicações através da troca de mensagens baseadas em XML utilizando os protocolos de comunicação padrão existentes na Internet" (AUSTIN *et. al*, 2002).

Assim como componentes de *software*, os serviços Web são utilizados através do conceito de "caixa-preta": clientes utilizam suas funcões sem se preocupar em como este serviço é implementado. Aplicações consumidoras de um serviço podem

ser implementadas em qualquer plataforma, utilizando qualquer linguagem de programação, devido ao fato de as mensagens definidas na interface de um serviço serem expressas em padrões conhecidos, baseados em XML.

Utilizando o conceito de serviços Web, podemos construir uma plataforma para a integração de aplicações de diferentes localidades. As principais funcões e protocolos utilizados na construção desse tipo de plataforma serão discutidos na próxima seção.

2.3 – Plataformas para Execução de Serviços Web

De acordo com (CURBERA *et al.*, 2001), a arquitetura de serviços Web pode ser vista como uma tentativa de transformar o conjunto das tecnologias utilizadas atualmente numa plataforma padrão para a integração de aplicações distribuídas. Essa plataforma poderia ser dividida em quatro conjuntos básicos de especificações, mencionados a seguir:

- **Protocolos de comunicação**: Contém a descrição dos formatos e protocolos para a comunicação entre aplicações.
- **Descrição de Serviços**: Contém a definição dos tipos de dados, operações e mensagens de um serviço, além de informações sobre como acessá-lo; em outras palavras, contém a descrição de um serviço.
- **Descrição de Composição de Serviços**: Contém os modelos e linguagens utilizadas para descrever como estes serviços interagem.
- **Publicação e Descoberta de Serviços**: Contém protocolos que permitam a localização da descrição desses serviços.

Todos esses quatro conjuntos de especificações apresentam uma característica comum: o uso da linguagem XML e suas descrições ou metadados (XML *Schema* (FALLSIDE, 2001)) como forma de garantir a interoperabilidade necessária para a arquitetura.

De forma a implementar esses conceitos, foram definidos diversos protocolos que possibilitam a descrição e localização de aplicações, bem como a comunicação

entre estas. A Figura 2.1 mostra como esses protocolos estão organizados, com o UDDI e linguagens para composição de serviços estando nas camadas superiores.

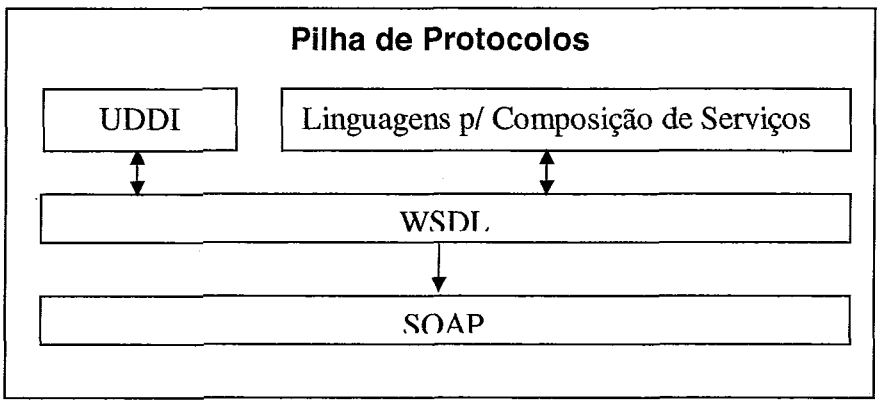


Figura 2.1 – Pilha de Protocolos utilizados numa arquitetura de Serviços *Web*

Nas próximas sub-seções, discutimos cada um desses protocolos, de uma forma simplificada.

2.3.1 – Protocolos de Comunicação

O protocolo de comunicação mais utilizado atualmente na comunicação entre serviços é o SOAP (BOX *et al.*, 2001). Anteriormente ao SOAP, os protocolos de comunicação utilizados nas chamadas de procedimentos remotos eram implementações específicas de fabricantes, obedecendo às orientações de algumas especificações como DCOM (MICROSOFT CORP. e DIGITAL EQUIPMENT CORP., 1995), CORBA (OMG, 1995) e o EJB(SUN MICROSYSTEMS, 2001). Apesar do relativo sucesso na comunicação entre aplicações hospedadas em plataformas homogêneas, o grau de interoperabilidade entre essas especificações era limitado, pois em um ambiente como a Web, a interação entre diferentes modelos é necessária, o que não era suportado por essas especificações (ex: programas utilizando o modelo COM podem querer se comunicar com uma aplicação utilizando o modelo CORBA ou EJB. Isso não era possível somente com essas especificações).

Utilizando padrões como XML em suas especificações e o protocolo HTTP na comunicação de dados, o SOAP vem sendo utilizado como o protocolo para a troca de informações em ambientes descentralizados e distribuídos. A especificação SOAP contém três partes principais:

- A definição de um invólucro para mensagens, denominado *SOAP Envelope*, que define um modelo para descrever o que está dentro de uma mensagem e como ela será processada.

- Um conjunto de regras de codificação, para expressar tipos de dados definidos pelo usuário.

- Uma convenção, para representar chamadas de procedimentos remotos e suas respectivas respostas.

O elemento *SOAP Envelope* representa quais mensagens serão transmitidas entre aplicações. A especicação SOAP (BOX *et al.*, 2001), define as seguintes partes componentes de uma mensagem:

- Um elemento *SOAP Envelope* obrigatório, que é o elemento raiz do documento XML que representa a mensagem.

- Um subelemento *SOAP Header* opcional, que define algumas características e acordos, especificados como atributos deste elemento, que deverão ser aceitos pelas aplicações que estiverem se comunicando.

- Um subelemento *SOAP Body* obrigatório, que contém todas as informações obrigatórias relativas a uma mensagem, como o nome da operação a ser chamada e seus parâmetros. O protocolo SOAP define também um subelemento para o *SOAP Body*, denominado *SOAP Fault*, que é utilizado para descrever informações de erro que possam ter ocorrido na chamada de um método remoto em outra aplicação.

A especificação do SOAP também define uma vinculação deste protocolo com o protocolo HTTP. Como este último é o protocolo padrão de comunicação utilizado em ambientes distribuídos como a Internet, é interessante que tenhamos uma especificação de como transmitir uma mensagem SOAP utilizando o protocolo HTTP. Assim, temos as diretrizes de como uma mensagem SOAP é vinculada a um pacote HTTP, para que esta seja utilizada na comunicação entre aplicações.

No Exemplo 2.1, é mostrada uma mensagem SOAP sendo transmitida utilizando-se o protocolo http.

```
POST /StockQuote HTTP/1.1
Host : www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn
SOAPAction: 'Some-URI'

<SOAP-ENV: Envelope
        xmlns: SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingstyle="http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Body>
                <m: GetLastTradePrice xmlns:m="Some-URI">
                        <symbol>DIS</symbol>
                </m:GetLastTradePrice>
        <SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo 2.1 - Mensagem SOAP

No exemplo, está sendo enviada uma solicitação de uma operação de nome "*GetLastTradePrice*". Essa operação possui um parâmetro, de nome "symbol", cujo valor é "DIS". Uma requisição SOAP é geralmente enviada através do protocolo HTTP e seu método POST (HTTP POST), com o tipo de conteúdo (content type) definido como "text/xml". O campo *SOAPAction*, na mensagem SOAP, pode ser nulo ou conter o nome de algum método SOAP, permitindo que o servidor Web que está hospedando o serviço faça o roteamento ou filtro necessário para a execução da requisição.

2.3.2 – Descrição de Serviços

A linguagem para definição de serviços Web é a WSDL (*Web Services Definition Language*) (CHRISTENSEN *et al.*, 2001), uma linguagem baseada em XML que define os serviços de uma forma padronizada e independente de plataforma. Um documento WSDL oferece a uma aplicação cliente detalhes sobre a interface de um serviço, bem como informações sobre "como" e "onde" acessá-lo.

A estrutura de um documento WSDL é definida da seguinte forma: há um elemento de nome "*definitions*", que é o elemento raiz do documento WSDL, possuindo alguns atributos que definem o nome do documento WSDL e seus espaços identificadores (*namespaces*). Imediatamente abaixo do elemento "*definitions*", temos os subelementos "*types*", "*message*", "*porttype*", "*binding*" e "*service*". Cada um desses subelementos tem uma seção específica no documento WSDL e possuem sua funcionalidade explicada a seguir :

- Elemento *Types*: Nesta seção, são feitas definições de tipos independentes de plataforma ou linguagens, possibilitando que esses tipos de dados sejam utilizados nas mensagens a serem trocadas entre aplicações.

- Elemento *Message*: Corresponde às definições de mensagens a serem trocadas entre as aplicações. Um elemento "*message*" pode possuir diversos subelementos do tipo "*part*", que correspondem aos atributos desta mensagem. A cada atributo representado por um elemento "*part*", está associado um nome e um tipo de dados.

- Elemento *PortType*: Aqui, é definido um conjunto de operações suportadas por um determinado serviço. A cada operação, podem estar associadas mensagens de entrada (definidas pelo subelemento "*input*"), de saída (definidas pelo subelemento "*output*") e de erro (definidas pelo subelemento "*fault*").

- Elemento *Binding* : Um *binding* é uma implementação de uma interface definida pelo elemento "porttype", utilizando algum protocolo de comunicação. Nesta seção, são especificados os detalhes da transmissão de dados, como o formato das mensagens e o protocolo utilizado na comunicação. Um elemento "*binding*" referencia um elemento "*porttype*" específico através do atributo *type*. Os subelementos "*operation*", "*input*", "*output*" e "*fault*" são relacionados aos subelementos de mesmo nome do elemento "*porttype*" referenciado por um elemento "*binding*". O protocolo de comunicação mais utilizado, conforme já discutido, é o protocolo SOAP. Nesta seção, detalhes do protocolo de comunicação a ser utilizado são incluídos como subelementos do elemento "binding".

- Elemento *Service* : Nesta seção, é definido o conceito de um serviço. Um elemento "*service*" possui vários subelementos "*port*". Cada elemento "*port*" está relacionado a um elemento "*binding*" particular, através do atributo *binding*, especificando qual interface e qual protocolo de comunicação estão sendo utilizados nessa implementação. Dependendo do protocolo de comunicação utilizado, um elemento "*port*" pode possuir sublementos específicos, indicando qual URL deve ser utilizada para se acessar a implementação do serviço. Por exemplo, caso estejamos utilizando o protocolo SOAP, definimos o elemento "*soap:address*", subelemento do elemento "*port*",

que irá conter informações sobre a URL pela qual um determinado elemento "*port*" de um serviço poderá ser acessado.

O exemplo 2.2 mostra um documento WSDL e seus principais elementos.

```
<definitions name="BNQuoteService"
      targetNamespace="http://www.xmethods.net/sd/BNQuoteService.wsdl"
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
      xmlns=http://schemas.xmlsoap.org/wsdl/
      xmlns:tns="http://www.xmethods.net/sd/BNQuoteService.wsdl">

      <message name="getPriceRequest">
            <part name="isbn" type="xsd:string" />
      </message>
      <message name="getPriceResponse">
            <part name="return" type="xsd:float" />
      </message>

      <portType name="BNQuotePortType">
            <operation name="getPrice">
               <input message="tns:getPriceRequest" name="getPrice" />
               <output message="tns:getPriceResponse" name="getPriceResponse" />
            </operation>
       </portType>

       <binding name="BNQuoteBinding" type="tns:BNQuotePortType">
            <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
            <operation name="getPrice">
               <soap:operation soapAction="" />
                 <input name="getPrice">
                 <soap:body use="encoded" namespace="urn:xmethods-BNPriceCheck"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
                  </input>
                  <output name="getPriceResponse">
                  <soap:body use="encoded" namespace="urn:xmethods-BNPriceCheck"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
                  </output>
            </operation>
       </binding>

        <service name="BNQuoteService">
          <documentation>Returns price of a book given an ISBN
                number</documentation>
          <port name="BNQuotePort" binding="tns:BNQuoteBinding">
             <soap:address
                location="http://services.xmethods.net:80/soap/servlet/rpcrouter" />
          </port>
        </service>
</definitions>
```

Exemplo 2.2 – Documento WSDL

No Exemplo 2.2, temos um exemplo de um documento WSDL. Ele contém apenas uma operação, "getPrice" (especificada no elemento "*portType*" de nome "BNQuotePortType"), que dado um número ISBN de um livro, recupera o seu valor

após a pesquisa em alguns *sites* de venda. A definição das mensagens para a execução da operação é feita no elemento "*message*". O único elemento "*binding*" presente indica que o protocolo de comunicação suportado por este serviço é o SOAP, mostrando detalhes sobre como devem ser formadas as mensagens a serem transmitidas. Finalmente, o elemento "*port*" representa a única implementação disponível desse serviço, que pode ser acessado através da URI identificada no elemento "*soap:address*".

2.3.3 – Descrição de Composições de Serviços

A automação de processos de negócios baseados em serviços Web requer uma notação para a especificação da troca de mensagens entre os serviços participantes. Dentro dessa notação, devem estar presentes algumas características, como a definição de quais serviços são obrigatórios e quais são opcionais, quais serviços são dependentes um do outro e em que ordem, qual caminho de execução seguir dado o sucesso ou insucesso na execução de um serviço, dentre outras.

Em (FLORESCU *et al.*, 2002) são descritas algumas características que devem estar presentes numa linguagem para composição de serviços, como :

- Concordância com os padrões da W3C: A linguagem deve se basear em padrões como XML Schema (FALLSIDE, 2001), WSDL (CHRISTENSEN *et al.*, 2001), entre outros.
- Processos de Negócios: A linguagem deve oferecer construções que facilitem a implementação de processos de negócios e a comunicação entre vários serviços.
- Composição: A linguagem deve permitir a criação de serviços de mais alto nível a partir da composição de serviços mais simples. A composição de serviços deve ser feita com um baixo acoplamento, de forma que se mantenha a máxima independência possível entre os serviços.
- Transações: A linguagem deve oferecer construtores que possibilitem a especificação de sequências de ações que devem ser executadas de uma forma isolada, garantindo sua atomicidade, ou seja: deve ser capaz de especificar procedimentos transacionais.

Na descrição de processos de negócios, especialmente em ambientes fracamente acoplados como a Internet, existem muitas questões a serem consideradas. A principal delas, é que ao contrário das arquiteturas de fluxo de trabalho tradicionais, não podemos assumir a existência de um serviço central que gerencie o processo de negócio entre as organizações participantes. Cada organização deve oferecer um ou mais serviços e a coordenação desses serviços ocorre implicitamente através dos resultados das operações executadas sobre eles. Através dessas chamadas de operações, são definidos caminhos alternativos no fluxo de trabalho, podendo ser chamadas diferentes operações de diferentes serviços. Concluindo, a descrição de um processo deve mostrar não apenas o comportamento de cada serviço participante, mas também, como esses comportamentos se relacionam para alcançar o objetivo desejado do processo.

Uma linguagem que descreva as características de processos de negócios de uma forma completa e estruturada é chamada de uma linguagem para a composição de serviços. Várias propostas desse tipo de linguagem foram feitas como a WSFL (*Web Services Flow Language*) (LEYMANN, 2001) e a XLANG (THATTE, 2001). A linguagem deve permitir um isolamento entre a forma como os serviços se comunicam, através de chamadas às suas operações, e a forma como estes são implementados.

Na tentativa de se chegar a um padrão para uma linguagem de composição de serviços, foi proposta a linguagem BPEL4WS (*Business Process Execution Language for Web Services*) (CURBERA *et al.*, 2002), que une características das linguagens XLANG e WSFL, tornando-se mais completa que as anteriores.

A linguagem BPFEL4WS define um modelo e uma gramática para a descrição do comportamento de um processo de negócios, baseando-se nas interações entre os participantes desse processo. A interação entre cada participante ocorre através de interfaces de serviços Web e a estrutura do relacionamento entre essas interfaces é encapsulada em uma estrutura denominada "*service link*". Um processo de negócios define como as múltiplas interações entre serviços são coordenadas para se alcançar um determinado objetivo, bem como a lógica necessária para se fazer essa coordenação.

O Exemplo 2.3 mostra a definição de um processo de negócios utilizando a linguagem BPEL4WS. Neste exemplo, um serviço Web cliente envia uma requisição

de um empréstimo bancário para um serviço Web de uma instituição financeira e este responde se este empréstimo foi aprovado ou não. Para simplificar a visualização do exemplo, foi omitida a descrição (documento WSDL) tanto do serviço cliente quanto do serviço da instituição financeira , bem como os *namespaces* utilizados na definição do processo de negócios.

```
<process>
        <partners>
                <partner        name="costumer"
                                serviceLinkType:"tns:loanApproveLinkType"
                                myRole="approver />
                <partner        name="approver"
                                serviceLinkType="tns:loanApprovalLinkType"
                                partnerRole="approver" />
        </partners>
        <containers>
                <container      name="request"
                                messagetype="lodef:CreditInformationMessage"/>
                <container      name="approvalInfo" messageType="apns:approvalMessage"/>
        </containers>
        <sequence>
                <receive        name="receive1"                 partner="costumer"
                                portType="apns:loanApprovalPT"
                                operation="approve"             container="request">
                </receive>
                <invoke         name="invokeapprover"           partner="approver"
                                portType="apns:loanApprovalPT"
                                operation="approve"
                                inputContainer="request"
                                outputContainer="approvalInfo">
                </invoke>
                <reply          name="reply"                    partner="costumer"
                                operation="approve"             container="approvalInfo">
                </reply>
        </sequence>
</process>
```

Exemplo 2.3 – Definição de um processo de negócio utilizando a linguagem BPEL4WS

No exemplo, o elemento "*partners*" define os participantes do processo de negócios (o serviço cliente e o serviço da instituição financeira). O elemento "*containers*" define os repositórios de dados que irão armazenar as mensagens que serão trocadas entre os serviços. Após esses dois elementos, tem início a definição do processo em si, onde temos três atividades sendo executadas seqüencialmente (definição feita através do elemento "*sequence*"): primeiramente, a aplicação cliente recebe uma mensagem, através da operação "*aprove*", solicitando um crédito. Os dados dessa mensagem são armazenados no repositório de nome "*request*". Após o recebimento dessa mensagem, é invocada a operação "*approve*" do serviço Web da instituição financeira (elemento "*invoke*"), com o conteúdo do repositório "*request*"

sendo utilizado como parâmetro de entrada. O repositório "*approvalInfo*" irá armazenar o resultado da execução desse serviço. Finalmente, o serviço cliente é notificado da execução do serviço da instituição financeira, mostrado através do elemento "*reply*".

2.3.4 – Publicação e Descoberta de Serviços

Um catálogo de serviços deve permitir a publicação e recuperação de informações sobre serviços conhecidos, bem como a descoberta de novos serviços através de descrições informadas por usuários. A especificação UDDI (*Universal Description Discovery and Integration Specification*) (UDDI.org, 2000), desenvolvida em conjunto por diversas organizações, define estruturas de dados para a descrição e classificação de serviços, bem como uma interface baseada no protocolo SOAP para permitir o acesso a essas informações.

O principal componente do projeto UDDI é conhecido como registro de negócio (*business registration*), um arquivo XML utilizado para descrever uma entidade de negócios e seus serviços na Web. Nele, temos tanto informações sobre a empresa que está oferecendo o serviço, como informações técnicas do serviço em si. A Figura 2.2 mostra o diagrama de classes, representados as estruturas do UDDI.

As estruturas de dados no projeto UDDI são descritas através de esquemas XML. Podemos identificar quatro estruturas principais para o tratamento de serviços :

- *businessEntity*: Contém informações sobre a organização que publicou informações sobre um serviço.
- *businessService*: Contém informações descritivas e conceituais sobre uma família de serviços em particular.
- *bindingTemplate*: Contém informações técnicas sobre como acessar um serviço (sua URL) e sobre suas construções.
- *tModel* : Contém as descrições de um serviço. Geralmente, esta estrutura é utilizada para armazenar também, a URL do documento WSDL que descreve o serviço, através de um subelemento de nome *overviewURL*.

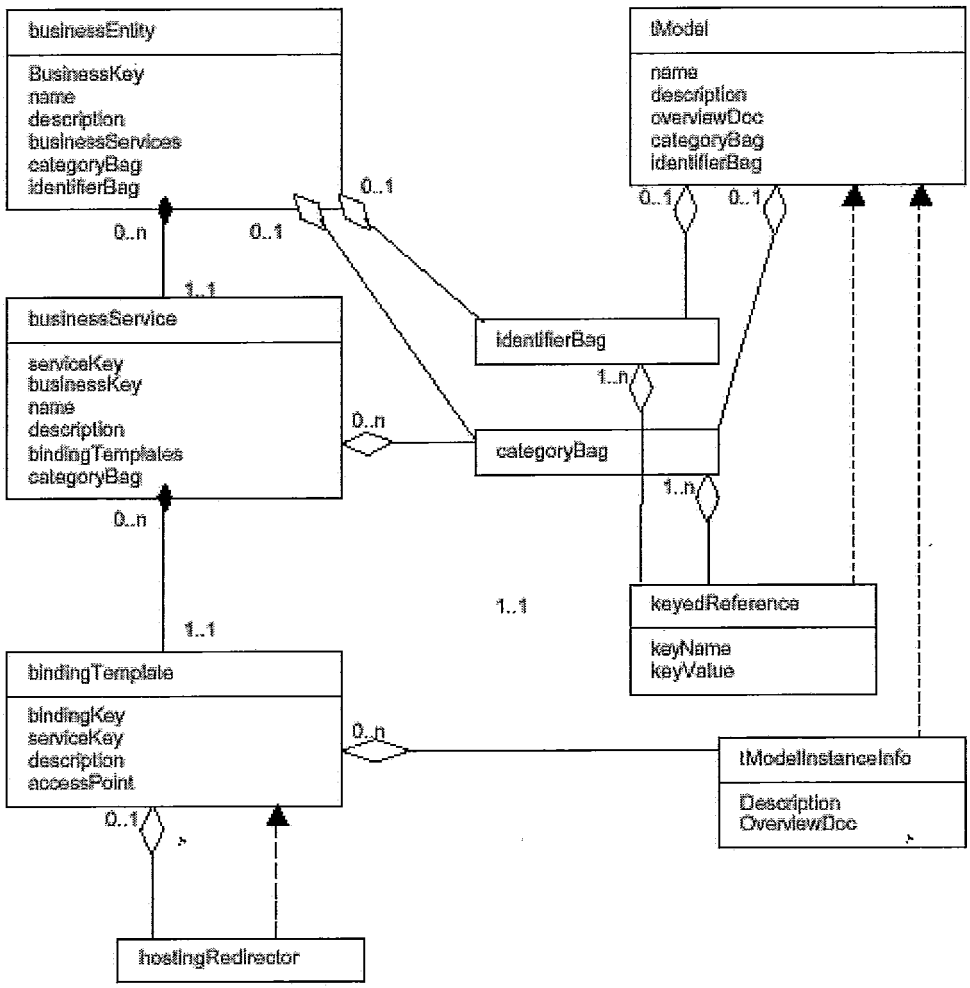


Figura 2.2 – Diagrama de classes contendo as estruturas do UDDI (UDDI.org , 2000)

Um exemplo de estrutura *tModel* pode ser vista no Exemplo 2.4. No exemplo, além das definições como a descrição do conteúdo do elemento *tModel*, temos a definição do endereço para o documento WSDL que descreve o serviço, especificado através do elemento *overviewURL*.

```
<tModel authorizedName = "..." operator = "..." tModelKey="...">
        <name> StockQuote Service </name>
        <description xml:lang="en">
                WSDL description of a standard stock quote service interface
        </description>
        <overviewDoc>
                <description xml:lang="en">
                        WSDL source document
                </description>
                <overview URL>
                        http://stockquote-definitions/stq.wsdl
                </overviewURL>
        </overviewDoc>
        <categoryBag>
                <keyedReference
                        tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
```

