Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Computação e Automação Industrial

Victor Alexandre Siqueira Marques de Souza

Uma Arquitetura Orientada a Serviços para Desenvolvimento, Gerenciamento e Instalação de Serviços de Rede

Campinas, SP
2006

Victor Alexandre Siqueira Marques de Souza

Uma Arquitetura Orientada a Serviços para Desenvolvimento, Gerenciamento e Instalação de Serviços de Rede

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Eleri Cardozo

Campinas, SP
2006

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Souza, Victor Alexandre Siqueira Marques de
So89a Uma arquitetura orientada a serviços para desenvolvimento, gerenciamento e instalação de serviços de rede / Victor Alexandre Siqueira Marques de Souza. – Campinas, SP: [s.n.], 2006.

Orientador: Eleri Cardozo.
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Serviços Web. 2. Redes de computação. 3. XML (Linguagem de marcação de documento). 4. Comunicações óticas. 5. Java (Linguagem de programação de computador). I. Cardozo, Eleri. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês:	A service oriented architecture for developing, managing and deploying network services.
Palavras-chave em Inglês:	Service Oriented Computing (SOC), Service Oriented Architecture (SOA), Web Services, Connection oriented networks, Orchestration, Choreography, Service composition, Optical networks.
Área de concentração:	Engenharia de Computação.
Titulação:	Mestre em Engenharia Elétrica.
Banca Examinadora:	Leonardo de Souza Mendes, Maurício Ferreira Magalhães e Rogério Drummond Burnier Pessoa de Mello Filho.
Data da defesa:	10/02/2006.

Victor Alexandre Siqueira Marques de Souza

Uma Arquitetura Orientada a Serviços para Desenvolvimento, Gerenciamento e Instalação de Serviços de Rede

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.
Aprovação em 10/02/2006.

Banca Examinadora:
Prof. Dr. Eleri Cardozo - UNICAMP
Prof. Dr. Leonardo de Souza Mendes - UNICAMP
Prof. Dr. Maurício Ferreira Magalhães - UNICAMP
Prof. Dr. Rogério D. B. P. de Mello Filho - UNICAMP

Campinas, SP
2006

Resumo

Novos serviços de rede devem ser desenvolvidos e gerenciados de acordo com os requisitos específicos dos clientes. Nesse contexto, provedores de serviço devem criar uma nova forma de projetar serviços de rede com tempo de desenvolvimento próximo a zero e alto nível de customização e evolução. A customização é necessária para criação de serviços que satisfaçam aos requisitos do cliente enquanto evolução é necessária para adaptar o serviço assim que esses requisitos se alterem. Além disso, clientes têm demandado a capacidade de gerenciar o serviço para manter o uso, configuração e evolução sob seu controle. Este trabalho apresenta uma abordagem baseada na arquitetura orientada a serviço para o desenvolvimento de serviços de rede capazes de cumprir os requisitos de rápida instalação, customização e gerenciamento por parte do cliente. Esta abordagem considera os serviços de rede como um conjunto de elementos interativos implementados como serviços *web*. A lógica do serviço é expressa através da orquestração de serviços *web*. Dois serviços para o gerenciamento de conexões em redes ópticas são apresentados como estudo de caso.

Palavras-chave: Computação Orientada a Serviço, Arquitetura Orientada a Serviço, Serviços *Web*, Orquestração, Coreografia, Redes Orientadas a Conexão.

Abstract

New generation network services must be developed and managed according to the customers' specific requirements. In this context, service providers must devise a way to design network services with near zero development time and high degrees of customization and evolution. Customization is necessary to fit the service according to the customers requirements, while evolution is necessary to adapt the service as soon as these requirements change. In addition, customers are demanding the ability to manage the service in order to keep the usage, configuration, and evolution under their control. This paper presents an approach based on Service Oriented Architecture (SOA) for developing network services able to fulfill the requirements of rapid deployment, customization, and customer-side manageability. The approach considers the network service as a set of interacting elements implemented as web services. The service logic is expressed in terms of web services orchestration. Two services for the management of connections in optical networks are presented as a case study.

Keywords: Service Oriented Computing, Service Oriented Architecture, Web Services, Orchestration, Choreography, Connection Oriented Networks.

Agradecimentos

Ao meu orientador, Prof. Dr. Eleri Cardozo, agradeço o auxílio e direcionamento.

À minha família, pelo apoio emocional, financeiro e espiritual.

Ao Prof. Dr. Maurício Ferreira Magalhães e ao Prof. Dr. Edmundo R. M. Madeira pelas sugestões ao trabalho.

Aos colegas de pós-graduação, pelas inúmeras colaborações. Em especial ao aluno Tomás A. C. Badan pela cessão do módulo MPLS.

À Ericsson Brasil, pelo suporte financeiro.

Dedico esse trabalho a Deus,
Fonte de todo conhecimento e sabedoria.

"Porque Deus dá a sabedoria;
Da sua boca procedem o conhecimento e o entendimento."
Pv. 2:6

Sumário

Lista de Figuras **xiii**

Lista de Tabelas **xv**

Glossário **xvii**

1 Introdução **1**
1.1 Motivações . . . 1
1.2 Contribuições . . . 2
1.3 Trabalhos Correlatos . . . 3
1.4 Organização do Texto . . . 4

2 Arquitetura Orientada a Serviço **5**
2.1 Visão Geral da Arquitetura Orientada a Serviços . . . 5
2.1.1 Características da Arquitetura Orientada a Serviços . . . 9
2.1.2 Tecnologias . . . 12
2.2 Composição de Serviços . . . 15
2.2.1 Orquestração . . . 15
2.2.2 Coreografia . . . 18
2.2.3 Diferenças Conceituais . . . 20
2.2.4 Comparativo entre Linguagens . . . 21
2.3 Business Process Execution Language . . . 22
2.3.1 Definições Iniciais . . . 22
2.3.2 Atividades Básicas . . . 23
2.3.3 Atividades Estruturadas . . . 24
2.3.4 Peculiaridades da Linguagem . . . 25
2.3.5 Plataformas Disponíveis . . . 27

3 A Arquitetura SOANet **31**
3.1 Visão Geral . . . 31
3.2 Composição Fim-a-Fim . . . 33
3.3 Arquitetura do Cliente . . . 34
3.4 Implementação dos Serviços . . . 35
3.5 Instalação dos Serviços . . . 36

3.6 Análise Comparativa 36
3.7 Considerações Finais 37

4 Estudo de Caso: Redes Ópticas **39**
4.1 Projeto 39
4.1.1 Especificação de Requisitos 40
4.1.2 Diagramas de Classes 42
4.1.3 Diagramas de Seqüência 49
4.2 Implementação 53
4.2.1 Serviço de Conexão Cruzada 57
4.2.2 Serviço de Gerenciamento de Recursos 58
4.2.3 Serviço de Roteamento 58
4.2.4 Serviço de *Logging* 60
4.2.5 Serviço de Autenticação 60
4.2.6 Serviço de Tarifação 60
4.3 Testes 62
4.4 Considerações Finais 64

5 Conclusões **65**
5.1 Trabalhos Futuros 65
5.2 Considerações Finais 66

Referências bibliográficas **68**

A Descrição das Interfaces **75**

Lista de Figuras

2.1 Visão Simplificada da Arquitetura Orientada a Serviços. 6
2.2 Mecanismo de Registro e Descoberta da Arquitetura Orientada a Serviço. 7
2.3 Arquitetura Orientada a Serviços Estendida. 8
2.4 Orquestração de Serviços. 16
2.5 Mecanismo Básico de Coreografia. 19
2.6 Interação entre os Mecanismos de Orquestração e Coreografia. 20
2.7 Comparativo de Linguagens de Composição. 21
2.8 Tecnologias para Composição de Serviços BPEL. 23
2.9 Interface de Administração do Servidor BPWS4J. 28
2.10 Interface de Desenvolvimento BPEL (*JDeveloper*). 29
2.11 Interface de Gerenciamento do Servidor *ActiveBPEL*. 29
2.12 Visualização do Estado de um Processo (*ActiveBPEL*). 30

3.1 Arquitetura Servidor. 32
3.2 Composição de Serviços Fim-a-Fim. 33
3.3 Interação com Provedores de Serviço ou de Rede. 35

4.1 Diagrama de Classes do Serviço de Gerenciamento de Conexões (SGC). 43
4.2 Diagrama de Classes do Serviço de Gerenciamento de Falhas (SGF). 44
4.3 Diagrama de Classes do Serviço de Autenticação (SA). 45
4.4 Diagrama de Classes do Serviço de Tarifação (ST). 45
4.5 Diagrama de Classes do Serviço de *Logging* (SL). 46
4.6 Diagrama de Classes do Serviço de Conexão Cruzada (SCC). 47
4.7 Diagrama de Classes do Serviço de Gerenciamento de Recursos (SGR). 48
4.8 Diagrama de Classes do Serviço de Roteamento (SR). 49
4.9 Diagrama de Seqüência de Criação de um Caminho de Luz. 51
4.10 Diagrama de Seqüência de Eliminação de um Caminho de Luz. 52
4.11 Diagrama de Seqüência de Restauração de Falha. 54
4.12 Diagrama de Seqüência de Eliminação de Falha. 55
4.13 Interface *Web* para Instalação da Rede. 56
4.14 Comutador Óptico Emulado. 57

Lista de Tabelas

2.1 Comparativo entre Linguagens de Composição de Serviços *Web*. 22

4.1 Tempos de Resposta. 63

Glossário

ACID	*Atomocity, Consistency, Isolation and Durability*
API	*Application Programming Interface*
ASP	*Active Server Pages*
ATM	*Asynchronous Transfer Mode*
BPEL4WS	*Business Process Execution Language for Web Services*
BPMI	*Business Process Management Initiative*
BPML	*Business Process Modeling Language*
BPSS	*Business Process Specification Schema*
BPWS4J	*Business Process Execution Language for Web Services Java Run Time*
COM+	*Component Object Model*
DWDM	*Dense Wavelength Division Multiplexing*
ebXML	*Electronic Business using Extensible Markup Language*
EJB	*Enterprise Java Beans*
ESOA	*Extended Service Oriented Architecture*
HTTP	*HiperText Transfer Protocol*
JSP	*Java Server Pages*
MPLS	*Multiprotocol Label Switching*
MPLS	*Multiprotocol Label Switching*
OADM	*Optical Add-Drop Multiplexer*
OC4J	*Oracle Application Server Containers for J2EE*
OIF	*Optical Internetworking Forum*

OXC	*Optical Cross-Connects*
PXC	*Photonic Cross-Connects*
RPC	*Remote Procedure Call*
RWA	*Routing and Wavelength Assignment*
SGC	Serviço de Gerenciamento de Conexões
SGF	Serviço de Gerenciamento de Falhas
SLA	*Service Level Agreement*
SNMP	*Simple Network Management Protocol*
SOA	*Service Oriented Architecture*
SOANet	*Service Oriented Architecture for Network Services*
SOAP	*Simple Object Access Protocol*
SOC	*Service Oriented Computing*
SONET	*Synchronous Optical Networks*
SPF	*Shortest Path First*
SSL	*Secure Socket Layer*
UCLP	*User Controlled LightPath*
UDDI	*Universal Description, Discovery and Integration*
UM	Unidade Monetária
WSBPEL	*Web Services Business Process Execution Language*
WSCDL	*Web Services Choreography Description Language*
WSCI	*Web Services Choreography Interface*
WSCL	*Web Services Conversation Language*
WSDL	*Web Services Description Language*
WSFL	*Web Services Flow Language*
XML	*Extensible Markup Language*
XSD	*XML Schema Definition*

Capítulo 1

Introdução

Este capítulo tem por objetivo apresentar as motivações do trabalho realizado, uma breve descrição de suas contribuições, alguns trabalhos correlatos e a organização do texto.

1.1 Motivações

A alta competividade, associada à necessidade de fidelização de clientes e à oferta de serviços diferenciados tem pressionado provedores de rede a considerarem aspectos antes ignorados no processo de desenvolvimento de novos serviços de rede. Primeiramente, o tempo de desenvolvimento de novos serviços de rede deve ser minimizado. Em outras palavras, o tempo entre a especificação do projeto e o uso efetivo do serviço deve ser muito pequeno (idealmente zero, não mais do que algumas horas em certos casos). Além disso, os serviços de rede devem levar em conta requisitos e expectativas específicos do cliente, por exemplo, aspectos relacionados à configuração, tarifação e qualidade de serviço. Clientes têm exigido a capacidade de gerenciar certas características do serviço para tirar proveito de peculiaridades de seus negócios, como padrões de tráfego, segurança dos dados, entre outros. Obviamente, preço é sempre uma importante variável que pode ser reduzida com a diminuição da complexidade de desenvolvimento, instalação e gerenciamento dos serviços.

Ao considerar as tecnologias empregadas, nota-se que os serviços de rede existentes apresentam uma grande separação entre entidades de software de negócio e de rede. Entidades no nível de rede são baseadas em protocolos de sinalização de baixo nível como RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*) e UNI (*User-to-Network Interface*) do *Optical Internetworking Forum* [1]. O acesso a esses protocolos é realizado via interface de operação, protocolos de gerenciamento de rede como SNMP (*Simple Network Management Protocol*) ou APIs (*Application Programming Interfaces*) proprietárias. Já as entidades no nível de negócios estão baseadas em artefatos de software de alto nível, como componentes *Enterprise Java Beans* (EJB), COM+ (*Component Object Model*) e

componentes *web*, como JSP (*Java Server Pages*) e ASP (*Active Server Pages*). Esta grande separação entre entidades de software no nível de rede e no nível de negócios é um complicante quando um serviço de rede com alto nível de automação e integração deve ser projetado, desenvolvido e instalado em um curto período de tempo.

Neste contexto, provedores de rede precisam desenvolver uma nova forma de projetar, instalar e gerenciar serviços de rede. Acredita-se que a composição de serviços seja uma solução para este problema [2]. Um novo serviço de rede pode ser criado a partir da composição de um grupo primitivo de serviços. Esta definição recorrente é importante pelo fato de possibilitar que um serviço mais complexo seja construído sobre um conjunto de serviços já existentes. Por exemplo, um serviço de rede privada virtual pode ser construído a partir da composição de um serviço de conexão, um serviço de autenticação, um serviço de gerenciamento de falhas e um serviço de gerenciamento de recursos. Estes serviços estão distribuídos pela rede do provedor, alguns executando no escritório central e outros próximos à rede de transporte.

Além disso, a computação orientada a serviço é uma solução atrativa para solucionar essa separação na medida que oferece um bom nível de automação, integração, customização e flexibilidade na criação, instalação e gerenciamento de novos serviços de rede. Este trabalho propõe uma arquitetura orientada a serviço para desenvolvimento, instalação e gerenciamento de serviços para redes orientadas a conexão. A arquitetura pressupõe que todos os serviços constituintes são serviços *web* e que a composição destes é governada pela orquestração e coreografia de serviços *web*. A criação de um serviço consiste na edição de um *script* de orquestração enquanto a instalação é a cópia deste *script* em uma máquina de software apropriada à sua execução. O gerenciamento do serviço é descrito via coreografia. A edição dos *scripts* de orquestração pode ser auxiliada por ferramentas especializadas, editores de texto de propósito geral ou mesmo por ferramentas de modelagem de software.

1.2 Contribuições

Como contribuições do trabalho proposto, destacam-se os seguintes pontos:

- Especificação de uma arquitetura orientada a serviços para desenvolvimento, instalação e gerenciamento de serviços em uma rede orientada a conexão;
- Análise e projeto de dois sistemas baseados nessa arquitetura, um para gerenciamento de conexões ópticas e outro para gerenciamento de falhas em redes ópticas;
- Implementação dos dois serviços de rede especificados e dos serviços que compõem os mesmos.

1.3 Trabalhos Correlatos

Um importante trabalho correlato está sendo desenvolvido no contexto do projeto de pesquisa *User Controlled LightPath* (UCLP) [3] da organização sem fins lucrativos *Canarie* [4]. Este projeto foi implementado e testado na rede Canadense de pesquisa CA*Net4. O sistema desenvolvido permite que clientes finais, tanto pessoas como aplicações de software, tratem os recursos de rede como objetos de software, gerenciando e reconfigurando-os dentro de um único domínio ou através de múltiplos domínios administrados independentemente. Tal projeto tem explorado novas funcionalidades e melhorias à implementação atual do sistema por intermédio do uso de mecanismos de orquestração para criação de redes privadas articuladas. As principais características dessa arquitetura são [5]:

- Todo software da rede, hardware, caminhos de luz e comutadores ópticos são expostos como serviços *web*;
- Composição de serviços *web* é empregada para construir um plano de controle universal ligando instrumentos, caminhos de luz, comutadores ópticos, redes e sistemas de software.

Diferentemente do modelo proposto por UCLP, nós não permitimos que o cliente interaja diretamente com elementos de rede dentro de um domínio por questões de segurança. Na arquitetura aqui proposta, uma interface contratual para cada domínio administrativo é exposta para prover serviços aos clientes, criando um encapsulamento dos serviços oferecidos pelo domínio. Dado que provedores de rede impõem sérias restrições quanto a abertura de suas redes para sinalização externa ou gerenciamento de elementos de rede esta arquitetura se torna mais realista. Além disso, usando o conceito de composição de serviços recursiva da arquitetura orientada a serviços pode-se prover serviços fim-a-fim de forma muito estruturada, como discutido na Seção 3.2.

Em [6] encontramos uma arquitetura para criação de serviços de rede baseada em técnicas de composição de serviços *web*. Neste trabalho foram definidos dois tipos de serviços de rede, serviços atômicos e compostos. O primeiro expõe as funcionalidades de equipamentos de rede (*e.g.*, roteador) como serviços *web*. O segundo possui operações que são expostas ao cliente final da rede, sendo implementado a partir da composição de serviços atômicos. A principal diferença entre a arquitetura proposta neste trabalho e naquele é a utilização de *wrappers* para interação com os elementos de rede. Estes *wrappers* traduzem as chamadas de serviço *web* em comandos proprietários (como SNMP ou Telnet). No trabalho corrente é proposta a adição de serviços *web* dentro do dispositivo de rede, sem a utilização de *wrappers* ou *gateways*. Dessa forma obtemos uma arquitetura que permite a composição de serviços de rede de forma rápida e flexível.

1.4 Organização do Texto

O Capítulo 2 tem por objetivo apresentar uma visão geral da arquitetura orientada a serviços. São discutidos os mecanismos de orquestração e coreografia de serviços, bem como linguagens e plataformas para desenvolvimento de software baseados nesta arquitetura.

O Capítulo 3 aborda a arquitetura de desenvolvimento, instalação e gerência de serviços de rede proposta neste trabalho. Esta arquitetura é independente de tecnologia, plataforma e linguagem utilizadas.

O Capítulo 4 descreve dois sistemas para gerência de redes ópticas desenvolvidos baseados na arquitetura proposta. São descritos também os testes realizados para avaliação dos sistemas e os resultados obtidos.

Finalmente, o Capítulo 5 apresenta as conclusões do trabalho realizado, uma análise dos resultados alcançados e possíveis trabalhos futuros.

Capítulo 2

Arquitetura Orientada a Serviço

Este capítulo descreve o paradigma de computação orientada a serviços e a arquitetura orientada a serviços. São apresentadas as tecnologias atuais que podem ser consideradas instâncias da arquitetura orientada a serviços. Ferramentas e plataformas para desenvolvimento de software orientado a serviços são brevemente descritas.

2.1 Visão Geral da Arquitetura Orientada a Serviços

O paradigma de computação orientada a serviço, *Service Oriented Computing* (SOC), e sua respectiva arquitetura de suporte, *Service Oriented Architecture* (SOA), são considerados atualmente os mais promissores na área de computação distribuída. Esse paradigma prega a interoperabilidade e o fraco acoplamento entre elementos básicos de software - chamados *serviços* - através da definição de interfaces neutras e da utilização de protocolos de transporte padronizados. Serviços são unidades lógicas de software, podendo encapsular um simples método ou um grande processo envolvendo múltiplos colaboradores. Em outras palavras,

> " Computação Orientada a Serviço é o paradigma de computação que utiliza serviços como elementos fundamentais para o desenvolvimento de aplicações." [7]

Serviços são componentes abertos, auto-descritivos que suportam a composição de aplicações distribuídas de forma rápida e com baixo custo [8]. Serviços devem utilizar um protocolo padrão e aberto para comunicação (*e.g.*, *HiperText Transfer Protocol*), provendo assim uma infra-estrutura de computação distribuída tanto dentro como entre empresas (*intra-enterprise* e *cross-enterprise*, respectivamente). Um serviço deve conter uma interface de comunicação que fornece a assinatura do serviço (parâmetros de entrada, saída, erro e tipos de mensagens). O comportamento esperado de um

serviço é descrito por fluxogramas. Resumidamente, um serviço é um componente de software auto contido que expõe funcionalidades para clientes através de interfaces (contratos) bem definidas.

> "Um serviço é um recurso abstrato que possui a capacidade de realizar tarefas que representam uma funcionalidade do ponto de vista de entidades provedoras e entidades requisitoras." [9]

A primeira menção à arquitetura SOA foi realizada em um artigo publicado pela equipe de desenvolvimento de serviços *web* da IBM no *site* developerWorks (*http://www.ibm.com/developerWorks*) [10]. Desde então ela tem sido de grande interesse entre pesquisadores no meio científico e tem recebido uma forte aceitação no mercado de desenvolvimento de software.

Apesar da arquitetura ter sido originada em uma equipe de desenvolvimento *web* ela não pressupõe o uso dessa tecnologia, sendo bem mais abrangente que esta. Grande confusão existe devido ao fato dos serviços *web* serem atualmente a tecnologia de maior destaque na implementação da arquitetura, entretanto, a utilização desta tecnologia não é obrigatória.

Em muitos aspectos a arquitetura SOA pode ser vista como uma aplicação da arquitetura cliente servidor em um novo contexto [11]. Simplificadamente, um requerente de serviço (cliente) envia uma mensagem a um provedor de serviço (servidor) e o provedor envia uma resposta, podendo ser uma informação requisitada ou a confirmação de que alguma ação foi tomada, conforme ilustra a Figura 2.1. A transação pode ser síncrona ou assíncrona (através de interfaces de *call back*) e é independente de protocolo de transporte utilizado.

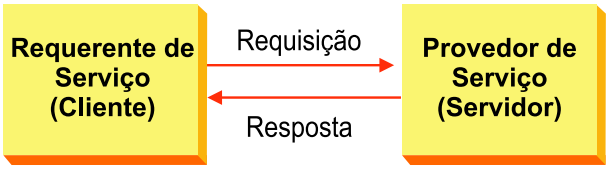

Fig. 2.1: Visão Simplificada da Arquitetura Orientada a Serviços.

Para possibilitar dinamismo na localização dos serviços, SOA prevê a utilização de um mecanismo de registro (*registry*) e descoberta de serviços (*discovery*) que resulta em três papéis bem definidos na arquitetura:

- Provedor de serviços: Entidade que provê serviços acessíveis pela rede. Responsável pelo gerenciamento, manutenção, publicação do serviço, entre outros.
- Cliente de serviços: Entidade que usa um serviço fornecido por um provedor de serviços. Para conhecimento das funcionalidades, localização e requisitos específicos do serviço (*e.g.*, protocolos) pode realizar consultas a uma agência de registro de serviços.

- Agência de registro de serviços: Entidade que mantém registro dos serviços disponíveis na rede, bem como funcionalidades, localização e protocolos utilizados pelo serviço, entre outras informações.

A interação entre estas três entidades se dá em quatro passos, segundo a Figura 2.2. Inicialmente o provedor publica o serviço na agência de registro, fornecendo informações relevantes quanto a utilização do mesmo, como localização do serviço, interfaces de interação, protocolos de transporte e aplicação utilizados, formato dos dados requeridos e parâmetros de qualidade do serviço. Em seguida o cliente do serviço solicita à agência de registro a busca por um serviço que satisfaça certos parâmetros. Esta, responde ao cliente uma lista (possivelmente nula) com serviços que satisfazem os parâmetros recebidos. Baseado em algum critério de avaliação, o cliente escolhe o serviço que mais lhe convém e envia uma requisição ao respectivo provedor do serviço, que por sua vez envia o resultado de tal requisição.

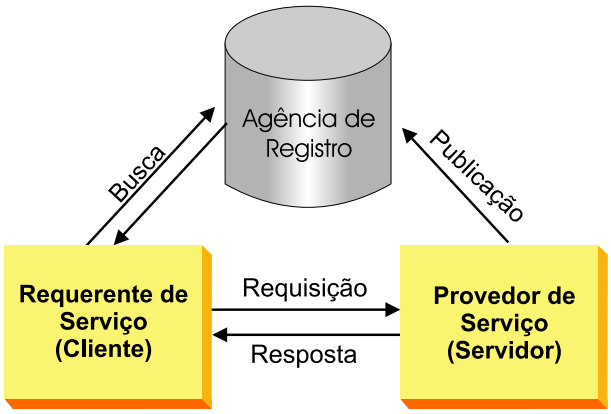

Fig. 2.2: Mecanismo de Registro e Descoberta da Arquitetura Orientada a Serviço.

Os elementos da arquitetura orientada a serviço mencionados compõem a base da arquitetura. Entretanto várias funcionalidades não são contempladas na base da arquitetura levando à necessidade de desenvolvimento de outras camadas que se beneficiam da infra-estrutura criada. Entre as funcionalidades não cobertas temos gerenciamento de transações, coordenação e segurança. Duas camadas adicionais foram definidas com o amadurecimento da arquitetura, resultando na arquitetura orientada a serviços estendida - *Extended Service Oriented Architecture* (ESOA) [8]. As funcionalidades de cada uma dessas camadas podem ser vistas na Figura 2.3 e serão descritas a seguir.

