Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática,
2010

Ricardo André
Martins Mendes

Suporte de gestão de Plataforma IMS baseado em NETCONF

Ricardo André Martins Mendes

Suporte de gestão de Plataforma IMS baseado em NETCONF

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Pedro Alexandre de Sousa Gonçalves, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda e do Professor Doutor Rui Luís Andrade de Aguiar, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president	**Professor Doutor José Carlos da Silva Neves** Professor Catedrático da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)
vogais / examiners committee	**Professor Doutor Rui Luís Andrade de Aguiar** Professor Auxiliar da Universidade de Aveiro (orientador)
	Professor Doutor Pedro Alexandre de Sousa Gonçalves Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda (co-orientador)
	Professor Doutor Paulo Simões Professor Associado da Universidade de Coimbra

agradecimentos / acknowledgements

Quando tomei a decisão de me candidatar ao Mestrado, tinha consciência das dificuldades em conciliar a actividade profissional com a actividade académica, o que eu não fazia ideia, era que, acrescido a tudo isto, ainda teria de conciliar com a actividade de de ser pai, a melhor de todas as actividades . . .

O meu maior agradecimento é destinado a minha companheira Tânia Tavares, por todo o apoio e compreensão que sempre me deu, mesmo nos momentos em que não pude estar presente durante a gravidez.

Agradeço ao meu orientador científico, Professor Pedro Gonçalves, pelo seu acompanhamento e disponibilidade. Agradeço-lhe ainda pela sua acessibilidade e facilidade de comunicação, que sempre sempre me fez sentir a vontade, e indicando-me o melhor caminho a seguir.

Agradeço ao meu orientador científico, Professor Rui Aguiar, pelo seu seu rigor e experiência.

A todos, muito obrigado.

palavras-chave Gestão de redes, Qualidade de Serviço, Redes de Próxima Geração.

resumo

As Redes de Próxima Geração ou NGN (*Next Generation Networks*) representam um novo conceito de rede que consiste na convergência da voz, dos dados e de novas aplicações multimédia. As vantagens são obvias, a NGN consegue associar a redução dos custos operacionais das redes de operador ao incremento da possibilidade de geração de novas fontes de receita, pois torna possível oferecer uma grande diversidade de serviços multimédia. No entanto, para que tal aconteça, torna-se essencial a existência de arquitecturas de rede que sejam extremamente flexíveis para a disponibilização de serviços diferenciados de acordo com as necessidades de cada utilizador.

A arquitectura das NGN tem vindo a ser desenvolvida por várias entidades normalizadoras como são exemplo o 3GPP e o TISPAN. No centro desta arquitectura está uma plataforma de distribuição de serviços multimédia para redes IP designada de *IP Multimedia Subsystem* (IMS). Os esforços de normalização têm-se centrado na especificação dos elementos intervenientes na operação da rede, não se tendo ainda dedicado estas organizações à definição da infraestrutura de gestão da rede.

Tem existido desde sempre esforços para o desenvolvimento de novas técnicas e modelos para gestão de redes. Actualmente a tendência que tem ganho força na área de gestão de redes é o recurso a tecnologias baseadas em XML. A principal razão é que as actuais tecnologias de gestão de redes não satisfazem todos os requisitos pretendidos por os operadores de telecomunicações, logo, novas alternativas estão sendo exploradas. Por outro lado, há uma significativa disseminação e uma clara aplicação de tecnologias baseadas em XML no domínio das tecnologia de informação (TI) e Internet. Um exemplo relevante de tecnologia XML aplicada à gestão de redes são os Web Services. Outros exemplos de aplicação de para gestão de redes são a arquitectura de gestão WBEM desenvolvida pelo DMTF e por fim, uma promissora iniciativa do IETF, que corresponde ao desenvolvimento do protocolo NetConf, também ele baseado em XML.

Inevitavelmente, o domínio de operação da gestão de redes também tem vindo a evoluir deparando-se com novos desafios, como sejam o controlo de qualidade de serviço (QoS), diferenciação de tráfego e o suporte multifacetado de serviços a que as redes actuais de dados têm sido sujeitas na convergência para a próxima geração.

O presente trabalho propõe uma solução de gestão de plataforma IMS baseado em NETCONF. A solução de gestão proposta consiste na interligação de um servidor de gestão CIM com os equipamentos de rede IMS utilizando uma API NETCONF *open-source* existente. Pretende-se assim estudar o desempenho do protocolo de gestão, especialmente no que diz respeito ao volume de sinalização e aos requisitos de memória, assim como avaliar a aplicabilidade desta tecnologia para um cenário de gestão de uma rede de operador de telecomunicações.

keywords Network Management, Quality of Service, Next Generation Networks.

abstract

The Next Generation Networks (NGN) represents a new network concept that consists in the convergence of voice, data and new multimedia applications. The advantages are obvious, the NGN can associate operator networks operation costs reduction to the increase possibility of new revenues sources, due to the diversity of the offered services. Although, for that happens, becomes mandatory that existence of network architectures very flexible to provide differentiated services according customers needs.

The NGN architecture is being developed by several standardization bodies like the 3GPP and TISPAN. In the center of this architecture there is one multimedia service delivery platform for IP networks, named as IP Multimedia Subsystem (IMS). The standardization efforts has been focusing on the network operating elements, not focusing yet the network management infrastructure.

Always has been efforts for the development of new techniques and models for the network management. Nowadays the tendency that has gain strength in the network management area is the use of XML based technologies. The main reason is that the current network management technologies do not fulfil all the requirement demanded by the network operators, so, new technologies has been investigated. By other hand, there is one significant dissemination and obvious application of XML based technologies in the information technologies domain and Internet. One relevant example of the XML based technologies applied to network management are the Web Services. Other application examples for the network management are the management architecture WBEM, developed by the DMTF and at last, one promising initiative from the IETF, that is the development of the NETCONF protoccol, also based in XML.

Inevitably, the network management operation domain has been evolve facing new challenges, as the Quality of Service (QoS) control, traffic differentiation and the multifaceted support of services that the current networks has been subject in the convergence process to the next generation.

The current work proposes one IMS platform management solution based on NETCONF. The proposed management solution consists in the integration of one CIM management server with the IMS network equipment trough one existing open-source NETCONF API. The purpose of the work is the study of the management protocol performance, specifically in what concerns about the traffic signalization volume and memory consumption, as well as validate the applicability of this technology to one telco operator network management scenario.

Conteúdo

Lista de Figuras

Lista de Tabelas

Índice Remissivo

HMOM *Hypermedia Object Manager*. 18

HTTP *HyperText Transfer Protocol*. 10, 12, 13, 16–19, 22, 24–26, 30, 44

HTTPS *HyperText Transfer Protocol Secure*. 30, 44

IETF *Internet Engineering Task Force*. 5, 9, 12, 29, 45

IM *Information Model*. 31

IP *Internet Protocol*. 5, 10

ISDN *Integrated Services Digital Network*. 5

ISO *International Organization for Standardization*. 1

ITU *International Telecommunication Union*. 5

LDAP *Lightweight Directory Access Protocol*. 11, 12, 15

MIB *Management Information Base*. 20, 31, 45

MOF *Managed Object Format*. 18, 20, 22, 23, 31, 44

NETCONF *Network Configuration*. 29–32, 41, 45

NGN *Next Generation Networks*. 2

NSIS *Next Steps In Signaling*. 7

OID *Object IDentification*. 31

OO *Object Oriented*. 20, 31

OSI *Open Systems Interconnection model*. 1

PBM *Policy Based Management*. 2, 6

PBN *Policy Based Networking*. 13, 44

PBNM *Policy Based Network Management*. 1–3, 5, 7, 9, 11, 13–15, 44

PDP *Policy Definition Point*. 10, 12–15

PEP *Policy Enforcement Point*. 12–15

PIB *Policy Information Base*. 31

PLMN *Public Land Mobile Network*. 5

PMT *Policy Management Tool*. 10, 11, 14

PP *Policy Proxie*. 14

PR *Policy Repository.* 10

PSTN *Public Switched Telephone Network.* 5

QoS *Quality of service.* 2, 6, 9

RADIUS *Remote Authentication Dial In User Service.* 30

RFC *Request For Comments.* 12, 29, 31, 45

RMON *Remote Network MONitoring.* 12

RPC *Remote Procedure Calls.* 29–32

RSVP *Resource ReSerVation Protocol.* 13

SLA *Service level agreement.* 6, 11

SMI *Structure of Management Information.* 18, 31

SNMP *Simple Network Management Protocol.* 4, 7, 10, 12, 14–16, 31, 45

SOA *Service Oriented Architecture.* 45

SOAP *Simple Object Access Protocol.* 29, 41, 45

SPPI *Structure of Policy Provising Information.* 31

SSH *Secure Shell.* 29, 41

SSL *Secure Sockets Layer.* 11, 12, 44

TCP *Transmission Control Protocol.* 30, 33, 34, 39

UDDI *Universal Description Discovery and Integration.* 45

UML *Unified Modeling Language.* 20, 31

URI *Uniform Resource Identifier.* 32

URN *Uniform Resource Name.* 32

VPN *Virtual Private Network.* 13

WBEM *Web-Based Enterprise Management.* 16–18, 20, 26, 44

WS *Web Services.* 5

WWW *World Wide Web.* 16

XML *eXtensible Markup Language.* 7, 11, 17–19, 22–24, 26, 28, 29, 31, 32, 37, 41, 44, 45

XPath *XML Path Language.* 45

XSL *EXtensible Stylesheet Language.* 23, 44

Capítulo 1

Introdução

A gestão de redes e sistemas consiste no processo de monitorizar e controlar redes e sistemas de modo a assegurar a sua operação e maximizar a qualidade dos serviços prestados aos utilizadores. As actividades de gestão de redes podem ser entendidas a vários níveis distintos, abrangendo aspectos que vão desde a monitorização de simples elementos de rede, até à gestão de aplicações de processamento distribuído.

Conceptualmente, a gestão de redes encontra-se dividida em áreas funcionais, que podem ou não estar todas incluídas num mesmo sistema de gestão. A *International Organization for Standardization* (ISO) propôs um modelo de gestão de redes conhecido como FCAPS [52] que define 5 áreas distintas, tal como ilustrado na Figura 1.1. São elas, a gestão de falhas (*Fault Management*), a gestão de configuração (*ConFiguration Management*), a gestão de contabilização (*Accounting Management*), a gestão de desempenho (*Performance Management*) e a gestão de segurança (*Security Management*) [55]. As áreas funcionais são endereçadas pelos dois modelos de gestão, o modelo *Open Systems Interconnection model* (OSI) e o modelo TCP/IP.

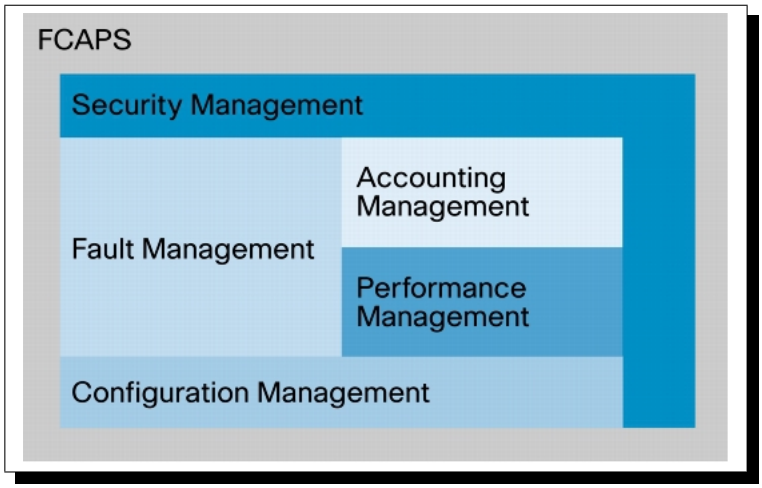

Figura 1.1: Áreas funcionais do FCAPS.

À medida que aumenta a complexidade e dimensão dos serviços suportados pelo operador de comunicações, as suas necessidades de resposta em termos de controlo, coordenação e monitorização aumentam, de modo a assegurar a oferta de serviços com a qualidade requerida.

Idealmente, essas necessidades apontam para uma forma de gestão integrada que permita o acesso à informação da rede a diversos níveis. Estes aspectos tornam evidente a necessidade de incorporar, de algum modo, nas infraestruturas de comunicações ferramentas para recolher, transferir, arquivar, analisar e apresentar informação de gestão da rede, e para monitorizar, controlar e coordenar os processos de comunicação.

Os sistemas de gestão deverão ser capazes de, para uma determinada situação, oferecer facilidades para reconfiguração do sistema na sua totalidade, se necessário, sem que o administrador da rede tenha que se preocupar com os detalhes de configuração dos diferentes equipamentos que constituem a rede.

A crescente evolução das infraestruturas, em termos de complexidade e heterogeneidade de tecnologias, carece obviamente da definição de mecanismos de configuração normalizados, de forma a garantir uma gestão homogénea dos diferentes equipamentos que constituem a rede. Neste contexto, a *Policy Based Network Management* (PBNM), tem vindo a ganhar popularidade e consistência [70], uma vez os sistemas PBNM permitem que a configuração e reconfiguração da rede seja executada de uma forma "automática", através da definição de regras de alto-nível e transparentes à tecnologia subjacente.

1.1 Enquadramento

À medida que as infraestruturas de rede, assim como o seu licenciamento, nomeadamente nas redes moveis, consomem mais recursos financeiros, maior é também a necessidade de encontrar novas formas de rentabilização dos investimentos efectuados por parte dos operadores de telecomunicações. Como nos últimos tempos a economia tem sido pouco generosa com as novas tecnologias, têm surgido novas tentativas para explorar pequenos nichos de mercado, ou soluções pontuais com algumas perspectivas de mercado, mas não as suficientes para uma clara evolução positiva do meio envolvente. *Quality of service* (QoS), poderá revelar-se uma clara oportunidade para devolver alguma estabilidade ao meio, por permitir oferecer gamas de serviço diferenciadas à medida do cliente e do seu negócio, permitindo a reutilização dos recursos para todos os serviços prestados. A junção das novas tecnologias de comunicação aliadas a formas de gestão eficientes, definidas com base no negócio e que simplificam razoavelmente a gestão das redes, parece potenciar um cenário promissor num futuro próximo.

