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

Fabrízzio Cabral de Lacerda

Uma proposta de arquitetura para o protocolo NETCONF sobre SOAP

Campinas, SP

2007

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

Lacerda, Fabrízzio Cabral de
L6116p Uma proposta de arquitetura para o protocolo NETCONF sobre SOAP / Fabrízzio Cabral de Lacerda. --Campinas, SP: [s.n.], 2007.

Orientador: Maurício Ferreira Magalhães
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computadores (Gerenciamento). 2. Serviços na Web. 3. Redes de computação – Protocolos. 4. XLM (Linguagem de marcação de documento). Java (Linguagem de programação de computador). I. Magalhães, Maurício Ferreira. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: An architecture proposal for the NETCONF protocol over SOAP
Palavras-chave em Inglês: Network management, Configuration management, Web services, Configuration protocol, Data Model, JAVA, XML, SOAP, NETCONF, SNMP
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Edmundo Roberto Mauro Madeira, Eleri Cardozo, Fábio Luciano Verdi
Data da defesa: 30/08/2007
Programa de Pós-Graduação: Engenharia Elétrica

Fabrízzio Cabral de Lacerda

Uma proposta de arquitetura para o protocolo NETCONF sobre SOAP

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: Prof. Dr. Mauricio Ferreira Magalhães

Banca Examinadora:
Prof. Dr. Edmundo Roberto Mauro Madeira – IC/UNICAMP
Prof. Dr. Eleri Cardozo – DCA/FEEC/UNICAMP
Dr. Fábio Luciano Verdi – DCA/FEEC/UNICAMP
Prof. Dr. Mauricio Ferreira Magalhães – DCA/FEEC/UNICAMP

Campinas, SP

2007

Ata de defesa

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Fabrízzio Cabral de Lacerda

Data da Defesa: 30 de agosto de 2007

Título da Tese: "Uma Proposta de Arquitetura para o Protocolo NETCONF sobre SOAP"

Prof. Dr. Maurício Ferreira Magalhães (Presidente): _____

Prof. Dr. Edmundo Roberto Mauro Madeira: _____

Prof. Dr. Eleri Cardozo: _____

Prof. Dr. Fábio Luciano Verdi: _____

Resumo

A gerência de redes é formada por cinco áreas funcionais: Falha, Configuração, Contabilidade, Desempenho e Segurança. A área de configuração é responsável pela operação e manutenção da rede, acompanhando as mudanças de configuração realizadas em cada dispositivo da rede. As principais ferramentas de gerência disponíveis, CLI e SNMP, não atendem aos requisitos de configurações das redes atuais. Novas tecnologias Web estão se tornando comuns na gerência de redes, com destaque para o uso da linguagem XML e do protocolo HTTP. Com o objetivo de aplicar estas novas tecnologias na configuração de redes foi definido, no âmbito do IETF, um novo protocolo de gerência de configuração denominado NETCONF. Este trabalho faz um estudo do protocolo NETCONF procurando destacar as suas vantagens e limitações. Este trabalho propõe, também, uma arquitetura de implementação para o NETCONF baseada no protocolo de transporte SOAP sobre HTTP, ou sobre HTTPs. Com o objetivo de validar a arquitetura, apresentamos a implementação de um protótipo totalmente aderente à proposta NETCONF para o qual foi especificado um modelo de dados para configuração de VLANs de switches de fabricantes diferentes. `

Palavras-Chaves: Gerência de Redes, Gerência de Configuração, CLI, SNMP, HTTP, HTTPs, XML, Web Services, SOAP, NETCONF, SWITCH, VLAN.

Abstract

Network management is formed by five functional areas: Failure, Configuration, Accounting, Performance and Security. The configuration area is responsible for the network's operation and maintenance, following the configuration changes done in each network's device. The main management tools available, CLI and SNMP, do not take care of the configuration requirements of current networks. New Web technologies are becoming widespread in network management, with prominence of XML language and HTTP protocol. A new protocol of configuration management named NETCONF has been defined, in the scope of the IETF, in order to apply these new technologies for configuration of networks. This work studies the NETCONF protocol aiming to highlight its advantages and limitations. This work also proposes an architecture of implementation for the NETCONF based on the transport protocol SOAP over HTTP, or HTTPs. Aiming to validate such architecture, we present the implementation of a prototype fully adherent to the NETCONF proposal, for which it has specified a model of data for configuration of VLANs of switches from different manufacturers.

Keywords: Network management, Configuration management, CLI, SNMP, HTTP, HTTPs, XML, Web Services, SOAP, NETCONF, SWITCH, VLAN.

Dedico este trabalho a minha esposa,

Renata Soares

Agradecimentos

Agradeço primeiramente a Deus por sempre estar comigo hoje e sempre nesta caminhada.

Agradeço aos meus pais, por tudo que me ensinaram. Meu pai como exemplo supremo de trabalho. Minha mãe pelos puxões de orelha (eu sei que foram para o meu bem).

Aos meus irmãos Maria Carolina e Paulo José, pelo carinho e atenção.

A toda a minha família, os Cabral, os Assis, os Lacerda. Todos os tios, primas, e tia-bisavós vivos são mais de 200!!

Aos amigos de longa data (e viva o Colégio de Aplicação da UFG!!). Fábio Oliveira, se conheço um pouco de SAMPA devo isto a você. Aos amigos recentes.

Aos amigos que gentilmente me receberam em suas casas em Campinas: Renato Cabral (Lilica), Vinicius (Muchacho) Junqueira, Corival do Carmo (mesmo não estado um segundo sequer em sua casa, por falta minha), o casal Yuri Mendonça (Gagarin) e Gabriela Meirelles, o casal Rodrigo Prado (Loção) e Michelle.

A todos os amigos do LCA, em especial: Fábio Verdi, Rafael Duarte, Rafael Pasquini, Luiz Gustavo, RossanoPablo (Ross), Nicola Zaaaagari e Rodrigo (Loção) de novo. E porque não?

Ao professor e agora colega Alexandre Ribeiro que me incentivou a mandar o formulário para entrar no programa de pós-graduação desta instituição.

Aos desafetos, onde quer que estejam.

Ao suporte financeiro do CNPq e da Ericsson.

"And last, but not least", minha esposa. Muito obrigado por seu carinho, seu suporte incondicional (em todas as áreas), sua paciência e o incentivo. Obrigado por me mostrar os USA. Obrigado por estar comigo.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

API	Application Programming Interface
ASCII	American Standard Code Information Interchange
ASN.1	Abstract Syntax Notation One
AT	Address Translation
BEEP	Blocks Extensible Exchange Protocol
BER	Basic Encoding Rule
CIM	Common Information Model
CLI	Command Line Interface
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DM	Data Model
DMTF	Distributed Management Task Force
DNS	Domain Name System
DOM	Document Object Model
DTD	Document Type Definition
EGP	External Gateway Protocol
ENSUITE	Extended Netconf Suite
FCAPS	Failure, Configuration, Accounting, Performance, Security
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IBM	International Business Machines
ID	Identifier
IETF	Intenet Engineering Task Force
IM	Information Model
INTAP	Interoperability Technology Association for Information Processing
IOS	Internetwork Operating System
IP	Internet Protocol
IPV4	Internet Protocol Version 4
ISO	International Organization for Standardization
JMX	Java Management Extensions
JUNOS	Juniper Operating Systems
LGPL	Lesser General Public License
MIB	Management Information Base
MIB-II	Management Information Base-II
MO	Managed Objects
MOF	Meta-Object Facility
MOM	Message-Oriented Model
MPLS	Multiprotocol Label Switching
NAT	Network Address Translation

NETCONF	Network Configuration
NMS	Network Management System
OASIS	Organization for the Advancement of Structured Information Standards
OID	Object Identifier
OSI	Open Systems Interconnection
OSMIC	Open Systems Management Industry Collaboration
PDU	Protocol Data Unit
PIB	Policy Information Base
PM	Policy Model
QNAME	Qualified Name
RADIUS	Remote Authentication Dial-In User Service
RFC	Request for Comments
RMI	Remote Method Invocation
ROM	Resource-Oriented Model
RPC	Remote Procedure Call
SAX	Simple Api for Xml
SGML	Standard Generalized Markup Language
SLA	Service Level Agreement
SMI	Structure of Management Information
SMIV2	Structure of Management Information Version 2
SNMP	Simple Network Management Protocol
SNMPV1	Simple Network Management Protocol Version 1
SNMPV2	Simple Network Management Protocol Version 2
SNMPV3	Simple Network Management Protocol Version 3
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPPI	Structure of Policy Provisioning Information
SSH	Secure Shell
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
TL1	Transaction Language 1
TMN	Telecommunications Management Network
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UML	Unified Modeling Language
URI	Uniform Resouce Identifier
URN	Uniform Resource Name
VLAN	Virtual Local Area Network
VOIP	Voice over Internet Protocol
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WBEM	Web-based Enterprise Management
WIMA	Web-based Integrated Management Architecture
WSDD	Web Services Deployment Descriptor
WSDL	Web Services Definition Language

WS-I	Web Services-Interoperability Organization
WSRF	Web Services Resource Framework
XML	Extensible Markup Language
XMLCONF	XML Configuration
XMLNS	Xml Namespace
XSLT	Extensible Stylesheet Language Transformations

Trabalhos publicados com participação do autor

1. F. L. Verdi, R. Duarte, F. C. de Lacerda, E. Madeira, E. Cardozo and M. Magalhães. Provisioning and Management of Inter-Domain Connections in Optical Networks: A Service Oriented Architecture-based Approach. *IEEE/IFIP Network Operations and Management Symposium* (NOMS 2006). Vancouver, April 2006.

2. F. L. Verdi, R. Duarte, F. C. de Lacerda, E. Madeira, E. Cardozo and M. Magalhães. Web Services-based Provisioning of Connections in GMPLS Optical Networks (in english). *The Brazilian Symposium on Computer Networks* (SBRC 2005). Fortaleza-CE, Brazil. May 2005.

3. M. Siqueira, R. Pasquini, F. L. Verdi, R. Duarte, F. C. de Lacerda, E. Madeira and M. Magalhães. Um Mecanismo Baseado em Web services para Divulgação de Topologias Virtuais Inter-Domínios em Redes GMPLS. *X Workshop on Network and Services Management and Operations* (WGRS 2005) co-located with SBRC 2005. Fortaleza-CE, Brazil. May 2005.

Capítulo 1

Introdução

Um modelo de gerência é definido pela *International Organization of Standardization* (ISO) [1]. As suas diversas funcionalidades são referenciadas pelas seguintes letras, do inglês, FCAPS [2]: falha (*fault*), configuração (*configuration*), contabilidade (*accounting*), desempenho (*performance*), segurança (*security*). A seguir são comentadas de forma resumida as áreas funcionais da gerência:

- Falha - O gerenciamento de falhas se caracteriza por reconhecer, isolar, corrigir e registrar as falhas ocorridas na rede.
- Contabilidade - O objetivo da contabilidade é coletar estatísticas de uso dos usuários. Está relacionada, geralmente, ao sistema de faturamento.
- Desempenho – O desempenho permite ao gerente preparar a rede para o futuro ao coletar e analisar os dados de desempenho, identificando gargalos e monitorando parâmetros da rede, como consumo da banda, ao longo do tempo e/ou em períodos específicos.
- Segurança - A segurança é responsável por identificar os riscos que uma rede está suscetível, tais como acessos não autorizados e invasões. Posteriormente, estes riscos devem ser corrigidos e controlados.

Por último, temos a área de configuração. A configuração diz respeito a toda e qualquer ação que visa à operação e manutenção da rede. Inclui como tarefas: a coordenação da eliminação de sistemas obsoletos, e a inclusão de novos hardwares tais como placas de expansão, e software, por exemplo, versões de firmware e sistemas operacionais. Os principais objetivos da área de configuração são:

1. Obter e armazenar as configurações de cada dispositivo de rede;
2. Simplificar a configuração do dispositivo;
3. Acompanhar as mudanças que são feitas na configuração de cada dispositivo.

A configuração dos dispositivos de rede nunca foi realizada com a eficiência necessária. O protocolo mais usual para gerência de redes é o *Simple Network Management Protocol* (SNMP) [3]. O SNMP se caracteriza pelo gerente realizar operações no agente. No paradigma gerente-agente, o gerente é o cliente e o agente o servidor. As operações mais usuais são:

GetRequest	operação de consulta;
GetResponse	resposta à operação de consulta;
SetRequest	operação de configuração;
GetNextRequest	requisita a próxima entidade à entidade especificada;
GetBulkRequest	requisita mais de uma seleção de uma única vez;
Trap	operação de notificação.

Mais detalhes são apresentados no Capítulo 2.

O uso do SNMP para consultas tem sido considerado um sucesso. Todos os fabricantes de dispositivos de rede, principalmente os que suportam o Protocolo Internet – *Internet Protocol* (IP) - o traz embutido em seus sistemas nativamente. Em [4] e [6], são apontadas algumas das suas vantagens. Destaca-se a sua simplicidade e a grande capacidade de interoperabilidade, principalmente quanto às Bases de Informação de Gerência (*Management Information Bases* – MIBs [5]) genéricas que são bastantes estáveis e estão presentes em praticamente todos os agentes.

No entanto, a lista de deficiências não é pequena, principalmente no que diz respeito à eficiência e escalabilidade [4]. Outros problemas do SNMP também são detalhados por [6], como a sua má reputação no mercado por ser inseguro, complexo, lento e com funções limitadas. No livro [7] outros problemas do SNMP também são apontados, principalmente quanto às mensagens G*etBulk* (*Simple Network Management Protocol version 2* - SNMPv2 [8]) e o tamanho máximo da mensagem SNMP.

Por isso, para a tarefa de configuração de dispostivos, os fabricantes de dispositivos de rede e os próprios usuários optaram por soluções proprietárias ou "caseiras" para os seus sistemas de gerência. Principalmente, porque uma das tarefas de configuração envolve o uso de arquivos de configuração que geralmente são extensos e o SNMP não é eficiente para estas trocas com as suas operações de *SetRequest* ou *GetNextRequest* [9]. Além disso, o SNMP não possui um modelo de transação atômico, principalmente pelo uso do protocolo *User Datagram Protocol* (UDP) como transporte, que não possui estado [9].

Apesar do protocolo de gerência mais comum ser o SNMP, principalmente para monitoramento e desempenho, a forma mais comum de se configurar é via interface de comando de linha (*Command Line Interface* – CLI). Sua principal vantagem é ser embutida no dispositivo gerenciado, podendo ser acessado a qualquer momento via sinalização fora-da-banda[1] (*out-of-band)*, mesmo que a sinalização dentro-da-banda[2] (*in-band)* não esteja disponível. A sua grande desvantagem é que cada fabricante adota uma forma, um estilo de comandos, dificultando a interoperabilidade. Outras vantagens e desvantagens da CLI são apresentadas no Capítulo 2.

Alternativas ao SNMP começaram a surgir com a popularidade da Internet, o uso de navegadores e do *HyperText Markup Language* (HTML [10]). Assim, os fabricantes passaram a fornecer um servidor *HyperText Transfer Protocol* (HTTP [11]) embutido nos dispositivos para acesso pelo gerente e administradores de rede. O seu apelo visual, facilitando, por exemplo, o uso de gráficos, logo se tornou comum. Pode-se dizer que isto aconteceu com a introdução da Internet comercial em 1993/1994.

Um avanço ao gerenciamento baseado em Web ocorreu com o uso de *applets* Java dentro do *web-browser* e também como aplicação gerente. Assim, também se tornou comum, por volta de 1997/1998[12], o fabricante do dispositivo entregar, junto com o equipamento, um cd com o software gerente baseado em *applets*,

Iniciativas apoiadas pelas operadoras de telecomunicações, basearam-se no uso de *Common Object Request Broler Architecture* (CORBA [13]), Java *Remote Method Invocation*

[1] Sinalização fora-da-banda: dados de controle/gerência são transportados em um canal separado dos demais dados.
[2] Sinalização dentro-da-banda: dados de controle/gerência são transportados no mesmo canal dos dados.

(RMI [14]) e de arquiteturas específicas. O CORBA foi adotado principalmente por sua robustez, por ser um sistema distribuído e ter suporte à transação.

Um exemplo de arquitetura definida pelo *Distributed Management Task Force* (DMTF [15]) foi o *Web-Based Enterprise Management* (WBEM [16]). Para a modelagem de dados baseada em UML [17] é utilizado o mapeamento WBEM/*Common Information Model* (CIM) [18]. Todas essas propostas foram ou são bastante utilizadas, apesar do alto custo de desenvolvimento.

Uma proposta de arquitetura baseada em tecnologias *Webs*: HTTP, *Extensible Markup Language* (XML [19]), *applets e* HTML foi o *Web-based Integrated Management Architecture* (WIMA [7]). O objetivo da arquitetura é resolver os problemas encontrados no SNMP e utilizar as novas tecnologias que começavam a se tornar comuns na época da divulgação do trabalho, no ínicio da década de 90. Outra proposta partiu da Sun com o *Java Management Extensions* (JMX [20]), baseado no Java.

Todas estas propostas caminharam para o uso de XML. O XML ganhou bastante espaço entre os desenvolvedores, pois as implementações de suas especificações, *Document Object Model* (DOM [21]) e *Simple API for XML* (SAX [22]), são tidas como maduras, robustas e rápidas, além de serem oferecidas de maneira gratuita. Assim, mais recentemente, tivemos uma explosão de soluções que transformam as MIBs SNMP em XML e vice-versa[9], bem como o uso de *gateways* XML-SNMP[4].

Na prática, no atual cenário das grandes operadoras de telecomunicações - com suas redes complexas e altamente heterogêneas, devido aos equipamentos de diferentes fabricantes - para se obter o estado atual da rede é necessário acessar os programas de gerência de cada fabricante. Somando-se a isso, tem-se a questão do custo envolvido em manter um operador de rede que seja especialista em rede, e muitas vezes, especialista neste ou naquele fabricante. Um profissional com estas capacidades é caro por exigir muitas horas de treinamento específico e uma alta remuneração.

Atualmente, a configuração não é realizada de forma automática, necessitando sempre da intervenção humana. Numa rede pequena isto não é problema. No entanto, para uma rede complexa - formada por vários elementos de rede, que necessita de alta interoperabilidade e ofereça uma sofisticada gama de serviços tais como: *Virtual Private Network* (VPN), *Voice over IP* (VoIP*), Multiprotocol Switching Service* (MPLS) , serviços aderentes a *Service Level*

Agreements (SLAs) - configurar os elementos de rede manualmente, ou mesmo com *scripts*, significa estar sujeito a uma alta margem de erro. Isto sem falar na própria demora em se entregar um novo serviço ao cliente, devido à necessidade de se configurar diversos dispositivos, de diversos fabricantes e posteriormente testar se a mudança foi satisfatória.

Por isso, também tem sido bastante discutido o uso dos *Web services* e de *Service Oriented Architecture* (SOA [23) para gerência de redes [24] [25]. Neste sentido, *o Web Services Resources Framework* (WSRF [26]) da *Organization for the Advancement of Structured Information Standards* (OASIS [27]) é o seu principal representante. A principal razão é justamente permitir que se possa interagir com os agentes de uma maneira mais automática, diminuindo a intervenção humana, e de forma programática.

Com o objetivo de contornar os problemas apontados anteriomente e ao mesmo tempo continuar seguindo as boas práticas já alcançadas, foi definido no âmbito do *Internet Engineering Task Force* (IETF) em Julho de 2002, o grupo de trabalho *XML Configuration* (XMLCONF). Posteriormente, o grupo de trabalho passou a se chamar NETwork CONFiguration: NETCONF [28], que também dá nome ao protocolo. A sua intenção é utilizar o XML como formato de codificação das mensagens e chamadas de métodos remotos (*Remote Procedure Call* - RPC) como paradigma de comunicação, de forma a permitir uma interação programática com os dispositivos e os softwares de gerência. Para isso, foram definidas capacidades e operações básicas, assim como, o formato das mensagens. Este protocolo define também como um agente deve reagir ao receber uma operação, levando em conta o seu estado atual, além de permitir facilmente agregar novas funcionalidades a partir de extensões às suas capacidades básicas. O seu transporte é definido para os seguintes protocolos: *Secure Shell* (SSH [29]), *Blocks Extensible Exchange Protocol* (BEEP [30]) e *Simple Object Access Protocol* (SOAP [31]).

Como registro, é importante salientar o papel da empresa Juniper [32]. Ela foi a primeira a permitir a configuração de um equipamento com mensagens de chamada de procedimento remoto encapsuladas em XML com o sistema operacional de rede *Juniper Networks Operating System* (JUNOS) . Sendo este o modelo utilizado para o protocolo NETCONF nascente.

Neste trabalho analisamos o uso do NETCONF para gerenciar *Virtual Local Area Network* (VLANs) em switches de diferentes fabricantes. Para isso, definimos uma arquitetura de implementação que respeita integralmente as mensagens definidas pelo protocolo e utilizamos como transporte o protocolo SOAP. O SOAP nesta implementação foi utilizado nativamente através da interface de programação de aplicação (*Application Programming Interface* – API) fornecida pela Apache no projeto AXIS para a linguagem de programação Java. O SOAP foi mapeado para HTTP *e HyperText Transfer Protocol Secure* (HTTPs [34]), no qual avaliamos o tamanho das mensagens e o tempo necessário para o protocolo realizar consultas à tabela de rotas do próprio agente e às VLANs dos switches. Outra contribuição deste trabalho é a definição de um modelo de dados através de um XML SCHEMA para se atingir o objetivo prático principal de configurar os switches. Uma contribuição final foi a implementação de um serviço de notificações. Este serviço foi implementado utilizando o modelo *push* clássico, no qual o gerente realiza a assinatura para receber eventos ocorridos no agente. Ocorrido o evento, o agente envia o evento para todos os agentes que assinaram o serviço.

O conteúdo deste trabalho é formado por cinco capítulos, incluindo esta introdução. A seguir, apresenta-se uma breve descrição dos assuntos abordados nos demais capítulos:

- O Capítulo 2 aborda as tecnologias utilizadas para a avaliação do protocolo NETCONF tais como CLI, SNMP, XML, HTTP, SOAP e *Web Services*. Revisa o protocolo SNMP com foco na questão de configuração e apresenta trabalhos relacionados. Por último, apresenta os requisitos necessários para a criação de um protocolo específico para a gerência de configuração.

- O Capítulo 3 apresenta o protocolo NETCONF em detalhes: suas mensagens, operações, modelo de comunicação, protocolos de transporte para os quais já foi mapeado, seu esquema de filtro e suas capacidades como forma de estender o protocolo base. Por último, discute as diferenças do modelo de dados para o modelo de informação e apresenta os pontos negativos do protocolo NETCONF.

- O Capítulo 4 descreve a arquitetura proposta, os detalhes da implementação da arquitetura, o modelo de dados proposto, o serviço de eventos e o funcionamento do sistema durante a validação da arquitetura no cenário de configuração de VLANs para switches de diferentes fabricantes e os resultados obtidos em três classes de testes distintas.

- O Capítulo 5 apresenta as conclusões do trabalho e descreve os trabalhos futuros a serem realizados.

Capítulo 2
Gerência de Redes

Neste capítulo são definidos os componentes básicos da Gerência de Redes. O foco é uma revisão das tecnologias de Gerência de Redes para a área funcional de Gerência de Configuração. São mostradas e comentadas as principais tecnologias para Gerência de Redes: CLI; SNMP e suas versões; as deficiências destas tecnologias para a área funcional de Gerência de Configuração; as novas tendências a partir de tecnologias Web, principalmente XML e HTTP; trabalhos acadêmicos de *gateways* SNMP-XML e SNMP-WEB SERVICES. Por último, serão apresentados os requisitos necessários para a definição de uma nova tecnologia de Gerência de Configuração.

2.1. Introdução

A gerência de redes é constituída por quatro componentes básicos: elementos gerenciados, estações de gerência, protocolos de gerência e informações de gerência.

Elementos gerenciados

Os elementos gerenciados (*managed elements*) constituem os componentes da rede que precisam operar adequadamente para que a rede ofereça os serviços para os quais foi projetada. Devem possuir um software especial, chamado agente para permitir que sejam gerenciados remotamente. O agente permite a monitoração e o controle de um componente

por uma ou mais estações de gerência, respondendo às solicitações dos indicadores solicitados. Exemplos:

- *Hardware*: equipamentos de interconexão, enlaces de comunicação, hospedeiros, nobreaks, modems, impressoras, etc;
- *Software*: sistemas operacionais, servidores de bancos de dados, servidores Web, servidores de correio, etc.

Estes elementos gerenciados em uma arquitetura de Gerência são as instâncias de objetos gerenciados (*managed objects* - MO). Um MO é a modelagem de um objeto. Oferece uma interface de acesso abstrata, que encapsula o recurso. Pode-se notar neste nível de abstração a importância do uso de conceitos de orientação a objetos como herança e polimorfismo.

Estações de gerência

As estações de gerência (*management workstation*) contêm o *software*, chamado gerente, que conversa diretamente com os agentes nos componentes gerenciados, com o objetivo de monitorá-los ou controlá-los. O gerente comunica-se com agentes através de *pollings* ou *trapings*.

- *Polling* (varredura): processo de obtenção das informações junto ao agente em que o gerente toma a iniciativa da comunicação;
- *Trapings* (notificações): processo onde o agente toma a iniciativa de enviar ao gerente (pré-configurado) uma notificação de ocorrência de eventos anormais, previamente configurados.

Normalmente as estações de gerência são centralizadas. O mais comum é que haja uma única estação de gerência. Possuem funções automáticas de gerência, como a consulta regular dos agentes. Outra função é obter e alterar informações de gerência presente nos agentes. Geralmente possuem uma interface gráfica com o usuário para facilitar a gerência.

Protocolos de gerência

De forma resumida, um protocolo de gerência define as mensagens usadas entre gerentes e agentes para trocar informações de gerência. Permite operações de monitoramento (*READ*) e operações de controle (*WRITE*). A maioria dos protocolos são baseados no modelo requisição-resposta. Exemplos de protocolos:

- SNMP, SNMPv2 e *Simple Network Management Protocol version 3* (SNMPv3 [38]) usados em redes *Transmission Control Protocol/Internet Protocol* (TCP/IP);
- CMIP (*Commom Management Information Protocol* [36]) protocolo do modelo *Open Systems Interconnection* (OSI [37]);
- NETCONF usado em redes TCP/IP;

Outros modelos de gerência importantes, que não são protocolos: CLI, WBEM, *Telecommunications Management Network* (TMN [38]), *Transaction Language* 1 (TL/1 [39]) e JMX.

Informações de gerência

Informações de gerência definem os dados que podem ser referenciados em operações do protocolo na comunicação entre gerentes e agentes. Corresponde ao conjunto das informações de gerência disponíveis em um agente. O principal exemplo é a MIB.

Tecnologias para Gerência de Redes TCP/IP

A seguir são comentadas as principais tecnologias de Gerência de Redes. A principal análise destas tecnologias diz respeito aos problemas, dificuldades e deficiências para realizar a funcionalidade de Gerência de Configuração do FCAPS.

2.2. CLI - Interface de comando de linha

A tecnologia para configuração de dispositivos de rede mais utilizada é a CLI. Apesar de cada fabricante adotar uma maneira/forma de implementar a ferramenta, na qual os comandos para a mesma tarefa geralmente divergem de um fabricante para o outro, continua sendo a primeira opção dos operadores, apesar de outras tecnologias (seções a seguir) estarem disponíveis. As principais razões para o seu sucesso são:

- Codificação dos comandos em *American Standard Code Information Interchange* (ASCII). Isto permite que o arquivo de configuração seja armazenado na forma de texto. Através de ferramentas simples, baratas e muitas vezes gratuitas, facilmente se pode pesquisar, comparar com outros arquivos, anexar numa mensagem eletrônica, copiar e colar e transferir para o dispositivo o arquivo de configuração;
- Similar ao Shell Unix;
- Orientado à sessão. Uma das formas para acessar o dispositivo é através da conexão à porta de console, geralmente uma porta serial, estabelecendo uma sessão informando usuário e senha;
- Sempre disponível. Quando acessado via porta serial de console a sinalização é fora-da-banda. Mesmo que a comunicação dentro-da-banda esteja indisponível, é possível acessar o dispositivo;
- Relativamente simples de se implementar. A máquina de estado é bastante reduzida. Basta verificar a sintaxe do comando e se os parâmetros estão corretos, um de cada vez;
- Modelo de transação orientado a comandos. No SNMP o modelo de transação é orientado a dados. Na CLI, cada comando é analisado pelo *parser* e encaminhado, se estiver correto. Uma seqüência de comandos define um arquivo de configuração. Todos os fabricantes aceitam receber este arquivo de configuração;
- Ferramentas de script. Existe uma vasta coleção de ferramentas de script (PERL, EXPECT), permitindo a automação da tarefa de gerência de configuração.

2.2.1. Deficiências da CLI

No entanto, a CLI apresenta algumas deficiências:

- Falta de um modelo de dados consistente;
- Coerência no processo de controle de mudanças. Ao se mudar a versão do software do dispositivo, é possível que a CLI tenha sido modificada;
- Falta de uma documentação estruturada;
- Falta de consistência para os códigos de erro;
- CLIs diferem entre os fabricantes;
- Tendência de ser voltada para a leitura humana. Dificulta a configuração do dispositivo de forma programática.

As deficiências que mais comprometem a CLI quanto à configuração de dispositivos são a primeira e as duas últimas. A primeira e a última afetam a automatização do processo de configuração. A penúltima deficiência gera a necessidade do operador conhecer mais de uma CLI. Ou seja, ele deve aprender como fazer a mesma coisa, de maneiras diferentes. Este profissional acaba necessitando de atualizações contínuas, via treinamento e dedicação, tornando este profissional caro.

2.3. SNMP

O protocolo SNMP foi desenvolvido para permitir que dispositivos de rede que utilizam o protocolo IP possam ser gerenciados remotamente, através de um conjunto de operações simples. É o protocolo mais utilizado para monitoramento e gerência de desempenho.

O SNMP utiliza o modelo gerente-agente (*manager-agent*) onde um servidor ou gerente (*manager*) com a função de *Network Management System* (NMS) comunica-se com o agente de Gerência de Rede (*agent*, instalado no dispositivo a ser gerenciado) através do protocolo de gerência.

O *manager* é responsável por *polling* (busca de informação no agente) ou por recebimento de *Traps* (enviada pelo agente, sem necessidade de solicitação prévia, para informar alterações de estado).

O *agent* é um software que roda no dispositivo gerenciado, podendo ser um programa separado (um "*daemon*", em um servidor Unix) ou incorporado (por exemplo, no Cisco IOS - *Internetwork Operating System* [40]), para prover informações ao NMS. A Figura 2.1. mostra as mensagens trocadas pelo protocolo SNMP no paradigma gerente-agente.

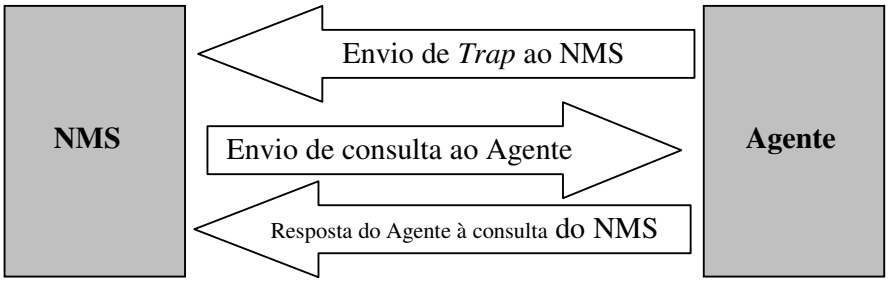


Figura 2.1: Troca de mensagens do protocolo SNMP.

2.3.1. Funcionamento do Protocolo

O protocolo SNMP utiliza-se do UDP como protocolo de transporte na comunicação entre NMS e agente. Um dos motivos pela escolha de um protocolo que não oferece garantia na comunicação é o fato do UDP ter menos *overhead*, reduzindo assim o impacto do sistema de gerência no desempenho da rede. Portas:

- UDP porta 161 – para envio e recebimento de requisições;
- UDP porta 162 – para receber *traps* dos elementos gerenciados.

As portas UDP utilizadas no protocolo SNMP são mostradas na Figura 2.2.

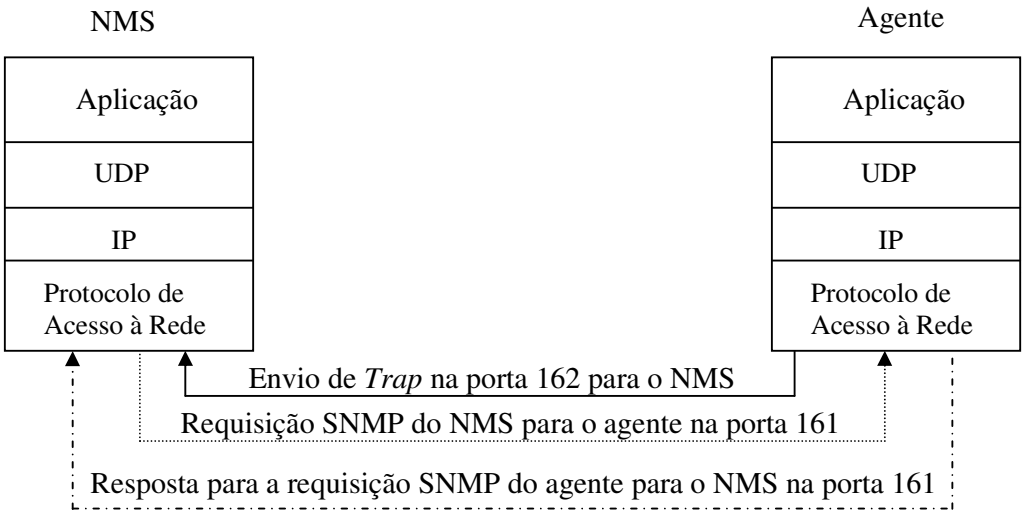


Figura 2.2: Funcionamento do protocolo SNMP.