Na Figura 2.3 são visualizadas três camadas. A camada na base da pirâmide representa as funcionalidades básicas de provisão, publicação e descoberta de serviços previamente descritas. Sobre esta encontramos a camada de composição de serviços, responsável pela consolidação de múltiplos servi-

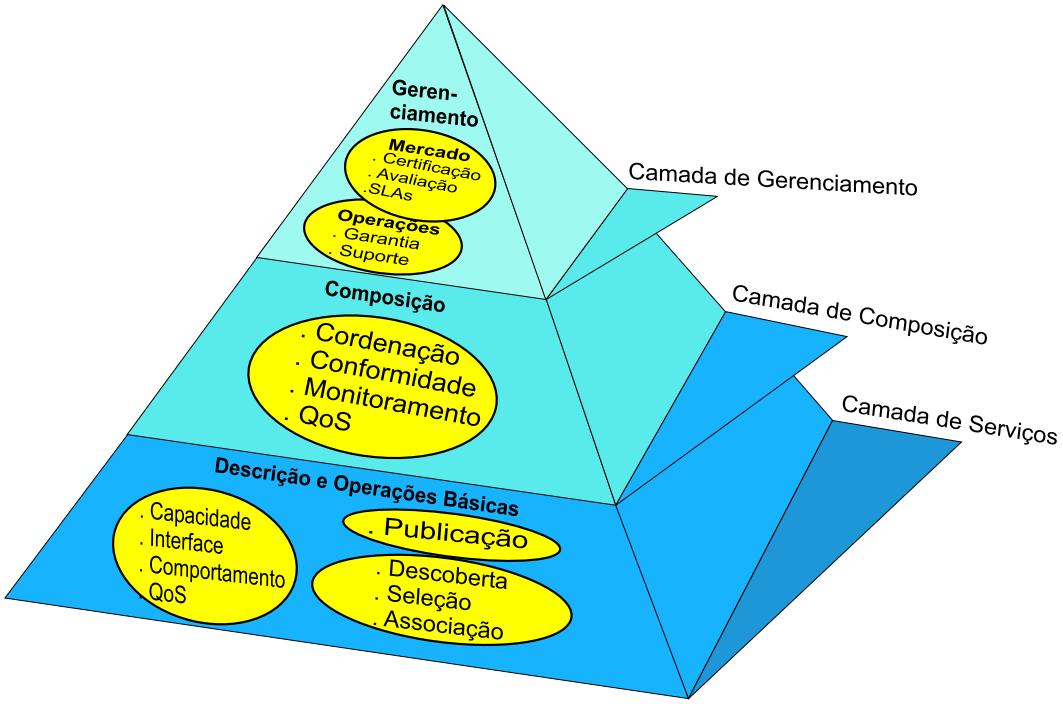

Fig. 2.3: Arquitetura Orientada a Serviços Estendida.

ços em um único serviço composto[1]. O *agregador de serviços* é a entidade responsável por compor os novos serviços. Os serviços compostos resultantes podem ser utilizados por outros agregadores de serviços como componentes (*i.e.*, serviços básicos) em futuras composições de serviços ou podem ser utilizados por clientes finais como soluções ou aplicações. Os agregadores de serviço se tornam provedores de serviço ao publicar os serviços compostos criados, possivelmente agregando valor ao compor o novo serviço. Agregadores de serviços devem prover as seguintes funcionalidades:

- Coordenação: controlar a execução e gerenciar o fluxo de dados entre os serviços componentes.
- Conformidade: garantir a integridade do serviço composto verificando compatibilidade entre tipos de dados, impondo restrições aos serviços componentes para garantir regras de negócio e realizando atividades de tratamento de dados.
- Monitoração: permitir mecanismos de subscrição a eventos ou informações produzidas pelos serviços componentes e publicar eventos compostos de mais alto nível.
- Qualidade de Serviço: computar a qualidade de serviço derivada da agregação da qualidade dos serviços componentes.

[1] Serviço composto é aquele implementado pela composição de outros serviços, chamados ao longo desse documento de serviços constituintes, serviços básicos, serviços agregrados ou serviços componentes.

Note que ao possibilitar que um agregador seja visto como um provedor temos uma definição recorrente quanto à criação de novos serviços. A recorrência permite que serviços mais abrangentes e significativos possam ser desenvolvidos sobre uma gama já existente de serviços disponíveis na rede. O nível de abstração e a funcionalidade do software cresce na medida em que a recorrência se propaga.

Sobre a camada de composição temos a camada de gerenciamento, no topo da pirâmide. O objetivo dessa camada é prover suporte a aplicações críticas que requerem que empresas gerenciem a plataforma de serviço, a instalação de serviços e aplicações. O gerenciamento de operações pode prover estatísticas de performance detalhadas que possibilitam a avaliação da efetividade da aplicação, permitem visibilidade completa das transações de negócio individuais e fornecem notificações de estado quando uma particular atividade é completada ou uma condição de decisão é alcançada. Além disso, o gerenciamento de operações pode ser utilizado para monitorar a corretude e funcionamento global do serviço composto evitando o risco de erros nos serviços. Isto pode ser feito, por exemplo, verificando compatibilidades dos contratos de serviço (*Service Level Agreements* - SLAs) dos serviços componentes com o contrato de serviço do serviço composto. O elemento da arquitetura responsável por implementar tais funções de gerenciamento é o *operador do serviço*. Dependendo dos requisitos da aplicação um operador de serviço pode ser um cliente ou um agregador de serviço.

Outra funcionalidade da camada de gerenciamento de serviços é prover um mercado com suporte à negociação aberta de serviços, criando oportunidades para vendedores e compradores conduzirem negócios eletronicamente. Um comprador pode compor serviços de procura/demanda oferecendo um serviço com valor agregado. O mercado de serviços deve oferecer certificação de serviços, garantia de qualidade, serviços de avaliação (*rating*), métricas dos serviços (*e.g.*, número corrente de clientes, tempo médio de resposta) e gerenciar a negociação e cumprimento dos contratos de serviço. Tais funcionalidades são providas pelo *operador de mercado*.

2.1.1 Características da Arquitetura Orientada a Serviços

O uso da arquitetura SOA traz várias vantagens no processo de desenvolvimento de software. Entre elas destacam-se:

Fraco acoplamento

Aplicações desenvolvidas com base na arquitetura SOA são inerentemente fracamente acopladas. Isso ocorre porque cada entidade responsável pelo desenvolvimento de uma unidade lógica de software especifica sua própria interface de acesso. A implementação é isolada da interface, *i.e.*, existe o encapsulamento da lógica de implementação do serviço que fica transparente ao cliente.

Através da utilização do mecanismo de descoberta evita-se acoplamento quanto a localização do serviço com o qual deseja-se interagir, ou seja, existe transparência de localização do serviço. Essa é uma das grandes vantagens do desenvolvimento baseado em SOA. O serviço com o qual deseja-se interagir pode ser descoberto em tempo de execução levando em conta parâmetros *a priori* desconhecidos.

Não existem restrições quanto a linguagem de programação e a plataforma de software a serem utilizados, uma vez que um protocolo de aplicação independente de plataforma é utilizado. Isso resulta em flexibilidade no desenvolvimento do serviço, podendo a escolha da plataforma e linguagem de programação ser baseada apenas em restrições do próprio serviço, não do sistema como um todo. Por exemplo, um serviço com restrições de desempenho pode ser implementado em uma linguagem de programação não-interpretada. Além disso, softwares legados podem ser reaproveitados através da exposição de suas funcionalidades em interfaces de serviço.

A necessidade de uma interação síncrona pode ser atenuada com o uso de interações assíncronas providas pela arquitetura. Dessa forma não existe acoplamento quanto ao tipo de interações entre o cliente e o provedor de serviços. Adicionalmente, transações de longa duração podem ser realizadas sem manutenção de uma conexão de transporte durante a execução do serviço, liberando o servidor para o recebimento de pedidos de conexões provenientes de outros clientes.

Interoperabilidade

Serviços idealmente devem ser implementados com uso de tecnologias padronizadas e disponíveis a um grande número de plataformas de software. Isto implica que os mecanismos de invocação (protocolos, descrições de interfaces, mecanismos de descoberta) devem ser neutros e ser compatíveis com padrões largamente aceitos.

O uso de um protocolo de aplicação independente de plataforma resulta em interoperabilidade entre serviços. Protocolos proprietários e fracamente padronizados apresentam problemas de interoperabilidade entre softwares desenvolvidos por diferentes fabricantes. Por vezes são bloqueados por *firewalls* e impossibilitados de serem aplicados em um ambiente multi-organizacional.

Composição

Serviços podem ser compostos para formar novos serviços com um nível maior de abstração e provendo funcionalidades agregadas. A flexibilidade na composição de novos serviços a partir de serviços já disponíveis na rede é grande atrativo da arquitetura. Através de plataformas de composição de serviços um projetista pode facilmente compor novos serviços.

Existem várias possibilidades quanto à entidade que implementará um dado serviço (ou conjunto de serviços) constituinte(s). São elas:

- Projeto e implementação locais: uma vez especificado, o projeto de suas interfaces ou conjunto de interfaces e implementação é realizado localmente.

- Compra, aluguel ou pagamento por serviços: certos serviços podem ser adquiridos de um provedor de serviços ao invés de implementados internamente. Esse tipo de transação é diferente do padrão conhecido de compra de soluções de software na medida em que o pagamento é realizado pela utilização e não pela posse do programa. A tarifação pode se dar por uso, por subscrição ao serviço ou por políticas específicas (*e.g.*, horário de utilização).

- Delegação do projeto e implementação: uma vez especificado, o projeto de suas interfaces ou conjunto de interfaces e implementação é realizado externamente, por exemplo, por uma empresa de desenvolvimento de software.

- Utilização de software legado: acesso a bancos de dados ou softwares legados podem ser realizados através de *wrappers* ou adaptadores. *Wrappers* utilizam código legado encapsulando suas funcionalidades em interfaces de serviço. Adaptadores fornecem novas lógicas de negócio e regras que complementam funcionalidades do software legado.

Reusabilidade

Por serem módulos independentes de código e fracamente acoplados à aplicação os serviços podem ser facilmente reutilizados, resultando em um alto nível de produtividade no desenvolvimento de software. O reuso de código gera redução no tempo de desenvolvimento de aplicações, bem como redução de custos.

Alta granularidade

O encapsulamento de funcionalidades no nível de serviço evoca um alto grau de granularidade nos componentes básicos da arquitetura. Um objeto individual apresenta operações muito finas para prover funcionalidades significativas no nível corporativo. Para o desenvolvimento de aplicações complexas e extensas a alta granularidade traz vantagens na medida em que detalhes de implementação são deixados à equipe de desenvolvimento responsável por aquele serviço.

Ubiqüidade

Os serviços devem ser acessíveis a partir de qualquer lugar e a qualquer momento facilitando a composição de serviços entre empresas.

2.1.2 Tecnologias

A tecnologia atual que mais conformidade possui com a arquitetura e com o paradigma de computação orientada a serviços é notadamente a de serviços *web*. A difusão de tal tecnologia alavancou a arquitetura orientada a serviços no mercado de desenvolvimento de software. As vantagens de utilização de SOA em um ambiente que possui altas restrições quanto ao tempo de desenvolvimento e reusabilidade de código são inúmeras.

É preciso inicialmente definir serviços *web*. Intuitivamente pode-se inferir que são serviços acessíveis via *web*, utilizando protocolos padrões da Internet. Baseado em tal premissa poderia-se erroneamente classificar como serviços *web* uma infinidade de tecnologias difundidas recentemente na rede, como *scripts* CGI, JSP e ASP. Entretanto, segundo [9]:

> "Um serviço *web* é um sistema de software projetado para suportar interações máquina-máquina interoperáveis sobre uma rede. O serviço *web* possui uma interface descrita em um formato passível de processamento pela máquina, especificamente *Web Service Description Language* (WSDL). Outros sistemas interagem com o serviço *web* da maneira definida na sua interface usando mensagens SOAP (*Simple Object Access Protocol*), tipicamente transportadas usando HTTP (*HiperText Transfer Protocol*) com serialização XML (*Extensible Markup Language*) e em conjunto com outros padrões relacionados à *web*."

Além disso, segundo [10], um serviço *web* deve possuir fraco acoplamento e alta granularidade. Estas características, combinadas com mecanismos de descoberta e composição presentes caracterizam a compatibilidade da tecnologia de serviços *web* com a arquitetura orientada a serviços.

A seguir são destacadas as tecnologias envolvidas na provisão da arquitetura SOA no contexto dos serviços *web*.

Extensible Markup Language (XML)

XML é uma linguagem extensível que utiliza marcações e atributos para representação de dados [12]. XML não impõe restrições quanto ao conjunto possível de marcações e atributos a serem utilizados em um dado documento. Conseqüentemente, é muito mais rica e descritiva que outras linguagens de marcação, que definem um conjunto limitado de elementos constituintes da linguagem. Dado que a interação entre cliente e serviço se dá de modo conversacional, a linguagem XML é utilizada na representação de mensagens de requisição e resposta de serviços *web*.

A característica mais destacável da linguagem XML é que as mensagens têm codificação em texto puro, ou seja, não há necessidade de protocolos proprietários para codificação ou decodificação

de mensagens e dados. Além disso, podem ser lidas por um ser humano, facilitando o processo de depuração de software. Em contrapartida, o uso de texto plano pode representar um grande impacto no consumo de banda necessário à transmissão da mensagem e trazer problemas de segurança, uma vez que o dado está completamente aberto. Para solucionar o primeiro problema pode-se utilizar compressão HTTP (usualmente compressão *gzip*). Já o segundo pode ser solucionado com o uso de criptografia de dados, criando um canal seguro entre cliente e provedor de serviço.

A linguagem XML não é apenas empregada na transmissão de mensagens, como será mostrado nas próximas seções. Ao oferecer um padrão de representação de dados flexível e inerentemente extensível a linguagem reduz significantemente o esforço de desenvolvimento de várias tecnologias necessárias para garantir o sucesso dos serviços *web*.

Web Services Description Language (WSDL)

WSDL é uma linguagem baseada em XML destinada à especificação de interfaces de serviços *web*. A implementação do serviço pode-se dar utilizando qualquer linguagem de programação e plataforma. Pode-se utilizar ferramentas para geração de código que, a partir da especificação da interface, geram os *stubs* e os *skeletons* do serviço, na linguagem de programação desejada.

Um documento WSDL define serviços como um conjunto de *endpoints* de rede ou portas. A definição abstrata de *endpoints* e de mensagens é separada da definição concreta quanto ao protocolo de rede utilizado e formatação dos dados [13]. Isto possibilita o reuso de definições abstratas como mensagens (descrições abstratas dos dados sendo trocados) e *Port Types* (coleções abstratas de operações). A especificação do protocolo de comunicação e a especificação da formatação dos dados para um *Port Type* constitui uma associação reutilizável. Uma porta é definida pela associação de um endereço de rede com uma associação reutilizável e uma coleção de portas define um serviço. Resumidamente, um documento WSDL usa os seguinte elementos XML na definição do serviço:

- Tipo (*Type*): contêiner para definições de tipos de dados. Usa algum sistema de definição de tipos como por exemplo *XML Schema Definition* (XSD).
- Mensagem (*Message*): definição abstrata dos dados a serem transmitidos e seus respectivos tipos.
- Operação (*Operation*): descrição abstrata de uma funcionalidade suportada pelo serviço.
- Tipo de Porta (*Port Type*): conjunto abstrato de operações suportadas por um ou mais *endpoints*.
- Associação (*Binding*): protocolo de comunicação concreto e especificação da formatação dos dados para um tipo de porta específico.

- Porta (*Port*): um único *endpoint* definido como a combinação de uma associação e um endereço de rede.

- Serviço (*Service*): uma coleção de portas agrupadas logicamente.

A linguagem WSDL define quatro tipos de interações entre o provedor e o cliente do serviço, baseado na quantidade e direção das mensagens trocadas. É possível que haja somente o *recebimento* de uma mensagem pelo provedor do serviço, caracterizando uma interação em sentido único (*one-way*). Outra possibilidade é que exista o *recebimento* e o *envio* de uma mensagem pelo provedor do serviço, caracterizando uma interação requisição-resposta (*request-response*). Ainda, pode-se ter o *envio* e *recebimento* de uma mensagem (*solicit-response*) ou apenas o *envio* de uma mensagem (*notification*) por parte do servidor.

Simple Object Access Protocol (SOAP)

SOAP [14] é um protocolo de comunicação no nível de aplicação que define a formatação e codificação de mensagens XML a serem enviadas pela rede. O protocolo define um mecanismo simples e leve para troca de informações estruturadas entre pares em um ambiente distribuído e descentralizado [15]. Foi projetado para comunicação na Internet, podendo utilizar HTTP ou SMTP para transporte de suas mensagens.

O objetivo desse protocolo é prover um meio de comunicação entre aplicações executando em diferentes sistemas operacionais, com diferentes tecnologias e linguagens de programação. Ao utilizar protocolos padrões da Internet ele não é bloqueado por *firewalls* nem por roteadores como outros protocolos de aplicação (*e.g.*, *Internet Inter-ORB Protocol* - IIOP).

A mensagem SOAP deve necessariamente conter um elemento *envelope*. Este por sua vez possui um elemento cabeçalho (*header*) opcional e um elemento corpo (*body*) mandatório. O elemento corpo representa a informação em si sendo transmitida, ou seja, parâmetros de entrada e saída do serviço. A mensagem ainda pode conter um elemento de falha (*fault*) que provê informações sobre erros ocorridos durante o processamento da mensagem.

Universal Description, Discovery and Integration (UDDI)

UDDI especifica protocolos necessários à descoberta de novos serviços disponíveis na rede [15, 16]. Funciona como uma agência de registro de serviços *web*, similar às páginas amarelas de uma lista telefônica. Dado um conjunto de parâmetros de especificação de um serviço (nome, qualidade, localização, entre outros) a agência é responsável por encontrar serviços cadastrados que satisfaçam os parâmetros fornecidos. A agência mantém um banco de dados centralizado dos serviços *web*

disponíveis na rede classificados por tipo de serviço, informações do provedor, qualidade dos serviços e assim por diante. O protocolo para requisição dessas informações também é definido, sendo este independente de plataforma pois utiliza SOAP para codificação das mensagens.

2.2 Composição de Serviços

Esta seção descreve os mecanismos de composição presentes na arquitetura orientada a serviço. São descritas as tecnologias e plataformas de maior destaque para implementação de tais mecanismos.

2.2.1 Orquestração

Orquestração é um mecanismo de composição ou agregação de serviços visando à obtenção de um novo serviço com funcionalidades mais significativas. É um componente importante da arquitetura orientada a serviços pois encapsula logicamente serviços de forma transparente ao cliente. A orquestração permite que o desenvolvimento de novos serviços seja realizado de forma dinâmica, flexível e adaptável a necessidades de negócios que se alteram freqüentemente [17, 18]. Não menos importante, o tempo de desenvolvimento de novos serviços é fortemente reduzido em função do alto índice de reuso de código [19].

A orquestração de serviços define um fluxo de execução (*workflow*) de um processo, ou seja, um *script* de orquestração que é processado por um programa denominado máquina de orquestração. Nesse *script* são definidas as chamadas a serviços componentes necessárias à obtenção do novo serviço. Como qualquer linguagem de programação, as linguagens de orquestração possuem fluxos condicionais, seqüenciais, atribuição de variáveis, interações, entre outros. Como a tecnologia mais compatível com a arquitetura orientada a serviços é a de serviços *web* as linguagens e tecnologias para composição de serviços que mais se desenvolveram foram aquelas que compõem serviços *web*.

É importante destacar que a orquestração tem por objetivo descrever um processo do ponto de vista de quem o executa, ou seja, somente *um* ponto de vista de execução do processo é conhecido. A menos que se tenha acesso ao código dos serviços componentes é impossível saber qual é o fluxo de execução de cada um deles pois somente as suas interfaces são conhecidas. Em outras palavras, as interfaces mostram o comportamento estático do serviço enquanto que a orquestração mostra o comportamento dinâmico do serviço. A Figura 2.4 mostra um exemplo de composição de serviços *web*. Note que a interação com serviços constituintes é feita através de uma interface WSDL bem definida. O fluxo central mostra o serviço composto ou orquestrador.

A composição de serviços pode ser realizada manualmente, utilizando qualquer linguagem de programação para gerar chamadas aos serviços componentes e gerenciar o fluxo de execução baseado

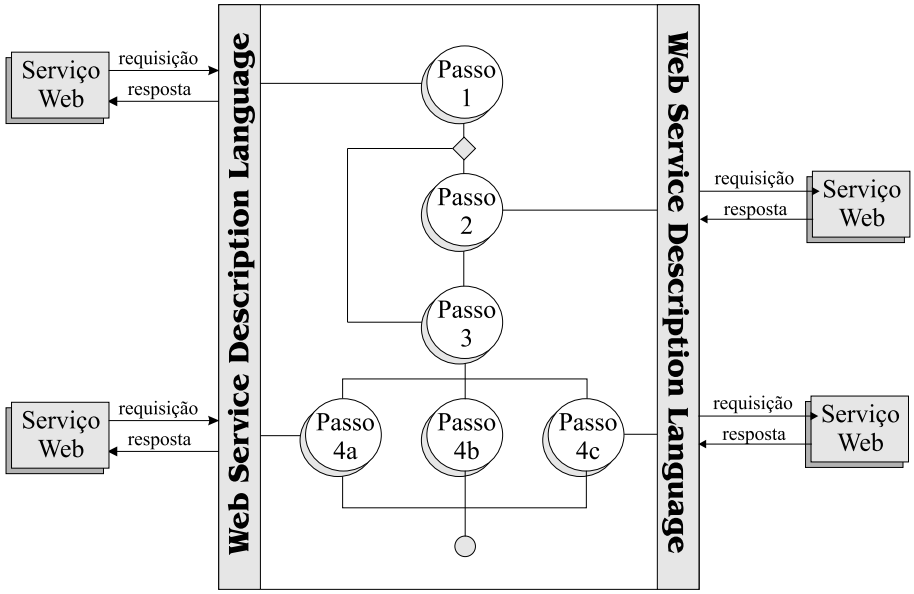

Fig. 2.4: Orquestração de Serviços.

nas respostas recebidas. Entretanto, ao compor serviços dessa forma o programador precisa dominar mecanismos de interação direta com serviços, por exemplo, montagem de *stubs* do serviço remoto e construção manual das mensagens a serem enviadas aos serviços. Para facilitar a composição de serviços foram definidas linguagens de programação específicas que livram o programador desses detalhes e permite que o foco seja dado à lógica de execução do serviço. Dessa forma, idealmente, é possível que um engenheiro de processos especialista em uma área de conhecimento especifique o seu próprio serviço a partir da composição de outros serviços já desenvolvidos.

Para a especificação de uma linguagem de composição de serviços alguns requisitos devem ser preenchidos. Primeiramente, a linguagem deve suportar interações assíncronas pois os processos podem ter longa duração [17]. A manutenção de conexões abertas nas interações síncronas em processos de longa duração representa um alto custo aos servidores de composição. A interação com serviços assíncronos leva à necessidade de um mecanismo de correlação, *i.e.*, um mecanismo que correlacione mensagens às instâncias de serviços. Adicionalmente, a capacidade de invocar serviços concorrentemente é vital por questões de desempenho, uma vez que esta gera a diminuição do tempo de execução do processo como um todo.

As linguagens de composição de serviços também devem prover um meio de gerenciar exceções e garantir a integridade transacional. A garantia de integridade gerada por transações ACID (*Atomocity, Consistency, Isolation and Durability*) não é apropriada para transações de longa duração, uma vez que não é razoável reservar (*lock*) recursos por um longo período de tempo. Para manter a integridade transacional é utilizado então o conceito de compensação, em que ações são tomadas e, caso algo de

errado ocorra, posteriormente estas ações são desfeitas.

Finalmente, as linguagens de composição devem permitir a composição recursiva de serviços, *i.e.*, o serviço composto pela agregação de serviços constituintes pode ser visto como um serviço componente em uma composição de maior nível de abstração. Isso é feito expondo as interfaces dos serviços compostos.

Histórico

Uma das primeiras linguagens de orquestração foi desenvolvida pela *Microsoft* e chamava-se XLANG [20]. Trata-se de uma linguagem baseada em XML para a especificação formal de processos de negócio como interações de longa duração com manutenção de estado. A descrição de um serviço em XLANG é um documento WSDL com extensões que descrevem o comportamento do serviço como parte do processo de negócio. O serviço composto poderia também delegar a serviços puramente descritos em WSDL funcionalidades básicas para implementação do processo de negócio. XLANG focava na criação de um processo de negócio e nas mensagens trocadas entre os serviços *web*. Possuía suporte ao tratamento de exceções. A execução desse código era realizada por um servidor de aplicações da *Microsoft*, o *BizTalk Server*.

Paralelamente a IBM desenvolveu a linguagem *Web Services Flow Language* (WSFL) [21]. Esta linguagem tinha por objetivo descrever fluxos de execução públicos (coreografia) e privados (orquestração). São definidos os dados trocados, a seqüência de execução (modelo de fluxo) e o mapeamento de cada um dos passos no fluxo para operações específicas (modelos globais). Uma interface WSDL do serviço composto é exposta. WSFL possuía suporte ao tratamento de exceções, entretanto não havia suporte direto a transações.