Tradicionalmente, a gestão QoS nas redes de telecomunicações tem sido alcançado por uma combinação de *best-effort* na entrega de pacotes, reserva de recursos de rede (*IntServ*) ou marcação de pacotes de dados (*DiffServ*) nos percursos da rede. O *design* das arquitecturas *Next Generation Networks* (NGN) não é compatível com este tipo de abordagem, tornando-a insuficiente. Até ao momento, os trabalhos de normalização das arquitecturas NGN têm-se limitado a especificar apenas os detalhes relativos ao funcionamento da rede não tendo ainda sido focada a arquitectura de gestão e consequentemente as suas áreas funcionais, como a gestão de desempenho, onde se encontra a gestão QoS.

Apesar de ainda não se encontrar definida a arquitectura de gestão, já foi adoptado o paradigma PBNM para a implementação do controlo de admissão em quase todas as arquitecturas NGN propostas [70]. Conceptualmente diferente da abordagem convencional, a paradigma de gestão PBNM consiste na definição de um conjunto de regras ou políticas descritas numa linguagem intuitiva, e que agregam os direitos de acesso e utilização dos recursos da rede com base no perfil estabelecido na aplicação.

Uma consequência de arquitecturas modulares e hierárquicas é o facto dos sistemas que

vão constituir a rede serem distribuídos em camadas (*layers*). Os sistemas comunicam entre si, existe controlo de erros e sincronização de processos. Na comunicação são necessárias regras que definem o que fazer em determinada situação e procedimentos que determinam o encaminhamento da informação. A comunicação envolve regras (sintácticas, semânticas, temporais, etc.) que são designadas de protocolos. Um protocolo regula a comunicação entre entidades homólogas em sistemas diferentes. Os protocolos estruturam-se igualmente em camadas, em que as camadas superiores encapsulam as inferiores. Uma interface regula a comunicação entre entidades em camadas adjacentes num mesmo sistema. Utilizando serviços fornecidos pela camada inferior, entidades homólogas cooperam, por meio de um protocolo, na realização de certas funções, o que lhes permite oferecer, através duma interface, um serviço de valor acrescentado à camada superior.

Devido à actual indefinição da arquitectura de gestão das redes NGN, o presente trabalho propõe uma solução de gestão de plataforma *IP Multimédia subSystem* (IMS) baseada em políticas, para a definição de regras de admissão. A solução de gestão proposta consiste na integração de tuas tecnologias de gestão *Web-Based Enterprise Management* (WBEM) e NETCONF, para o servidor de gestão e configuração dos equipamentos de rede IMS respectivamente. O WBEM não é uma tecnologia recente, no entanto apresenta um modelo de informação actualizado, muito versátil e poderoso, tratando-se de um protocolo bem disseminado com a existência de varias implementações *open-source*. Por seu lado o NETCONF é um protocolo bastante recente, em fase de aceitação, e que pretende colmatar as actuais lacunas do *Simple Network Management Protocol* (SNMP) ao nível da segurança e escalabilidade.

1.2 Objectivos

Com este trabalho pretende-se apresentar uma solução de gestão baseada em políticas, para gestão dos equipamentos de rede IMS. A arquitectura da solução consiste na interligação de um servidor de gestão, concretamente um servidor *Common Information Model* (CIM), com os equipamentos de rede IMS recorrendo a uma implementação *open-source* do protocolo NETCONF.

Depois de implementada a solução, pretende-se averiguar o desempenho do protocolo de gestão NETCONF, no que diz respeito à sinalização, uma vez que que se trata de uma arquitectura distribuída, assim como os requisitos em termos de consumo memória por parte do vários módulos que compõem a solução.

Com base nos resultados obtidos, decorrentes da avaliação de desempenho, pretende-se validar a aplicabilidade desta tecnologia para um cenário de gestão de uma rede de operador de telecomunicações.

1.3 Organização do Documento

O presente capítulo e primeiro, contextualiza o âmbito do projecto, introduzindo os conceitos, estabelecendo o enquadramento e definindo os objectos a que o trabalho se propõe.

O segundo capítulo analisa o estado da arte, descrevendo: o paradigma de gestão de redes baseado em políticas e os vários componentes que integram a arquitectura PBNM; as tecnologias de gestão de redes relevantes para a execução deste trabalho, identificando para cada uma delas, o modelo de dados, a codificação da informação e o transporte da informação.

O terceiro capítulo apresenta o IMS como arquitectura de referência para as redes de próxima geração, focando sua génese e descrevendo a sua arquitectura e os respectivos elementos que a constituem.

O quarto capítulo descreve em detalhe o sistema desenvolvido; são identificados os requisitos, é definido o modelo de informação de acordo com requisitos, são apresentadas as implementações *open-source* das tecnologias de gestão usadas e as ferramentas de suporte ao desenvolvimento; e é descrita a arquitectura da solução bem como os seu componentes.

No quinto capítulo é efectuada a análise da resposta do sistema resultante dos testes efectuados e é validado o sistema de acordo com os requisitos propostos. Neste capítulo é também analisado o comportamento dinâmico do sistema nas componentes de *profiling* e testes de carga.

Finalmente, no sexto capítulo são tiradas conclusões e é apresentado o trabalho futuro.

Capítulo 2

Tecnologias de gestão de redes

2.1 Introdução

Até há bem pouco tempo, a gestão de redes tem sido bastante centralizada nos equipamentos específicos, digamos que de forma elementar, actuando directamente sobre estes equipamentos de uma forma desintegrada do resto da infraestrutura, aplicando configurações especificas e obtendo do equipamento relatórios e estatísticas.

Esta abordagem carece obviamente de integridade com o resto da estrutura, além de inevitavelmente encarecer o preço de manutenção global da infraestrutura. A necessidade de pessoal cada vez mais especializado para fazer a manutenção destes equipamentos é também algo a ter em conta.

Existe contudo, ainda alguma falta de uniformização em termos protocolares. Enquanto que determinados fabricantes privilegiam o suporte nativo a SNMP, outros ou não o fazem, ou enveredam por soluções proprietárias, incompatíveis com com os *standards*. O saldo final de tudo isto não é de modo algum positivo, obrigando que para garantir determinados níveis de serviço, seja necessário recorrer a um portfólio de ferramentas e consolas, dependendo é claro da complexidade da rede em causa.

Juntamente com os problemas de gestão propriamente ditos, emergentes da complexidade crescente das redes, a evolução dos mercados em termos de comunicações de voz e de dados (novos serviços, novos requisitos de qualidade de serviço e necessidade de fornecimento de garantias), assim como o aumento brutal da concorrência, têm demonstrado a vantagem que seria em possuir um modelo de gestão que permitisse gerir todo o equipamento de uma forma simples e integrado com as necessidades do negócio.

A PBNM veio colmatar esta necessidade, tornando-se por isso bastante aliciante, uma vez que os sistemas PBNM permitem que a configuração da rede seja executada de forma autónoma e coordenada de todos os equipamentos da rede, sem o factor de erro associado à intervenção humana. As vantagens são evidentes quer para empresas, quer para fornecedores de serviços. Como resultado, o PBNM tem tido nos últimos anos um grande alento em virtude de conceptualmente apresentar um grande potencial de aplicação. No entanto, são ainda poucas as implementações concretas que materializam esta abordagem, tirando dela partido, como por exemplo o CiscoWorks QoS Policy Manager (QPM) [6] ou o Extreme Networks Policy Manager (EPM) [16].

O *Internet Engineering Task Force* (IETF) e o *Distributed Management Task Force* (DMTF) defendem principalmente um desacoplamento entre a gestão e as especificades das tecnologias

subjacentes. Assim, o interesse em uniformizar a gestão destes recursos, é claramente importante e permite definir a um nível de abstracção superior, onde se encontram as políticas de gestão da rede e as definições concretas dos serviços pretendidos, ocultando o detalhe técnico prescindível.

A convergência das redes de comunicações tradicionais (ISDN/PSTN/PLMN) com as redes *Internet Protocol* (IP) tem sido objecto de trabalhos de normalização em várias organizações internacionais, como o grupo *Third Generation Partnership Project* (3GPP), o *European Telecommunications Standards Institute* (ETSI) e o *International Telecommunication Union* (ITU), com o objectivo de diminuir progressivamente a proliferação de redes dedicadas a serviços específicos, por um lado, e promover a convergência ao nível aplicacional e da infraestrutura, por outro. Desta convergência resulta o conceito de redes de nova geração que pode ser definido como redes de comutação de pacotes, para prestação de serviços de voz, de dados e multimédia, recorrendo ao uso de múltiplas tecnologias (fixas e móveis) de transporte e permitindo uma gestão centralizada de diferentes níveis de qualidade de serviço, independentes das tecnologias de transporte. Estas redes permitem livre acesso por parte dos utilizadores, às redes concorrentes e aos serviços multimédia, suportando assim mobilidade generalizada.

Nas redes de próxima geração, as entidades que controlam as políticas, as sessões, os serviços multimédia, os recursos, entrega de serviços e a segurança estão distribuídas pela infraestrutura de rede. A interoperabilidade entre estas entidades está definida por interfaces normalizadas. O PBNM poderá ser uma boa opção para a gestão deste tipo de redes, pois suporta um vasto leque de serviços distribuídos e esconde de forma transparente os detalhes técnicos da implementação assim como reduz o tempo de integração. Por esta razões, as principais arquitecturas de redes de nova geração, 3GPP, *Telecommunications and Internet converged Services and Protocols for Advanced Networking* (TISPAN) ou *Telecommunication Standardization Sector* (ITU-T) baseiam os mecanismos de controlo de admissão no paradigma de gestão baseada em políticas, utilizando esquemas de regras para definir critérios de admissão de novos fluxos [70].

Conceptualmente, o paradigma *Policy Based Management* (PBM) constitui assim, a resposta promissora ao novos desafios da gestão, que acompanham a evolução das redes, rumo a próxima geração. O sistema de gestão terá de interpretar as políticas e mapeá-las na semântica e sintaxe específica das linguagens dos equipamentos e sistemas. Esta tarefa é tanto mais fácil, quanto maior for a adesão por parte dos fabricantes de equipamento em adoptar protocolos e modelos de dados normalizados. É importantíssimo o papel dos organismos de estandarização, com o desenvolvimento de novas técnicas e modelos de gestão.

O SNMP é um dos protocolos mais conhecido e utilizados pela comunidade académica, organizações e ambientes corporativos. Como o próprio nome sugere, a arquitectura e a forma de aplicação deste protocolo são bastante simples e podem ser implementadas em hardware. Esses são alguns dos motivos que contribuíram para a sua ampla aceitação e aplicação no mercado de redes de computadores. Contudo, até sua versão actual, este protocolo ainda apresenta algumas deficiências e problemas que motivaram, inclusive, o desenvolvimento de novos protocolos e ferramentas de gestão como o *Network ConFiguration* (NETCONF). Entre as maiores deficiências do SNMP, destacam-se a segurança e a dificuldade de aplicação em alguns ambientes complexos (com numerosos e/ou diversos sistemas e componentes de rede) e que exigem suportes de gestão mais dinâmicos e flexíveis.

2.2 Gestão de redes baseada em políticas

Limitando a definição de política a um contexto tecnológico, uma política é um conjunto de regras e instruções que determinam o funcionamento de uma infraestrutura. Mesmo assim a abrangência desta definição é vasta, dependendo da dimensão do sistema em causa, as políticas podem estar agrupadas funcionalmente por áreas mais específicas, ou geridas por vários administradores, cada um com a sua responsabilidade.

A Gestão de Redes Baseada em políticas é acima de tudo uma mudança de paradigma relativamente ao conceitos de gestão tradicionais. Tal como descrito anteriormente, em vez do focar o controlo nos equipamentos, foca-se na definição de regras que permitem a gestão dos equipamentos, permitindo a integração de vários modelo e fabricantes. Baseia-se principalmente na gestão dos utilizadores e serviços. Conceptualmente um sistema PBNM consiste num conjunto de entidades distribuídas pela rede que dinamicamente fazem associações e mapeiam os utilizadores aos equipamentos e serviços, de acordo com as políticas definidas, tornando transparente ao administrador da rede, o detalhe técnico desses mapeamentos, nomeadamente as configurações.

O recurso a este tipo de gestão tem sido adoptado de forma abrangente, principalmente em domínios como QoS e segurança. Os exemplos típicos de aplicação incluem:

- configuração de dispositivos, mecanismos de gestão de filas de espera, estratégias de descarte e listas de acesso;
- classificação de tráfego;
- reserva de recursos e controlo de admissão baseados em parâmetros, como identidade dos utilizadores e tipo de aplicação;
- estabelecimento de *Service level agreement* (SLA) entre domínios adjacentes.

O modelo de gestão PBM não é apenas uma substituição aos modelos tradicionais, mas sim uma importante extensão aos métodos de gestão existentes. Esta nova abordagem aos princípios da gestão apresenta algumas vantagens importantes, entre elas destacam-se por exemplo as seguintes:

- permite ao administrador de rede mapear recursos da rede de acordo com as necessidades de negócio, assegurando perfis de qualidade de serviço para processos críticos de negócio;
- simplifica, de uma forma geral, a gestão da rede tornando os operadores mais produtivos ao simplificar a gestão e configuração dos equipamentos de rede, assim como a implementação de novos serviços;
- ao permitir um controlo central relativamente à segurança, prioridades de tráfego, acessos, etc, garante-se que todas estas políticas são aplicadas de uma forma eficiente por todo o domínio de gestão, mantendo a rede mais consistente e sob controlo.

Os objectivos são nobres, e evidenciam a importância do esforço de normalização, compatibilização e mapeamento entre toda a gama de serviços de informação e equipamentos existentes no mercado.