SNMPv1 e SNMPv2 utilizam a *string* denominada como *community* para o estabelecimento de comunicação confiável entre NMS e agente. O agente pode ser configurado com três tipos de *community*: *read-only, read-write e trap*. A *community read-only* permite apenas leitura de dados no agente, a *read-write* permite leitura e modificação de dados e valores no agente, e *trap* permite envio de informações do agente para o NMS de forma assíncrona.

Uma das chaves para o entendimento do SNMP é a *Structure of Management Information* (SMI [41]), subconjunto da *Abstract Syntax Notation One* (ASN.1 [42]). O ASN.1 especifica como os dados são representados dentro do contexto do protocolo SNMP, o que torna esta notação independente do tipo de máquina. O SMI define precisamente como os objetos gerenciados são denominados e especifica a qual tipo de dados eles estão associados. Esta definição pode ser resumida em três atributos:

1. **Name**

O nome ou *Object Identifier* (OID) define um objeto como único. Pode aparecer na forma numérica ou amigável (*human-readable*). Em ambos os casos os nomes são grandes e inconvenientes. Assim, as aplicações SNMP buscam facilitar a navegação através destes nomes de forma simplificada.

A notação utilizada para definir um objeto é organizada em uma hierarquia em árvore (tal como a utilizada pelos servidores de nomes – *Domain Name Systems* - DNS), onde o nome do objeto é formado pelo nome do caminho percorrido do nó principal até o ponto desejado, sendo estes separados por ".". O nó principal denominado como "*root*" ou "*root-node*" não aparece no nome final, diz-se que este é definido como " ".

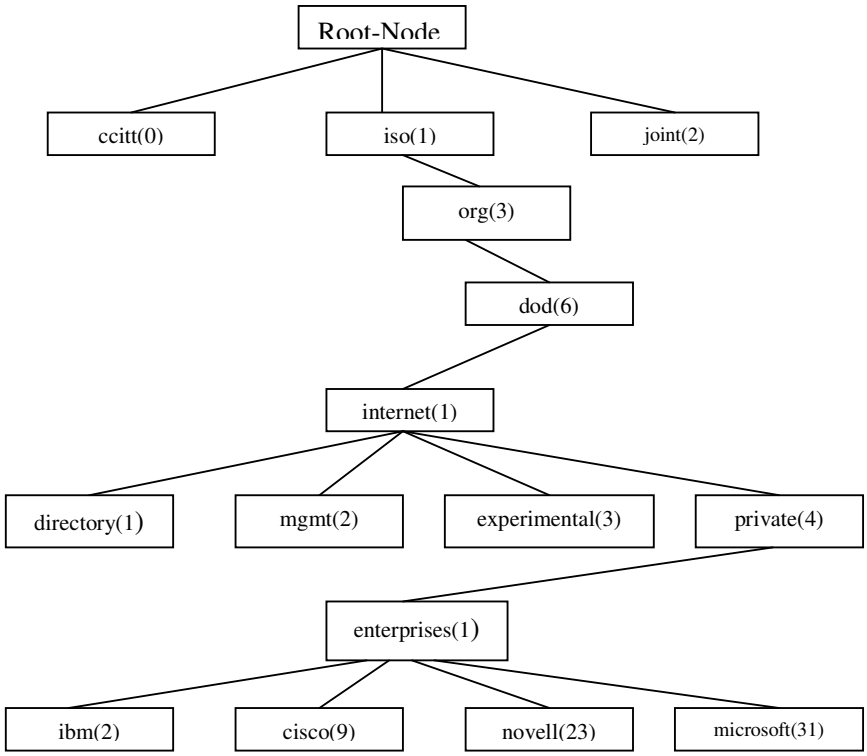


Figura 2.3: Hierarquia de nomes em árvores SMI.

Assim, na Figura 2.3,por exemplo, percorrendo a árvore iso(1), org(3), dod(6), internet(1), private(4), enterprise(1), cisco(9) temos a representação deste OID na forma 1.3.6.1.4.1.9 ou iso.org.dod.internet.private.enterprise.cisco.

2. Type ou Syntax

Os seguintes tipos de dados são definidos:

- INTEGER: Um contador numérico de 32 bits. Pode representar, por exemplo, o estado de uma interface de um roteador (*up* (1), *down* (2) ou *testing* (3)). O valor zero (0) não deve ser usado de acordo com a *Request for Comments* (RFC) 1155;
- OCTET STRING: Uma string de zero ou mais octetos, geralmente usada para representar strings de textos ou endereços físicos;

- COUNTER: Um contador com valor mínimo de 0 e máximo de $2^{32} - 1$ (4.294.967.295). Quando um roteador é ligado, ou reiniciado, este valor começa com 0; quando o valor máximo é alcançado, a contagem é reiniciada com valor 0. Pode ser usado para representar, por exemplo, octetos enviados e recebidos em uma interface;
- OBJECT IDENTIFIER: Uma *string* decimal separada por pontos e que representa um objeto gerenciado na árvore de objetos. Por exemplo, a Cisco é 1.3.6.1.4.1.9;
- NULL: Atualmente não utilizado pelo SNMP;
- SEQUENCE: Define listas que contêm 0 ou mais tipos de dados ASN.1;
- SEQUENCE OF: Define tabelas que são objetos gerenciados produzidos de um tipo SEQUENCE do ASN.1;
- IPADDRESS: Representa um endereço *Internet Protocol version 4* (Ipv4) de 32 bits;
- NETWORKADDRESS: O mesmo do IpAddress, mas pode representar diferentes endereços de redes;
- GAUGE: Um contador com valor mínimo de 0 e máximo de $2^{32} - 1$ (4.294.967.295). Diferentemente do COUNTER, ele pode incrementar ou decrementar, mas nunca pode exceder o valor máximo. A velocidade de uma interface de um roteador é medida pelo GAUGE;
- TIMETICKS: Um contador com valor mínimo de 0 e máximo de $2^{32} - 1$ (4.294.967.295). Usado para medidas de tempo em centenas de segundos. O tempo de *UpTime* de um roteador é medido com este valor;
- OPAQUE: Permite qualquer notação ASN.1 codificada como OCTET STRING;

3. Enconding

Uma instância de um objeto gerenciado é codificado em uma string de octetos usando *Basic Enconding Rules* (BER). BER define como os objetos são codificados e decodificados para permitir sua transmissão em um meio físico qualquer.

2.3.2. SMIv2

Amplia a árvore de objetos SMI adicionando o braço SNMPv2 à sub-árvore Internet, conforme mostrado na Figura 2.4. Adiciona novos tipos de dados e modifica a definição de um objeto adicionando novos campos, dando mais controle sobre a forma como um objeto é acessado. Permite aumentar uma tabela adicionando mais colunas e oferece melhores descrições.

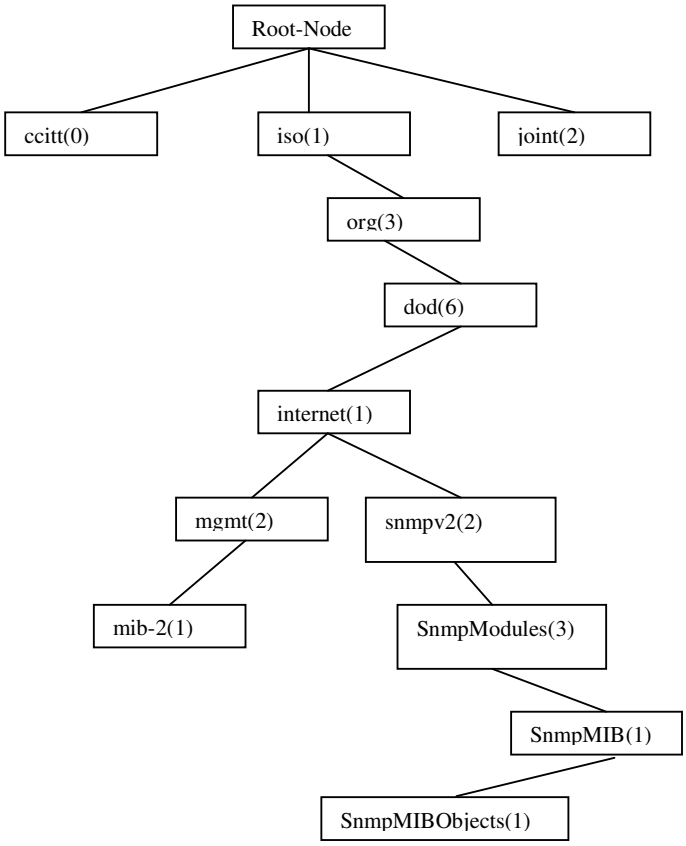


Figura 2.4: Hierarquia de nome em árvore SMIv2.

Os seguintes tipos de dados são definidos:

- INTEGER32: Valores númericos inteiros de -2^{31} a $2^{31} - 1$.
- COUNTER32: O mesmo valor numérico de COUNTER;

- GAUGE32: O mesmo valor numérico de GAUGE;
- UNSIGNED32: Representa valores decimais entre 0 e 2^{32} –1 (4.294.967.295);
- COUNTER64: Similar ao COUNTER32, valor mínimo de 0 e máximo de 2^{64} -1 (18.446.744.073.709.551.615). É ideal onde COUNTER32 atinge seu valor máximo num curto intervalo de tempo;
- BITS: Uma enumeração não negativa chamada bits.

MIB-II (*Management Information Base II* [43])

Um dos principais grupos de gerenciamento é a MIB-II cujo OID é o 1.3.6.1.2.1 ou iso.org.dod.internet.mgmt.mib-2, conforme Figura 2.5.

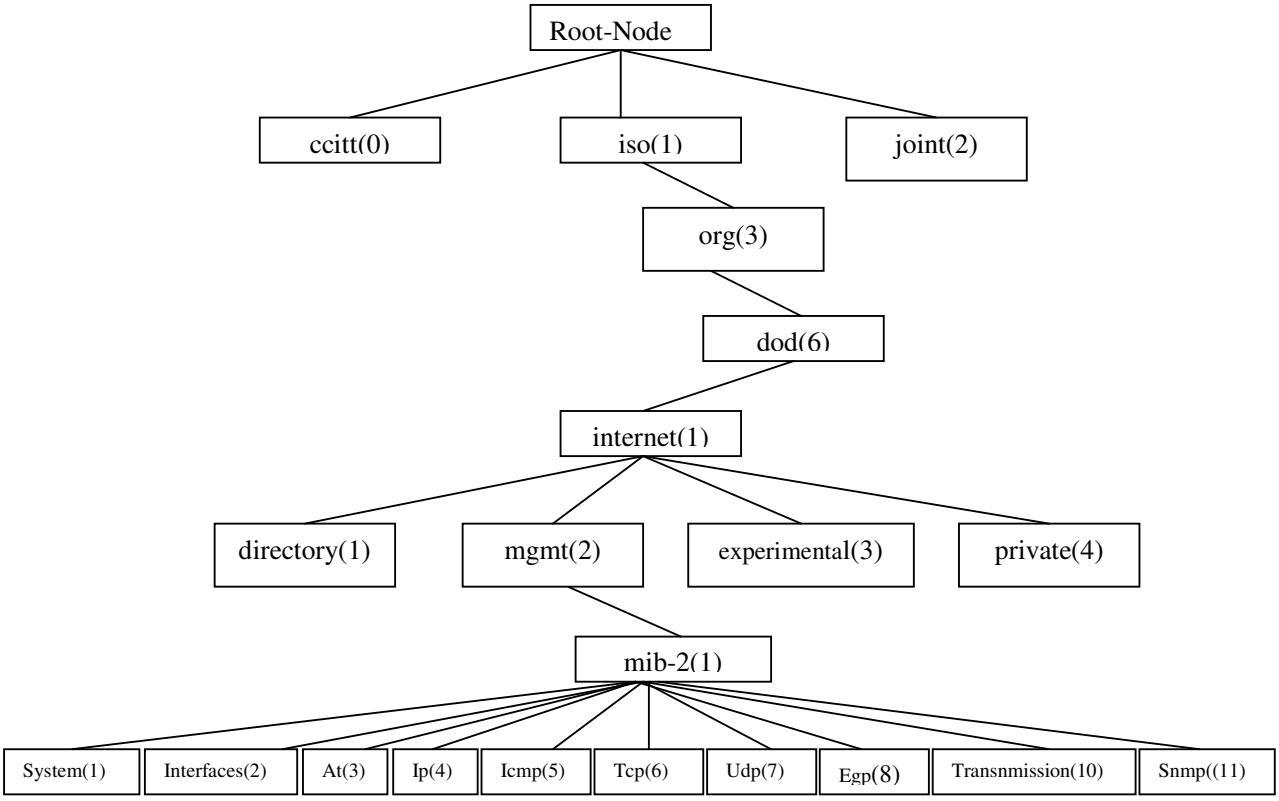


Figura 2.5: MIB-II.

As seguintes sub-árvores, com seus respectivos OIDs, são definidos:

- *System*: 1.3.6.1.2.1.1. Define uma lista de objetos que pertencem ao sistema operacional, como *UpTime* do sistema, contato do sistema e nome do sistema;
- *Interfaces*: 1.3.6.1.2.1.2. Mantém históricos de status de cada interface da entidade gerenciada, como status de *up* ou *down*, octetos enviados e recebidos, erros, descartes, etc;
- At: 1.3.6.1.2.1.3. O grupo *address translation* (at) está obsoleto e existe para manter compatibilidade;
- Ip: 1.3.6.1.2.1.4. Mantém históricos de diversos aspectos referentes ao protocolo IP, inclusive roteamento;
- Icmp: 1.3.6.1.2.1.5. Trata aspectos como pacotes de erro *Internet Control Message Protocol* (ICMP), descartes, etc;
- Tcp: 1.3.6.1.2.1.6. Trata entre outras coisas do estado do TCP: *closed*, *listen*, *synSent*, etc;
- Udp: 1.3.6.1.2.1.7. Trata de estatísticas do UDP, datagramas recebidos enviados, etc;
- Egp: 1.3.6.1.2.1.8. Trata de várias estatísticas sobre *External Gateway Protocol* (EGP) e mantém uma tabela de vizinhança EGP;
- *Transmission*: 1.3.6.1.2.1.10. Atualmente não há nenhuma definição de objeto neste grupo, mas outras especificações de mídia são definidas usando esta sub-árvore;
- Snmp: 1.3.6.1.2.1.11. Medição do desempenho SNMP na entidade gerenciada, tratando informações como número de pacotes SNMP recebidos e enviados.

2.3.3. Operações SNMP

O envio e recebimento de mensagens entre NMS e agentes é feito no formato de *Protocol Data Unit* (PDU), existindo um formato padrão de PDU para cada operação SNMP.

GetRequest

A requisição *GetRequest* é iniciada pelo NMS e é enviada ao agente, contendo a variável procurada - correspondente ao OID desejado, Figura 2.6.

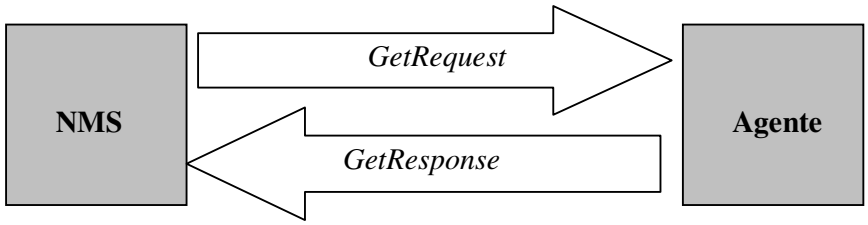


Figura 2.6: SNMP – Operação *GetRequest*.

GetNextRequest

Esta operação permite obter mais de um valor de uma determinada MIB. O funcionamento do *GetNextRequest* baseia-se em percorrer a árvore SMI do *Root-Node* até encontrar o OID desejado, enviando um *GetNextRequest* a cada *GetResponse*. Após localizar o OID desejado ele continua enviando requisições *GetNextRequest* até receber um erro, neste ponto ele finaliza a busca. Trata-se de uma operação útil para percorrer tabelas que variam dinamicamente (ex.: tabela de roteamento).

***GetBulkRequest* (SNMPv2 e SNMPv3)**

Permite requisitar mais de uma seleção de OID de uma única vez, e sua resposta vai depender da capacidade do agente. Se o agente não pode enviar todas as respostas requisitadas ele retorna uma mensagem de erro sem dados, Figura 2.7.

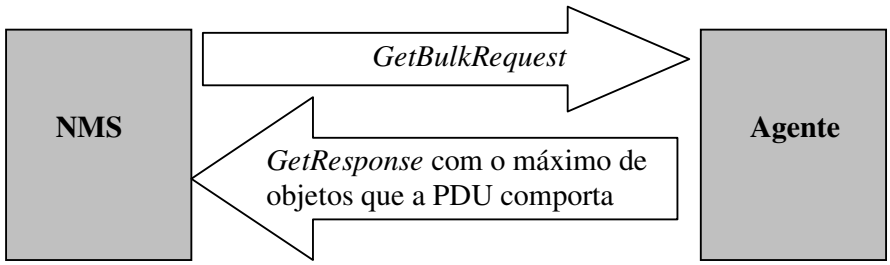


Figura 2.7: SNMP – Operação *GetBulkRequest*.

Set

É utilizado para mudar o valor de um objeto gerenciado ou criar uma nova coluna em uma tabela (mecanismo *RowStatus*, variável *CreateAndGo* e várias operações *set* em sequência). Precisa que a *community* seja *read-write ou write-only*. Mais de um objeto pode ser configurado por vez, Figura 2.8.

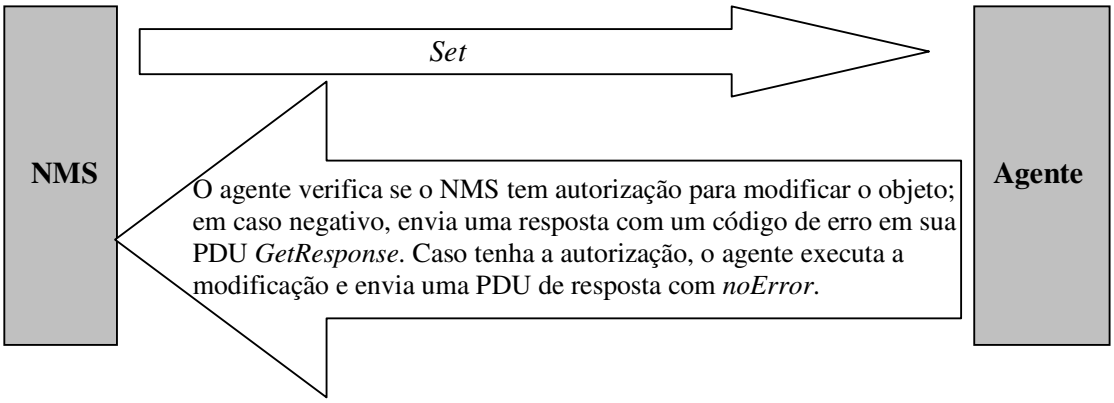


Figura 2.8: SNMP - Operação *SetRequest*.

GetResponse

Refere-se à PDU enviada pelo agente ao NMS em resposta a alguma requisição.

Trap

É utilizado pelo agente para envio de informações não solicitadas. De acordo com a configuração do agente, ele passa a enviar *Traps* para eventos como mudança de estado da interface (*up* ou *down*), etc.

As *Traps* definidas são:

- *Coldstart*: (0). Indica que o agente foi reiniciado e, portanto, todas as variáveis gerenciadas são reiniciadas; no caso das variáveis COUNTER ou GAUGE, os respectivos valores são reiniciados como 0 (zero);
- *WarmStart*: (1). Indica que o agente reiniciou a si próprio. Nenhuma variável é reiniciada;
- *LinkDown*: (2). Enviado quando uma interface muda de estado para *down*; a primeira variável da PDU *trap* identifica a interface em questão;
- *LinkUp*: (3). Enviado quando uma interface muda de estado para *up*; a primeira variável da PDU *trap* identifica a interface em questão;
- *AuthenticationFailure*: (4). Indica quando alguma requisição SNMP é feita com uma *community* incorreta;
- *EgpNeighborLoss*: (5). Identifica quando um vizinho EGP muda de estado para *down*;
- *EnterpriseSpecific*: (6). Identifica que o *trap* é específico de determinado fabricante.

***Notification* (SNMPv2 e SNMPv3)**

Resultado do esforço para padronizar o formato da PDU do *Trap* SNMPv1. Assim, a definição de *NOTIFICATION-TYPE* pelo SNMPv2 faz com que o formato da PDU do *Trap* seja idêntico ao *GetRequest* ou *SetRequest*.

***Report* (SNMPv2 e SNMPv3)**

Definido pelo SNMPv2, porém nunca implementado. A intenção é permitir que mecanismos SNMP interoperem.

2.3.4. Formato das Mensagens

Uma mensagem SNMPv1 consiste em um cabeçalho (*Message Header*) e uma PDU, Figura 2.9:

Message Header	PDU Type	PDU

Figura 2.9: Formato da mensagem SNMPv1.

Sendo que o cabeçalho contém dois campos:

- ***Version number*** – especifica a versão do protocolo;
- ***Community name*** – define a *string* de acesso do NMS.

PDU Type – especifica o tipo da PDU transmitida;

O formato da PDU, Figura 2.10:

Request ID	Error Status	Error Index	Object 1 Value 1	Object 2 Value 2	Object X Value X

Variable bindings

Figura 2.10: Formato PDU SNMPv1.

Sendo:

- *Request ID* – Associa a requisição SNMP com a resposta;
- *Error status* – Identifica um dos números de erro e o tipo, apenas na resposta;
- *Error index* – Associa um erro com um objeto em particular;

- *Variable bindings* – Representa o campo de dados da PDU SNMPv1, com o objeto e seu valor.

Uma mensagem *Trap* SNMPv1 consiste em 8 campos, Figura 2.11:

Enterprise	Agent Address	Generic TrapType	Specific Trap Code	Time Stamp	Object 1 Value1	Object 2 Value 2	Object X Value X
					←——*Variable bindings*——→		

Figura 2.11: Formato mensagem *Trap* SNMPv1.

Sendo:

- *Enterprise* – Identifica o tipo do objeto gerenciado que gerou o *trap*;
- *Agent address* – Provê o endereço do objeto gerenciado que gerou o *trap*;
- *Generic trap type* – Indica um dos números de tipo de traps genéricos;
- *Specific trap code* - Indica um dos números de código de traps específicos;
- *Time stamp* – Insere o valor do tempo referente à ocorrência do evento;
- *Variable bindings* - Representa o campo de dados da PDU SNMPv1, com o objeto e seu valor.

Assim como no SNMPv1, uma mensagem SNMPv2 consiste em um cabeçalho e uma PDU. O formato da PDU SNMPv2 é igual *para Get, GetNext, Inform, Response, Set e Trap*. A operação *Inform* é definida no SNMPv2 para possibilitar a comunicação entre entidades gerentes, onde um dos gerentes é remoto com relação ao agente monitorado.

Uma mensagem *GetBulk* SNMPv2 consiste em 7 campos, Figura 2.12:

Request ID	Non Repeaters	Max Repetitions	Object 1 Value 1	Object 2 Value 2	Object X Value X
			←—— V*ariable bindings* ——→		

Figura 2.12: Formato mensagem operação *GetBulk* SNMPv2.

Sendo:

- *Request ID* – Associa a requisição SNMP com a resposta;

- *Non repeaters* – Especifica o número de instâncias no campo *variable bindings* que deve ser recuperada não mais do que uma vez a partir do começo do pedido. É usado quando uma das instâncias do objeto é do tipo escalar;
- *Max repetitions* – Define o número máximo de vezes que outras variáveis, além do que foi especificado no campo *Non repeaters*, devem ser recuperadas;
- *Variable bindings* - Representa o campo de dados da PDU SNMPv2, com o objeto e seu valor.

2.3.5. SNMP versão 1

O SNMP versão 1 é definido em quatro RFCs e é totalmente padronizado pelo IETF:

- RFC 1155 - define a SMI. Ou seja, os mecanismos usados para descrever e nomear os objetos que serão gerenciados;
- RFC 1156 – define as MIBs e as definições dos objetos;
- RFC 1157 - define o protocolo SNMP.
- RFC 1212 - define um mecanismo de descrição mais conciso, mas é inteiramente consistente ao SMI;

A segurança não é o forte desta versão que baseia-se em "*community strings*", que nada mais são que simples *passwords*, *strings* em formato texto aberto, que permitem que qualquer ferramenta de gerência que conheça esta *string* obtenha acesso aos dados deste dispositivo.

2.3.6. SNMP versão 2

O SNMP versão 2 também denominado SNMPv2c (*community string-based* SNMPv2) é definido pelas RFCs 1905, 1906 e 1907 pelo IETF. Esta versão busca implementar e corrigir algumas deficiências da versão anterior, como: adicionar mais segurança, novas operações, comunicação entre servidores com a função de *manager* e configuração remota via SNMP.

Para facilitar a coexistência com a versão 1 são definidas algumas práticas. Estas práticas se resumem em duas categorias:

- Informação de gerência – Os módulos definidos a partir de SNMPv1 SMI devem ser entendidos pelo protocolo SNMPv2. Para isto é definido que o SNMPv2 SMI é um superconjunto do SNMPv1 SMI;
- Operações do protocolo – Um agente SNMPv1 pode ser gerenciado por um gerente SNMPv2. Para isso deve ser fornecido um *proxy*. Outra maneira é através de um gerente bilíngüe que "fale" SNMPv1 e SNMPv2.

Uma grande controvérsia desta versão foi a definição do mecanismo *RowStatus.* Esta convenção textual foi definida para permitir a criação e remoção de linhas em tabelas conceituais. De acordo com a RFC 1902, tabelas conceituais são estruturas de tabela de uma coleção de objetos ordenados dentro de uma MIB. Por definição, o protocolo SNMP não permite manipular tabelas ou linhas, apenas itens individuais. Seu principal uso é por desenvolvedores de aplicações de gerência que necessitam de estruturas aumentadas, uma vez que tabelas conceituais são formadas com linhas conceituais base e linhas conceituais de extensão. Assim, é possível aumentar o número de colunas na tabela sem reescrever a sua definição.

Os principais métodos de criação de linhas são: *CreateAndWait* e *CreateAndGo*. Outros métodos são: *Active*, *notInService*, *notReady* e *Destroy*.

- *CreateAndWait* – para a criação de uma nova instância de uma linha conceitual. O dispositivo gerenciado não tem acesso imediato;
- *CreateAndGo* – para a criação de uma nova instância de uma linha conceitual. O dispositivo gerenciado tem acesso imediato à ela.

Um dos usos do mecanismo *RowStatus* foi para simular outras operações além das já existentes. No entanto, o algoritmo do mecanismo é muito complexo. Por isso, os agentes não suportam todas as operações definidas. O seu uso, por exemplo, com objetos agregados para configuração é bastante complicado.

A versão 2 do SNMP não teve ampla aceitação, pois apesar de implantar algumas correções do SNMPv1, falhou em não corrigir as deficiências nas questões de segurança.

2.3.7. SNMP versão 3

O SNMP versão 3 é um padrão proposto pelas RFCs 2570, 2571, 2572, 2573, 2574 e 2575.

Novas funcionalidades foram adicionadas ao SNMPv3. São elas:

- Segurança
  - Autenticação e privacidade;
  - Autorização e controle de acesso.
- Modelo administrativo
  - Nomeação das entidades;
  - Gerência das chaves;
  - Notificação dos destinos;
  - Relação dos *proxies*;
  - Configuração remota através de operadores SNMP.

A co-existência do SNMPv1, SNMPv2 e SNMPv3 está descrita na RFC 2576.

A ênfase do SNMPv3 foi resolver os problemas de segurança que ficaram em aberto na versão 2. No entanto, o protocolo ainda se encontra em fase de evolução. Principalmente porque a PDU SNMP foi alterada para comportar dados de segurança, tornando sua implementação muito mais complexa que as anteriores.

2.3.8. Deficiências do SNMP para Configuração

Embora o SNMP seja usado para configuração de algumas características e plataformas, ele não é amplamente usado para a configuração de dispositivos de rede. A primeira opção é a interface de linha de comando (CLI). Uma das razões para isto é a questão de segurança. No entanto, embora importante, é apenas um dos pré-requisitos que definem se uma tecnologia vai ser adotada ou não para a tarefa de Gerência de Redes e, mais especificamente, de configuração. Interessante notar que a maioria das CLIs, antes da adoção do protocolo SSH, utilizavam TELNET como protocolo de aplicação, e as senhas eram passadas pela rede na

forma de texto puro. Da mesma maneira, as *strings* da comunidade SNMP também são passadas como texto puro, exceto para o SNMPv3. Infelizmente para o SNMPv3, o protocolo foi completado depois que o SSH já tinha sido adotado pelos operadores de rede.

No capítulo de Introdução apresentamos algumas deficiências do SNMP. Além daquelas deficiências, a seguir são discutidas outras deficiências do SNMP com ênfase nos aspectos relacioanados à Gerência de configuração.

Operação SET do SNMP não é tão simples

Ao contrário do que o nome do protocolo diz, desenvolver e manter um agente SNMP não é simples, comparado com CLI. A primeira razão é que a máquina de estados do protocolo para a operação *SET* é muito mais complicada do que para CLI. O protocolo permite, arbitrariamente, transações complexas, na qual múltiplas linhas não relacionadas e parciais podem ser passadas para o agente. Espera-se que o agente complete as transações da maneira "ou todas ou nenhuma" e de forma simultânea. Ao invés do comando *GetRequest*, por exemplo, ser a unidade básica da transação, cada parâmetro para o comando (um dos vários OIDs que podem ser informados) é tratado como uma unidade básica de transação. Esta complexa máquina de estados, sem garantias, é muito mais difícil de projetar e testar comparativamente com o código para uma CLI que realiza a mesma tarefa.

Outro ponto negativo, é que não existe distinção no protocolo entre as operações de criação (*create*), edição (*edit*), e remoção (*delete*). Ao invés disto, estas operações do protocolo são codificadas como dados, representados pelo objeto MIB *RowStatus* (Seção 2.2.2.2), cada uma em uma linha de uma tabela configurável.

Há outras razões para que o desenvolvimento de aplicações SNMP seja mais complicado do que ferramentas de *script* para CLI, como a falta de elementos de alto nível do procedimento para configuração. MIBs não especificam consistentemente a ordem de criação, ou qualquer outro requisito operacional de alto nível, para completar uma dada tarefa funcional. As CLIs e suas documentações têm a tendência de ser muito mais orientada a tarefas e, dos dispositivos, espera-se o suporte a um procedimento específico para completar uma dada tarefa.

Definições dos dados da gerência

MIBs são definidas através do modelo especificado pelo SMI. O SMI é parcialmente responsável pelo alto custo de desenvolvimento do SNMP. O único mecanismo para o reuso de dados no SMI é o refinamento dos tipos básicos. A capacidade de definição para as estruturas de dados é limitada a simples *arrays* de tipos de dados escalares. Para se criar estruturas de dados complexas, é necessário associar *arrays* de objetos escalares, conectados por componentes de índices comuns. O resultado são estruturas de dados que são difíceis de ler e de implementar. Isto é especialmente prejudicial para a configuração, porque a complexidade adicional das múltiplas tabelas para escrita causa dependências da ordem da criação.

Protocolo de Transporte

Mensagens SNMP são transportadas sob UDP. Como se trata de protocolo sem conexão, limitou-se o tamanho máximo da mensagem para um valor pequeno. Isto gerou impacto no projeto das MIBs e das aplicações. A operação *CreateAndWait* para *RowStatus* existe para contornar esta limitação. Permite que operações de criação de configuração sejam divididas em duas ou mais transações. As próprias MIBs devem ser projetadas a partir desta restrição. Objetos individuais ficam restritos a uma única instância de 484 octetos de carga útil (*payload*), que é o tamanho da PDU do SNMP.

O UDP não garante que os pacotes serão entregues em ordem, ou que apenas uma única cópia do pacote será entregue. O que torna o projeto das MIBs, aplicações e agentes mais complexos.

O desenvolvimento da aplicação se torna mais difícil porque, para recuperar um objeto grande como, por exemplo, uma tabela de roteamento completa, são necessárias várias transações com a operação *GetNext* ou *GetBulk*. Na camada de aplicação isto acarreta em lidar com a fragmentação e remontagem dos pacotes, o que é bastante caro. Uma opção seria o uso do TCP, que é orientado a conexão, e o projeto da aplicação poderia ser simplificado.

2.4. Tecnologias *Web* na Gerência de redes

A seguir são apresentadas as principais tecnologias *Web* para a Gerência de Redes discutidas recentemente na literatura. A principal preocupação é corrigir os problemas apresentados pelo CLI e SNMP.

2.4.1 HTTP

O gerenciamento baseado em HTTP consiste em usar o protocolo HTTP, HTTP/1.0 ou HTTP/1.1, ao invés de uma das três versões do SNMP para transferir dados de gerência entre os agentes e os gerentes. A partir de 1994, tornou-se comum o dispositivo de rede ter um servidor HTTP embutido.