Além de XLANG e WSFL, uma repartição das Nações Unidas responsável pela padronização de mercados e negócios eletrônicos (*UN/CEFACT - United Nations Center for Trade Facilitation and Eletronic Business*) [22] desenvolveu o padrão *Electronic Business using Extensible Markup Language* (ebXML) [23]. Este padrão provê um conjunto de componentes de *middleware* que facilitam a colaboração entre parceiros de negócio. Ainda dentro deste padrão foi criada a linguagem *Business Process Specification Schema* (BPSS) [24], com o objetivo de definir padrões de coreografia e protocolos de comunicação entre os serviços *web*.

A IBM e a *Microsoft* posteriormente se uniram para especificação de uma única linguagem de orquestração e coreografia. Esta linguagem nasceu essencialmente da combinação das linguagens WSFL e XLANG, provendo suporte assim à orquestração e à coreografia. A linguagem resultante dessa junção foi batizada de *Business Process Execution Language for Web Services* (BPEL4WS ou apenas BPEL). Sua versão 1.0 foi lançada em Julho de 2002. Foi formado então um consórcio com a *Siebel Systems*, BEA e SAP para especificação da versão 1.1 da linguagem [25], lançada em Maio

de 2003. Atualmente a manutenção da especificação dessa linguagem é de responsabilidade da organização OASIS (*Organization for the Advancement of Structured Information Standards*) [26], um consórcio internacional sem fins lucrativos cujo objetivo é dirigir o desenvolvimento, convergência e adoção de padrões no comércio eletrônico. Ao assumir responsabilidade pela manutenção da linguagem a OASIS renomeou-a para *Web Services Business Process Execution Language* (WSBPEL). O comitê técnico responsável pela especificação da próxima versão da linguagem inclui um número bem maior de empresas, entre elas a *Adobe System*, *Hewllet-Packard*, NEC, *Oracle* e *Sun*.

Além destas existe ainda uma organização sem fins lucrativos chamada *Business Process Management Initiative* (BPMI - `http://www.bpmi.org`) que especificou um modelo abstrato e uma linguagem baseada em XML para especificação de processos de negócio de longa duração. A linguagem é chamada BPML (*Business Process Modeling Language*) [27] e foi lançada em Novembro de 2002. O modelo da linguagem BPML não define em si mesmo qualquer semântica de aplicação em um domínio específico; ao invés disso, define um modelo abstrato e uma gramática para especificação de processos genéricos. Isto permite que BPML seja utilizado em diferentes aplicações, incluindo colaborações multi-organizacionais e composição de serviços *web*. Entretanto, BPML recebeu pequena aceitação no mercado de desenvolvimento de software.

2.2.2 Coreografi a

A integração de sistemas requer mais do que a simples habilidade de conduzir interações usando protocolos padronizados de comunicação [25]. Quando um processo envolve múltiplas empresas com interesses até mesmo conflitantes é essencial que a interação entre elas seja definida formalmente, através de contratos baseados em um modelo de interação de processos padrão. Tais contratos poderiam ser especificados em linguagem natural (*e.g.*, língua portuguesa), o que é freqüentemente feito. Entretanto, o uso de contratos em linguagem natural resulta em ambigüidades indesejadas.

As interações entre serviços são tipicamente troca de mensagens ponto-a-ponto, síncronas e assíncronas, envolvendo duas ou mais partes, possivelmente de longa duração e com manutenção de estado (*stateful*). É importante notar que cada parte possui uma interface de serviço especificada. Entretanto, as interfaces de serviço provêem uma visão estática da execução de um serviço. Elas não conseguem especificar uma seqüência de execução de métodos que resultem em uma interação semanticamente correta.

Foi nesse contexto que a coreografia de serviços foi criada. Coreografia é um mecanismo de especificação de um protocolo de troca de mensagens entre duas entidades participando em uma interação. Seu objetivo é especificar um contrato (protocolo de troca de mensagens) entre duas entidades independentes de software (típica mas não exclusivamente) localizadas em corporações distintas. O uso da coreografia para especificar contratos dentro de um domínio, *i.e.*, entre serviços dentro de uma

mesma organização traz vantagens. Por exemplo, é possível que diferentes equipes de programadores de uma empresa desenvolvam diferentes serviços capazes de interoperar, desde que ambos sejam compatíveis com o contrato especificado. Entretanto, tipicamente ela é utilizada entre corporações porque nesse ambiente ambigüidades não podem existir em qualquer forma.

Na coreografia, o comportamento externo (público) do serviço é especificado sem revelar qualquer detalhe sobre sua implementação interna (privada). Existem dois motivos para tal desacoplamento. Primeiramente, o provedor de um serviço não tem interesse em revelar suas decisões de implementação aos seus parceiros. Em segundo lugar, o desacoplamento provê liberdade quanto a mudança de aspectos privados de implementação do serviço sem afetar o protocolo já estabelecido.

Um *script* de coreografia provê uma visão global da interação em um ambiente colaborativo [17]. Por global entende-se que ele não representa a visão de uma parte, mas sim a interação entre partes. Isso pode ser visto na Figura 2.5. Cada parte envolvida no processo tem o seu papel na interação descrito, possivelmente mediante um acordo entre as partes.

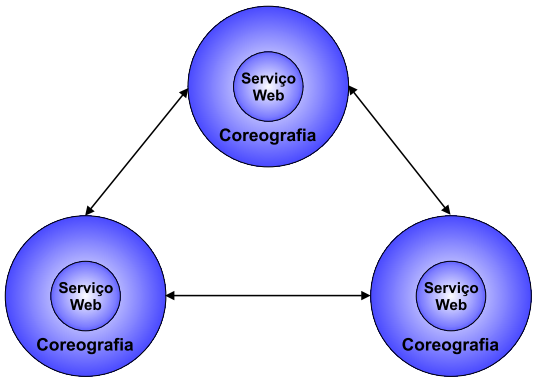

Fig. 2.5: Mecanismo Básico de Coreografia.

Histórico

Uma das primeiras linguagens para coreografia de serviços *web*, especificada pela *Hewlett-Packard*, foi a linguagem *Web Services Conversation Language* (WSCL) [28]. Submetida ao W3C [29], tornou-se uma nota em março de 2002. WSCL permite que as conversações no nível de negócios ou processos públicos suportados por um serviço *web* sejam definidos. São especificados os documentos XML trocados e a seqüência permitida de troca desses documentos. A linguagem é baseada em XML. Tal linguagem não foi absorvida pelo mercado.

Outra linguagem de coreografia especificada pelas empresas *Intalio*, *Sun Microsystems*, SAP e *BEA Systems* foi *Web Services Choreography Interface* (WSCI, lê-se *whisky*). Também baseada em XML, "WSCI é uma linguagem que descreve o fluxo de mensagens trocadas por um serviço *web* participando de interações coreografadas com outros serviços" [30].

O W3C criou um grupo para padronização de um modelo e uma linguagem de coreografia chamado *Web Services Choreography Working Group*. Em março de 2004 foi lançado a especificação do modelo básico de coreografia. Em dezembro de 2004, foi publicado um *Working Draft* da linguagem *Web Services Choreography Description Language* (WSCDL) [31], trabalho corrente deste grupo.

Certamente a linguagem de coreografia mais difundida é BPEL. Através de processos abstratos a linguagem BPEL provê suporte à coreografia. Atualmente a OASIS está trabalhando na especificação da versão 2.0 da linguagem, que tem por objetivo resolver os principais problemas identificados na versão 1.1.

2.2.3 Diferenças Conceituais

Os conceitos de coreografia e orquestração são bem próximos, o que causa algumas divergências quanto a aplicação de cada um. Entretanto, certas diferenças são consenso e serão aqui consideradas para melhor elucidação de cada conceito.

A primeira diferença é que um *script* de orquestração é executável, enquanto que um de coreografia não é. O *script* de coreografia define um protocolo de interação, logo, não pode ser executado.

A coreografia tem uma natureza colaborativa, isto é, na coreografia é definido um contrato de interação em que ambas as partes *colaboram* para atingir um objetivo comum. A coreografia descreve colaborações ponto-a-ponto de um ponto de vista global, onde trocas de mensagens ordenadas resultam em um objetivo comum de negócio com a semântica desejada. Já a orquestração revela o ponto de vista de uma única entidade interagindo com outras entidades. A interação entre os mecanismos de orquestração e coreografia pode ser vista na Figura 2.6. Este é um cenário representativo, em que a coreografia foi utilizada *entre* processos de negócio, possivelmente executando em diferentes empresas.

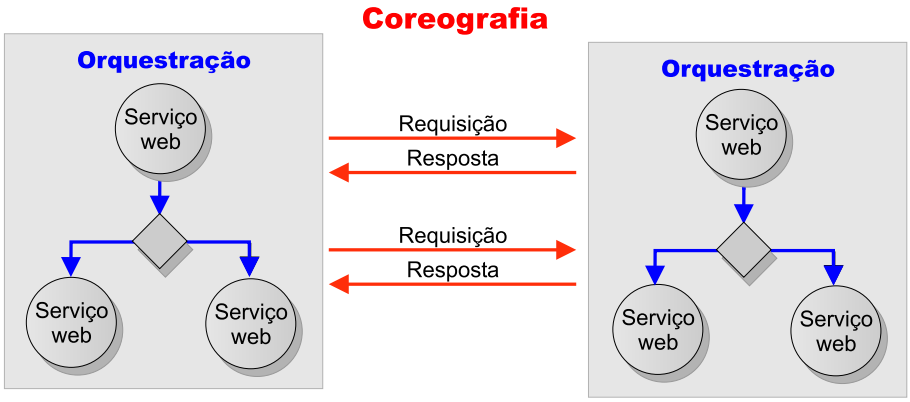

Fig. 2.6: Interação entre os Mecanismos de Orquestração e Coreografia.

2.2.4 Comparativo entre Linguagens

A Figura 2.7 mostra um comparativo de algumas linguagens de composição de serviços com relação à funcionalidade das mesmas.

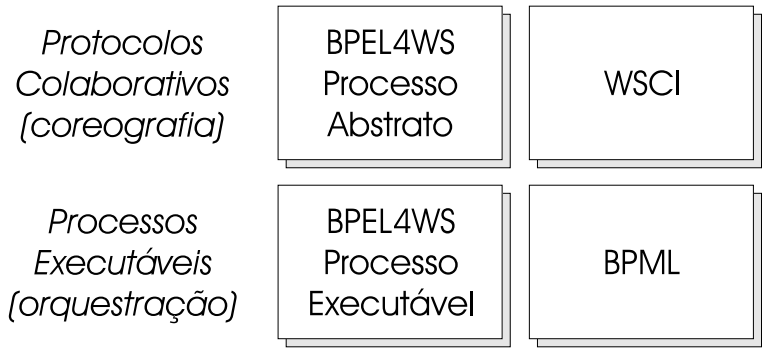

Fig. 2.7: Comparativo de Linguagens de Composição.

A linguagem BPEL pode sozinha realizar tanto a orquestração quanto a coreografia. Juntas as linguagens WSCI e BPML provêm a mesma funcionalidade que a linguagem BPEL. Ambas compartilham o mesmo modelo de execução de processos [17], permitindo dessa forma que desenvolvedores possam usar WSCI para descrever as interações públicas entre processos de negócio e BPML para desenvolvimento de implementações privadas.

Em [32] encontramos um estudo comparativo entre as linguagens BPEL, XLANG, WSFL, BPML e WSCI. Diferentemente dos estudos comparativos realizados, este faz comparações objetivas quanto à capacidade da linguagem em expressar certas construções comuns em linguagens de especificação de fluxogramas genéricos (*workflows*). Tais construções são utilizadas na avaliação dessas linguagens, funcionando como padrões de comparação (*benchmarks*) para as mesmas. Como não existem ainda padrões para avaliação de linguagens de composição de serviço, a utilização desses padrões é válida dada a semelhança funcional de ambas as linguagens. O padrão Escolha Atrasada (*Defereed Choice*), por exemplo, avalia a capacidade de escolha de um dentre vários ramos de execução baseado em informações que não estão necessariamente disponíveis quando este ponto do código é atingido. A Tabela 2.1 mostra um subconjunto dos padrões analisados neste trabalho. As possibilidades de avaliação de cada padrão são as seguintes: (1) a linguagem possui suporte nativo (símbolo "+" na tabela); (2) a linguagem não possui suporte nativo, mas usando uma combinação de outras construções é possível realizá-la ("+/-" na tabela); (3) a linguagem não possui qualquer forma de suporte ao padrão ("-" na tabela).

Ao analisarmos os resultados deste trabalho concluímos que a linguagem BPEL é a que mais provê suporte nativo aos padrões considerados.

Padrão	BPEL	BPML	WSCI
Execução Seqüencial	+	+	+
Execução Paralela	+	+	+
Sincronização	+	+	+
Escolha Exclusiva	+	+	+
Escolha Múltipla	+	-	-
Escolha Atrasada	+	+	+
Junção Simples	+	+	+
Junção Sincronizada	+	-	-
Junção Múltipla	-	+/-	+/-
Finalização Implícita	+	+	+
Requisição-Resposta	+	-	-
Envio de Mensagem	+	-	-

Tab. 2.1: Comparativo entre Linguagens de Composição de Serviços *Web*.

2.3 Business Process Execution Language

Notoriamente, BPEL é a linguagem de orquestração e coreografia de maior destaque atualmente. Devido à alta aceitação da linguagem no mercado, várias ferramentas, tanto abertas quanto proprietárias, foram criadas para o desenvolvimento, instalação e execução de *scripts* BPEL. A presença de grandes empresas no comitê técnico de especificação da próxima versão da linguagem mostra o forte interesse destas no desenvolvimento da linguagem.

2.3.1 Definições Iniciais

Na linguagem BPEL são definidos processos de negócio (*Business Processes*) e protocolos de negócio (*Business Protocols*)[2]. O primeiro é um *script* que define um fluxo de execução sob o ponto de vista de uma entidade que gerencia chamadas a serviços componentes. Uma máquina de software é capaz de interpretar tal *script*, coordenando atividades e compensando-as quando erros ocorrerem. Essencialmente, este *script* modela um fluxo de execução privado. O segundo é um *script* que modela as mensagens trocadas entre os serviços, proporcionando uma visão global entre duas entidades (*e.g.*, provedor de serviço composto e provedor de serviço componente) que interagem em um processo. Um protocolo de negócio não pode ser executado e não mostra detalhes internos de implementação dos participantes na interação. Dessa forma, processos de negócio são identificados como um mecanismo de orquestração e protocolos de negócio como um mecanismo de coreografia [17].

[2]Um processo de negócio pode também ser chamado de processo de negócio executável e um protocolo de negócio pode ser chamado de processo de negócio abstrato. Quando somente o termo *processo* é utilizado, fica implícita a referência a um processo de negócio executável.

Qualquer serviço que participa da composição do serviço orquestrado é considerado um parceiro (*partner*). Parceiros podem realizar chamadas ao processo, prover serviços ao processo ou ambos simultaneamente. As interfaces dos parceiros, bem como do serviço composto, são definidas na linguagem WSDL (adicionada de algumas extensões). Por esse motivo, BPEL é considerado uma camada sobre WSDL. A interface WSDL define operações específicas permitidas sobre aquela entidade de software e o *script* BPEL define como sequenciá-las. A Figura 2.8 mostra a estrutura lógica em camadas das tecnologias envolvidas na composição de serviços BPEL. Vale a pena ressaltar que a linguagem é baseada em XML.

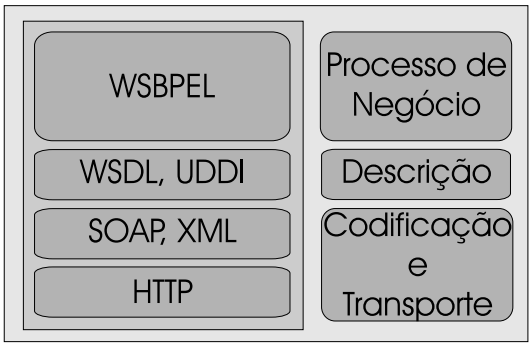

Fig. 2.8: Tecnologias para Composição de Serviços BPEL.

2.3.2 Atividades Básicas

Na linguagem BPEL cada instrução é uma atividade. As instruções são representadas no documento por elementos XML. Atividades podem ser básicas ou estruturadas. Atividades básicas são instruções que interagem com um elemento externo ao processo como, por exemplo, o recebimento, resposta ou invocação de um serviço *web* parceiro. Atividades estruturadas gerenciam o fluxo de execução do processo como um todo. Tais atividades produzem iteração condicional (análogas ao *while* de linguagens de programação conhecidas), execução condicional (*if*), atribuição de variáveis, entre outras. Elas constituem a lógica de programação da linguagem.

As atividades básicas da linguagem BPEL são:

- *receive*: instrução para o recebimento de uma requisição de serviço. É especificado o parceiro que enviou a requisição (*partnerLink*), a porta que recebeu a requisição (*portType*) e a operação requisitada (*operation*). Opcionalmente, uma mensagem de entrada (*variable*) é especificada, caso a operação assim o exija. Pode-se indicar caso o recebimento de tal requisição de serviço deva iniciar uma nova instância do serviço. É uma instrução bloqueante.

- *invoke*: instrução para envio de uma requisição de serviço. É especificado a que parceiro a requisição é destinada (*partnerLink*), a porta que receberá a requisição (*portType*) e a operação requisitada (*operation*). Opcionalmente, uma mensagem de entrada (*inputVariable*) e uma mensagem de saída (*outputVariable*) são especificadas, dependendo da natureza da chamada (*one-way*, *request-response*, *solicit-response* ou *notification*).

- *reply*: instrução para resposta a uma requisição de serviço realizada anteriormente. É especificado a que parceiro a resposta é destinada (*partnerLink*), a porta que recebeu a requisição (*portType*) e a operação requisitada (*operation*). Opcionalmente, uma mensagem de saída (*variable*) é especificada, caso a operação assim o exija. Também pode-se especificar uma falha (*fault*) a ser reportada ao cliente (*faultName*).

- *throw*: instrução utilizada para sinalizar a ocorrência de uma falha interna. Semelhante à construção da linguagem Java. Tem como atributos o nome da falha ocorrida (*faultName*) e uma variável (opcional) que contém detalhes da falha (*faultVariable*).

- *wait*: permite que um processo especifique um atraso por um certo período de tempo (*for*) ou até que um prazo final (*until*) seja atingido.

- *empty*: instrução que não realiza ação alguma.

2.3.3 Atividades Estruturadas

As atividades estruturadas da linguagem BPEL são:

- *sequence*: instrução que contém uma ou mais atividades que são executadas seqüencialmente, na ordem em que são apresentadas dentro do elemento. A atividade completa quando a última atividade terminar sua execução.

- *switch*: instrução que provê comportamento condicional. Consiste de uma lista ordenada de um ou mais ramos condicionais definidos por elementos *case*.

- *while*: instrução que provê suporte à execução repetida de uma dada atividade iterativa. A atividade iterativa é executada até que a condição booleana de verificação do *while* seja falsa.

- *pick*: instrução que aguarda a ocorrência de um dentre um conjunto de eventos e então executa a atividade associada ao evento ocorrido. A ocorrência dos eventos deve ser mutualmente exclusiva. Caso dois ou mais eventos ocorram a escolha da atividade a ser executada é baseada na

ordem de chegada dos eventos. Os eventos podem ser alarmes de tempo ou chegada de mensagens. A atividade termina quando um dos seus ramos de execução é escolhido e a atividade associada àquele ramo termina.

- *flow*: instrução que provê concorrência e sincronização. As atividades incluídas em seu corpo serão executadas paralelamente, com a possibilidade de sincronização de atividades paralelas através da especificação de *links*. Ou seja, quando a execução de uma dada tarefa depende da execução de uma outra tarefa sendo executada em paralelo por outra atividade deve-se usar um *link* para especificar esta relação de dependência.

2.3.4 Peculiaridades da Linguagem

Para manter o estado do processo são utilizadas *variáveis*. Uma variável pode conter integralmente a mensagem XML trocada com um parceiro. Quando um processo recebe uma mensagem proveniente de um parceiro tal mensagem populará uma variável para posterior utilização no processo. Os tipos de dados customizados, *e.g.*, as mensagens trocadas com os parceiros, são definidos nos arquivos WSDL, utilizando a linguagem *XML Schema Definition* (XSD). Variáveis podem representar também dados privados relevantes ao processo ou dados utilizados no controle de iterações. Estas são declaradas no próprio *script* BPEL, utilizando tipos de dados primitivos da linguagem XSD ou tipos customizados definidos nos arquivos WSDL incluídos no *script* BPEL.

A comunicação com os parceiros é descrita utilizando *partnerLinks* que especificam os papéis (*roles*) desempenhados por cada participante da interação. Para definir um papel, deve-se especificar qual tipo de porta (*portType*) a entidade que o implementa necessariamente deve possuir. Por exemplo, para um serviço de aluguel de carros podemos definir os papéis locadora e locatário. A interface WSDL da locadora contém um tipo de porta chamado *LocatorService* com um método *LocateCar*, assíncrono. A interface WSDL do cliente especifica um tipo de porta chamado *ClientPort* com um método *LocateCarCallBack*. O *partnerLink* de interação com o cliente, nesse caso, especificará que a interação envolve os papéis locadora e locatário e que, locadora deve necessariamente implementar o tipo de porta *LocatorService*, bem como o locatário deve implementar o tipo de porta *ClientPort*.

A linguagem possui também um mecanismo muito sofisticado de compensação de atividades. Pode-se definir um bloco de atividades que é totalmente executado com sucesso ou nenhuma das atividades é executada. Esse bloco de código define um escopo de execução (*scope*) onde podem ser criadas variáveis locais, tratadores de exceções (*exceptionHandler*) e tratadores de compensação (*compensationHandler*) com alcance limitado às atividades do escopo. Na prática a compensação é realizada através do *rollback* de atividades terminadas com sucesso quando uma falha ocorre em outra. Tal mecanismo é descrito em detalhes no parágrafo a seguir.

Quando uma falha ocorre dentro de um escopo, todas as atividades em execução dentro do escopo são terminadas. Em seguida é executado o tratador de exceções correspondente àquele escopo, caso algum exista. Caso esse não exista, um tratador de exceções padrão é executado. O tratador de exceções toma ações em função da exceção ocorrida. Se tal exceção comprometer o restante da execução do processo ele pode, por exemplo, terminá-lo. Caso a exceção não seja tão grave, a execução do processo pode ser continuada. O tratador de exceções (e somente ele) pode também solicitar a execução do tratador de compensação do escopo. Este, por sua vez, pode solicitar a execução de atividades compensatórias correspondentes a cada atividade encapsulada no escopo, entretanto somente atividades que terminaram com sucesso podem ser compensadas. Atividades que terminaram com falha ou foram terminadas devido à ocorrência de falha em outra atividade do escopo não podem ser compensadas.

Um processo BPEL pode interagir assincronamente com seus parceiros. Neste tipo de interação o parceiro executa uma chamada em uma interface do serviço e fecha a conexão. Posteriormente o serviço responde realizando uma chamada em uma interface do parceiro. A recíproca também é possível, ou seja, um serviço BPEL pode realizar uma chamada a um parceiro e prover uma interface para recebimento de resposta. Em ambos os casos, cada parceiro deve prover uma interface de serviço *web*. As trocas de mensagens assíncronas dão ao desenvolvedor grande flexibilidade quanto ao momento em que mensagens são enviadas, não impondo restrições quanto ao tempo de resposta da requisição. Isto não ocorre nas arquiteturas cliente servidor, que são síncronas.

O tempo de vida de um processo de negócio pode ser muito longo, quando comparado com aplicações comuns. Em alguns casos não existe previsão quanto ao tempo de resposta do parceiro no negócio. Parte do processo de negócio pode envolver intervenção humana, como um serviço de empréstimos que depende da aprovação final de um gerente bancário. Por esse motivo, processos de negócio podem ser persistentes. Durante longos períodos de inatividade o processo e o seu estado corrente são armazenados em disco, liberando recursos e garantindo escalabilidade. A persistência do processo é totalmente transparente ao programador, devendo ser gerenciada pela máquina de orquestração. Embora a forma de persistência não seja especificada pela linguagem, a maior parte das máquinas de orquestração usa banco de dados.

Quando mensagens são enviadas ao servidor de orquestração uma nova instância de um serviço pode ser criada para tratar tal mensagem. Outra possibilidade é que essa mensagem seja enviada para um processo já instanciado e em execução no servidor. Várias instâncias de um mesmo processo podem estar simultaneamente esperando pelo mesmo tipo de mensagem do mesmo parceiro. Para identificar qual mensagem pertence a qual instância do serviço a linguagem BPEL utiliza um mecanismo de correlação. Ele funciona como um filtro que garante que somente uma mensagem com um dado valor de uma propriedade de correlação pode ser enviada a uma dada instância de serviço

BPEL. Novamente, o envio de mensagens às correspondentes instâncias de serviço é transparente ao programador, ficando a cargo da máquina de orquestração.

Os *links* utilizados na sincronização de atividades também são utilizados para gerar uma árvore de execução. Um *link* pode possuir uma condição de transição. Para que a atividade dependente de uma primeira seja executada, a condição de transição deve ser verdadeira no momento em que a primeira atividade for concluída. Dessa forma constituem-se os caminhos-mortos (*dead-paths*), que são ramos dessa árvore que jamais serão executados pois uma condição de transição de um nó pai foi falsa. Um caminho-morto tem suas condições de transição por definição falsas, portanto, existe a propagação de caminhos-mortos até que sejam atingidos todos os nós folha daquele caminho na árvore. Outro caso de caminho-morto é quando um dos ramos de execução de um *switch* é escolhido. Os outros serão automaticamente considerados como caminhos-mortos.