2.2.1 Requisitos para PBNM

Os quatro requisitos básicos que devem ser satisfeitos para uma completa implementação de PBNM [57] são:

i) Um modelo de informação extensível a toda a estrutura e topologia da organização a gerir. Este modelo deverá incluir, por exemplo, os equipamentos, serviços, aplicações, utilizadores e a forma como se relacionam os objectos entre si.

ii) Uma linguagem de especificação para a representação das necessidades de negócio e funções, com alguma abstracção, de modo que garanta a total independência das questões e sintaxes específicas dos equipamentos e das soluções proprietárias dos diferentes fornecedores. Esta questão está longe de ser simples e implica métodos de compatibilização entre distintas formas de armazenamento e relacionamento da informação, por exemplo, entre serviços de directório ou bases de dados diferentes.

iii) Uma plataforma de gestão completa para administração e controlo, que permita a resolução de conflitos e distribuição, onde se inclui a definição da arquitectura, com um peso importante em toda a escalabilidade e consistência do modelo. Este é provavelmente um requisito crucial para o sucesso do modelo, e nesse sentido muito desenvolvimento está a ser feito com relação a protocolos de comunicação intra domínios e entre as diversas entidades do sistema de políticas baseadas em *Common Open Policy Service* (COPS) [41], *Next Steps In Signaling* (NSIS) [53], *eXtensible Markup Language* (XML), etc. A comunicação entre domínios PBNM diferentes é também outra funcionalidade importante, nomeadamente no que respeita a toda a interoperacionalidade entre eles, partilha de repositórios, extensão de políticas de segurança e relações de confiança entre eles.

iv) Uma forma escalável de transposição ou tradução das especificações de alto nível independentes dos equipamentos, para configurações dependentes e específicas dos equipamentos. Protocolos como SNMP, COPS, NETCONF ou até *Command Line Interface* (CLI), podem ser uma solução interessante para distribuir as políticas pelos diferentes equipamentos na rede. No entanto, isto implica que cada fabricante implemente funcionalidades de tradução das políticas recebidas, para a semântica e sintaxe específica das linguagens dos seus equipamentos e sistemas. Este ponto é obviamente de progressão mais lenta e depende do envolvimento de muitas entidades e da adopção do standard por parte delas.

2.2.2 A Estrutura

O bloco mais elementar de uma política é uma regra (*Policy Rule*). As regras podem conter uma ou mais condições e uma ou mais acções. As condições filtram os eventos em que as acções definidas devem ser aplicadas. No exemplo da Figura 2.1, a condição está entre o *if* e o *then*, e a acção entre o *then* e o *endif*.

O exemplo apresentado é bastante simples e serve apenas como ilustração. Fica, no entanto, a noção de que as combinações possíveis na construção das condições são bastante extensas, potenciando uma gestão funcionalmente apreciável. Além disso, as regras podem ser combinadas dando origem a novas regras, num processo de agregação de regras mais complexas. As regras simples contém um conjunto de condições e acções, mais ou menos extenso,

```
if (((trafficToOrFrom AccountingSubnet) and
  (dayOfMonth is last10days))
then
  priority = hight;
endif
```

Figura 2.1: Exemplos genéricos de políticas.

como ilustrado anteriormente. Exemplos de políticas complexas podem incluir reconfigurações dinâmicas da rede, baseadas em determinados eventos. As utilizações são infinitas. As políticas podem apresentar ainda outra propriedade importante: a herança, facilitando a construção de políticas mais complexas a partir de outras já criadas e das definições herdadas. A Figura 2.2 ilustra a árvore hierárquica das políticas de acordo com o [64] [63].

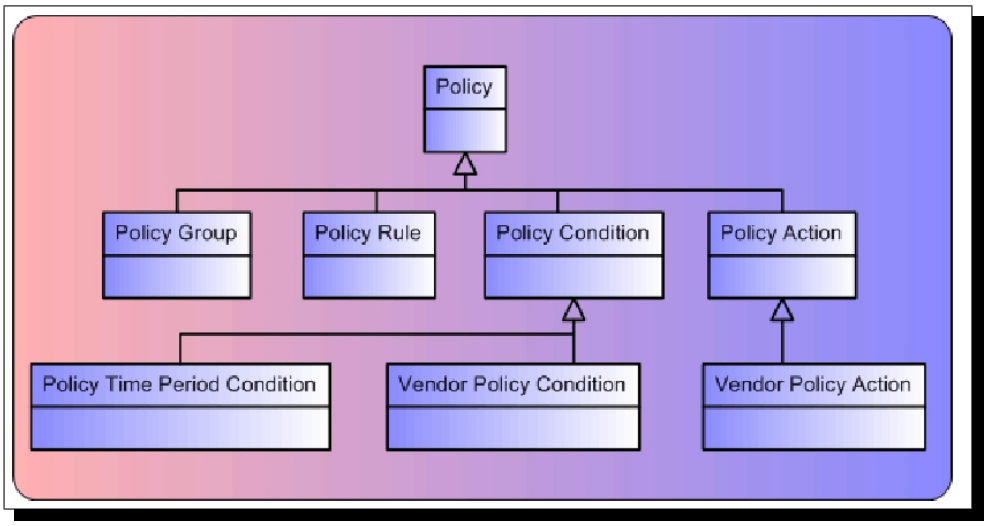

Figura 2.2: Estrutura hereditária das políticas.

A reutilização e a simplificação da criação são duas fortes vantagens, pela estrutura hierárquica existente. Além disso, a actualização do repositório torna-se bem mais simples. Vejamos o seguinte exemplo da Figura 2.3.

Se por motivos de reorganização, a rede DMZ for alterada para outra gama de endereços, a actualização de todas as políticas que lhe fizessem referência seria incontornável. No entanto, desta forma simples, bastará actualizar uma propriedade, para que todas as políticas que fazem referência à rede DMZ sejam actualizadas.

```
Subnet_DMZ = 192.168.1.0/24
...
if (Subnet_DMZ contains MACaddress)
then ...
```

Figura 2.3: Exemplo de reutilização.

Outra das características interessantes, presente em algumas implementações de políticas, no que respeita à utilização de políticas, são as *roles*. O conceito de *role* é bastante

importante por permitir agregar grupos de objectos aos quais são aplicadas as políticas, minimizando desta forma o trabalho necessário de aplicação de políticas, evitando a replicação e aumentando a organização e controlo.

É possível, por exemplo, agregar todas as interfaces de um determinado tipo e aplicar de uma só vez a esse grupo, as políticas pretendidas, em vez de as aplicar repetidamente às várias interfaces. Outra das vantagens que advêm da sua utilização é a diminuição de algum tráfego em comunicações, uma vez que a mesma política pode ser aplicada a um grupo de componentes.

2.2.3 Arquitectura Subjacente

A Figura 2.4 ilustra uma arquitectura alto nível, típica de um sistema PBNM proposta pelo IETF [64].

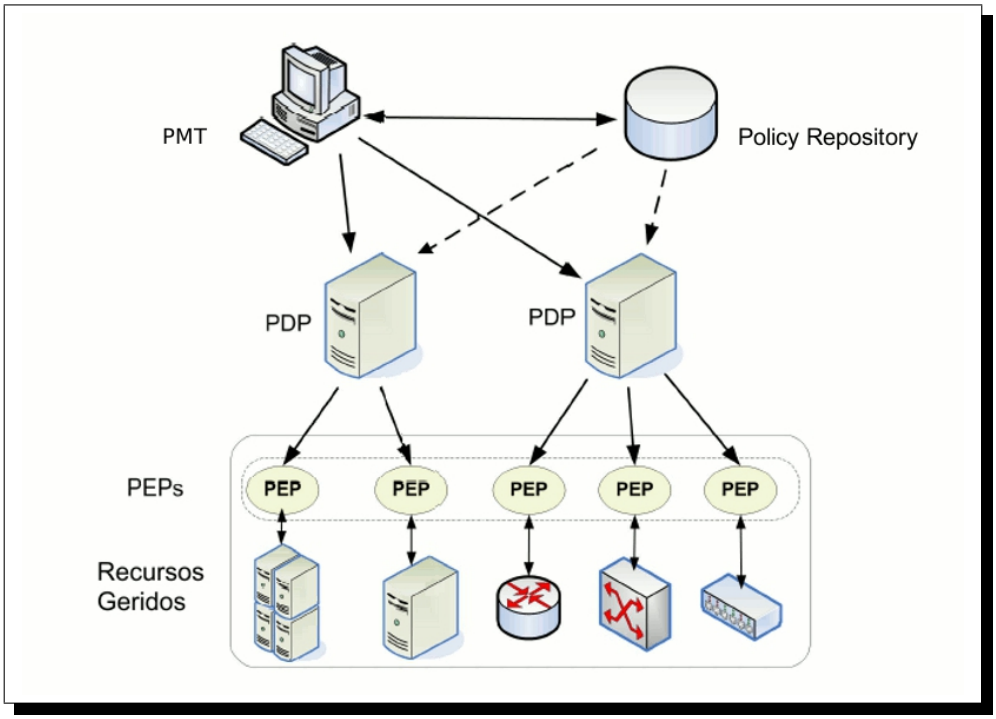

Figura 2.4: Arquitectura PBNM proposta pelo IETF.

De forma simplificada, a arquitectura proposta pelo IETF, que tem sido a implementação mais utilizado nas abordagens actuais, especialmente no domínio de gestão de QoS, é composta por quatro elementos principais: o *Policy Management Tool* (PMT), *Policy Repository* (PR), *Policy Definition Point* (PDP) e o *Policy Enforcement Point* (PEP).

O PMT fornece uma interface do sistema de gestão ao administrador de rede, de forma a permitir que este crie ou edite as políticas desejadas de forma gráfica ou utilizando uma linguagem padrão. Esta ferramenta é responsável também por converter, se necessário, a forma de representação da política para o formato aceite pelo repositório de políticas e também obter uma política do repositório para edição. Este elemento deve ainda, verificar a validade da política inserida, assim como, opcionalmente, verificar possíveis conflitos entre a nova política e as políticas existentes. Para que uma política seja aplicada, a ferramenta deve sinalizar

o PDP, indicando que este deve consultar, junto ao repositório de políticas, as informações referentes à política em questão.

O PR é o elemento do sistema responsável pelo armazenamento das políticas, das informações dos recursos e dos utilizadores. Os repositórios em PBNM podem ser baseados em bases de dados relacionais ou em directórios (*Lightweight Directory Access Protocol* (LDAP) por exemplo).

O PDP é o elemento do sistema responsável pela gestão de políticas, podendo também ser chamado de servidor de políticas. O PDP analisa os eventos enviados pelos PEP controlados por ele, de acordo com o conjunto de regras armazenadas no repositório de políticas. A partir do evento recebido, o PDP reavalia as políticas adoptadas e envia as acções a serem tomadas pelos PEP. Esta comunicação é feita recorrendo um protocolo normalizado entre PDP e PEP, tal como o COPS [41], *Common Open Policy Service for Policy Provisioning* (COPS-PR) [44], Command Line Interface/Telnet, SNMP, ou até NETCONF e *Simple Object Access Protocol* (SOAP).

O PEP é o elemento no qual a política é aplicada, podendo residir tipicamente no dispositivo gerido. Com base na informação fornecida pelo PDP, o PEP irá configurar o dispositivo gerido, concretizando as acções da política, tal como filtragem de pacotes, reserva de banda, prioridade de tráfego e múltiplas filas de saída. As implementação actuais requerem este conhecimento detalhado dos elementos para determinar a possibilidade de implantação da política.

Segundo [67], existem dois modelos principais utilizados no gestão baseado em políticas: *outsourcing* e *provisioning*. No modelo *outsourcing*, quando há ocorrência de um evento no PEP, este delega ao PDP a tarefa de tomar uma decisão de que acção executar (e se deve ser executada alguma acção) na ocorrência deste evento. Por exemplo, este é um caso típico de controlo de admissão gerado a partir de um pedido *Resource ReSerVation Protocol* (RSVP) [40] por reserva de recursos. O PDP decide se a reserva deve ser efectuada ou não, baseado nas políticas relativas aos objectivos corporativos localizados no repositório, contendo a informação de que tipo de utilizador/aplicação possui o privilégio de reserva de recursos.

Já no modelo *provisioning*, na medida em que novas políticas são inseridas no editor de políticas, as suas regras vão sendo distribuídas em tempo real ao PDP. A partir daí, o PDP decide se a política deve ser aplicada utilizando um conjunto de critérios, tais como se as condições temporais são satisfeitas ou se os elementos controlados possuem características de software/hardware que permitam a aplicação da política. Se todas as características forem satisfeitas, o PDP decompõe as regras da política em comandos de configuração a serem interpretados pelos dispositivos e envia as instruções a estes. Todas as decisões enviadas pelo PDP são acompanhadas de mensagens de confirmação para garantir que as instruções foram devidamente instaladas ou para detectar um erro na instalação. Esta abordagem é mais comum ser aplicada no controlo de redes que utilizam protocolos sem sinalização, tais como *DiffServ*, ou para a configuração de dispositivos - tais como *Virtual Private Network* (VPN) e glsvoip. O protocolo COPS-PR [44] implemente este modelo de aplicação de políticas.

2.2.4 Linguagens de definição de políticas

A tarefa de converter políticas abstractas em políticas de serviços específicas para, então, aplicá-las em dispositivos que oferecem suporte a tais aplicações não é simples. De entre as várias linguagens e ferramentas em desenvolvimento e já existentes, serão descritas a seguir a *Policy Definition Language* (PDL), Ponder e o *CIM Simplified Policy Language* (CIM-SPL).

A primeira será abordada por se tratar de uma linguagem a partir da qual, diversas outras se basearam, aproveitando seus conceitos. Já a linguagem Ponder será descrita principalmente por ser uma das mais completas, embora descontinuada. Por fim o CIM-SPL, será descrito, pois é relativamente recente.