```
                    keyName="uddi-org:types"
                    keyValue="wsdlSpec"/>
        < /categoryBag>
</tModel>
```

Exemplo 2.4 – Estrutura tModel em um registro

A maioria das plataformas comerciais para a construção e utilização de serviços Web disponíveis contém classes e funções que facilitam a interação com as estruturas de dados da especificação UDDI. Apesar de ainda não existir um padrão "de fato" para o mapeamento das estruturas de um documento WSDL para o UDDI, o modelo baseado na publicação de interfaces como elementos "tModel" e de implementações de um serviço como elementos "bindingTemplate" é o mais utilizado (BRITTENHAM *et. al*, 2002 , UDDI.org, 2000).

Basicamente, uma interface de um serviço é publicada como um elemento "tModel", contendo um subelemento "overviewURL" que faz referência ao documento WSDL que contém a definição dessa interface. Geralmente, também é associado a um elemento "tModel" um protocolo de comunicação específico que deve ser utilizado na implementação dessa interface. Um elemento "bindingTemplate" do UDDI é criado para cada implementação dessa interface, com o subelemento "acessPoint" fazendo referência a uma URL pela qual aquela implementação pode ser acessada. Finalmente, um elemento "businessService" é criado, de forma a agrupar todas as interfaces de um serviço e suas respectivas implementações.

Apesar de incentivar a homogeneização de interfaces, acreditamos que tal modelo de mapeamento de um documento WSDL para o UDDI ainda não é ideal. O que tem sido notado atualmente é a geração de um grande número de interfaces distintas para serviços de funcionalidade equivalente, com cada interface possuindo implementações apenas por parte da própria empresa que desenvolveu aquela interface. Além disso, a busca por uma interface já existente que tenha a mesma funcionalidade de uma que estejamos interessados em implementar não é uma tarefa simples, dada a dificuldade de se expressar semanticamente esse tipo de consulta no UDDI.

Conhecidos os protocolos e linguagens que compõem uma arquitetura para a execução de serviços na Web, mostraremos no próximo capítulo os detalhes da arquitetura do *WebTransact*, uma plataforma para a execução e composição de serviços Web que possui entre suas características, o gerenciamento de transações.

3 – A Plataforma *WebTransact*

3.1 – Introdução

A maioria das plataformas comerciais com suporte para a utilização de serviços Web existentes atualmente (Microsoft .NET Framework (MICROSOFT CORP. 2000), Sun One (SUN MICROSYSTEMS, 2002), IBM Web Services Toolkit (IBM CORP. 2002), entre outras) contém ferramentas que auxiliam a criação de novos serviços, bem como a utilização de serviços já existentes na Web. O processo de publicação e descoberta de novos serviços utilizando o registro UDDI (UDDI.org, 2000), também é bastante facilitado.

Entretanto, essas plataformas oferecem pouco ou nenhum auxílio para a criação de aplicações baseadas na composição de serviços Web. Características como definições de fluxos de trabalho, atividades, garantias de qualidade de serviço e gerenciamento de transações são ainda carentes nessas plataformas.

De forma a atender aos requisitos necessários para facilitar a composição de serviços, algumas plataformas foram propostas na literatura. Em (TOSIC *et al.*,2001), é proposta a arquitetura DAMSC(*Dynamically Adaptable and Manageable Service Compositions*), que tem como principal característica a possibilidade de definição de múltiplas classes de serviços, contendo diferentes critérios de qualidade oferecido às aplicações clientes. Já em (KEIDL *et al.*, 2002A , KEIDL *et al.*, 2002B), é proposta uma plataforma conhecida como ServiceGlobe, que além da construção e utilização de serviços, permite a alocação dinâmica de serviços Web em computadores espalhados pela rede.

Em (PIRES, 2002, PIRES *et al.* 2002, PIRES *et al.*, 2003), é definida uma plataforma para a execução de serviços Web conhecida como *WebTransact*. Entre suas principais características, podemos citar o gerenciamento de transações (onde para cada serviço, pode estar associado um comportamento transacional, descrito em uma linguagem conhecida como WSTL (*Web Services Transaction Language*)) e a agregação de serviços de funcionalidade equivalente em uma mesma classe de serviço, implementada como um mediador nesta plataforma.

Apesar de permitirem a execução dinâmica de serviços com uma mesma interface (*ServiceGlobe*, *DAMSC*) e com interfaces distintas (*WebTransact*, dada a existência de um mecanismo de mapeamento da interface de um mediador para a interface de um serviço remoto existente), nenhuma das plataformas existentes contempla um modelo de execução que, baseado nos critérios de qualidade e parâmetros de custo de um serviço Web, decida qual serviço deve ser executado num determinado instante de tempo. Basicamente, nessas arquiteturas, uma das portas (implementações) de um serviço é escolhida para execução e no caso de uma falha, outras portas são invocadas até que a execução do serviço seja finalizada com sucesso. Esse tipo de execução não leva em consideração a possibilidade da seleção dinâmica de serviços baseada em critérios de qualidade de um serviço.

Dada a facilidade de se representar a agregação de serviços Web de funcionalidade equivalente na plataforma *WebTransact* (onde estes serviços são agrupados por um mesmo mediador), esta plataforma foi escolhida para a implementação do modelo de execução de serviços apresentado nesta dissertação, conhecido como WebTransact-EM. Além disso, a *WebTransact* também oferece um mecanismo para o gerenciamento de transações, garantindo a confiabilidade das composições de serviços efetuadas nesta plataforma, livrando nosso modelo de ter de se preocupar com este gerenciamento e fazendo com que o comportamento transacional de um serviço Web possa ser mais um critério de qualidade aplicável em nosso modelo.

Nas próximas seções, discutiremos as principais características da plataforma *WebTransact.* Sua implementação, que faz parte do escopo desta dissertação, é apresentada no Capítulo 5.

3.2 – Mediadores e Serviços Web

Intuitivamente, um mediador é um programa que acessa e integra múltiplos bancos de dados ou pacotes de software (WIEDERHOLD, 1995, ADALI *et al.*, 1996). Sistemas baseados na utilização de mediadores vêm sendo largamente adotados nos últimos anos, como na implementação de sistemas de bancos de dados heterogêneos. Neste tipo de sistema, mediadores são responsáveis por fazer a integração de múltiplas fontes de dados distintas, oferecendo uma visão unificada dessas fontes para um usuário externo. Como exemplos de sistemas de bancos de dados

heterogêneos, podemos citar: HERMES (ADALI *et al.*, 1996), GARLIC (ROTH *et al.*, 1999), DISCO (TOMASIC *et al.*, 1998), entre outros.

Podemos estender o conceito de mediadores para a construção de uma plataforma para a execução de serviços Web. Nesse caso, mediadores seriam responsáveis pela integração de serviços de funcionalidade equivalente (com serviços de validação de cartões de crédito, obtenção da temperatura ambiente, entre outros), oferecendo ainda, facilidades para a composição de serviços e o gerenciamento de transações.

Em (PIRES, 2002), a composição de serviços é definida como uma aplicação composta de serviços Web autônomos, conectados por várias redes de comunicação especializadas, sendo coordenados por um mediador responsável pelo gerenciamento de transações. Aplicações podem ser vistas como especificações de programas, construídas a partir de serviços Web. Elas descrevem as interações entre os serviços participantes e o comportamento transacional esperado por parte da aplicação.

Essencialmente, a mediação de transações distribuídas em ambientes para a composição de serviços consiste da escolha dos serviços Web disponíveis de acordo com o fluxo de controle da aplicação e com o comportamento transacional de cada serviço, de forma a garantir que a aplicação irá atingir um estado consistente quando de seu término.

Na próxima seção, serão mostradas algumas características de serviços compostos, bem como os possíveis comportamentos transacionais destes serviços, na plataforma *WebTransact*.

3.3 – Comportamento Transacional de Serviços Web na Plataforma *WebTransact*

Algumas características de sistemas para a composição de serviços influenciam o modelo de transações distribuídas destes ambientes. Entre tais características, podemos citar :

- Atomicidade das Aplicações: Algumas vezes, não é possível obtermos o conceito tradicional de atomicidade na execução de aplicações (ou a aplicação

toda é executada com sucesso ou nenhuma de suas atividades é considerada). Se a especificação da aplicação oferece caminhos de execução alternativos que tratam de possíveis falhas, é possível evitar que tenhamos que finalizar toda a aplicação, desfazendo suas atividades, quando uma falha ocorrer. Portanto, a linguagem utilizada para a descrição do controle de fluxo de aplicações necessita de construções que permitam uma correta e precisa definição de caminhos alternativos, quando da falha na execução de um serviço.

- Suporte Transacional por parte de Provedores de Serviços: Como provedores de serviços Web são independentes e autônomos, não podemos utilizar tais provedores para o gerenciamento de transações. Essa funcionalidade pode não estar presente, devido ao fato de estarmos tratando de serviços executados dentro de sistemas arbitrários, que podem não possuir suporte a transações, ou não exportar interfaces que permitam esse gerenciamento por parte de clientes externos. Mesmo que esse tipo de interface fosse oferecido, caso as utilizássemos, estaríamos bloqueando recursos desses provedores até o término da execução da aplicação. Para aplicações executando na Web, com potencialmente milhares de usuários acessando-as concorrentemente, o bloqueio de recursos nos provedores (através de protocolos como o de consolidação em duas fases – *two-phase commit* (GRAY, 1978, LAMPSON e STURGIS, 1976), por exemplo), não seria uma solução eficiente. Dessa forma, serviços Web precisam ser executados como uma transação única, onde após serem finalizados, seus resultados terão se tornados duráveis.

Para contemplar as características e requisitos da execução de transações sobre serviços Web, a WebTransact define quatro tipos de comportamento transacional de serviços, apresentando ainda os estados de execução válidos para cada tipo de comportamento e especificando uma linguagem de explicitação para esses comportamentos transacionais, a WSTL (*Web Services Transaction Language*).

Os quatro diferentes tipos de comportamento transacional definidos para um serviço na *WebTransact* são:

- Compensável: Um serviço remoto é compensável se, após sua execução, seus efeitos podem ser desfeitos através da execução de um outro serviço.

- *Retriable*: Um serviço remoto é dito *retriable* se podemos garantir que ele irá ser finalizado com sucesso após um conjunto finito de repetidas execuções.

- Pivô : Um serviço é dito pivô se ele não é compensável, nem do tipo *retriable*.

- Virtualmente Compensável: Serviços virtualmente compensáveis representam todos os serviços que suportam o protocolo padrão de consolidação em 2 fases (*two-phase commit*) (GRAY, 1978, LAMPSON e STURGIS, 1976), sendo tratados da mesma forma que os serviços compensáveis. Entretanto, seus efeitos não são compensados através da execução de um outro serviço. Ao invés disso, eles aguardam em um estado intermediário (*prepare-to-commit state*) até que a aplicação atinja um estado onde seja segura a consolidação do serviço.

A Figura 3.1 mostra os diagramas de transição de estados para cada um dos quatro tipos de comportamento transacional :

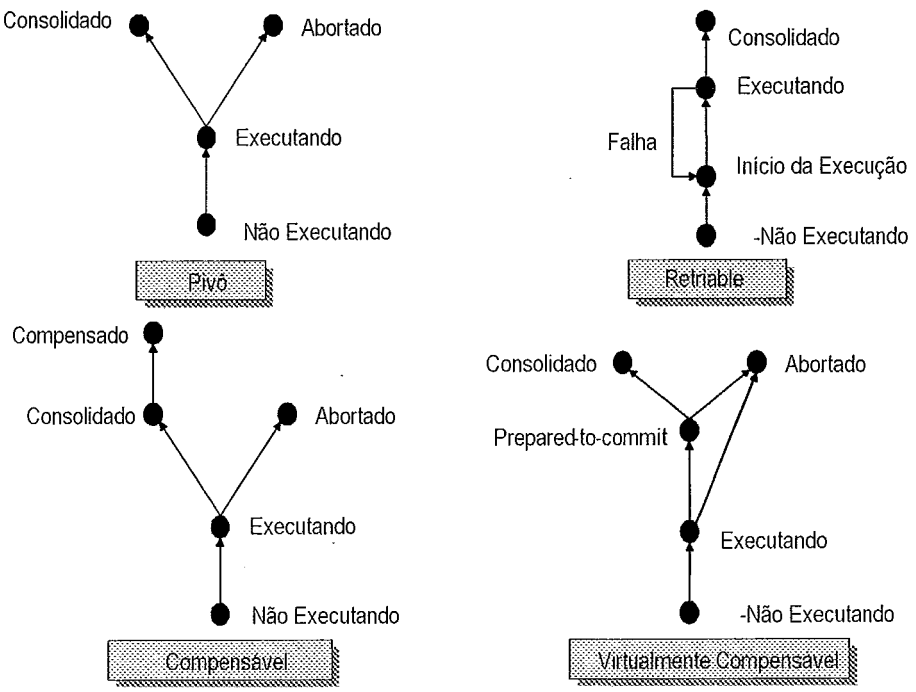


Figura 3.1 – Diagrama de transição de estados para os quatro possíveis comportamentos transacionais (PIRES,2002)

Um serviço pivô apresenta apenas dois tipos de estado final possíveis, *consolidado* e *abortado*, dada a incapacidade de desfazermos seus resultados. Um serviço do tipo *retriable*, dada a garantia de sucesso em sua execução após um

número finito de tentativas, não possui o estado final *abortado*, possuindo apenas o estado final *consolidado*. Serviços compensáveis, além dos estados finais *abortado* e *consolidado*, também apresentam um estado final *compensado*, obtido a partir de uma transição do estado *consolidado*, dado que podemos desfazer os resultados deste tipo de serviço. Finalmente, serviços virtualmente compensáveis possuem um estado intermediário denominado *prepared-to-commit*, onde dependendo da resposta dos demais serviços participantes de uma transação, podemos ter a transição para o estado final *abortado* ou *consolidado*.

O comportamento transacional de um serviço Web é expresso através de uma extensão da linguagem WSDL, conhecida como WSTL (*Web Services Transaction Language*), definida em (PIRES, 2002). Na linguagem WSTL, foi criado um elemento "*transactionDefinitions*", subelemento do elemento raiz de um documento WSDL (elemento "*definitions*"), que descreve o comportamento transacional das operações de um serviço.

Um elemento "*transactionDefinitions*" contém um conjunto de subelementos do tipo "*transactionBehavior*" que, para a operação identificada no atributo *operationName*, descreve o comportamento transacional desta operação em um atributo de nome *type*, que pode assumir quatro valores distintos : *pivot* (para serviços do tipo pivô), *retriable* (para serviços do tipo retriable), *compensable* (para serviços compensáveis) e *virtualcompensable* (para serviços virtualmente compensáveis).

As subseções seguintes apresentam exemplos de serviços dos quatro tipos de comportamento transacional definidos na *WebTransact*.

3.3.1 – Operações do Tipo Pivô ou *Retriable*

Dada uma definição de um serviço Web através de um documento WSDL, para operações do tipo pivô ou *retriable*, basta definirmos o nome da operação e seu comportamento transacional, conforme mostrado no Exemplo 3.1. No exemplo, a operação "*cancelReservation*" é do tipo *retriable*, enquanto a operação "*checkReservation*" é do tipo pivô.

```
<wstl:transactionDefinitions>
        <wstl:transactionBehavior operationName="absd:cancelReservation" type="retriable">
        </wstl:transactionBehavior>

        <wstl:transactionBehavior operationName="absd:checkReservation" type="pivot">
        </wstl:transactionBehavior>
</wstl:transactionDefinitions>
```

Exemplo 3.1 – Operações *Retriable* e Pivô.

3.3.2 – Operações Compensáveis

Para operações que são compensáveis, precisamos de informações sobre qual operação precisará ser executada (operação compensatória) para desfazer as ações destas operações (operação compensável). Essas informações estão armazenadas em um elemento do tipo "*activeAction*", subelemento de "*transactionBehavior*". Nesse elemento, estão especificadas as operações compensatórias que devem ser executadas para a compensação de um serviço, bem como os mapeamentos necessários entre os parâmetros da operação compensável e os parâmetros da operação compensatória relacionada. Esses mapeamentos estão descritos num elemento "*msgParamLink*", subelemento de "*activeAction*".

Existem ainda, algumas operações compensáveis que não necessitam da execução de nenhuma operação compensatória para desfazer seus efeitos. Esse tipo de operação não é explicitamente compensável: ao invés disso, seus efeitos possuem um tempo de vida pré-definido. Como exemplo, podemos citar um serviço de reservas de carros, que precisa de uma confirmação até um determinado período de tempo para homologação de uma reserva. Após esse período, caso nenhuma confirmação tenha sido enviada, a reserva é descartada.

Para representarmos operações compensáveis desta forma, utilizamos o elemento "*passiveAction*", também subelemento de "*transactionBehavior*". Cada elemento "*passiveAction*" possui um atributo *expiration*, que define o tempo a ser transcorrido para que uma operação expire, e um atributo *unit*, que identifica a unidade de tempo a ser levada em consideração neste processo (minutos, horas, dias, semanas, meses ou anos).

No Exemplo 3.2, temos um exemplo de uma operação compensável de nome *Reservation*. No exemplo, esta operação pode ser compensada pela operação

"cancelReservation", especificada no atributo *compensatoryOper*. Neste caso, o parâmetro de saída "reservationResult" (elemento *sourceParamLink*), da operação "Reservation", será utilizado como o parâmetro de entrada de nome "reservationCode" (*elemento targetParamLink*) da operação compensatória "cancelReservation".

```
<wstl:transactionDefinitions>
        <wstl:transactionBehavior operationName="absd:Reservation" type="compensable">
                <wstl:activeAction porttypeName="svc:reservationSoap"
                                                compensatoryOper="cancelReservation">
                        <wstl:paramLink>
                                <wstl:sourceParamLink msgname="reservationSoapOut"
                                param="absd:reservationResponse/@reservationResult" />
                                <wstl:targetParamLink name="cancelReservationSoapIn"
                                        param="absd:cancelReservation/@reservationCode" />
                        </wstl:paramLink>
                </wstl:activeAction>
        </wstl:transactionBehavior>
</wstl:transactionDefinitions>
```

Exemplo 3.2 – Operação Compensável

3.3.3 – Operações Virtualmente Compensáveis

Serviços virtualmente compensáveis representam todos os serviços que suportam o protocolo padrão de consolidação em duas fases (*two-phase commit protocol*) (GRAY, 1978, LAMPSON e STURGIS, 1976), sendo tratados da mesma forma que os serviços compensáveis.

Dessa forma, esses serviços precisam ser suportados por algum tipo de sistema de processamento de transações distribuídas que ofereça o protocolo de consolidação em duas fases. Neste protocolo, o processamento de transações distribuídas é dividido em duas fases, sob a supervisão de um coordenador ou gerente de transações. Numa primeira fase, o coordenador envia uma mensagem a cada um dos nós participantes da transação perguntando se eles "concordam" em consolidar a transação. Após ter recebido a resposta de todos os participantes, tem início a segunda fase: se todos os participantes tiverem concordado em consolidar a transação, o coordenador decide por uma consolidação global desta, enviando uma mensagem de consolidação (*commit*) a cada um dos participantes. Entretanto, se pelo menos um dos participantes não tiver concordado em consolidar a transação, o coodernador decide por abortá-la, enviando uma mensagem para cada um dos participantes solicitando que abortem a transação.

Quando da execução em paralelo de serviços Web de funcionalidade equivalente, apenas um dos serviços terá seus resultados consolidados, sendo denominado como serviço vencedor, enquanto os demais serão abortados compensados. Caso os serviços a terem seus resultados desfeitos sejam virtualmente compensáveis, um protocolo como o de consolidação em 2 fases deve ser utilizado para realizar essa compensação.

Assim, para que possamos implementar serviços Web virtualmente compensáveis (ou seja, que suportem o protocolo padrão de consolidação em duas fases), precisamos que três requisitos sejam atendidos (PIRES, 2002):

- Para cada operação compensável *op* de um serviço Web *S*, precisamos de um gerente de transações *GT* que ofereça suporte transacional para *op*.
- Para cada gerenciador de transações *GT*, deve existir um documento WSDL que defina a sua interface.
- Cada mediador M que agrega um serviço Web *S* deve suportar a interface exposta pelo gerenciador de transações GT de forma a coordenar transações que envolvam este gerente. A linguagem WSTL define uma interface padrão para o protocolo de consolidação em duas fases que deve ser suportada por todos os gerentes de transações que coordenam serviços Web virtualmente compensáveis.

Assim, quando definimos operações virtualmente compensáveis, precisamos que esta operação referencie um serviço que faça o papel de gerente de transações, de forma que este ofereça suporte transacional para essa operação. Na linguagem WSTL, uma operação virtualmente compensável é representada por um elemento "*tmSrv*", que utiliza um serviço gerente de transações através do atributo "*tmElemName*". Este atributo, por sua vez, referencia um determinado elemento "*tmElem*". Este elemento "*tmElem*" contém um atributo de nome "*tmMap*", que é responsável por explicitar o mapeamento da interface exposta pelo gerente de transações para a interface padrão de consolidação em duas fases que é oferecida pela linguagem WSTL.

No Exemplo 3.3, temos um exemplo de definição de operações virtualmente compensáveis, representando um serviço bancário. Temos três operações, "balance",

"deposit" e "withdraw", todas elas virtualmente compensáveis. O serviço gerente que coordena as transações em que estas operações participam é o mesmo, e está especificado pelo elemento *tmElem* de nome "tmSoapMap". Este elemento, através do atributo *tmPort*, indica a porta pela qual devemos acessar o serviço desse gerente de transações, enquanto o atributo *tmMap* referencia o elemento que fará o mapeamento da interface exposta pelo gerente de transações para a interface padrão de consolidação em duas fases oferecida pela linguagem WSTL.