2.3.5 Plataformas Disponíveis

O forte interesse de grandes empresas no desenvolvimento de aplicações baseadas na arquitetura orientada a serviço e mais especificamente na linguagem BPEL impulsionou a criação de plataformas de desenvolvimento e das máquinas de orquestração. Encontram-se disponíveis soluções comerciais e de código aberto.

A IBM desenvolveu um servidor de orquestração que deve ser instalado no servidor de aplicação Tomcat [33] chamado *Business Process Execution Language for Web Services Java Run Time* (BPWS4J) [34]. A interface de administração desse servidor de orquestração pode ser vista na Figura 2.9. Foi desenvolvido também um *plug-in* Eclipse para edição de documentos BPEL. Ambos foram desenvolvidos pela equipe de pesquisa *Component Systems Group*, disponibilizados no *site* de tecnologias emergentes (*alphaWorks*) e são livres (não-comerciais, embora não tenham código aberto). Além desses, o servidor comercial da IBM *WebSphere* possui também suporte a orquestração BPEL.

A *Oracle* também desenvolveu uma plataforma de desenvolvimento BPEL, chamada inicialmente de *Collaxa BPEL Server* e recentemente de *Oracle BPEL Process Manager* [35]. Essa plataforma inclui um servidor com suporte à orquestração, batizado *Oracle Process Manager Server*, que é instalado no servidor de aplicação OC4J (*Oracle Application Server Containers for J2EE*) ou *WebLogic* da BEA. É disponibilizada também uma interface gráfica de desenvolvimento chamada *Oracle BPEL Process Design*, essencialmente um *plug-in* para o *JDeveloper* [36]. Esta interface pode ser visualizada na Figura 2.10. Por meio da interface para gerenciamento de processos (*Oracle BPEL Process Manager Console*) é possível instalar, verificar estado de execução e até mesmo depurar processos. Esta plataforma possui um alto grau de integração, resultando em rápida instalação, teste e validação de serviços orquestrados. Alternativamente, também é disponibilizado um *plug-in* Eclipse para

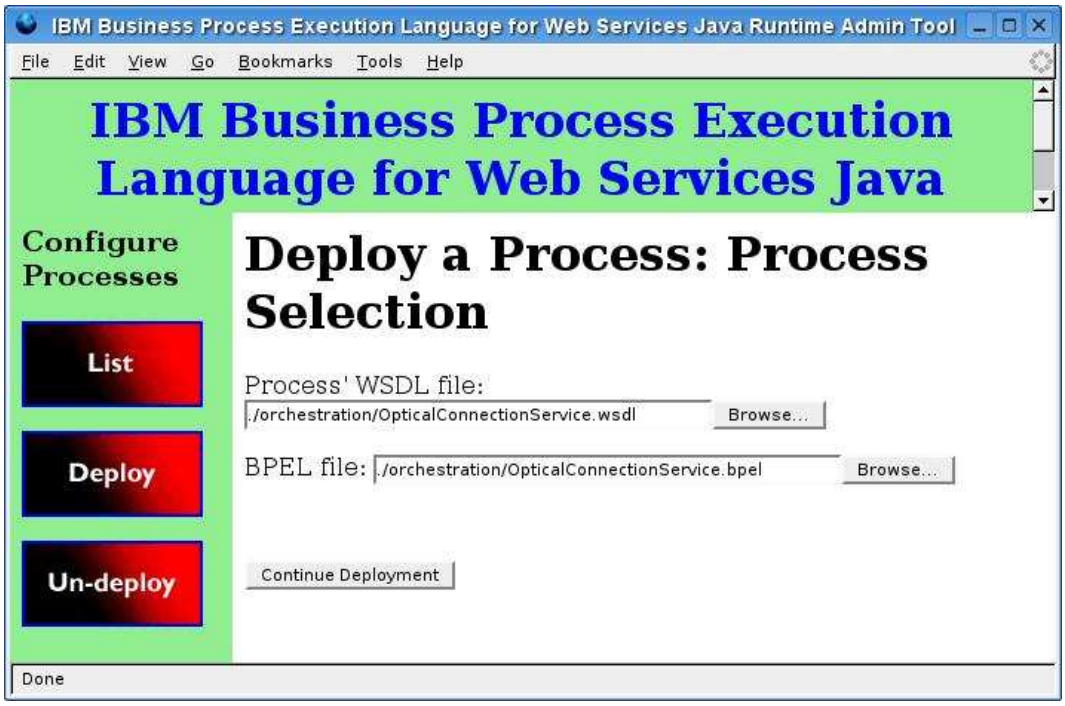

Fig. 2.9: Interface de Administração do Servidor BPWS4J.

edição de documentos BPEL.

A *Microsoft* provê suporte a linguagem BPEL através do servidor comercial *BizTalk Server 2004* [37].

A máquina de orquestração com código aberto mais difundida é a *ActiveBPEL* [38], desenvolvida pela *Active Endpoints* e instalada no Tomcat. Esta empresa possui uma interface gráfica de desenvolvimento chamada *ActiveWebFlow Professional* [39], um *plug-in* para o Eclipse. Tal interface possui integração com a máquina de orquestração, facilitando o processo de instalação dos serviços. Também possui ferramentas para depuração de serviços. Uma interface *web* mostra dados estatísticos sobre o servidor de orquestração, tais como número de processos ativos, número de serviços instalados, estado de execução dos processos, entre outros. Esta interface pode ser visualizada na Figura 2.11.

Por meio desta interface de gerenciamento é possível obter o estado de execução dos processos instanciados desde o momento em que a máquina de orquestração foi iniciada. O valor de todas as variáveis pode ser inspecionado, bem como o resultado de cada atividade executada. É exibido também uma imagem que representa o processo, com semântica definida pela próprio fabricante. Tais dados podem ser utilizados na depuração dos processos (Figura 2.12).

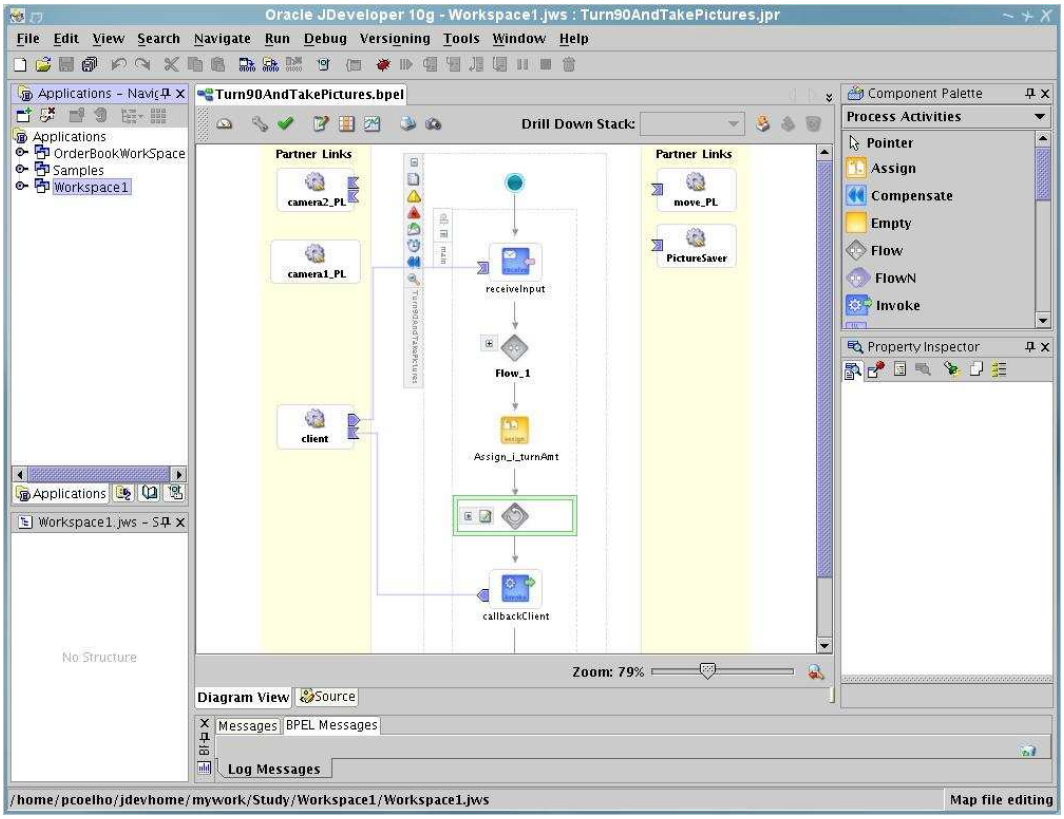

Fig. 2.10: Interface de Desenvolvimento BPEL (*JDeveloper*).

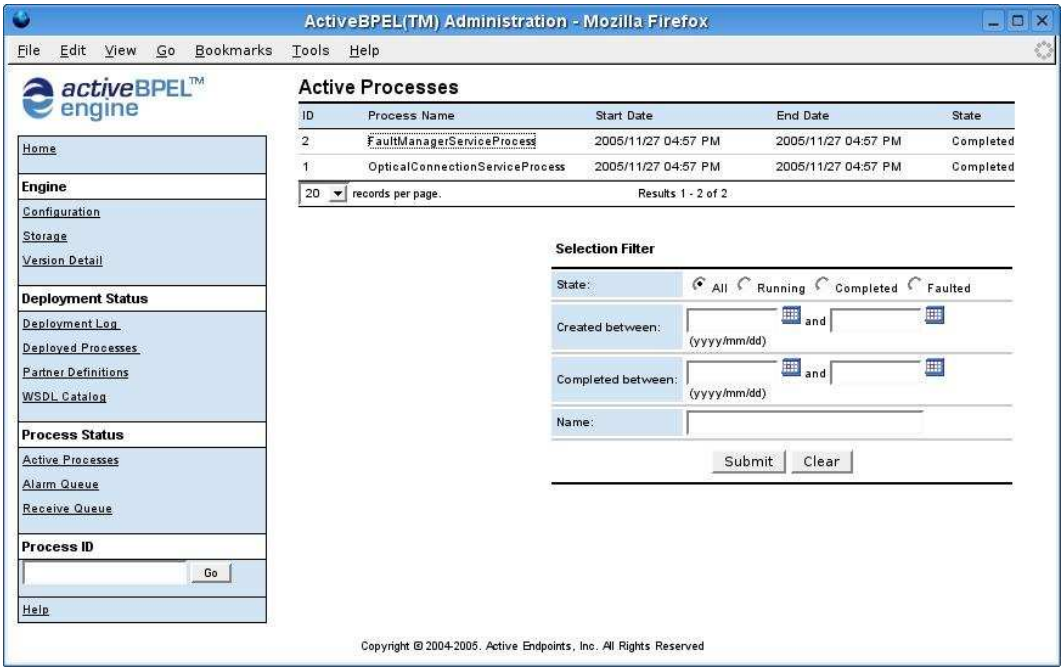

Fig. 2.11: Interface de Gerenciamento do Servidor *ActiveBPEL*.

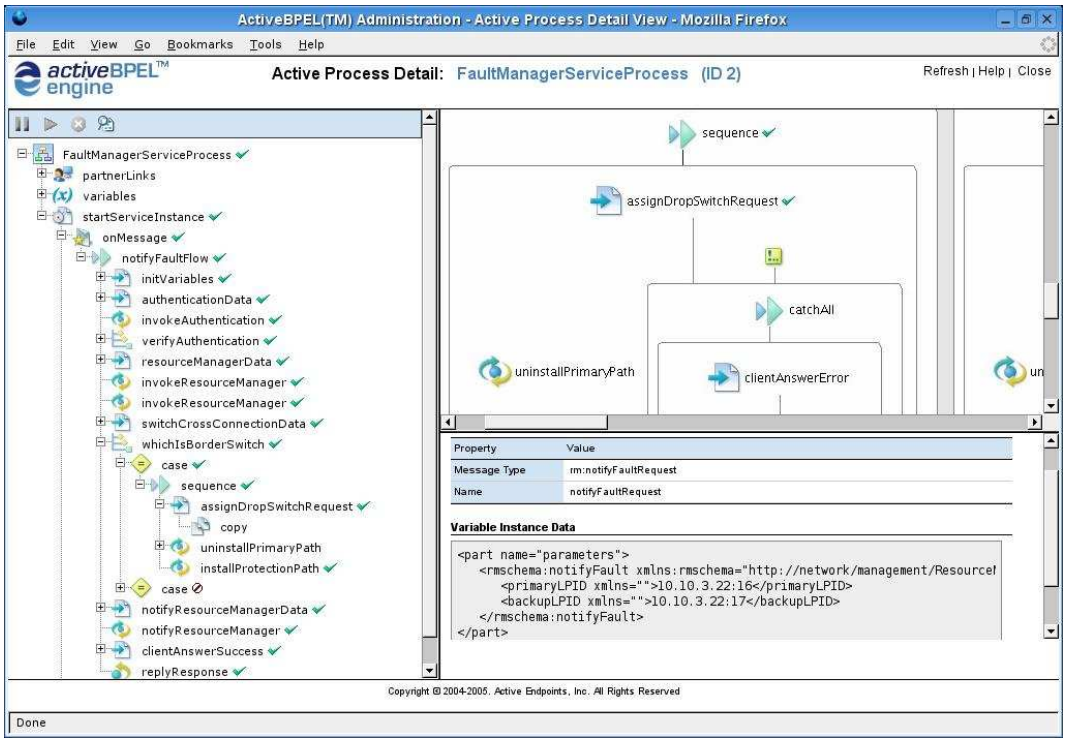

Fig. 2.12: Visualização do Estado de um Processo (*ActiveBPEL*).

Capítulo 3

A Arquitetura SOANet

Este capitulo traz a descrição da arquitetura SOANet (*Service Oriented Architecture for Network Services*) proposta neste trabalho. A arquitetura é baseada no paradigma de computação orientada a serviço e na arquitetura orientada a serviços, detalhados no Capítulo 2.

3.1 Visão Geral

A arquitetura SOANet tem por principal objetivo possibilitar o desenvolvimento rápido e flexível de novos serviços de rede. A arquitetura pode ser aplicada a qualquer rede orientada a conexão, como redes MPLS (*Multiprotocol Label Switching*), ATM (*Asynchronous Transfer Mode*) e ópticas. Conforme mencionado na Seção 1.1, novos serviços de rede precisam ser desenvolvidos em um período de tempo estritamente pequeno. A reusabilidade de código presente na arquitetura orientada a serviços contribui para a solução deste problema. Além disso, o alto grau de granularidade e o fraco acoplamento permite que serviços complexos sejam desenvolvidos com maior facilidade na arquitetura SOA.

Este trabalho propõe a aplicação da arquitetura orientada a serviços ao domínio dos serviços de rede. Na arquitetura SOA cada entidade lógica é vista como um serviço, podendo este ser construído a partir de um sistema legado existente ou implementado sem qualquer código inicial. Genericamente falando, serviços básicos de rede podem ser classificados funcionalmente em dois níveis, o nível de negócios e o de rede. Serviços no nível de negócios estão relacionados à empresa em si (*e.g.*, subscrição), enquanto serviços no nível de rede estão relacionados à rede de transporte (*e.g.*, roteamento). Como mencionado anteriormente, existe uma separação entre serviços localizados nesses dois níveis em termos de tecnologias de software na qual estes serviços são baseados. Uma abordagem comum para solucionar tal separação é empregar um programa *gateway* que converte as decisões de negócio (de alto nível) em sinalização de rede (de baixo nível). Esta não é uma solução apropriada pois *ga-*

teways são sempre gargalos para interoperabilidade, confiabilidade e desempenho. Além disso, esta abordagem amarra fortemente as camadas de negócios e de rede, comprometendo a automatização, customização e evolução dos serviços.

Para evitar a presença de *gateways* é proposta a adição de uma interface de serviço nos elementos de rede como roteadores e comutadores. Cada elemento de rede ofereceria funções de controle e gerenciamento através de uma interface de serviço *web*. Como atualmente a maior parte dos elementos de rede já hospedam um servidor *web*, estendê-lo para realizar o processamento de mensagens SOAP é um passo pequeno sem impacto significante ao custo do equipamento nem ao tempo de desenvolvimento do software embarcado. A presença destes serviços nos elementos de rede pode eliminar a necessidade de complexos protocolos de sinalização de rede, possibilitando uma integração direta entre as camadas de negócios e de rede. Por exemplo, o protocolo de sinalização para estabelecimento de conexões em redes ópticas (usualmente RSVP-TE) é substituído por um *script* de orquestração muito mais simples. Desta forma, é construído um único barramento de comunicação baseado em serviços dentro de um domínio de rede, onde um serviço composto central é responsável por receber as requisições dos clientes e coordenar todas as chamadas aos serviços componentes. A arquitetura SOANet é exibida na Figura 3.1.

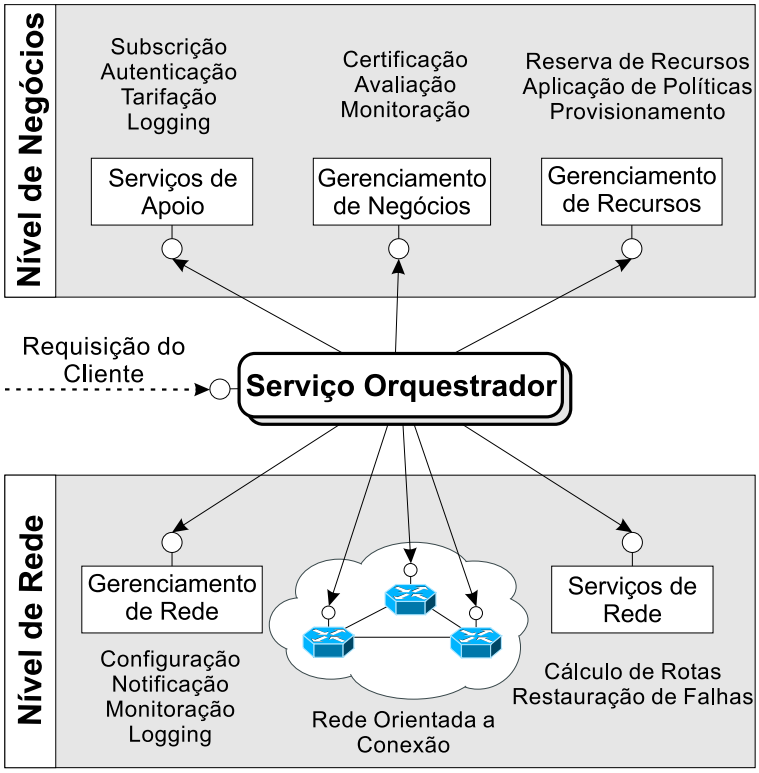

Fig. 3.1: Arquitetura Servidor.

Dentro de cada nível os serviços foram agrupados funcionalmente por clareza e simplicidade.

Cada círculo representa uma interface de serviço *web* que pode ser acessada pelo serviço central de composição. Apesar dos serviços terem sido agrupados vale a pena ressaltar que *cada* um deles expõe *uma* ou *mais* interfaces de serviço, *i.e.*, não é o conjunto funcional inteiro que exibe uma interface de serviço.

Com todos os serviços componentes especificados e implementados é possível realizar a composição de um serviço agregado de forma muito flexível, levando em conta requisitos específicos de um cliente. É importante que durante a etapa de especificação de cada serviço não sejam introduzidas dependências não reais entre serviços. Isto é, os dados que são entregues ao serviço para execução de uma tarefa devem estar *estritamente* ligados ao mesmo, favorecendo o fraco acoplamento entre serviços.

Utilizando serviços *web* é possível realizar a escolha de serviços a serem utilizados em tempo de execução [40], levando em conta requisitos do cliente. Para isso é necessário o uso de mecanismos de registro e busca de serviços. Entretanto, a escolha de serviços em tempo de execução parece ainda estar em sua infância [11], isso porque é preciso estabelecer formalmente meios de classificar [41] e atribuir semântica [42, 43] aos serviços. Vários mecanismos e linguagens já foram propostos, entretanto as soluções se mostram ainda pouco desenvolvidas.

3.2 Composição Fim-a-Fim

Note que a arquitetura proposta na Seção 3.1 é aplicável dentro de um domínio de rede. Entretanto, utilizando o conceito de composição recursiva de serviços, pode-se facilmente estabelecer serviços fim-a-fim através da composição dos serviços fornecidos por cada domínio de rede. Isto pode ser visto na Figura 3.2.

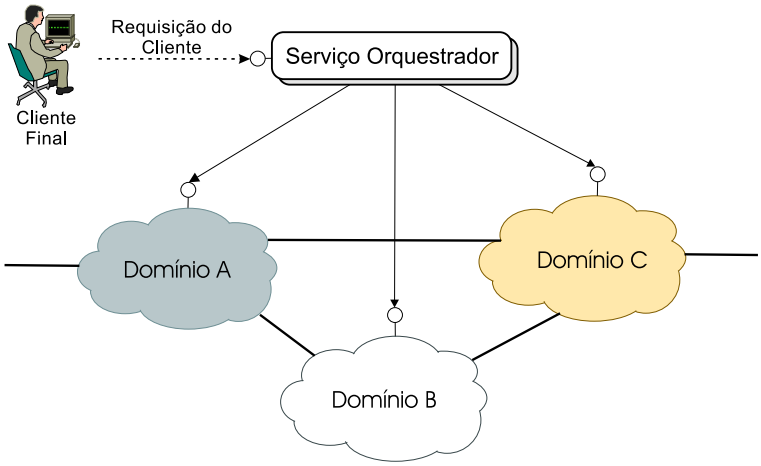

Fig. 3.2: Composição de Serviços Fim-a-Fim.

Soluções que utilizam sinalização de rede entre domínios não são bem vistas pelos provedores de rede. A abertura da rede para sinalização externa parece nunca realmente ter ocorrido por questões de segurança. Administradores de rede não se sentem confortáveis com fato de a gerência de seus equipamentos internos poder ser realizada externamente, por exemplo, com estabelecimento de circuitos com rota na origem. A solução aqui apresentada resolve tais questões, na medida em que todo o gerenciamento dos equipamentos internos ao domínio é realizado pelo serviço do domínio. Baseado em políticas internas ele é capaz de aceitar ou rejeitar requisições realizadas.

O serviço fim-a-fim não necessariamente precisa ser implementado por um provedor de rede, isto é, ele pode ser provido por uma terceira parte. Essa entidade precisa ter contratos estabelecidos com cada um dos provedores de serviço de cada domínio para vender um serviço de valor agregado a clientes finais. Outra possibilidade é que cada domínio implemente um serviço orquestrado para provimento de serviços fim-a-fim, permitindo que seus clientes possam ter acesso a tais serviços. Nesta abordagem o provimento dos serviços fim-a-fim fica distribuído, resultando em maior escalabilidade do sistema como um todo.

3.3 Arquitetura do Cliente

O cliente do serviço realiza suas requisições através da interface exibida na Figura 3.1. Para o cliente, a implementação do serviço não é revelada, somente a interface do serviço. É papel do cliente da rede obter a especificação da interface com a qual ele deseja interagir e montar sua requisição ao servidor, de forma coerente com a especificada na interface.

Para localização do serviço pode ser usado um mecanismo de registro e descoberta. No contexto de serviços de rede, acredita-se que tal mecanismo não traga muitas vantagens, na medida em que, geralmente, um dado cliente está conectado a uma rede de acesso apenas. Neste caso a localização do serviço com o qual ele deseja interagir pode ser fornecida pelo próprio provedor no momento de assinatura do contrato. Entretanto, caso o provedor do serviço deseje alterar freqüentemente a localização do serviço tal mecanismo pode ser de grande valia.

Os clientes de um domínio de rede podem ser usuários finais, provedores de rede compondo serviços fim-a-fim ou provedores de serviço que apenas compõem serviços sem necessariamente ser um provedor de rede. Um usuário final está interessado em estabelecer conexões com uma dada máquina que se encontra atrás do domínio com o qual ele interage, diretamente conectada a esse domínio. Já o provedor de rede ou de serviços possui um processo de negócio, um programa que utiliza serviços de diferentes domínios para atingir seu objetivo. Neste caso, é necessário que um contrato muito bem definido seja utilizado para especificar sem ambigüidades a interação entre os serviços. Elabora-se então um *script* de coreografia, definindo formalmente a interação entre o cliente

do serviço (que no caso é um provedor) e o servidor. A Figura 3.3 mostra tal cenário.

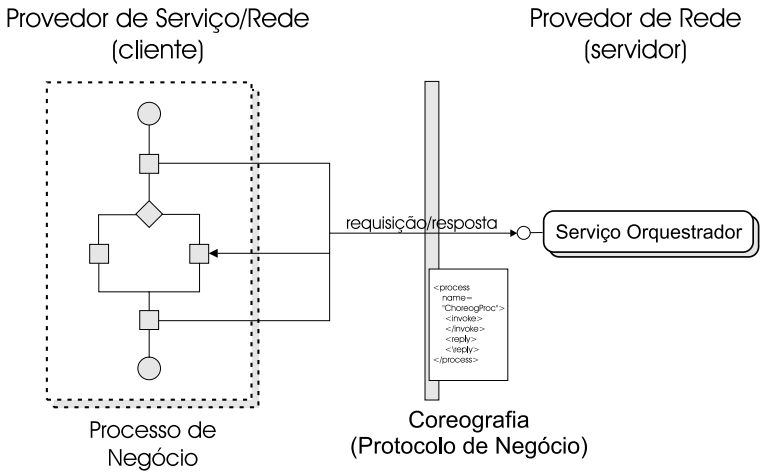

Fig. 3.3: Interação com Provedores de Serviço ou de Rede.