PDL

A PDL [60] é uma linguagem baseada em eventos, originada no departamento de *network computing* da Bell-Labs. políticas em PDL usam o paradigma de regra *Event-Condition-Action* (ECA), ou seja, define-se uma política como uma função que mapeia uma série de eventos em uma série de acções. Para isso, a linguagem tem semânticas claramente definidas e uma arquitectura especifica para aplicar políticas PDL. Esta linguagem não suporta políticas de controlo de acesso, assim como também não permite a composição de regras de políticas em *roles*, ou outras estruturas de agrupamento [46]. Na Figura 2.5 encontra-se um exemplo de uma política PDL.

```
1 // policy for product order processing
2 defectiveProduct causes stop
3 orderReceived causes mailProduct
4 orderReceived causes chargeCreditCard
5
6 /* policy for Soft-switch overload control in the presence of an excessive number
      of signalling network time outs */.
7 (normalmode, group(callmade|timeout)) triggers overloadmode if (Count(timeout)/
     Count(callmade)>ThOvld)
```

Figura 2.5: Exemplo de política em PDL.

Ponder

O Ponder [47] foi desenvolvida pelo *Policy Research Group* do Departamento de Engenharia de Software Distribuído do Imperial College, Londres. É uma linguagem declarativa, orientada a objectos, utilizada para a especificação de políticas de segurança para sistemas distribuídos. A linguagem é flexível, expressiva e extensível para cobrir uma grande quantidade de requisitos exigidos pelos sistemas distribuídos da actualidade. Contudo, conforme consta em sua página oficial, o projecto foi descontinuado. No entanto, foi uma das primeiras linguagens a ser relativamente funcional.

Segundo [46], os principais tipos de políticas suportados pelo Ponder são:

- Políticas de controlo de acesso: limitam a actividade de utilizadores legítimos que tenham sido autenticados com sucesso através de políticas de autorização, delegação, filtragem de informações e proibição;
- Políticas de obrigação: especificam as acções que devem ser realizadas pelos gestores do sistema quando certos eventos ocorrerem e fornece a habilidade de responder a circunstâncias mutáveis.

Na Figura 2.6, pode-se visualizar um exemplo de política em Ponder. A política de grupo loginGroup autoriza o grupo a aceder computadores no domínio research, armazena em log as tentativas de login, carrega o ambiente de utilizador e lida com falhas de *login*.

```
inst group loginGroup {
  inst auth+ staffLoginAuth {
    subject  /dept/users/staff ;
    target /dept/computers/research;
    action login;
  }
  inst oblig loginactions {
    subject  s = /dept/users/staff;
    on loginevent (userid, computerid);
    target t = computerid ^ {/dept/computers/}
    do s.log (userid, computerid) -> t.loadenviroment (userid);
  }
  inst oblig loginFailure {
    on 3*loginfail(userid);
    subject s = /NRegion/SecAdmin;
    target <userT> t = /NRegion/users ^(userid);
    do t.disable() -> s.log(userid);
  }
}
```

Figura 2.6: Exemplo de política em Ponder.

CIM-SPL

O elevado nível de detalhe do modelo CIM torna a definição de políticas pouco intuitiva. A fim de simplificar a utilização dos modelos de políticas CIM, o CIM-SPL foi criado por Jorge Lobo dentro do DMTF *Policy Working Group* em Janeiro de 2007 [59], tornando-se uma norma proposta.

O objectivo do CIM-SPL é proporcionar um meio para especificar regras baseadas em *if-condition-then-action* para a gestão de recursos computacionais utilizando elementos definido pelo CIM. O desenvolvimento do CIM-SPL foi inspirado em linguagens de definição de políticas e modelos existentes, como o PDL, Ponder e o *Autonomic Computing Policy Language* (ACPL) [74] da IBM Research.

Um dos aspectos fundamentais do projecto CIM-SPL é o fornecimento de uma linguagem de definição de políticas compatível com o modelo de políticas CIM e o esquema CIM.

A Figura 2.7 mostra a especificação CIM-SPL de uma política que é invocado quando a memória de um sistema de ficheiros esta 85 por cento ocupada. A política amplia a *pool* de armazenamento em 25 por cento.

```
Import CIM_X_XX_XXXX::CIM_LocalFileSystem;
Strategy Execute_All_Applicable;
Policy {
  Declaration {
    computer_system = collect(Self, CIM_HostedFileSystem, PartComponent,
        GroupComponent, null, true)[0];
    storage_config_service = collect(computer_system, CIM_HostedService,
        Antecedent, Dependent,
    CIM_StorageConfigurationService, true)[0];
    logical_disk = collect(Self, CIM_ResidesOnExtent, Dependent, Antecedent, null
        , true)[0];
    storage_pool = collect(logical_disk, CIM_AllocatedFromStoragePool, Dependent,
         Antecedent,
    null, true)[0];
    fs_goal = collect(Self, CIM_ElementSettingData, ManagedElement, SettingData,
    CIM_FileSystemSetting true)[0]; }
  Condition { (AvailableSpace / FileSystemSize) < 0.25 }
  Decision {
    storage_conf_service.CreateOrModifyElementFromStoragePool("LogicalDisk",
    /* ElementType Volume */
    8, job, fs_goal,
    1.25 * FileSystemSize,
    storage_pool,
    logical_disk)
     | DoSomethingOnFailure() }
 }:1 /* policy priority*/;
```

Figura 2.7: Exemplo de política em CIM-SPL.

2.2.5 Distribuição

A arquitectura PBNM pode ser distribuída vulgarmente em duas soluções diferentes: *2-tier* ou *3-tier*. As soluções são distintas, dependendo se o PDP e o PEP coexistem na mesma entidade ou não. Os dois modelos apontam para a concentração na primeira camada, PMT e PR. Na versão *2-tier*, a segunda camada inclui o PDP e o PEP, os quais estão separados cada um em diferentes *tier's* na versão *3-tier*. Esta última tem tido maior aceitação, pelo facto de se poder incluir nela a integração de equipamentos tradicionais, que não suportem o controlo via políticas. Isto é conseguido com a inclusão do *Policy Proxie* (PP), tal como ilustrado na Figuras 2.8.

Outra limitação da versão *2-tier* diz respeito ao número de equipamentos preparados para suportar esta arquitectura, pois o equipamento tem necessidade de assumir as funções de PDP e PEP, o que não existe nos equipamentos actuais. Por outro lado, integrar desde já alguns equipamentos via SNMP, CLI ou NETCONF através dos *proxies* parece uma solução preferível de início. Embora, como é óbvio, existam vantagens em ter um ambiente homogéneo a operar com equipamentos compatíveis PBNM. No entanto, é bastante difícil que aconteça e as soluções com o *proxy* são uma solução a ter em conta. Vejamos por exemplo algumas vantagens do uso de equipamentos PBNM.

Contrariamente ao que acontece com SNMP onde tem de haver algoritmos de sincronização de conteúdo e de controlo de concorrência, dada a possibilidade de existirem vários gestores para os mesmos agentes. Em PBNM o uso de COPS confere à arquitectura uma leveza importante por associar a cada PEP um e um só PDP, e por essa razão os algoritmos de

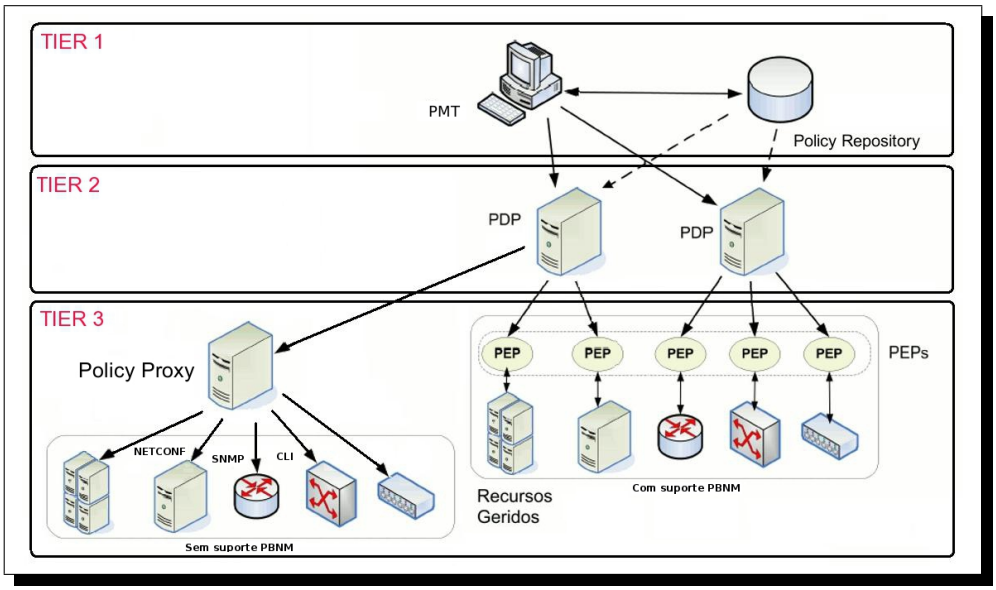

Figura 2.8: Distribuição em PBNM no modelo *3-tier*.

controlo, quer no PEP, quer no PDP são manifestamente mais simples e eficientes [57], embora haja obviamente capacidade para referenciar PDP alternativos como medida de protecção contra falhas.

Outra grande vantagem justifica o uso dos *roles*, com os quais pode ser diminuído o tráfego na rede dependendo da arquitectura implementada, uma vez que uma só política pode ser aplicada a um grupo de objectos. O uso de COPS controla de forma mais eficiente a arquitectura do sistema e a forma como funciona pois através deste protocolo, os PEP têm capacidade para informar o PDP da sua capacidade e funções, o que com outros dispositivos tradicionais não acontece. Do mesmo modo, o PDP tem um protocolo nativo para enviar as políticas para o PEP.

2.3 WBEM

O WBEM [13] corresponde a uma iniciativa de várias empresas que surgiu em 1996 por um grupo de empresas liderado pela Microsoft, Compaq Computer, BMC Software, Cisco Systems e Intel. A motivação foi a definição de um ambiente aberto para a gestão, onde todos os sistemas de gestão e aplicação pudessem aceder, controlar e partilhar informações de gestão uns com os outros, assim como a um qualquer agente de gestão de um dispositivo a gerir, utilizando a tecnologia existente e recorrendo aos *standards* tanto quanto possível. Em muitos aspectos, as pretensões do consorcio reflectiam os avanços tecnológicos da *World Wide Web* (WWW), onde, pela primeira vez, os dispositivos ligados a Internet podiam actuar como fontes e consumidores de informação, sem qualquer conhecimento dos contextos específicos em que cada componente é gerido. Devido a esta visão partilhada, juntamente com a possibilidade de utilização de tecnologias baseadas na *Web*, como complemento de instrumentos de gestão mais convencionais para criar um ambiente de gestão aberta, o nome da iniciativa da *Web* tornou-se *Web Based Enterprise Management* (WBEM).

O WBEM consiste num conjunto de tecnologias de gestão e *Internet* normalizadas e desenvolvidas para unificar a gestão de ambientes de computação empresariais.

A arquitectura WBEM foi projectada para:

- proporcionar o desenvolvimento de padrões industriais que permitissem aos administradores recorrer a qualquer *Web Browser* para proceder à gestão de redes, sistemas, aplicações e serviços;
- permitir que soluções de gestão fossem construídas cobrindo as áreas funcionais da gestão, como: a gestão de configuração, gestão de falhas, gestão de contabilização, gestão de desempenho e gestão de segurança;
- fornecer um modelo de dados que permitisse uma modelação uniforme do ambiente a ser gerido;
- atender às necessidades de um grande conjunto distribuído de elementos de gestão, fornecendo uma solução flexível e escalável.

Durante o decorrer do desenvolvimento da arquitectura WBEM, um protocolo de gestão chamado *HyperMedia Management Protocol* (HMMP) foi especificado. Contrariamente às intenções originais, o HMMP não era baseado em *HyperText Transfer Protocol* (HTTP), mas em outros protocolos de transporte, tendo muito pouco em comum com o HTTP. Com as dificuldades em desenvolver uma arquitectura de gestão ao nível pretendido, o DMTF [17] ficou com a responsabilidade de liderar o desenvolvimento da arquitectura WBEM, até que Junho de 1998, o DMTF anunciou que aceitava a transferência da iniciativa WBEM por parte das restantes empresas fundadoras. O conceito principal do WBEM, a unificação, manteve-se. Contudo, houve outros aspectos da iniciativa que foram alterados, nomeadamente a representação do modelo de informação e o protocolo usado para a troca de informação. Com os trabalhos do DMTF, o objectivo original de recorrer ao HTTP como protocolo de gestão foi reestabelecido, sendo a proposta do HMMP suspensa.

O DMTF desenvolveu um protótipo de um conjunto de especificações, independentes do ambiente de gestão, de como descrever e aceder a qualquer tipo de tecnologias de gestão,

incluindo normas tais como SNMP [28] e o *Desktop Management Information* (DMI) [9]. O elemento central desta especificação consiste num mecanismo de descrição de dados que mais tarde se tornou o padrão do DMTF conhecido como o CIM [8].

O DMTF constitui a partir daí o ponto central para os esforços de iniciativa WBEM, fornecendo uma estrutura organizacional para uma participação mais ampla da indústria no desenvolvimento de tecnologias WBEM e respectivas normas compatíveis. Sendo detentor da especificação CIM e CIM *Schema*, posicionou-as como padrões no sector para o acesso e partilha de dados de gestão de rede.

A Tabela 2.1 fornece um resumo da história do WBEM com as datas da disponibilização das versões mais importantes da norma.

Tabela 2.1: História do WBEM.