A forma mais simples de gerência baseada em HTTP consiste em páginas Web escritas em HTML. O gerente acessa uma página HTML do agente a partir de um *Web-Browser*. Esta página pode ser estática ou dinâmica. Páginas estáticas não mudam, como o próprio nome diz, e são armazenadas no agente. O seu principal uso é para gerência de configuração. Páginas dinâmicas refletem, principalmente, o estado do agente. Por isso, são ideais para mostrar dados de desempenho.

Outra forma de uso do HTTP é a partir de uma aplicação do lado do gerente, escrita numa linguagem de programação. Ou seja, do lado do gerente não é necessário o uso de um *Web-Browser*. Uma linguagem bastante utilizada é o Java. A comunicação entre este gerente e agente se dá via HTTP. Uma vantagem é que dentro das mensagens HTTP os dados podem ser codificados em XML, HTML ou strings.

Importante salientar que o uso do HTTP reflete apenas a mudança do protocolo de comunicação entre as entidades gerente/agente. O modelo de gerência não muda.

A grande vantagem do HTTP

A grande vantagem do HTTP é que a sua porta TCP 80 é conhecida por todo admistrador de rede e geralmente passa sem restrições pelos *firewalls* e *proxies.*

Deficiências do HTTP para configuração

É possível fazer gerência de configuração com o HTTP. É uma forma bastante simples e específica, mas com inconveniências. Nesta ferramenta, o operador utiliza formulários HTML específicos de configuração de um servidor de configuração. Após preencher o formulário é gerado automaticamente um arquivo de configuração. Este arquivo é transferido para um servidor que armazena todas as configurações dos dispositivos. Por último, o servidor carrega o arquivo no agente. Após o *reboot* do dispositivo, ele está configurado com os parâmetros definidos anteriormente no formulário. O passo final pode ser variado. Nesta outra forma, o próprio agente carrega o arquivo, ao ser reinicializado automaticamente ou manualmente pelo operador.

Os inconvenientes se referem à manutenção do SNMP e CLI para gerência, com a manutenção de suas deficiências, como visto nas seções anteriores. Uma vez que o HTTP, como dito anteriormente, é utilizado, como é de se esperar, apenas como protocolo de transporte entre o gerente e o agente, sem mudar o modelo de gerência; a impossibilidade de se configurar os dispositivos de uma maneira interativa, já que se fica amarrado aos formulários HTML disponibilizados pelo servidor de configuração; e a presença de um intermediário entre o gerente e o agente, podendo ser tornar um ponto de gargalo.

2.4.2 XML

XML é uma meta-linguagem de marcação padronizada pelo *World Wide Web Consortium* (W3C [44]) para troca de documentos na Web. É uma versão reduzida do *Standard Generalized Markup Language* (SGML [45]). Sua intenção é ser simples de transmitir, como

o HTML no HTTP, sendo ao mesmo tempo fácil de estender como SGML. É acompanhado por uma série de padrões:

- DTD (*Document Type Definition*): utilizado para a definição da estrutura de informação de maneira flexível;
- XML Schema: Mesmo objetivo do DTD, contudo, fácil de entender pelo ser humano e permite definições de dado mais ricas. É a forma mais comum;
- APIs Document Objetct Modelo (DOM) e Simple API for XML (SAX): para facilitar a manipulação dos dados a partir das aplicações;
- XPATH: define expressões para endereçar os objetos dentro do documento;
- XSLT (*Extensible StyleSheet Language Transformations*): Para transformação do XML em outros textos formatados.

Vantagens do XML

- XML é auto-descritivo: pode ser lido tanto por máquinas como por seres humanos;
- XML é independente de plataforma, tornando soluções baseadas em XML muito flexíveis;
- XML é uma linguagem de descrição de uso geral que pode codificar qualquer tipo de informação, como dados e tipos de dados.

Vantagens do XML para Gerência de Redes

O uso do XML para Gerência de Redes traz algumas vantagens importantes:

- Facilita a modelagem da informação de gerência. É possível, por exemplo, converter uma MIB especificada em SMI, para uma MIB em XML e vice-versa;
- Pode ser transmitido utilizando o protocolo HTTP disponível em quase todos dispositivos de rede;
- Possibilita a compatibilidade entre qualquer hardware ou plataforma de software que suporte HTTP;

- Possui uma lista grande de ferramentas de desenvolvimento, muitas delas disponíveis de forma gratuita;
- Não possui restrição quanto ao tamanho do documento.

2.4.3. WBEM

O WBEM é uma iniciativa do DMTF que inclui um conjunto de tecnologias para possibilitar a gerência de softwares e hardwares. Consiste em um modelo comum de informação (CIM), um DTD para representar o CIM em XML e uma especificação para a transmissão do CIM no HTTP. Provê uma forma de compartilhamento de dados independente do fabricante, protocolo, sistema operacional ou padrão de gerenciamento.

CIM é um modelo de informação orientado a objetos na forma de uma hierarquia de classes e associações. Provê um *framework* conceitual para classificar e definir partes de um ambiente de rede e descrever como são integrados. O modelo captura noções que são aplicáveis a todas as áreas de gerência, independentemente da tecnologia de implementação.

Deficiências do WBEM para Gerência

A principal deficiência do WBEM é a falta de definição de um modelo organizacional. Este modelo define como devem ser as interações entre o gerente e o agente. Um exemplo é o modelo gerente/agente do SNMP. Assim, cada desenvolvedor do protocolo acaba tomando suas próprias decisões de organização na forma de tentativa e erro. A interoperabilidade entre sistemas de gerência baseados em WBEM fica prejudicada.

O CIM é muito detalhado, mas também muito complexo. Entender o modelo de informação, mesmo com o auxílio das suas definições em modo gráfico, via UML, não é uma tarefa fácil.

A sua especificação baseia-se na filosofia *top-down*. Para se tornar útil, necessita de extensões concretas ao protocolo.

2.4.4. Gateway SNMP-XML

Constatada as deficiências do SNMP para Gerência de Redes, um grande esforço foi empreendido no sentido de aplicar as tecnologias Web ao gerenciamento de redes, notadamente o HTTP e o XML.

As primeiras propostas foram no sentido de traduzir as MIBs. Um dos primeiros trabalhos foi proposto por Martin Flatin[7]. Ele mostrou a tradução de alguns grupos da MIB-II para XML. Posteriormente, Frank Strauss desenvolveu o libsmi [46], uma biblioteca para tradução de MIBs SNMP para outras linguagens, como Java e XML.

Um trabalho importante foi publicado e proposto pelo grupo POSTECH da Korea em 2003 [4]. Inicialmente eles identificaram que as propostas de traduções anteriores eram deficientes, então propuseram um algoritmo para a tradução de qualquer MIB e que entendesse os módulos presentes. Em seguida, eles propuseram um modelo de interação, mapeando cada operação em HTTP. Por último, definiram uma arquitetura de *gateway* SNMP-XML.

No mesmo ano de 2003, um outro trabalho deste grupo também foi publicado [47]. A sua contribuição é quanto à definição de uma arquitetura para sistemas de gerência baseada em XML, chamado genericamente de XNMS. Eles definiram um gerente baseado em XML, um agente baseado em XML e utilizaram o trabalho do *gateway* SNMP-XML. As tecnologias XML utilizadas foram o SAX para análise do XML e XPATH para endereçar objetos dentro do XML. O SAX substituiu o DOM que fora utilizado no trabalho anterior, uma vez que o DOM carrega todo o XML em memória para análise, consumindo mais memória que o SAX, que processa o XML de forma orientada a eventos.

Métricas foram realizadas comparando operações do sistema utilizando SNMP, XML puro (o agente entende XML) e o *gateway* SNMP-XML. Os resultados obtidos em XML puro apontam para uma melhora em termos de tráfego de rede comparado com o SNMP. Com relação ao *gateway,* ele gera um *overhead* de tráfego inerente ao trabalho de tradução. Entretanto, o tempo de processamento não é muito maior com relação ao processamento direto entre o gerente e o agente.

2.4.5. *Web Services*

Uma definição simples para o que é um *Web Services*: programas acessíveis via rede. No entanto, outras tecnologias já existem para este fim: CORBA, Microsoft DCOM (*Distributed Component Object Model*) [48], Sun JAVA RMI e outras. Contudo, estas tecnologias adotam um mecanismo fortemente acoplado para o modelo de comunicação síncrono – requisição/resposta. Alguns problemas desta característica são:

- Estas tecnologias são complexas e necessitam de um investimento pesado em infra-estrutura;
- Não passam por *firewalls*;
- Baixa interoperabilidade com plataformas de diferentes fabricantes da mesma tecnologia e quase nenhuma interoperabilidade com tecnologias diferentes;
- Uma integração entre plataformas de *business-to-business* é muito cara e complexa;

Por outro lado, diferentemente das tecnologias citadas acima, ao utilizar XML e protocolos de Internet – como HTTP, FTP SMTP - *Web services* são, naturalmente, um modelo de comunicação assíncrono fracamente acoplado. Isto facilita a interoperabilidade e permite o desenvolvimento de aplicações em qualquer linguagem de programação, utilizando protocolos abertos.

Uma definição mais completa é dada pelo W3C:

"Um *Web service* é um sistema de software projetado para suportar interações máquina-máquina interoperáveis sobre uma rede. Tem uma interface descrita em formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o *Web service* da maneira prescrita por sua descrição utilizando mensagens SOAP, tipicamente transportado utilizando HTTP com uma serialização XML em conjunção com outros padrões Web relacionados."

Da própria definição do W3C fica claro que outras tecnologias e padrões baseados em XML devem ser utilizado: Simple Object Access Protocol (SOAP), *Web Services Definition Language* (WSDL [49]) e *Universal Description, Discovery and Integration* (UDDI [50]).

Estas tecnologias formam os fundamentos dos *Web Services*. SOAP é o protocolo de mensagem para *Web Services*. WSDL é um documento XML que descreve as interfaces, o conjunto de mensagens SOAP e como as mensagens XML são trocadas. UDDI é um diretório onde se pode achar um serviço após o proprietário do serviço publicá-lo no UDDI. Todas estas tecnologias são padrões abertos.

Um *Web Services* é definido a partir de quatro modelos de arquitetura [23]: MOM (*Message-Oriented Model*) modelo orientado a mensagens; SOA (*Service-Oriented Architecture*) arquitetura orientada a serviços; ROM (*Resource-Oriented Model*) modelo orientado a recursos; e PM (*Policy Model*) modelo de políticas. O modelo mais utilizado para o desenvolvimento de aplicações é o SOA, que será descrito a seguir.

2.4.6. SOA

Uma boa maneira de entender um *Web Services* é a partir do modelo SOA. SOA é uma forma de arquitetura de sistema distribuído que define os seguintes papéis e operações, Figura 2.13:

Papéis:

- ***Service Broker/Provider***: O provedor de serviço é responsável por desenvolver e disponibilizar o *Web Services*. Ele também define os serviços e os publica no serviço de registro;
- ***Service Registry***: O registro lista os vários tipos de serviços, descrições e localidades. O *Service registry* é responsável por registrar o serviço e descobrir outros *Web services*. Ajuda os requisitantes de serviços a localizar e assinar o serviço requerido;
- ***Service Requestor***: O requisitante do serviço é responsável pela sua invocação.

Operações:

- **Publish**: O *Service Provider* deve publicar o *Web Service* no *Service Registry* para que um *Service Requestor* localize um *Web Service* e o acesse;
- **Find**: Na operação *find*, o *Service Requestor* obtém uma descrição do serviço inquirindo o *Service Registry*. A operação *find* pode ocorrer de duas maneiras: no desenvolvimento da aplicação cliente, de forma estática (*static binding*); ou em tempo de execução, de forma dinâmica (*dynamic binding*);
- **Bind**: Na operação *bind*, *o Service Requestor* invoca o serviço em tempo de execução usando os detalhes de ligação para localizar, contactar e invocar o serviço.

Os papéis e operações são integrados na Figura a seguir.

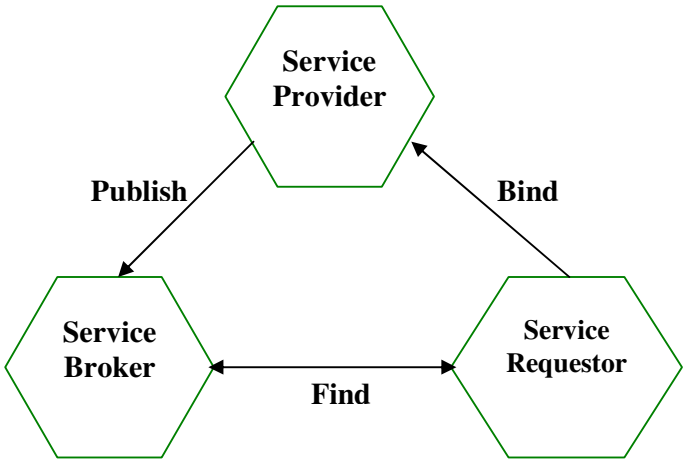


Figura 2.13: Papéis e operações arquitetura *SOA*.

2.4.7. SOAP

Quando o SOAP nasceu sua intenção era ser simples. Por isso, naquele momento, SOAP significava "Simple Object Access Protocol". A sua história começou com RPC e a necessidade de resolver a questão de interoperabilidade entre diferentes plataformas como CORBA e DCOM. Este protocolo foi proposto pela Microsoft em 1998. Em 1999 o W3C

liberou a versão 1.0. Em Junho de 2003 foi liberada a versão 1.2. Hoje SOAP é um acrônimo que dá nome a uma tecnologia de Internet.

SOAP é um protocolo leve para a troca de informação estruturada de forma descentralizada, num ambiente distribuído. SOAP utiliza técnicas XML para definir um *framework* de mensagens extensível sob vários protocolos de transporte. O *framework* foi projetado para ser independente de qualquer modelo de programação ou semântica de implementação.

Em um cenário básico, a mensagem SOAP é transmitida em uma única via entre o nó SOAP emissor e o nó SOAP receptor, conforme Figura 2.14 a seguir.

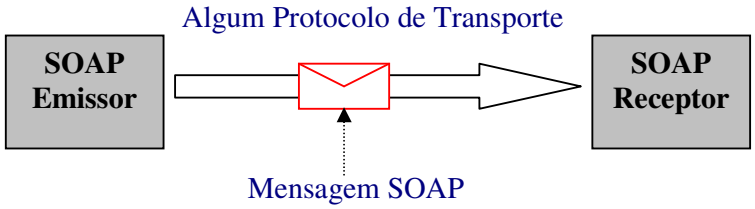


Figura 2.14: Cenário básico da mensagem SOAP.

O *framework* de mensagens define um conjunto de elementos XML para codificar todas as mensagens. Os principais elementos são: *Envelope* (mandatório), *Header* (opcional), *Body* (mandatório) and *Fault* (opcional) definido em HTTP://www.w3.org/2003/05/soap-envelope. A seguir uma representação da mensagem SOAP, Figura 2.15.

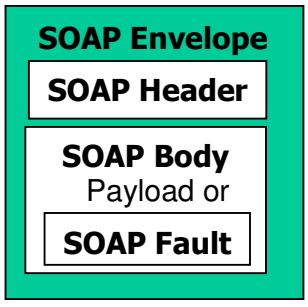


Figura 2.15: Representação da mensagem SOAP.

SOAP Envelope

O SOAP *envelope* é o elemento de nível superior de uma mensagem SOAP. É como um envelope de uma carta no sistema postal conforme ilustrado no exemplo a seguir.

```
<env:Envelope>
    <env:Header>
    …       //Optional
    </env:Header>
    <env:Body>
    …       //Mandatory
            <env:Fault>
            …       //Optional
            </env:Fault>
    </env:Body>
</env:Envelope>
```

SOAP *header*

O elemento SOAP *header* é opcional. É um mecanismo de extensão que provê os meios para passar informação em mensagens SOAP que não são dados (*payload*) como autenticação e gerenciamento de transação.

SOAP *body*

O elemento SOAP *body* é obrigatório dentro de um SOAP *envelope*. Dentro dele está inserida a principal informação fim-a-fim de uma mensagem SOAP. Uma mensagem SOAP pode conter mais que um SOAP *body*. Também pode conter um elemento SOAP *fault*.

SOAP *fault*

Toda falha específica do SOAP ou da aplicação usa o elemento simples, env:Fault, dentro de um elemento env:Body.

Chamadas de Procedimentos Remotos – RPC

A especificação 1.2 do SOAP provê uma forma de escrever métodos RPC como em qualquer outro sistema distribuído. Para invocar o RPC SOAP as seguintes informações são necessárias:

1 – O endereço do nó SOAP alvo;

2 – O procedimento ou nome do método;

3 – As identidades e os valores de qualquer parâmetro de entrada ou saída e o valor de retorno;

4 – Uma separação clara dos argumentos utilizados para identificar o recurso WEB que é o alvo do RPC;

5 – O padrão de troca de mensagens empregado para o RPC;

6 – Informações opcionais: dados que podem ser carregados como parte de blocos SOAP *header*.

A ligação SOAP RPC define como encapsular e representar esta informação dentro de um SOAP *body*.

2.4.8. WSDL

O WSDL é uma linguagem para representar informações sobre a interface e semânticas de como invocar ou chamar um *Web Service*. As definições WSDL contêm quatro importantes informações sobre o *Web Service*:

- Informação da Interface descrevendo todas as funções públicas disponíveis;
- Informação do tipo de dado para as mensagens de requisição e resposta para estas funções;
- Informação de ligação sobre o protocolo a ser utilizado para invocar o *Web Service* especificado;
- Informação de endereço para localizar o *Web Service* especificado.

WSDL é um relatório técnico do W3C. A versão atual é a 2.0.

Anatomia do documento de definição WSDL

Um documento de definição WSDL consiste de sete elementos, Figura 2.16. Estas definições são incluídas dentro do elemento <definitions> , que é o elemento raiz em um documento WSDL. Define, principalmente, os espaços de nomes (*namespaces*) que são utilizados no restante do documento.

Types – Este elemento define todos os tipos de dados que devem ser utilizados para descrever as mensagens que são trocadas entre o Web Service e o serviço do usuário. XML Schema é o sistema padrão para tipos.

Message – Este elemento representa a definição lógica dos dados transmitidos entre o *Web Service* e o usuário do serviço. Este elemento descreve uma mensagem *one-way*, que deve representar uma requisição ou resposta enviada de/para o *Web Service*. Contém zero ou mais elementos de mensagem <part>, que refere-se aos parâmetros de requisição ou valores de retorno para a resposta.

portType – Este elemento refere-se à definição abstrata das operações suportadas pelo *Web Service* através da combinação de várias mensagens de requisição e resposta definidas pelo elemento <message>. Cada operação refere-se a operações de entrada (*input*) e saída (*output*).

Binding – Este elemento especifica o protocolo concreto e o formato dos dados utilizados para representar as operações e as mensagens definidas em um <portType> particular.

Port – Define o endereço de ligação para o *Web Service*.

Service – Agrega um conjunto de elementos <port> relacionados. Um <service> consistituído de múltiplos elementos <port> representa a capacidade do serviço de ser invocado sob múltiplas ligações.

```
<wsdl:definitions …>
  <wsdl:types>
    <!-- Esquema XML que descreve os tipos de dados utilizados -->
  </wsdl:types>
  <wsdl:message ...>
    <!—Descrição da mensagem para o Web Service -->
  </wsdl:message>
  <wsdl:portType ...>
      <wsdl:operation ...>
         <!—Referência aos parâmetros de entrada e saída -->
       </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding..>
    <!—Descrição do protocolo de rede para invocação -- >
  </wsdl:binding>
  <wsdl:service ...>
      <wsdl:port ...>
         <!—Referência para a localização real do serviço-->
       </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figura 2.16: Representação do documento WSDL.

2.4.9. UDDI

UDDI é uma das tecnologias básicas dos *Web Services*. Trata-se da especificação para criar um serviço de registro que cataloga organizações e seus *Web services*. Inicialmente, foi um esforço conjunto da *International Business Machines* (IBM), Microsoft e Ariba. A comunidade do projeto UDDI liberou a versão 2.0 em Junho de 2001. A versão 3.0 foi lançada em Julho de 2002.

Um registro UDDI pode ser operado de duas maneiras: modo público e modo privado. Um registro UDDI público é disponível para qualquer um publicar seu *Web Service* ou pesquisar informações de *Web Services* disponíveis na Internet.

Um registro UDDI é operado, geralmente, por uma única organização. Este registro pode ser do tipo privado. O administrador do registro geralmente cria o registro neste modo para poder impor controles de segurança adicionais, proteger a integridade dos registros, prevenir acesso por usuários não autorizados e, principalmente, restringir quem irá utilizar o seu registro.

Um registro UDDI é, por si só, um *Web Service*. Um usuário *Web Service* pesquisa o registro UDDI utilizando uma API SOAP. Além disto, a especificação UDDI publica uma descrição WSDL do serviço de registro UDDI.

Usos do Registro UDDI

Um Registro UDDI pode ser utilizado como um catálogo telefônico de três maneiras:

Páginas Brancas: Organizações que pretendem registrar apenas poucas informações básicas sobre elas, como: nome da empresa, endereço, informação de contato;

Páginas Amarelas: Organizações que pretendem classficar suas informações baseadas em categorias de serviço;

Páginas Verdes: Organizações que publicam informações técnicas descrevendo o comportamento e funções suportadas por seus *Web Services.*

A especificação do UDDI consiste de quatro partes principais [41]:

• **Estrutura de dados**: descreve quais tipos de dados são armazenados no UDDI. Como outra tecnologia *Web services*, a estrutura de dados do UDDI é baseada no XML. Desta forma, a interoperabilidade de linguagem de programação e plataforma é alcançada pela descrição destas estruturas de dados em documentos XML. As estruturas de dados são descritas através de um XML *Schema*;

• **API do programador**: define os tipos de acesso ao registro UDDI. Dois tipos de APIs estão definidas: API de publicação e API de investigação. A API de publicação é utilizada para criar e atualizar entradas existentes no registro. A API de investigação permite que entradas existentes sejam consultadas.

A API é independente de linguagem de programação, pois as descrições dos dados de requisição e retorno são baseadas em XML. Estas estruturas de requisição e retorno mapeiam o conteúdo real do registro. Os registros existentes oferecem acesso via SOAP sobre HTTP. Isto significa que os documentos XML de requisição e retorno são encapsulados dentro do envelope SOAP. As funções de investigação estão disponíveis sobre o HTTP, enquanto que as funções de publicação estão acessíveis via HTTPs e requer um identificador (Identifier - Id) de usuário e senha para processar a requisição. Cada registro UDDI define a forma para um usuário obter identificador e senha válidos.

• **Especificação de replicação**: contém descrições de como registros replicam informações entre eles. Esta informação é somente necessária para quem quer implementar seu próprio registro e integrar com outros registros existentes;

• **Especificação do operador**: é voltada para aqueles que estão implementando ou executando um registro UDDI. Esta especificação define políticas para segurança (por exemplo, Id do usuário requerido para fazer modificações) e para gerenciamento de dados (por exemplo, quantas entradas podem ser criadas por uma conta). Enquanto a especificação não obriga que um operador siga certas políticas, isto requer que cada operador publique quais políticas estarão utilizando.

Atualmente o UDDI é um padrão OASIS.

2.4.10. Gateway SNMP- *Web Services*

Alguns trabalhos foram realizados visando comparar o desempenho do SNMP e dos Web Services, notadamente SOAP. Estes trabalhos propuseram um *gateway* SNMP-SOAP.

Em [51], Bernhard compara o NET-SNMP com AXIS, a implementação de SOAP da fundação Apache na linguagem Java. Em [52] Aiko Pras também utiliza o NET-SNMP, mas agora o compara com gSoap, implementação do SOAP em linguagem C (assim como o NET-SNMP). O trabalho de Bernhard não é tão completo e detalhado quanto o de Aiko, mas chega à conclusão que SOAP é adequado onde se exige funcionalidades complexas como, por exemplo, a possibilidade de realizar várias operações numa mesma chamada SOAP. Enquanto

no SNMP, para realizar estas várias operações, gera-se um grande *overhead* em função da necessidade de se instanciar várias mensagens (PDUs) SNMP.

O trabalho de Aiko chega à conclusão de que as métricas obtidas permitem dizer que o SOAP é uma boa opção para gerência. Principalmente, porque com SOAP obtém-se a facilidade de criação de estruturas de dados avançadas.

Observação quanto aos trabalhos comentados

Importante dizer que as duas classes de trabalhos comentadas, *gateways* SNMP-XML e *gateways* SNMP-*Web Services*, referem-se a sistemas de gerenciamento de rede voltados principalmente para o monitoramento e desempenho dos dispositivos de rede. Por isso, a questão da área funcional de Gerência de Configuração mantêm-se em aberto.

Desta maneira, são apresentados, a seguir, os requisitos necessários para um protocolo de gerência de configuração que utilize as novas tecnologias de Internet (HTTP e XML) e resolva as deficiências apontadas para a CLI e o SNMP, principalmente. Este requisitos são baseados no trabalho desenvolvido por [53].

2.5. Requisitos da configuração

Em virtude das deficiências discutidas nas seções anteriores, um novo protocolo de Gerência de Redes é desejável de modo a contemplar a área funcional voltada à configuração de redes. A definição deste protocolo deve seguir alguns requisitos básicos. Estes requisitos devem manter as boas práticas do SNMP, ao mesmo tempo em que oferecem novas potencialidades. Os principais requisitos são:

a. Prover um modelo de transação orientado à conexão, com segurança;

b. Permitir que um grande volume de dados de gerência e configuração sejam transferidos;

c. Prover uma forma otimizada de obter uma parte específica da configuração;

d. Utilizar uma codificação das mensagens que seja mais natural para o ser humano, ao invés do ASN.1/BER, por exemplo;

e. Substituir o mecanismo *RowStatus* por operações explícitas para criação, edição e remoção de objetos de dados;

f. Facilitar o tratamento de erro através da distinção entre as operações, principalmente para a criação e edição;

g. Permitir a criação de operações complexas a partir de duas ou mais operações simples;

h. Permitir, de maneira simples, que a operação de configuração possa ser do tipo "tudo ou nada" (transação atômica) ou "pare se houver erro";

i. Permitir que a consulta por um objeto de dados utilize critérios de pesquisa.

2.5.1. Outros requisitos desejáveis

Além dos requisitos anteriores, que são mandatórios, outros requisitos de mais alto nível são desejáveis. A seguir a relação destes requisitos:

j. Suportar modelos de configuração, da mesma maneira que, para o desenvolvimento de software, é necessário ter reusabilidade controlada e controle de acesso em camadas para parâmetros específicos;

k. Configurações nomeadas para permitir o armazenamento no dispositivo de mais de um arquivo de configuração simultaneamente;

l. Permitir uma forma fácil de autenticação que não gere a necessidade de se ter uma administração de segurança em separado.

2.6. Considerações finais

Neste capítulo foram analisadas as tecnologias de Gerência de Redes com foco na área funcional de Gerência de Configuração.

Duas tecnologias são as mais comuns e utilizadas para esta tarefa: CLI e SNMP. O CLI é mais utilizado para a configuração do dispositivo de rede. O SNMP é mais utilizado para o monitoramento e gerência de desempenho do dispositivo de rede.

No entanto, as duas tecnologias apresentam deficiências importantes para a tarefa de gerenciamento de configuração. Uma outra tecnologia se faz necessária para substituir as outras duas, preservando as boas práticas de cada uma e melhorando seus pontos fracos.

Como proposta de tecnologia desenvolvida recentemente temos o NETCONF. Protocolo de configuração de dispositivos definido no âmbito do IETF. O protocolo, como será visto no Capítulo 3, atende os requisitos de configuração listados anteriormente e adota duas tecnologias importantes da Internet atual: o protocolo HTTP, com sua altíssima aceitabilidade pelos usuários e administradores de rede, e o XML com a sua vasta quantidade de ferramentas de programação, muitas delas disponíveis de maneira gratuita.

Capítulo 3
O protocolo NETCONF

Neste capítulo detalhamos o protocolo NETCONF. Principalmente, como deve ser uma mensagem NETCONF, quais os elementos que ela deve apresentar, como se estabelece uma sessão NETCONF e como deve reagir o agente quando da solicitação de um gerente.

3.1. Introdução

Atualmente, o protocolo NETCONF é uma RFC de número 4741 no IETF dentro da área de operações e gerência, sob a responsabilidade do grupo de trabalho que dá nome ao protocolo, NETCONF (*Network Configuration*).

O protocolo NETCONF utiliza chamadas de procedimentos remotos (RPC) como paradigma para a comunicação entre a entidade gerente e a entidade agente, cliente e servidor, respectivamente, numa visão clássica de sistemas distribuídos. O gerente é quem inicia uma sessão NETCONF. Esta sessão é caracterizada por ser segura e por haver persistência entre os pares. Para isso, deve-se usar um esquema de autenticação, autorização e confidencialidade. As mensagens são codificadas em XML a fim de facilitar a leitura pelo homem.

3.1.1. Visão geral do Protocolo

Conceitualmente o NetConf é divido em quatro camadas, Figura 3.11:

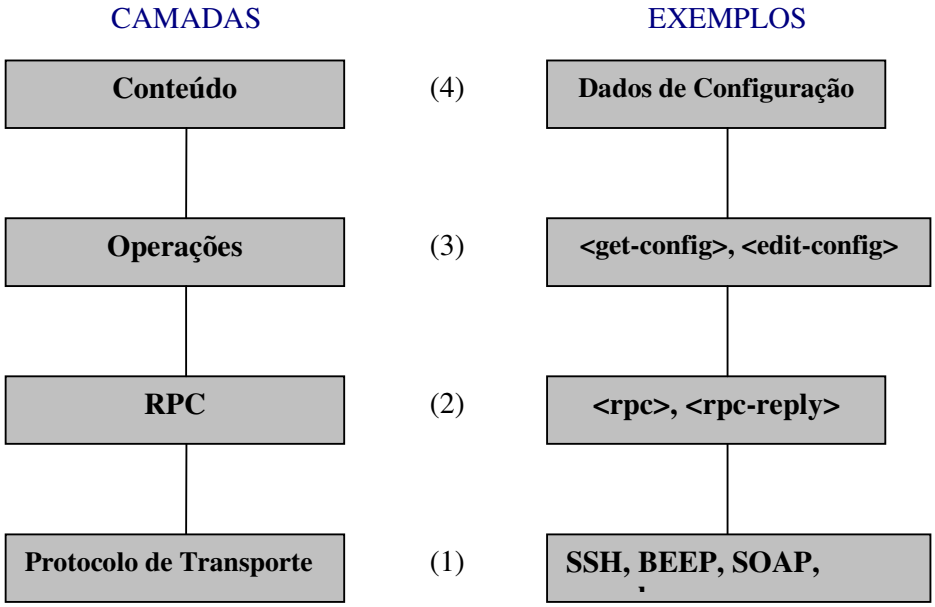


Figura 3.1: Camadas do modelo NETCONF.

- Protocolo de transporte - provê a comunicação entre o cliente e o servidor. Atualmente está mapeado para SSH, BEEP e SOAP. Desde que sejam seguidos os requisitos básicos do NETCONF, qualquer outro transporte pode ser usado;
- RPC - provê um mecanismo simples para codificação das chamadas remotas;
- Operações - conjunto básico de operações invocadas como métodos remotos com parâmetros codificados em XML;
- Conteúdo - fora do escopo do protocolo. Refere-se ao modelo de informação de cada dispositivo.

3.1.2.Capacidades

Uma capacidade NETCONF é um conjunto de funcionalidades que modifica a especificação e as operações básicas do NETCONF. A capacidade é identificada por uma URI. Elas incrementam as operações básicas do dispositivo, com novas operações e novos conteúdos dentro das operações pré-existentes. O cliente descobre as capacidades do servidor e usa as operações adicionais de acordo com os parâmetros e conteúdo definidos por estas capacidades.

O protocolo NETCONF define como se dá a troca de mensagens de capacidades entre o cliente e o servidor através do elemento <hello>. Ver Seção 3.5.

A URI de uma capacidade deve ser suficientemente distinta de outra através de uma autoridade de nomeação a fim de evitar colisões de nomes.

3.1.3. Requisitos do Protocolo de Transporte

O protocolo NETCONF é baseado no paradigma de comunicação RPC. Um cliente envia uma ou mais operações RPC de requisição, e o servidor responde à série com respostas RPC. Uma vez que o protocolo de transporte suporte chamadas RPC e respeite alguns pré-requisitos básicos, ele pode ser usado para transportar as mensagens NETCONF. Os pré-requisitos são:

1. Operação orientada a conexão: O NETCONF é orientado a conexão e requer, portanto, uma conexão persistente entre o cliente e o servidor. Esta conexão deve prover uma comunicação confiável para os dados, além da sua entrega em sequência. As conexões são de longa vida, persistindo durante as várias operações do protocolo. Um exemplo seria a informação de autenticação que deve ser a mesma até que uma das partes feche a conexão. Assim que a conexão é encerrada, os recursos em uso por ela no servidor devem ser liberados de forma que, na eventualidade de uma falha, a recuperação do estado anterior à falha seja simples e robusta;

2. Autenticação, Integridade e Confidencialidade: Estas características são obrigatórias e o NETCONF depende única e exclusivamente dos mecanismos existentes no próprio protocolo de transporte;

3. Autenticação: As conexões NETCONF devem ser autenticadas e isto é de responsabilidade do protocolo de transporte. Além do mais, o NETCONF sempre faz uso do mecanismo já utilizado pelo dispositivo. Por exemplo, se este usa *Remote Authentication Dial-In User Service* (RADIUS) [54] para autenticação, então as sessões NETCONF devem ser autenticadas por aquele protocolo. Como resultado da autenticação, uma identidade é gerada, cujas permissões são conhecidas pelo dispositivo. Estas permissões devem ser usadas durante toda a sessão NETCONF.

Estes requisitos são compatíveis com os itens a, b, m da Seção 2.3. Através de um protocolo de transporte que suporte conexão, como o HTTP sobre TCP, ou o uso do HTTPs é possível estabelecer uma transação orientada à conexão que seja segura. No caso do HTTPs, é o próprio protocolo que oferece a infra-estrutrua de segurança. O NETCONF apenas utiliza o que é disponibilizado para ele.

3.1.4. Codificação das mensagens

As mensagens NETCONF são codificadas em XML. Isto permite que os dados hierárquicos complexos sejam expressos no formato texto, o qual pode ser lido, salvado e manipulado por ferramentas de manipulação de texto e/ou ferramentas específicas do XML. Um *namespace* para todos os elementos do NETCONF é definido por um *Uniform Resource Name* (URN): ***urn:ietf:params:xml:ns:NETCONF:base:1.0***. Também as capacidades devem ser nomes do tipo *Uniform Resource Identifiers* (URIs).

O uso do XML atende o requisito d da Seção 2.3. Como o XML pode ser lido tanto por seres humanos como por máquinas, ele é um bom substituto para o BER. Ele substitui o ASN.1 ao definir a semântica dos dados a partir de um XML SCHEMA.

3.2. Modelo RPC

As mensagens RPCs são codificadas em XML. Uma requisição RPC é dada pelo elemento <rpc>. Este elemento encapsula os nomes e parâmetros da requisição como conteúdo do elemento <rpc>. O retorno é dado pelo elemento <rpc-reply>. Após o agente receber uma requisição <rpc>, ele a executa. Se o processamento ocorrer com sucesso, o elemento <ok> retornará encapsulado como conteúdo do <rpc-reply>. O último elemento é o <rpc-error>. Este é encapsulado dentro de um <rpc-reply> caso haja alguma falha de processamento. Na Figura 3.2, abaixo, é mostrada a sequência de mensagens para o estabelecimento da sessão NETCONF e a troca de mensagens do modelo RPC:

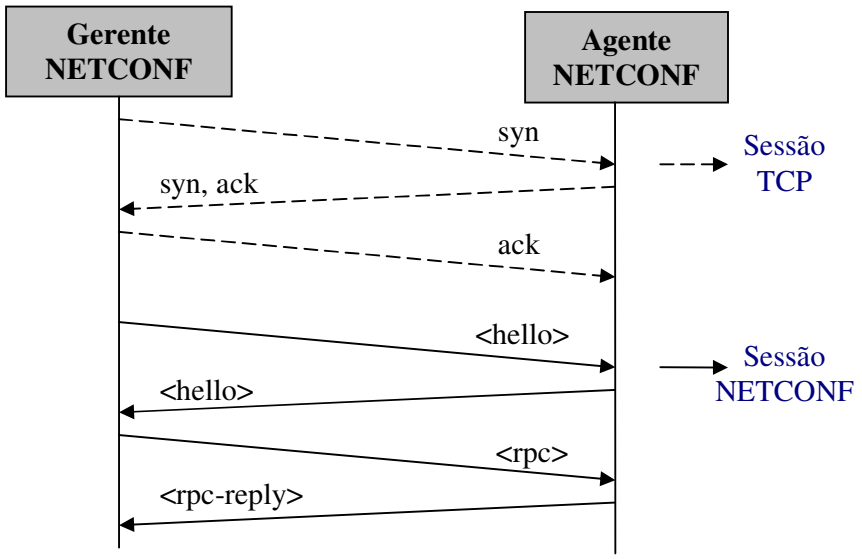


Figura 3.2: Estabelecimento da sessão NETCONF.

Nesta figura, primeiro é estabelecida a Sessão TCP, através do mecanismo conhecido como *three-handshaking* [55]. Da mesma maneira que duas pessoas se cumprimentam antes de começar um diálogo, estabelecendo assim um canal de comunicação, é necessária uma forma de duas entidades computacionais conversarem. Uma vez que o TCP é um protocolo orientado à conexão, para se estabelecer a comunicação entre um cliente (Gerente NETCONF) e um servidor (Agente NETCONF) são trocados os sinais: *SYN; SYN/ACK;*

ACK. O sinal *SYN* é utilizado para sincronismo entre o cliente e o servidor. O servidor reconhece o pedido de estabelecimento de conexão e responde com o sinal *SYN/ACK*. O cliente responde com o sinal *ACK*. Por isso, o mecanismo ser conhecido por "aperto de mão em três passos". A partir daí a Sessão TCP está estabelecida.

Após o estabelecimento da Sessão TCP, o Gerente (sempre o Gerente) envia a mensagem de <hello> para divulgar as suas capacidades (Seção 3.5). O agente retorna uma mensagem <hello> divulgando suas capacidades. A partir deste momento o Gerente pode enviar as suas solicitações NETCONF ao Agente, encapsuladas em mensagens <rpc> e receber as respostas em mensagens <rpc-reply>.

O mecanismo RPC atende os requisitos g e i da Seção 2.3. Ao definir os elementos básicos <rpc> e <rpc-reply> para uma mensagem RPC, é fácil criar outras operações a partir delas. Estas outras operações são definidas na Seção 3.4. Sendo o modelo RPC baseado no modelo requisição-resposta, facilita a criação de transações atômicas. Note que o uso do modelo RPC por si só não define completamente uma transação atômica. Esta opção se completa através de capacidades adicionais como *rollback*, ver Seção 3.4, e através da operação <lock>.

3.2.1. O elemento <rpc>

O elemento <rpc> é usado para confinar a requisição NETCONF enviada sempre do cliente para o servidor. Este elemento possui um atributo obrigatório chamado *message-id*. Este atributo é um inteiro que é incrementado pelo cliente à medida que as requisições são enviadas. O servidor armazena o valor do atributo recebido e o retorna na mensagen <rpc-reply> correspondente. Por exemplo:

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <algum-método>
       <!—parâmetros do método... -->
      </algum-método>
</rpc>
```