3.4 Implementação dos Serviços

Para implementação dos serviços compostos podem ser utilizadas ferramentas para agilizar o processo de desenvolvimento. Primeiramente é possível que o programador, que conheça bem o funcionamento dos serviços componentes e uma linguagem de orquestração, edite manualmente os *scripts* de orquestração. Este é um trabalho árduo, na medida em que os *scripts* podem ser tornar realmente extensos, dificultando sua manutenção.

Uma abordagem extremamente elegante é a utilização de uma linguagem de modelagem genérica como UML (*Unified Modeling Language*) para especificação das interações com os serviços componentes, usando, por exemplo, um diagrama de atividades UML [44, 45]. Para isto, é necessária a especificação de um novo perfil UML capaz de representar o domínio de interesse, no caso, composição de serviços. É também necessário utilizar transformações que tenham como entrada os modelos UML e produzam o código de composição na linguagem de orquestração alvo.

Finalmente, a maneira mais conveniente de criar serviços compostos é utilizando um ambiente de criação de serviços, um programa projetado especificamente para esta tarefa [39, 36, 46]. Esses ambientes empregam interfaces gráficas dedicadas, com termos, ícones e diagramas que devem ser conhecidos pelo desenvolvedor do sistema. Em geral, o aprendizado de tais ferramentas demanda menor esforço do que as linguagens de composição. A saída dessas ferramentas é o *script* de orquestração pronto para execução em uma máquina de orquestração.

Em qualquer uma das possibilidades de criação de serviços aqui apresentada, a composição de serviços leva a uma redução no tempo de desenvolvimento, um dos objetivos da arquitetura SOANet.

3.5 Instalação dos Serviços

Ao utilizar-se a composição de serviços a instalação se torna um passo extremamente simples. Basta que o serviço composto seja instalado na máquina de orquestração que vai executá-lo. Tal procedimento pode ser realizado via interface *web* fornecida pela máquina de orquestração. Pode também ser realizada pela simples cópia dos arquivos do serviço em um diretório apropriado da máquina de orquestração. Alguns ambientes de desenvolvimento de serviços já possuem integração total com a máquina de orquestração, permitindo que a instalação do serviço seja feita de forma direta, através de uma simples solicitação realizada ao ambiente.

As máquinas de orquestração da linguagem BPEL, em geral, necessitam de um arquivo de instalação com detalhes relevantes à instalação que não podem ser expressos pela linguagem. Esse arquivo de instalação não possui um padrão especificado e, por conseqüência, cada fabricante utiliza um formato próprio. A edição desses arquivos pode ser realizada pelos próprios ambientes de desenvolvimento de serviços ou manualmente. Entre as informações que devem estar presentes nesses arquivos está a localização dos arquivos WSDL dos clientes.

3.6 Análise Comparativa

A arquitetura SOANet propõe um novo paradigma para o desenvolvimento de serviços de rede. Trata-se de uma proposta radical, pois ela é capaz de substituir serviços e protocolos de rede que, embora apresentem problemas de interoperabilidade e forte acoplamento, são amplamente aceitos e bem estabelecidos entre os provedores de rede.

A característica mais inovadora da arquitetura é a exposição de funcionalidades de gerenciamento e controle de cada elemento de rede como um serviço. Dessa forma um *script* de orquestração pode substituir protocolos de sinalização utilizados, por exemplo, para o estabelecimento de conexões. Tais protocolos de sinalização de rede apresentam a característica de distribuição da inteligência do sistema na medida em que cada nó pode processar mensagens de sinalização e tomar decisões baseadas nestas. A arquitetura SOANet, por sua vez, possui um certo grau de centralização, pois um serviço orquestrador é responsável pelo provimento de serviços ao cliente dentro de um domínio. A presença de um único serviço orquestrador em um domínio pode ser considerado um elo fraco com respeito ao desempenho frente a um grande número de requisições e confiabilidade do sistema como um todo.

Entretanto, a arquitetura poderia ser relaxada para permitir a presença de mais de um serviço orquestrador por domínio. Dessa forma, problemas de escalabilidade, desempenho e confiabilidade do serviço são amenizados. Cada elemento de borda da rede poderia expôr aos clientes serviços

compostos dentro do domínio. Contudo, ao utilizar serviços compostos distribuídos é preciso realizar o controle de acesso a recursos (ou serviços) compartilhados tendo em vista a manutenção de um estado consistente.

Por outro lado, a presença de um elemento central apresenta vantagens. Uma delas é a possibilidade de instalação concorrente de conexões em cada elemento de rede. Protocolos de sinalização em geral instalam conexões serialmente, ou seja, cada elemento realiza a instalação da conexão após o elemento anterior tê-la instalado. Tal fato pode representar um ganho significativo no tempo de estabelecimento de uma conexão, principalmente quando o número de elementos de rede em um circuito é grande. Neste caso, a arquitetura SOANet não sofre degradação, apresentando um comportamento constante (*O(1)*), enquanto protocolos tradicionais apresentam um comportamento linear (*O(n)*).

3.7 Considerações Finais

Neste capítulo foi apresentada a arquitetura SOANet. Esta arquitetura é independente de linguagem e de plataforma. Ela pode ser instanciada utilizando qualquer linguagem de orquestração e coreografia desejadas.

Serviços de rede de complexidade arbitrária podem ser facilmente construídos com base nesta arquitetura. Isto porque a composição de serviços fornece grande flexibilidade e um alto índice de reusabilidade de código. A composição realizada através de ambientes de criação de serviço reduz drasticamente o tempo de desenvolvimento de novos serviços, possibilitando que estes sejam implementados levando em conta requisitos específicos de um cliente.

Fazendo uso da arquitetura é possível criar serviços de rede fim-a-fim de forma muito estruturada e segura para o provedor do domínio. Apenas utilizando uma interface provida por um domínio de rede, um agregador de serviços é capaz de compor serviços com a mesma rapidez e flexibilidade obtidas no ambiente intra-domínio.

Capítulo 4

Estudo de Caso: Redes Ópticas

Este capítulo apresenta o projeto, implementação e testes de dois serviços de rede desenvolvidos para validação da arquitetura SOANet. Esses sistemas têm por objetivo o gerenciamento de conexões e de falhas em redes ópticas.

4.1 Projeto

Para validação da arquitetura SOANet foram desenvolvidos dois serviços para redes ópticas comutadas por comprimento de onda construídas com equipamentos dotados de tecnologia DWDM (*Dense Wavelength Division Multiplexing*). Estas redes também são chamadas de redes ópticas de segunda geração. Como em qualquer rede orientada a conexão, em redes ópticas antes que o tráfego de dados possa ser enviado pela rede é necessário o estabelecimento de uma conexão (ou circuito). Nestas redes ópticas, conexões são caminhos de luz (*lightpaths*) que podem atravessar um número ilimitado de elementos de rede. Esses podem ser multiplexadores OADM (*Optical Add-Drop Multiplexer*) ou comutadores ópticos (*optical switches*) - tipo OXC (*Optical Cross-Connects*) ou tipo PXC (*Photonic Cross-Connects*).

O primeiro serviço especificado e desenvolvido foi o Serviço de Gerenciamento de Conexões (SGC) cujo objetivo é permitir que o cliente de rede possa criar e destruir caminhos de luz. O segundo, Serviço de Gerenciamento de Falhas (SGF), é responsável pela restauração de conexões em caso de falha.

Seguindo a arquitetura SOANet, cada comutador óptico no domínio exporá um serviço de conexão cruzada (*cross connection*), possibilitando que a máquina de composição instale as conexões necessárias nos comutadores. Como o tempo requerido para instalar uma conexão em um comutador óptico é alto (da ordem de milisegundos), é esperado que os serviços *web* não introduzam carga considerável no tempo de estabelecimento de uma conexão. Todos os outros serviços componentes

necessários poderiam ser instalados em qualquer local na rede, até mesmo em um domínio externo (*e.g.*, um serviço de cobrança pode ser provido por uma operadora de cartão de crédito).

4.1.1 Especificação de Requisitos

Requisitos Funcionais

Os requisitos funcionais de cada um dos serviços a serem desenvolvidos são:

- Serviço de Gerenciamento de Conexões: serviço que oferece funcionalidades de criação e eliminação de caminhos de luz primários (ou principais). Pode ser solicitada a provisão de um caminho de luz de proteção associado a este caminho primário. O caminho de luz de proteção pode ser dedicado (protege um único caminho primário) ou compartilhado (protege mais de um caminho primário). Os caminhos primário e de proteção devem ser disjuntos. Pode ser solicitada a instalação de um caminho de luz com rota e alocação de comprimentos de onda explícitas, *i.e.*, todos os nós no caminho e os comprimentos de onda associados a cada enlace são fornecidos pelo requerente do serviço. Tal recurso pode ser útil para o administrador do domínio para fins de balanceamento de carga. Quando erros ocorrem durante o processamento do serviço detalhes devem ser armazenados de forma persistente para posterior processamento. O cliente do serviço deve ser tarifado em função do uso do serviço e contrato previamente acordado. O serviço deve ser capaz de autenticar o cliente da rede realizando uma requisição.

- Serviço de Gerenciamento de Falhas: serviço que oferece funcionalidades de restauração e eliminação de falhas. Deve ser capaz de restaurar caminhos dedicados e compartilhados. Por eliminação de falhas[1] deve-se entender o retorno da rede ao estado anterior à ocorrência da falha, ou seja, os dados que estão sendo transmitidos pelo caminho de proteção devem passar a ser enviados novamente pelo caminho principal. Quando erros ocorrem durante o processamento do serviço estes devem ser armazenados de forma persistente para posterior processamento.

Para que os requisitos acima pudessem ser atingidos foram identificados os seguintes serviços componentes e seus requisitos funcionais:

- Serviço de Conexão Cruzada (SCC): serviço exposto pelo comutador óptico provendo facilidades de comutação cruzada de comprimentos de onda.

- Serviço de Gerenciamento de Recursos (SGR): serviço responsável por armazenar o estado atual da rede, ou seja, armazenar dados dos caminhos de luz instalados, recursos disponíveis e

[1] Tradução do termo *clear fault* (em inglês).

contratos de provisão de serviço (*Service Level Agreements* - SLAs). Armazena informações sobre caminhos de proteção associados a um caminho de luz primário.

- Serviço de Roteamento (SR): serviço responsável pela computação de uma rota dentro da rede óptica, dados um nó de ingresso, um nó de egresso e os recursos disponíveis na rede. Além da rota ele deve ser capaz de alocar um comprimento de onda disponível nesta rota. Responsável também por encontrar um caminho de luz de proteção para um caminho primário, sendo que estes devem necessariamente ser disjuntos.

- Serviço de Autenticação (SA): serviço responsável por autenticar os usuários do serviço de rede.

- Serviço de *Logging* (SL): serviço responsável por armazenar persistentemente dados referentes a erros ocorridos durante a execução de um serviço orquestrado.

- Serviço de Tarifação (ST): serviço responsável por contabilizar os recursos efetivamente utilizados por um cliente e realizar a tarifação sobre os mesmos.

Requisitos Não-Funcionais

O principal aspecto não-funcional envolvido é segurança e privacidade dos dados. Para que seja possível o uso do serviço o cliente ou administrador da rede deve ser autenticado através de um nome de usuário e senha. As requisições e os dados de autenticação devem ser enviados de forma segura ao provedor de serviço garantindo a privacidade dos dados.

O tempo de instalação de um caminho de luz e o tempo de restauração de um caminho de luz em falha não devem ser excessivamente grandes. Operadoras de rede consideram que a restauração deva ser realizada em no máximo 50 milisegundos. Entretanto, no sistema proposto, um valor aceitável seria o de 500 milisegundos para restauração, 10 vezes o valor padrão das operadoras. Vale a pena ressaltar que a restauração é realizada via software, por isso tal valor pode ser considerado aceitável. Já o tempo de instalação de um caminho de luz não possui restrições severas e 1000 milisegundos pode ser considerado um valor compatível. Em geral, como os caminhos de luz permanecem instalados por um longo período de tempo na rede, existe uma tolerância maior quanto ao tempo de instalação. Se comparado com tempos de instalação de caminhos de luz criados manualmente por gerência - que podem levar horas ou até mesmo dias - esse valor é extremamente pequeno.

Ambos os sistemas devem ser altamente estáveis e robustos. Os sistemas devem possuir alta disponibilidade, *i.e.*, devem permanecer o menor período de tempo desligados (por exemplo em caso de atualizações).

4.1.2 Diagramas de Classes

Os diagramas de classes dos serviços SGC e SGF podem ser vistos nas Figuras 4.1 e 4.2, respectivamente. Os serviços compostos são representados pelo nome do serviço, em inglês[2], adicionado da palavra *Engine* (máquina), pois na verdade quem realiza as chamadas aos serviços componentes é a máquina de orquestração, alimentada do *script* de orquestração. Ambos os diagramas mostram somente a interação dos serviços compostos com a interface dos serviços componentes, por facilidade de visualização. Os parâmetros de cada método foram omitidos nos diagramas mas serão detalhados textualmente, quando relevantes.

Estes diagramas de classes foram obtidos tendo em vista os requisitos funcionais e não-funcionais explicitados na Subseção 4.1.1. Cada um dos serviços lá especificados teve suas funcionalidades expostas diretamente em uma interface de serviço. As interfaces foram nomeadas com o nome do serviço, em inglês, adicionado do sufixo *PortType*. Os dados que devem ser armazenados de forma persistente são gerenciados por Java *beans*, que realizam o interfaceamento entre o sistema e o banco de dados. Desta forma obtém-se uma arquitetura em três camadas (*3-tier architecture*), com classes de interface, controle e persistência. As classes de controle são as que efetivamente implementam a lógica exposta nas classes de interface, e realizam chamadas às classes de persistência quando necessário.

Foi especificado um diagrama de classes para cada um dos serviços componentes. Estes serão explicitados a seguir. Entretanto, inicialmente é útil definir uma classe que represente um caminho de luz (classe *Lightpath*), dado que este é um parâmetro muito usado. Os atributos de um caminho de luz são os seguintes:

- *switchingType*: tipo de comutação do caminho de luz (comprimento de onda, fibra, faixa de comprimentos de onda);
- *ingressNode*: nó de ingresso na rede;
- *egressNode*: nó de egresso da rede;
- *color*: classe de serviço a que o caminho de luz pertence;
- *setupPriority*: prioridade de instalação do caminho de luz;
- *holdingPriority*: prioridade de manutenção do caminho de luz;
- *protectionLevel*: nível de proteção do caminho de luz (inexistente, dedicado, compartilhado);
- *status*: estado atual do caminho de luz (“em uso” ou “falhou”);
- *route*: coleção de nós, fibras, interfaces e comprimentos de onda que definem uma conexão.

Os diagramas de classes para cada um dos serviços componentes se encontram a seguir.

[2]Serviço de Gerenciamento de Conexões é tradução do termo *Optical Connection Service* (OCS) e Serviço de Gerenciamento de Falhas é tradução do termo *Fault Manager Service* (FMS).

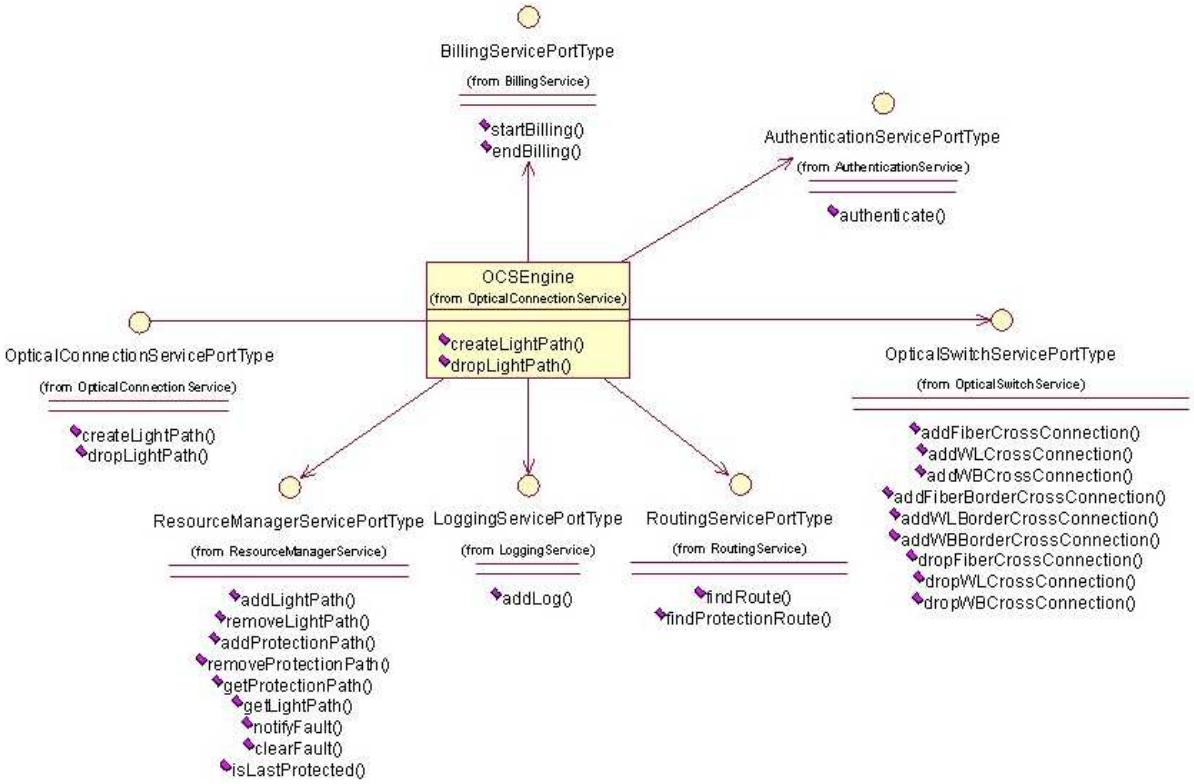

Fig. 4.1: Diagrama de Classes do Serviço de Gerenciamento de Conexões (SGC).

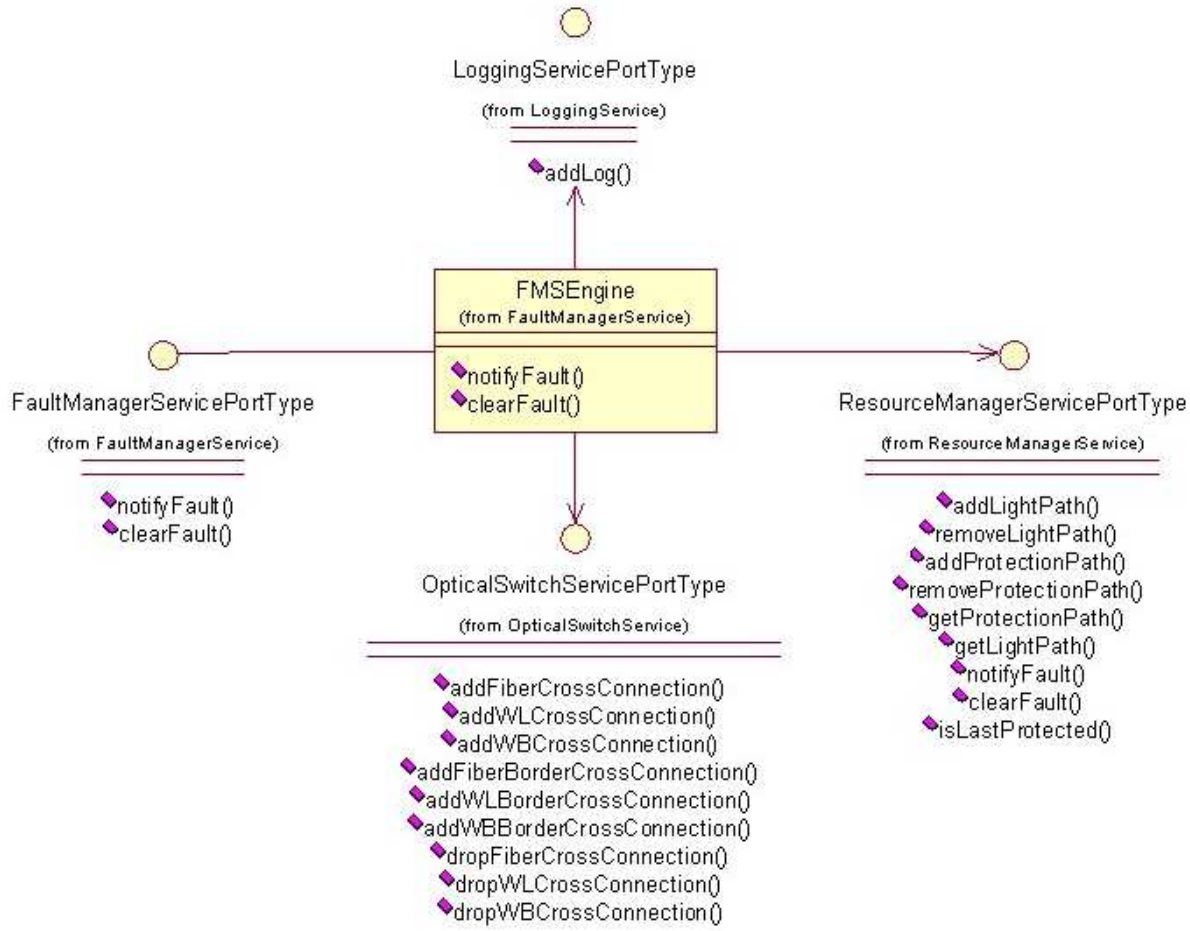

Fig. 4.2: Diagrama de Classes do Serviço de Gerenciamento de Falhas (SGF).

Serviço de Autenticação

O serviço possui somente um método para autenticação de usuários. Recebe como parâmetros o nome do usuário e senha e valida estes dados. Caso estes estejam presentes na base de dados este serviço retorna verdadeiro; caso contrário retorna falso. O diagrama de classes do serviço de autenticação pode ser visto na Figura 4.3.

Fig. 4.3: Diagrama de Classes do Serviço de Autenticação (SA).

Serviço de Tarifação

O serviço de tarifação possui dois métodos. O primeiro (*startBilling*) deve ser executado no momento em que um caminho de luz é instalado. Os parâmetros desse método incluem os atributos do caminho de luz principal e de proteção, o identificador do caminho de luz e o nome do usuário. O segundo (*endBilling*) deve ser executado no momento em que um caminho de luz é desinstalado. Recebe como parâmetro somente o identificador do caminho de luz. Ambos os métodos retornam verdadeiro caso a execução do serviço ocorra com sucesso e falso caso contrário. O diagrama de classes do serviço de tarifação pode ser visto na Figura 4.4.

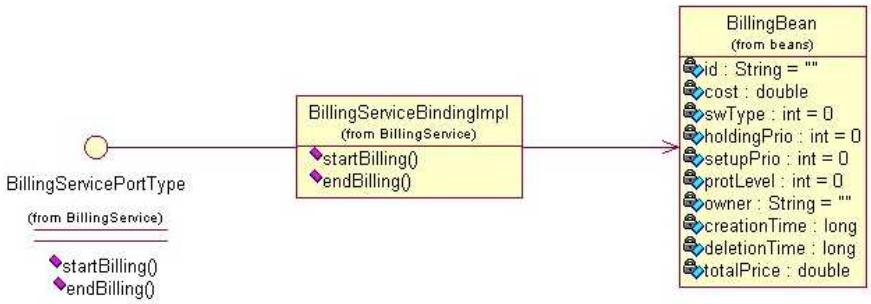

Fig. 4.4: Diagrama de Classes do Serviço de Tarifação (ST).

Serviço de *Logging*

Serviço responsável por armazenar dados de *log* persistentemente. Possui somente o método *addLog* que recebe como parâmetro uma seqüência de caracteres (*string*) com os dados a serem armazenados. Armazena também o momento em que o serviço foi executado, o que pode fornecer uma estimativa do momento de ocorrência do evento sendo armazenado. O diagrama de classes do serviço de *logging* pode ser visto na Figura 4.5.

Fig. 4.5: Diagrama de Classes do Serviço de *Logging* (SL).

Serviço de Conexão Cruzada

Serviço instalado dentro de cada comutador óptico no domínio. Provê facilidades de conexão cruzada no comutador. Os métodos expostos cobrem os três tipos de comutação possíveis - por comprimento de onda, por fibra e por faixa de comprimento de onda. Para instalação de uma comutação cruzada são recebidas as portas (fibras) de entrada e de saída, os comprimentos de onda (ou faixa de comprimentos de onda) de entrada e de saída (quando aplicável) e o identificador da conexão. Para remoção da conexão cruzada o único parâmetro recebido é o identificador da conexão.

Um método especial (com sufixo *BorderCrossConnection*) deve ser executado para instalação de uma conexão cruzada em um nó de borda. Isto porque na borda deve ser realizada a classificação dos pacotes que adentrarão o caminho de luz. Logo, este método possui um parâmetro adicional, o classificador de pacotes. Um classificador pode ser, por exemplo, o endereço IP da máquina destino. O diagrama de classes do serviço de conexão cruzada pode ser visto na Figura 4.6.

A classe *OpticalSwitchServiceBindingImpl* deve ser capaz de interagir com uma API de baixo nível ou diretamente com o hardware para implementação efetiva da conexão cruzada no comutador óptico.