Ano	**Data**	**Evento**
1996	7/17	WBEM é anunciado
1996	9/18	DMTF adopta o CIM
1997	4/9	CIM V1 é publicada
1997	12/11	CIM V2.0a é publicada
1998	6/2	DMTF adopta o WBEM
2003	12/15	Especificação CIM v1.1 é publicada
2004	12/9	Especificação de operações CIM sobre HTTP v1.2 é publicada (DSP0200)
2004	12/9	Especificação da representação CIM em XML v2.2 é publicada
2005	10/4	Especificação da infraestrutura CIM v2.3 é publicada
2007	11/11	Especificação da infraestrutura CIM v2.4 é publicada
2009	5/1	Especificação da infraestrutura CIM v2.5 é publicada

Partindo inicialmente do projecto *HyperMedia Management Schema* (HMMS), a especificação CIM descreve a linguagem de modelação, a nomenclatura e técnicas de mapeamento usadas para recolher e transferir informações de fornecedores de dados e modelos de gestão. O esquema CIM fornece as descrições do modelo real e o enquadramento da informação. Este define um conjunto de classes com propriedades e associações, tornando possível a organização de toda a informação sobre o ambiente gerido.

Um ambiente de gestão pressupõe a existência de três elementos:

i) uma descrição dos dados;

ii) codificação dos dados para o seu transporte;

iii) conjunto de operações para manipular esses dados (acções de *get, set, action, event*).

Estes três aspectos, em conjunto com algum mecanismo de identificação que permita que os gestores e agentes tenham conhecimento de quem comunica com quem, são os requisi-

tos essenciais para que um ambiente de gestão seja operacional. A qualidade, e de alguma forma, a quantidade dos dados de gestão, determinam quão compreensivo ou sofisticado é esse ambiente.

Depois de se ter tornado detentor do WBEM, o DMTF redefiniu alguns dos seus conceitos iniciais de modo a promover o uso de tecnologias *Web* que fossem acessíveis e estivessem bem divulgadas. O HMMS corresponde actualmente ao CIM, o *Hypermedia Object Manager* (HMOM) corresponde actualmente ao *Common Information Model Object Manager* (CIMOM) e o HMMP corresponde actualmente ao CIM/XML sobre HTTP.

Os novos elementos de gestão são:

- CIM – Corresponde a um modelo orientado a objectos para a descrição de informações de gestão;
- *Managed Object Format* (MOF) – Corresponde a uma especificação sintáctica para objectos a gerir modelados através do CIM. A notação MOF é uma linguagem *template*, comparável ao *Guidelines for the Definition of Managed Objects* (GDMO) ou a *Structure of Management Information* (SMI) da Internet;
- Uma especificação para mapeamento do CIM em XML, uma linguagem escrita em *Document Type Definition* (DTD) que pode ser usada para representar classes e instâncias do CIM.

Tal como ilustra a Figura 2.9, foi definida uma tradução (*mapping*) do CIM em XML, o que permite a representação da informação de gestão numa forma textual e susceptível de ser trocada assim como a definição de um mecanismo de transporte de operações CIM sobre o protocolo HTTP.

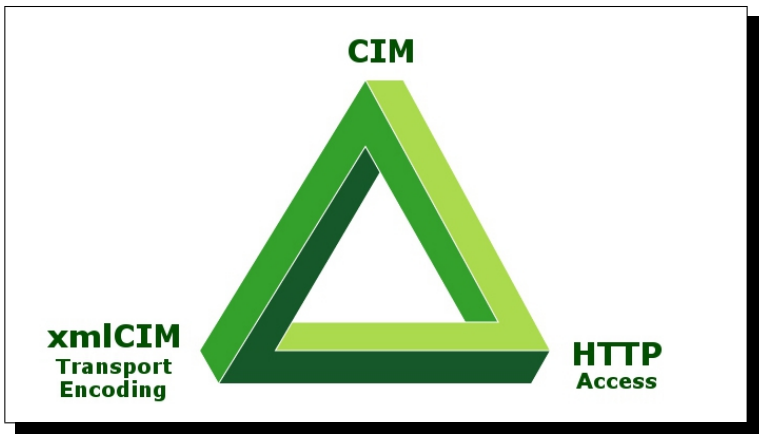

Figura 2.9: Aspectos de gestão definidos pelo WBEM.

Através do mapeamento do CIM em XML, qualquer *browser* com implementação da linguagem XML pode ser usado para mostrar e capturar informações de gestão. Nenhum protocolo de gestão especializado é necessário nesse contexto.

O ambiente de gestão WBEM resulta então em:

- Definição de um modelo de dados – CIM [8];
- Uma codificação para transporte de dados – xmlCIM [11];
- Um conjunto de operações HTTP que definem o procedimento para a aquisição de dados de gestão por parte aplicação de gestão, assim como a a reposta de um agente às operações de gestão [10].

2.3.1 Arquitectura Subjacente

O CIMOM corresponde ao núcleo da arquitectura WBEM, e é encarregue da gestão dos objectos CIM e o seu armazenamento no repositório. O CIMOM tem a capacidade de gerir vários *namespaces* diferentes, contendo classes e instâncias distintas. É responsável pelo roteamento de informações sobre eventos e objectos entre os componentes, responde aos pedidos dos clientes CIM, verifica a sintaxe e a semântica das mensagens, e garante a segurança. O CIMOM possui a sua própria linguagem para definição de objectos geridos designada por MOF.

A Figura 2.10 ilustra a arquitectura WBEM focando o CIMOM.

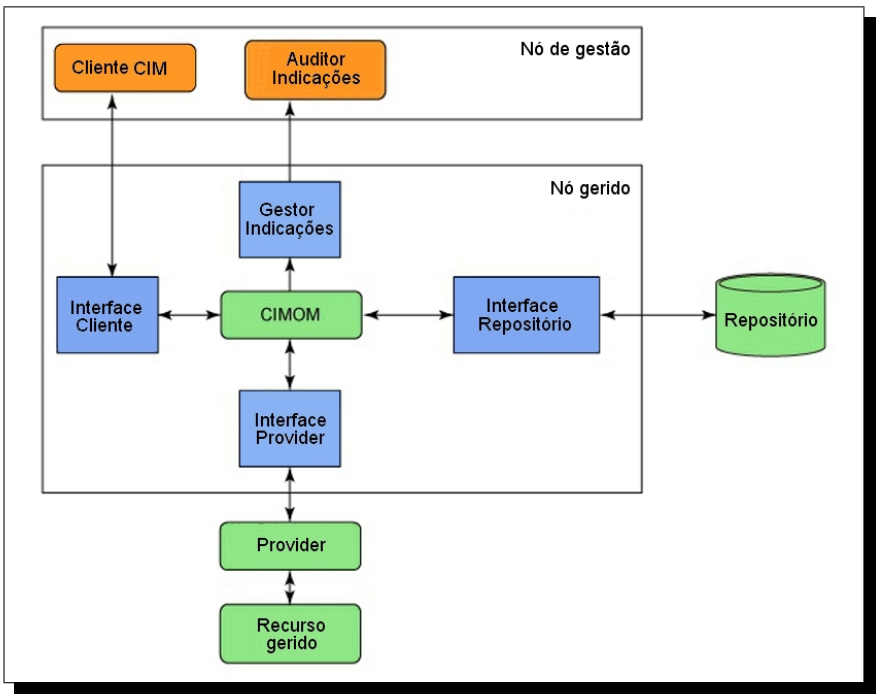

Figura 2.10: Arquitectura WBEM.

A arquitectura WBEM é composta por cinco componentes básicos, o Cliente CIM no lado do cliente e quatro componentes no lado do servidor WBEM, as aplicações de gestão, o CIMOM, o repositório de dados e os *providers* [54].

Os *providers* correspondem a um conjunto de processos que comunicam com os agentes de gestão dos sistemas convencionais nos dispositivos geridos. Os *providers* são responsáveis pela obtenção e definição dos dados de gestão, estabelecendo a comunicação com o agente do dispositivo gerido. Existem *providers* que utilizam arquitecturas normalizadas como SNMP,

Common Management Information Protocol (CMIP) e DMI, estes *providers* são denominados "*Standard Providers*".

Os *providers* podem criar, modificar ou remover instâncias de objectos ou classes. As instâncias podem existir em armazenamento persistente ou podem ser voláteis. Estes podem ser distinguidos de acordo com o tipo de pedidos efectuados ao CIMOM:

- ***Providers* de método**: utilizados para chamadas a métodos externos. Estas chamadas podem executar algo relevante para a classe que está sendo invocada;
- ***Providers* de instância**: executam a criação, enumeração e remoção de instâncias de classes particulares;
- ***Providers* de propriedade**: executam chamadas de consulta e modificação de propriedades de uma instância (get's e set's);
- ***Providers* de associação**: possuem a função de criar, remover e executar outras operações de associação entre classes ou instâncias;
- ***Providers* de indicação**: recebem eventos ou alarmes provenientes do sistema que está sendo monitorizado e enviam as indicações para os objectos que as subscreveram.

2.3.2 Modelo de dados

As tecnologias do DMTF foram projectados para trabalhar em conjunto, como ilustrado na Figura 2.11, de modo a fazer face às necessidades da indústria e aos requisitos para a gestão distribuída de forma interoperável. Estes padrões fornecem interfaces bem definidas que se complementam umas às outras, entregando capacidades transversais de gestão e interoperabilidade. As inter-relações entre as tecnologias DMTF neste diagrama apresentam um valor incremental através da stack, providenciando uma mais valia tanto maior quando o numero de camadas adicionais que forem executadas.

Como o diagrama da Figura 2.11 [12] ilustra, a base das tecnologias do DMTF é o CIM. A especificação da infra-estrutura CIM define regras CIM e fornece os detalhes para a integração com outros modelos de gestão. A camada seguinte é o esquema CIM, que oferece um modelo *Orientado a Objectos* (OO) de descrições semanticamente rico e orientado para todos os elementos geridos [8].

O esquema CIM facilita a integração simplificada bem como a redução de custos, permitindo a troca de informação de gestão através de uma plataforma independente da tecnologia de uma forma neutra.

O principal objectivo do modelo CIM consiste na descrição dos dados necessários para efectuar a gestão de sistemas de informação assim como nas relações existentes entre os mesmos, estando este modelo definido recorrendo a linguagem *Unified Modeling Language* (UML) [50].

Esta abordagem define um modelo de dados e não uma linguagem, ou implementação, para a definição desses mesmos dados. O modelo CIM utiliza os conceitos do paradigma orientado a objectos para a descrição das entidades existentes no sistema, como por exemplo os computadores, software, utilizadores e redes. Sendo baseado no paradigma orientado a objectos, existem no modelo CIM noções de classes, instâncias dessas classes, tipos de dados, herança, regras de associação entre objectos e métodos.

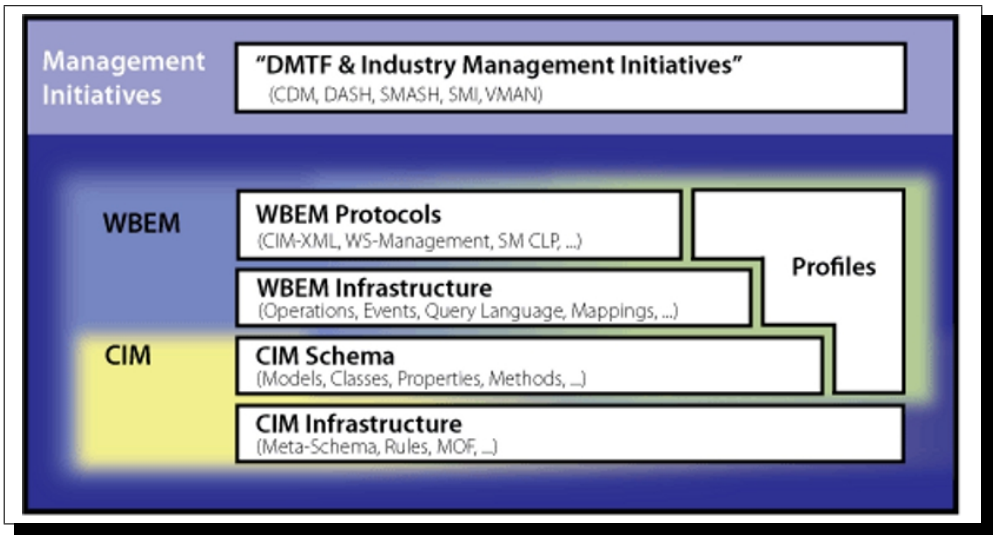

Figura 2.11: Mapa da stack WBEM.

Uma das características que torna este modelo tão importante, é o facto de ser extensível, tal como todos os modelos OO. A sua criação, no âmbito do WBEM, prevê que cada fabricante crie a sua implementação sobre o CIM, o que significa partir de classes já existentes, necessariamente genéricas, e criar classes que derivem dessas, e que definam o comportamento inerente ao sistema a que se destinam. Estas implementações devem permitir criar classes que representem discos rígidos, *routers* de rede ou mesmo tecnologias definidas pelo utilizador como seja por exemplo um ar-condicionado gerido através da rede.

Ao ver e alterar classes do CIM, o gestor pode controlar os diferentes aspectos da plataforma de gestão. Veja-se o exemplo em que o gestor executa uma *query* a uma classe CIM que representa uma *workstation*. Ele pode correr um *script* que altere uma instância dessa classe. É objectivo da implementação propagar essa alteração desde a instância da classe até ao dispositivo físico que ela representa. Este nível de abstracção torna o CIM totalmente independente da linguagem de programação.

O CIM é composto por:

- um modelo de informação básica referenciado como *metaschema*;
- uma sintaxe para a descrição dos objectos geridos MOF;
- dois níveis de classes genéricas dos objectos geridos, designadas por *Core Model* e *Common Model*.

O *metaschema* é definido com ajuda da linguagem UML e contém classes, métodos, qualificadores (*qualifiers*), associações, referências, classes de eventos e esquemas. As classes caracterizam um conjunto de objectos com propriedades comuns. Somente herança simples é possível entre as classes. Métodos definem procedimentos que podem ser executados em objectos de uma classe. Qualificadores (*qualifiers*) são usados para especificar características adicionais das classes ou propriedades e métodos das classes. Exemplos incluem [55]: opções de acesso (*READ*, *WRITE*), informações de atributo de mapeamento para uma *Management*

Information Base (MIB) existente (*MappingStrings*), classificação de atributos (*names*) como chaves (*Keys*) para entidades e identificação de classes que não podem ser instanciadas (*ABSTRACT*). Associações são classes (ou instância delas) que representam relações entre dois um mais objectos. Elas contém pelo menos duas referências. Referências são propriedades especiais. Classes de eventos são usadas para definir tipos diferentes de notificação de eventos. Esquemas são grupos de elementos combinados para um propósito administrativo.