Se atributos adicionais estiverem presentes num elemento <rpc>, o servidor deverá retorná-los no elemento <rpc-reply> sem modificações. O nome e os parâmetros de um RPC são tratados como conteúdos do elemento <rpc>. O nome da chamada remota é um elemento dentro do elemento <rpc> e os seus parâmetros são codificados dentro daquele elemento. No exemplo a seguir é invocado o método chamado <meu-método> que têm dois parâmetros: <primeiro>, com o valor "14", e <segundo>, com o valor "fred":

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
     <meu-método xmlns="HTTP://example.net/me/my-own/1.0">
      <primeiro>14</primeiro>
      <segundo>fred</segundo>
     </meu-método>
</rpc>
```

3.2.2. O elemento <rpc-reply>

O elemento <rpc-reply> é enviado do servidor para o cliente em resposta a uma operação <rpc>. Ele também possui o atributo mandatório *message-id*, cujo valor é o mesmo do atributo do elemento <rpc> do qual o <rpc-reply > é a resposta.

Como dito na Seção 3.2.1, se atributos adicionais estiverem presentes num elemento <rpc>, o servidor deverá retorná-los no elemento <rpc-reply> sem modificação. O nome da resposta e os dados da resposta são tratados como conteúdos do elemento <rpc-reply>. Este nome é um elemento dentro do elemento <rpc-reply> e os dados são codificados dentro daquele elemento.

No exemplo a seguir é enviada uma requisição RPC com o atributo "user-id". Esta requisição invoca o método <get> do NETCONF.

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
        xmlns:ex="HTTP://example.net/content/1.0"
        ex:user-id="fred">
      <get/>
</rpc>
```

O conteúdo do retorno segue encapsulado no elemento <data>. O retorno segue abaixo:

```
<rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
        xmlns:ex="HTTP://example.net/content/1.0"
        ex:user-id="fred">
      <data>
       <!—Conteúdo aqui... -->
      </data>
     </rpc-reply>
```

3.2.3.O elemento <rpc-error>

O elemento <rpc-error> é enviado encapsulado no elemento <rpc-reply> se um erro ocorrer durante o processamento de uma requisição <rpc>.

O elemento <rpc-error> possui outros elementos que são encapsulados dentro dele com informações a cerca do erro:

- <*error-type*> - Define a camada conceitual onde o erro ocorreu. Pode ser uma das quatro possibilidades: *transport, rpc, protocol* ou *application;*
- <*error-tag*> - Contém a *string* que identifica a condição de erro. O apêndice A de [30] possui uma lista dos possíveis valores;
- <*error-severity*> - Contém a *string* que identifica a severidade do erro. Pode ser uma das duas possibilidades: *error, warning;*
- <*error-app-tag*> - Contém a string que identifica a condição de erro da implementação ou do modelo de dados específico, se existir;

- <*error-xpath*> -Contém a expressão XPATH absoluta que identifica o caminho do elemento para o nó que está associado com o erro que está sendo reportado;
- <*error-message*> - Contém a string que descreve a condição de erro na forma da linguagem escrita do usuário;
- <*error-info*> - Contém o conteúdo do erro do protocolo ou do modelo de dados específico.

Um erro é retornado se o elemento <rpc> for recebido sem o atributo *message-id*, por exemplo. Veja a seguir:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
      <source>
        <running/>
      </source>
    </get-config>
</rpc>
```

Na mensagem acima, após a palavra rpc e antes da palavra xmlns, está faltando o atributo obrigatório *message-id*.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error>
      <error-type>rpc</error-type>
      <error-tag>missing-attribute</error-tag>
      <error-severity>error</error-severity>
      <error-info>
        <bad-attribute>message-id</bad-attribute>
        <bad-element>rpc</bad-element>
      </error-info>
    </rpc-error>
</rpc-reply>
```

Por isso, é gerada a mensagem de erro acima. Nela é informado o tipo do erro: rpc (*tag error-type*). Significa que o modelo rpc não foi respeitado e o processamento da mensagem recebida não pode ser realizado. É indicado o tipo de erro, no caso a falta de um atributo (*tag error-tag*). A gravidade do erro é dada pela *tag* <error-severity>, no caso apenas um erro. A seguir, na mensagem é detalhada as informações de erro: atributo faltando, *message-id*; e que o elemento <rpc> foi mal formado.

3.2.4.O elemento <ok>

Como dito anteriormente no início da seção, este elemento é encapsulado dentro do elemento <rpc-reply> caso o resultado do processamento tenha ocorrido com sucesso. Exemplo:

```
<rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <ok/>
</rpc-reply>
```

3.3. Filtro

O protocolo define um esquema próprio de filtro das informações chamado *subtree-filtering*. Este filtro pode estar presente nas requisições <rpc> como um atributo <filter=subtree>. Quando presente na requisição, o seu formato define como a informação deve ser solicitada. Com isso, define-se também como o agente deve processar o pedido e como as informações devem ser retornadas. Outro esquema de filtro possível é apresentado na Seção 3.5.

O uso de filtro atende os requisitos c e j da Seção 2.3. Ao disponibilizar um filtro é possível o gerente selecionar partes específicas da configuração. Uma vez que um arquivo de configuração completo pode ser muito extenso e todo o seu conteúdo pode não ser necessário em determinadas situações. Exemplo: um gerente gostaria de saber as configurações de

VLANs de seu switch. Ao utilizar um filtro é selecionada apenas esta parte da configuração do dispositivo, diminuindo a quantidade de dados trafegados pela rede.

3.4. Operações básicas

As operações básicas do protocolo são: *get, get-config, edit-config, copy-config, delete-config, lock, unlock, close-session, kill-session*. Estas operações são solicitadas pelo gerente. Sua execução pelo agente é realizada sobre a configuração atual do dispositivo, chamada *running*. Outras configurações são possíveis (Seção 3.5). Todas essas operações são *tags* XML encapsuladas dentro de um elemento <rpc>.

Ao definir este conjunto de operações o NETCONF atende os seguintes requisitos definidos na Seção 2.3: e,f. As operações básicas permitem que uma tarefa de configuração seja acompanhada do seu início ao fim.

Ao suportar operações explicítas para a criação, edição ou remoção de objetos de dados, o NETCONF permite que o agente informe claramente que tipo de erro ocorreu. Isto facilita o tratamento de erros tanto no agente, que pode mais facilmente se recuperar de uma falha e retornar ao estado anterior, como no gerente que vai saber exatamente o que aconteceu e que medidas paliativas deve tomar para contornar o problema ou evitar que aconteça novamente.

Operações

No documento do NETCONF, cada operação é especificada da seguinte maneira: inicialmente é apresentada uma breve descrição; em seguida, são mostrados e comentados os parâmetros obrigatórios e opcionais; as respostas positivas (em caso de sucesso) e as respostas negativas (em caso de falhas).

As operações, suas descrições e alguns comentários sobre elas são mostrados a seguir (exemplos das mensagens de operações básicas podem ser vistas no Anexo A):

<get>

Obtém a configuração *running* e os dados de estado do dispositivo.

<get-config>

Obtém toda ou parte de uma configuração.

<edit-config>

Carrega parte ou toda uma configuração para o dispositivo. Define operações para atualizar, criar, apagar ou fazer o *merge* da configuração atual com os novos dados que estão sendo enviados encapsulados no elemento <*edit-config*>.

<copy-config>

Copia inteiramente uma nova configuração ou sobrescreve a atual, se permitido.

<delete-config>

Apaga a configuração indicada. A configuração *running* não pode ser apagada. Por isso, depende de capacidades adicionais, como *candidate*.

<lock>

Retém o acesso à configuração indicada. Utilizado para garantir o acesso exclusivo à configuração e evitar que outras sessões tenham acesso a esta mesma configuração.

A operação <*lock*> atende o requisito i da Seção 2.3. Ao permitir que em um dado momento o arquivo de configuração seja acessado de forma exclusiva por uma sessão NETCONF (ou seja, um gerente) esta operação auxilia a criação de transações atômicas ao suportar um mecanismo básico de exclusão mútua. Para criar uma transação atômica é necessário novas capacidades. Ver Seção 3.5.

<unlock>

Libera a configuração para que outras sessões a usem.

A operação <*unlock*> atende o requisito i da Seção 2.3. Após realizar a operação de alteração da configuração, via <*edit-config*> ou <*copy-config*>, utilizando antes o <*lock*>, a sessão NETCONF deve liberar o arquivo de configuração para outras sessões/gerentes. Desta forma, a operação <*unlock*> auxilia na criação de uma transação atômica.

<close-session>

Termina a própria sessão, liberando todos os recursos associados a ela.

<kill-session>

Força o término de uma sessão NETCONF através da indicação do número de identficação da sessão que se deseja encerrar.

Os possíveis parâmetros que as operações podem receber são mapeados em elementos XML. Os parâmetros são mostrados a seguir:

- *Session-id*: Número de identificação da Sessão NETCONF que deseja-se encerrar;
- *Filter*: Especifica a parte da configuração do sistema ou dos dados de estado que se deseja consultar ou alterar, conforme a operação. Pode ser do tipo *subtree* (padrão) ou outro, conforme capacidade adicional (*xpath*, por exemplo);
- *Config*: Hierarquia de dados de configuração definido pelo modelo de dados do dispositivo. Identifica o ponto de substituição da configuração atual pelos novos dados;
- *Target*: Nome do arquivo de configuração que será o alvo de uma edição, cópia, remoção ou travamento/destravamento, conforme a operação. Pode ser a configuração padrão *running* ou outra, conforme capacidade adicional (*candidate*, por exemplo);
- *Source*: Nome do arquivo de configuração de origem a ser consultado ou copiado, conforme a operação;
- *Default-Operation*: Parâmetro específico da operação <edit-config>. Define a forma como a edição do arquivo de configuração *target* será realizada. Pode ser um dos seguintes tipos:
    - *Merge* – A configuração atual é fundida com os novos dados de configuração. Esta é a forma padrão;
    - *Replace* – Repõe completamente um arquivo de configuração por um totalmente novo;
    - *None* – O arquivo de configuração não é afetado pelos novos dados de configuração.
- *Test-Option**: Define se o agente deve validar ou não a nova configuração recebida. Válido somente se for definida a capacidade *validate*;
- *Error-Option**: Define se o agente deve parar ou continuar a configuração se um erro acontecer. O padrão é parar se ocorrer erro: *stop-on-error*.

Uma resposta positiva pode ser uma mensagem de <ok> (Seção 3.2.4) ou dados encapsulados na *tag* <*data*>, todas embutidas na *tag* <rpc-reply>.

Uma resposta negativa é dada em virtude da falha de processamento por algum motivo (mensagem mal-formada ou dados informados incorretos), encapsulada na mensagem <rpc-error> (Seção 3.2.3).

A seguir uma tabela onde, para cada operação, são informados os parâmetros que devem estar presentes na mensagem e as respostas esperadas.

Tabela 3.1 – Operações básicas e parâmetros.

	PARÂMETROS							
Operação	**Session-id**	**Filtro**	**config**	**target**	**source**	**Default-operation**	**Resposta Positiva**	**Resposta Negativa**
<get>	NÃO	OPCIONAL	NÃO	NÃO	NÃO	NÃO	<data>	<rpc-error>
<get-config>	NÃO	OPCIONAL	NÃO	NÃO	OPCIONAL	NÃO	<data>	<rpc-error>
<edit-config>**	NÃO	OPCIONAL	OPCIONAL	OPCIONAL	NÃO	OPCIONAL	<data>	<rpc-error>
<copy-config>	NÃO	NÃO	NÃO	OPCIONAL	OPCIONAL	NÃO	<ok>	<rpc-error>
<delete-config	NÃO	NÃO	NÃO	OBRIGATÓRIO	NÃO	NÃO	<ok>	<rpc-error>
<lock>	NÃO	NÃO	NÃO	OBRIGATÓRIO	NÃO	NÃO	<ok>	<rpc-error>
<unlock>	NÃO	NÃO	NÃO	OBRIGATÓRIO	NÃO	NÃO	<ok>	<rpc-error>
<close-session>	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	<ok>	<rpc-error>
<kill-session>	OBRIGATÓRIO	NÃO	NÃO	NÃO	NÃO	NÃO	<ok	<rpc-error>

* Os parâmetros test-option e error-option são exclusivos da operação <*edit-config*>.

** Para a operação <*edit-config*> também é definido o atributo *operation* para qualquer elemento da configuração. Este atributo pode estar presente em qualquer ponto do arquivo de configuração. Pode assumir os seguintes valores: *merge* e *replace* (como definido para o parâmetro *default-operation*); *create*, para criar uma entrada totalmente nova na configuração; *delete*, para apagar o elemento indicado no arquivo de configuração.

3.5. Capacidades

Capacidades são maneiras de estender o protocolo. O protocolo base, formado basicamente pelo conjunto de operações citadas anteriormente, é definido pela seguinte URN: ***urn:ietf:params:xml:ns:NETCONF:base:1.0***. Outras capacidades são possíveis. Por isso, assim que uma sessão NETCONF é iniciada, a primeira troca de mensagens que ocorre é a troca de capacidades, identificada pelo elemento <hello>. Nele estão encapsuladas as capacidades que o gerente e o agente suportam.

Segue a descrição de algumas capacidades já propostas pela especificação:

- *xpath* - indica que a entidade aceita expressões XPATH como filtro, identificada da seguinte forma: ***urn:ietf:params:xml:ns:netconf:xpath:1.0***;
- *candidate* - indica que a entidade aceita um segundo arquivo de configuração, no qual alterações não impactam na configuração atual, *running*, que está em execução, identificada da seguinte forma: ***urn:ietf:params:xml:ns:netconf:candidate:1.0***;
- *validate* – indica que a entidade aceita que a configuração *candidate* seja validada quanto à sintaxe e semântica, antes de se tornar a configuração *running*, identificada da seguinte forma: ***urn:ietf:params:xml:ns:netconf:validate:1.0****;*
- *rollback-on-error* – indica que a entidade aceita, em caso de falha na atualização, o retorno à última configuração *running* válida, identificada da seguinte forma: ***urn:ietf:params:xml:ns:netconf:rollback-on-error:1.0***.

Cada nova capacidade pode alterar ou não como uma operação base é executada. Para isso é definido na especificação um modelo, *template*, como forma de documentar a nova capacidade e como ela afeta a operação base.

As capacidades atendem os requisitos g, h,l, k da Seção 2.3. A partir das mensagens básicas <rpc> e <rpc-reply> do modelo RPC são definidas as operações básicas. Com o auxílio de novas capacidades, uma delas a rollback-on-error, é possível criar operações mais complexas. A partir destas operações complexas, auxiliadas pelas operações <lock> e

<unlock> é possível criar transações atômicas, onde a configuração é realizada por completo ou o dispositivo retorna ao estado anterior.

A capacidade <candidate> permite que se tenha mais de um arquivo configuração. Este outro arquivo é diferenciado por seu nome, *candidate*, diferente do nome do arquivo de configuração atual, *running*, facilitando a nomeação dos arquivos de configuração.

3.6. Protocolos de transporte

O grupo de trabalho do IETF definiu três propostas referentes ao protocolo de transporte das mensagens NETCONF: SSH, BEEP, SOAP.

Transporte via SSH

O transporte via SSH é definido como obrigatório. Depois que o serviço de conexão SSH é estabelecido, o cliente abrirá um canal do tipo sessão. Uma vez que a sessão SSH tenha sido estabelecida, o usuário irá invocar o NETCONF como um subsistema chamado "NETCONF". Caracteriza-se por indicar o fim da mensagem pela seguinte sequência de caracteres]]>]]>.

Transporte via BEEP

O outro transporte definido é sobre BEEP. Este mapeamento é muito pouco usado, pois o protocolo não é tão comum. Mas permite que qualquer uma das entidades gerente ou agente inicie a sessão NETCONF.

Transporte via SOAP

Por último, é definido o mapeamento para SOAP. A grande vantagem deste mapeamento é que este é usualmente encapsulado em HTTP. O uso do HTTP possibilita que a configuração possa ser realizada pela Internet, uma vez que se trata de uma porta bem conhecida por qualquer *firewall.* Um encapsulamento de SOAP sob BEEP também é possivel.

Mas este sofre dos mesmos defeitos e qualidades do mapeamento do NETCONF diretamente sob BEEP. Outra vantagem é que também as mensagens SOAP são codificadas em XML.

A mensagem SOAP é composta por um envelope. O envelope é formado por um cabeçalho e um corpo, respectivamente *header* e *body*. O *body* é encapsulado dentro do *header*. No *header* são inseridas informações pertinentes ao servidor SOAP como, por exemplo, usuário e senha ou se a mensagem recebida vai ser processada ou encaminhada para outro servidor SOAP. No *body* estão presentes os dados da mensagem, ou seja o *payload*.

A proposta de mapeamento para SOAP com HTTP propõe que o *header* SOAP não seja usado, já que é específico da aplicação. Outra recomendação importante é o uso de mecanismos de segurança. O requisito mínimo é estabelecer conexões persistentes na troca de mensagens através do HTTPs. O uso da versão 1.1 do HTTP também é recomendado como forma de evitar o excesso de *three-handshaking* do TCP, como mostrado na Figura 3.2. Além destas e outras recomendações, a proposta fornece um WSDL e uma amostra de um serviço WSDL. Na proposta do protocolo NETCONF é fornecido um XML SCHEMA . A união destes três, em um único arquivo WSDL, fornece os meios para se construir uma implementação do protocolo NETCONF usando SOAP. Ver detalhes no Capítulo 4.

No Anexo A é mostrado o formato das mensagens para a troca de <hello> entre o cliente (gerente) e o servidor (agente). A mensagem do NETCONF é encapsulada no envelope SOAP e este, por sua vez, encapsulado no HTTP:

Falhas no SOAP

No caso de falha no processamento da requisição, uma falha SOAP deve ser construída conforme definido pelo W3C como forma de encapsular a mensagem <rpc-error> definida no protocolo NETCONF. Na Seção 3.2.3. é apresentada uma mensagem de erro devido à mensagem de requisição não possuir o atributo obrigatório *message-id*. A seguir é apresentado o mesmo exemplo agora com o uso do SOAP.

```
<soapenv:Envelope
      xmlns:soapenv          "HTTP://www.w3.org/2003/05/soap-envelope"
      xmlns:xml="HTTP://www.w3.org/XML/1998/namespace">
    <soapenv:Body>
      <soapenv:Fault>
        <soapenv:Code>
          <soapenv:Value>env:Receiver</soapenv:Value>
        </soapenv:Code>
        <soapenv:Reason>
          <soapenv:Text
             xml:lang="en">MISSING_ATTRIBUTE</soapenv:Text>
        </soapenv:Reason>
        <detail>
          <rpc-error xmlns              "urn:ietf:params:xml:ns:netconf:base:1.0">
            <error-type>rpc</error-type>
            <error-tag>MISSING_ATTRIBUTE</error-tag>
            <error-severity>error</error-severity>
            <error-info>
              <bad-attribute>message-id</bad-attribute>
              <bad-element>rpc</bad-element>
            </error-info>
          </rpc-error>
        </detail>
      </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

Como visto no exemplo acima, a mensagem SOAP de falha é construída a partir da mensagem <rpc-error> da seguinte maneira: o valor do código de falha é "*Receiver*", o texto da razão da falha é o conteúdo da *tag error-tag* do <rpc-error> e o detalhe da falha é a estrutura original do <rpc-error>. Sendo o elemento <*detail*> pertencente ao *body* da mensagem SOAP.

3.7. Configuração de múltiplos dispositivos

A especificação do protocolo define passo-a-passo, como se fosse um roteiro de boas práticas, as operações para a configuração de um único dispositivo. O mesmo também é feito para a configuração de múltiplos dispositivos.

Configuração de um único dispositivo

A tarefa aqui é atualizar o dispositivo. O mais importante nesta tarefa é ter certeza que ao fim do processo a rede estará operacional como anteriormente.

Neste passo-a-passo é prevista a implementação de capacidades adicionais ao protocolo base.

Passos:

1. Adquirir o *lock* da configuração atual – enviar uma mensagem <*lock*> indicando a configuração que deseja alterar;
2. Carregar a nova configuração – para este passo deve ser implementada a capacidade :*candidate*;
3. Validar a nova configuração – para este passo deve ser implementada a capacidade :validate e utilizar o parâmetro *test-option* na operação <*edit-config*>;
4. Salvar a configuração *running* atual – para este passo pode-se utilizar a operação <*copy-config*>, indicando como *source* a configuração *running* e um ponto qualquer da rede, como uma url, por exemplo, se o agente suportar esta capacidade.;
5. Trocar a configuração *running* atual pela nova configuração – para este passo pode-se utilizar a operação <*copy-config*>, indicando como *source* a configuração *candidate* e *target* a configuração *running*;
6. Testar a nova configuração *running* – para este passo pode-se utilizar a operação <*edit-config*>, indicando o parâmetro *test-option;*
7. Tornar a nova configuração permanente – uma opção para este passo é realizar um <*commit*>, se esta capacidade for suportada;

8. Liberar o *lock* da configuração – por último, enviar a mensagem <*unlock*> para liberar a configuração para que outros gerentes tenham acesso,

Configuração de múltiplos dispositivos

A configuração de múltiplos dispositivos deve ser efetuada com cuidado, uma vez que, desta maneira, não só um dispositivo de rede, mas toda a rede pode ficar inoperante devido às falhas no processo de configuração. A implementação do protocolo pode prever o suporte à transação para esta tarefa. Embora seja dispendiosa, esta opção pode reduzir a quantidade de falhas no processo.

Duas classes de ações podem ser utilizadas para a configuração de múltiplos dispositivos. Na primeira forma, se a configuração de um dispositivo falhar esta é ignorada. A configuração pode ser realizada posteriormente e/ou o operador ser informado do problema. A outra forma é mais radical: ou a configuração é realizada com sucesso em todos os dispositvos ou, em caso de falha, revertida em todos.

Para a segunda forma, os mesmos passos do item 3.7.2. são realizados em paralelo em todos os dispositivos. Na presença de alguma falha, todos os dispositivos envolvidos voltam ao seu estado anterior.

3.8. Notificação de eventos

A partir de uma submissão, foi proposta ao grupo de trabalho do NETCONF uma forma de enviar mensagens assíncronas ou notificação de eventos compatível com o protocolo NETCONF. A proposta atual está na sua versão 08 [56]. No entanto, este trabalho baseia-se na versão 2 [5].

A proposta define o procedimento no qual um cliente NETCONF se inscreve para receber eventos de seu interesse. A sua implementação é definida como uma nova capacidade suportada pelas entidades cliente e servidor. É identificada da seguinte forma: ***urn:ietf:params:xml:ns:netconf:notification:1.0***. A seguir é detalhado este processo, os

elementos XML definidos, os tipos de classes de eventos apresentados pela proposta e o mapeamento para os transportes SSH, BEEP e SOAP, este último em destaque.

3.8.1. Assinando o Serviço

Ao se mostrar interessado em algum evento, o cliente envia ao servidor a mensagem de <*create-subscription*>. Nela é informada a classe de evento de interesse. No caso do servidor poder atender a requisição, este envia um <*rpc-reply*> contendo o *id* da subscrição no elemento <*data*>.

A partir deste momento, sempre que houver o evento para o qual uma subscrição foi solicitada, o servidor envia a notificação utilizando o elemento <notification>, contendo as seguintes informações: classe do evento, *id* da subscrição, número de sequência, dia e hora.

É prevista também a modificação das propriedades de uma subscrição, usualmente da classe de evento. Para isto é definido o elemento <modify-subscription> contendo as seguintes informações: *id* da subscrição, classe do evento. O retorno é um <rpc-reply>. No caso de sucesso, o servidor encapsula um <ok>; e no caso de falha este encapsula um <rpc-error>.

Por último, pode-se cancelar a subscrição, através do elemento <*cancel-subscription*>, contendo o *id* da subscrição. O retorno é um <rpc-reply>. No caso de sucesso, o servidor encapsula um <ok>; no caso de falha este encapsula um <rpc-error>.

Esta troca de mensagens é mostrada de forma esquemática na Figura 3.3 abaixo. De maneira a simplificar a figura, não é mostrado os elementos <rpc> e <rpc-reply> que encapsulam, respectivamente, as requisições e as respostas após a troca de mensagens <hello>.

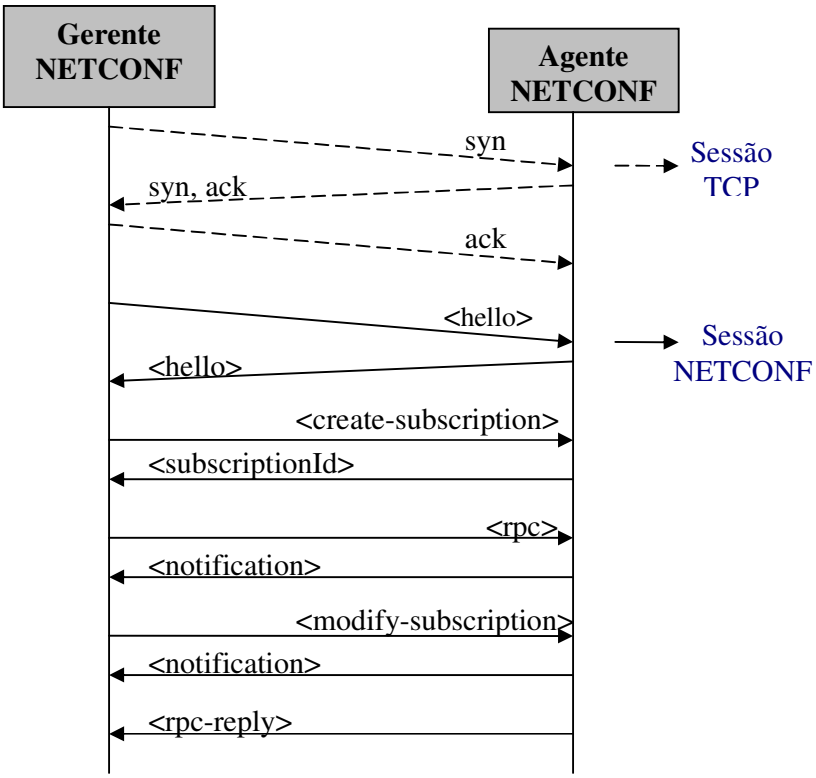


Figura 3.3: Serviço de Eventos.

3.8.2. Classes de eventos

Algumas classes de eventos são identificadas e comentadas a seguir:

- *Configuration*: Refere-se ao uso de uma das seguintes operações: <copy-config>, <delete>, <edit-config>. No geral, corresponde à troca, remoção ou adição de um hardware ou software do dispositivo;
- *Fault*: Refere-se a qualquer falha ou erro que normalmente gera um alarme;
- *State*: Indica a mudança de estado da entidade, sendo o estado uma condição ou estágio na existência desta entidade;
- *Audit*: São informações de qualquer ação específica;
- *Data dump*: Contém informação do sistema, sua configuração e seu estado;

- *Maintenance*: Indica o início, o meio e/ou o fim de uma ação de manutenção manual ou automática. Por exemplo, pode-se gerar um evento notificando o início de um *backup* e outro evento que notifica o fim da ação de *backup*;
- *Metrics*: Contém informações geralmente de métricas de desempenho;
- *Heart beat*: É um evento gerado periodicamente para manter ou testar um canal de comunicação
- *Information*: É um evento de interesse que está dentro do comportamento operacional previsto, mas que, no entanto, não se encaixa nas definições anteriores.

3.8.3. Mapeamento para os protocolos de transporte

A notificação de eventos é mapeada para os protocolos de transporte já definidos para o NETCONF: SSH, BEEP e SOAP. Um exemplo de mapeamento do serviço de evento para SOAP encontra-se no Anexo A.

A grande questão é que uma notificação é um evento de mão única. Ao acontecer o evento, o Agente NETCONF deve enviar a notificação como uma entidade única que não necessita de resposta. Este modelo de comunicação é o oposto do modelo tradicional RPC, no qual para uma requisição (do cliente) sempre existe uma resposta (do servidor). Como o SOAP é transportado por HTTP, este problema também existe para o mapeamento para SOAP. Por isso, esta questão deve ser resolvida ao se implementar o serviço de notificação de eventos.

3.9. NETCONF x Modelo de Dados

Uma decisão importante da especificação foi separar o protocolo do Modelo de Dados. Esta separação é vista na Figura 3.1, onde a camada 4 (conteúdo) é mostrada mas não é detalhada no protocolo. No próprio protocolo é explicado que o NETCONF deve ser independente da linguagem de definição de dados e do modelo de dados utilizado para descrever a configuração e o estado dos dados. Mais ainda, dada à natureza proprietária dos

dados de configuração a serem utilizados, a especificação deste conteúdo depende da implementação do NETCONF. Por isso, um esforço futuro deve ser empreendido no sentido de padronizar um modelo de dados.

No entanto, a definição de um modelo de dados é fator crucial para adoção do protocolo, de modo que em um futuro próximo a junção de um modelo de dados padronizado com a implementação do protocolo não seja um obstáculo. A adoção de um modelo de dados é necessária para atingir a interoperabilidade entre entidades NETCONF (agentes NETCONF).

Importante neste ponto esclarecer as diferenças entre um Modelo de Informação e um Modelo de Dados [58]. Este dois modelos são muito parecidos, causando uma grande confusão ao utilizar os dois termos. O mais importante é que os dois diferem quanto ao propósito de cada um.

Um Modelo de Informação (*Information Model* - IM) define em um nível alto de abstração o modelo conceitual dos objetos gerenciados, independentes de uma implementação específica ou protocolos utilizados para o transporte dos dados. De forma a manter o IM o mais claro possível, este deve esconder qualquer detalhe de protocolo e implementação. Uma característica importante de todo IM é a definição das relações entre os objetos gerenciados.

O IM é utilizado pelo projetista para descrever o ambiente gerenciado, pelo operador para entender os objetos modelados e pelos desenvolvedores como um guia para as funcionalidades que devem ser descritas e codificadas em Modelos de Dados (*Data Model* - DM).

A linguagem para a descrição de um IM pode ser informal, como o Português ou Inglês. Outra forma de descrever o modelo é via UML. Esta tem a vantagem de utilizar conceitos da orientação a objetos como: abstração, herança, encapsulamento e métodos. Outra vantagem do UML é que o resultado da modelagem é mostrado em figuras geométricas específicas da linguagem, interligadas por linhas com vários tipos de terminações, oferecendo assim uma visão gráfica do modelo que facilita o seu entendimento.

No entanto, a maioria dos modelos de gerência padronizados são DM. Exemplos de DM são:

- SMI para a definição de módulos das MIBs;
- SPPI (*Structure of Policy Provising Information* [59]) para a definição de módulos das *Policy Information Bases* (PIBs [60]);
- CIM, desenvolvido no DMTF. Apresenta a parte gráfica em UML e a parte textual em *Meta-Object Facility* (MOF [61]).

Um DM é descrito numa linguagem formal, na qual são definidos os objetos gerenciados num nível de abstração baixo. Incluem detalhes específicos da implementação e do protocolo. É voltado, como dito anteriormente, para desenvolvedores.

Existe uma relação entre um IM e um DM. O IM pode ser instanciado em vários DMs. Por exemplo, um IM pode ser instanciado em SMI, SPPI ou MOF.

Consideremos os módulos da MIB. Uma MIB é definida numa RFC. Esta é um DM, pois detalha as OID, estruturas de índices, valores máximos de acesso para as variáveis e outras tantas informações de implementação. No entanto, algumas RFCs contêm algum tipo de descrição informal explicando partes do modelo por trás do módulo da MIB. Esta informação é um IM. Um exemplo é a RFC 2863 para o grupo *Interfaces*.

3.10. Considerações finais

Neste capítulo foi detalhado o protocolo NETCONF. Foram mostradas as suas camadas; as operações básicas: *get, get-config, edit-config, kill-session, lock, unlock, copy-config, delete-config*; o modelo de comunicação RPC; os requisitos do protocolo de transporte; a maneira de se estender o protocolo a partir de novas capacidades; o mapeamento para SOAP; e como deve funcionar o serviço de notificações e eventos.

No entanto, a especificação do protocolo não define dois aspectos importantes na busca por sua ampla aceitação e utilização: um modelo de dados e uma arquitetura de implementação. Por isso, no capítulo seguinte é apresentada uma proposta de arquitetura de implementação e uma proposta de modelo de dados. A arquitetura foi validada num experimento prático de configuração de VLANs para switches de diferentes fabricantes. O protótipo da arquitetura é apresentado, bem como o modelo de dados para a configuração das VLANS. Tabelas e gráficos são apresentados para avaliar o custo do protocolo NETCONF ao se utilizar HTTP e HTTPS como transporte para as mensagens do protocolo no mapeamento SOAP.

Capítulo 4

Arquitetura, Modelo de Dados e Implementação

Neste capítulo é proposta uma arquitetura para implementação e validação do protocolo NETCONF utilizando SOAP como transporte. É proposto um modelo de dados e são descritos os detalhes do protótipo e da implementação durante a fase de validação da arquitetura nos cenários de consulta à tabela de rotas IP em computadores, e na configuração de VLANs para switches de diferentes fabricantes.

4.1. Arquitetura Proposta

A arquitetura proposta tem por objetivo primordial o respeito total às mensagens definidas pelo protocolo, sendo ao mesmo tempo simples e flexível. Este pré-requisito é posto porque na fase de estudos e pesquisa foi verificado que nas poucas implementações disponíveis, muitas delas não respeitavam o protocolo quanto ao formato das mensagens [62]. Outras implementações não utilizam qualquer dos mapeamentos definidos para o transporte das mensagens.

A implementação de um fabricante de equipamentos de rede baseia-se na idéia de um servidor de configuração [63]. O gerente se conecta a este servidor, autentica-se e depois é habilitado a trocar as mensagens. A arquitetura proposta neste trabalho não segue esta linha

por acreditar que este servidor pode se tornar um provável gargalo para a execução do protocolo ao servir vários gerentes, tornando-se assim sobrecarregado. No caso de uma falha, não se poderia mais acessar o dispositivo para obter e/ou alterar algum aspecto da sua configuração.

Além das mensagens, levou-se em conta a questão do estabelecimento da sessão NETCONF. A sessão é o meio pelo qual garante-se segurança, principalmente nas trocas das mensagens, a partir da validação do usuário. A sessão também é responsável por atender as solicitações na ordem que foram recebidas pelo agente. Ao usar o HTTP para encapsular o SOAP (método mais comum), torna-se possível acessar o objeto gerenciado via Internet. Haja visto que a configuração de dispositivo é uma informação sensível e confidencial, é necessário um meio de garantir que pessoas não-autorizadas acessem os dados. Este meio é via o estabelecimento da sessão.

4.1.1. Outras considerações sobre a arquitetura

Nas propostas iniciais do protocolo NETCONF não havia um transporte mandatório para as mensagens entre o gerente e o agente. A partir da versão 10 o uso de SSH tornou-se padrão. Como um dos objetivos do grupo de pesquisa é estudar a viabilidade dos *Web services* para alguns problemas na área de Gerência de Redes [64][65], a arquitetura proposta concentrou-se no mapeamento do protocolo para SOAP, apesar de acrescentar um *overhead* no processamento da mensagem e aumentar o tráfego gerado. Ainda sobre os *Web Services*, assim como na especificação do NETCONF, não é definido o uso de UDDI ou qualquer outro serviço de registro e localização do agente, como previsto na arquitetura SOA.

Outro fator importante quando da definição da arquitetura foi a possibilidade de, no futuro, ter o protótipo (agente) implementado no próprio dispositivo gerenciado. Como a proposta principal do trabalho é implementar e validar o protocolo NETCONF via SOAP, onde é instalado o agente não é de relevância no momento, conforme pode ser observado na Figura 4.1 abaixo. Pode-se observar que o computador onde está o agente também pode ter ou não o gerente instalado. Sendo este mais um detalhe na tentativa de manter a arquitetura flexível.

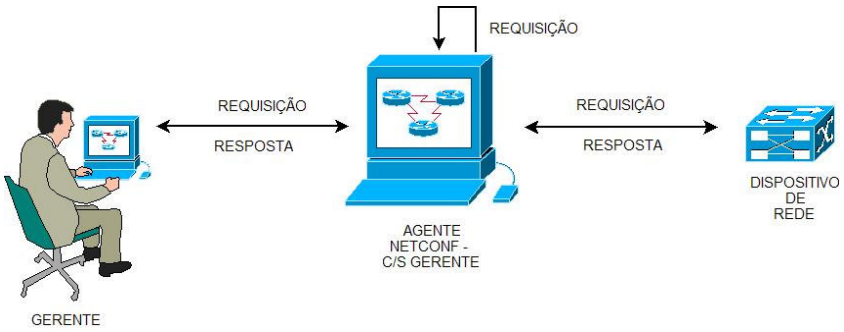


Figura 4.1: Localização do agente NETCONF.

Além do mais, não temos disponível - na forma de uma parceria, por exemplo - o acesso ao código fonte de um dispositivo de rede no qual pudéssemos alterar ou acrescentar as bibliotecas e o código implementado. Assim, a arquitetura reflete esta realidade e é definida pensando em ser executada em um computador comum. Sendo possível, neste caso, que o próprio computador seja o objeto gerenciado ou, a partir dele, acessar um dispositivo de rede qualquer (*gateway*).

O protocolo adota implicitamente um mapeamento de um gerente para vários agentes e um agente para cada dispositivo. Um incremento ao protocolo, proposto nesta arquitetura, é que o mesmo agente possa lidar com mais de um dispositivo, criando assim o mapeamento de um agente para vários dispositivos. A maior vantagem é a flexibilidade de manutenção dos dispositivos de rede. O que atende, principalmente, à situação prática de várias operadoras de telecomunicações, na qual vários operadores precisam ter acesso a vários dispositivos. Problemas decorrentes desta proposta ainda não foram analisados mais profundamente, ficando para um trabalho futuro.

No documento do protocolo não é especificado nada com relação à notificação de eventos, como existe no SNMP na forma da mensagem TRAP. Na lista de discussão do grupo muito se debateu sobre a necessidade ou não de se acrescentar ao protocolo esta possibilidade. Alguns participantes ponderaram que, no âmbito do IETF, já se tem o SYSLOG para este fim. Desta maneira, não se chegou a um consenso sobre a questão. Assim, a notificação de eventos foi definida a partir de uma submissão na forma de um *draft*, como complemento ao protocolo.

Novamente, na fase de estudos, foi definido que seria importante para esta arquitetura contemplar esta função. Principalmente, porque se poderia ter mensagens assíncronas a partir do agente para um outro qualquer. O que é o funcionamento inverso do protocolo, que estabelece que as suas mensagens sejam síncronas, iniciando sempre a partir do gerente para o agente.

Boa parte do documento do protocolo é a respeito do filtro de mensagens *subtree.* Na fase de estudos e pesquisa foi constatado que todas as opções definidas para este filtro são perfeitamente substituíveis por expressões XPATH. No entanto, para o agente entender tais expressões, uma nova capacidade deve ser adicionada ao conjunto de capacidades base. Assim, de forma a manter a arquitetura flexível, um módulo de filtro é oferecido. Fica, então, sob responsabilidade da implementação decidir qual filtro será utilizado: *subtree*, *xpath* ou os dois.

Outra parte essencial do trabalho é o respeito às operações pré-definidas, além das próprias mensagens. O agente deve refletir esta premissa. A dúvida para esta arquitetura, quanto a este aspecto, seria se existiria ou não um módulo para cada operação. Para manter a arquitetura simples, optou-se por manter as operações num único módulo.

4.1.2. Detalhando a Arquitetura

A seguir é mostrada a Figura 4.2 com a arquitetura proposta por este trabalho e os seus módulos funcionais.

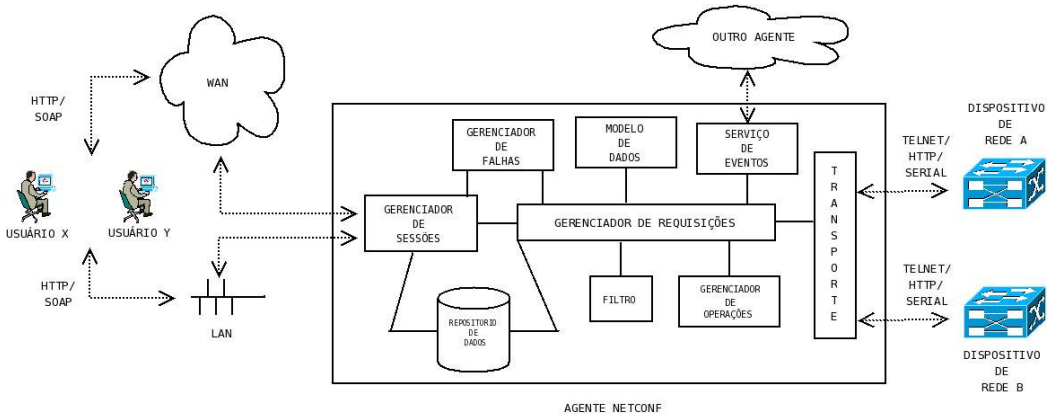


Figura 4.2: Arquitetura proposta.

- **Serviço de Eventos**

Este módulo é responsável por montar e enviar um evento para os gerentes que se inscreveram no serviço de eventos. Ao ser acionado pelo Gerenciador de Requisições, ele monta a mensagem de evento e a envia para o destino especificado quando da subscrição ao serviço de eventos. Deve suportar pelo menos a classe de eventos *configuration*.

- **Gerenciador de Requisições**

Este módulo é responsável por:

- Receber as mensagens <rpc>, validá-las acionando o módulo Modelo de Dados, repassá-las ao Gerenciador de Operações e devolver as respostas ao Gerente;
- Ser um intermédiario entre os demais módulos;
- Acionar o Gerenciador de Falhas em caso de não-conformidade da mensagem com o protocolo ou com o Modelo de Dados;
- Analisar a requisição, verificar se existe algum evento registrado e acionar o Serviço de Eventos. Para cada operação e evento, o Gerenciador de

Requisições deve consultar o módulo Repositório de Dados para identificar se há algum consumidor para aquele evento e acionar o Serviço de Eventos.

- **Repositório de Dados**

  - Armazena informações sobre cada sessão NETCONF aberta. Para cada sessão é armazenado o seu *id*, sendo que toda mensagem deve estar associada a uma sessão;
  - Armazena as versões das configurações, dando suporte às capacidades adicionais como a *candidate*, por exemplo;
  - Armazena os consumidores subscritos no serviço de eventos e a classe de eventos de cada um;
  - Armazena os modelos de dados para posterior validação das mensagens recebidas pelo módulo Modelo de Dados.

- **Gerenciador de Falhas**

Ao ser verificada uma não-conformidade pelo Modelo de Dados da mensagem recebida, ou a incapacidade do agente de responder ou de entender a mensagem recebida, este módulo gera a mensagem de falha e a envia de volta ao gerente.

- **Gerenciador de Sessões**

Gera, valida e gerencia a sessão NETCONF. A cada requisição é verificado se a sessão expirou por tempo excedido, ou se o usuário e senha informados já estão em uso. Nestes casos, o Gerenciador de Falhas é acionado e gerada uma falha. Nenhum outro módulo é acionado. As sessões são armazenadas no Repositório de Dados.

- **Gerenciador de Operações**

Ao receber a mensagem <rpc> este encapsula uma das 9 (nove) operações do protocolo. Este módulo é responsável por:

  - acionar a classe ou método que trata a operação;
  - chamar o Transporte;
  - receber a resposta do dispositivo;
  - acionar o módulo Filtro para filtrar os dados recebidos, se um filtro foi indicado na operação;
  - devolver os dados ao Gerenciador de Requisições.

- **Transporte**

Este módulo é responsável por realizar o acesso ao dispositivo gerenciado utilizando o transporte por ele suportado, tais como: TELNET, porta serial, http, TFTP, SMTP ou outros.

- **Filtro**

Módulo responsável por tratar a mensagem recebida na presença do elemento <filter>. Após obter a resposta do dispositivo, o filtro trata os dados recebidos e monta a mensagem de resposta. Esta é encaminhada ao Gerente de Operações que a envia para o Gerente. Deve suportar o filtro básico do protocolo, *subtree*. Qualquer outro filtro suportado, exemplo XPATH, deve ser de sua responsabilidade.

- **Modelo de Dados**

O módulo Modelo de Dados recupera o XML Schema do Repositório de Dados. A mensagem recebida então é validada no módulo. Se estiver conforme, é devolvida ao Gerenciador de Requisições. Se estiver não-conforme, é informado para o Gerenciador de Requisições que um erro foi encontrado.

4.2. Proposta de Modelo de Dados

Conforme comentado na Seção 3.9, a grande crítica ao protocolo NETCONF refere-se à ausência de um modelo de dados [66]. Por isso, este trabalho tem como um dos seus objetivos definir um modelo de dados. Inicialmente, o modelo refere-se à configuração de VLANs. As razões para isto são explicadas a seguir.

Na fase de implementação, ver Seção 4.3, o principal objetivo foi realizar uma configuração real de algum dispositivo de rede através do protocolo NETCONF. No laboratório existem dois switches do nosso grupo de pesquisa em produção. Nos dois, foi realizado um estudo de como são os comandos da interface de linha e como funciona a interação via *Web-Browser*. O próximo passo foi definir se o teste a ser realizado envolveria a totalidade ou parte de suas configurações. Neste momento foi definido que um bom teste seria configurar as suas VLANs, principalmente porque a sintaxe dos comandos de linha diferem de um para o outro, como diferem nos passos para configurar via *Web-Browser*.

Como o mecanismo de transporte utilizado é o SOAP, foi natural pensar num arquivo XML que representasse os comandos de linha de cada um dos switches. Desta maneira, o gerente ao configurar a VLAN deveria escrever um arquivo XML e, ao realizar a operação <*edit-config*>, ele informaria este arquivo com a configuração. Do lado do agente, ele receberia a operação e o módulo Gerente de Operações ficaria responsável por montar os comandos da interface de linha, realizando um *parser* e, depois, acionaria o módulo de transporte para repassá-los via TELNET.

No entanto, foi verificado que desta forma, para cada fabricante de *switch*, o gerente

deveria saber exatamente a sintaxe de cada um deles para montar o arquivo XML de configuração. Por isso, foi estudada a especificação das VLANs [67] e uma contribuição adicional deste trabalho foi a definição de um modelo de dados baseado em XML SCHEMA, conforme indicado no Apêndice B, para configuração básica de VLANs para o protocolo 802.1Q (VLANs do tipo *tag*) e VLANs baseada em porta. O objetivo é facilitar o trabalho do gerente ao permitir que ele defina apenas as operações que deseja realizar, tais como: criar, remover ou alterar uma VLAN; ficando livre de se preocupar com a sintaxe. Com o uso desta abstração, o gerente precisa informar apenas o fabricante e os dados de configuração. Na Figura 4.3 abaixo é apresentado um exemplo de configuração de VLANs em XML.

```
<vlans>
      <vlan>
            <operation>create</operation>
            <name>f_1</name>
            <tagid>10</tagid>
            <address>
                  <ip>10.10.1.10</ip>
                  <mask>255.255.255.255</mask>
            </address>
            <port>
                  <option>add</option>
                  <portlist>30-32</portlist>>
                  <type>tagged</type>
            </port>

      <advertisement>true</advertisement>
      </vlan>
</vlans>
```

Figura 4.3: XML para configurar uma VLAN.

O XML Schema para configuração de VLANs define os seguintes campos:

- Operation: Operação a ser realizada: create - criar uma nova VLAN; delete - remover uma VLAN existente; config - para configurar uma VLAN existente;
- Name: Nome da VLAN para facilitar a identificação. Geralmente o nome reflete o papel da VLAN na rede;
- Tagid: Identificação numérica - *tag*, uma marca a ser adicionada ao frame Ethernet se for utilizar o protocolo 802.1Q . A faixa de numeração geralmente é

diferente para cada fabricante;

- Address: Composto por dois campos. Utilizado principalmente por switches que também desempenham papel de roteadores:
    - Ip: Define o endereço IP ;
    - Mask: Define a máscara do endereço IP;
- Port: Uma VLAN é associada às portas de um switch:
    - Option: Define a operação a ser realizada: add – adicionar a porta; delete - remover a porta da VLAN;
    - Portlist – Relação de porta(s). Como as portas são separadas pelo sinal de menos, no XML Schema foi definido como sendo uma string;
    - Type: Define o tipo de porta: tagged – porta utiliza *tag*; untagged – porta não utiliza *tag*; forbidden – para alguns fabricantes, define que a porta não pode ser membro de uma VLAN e que não pode se tornar membro de uma VLAN dinamicamente;
- Advertisement: Para alguns fabricantes, se o campo estiver presente e verdadeiro indica que o switch pode enviar pacotes externos para outras VLANs que se juntarem à VLAN;
- NoBroadcast: Para alguns fabricantes, se o campo estiver presente e verdadeiro, indica que o switch deve prevenir o encaminhamento de tráfego broadcast, multicast e tráfego unicast desconhecido;

Da maneira que o XML Schema foi definido, cada um dos campos é uma sequência que pode estar ou não presente no XML. Assim, consegue-se a flexibilidade para construir XMLs que diferem entre si do lado do gerente.

Do lado do agente, a decisão de implementação foi manter o Gerenciador de Operações responsável por montar o comando da CLI de cada fabricante. O XML de configuração da VLAN é encapsulada no elemento <*device*>. Dentro deste elemento é definido o nome do dispositivo através da tag <*name*>, no caso “extreme” ou “dlink”. O elemento <device> é encapsulado no parâmetro <*config*> da operação <*edit-config*>. O Gerenciador de Operações verifica que se trata de uma solicitação de configuração de VLAN através da execução da expressão XPATH “/device/vlans” no XML recebido . Se o resultado da expressão não for

vazio, é extraído o nome do fabricante informado através da expressão XPATH "/device/name". A partir deste nome, é instanciada a classe correspondente ao dispositivo, ela transforma o XML em comandos de linha e chama o módulo Transporte para conectar ao *switch* e passar o comando de configuração.

4.3. Protótipo e Implementação

Alguns protótipos do NETCONF já estão disponíveis [68]. Um deles é o Ensuite (*Extended NETCONF Suite*) [69] distribuído sob a licença *Lesser General Public License* (LGPL). Este foi implementado na linguagem de programação PYTHON, usando como transporte o SSH. Foi desenvolvido pelo mesmo grupo que implementou o YENCA [62].

A principal diferença do nosso protótipo, além de usar a linguagem de programação Java, é o uso do HTTP e do HTTPs como transporte, abrindo a perspectiva para o uso do protótipo com os conceitos de *web-services* e SOA. Principalmente, porque o WSDL da proposta já prevê o uso do estilo de troca de mensagens como *document*, que é definido pela Organização de Interoperabilidade para W*eb-services* – (*Web Services – Interoperability Organization* (WS-I) [70] - como o adequado para interoperabilidade entre linguagens de programação e plataformas.

Outra diferença é a implementação de um sistema básico de evento/notifacação através da extensão ao WSDL original. As mensagens suportadas são a <*create-subscription*> e <*cancel-subscription*>. Por enquanto, é possível gerar eventos apenas do tipo *configuration* para as operações <*get*>, <*get-config*>, <*edit-config*>, <*copy-config*>, <*delete-config*> realizadas no agente, enviando o evento para o consumidor informado em <*create-subscription*>, que pode ser um terceiro computador no qual o agente ou o gerente NETCONF esteja instalado, Figura 4.2.

O protótipo pode ser visto na Figura 4.4 abaixo:

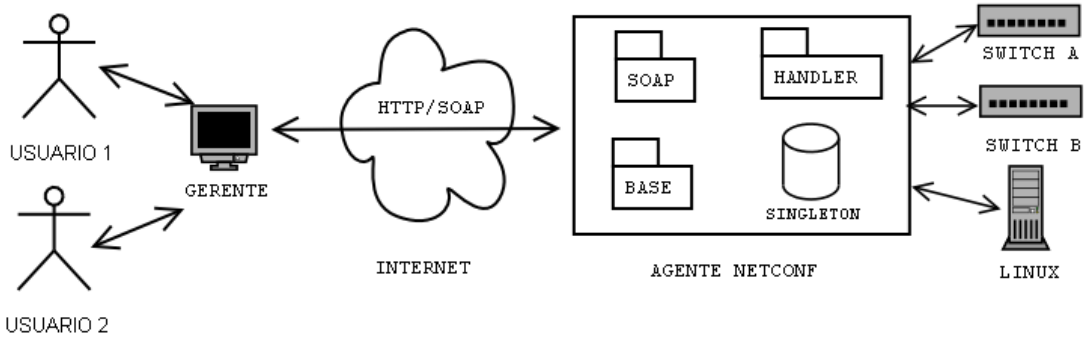


Figura 4.4: Protótipo NETCONF.

O ponto de partida para a implementação do protótipo foi utilizar o WDSL definido em [netconfsoap]. A partir do WSDL, mais o serviço definido na própria especificação, foi utilizada a ferramenta WSD2Java do Axis para criar as classes do protótipo. Como o mapeamento aponta para a seguinte URN: ***urn:ietf:params:xml:ns:NETCONF:base:1***, a ferramenta criou o pacote no seguinte caminho /1/base/netconf/ns/xml/params/ietf. Neste pacote, representado na Figura 4.4 pelo pacote BASE, estão as classes das operações do protocolo. Todas as classes, de acordo com a seção de tipos do WSDL, têm concatenado ao seu nome a palavra Type. Temos, então, as classes: *GetType*, *GetConfigType*, *EditConfigType*, por exemplo. Mais detalhes podem ser vistos no Apêndice A.

O outro pacote é o SOAP, também representado na Figura 4.4. Este foi criado no seguinte caminho: /1/soap/netconf/ns/xml/params/ietf. Neste pacote temos as classes *stubs* e *skeletons* e a implementação da interface do serviço NETCONF.

Originalmente, o WDSL define duas portas (*ports*) para o serviço: hello e rpc. A porta hello é a chamada de procedimento remoto para a troca de mensagens <hello>. Para a troca de mensagens RPC, onde são encapsuladas as operações do protocolo, é definida a chamada de procedimento remoto <rpc>. Para implementar o Serviço de Notificação e Eventos, foi necessário adicionar a porta *rpcOneWay* que permite a troca de mensagens assíncronas. Uma vez que a mensagem de notificação é enviada pelo agente e nenhuma resposta é esperada, duas ações foram tomadas: a resposta à chamada *rpcOneWay* foi definida como *void*. Na

classe *stub NetConfSoapStub*, a porta *rpcOneWay* invoca o método *invokeOneWay* da instância da classe *CALL*. Desta maneira, resolveu-se o problema apontado na Seção 3.8.3.

Mapeamento da Arquitetura no Protótipo

Uma vez que a ferramenta WSD2Java criou os pacotes SOAP e BASE, o protótipo foi implementado baseado nestes dois pacotes. No pacote BASE ficaram as classes das operações do serviço, como gerado pela ferramenta.

No pacote SOAP estão os módulos: Repositório de Dados, representado na classe SingletonDataBase que implementa o *design pattern singleton* (Seção 4.3.1); o Gerenciador de Sessões e o Serviço de Eventos. Os dois primeiros módulos foram introduzidos neste pacote, porque ele é a porta de entrada do Serviço Web. Assim, o controle de Sessão NETCONF fica facilitado. A Sessão NETCONF utiliza a infra-estrutura de Sessão do próprio HTTP (Seção 4.3.1) para verificar, por exemplo, se a Sessão expirou por falta de uso ou se o usuário e senha informados já estão autenticados no agente. O Serviço de Eventos está no pacote SOAP, porque a porta *rpcOneWay* precisa ser especificada no STUB para que o cliente possa invocá-lo.

No pacote HANDLER estão os demais módulos da arquitetura: Filtro, Gerenciador de Falhas, Gerenciador de Requisições, Modelo de Dados, Gerenciador de Requisições e Transporte.

De forma a possibilitar o uso de vários usuários ao mesmo tempo e que as mensagens RPC sejam processadas de forma sequêncial, como o protocolo estabelece, foi utilizado o padrão de desenvolvimento *singleton* para o módulo Repositório de Dados, ver Seção 4.3.1. Este padrão garante que uma única instância de uma classe seja instanciada enquanto o sistema estiver ativo. Neste *singleton* é armazenado, por exemplo, as sessões NETCONF. A partir dele também é possível saber qual é o usuário responsável por um <lock>. Assim, garante-se que apenas ele possa alterar um arquivo de configuração.

Do lado do gerente, o protótipo fornece uma interface de linha de comando via menu, conforme Figura 4.5 a seguir.

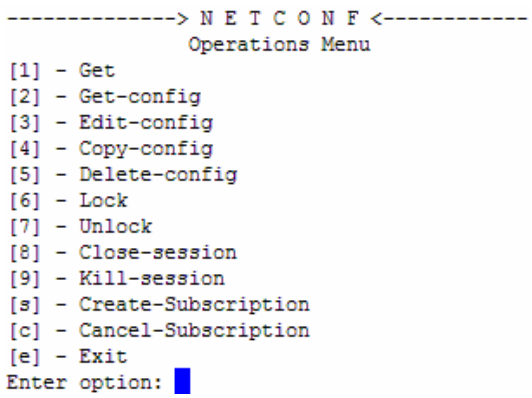


Figura 4.5: Gerente NETCONF.

Neste menu, o gerente informa a operação que deseja realizar. Uma decisão de projeto importante, consoante com o suporte de várias sessões simultâneas, foi permitir que as operações de consulta - <*get*> ou <*get-config*> - só fossem possíveis se não houvesse um <*lock*> na configuração *running*.

Para o gerente realizar alguma operação que altera a configuração - <*edit-config*>, <*copy-config*>, <*delete-config*> - deve-se primeiro realizar um <*lock*>. Após realizar a alteração desejada, o gerente deve fazer um <*unlock*>, caso contrário, nenhum outro usuário conseguirá alterar a configuração.

Outra opção oferecida ao gerente é fazer uma subscrição no sistema de eventos. Neste momento, o gerente informa o endereço IP do consumidor de eventos. No protótipo o produtor de eventos é sempre o agente ao qual o gerente se conectou assim que abriu a interface.

Na Figura 4.6 é apresentado um exemplo de uma mensagem do protótipo:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <rpc message-id="5" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
            <createSubscription>
                <eventClass>
                    <configuration/>
                </eventClass>
                <consumer>10.1.1.2</consumer>
            </createSubscription>
        </rpc>
    </soapenv:Body>
</soapenv:Envelope>
```

Figura 4.6: Mensagem Netconf para a subscrição no serviço de eventos.

Nesta mensagem <rpc> é solicitada a subscrição no Serviço de Notificação e Eventos. A solicitação é realizada pelo menu da interface selecionando a opção "s" A mensagem <*createSubscription*> é enviada do gerente para o agente. Nela é informada automaticamente a classe de evento <configuration>, já que é a única classe de eventos suportada neste protótipo. É repassado também o IP ou o nome da máquina que será a consumidora do evento, elemento <consumer>.

4.3.1. Detalhes da implementação

Na Seção 4.1 foi descrita a arquitetura proposta. Um dos componentes da arquitetura é o módulo Repositório de dados. Na implementação, optou-se por utilizar o *design pattern singleton* para implementar tal componente ao invés de se utilizar um banco de dados nativamente XML. Este padrão de desenvolvimento garante que em uma linguagem orientada a objetos existe uma única instância da classe durante todo o tempo de execução da aplicação. A opção pelo *singleton* se deu, principalmente, para acelerar o desenvolvimento do agente.

Neste *singleton* são utilizadas tabelas *hash* como estrutura de dados. As tabelas *hash* servem para controlar:

- as sessões – chave: nome do usuário; conteúdo: objeto da sessão HTTP;
- os locks das configurações - chave: nome da configuração (*running*,*candidate*

ou *validate*, as duas últimas se suportadas); conteúdo: objeto da sessão HTTP;

- inscrições no serviço de evento - chave: número único de identificação por sessão; conteúdo: objeto que representa detalhes da inscrição.

Com relação ao objeto da sessão HTTP, este é extraído a partir do *cast* da sessão HTTP do TOMCAT para o tipo *AxisHTTPSession* retirado do contexto da mensagem da infra-estrutura do AXIS [71]. A partir do contexto da mensagem também é possível retirar o nome do usuário informado pelo gerente.

O filtro *subtree-filtering* foi disponibilizado a partir de um arquivo XSLT disponível no ENSUITE.

4.3.2. Dificuldades encontradas na Implementação

Uma grande dificuldade foi encontrada na fase de desenvolvimento: o WSDL fornecido na documentação do mapeamento do protocolo NETCONF para o mecanismo de transporte SOAP [31]. Esta dificuldade também foi reportada por [72]. O WSDL fornecido faz uso abundante de tipos abstratos e herança. No artigo citado, o problema foi contornado reescrevendo o WDSL mantendo-se fiel ao formato da mensagem. Neste trabalho, a dificuldade foi contornada de outra maneira, o que resultou num WSDL diferente do original mas, também, teve-se a preocupação de manter as mensagens como definidas pelo protocolo.