Serviço de Gerenciamento de Recursos

Serviço responsável por manter um banco de dados com o estado da rede atualizado. Quando um caminho de luz primário ou de proteção for instalado este serviço deve ser notificado por meio

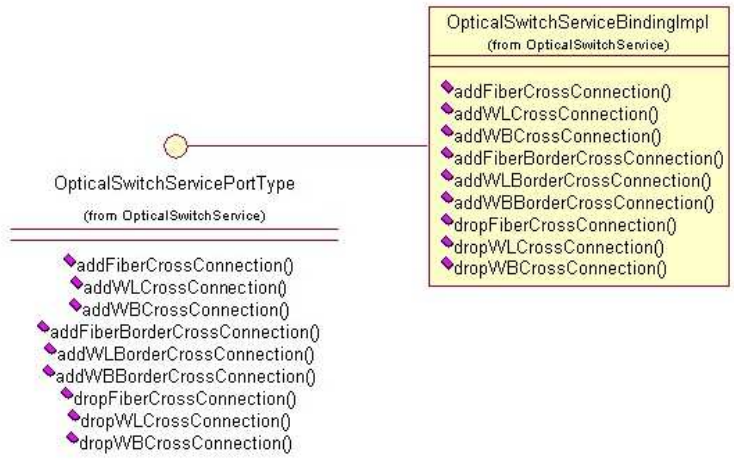

Fig. 4.6: Diagrama de Classes do Serviço de Conexão Cruzada (SCC).

dos métodos *addLightPath* e *addProtectionPath*, respectivamente. Esse métodos recebem como parâmetro os atributos do caminho de luz e o armazenam em uma base de dados (através da classe *LightpathBean*), criando um identificador único que é então retornado ao cliente do serviço. Os métodos para eliminação de caminho de luz primário e de proteção, respectivamente *removeLightPath* e *removeProtectionPath*, recebem como parâmetro esse identificador.

O gerenciador de recursos é capaz de retornar todos os atributos de um caminho de luz criado previamente através dos métodos *getLightPath* e *getProtectionPath*. Ambos os métodos recebem como entrada o identificador do caminho de luz *primário*. O primeiro retorna o caminho de luz com o dado identificador e o segundo retorna o caminho de luz que protege o caminho de luz com tal identificador.

Quando uma falha ocorre em algum enlace o gerenciador de recursos deve ser notificado por meio do método *notifyFault* para atualização do estado (*status*) do caminho de luz. A ação simétrica (*clearFault*) deve ser executada quando a falha não mais ocorrer. Ambos os métodos recebem como parâmetros os identificadores do caminho de luz principal e de proteção (quando existente).

Quando houver uma requisição de remoção de um caminho de luz principal com proteção compartilhada o método *isLastProtected* pode ser chamado. Este método verifica se o caminho de proteção correntemente só protege um dado caminho principal. Caso isso seja verdade, o caminho de proteção pode ser removido, dependendo da política adotada pelo provedor do domínio. No caso do sistema implementado foi realizada a opção pela remoção desses caminhos de proteção. O diagrama de classes do serviço gerenciador de recursos pode ser visto na Figura 4.7.

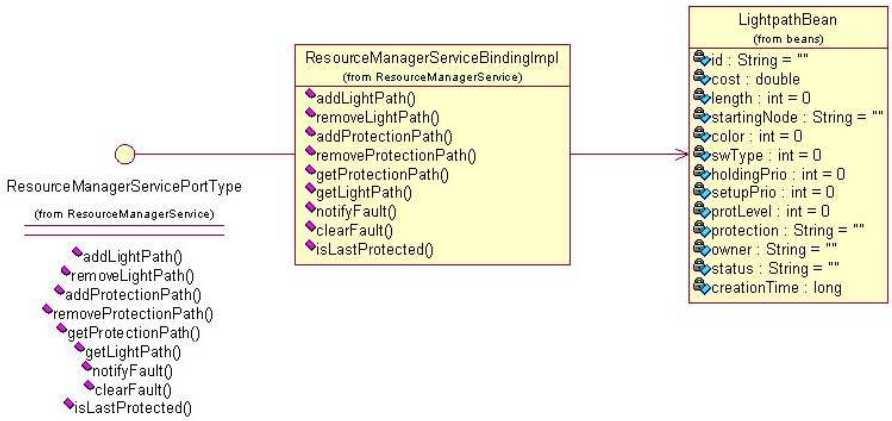

Fig. 4.7: Diagrama de Classes do Serviço de Gerenciamento de Recursos (SGR).

Serviço de Roteamento

O serviço de roteamento expõe dois métodos: um para busca de um caminho de luz principal (*findRoute*) e outro para busca de um caminho de luz de proteção (*findProtectionRoute*). Ambos recebem como parâmetros os atributos do caminho a ser procurado como nó de ingresso na rede, nó de egresso da rede, tipo de comutação, classe de serviço, prioridade de manutenção e de instalação. Adicionalmente, para busca do caminho de proteção o caminho principal deve ser passado como parâmetro, dado que ambos devem ser disjuntos.

Duas classes auxiliares são utilizadas por esse serviço (*ReadNet* e *OpticalNet*). A primeira é responsável por carregar a rede em memória (com seu estado atual). Essa classe retorna uma instância da classe *OpticalNet* que representa a rede em si, ou seja, contém a topologia e o estado atual. Note que a classe *OpticalNet* possui vários atributos (inseridos em vetores) que armazenam tais dados. Estes são utilizados para cálculo das rotas e comprimentos de onda a serem utilizados. O diagrama de classes do serviço de roteamento pode ser visto na Figura 4.8.

A classe SPF (abreviação de *Shortest Path First*) possui dois métodos (*shortestPath* e *shortestBackupPath*). Ambos recebem como parâmetros a classe *OpticalNet* e os atributos do caminho a ser encontrado, retornando a rota (classe *Route*) com menor custo (caso alguma exista). O método para cálculo da rota de proteção de menor custo recebe também como parâmetro a rota principal.

De posse da rota de menor custo (*Route*), a classe *RoutingServiceBindingImpl* invoca a classe *FirstFit* para alocação do(s) comprimento(s) de onda dentro dessa rota utilizando o algoritmo *First Fit*. Qualquer outro algoritmo poderia ser utilizados para tal cômputo. Nesse diagrama foi utilizado o algoritmo de alocação seqüencial fixa (*FirstFit*) pela sua simplicidade e bom desempenho. Na etapa de implementação outro algoritmo pode ser escolhido sem qualquer impacto ao restante do projeto.

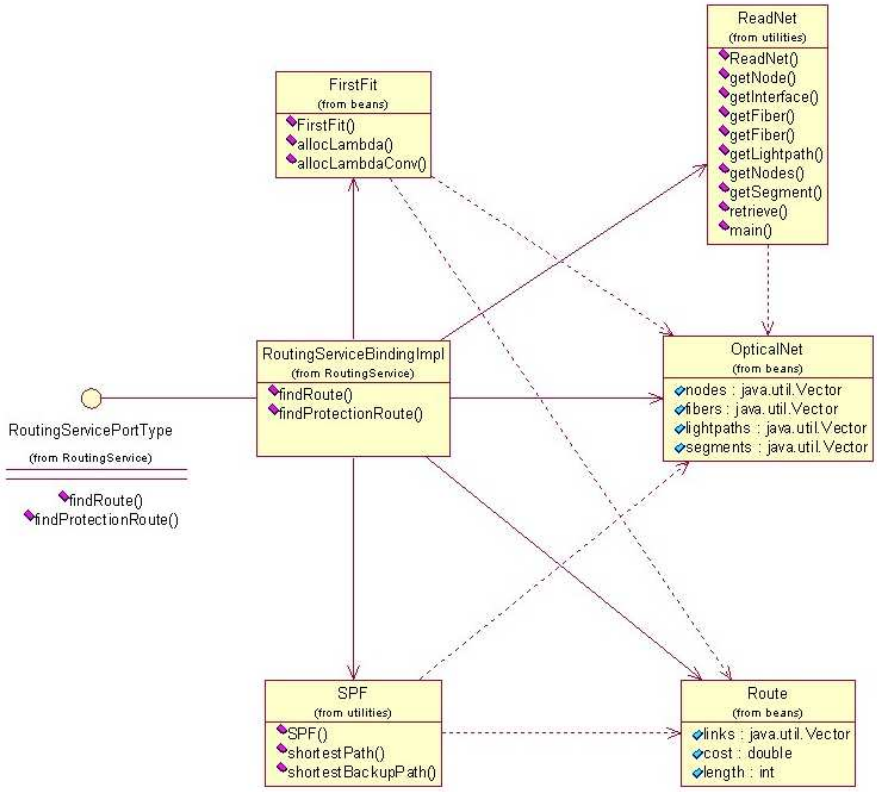

Fig. 4.8: Diagrama de Classes do Serviço de Roteamento (SR).

4.1.3 Diagramas de Seqüência

Foram especificados dois diagramas de seqüência para cada serviço orquestrado, cobrindo assim todos os possíveis casos de uso dos mesmos. Novamente, para facilidade de visualização todos os diagramas apresentam um alto grau de abstração, não detalhando os parâmetros de cada chamada. Os parâmetros utilizados são aqueles já mencionados na Subseção 4.1.2.

Para o serviço SGC foram desenvolvidos os diagramas de seqüência para criação e eliminação de um caminho de luz. Estes diagramas de seqüência podem ser vistos nas Figuras 4.9 e 4.10.

O primeiro passo para criação de um caminho de luz é a realização de uma chamada ao serviço de autenticação para verificação da validade dos dados do cliente (usuário e senha). Em seguida o serviço de roteamento é invocado para computação de um caminho primário e de proteção (caso tenha sido solicitado). Ambos levam em conta os parâmetros do caminho de luz solicitados pelo cliente. O gerenciador de recursos é então notificado sobre a instalação de um caminho de proteção. Este, por sua vez, instala os dados sobre o caminho de proteção na base de dados e retorna ao serviço orquestrador um identificador do caminho (LPPID). Neste momento, cada nó no caminho de proteção é contactado para adição da conexão cruzada referente ao caminho de proteção. Os nós devem preferencialmente

ser acessados concorrentemente, gerando um ganho de desempenho no tempo de estabelecimento de conexões. Isso é possível pois a maior parte das linguagens de orquestração possui suporte à execução concorrente.

Logo após, o gerenciador de recursos é notificado sobre a instalação de um caminho de luz primário. Além dos dados do caminho de luz primário é fornecido o identificador do caminho de luz que protege este, para que o gerenciador de recursos seja capaz de associá-los. Novamente os dados são armazenados na base de dados e um identificador da conexão (LPID) é retornado ao serviço orquestrador. Cada nó no caminho de luz é então contactado para adição da conexão cruzada referente ao caminho primário. Os nós devem preferencialmente ser acessados concorrentemente.

O último passo é o início da tarifação. Para isso o serviço de tarifação é notificado sobre a instalação dos caminhos de luz primário e de proteção, recebendo como parâmetros os detalhes das conexões e o nome do cliente a ser tarifado. O identificador da conexão é finalmente retornado ao cliente. Este identificador deve ser guardado para posterior remoção do caminho de luz.

Para eliminação de um caminho de luz é necessária a autenticação dos dados do cliente inicialmente. Em seguida os dados sobre o caminho de luz primário são obtidos do gerenciador de recursos. Para eliminação de caminhos de luz de proteção compartilhados que ficariam instalados sem um caminho de luz principal o método *isLastProtected* do gerenciador de recursos é chamado. Caso o caminho de luz de proteção só esteja protegendo o caminho luz sendo eliminado ele também será desinstalado. Os dados do caminho de proteção são então obtidos do gerenciador de recursos, e o mesmo é notificado da remoção do caminho de luz principal. Cada nó é então contactado para eliminação da conexão cruzada relativa à conexão primária. Em seguida o gerenciador de recursos é notificado sobre a remoção do caminho de proteção[3]. Cada nó é novamente contactado para eliminação da conexão cruzada relativa à conexão de proteção. Em ambos os casos, os nós devem ser preferencialmente acessados concorrentemente. O serviço de tarifação é notificado sobre a desinstalação do caminho de luz e o cliente recebe uma mensagem de sucesso.

Para o serviço SGF foram desenvolvidos os diagramas de seqüência para restauração e eliminação de falhas. Estes diagramas de seqüência podem ser vistos nas Figuras 4.11 e 4.12.

Para notificação e restauração de falhas ocorridas o identificador do caminho de luz que falhou deve ser fornecido. Inicialmente são obtidos os caminhos de luz principal e de proteção junto ao gerenciador de recursos. De posse de tais dados, o gerenciador de falhas contacta o nó de borda do caminho para eliminação da conexão cruzada relativa ao caminho de luz principal. Em seguida, a conexão cruzada referente ao caminho de proteção é instalada nesse nó de borda. O gerenciador de recursos é notificado sobre a falha, para atualização do estado dos caminhos de luz e uma mensagem

[3] Aqui supõe-se a presença de um caminho de proteção dedicado ou compartilhado na situação em que o caminho de proteção correntemente só protege o caminho de luz sendo removido.

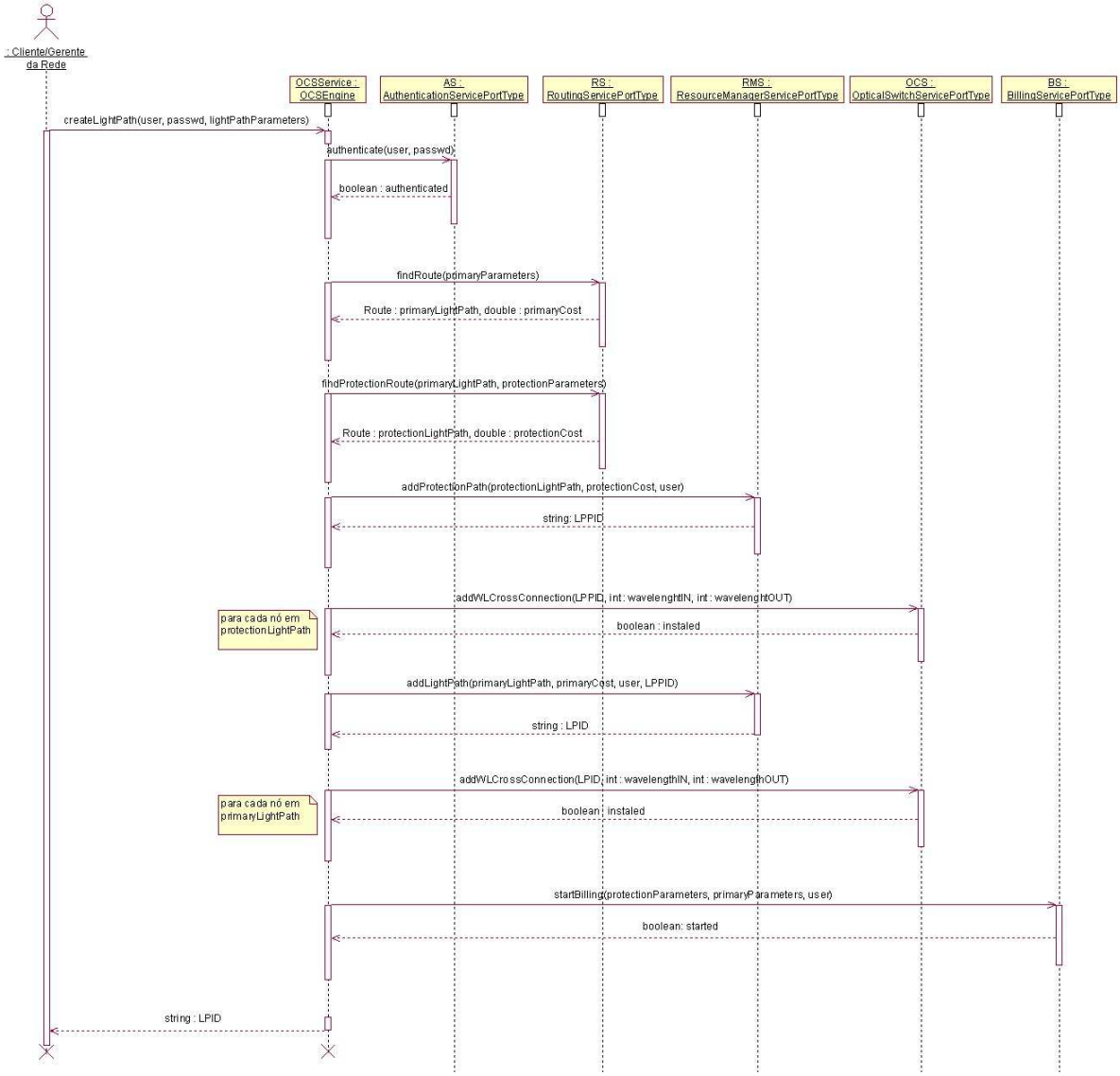

Fig. 4.9: Diagrama de Seqüência de Criação de um Caminho de Luz.

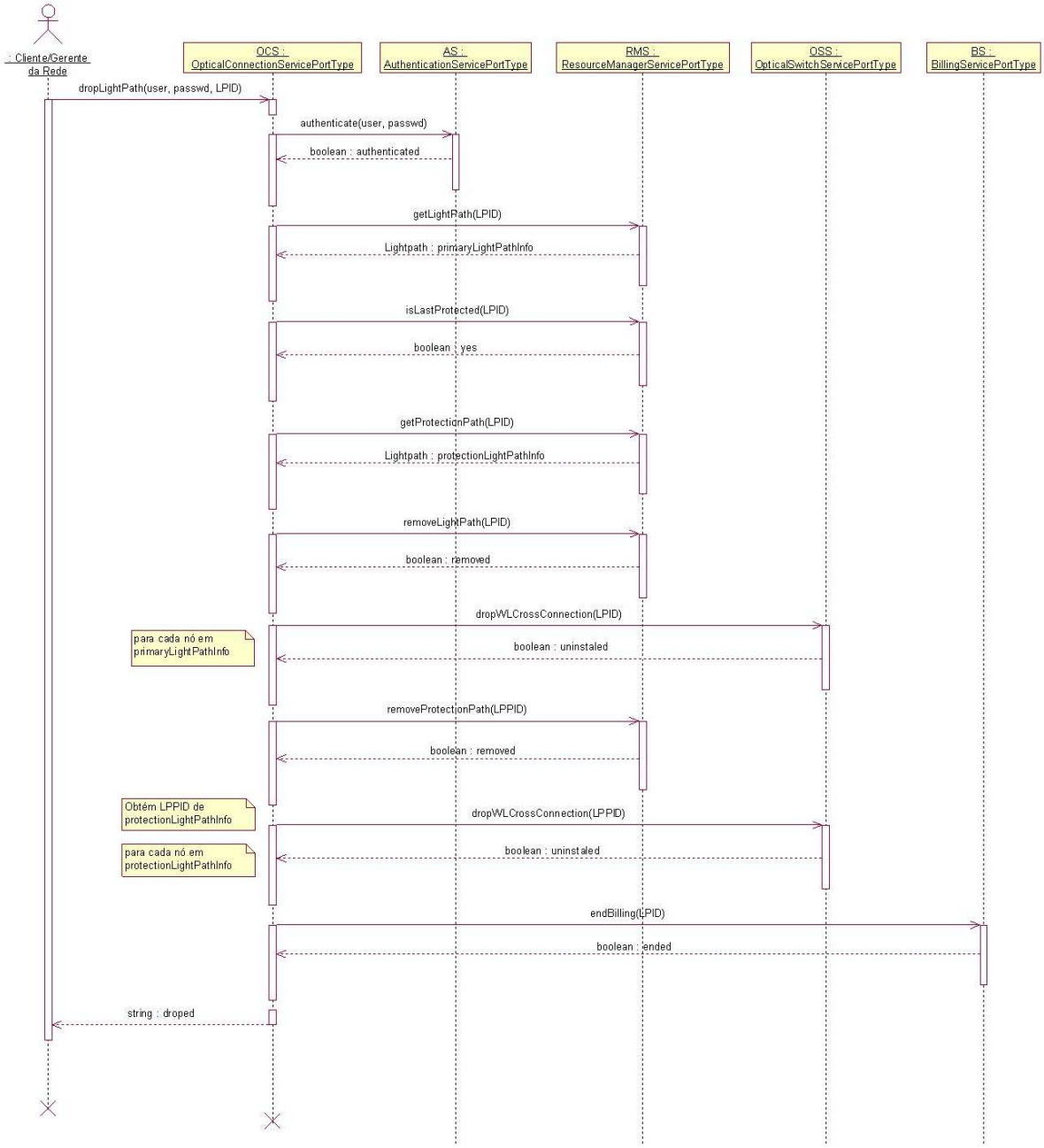

Fig. 4.10: Diagrama de Seqüência de Eliminação de um Caminho de Luz.

de sucesso é retornada ao cliente.

O processo de eliminação de falhas é análogo. Inicialmente são obtidos os caminhos de luz principal e de proteção junto ao gerenciador de recursos. O gerenciador de falhas contacta o nó de borda do caminho para eliminação da conexão cruzada relativa ao caminho de proteção e adição da conexão cruzada referente ao caminho principal. O gerenciador de recursos é notificado sobre a eliminação da falha e uma mensagem de sucesso é retornada ao cliente.

4.2 Implementação

Os serviços componentes SCC, SGR, SR, SA, ST e SL foram completamente implementados a partir do projeto especificado na Seção 4.1. No ambiente de um provedor de serviços, a maior parte dos serviços componentes já existem, embora não sejam implementados como serviços *web*. Para expor um serviço existente como um serviço *web* o desenvolvedor pode utilizar muitas ferramentas de software existentes. Por exemplo, algumas plataformas EJB geram interfaces de serviços *web* para componentes EJB existentes. No caso, nenhum dos serviços possuía qualquer implementação, portanto tiveram que ser implementados integralmente.

Foi utilizada a linguagem BPEL para especificação dos *scripts* de orquestração, pelo fato de ser a linguagem mais madura e difundida atualmente. Além disso, conforme mencionado na Subseção 2.2.4, a linguagem BPEL é a que mais provê suporte nativo a padrões de instruções de linguagens de fluxograma.

Inicialmente foi utilizada a plataforma BPWS4J da IBM para desenvolvimento (utilizando o *plug-in* para Eclipse) e instalação dos serviços (máquina de orquestração instalada no Tomcat). Devido à instabilidade do *plug-in* e não completude da máquina de orquestração frente à especificação da linguagem esta plataforma foi abandonada. Foi utilizada então a máquina de orquestração *ActiveBPEL*, pelo fato de ser livre (código aberto) e ser totalmente compatível com a versão 1.1 da linguagem BPEL.

Usando o mecanismo de compensação da linguagem BPEL foi possível realizar o *rollback* no estabelecimento de conexões. Quando um comutador óptico, pertencente ao caminho de uma conexão, por algum motivo não puder realizar a comutação requerida, os comutadores que já a realizaram devem desfazê-la. O *rollback* é realizado pela máquina de orquestração, o programador somente precisa especificar qual ação será realizada em caso de falha pois tal ação é dependente de implementação. Por exemplo, ao especificar a chamada ao método de instalação de uma conexão cruzada em um comutador óptico (*addWLCrossConnection*) o programador especifica qual a ação simétrica a essa, *i.e.*, qual ação deve ser tomada caso alguma falha ocorra no escopo corrente[4]. Por exemplo, o método

[4]Para mais detalhes sobre a relação entre escopos e *rollback* ver Subseção 2.3.4.

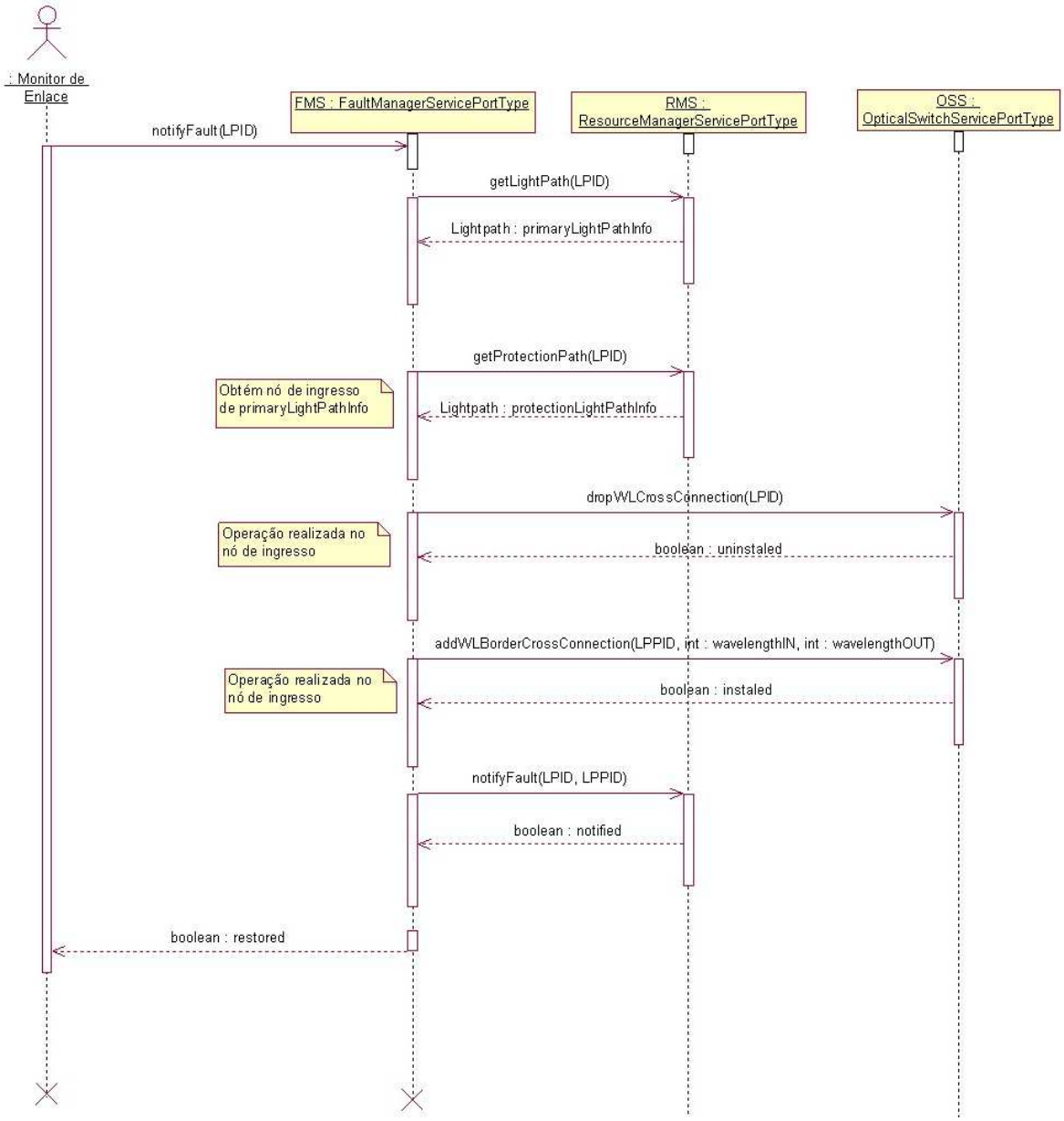

Fig. 4.11: Diagrama de Seqüência de Restauração de Falha.