Figura 2.12: Hierarquia de classes do CIM.

Existem três tipos de classes no CIM, com níveis crescentes de especificidade. Isto é, os dois primeiros níveis fazem parte da norma e definem aspectos genéricos da plataforma de gestão, enquanto que o terceiro nível é dependente da implementação de cada fabricante e, como tal, é constituído por classes que descendem do nível anterior e que incluem comportamento específico do elemento a gerir. Esses três tipos de classe são:

- O *Core Model*, representado como um círculo no centro da Figura 2.12, é constituído por um conjunto de associações e de classes abstractas que modelam as características genéricas e comuns a todas as áreas de gestão. Os sistemas, as aplicações e as redes são modeladas como extensões ao *Core Model*.

- O *Common Model*, representado como elipses na Figura 2.12, é um conjunto de modelos de informação relacionados com áreas específicas de gestão, mas de uma forma independente das tecnologias ou implementações. São exemplos de *Common Models* constantes na versão 2.20.2 do *Schema* CIM : CIM_Application, CIM_Application-J2eeAppServer, CIM_Database, CIM_Device, CIM_Event, CIM_Interop, CIM_IPsecPolicy, CIM_Metrics, CIM_Network, CIM_Physical, CIM_Policy, CIM_Security, CIM_Support, CIM_System e CIM_User.

- A camada *Extension Schema*, que não se encontra representada na Figura 2.12, é constituída por extensões às classes do *Comon Model*, de forma a representar os detalhes específicos de uma determinada tecnologia ou fabricante.

O *Common Model* refina o *Core Model* em classes não dependentes de tecnologias e implementações específicas, mas que são concretas o suficiente para servirem como base para muitas aplicações de gestão. O *Extension Schema* refina o modelo comum para tecnologias especiais. Por exemplo, a Microsoft especificou uma extensão de esquema, denominada *Win32Schema*, que especializa as classes do *Common Model* para Windows 95/98 e Windows NT.

2.3.3 Codificação da informação

O modelo CIM recorre a conceitos típicos de tecnologias orientadas a objectos para representar a informação acerca dos sistemas geridos. Devido a necessidade em obter uma representação dos elementos e mensagens CIM de modo a poderem ser transmitidos sobre HTTP e mediante algum tipo de codificação, foi elegido o XML por ser uma norma que permite representar informação de forma extensível, potente, e que acima de tudo porque foi concebido para ser usado sobre HTTP [10].

O XML é particularmente adequado como mecanismo de representação de dados em ambientes heterogéneos, porque o XML é baseado numa norma aberta, existem implementações de *parsers* para várias plataformas e para muitas linguagens de programação. Existem implementações para UNIX, Windows, desenvolvidas em linguagens C++ e Java. Adicionalmente, algumas implementações de *parsers* permitem aceder à árvore de elementos XML a partir de diferentes ambientes e linguagens de *script*.

Existem também várias implementações de *parsers* e *Application Programming Interface* (API) para os *browsers* mais populares. Estes factos, tornam o XML numa forma normalizada ideal para transferir dados entre plataformas heterogéneas. Ou seja, em ambientes de sistemas de gestão, a representação XML da informação de gestão pode ser usada para transferir dados entre plataformas de gestão que não precisam necessariamente de estar a correr necessariamente os mesmos sistemas operativos, mas onde a interoperabilidade é conseguida através de um entendimento comum da representação XML da informação de gestão.

O CIM pode ser usado como base para o DTD que define a representação XML (vocabulário) da informação de gestão. A este mapeamento do meta-modelo CIM num DTD XML chama-se CIM XML *Vocabulary*. Ao usar esta técnica, pode ser definido um *mapping* entre a informação de gestão que pode ser modelada usando o CIM e o vocabulário CIM XML. Claro que o inverso também seria verdadeiro, visto que qualquer informação que é representada usando o vocabulário CIM XML também pode ser representada usando a sintaxe MOF. De facto, pode ser escrito um *EXtensible Stylesheet Language* (XSL) de modo a que transforme informação de gestão representada num vocabulário CIM XML em informação que siga a sintaxe MOF.

Ao fazer-se uma representação do CIM em XML, está-se a permitir que se tire partido de todas as vantagens de usar o XML e as tecnologias que lhe estão associadas. Existem duas escolhas que podem ser tomadas em conta na tradução de dados CIM para XML [8]:

- *Schema Mapping* – neste tipo de tradução, o esquema XML é usado para descrever as classes CIM, e as instâncias CIM seriam traduzidas para documentos XML válidos relativos a esse esquema. Ou seja, cada classe CIM teria o seu DTD, nos quais o nome dos elementos CIM seriam tirados directamente dos nomes dos elementos CIM. Um exemplo de *Schema Mapping* seria algo como o representado na figura 2.13.

- *MetaSchema Mapping* – neste tipo de tradução, o esquema XML é usado para descrever o meta-esquema CIM, e quer as classes CIM, quer as suas instâncias são documentos

```
<Disk>
  <Properties>
    <Freespace>200</Freespace>
    </Properties>
</Disk>
```

Figura 2.13: Exemplo de Schema Mapping.

XML válidos relativamente a esse esquema. Ou seja, o DTD descreve de uma forma genérica a notação de uma classe ou instância CIM. Os nomes dos elementos CIM são traduzidos em atributos XML ou valores de elementos, ao invés de nomes de elementos XML. Um exemplo de *MetaSchema Mapping* seria algo como seria algo como o representado na figura 2.14.

```
<CLASS Name="Disk">
        <Property Name="Freespace">
          200
        </Property>
<CLASS>
```

Figura 2.14: Exemplo de MetaSchema Mapping

Apesar de poderem existir algumas vantagens em utilizar uma tradução do tipo *Schema Mapping*, foi escolhido utilizar o *MetaSchema Mapping* pelas seguintes razões [11]:

- Só requer um meta-esquema DTD normalizado para CIM, em vez de um número ilimitado de DTD. Este facto reduz consideravelmente a complexidade de gestão e administração das traduções XML;
- Um DTD XML não permite uma lista desordenada de elementos. Numa tradução estática tal significaria: fixar uma ordem arbitrária para listas de propriedades, métodos e qualificadores; definir uma tradução que tivesse em consideração todas as ordenações de listas explicitamente (e que cresceria em tamanho de forma exponencial com o número de elementos da lista);
- Outra desvantagem das traduções do tipo *Schema Mapping* é que os nomes dos elementos do esquema CIM (classes, propriedades, qualificadores e nomes de métodos) fazem parte do *namespace* de um elemento XML. De modo a replicar as regras de *scoping* nos nomes dos elementos CIM num DTD XML, seria necessário usar *namspaces* XML para definir um esquema XML com uma granularidade ao nível das propriedades;
- Embora o *Schema Mapping* permitisse a validação de instâncias sobre classes, tal só seria possível se toda a hierarquia de classes tivesse sido atravessada antes de traduzir uma classe CIM num esquema XML. De outra forma, as propriedades das classes que são herdadas estariam ausentes no DTD e a validação contra uma instância que incluía valores de propriedades herdadas falharia.

A metodologia para fazer o *MetaSchema mapping* seria definir o CIM num DTD representando valores, qualificadores, propriedades, métodos, *Paths* e Classes de instância e associação.

Depois de descrever em primeiro lugar nove entidades de parâmetros usados no vocabulário do esquema (*CIMName, CIMType, QualifierFlavor, ClassOrigin, Propagated, ArraySizes, SuperClass, ClassName e ReferenceClass*), o DTD descreve uma série de elementos que se apresentam a seguir [11].

- Elementos de alto nível – o elemento raíz de um documento XML CIM é o elemento CIM. Este elemento pode conter um só elemento *MESSAGE*, que define uma mensagem CIM (para ser usado em operações HTTP), ou uma declaração, *DECLARATION*, usada para declarar um conjunto de objectos CIM;

- Elementos declarativos – o elemento *DECLARATION* por sua vez, pode conter os elementos, *DECLGROUP.WITHNAME, DECLGROUP.WITHPATH* para declarar classes, instâncias e qualificadores com *QUALIFIER.DECLARATION*;

- Elementos de valores – os elementos de valores definem aqueles elementos do esquema que expressam o valor dos objectos CIM. O de mais alto nível é *VALUE* que pode englobar *VALUE.REFERENCE, VALUE.REFARRAY, VALUE.OBJECT* e outros;

- Elementos de Nome e Localização – são aqueles que expressam o nome e localização de objectos CIM (classes, instâncias, propriedades, métodos e qualificadores), por exemplo : *NAMESPACEPATH, LOCALNAMESPACEPATH, HOST, NAMESPACE, CLASSPATH*;

- Elementos de definição de objectos – estão relacionados com a expressão da definição de objectos CIM (classes, instâncias, propriedades, métodos e qualificadores). Existem os elementos *CLASS, INSTANCE, QUALIFIER, PROPERTY, METHOD, PARAMETER*;

- Elementos de mensagens (Operações, Invocação de métodos, retornos) – os elementos de mensagens são aqueles que servem para criar mensagens CIM para serem enviadas sobre HTTP. O elemento de mais alto nível é *MESSAGE* que pode conter um pedido, com *SIMPLEREQ* ou *MULTIREQ*, ou uma resposta com *SIMPLERSP* ou MULTIRSP. Os pedidos, por sua vez, têm invocações a métodos extrínsecos e intrínsecos com *(I)METHODCALL*, passando parâmetros com *(I)PARAMVALUE*, e as respostas contém *(I)METHODRESPONSE*. Por último, estas contém valores de retorno com *(I)RETURNVALUE* ou erros com *ERROR*;

2.3.4 Transporte da Informação

Todos os pedidos de mensagens CIM devem de ser executados recorrendo a métodos HTTP *M-POST* ou *POST*, sendo o mecanismo aconselhado o *M-POST*. O recurso a outros métodos HTTP encontram-se fora da especificação [10]. Todas as respostas de mensagem CIM são obtidas na correspondente mensagem HTTP de resposta resultante dos pedidos *M-POST* ou *POST*.

Um cliente CIM ao tentar uma invocação de mensagem CIM em conformidade com as especificações deve primeiro tentar a invocação usando o método *HTTP M-POST*":

- se a invocação *M-POST* falha com um *status* HTTP de "*501 Not Implemented*"ou "*510 Not Extended*", o cliente deve repetir a solicitação usando o método HTTP *POST* com

as modificações apropriadas. A intenção é que o *POST* deva ser usado apenas como um mecanismo de retorno em ambientes onde *firewalls* ou *proxies* não tenham ainda a capacidade de compreender pedidos *M-POST*.

- se a invocação *M-POST* falhar com um *status* HTTP de "*405 Method Not Allowed*", o cliente deve deixar o pedido.

- para todos os outros códigos de estado que o cliente deve agir em conformidade com a norma HTTP [39] [49].

Este mecanismo extenso de invocação permite que *proxies* de Internet e *firewalls* de filtragem possuam um maior controlo e flexibilidade administrativa sobre invocações de mensagem CIM.

No caso de um cliente que recebe um *status* 501 ou 510 em resposta a uma solicitação *M-POST*, de seguida, em chamadas subsequentes para o mesmo servidor HTTP, o cliente pode omitir a tentativa de invocações *M-POST* para um período adequado, evitando assim a necessidade de uma viagem de ida extra em cada invocação do método.

Dado este algoritmo, as *firewalls* podem, efectivamente forçar o uso do *M-POST*, proibindo invocações *POST* contendo a extensão do cabeçalho *CIMOperation* para mensagens CIM de operação e a extensão do cabeçalho *CIMExport* para exportar mensagens CIM. Em mensagens de solicitação *M-POST*, bem como as suas respostas, os cabeçalhos de extensão CIM devem ser declarados usando o *namespace* do prefixo atribuído pelo cabeçalho de extensão "*Man*"[65], que neste caso, se refere ao *namespace* "http://www.dmtf.org/cim/mapping/http/v1.0".

O formato completo para a declaração do cabeçalho "*Man*"é descrito na figura 2.15.

```
1 Man = "Man" ":" "http://www.dmtf.org/cim/mapping/http/v1.0"
2 ";" "ns" "=" header-prefix
3 header-prefix    = 2*DIGIT
```

Figura 2.15: Declaração do cabeçalho Man

Este *header-prefix* deve ser gerado aleatoriamente por cada mensagem HTTP, e não deve ser necessariamente um número específico.

```
1 M-POST /cimom HTTP/1.1
2 Man: http://www.dmtf.org./cim/mapping/http/v1.0 ; ns=23
3 23-CIMOperation: MethodCall
4 ...
```

Figura 2.16: Exemplo 1 - Utilizando M-POST

De acordo com [65], todas as mensagens de solicitação *POST*, bem como as suas respostas não devem incluir esta extensão. Em mensagens de solicitação *POST*, bem como as suas respostas, os *header-prefix* para nome não devem ser utilizados.

Operações WBEM [10], definem um conjunto de operações que um cliente WBEM implementa de forma a operar de uma forma aberta e padronizada. As operações WBEM são completamente independentes dos protocolos utilizados para o seu transporte.

```
POST /cimom HTTP/1.1
CIMOperation: MethodCall
...
```

Figura 2.17: Exemplo 1 - Utilizando POST

As operações WBEM podem ser simples (individuais) ou complexas sendo compostas por conjuntos de de várias operações (*batches*). Uma operação inclui os tipos de dados, os meta dados, as consultas e os métodos.

As operações CIM são invocações de métodos que podem ser intrínsecos ou extrínsecos conforme pertençam à especificação aqui referida (métodos invocados com vista à manipulação de um *namespace* CIM), ou sejam métodos pertencentes a classes do esquema CIM, respectivamente.