```
<wstl:trasactionDefinitions>
    <wstl:transactionBehavior type="virtualCompensable"
                                    operationName="tns:balance">
      <wstl:tmSrv> <wstl:tmRef tmElemName="tns:tmSoapMap"/> </wstl:tmSrv>
    </wstl:transactionBehavior>
    <wstl:transactionBehavior type="virtualCompensable"
                                    operationName="tns:deposit">
      <wstl:tmSrv> <wstl:tmRef tmElemName="tns:tmSoapMap"/> </wstl:tmSrv>
    </wstl:transactionBehavior>
    <wstl:transactionBehavior type="virtualCompensable"
                                    operationName="tns:withdraw">
      <wstl:tmSrv> <wstl:tmRef tmElemName="tmSoapMap"/> </wstl:tmSrv>
    </wstl:transactionBehavior>
    <wstl:tmElem name="tmSoapMap" tmPort="tmSrv:TMSrvSoapPort"
                    tmMap="tmSrvMap:TMSrvSoapMap"/>
  </wstl:trasactionDefinitions>
</definitions>
```

Exemplo 3.3 – Operações Virtualmente Compensáveis

3.4 – Arquitetura da Plataforma *WebTransact*

A plataforma WebTransact viabiliza a composição de serviços Web através da adoção de uma arquitetura com múltiplas camadas formadas por vários componentes especializados. Os programas de aplicação interagem com *composições de serviços de mediador* escritas por desenvolvedores de composições. Tais composições são definidas através de padrões de interação transacionais envolvendo os serviços de mediador. *Serviços de mediador* fornecem uma interface homogênea de (múltiplos) serviços remotos semanticamente equivalentes. *Serviços remotos* encapsulam a interface de um serviço Web específico bem como o seu comportamento, fornecendo as informações de mapeamento necessárias para converter mensagens do formato heterogêneo de serviço Web para o formato do mediador. A Figura 3.2 ilustra esses componentes da arquitetura.

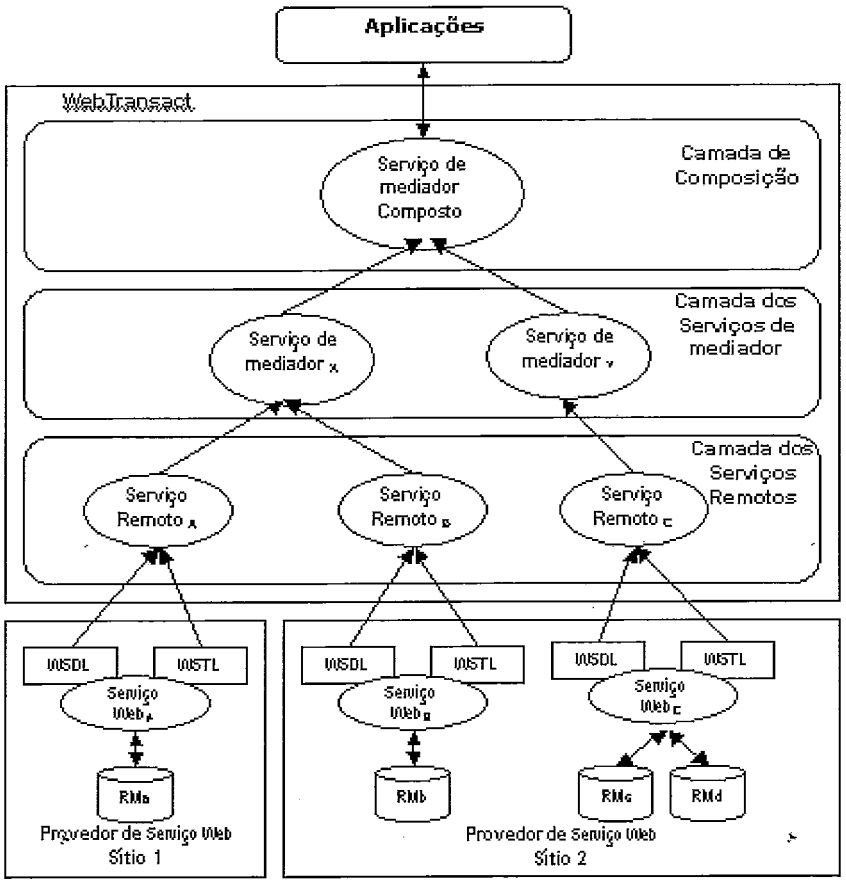


Figura 3.2 – Arquitetura da Plataforma WebTransact

Uma vez apresentados os componentes da arquitetura WebTransact, na próxima seção, mostraremos como estes interagem para a execução de uma composição formada por vários serviços.

3.4.1 – Modelo de Execução de Aplicações

Uma composição de serviços descreve a comunicação entre os serviços de mediadores participantes da execução de uma determinada tarefa. Ao invés de chamar um serviço Web diretamente, uma composição invoca uma operação de um mediador, que agrega um conjunto de serviços de funcionalidade equivalente, mas com comportamentos transacionais possivelmente distintos. Cabe, portanto, ao mediador, a escolha de qual serviço remoto deve ser executado em um determinado momento. Daí, a importância de um modelo de custo e execução que possa auxiliar o mediador nesta decisão.

As composições de serviços são definidas através das seguintes informações : i) um conjunto de serviços de mediadores; ii) um conjunto de dependências de

execução; iii) um conjunto de serviços de mediadores obrigatórios. Essas informações são mostradas a seguir:

- Conjunto de Serviços de Mediadores: Representa cada serviço de um mediador que participa na execução de uma tarefa definida na composição.

- Conjunto de Dependências de Execução: Define como os mediadores interagem durante a execução de uma composição. Uma dependência de execução é definida entre serviços de mediadores e está relacionada à ocorrência de eventos de inicialização e término deste tipo de serviços. Dependências de execução definem a ordem em que um serviço de mediador deve ser executado, isto é, definem o fluxo da aplicação. Uma dependência de execução pode ser especificada baseada no estado de execução de um ou mais serviços de mediadores (exemplo: quando a operação o1 de um mediador M1 terminar com sucesso, execute a operação o2 de M2), ou baseada em valores de saída gerados por um serviço de mediador.

- Conjunto de Serviços de Mediadores Obrigatórios → Especifica, para cada caminho de execução da composição, o conjunto de serviços de mediadores que precisam ser executados com sucesso de forma que a aplicação também termine com sucesso. Uma composição pode ser especificada através da agregação de serviços que são desejáveis, mas não essenciais, para que esta aplicação alcance seu objetivo. Logo, o conjunto de serviços de mediadores permite a distinção entre serviços *desejáveis* e *obrigatórios*, oferecendo uma maior flexibilidade na especificação de uma composição. Dessa forma, esta pode ter sua execução finalizada com sucesso mesmo que um subconjunto de seus serviços falhe, desde que esse subconjunto contenha apenas serviços não obrigatórios.

Das três camadas da arquitetura da *WebTransact*, foram implementadas a camadas dos mediadores e a camada de serviços remotos, necessárias para a implementação do modelo de execução de serviços Web (*WebTransact-EM*) discutido nesta dissrtação. A camada de composição de serviços, que tratará da especificação de aplicações baseadas na execução de múltiplos serviços, não será discutida.

Apesar de prever a execução de serviços Web de forma dinâmica, a WebTransact não dispõe de um modelo de execução baseado em critérios de

qualidade que faça a escolha de qual serviço executar. Dada a capacidade dessa arquitetura de agrupar serviços equivalentes através de um mediador e a existência de um mecanismo de gerenciamento de transações, escolhemos esta plataforma para implementarmos esse modelo de execução, tornando-a mais flexível.

3.5 – Outras Propostas de Linguagens para Gerenciamento de Transações

A linguagem WSTL (PIRES, 2002) não é a única que trata da descrição de aspectos transacionais de serviços Web. Na tentativa de se chegar a um padrão de linguagem para o gerenciamento de transações distribuídas, foram propostas as especificações WS-Coordination (*Web Services Coordination*) (CABRERA et. al, 2002a) e WS-Transaction (*Web Services Transaction*) (CABRERA et. al. , 2002b).

A especificação WS-Coordination define uma arquitetura extensível para a coordenação de atividades de aplicações distribuídas, utilizando um serviço coordenador e um conjunto de protocolos. Essa arquitetura permite que uma aplicação crie o contexto necessário para a propagação de uma atividade para outros serviços. A estrutura desse contexto bem como os requisitos necessários para sua propagação também fazem parte da especificação WS-Coordination.

Já a especificação WS-Transaction define modelos de coordenação, contendo os principais protocolos para o controle de transações. Esses modelos são utilizados em conjunto com os serviços coordenadores, definidos na especificação WS-Coordination, auxiliando no processo de gerenciamento de transações. Os dois modelos de coordenação definidos na especificação WS-Transaction são :

- Transação Atômica (*Atomic Transaction*): Utilizada para coordenar atividades tendo um período curto de duração e que são executadas dentro de um mesmo domínio. Esse modelo garante a atomicidade no processamento de transações : ou toda a atividade é finalizada com sucesso ou todos os seus resultados são desfeitos.

- Atividade de Negócios (*Business Activity*): Utilizada para coordenar atividades que possuem uma duração longa e que desejam aplicar regras de negócio para tratar possíveis exceções. A longa duração das atividades inibe a

retenção de recursos, deixando-os indisponíveis para outras aplicações. Ao invés disso, as ações são consolidadas assim que terminam e são feitas permanentes. No caso de uma possível falha da atividade, estas deverão ser compensadas.

Muitos dos conceitos presentes na linguagem WSTL também estão presentes nas especificações WS-Coordination e WS-Transaction, como a utilização de um serviço Web que funcione como coordenador para a execução de transações distribuídas e a utilização do conceito de compensação de atividades para evitar a retenção de recursos por um grande período de tempo. Entretanto, a definição de um comportamento transacional para cada operação de um serviço Web, feita na linguagem WSTL, torna mais simples o gerenciamento de transações que não utilizam um protocolo como o de consolidação em duas fases, pois estas não precisam se basear num serviço coordenador, sendo tarefa de uma plataforma (no caso da WSTL, o *WebTransact*) o gerenciamento desse tipo de transações. Como os conceitos utilizados em ambas as especificações são bastante semelhantes, acreditamos que estas possam coexistir sem nenhum problema de compatibilidade no caso de um serviço que utilize a linguagem WSTL queira se comunicar com outro que se baseie nas especificações WS-Coordination e WS-Transaction para o gerenciamento de transações distribuídas.

4 – Seleção Dinâmica de Serviços Web

4.1 - Introdução

Com a evolução da tecnologia de serviços Web, vários serviços encontram-se disponíveis para serem utilizados. Dada essa grande oferta de serviços, é muito comum que tenhamos serviços distintos, oferecidos por diferentes organizações, porém, com funcionalidades equivalentes.

Uma aplicação cliente poderia escolher qual desses serviços utilizar baseada em critérios como o tempo de resposta ou o custo monetário para execução desse serviço. Para que este tipo de escolha seja possível, serviços de funcionalidade equivalente devem ser agrupados, de forma que baste à aplicação cliente informar qual grupo de serviços ela deseja executar e quais as suas restrições (ou seja, quais critérios de qualidade uma aplicação cliente deseja que um serviço Web possua). Dessa forma, ficaria sob a responsabilidade de um otimizador a escolha de qual serviço deve ser executado num determinado instante de tempo. Em outras palavras, ficaria a cargo de um otimizador a seleção dinâmica de serviços Web.

Essa seleção dinâmica de serviços pode ser vista como uma etapa adicional dentro do processo de composição de serviços Web, porém, transparente ao usuário, feita por um otimizador. Ao invés de explicitarmos um serviço específico numa composição, indicaríamos apenas uma classe de serviços, representando a funcionalidade que desejamos executar. Seria tarefa do otimizador a seleção de um ou mais serviços remotos que devem ser executados de forma a atender a requisição efetuada pela aplicação cliente. Esse processo de seleção seria então, repetido para cada classe de serviço presente numa composição.

Para que um otimizador possa fazer essa seleção de uma forma adequada, três requisitos devem ser atendidos:

- Parâmetros de custo aplicáveis a serviços Web e suas operações devem ser definidos;
- Um modelo de execução, que leve em conta esses parâmetros deve ser elaborado, de forma que serviços possam ser escolhidos baseados nesses parâmetros;

- Uma plataforma que permita o agrupamento de serviços em classes, e a especificação de restrições de custo e qualidade sob estas classes deve ser utilizada como base para implantação de tal modelo.

A princípio, o problema da seleção dinâmica de serviços Web (qual serviço executar num determinado instante de tempo), se assemelha ao problema da otimização de consultas em bancos de dados heterogêneos (DU *et al.*, 1992, HASS *et al.*, 1997, ROTH *et al.*, 1999), dado o caráter autônomo e distribuído das fontes de dados. Entretanto, mediadores em banco de dados heterogêneos são utilizados para o agrupamento de dados, possivelmente oferecendo uma visão homogeneizada destes através de um esquema global, além de tratar das diferenças na forma de representação desses dados (dados relacionais, orientados a objetos, semi-estruturados, entre outros). No caso de serviços Web, mediadores são utilizados no agrupamento de serviços de funcionalidade equivalente, oferecendo uma visão homogeneizada destes através de uma outra interface, sendo tarefa do mediador o mapeamento necessário entre essas interfaces.

Além disso, algumas outras diferenças existem entre essas duas abordagens: uma consulta pode ser decomposta em vários operadores (junções, seleções, projeções), que podem ser executados em nós distintos na rede, gerando um grande espaço de soluções e uma grande possibilidade de paralelismo. Já uma operação de um serviço não pode ser tão facilmente decomposta, pois depende da lógica de cada operação e de sua implementação. Dessa forma, uma operação não pode ser decomposta por um otimizador em operações menores. Além disso, a estimativa de custos de operadores conhecidos como o operador de junção ou o de seleção pode ser realizada de forma mais simples do que a estimativa de custo de uma operação arbitrária de um serviço, cujo custo pode variar dependendo dos parâmetros de entrada.

Na discussão sobre os parâmetros do modelo de custo, também serão levados em consideração alguns critérios de qualidade de serviço (QoS) aplicáveis a serviços e suas operações. Inicialmente, o termo "qualidade de serviço" se referia a algumas características de desempenho que uma rede deveria possuir, sem incluir qualquer tipo de controle ou interferência do usuário final (CAMPBELL *et al.*, 1994). Porém, a especificação de critérios de qualidade se generalizou e atualmente, abrange várias áreas da computação distribuída, permitindo agora, a definição de critérios de

qualidade garantidos ao usuário, possibilitando que um cliente especifique os níveis de qualidade que deseja na execução de uma aplicação.

Alguns trabalhos na literatura já tratam da definição de critérios de qualidade para serviços Web (MANI e NAGARAJAN, 2002, NOTTINGHAM, 2001, SHETH *et al.*, 2002). Entretanto, eles não utilizam esses critérios na elaboração de um modelo de custo. Podemos utilizar esse conceito para permitir que uma aplicação cliente defina quais critérios de qualidade são desejáveis na execução de um serviço, de forma que seja feita uma seleção inicial baseada nesses critérios, dos serviços candidatos a execução (serviços que atendem a critérios pré-estabelecidos selecionados por aplicações clientes e que podem ser escolhidos por um escalonador para serem executados).

O restante deste capítulo está dividido da seguinte forma. Na Seção 4.2, são identificados os parâmetros de custo e critérios de qualidade aplicáveis a serviços Web e suas operações. Na Seção 4.3, é feita uma introdução ao conceito de modelos de custo e baseado nos parâmetros discutidos na seção 4.2, é apresentado um modelo para o cálculo do tempo total de execução de um serviço Web. Na Seção 4.4, são discutidas as possibilidades para a execução paralela de serviços, de forma que possamos obter o máximo de desempenho em nossa execução. Finalmente, na Seção 4.5 é mostrada uma gramática XML para a representação dos parâmetros de custo e critérios de qualidade discutidos anteriormente. A representação dessa gramática no registro UDDI é discutida na Seção 4.6.

4.2 – Parâmetros e Critérios de Qualidade Utilizados no Modelo de Custo

4.2.1 – Introdução

Nesta seção, são mostrados os critérios de qualidade e parâmetros de custo aplicáveis a serviços Web e suas operações. Entretanto, antes de apresentarmos estes critérios, falaremos da definição de serviço Web feita através da linguagem WSDL e do conceito de serviço utilizado nesta dissertação.

Na linguagem WSDL, um serviço (representado pelo elemento *"service"*) é mostrado como um conjunto de portas (elementos *"port"*) que representam uma

associação entre um endereço físico e um elemento *"binding"* (uma implementação de uma interface utilizando um determinado protocolo de comunicação). Utilizando esse conceito, podemos visualizar um serviço como sendo um conjunto de diferentes implementações de uma ou mais interfaces.

Neste trabalho, identificaremos um serviço como um sendo uma porta (um elemento *"port"*) de um documento WSDL. Tal diferenciação se faz necessária devido ao fato de uma porta referenciar um endereço físico (um computador na rede), representando uma implementação, e diferentes implementações podem apresentar custos e critérios de qualidade distintos, que irão afetar a seleção dinâmica de serviços e conseqüentemente o modelo de custo proposto. Portanto, quando dizemos que iremos executar um serviço, na verdade estaremos executando uma implementação deste, referenciada por uma porta no documento WSDL.

Definido o conceito de serviço, podemos agora mostrar como será elaborado o modelo de custo sendo proposto, inicialmente definindo os conjuntos S, de serviços e O, de operações, que serão utilizados no restante deste capítulo.

Seja $S = \{S_0 , S_1 , S_2 , \ldots , S_{n-1}\}$ o conjunto dos *n* serviços de funcionalidade equivalente agrupados por um mediador M. Para cada serviço S_i, considere também o conjunto $O = \{O_0 , O_1 , O_2 , \ldots , O_{m-1}\}$, conjunto das *m* operações oferecidas por cada serviço S_i .

Podemos definir $C_T^{i,j}$ como o custo total de execução de uma determinada operação $O^{i,j}$ de um serviço S_i. De uma forma simplificada, podemos definir $C_T^{i,j}$ conforme a Equação 1:

$$C_T^{i,j} = C_I^i + C_E^{i,j} + C_R^{i,j} \qquad \text{(Eq. 1)}$$

onde C_I^i é o custo de inicialização do serviço S_i, $C_E^{i,j}$ é o custo de execução da operação $O^{i,j}$ do serviço S_i e $C_R^{i,j}$ é o custo de transmissão envolvido na execução da operação $O^{i,j}$ do serviço S_i.

Na próxima seção, iremos definir critérios de qualidade aplicáveis tanto ao conjunto de serviços S quanto ao conjunto de operações O, de forma a identificarmos os parâmetros necessários para calcularmos os custos de inicialização, transmissão e execução de uma determinada operação de um serviço.

4.2.2 – Parâmetros e Critérios de Qualidade Aplicáveis a um Serviço

a) *Disponibilidade*

A disponibilidade de um serviço, denotada por $D(S_i)$, corresponde à porcentagem de tempo em que um serviço S está disponível, considerando-se um intervalo de tempo T. Ou seja,

$$D(S_i) = \frac{Q}{T}, \quad Q \leq T \qquad \text{(Eq. 2)}$$

onde T é um intervalo de tempo e Q é o intervalo de tempo, contido em T em que o serviço S esteve disponível.

b) *Confiabilidade*

Um serviço pode estar disponível, mas pode não ser capaz de atender a uma determinada requisição por problemas de congestionamento de rede ou por problemas em sua implementação. Definiremos confiabilidade de um serviço ($R(S_i)$) como sendo a razão entre o número de requisições atendidas por este serviço sobre o número de requisições efetuadas. Ou seja,

$$R(S_i) = \frac{N_{atendidas}}{N_{efetuadas}} \qquad \text{(Eq. 3)}$$

onde $N_{atendidas}$ corresponde ao número de requisições atendidas e $N_{efetuadas}$ ao número total de requisições efetuadas.

c) *Custo de Inicialização*

O custo de inicialização de um serviço S_i, denotado por $I(S_i)$, corresponde ao intervalo de tempo máximo para que um determinado serviço seja inicializado. Ele inclui o tempo necessário para a autenticação de usuários, verificação de permissões e inicialização dos itens de hardware e software necessários.

d) *Informações de Rede*

Quando estamos tratando de uma rede remota (como a Internet, por exemplo), o custo de comunicação é, em geral, o fator dominante na determinação do custo de execução de um serviço.

Para que possamos calcular de forma eficiente o custo de transmissão envolvido na execução de um serviço S, precisamos obter algumas informações a respeito da rede onde este serviço está hospedado.

Em (ÖZSU e VALDURIEZ, 1999), é apresentado um modelo de custo para sistemas de bancos de dados distribuídos. Nesse modelo, o custo de comunicação é representado através da fórmula da Equação 4:

$$C = (T_{MSG} * \#msgs) + (T_{TR} * \#bytes) \qquad \text{(Eq. 4)}$$

onde T_{MSG} é o custo fixo de inicialização de uma mensagem, #msgs a quantidade de mensagens, T_{TR} o tempo que se leva para transmitir uma unidade de dados de um *site* para outro e #bytes a quantidade de bytes a serem transmitidos. Usaremos esse custo de comunicação de dados em nosso modelo de custo, por ser esta uma fórmula clássica para o custo de transmissão em sistemas de banco de dados distribuídos, podendo ser facilmente adaptada para o tratamento de custos de transmissão de serviços Web.

e) *Segurança*

Serviços podem ser implementados de forma que apenas clientes previamente autorizados possam acessá-los. Para restringir o acesso a um serviço, precisamos de algum método para a autenticação de clientes, de forma que possamos decidir se um cliente será autorizado ou não a utilizar determinado serviço.

Na maioria das plataformas comerciais que suportam serviços Web, a segurança desses serviços não difere muito da segurança que é oferecida para páginas Web. (ou seja, o acesso pode ser dado para determinados endereços IP ou obtido através da digitação de uma combinação de usuário/senha válida). Isso pode ser visto como uma limitação corrente da tecnologia de serviços Web, pois estes diferem de simples páginas Web.

Para resolver esse problema, alguns trabalhos tratam de modelos de segurança específicos para serviços Web (HONDO *et al.*, 2002, WEBMETHODS, 2002), abordando requisitos de segurança necessários (como autenticação de usuários, controle de acesso, confidencialidade de dados), sugerindo a inclusão de mais uma camada nas atuais plataformas para tratar especificamente desses requisitos. Para compor essa nova camada de segurança, algumas propostas têm sido feitas, como a especificação WS-Security (ATKINSON *et al.*, 2002), que através da inclusão de cabeçalhos (*headers*) nas mensagens SOAP, contendo assinaturas e

métodos de encriptação, oferece mecanismos para a proteção da integridade e confidencialidade de mensagens, entre outros aspectos de segurança.

Nesta dissertação, estamos tratando da segurança como mais um critério de qualidade aplicável a um serviço Web. Sendo assim, estamos interessados apenas em saber se um determinado serviço apresenta algum mecanismo de segurança. Mais especificamente, estamos interessados em descobrir se um serviço oferece algum tipo de autenticação de usuários (como autenticação por usuário/senha ou através do uso de protocolos como o HTTPS) e algum método para garantir a confidencialidade dos dados (geralmente obtida através da utilização de protocolos de encriptação, como o SSL, SSH, entre outros). Uma descrição dos mecanismos de segurança utilizados por um serviço também poderia ser oferecida pelo provedor do serviço Web, de forma a facilitar a escolha por um determinado serviço baseada nos critérios de segurança apresentados.

4.2.3 – Parâmetros e Critérios de Qualidade Aplicáveis a uma Operação de um Serviço

a) *Prioridade*

A prioridade de uma operação $O^{i,j}$ de um serviço S_i, denotada por $P(O^{i,j})$, é definida pela Equação 5:

$$P(O^{i,j}) = x,\ 0 < x \leq 1. \qquad \text{(Eq. 5)}$$

De forma a simplificar cálculos e impedir a existência de valores muito dispersos de x, podemos definir alguns valores padrão de prioridade. Por exemplo: prioridade de tempo real, $P(O^{i,j}) = 1$, alta prioridade, $P(O^{i,j}) = 0{,}8$, prioridade normal, $P(O^{i,j}) = 0{,}5$, e assim por diante.

Uma quantidade maior de recursos computacionais será alocada para a execução de operações com maior prioridade. Isto pode significar que o serviço está hospedado em ambientes com maior poder de processamento, ou que será destinado uma atenção maior em sua execução (SAHAI *et al.*, 2001).

b) *Custo Monetário*

Muitos serviços existentes atualmente exigem que em sua utilização, paguemos uma determinada quantia, dependendo da operação a ser utilizada e da

prioridade solicitada na execução desta operação. Sendo assim, definiremos como $M(O^{i,j})^p$, o custo monetário para a utilização de um determinada operação $O^{i,j}$ de um serviço S_i , dada uma prioridade p.

Neste trabalho, estaremos levando em consideração apenas serviços cujo modelo para o cálculo de seu custo monetário seja baseado no sistema "pague por utilização" (*pay-per-use model*), ou seja: a cada execução do serviço, uma taxa fixa é cobrada, independentemente do tempo de utilização do serviço. Ainda não existe um padrão sobre como deve ser feita a cobrança sobre a utilização de serviços Web, embora outras possibilidades já tenham sido abordadas, como um modelo baseado no tempo de utilização do serviço ou uma quantia fixa por mês ou determinado período a ser paga por um consumidor do serviço. (EIBACH e KUEBLER, 2001).

c) *Custo de Execução*

O custo de execução de uma operação $O^{i,j}$ de um serviço S_i compreende o tempo gasto desde o início da execução da operação até o seu término, desconsiderando-se os custos de inicialização e transmissão envolvidos.

Como o tempo de execução de uma operação pode variar dependendo de seus parâmetros de entrada, em nosso modelo de custo optamos por tratar do tempo médio de execução de uma operação. Sendo assim, idealmente, cada operação de um determinado serviço deve expor o seu custo médio de execução, calculado tomando por base os diversos valores possíveis dos parâmetros de entrada e suas respectivas freqüências. Em (BOULOS e ONO, 1999) é mostrado um modelo para o cálculo do custo médio de execução de métodos em sistemas de banco de dados objeto-relacionais que pode ser aplicado também em outros ambientes, incluindo-se plataformas para execução de serviços Web.

Também levamos em consideração o fato de um determinado serviço possuir diversos níveis de prioridade. Sendo assim, pode ser feita uma diferenciação nos tempos de execução de um serviço dependendo de sua prioridade, pois serviços com uma prioridade mais alta terão custos de execução menores que aqueles com prioridade mais baixa.

Finalmente, convencionamos por:

$$T_E (O^{i,j})^p$$

o tempo de execução de uma determinada operação $O^{i,j}$, de um serviço S_i, dada uma prioridade p.

d) *Granularidade da Requisição*

A granularidade da requisição corresponde à quantidade média de dados a serem enviados em uma requisição de execução de uma determinada operação $O^{i,j}$ de um serviço S_i. No caso de serviços Web, corresponde ao tamanho médio da mensagem (geralmente uma mensagem SOAP) a ser enviada ao provedor do serviço.

Na maior parte das vezes, essa informação pode ser obtida a partir da própria definição da operação e suas mensagens, contida no documento WSDL que descreve o serviço. Entretanto, para operações que possuam vetores como parâmetros de entrada ou tipos de dados definidos pelo usuário, o cálculo da quantidade de dados a ser enviada pode se tornar uma tarefa complexa. Dessa forma, idealmente, um serviço deve expor esse parâmetro como parte da descrição dos critérios de qualidade do serviço.

Sendo assim, denotamos por:

$$Q_{Env}(O^{i,j})$$

a quantidade média de dados a serem enviados numa requisição para a execução de uma operação O_j de um serviço S_i.

e) *Granularidade da Informação de Resposta*

A granularidade da informação de resposta corresponde à quantidade média de dados a serem transmitidos como resposta à execução de uma determinada operação $O^{i,j}$. No caso de serviços Web, corresponde ao tamanho médio da mensagem (geralmente uma mensagem SOAP) a ser enviada como resposta à aplicação cliente.

Dependendo mais uma vez dos parâmetros de entrada, a quantidade de dados a serem transmitidos como resposta pode variar bastante. Podemos comparar essa situação com a variação nos valores dos predicados de uma consulta feita a um banco de dados. Dependendo dos valores dos predicados, a cardinalidade da relação resultante da consulta pode aumentar ou diminuir, sendo que o mesmo acontece na execução de uma operação de um serviço, variando-se seus parâmetros de entrada.

Sendo assim, denotamos por:

$$Q_{Resp}\ (O^{i,j})$$

a quantidade média de dados a serem enviados como resposta à execução de uma operação $O^{i,j}$ de um serviço S_i.

4.3 – Um Modelo de Custo para a Execução Dinâmica de Serviços Web

4.3.1 - Introdução

Modelos de custo vêm sendo utilizados com sucesso para estimar os custos envolvidos em diversas áreas da computação, como em sistemas de banco de dados (ROTH *et al.*, 1999, ÖZSU e VALDURIEZ, 1999), arquiteturas de redes (NAJAFI e LEON-GARCIA,2000) , entre outras.

Um modelo de processamento consiste em um conjunto de fórmulas que permite estimar os custos envolvidos em um determinado ambiente computacional (ELMASRI e NAVATHE, 1994). Ele pode ser utilizado para uma avaliação quantitativa de cada configuração possível do ambiente, permitindo assim, que as melhores opções sejam identificadas objetivamente, evitando o processamento real das atividades previstas no modelo.

Alguns critérios são utilizados na análise dos custos envolvidos em um determinado ambiente: um modelo é especificado de acordo com um conjunto de decisões sobre quais aspectos serão abordados e qual será o nível de detalhamento dessa abordagem. Dessa forma, definimos a fronteira de aplicação do modelo de custo. Além disso, para que os resultados de um modelo possuam significado, é necessário definir quais métricas de desempenho serão adotadas. Em otimizadores de sistemas de banco de dados, métricas como o tempo total (soma do tempo gasto em todas as operações realizadas durante uma consulta) e tempo de resposta (tempo registrado entre o início e o encerramento da consulta) são as mais utilizadas (NICOLA e JARKE, 2000).

Muito frequentemente, também são adotadas suposições e restrições que simplificam o universo analisado pelo modelo. Essa prática é justificada, pois um modelo de custo visa proporcionar uma análise específica, permitindo que

determinados aspectos do contexto sejam estudados isoladamente. É importante que as suposições adotadas no modelo não representem simplificações exageradas da realidade, as quais poderiam comprometer os resultados obtidos na análise (RUBERG, 2001). Um exemplo muito comum de simplificação utilizado é admitir, numa arquitetura de rede, que o tempo de transmissão é sempre constante, o que nem sempre é verdade, devido às diferentes cargas a que uma rede é submetida, criando "gargalos" na comunicação.

Finalmente, o custo envolvido na utilização de um modelo por um otimizador não pode ser alto, pois embora considerado insignificante na maioria dos trabalhos na literatura, ele pode ter um peso importante em ambientes distribuídos onde altos custos de comunicação podem estar envolvidos na obtenção de informações sobre as bases de dados (DESHPANDE e HELLERSTEIN, 2002). Sendo assim, precisamos de um modelo que leve em consideração as características de serviços e suas operações, sem que o custo para aplicarmos tal modelo tenha uma influência significativa no desempenho do sistema que o utilize.

4.3.2 – Apresentação do Modelo de Custo

Considere o conjunto $S^f = \{S_0 , S_1 , S_2 , ... , S_{n-1}\}$, como sendo um conjunto de n serviços de funcionalidade equivalente. Para cada serviço S_i^f, considere também o conjunto $O_i^f = \{O_0 , O_1 , O_2 , ... , O_{m-1}\}$, conjunto das *m* operações oferecidas por cada serviço S_i^f.

Dada a similaridade das funções oferecidas por cada serviço, podemos elaborar um modelo de custo que auxilie na decisão de qual serviço executar em um determinado instante de tempo, levando-se em conta os parâmetros discutidos na seção anterior. Dessa forma, podemos pensar num modelo dinâmico para a execução de serviços Web, onde ao invés de invocarmos um serviço específico, apenas escolhemos a classe de serviços que queremos executar e os critérios de qualidade desejados, sendo papel do modelo de custo a decisão de qual serviço pertencente à classe selecionada deve ser efetivamente executado.

O custo total de execução de uma operação $O^{i,j}$ de um serviço S_i, denotado por $C_T^{i,j}$ e definido na Equação 1 é semelhante ao cálculo do tempo total de execução de um serviço Web, apresentado em (CHANDRASEKARAN *et al.*, 2002, CARDOSO e SHETH, 2002). Nesse cálculo, são considerados o tempo de execução do serviço, os

custos envolvidos nas transmissões de mensagens (custo de rede) e o custo de espera pelo serviço, dependendo da carga a qual o sistema esteja submetido. Esse custo de espera pode ser incluído como parte do custo de inicialização apresentado na Equação 1.

Podemos estender essa equação de forma que esta englobe também, os critérios de qualidade "Disponibilidade" e "Confiabilidade", definidos na Seção 4.2.2. Dessa forma, estaremos garantindo que o modelo de custo não será estático pois, após sucessivas execuções de serviços, os parâmetros "Disponibilidade" e "Confiabilidade" podem variar bastante, fazendo com que serviços constantemente indisponíveis ou não confiáveis tenham sua execução preterida pela execução de um outro serviço, com um tempo de execução um pouco maior, porém, mais confiável.

Assim, a equação para o custo total de execução ficaria:

$$C_T^{i,j} = \frac{C_I^i + C_E^{i,j} + C_R^{i,j}}{D(S_i) * R(S_i)} \qquad \text{(Eq. 6)}$$

Como os valores de $D(S_i)$ e $R(S_i)$ variam entre 0 e 1, ao dividirmos o tempo total de execução por esses critérios de qualidade, estaremos aumentando o tempo de execução de serviços menos confiáveis ou constantemente indisponíveis. Inicialmente, deve ser considerado que cada serviço é completamente confiável e disponível, ou seja, $D(S_i) = 1$ e $R(S_i) = 1$. A partir de sucessivas execuções de serviços, esses parâmetros devem ser ajustados de acordo com um modelo matemático que reflita os verdadeiros parâmetros de confiabilidade e disponibilidade de um serviço.

Nas próximas seções, mostraremos como são calculados cada um dos custos envolvidos na execução de um serviço.

4.3.3 – Custo de Inicialização (C_I^i)

O custo de inicialização de um serviço (C_I^i na Equação 6) corresponde ao intervalo de tempo máximo para que um determinado serviço seja inicializado, tendo sido definido como $I(S_i)$ na Seção 4.2.2. Idealmente, cada serviço Web deve oferecer essa informação para aplicações clientes que desejem executar esse serviço.

Sendo assim, podemos definir simplesmente:

$$C_I^i = I(S_i) \quad \text{(Eq. 7)}$$

4.3.4 – Custo de Execução ($C_E^{i,j}$)

Na Seção 4.2.2, foi definido $T_E (O^{i,j})^p$ como sendo o tempo gasto desde o início da execução de uma operação $O^{i,j}$ de um serviço S_i até o seu término, dada uma prioridade *p*. Dessa forma, também podemos assumir que:

$$C_E^{i,j} = T_E(O^{i,j})^p \quad \text{(Eq. 8)}$$

Idealmente, cada provedor de um serviço Web deve oferecer os custos médios de execução de cada operação de um serviço, com estes custos podendo variar de acordo com a prioridade na execução.

4.3.5 – Custo de Transmissão ($C_R^{i,j}$)

Na execução de um serviço Web, podemos considerar dois custos de transmissão envolvidos:

- Custo de envio de uma requisição para a execução de uma operação $O^{i,j}$ de um serviço S_i : $Env(O^{i,j})$.
- Custo de envio da resposta da execução de uma operação $O^{i,j}$ de um serviço S_i: $Resp(O^{i,j})$.

Assim, podemos assumir que :

$$C_R^{i,j} = Env(O^{i,j}) + \mathrm{Re}\,sp(O^{i,j}) \quad \text{(Eq. 9)}$$

Utilizaremos nesta dissertação o modelo de custo proposto em (ÖZSU e VALDURIEZ, 1999), cujo custo de comunicação é representado através da Equação 4 mostrada anteriormente. Iremos considerar o custo para a inicialização de uma mensagem (T_{MSG} ou, generalizando para todas as mensagens, T_{MSG} * #msgs) como estando embutido em T_{TR}, ou seja, fazendo parte do tempo de transmissão de uma unidade de dados. Logo, de maneira simplificada, nosso custo de comunicação passará a ser representado por:

$$C = (T_{TR} * \#bytes) \quad \text{(Eq. 10)}$$

Portanto, podemos agora definir Env($O^{i,j}$) como sendo:

$$Env(O^{i,j}) = T_{TR1} * Q_{ENV}(O^{i,j}) \quad \text{(Eq. 11)}$$

onde:

T_{TR1} = Custo de transmissão de uma unidade de dados do cliente para o provedor do serviço Web.

Q_{Env} ($O^{i,j}$) = Quantidade média de dados a serem enviados numa requisição para a execução de uma operação $O^{i,j}$ de um serviço S_i (definido na Seção 4.2.3).

De forma análoga podemos agora definir Resp($O^{i,j}$) como sendo:

$$\text{Re}sp(O^{i,j}) = T_{TR2} * Q_{RESP}(O^{i,j}) \quad \text{(Eq. 12)}$$

onde:

T_{TR2} = Custo de transmissão de uma unidade de dados do provedor do serviço Web para o cliente.

Q_{Resp} ($O^{i,j}$) = Quantidade média de dados a serem enviados como resposta à execução de uma operação O_j de um serviço S_i (definido na Seção 4.2.3).

Finalizando, podemos agora definir o custo de transmissão ($C_R^{i,j}$) envolvido na execução de um serviço como sendo:

$$C_R^{i,j} = (T_{TR1} * Q_{ENV}(O^{i,j})) + (T_{TR2} * Q_{RESP}(O^{i,j})) \quad \text{(Eq. 13)}$$

4.3.6 – Custo Monetário

Ao invés de considerarmos apenas o tempo total de execução como o único parâmetro para a escolha de qual serviço Web executarmos em um determinado momento, podemos considerar também, o custo monetário envolvido na execução de um serviço como um outro parâmetro nesta escolha.

Assim, para cada execução de um serviço Web, uma aplicação cliente poderia decidir se a escolha de um serviço seria baseada em:

a) Tempo de Execução: Nesse caso, a escolha do serviço Web a ser executado independe do custo monetário envolvido, pois estamos dando preferência ao serviço com o menor tempo execução.

b) Custo Monetário: Aqui, a situação se inverte, a escolha do serviço Web a ser executado irá levar em conta apenas o custo monetário envolvido, não importando os tempos de execução dos serviços.

c) Melhor Relação Custo / Desempenho: Nessa última opção, levamos em consideração ambos os parâmetros na escolha do serviço Web a ser executado: o tempo de execução do serviço e seu custo monetário. Dividimos o custo monetário envolvido pelo tempo de execução do serviço, para todos os serviços candidatos a execução. Aquele que possuir a menor relação custo / tempo de execução deverá ser o serviço escolhido.

4.4 – Execução Paralela de Serviços Web

De forma a melhorarmos o desempenho de composições de serviços Web, diminuindo seu tempo de resposta, podemos pensar na execução em *paralelo* de serviços. Dadas duas operações, O_1 (pertencente a um serviço S_1) e O_2 (pertencente a um serviço S_2), estas podem potencialmente ser executadas concorrentemente se forem mutuamente independentes, ou seja, se os resultados obtidos na execução de uma delas não interferir na execução da outra e se os recursos computacionais adquiridos exclusivamente na execução de uma não forem necessários para a execução da outra. A possibilidade de falhas que levem a aplicação a um estado inconsistente também deve ser levada em consideração quando do escalonamento paralelo de operações de serviços.

Numa plataforma que possibilite a composição e execução de serviços Web, podemos pensar em três diferentes níveis de paralelismo:

- Execução paralela de serviços de uma composição: Uma composição formada por vários serviços Web pode ter a execução destes serviços feita de forma concorrente. O escalonamento de serviços e a possibilidade de execução em paralelo, se assemelham neste caso, ao problema da execução concorrente de processos e atividades em sistemas de *workflow*, podendo ser vistos em (ALONSO et al.,1996) e (KAMATH e RAMAMRITHAM, 1998), e mais

recentemente, em plataformas e linguagens para a composição de serviços Web (PIRES,2002, *SHENG et al.*, 2002, THATTE, 200, LEYMANN, 2001).

- Execução Paralela de Serviços Contidos em Outro Serviço: Um serviço Web pode em sua implementação chamar vários outros serviços, podendo estes últimos, teoricamente, serem executados em paralelo. Como esta possibilidade depende da lógica e da implementação de cada serviço, este tipo de paralelismo não será tratado nesta dissertação.

- Execução em Paralelo de Serviços de Funcionalidade Equivalente: Quando uma composição solicita a execução de um serviço em plataformas que possibilitem a seleção dinâmica de qual serviço efetivamente executar, teremos vários serviços candidatos à execução. Estes, por sua vez, podem ser executados em paralelo, com o retorno de um deles sendo aproveitado como resultado e os outros, descartados. Entretanto, como apenas um desses serviços terá seu resultado consolidado, os efeitos dos outros serviços precisam ser desfeitos. Dessa forma, apenas serviços compensáveis (serviços cujos efeitos podem ser desfeitos, possivelmente através da execução de outro serviço, como mostrado na Seção 3.3.1) podem ser executados em paralelo.

Quando custos como tempo de resposta e preço estão envolvidos na execução de serviços, a escolha de quais serviços devem ser escalonados em paralelo pode não ser uma tarefa simples. Algumas vezes, serviços não compensáveis possuem custos de execução menores do que serviços compensáveis, o que torna a execução seqüencial destes mais atraente do que a execução em paralelo de serviços compensáveis. O custo de compensação de um serviço também deve ser levado em consideração, pois se este for muito alto, talvez não seja uma boa opção a sua execução em paralelo com outro serviço. Estes e outros aspectos serão levados em consideração na criação de uma estratégia que permita um melhor escalonamento de serviços. Essa estratégia será discutida nas seções seguintes.

4.4.1 – Estratégia para a execução em paralelo de serviços de funcionalidade equivalente

Normalmente, quando uma tarefa composta por vários serviços que podem ser escalonados em paralelo está sendo executada, o custo total de sua execução corresponde ao maior custo de execução dentro dos serviços sendo executados. Isso

ocorre devido ao fato de estarmos interessados nos resultados de todos os serviços, ou seja: somente teremos a tarefa finalizada quando todos os serviços sendo executados concorrentemente terminarem.

Entretanto, como agora estamos tratando de serviços de funcionalidade equivalente, basta que um deles seja finalizado com sucesso para que possamos dar nossa execução como terminada com sucesso. Esse serviço será então eleito como "serviço vencedor" e terá o seu resultado consolidado, enquanto os demais serviços serão abortados com seus resultados sendo desfeitos ou compensados. Dada a necessidade de se compensar os serviços que não o vencedor, somente serviços compensáveis podem ser executados em paralelo, com os demais tipos de serviço sendo obrigados a realizar uma execução seqüencial.

No caso de estarmos lidando com serviços compensáveis, devemos considerar também, o custo de compensação associado a ele, pois caso este não seja o vencedor, uma operação compensatória deverá ser executada. Entretanto, como já temos os resultados de um serviço (o vencedor) sendo consolidado e retornado à aplicação que invocou o serviço, esta poderá continuar com seu fluxo normal de execução, enquanto que as operações compensatórias dos serviços compensáveis (caso existam) podem ser executadas em segundo plano (execução em *background*), sem comprometer o fluxo normal da aplicação e seu correto funcionamento.

Porém, mesmo sendo executadas em segundo plano, essas operações podem vir a prejudicar o desempenho de todo o sistema, visto que recursos computacionais para o gerenciamento dessas operações estão sendo alocados. Caso o custo de uma operação compensatória de um serviço seja excessivamente elevado, esse problema é ainda maior, pois recursos podem ser alocados indefinidamente. Essa questão foi identificada em (SCHULDT, 2001) e neste caso, tais serviços são conhecidos como "pseudo-pivôs": embora teoricamente sejam compensáveis, são tratados como serviços do tipo pivô, com execução seqüencial, dado o alto custo de sua operação compensatória.

O conceito de serviços "pseudo-pivôs" também será utilizado nesta dissertação, de forma a evitarmos a execução de operações de compensação com um alto custo. Além disso, os serviços compensáveis que não necessitam da execução de nenhuma operação compensatória (na linguagem WSTL, são aqueles que possuem o elemento "*PassiveAction*" – Seção 3.3.2), terão sua execução privilegiada, pois podem

ser escalonados concorrentemente sem nenhum custo adicional para sua compensação.

4.4.2 – Modos de Execução de um Serviço

Conforme identificado na Seção 4.3.4, temos três possíveis modos para a execução de um serviço: privilegiar o preço (custo monetário) envolvido na execução, privilegiar o menor custo (tempo) de execução ou privilegiar a melhor relação custo/desempenho. Para cada um desses modos, uma estratégia de execução que utilize ou não a execução paralela de serviços será indicada e apresentada nas próximas seções :

4.4.2.1 – Privilegiar o Menor Custo Monetário de Execução

Neste caso, estamos interessados em minimizar o custo monetário envolvido na execução do serviço. Dessa forma, o tempo de execução somente será levado em consideração no caso de termos serviços com custos monetários semelhantes. Os seguintes passos devem ser seguidos por um módulo que seja responsável por escalonar os serviços a serem executados num determinado instante de tempo:

1. Ordenar de forma crescente os serviços Web candidatos à execução pelo seu custo monetário de execução.

2. Executar o serviço Web de menor custo. Caso haja alguma falha na execução deste serviço, seguir executando seqüencialmente o próximo até que um deles termine com sucesso (não há execução paralela de serviços). No caso de serviços com o mesmo custo monetário, escolher aquele de menor tempo de execução.

Caso tenhamos serviços Web de custo zero (serviços cuja execução não envolve um custo monetário relacionado), estes podem ser potencialmente executados em paralelo. Neste caso, estaremos dando preferência ao de menor tempo de execução, e devemos seguir os passos apresentados na Seção 4.4.2.2.

4.4.2.2 – Privilegiar o Menor Tempo de Execução

Neste caso, estamos interessados em executar o serviço da forma mais eficiente possível, com o menor tempo de resposta. Para alcançar esse objetivo, podemos executar serviços compensáveis em paralelo, escolhendo ao final do processamento um serviço como o sendo o vencedor e validando seus resultados.

Nesta dissertação, definimos serviços "pseudo-pivôs" como sendo aqueles cujo custo de sua operação compensatória é mais alto que o custo de execução de todas operações de serviços de funcionalidade equivalente. Explicitando, temos :

Seja $O = \{ O_0, O_1, O_2, ..., O_{m-1} \}$, *um conjunto de m operações de funcionalidade equivalente, cada uma delas pertencente a um serviço* S_i. *Dada uma operação* $O_j \in O$ *e sua operação compensatória* O_j^{-1} *que desfaz seus resultados, se* $C_T(O_j^{-1}) > C_T(O_k)$, $\forall$ k, $0 \leq k < m$ *e* $k \neq j$, *dizemos que o serviço* S_j *que contém a operação* O_j *é um serviço "pseudo-pivô".*

Serviços "pseudo-pivôs" serão tratados da mesma forma que serviços pivôs na estratégia a ser mostrada. Portanto, teremos dois tipos de serviços compensáveis: serviços que não precisam de uma operação compensatória para desfazer seus resultados (serviços do tipo "*PassiveAction*") e serviços com uma operação compensatória associada, mas que não sejam "pseudo-pivôs".

O número máximo de serviços que poderão ser executados em paralelo será definido por um fator de agrupamento, um parâmetro do sistema que tem como objetivo decidir, dada a carga de utilização do sistema num determinado momento, qual o número ótimo de serviços que devem ser executados em paralelo para obtermos o máximo de desempenho. Esse fator de agrupamento deve ser calculado pelo sistema e passado como parâmetro para um escalonador que utilize o modelo de custo e as estratégias de execução paralela discutidas nessa dissertação.

Os seguintes passos devem ser seguidos quando o modo de execução escolhido for privilegiar o menor tempo de resposta:

1. Ordenar de forma crescente os serviços Web candidatos à execução de acordo com seu tempo total de execução, calculado através de um modelo de custo como o apresentado na Seção 4.3.2

2. A partir do serviço de menor tempo de execução, verificar o seu comportamento transacional e tomar uma das seguintes decisões:

- Caso o serviço de menor custo seja não compensável ou "pseudo-pivô", executá-lo sozinho. Caso aconteça alguma falha em sua execução, retirá-lo da lista de serviços candidatos e reaplicar a estratégia a partir do passo 2.

- Caso o serviço de menor custo seja compensável do tipo "*PassiveAction*" (serviços compensáveis que não precisam de uma operação compensatória para desfazer seus resultados), executar os primeiros *f* serviços deste tipo em paralelo , onde *f* é o fator de agrupamento do sistema.

- Caso o serviço de menor custo seja compensável possuindo uma operação de compensação associada, podemos executar os *f* primeiros serviços compensáveis em paralelo, onde *f* é o fator de agrupamento do sistema (levando em consideração tanto os do tipo "*PassiveAction*" quanto aqueles que não sejam "pseudo-pivôs").

Caso tenhamos executado serviços em paralelo, podemos ter ao final de nossa execução o resultado de mais de um serviço. Neste caso, temos que escolher um serviço como sendo o vencedor e compensar os demais. Podemos assumir que o primeiro serviço a ser finalizado com sucesso será o vencedor e já iniciar o processo de compensação dos demais. No caso de dois ou mais serviços terminarem sua execução simultaneamente, ou num curto intervalo de tempo, escolheremos como vencedor aquele com o maior custo de compensação, de forma a evitarmos a execução desta operação.

4.4.2.3 – Privilegiar a Melhor Relação Custo/Desempenho :

Neste caso, estamos interessados na melhor relação custo/desempenho na execução de serviços. Para calcularmos esta relação, dividimos o custo monetário da execução de um serviço pelo seu tempo total de execução. A estratégia a ser adotada nesta situação é mostrada a seguir:

1. Ordenar de forma crescente os serviços de acordo com sua relação custo/desempenho.

2. A partir do serviço de melhor relação custo/desempenho, verificar seu custo monetário e seu comportamento transacional e tomar uma das seguintes decisões:

- Caso o serviço possua um custo monetário diferente de zero ou seja não compensável, executa-lo sozinho. Nesse caso, se não houver uma falha na execução do serviço, temos a garantia de estarmos obtendo a melhor relação custo/desempenho.
- Caso o serviço seja compensável e não tenha nenhum custo monetário associado (ou seja, serviço de custo zero), procurar na lista de serviços candidatos todos os serviços que tenham essas mesmas características e executá-los em paralelo. Fazendo isso, estaremos executando todos os serviços de custo zero em paralelo, garantindo que teremos a melhor relação custo/desempenho.

4.5 – Representação dos Parâmetros de Custo e Critérios de Qualidade

Idealmente, um serviço Web deve fornecer às aplicações clientes informações sobre seus custos, discutidos na Seção 4.2, para que estas aplicações possam ter um maior conhecimento do serviço sendo executado, permitindo ainda o desenvolvimento de modelos de custo como o mostrado na Seção 4.3.

De forma a padronizar o modo como as informações de custo e de qualidade serão oferecidas às aplicações clientes, um modelo para representação desses dados será discutido nesta seção. Como a tecnologia de serviços Web utiliza a linguagem XML para a descrição de serviços e documentos, o modelo de representação proposto também se utiliza da linguagem XML e do XML *Schema* em particular. Finalmente, convencionamos chamar esse modelo de definição de custos e critérios de qualidade de WSQD (*Web Services Quality Definitions*).

Um documento com informações de custo sobre serviços pode ser obtido de diversas formas. Ele pode ser um arquivo fixo, disponibilizado em algum servidor específico na Internet (como o servidor que hospeda o próprio serviço ou um nó do registro UDDI) ou ser retornado em resposta à execução de um outro serviço. Nessa última abordagem, podemos ter um serviço Web oferecendo suas próprias informações de custo através da implementação de uma interface (ou um elemento

"*porttype*", considerando um documento WSDL) que contenha uma operação que retorne esses dados.

Qualquer que seja a forma de obtenção das informações de custo de um serviço, estas devem ser obtidas anteriormente a uma solicitação de execução de uma operação de um serviço. Embora seja possível que essa informação seja solicitada dinamicamente, o custo envolvido nessa operação pode ser, às vezes, bastante considerável. Em ambientes onde um modelo de custo para a execução de serviços como o proposto neste trabalho esteja presente, dependendo do número de serviços semelhantes existentes, o custo para se obter dados sobre todos esses serviços pode ser inclusive maior que o custo para efetivamente executar um deles, inviabilizando essa obtenção dinâmica de informações. Logo, em nossa implementação, os documentos contendo as descrições os critérios de qualidade de serviços são obtidos anteriormente ao processo de execução.

A Figura 4.1 mostra, graficamente, o modelo de definição WSQD proposto (detalhes do modelo gráfico de representação utilizado podem ser visualizados no Apêndice 3).

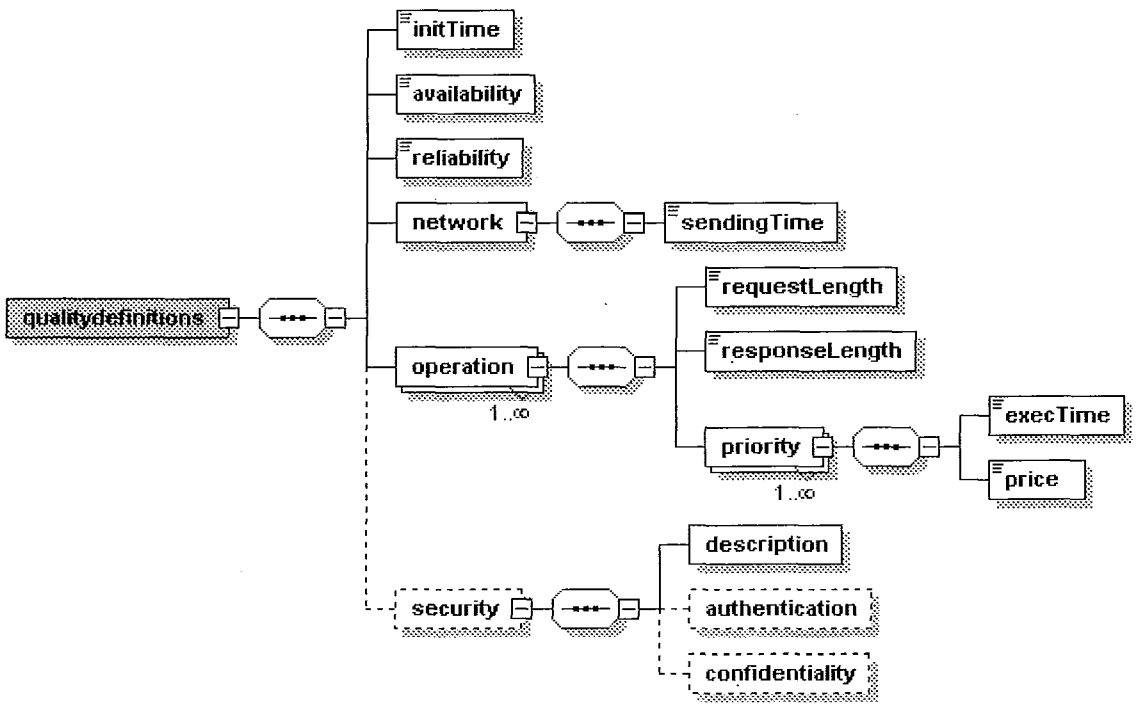


Figura 4.1 – Modelo de Definição WSQD – *Web Services Quality Definitions*

O elemento raiz do documento, denominado "*qualityDefinitions*" contém dois atributos:

- *id*, um nome que identifica o documento em questão;

- *endpoint*, que referencia um elemento "*port*" do documento WSDL do qual as informações de custo se referem. Como uma interface de um serviço pode possuir diversas implementações (tendo sua localização explicitada através de um elemento "*port*" de um documento WSDL), torna-se necessário termos um documento com as informações de custo para cada uma dessas implementações, pois essas podem estar hospedadas em servidores com diferentes capacidades de processamento e portanto com diferenciados custos de execução e comunicação.

O elemento "*qualityDefinitions*" possui ainda quatro subelementos: "*initTime*", "*availability*", "*reliability*", "*network*", "*operation*" e "*security*". Eles serão descritos nas próximas seções.

4.5.1 – Elemento "initTime"

O elemento initTime contém informações sobre o custo de inicialização de um serviço, especificado na Seção 4.2.2. Ele contém um atributo *unit*, que identifica a unidade de tempo utilizada para representar o custo de inicialização (*ms* para milisegundos, *s* para segundos).

4.5.2 – Elemento "availability"

O elemento "*availability*" contém informações sobre a disponibilidade de um serviço, conforme especificado na Seção 4.2.2. O valor da disponibilidade varia entre 0 e 1, sendo que, inicialmente, caso esse valor não seja informado pelo provedor de serviço, podemos assumir que ele seja igual a 1 (serviço 100% disponível). Conforme sucessivas execuções deste serviço vão ocorrendo, é armazenado num arquivo de log o resultado dessas execuções e consequentemente, quantas vezes obtivemos um erro de indisponibilidade do serviço. Esse valor armazenado no arquivo de log é periodicamente utilizado para atualizar o valor contido no documento WSQD. O monitoramento de execuções e a gravação de arquivos de log serão discutidos na Seção 5.5.

4.5.3 – Elemento "reliability"

O elemento *"reliability"* contém informações sobre a confiabilidade de um serviço, conforme especificado na Seção 4.2.2. O valor da confiabilidade varia entre 0 e 1, sendo que, inicialmente, caso esse valor não seja informado pelo provedor de serviço, podemos assumir que ele seja igual a 1 (serviço 100% confiável). Conforme sucessivas execuções deste serviço vão ocorrendo, é armazenado num arquivo de log o resultado dessas execuções e consequentemente, quantas vezes obtivemos algum problema com a confiabilidade do serviço. Esse valor armazenado no arquivo de log é periodicamente utilizado para atualizar o valor contido no documento WSQD.

4.5.4 – Elemento "network"

Este elemento contém informações a respeito da rede onde esta implementação de um serviço Web está hospedada, conforme mostrado na Seção 4.2.2.

Para o cálculo do custo de comunicação envolvido na execução de um serviço, estamos interessados no tempo necessário para transmitir uma unidade de dado do servidor onde o serviço está hospedado para a máquina executando a aplicação cliente. Para representar tal informação, temos um subelemento "*sendingTime*", que indica o tempo necessário para a transmissão de uma determinada unidade de dados, unidade essa especificada no atributo "*dataunit*", deste subelemento (como possíveis unidades, temos *bytes*, *Kbytes*, entre outras). O atributo "*unit*" deste subelemento representa a unidade de tempo que está sendo utilizada para representar esse tempo de transmissão.

4.5.5 – Elemento "operation"

Um elemento "operation" corresponde a uma operação existente em um elemento "porttype" do documento WSDL que descreve o serviço em questão. Além do atributo "name", que identifica o nome da operação, um elemento "operation" possui ainda três subelementos:

- "*requestLength*": Corresponde à quantidade média de dados a serem enviados em uma requisição de execução dessa operação (conforme descrito na Seção

4.2.3). Esse elemento possui um atributo de nome *unit,* identificando a unidade de dados utilizada para representar esse valor (geralmente, bytes).

- "*responseLength*": Corresponde à quantidade média de dados a serem transmitidos como resposta à execução dessa operação (conforme descrito na Seção 4.2.3). Esse elemento também possui um atributo de nome *unit*, identificando a unidade de dados utilizada para representar esse valor (geralmente, *bytes*).

- "*Priority*": Identifica os vários níveis de prioridade que podem estar associados a uma operação. Esse elemento possui um atributo de nome *value*, que identifica o grau de prioridade ("low", para baixa prioridade, "normal", para prioridade normal, "high", para prioridade alta, podendo existir ainda, valores intermediários a estes). Um elemento "*priority*" possui ainda dois subelementos:

  - "*execTime*": Corresponde ao custo de execução da operação especificada no elemento "*operation*", dada a prioridade especificada no elemento "*priority*" (conforme descrito em 4.2.3). Esse elemento possui um atributo de nome *unit*, que identifica a unidade de tempo utilizada para representar o custo de execução (*ms* para milisegundos, *s* para segundos).

  - "*price*": Corresponde ao custo monetário associado à execução da operação especificada no elemento "*operation*", dada a prioridade especificada no elemento "*priority*" (conforme descrito em 4.2.3). Esse elemento possui ainda um atributo de nome *unit*, que identifica a unidade monetária utilizada para representar este custo (exemplo: USD para dólares americanos, R para reais).

4.5.6 – Elemento "security"

Este é um elemento opcional, devendo ser preenchido somente por serviços que apresentem algum mecanismo de segurança, como os apresentados na Seção 4.2.2. O subelemento "*description*" contém uma descrição destes mecanismos de segurança, podendo trazer informações sobre os protocolos e métodos utilizados para tornar o serviço Web mais seguro.

O elemento "*security*" contém ainda mais dois subelementos : "*authentication*" e "*confidentiality*". O primeiro se refere ao modelo de autenticação utilizado pelo serviço Web (se a autenticação de usuários é feita através de um protocolo como o HTTPS ou se é solicitado um par usuário/senha), enquanto o segundo se refere a como a confidencialidade dos dados é garantida na comunicação entre o servidor Web e a aplicação cliente (utilizando um protocolo como o SSL, por exemplo). Ambos subelementos são opcionais, devendo ser preenchidos apenas por serviços Web que garantam a autenticação de usuários e/ou a confidencialidade na transmissão dos dados.

O Apêndice 1 mostra listagem completa do XML Schema utilizado para representar a linguagem WSQD.

4.6 – Representação dos Parâmetros de Custo e Critérios de Qualidade no Registro UDDI

Assim como a maior parte das especificações relacionadas a serviços Web (WSDL, SOAP, entre outras), a especificação UDDI vem sofrendo constantes atualizações de forma a aumentar sua eficiência. Para facilitarmos o acesso às informações de custo e critérios de qualidade discutidos na seção anterior, podemos estender a especificação UDDI de forma que esta possua uma referência para a URL do documento WSQD que contenha essas definições de custo.

Alguns trabalhos na literatura também propõem atualizações no UDDI, mais especificamente, na área relativa a critérios de qualidade de um serviço. Em (PAOLUCCI *et al.*, 2002b), é proposta uma ligação da especificação DAML-S com a especificação UDDI, buscando uma melhor representação semântica de serviços, contemplando também seus critérios de qualidade. Em (SHAIKH *et al.*, 2003), é proposta a especificação UDDIe (UDDI Extension) que além de outras atualizações, sugere a criação de um elemento "*propertyBag*", que conteria as propriedades de um determinado serviço. Essas propriedades poderiam contemplar os critérios de qualidade de um serviço apresentados anteriormente. Em ambos os trabalhos, a definição de quais critérios de qualidade devem estar contidos no registro UDDI é deixada a cargo do provedor do serviço. Apesar de garantir uma maior flexibilidade, esse tipo de representação dificulta a tarefa de um modelo de execução que trate de características específicas de um serviço, que podem não estar presentes no registro.

Conforme mostrado na Seção 2.3.4, os elementos de um documento WSDL podem ser mapeados para o registro UDDI da seguinte forma: uma interface de um serviço, geralmente ligada a um protocolo de comunicação (ou seja, um documento WSDL com os elementos "types", "import" , "messages" , "porttype" , "binding") é utilizada como base para a criação de um elemento "tModel" do UDDI, enquanto informações sobre a implementação de um serviço (um elemento "port" de um documento WSDL) são utilizadas como base para a criação de elementos "bindingTemplate" do UDDI (um elemento "bindingTemplate" para cada implementação).

Como as informações de custo e critérios de qualidade são específicas a uma implementação de um serviço, podemos estender o elemento "bindingTemplate" do UDDI de forma que este contenha essas informações. Da mesma forma que o subelemento "accessPoint" (de um "bindingTemplate") faz referência à URL pela qual o serviço deve ser acessado, podemos criar mais um subelemento para um "bindingTemplate", que faça referência à URL pela qual o documento WSQD contendo as informações de custo e qualidade possa ser obtido. Este subelemento receberá o nome "qualityDefinitions".

O Exemplo 4.1 mostra um elemento "bindingTemplate" do UDDI estendido com um elemento *qualityDefinitions*.