A maneira encontrada para contornar o problema anterior gerou uma segunda dificuldade. Entender como a plataforma AXIS faz o mapeamento do XML para tipos de dados da linguagem Java. A forma mais simples é o mapeamento básico do XML definido pela especificação JAX-RPC[73]. Esta especificação da SUN define uma API Java para chamadas de procedimentos remotos, RPC, baseada em XML.

Para serializar e desserializar, sem que o programador precise escrever seu próprio código para isto, as classes Java devem seguir o padrão JavaBeans [74]. Neste padrão, é necessário definir os métodos acessores get/set para os campos da classe. Após isto, é preciso indicar no *Web Service Deployment Descriptor* (WSDD) - arquivo que contêm todos os *Web Services* e suas particularidades em qualquer instalação do AXIS [75] - qual o mapeamento desejado

para a classe Java entre os tipos básicos suportados no XML SCHEMA.

No entanto, para este projeto, apenas o mapeamento anterior não foi suficiente. Foi necessário aprender um outro mapeamento, principalmente, para obter um XML como o definido pelo protocolo, e para conseguir inserir uma *tag* XML dentro da outra. Este mapeamento utiliza a tag <*typeMapping*> no WSDD. Através dela, é possível construir novos serializadores e desserializadores, e mapear classes Java para tipos específicos do XML SCHEMA. Na figura abaixo segue um exemplo de um mapeamento de classe Java para XML.

```
<typeMapping
   xmlns:ns="urn:ietf:params:xml:ns:netconf:base:1.0"
   qname="ns:eventClassType"
   type="java:_0._1.base.netconf.ns.xml.params.ietf.EventClassType"
   serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
   deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
   encodingStyle=""
/>
```

Figura 4.7: Mapeamento classe Java para tipo XML SCHEMA.

No exemplo de Figura 4.7, o campo **xmlns** *(*XML *Namespace)* define o *namespace* (espaço de nomes) do XML. No caso, este foi retirado do WSDL original. O campo **qname** *(Qualified Name)* define o nome qualificado da classe Java. O campo **type** define a classe Java a ser mapeada para o *qname* definido anteriormente. Os campos ***serializer*** e ***deserializer*** definem as classes encarregadas de realizar o *marshaling* e *unmarshaling* das classes Java para XML e vice-versa. O campo ***encondingStyle*** define o estilo de codificação, neste caso o campo está vazio. Uma codificação possível seria: HTTP://schemas.xmlsoap.org/soap/encoding/ para codificação SOAP.

Além de informar o mapeamento desejado para a classe Java no WSDD todas as vezes que este arquivo é alterado, é necessário utilizar as ferramentas de administração do AXIS para realizar o *undeploy* do *Web Service* e depois realizar o *deploy* do *Web Service* para o AXIS entender o novo mapeamento. Aliado à alteração do WSDD e ao uso de métodos acessores para os campos da classe, foi preciso entender que, na versão do AXIS utilizada, a classe criada e que será mapeada em XML, necessita fazer a chamada da classe estática *org.apache.axis.description.TypeDesc*. Esta classe define os metadados da classe da qual se deseja fazer o mapeamento. Ela define também o relacionameno entre o(s) campo(s) da classe

e a(s) tag(s) XML, de maneira que é possível definir um nome diferente para a tag XML em relação ao nome da variável da classe mapeada. Na Figura 4.8 é ilustrado este relacionamento.

```
private static org.apache.axis.description.TypeDesc typeDesc =
      new org.apache.axis.description.TypeDesc(EventType.class, true);

static {
      typeDesc.setXmlType(new javax.xml.namespace.QName("urn:ietf:params:xml:ns:netconf:base:1.0",
"eventType"));
      org.apache.axis.description.ElementDesc elemField = new org.apache.axis.description.ElementDesc();

      elemField.setFieldName("eventClass");
      elemField.setXmlName(new javax.xml.namespace.QName("urn:ietf:params:xml:ns:netconf:base:1.0",
"eventClass"));
      elemField.setXmlType(new javax.xml.namespace.QName("urn:ietf:params:xml:ns:netconf:base:1.0",
"eventClassType"));
      elemField.setMinOccurs(0);
      elemField.setNillable(false);
      typeDesc.addFieldDesc(elemField);
}
```

Figura 4.8: Mapeamento variável da classe em tag XML.

A variável estática **typeDesc,** ao ser criada, é relacionada à classe fornecendo o nome da classe como parâmetro, no exemplo: EventType.class. Por sua vez, a partir da realização do <*typeMapping*> da classe no WSDD e o deploy do WEB SERVICE, o AXIS é capaz de associar o .class ao tipo XML SCHEMA adequado.

Na seção ***static*** da Figura 4.8 são definidos para ***typeDesc***: o tipo XML, novamente relacionado ao <*typeMapping*>, e o(s) elemento(s) que deve(m) aparecer no XML e o(s) seu(s) campo(s) correspondente(s). Para isto, é criada a variável **elemField**. Esta variável aceita configurar os seguintes estados:

- *setFieldName* – configura o nome do campo;
- *setXmlName* – configura o nome da tag XML;
- *setXmlType* – relaciona com a classe Java mapeada no WSDD;
- *setMinOccurs* – define o número mínimo de ocorrência do caso; se for zero, indica que não é obrigatório a tag aparecer no XML;
- *setNillable* – define que, se o campo estiver vazio, deve aparecer ou não como

conteúdo a palavra NILL; *false* indica que não é para aparecer NILL.

Após o entendimento de como funciona a ferramenta de desenvolvimento AXIS para o mapeamento de classes e atributos em *tags* XML, o desenvolvimento do protótipo ganhou em velocidade.

4.4. Cenários de uso do protótipo

Nesta seção são mostrados dois cenários de uso do prótotipo. No primeiro cenário, é mostrado e comentado o diagrama de sequência para a realização da operação <*edit-config*>. No segundo cenário, é mostrado o diagrama de sequência para a subscrição no Serviço de Eventos, o envio do evento e o cancelamento da subscrição.

4.4.1. Cenário da operação <edit-config>

Na Figura 4.9., a seguir, é mostrado o diagrama de sequência para a realização da operação <edit-config>.

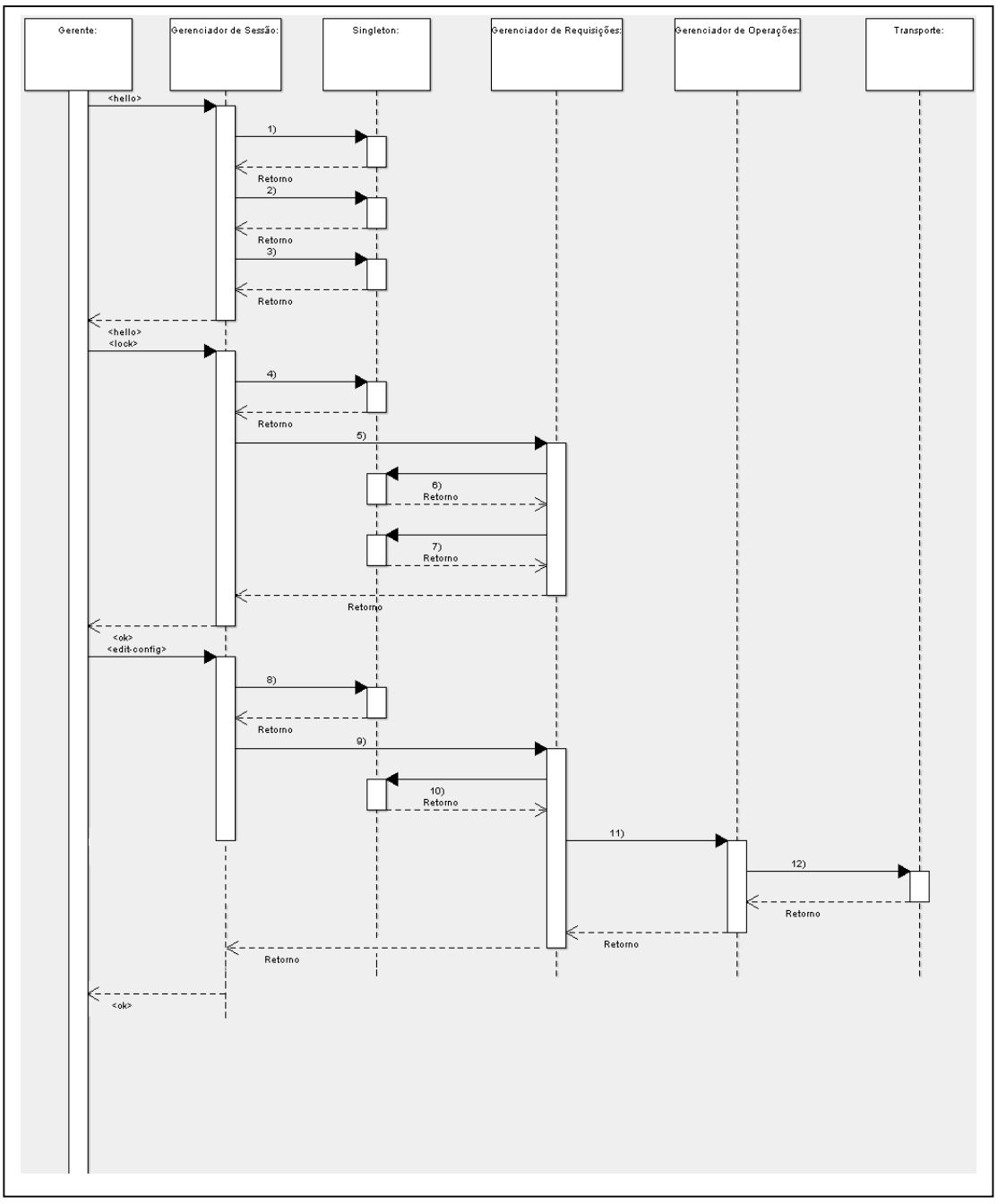


Figura 4.9: Diagrama de sequência operação <edit-config>.

A seguir, são descritos os passos deste diagrama de sequência para um caso de sucesso:

a. O Gerente abre a interface de gerência, informa seu usuário e senha. Neste momento, é estabelecida a sessão NETCONF e é realizado o envio da mensagem <hello> do Gerente para o Gerenciador de Sessão;

b. No passo 1), o Gerenciador de Sessão instancia a classe Singleton e verifica se o usuário já está autenticado no sistema;

c. Caso o usuário não esteja autenticado, retorna para o Gerenciador de Sessão;

d. No passo 2), o Gerenciador de Sessão solicita o acréscimo da nova sessão no Singleton, conforme descrito na Seção 5.3;

e. Retorna para o Gerenciador de Sessões;

f. No passo 3), o Gerenciador de Sessões estabelece o ID (identificação) da sessão. Valor que será usado durante toda a vida da sessão recém-estabelecida para identificá-la das demais;

g. Retorna para o Gerenciador de Sessões;

h. O Gerenciador de Sessões fecha o ciclo da troca de capacidades enviando a mensagem <hello> para o Gerente;

i. O Gerente envia a mensagem de <lock> para o Gerenciador de Sessões para obter acesso exclusivo na configuração informada (padrão: running);

j. No passo 4), o Gerenciador de Sessão aciona o Singleton para verificar se esta é uma nova sessão HTTP. Isto pode ocorrer por *time out*, se o tempo da sessão expirar;

k. Caso não seja uma nova sessão, retorna para o Gerenciador de Sessão;

l. No passo 5), o Gerenciador de Sessão aciona o Gerenciador de Requisições para realizar o *lock* da configuração;

m. No passo 6), o Gerenciador de Requisições aciona o Singleton e verifica se a configuração já está travada ou não;

n. Caso não esteja travada, retorna para o Gerenciador de Requisições;

o. No passo 7), o Gerenciador de Requisições aciona o Singleton e faz o *lock* da configuração solicitada;

p. Retorna para o Gerenciador de Requisições;

q. O Gerenciador de Requisições retorna para o Gerenciador de Sessão

informando que o lock foi realizado;

r. O Gerenciador de Sessão envia a mensagem de <ok> para o Gerente;

s. O Gerente envia a mensagem <edit-config> para o Gerente de Sessão;

t. No passo 8), o Gerenciador de Sessão aciona o Singleton para verificar se esta é uma nova sessão HTTP;

u. Retorna para o Gerenciador de Sessão;

v. No passo 9), o Gerenciador de Sessão aciona o Gerenciador de Requisições para realizar a operação <edit-config>;

w. No passo 10), o Gerenciador de Requisições aciona o Singleton para checar se a configuração está travada;

x. Retorna para o Gerenciado de Requisições;

y. No passo 11), o Gerenciador de Requisições extrai as informações de configuração da operação <edit-config>, define o dispositivo a ser chamado e aciona o Gerenciador de Operações;

z. No passo 12), o Gerenciador de Operações traduz a configuração XML para o comando de linha correspondente; aciona o Transporte, definindo principalmente o meio de comunicação entre o protótipo e o dispositivo - no caso, o TELNET. A sessão TELNET é aberta, o comando enviado e o dispositivo retorna para o Transporte;

aa. O Transporte retorna para o Gerenciador de Operações;

bb. O Gerenciador de Operações retorna para o Gerenciador de Requisições;

cc. O Gerenciador de Operações retorna para o Gerenciador de Sessão;

dd. O Gerenciador de Sessão envia a mensagem de <ok> para o Gerente informando que a configuração foi realizada com sucesso.

4.4.2. Cenário para subscrição no Serviço de Eventos

Na Figura 4.10, a seguir, é mostrado o diagrama de sequência para a subscrição no Serviço de Eventos, o evento gerado e o cancelamento da subscrição.

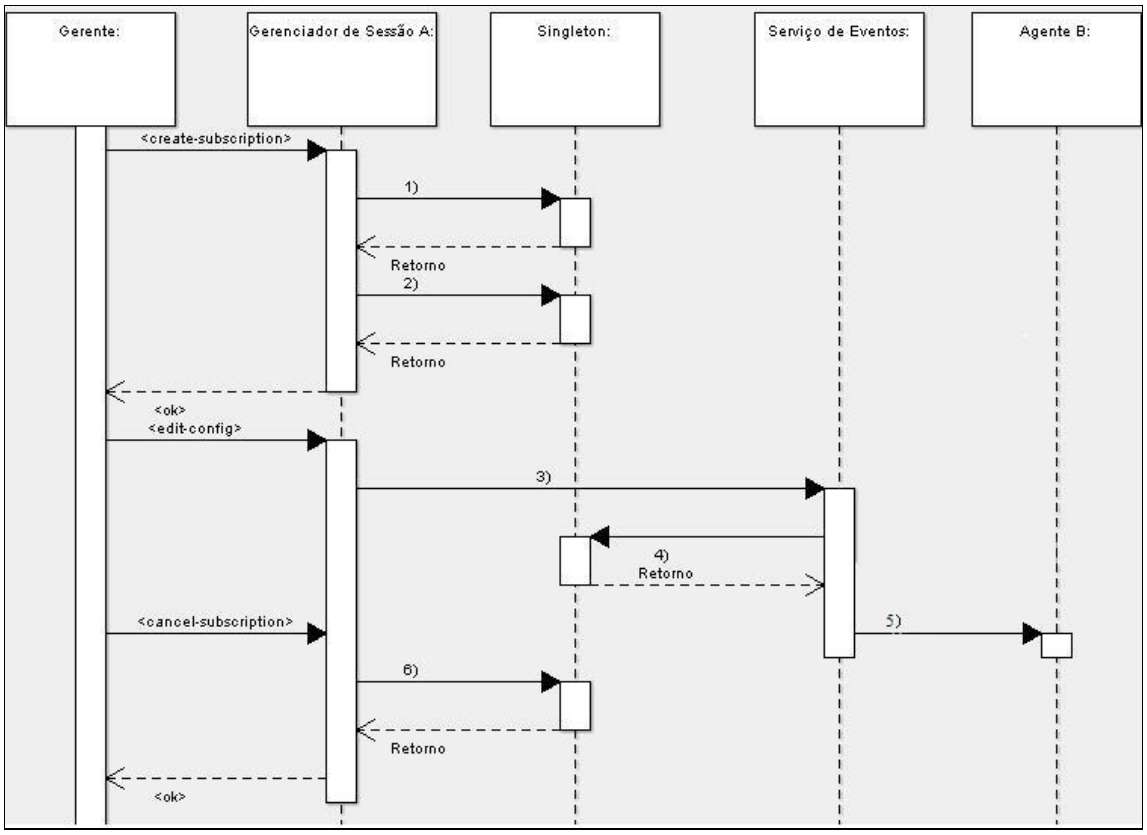


Figura 4.10: Diagrama de sequência para o serviço de eventos.

Nesta figura assume-se que a sessão NETCONF já está estabelecida de acordo com os passos iniciais mostrados na figura anterior. A seguir, são descritos os passos deste diagrama de sequência:

a. O Gerente envia a mensagem <create-subscription> para se inscrever no Serviço de Eventos;

b. No passo 1), o Gerenciador de Sessão aciona o Singleton para adicionar a

subscrição na base de dados;

c. Retorna para o Gerenciador de Sessão;

d. No passo 2), o Gerenciador de Sessão aciona o Singleton e incrementa o contador de identificação de subscrição;

e. Retorna para o Gerenciador de Sessão;

f. O Gerenciador de Sessão envia a mensagem <ok> informando que foi realizada com sucesso a subscrição no Serviço de Eventos;

g. O Gerente envia uma mensagem <edit-config> para o Gerenciador de Sessão. Neste momento, são realizados os passos da operação <edit-config> mostrados na figura anterior;

h. No passo 3), paralelamente ao item g, o Gerenciador de Sessão aciona o Serviço de Eventos para processar o evento;

i. No passo 4), o Serviço de Eventos aciona o Singleton para obter a relação de consumidores do evento;

j. O Singleton retorna para o Serviço de Eventos a relação de consumidores do evento;

k. No passo 5), o Serviço de Eventos monta a mensagem <event> e a envia para o consumidor. Neste exemplo, outro agente NETCONF;

l. O Gerente envia a mensagem <cancel-subscription> para o Gerenciador de Sessão para cancelar a subscrição no Serviço de Eventos;

m. No passo 6) o Gerenciador de Sessão aciona o Singleton para remover a subscrição indicada pelo ID da subscrição;

n. O Singleton retorna para o Gerenciador de Sessão;

o. O Gerenciador de Sessão envia a mensagem <ok> informando que o cancelamento da subscrição foi realizado com sucesso.

4.5. Cenário de Testes

Para a realização dos testes do protótipo foi utilizada uma rede local de 100Mbps em produção. Esta rede é composta por 6 máquinas, todas com a seguinte configuração: Pentium 4 3.0 GHz HT, com 1 GB de memória, HD SATA 160 GB. O sistema operacional é o Linux versão 2.6.15-SMP-LGZ. Uma das máquinas desta rede, chamada alfaromeo, foi utilizada como gerente.

Uma outra rede local de 100Mbps também foi utilizada. Essa contém 16 máquinas, todas com a seguinte configuração: Pentium 4 2.8 GHz HT, com 1 GB de memória, HD SATA 160 GB. O sistema operacional é o Linux versão 2.6.13.3-SMP-LGZ. Uma das máquinas desta rede, chamada dca09, foi utilizada como agente.

A comunicação entre as duas redes é realizada por um *switch-router* da Extreme Networks que faz o papel de roteador, já que as duas redes têm faixa de endereçamento IP distintos. Ver Figura 4.11 abaixo.

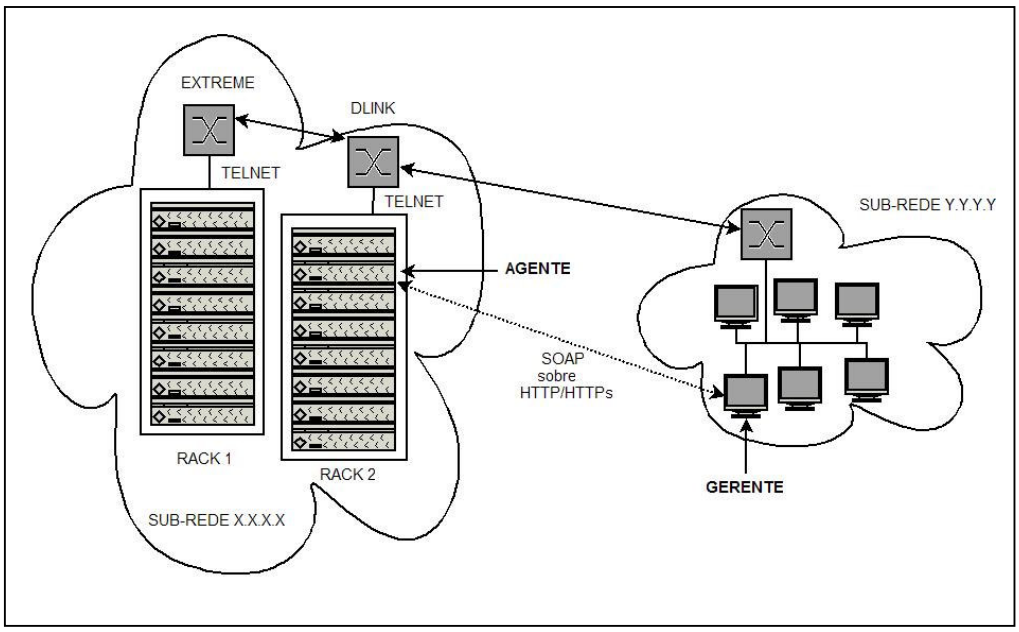


Figura 4.11: Cenário de validação da arquitetura.

Nas duas máquinas foi utilizada a linguagem de programação Java versão 1.5.0_04,

servidor Tomcat versão 5.5 como implementação do Java Servlet e JavaServer Pages [76]. Como implementação da especificação SOAP do W3C foi utilizada a API AXIS da fundação Apache versão 1.3 [77]. Foi utilizada também a API commons-net-1.1.4 do projeto Apache como cliente TELNET para o agente NETCONF consultar ou alterar a configuração das VLANs dos switches [78].

Para o HTTPs foi utilizada a infra-estrutura do próprio Java com a criação de certificados para o gerente e o agente. No Tomcat do agente foi ativado o container de suporte ao HTTPs.

Para realização dos testes foi escrita uma interface de teste específica. Nela é informado o protocolo a ser utilizado, HTTP ou HTTPs; o endereço IP do agente; o número de requisições/consultas; e as expressões XPATH de consulta.

4.6. Resultados obtidos

Para testar o protótipo foram realizadas três classes de testes. A primeira classe de teste é de comparação de desempenho do protótipo com a CLI. Tem como objetivo realizar séries de consultas em seqüência via protótipo e depois a mesma séria via CLI para verificar o desempenho da implementação com relação à CLI. A segunda classe de teste é de consistência. Baseia-se em alterar as configurações de dois switches em produção do laboratório. Realizada a configuração via protótipo, a alteração é em seguida lida via CLI e vice-versa. Espera-se deste teste que os resultados obtidos sejam os mesmos para as duas formas de configuração. A última classe é de tráfego gerado. O objetivo é verificar o tamanho das mensagens enviadas e recebidas entre o gerente e o agente via protótipo e comparar com o tamanho das mensagens ao consultar os equipamentos via CLI.

4.6.1. Testes de desempenho

Para a primeira classe de testes foram realizados dois testes. O primeiro teste consistiu em realizar 5 séries de consultas em sequência às tabelas de rotas da máquina dca09 com sistema operacional LINUX, onde está instalado o agente. Este teste tem a intenção apenas de verificar se a interface de teste, o agente e a transformação da tabela de rotas em XML estão funcionando corretamente.

O segundo teste consistiu em realizar 5 séries de consultas em seqüência às VLANs de dois *switches* do laboratório de fabricantes diferentes. Um *switch* é da Dlink [79]; o outro, é um *switch* de camada 3, do modelo TCP/IP, conhecido como *switch-router*, da *Extreme Networks* [80].

Assim, foram informadas à interface de teste (Seção 4.5) as seguintes expressões XPATH de consulta:

/route para a tabela de rotas do agente;

/dlink/vlan para a tabela de VLANs do switch da DLink;

/extreme/vlan para a tabela de VLANS do switch da Extreme.

Como protocolo de transporte foram informados o HTTP e, depois, o HTTPs.

Cada vez que a interface de teste é executada, uma mensagem <hello> é enviada do gerente ao agente. Este, por sua vez, responde com outra mensagem <hello>. Em seguida são realizadas as consultas, de acordo com a quantidade informada na interface de teste, encapsuladas numa mensagem <rpc></get></rpc>. Para cada operação <get> recebida pelo agente, este retorna com uma mensagem <rpc-reply></data></rpc-reply>. Encapsulado na tag <data> encontra-se o conteúdo da resposta. Após as consultas serem atendidas, o gerente envia uma mensagem <rpc></close-session></rpc> para fechar a sessão NETCONF. Por fim, o agente retorna com uma mensagem <rpc-reply></ok></rpc-reply> indicando o reconhecimento do fim da sessão.

Consulta à Tabela de Rotas

Internamente no agente, para realizar as consultas à tabela de rotas do computador, é realizado o comando "/sbin/route" do Linux no computador onde está instalado o agente. A cadeia de caracteres recebida é tratada, realizado o *parser*, transformada em XML, enviada ao gerente via SOAP sobre HTTPs ou via SOAP sobre HTTP e mostrada na tela da interface de teste.

Tabela 4.1: Consulta à tabela de rotas IP da máquina agente: HTTP x HTTPs.

Rotas	Diferença Tempo HTTP X HTTPs(s)			Tempo médio da consulta(s)	
Consultas	**HTTPs**	**HTTP**	**HTTPS/HTTP**	**HTTPs/Consultas**	**HTTP/Consultas**
10x	2,719	1,812	1,500551656	0,27189996	0,1812
30x	7,053001	3,0220003	2,333884778	0,235100017	0,100733343
60x	12,495	4,775	2,616756119	0,20825	0,079583267
90x	16,47	8,41	1,95838311	0,183	0,093444433
500x	50,689	29,419	1,723002144	0,101377976	0,058837986
		Média	2,026515561	0,199925591	0,102759806

A Tabela 4.1 é formada por seis colunas. A primeira coluna indica a quantidade de consultas realizadas. Foram realizadas 10, 30, 60, 90 e 500 consultas em seqüência. A coluna HTTPs mostra o tempo em segundos necessário para realizar a quantidade de consultas indicadas utilizando o HTTPs para uma conexão segura a partir do gerente. Para cada consulta, é obtida a tabela de rotas IP da máquina onde está o agente, máquina dca09, e mostrado em XML o resultado na tela da interface de teste do gerente. A terceira coluna mostra o tempo em segundos dos mesmos passos anteriores para a consulta com HTTPs, mas agora sem segurança, via HTTP puro. A quarta coluna mostra quantas vezes o HTTPS é mais lento que o HTTP para cada série. As duas últimas colunas mostram o tempo médio das consultas.

A última linha mostra as médias das três últimas colunas.

Este teste inicial atingiu o objetivo proposto, confirmando a operação correta do protótipo.

A última linha, quarta coluna, mostra que o HTTPs é 2 vezes mais lento que o uso do HTTP. No entanto, com o uso do HTTPs abre-se a oportunidade de consultar e configurar o

dispositivo de qualquer computador através da Internet. Internamente, se o ambiente for seguro (uma rede exclusiva para configurar os equipamentos, por exemplo) pode-se dispensar o HTTPs. Vale ressaltar, a partir das duas últimas colunas, que o tempo por consulta é baixo, mesmo para o HTTPs. Se a ação de configuração não for uma constante, vale a pena utilizar o HTTPs em função da segurança. Interessante notar que o tempo por consulta para HTTP e, principalmente para HTTPs, mostra uma tendência de queda, provavelmente por algum mecanismo interno de CACHE do servidor TOMCAT Apache.

Outra conclusão deste teste é que a consulta foi realizada no próprio equipamento onde está o agente, sem intermediários, mostrando o real desempenho do protótipo.

Consulta às VLANs

Para este teste, duas medições foram realizadas. Na primeira, as consultas foram realizadas via protótipo. A partir da máquina gerente é aberta uma sessão NETCONF com o agente. Os comandos de consulta são enviados ao agente via SOAP sob HTTP. No agente, é aberta uma sessão TELNET ao *switch* indicado. É passado o comando de linha de interface "*show vlan*". O comando de consulta é o mesmo para os dois *switches* testados. A cadeia de caracteres recebida do *switch* é tratada, realizado o *parser*, transformada em XML, enviada ao gerente via SOAP sob HTTP e mostrada na tela da interface de teste.

O HTTP foi escolhido em detrimento ao HTTPs por ter tido, como se esperava, um desempenho melhor do que o HTTPs.

A segunda consulta é feita da máquina gerente, diretamente aos switches, sem intermediários, via CLI através de TELNET.

Tabela 4.2: Consulta às VLANs do switch Extreme: protótipo x CLI.

Extreme

Consultas	NetConf HTTP(s)	CLI Puro(s)	Netconf/ CLI	OverHead/Consultas
10x	2,603	0,855	3,04	0,1748
30x	6,823001	1,9	3,591	0,164100033
60x	13,294002	3,469	3,832	0,163750033
90x	19,220999	5,059	3,799	0,157355544
500x	106,93402	26,864	3,98	0,16014004

A Tabela 4.2 é composta por cinco colunas. A primeira coluna, consultas, mostra a quantidade de consultas em sequência.. A segunda coluna, NETCONF HTTP(s), mostra o tempo, em segundos, para se realizar a consulta ao switch via protótipo. A terceira coluna, CLI Puro(s) mostra, em segundos, o tempo da consulta via CLI através de TELNET do gerente diretamente ao *switch*. A quarta coluna, NETCONF/CLI, informa quantas vezes o NETCONF é mais lento que o CLI para cada série de consulta. A quinta coluna, OverHead/Consultas, é a divisão do *OverHead* do NETCONF (coluna 2 – coluna 3) pela quantidade de consultas (coluna 1).

Tabela 4.3: Consulta às VLANs do switch Dlink: protótipo x CLI.

Dlink

Consultas	NetConf HTTP(s)	CLI Puro(s)	Netconf/ CLI	OverHead/Consultas
10x	8,541	5,84	1,4625	0,2701
30x	24,497002	16,875	1,451	0,254066733
60x	47,923992	33,503	1,4304	0,240349867
90x	72,14198	50,161	1,438	0,244233111
500x	394,37106	276,321	1,427	0,23610012

A Tabela 4.3 é composta das mesmas colunas da tabela 4.2

Comparando a Tabela 4.2 com a Tabela 4.3, verifica-se que o tempo do CLI puro, coluna 3, do DLink é muito maior que do Extreme. O primeiro motivo é que o DLink tem 15 VLANs configuradas, enquanto o Extreme tem 6 VLANs. O XML gerado para o DLink tem 5,1KBytes de tamanho, enquanto o XML para o Extreme tem 1,4KB. Ou seja, o XML do DLink é cerca de 3,6 vezes maior que o do Extreme. Outro fator que impacta no resultado, é que para o DLink, após enviar o comando "show vlan", é necessário enviar mais um comando com o caracter "a", indicando que se deseja receber todas as VLANs. Isto não é necessário no Extreme, basta enviar um comando: "show vlan". Depois é necessário enviar mais um comando no Dlink indicando que se deseja fazer o *logout* do switch, comando "*logout*".

Outra constatação é que os valores das duas últimas colunas se mantêm praticamente constante, indicando uma previsibilidade do protótipo para outras séries de consultas.

Por último, da quarta coluna, verifica-se que o protótipo é em média 3,65 mais lento que a CLI para o Extreme. Enquanto para o Dlink o protótipo é em média 1,44 mais lento que a CLI. Esta diferença reflete a maior demora para se realizar a CLI via TELNET no Dlink, diluindo assim o custo do protótipo com relação ao Extreme, onde a consulta via CLI é bem mais rápida.

4.6.2. Testes de consistência

Neste teste, a intenção é verificar que o protótipo implementado pode (e deve) conviver com outras formas de acesso aos dispositivos. Os dispositivos neste teste continuam sendo os mesmos *switches* utilizados no teste anterior. A outra forma de consulta continua sendo via CLI através de uma conexão TELNET.

Uma ação de configuração via protótipo deve ser lida via CLI. De forma inversa, uma ação de configuração via CLI deve ser lida via protótipo. Nos dois casos, os resultados obtidos devem ser idênticos para as duas formas de acesso.

Para este teste foi utilizada a interface de menu da Figura 4.5. O teste foi realizado primeiro para o DLink. Os mesmos passos foram realizados depois para o Extreme.

Este teste de consistência comprovou que o protótipo implementado funciona corretamente. As configurações via NETCONF são realizadas com sucesso e confirmadas via CLI e vice-versa. A convivência com outros meio de acesso ao dispositivo, tais como SNMP e *Web-Browser* não deve ser um problema. No entanto, não foram testadas, uma vez que a principal forma de configuração ainda é via CLI.

Outra conclusão a partir do teste de consistência

A outra conclusão que se pode chegar, de forma implícita, é que, a partir do uso do NETCONF, mais a definição de uma modelo de dados, configurar vários dispositivos se torna uma tarefa muito menos complicada. Vejamos mais alguns detalhes do teste anterior para confirmar esta afirmação.