Métodos intrínsecos

O elemento < *IMETHODCALL* > corresponde à representação em XML da chamada a um método intrínseco, e a resposta a essa chamada è efectuada pelo elemento < *IMETHODRESPONSE* >. Os métodos intrínsecos definidos na especificação encontram-se descritos na tabela 2.2.

A especificação usa uma convenção para definir a assinatura dos métodos intrínsecos que é uma notação pseudo-MOF, com a qual se descrevem os métodos CIM com um número de pseudo-tipos de parâmetros (colocados entre "<"e ">"). Esta notação permite a classificação dos parâmetros com pseudo-qualificadores (*IN*, *OUT*, *OPTIONAL* e *NULL*) para definir uma semântica de invocação. *IN* denota que se trata de um parâmetro de entrada; *OUT* denota que se trata de um parâmetro de saída;*OPTIONAL* é usado para indicar que a presença de um determinado parâmetro não é obrigatória, indicando os parâmetros cujos valores podem ser especificados como sendo *NULL*. Segue-se um exemplo na Figura 2.18 de como é que a especificação define um método intrínseco:

```
GetClass
<class> GetClass (
     [IN]<className> ClassName,
     [IN, OPTIONAL] boolean LocalOnly = true,
     [IN, OPTIONAL] boolean IncludeQualifiers = true,
     [IN, OPTIONAL] boolean IncludeClassOrigin = false,
     [IN, OPTIONAL, NULL] string PropertyList [] = NULL
)
```

Figura 2.18: Método intrínseco

A tabela 2.3 ilustra como é que cada um dos pseudo-parâmetros usados pelos métodos intrínsecos deve ser traduzidos para XML, quer no contexto do valor de um parâmetro, quer como o valor de retorno.

Tabela 2.2: Métodos intrínsecos definidos na especificação

GetClass	Usado para retornar uma única classe CIM do *namespace* alvo.
EnumerateClasses	Usado para enumerar as subclasses de uma classe CIM no *namespace* de destino.
EnumerateClassNames	Usado para enumerar os nomes das subclasses de uma classe CIM no *namespace* de destino.
GetInstance	Usado para retornar uma única instância CIM do *namespace* de destino.
EnumerateInstances	Usado para enumerar as instâncias de uma classe CIM no *namespace* de destino.
EnumerateInstanceNames	Usado para enumerar os nomes de caminhos (modelo) das instâncias de uma classe CIM no *namespace* de destino.
GetProperty	Usado para recuperar um único valor da propriedade de uma instância CIM no *namespace* de destino.
SetProperty	Usado para definir um valor da propriedade de uma instância CIM no *namespace* de destino.
CreateInstance	Usado para criar um único instância CIM no *namespace* de destino.
ModifyInstance	Usado para modificar uma instância CIM existente no *namespace* de destino.
DeleteInstance	Usado para excluir uma única instância CIM do *namespace* de destino.
CreateClass	Usado para criar uma única classe CIM no *namespace* de destino.
ModifyClass	Usado para modificar uma classe CIM existente no *namespace* alvo.
Deleteclass	Usado para excluir um único CIM classe do *namespace* alvo.
Associators	Usado para enumerar objectos CIM (classes ou instâncias) que estão associados a uma determinada origem objectos CIM .
AssociatorNames	Usado para enumerar os nomes de objectos CIM (classes ou instâncias) que estão associados a uma determinada origem objectos CIM.
References	Usado para enumerar os objectos de associação que se referem a um alvo particular objectos CIM (Classe ou instância).
ReferenceNames	Usado para enumerar os objectos de associação que se referem a um alvo particular objectos CIM (Classe ou instância).
ExecQuery	Usado para executar uma consulta no *namespace* de destino.
GetQualifier	Usado para recuperar uma única declaração de *Qualifier* no *namespace* de destino.
SetQualifier	Usado para criar ou actualizar uma única declaração *Qualifier* no *namespace* de destino.
DeleteQualifier	Usado para excluir uma única declaração de *Qualifier* no *namespace* de destino.
EnumerateQualifiers	Usado para enumerar as declarações de *Qualifier* no *namespace* de destino.

Tabela 2.3: Tradução de pseudo-parâmetros de mensagens CIM em elementos XML.

Tipo	**Elemento XML**
<object>	(VALUE.OBJECT|VALUE.OBJECTWITHLOCALPATH|VALUE.OBJECTWITHPATH)
<class>	CLASS
<instance>	INSTANCE
<className>	CLASSNAME
<namedInstance>	VALUE.NAMEDINSTANCE
<instanceName>	INSTANCENAME
<objectWithPath>	VALUE.OBJECTWITHPATH
<objectName>	(CLASSNAME|INSTANCENAME)
<propertyValue>	(VALUE|VALUE.ARRAY|VALUE.REFERENCE)
<qualifierDecl>	QUALIFIER.DECLARATION

Métodos extrínsecos

O elemento <*METHODCALL*> corresponde a representação em XML da chamada a um método extrínseco, e a resposta a essa chamada pelo elemento <*METHODRESPONSE*>. Supõe-se que os servidores CIM devem suportar a chamada aos métodos extrínsecos que figuram no esquema CIM.

Se um servidor CIM não suportar invocações de métodos extrínsecos, este deve retornar o código de erro *CIM_ERR_NOT_SUPPORTED* a qualquer pedido de execução de um método extrínseco. Isso permite ao cliente CIM determinar que todas as tentativas para executar métodos extrínsecos irão falhar.

Se o servidor CIM for incapaz de realizar a invocação de um método extrínseco, um dos seguintes códigos de *status* deve ser retornado pelo servidor CIM, onde o primeiro erro aplicável na lista (começando com o primeiro elemento da lista, e descendo) é o erro retornado. Qualquer interpretação mais específica do erro é dado entre parêntesis.

- CIM_ERR_ACCESS_DENIED
- CIM_ERR_NOT_SUPPORTED (o servidor CIM não suporta invocações de métodos extrínsecos)
- CIM_ERR_INVALID_NAMESPACE
- CIM_ERR_INVALID_PARAMETER (incluindo falta, duplicação, desconhecimento ou então parâmetros inválidos)
- CIM_ERR_NOT_FOUND (a classe ou instância CIM de destino não existe no *namespace* especificado)
- CIM_ERR_METHOD_NOT_FOUND

- CIM_ERR_METHOD_NOT_AVAILABLE (o servidor CIM é incapaz de cumprir a invocação pedida)
- CIM_ERR_FAILED (um outro erro não especificado ocorreu)

2.4 NETCONF

O NETCONF [48] corresponde a uma proposta de protocolo de configuração de dispositivos de rede, que se encontra ainda em fase de normalização, dentro da área de operações e gestão do IETF, sob a responsabilidade do grupo de trabalho que dá nome ao protocolo, NETCONF.

O NETCONF pretende unificar o processo de configuração dos dispositivos de rede, e simultaneamente suprimir as deficiências das tecnologias actualmente disponíveis, como é o caso do SNMP. A sua proposta apresenta um conjunto de instruções simples e uniformes, mas suficientemente genéricas de forma a abranger diversos tipos de dispositivos. Essas instruções definem mecanismos para criar, editar, copiar e remover configurações, bem como obter informação de estado e estatísticas dos dispositivos.

2.4.1 Arquitectura Subjacente

O protocolo NETCONF recorre a chamadas de procedimentos remotos *Remote Procedure Calls* (RPC) como paradigma para a comunicação entre a entidade gestora (NETCONF *Manager*) e a entidade gerida (NETCONF *Agent*), cliente e servidor respectivamente, numa visão clássica de sistemas distribuídos. O NETCONF *Manager* é quem inicia uma sessão NETCONF. Esta sessão é caracterizada por ser segura e por uma ligação persistente entre o cliente e o servidor, para isso, deve existir um esquema de autenticação, autorização e confidencialidade.

As mensagens são codificadas em XML a fim de facilitar a processo de leitura e interpretação. Conceptualmente o NETCONF é divido em quatro camadas, de acordo com a Figura 2.19:

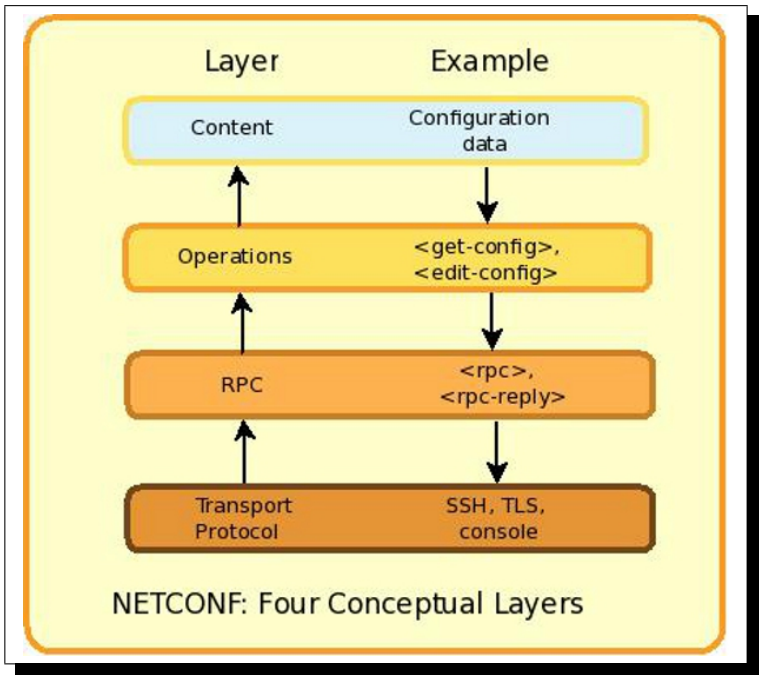

Figura 2.19: Camadas do modelo NETCONF.

1. ***Transport*** : Fornece a comunicação entre o cliente e o servidor. Actualmente está mapeado para *Secure Shell* (SSH) [76], *Blocks Extensible Exchange Protocol* (BEEP) [35], SOAP [51] e *Transport Layer Security* (TLS) [37], sendo obrigatório o suporte de SSH. Desde que sejam seguidos os requisitos básicos do NETCONF, qualquer outro transporte pode ser usado;

2. ***RPC*** : Fornece um mecanismo simples para codificação das chamadas remotas;

3. ***Operations*** : Conjunto básico de operações invocadas como métodos remotos através de parâmetros codificados em XML;

4. ***Content*** : Encontra-se fora do âmbito do protocolo. Refere-se ao *Modelo de Dados* (MD) de cada dispositivo.

2.4.2 Modelo de Dados

Uma decisão importante da especificação foi a separação do protocolo relativamente ao MD. Esta separação é ilustrada na Figura 2.19, onde a camada (*Content*) é mostrada mas não é detalhada no protocolo. Na proposta é explicado que o NETCONF deve ser independente da linguagem de definição de dados e do MD utilizado para descrever a configuração e o estado dos dados. Mais ainda, dada a natureza proprietária dos dados de configuração a serem utilizados, a especificação deste conteúdo depende da implementação do NETCONF. Por isso, um esforço está a ser empreendido no sentido de normalizar um MD, pois a definição de um MD corresponde ao factor crucial para adopção do protocolo, de modo que num futuro próximo a coexistência de um MD normalizado com a implementação do protocolo não seja um obstáculo à sua aceitação. A adopção de um MD é necessária para atingir a interoperabilidade entre entidades NETCONF a gerir (NETCONF *Agent*).

É importante esclarecer as diferenças entre um *Modelo de Informação* (MI) e um MD [71]. Este dois modelos são muito parecidos, podendo suscitar alguma confusão. O mais importante é que os dois diferem quanto ao propósito de cada um.

O MI define um alto nível de abstracção ou seja o modelo conceptual dos objectos a gerir, independentes de uma implementação específica ou dos protocolos utilizados para o transporte dos dados. De forma a manter o MI o mais claro possível, este deve esconder qualquer detalhe do protocolo e da implementação. Uma característica importante de todo MI é a definição das relações entre os objectos a gerir.

A linguagem para a descrição de um MI pode ser informal, como o Português ou Inglês. Outra forma de descrever o modelo é via UML. O UML tem a vantagem de utilizar conceitos OO como: abstracção, herança, encapsulamento e métodos. Como resultado do processo de modelação, oferece uma série de figuras geométricas específicas da linguagem, interligadas por linhas com vários tipos de terminações, oferecendo assim uma visão gráfica do modelo que facilita o sua interpretação. No entanto, a maioria dos modelos de gestão normalizados são MD.

Exemplos de Modelos de Dados são:

- SMI para a definição de módulos das MIB;

- *Structure of Policy Provising Information* (SPPI) para a definição de módulos das *Policy Information Base* (PIB);

- CIM, desenvolvido no DMTF. Tal como descrito na secção 2.3, apresenta a parte gráfica em UML e a parte textual em MOF.

Um MD é descrito numa linguagem formal, na qual são definidos os objectos a gerir a um nível de abstracção baixo. Incluem detalhes específicos da implementação e do protocolo. Existe uma relação entre um MI e um MD: o MI pode ser instanciado em vários MD. Por exemplo, um MI pode ser instanciado em SMI, SPPI ou MOF.

Consideremos por exemplo os módulos de um MIB SNMP. Uma MIB é definida numa *Request For Comments* (RFC) [34]. Esta é um MD, pois detalha as *Object IDentification* (OID), estruturas de índices, valores máximos de acesso para as variáveis e outras tantas informações de implementação. No entanto, algumas RFC contém algum tipo de descrição informal explicando partes do modelo por trás do módulo da MIB. Esta informação é um MI. Um exemplo é a RFC 2863 [56] para o grupo Interfaces.

Actualmente o protocolo de gestão de redes mais difundido é o SNMP, no entanto este apresenta alguns pontos fracos, nomeadamente a segurança e a escalabilidade, não atendendo assim, aos requisitos de configuração das redes actuais. O NETCONF através do XML, como linguagem de codificação de dados, é visto como uma solução promissora para o futuro da gestão de configurações de rede. Contudo, actualmente, o protocolo NETCONF, carece ainda de uma forma normalizada para representar o MD. Como consequência, os vendedores são forçados a usar soluções proprietárias. De forma a tornar o NETCONF um protocolo interoperável, os modelos devem ser definidos de forma neutra e independente dos fabricantes.