```
<bindingTemplate>
        (...)
        <acessPoint urlType="http">http://www.stockquoteservice.com/service.asmx
        </acessPoint>
        <qualityDefinitions>http://www.stockquoteservice.com/stockquote.wsqd
        </qualityDefinitions>
        <tModelInstanceDetails>
                <tModelInstanceInfo tModelKey="...">
                        (...)
                </tModelInstanceInfo>
                (...)
        </tModelInstanceDetails>
</bindingTemplate>
```

Exemplo 4.1 – "Estendendo o bindingTemplate com o parâmetro "qualityDefinitions".

No próximo capítulo são discutidos os detalhes da implementação da plataforma *WebTransact,* apresentada no Capítulo 3. Os conceitos apresentados nas seções anteriores (parâmetros de custo e critérios de qualidade, modelo de custo para o cálculo do tempo de resposta de um serviço e os algoritmos para execução paralela de serviços) também terão sua implementação detalhada nesse capítulo.

5 – Implementação do *WebTransact-EM*

5.1 – Introdução

Conforme apresentado no Capítulo 3, a plataforma *WebTransact* (utilizada como base para a implementação do modelo WebTransact-EM) foi especificada com o objetivo de gerenciar de forma eficiente a composição e execução de serviços Web. Essa plataforma é composta por três elementos principais:

- Uma arquitetura dividida em três camadas, com uma camada para a especificação de aplicações compostas por vários serviços, outra para a definição de mediadores, que agrupam serviços Web de funcionalidade equivalente e outra para a representação dos serviços remotos disponíveis na rede, possivelmente contendo mapeamentos de suas interfaces para a interface do serviço definida pelo mediador.

- Uma linguagem baseada em XML, a WSTL, utilizada para representar todos os elementos da arquitetura, como a especificação de aplicações, serviços remotos e o comportamento transacional de cada serviço Web.

- Um modelo transacional que garante a confiabilidade na execução de aplicações compostas por serviços Web de comportamentos transacionais distintos.

Das três camadas da arquitetura do *WebTransact*, foram implementadas nesta dissertação a camadas dos mediadores e a camadà de serviços remotos (além do modelo de execução de serviços proposto, o *WebTransact-EM*). A camada de composição de serviços, que trata da especificação de aplicações baseadas na execução de múltiplos serviços, não será discutida nesta dissertação. As estratégias para a seleção dinâmica de serviços Web discutidas no Capítulo 4, bem como o modelo de custo apresentado terão suas implementações discutidas ao longo deste capítulo.

5.2 – Diagrama de Classes

A Figura 5.1 mostra o diagrama de classes em UML utilizado como base para a implementação da *WebTransact*. Cada uma das classes presentes no diagrama terá sua funcionalidade descrita a seguir:

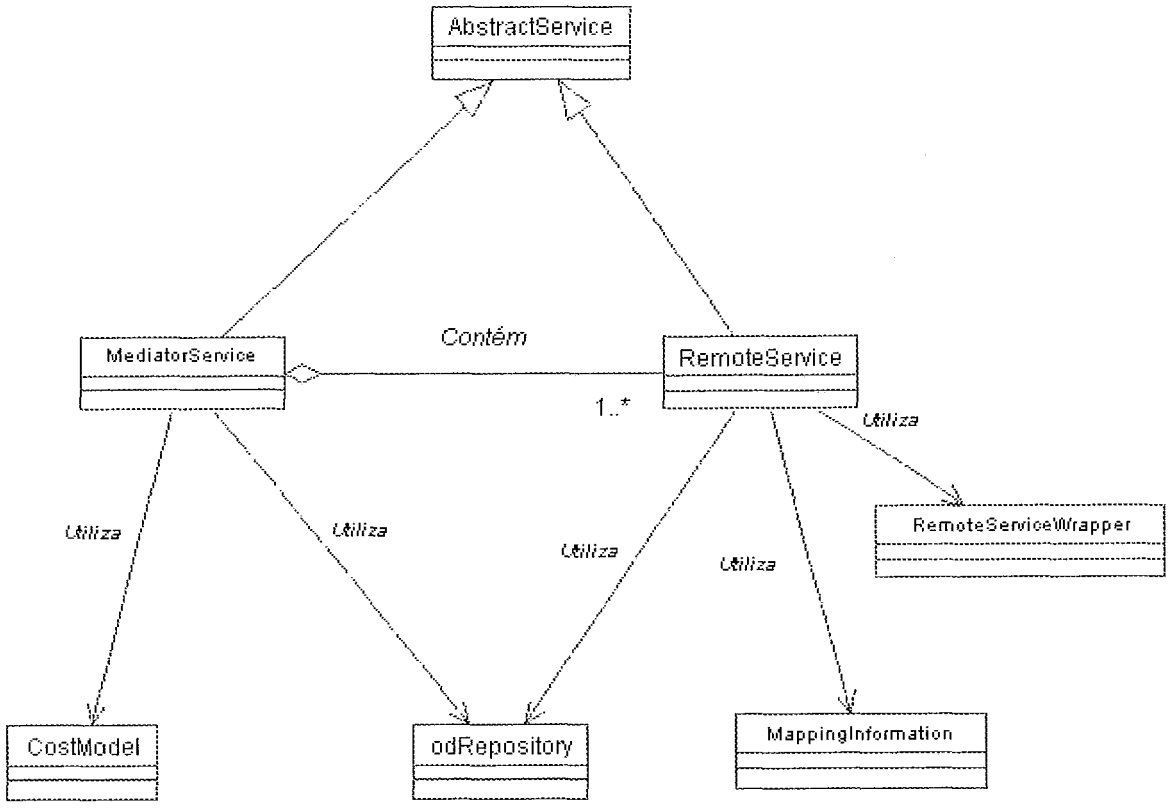


Figura 5.1 – Diagrama de classes do WebTransact

- Classe *MediatorService*: Representa um mediador em nossa arquitetura, podendo agregar diversos serviços remotos semanticamente equivalentes (representados pela classe *RemoteService*). Contém métodos responsáveis pela escolha, ordenação e escalonamento dos serviços remotos candidatos à execução. Cada mediador está relacionado a um arquivo WSTL que descreve sua funcionalidade.

- Classe *RemoteService* : Representa um serviço remoto na arquitetura. Contém métodos responsáveis pela verificação dos parâmetros de qualidade do serviço, bem como pela obtenção de seu comportamento transacional. Cada serviço remoto está associado a um arquivo WSTL que descreve sua interface e o mapeamento desta para a interface do mediador que agrega esse serviço.

- Classe *AbstractService*: Classe abstrata, representa um serviço genérico em nossa arquitetura, contendo funcionalidades comuns tanto a um mediador quanto a um serviço remoto.

- Classe *RemoteServiceWrapper* : Responsável pela geração das mensagens SOAP a serem enviadas aos provedores de serviços remotos, solicitando a execução de alguma operação. Implementada com o auxílio de classes contidas na biblioteca Apache Axis (APACHE GROUP,2002).

- Classe *odRepository*: Representa o repositório de dados da arquitetura, contendo métodos responsáveis pela recuperação e armazenamento das definições de serviços remotos, mediadores e aplicações.

- Classe *CostModel*: Classe contendo a implementação do modelo de custo utilizado para calcular o custo total de execução de um serviço Web, conforme mostrado na Seção 4.3.

- Classe *MappingInformation*: Classe responsável pelo mapeamento da interface de um mediador para uma interface de um serviço remoto específico, caso existam diferenças entre as interfaces.

5.3 – Definição de Critérios de Qualidade na Execução de um Serviço

Nas Seções 4.2 e 4.5 foram mostrados os critérios de qualidade e parâmetros de custo que devem ser fornecidos pelos provedores de serviço de forma a se permitir uma seleção dinâmica de serviços baseada nesses parâmetros. Uma aplicação cliente, interessada na execução de uma classe de serviços em particular, pode especificar quais critérios de qualidade um determinado serviço necessita atender. Dessa forma, somente serviços que preencham os requisitos solicitados por essa aplicação poderão ser selecionados para uma possível execução.

A Figura 5.2 mostra o formato de um documento XML utilizado para representar a chamada de serviços remotos (mais especificamente, para a execução de uma operação de um mediador em nossa arquitetura). Além do nome da operação a ser executada e seus parâmetros, diversos requisitos de qualidade podem ser preenchidos.