Para configurar os dois switches via NETCONF, foi utilizado o mesmo arquivo para a

criação das VLANs f_1 e f_2. De acordo com o XML SCHEMA definido, todas os elementos do XML são seqüências que podem ou não estar presentes. Assim, se o operador, por exemplo, não sabe o endereço IP, ou não sabe se o switch aceita ou não informar o endereço IP, isto fica transparente para o operador. Ao receber o XML de configuração, o agente atua como um PROXY. Ele verifica qual *switch* que se deseja configurar, os dados são enviados para a classe responsável pela tradução do XML em comando de linha (CLI), o *switch* é acionado via CLI através de TELNET e configurado.

Ao realizar o papel de PROXY, o agente retirou do operador a necessidade de saber como são os comandos de linha de cada switch. Por exemplo, para criar uma VLAN do tipo *tagged* no DLink são necessários dois comandos:

- create vlan f_1 tag 100;
- config vlan f_1 add tagged 49-50.

O primeiro comando cria a VLAN de nome f_1 com a *tag* 100. O segundo comando configura a VLAN criada, adicionando as portas 49 e 50 do tipo *tagged*.

Já no Extreme são necessários quatro comandos:

- create vlan f_1;
- config vlan f_1 ipaddress 10.10.1.10 255.255.255.255;
- config vlan f_1 tag 10;
- config vlan f_1 add port 30-32 tagged.

O primeiro comando cria a VLAN f_1. A seguir ela é configurada com o endereço IP e a respectiva máscara. Depois é configurado o valor 10 para a *tag* desta VLAN. Por último, são adicionadas as portas 30 até 32 à vlan com a opção *tagged*.

Note que para configurar o DLink são necessários 2 comandos, no Extreme 4. Esta não é a única diferença. O Extreme é um *switch-router*, por isso é possível configurar um IP para a VLAN f_1. Este passo é opcional, não precisa ser feito no DLink. No Dlink, pode-se criar a VLAN e no mesmo comando configurar a sua *tag*. No Extreme, a *tag* deve ser informada em outro comando de linha. Por último, no DLink, para adicionar as portas que participarão da VLAN, indica-se as portas no final do comando. No Extreme, é a palavra *tagged* que fica no final do comando, e entre a palavra *add* e as portas deve-se colocar a palavra *port*.

Ou seja, apenas para configurar a parte de VLANs destes switches existem 7 diferenças.

Na hipótese que se tivesse que configurar via CLI outros switches de fabricantes diferentes, pode-se extrair do exemplo acima que a quantidade de diferenças tende a aumentar.

A configuração das VLANs é apenas uma parte do processo de configuração do swtich. Uma operadora de telecomunicações, uma grande empresa ou um provedor de Internet geralmente necessita configurar vários equipamentos de rede, muitas vezes de diferentes fabricantes, para tornar a rede operacional ou para oferecer um novo serviço. Para isto, precisa-se configurar vários tipos de informações dos equipamentos de rede. Sem o suporte de um sistema de configuração eficiente esta tarefa tende a ser estressante, demorada, suscetível a erros, complexa e pouco produtiva devido à quantidade de comandos diferentes que este cenário apresenta.

Por isso, o NETCONF, auxiliado por modelos de dados bem desenhados, pode ser a chave para resolver os problemas atuais da área funcional de Gerência de Configuração.

4.6.3. Tráfego gerado

Neste teste, são verificados o tamanho das mensagens geradas para as consultas às VLANs do DLink e do Extreme via CLI e via protótipo.

Tabela 4.4: Extreme: tamanho das mensagens via CLI.

Extreme - CLI

Operações	Tamanho (Bytes)
login	7
usuário	9
password	19
senha	8
texto de entrada	100
prompt	21
comando	11
dados	668
Total	843

Tabela 4.5: Extreme: tamanho das mensagens via protótipo.

Extreme – Operações Netconf	Tamanho (bytes)		
	HTTP	Xml	Total
<hello> – gerente para o agente	323	504	827
<hello> – agente para o gerente	209	606	815
<rpc> </get> </rpc>	376	394	770
<rpc-reply> <data> </rpc-reply>	140	2307	2447
<rpc> <close-session> </rpc>	376	352	728
<rpc-reply> <ok> </rpc-reply>	140	353	493
		Total Geral	6080

Tabela 4.6: Dlink: tamanho das mensagens via CLI.

DLink - CLI

Operações	**Tamanho (Bytes)**
login	244
usuário	13
password	19
senha	11
prompt	23
comando	14
dados	4343
logout	8
Total	4675

Tabela. 4.7: Dlink: tamanho das mensagens via protótipo.

DLink – Operações Netconf	Tamanho (bytes)		
	HTTP	Xml	Total
<hello> – gerente para o agente	323	504	827
<hello> – agente para o gerente	209	606	815
<rpc> </get> </rpc>	376	396	772
<rpc-reply> <data> </rpc-reply>	1402	6408	7810
<rpc> <close-session> </rpc>	376	352	728
<rpc-reply> <ok> </rpc-reply>	140	353	493
		Total Geral	11445

Das Tabelas 4.5 e 4.7 pode-se notar, independemente da consulta realizada, a constância no tamanho das mensagens do protocolo NETCONF de acordo com cada operação. Por exemplo, para os três tipos de consultas realizadas, a operação <*get*> apresentou os tamanhos: 770, 772. A diferença aqui se deve, única e exclusivamente, ao tamanho da expressão XPATH utilizada.

Pode-se concluir também que o tráfego gerado pelas mensagens do protocolo tende a diluir à medida que os dados enviados e, principalmente, recebidos são maiores. No Extreme, o XML das VLANS tem 1,4 KB de tamanho. Ainda que estes dados devam ser encapsulados no elemento <*data*> da mensagem <rpc-reply>, são apenas 40,25% do tráfego gerado. Mais da metade do tráfego se deve às próprias mensagens do protocolo.

Já no DLink, só o XML das VLANs tem 5,1 KB. Assim, 68,24% do tráfego é gerado pelos próprios dados, diluindo o impacto das mensagens do protocolo. Conclui-se, assim, que o ideal para o protocolo é seu uso para grandes trocas de dados. Como esta é uma característica dos arquivos de configuração, o impacto do NETCONF não é significativo para a área funcional de Gerência de Configuração.

A mensagem do Extreme via protótipo é 7,21 vezes maior que a mensagem gerada via CLI. No Dlink a mensagem é 2,5 vezes maior via protótipo com relação à mensagem gerada via CLI. Esta diferença mostra mais uma vez que o *overhead* das mensagens do protocolo NETCONF é diluída quanto maior forem os dados transferidos.

4.7. Trabalhos Relacionados

Em Julho de 2004 o grupo POSTECH publicou um artigo sobre uma proposta de arquitetura para o NETCONF e suas sugestões e experiências na implementação da proposta [81]. A tarefa em questão foi realizar a configuração de um dispositivo de rede para compartilhamento de IP via *Network Address Translation* (NAT). O gerente foi implementado com AXIS e o agente com gSOAP, pois o sistema foi embutido no dispositivo. As maiores diferenças quanto ao nosso trabalho foram: eles propuseram um modelo de dados mas não o tornaram disponível; a arquitetura não prevê claramente o tratamento de mensagens de notificação; a arquitetura não prevê claramente o tratamento para novos

modelos de dados; e, por último, e mais importante, de acordo com a Figura 3.a, a mensagem NETCONF gerada não é compatível com a especificação do protocolo: falta na mensagem de requisição a identificação da mensagem (atributo *message-id*); foi incluído o elemento <*sessionId*>. Primeiro, a maneira correta é <*session-id*>. Segundo, o elemento <*session-id*> é passado na mensagem de <*hello*>. Este valor é utilizado para identificar as várias sessões no agente e distinguir as mensagens recebidas através da manutenção da sessão via HTTP.

Outro trabalho foi publicado em 2006 [72]. Neste trabalho, eles utilizam o NETCONF para configurar o equipamento *Netflow* da Cisco para monitoramento do tráfego de rede via o monitoramento de *probes*. A implementação utilizou gSOAP e definiu-se um modelo de dados. Diferente da nossa proposta, não foi apresentada uma arquitetura e o modelo de dados também não foi apresentado.

Em um trabalho mais recente, de Fevereiro de 2007, Iijima, et.al, da Hitachi Japão, propõem e apresentam, da mesma maneira que nosso trabalho, um modelo de dados para VLAN [82]. É uma submissão ao grupo NETCONF na forma de *draft*. Nele, todo o XML Schema é apresentado. Este Schema é definido para quatro tipos de VLANs: tagged, untagged, baseado em endereço MAC e baseado em endereço de subrede IP. É reportado também que este Schema foi apresentado e aceito pela organização japonesa *Interoperability Technology Association for Information Processing / Open Systems Management Industry Collaboration* (INTAP/OSMIC) [83], responsável por promover a pesquisa e desenvolvimento naquele país. OSMIC é um sub-comitê da INTAP responsável por interoperabilidade.

Deste mesmo grupo, em Abril de 2007, também na forma de *draft* no NETCONF, são apresentadas as experiências de se implementar o NETCONF e o modelo de dados para VLANs utilizando SOAP [84]. Eles também utilizaram o Axis como implementação do SOAP e propuseram uma arquitetura de implementação.

Muitas concidências são compartilhadas entre o trabalho do Iijima e o nosso: proposta de um modelo de dados para VLAN, proposta de arquitetura e implementação do NETCONF sob SOAP utilizando Axis. Com relação à arquitetura, a nossa proposta é mais detalhada. Uma vez que define módulos claros para o armazenamento das configurações (Repositório de

Dados), um módulo de filtro das mensagens, um módulo para o tratamento dos modelos de dados e um módulo para o serviço de eventos. A arquitetura por eles proposta está relacionada, principalmente, com a ferramenta utilizada para a implementação do NETCONF sob SOAP, o Axis.

Importante salientar que todos os trabalhos apresentados e discutidos atestam a viabilidade do NETCONF sob SOAP como protocolo para a gerência de configuração, solucionando os problemas dos protocolos e ferramenttas atualmente utilizados, principalmente SNMP e CLI, respectivamente.

4.8 Considerações finais

Neste capítulo foi apresentada a arquitetura para implementação e validação do protocolo NETCONF utilizando SOAP como transporte e a descrição de seus módulos funcionais.

Foi apresentado o projeto, as tecnologias e as ferraementas utilizadas para a implementação da arquitetura proposta para validação do protocolo NETCONF, e os desafios encontrados na fase de implementação. Foi descrito o cenário utilizado, a contribuição adicional do trabalho na forma do SCHEMA XML para configuração de VLANs, o serviço de evento, detalhado os módulos do sistema/protótipo e apresentado os resultados obtidos.

Como avaliação da proposta foram realizados testes de desempenho, consistência e de tráfego gerado. Os dados recolhidos nos testes apontam para a viabilidade da arquitetura proposta em comparação com a CLI, ainda uma das ferramentas mais utilizadas para gerência de configuração.

Por último, trabalhos relacionados recentes foram apresentados atestando a viabilidade do NETCONF para a gerência de configuração, cobrindo as lacunas principalmente do SNMP e da CLI para esta gerência.

Capítulo 5
Conclusão e trabalhos futuros

5.1.Conclusão

Atualmente, o protocolo NETCONF é uma RFC no IETF dentro da área de operações e gerência, sob a responsabilidade do grupo de trabalho que dá nome ao protocolo, NETCONF (*Network Configuration*).

Neste trabalho foi estudado o protocolo NETCONF e seu mapeamento para o transporte sobre SOAP, e uma proposta para arquitetura e a validação da mesma. Na fase de implementação, o principal teste foi a configuração de VLANs para switches de diferentes fabricantes. Outra contribuição importante foi estabelecer um modelo de dados via XML SCHEMA para configuração de VLANs, no qual a pessoa responsável pela manutenção do equipamento pode basear-se sem ter que saber detalhes específicos de comandos de linha de cada um dos switches.

A partir do estudo e implementação do protocolo, a principal critica recai principalmente sobre dois itens: a falta de uma definição para os modelos de dados e a existência de toda uma Seção no documento para tratar do filtro chamado *subtree filter*.

Com relação aos modelos de dados, o protocolo não define o que vai ser gerenciado, o formato, ou os campos que devem existir e seus tipos. Como o protocolo nasceu baseado em XML, a visão daqueles que o escreveram é que a interoperabilidade entre diferentes agentes, ao configurar dispositivos diferentes, é conseguida através das ferramentas disponíveis para XML. Isto é verdade, no entanto, esta situação gera um enorme trabalho adicional por parte daqueles que implementam o protocolo. Cada fabricante tenderá a ter como estratégia escrever o seu próprio modelo de dados para a mesma área de configuração; VLANs, por

exemplo. Esta idéia é o oposto do que se vê no SNMP e que pode ser entendido como um ponto positivo: a definição de um modelo de dados através das MIBs.

Uma solução para esta questão é agregar a arquitetura SOA ao mapeamento do NETCONF para SOAP e utilizar o UDDI para registrar, pesquisar e obter os modelos de dados existentes. Desta forma, o gerente pode adicionar de forma manual, ou o agente de forma programática, o modelo de dados para o gerenciamento de uma nova área de configuração.

Com relação ao filtro s*ubtree filter*, boa parte do documento NETCONF preocupa-se em mostrar um novo esquema de filtragem. Este filtro teria a função de retirar da configuração total do dispositivo apenas alguns dados de interesse do gerente. A principal observação é que o XPATH é plenamente capaz de realizar esta tarefa. É um recurso bem conhecido e bastante usado no mundo XML. O custo de se implementar o filtro talvez não compense quando se leva em conta este fato.

As conclusões retiradas deste trabalho foram:

- O protocolo NETCONF e o seu mapeamento para SOAP são plenamente capazes de substituir o protocolo SNMP na tarefa de configuração de dispositivos de rede. Ao nascer baseado em XML, o protocolo ganha a possibilidade de gerenciar qualquer informação que seja baseada em XML. Ao definir o uso do SOAP, o protocolo abre a possibilidade de se fazer gerência de configuração a partir de sites remotos. Com SOAP, o protocolo abre também a possibilidade de agregar as idéias de uma arquitetura SOA e, através disto, oferecer a oportunidade dos gerentes e agentes NETCONF conversarem com outros gerentes e agentes, principalmente para troca, no futuro, de modelos de dados;
- Arquitetura proposta é factível e atende a necessidade de ser um ponto de partida para as implementações do protocolo;
- A definição de um modelo de dados através de um XML Schema para a configuração de VLANs para fabricantes diferentes foi uma parte não prevista no

início dos estudos. Este modelo agregou bastante ao trabalho e corrobora com a idéia geral de que um padrão para o modelo de dados seria bem vindo ao protocolo;

- Serviço de Eventos agrega valor ao protótipo, possibilitando o monitoramento das ações executadas internamente no agente;
- Os objetivos da implementação foram atingidos. Exceto pelas operações básicas <delete-config> e <copy-config> que ainda não estão operacionais.

As contribuições do trabalho foram:

- Implementação compatível com a especificação;
- Serviço de eventos onde se pode indicar o consumidor do evento;
- Proposta de modelo de dados para VLANs;
- Proposta de arquitetura de implementação;
- Análise do custo do NETCONF com relação a mecanismos clássicos de configuração de dispositivos.

5.2. Trabalhos futuros

Com relação ao modelo de dados, seria interessante as seguintes atividades como trabalho futuro:

- Propor um XML Schema para a configuração básica de um switch, incluindo, por exemplo, a configuração dos parâmetros de cada porta. Excluir da proposta detalhes específicos de um switch. Se um switch for do tipo switch-router, por exemplo, os detalhes a mais de roteamento seriam retirados do XML Schema proposto;
- Agregar a arquitetura SOA e UDDI à arquitetura para facilitar a troca de modelos de dados entre agentes e gerentes (principalmente, outros fabricantes).

Com relação ao próprio protótipo, realizar as seguintes melhorias:

- Oferecer ao gerente uma interface gráfica;
- Oferecer ao gerente uma interface web;
- Disponibilizar o SSH como mais um meio de transporte entre o gerente e o agente;
- Disponibilizar outros meios de transporte entre o agente e o dispositivo gerenciado além do telnet, como, por exemplo, o *Trivial File Transport Protocol* [TFTP] e comunicação via porta serial;
- Armazenar as configurações e as informações, que hoje estão guardadas de maneira volátil na classe *Singleton*, num banco de dados XML;
- Separar as mensagens de erro descritas no protocolo num arquivo de constantes.

A partir da implementação do banco de dados XML o sistema ganha em persistência. O serviço de notificação de eventos pode se aproveitar disto para em um trabalho futuro acrescentar o modelo *pull* para os eventos. Desta forma, os eventos são armazenados no banco de dados do agente. Um gerente com baixo poder de processamento, por exemplo um PDA, consultaria os eventos que desejasse, ao invés de assinar o serviço e receber todos os eventos gerados, o que sobrecarregaria o dispositivo.

Outro trabalho futuro para o serviço de notificação de eventos é realizar duas classes de testes. A primeira seria obter métricas de desempenho comparando com o desempenho das *traps* do SNMP. A segunda classe envolveria estudar o tamanho das mensagens geradas comparando, novamente, com o tamanho das *traps* do SNMP.

Com relação ao próprio protocolo, seria interessante desenvolver uma outra abordagem na qual as mensagens do protocolo seriam transportadas diretamente sobre o HTTP, sem a intermediação do SOAP. Uma vez que o próprio SOAP também deve ser transportado, geralmente sobre HTTP, gerando um *overhead*. Duas classes de testes devem ser realizadas para investigar a realidade da afirmação anterior. A primeira para obter métricas de desempenho, comparando o desempenho da implementação atual do NETCONF sobre SOAP sobre HTTP versus o desempenho do NETCONF sobre HTTP. A segunda, comparando o tamanho das mensagens nas duas implementações.

Submissões mais recentes no grupo de trabalho do NETCONF no IETF, relatam a importância de se aproveitar a experiência e as vantagens do SMI e, principalmente das MIBs, para definição do modelo de dados no SNMP e trazer isto para o mundo NETCONF [90-andy]. Assim, preenchendo a lacuna da camada de conteúdo que o protocolo deixa em aberto.

Esta nova abordagem tem o nome de NETCONF *Extensions*. Visa principalmente estender o protocolo para as questões de modelo de dados [90], funcionalidades do próprio protocolo [91] e um modelo de controle de acesso [92]. Todas estas propostas são de agosto de 2007.

Como última proposta para trabalho futuro, é sugerido o estudo destes *drafts* e, principalmente o uso de SMI para definir MIBs NETCONF como forma de propiciar a efetiva adoção do protocolo, interoperabilidade entre implementações e que permita que sejam controláveis a criação das interfaces de gerência.

Referências Bibliográficas

[1] ISO. Página na Internet, Julho de 2007. http://www.iso.org

[2] Open Systems Interconnection | Structure of Management Information | Part 4: Guidelines for the Definition of Managed Objects. International Standard ISO/IEC, 10165-4 : 1992

[3] J. Case, M. Fedor, M. Schoffstall, J. Davin. RFC 1157, A Simple Network Management Protocol. Maio 1990.

[4] Jeong-Hyuk Yoon, Hong-Taek Ju, W. Hong. Development of SNMP-XML translator and gateway for XML-based integrated network management. *International Journal of Network Management,* Volume 13(4): 259-276, Agosto 2003.

[5] K. McCloghrie, M. Rose. RFC 1156, Management Information Base for Network Management of TCP/IP-based internets. Maio 1990.

[6] Jurgen Schonwalder, Aiko Pras, Jean-Philippe Martin-Flatin. On the future of Internet Management Technologies. *IEEE Communications Magazine*, Volume 41(10):90-97, Outubro 2003.

[7] Jean-Philippe Martin-Flatin. Web-Based Management of IP Networks and Systems. Wiley 2002.

[8] J. Case, K. McCloghrie, M. Rose, S. Waldbusser. RFC 1905, Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2). Janeiro 1996.

[9] Frank Strauß, Torsten Klie. Towards XML Oriented Internet Management. IFIP/IEEE Eighth International Symposium on Volume , Issue , 24-28 March 2003 Page(s): 505 - 518

[10] HTML. W3C HTML especificação 4.01. http://www.w3.org/TR/1999/REC-html401-19991224. Dezembro 1999.

[11] T. Berners-Lee, R. Fielding, H. Frystyk. RFC 1945, Hypertext Transfer Protocol -- HTTP/1.0. Maio 1996.

[12] Applets. Sun Applets. Página na Internet, Julho 2007. http://java.sun.com/applets/

[13] Corba. OMG. Página na Internet, Julho 2007. http://www.omg.org/technology/documents/corba_spec_catalog.htm.

[14] RMI. Sun RMI. Página na Internet, Julho 2007. http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[15] DMTF. DMTF. Página na Internet, Julho 2007. http://www.dmtf.org/home

[16] WBEM. DMTF. Página na Internet. Julho 2007. http://www.dmtf.org/standards/wbem/

[17] UML. OMG UML. Página na Internet. Julho 2007. http://www.uml.org/

[18] DMTF, CIM. Common Information Model 2.2. http://www.dmtf.org/standards/cim/cim_spec_v22. Junho 1999.

[19] XML. W3C XML. Página na Internet, Julho 2007. http://www.w3.org/XML/

[20] JMX. Sun JMX. Página na Internet, Julho 2007. http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[21] W3C. Document ObjectModel (DOM) Technical Reports. Página na Internet,W3C, Julho 2007. http://www.w3.org/DOM/.

[22] XML-DEV. SAX. Página na Internet, XML-DEV, Julho 2007. http://www.

saxproject.org/.

[23] SOA. Reference Model for Service Oriented Architecture 1.0, OASIS, Agosto 2006.

[24] Najla Al-Rawahi, Youcef Baghdadi. Approaches to Identify and Develop Web Services as Instance of SOA Architecture. *Services Systems and Services Management*, 2005. Proceedings of ICSSSM apos;05. 2005 International Conference on Volume 1, Issue , 13-15 June 2005 Page(s): 579 - 584 Vol. 1

[25] Lin Chen, Minglu Li. Using Web Services in TMN Environment. Shangai Jiao Tong University, 2005.

[26] WSRF. OASIS WSRF. Página na Internet, Julho 2007. http://www.oasis-open.org/committees/wsrf/

[27] OASIS. OASIS. Página na Internet, Julho 2007. http://www.oasis-open.org/

[28] R. Enns, Ed. NETCONF Configuration Protocol. *draft-ietf-netconf-prot-12.txt,* Fevereiro 2006.

[29] [NETCONF/SSH] M. Wasserman, T. Goddard. Using the NETCONF Protocol over Secure Shell (SSH). *draft-ietf-netconf-ssh-06.txt,* Março 2006.

[30] E.Lear, K. Crozier. Using the NETCONF Protocol over Blocks Extensible Exchange Protocolo (BEEP). *draft-ietf-netconf-beep-10.txt*, Março 2006.

[31] T. Goddard. Using the Network Configuration Protocolo (NETCONF) over the Simple Object Access Protocol (SOAP). *draft-ietf-netconf-soap-08.txt*, Março 2002.

[32] Juniper Network. Página da Internet, Julho 2007. http://juniper.net.

[33] SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/, Junho 2003

[34] HTTPs. E. Rescorla. RFC 2818, HTTP over TLS. Maio 2000.

[35] SNMPv3. J. Case, R. Mundy, D. Partain, B. Stewart. RFC 2570, Introduction to Version 3 of the Internet-standard Network Management Framework. Abril 1999.

[36] CMIP. OSI. ISO/IEC 7498-4. 1989.

[37] OSI. I*SO 7498:1984 Open Systems Interconnection - Basic Reference Model.* 1984.

[38] TMN. International Telecommunications Union–Telecommunications Services Sector (ITU). M.3000 recommendation series. Outubro 1994.

[39] TL1. BellCore. GR-831-Core. Novembro, 1996

[40] Cisco IOS. CISCO. Página na Internet, Julho 2007. http://www.cisco.com/en/US/products/sw/iosswrel/products_ios_cisco_ios_software_category_home.html

[41] M. Rose, K. McCloghrie. RFC 1155, Structure and Identification of Management Information for TCP/IP-based Internets. Maio 1990.

[42] Information processing systems - Open Systems Interconnection, Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard

8824, Dezembro 1987.

[43] K. McCloghrie, M. Rose. RFC 1213, Management Information Base for Network Management of TCP/IP-based internets:MIB-II. Março 1991.

[44] W3C. The World Wide Web Consotium. Página na Internet, Julho 2007. http://www.w3c.org.

[45] ISO 8879. Information Processing -- Text and Office Systems - Standard Generalized Markup Language (SGML),1986. <URL:http://www.iso.ch/cate/d16387.html>

[46] Libsmi. Ibr, Tu-Bs. Página na Internet, Julho 2007. http://www.ibr.cs.tu-bs.de/projects/libsmi/

[47] Mi-Jung Choi, et.al. XML-Based Network Management for IP Networks. ETRI Journal, Volume 25, Number 6. Dezembro 2003.

[48] COM/DCOM. Microsoft DCOM. Página na Internet, Julho 2007. http://www.microsoft.com/com/

[49] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, Microsoft, IBM Research, Março 2001. http://www.w3c.org/TR/wsdl.

[50] A. Hately, C. Riegen, and T. Rogers. UDDI Version 3.0.2. UDDI Spec Technical, IBM, SAP AG,Computer Associates, Fevereiro 2005. http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-%20041019.htm.

[51] B. Thurm. Analysis of Web Service Performance for Network Management. University of Karlsruhe. Security and Management, Julho 2004.

[52] A. Pras, T. Drevers, R. Meent, and D. Quartel. Comparing the Performance of SNMP and Web Services-Based Management. In *ETRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT*,Fall 2004.

[53] A. Bierman. Network Management Observations. *draft-bierman-nm-observations-00.txt*. Junho 2002.

[54] C. Rigney, S. Willens, A. Rubens, W. Simpson. RFC 2865, Remote Authentication Dial In User Service (RADIUS). Junho 2006.

[55] TCP. RFC 793, Transmission Control Protocol. Darpa Internet Program Protocol Specification. Setembro 1981.

[56] S. Chisholm, H. Trevino. NETCONF Event Notifications. *draft-ietf-netconf-notification-08.txt.pre-release*. Julho 2007.

[57] S. Chisholm, H. Trevino. NETCONF Event Notifications. *draft-ietf-netconf-notification-02*. Junho 2006.

[58] A. Pras, J. Schoenwaelder. RFC 3444, On the Difference between Information Models and Data Models. Janeiro 2003

[59] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R.Sahita, A. Smith, F. Reichmeyer, "Structure of Policy Provisioning Information (SPPI)", IETF RFC 3159, Proposed Standard, Agosto 2001.

[60] M. Fine, K. McCloghrie, J. Seligson, K. Chan; S. Hahn, R. Sahita, A. Smith, F. Reichmeyer, "Framework Policy Information Base", IETF, Internet-Draft, draft-ietf-rap-frameworkpib-07.txt, Janeiro 2002.

[61] MOF. OMG. Página na Internet, Julho 2007. http://www.omg.org/mof/

[62] Yenca. SourceForge. Página na Internet, Julho 2007. http://sourceforge.net/projects/yenca/

[63] Cisco EDI. Cisco. Página na Internet, Julho 2007. http://www.cisco.com/en/US/products/ps6456/products_qanda_item0900aecd802f2038.shtml

[64] F. L. Verdi, R. Duarte, F. C. de Lacerda, E. Madeira, E. Cardozo, and M. Magalhães. Web Services-based Provisioning of Connections in GMPLS Optical Networks. In *Simpósio Brasileiro de Redes de Computadores (SBRC 2005)*, Fortaleza, CE, Maio 2005.

[65] Rafael L. Duarte. Provisionamento baseado em Web Services de conexões fim-a-fim em Redes Ópticas GMPLS. Tese de Mestrado, Faculdade de Engenharia Elétrica e Computação, UNICAMP, Junho 2006.

[66] S. Admankar, S. Chisholm. Using XML Schema to define Netconf Content. *draft-chisholm-netconf-model-06.txt*. Fevereiro 2007.

[67] 802.1Q VLANs. IEEE. Página na Internet, Julho 2007. http://www.ieee802.org/1/pages/802.1Q.html

[68] Implementações Netconf. Ietf, Netconf. Página na Internet, Julho 2007. http://www.ops.ietf.org/netconf/

[69] Ensuite. SourceForge. Página na Internet, SourceForge, Julho 2007. http://ensuite.sourceforge.net/index.html.

[70] WS-I. Web Services Interoperability Organization. Página na Internet, Julho 2007. http://www.ws-i.org.

[71] Axis Architecture Guide. Apache Axis. Página na Internet, Apache, Julho 2007. http://ws.apache.org/axis/java/architecture-guide.html

[72] Carle et. Al.Using Netconf for Configuring Monitoring Probes. *Network Operations and Management Symposium*, 2006. NOMS 2006. 10th IEEE/IFIP Volume , Issue , 0-0 0 Page(s):1 - 4

[73] Jax-RPC. Sun Jax-RPC. Página na Internet, Sun, Julho 2007. http://java.sun.com/webservices/jaxrpc/docs.html

[74] JavaBeans. Sun JavaBeans. Página na Internet, Sun, Julho 2007. http://java.sun.com/products/javabeans/

[75] Axis User's Guide. Apache Axis. Página na Internet, Apache, Julho 2007. http://ws.apache.org/axis/java/user-guide.html.

[76] Tomcat. Apache Tomcat. Página na Internet, Apache, Julho 2007. http://tomcat.apache.org/.

[77] AXIS. Web services - AXIS. Página na Internet, Apache, Julho 2007. http://ws.apache.org/axis/.

[78] Commons-net. Apache Internet Protocols client. Página na Internet, Apache, Julho 2007. http://commons.apache.org/net/.

[79] DLink. Página na Internet, Julho 2007. http://www.dlink.com.

[80] Extreme Networks. Página na Internet, Julho 2007. http://www.extremenetworks.com.

[81] James W. Hong, et.al. XML-Based Configuration Management for IP Network Devices. IEEE Communications Magazine ,Volume 42, Issue 7, July 2004 Page(s): 84 - 91.

[82] T. Iijima, et al. VLAN data model for NETCONF. *draft-iijima-ngo-vlandatamodel-00.txt*. Fevereiro 2007.

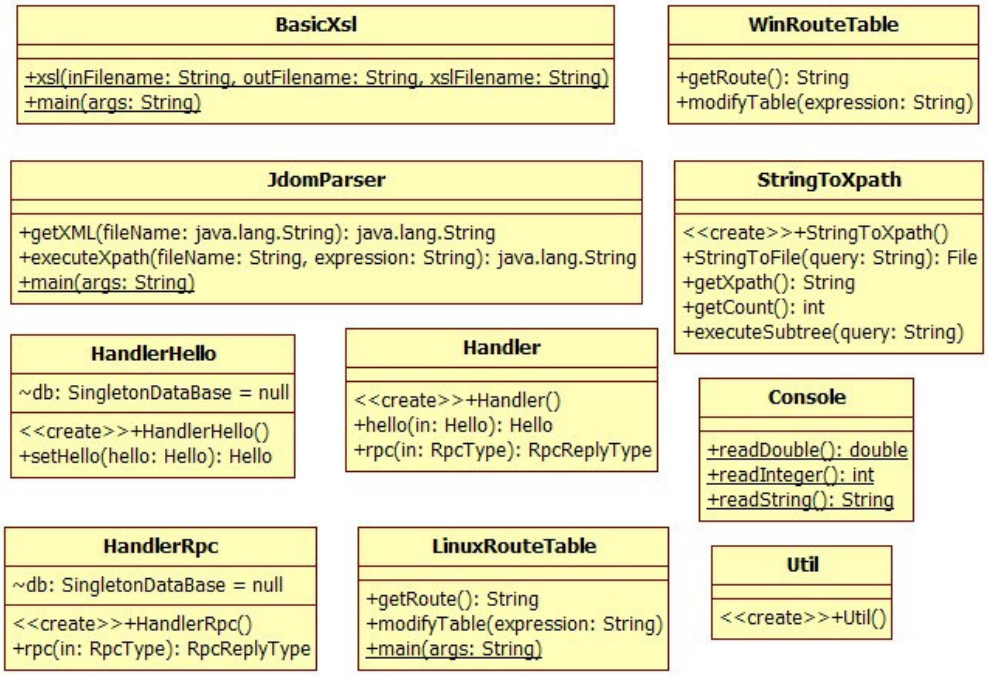
[83] INTAP/OSMIC. Intap. Página na Internet, Julho 2007. http://www.intap.or.jp/e/.

[84] I. Tomoyuki, et al. Experience of implementing NETCONF over SOAP. draft-iijima-netconf-soap-implementation-02.txt. Abril 2007.