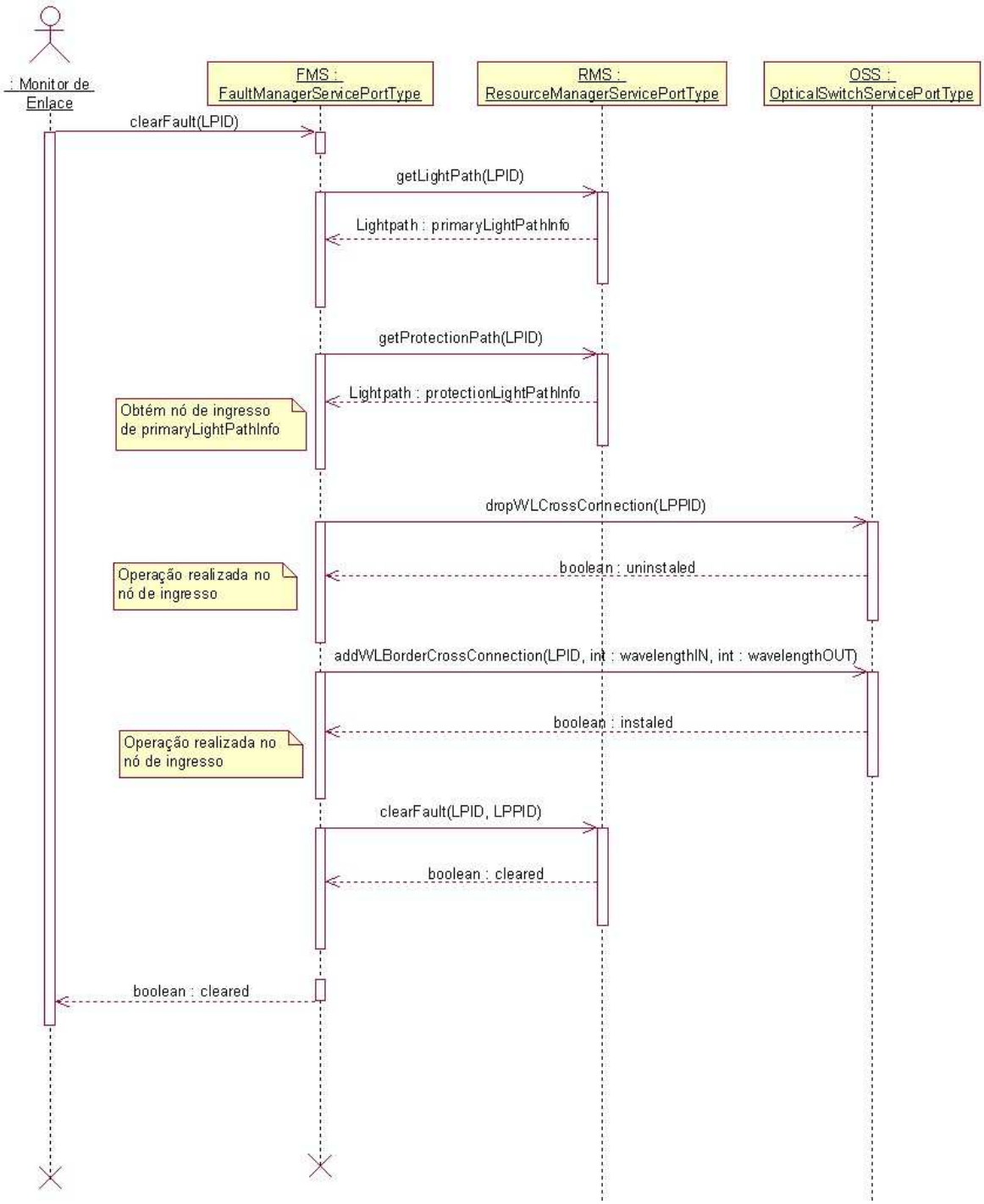

Fig. 4.12: Diagrama de Seqüência de Eliminação de Falha.

dropWLCrossConnection do mesmo serviço pode ser chamado. Situação análoga ocorre com o gerenciador de recursos. Quando um comutador de um dado caminho de luz falhar durante a instalação da conexão cruzada, os detalhes da conexão previamente inseridos no gerenciador de recursos devem ser removidos.

Todas as informações de topologia da rede óptica são armazenadas em um banco de dados. Foi implementada uma interface *web* onde o administrador da rede pode inserir informações sobre a rede óptica local, como nós, fibras, comprimentos de onda por fibras, custo por fibra, capacidade de comutação dos nós e topologia da rede. Esta interface pode ser vista na Figura 4.13.

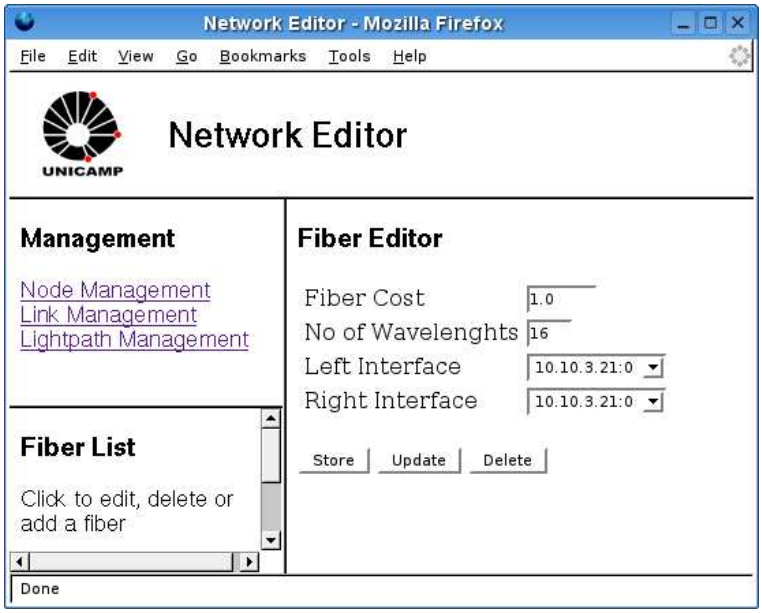

Fig. 4.13: Interface *Web* para Instalação da Rede.

Todos os dados transmitidos entre o cliente e os serviços SGC e SGF são enviados através de um canal seguro, utilizando *Secure Socket Layer* (SSL). Isto foi feito para proteger os dados de autenticação do cliente e prover privacidade às requisições realizadas.

A comunicação com serviços *web* foi realizada utilizando o protocolo SOAP. Foi utilizado o estilo documento literal *wrapped* para todas as associações (*bindings*) SOAP pois o uso do estilo codificado (*Section 5 encoding*) tem sido não recomendado por questões de interoperabilidade. Mensagens utilizando o estilo documento literal podem ser totalmente validadas, pois o campo *body* não contém qualquer tipo de informação codificada e a mensagem deve ser compatível com um *XML Schema* acordado [47, 48]. O nome do método sendo chamado está presente na mensagem SOAP (propriedade proveniente do estilo *wrapped*), o que simplifica o despacho da mensagem no lado servidor para o serviço devido. Além disso, o estilo documento apresenta escalabilidade superior quando comparado ao estilo RPC (*Remote Procedure Call*). Quando o tamanho da mensagem cresce o estilo documento

sofre uma degradação muito menor quanto ao número de requisições servidas por segundo [49]. Uma indicação de que a indústria está abandonando o estilo codificado pode ser vista pela sua omissão na especificação *WS-I Basic Profile* [50] que provê um guia para implementação de serviços *web* interoperáveis.

4.2.1 Serviço de Conexão Cruzada

Esse serviço é instalado dentro de cada comutador óptico do domínio e provê funcionalidades de conexão cruzada no equipamento. O comutador óptico foi emulado utilizando roteadores com kernel Linux MPLS-capaz (*Multiprotocol Label Switching*) [51]. As entradas na tabela de rótulos MPLS emulam uma conexão óptica com uma equivalência de um para um, ou seja, um comprimento de onda equivale a um rótulo. A Figura 4.14 ilustra o comutador óptico emulado.

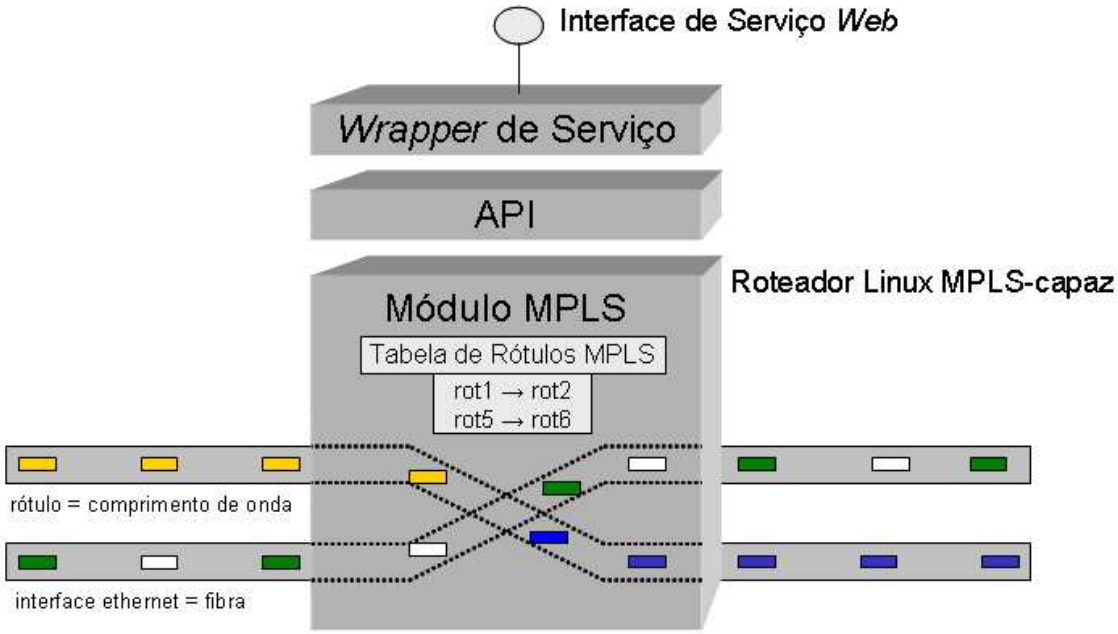

Fig. 4.14: Comutador Óptico Emulado.

Neste cenário emulado o SCC deve alterar a tabela de rótulos MPLS no kernel Linux via chamadas de sistema. Dessa forma, o serviço deve ser capaz de realizar ações de baixo nível no kernel. Num cenário real, este serviço deve executar dentro de um comutador óptico, devendo ser eficiente e capaz de trabalhar com recursos limitados de memória e processamento. Devido a estes fatos, foi utilizada a linguagem C++ para implementação desse serviço, dado que esta é uma linguagem orientada a objetos eficiente que permite interações de baixo nível. O servidor HTTP *Apache* e a máquina SOAP *Axis C++* foram utilizados para o desenvolvimento desse serviço.

As funcionalidades do protocolo MPLS foram implementadas como um módulo Linux usando a linguagem C. A implementação de tal módulo não foi parte deste trabalho. Ao iniciar cada máquina que emula um comutador óptico, deve-se adicionar este módulo ao kernel através do comando *insmod*. Para acesso a esse módulo foi desenvolvida uma API em C++. Tal API realiza chamadas de sistema *ioctl*, um tipo de *socket* de sistema, para comunicação com o módulo MPLS. Sobre essa camada vem a de serviço *web*, também implementada em C++, pelos motivos já mencionados.

4.2.2 Serviço de Gerenciamento de Recursos

O serviço SGR tem por objetivo manter um banco de dados dos recursos efetivamente disponíveis na rede. Ou seja, ele deve ser consultado sobre a disponibilidade de recursos e deve também ser notificado quando qualquer recurso da rede for utilizado.

Quando uma falha ocorre o SGR deve ser notificado para que o estado do caminho de luz seja alterado (de "em uso" para "falhou"). Todos os dados referentes à proteção de caminhos de luz são armazenados pelo SGR. Ele fornece ao gerenciador de falhas qual o caminho de proteção associado a um dado caminho principal. Quando a falha desaparecer o serviço deve ser notificado para que o estado do caminho de luz possa ser restaurado e uma consulta deve ser realizada para que os dados sobre o caminho principal sejam obtidos.

Quando um caminho de luz é desinstalado, seja de proteção ou principal, o SGR também deve ser notificado.

4.2.3 Serviço de Roteamento

Um algoritmo de roteamento e alocação de comprimentos de onda em redes ópticas (*Routing and Wavelength Assignment* - RWA) pode ser classificado em termos da forma de controle da rede e do modo de estabelecimento da conexão. Quanto à forma de controle ele pode ser distribuído ou centralizado. No controle centralizado existe um agente central responsável pelo gerenciamento da rede, manutenção do estado atual de uso, topologia dos nós, enlaces disponíveis, entre outros. No controle distribuído não existe este elemento central e o estado global da rede só é disponibilizado aos nós da rede com algum tempo de atraso. Já quanto à forma de estabelecimento da conexão, os algoritmos podem ser classificados em [52]:

- estáticos: o conjunto de solicitações de caminhos ópticos está disponível *a priori* e o problema consiste em estabelecer os caminhos ópticos de forma a minimizar os recursos alocados na rede;

- semi-permanentes: os caminhos ópticos são estabelecidos seqüencialmente e permanecem indefinidamente ou por um longo período de tempo na rede;

- dinâmicos: os caminhos ópticos são estabelecidos quando a chamada chega e liberados após um período finito de tempo.

O serviço de roteamento implementado utiliza a técnica de controle centralizado. É utilizado o banco de dados atualizado provido pelo gerenciador de recursos para busca de um caminho de luz dentro de um domínio. A rede, juntamente com seu estado atual, é carregada do banco de dados em memória para processamento. Quanto à forma de estabelecimento de conexões assumiu-se o uso do modo semi-permanente, dado que os clientes vão solicitar dinamicamente a criação e eliminação dos caminhos de luz.

O processo para encontrar um caminho de luz disponível dentro de uma rede óptica se dá em dois passos. O primeiro é o roteamento, cujo objetivo é encontrar uma rota (seqüência de nós conectados fisicamente). O passo seguinte é a alocação de comprimentos de onda dentro desse rota. Para ambas etapas várias técnicas são possíveis.

O roteamento pode ser fixo, fixo alternativo ou adaptativo. No fixo, sempre a mesma rota é escolhida dada uma requisição. São utilizados algoritmos de caminho mais curto (*e.g.*, Dijkstra ou Bellman-Ford) ou tabelas programadas em cada nó da rede ou em uma base de dados global. No roteamento fixo alternativo, deve-se manter uma tabela de roteamento contendo uma rota direta (preferencial) e um conjunto de rotas alternativas para cada par de nós de origem e destino. Caso a rota direta não possua comprimentos de onda disponíveis alguma rota alternativa deve ser escolhida baseado em algum critério, por exemplo, aleatoriamente. Já no roteamento adaptativo, a rota é escolhida de forma dinâmica em função do estado da rede, levando em conta por exemplo, o número de elementos no caminho (número de *hops*), o custo ou a carga dos enlaces.

Para escolha dos comprimentos de onda existem várias técnicas disponíveis: Alocação Aleatória (*Random*), Alocação Seqüencial Fixa (*First Fit*), Menos Usado (*Least Used*), Mais Usado (*Most Used*), Menos Ocupado (*Least Loaded*), MAX-SUM, Reserva de Comprimento de Onda (*Wavelength Reservation*), Proteção de Limite (*Threshold Protection*), entre outras. O detalhamento do funcionamento de cada técnica pode ser encontrado em [52, 53].

Para implementação desse serviço foi utilizado um algoritmo de roteamento que leva em conta o custo de um enlace para escolha do caminho mais apropriado. Inicialmente este serviço encontra o caminho mais barato entre os nós de ingresso e egresso na rede, fazendo uso de um algoritmo tipo *Shortest Path First* (SPF). Posteriormente é efetuada uma busca por comprimentos de onda disponíveis nessa rota. O algoritmo de alocação de comprimento de onda escolhido foi o de Alocação Seqüencial Fixa, ou seja, o primeiro comprimento de onda disponível na rota é utilizado. A combina-

ção da rota e do comprimento de onda constituem o caminho de luz que é então retornado ao cliente do serviço.

4.2.4 Serviço de *Logging*

Serviço responsável pelas atividades de *logging* do sistema. É capaz de armazenar qualquer dado que lhe for solicitado, no formato de uma seqüência de caracteres. Juntamente com o dado é armazenada a data e o horário da ocorrência do evento (o momento em que o serviço foi chamado). Os dados são armazenados em um arquivo texto para posterior processamento. No contexto do serviço SGC e SGF informações sobre exceções e falhas estão sendo armazenadas para fins estatísticos.

4.2.5 Serviço de Autenticação

Serviço responsável por autenticar os dados de um cliente. Os dados são constituídos de um nome de usuário e uma senha. Os dados obtidos do cliente no momento de subscrição ao serviço são armazenados em um banco de dados, não criptografados. Dado que o banco de dados não está aberto para acesso remoto e o acesso local é protegido por senha, o fato de os dados estarem abertos não pode ser considerado uma falha de segurança severa.

4.2.6 Serviço de Tarifação

Este serviço é responsável por tarifar o cliente pelos recursos efetivamente utilizados da rede. Quando algum recurso da rede é alocado a um cliente este serviço deve ser notificado. Devem ser informados quais os recursos que estão sendo utilizados, com todas as características dos mesmos. Por exemplo, quando um caminho de luz é instalado na rede, o serviço deve ser acionado. Devem ser passados parâmetros referentes ao caminho de luz instalado, como nível de proteção, tipo de comutação, prioridade de instalação, prioridade de manutenção e custo do caminho[5]. Obviamente, o nome do cliente que requisitou os recursos deve também ser fornecido.

Cada provedor de serviço tem o seu modelo de tarifação e portanto, a lógica de implementação de tal serviço é diferente para cada provedor. Foi criado um modelo de tarifação para implementação desse serviço. Basicamente são utilizados dois valores no cômputo dos preços associados a um recurso utilizado: o preço de instalação (um valor fixo para cada tipo de recurso utilizado) e o preço de manutenção (dependente do período de tempo durante o qual o recurso de rede foi utilizado). O serviço não trabalha com uma moeda especifica, ao invés disso, foi utilizada uma moeda fictícia chamada de Unidade Monetária (UM). Uma Unidade Monetária pode ser convertida para qualquer

[5]O custo do caminho é uma informação proveniente do serviço de cômputo de rotas. Pode estar associado, por exemplo, ao número de comutadores atravessados pelo caminho de luz.

moeda, com uma taxa (câmbio) fornecida pelo provedor do serviço. Dessa forma, o preço do serviço pode ser reajustado sem a necessidade de alteração do serviço em si, sendo preciso somente realizar a alteração do câmbio.

Foi decidido que a tarifação sobre a requisição de um caminho de luz de proteção teria um valor associado à reserva do caminho de luz por unidade de tempo (dependente do tipo de proteção utilizada), multiplicado pelo tempo de alocação total do caminho de proteção. Ou seja, independente do fato do caminho de proteção ser utilizado o cliente será tarifado por ter o recurso alocado para si. É claro que a tarifação sobre um caminho de proteção compartilhado é bem menor do que um dedicado. Sendo:

pc_{prot} = preço da reserva do caminho de luz por unidade de tempo, dependente do tipo de proteção (UM/hora);

t_{prot} = período de tempo em que o caminho de proteção permaneceu reservado (horas);

Temos que:

$$p_{prot} = pc_{prot} * t_{prot} \tag{4.1}$$

Onde p_{prot} é o valor total de tarifação referente ao caminho de proteção.

O valor cobrado pela instalação do caminho principal é calculado a partir de um valor constante para cada tipo de comutação utilizada (*switchingType*). Na realidade os tipos de comutação por fibra e por faixa de comprimentos de onda são tarifados como *n* comprimentos de onda, onde *n* é o número de comprimentos de onda da fibra e da faixa alocados, respectivamente. Esse valor constante é multiplicado pela prioridade de instalação adicionada do custo (retornado pelo serviço de roteamento). Ou seja, caso o cliente solicite uma prioridade de instalação alta (capaz de preemptar outros caminhos de luz com prioridade de manutenção menor que tal prioridade de instalação) ele será tarifado por isso. Além disso, caso a rede esteja sobrecarregada e o caminho utilizado seja longo, o custo retornado pelo algoritmo de roteamento será alto e, por conseguinte, a componente do preço referente à instalação do caminho também o será. Matematicamente, sendo:

pc_{prim} = preço constante dependente do tipo de comutação (UM);

pi = prioridade de instalação do caminho principal (adimensional);

c = custo retornado pelo algoritmo de roteamento (adimensional);

Temos que:

$$pinst_{prim} = (pi + c) * pc_{prim} \tag{4.2}$$

Onde $pinst_{prim}$ é a componente de tarifação referente à instalação do caminho de luz primário.

Já o valor cobrado pela manutenção do caminho principal é calculado a partir do preço de utilização de um caminho de luz por hora (UM/hora). Esse valor depende também do tipo de comutação utilizada. É levada em conta também a prioridade de manutenção (*holdingPriority*) do caminho de luz principal. Finalmente, o tempo de utilização do caminho principal é utilizado. Sendo:

tx_{prim} = preço de utilização por unidade de tempo, dependente do tipo de comutação (UM/hora);
pm = prioridade de manutenção do caminho principal (adimensional);
t_{prim} = tempo total em que o caminho primário permaneceu alocado (horas);

Temos que:

$$pman_{prim} = tx_{prim} * pm * t_{prim} \tag{4.3}$$

Onde $pman_{prim}$ é a componente de tarifação referente à manutenção do caminho de luz primário. O preço total (p_{tot}) de tarifação é então calculado da seguinte forma:

$$p_{tot} = p_{prot} + pinst_{prim} + pman_{prim} \tag{4.4}$$

Apesar do sistema de tarifação aqui proposto ser simplista acredita-se que ele seja suficiente para avaliação do sistema como um todo, dado que o foco deste trabalho é a composição dos serviços.

4.3 Testes

Com objetivo de testar os sistemas desenvolvidos vários programas clientes de linha de comando foram implementados para interagir com os serviços orquestrados. A linguagem utilizada para codificação do cliente não é relevante, dado que a mensagem SOAP trocada com o serviço é compatível com o *XSD Schema* especificado. No caso foi utilizada a linguagem Java e a API Axis Java como máquina SOAP. Os seguintes testes foram realizados para avaliação dos serviços orquestrados (SGC e SGF):

- Criação de caminho de luz;
- Eliminação de caminho de luz;
- Restauração de falha;
- Eliminação de falha.

Para avaliação do impacto no desempenho proveniente do uso de uma arquitetura SOA para desenvolvimento de serviços de rede foi dado foco especial aos tempos de resposta. Como tempo de resposta foi considerado a diferença entre o instante em que a mensagem foi enviada pelo cliente e o instante em que a mensagem de resposta foi entregue ao cliente. Nestes testes não foi realizada emulação do atraso para instalação de uma conexão cruzada nos elementos ópticos e, como as conexões são instaladas concorrentemente, não existe uma diferença significante no tempo de resposta em função do aumento do número de nós no caminho de luz. Os testes foram conduzidos em um ambiente controlado (uma rede local) com condições ideais (*e.g.*, baixo tráfego). Por se tratar de uma rede local não existe atraso significativo na comunicação entre o cliente da rede e o servidor de orquestração. Os valores obtidos devem portanto ser considerados como otimistas. Os resultados de 100 medições realizadas são mostrados na Tabela 4.1.

Teste	**Tempo de Resposta (*ms*)**	**Desvio Padrão (*ms*)**
Criação de caminho de luz	122	13
Eliminação de caminho de luz	101	6
Restauração de falha	125	5
Eliminação de falha	136	8

Tab. 4.1: Tempos de Resposta.