```
<operation name="..." maxresponsetime="..." maxprice="..."
        executionmode="..." priority="..." mandatorycompensate="..."
        security="..." broadcast="...">

        <param name="..."></param>
        ...
        <param name="..."</param>
</operation>
```

Figura 5.2 – XML com a representação da chamada de um método de um serviço remoto.

Ao executarmos uma operação, além do nome da mesma (indicado no atributo *name*), podemos definir os seguintes critérios de qualidade :

- *maxresponsetime*: Tempo de resposta máximo desejado na execução do serviço. Esse tempo será calculado utilizando-se o modelo discutido na Seção 4.3. Somente serviços cujo tempo de execução seja menor ou igual ao tempo definido pela aplicação cliente poderão ser escolhidos para execução.

- *maxprice*: Preço máximo (custo monetário) que o cliente aceita pagar pela execução do serviço. Somente serviços cujo custo monetário seja menor ou igual ao preço definido pela aplicação cliente poderão ser escolhidos para execução.

- *executionmode*: Indica o modo de execução selecionado pela aplicação cliente. Conforme discutido na Seção 4.3.4, temos três possíveis modos de execução : 1 - Minimizar Preço, 2 – Minimizar Tempo de Resposta, 3 – Maximizar Relação Desempenho/Custo.

- *priority*: Corresponde à prioridade desejada na execução de um serviço. Atualmente no *WebTransact-EM*, um dos três níveis de prioridade pode ser escolhido: Baixa, Normal e Alta. Apesar de ainda não existir um formato padrão de como informar ao provedor de serviço que queremos uma execução sob uma diferente prioridade, é um consenso que, no caso de estarmos utilizando o protocolo SOAP, devemos fazer uso do elemento de cabeçalho (*SOAP Header*) para passarmos essa informação. Utilizaremos em nossa arquitetura um formato de mensagem equivalente ao proposto em (LAU, 2001), onde é criado um elemento específico para itens de qualidade de serviço (QoS) no cabeçalho da mensagem SOAP enviada ao provedor do serviço remoto.

- *mandatorycompensate*: Esse atributo indica se a operação do serviço remoto a ser escolhido precisa ser compensável (operação cujas ações podem ser desfeitas, possivelmente através da execução de uma outra operação do mesmo serviço, conforme definido na Seção 3.3.2). Esse atributo aceita apenas os valores "*yes*" (se a operação deve ser compensável) ou "*no*" (se nenhuma restrição é feita sobre o comportamento transacional da operação).

- *security*: Esste atributo indica se o serviço remoto a ser escolhido deve apresentar algum mecanismo de segurança, como os descritos na Seção 4.2.2. Esse atributo aceita apenas os valores "*yes*" (se o serviço deve apresentar algum mecanismo de segurança) ou "*no*" (se nenhuma restrição é feita sobre a segurança do serviço).

- *broadcast* : Esse atributo, se selecionado, indica que queremos executar todos os serviços remotos agregados por um mediador e que queremos os resultados de todos eles. Nesse caso específico, não teríamos apenas um serviço vencedor, e sim, vários, correspondendo a todos os serviços executados. Esse atributo é útil no caso de serviços que simulem consultas a banco de dados (ex : consultas a preços de passagens de companhias aéreas, onde queremos saber o preço de todas as companhias, para depois escolhermos em qual iremos viajar. Teríamos um mediador agregando os serviços de cada companhia e indicaríamos que desejamos um *broadcast* nesse mediador). No caso desse atributo ser selecionado, todas as demais escolhas feitas pela aplicação cliente (preço, tempo de resposta, segurança, entre outras) são ignoradas, e todos os serviços remotos agregados pelo mediador são executados em paralelo.

- *param* : Elementos que simbolizam os parâmetros que são passados na execução de uma operação. O atributo *name*, nesse elemento, indica o nome do parâmetro.

A Figura 5.3 mostra a tela de apresentação do WebTransact-EM, onde além do mediador e da operação a ser executada, uma operação cliente pode escolher os critérios de qualidade descritos anteriormente.

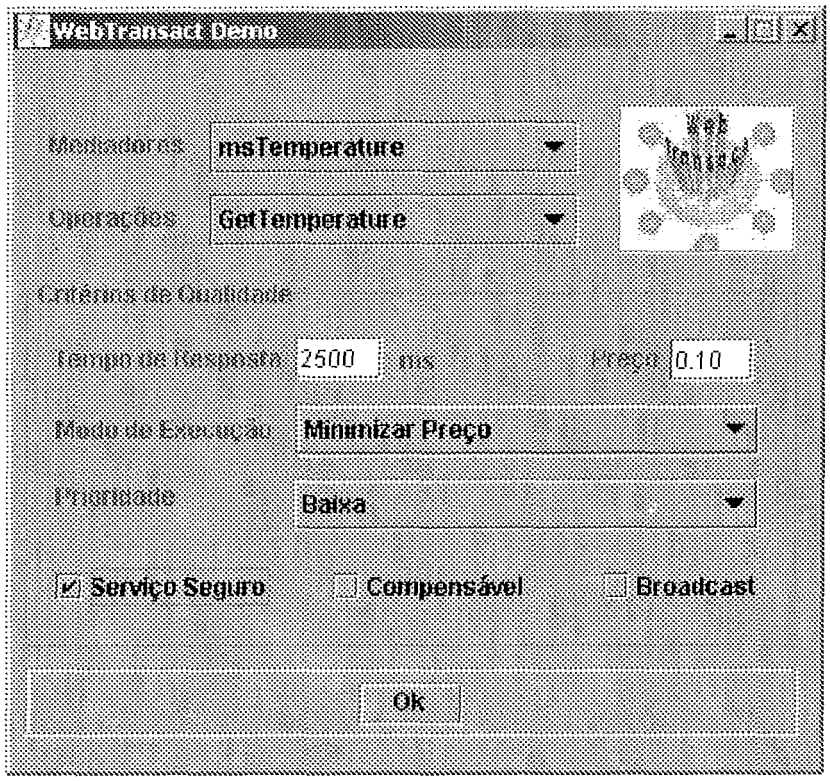


Figura 5.3 - Apresentação da escolha dos critérios de qualidade na *WebTransact-EM*

5.4 – Algoritmos para a Execução Dinâmica de Serviços do WebTransact-EM

O processo de execução de serviços no *WebTransact-EM* envolve diversas atividades, como a seleção inicial dos serviços candidatos à execução, a ordenação desses serviços, o escalonamento de um ou mais serviços para serem efetivamente executados e por último, a escolha de um serviço vencedor.

Apresentaremos em seguida o conjunto de algoritmos do *WebTransact-EM*. Como diferentes implementações de serviços podem apresentar diferentes custos e critérios de qualidade, em nosso modelo, trabalhamos individualmente com cada implementação de um serviço. Portanto, nos algoritmos, quando nos referimos a um *serviço remoto*, estamos nos referindo a um conjunto de portas (implementações) de uma mesma interface de um documento WSDL. Já quando nos referimos a um *serviço*, estamos tratando de uma porta (implementação) específica de um serviço remoto. O Algoritmo 1 representa o algoritmo principal do *WebTransact-EM*, mostrando como as atividades do processo de execução de serviços estão logicamente relacionadas em nosso modelo e o nome das operações que efetuam cada uma dessas atividades.

```
Entrada :   M = mediador
            p = conjunto de parâmetros passados para a execução da operação.
            q = conjunto dos critérios de qualidade definidos pela aplicação cliente.
            m = modo de execução desejado.
            f = fator de agrupamento de serviços
Saída : R = Resultado(s) da Execução do Serviço.

1:    S  ← SelectCandidateServices(M, p , q)
2:    CO ← SortCandidateServices (S , m)
3:    Enquanto (CO != ∅ )
4:          S ← ScheduleServices (m,CO ,f )
5:          ExecuteServices(S)
6:          Se algum dos serviços em S foi finalizado com sucesso
7:                R ← SelectWinnerService(S)
8:                retornar R
9:          Senão
10:               CO ← CO – S
11:         Fim-Se
12:   Fim-Enquanto
```

Algoritmo 1 – Execução de um serviço no WebTransact-EM.

O algoritmo recebe como entrada:

- Um mediador M (na verdade, um identificador que será utilizado para recuperar o arquivo WSTL desse mediador no repositório de dados).
- Um conjunto de parâmetros p, correspondentes aos parâmetros da operação, definidos no arquivo WSDL que descreve o serviço.
- Um conjunto de critérios de qualidade q, contendo os critérios apresentados na Seção 5.3.
- O modo de execução desejado (Minimizar Preço, Minimizar Tempo de Resposta ou Melhor Relação Custo/Desempenho).
- O fator de agrupamento de serviços, correspondendo ao número máximo de serviços que podem ser executados em paralelo.

Como saída do algoritmo, temos um conjunto R de resultados obtidos através da execução do serviço vencedor (que terá os seus resultados consolidados). Nos próximos parágrafos, discutiremos o funcionamento do algoritmo.

Inicialmente, é feita uma seleção dos serviços candidatos à execução baseada principalmente nos parâmetros de qualidade definidos pela aplicação cliente. O Algoritmo 2 detalha como é feita essa seleção.

```
Método:     SelectCandidateServices
Entrada:    M = Mediador
            p : conjunto de parâmetros passados para a execução da operação.
            q = conjunto dos critérios de qualidade definidos pela aplicação cliente.
Saída:      R = conjunto dos serviços candidatos à execução

1:  Se VerifiyContentDescription(Mediador,p) == falso
2:      retornar vazio
3:  Fim-Se
4:  S ← conjunto dos serviços remotos agregados por M.
5:  Para todo serviço Si em S
6:      Se VerifyContentDescription(Si ,p) == verdadeiro
7:          Se VerifyQualityDefinitions(Si , q) == verdadeiro
8:              Acrescentar Si em R
9:          Fim-Se
10:     Fim-Se
11: Fim-Para
12: retornar R
```

Algoritmo 2 – Seleção dos serviços candidatos à execução.

O método *VerifyContentDescription* verifica se os valores dos parâmetros passados na execução da operação estão dentro de um domínio pré-estabelecido (chamado de descritor de conteúdo), definido nos arquivos WSTL tanto do mediador quanto do serviço remoto. A descrição de conteúdo especifica se um determinado serviço remoto possui a capacidade de responder à determinada solicitação. Por exemplo, considere os serviços remotos de validação de cartão de crédito rm_1 e rm_2. Considere ainda que o serviço remoto rm_1 consegue validar cartões das operadoras "American Express" e "MasterCard", enquanto que o serviço remoto rm_2 aceita somente a operadora "Mastercard". Agora, considere que o serviço do mediador ms_1 agrega rm_1 e rm_2. Se ms_1 recebe uma solicitação para efetuar uma validação de um cartão "American Express", então ms_1 deverá delegar essa solicitação somente para rm_1. O serviço de mediador ms_1 sabe, consultando a descrição de conteúdo, que rm_2 não é capaz de validar cartões deste tipo. Já o descritor de conteúdo do mediador ms_1 corresponde ao conjunto formado pela união dos descritores de conteúdo de seus serviços remotos agregados. Sendo assim, uma solicitação de validação de um cartão "VISA" não seria suportada pelo mediador, pois nenhum de seus serviços remotos suporta esta operadora. O descritor de conteúdo do mediador é verificado na linha 1 do Algoritmo 2, enquanto a verificação dos descritores de conteúdo dos serviços remotos é verificada na linha 6 do algoritmo.

É verificado ainda, no método *VerifyQualityDefinitions*, se alguma porta do serviço atende aos critérios de qualidade especificados pela aplicação cliente. Essa verificação é feita a partir da leitura das informações contidas no arquivo WSQD oferecido pelo provedor do serviço. No caso de uma execução em *broadcast*, esse método retorna todas as portas do serviço, com os critérios de qualidade sendo ignorados, visto que queremos executar todos os serviços.

Após a seleção do conjunto de serviços candidatos, é feita através do método *SortCandidateServices*, uma ordenação desse conjunto baseada no modo de execução escolhido pela aplicação cliente (minimizar preço, maximizar tempo de resposta ou maximizar relação desempenho/custo). O Algoritmo 3 mostra como essa seleção é feita.

Método: SortCandidateServices

Entrada: m = modo de execução desejado S = conjunto dos serviços candidatos
Saída : R = conjunto ordenado dos serviços candidatos a execução

```
1:  Se m == 'Minimizar preço'
2:      R ← Ordenar S pelo custo monetário de cada serviço, de forma crescente
3:  Se m == 'Maximizar Tempo de Resposta'
4:      R ← Ordenar S pelo tempo de resposta de cada serviço, de forma crescente.
5:  Se m == 'Maximizar Relação Custo/Desempenho'
6:      R ← Ordenar S pela relação tempo de resposta/custo, de forma decrescente.
7:  Fim-Se
8:  retornar R
```

Algoritmo 3 – Ordenação dos serviços candidatos.

A partir de uma lista de serviços já ordenada, é feito o escalonamento de quais serviços serão efetivamente executados através do método *ScheduleServices*. Tal método leva em consideração o modo de execução selecionado pela aplicação cliente, bem como o comportamento transacional dos serviços para fazer esse escalonamento, conforme é mostrado no Algoritmo 4.

O método *ScheduleServices* funciona conforme descrito a seguir. Se o cliente tiver escolhido uma execução em *broadcast*, todos os serviços candidatos serão executados. Caso contrário, se o modo de execução escolhido for o de minimizar o preço, apenas o serviço de menor preço (o primeiro do conjunto ordenado S) será escalonado.

Método: ScheduleServices

Entrada: m = modo de execução desejado
S = conjunto ordenado dos serviços candidatos
f = fator de agrupamento de serviços

Saída : R = conjunto dos serviços que devem ser executados.

```
1:  Se broadcast = true
2:      R ← S
3:      Retonar R
4:  Fim-Se
5:  Se m == 'Minimizar preço'
6:      Acrescentar S0 em R
7:      retornar R
8:  Fim-Se
9:  Se m == 'Minimizar Tempo de Resposta'
10:     Se S0.transactionbehavior != "Compensable" ou S0.pseudopivot == true
11:         Acrescentar S0 em R
12:         Retornar R
13:     Fim-Se
14:     Se S0.transactionbehavior == "Compensable|PassiveAction"
15:         R ← f primeiros elementos de S onde Si.transactionbehavior ==
            "Compensable|PassiveAction"
16:     Senão
17:         R ← f primeiros elementos de S onde Si.transactionbehavior ==
            "Compensable" e Si.pseudopivot == false
18:     Fim-Se
19: Fim-Se

20: Se m == 'Melhor Relação Custo/Desempenho'
21:     Se (S0.cost !=0)
22:         Acrescentar S0 em R
23:         Retornar R
24:     Senão
25:         Se S0.transactionbehavior != "Compensable" ou S0.pseudopivot == true
26:             Acrescentar S0 em R
27:             Retornar R
28:         Fim-Se
29:         Se S0.transactionbehavior == "Compensable|PassiveAction"
30:             R ← f primeiros elementos de S onde
                Si.transactionbehavior == "Compensable|PassiveAction"
31:         Senão
32:             R ← f primeiros elementos de S onde
                Si.transactionbehavior=="Compensable" e
                Si.pseudopivot==false
33:         Fim-Se
34:     Fim-Se
35: Fim-Se
```

Algoritmo 4 – Escalonamento de Serviços

Caso outro modo de execução seja escolhido, existe a possibilidade de escalonarmos mais de um serviço e termos execução paralela. Assim, usamos as

estratégias definidas nas Seções 4.4.2.2 e 4.4.2.3. Se o primeiro serviço da lista não for compensável ou for um serviço pseudo-pivô, apenas ele é escalonado para execução. Caso ele seja compensável do tipo "*PassiveAction*", são escalonados os f primeiros serviços com esse mesmo comportamento transacional, onde f é o fator de agrupamento do sistema. Por último, caso o serviço seja compensável com uma operação compensatória associada, escolhemos os f primeiros serviços compensáveis (do tipo "*PassiveAction*" ou não) para serem executados.

No caso de termos escolhido a melhor relação custo/desempenho, devemos observar também o custo monetário dos serviços. Caso tenhamos serviços compensáveis e de custo zero, estes podem ser executados em paralelo. Porém, se não existirem serviços de custo zero, devemos realizar uma execução seqüencial para garantirmos a melhor relação custo/desempenho. Neste tipo de modo de execução, outras combinações possíveis poderiam ser observadas, como por exemplo, executar em paralelo serviços cujo custo fosse não nulo, embora muito próximo de zero. Entretanto, optamos pela execução concorrente apenas de serviços de custo zero, devido a dificuldade de se definir uma métrica que nos indicasse qual o custo mínimo, próximo de zero, até onde serviços seriam executados em paralelo.

Os serviços retornados pela função *ScheduleServices* são efetivamente executados (conforme representado na linha 5 do Algoritmo 1). Quando o primeiro dos serviços executados indica o término com sucesso de sua execução, é chamado o método *SelectWinnerService*, onde é feita a escolha do serviço vencedor.

Se o cliente tiver escolhido uma execução em *broadcast*, esperaremos pelo término da execução de todos os serviços e retornaremos os resultados de todos eles. Caso contrário, se apenas um serviço já tiver terminado, iremos determinar que esse serviço será o vencedor, retornando o seu resultado. Os demais serviços sendo executados (caso existam) serão abortados (se possível) ou compensados. Se mais de um serviço Web já tiver sido finalizado, escolheremos como serviço vencedor aquele que tiver o maior custo de compensação (para evitarmos o alto custo dessa compensação). Os demais serviços, também serão abortados ou compensados.

```
Método: SelectWinnerService
Entrada:    S = conjunto de serviços executados ou ainda em execução
Saída:      R = conjunto de resultados da execução do serviço.