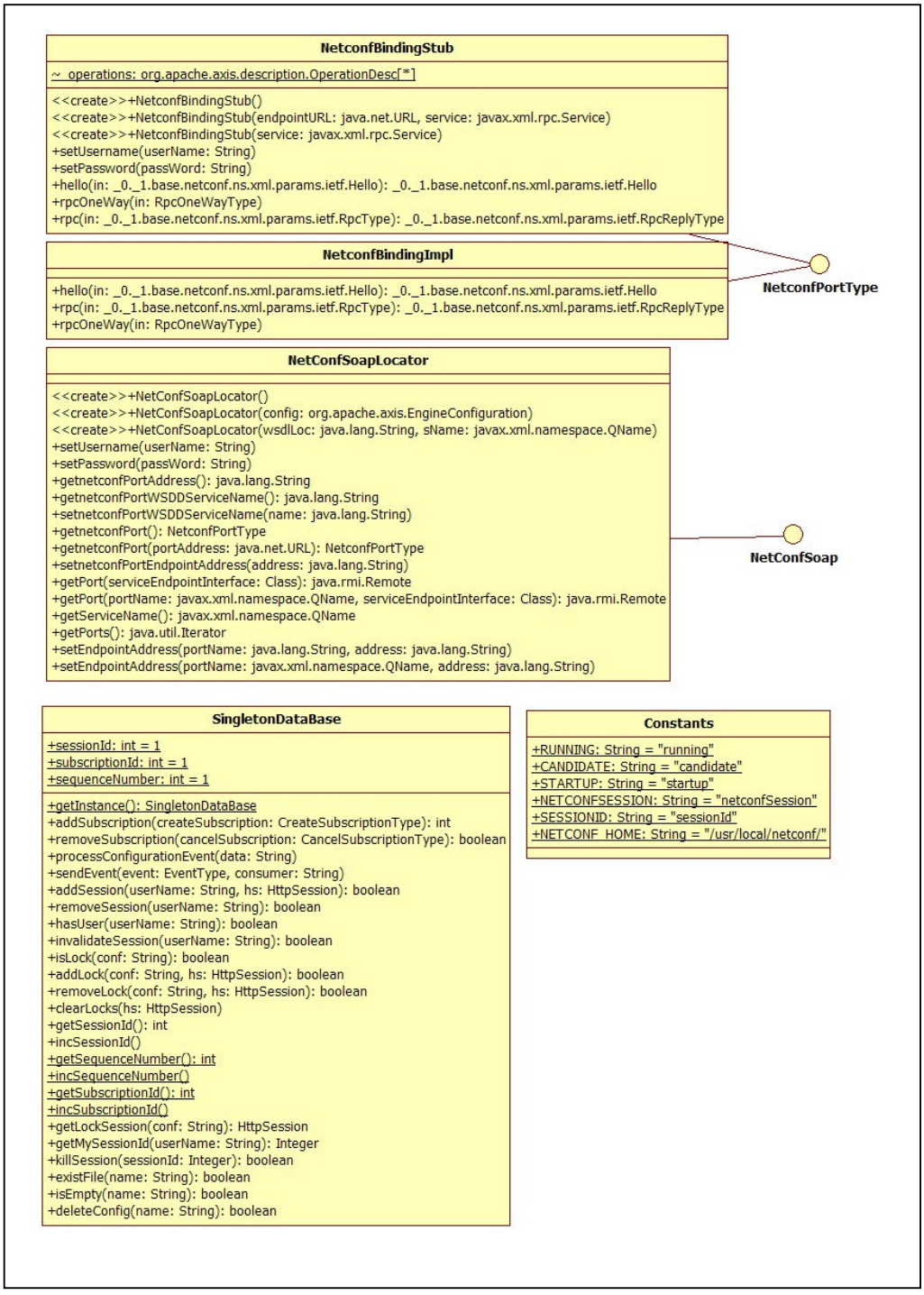
Apêndice A

Diagramas de Classes

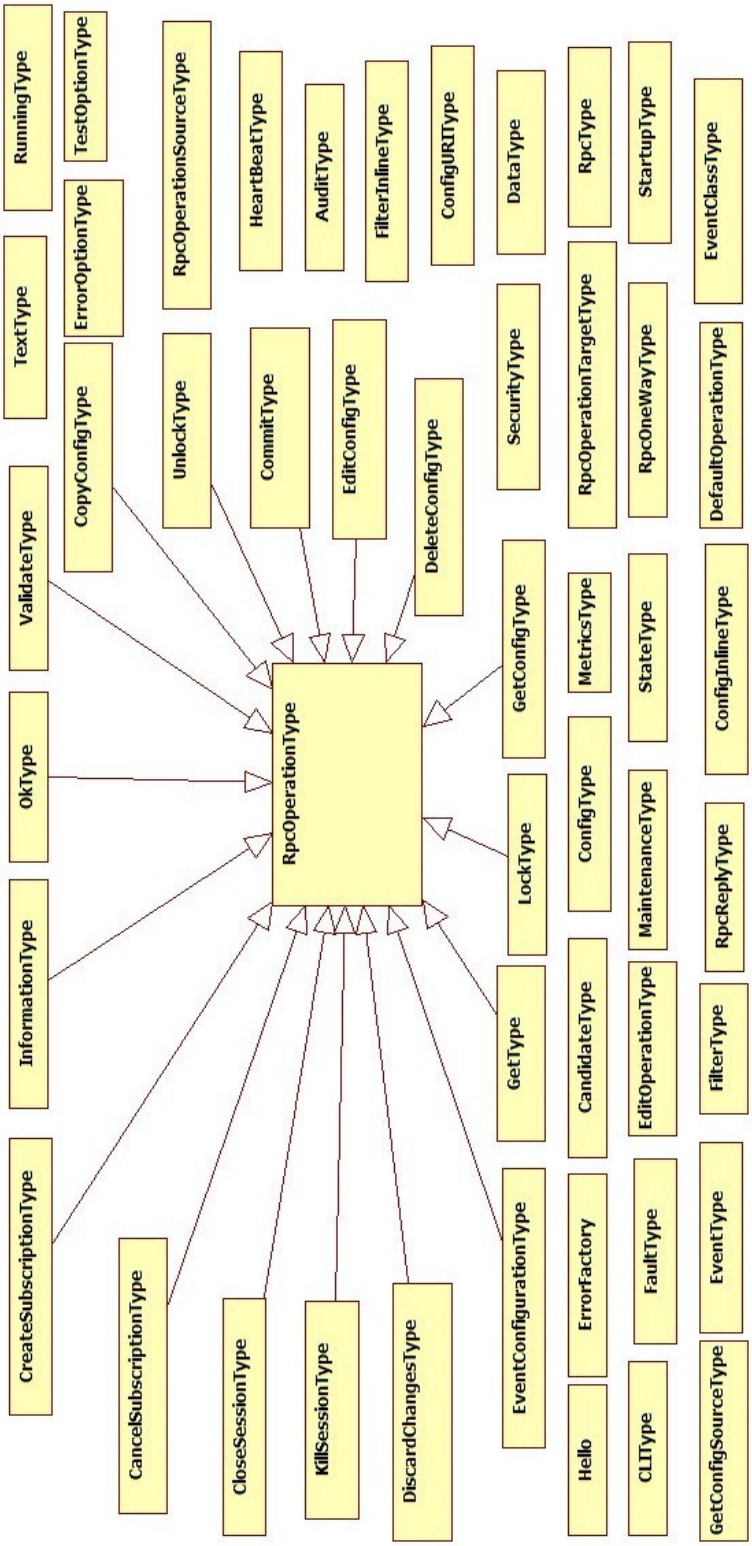
Pacote Handler



Pacote SOAP


NetconfBindingStub
~ operations: org.apache.axis.description.OperationDesc[*]
<<create>>+NetconfBindingStub()
<<create>>+NetconfBindingStub(endpointURL: java.net.URL, service: javax.xml.rpc.Service)
<<create>>+NetconfBindingStub(service: javax.xml.rpc.Service)
+setUsername(userName: String)
+setPassword(passWord: String)
+hello(in: _0._1.base.netconf.ns.xml.params.ietf.Hello): _0._1.base.netconf.ns.xml.params.ietf.Hello
+rpcOneWay(in: RpcOneWayType)
+rpc(in: _0._1.base.netconf.ns.xml.params.ietf.RpcType): _0._1.base.netconf.ns.xml.params.ietf.RpcReplyType
NetconfBindingImpl
+hello(in: _0._1.base.netconf.ns.xml.params.ietf.Hello): _0._1.base.netconf.ns.xml.params.ietf.Hello
+rpc(in: _0._1.base.netconf.ns.xml.params.ietf.RpcType): _0._1.base.netconf.ns.xml.params.ietf.RpcReplyType
+rpcOneWay(in: RpcOneWayType)
NetconfPortType
NetConfSoapLocator
<<create>>+NetConfSoapLocator()
<<create>>+NetConfSoapLocator(config: org.apache.axis.EngineConfiguration)
<<create>>+NetConfSoapLocator(wsdlLoc: java.lang.String, sName: javax.xml.namespace.QName)
+setUsername(userName: String)
+setPassword(passWord: String)
+getnetconfPortAddress(): java.lang.String
+getnetconfPortWSDDServiceName(): java.lang.String
+setnetconfPortWSDDServiceName(name: java.lang.String)
+getnetconfPort(): NetconfPortType
+getnetconfPort(portAddress: java.net.URL): NetconfPortType
+setnetconfPortEndpointAddress(address: java.lang.String)
+getPort(serviceEndpointInterface: Class): java.rmi.Remote
+getPort(portName: javax.xml.namespace.QName, serviceEndpointInterface: Class): java.rmi.Remote
+getServiceName(): javax.xml.namespace.QName
+getPorts(): java.util.Iterator
+setEndpointAddress(portName: java.lang.String, address: java.lang.String)
+setEndpointAddress(portName: javax.xml.namespace.QName, address: java.lang.String)
NetConfSoap
SingletonDataBase
+sessionId: int = 1
+subscriptionId: int = 1
+sequenceNumber: int = 1
+getInstance(): SingletonDataBase
+addSubscription(createSubscription: CreateSubscriptionType): int
+removeSubscription(cancelSubscription: CancelSubscriptionType): boolean
+processConfigurationEvent(data: String)
+sendEvent(event: EventType, consumer: String)
+addSession(userName: String, hs: HttpSession): boolean
+removeSession(userName: String): boolean
+hasUser(userName: String): boolean
+invalidateSession(userName: String): boolean
+isLock(conf: String): boolean
+addLock(conf: String, hs: HttpSession): boolean
+removeLock(conf: String, hs: HttpSession): boolean
+clearLocks(hs: HttpSession)
+getSessionId(): int
+incSessionId()
+getSequenceNumber(): int
+incSequenceNumber()
+getSubscriptionId(): int
+incSubscriptionId()
+getLockSession(conf: String): HttpSession
+getMySessionId(userName: String): Integer
+killSession(sessionId: Integer): boolean
+existFile(name: String): boolean
+isEmpty(name: String): boolean
+deleteConfig(name: String): boolean
Constants
+RUNNING: String = "running"
+CANDIDATE: String = "candidate"
+STARTUP: String = "startup"
+NETCONFSESSION: String = "netconfSession"
+SESSIONID: String = "sessionId"
+NETCONF_HOME: String = "/usr/local/netconf/"

Pacote Base


RunningType
TestOptionType
TextType
ErrorOptionType
RpcOperationSourceType
HeartBeatType
AuditType
FilterInlineType
ConfigURIType
DataType
RpcType
StartupType
EventClassType
ValidateType
CopyConfigType
UnlockType
CommitType
EditConfigType
DeleteConfigType
SecurityType
RpcOperationTargetType
RpcOneWayType
DefaultOperationType
OkType
InformationType
CreateSubscriptionType
CancelSubscriptionType
CloseSessionType
KillSessionType
DiscardChangesType
RpcOperationType
GetConfigType
MetricsType
StateType
ConfigInlineType
LockType
ConfigType
MaintenanceType
RpcReplyType
GetType
CandidateType
EditOperationType
FilterType
EventConfigurationType
ErrorFactory
FaultType
EventType
Hello
CLIType
GetConfigSourceType

Apêndice B

XML Schema para configuração de vlans

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="HTTP://www.w3.org/2001/XMLSchema">

<!-- definition of elements -->
<xs:simpleType name="OperationType">
      <xs:restriction base="xs:string">
        <xs:pattern value="create|delete|config"/>
      </xs:restriction>
</xs:simpleType>
<xs:element name="name" type="xs:string"/>
<xs:element name="tagid" type="xs:positiveInteger"/>
<xs:simpleType name="OptionType">
      <xs:restriction base="xs:string">
        <xs:pattern value="add|delete"/>
      </xs:restriction>
</xs:simpleType>
<xs:element name="portlist" type="xs:string"/>
<xs:simpleType name="TypeType">
      <xs:restriction base="xs:string">
        <xs:pattern value="tagged|untagged|forbidden"/>
      </xs:restriction>
</xs:simpleType>

<xs:complexType name="PortType">
 <xs:sequence>
  <xs:element name="option" type="OptionType"/>
```

```
  <xs:element ref="portlist"/>
  <xs:element name="type" type="TypeType"/>
 </xs:sequence>
</xs:complexType>

<xs:simpleType name="IPType">
    <xs:restriction base="xs:string">
       <xs:pattern
            value="(([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([1-9]?[0-9]|1[0-
                  9][0-9]|2[0-4][0-9]|25[0-5])"/>
    </xs:restriction>
</xs:simpleType>
<xs:element name="mask" type="xs:string"/>


<xs:complexType name="AddressType">
 <xs:sequence>
  <xs:element name="ip" type="IPType"/>
  <xs:element ref="mask"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="newname" type="xs:string"/>
<xs:element name="adverstisement"/>
<xs:element name="nobroadcast"/>

<xs:complexType name="VlanType">
 <xs:sequence>
  <xs:element name="operation" type="OperationType"/>
  <xs:element ref="name"/>
  <xs:element ref="tagid"/>
  <xs:element name="port" type="PortType"/>
  <xs:element name="address" type="AddressType"/>
```

```
  <xs:element ref="newname"/>
  <xs:element ref="adverstisement"/>
  <xs:element ref="nobroadcast"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="VlansType">
 <xs:sequence>
  <xs:element name="vlan" minOccurs="1" maxOccurs="unbounded" type="VlanType"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="vlans" type="VlansType"/>

</xs:schema>
```

Apêndice C

WSDL do Protótipo

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:ietf:params:xml:ns:netconf:soap:1.0"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:ietf:params:xml:ns:netconf:soap:1.0"
xmlns:intf="urn:ietf:params:xml:ns:netconf:soap:1.0"
xmlns:tns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!--WSDL created by Apache Axis version: 1.2
   Built on May 03, 2005 (02:20:24 EDT)-->
    <wsdl:types>
     <schema elementFormDefault="qualified"
targetNamespace="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="hello">
       <sequence>
        <element name="capabilities" type="xsd:anyURI"/>
        <element maxOccurs="1" minOccurs="0" name="session-id"
nillable="true" type="xsd:unsignedInt"/>
       </sequence>
      </complexType>
      <element name="hello" type="tns1:hello"/>
      <element name="hello1" type="tns1:hello"/>
      <complexType name="rpcOperationType">
       <sequence/>
      </complexType>
      <simpleType name="FilterType">
       <restriction base="xsd:string">
        <enumeration value="subtree"/>
        <enumeration value="xpath"/>
       </restriction>
      </simpleType>
      <complexType name="filterInlineType">
       <sequence/>
       <attribute name="type" type="tns1:FilterType"/>
      </complexType>
      <complexType name="getType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
```

```
         <sequence>
          <element maxOccurs="1" minOccurs="0" name="filter"
nillable="true" type="tns1:filterInlineType"/>
          <element maxOccurs="1" minOccurs="0" name="query" nillable="true"
type="xsd:string"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="runningType">
       <sequence/>
      </complexType>
      <complexType name="candidateType">
       <sequence/>
      </complexType>
      <complexType name="startupType">
       <sequence/>
      </complexType>
      <complexType name="configURIType">
       <simpleContent>
        <extension>
         <attribute name="value" type="xsd:anyURI"/>
        </extension>
       </simpleContent>
      </complexType>
      <complexType name="getConfigSourceType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="running" nillable="true"
type="tns1:runningType"/>
        <element maxOccurs="1" minOccurs="0" name="candidate"
nillable="true" type="tns1:candidateType"/>
        <element maxOccurs="1" minOccurs="0" name="startup" nillable="true"
type="tns1:startupType"/>
        <element maxOccurs="1" minOccurs="0" name="url" nillable="true"
type="tns1:configURIType"/>
       </sequence>
      </complexType>
      <complexType name="getConfigType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element maxOccurs="1" minOccurs="0" name="source"
type="tns1:getConfigSourceType"/>
          <element maxOccurs="1" minOccurs="0" name="filter"
type="tns1:filterInlineType"/>
          <element maxOccurs="1" minOccurs="0" name="query" nillable="true"
type="xsd:string"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="rpcOperationTargetType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="running" nillable="true"
type="tns1:runningType"/>
```

```
        <element maxOccurs="1" minOccurs="0" name="candidate"
nillable="true" type="tns1:candidateType"/>
        <element maxOccurs="1" minOccurs="0" name="startup" nillable="true"
type="tns1:startupType"/>
        <element maxOccurs="1" minOccurs="0" name="url" nillable="true"
type="tns1:configURIType"/>
       </sequence>
      </complexType>
      <simpleType name="defaultOperationType">
       <restriction base="xsd:string">
        <enumeration value="merge"/>
        <enumeration value="replace"/>
        <enumeration value="none"/>
       </restriction>
      </simpleType>
      <simpleType name="testOptionType">
       <restriction base="xsd:string">
        <enumeration value="test-then-set"/>
        <enumeration value="set"/>
       </restriction>
      </simpleType>
      <simpleType name="errorOptionType">
       <restriction base="xsd:string">
        <enumeration value="stop-on-error"/>
        <enumeration value="ignore-error"/>
        <enumeration value="rollback-on-error"/>
       </restriction>
      </simpleType>
      <complexType name="cliType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="application"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="cmd" nillable="true"
type="xsd:string"/>
       </sequence>
      </complexType>
      <complexType name="configInlineType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="configuration"
nillable="true" type="tns1:configurationType"/>
        <element maxOccurs="1" minOccurs="0" name="text" nillable="true"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="cli" nillable="true"
type="tns1:cliType"/>
       </sequence>
      </complexType>
      <complexType name="editConfigType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element name="target" type="tns1:rpcOperationTargetType"/>
          <element maxOccurs="1" minOccurs="0" name="default-operation"
nillable="true" type="tns1:defaultOperationType"/>
          <element maxOccurs="1" minOccurs="0" name="test-option"
nillable="true" type="tns1:testOptionType"/>
```

```
          <element maxOccurs="1" minOccurs="0" name="error-option"
nillable="true" type="tns1:errorOptionType"/>
          <element maxOccurs="1" minOccurs="0" name="config"
type="tns1:configInlineType"/>
          <element maxOccurs="1" minOccurs="0" name="url"
type="tns1:configURIType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="rpcOperationSourceType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="config" nillable="true"
type="tns1:configInlineType"/>
        <element maxOccurs="1" minOccurs="0" name="running" nillable="true"
type="tns1:runningType"/>
        <element maxOccurs="1" minOccurs="0" name="candidate"
nillable="true" type="tns1:candidateType"/>
        <element maxOccurs="1" minOccurs="0" name="startup" nillable="true"
type="tns1:startupType"/>
        <element maxOccurs="1" minOccurs="0" name="url" nillable="true"
type="tns1:configURIType"/>
       </sequence>
      </complexType>
      <complexType name="copyConfigType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element maxOccurs="1" minOccurs="0" name="source"
type="tns1:rpcOperationSourceType"/>
          <element maxOccurs="1" minOccurs="0" name="target"
type="tns1:rpcOperationTargetType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="deleteConfigType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element name="target" type="tns1:rpcOperationTargetType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="lockType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element name="target" type="tns1:rpcOperationTargetType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="unlockType">
       <complexContent>
```

```
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element name="target" type="tns1:rpcOperationTargetType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="closeSessionType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence/>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="SessionId">
       <simpleContent>
        <extension/>
       </simpleContent>
      </complexType>
      <complexType name="killSessionType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element maxOccurs="1" minOccurs="0" name="session-id"
nillable="true" type="tns1:SessionId"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="faultType">
       <sequence/>
      </complexType>
      <complexType name="informationType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence/>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="stateType">
       <sequence/>
      </complexType>
      <complexType name="auditType">
       <sequence/>
      </complexType>
      <complexType name="eventConfigurationType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence/>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="dataType">
       <sequence/>
      </complexType>
      <complexType name="maintenanceType">
```

```
       <sequence/>
      </complexType>
      <complexType name="metricsType">
       <sequence/>
      </complexType>
      <complexType name="securityType">
       <sequence/>
      </complexType>
      <complexType name="heartBeatType">
       <sequence/>
      </complexType>
      <complexType name="eventClassType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="fault" nillable="true"
type="tns1:faultType"/>
        <element maxOccurs="1" minOccurs="0" name="information"
nillable="true" type="tns1:informationType"/>
        <element maxOccurs="1" minOccurs="0" name="state" nillable="true"
type="tns1:stateType"/>
        <element maxOccurs="1" minOccurs="0" name="audit" nillable="true"
type="tns1:auditType"/>
        <element maxOccurs="1" minOccurs="0" name="configuration"
nillable="true" type="tns1:eventConfigurationType"/>
        <element maxOccurs="1" minOccurs="0" name="data" nillable="true"
type="tns1:dataType"/>
        <element maxOccurs="1" minOccurs="0" name="maintenance"
nillable="true" type="tns1:maintenanceType"/>
        <element maxOccurs="1" minOccurs="0" name="metrics" nillable="true"
type="tns1:metricsType"/>
        <element maxOccurs="1" minOccurs="0" name="security"
nillable="true" type="tns1:securityType"/>
        <element maxOccurs="1" minOccurs="0" name="audit" nillable="true"
type="tns1:auditType"/>
        <element maxOccurs="1" minOccurs="0" name="heartBeat"
nillable="true" type="tns1:heartBeatType"/>
       </sequence>
      </complexType>
      <complexType name="createSubscriptionType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element maxOccurs="1" minOccurs="0" name="eventClass"
type="tns1:eventClassType"/>
          <element maxOccurs="1" minOccurs="0" name="consumer"
type="xsd:string"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="cancelSubscriptionType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence>
          <element maxOccurs="1" minOccurs="0" name="subscriptionId"
type="xsd:string"/>
         </sequence>
```

```
        </extension>
       </complexContent>
      </complexType>
      <complexType name="rpcType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="get" nillable="true"
type="tns1:getType"/>
        <element maxOccurs="1" minOccurs="0" name="get-config"
nillable="true" type="tns1:getConfigType"/>
        <element maxOccurs="1" minOccurs="0" name="edit-config"
nillable="true" type="tns1:editConfigType"/>
        <element maxOccurs="1" minOccurs="0" name="copy-config"
nillable="true" type="tns1:copyConfigType"/>
        <element maxOccurs="1" minOccurs="0" name="delete-config"
nillable="true" type="tns1:deleteConfigType"/>
        <element maxOccurs="1" minOccurs="0" name="lock" nillable="true"
type="tns1:lockType"/>
        <element maxOccurs="1" minOccurs="0" name="unlock" nillable="true"
type="tns1:unlockType"/>
        <element maxOccurs="1" minOccurs="0" name="close-session"
nillable="true" type="tns1:closeSessionType"/>
        <element maxOccurs="1" minOccurs="0" name="kill-session"
nillable="true" type="tns1:killSessionType"/>
        <element maxOccurs="1" minOccurs="0" name="createSubscription"
nillable="true" type="tns1:createSubscriptionType"/>
        <element maxOccurs="1" minOccurs="0" name="cancelSubscription"
nillable="true" type="tns1:cancelSubscriptionType"/>
       </sequence>
       <attribute name="message-id" type="xsd:string"/>
      </complexType>
      <element name="rpc" type="tns1:rpcType"/>
      <complexType name="okType">
       <complexContent>
        <extension base="tns1:rpcOperationType">
         <sequence/>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="rpcReplyType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="ok" nillable="true"
type="tns1:okType"/>
        <element maxOccurs="1" minOccurs="0" name="data" nillable="true"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="subscriptionId"
nillable="true" type="xsd:string"/>
       </sequence>
       <attribute name="message-id" type="xsd:string"/>
      </complexType>
      <element name="rpc-reply" type="tns1:rpcReplyType"/>
      <complexType name="eventType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="subscriptionId"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="eventClass"
type="tns1:eventClassType"/>
```

```
        <element maxOccurs="1" minOccurs="0" name="supplier"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="sequenceNumber"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="dateAndTime"
type="xsd:string"/>
        <element maxOccurs="1" minOccurs="0" name="data"
type="xsd:string"/>
       </sequence>
      </complexType>
      <complexType name="rpcOneWayType">
       <sequence>
        <element maxOccurs="1" minOccurs="0" name="event" nillable="true"
type="tns1:eventType"/>
       </sequence>
       <attribute name="message-id" type="xsd:string"/>
      </complexType>
      <element name="rpcOneWay" type="tns1:rpcOneWayType"/>
     </schema>
    </wsdl:types>

      <wsdl:message name="rpcRequest">

         <wsdl:part element="tns1:rpc" name="rpc"/>

      </wsdl:message>

      <wsdl:message name="helloResponse">

         <wsdl:part element="tns1:hello1" name="hello"/>

      </wsdl:message>

      <wsdl:message name="rpcResponse">

         <wsdl:part element="tns1:rpc-reply" name="rpc-reply"/>

      </wsdl:message>

      <wsdl:message name="helloRequest">

         <wsdl:part element="tns1:hello" name="hello"/>

      </wsdl:message>

      <wsdl:message name="rpcOneWayResponse">

      </wsdl:message>

      <wsdl:message name="rpcOneWayRequest">

         <wsdl:part element="tns1:rpcOneWay" name="rpcOneWay"/>

      </wsdl:message>

      <wsdl:portType name="netconfPortType">
```

```
         <wsdl:operation name="hello" parameterOrder="hello">

            <wsdl:input message="impl:helloRequest" name="helloRequest"/>

            <wsdl:output message="impl:helloResponse"
name="helloResponse"/>

         </wsdl:operation>

         <wsdl:operation name="rpc" parameterOrder="rpc">

            <wsdl:input message="impl:rpcRequest" name="rpcRequest"/>

            <wsdl:output message="impl:rpcResponse" name="rpcResponse"/>

         </wsdl:operation>

         <wsdl:operation name="rpcOneWay" parameterOrder="rpcOneWay">

            <wsdl:input message="impl:rpcOneWayRequest"
name="rpcOneWayRequest"/>

            <wsdl:output message="impl:rpcOneWayResponse"
name="rpcOneWayResponse"/>

         </wsdl:operation>

      </wsdl:portType>

      <wsdl:binding name="netconfPortSoapBinding"
type="impl:netconfPortType">

         <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

         <wsdl:operation name="hello">

            <wsdlsoap:operation soapAction=""/>

            <wsdl:input name="helloRequest">

               <wsdlsoap:body use="literal"/>

            </wsdl:input>

            <wsdl:output name="helloResponse">

               <wsdlsoap:body use="literal"/>

            </wsdl:output>

         </wsdl:operation>

         <wsdl:operation name="rpc">
```

```
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="rpcRequest">
               <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="rpcResponse">
               <wsdlsoap:body use="literal"/>
            </wsdl:output>
         </wsdl:operation>
         <wsdl:operation name="rpcOneWay">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="rpcOneWayRequest">
               <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="rpcOneWayResponse">
               <wsdlsoap:body use="literal"/>
            </wsdl:output>
         </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="NetConfSoap">
         <wsdl:port binding="impl:netconfPortSoapBinding"
name="netconfPort">
            <wsdlsoap:address
location="http://10.1.4.5:8080/axis/services/NetConfSoap"/>
         </wsdl:port>
      </wsdl:service>
   </wsdl:definitions>
```

Anexo A

NETCONF – Exemplos de Mensagens

1. Operações Básicas

a. <get>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
      <filter type="subtree">
        <top xmlns="HTTP://example.com/schema/1.2/stats">
          <interfaces>
            <interface>
              <ifName>eth0</ifName>
            </interface>
          </interfaces>
        </top>
      </filter>
    </get>
</rpc>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      <top xmlns="HTTP://example.com/schema/1.2/stats">
        <interfaces>
          <interface>
            <ifName>eth0</ifName>
            <ifInOctets>45621</ifInOctets>
            <ifOutOctets>774344</ifOutOctets>
          </interface>
        </interfaces>
      </top>
    </data>
</rpc-reply>
```

b. <get-config>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
      <source>
        <running/>
      </source>
      <filter type="subtree">
        <top xmlns="HTTP://example.com/schema/1.2/config">
          <users/>
        </top>
      </filter>
    </get-config>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      <top xmlns="HTTP://example.com/schema/1.2/config">
        <users>
          <user>
            <name>root</name>
            <type>superuser</type>
            <full-name>Charlie Root</full-name>
            <company-info>
              <dept>1</dept>
              <id>1</id>
            </company-info>
          </user>
          <!—demais elementos <user> aparece aqui... -->
        </users>
      </top>
    </data>
</rpc-reply>
```

c. <edit-config>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
      <target>
       <running/>
      </target>
      <config>
       <top xmlns="HTTP://example.com/schema/1.2/config">
        <interface>
         <name>Ethernet0/0</name>
         <mtu>1500</mtu>
        </interface>
       </top>
      </config>
     </edit-config>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <ok/>
</rpc-reply>
```

d. <copy-config>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <copy-config>
      <target>
       <running/>
      </target>
      <source>
       <url>HTTPs://user@example.com:passphrase/cfg/new.txt</url>
      </source>
     </copy-config>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <ok/>
</rpc-reply>
```

e. <delete-config>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <delete-config>
      <target>
        <startup/>
      </target>
    </delete-config>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>
```

f. <lock>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
      <target>
        <running/>
      </target>
    </lock>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/> <!-- lock com sucesso -->
</rpc-reply>
```

g. <unlock>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <unlock>
      <target>
       <running/>
      </target>
    </unlock>
</rpc>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>
```

h. <close-session>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <close-session/>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>
```

i. <kill-session>

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <kill-session>
      <session-id>4</session-id>
    </kill-session>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>
```

2. Mensagem NETCONF SOAP sobre HTTP

A seguir é mostrada a mensagen <hello> enviada do gerente (cliente - C) para o agente (Servidor - S) e a resposta do agente para o gerente.

```
C: POST /netconf HTTP/1.1
C: Host: netconfdevice
C: Content-Type: text/xml; charset=utf-8
C: Accept: application/soap+xml, text/*
C: Cache-Control: no-cache
C: Pragma: no-cache
C: Content-Length: 376
```

```
C:
C: <?xml version="1.0" encoding="UTF-8"?>
C: <soapenv:Envelope
C:   xmlns:soapenv="HTTP://www.w3.org/2003/05/soap-envelope">
C:   <soapenv:Body>
C:     <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:       <capabilities>
C:         <capability>
C:           urn:ietf:params:netconf:base:1.0
C:         </capability>
C:       </capabilities>
C:     </hello>
C:   </soapenv:Body>
C: </soapenv:Envelope>

S: HTTP/1.1 200 OK
S: Content-Type: application/soap+xml; charset=utf-8
S: Content-Length: 600
S:
S: <?xml version="1.0" encoding="UTF-8"?>
S: <soapenv:Envelope
S:   xmlns:soapenv="HTTP://www.w3.org/2003/05/soap-envelope">
S:   <soapenv:Body>
S:     <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:       <capabilities>
S:         <capability>
S:           urn:ietf:params:netconf:base:1.0
S:         </capability>
S:         <capability>
S:           urn:ietf:params:netconf:capability:startup:1.0
S:         </capability>
S:       </capabilities>
S:       <session-id>4</session-id>
S:     </hello>
S:   </soapenv:Body>
S: </soapenv:Envelope>
```

3. Mensagem de Notificação para SOAP sobre HTTP

```
C: POST /netconf HTTP/1.1
C: Host: netconfdevice
C: Content-Type: text/xml; charset=utf-8
C: Accept: application/soap+xml, text/*
C: Cache-Control: no-cache
C: Pragma: no-cache
C: Content-Length: 465
C:
C: <?xml version="1.0" encoding="UTF-8"?>
C: <soapenv:Envelope
C:   xmlns:soapenv="HTTP://www.w3.org/2003/05/soap-envelope">
C:   <soapenv:Body>
C:     <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
C:       <create-subscription>
C:       </create-subscription>
C:     </rpc>
C:   </soapenv:Body>
C: </soapenv:Envelope>
```

A resposta:

```
S: HTTP/1.1 200 OK
S: Content-Type: application/soap+xml; charset=utf-8
S: Content-Length: 917
S:
S: <?xml version="1.0" encoding="UTF-8"?>
S: <soapenv:Envelope
S:   xmlns:soapenv="HTTP://www.w3.org/2003/05/soap-envelope">
S:   <soapenv:Body>
S:     <rpc-reply message-id="101"xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
S:       <data>
S:         <top xmlns=
                  "HTTP://example.com/schema/1.2/notification">
S:           <subscriptionId>123456</subscriptionId>
S:         </top>
S:       </data>
S:     </rpc-reply>
```

```
S:   </soapenv:Body>
S: </soapenv:Envelope>
```

Algum tempo depois, um evento:

```
S: HTTP/1.1 200 OK
S: Content-Type: application/soap+xml; charset=utf-8
S: Content-Length: 917
S:
S: <?xml version="1.0" encoding="UTF-8"?>
S: <soapenv:Envelope
S:   xmlns:soapenv="HTTP://www.w3.org/2003/05/soap-envelope">
S:   <soapenv:Body>
S:     <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
S:      <subscriptionID>123456</subscriptionID>
S:      <eventClasses><configuration/><audit/></eventClasses>
S:      <sequenceNumber>2</sequenceNumber>
S:      <dateAndTime>2000-01-12T12:13:14Z</dateAndTime>
S:        <data>
S:          <user>Fred Flinstone</user>
S:             <operation>
S:              <edit-config>
S:             <target>
S:              <running/>
S:             </target>
S:            <config>
S:             <top xmlns="HTTP://example.com/schema/1.2/config">
S:                 <interface>
S:                   <name>Ethernet0/0</name>
S:                   <mtu>1500</mtu>
S:                 </interface>
S:              </top>
S:           </config>
S:          </edit-config>
S:        </operation>
S:       </data>
S:    </notification>
S:   </soapenv:Body>
S: </soapenv:Envelope>
```