Como conexões ópticas possuem geralmente longa duração o tempo de resposta para a criação de caminhos ópticos é aceitável. É importante lembrar que este inclui o tempo de todo processamento necessário ao estabelecimento da conexão, mesmo aquele relacionado à camada de negócios (*e.g.*, tarifação). Dependendo do tipo de aplicação, notoriamente conexões telefônicas, o tempo de restauração de falhas pode ser altamente restritivo. O tempo de resposta do SGF para este tipo de aplicações pode ser considerado inadequado se comparado aos tempos de restauração obtidos em camada 2 (como nas redes SONET). O desempenho obtido pelo SGF é adequado a aplicações com requisitos mais tolerantes quanto ao tempo de restauração, como navegação na internet ou vídeo sob-demanda.

Os tempos de eliminação de caminho de luz e eliminação de falhas não apresentam grande impacto no desempenho do sistema como um todo, exceto no caso em a rede se encontra sobrecarregada (todos os possíveis caminhos de luz estão instalados) e existe a necessidade de instalação de novas conexões. Mesmo nestes casos, os tempos obtidos foram muito satisfatórios.

O objetivo da arquitetura proposta é possibilitar o desenvolvimento rápido e flexível de novos serviços de rede. Com a ajuda de ferramentas especializadas tal objetivo foi atingido. As soluções são simples e robustas, e a implementação valida completamente a arquitetura proposta.

4.4 Considerações Finais

Neste capítulo foram descritos o projeto e a implementação de dois serviços de rede para validação da arquitetura SOANet. Os detalhes de implementação mais relevantes foram mencionados, bem como razões para escolha das plataformas, linguagens e ferramentas. Os testes realizados mostraram tempos de resposta satisfatórios, mostrando que o impacto da utilização de uma arquitetura orientada a serviço neste domínio é pequeno.

Capítulo 5

Conclusões

Este capítulo apresenta alguns possíveis trabalhos futuros, bem como apresenta as considerações finais relativas a avaliação do trabalho como um todo.

5.1 Trabalhos Futuros

Apesar de ter cumprido os objetivos propostos este trabalho é passível de várias adições que o tornariam mais interessante. Entre estes estão:

- Interface gráfica para cliente final: uma interface gráfica para que o cliente pudesse solicitar recursos ao provedor de rede seria útil. Tal interface faz sentido somente ao cliente final (usuário) da rede. Um provedor de rede ou de serviços implementando serviços fim-a-fim vai interagir diretamente com o serviço intra-domínio, isto é, sem fazer uso de uma interface gráfica;

- Implementação de um serviço fim-a-fim: a implementação, teste e avaliação da arquitetura SOANet na provisão de um serviço fim-a-fim envolvendo vários domínios é necessária. Acredita-se que as mesmas vantagens obtidas no desenvolvimento do serviço intra-domínio seriam obtidas, entretanto, uma avaliação prática seria adequada para validar tal crença.

- Adições de políticas no serviço de conexões ópticas: a política de admissão atualmente implementada é simplista. Políticas de admissão mais elaboradas, levando em conta por exemplo classes de serviços, são mais compatíveis com um cenário real de um provedor de serviços.

- Avaliação do consumo de banda: um ponto de discussão quanto ao gerenciamento via serviços *web* é quanto ao consumo de banda, dado que as mensagens SOAP podem ser grandes. Uma análise profunda do consumo de banda proveniente do uso da arquitetura orientada a serviços na provisão de serviços de rede seria de grande valia.

- Algoritmo de roteamento: utilização de um algoritmo de roteamento (RWA) mais eficiente [52].
- Replicação do serviço de gerenciamento de conexões: conforme mencionado na Seção 3.6, a replicação do serviço de gerenciamento de conexões dentro do domínio de rede aumenta a escalabilidade, o desempenho e a confiabilidade do sistema como um todo.
- Distribuição do serviço de gerenciamento de falhas: tendo em vista a redução do tempo de resposta na restauração de falhas, é possível que o próprio nó que detectou a falha realize a restauração. Desta forma, ao invés de notificar um serviço orquestrador central, cada nó possuiria um serviço composto capaz de contactar o nó de ingresso do caminho de luz em falha solicitando a comutação para o caminho de luz de proteção.
- Registro e busca de serviços: a adição de uma agência de registro de serviços possibilita a relocação do serviço sem a necessidade de notificação explícita ao cliente por parte do servidor.

5.2 Considerações Finais

Este trabalho propôs uma arquitetura orientada a serviços (SOANet) para o desenvolvimento, instalação e gerenciamento de serviços em redes orientadas a conexão. Esta arquitetura proporciona ao provedor de serviço de rede um alto grau de automação, integração e flexibilidade no projeto e desenvolvimento de novos serviços de rede. Utilizando a composição de serviços o tempo de desenvolvimento pode ser reduzido drasticamente, resultando também em redução de custos, fator extremamente importante no desenvolvimento de software atualmente.

Certamente, o ponto mais inovador proposto neste trabalho é a adição de interfaces de serviço *web* nos equipamentos de rede. A arquitetura SOANet é horizontal, na medida em que todos os serviços dentro de um domínio se relacionam através da composição em um mesmo nível (sem camadas de abstração). Em função do uso de protocolos padronizados para composição dos serviços tem-se a vantagem adicional da eliminação de gargalos de interoperabilidade diante de uma rede composta por equipamentos de diferentes fabricantes. Ao invés de implementar protocolos de sinalização e gerência complexos e às vezes fracamente padronizados, fabricantes de equipamentos de rede podem implementar interfaces de serviço *web* para controle e gerenciamento de seus equipamentos. Ao publicar esta interface, o controle e gerência desses equipamentos é realizado incorporando estas interfaces em *scripts* de composição. Diferentemente dos protocolos de rede, as interfaces de serviço *web* não precisam ser totalmente padronizadas, precisam somente expor funcionalidades similares.

As características inerentes à arquitetura orientada a serviço, notoriamente alta granularidade, alto índice de reuso de código, interoperabilidade, fraco acoplamento e ubiqüidade puderam ser verificadas através do desenvolvimento dos serviços SGC e SGF. A flexibilidade e o rápido desenvolvimento

de novos serviços deve ser enfatizado, dado que são altamente valorizados em qualquer processo de desenvolvimento de software.

Referências Bibliográfi cas

[1] User-to-Network Interface. `http://www.oiforum.com/`.

[2] Victor A. S. M. de Souza and Eleri Cardozo. A Service Oriented Architecture for Deploying and Managing Network Services. In F. Casati e P. Traverso B. Benatallah, editor, *Proceedings of the 3rd International Conference on Service Oriented Computing*, pages 465–477. Springer-Verlag, Dezembro 2005.

[3] Bill St. Arnaud, Andrew Bjerring, Omar Cherkaoui, Raouf Boutaba, Martin Potts, and Wade Hong. Web Services Architecture for User Control and Management of Optical Internet Networks. *Proceedings of the IEEE*, 92(9):1490–1500, Setembro 2004.

[4] Canarie Inc. `http://www.canarie.ca`.

[5] Bill St. Arnaud. Web Services Workflow for Connecting Research Instruments and Sensors to Networks. `http://www.canarie.ca/canet4/uclp/UCLP_Roadmap.doc`, Dezembro 2004.

[6] Nobuhide Nishiyama, Kenichi Nishikawa, Fumihiko Ito, and Yasuhiro Suzuki. Composing User Network Operation Services Using Web Service Composition Techniques. In *Second IEEE Consumer Communications and Networking Conferece (CCNC 2005)*, Janeiro 2005.

[7] Mike P. Papazoglou and Dimitris Georgakopoulos. Service-oriented computing: Introduction. *Communications of the ACM*, 46(10):24–28, Outubro 2003.

[8] Mike P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *WISE 2003: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, Dezembro 2003.

[9] Web Services Architecture. `http://www.w3.org/TR/ws-arch/`.

[10] David A. Chappell and Tyler Jewell. *Java Web Services*. O'Reilly & Associates, First edition, Março 2002.

[11] Kevin J. Ma. Web Services: What's Real and What's Not? *IT Professional*, 2005.

[12] World Wide Web Consortium. *Extensible Markup Language (XML) Specification*, Agosto 2004. `http://www.w3c.org/XML/`.

[13] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`.

[14] World Wide Web Consortium. *Simple Object Access Protocol (SOAP) Version 1.2*, Junho 2003. `http://www.w3.org/TR/soap/`.

[15] Aaron E. Walsh, editor. *UDDI, SOAP, and WSDL: The Web Services Specification Reference Book*. Prentice Hall, 2002.

[16] Eric Newcomer. *Understanding Web Services - XML, WSDL, SOAP, and UDDI*. Addison Wesley, 2002.

[17] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 2003.

[18] Nizamuddin Channa, Shanping Li, Abdul Wasim Shaikh, and Xiangjun Fu. Constraint satisfaction in dynamic web service composition. In *Proceedings of the Sixteenth International Workshop on Database and Expert Systems Applications*, pages 658–664. IEEE, Agosto 2005.

[19] Nikola Milanovic and Miroslaw Malek. Current Solutions for Web Service Composition. *IEEE Internet Computing*, 2004.

[20] XLANG: Web Services for Business Process Design. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`.

[21] International Business Machines (IBM). *Web Services Flow Language (WSFL 1.0)*, Maio 2001. `www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`.

[22] United Nations Centre for Trade Facilitation and Electronic Business. `http://www.unece.org/cefact/`.

[23] ebXML. `http://www.ebxml.org`.

[24] UN/CEFACT e OASIS. *ebXML Business Process Specification Schema Version 1.01*, Maio 2001. `www.ebxml.org/specs/ebBPSS.pdf`.

[25] BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems. *Business Process Execution Language for Web Services Version 1.1*, Maio 2003.

[26] Organization for the Advancement of Structured Information Standards. `http://www.oasis-open.org/`.

[27] Assaf Arkin. *Business Process Modeling Language Version 1.0*. Business Process Management Initiative, Novembro 2002.

[28] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams. Web Services Conversation Language (WSCL) 1.0. `http://www.w3.org/TR/2002/NOTE-wscl10-20020314/`.

[29] World Wide Web Consortium (W3C). `http://www.w3.org`.

[30] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek. Web Services Choreography Interface (WSCI) 1.0. `http://www.w3.org/TR/wsci/`.

[31] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, and Yves Lafon. *Web Services Choreography Description Language Version 1.0*. World Wide Web Consortium (W3C), Dezembro 2004. `http://www.w3.org/TR/ws-cdl-10/`.

[32] Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. Web Services Composition Languages: Old Wine in New Bottles? In *EUROMICRO'03: Proceedings of the 29th EUROMICRO Conference N̈ew Waves in System Architecture:*, Setembro 2003.

[33] Apache Software Foundation. *Apache Jakarta Tomcat*, 2005. `http://jakarta.apache.org/tomcat/`.

[34] Business Process Execution Language for Web Services Java Run Time. `www.alphaworks.ibm.com/tech/bpws4j`.

[35] Oracle BPEL Process Manager. `http://www.oracle.com/appserver/bpel_home.html`.

[36] Oracle JDeveloper 10g. `http://www.oracle.com/`.

[37] Microsoft BizTalk Server 2004. `http://www.microsoft.com/biztalk/`.

[38] ActiveBPEL. `http://www.activebpel.org/`.

[39] ActiveWebflow Professional. `http://www.active-endpoints.com/`.

[40] Dirk Thissen. Flexible Service Provision Considering Specific Customer Needs. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP'02)*, pages 253–260. IEEE, 2002.

[41] Daniel A. Menascé. Composing Web Services: A QoS View. *IEEE Internet Computing*, 2004.

[42] Shalil Majuthia, David W. Walker, and W. A. Gray. Automated Web Service Composition Using Semantic Web Technologies. In *Proceedings of the International Conference on Autonomic Computing (ICAC'04)*, Maio 2004.

[43] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 2001.

[44] Keith Mantell. From UML to BPEL. `http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/`, Setembro 2003.

[45] David Skogan, Roy Groenmo, and Ida Solheim. Web Service Composition in UML. In *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 47–57. IEEE, 2004.

[46] Vikas Agarwal, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A Service Creation Environment Based on End to End Composition of Web Services. In *Proceedings of the 14th International World Wide Web Conference (WWW 2005)*, pages 128–137. ACM, Maio 2005.

[47] Russell Butek. Which style of WSDL should I use? `http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/%index.html`, Maio 2005.

[48] Tim Ewald. The Argument Against SOAP Encoding. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap%/html/argsoape.asp`, Outubro 2002.

[49] Frank Cohen. Discover SOAP Encoding's Impact on Web Services Performance. `http://www-128.ibm.com/developerworks/webservices/library/ws-soapenc/`, Março 2003.

[50] Web Services Interoperability Organization. *Basic Profile Version 1.1*, Agosto 2004. `http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html`.

[51] Tomás A. C. Badan, Rodrigo C. M. do Prado, Eduardo N. F. Zagari, Eleri Cardozo, and Maurício F. Magalhães. Uma Implementação de MPLS para Redes Linux. In *Simpósio Brasileiro de Redes de Computadores (SBRC 2001)*, volume 1, maio 2001.

[52] Silvio Mauro Tamashiro. Estudo de algoritmos de alocação de rota e comprimento de onda em redes Ópticas. Master's thesis, Universidade Estadual de Campinas, Campinas, Novembro 2003.

[53] Hui Zang, Jason P. Jue, and Biswanath Mukherjee. A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks. *Optical Networks Magazine*, 1, Janeiro 2000.

Apêndice A

Descrição das Interfaces

Neste apêndice foram incluídos os arquivos de descrição de interface dos serviços orquestrados (SGC e SGF). Os arquivos são escritos na linguagem de descrição de serviços *web* WSDL.

O arquivo WSDL do serviço de gerenciamento de conexões (SGC) é:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="OpticalConnectionService"
    targetNamespace="http://cururupe/ocs/control/OpticalConnectionService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://cururupe/ocs/control/OpticalConnectionService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://cururupe/ocs/control/OpticalConnectionService.xsd1">
    <wsdl:types>
        <xsd:schema
            targetNamespace="http://cururupe/ocs/control/OpticalConnectionService.xsd1"
            xmlns="http://www.w3.org/2000/10/XMLSchema"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://cururupe/ocs/control/OpticalConnectionService.xsd1">
            <xsd:element name="createLightPath" type="xsd1:createLightPathType"/>
            <xsd:complexType name="createLightPathType">
                <xsd:sequence>
                    <xsd:element name="user" type="xsd:string"/>
                    <xsd:element name="passwd" type="xsd:string"/>
                    <xsd:element name="switchingType" type="xsd:int"/>
                    <xsd:element name="ingressNode" type="xsd:string"/>
                    <xsd:element name="egressNode" type="xsd:string"/>
                    <xsd:element name="color" type="xsd:int"/>
                    <xsd:element name="setupPriority" type="xsd:int"/>
                    <xsd:element name="holdingPriority" type="xsd:int"/>
                    <xsd:element name="bandwidth" type="xsd:int"/>
                    <xsd:element name="protectionLevel" type="xsd:int"/>
                    <xsd:element name="fec" type="xsd:string"/>
                    <xsd:element name="route" type="xsd1:nodeArray"/>
                </xsd:sequence>
```

```
        </xsd:complexType>
        <xsd:complexType name="nodeArray">
            <xsd:sequence>
                <xsd:element name="node"
                    maxOccurs="unbounded"
                    minOccurs="0"
                    type="xsd1:nodeInfo"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="nodeInfo">
            <xsd:sequence>
                <xsd:element name="nodeIP" type="xsd:string"/>
                <xsd:element name="inInterface" type="xsd:string"/>
                <xsd:element name="outInterface" type="xsd:string"/>
                <xsd:element name="inInitialWL" type="xsd:int" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="inFinalWL" type="xsd:int" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="outInitialWL" type="xsd:int" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="outFinalWL" type="xsd:int" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:element
            name="createLightPathResponse"
            type="xsd1:createLightPathResponseType"/>
        <xsd:complexType name="createLightPathResponseType">
            <xsd:sequence>
                <xsd:element name="LPID" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="dropLightPath" type="xsd1:dropLightPathType"/>
        <xsd:complexType name="dropLightPathType">
            <xsd:sequence>
                <xsd:element name="user" type="xsd:string"/>
                <xsd:element name="passwd" type="xsd:string"/>
                <xsd:element name="LPID" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="dropLightPathResponse" type="xsd1:dropLightPathResponseType"/>
        <xsd:complexType name="dropLightPathResponseType">
            <xsd:sequence>
                <xsd:element name="droped" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="faultMsg" type="xsd1:faultMsgType"/>
        <xsd:complexType name="faultMsgType">
            <xsd:sequence>
                <xsd:element name="reason" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:schema>
</wsdl:types>
<wsdl:message name="createLightPathResponse">
    <wsdl:part element="xsd1:createLightPathResponse" name="response"/>
</wsdl:message>
<wsdl:message name="dropLightPathResponse">
```

```
        <wsdl:part element="xsd1:dropLightPathResponse" name="response"/>
    </wsdl:message>
    <wsdl:message name="dropLightPathRequest">
        <wsdl:part element="xsd1:dropLightPath" name="request"/>
    </wsdl:message>
    <wsdl:message name="createLightPathRequest">
        <wsdl:part element="xsd1:createLightPath" name="request"/>
    </wsdl:message>
    <wsdl:message name="faultMessage">
        <wsdl:part element="xsd1:faultMsg" name="reason"/>
    </wsdl:message>
    <wsdl:portType name="OpticalConnectionServicePortType">
        <wsdl:operation name="createLightPath">
            <wsdl:input message="tns:createLightPathRequest"/>
            <wsdl:output message="tns:createLightPathResponse"/>
        </wsdl:operation>
        <wsdl:operation name="dropLightPath">
            <wsdl:input message="tns:dropLightPathRequest"/>
            <wsdl:output message="tns:dropLightPathResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="OCSSOAPBinding" type="tns:OpticalConnectionServicePortType">
     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="createLightPath">
      <soap:operation
       soapAction=
       "server:OpticalConnectionServiceProcess:OpticalConnectionServicePortType#createLightPath"/>
       <wsdl:input>
        <soap:body use="literal"/>
       </wsdl:input>
       <wsdl:output>
        <soap:body use="literal"/>
       </wsdl:output>
      </wsdl:operation>
     <wsdl:operation name="dropLightPath">
      <soap:operation
       soapAction=
       "server:OpticalConnectionServiceProcess:OpticalConnectionServicePortType#dropLightPath"/>
       <wsdl:input>
        <soap:body use="literal"/>
       </wsdl:input>
       <wsdl:output>
        <soap:body use="literal"/>
       </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="OpticalConnectionServiceProcess">
      <wsdl:port binding="tns:OCSSOAPBinding" name="OpticalConnectionServiceProcessPort">
       <soap:address
        location="http://10.10.1.24:8080/active-bpel/services/OpticalConnectionServiceProcess"/>
       </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

O arquivo WSDL do serviço de gerenciamento de falhas (SGF) é:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="FaultManagerService"
    targetNamespace="http://cururupe/fms/control/FaultManagerService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://cururupe/fms/control/FaultManagerService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://cururupe/fms/control/FaultManagerService.xsd1">
    <wsdl:types>
        <xsd:schema
            targetNamespace="http://cururupe/fms/control/FaultManagerService.xsd1"
            xmlns="http://www.w3.org/2000/10/XMLSchema"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://cururupe/fms/control/FaultManagerService.xsd1">
            <xsd:element name="notifyFault" type="xsd1:notifyFaultType"/>
            <xsd:complexType name="notifyFaultType">
                <xsd:sequence>
                    <xsd:element name="LPID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element
                name="notifyFaultResponse"
                type="xsd1:notifyFaultResponseType"/>
            <xsd:complexType name="notifyFaultResponseType">
                <xsd:sequence>
                    <xsd:element name="response" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="clearFault" type="xsd1:clearFaultType"/>
            <xsd:complexType name="clearFaultType">
                <xsd:sequence>
                    <xsd:element name="LPID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="clearFaultResponse" type="xsd1:clearFaultResponseType"/>
            <xsd:complexType name="clearFaultResponseType">
                <xsd:sequence>
                    <xsd:element name="response" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="faultMsg" type="xsd1:faultMsgType"/>
            <xsd:complexType name="faultMsgType">
                <xsd:sequence>
                    <xsd:element name="reason" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="notifyFaultResponse">
        <wsdl:part element="xsd1:notifyFaultResponse" name="response"/>
```

```
</wsdl:message>
<wsdl:message name="clearFaultResponse">
    <wsdl:part element="xsd1:clearFaultResponse" name="response"/>
</wsdl:message>
<wsdl:message name="clearFaultRequest">
    <wsdl:part element="xsd1:clearFault" name="request"/>
</wsdl:message>
<wsdl:message name="notifyFaultRequest">
    <wsdl:part element="xsd1:notifyFault" name="request"/>
</wsdl:message>
<wsdl:message name="faultMessage">
    <wsdl:part element="xsd1:faultMsg" name="reason"/>
</wsdl:message>
<wsdl:portType name="FaultManagerServicePortType">
    <wsdl:operation name="notifyFault">
        <wsdl:input message="tns:notifyFaultRequest"/>
        <wsdl:output message="tns:notifyFaultResponse"/>
    </wsdl:operation>
    <wsdl:operation name="clearFault">
        <wsdl:input message="tns:clearFaultRequest"/>
        <wsdl:output message="tns:clearFaultResponse"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="FMSSOAPBinding" type="tns:FaultManagerServicePortType">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="notifyFault">
   <soap:operation
    soapAction=
    "server:FaultManagerServiceProcess:FaultManagerServicePortType#notifyFault"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="clearFault">
    <soap:operation
     soapAction=
     "server:FaultManagerServiceProcess:FaultManagerServicePortType#clearFault"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<wsdl:service name="FaultManagerServiceProcess">
  <wsdl:port binding="tns:FMSSOAPBinding" name="FaultManagerServiceProcessPort">
   <soap:address
    location="http://10.10.1.24:8080/active-bpel/services/FaultManagerServiceProcess"/>
  </wsdl:port>
</wsdl:service>
```

```
</wsdl:definitions>
```

FaultManagerService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="FaultManagerService"
    targetNamespace="http://cururupe/fms/control/FaultManagerService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://cururupe/fms/control/FaultManagerService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://cururupe/fms/control/FaultManagerService.xsd1">
    <wsdl:types>
        <xsd:schema
            targetNamespace="http://cururupe/fms/control/FaultManagerService.xsd1"
            xmlns="http://www.w3.org/2000/10/XMLSchema"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://cururupe/fms/control/FaultManagerService.xsd1">
            <xsd:element name="notifyFault" type="xsd1:notifyFaultType"/>
            <xsd:complexType name="notifyFaultType">
                <xsd:sequence>
                    <xsd:element name="LPID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element
                name="notifyFaultResponse"
                type="xsd1:notifyFaultResponseType"/>
            <xsd:complexType name="notifyFaultResponseType">
                <xsd:sequence>
                    <xsd:element name="response" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="clearFault" type="xsd1:clearFaultType"/>
            <xsd:complexType name="clearFaultType">
                <xsd:sequence>
                    <xsd:element name="LPID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="clearFaultResponse" type="xsd1:clearFaultResponseType"/>
            <xsd:complexType name="clearFaultResponseType">
                <xsd:sequence>
                    <xsd:element name="response" type="xsd:string"/>
                </xsd:sequence>
```

```
        </xsd:complexType>
        <xsd:element name="faultMsg" type="xsd1:faultMsgType"/>
        <xsd:complexType name="faultMsgType">
            <xsd:sequence>
                <xsd:element name="reason" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:schema>
</wsdl:types>
<wsdl:message name="notifyFaultResponse">
    <wsdl:part element="xsd1:notifyFaultResponse" name="response"/>
</wsdl:message>
<wsdl:message name="clearFaultResponse">
    <wsdl:part element="xsd1:clearFaultResponse" name="response"/>
</wsdl:message>
<wsdl:message name="clearFaultRequest">
    <wsdl:part element="xsd1:clearFault" name="request"/>
</wsdl:message>
<wsdl:message name="notifyFaultRequest">
    <wsdl:part element="xsd1:notifyFault" name="request"/>
</wsdl:message>
<wsdl:message name="faultMessage">
    <wsdl:part element="xsd1:faultMsg" name="reason"/>
</wsdl:message>
<wsdl:portType name="FaultManagerServicePortType">
    <wsdl:operation name="notifyFault">
        <wsdl:input message="tns:notifyFaultRequest"/>
        <wsdl:output message="tns:notifyFaultResponse"/>
    </wsdl:operation>
    <wsdl:operation name="clearFault">
        <wsdl:input message="tns:clearFaultRequest"/>
        <wsdl:output message="tns:clearFaultResponse"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="FMSSOAPBinding" type="tns:FaultManagerServicePortType">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="notifyFault">
   <soap:operation
    soapAction=
    "server:FaultManagerServiceProcess:FaultManagerServicePortType#notifyFault"/>
    <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
```

```
        <wsdl:output>
         <soap:body use="literal"/>
        </wsdl:output>
       </wsdl:operation>
       <wsdl:operation name="clearFault">
        <soap:operation
         soapAction=
         "server:FaultManagerServiceProcess:FaultManagerServicePortType#clearFault"/>
        <wsdl:input>
         <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
         <soap:body use="literal"/>
        </wsdl:output>
       </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="FaultManagerServiceProcess">
      <wsdl:port binding="tns:FMSSOAPBinding" name="FaultManagerServiceProcessPort">
       <soap:address
        location="http://10.10.1.24:8080/active-bpel/services/FaultManagerServiceProcess"
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```