1:    Se broadcast = true
2:        Esperar pelo término da execução de todos os serviços
3:        Adicionar a R o resultado da execução de todos os serviços
4:        Retornar R
5:    Fim-Se
6:    Se apenas 1 serviço Si em S já tiver terminado
7:        Adicionar o resultado de Si a R
8:        Abortar os serviços em estado de execução e compensar os demais serviços
9:        Retornar R
10:   Senão
11:       Abortar os serviços em estado de execução
12:       Para cada serviço Si em S que já tiver terminado
13:           Calcular seu custo de Compensação
14:       Fim-Para
15:       Sm ← Serviço com o maior custo de compensação
16:       Adicionar o resultado de Sm a R
17:       Abortar ou compensar os demais serviços
18:       Retornar R
19:   Fim-Se
```

Algoritmo 5 – Escolha do serviço vencedor

Após a execução do método *SelectWinnerService*, teremos os resultados da execução do serviço. Entretanto, caso nenhum serviço pertencente ao conjunto dos serviços escalonados na função *ScheduleServices* tenha sido finalizado com sucesso, precisamos escolher um novo conjunto de serviços para ser executado. Para isso, retiramos da lista de serviços candidatos aqueles que não foram finalizados com sucesso (conforme indicado na linha 10 do Algoritmo 1) e damos prosseguimento ao processo de escalonamento e execução de serviços, até que um desses seja executado com sucesso ou o número de serviços candidatos ainda não executados seja nulo. (conforme o *laço* mostrado entre as linhas 3 e 12 do Algoritmo 1). Será retornado como resposta da execução do serviço, o resultado do serviço vencedor (ou resultados, caso tenhamos mais de um serviço vencedor). Se nenhum serviço tiver sido finalizado com sucesso, um erro será gerado pela aplicação. Dessa forma, completamos o processo de execução de serviços.

5.5 – Monitoramento de Execuções na *WebTransact*

Conforme visto nas Seções 4.3 e 4.5, o modelo de custo utilizado para calcular o tempo total de execução dos serviços Web necessita de informações de custo e

qualidade, oferecidas pelo provedor do serviço na forma de um arquivo WSQD (*Web Services Quality Definitions*). Entretanto, alguns dos parâmetros utilizados no modelo (como a disponibilidade e a confiabilidade do serviço) não encontram-se presentes nesse arquivo, visto que são dependentes de execuções anteriores desse mesmo serviço. Sendo assim, precisamos realizar o monitoramento de execuções anteriores no *WebTransact*, para que possamos calcular esses valores e utiliza-los em nosso modelo.

Alguns trabalhos na literatura tratam do monitoramento de execuções de serviços Web (CHANDRASEKARAN *et al.*, 2002, DIALANI *et al.*, 2002). Entretanto, nenhum deles se preocupa em atualizar as características de qualidade desses serviços baseado nos resultados obtidos através desse monitoramento.

Nesta dissertação, utilizamos as informações obtidas através do monitoramento de execuções para verificar se determinados parâmetros oferecidos pelo provedor do serviço (como o tempo de execução) estão consistentes com os tempos obtidos em execuções anteriores. Caso não estejam, esses valores serão atualizados com os dados contidos nos registros de *log* obtidos através do monitoramento das execuções dos serviços. Isto irá conferir um caráter mais dinâmico a nosso modelo, levando a resultados mais confiáveis.

5.5.1 – Tipos de Finalização de um Serviço

Para calcularmos a disponibilidade de um serviço, precisamos saber, dado o número total de execuções já efetuadas para aquele serviço, quantas retornaram um erro de indisponibilidade (erro do tipo "Serviço Indisponível"). Já para o cálculo da confiabilidade, precisamos saber quantas execuções retornaram um erro "inesperado", ou seja, um erro não identificado nos elementos *fault* do documento WSDL que descreve o serviço.

De forma a obtermos essas informações, identificamos quatro possíveis tipos de finalização de uma operação (executada a partir de um serviço Web) :

- Serviço finalizado com sucesso: Indica que a operação foi finalizada com sucesso, retornando os resultados desejados;

- Serviço Indisponível: Indica que não foi possível fazer a comunicação com o provedor do serviço remoto;

- Erro Inesperado: A operação retornou um erro que não foi identificado nos elementos *fault* do documento WSDL que descreve o serviço;

- Erro Semântico: A operação retornou uma falha catalogada no documento WSDL que descreve o serviço. Apesar de não ser utilizado no cálculo da confiabilidade de um serviço, esse tipo erro também é monitorado nos arquivos de log do *WebTransact-EM.*

5.5.2 – Calculando a disponibilidade e a confiabilidade de um serviço

Para o cálculo da disponibilidade, levamos em conta o número de vezes em que conseguimos nos comunicar com o provedor do serviço (ou seja, o número total de execuções menos a quantidade de vezes em que obtivemos o erro de "Serviço Indisponível"). Então, dividimos esse valor pelo número total de execuções.

Sendo assim, temos que a disponibilidade D(S) de um serviço corresponde a:

$$D(S) = \frac{T - I}{T}$$

onde:

T = Número Total de Execuções de um Serviço.

I = Número de vezes em que obtivemos o erro de "Serviço Indisponível".

O cálculo da confiabilidade é feito de forma análoga. Entretanto, levamos em consideração o número de erros inesperados, ao invés do número de erros de indisponibilidade do serviço.

Sendo assim, temos que a confiabilidade C(S) de um serviço corresponde a:

$$C(S) = \frac{T - E}{T}$$

onde:

T = Número Total de Execuções de um Serviço.

E = Número de Erros Inesperados ocorridos na execução de um serviço.

5.5.3 – Monitoramento do Tempo de Resposta de um Serviço

Podemos monitorar o tempo de resposta da execução de operações de serviços e comparar o tempo médio obtido através dessas execuções com o tempo médio prometido pelo provedor do serviço no arquivo de informações de custo WSQD. Os valores existentes nesse arquivo podem ser atualizados a partir dos valores existentes no *log* de execuções, de forma a obtermos tempos de execução mais confiáveis e de acordo com a realidade.

Ao invés de gravarmos o tempo de cada execução de uma operação de um serviço, gravamos no arquivo de *log* apenas o tempo médio das execuções ocorridas até o momento (ou seja, caso três execuções tenham sido feitas obtendo-se os tempos de 1,6s , 1,1s e 1,2s, no arquivo de *log* estará gravado apenas o valor de 1,3s, correspondendo ao tempo médio dessas execuções). O parâmetro de sistema *UpdateFactor*, definido no arquivo de configuração do *WebTransact* indica o número de execuções de um serviço que devem ser feitas antes de atualizarmos o tempo de execução no arquivo WSQD. Esse número não deve ser muito pequeno, pois apenas após um número razoável de execuções de um serviço obteremos um tempo de resposta que realmente corresponda ao tempo médio de execução daquele serviço.

5.5.4 – Estrutura do Arquivo de *Log* do *WebTransact*

Os resultados do monitoramento de execuções são gravados em arquivos de *log*, que são documentos XML (sendo um para cada serviço) que contém as informações a respeito das execuções de serviços. A Figura 5.4 mostra a estrutura desses arquivos.

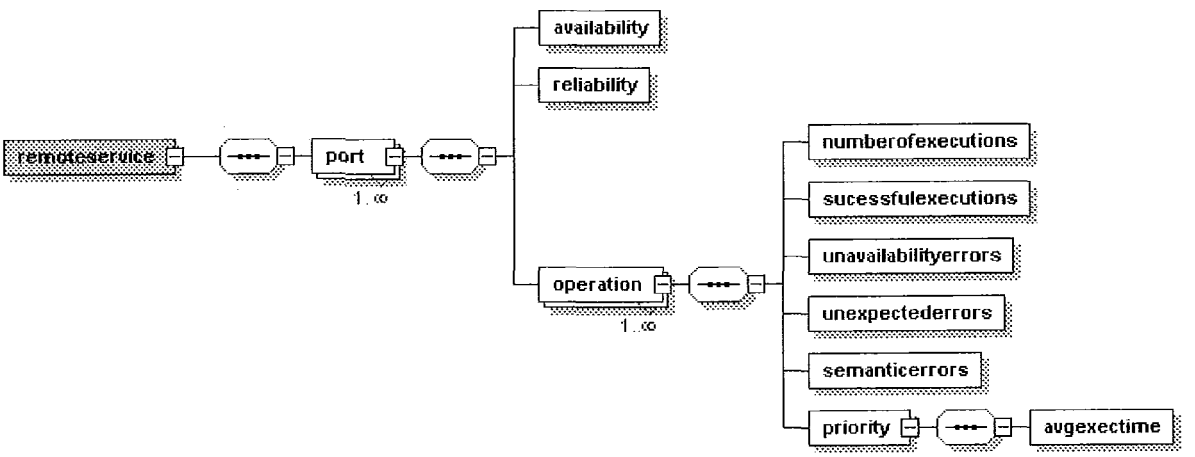


Figura 5.4 – Estrutura dos arquivos de *log* do *WebTransact*

O elemento raiz do documento, de nome "*remoteservice*", possui um atributo de nome *id*, que identifica unicamente o serviço remoto no sistema. Um serviço remoto pode possuir várias portas, representadas pelo elemento "*port*". Um elemento "*port*" também possui um atributo de nome *id*, que identifica unicamente essa porta dentro do serviço. Além disso, esse elemento possui ainda três subelementos :

- *availability*: Corresponde à disponibilidade da porta do serviço. Como os serviços Web podem possuir diferentes implementações em diferentes portas, os valores de disponibilidade e confiabilidade estão associados a cada porta do serviço. Inicialmente, o valor da disponibilidade é definido como sendo igual a um (100% de disponibilidade). Conforme execuções de serviços nessa porta vão ocorrendo, esse valor é atualizado de acordo com a fórmula mostrada na seção 5.5.2.

- *reliability*: Corresponde à confiabilidade da porta do serviço. Inicialmente, o valor da confiabilidade é definido como sendo igual a um (100% de confiabilidade). Conforme execuções de serviços nessa porta vão ocorrendo, esse valor é atualizado de acordo com a fórmula mostrada na seção 5.5.2.

- *operation*: Cada elemento "operation" corresponde à uma operação de um serviço Web. Esse elemento possui um atributo *name*, que indica o nome da operação.

Cada elemento "operation" possui ainda os seguintes subelementos:

- *numberofexecutions*: Corresponde ao número total de execuções dessa operação já efetuados.

- *successfulexecutions*: Corresponde ao número de execuções com sucesso dessa operação.

- *unavailabilityerrors*: Corresponde ao número de vezes em que obtivemos o erro de "Serviço Indisponível" na execução dessa operação.

- *unexpectederrors*: Corresponde ao número de vezes em que obtivemos um erro inesperado (erro não previsto) na execução dessa operação.

- *semanticerrors*: Corresponde ao número de vezes em que obtivemos um erro semântico na execução dessa operação.

- *priority*: Cada elemento "priority" corresponde a uma prioridade na execução dessa operação. Esse elemento possui dois atributos: *value* , que indica o valor da prioridade e *count* que indica quantas vezes a operação do serviço foi executada com essa prioridade.

O elemento "priority" possui ainda um subelemento "avgexectime" que corresponde ao tempo médio de execução dessa operação, dada a prioridade indicada através do atributo *value*. Esse tempo médio é calculado de acordo com a fórmula mostrada na Seção 5.5.3.

O Exemplo 5.1 mostra um exemplo de um arquivo de *log*.

```
<remoteservice id="rsBNQuote">
        <port id="BNQuotePort">
                <availability>1.0</availability>
                <reliability>1.0</reliability>
                <operation name="getPrice">
                        <numberofexecutions>6</numberofexecutions>
                        <successfulexecutions>6</successfulexecutions>
                        <unexpectederrors>0</unexpectederrors>
                        <semanticerrors>0</semanticerrors>
                        <unavailabilityerrors>0</unavailabilityerrors>
                        <priority count="4" value="Low">
                                <avgexectime>3.48525</avgexectime>
                        </priority>
                        <priority count="2" value="Normal">
                                <avgexectime>3.7305</avgexectime>
                        </priority>
                </operation>
        </port>
</remoteservice>
```

Exemplo 5.1 – Arquivo de log de um serviço no WebTransact

O Apêndice 2 mostra a listagem completa do XML Schema utilizado para representar os arquivos de log do *WebTransact*. No próximo capítulo, serão apresentados exemplos da utilização do modelo *WebTransact-EM*, utilizando os algoritmos apresentados na Seção 5.4.

6 – Exemplos de Utilização do Modelo *WebTransact-EM*

De forma a mostrarmos a utilização das principais funções da *WebTransact*, serão apresentados neste capítulo exemplos de serviços Web de funcionalidade equivalente e como a execução destes é efetuada nesta plataforma.

6.1 – Serviços de Validação de Cartão de Crédito

Neste exemplo, um mediador agrupa serviços de validação de cartão de crédito, onde possuímos uma única operação (de nome CheckCreditCardNumber, na interface do mediador) que recebe como parâmetro um número de cartão de crédito e o tipo do cartão (o código da operadora) e retorna se o cartão é válido ou não.

O Exemplo 6.1 mostra o arquivo WSTL do mediador *msCreditCard* (foram omitidos os espaços identificadores (*namespaces*)). A listagem completa dos arquivos utilizados neste exemplo pode ser visualizada no Apêndice 4.

```
<wstl:mediatorService id="msCreditCard">
        <wstl:operation name="CheckCreditCardNumber">
            <wstl:inputMsg>
                <wstl:param name="CCNumber" type="xsd:string"/>
                <wstl:param name="Type" type="xsd:string"/>
            </wstl:inputMsg>
            <wstl:outputMsg>
                <wstl:param name="validation" type="xsd:boolean"/>
            </wstl:outputMsg>
            <wstl:faultMsg errorCode="ERROR_101" description="Invalid card
type."/>
            <wstl:contentDescription medParam="inputMsg/@Type">
                <wstl:domain value="VISA"/>
                <wstl:domain value="AMEX"/>
                <wstl:domain value="MASTERCARD"/>
            </wstl:contentDescription>
        </wstl:operation>
</wstl:mediatorService>
```

Exemplo 6.1 – Arquivo WSTL do mediador msCreditCard

O elemento "*contentDescription*", definido para o parâmetro *type* identifica o domínio de valores aceitos pelo serviço. Sendo assim, somente cartões cujo tipo seja "VISA", "AMEX" ou "MASTERCARD" serão aceitos pelo mediador. Esse elemento também pode ser definido para um serviço remoto, indicando que aquele serviço

somente aceita um determinado conjunto de valores (no caso de serviços remotos, esse conjunto deve ser obrigatoriamente um subconjunto do domínio apresentado pelo mediador).

Em nosso exemplo, o mediador *msCreditCard* agrega quatro serviços, três deles remotos, já existentes, localizados através do registro UDDI, e um serviço local (*rsLocalCreditCard*), criado para efeitos de teste. Os nomes dos serviços remotos disponíveis na Web e os endereços de seus documentos WSDL estão listados a seguir:

- *rsMSugsCreditCard*
http://hosting.msugs.ch/eyesoft/ws/v1/ccvalidator/ccvalidator.asmx?WSDL

- *rsRichCardValidator*
http://www.richsolutions.com/richpayments/RichCardValidator.asmx?WSDL

- *rsCreditCardVerifier*
http://www.baccar-
inc.com/ws/Financial/CreditCardVerifier/CreditCardVerifier.asmx?WSDL

Para todos os serviços, foi definido um arquivo WSQD com as características de qualidade de cada serviço, a serem utilizadas pelo *WebTransact-EM*. A interface dos serviços remotos pode ser diferente daquela apresentada pelo mediador. Sendo assim, precisamos mostrar como o mapeamento entre as interfaces é efetuado, tanto para a mensagem a ser enviada ao provedor do serviço remoto quanto para a mensagem recebida como resposta do provedor. O exemplo 6.2 mostra a definição desse mapeamento para o serviço remoto *rsMSugsCreditCard*.

No Exemplo 6.2, o nome da operação é mapeado para "*CheckCardNumber*", enquanto o nome dos parâmetros é mapeado para "*cardNumber*" e "*cardType*". A operação do mediador retorna como resultado um valor *booleano* (verdadeiro ou falso). Entretanto, o serviço remoto retorna como resultado um valor igual a "*Valid Credit Card*" ou "*Invalid Credit Card*". Sendo assim, precisamos mapear esses valores para um valor do tipo *booleano*. Esse mapeamento é indicado através do elemento "*outputMap*" e a função responsável por essa tarefa tem o nome de "*ConvertCreditCardResults*".

```
<wstl:remoteService    id="rsMSugsCreditCard"          medServ="tns:msCreditCard"
                                                       portType="rs:CCCheckerSoap">
        <wstl:operationMap      name="CreditCardMap"
                                ptOperation="rs:CheckCardNumber
                                medOperation="tns:CheckCreditCardNumber">
            <wstl:inputMap>
                <wstl:paramMap          targetParam="cardNumber">
                    <wstl:sourceParam xpath="CCNumber"/>
                </wstl:paramMap>
                <wstl:paramMap          targetParam="cardType">
                    <wstl:sourceParam xpath="Type"/>
                </wstl:paramMap>
            </wstl:inputMap>
            <wstl:outputMap>
                <wstl:paramMap targetParam="validation"
                    mapFunction="ConvertCreditCardResults">
                    <wstl:sourceParam
                                                       xpath="CheckCardNumberResponse"/>
                </wstl:paramMap>
            </wstl:outputMap>
        </wstl:operationMap>
    </wstl:remoteService>
```

Exemplo 6.2 – Arquivo WSTL com as informações de mapeamento do serviço rsMSugsCreditCard

Além das informações sobre a interface do mediador e sobre como o mapeamento entre as interfaces é efetuado, precisamos das informações sobre os custos e critérios de qualidade aplicáveis a um serviço, presentes no arquivo WSQD de serviço (sendo um arquivo para cada porta contendo uma implementação desse serviço). O Exemplo 6.3 mostra o arquivo WSQD do serviço local *rsLocalCreditCard*. Por estar hospedado numa máquina local, consideramos o custo de transmissão de dados (indicado no elemento *network*) como sendo igual a zero. Este serviço é o único dos quatro agregados pelo mediador que permite sua execução sob uma prioridade alta. Dois serviços (*rsRichCardValidator* e *rsCreditCardVerifier*) são ainda considerados seguros (suas execuções são feitas sobre o protocolo HTTPS).

```
<definitions>
    <qualitydefinitions id="rsLocalCreditCardICreditCard" endpoint="CreditCard">
        <initTime unit="ms">100</initTime>
        <availability>1.0</availability>
        <reliability>1.0</reliability>
        <network>
            <sendingTime unit="ms">0</sendingTime>
        </network>
        <operation name="CheckCreditCardNumber">
            <requestLength unit="bytes">600</requestLength>
            <responseLength unit="bytes">700</responseLength>
            <priority value="normal">
                <execTime unit="ms">3000</execTime>
                <price unit="USD">0.001</price>
            </priority>
            <priority value="high">
```

```
                    <execTime unit="ms">1000</execTime>
                    <price unit="USD">0.03</price>
                </priority>
            </operation>
        </qualitydefinitions>
</definitions>
```

Exemplo 6.3 – Arquivo WSQD do serviço rsLocalCreditCard

Apresentados o mediador, os serviços remotos por ele agregados e os documentos de qualidade associados a esses serviços, serão mostrados agora as diferentes formas de execução que podem ser efetuadas no *WebTransact*-EM.

a) Minimizando o Custo Monetário.

Ao escolhermos o modo de execução "Minimizar Preço", estamos interessados em executar o serviço que possua o menor custo associado. A aplicação cliente pode ainda especificar qual o tempo de resposta máximo aceito por ela, e o preço máximo que ela está disposta a pagar pela execução do serviço, além de outros critérios como segurança e comportamento transacional do serviço. A partir dos critérios solicitados pela aplicação, será escolhido para execução o serviço de menor custo monetário associado que atenda a esses critérios.

Caso tenhamos uma solicitação de execução como a mostrada no Exemplo 6.4, o serviço escolhido para execução será *rsRichCardValidator*, pois este é o que apresenta o menor custo associado e atende aos demais critérios especificados pela aplicação. O serviço *rsLocalCreditCard*, apesar de possuir o menor custo monetário entre todos os serviços, não será escalonado para execução, pois não apresenta um mecanismo de segurança, que é essencial para a execução de acordo com a solicitação mostrada no Exemplo 6.4.

```
<operation name="CheckCreditCardNumber" maxresponsetime="2000" maxprice="0.005"
        executionmode="1" priority="normal" mandatorycompensate="false"
        security="yes" broadcast="no">
        ....
</operation>
```

Exemplo 6.4 – Exemplo de modo de execução igual a minimizar preço

A figura 6.1 mostra como essa solicitação é efetuada no *WebTransact-EM*. A partir dos dados selecionados na tela inicial da aplicação, é montado um arquivo XML como o do Exemplo 6.4. Finalmente, a figura 6.2 mostra o resultado dessa execução.

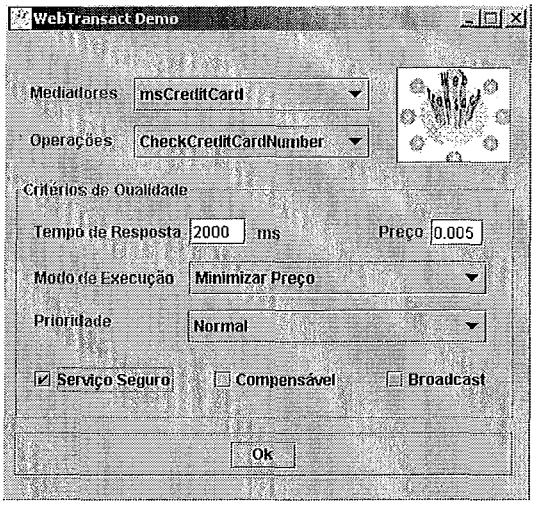


Figura 6.1 – Execução para "Minimizar Preço no WebTransact-EM.

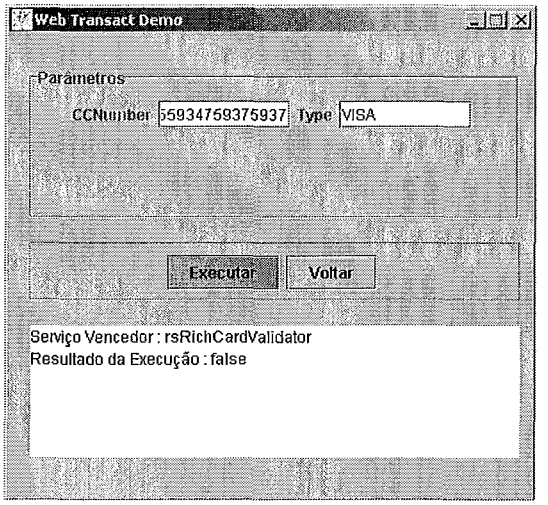


Figura 6.2 – Resultados da Execução para "Minimizar Preço"

b) Minimizando o Tempo de Resposta

Ao escolhermos o modo de execução "Minimizar Tempo de Resposta", estamos interessados em executar o serviço com o menor tempo de resposta associado. Nesse caso, temos a possibilidade de execução paralela de serviços, de forma que possamos obter esse melhor tempo. Como pela própria lógica da aplicação (validação de cartão de crédito) todos os serviços são compensáveis (sem a necessidade de execução de nenhuma operação compensatória), todos os serviços que atendam aos critérios escolhidos pela aplicação cliente serão escalonados para execução, sendo executados de forma paralela. Um dos serviços será eleito como o serviço vencedor, enquanto os demais terão sua execução cancelada ou compensada.

O Exemplo 6.5 ilustra uma solicitação de execução onde queremos minimizar o tempo de resposta. O serviço *rsCreditCardVerifier* não será escalonado para execução, pois seu custo monetário é maior que o indicado pela aplicação cliente. Nesta situação, os outros três serviços serão executados em paralelo.

```
<operation name="CheckCreditCardNumber" maxresponsetime="5000" maxprice="0.005"
        executionmode="2" priority="normal" mandatorycompensate="false"
        security="no" broadcast="no">
        ....
</operation>
```

Exemplo 6.5 – Exemplo de modo de execução igual a minimizar tempo de resposta

As figuras 6.3 e 6.4 mostram o resultado de uma execução feita no WebTransact-EM, gerando um arquivo XML como o do Exemplo 6.5. Nesse caso, apesar de termos executado todos os serviços em paralelo, o serviço vencedor

(rsLocalCreditCard no exemplo) foi o único a ter seus resultados retornados para a aplicação cliente.

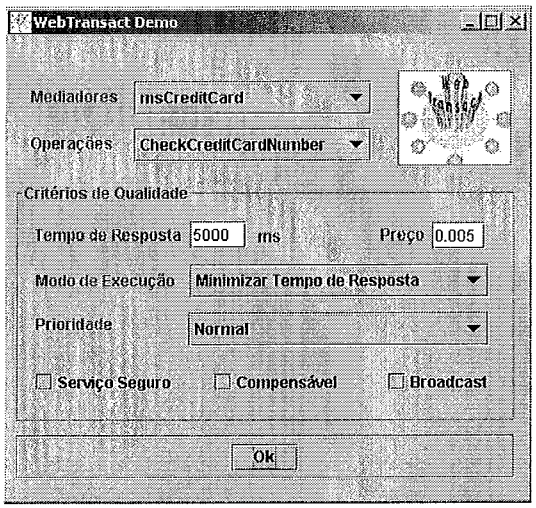


Figura 6.3 – Execução para "Minimizar Tempo de Resposta" no WebTransact-EM.

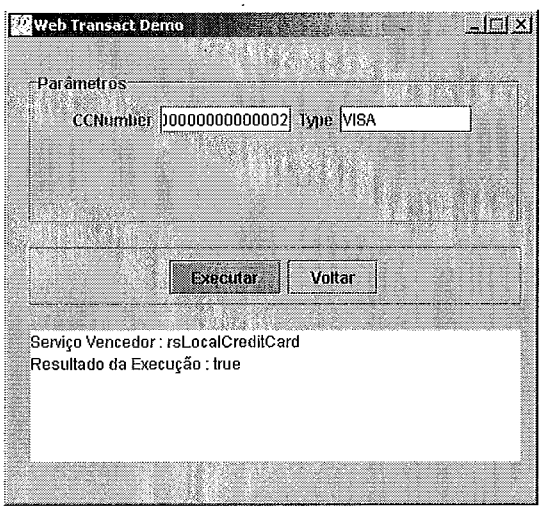


Figura 6.4 – Resultados da Execução para "Minimizar Tempo de Resposta"

c) Melhor Relação Custo/Desempenho

Neste caso, estamos interessados em executar o serviço com a melhor relação custo/desempenho. Como todos os serviços possuem um custo monetário associado, não teremos execução paralela neste caso, de acordo com o apresentado no Algoritmo 4.

Após a seleção dos serviços candidatos baseados nos parâmetros estabelecidos pelo usuário (tempo máximo de resposta, custo monetário máximo, segurança e comportamento transacional), será escolhido aquele que apresenta a melhor relação custo monetário por tempo de resposta. Para os serviços agrupados pelo mediador *msCreditCard*, o serviço *rsMSugsCreditCard* foi o que apresentou a melhor relação (com um tempo total de 1901 milissegundos e um custo monetário de 0,005 dólares). Caso algum erro ocorra na execução desse serviço, o segundo serviço de melhor relação será executado.

d) Execução de Serviço Sob Diferente Prioridade

Em todos os exemplos anteriores, os serviços estavam sendo executados sob uma prioridade normal. Entretanto, podem existir casos onde necessitamos de um tempo de resposta ainda melhor na validação do cartão de crédito, onde queremos que nossa execução receba uma atenção especial por parte do provedor do serviço.

Nestes casos, caso o serviço ofereça essa diferenciação, podemos fazer com que a execução do serviço aconteça sob uma prioridade diferente (mais alta ou mais baixa). Execuções mais prioritárias possuem um custo maior e um melhor tempo de resposta, enquanto execuções menos prioritárias possuem um baixo custo e um tempo de resposta inferior.

O Exemplo 6.6 ilustra uma solicitação de execução de alta prioridade, com um tempo máximo de resposta de 1 segundo e onde aceitamos pagar um valor máximo de 5 centavos de dólar (apesar de querermos minimizar o custo monetário dessa execução) :

```
<operation name="CheckCreditCardNumber" maxresponsetime="1000" maxprice="0.05"
        executionmode="1" priority="high" mandatorycompensate="false"
        security="no" broadcast="no">
        ....
</operation>
```

Exemplo 6.6 – Execução com prioridade alta.

O único serviço que suporta os critérios de qualidade especificados pela aplicação cliente, fazendo uma diferenciação por prioridade é o serviço *rsLocalCreditCard*. Esse será o único serviço candidato à execução selecionado pelo *WebTransact-EM*. Conforme especificado na Seção 5.3, a prioridade desejada na execução de uma operação é transmitida no cabeçalho da mensagem SOAP a ser enviada ao serviço remoto. O Exemplo 6.7 mostra a mensagem SOAP gerada, para uma execução com uma prioridade alta.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Header>
                <t:QoS SOAP-ENV:mustUnderstand="1">
                        <priority>High</priority>
                </t:QoS>
        </SOAP-ENV:Header>
        <SOAP-ENV:Body>
                <m:CheckCreditCardNumber
xmlns:m="http://localhost:8080/axis/CreditCard.jws">
                        <CCNumber xsi:type="xsd:string">4000000000000002</CCNumber>
                        <Type xsi:type="xsd:string">VISA</Type>
                </m:CheckCreditCardNumber>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo 6.7 –Mensagem SOAP com cabeçalho indicando o valor de prioridade.

6.2 – Livraria Virtual

Neste exemplo, temos serviços que executam funções equivalentes a uma livraria virtual, onde existem três operações distintas: uma para a obtenção do preço de um livro dado seu número ISBN, outra para a compra de um livro e por último, uma operação para o cancelamento de uma compra previamente efetuada. O objetivo é verificar como o WebTransact-EM se comporta quando queremos executar todos os serviços de um mediador (*execução em broadcast*) e mostrar o funcionamento de serviços compensáveis, fazendo do comportamento transacional mais um critério de qualidade aplicável a um serviço.

Uma aplicação cliente poderia pesquisar o preço de um livro em todos os serviços existentes e efetuar a compra naquela com o preço mais interessante. Da mesma forma, poderia cancelar essa compra no caso de qualquer imprevisto. Sendo assim, uma aplicação cliente poderia não estar interessada em parâmetros como o tempo de resposta ou o custo monetário de execução do serviço (que possivelmente, deverá ser nulo para este tipo de serviço). Entretanto, parâmetros como segurança e comportamento transacional de um serviço continuam sendo importantes neste domínio de aplicação (poderiam ser selecionados apenas serviços seguros e que sejam compensáveis, ou seja, cujo processo de compra possa ser cancelado a partir da execução de uma outra operação, restituindo os valores já pagos ao cliente).

O mediador *msBookServices*, mostrado no Exemplo 6.8, agrega quatro serviços distintos, de quatro livrarias diferentes: LivrosSA , LivrariaWeb, WebVendas e BookExpress. As três primeiras livrarias listadas implementam a mesma interface de serviço. Neste caso, temos um mesmo documento WSDL com portas diferentes, cada uma representando um serviço. A livraria BookExpress implementa uma interface, diferente, oferecendo apenas duas operações: GetBookPrice e BuyBook. Assim, temos que a compra de livros nessa livraria representa uma operação não compensável, ou seja, não é possível cancelarmos uma compra previamente efetuada, visto que não é disponibilizada uma operação com tal finalidade.

```
<wstl:mediatorService id="msBookServices">
        <wstl:operation name="ObterPrecoLivro">
            <wstl:inputMsg>
                <wstl:param name="ISBN" type="xsd:string"/>
            </wstl:inputMsg>
            <wstl:outputMsg>
                <wstl:param name="preco" type="xsd:float"/>
            </wstl:outputMsg>
            <wstl:faultMsg errorCode="ERROR_101" description="ISBN Inválido."/>
        </wstl:operation>
        <wstl:operation name="ComprarLivro">
            <wstl:inputMsg>
                <wstl:param name="ISBN" type="xsd:string"/>
                <wstl:param name="NumeroCartao" type="xsd:string"/>
                <wstl:param name="Tipo" type="xsd:string"/>
                <wstl:param name="Nome" type="xsd:string"/>
                <wstl:param name="Livraria" type="xsd:string"/>
            </wstl:inputMsg>
            <wstl:outputMsg>
                <wstl:param name="idcompra" type="xsd:string"/>
            </wstl:outputMsg>
            <wstl:faultMsg errorCode="ERROR_102" description="Cartão
Inválido."/>
            <wstl:faultMsg errorCode="ERROR_103" description="Tipo de cartão
não aceito."/>
            <wstl:contentDescription medParam="inputMsg/@Livraria">
                <wstl:domain value="LivrosSA"/>
                <wstl:domain value="WebVendas"/>
                <wstl:domain value="LivrariaWeb"/>
                <wstl:domain value="BookExpress"/>
            </wstl:contentDescription>
        </wstl:operation>
        <wstl:operation name="CancelarOperacao">
            <wstl:inputMsg>
                <wstl:param name="idcompra" type="xsd:string"/>
                <wstl:param name="Livraria" type="xsd:string"/>
            </wstl:inputMsg>
            <wstl:outputMsg>
                <wstl:param name="retorno" type="xsd:string"/>
            </wstl:outputMsg>
            <wstl:faultMsg errorCode="ERROR_104" description="O período de
cancelamento expirou."/>
            <wstl:contentDescription medParam="inputMsg/@Livraria">
                <wstl:domain value="LivrosSA"/>
                <wstl:domain value="WebVendas"/>
                <wstl:domain value="LivrariaWeb"/>
            </wstl:contentDescription>
        </wstl:operation>
    </wstl:mediatorService>
```

Exemplo 6.8 – Arquivo WSTL do mediador msBookServices.

Para os serviços que permitem o cancelamento de uma compra (LivrosSA, LivrariaWeb e WebVendas), é disponibilizado um método de nome "CancelarOperação", que representa um operação compensatória em relação a operação "ComprarLivro". O Exemplo 6.9 mostra o fragmento de arquivo WSTL

contendo a definição do comportamento transacional de cada serviço que permite esta operação compensatória. Neste arquivo, é indicado que a operação "ComprarLivro" é compensável, tendo como operação compensatória a operação "CancelarOperação".

```
<wstl:transactionDefinitions>
        <wstl:transactionBehavior operationName="ComprarLivro"
                                                    type="compensable">
            <wstl:activeAction      portTypeName="bookServices"
                                    compensatoryOper="CancelarOperacao">
                <wstl:paramLink>
                    <wstl:sourceParamLink
                                msgName="ComprarLivroResponse"
                                param="intf:ComprarLivroReturn"/>
                    <wstl:targetParamLink
                                msgName="CancelarOperacaoRequest
                                param="intf:id"/>
                </wstl:paramLink>
            </wstl:activeAction>
        </wstl:transactionBehavior>

        <wstl:transactionBehavior operationName="CancelarOperacao"
                                type="retriable"/>
</wstl:transactionDefinitions>
```

Exemplo 6.9 –Arquivo WSTL com o comportamento transacional das operações .

Uma aplicação cliente, inicialmente, poderia querer pesquisar o preço de um determinado livro em todas as livrarias disponíveis. Assim, poderia ser feita uma escolha daquela livraria onde o livro encontra-se à venda com o menor preço. No *WebTransact-EM*, isso é possível através de uma execução em *broadcast* de todos os serviços agregados pelo mediador *msBookServices*. Neste caso, todos os serviços agregados pelo mediador serão executados em paralelo, com cada um deles retornando o preço do livro indicado através do parâmetro ISBN. Assim, não precisaríamos identificar o tempo de resposta máximo nem o custo monetário, visto que todos os serviços serão executados. As figuras 6.5 e 6.6 mostram um exemplo desse tipo de pesquisa. No exemplo, a livraria que apresenta o menor preço para o livro foi a LivrosSA.

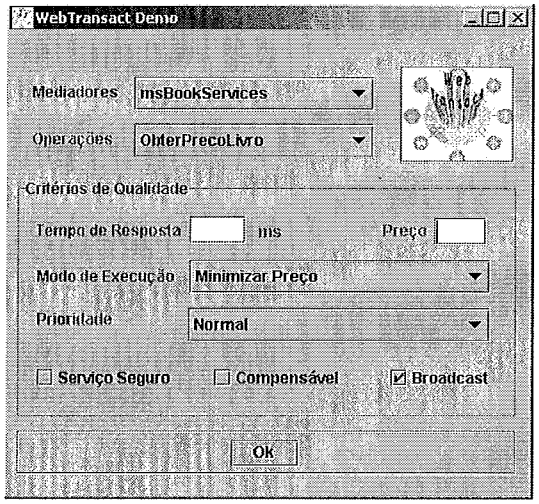


Figura 6.5 – Execução em *broadcast* em todas livrarias

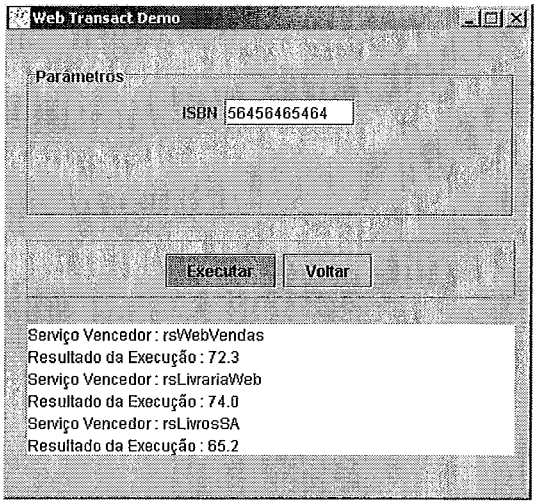


Figura 6.6 – Resultados da Execução em *broadcast*

Após todos os serviços terem retornado o preço do livro, a aplicação cliente seleciona em qual livraria quer efetuar a compra. Essa livraria é indicada através do parâmetro "Livraria", da operação "ComprarLivro" do mediador. Caso o cliente queira ter a possibilidade de cancelar a compra do livro após um determinado período de tempo, a livraria "BookExpress" não poderá ser a escolhida, visto que ela não permite essa funcionalidade. Uma aplicação cliente indica que quer ter a possibilidade de compensar essa operação definindo o parâmetro *mandatorycompensate* como sendo verdadeiro na solicitação de execução da operação "ComprarLivro".

Finalmente, se desejar cancelar a compra efetuada, o cliente poderá executar a operação "CancelarOperação", indicando além do número de identificação da compra, a livraria onde tal compra foi efetuada.

Através dos exemplos pudemos verificar a adequação do *WebTransact-EM* na seleção e dinâmica de serviços. Com o primeiro exemplo, mostramos a utilidade dos critérios de qualidade, especialmente, o tempo de resposta e o custo monetário. Sem nosso modelo, uma aplicação deveria pesquisar em todos os serviços de validação de cartão de crédito por aquele com o menor custo monetário ou com o menor tempo de execução (ou ainda, com a melhor relação entre ambos) e executá-lo de forma explícita. Dada a potencial existência de vários serviços equivalentes, a busca através destes ou de outros critérios de qualidade (como a segurança, importante nesse domínio de aplicação), poderia ser extremamente complexa. A possibilidade de execução paralela de serviços (no caso de termos selecionado o menor tempo de resposta), livra a aplicação cliente de ter de se preocupar com este critério durante

uma execução. Mostramos ainda, como prioridades mais altas (ou mais baixas) podem ser facilmente selecionadas através de nosso modelo (com a inclusão da prioridade no cabeçalho da mensagem SOAP enviada ao provedor do serviço).

Com o segundo exemplo, mostramos a adequação das execuções em *broadcast*, livrando uma aplicação cliente de ter de invocar vários serviços, com uma execução para cada serviço de funcionalidade equivalente. Mostramos ainda, a utilização de serviços compensáveis, e como o comportamento transacional de serviços pode ser utilizado como mais um critério de qualidade. No exemplo, dado que queremos ter a possibilidade de cancelar a compra após um determinado período de tempo, somente serviços compensáveis são candidatos à execução.

Como na execução de um serviço Web, os custos dominantes são os custos de rede e de execução, o *overhead* causado pela aplicação de nosso modelo de execução pode ser considerado irrelevante. Entretanto, para a execução de serviços com baixo tempo de resposta em redes de alta velocidade de transmissão, a aplicação do modelo pode causar um atraso indesejável, talvez sendo mais conveniente a execução de um serviço específico do que a própria aplicação do modelo.

Estando implantado sobre uma plataforma que permita a composição de serviços, o mecanismo de seleção e execução dinâmica oferecido pelo *WebTransact-EM* facilita esse processo de composição, possibilitando a escolha de serviços de acordo com seus critérios de qualidade, com classes de serviços equivalentes sendo incluídas na composição ao invés de serviços específicos, tornando a plataforma mais flexível e adaptável a mudanças.

7 – Conclusões

A tecnologia de serviços Web vem tornando mais simples a interação entre aplicações e serviços desenvolvidos por diferentes organizações, dada a grande interoperabilidade por ela proporcionada. Vários serviços estão disponíveis atualmente na Web, podendo ser utilizados por diferentes aplicações e empresas. Sendo a tecnologia de serviços Web baseada no uso de componentes fracamente acoplados, serviços podem ser escolhidos e acoplados a uma aplicação cliente em tempo de execução, dando um caráter mais dinâmico a esta tecnologia.

Um problema presente atualmente nos mecanismos para a execução de serviços Web é a necessidade de se explicitar a localização, interface, comportamento e outras propriedades de uma implementação específica de um serviço remoto quando queremos utilizá-la em uma aplicação. Certamente, soluções mais flexíveis e que sejam mais facilmente adaptáveis a mudanças são necessárias para facilitar o uso desta tecnologia, possibilitando que serviços sejam dinamicamente escalonados em tempo de execução. Além disso, dada a existência de vários serviços semanticamente equivalentes na Web, precisamos de algum mecanismo que possibilite a escolha em tempo de execução de um ou mais desses serviços para ser executado. Apesar de alguns avanços já terem sido feitos na área de especificação de serviços (CHRISTENSEN *et al.*, 2001) e na sua composição (LEYMANN, 2001, THATTE, 2001, CURBERA *et al.*, 2002, PIRES,2002), o problema da seleção e execução dinâmica de serviços tem recebido pouca atenção. Atualmente, não existem modelos de execução para serviços Web que, dado um conjunto de serviços de funcionalidade equivalente, realize o processo de escolha de qual serviço executar, baseado nos critérios de qualidade de cada serviço Web.

Nesta dissertação, propomos um modelo para a execução dinâmica de serviços Web que leva em consideração a existência de serviços de funcionalidade equivalente, sendo baseado nas características de qualidade e parâmetros de custo aplicáveis a um serviço. Alguns trabalhos na literatura permitem a execução dinâmica de serviços (KEIDL *et al.*, 2002a, TOSIC *et al.*, 2001), porém, eles não consideram serviços com diferentes interfaces durante o processo de seleção de serviços candidatos à execução. A partir do agrupamento de serviços de funcionalidade equivalente em classes, é necessário a uma aplicação cliente apenas especificar a classe de serviços que deseja executar e os critérios de qualidade que um serviço

deve atender. É tarefa do modelo de execução proposto a decisão de quais serviços dentro dessa classe devem ser executados.

Foram identificados os parâmetros de custo e critérios de qualidade aplicáveis a serviços Web e suas operações, de forma que o processo de seleção dinâmica de serviços possa se basear nesses critérios. Alguns trabalhos na literatura (MANI e NAGARAJAN, 2002, SAHAI *et al.*, 2001) também identificam critérios de qualidade de serviços Web, porém, eles não utilizam tais critérios na elaboração de um modelo de execução que selecione serviços baseados nesses critérios. Além de características como o tempo de resposta e o custo monetário de execução dos serviços, foram identificados outros critérios importantes para serem utilizados em nosso modelo, como a disponibilidade e a confiabilidade de um serviço, aspectos de segurança e custos de transmissão. Aplicações clientes podem especificar restrições sobre esses parâmetros de qualidade, limitando o número de serviços candidatos à execução, facilitando o trabalho do modelo proposto. De forma a padronizar a forma como estes critérios são apresentados a uma aplicação cliente, propomos a linguagem WSQD (Web Services Quality Definitions), que representa esses critérios para uma implementação específica de um serviço.

Uma vez apresentados os critérios de qualidade aplicáveis a serviços Web, propomos o modelo de execução *WebTransact-EM*. Em (HOSCHEK, 2002), são identificadas as etapas para a execução de serviços Web, desde a localização de serviços, até a apuração de seus resultados, sem porém, uma implementação dessas etapas. Nesta dissertação, elaboramos um modelo de execução composto por quatro diferentes fases, também contempladas em (HOSCHEK, 2002): seleção dos serviços candidatos à execução (baseado nos critérios de qualidade solicitados por uma aplicação cliente), ordenação desses serviços, o escalonamento destes, e a escolha de um serviço vencedor. A elaboração de tal modelo constitui uma tarefa complexa, dada a grande variedade de planos de execução que pode ser gerada a partir do escalonamento de serviços, dependendo das restrições feitas por uma aplicação cliente. Além da dificuldade na geração de um plano que atenda a essas restrições, no caso de execução paralela de serviços, deve ser realizado um controle transacional sobre essas execuções, de forma que apenas um serviço tenha seus resultados consolidados (excetuando-se o caso de execuções em broadcast, onde todos os serviços têm seus resultados consolidados).

A utilização de tal modelo em uma plataforma que suporte a especificação de composições de serviços Web facilita o uso desta tecnologia, dado que não precisamos explicitar qual serviço utilizar numa composição (basta ser indicada que classe de serviço deve ser executada). Para verificarmos a validade de nosso modelo, este foi implementado na plataforma WebTransact (PIRES, 2002, PIRES *et al.* 2002, PIRES *et al.*, 2003), que permite a composição de serviços de forma confiável, dada a existência de um mecanismo de gerenciamento de transações. Foram apresentados exemplos de serviços semanticamente equivalentes sendo executados utilizando nosso modelo, mostrando sua utilidade na geração de diferentes planos de execução dependendo das restrições estabelecidas por aplicações clientes. Acreditamos que a maior flexibilidade na seleção e execução de serviços Web, proporcionada através da utilização deste modelo de execução, é um grande diferencial que pode tornar a tecnologia de serviços Web mais atraente para um número cada vez maior de organizações.

Nas próximas seções, mostraremos as principais contribuições desta dissertação e sugestões de trabalhos futuros.

7.1 – Contribuições

a) *Identificação de parâmetros de qualidade aplicáveis a serviços Web e suas operações*

Neste trabalho, foram identificados os parâmetros de custo e critérios de qualidade aplicáveis a serviços Web e suas operações. A representação desses parâmetros foi padronizada numa linguagem baseada em XML conhecida como *Web Services Quality Definitions*.

Identificamos um serviço como um sendo uma porta (um elemento *"port"*) de um documento WSDL, ao contrário da representação normalmente utilizada, que identifica um serviço como um conjunto de portas. Tal diferenciação se faz necessária devido ao fato de uma porta referenciar um endereço físico (um computador na rede), representando uma implementação, e diferentes implementações podem apresentar custos e critérios de qualidade distintos, que irão afetar a seleção dinâmica de serviços e as estratégias sendo propostas. Portanto, quando dizemos que iremos executar um serviço, na verdade estaremos executando uma implementação deste, referenciada por uma porta no documento WSDL.

b) *Definição da linguagem WSQD (Web Services Quality Definitions)*

De forma a representar de uma forma padronizada os parâmetros de custo e de qualidade aplicáveis a serviços e suas operações, foi proposta a linguagem WSQD (*Web Services Quality Definitions*). Cada documento WSQD está associado a um elemento "port" de um documento WSDL, representando as características de custo e qualidade aplicáveis àquela implementação de um serviço Web. Idealmente, esperamos que os provedores de serviço ofereçam às aplicações cliente informações de custo num documento semelhante a um arquivo WSQD, da mesma forma que as informações sobre serviço, suas operações e suas implementações são descritas num arquivo WSDL.

c) *Implementação da plataforma WebTransact*

A plataforma WebTransact, definida em (PIRES, 2002,) e apresentada no Capítulo 3, teve seu projeto e sua implementação realizados como parte desta dissertação. Foram desenvolvidas as camadas de serviço remoto e mediadores, de forma que pudesse ser trabalhado o processo de agrupamento de serviços de funcionalidade equivalente. O desenvolvimento da camada de composição de serviços da plataforma permanece como um trabalho futuro.

d) *Algoritmos para execução dinâmica de serviços Web*

Quando dispomos de vários serviços Web de funcionalidade equivalente que podem ser executados, a seleção de qual (is) serviço(s) executar não é uma tarefa simples. De forma a resolver esse problema, foram propostos algoritmos para a execução dinâmica de serviços Web, baseados nos parâmetros de custo e qualidade selecionados pelo usuário e nas características de cada serviço. Estes foram implementados na plataforma *WebTransact*, de forma a terem seus resultados validados.

De uma forma simplificada, o processo de execução dinâmica de serviços Web de funcionalidade equivalente contempla as seguintes etapas: a localização de serviços (feita atualmente de forma manual na *WebTransact*), o agrupamento de serviços Web de funcionalidade equivalente, utilizando-se a tecnologia de mediadores,

a seleção de serviços candidatos à execução baseado em seus critérios de qualidade, a ordenação desses serviços, o escalonamento de um ou mais serviços para serem executados e a seleção de um ou mais serviços vencedores que terão seus resultados consolidados.

Com a utilização dos algoritmos propostos, o processo de seleção e execução dinâmica de serviços Web torna-se um processo transparente para uma aplicação cliente. Tal processo pode ser utilizado em composições de serviços, onde ao invés de especificarmos um serviço específico em uma composição, identificamos apenas uma classe de serviços semanticamente equivalentes, com os serviços a serem efetivamente executados sendo apenas escalonados em tempo de execução, dependendo de seus critérios de qualidade e as restrições especificadas por uma aplicação cliente.

e) *Definição de modelo para o log de execução da WebTransact*

Poucos trabalhos na literatura abordam a questão do monitoramento de execuções de serviços Web (DIALANI *et al.*, 2002 , CHANDRASEKARAN *et al.*, 2002). Entretanto, em nenhum deles é definido um modelo para um log de execuções desses serviços, de forma a realizar esse monitoramento de execuções de uma maneira padronizada.

Nesta dissertação, propomos um modelo baseado na linguagem XML de log de execuções de serviços Web. A cada serviço está associado um arquivo de log, indicando os resultados das execuções nas portas destes serviços. Como possíveis tipos de finalização de uma execução, temos: execução com sucesso, erro de indisponibilidade, erro semântico e erro inesperado. Esses tipos são utilizados como base para o cálculo da disponibilidade e confiabilidade de um serviço, conforme discutido na Seção 5.5.

7.2 – Trabalhos Futuros

A tecnologia de serviços Web viabiliza a interoperabilidade entre aplicações que são acessadas via Web, fornecendo a infraestrutura básica para construção de serviços mais complexos, compostos de outros serviços mais simples. Contudo, somente a viabilização da comunicação de forma interoperável entre programas

clientes e serviços Web não é suficiente para o desenvolvimento de composições de serviços que precisam ser robustas, flexíveis e escaláveis.

Por ser a tecnologia de serviços Web ainda recente, várias pesquisas vêm sendo desenvolvidas nessa área, mais especificamente na parte de composição de serviços. Nesta dissertação, buscamos resolver o problema da execução dinâmica de serviços Web semanticamente equivalentes, utilizando características de custo e qualidade de serviços em nosso processo de decisão. Entretanto, outros trabalhos ligados à área de composição de serviços Web ainda permanecem em aberto. Alguns destes trabalhos, também relacionados à plataforma *WebTransact*, são listados a seguir.

a) Finalização da implementação da plataforma WebTransact

A plataforma *WebTransact* possibilita a execução e composição de serviços Web de forma dinâmica (utilizando as estratégias para seleção e execução dinâmica de serviços discutidas nesta dissertação) e confiável (através do gerenciamento de transações tendo por base a linguagem WSTL, definida em (PIRES,2002)).

A arquitetura da *WebTransact* é dividida em três camadas: uma para a representação de serviços remotos, uma para a definição de mediadores e uma para a composição de serviços. Nesta dissertação, foram implementadas as camadas de serviços remotos e mediadores, desenvolvidas em Java com o auxílio da biblioteca Apache Axis (APACHE GROUP, 2002) para a criação das mensagens SOAP a serem enviadas aos provedores de serviços remotos. A camada de composição de serviços, bem como a tarefa de gerenciamento de transações permanecem em aberto, constituindo-se de possíveis trabalhos futuros.

b) Aplicação dos parâmetros de custo e qualidade em composições de serviços

Uma tarefa pode ser vista como uma aplicação composta por vários serviços Web. Ao invés de aplicarmos os parâmetros de custo e qualidade em apenas um único serviço (um único mediador), podemos aplicá-los em uma tarefa. Assim, poderíamos definir que uma tarefa deve ser executada em um tempo de resposta máximo selecionado, que aceitamos pagar uma determinada quantia máxima pela execução de uma tarefa e assim por diante, para os demais critérios de qualidade.

As estratégias para seleção dinâmica de serviços seriam primeiro aplicadas, de forma individual, a cada serviço (representado por um mediador) que faz parte de uma determinada tarefa, sendo depois aplicadas, de uma forma mais genérica, a uma tarefa como um todo. Como para o caso de composições de serviços, temos a possibilidade de execução paralela de serviços contidos numa aplicação (discutida na Seção 4.4), este torna-se um problema mais interessante, tornando mais complexa a aplicação das estratégias definidas nesta dissertação.

c) Localização automática de serviços

Atualmente, o processo de localização de serviços de funcionalidade equivalente é feito de forma manual na *WebTransact*. O desenvolvimento de uma ferramenta que realize uma busca desses serviços e os integre através de um mediador é de grande importância para uma utilização mais fácil das estratégias aqui apresentadas. Dada a dificuldade de se expressar semanticamente uma consulta a serviços de funcionalidade equivalente sob o registro UDDI, algumas técnicas que possibilitem uma maior capacidade semântica a esse registro devem ser utilizadas, como as apresentadas em (ANKOLEKAR *et al.*, 2001, MCILRAITH *et al.*, 2001, TRASTOUR *et al.*, 2001).

d) Qualidade de Dados

Um serviço Web pode ter sido finalizado com sucesso; entretanto, os resultados retornados por ele podem não ter sido aqueles desejados por uma aplicação cliente. Isto se deve principalmente, à baixa confiabilidade das fontes de dados do provedor de serviço, que podem não atender ao grau de exigência de uma aplicação cliente.

A cada resultado de uma execução com sucesso de um serviço, seriam atribuídos valores que categorizassem aquele resultado. Quanto mais altos esses valores, melhor seria a avaliação desse serviço por parte de uma aplicação cliente. Este poderia ser mais um critério de qualidade aplicável a uma operação de um serviço web, com serviços sendo escolhidos tendo por base esses valores de qualidade atribuídos por uma aplicação cliente em execuções anteriores. Alguns trabalhos na literatura já abordam técnicas de como quantificar os resultados obtidos na execução de serviços (MECELLA *et al.*, 2002, SCANNAPIECO *et al.*,2002).

Um exemplo de utilização deste critério seria para o caso de serviços de meteorologia, onde quanto maior a qualidade dos dados retornados por um serviço, mais indicada seria a sua execução em detrimento de outros serviços, possivelmente com menor custo monetário e tempo de resposta.

e) Integração com provedores de estatísticas

Idealmente, os parâmetros de custo e qualidade devem ser fornecidos pelos provedores de serviços Web, da mesma forma que informações sobre as interfaces e operações de um serviço são disponibilizadas através de um documento WSDL.

Entretanto, sistemas provedores de estatísticas, que inferem os custos associados a determinadas fontes de dados na Web também podem ser utilizados na tarefa de obtenção desses custos. Como exemplo de tal sistema, podemos citar o DIG (*Distributed Informathion Gathered*) (RUBERG *et al.* ,2002), um provedor de custos e estatísticas que auxilia o processo de otimização global de consultas em um ambiente distribuído, com fontes de dados autônomas e distribuídas. Este sistema poderia ser estendido para possibilitar a obtenção de custos relativos a serviços Web, de forma que pudesse ser agregado a plataformas como a *WebTransact*, para a obtenção desses custos.

f) Utilização das estratégias para seleção e execução de serviços em outros ambientes computacionais

As estratégias para seleção e execução dinâmica de serviços apresentadas nesta dissertação foram aplicadas na plataforma WebTransact, em um ambiente Web. Entretanto, elas podem ser úteis também em outras plataformas da computação distribuída, como por exemplo, no ambiente de *grid* de computadores (FOSTER e KESSELMAN, 1999). Um grid oferece mecanismos para acoplar recursos geograficamente distribuídos de forma consistente, provendo a execução paralela de tarefas em qualquer máquina que dele faça parte. Permite ainda, o compartilhamento, seleção e agregação de uma grande variedade de recursos computacionais, como processadores, sistemas de armazenamento, fontes de dados, entre outros.

Recentemente, a área de *grid* de computadores tem usado a tecnologia de serviços Web para garantir a interoperabilidade na comunicação entre esses recursos

(FOSTER *et al.*, 2002, MOREAU, 2002). As estratégias aqui apresentadas podem ser úteis para auxiliar a decidir em qual máquina alocar uma tarefa num *grid*. Além disso, uma linguagem como a WSQD poderia ser utilizada para representar os aspectos de qualidade de um determinado recurso de um *grid*.

Referências Bibliográficas

ADALI, S., CANDAN, K. S., PAPAKONSTANTINOU, Y., *et al.*, 1996, "Query Caching and Optimization in Distributed Mediator Systems". In: *Proceedings of SIGMOD Conference on Management of Data*, pp. 137-148, Montreal, Canada, June.

AIELLO, M., PAPAZOGLOU., M., CARMAN, M., *et al.* , 2002 , "A request language for web-services based on planning and constraint satisfaction". *In: Proc. of the VLDB Workshop on Technologies for E-Services (TES02)*, pp. 76-85, Hong Kong, China, August.

ALONSO, G., AGRAWAL, D., ABBADI, A., 1996, "Process Synchronization in Workflow Management Systems". *8th IEEE Symposium on Parallel and Distributed Processing*, New Orleans, USA, October.

ANKOLEKAR, A., BURSTEIN, M., HOBBS, J., *et al.*, 2001, "DAML-S: Semantic Markup for Web Services". In: *Proceedings of the International Semantic Web Working Symposium*, Stanford, CA, USA, August.

APACHE GROUP, 2002, "The Axis Project", http://xml.apache.org/axis

ATKINSON, B., DELLA-LIBERA, G., HADA, S., *et al.*, 2002, "Web Services Security (WS-Security) 1.0". Disponível em: http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

AUSTIN, D., BARBIR, A., FERRIS, C., *et al.*, 2002, *Web Services Architecture Requirements*, W3C Working Draft, August.

BERNERS-LEE, T., HENDLER, J., LASSILA, O., 2001, "The Semantic Web". *Scientific American*, 284(5), pp. 34-43.

BOAG, S., CHAMBERLIN D., FERNANDEZ, M., *et al.*, 2002, *XQuery 1.0: An XML Query Language*, W3C Working Draft, November. Disponível em: http://www.w3.org/XML/Query

BOULOS, J., ONO, K., 1999, "Cost Estimation of User-Defined Methods in Object Relational Database Systems", *SIGMOD Record* 28(3), pp. 22-28.

BOX, D., EHNEBUSKE, D., KAKIVAYA, A., *et al.*, 2000, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, May. Disponível em http://www.w3.org/TR/soap

BRAUMANDL, R., KEIDL, M., KEMPER, A., *et al.*, 2001, "ObjectGlobe: Ubiquitious Query Processing on the Internet". *VLDB Journal*, n.10, pp. 48-71, Springer-Verlag.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, M., *et al.*, 2000, *Extensible Markup Language (XML) 1.0 Second Edition*, W3C Recommendation, October. Disponível em: http://www.w3.org/TR/REC-xml

BRITTENHAM, P., CURBERA, F., EHNEBUSKE, D., *et al.*, 2001, "Understanding WSDL in a UDDI Registry", IBM White Paper. Disponível em: http://www-106.ibm.com/developerworks/library/ws-wsdl/

CABRERA, F., COPELAND, G., FREUND, T., 2002, "Web Services Coordination (WS-Coordination)". Disponível em: http://www-106.ibm.com/developerworks/library/ws-coor

CABRERA, F., COPELAND, G., COX, B., 2002, "Web Services Transaction (WS-Transaction)". Disponível em: http://www-106.ibm.com/developerworks/library/ws-transpec

CALADO, P., SILVA, A., VIEIRA, R., *et al.*, 2002, "Searching Web databases by structuring keyword based queries". In: *Proc. of the ACM CIKM Intl. Conf. on Information and Knowledge Management*, USA, pp. 26-33, November.

CAMPBELL, A., COULSON, G., HUTCHISON, D., 1994, "A Quality of Service Architecture". *ACM Computer Communications Review*, vol. 24, pp. 6-27, April.

CARDOSO, J., SHETH, A., 2002, *Semantic e-Workflow Composition*, Technical Report LSDIS Lab, University of Georgia, July.

CHANDRASEKARAN, S., SILVER, G., MILLER, J., *et al.*, 2002, "Web Service Technologies and their Synergy with Simulation". *Proc. of the Winter Simulation Conference*, San Diego, CA.

CHRISTENSEN, E., CURBERA, F., MEREDITH, G., *et al.*, 2001, *Web Services Description Language (WSDL) 1.1*, W3C Note, March. Disponível em: http://www.w3.org/TR/wsdl.html

CURBERA, F., MUKHI, N., WEEERAWARANA, S. , 2001, "On the Emergence of a Web Services Component Model". *Proc. of the 6th International Workshop on Component-Oriented Programming at ECOOP 2001*, Budapest, Hungary, June.

CURBERA, F., GOLAND, Y., KLEIN, J., *et al.* , 2002 , "Business Process Execution Language for Web Services Version 1.0". Disponível em http://www.ibm.com/developerworks/library/ws-bpel/

DIALANI, V., MILES, S.,MOREAU, L., *et al.*, 2002, "Transparent Fault Tolerance for Web Services Based Architectures". In: *Proceedings of the 8th International Europar Conference*, Paderborn, Germany, pp. 889-898, August.

DESHPANDE, A., HELLERSTEIN, J., 2002, "Decoupled Query Optimization for Federated Database Systems". In: *18th International Conference on Database Engineering*, pp. 716-728, San Jose, CA, USA, March.

DU, M., KRISHNAMURTHY, R., SHAN, M., 1992, "Query Optimization in a Heterogeneous DBMS". In: *Proceedings of the 18th VLDB Conference*, pp.297-29, Vancouver, Canada, 1992, August.

EIBACH, W., KUEBLER, D., 2001, "Metering and accounting for Web Services: A dynamic e-business solution", IBM White Paper. Disponível em: http://www-106.ibm.com/developerworks/webservices/library/ws-maws

ELMASRI, R., NAVATHE S., 1994, *Fundamentals of Database Systems*. Second Edition, California, Addison-Wesley.

FALLSIDE, D., 2001, *XML Schema Part 0: Primer*, W3C Recommendation, May. Disponível em : http://www.w3.org/TR/xmlschema-0/

FERGUSON, D. F., 2001, "Web Services Architecture: Direction and Position Paper". *Proc. of the W3C Workshop on Web Services*, San Jose, CA, April.

FLORESCU, D., GRÜNHAGEN, A., KOSSMANN, D., 2002, "XL: An XML Programming Language for Web Service Specification and Composition". In: *WWW2002, International World Wide Web Conference*, pp. 65-76, Honolulu, HI, USA, May.

FOSTER, I., KESSELMAN, C., 1999, *The Grid: Blueprint for a New Computing Infrastructure* , Morgan-Kaufman.

FOSTER, I., KESSELMAN, C., NICK, J., *et al.*, 2002, "The Physiology of the Grid: An Open Grid Service Architecture for Distributed Systems Integration".*Open Grid Service Infrastructure WG, Global Grid Forum*, June.

GRAY, J., 1978, "Notes on database Operation Systems". In: *Operation Systems: An Advanced Course*, v. 60, Lecture Notes in Computer Science, Springer-Verlag, New York.

HASS, L., KOSSMAN, D., WIMMERS, E., *et al.*, 1997, "Optimizing Queries across Diverse Data Sources". In: *Proc. of the 23rd VLDB Conference*, pp. 276-285, Atenas, Greece, August.

HENDLER, J., BERNERS-LEE, T., MILLER, E., 2002, "Integrating Applications on the Semantic Web". Journal of the Institute of Electrical Engineers of Japan, vol. 122(10), pp. 676-680, October.

HOSCHEK, W., 2002, "Web Service Discovery Processing Steps". *Proc. of the International WWW/Internet Conference*, Lisboa, Portugal, November.

HONDO, M., NAGARATNAM, N., NADALIN, A., 2002 , "Securing Web Services". *IBM Systems Journal* , vol. 41, n. 2.

IBM CORP., 2002, "IBM Web Services Toolkit – A showcase for emerging web services technologies". IBM White Paper. Disponível em: http://www-3.ibm.com/software/solutions/webservices/wstk-info.html

IBM CORP., 2000, "The IBM WebSphere software platform and patterns for e-business: invaluable tools for IT architects of the new economy". IBM White paper. Disponível em: http://www-3.ibm.com/software/info/websphere/docs/wswhitepaper.pdf

KAMATH, M., RAMAMRITHAM, K., 1998, "Failure Handling and Coordinated Execution of Concurrent Workflows". In: *Proc. of the 14th International Conference on Data Engineering (ICDE98)*, pp. 334-341, Orlando, Florida, February.

KEIDL, M., SELTZAM, S., STOCKER, K., *et al.*, 2002, "ServiceGlobe: Distributing E-Services Across the Internet". *Proc. of the 28th VLDB Conference*, Hong Kong, China, August.

KEIDL, M., SELTZSAM, S., KEMPER, A., 2002, "Flexible and Reliable Web Service Execution". *Proc. of the 1st Workshop on "Entwicklung von Anwendungen auf der Basis der XML Web-Service Technologie"*, Germany, July.

LAENDER, A., RIBEIRO-NETO, B., SILVA, A., *et al.*, 2002, "A Brief Survey of Web Data Extraction Tools", *SIGMOD Record* 31(2), pp. 84-93.

LAMPSON, B. W., STURGIS, H., 1976, *Crash Recovery in a Distributed Data Storage System*. Technical Report, Computer Sciences Laboratory, Xerox Palo Alto Research Center, Palo Alto, California.

LAU, T., 2001, "QoS for B2B Commerce in the New Web Services Economy". *ISEC Workshop on Quality of Services*, Hong Kong, China, April.

LEYMANN, F., 2001, "Web Services Flow Language (WSFL) 1.0", Maio. Disponível em : http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

MANI, A., NAGARAJAN, A., 2002, "Understanding Quality of Service for Web Services", IBM White Paper, Disponível em: http://www-106.ibm.com/developerworks/webservices/library/ws-quality.html?dwzone=webservices

MCILRAITH, S., SON, C., ZENG, H., 2001, "Semantic Web Services". In: *IEEE Intelligent Systems. Special Issue on the Semantic Web*, n. 16(2), pp. 46-53, March.

MECELLA, M., PERNICI, B., 2001 , "Designing wrapper components for e-services in integrating heterogeneous systems". *VLDB Journal* , n.10 , pp. 2-15 , Springer-Verlag.

MECELLA, M., SCANNAPIECO, M., VIRGILLITO, A., *et al.*, 2002, "Managing Data Quality in Cooperative Information Systems". *Proc. of the 10th Intl. Conf. on Cooperative Information Systems (CoopsIS)*, CA, USA.

MICROSOFT CORP., 2000, ".NET Framework – Home Page". Disponível em: http://msdn.microsoft.com/netframework/default.asp

MICROSOFT CORP., DIGITAL EQUIPMENT CORP., 1995, "The Component Object Model Specification", Outubro. Disponível em: http://www.opengroup.org/pubs/catalog/ax01.htm

MOREAU, L., 2002, "Agents for the Grid: A Comparison for Web Services (Part 1: The Transport Layer)". *IEEE International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May.

MOURA, A. M., CAMPOS, M. L., BARRETO, C., 1998, A Survey on Metadata for Describing and Retrieving Internet Resources, *World Wide Web 1(4)*, pp. 221-240.

NAJAFI, K., LEON-GARCIA, A., 2000, "A Novel Cost Model For Active Networks". *Proc. of the International Conference on Communication Technologies, World Computer Congress*, Beijing, China, August.

NICOLA, M., JARKE, M., 2000, "Performance Modeling of Distributed and Replicated Databases", *IEEE Transactions on Knowledge and Data Engineering*, v. 12, n. 4, pp. 645-672, July/August.

NOTTINGHAM, M. , 2001 , "Web Service Scalability and Performance with Optimising Intermediaries". *Proc. of the W3C Workshop on Web Services*, San Jose, CA, Apbril.

OMG (Object Management Group), 1995, "The Common Object Request Broker: Architecture and Specification", Julho.

ÖZSU, M., VALDURIEZ, P., 1999, *Principles of Distributed Database Systems*. Second Edition, New Jersey, Prentice-Hall.

PAOLUCCI, M., KAWAMURA, T., PAYNE, T.R., *et al.*, 2002, "Semantic Matching of Web Services Capabilities". *Proc. of the 1st International Semantic Web Conference (ISWC)*, Sardenha, Italia.

PAOLUCCI, M., KAWAMURA, T., PAYNE, T.R., *et al.*, 2002, "Importing the Semantic Web in UDDI". *Proc. of the CAISE Workshop on Web Services, e-Business and the Semantic Web (WES),* Toronto, Canada, May.

PAPAZOGLOU, M., AIELLO, M., PISTORE, M., *et al.*, 2002, *XSRL: An XML Web-Services Request Language*. Technical Report DIT-02-0079, University of Trento, Italy.

PIRES, P., 2002, *WebTransact: A Framework for Specifying and Coordinating Reliable Web Services Composition*. Tese de D. Sc., Relat. Tec. ES-578/02, COPPE/UFRJ, Brasil.

PIRES, P., MATTOSO, M., BENEVIDES, M., 2002, "Building Reliable Web Services Compositions" In: *Proc. of the NET.Object Days Conference (WS-RDS'02)*, Erfurt, Germany. Lecture Notes in Computer Science, v. 2593, Springer-Verlag.

PIRES, P., MATTOSO, M., BENEVIDES, M., 2003, "Mediating Heterogeneous Web Services". In: *Intl. Symposium on Applications and the Internet, SAINT-03*, IEEE Computer Society, Orlando, USA, January.

ROCCO, D., CRITCHLOW, T., 2002, *Discovery and Classification of BioInformatics Web Services*. Technical Report, Lawrence Livermore National Laboratory, September.

ROTH, M. T., ÖZCAN, F., HASS, L. M., 1999, "Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System". In: *Proc. of the 25th VLDB Conference*, pp 599-610, Edinburgh, Scotland, September.

RUBERG, G. , 2001 , *Um Modelo de Custo para o Processamento de Consultas em Bases de Objetos Distribuídos*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

RUBERG, N., RUBERG, G., MATTOSO, M., 2002, "DIG: Um Serviço de Custos e Estatísticas para o Processamento Distribuído de Consultas". In: *Anais do XVII Simpósio Brasileiro de Banco de Dados*, pp. 121-135, Gramado, Outubro.

SAHAI, A., OUYANG, J., MACHIRAJU, V., *et al.*, 2001, *Specifying and Guaranteeing Quality of Service for Web Services through Real Time Measurement and Adaptative Control*, HP Labs Technical Report HPL-2001-134, May.

SCANNAPIECO, M., MIRABELLA, V., MECELLA, M., *et al.*, 2002, "Data Quality in e-Business Applications". *Proc. of the CAISE Workshop on Web Services, e-Business and the Semantic Web (WES)*, Toronto, Canada, May.

SCHULDT, H., 2001, "Process Locking: A Protocol based on Ordered Shared Locks for the Execution of Transactional Processes". *Proc. of the Symposium on Principles of Database Systems (PODS)*, Santa Barbara, California, May.

SHAIKH, A., RANA, O.F., RASHID, A., *et al.*, 2003, "UDDIe: An Extended Registry for Web Services". *International Symposium on Applications and the Internet, SAINT03*, Florida, USA, January.

SHENG, Q., BENATALLAH, B., DUMAS, M., *et al.*, 2002, "SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment". *Proc.of the 28th VLDB Conference*, Hong Kong, China, August.

SHETH, A., CARDOSO, J., MILLER, J., *et al.*, 2002, "QoS for Service-oriented Middleware". In: *Proceedings of the Conf. on Systemics, Cybernetics and Informatics*, Orlando, Florida, July.

SUN MICROSYSTEMS, 2001, "Enterprise JavaBeans Specification 2.0", Agosto. Disponível em: http://java.sun.com/products/ejb/docs.html

SUN MICROSYSTEMS, 2002, "Developing Web Services with the Sun Open Net Environment". Sun White Paper. Disponível em: http://wwws.sun.com/software/sunone/wp-spine/spine.pdf

THATTE, S., 2001, "XLANG : Web Services for Business Process Design". Disponível em: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

TOMASIC, A., RASCHID, L., VALDURIEZ, P., 1998, "Scaling Access to Heterogeneous Data Sources with DISCO", *IEEE Transactions on Knowledge and Data Engineering*, 10(5), pp. 808-823.

TOSIC, V., PAGUREK, B., ESFANIARI, B., *et al.*, 2001 , "On the Management of Compositions of Web Services". Position Paper at the *Workshop on Object-Oriented Web Services* (at OOPSLA 2001), Tampa, USA, October.

TOSIC, V., PATEL, K., PAGUREK, B., 2002, "WSOL – Web Services Offerings Language". *Proc. of the CAISE Workshop on Web Services, e-Business and the Semantic Web (WES)*, Toronto, Canada, May.

TRASTOUR, D., BARTOLINI, C., GONZALEZ-CASTILLO, J., 2001 , "A Semantic Web Approach to Service Description for Matchmaking of Services". *Proc. of the International Semantic Web Working Symposium* , Stanford, CA, USA, August.

UDDI.org, 2000, "UDDI Technical White Paper", Setembro. Disponível em: http://www.uddi.org/pubs/lru_UDDI_Technical_White_Paper.PDF

WEBMETHODS, 2002 , "Requirements for Securing Enterprise Web Services". Disponível em: http://www.webmethods.com/PDF/Web_Services_Security_White_Paper.pdf

WIEDERHOLD, G., 1995, "Mediation in Information Systems", *ACM Computing Surveys*, v. 27, n. 2, pp. 265-267.

Apêndice 1 – XML Schema da linguagem WSQD

XML Schema representando os elementos da linguagem WSQD (Web Services Quality Definitions Language), que define os parâmetros de custo e critérios de qualidade aplicáveis a serviços Web e suas operações.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="qualitydefinitions">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="initTime">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:long">
                                <xs:attribute name="unit" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="availability">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:long">
                                <xs:attribute name="unit" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="reliability">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:long">
                                <xs:attribute name="unit" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="network">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="sendingTime">
                                <xs:complexType>
                                    <xs:simpleContent>
                                        <xs:extension base="xs:long">
                                            <xs:attribute name="unit" type="xs:string"/>
                                            <xs:attribute name="dataunit" type="xs:string"/>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="operation" maxOccurs="unbounded">
```

```
        <xs:complexType>
            <xs:sequence>
                <xs:element name="requestLength">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:long">
                                <xs:attribute name="unit" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="responseLength">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:long">
                                <xs:attribute name="unit" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="priority" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="execTime">
                                <xs:complexType>
                                    <xs:simpleContent>
                                        <xs:extension base="xs:long">
                                            <xs:attribute name="unit" type="xs:string"/>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="price">
                                <xs:complexType>
                                    <xs:simpleContent>
                                        <xs:extension base="xs:float">
                                            <xs:attribute name="unit" type="xs:string"/>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="value" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:QName" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="security" minOccurs="0">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="description"/>
                <xs:element name="authentication" minOccurs="0"/>
                <xs:element name="confidentiality" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:sequence>
<xs:attribute name="serviceName" type="xs:string" use="optional"/>
```

```
            <xs:attribute name="endpoint" type="xs:QName" use="optional"/>
            <xs:attribute name="id" type="xs:QName" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Apêndice 2 – XML Schema dos arquivos de log do WebTransact

XML Schema representando os elementos de um arquivo de log do WebTransact, contendo detalhes das execuções de um serviço específico.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="remoteservice">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="port" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="availability"/>
                            <xs:element name="reliability"/>
                            <xs:element name="operation" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="numberofexecutions"/>
                                        <xs:element name="sucessfulexecutions"/>
                                        <xs:element name="unavailabilityerrors"/>
                                        <xs:element name="unexpectederrors"/>
                                        <xs:element name="semanticerrors"/>
                                        <xs:element name="priority">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="avgexectime"/>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                    <xs:attribute name="name" type="xs:string"/>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

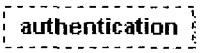
Apêndice 3 – Notação Gráfica para elementos de um XML Schema

Notação gráfica utilizada para representar elementos de um XML Schema (figuras 4.1 e 5.4)

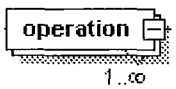
Tipos de Elementos :

Elemento simples obrigatório

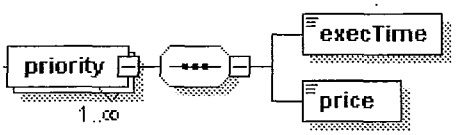
Elemento simples obrigatório, com conteúdo do tipo #PC-DATA(*Parsed Character Data*)



Elemento simples opcional



Elemento múltiplo obrigatório.
Detalhes: número mínimo de ocorrências = 1
número máximo de ocorrências = n



Especificação de um seqüência ordenada de subelementos
(os elementos "exectime" e "price" são sublementos de "priority", devendo aparecer nesta ordem)

Apêndice 4 – Listagem dos Arquivos para o Exemplo de Serviços de Validação de Cartão de Crédito

Definição do Mediador:

```
<wstl:mediatorService id="msCreditCard">
        <wstl:operation name="CheckCreditCardNumber">
                <wstl:inputMsg>
                        <wstl:param name="CCNumber" type="xsd:string"/>
                        <wstl:param name="Type" type="xsd:string"/>
                </wstl:inputMsg>
                <wstl:outputMsg>
                        <wstl:param name="validation" type="xsd:boolean"/>
                </wstl:outputMsg>
                <wstl:faultMsg errorCode="ERROR_101" description="Invalid card type."/>
                <wstl:contentDescription medParam="inputMsg/@Type">
                        <wstl:domain value="VISA"/>
                        <wstl:domain value="AMEX"/>
                        <wstl:domain value="MASTERCARD"/>
                </wstl:contentDescription>
                </wstl:operation>
</wstl:mediatorService>
```

Mediador msCreditCard

Definição dos Serviços Remotos Agregados pelo Mediador:

```
<wstl:remoteService id="rsMSugsCreditCard" medServ="tns:msCreditCard"
portType="rs:CCCheckerSoap">
        <wstl:operationMap name="CreditCardMap" ptOperation="rs:CheckCardNumber"
                        medOperation="tns:CheckCreditCardNumber">
                <wstl:inputMap>
                        <wstl:paramMap targetParam="cardNumber">
                                <wstl:sourceParam xpath="CCNumber"/>
                        </wstl:paramMap>
                        <wstl:paramMap targetParam="cardType">
                                <wstl:sourceParam xpath="Type"/>
                        </wstl:paramMap>
                </wstl:inputMap>
                <wstl:outputMap>
                        <wstl:paramMap targetParam="validation"
                                        mapFunction="ConvertCreditCardResults">
                                <wstl:sourceParam xpath="CheckCardNumberResponse"/>
                        </wstl:paramMap>
                </wstl:outputMap>
        </wstl:operationMap>
</wstl:remoteService>
```

Informações de Mapeamento para o Serviço Remoto rsMSugsCreditCard

```
<wstl:remoteService     id="rsRichCardValidator" medServ="tns:msCreditCard"
                        portType="rs:CreditCardValidatorSoap">
        <wstl:operationMap name="CreditCardMap" ptOperation="rs:ValidMod10"
                           medOperation="tns:CheckCreditCardNumber">
                <wstl:inputMap>
                        <wstl:paramMap targetParam="CardNumber">
                                <wstl:sourceParam xpath="CCNumber"/>
                        </wstl:paramMap>
                </wstl:inputMap>
        </wstl:operationMap>
</wstl:remoteService>
```

Informações de Mapeamento para o Serviço Remoto rsRichCardValidator

```
<wstl:remoteService     id="rsLocalCreditCard" medServ="tns:msCreditCard"
                        portType="rs:CreditCard">
        <wstl:operationMap name="CreditCardMap" ptOperation="rs:CheckCreditCardNumber"
                           medOperation="tns:CheckCreditCardNumber">
                <wstl:outputMap>
                        <wstl:paramMap targetParam="validation"
                                       mapFunction="ConvertCreditCardResults">
                                <wstl:sourceParam xpath="CheckCreditCardNumberReturn"/>
                        </wstl:paramMap>
                </wstl:outputMap>
                <wstl:contentDescription medParam="Type">
                        <wstl:domain value="VISA"/>
                        <wstl:domain value="MASTERCARD"/>
                </wstl:contentDescription>
        </wstl:operationMap>
</wstl:remoteService>
```

Informações de Mapeamento para o Serviço Remoto rsLocalCreditCard

```
<wstl:remoteService     id="rsCreditCardVerifier" medServ="tns:msCreditCard"
                        portType="rs:CreditCardVerifierSoap">
        <wstl:operationMap  name="CreditCardMap" ptOperation="rs:VerifyCreditCard"
                            medOperation="tns:CheckCreditCardNumber">
                <wstl:inputMap>
                        <wstl:paramMap targetParam="AccountNumber">
                                <wstl:sourceParam xpath="CCNumber"/>
                        </wstl:paramMap>
                        <wstl:paramMap targetParam="LicenseKey"
                                       mapFunction="GetLicenseKey">
                                <wstl:sourceParam xpath="Type"/>
                        </wstl:paramMap>
                </wstl:inputMap>
                <wstl:outputMap>
                        <wstl:paramMap targetParam="validation"
                                       mapFunction="ConvertCreditCardResults">
                                <wstl:sourceParam xpath="VerifyCreditCardResult"/>
                        </wstl:paramMap>
                </wstl:outputMap>
        </wstl:operationMap>
</wstl:remoteService>
```

Informações de Mapeamento para o Serviço Remoto rsCreditCardVerifier

Definição dos Critérios de Qualidade de cada serviço:

```
<qualitydefinitions id="rsCreditCardVerifier " endpoint=" CreditCardVerifierSoap">
        <initTime unit="ms">100</initTime>
        <availability>1.0</availability>
        <reliability>1.0</reliability>
        <network>
                <sendingTime unit="ms">0.001</sendingTime>
        </network>
        <operation name="VerifyCreditCard">
                <requestLength unit="bytes">500</requestLength>
                <responseLength unit="bytes">500</responseLength>
                <priority value="normal">
                        <execTime unit="ms">1200</execTime>
                        <price unit="USD">0.008</price>
                </priority>
        </operation>
        <security>
                <description>This service uses the HTTPS protocol</description>
                <authentication>HTTPS</authentication>
                <confidentiality>SSL</confidentiality>
        </security>
</qualitydefinitions>
```

Arquivo WSQD para o serviço rsCreditCardVerifier

```
<definitions>
        <qualitydefinitions id="rsLocalCreditCard " endpoint="CreditCard">
                <initTime unit="ms">100</initTime>
                <availability>1.0</availability>
                <reliability>1.0</reliability>
                <network>
                        <sendingTime unit="ms">0</sendingTime>
                </network>
                <operation name="CheckCreditCardNumber">
                        <requestLength unit="bytes">600</requestLength>
                        <responseLength unit="bytes">700</responseLength>
                        <priority value="normal">
                                <execTime unit="ms">3000</execTime>
                                <price unit="USD">0.001</price>
                        </priority>
                        <priority value="high">
                                <execTime unit="ms">1000</execTime>
                                <price unit="USD">0.03</price>
                        </priority>
                </operation>
        </qualitydefinitions>
</definitions>
```

Arquivo WSQD para o serviço rsLocalCreditCard

```
<qualitydefinitions id="rsMSugsCreditCard endpoint="CCCheckerSoap" >
        <initTime unit="ms">100</initTime>
        <availability>1.0</availability>
        <reliability>1.0</reliability>
        <network>
                <sendingTime unit="ms">0.001</sendingTime>
        </network>
        <operation name="CheckCardNumber">
```

```
            <requestLength unit="bytes">500</requestLength>
            <responseLength unit="bytes">500</responseLength>
            <priority value="normal">
                  <execTime unit="ms">1800</execTime>
                  <price unit="USD">0.005</price>
            </priority>
      </operation>
</qualitydefinitions>
```

Arquivo WSQD para o serviço rsMSugsCreditCard

```
<qualitydefinitions id="rsRichCardValidator" endpoint="CreditCardValidatorSoap" >
      <initTime unit="ms">100</initTime>
      <availability>1.0</availability>
      <reliability>1.0</reliability>
      <network>
            <sendingTime unit="ms">100</sendingTime>
      </network>
      <operation name="ValidMod10">
            <requestLength unit="bytes">400</requestLength>
            <responseLength unit="bytes">500</responseLength>
            <priority value="normal">
                  <execTime unit="ms">1400</execTime>
                  <price unit="USD">0.005</price>
            </priority>
      </operation>
      <security>
            <description>This service uses the HTTPS protocol</description>
            <authentication>HTTPS</authentication>
            <confidentiality>SSL</confidentiality>
      </security>
</qualitydefinitions>
```

Arquivo WSQD para o serviço rsMSugsCreditCard