Para garantir a interoperabilidade do NETCONF e a capacidade de manipular dados de forma normalizada, o IETF criou um grupo de trabalho designado de NETMOD [21] com o objectivo de apoiar o desenvolvimento contínuo do IETF fornecendo modelos de dados que sejam adoptados pela industria. O NETMOD tem vindo a desenvolver a linguagem YANG [45], que foi proposta como linguagem de definição de dados, incluindo dados de configuração e de estado, manipulados pelo protocolo NETCONF, através de RPC e notificações [45]. Até a data foram lançados os seguintes documentos para revisão da futura norma YANG: *draft-ietf-netmod-yang*, *draft-ietf-netmod-yang-types*, *draft-ietf-netmod-dsdl-map*, *draft-ietf-netmod-arch*, *draft-ietf-netmod-yang-usage* [21].

A linguagem YANG é simples e intuitiva, devido à sua sintaxe simples e regular. Apresenta uma hierarquia clara, onde o papel dos elementos estão bem definidos e o mapeamento da informação é coerente. O YANG privilegia a legibilidade, mas assegura flexibilidade e extensibilidade suficiente que justifiquem a sua utilidade para esquemas complexos e extensos.

2.4.3 Codificação da informação

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 Identifier* (URI).

Modelo RPC

As mensagens RPC são codificadas em XML. Um pedido RPC é dado pelo elemento <*rpc*>. Este elemento encapsula os nomes e parâmetros do pedido, como conteúdo do ele-

mento <*rpc*>. O retorno é dado pelo elemento <*rpc-reply*>. Após o NETCONF *Agent* receber um pedido <*rpc*>, ele executa-o. Se o processamento ocorrer com sucesso, retornará o elemento <*ok*>, encapsulado como conteúdo do <*rpc-reply*>. Outro elemento RPC é o <*rpc-error*>. Este é encapsulado dentro de um <*rpc-reply*> caso haja alguma falha de processamento. Na Figura 2.20, é mostrada a sequência de mensagens para o estabelecimento da sessão NETCONF e a troca de mensagens do modelo RPC:

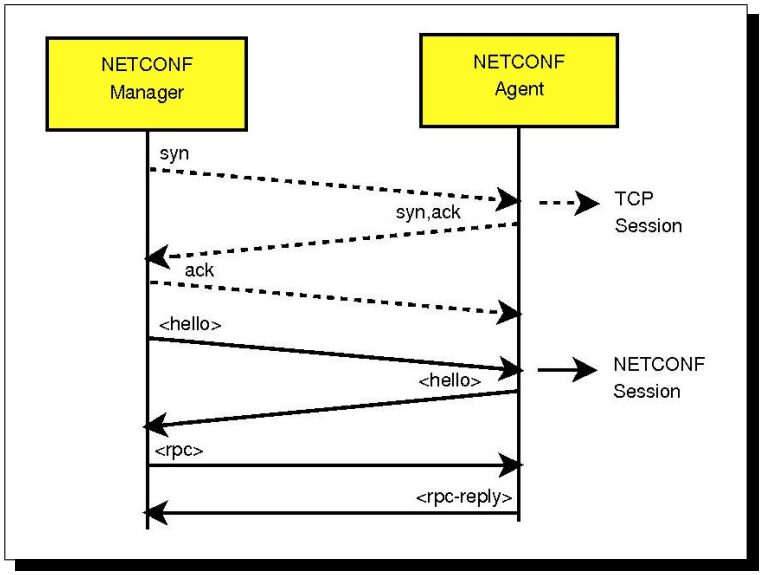

Figura 2.20: Estabelecimento da sessão NETCONF.

De acordo com a Figura 2.20, primeiro é estabelecida a sessão *Transmission Control Protocol* (TCP), através do mecanismo conhecido como *three-handshaking* [68]. Uma vez que o TCP é um protocolo orientado à ligação, para se estabelecer a comunicação entre um cliente e um servidor são trocadas as mensagens: *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 ligação e responde com o sinal *SYN/ACK*. O cliente responde com o sinal *ACK*, marcando o estabelecimento da sessão TCP.

Após o estabelecimento da sessão TCP, o NETCONF *Manager* (sempre o requisitante) envia a mensagem de <hello> para interrogar as capacidades do NETCONF *Agent*. O NETCONF *Agent* retorna uma mensagem <hello> divulgando suas capacidades. A partir deste momento o NETCONF *Manager* pode enviar os seus pedidos NETCONF ao NETCONF *Agent*. Os pedidos são encapsuladas em mensagens <rpc> e as respostas em mensagens <rpc-reply>.

Ao definir os elementos básicos <rpc> e <*rpc-reply*> para uma mensagem TCP, é fácil criar outras operações a partir destes elementos. As operações suportadas pelo NETCONF são definidas na Secção 2.4.3. Sendo o modelo TCP baseado no modelo pedido-resposta, facilita a criação de transacções atómicas. É de salientar que o uso do modelo TCP por si só não define completamente uma transacção atómica, para isso, são necessárias capacidades adicionais como a de *rollback* (2.4.3), através da operação <lock>.

O elemento <*rpc*>

O elemento <rpc> é usado para encapsular o pedido NETCONF enviada sempre do NETCONF *Manager* para o NETCONF *Agent*. Este elemento possui um atributo obrigatório chamado *message-id*. Este atributo é do tipo inteiro, e é incrementado pelo cliente à medida que os pedidos são enviadas. O servidor armazena o valor do atributo recebido e o retorna na mensagem <*rpc-reply*> correspondente. A Figura 2.21 ilustra o *template* do pedido:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <algum-metodo>
    <h-- parametros do metodo... -->
  </algum-metodo>
</rpc>
```

Figura 2.21: Template de pedido <rpc>.

Se existirem atributos adicionais presentes num elemento <rpc>, o servidor deverá retorná-los no elemento <*rpc-reply*> sem modificações. O nome da chamada remota é um elemento dentro do elemento <rpc> e os seus parâmetros são codificados dentro daquele elemento. No exemplo da Figura 2.22 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-metodo xmlns="HTTP://example.net/me/my-own/1.0">
    <primeiro>14</primeiro>
    <segundo>fred</segundo>
  </meu-metodo>
</rpc>
```

Figura 2.22: Exemplo de pedido <rpc>.

O elemento <*rpc-error*>

O elemento <*rpc-error*> é enviado encapsulado no elemento <*rpc-reply*> se um erro ocorrer durante o processamento de um pedido <*rpc*>. O elemento <*rpc-error*> possui outros elementos que são encapsulados dentro dele com informações a cerca do erro, tais como:

- <*error-type*> Define a camada conceptual onde o erro ocorreu. Pode ser uma de quatro possibilidades: *transport*, *rpc*, *protocol* ou *application*;
- <*error-tag*> Contém a string que identifica a condição de erro. O apêndice A de [14] possui uma lista dos possíveis valores;
- <*error-severity*> Contém a string que identifica a gravidade do erro. Pode ser uma de duas possibilidades: *error*, *warning*;
- <*error-app-tag*> Contém a string que identifica a condição de erro da implementação ou do MD específico, se existir;

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

Um erro é retornado se o elemento <*rpc*> for recebido sem o atributo *message-id*, por exemplo. Na mensagem da Figura 2.23, após a palavra *rpc* e antes da palavra *xmlns*, está em falta o atributo mandatório *message-id*.

```
1  <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2    <get-config>
3      <source>
4        <running/>
5      </source>
6    </get-config>
7  </rpc>
```

Figura 2.23: Pedido <rpc> com erro.

Como resultado, é gerada a mensagem de erro da figura 2.24. Na mensagem de erro 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 de existir apenas um erro no pedido.

```
1  <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2    <rpc-error>
3      <error-type>rpc</error-type>
4      <error-tag>missing-attribute</error-tag>
5      <error-severity>error</error-severity>
6      <error-info>
7        <bad-attribute>message-id</bad-attribute>
8        <bad-element>rpc</bad-element>
9      </error-info>
10   </rpc-error>
11 </rpc-reply>
```

Figura 2.24: Resposta ao pedido <rpc> com erro.

O elemento <*rpc-reply*>

O elemento<*rpc-reply*> é enviado do NETCONF *Agent* para o NETCONF *Manager* em resposta a uma operação <*rpc*>. Este 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 mencionado anteriormente, se existirem atributos adicionais presentes num elemento <*rpc*>, o NETCONF *Agent* deverá retorná-los no elemento <*rpc-reply*> sem modificação. O nome

da resposta bem como os seus dados 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 da Figura 2.25 é enviada um pedido TCP com o atributo *"user-id"*. Este pedido 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>
```

Figura 2.25: Pedido <rpc>.

O conteúdo do retorno segue encapsulado no elemento <*data*> e é ilustrado na Figura 2.26.

```
<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>
    <h-- Conteudo aqui... -->
  </data>
</rpc-reply>
```

Figura 2.26: Pedido <rpc>.

O elemento <ok>

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

A Figura 2.27 ilustra a resposta <ok>.

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

Figura 2.27: Resposta <ok>

Repositórios

O protocolo NETCONF contém várias operações *standard* que operam sobre uma ou mais configurações conceptuais de repositórios. Por exemplo, o parâmetro <*target*> para a operação <*edit-config*> especifica qual o repositório a editar. Os repositórios de configuração são representados (em XML) nas operações NETCONF tanto como um elemento vazio, identificação um repositório *standard*, ou um elemento <*url*> identificando um repositório não-*standard*, como por exemplo o repositório *offline*.

Existem três repositórios conceptuais *standard* para configuração:

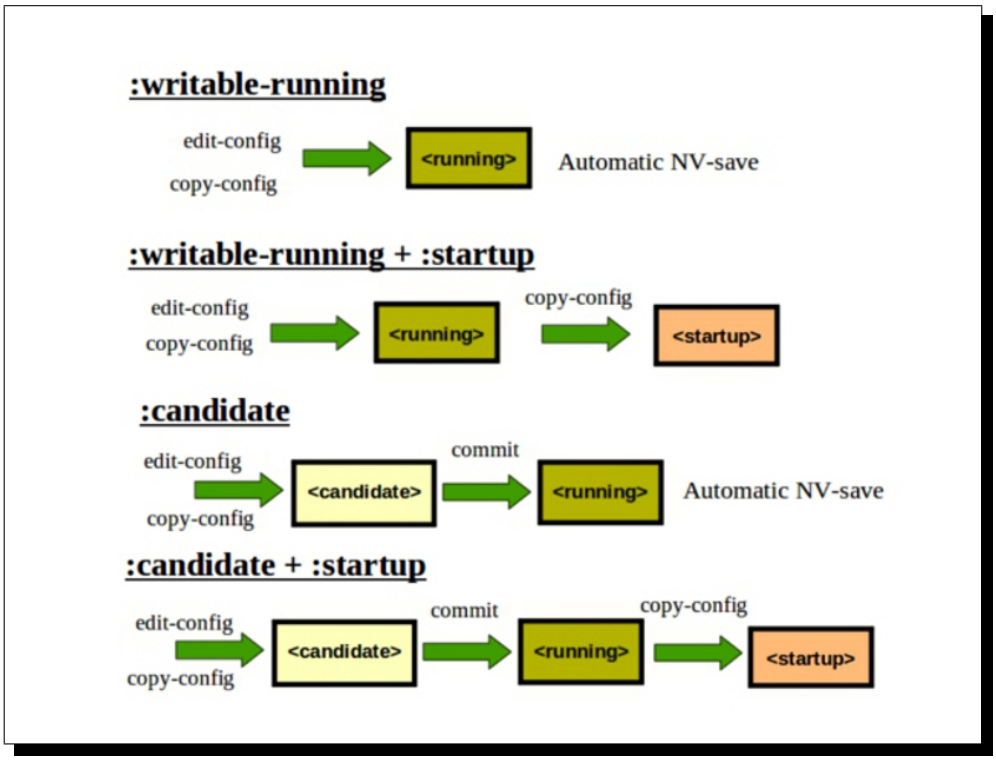

Figura 2.28: Repositórios NETCONF.

- *<running/>* Este repositório representa toda a configuração actualmente activa no dispositivo. É o único repositório obrigatória segundo a norma. A menos que o servidor suporte a capacidade de **:candidate**, o servidor deve permitir que este repositório seja editada directamente. Caso contrário, o servidor não é obrigado a suportar a sua edição directa. Se ele a suportar, então a capacidade **:writable-running** será anunciada pelo servidor para indicar este suporte. O repositório *<running/>* também é usada para conter todas as informações sobre o estado actual dispositivo.

- *<candidate/>* Este repositório está disponível, se a capacidade de **:candidate** for suportado pelo servidor. É um repositório global, mas temporária. É usada para guardar as edições efectuadas por uma ou mais operações *<edit-config>*. Um NETCONF *Manager* pode construir um conjunto de alterações, que podem ou não, ser validadas pelo servidor, até que sejam explicitamente aplicadas à configuração actual, de uma só vez. Ao contrário do repositório *<running/>*, as alterações feitas ao repositório *<candidate/>* não têm efeito imediato no dispositivo de rede. Quando a edição estiver completa, o NETCONF *Manager* pode usar a operação *<commit>* para activar as mudanças inseridas no repositório *<candidate/>*, e torná-las parte da configuração *<running/>*. Após uma operação bem sucedida *<commit>*, o repositório *<candidate/>* e o repositório *<running/>* têm o mesmo conteúdo de configuração.

  Esta condição é especialmente importante porque uma operação de *<lock>* no repositório *<candidate/>* não limita o seu acesso. Não importa qual a sessão que faz alterações no repositório *<candidate/>*. Se ela é diferente do repositório *<running/>*, então não pode ser bloqueada. Como este é um repositório global, o NETCONF *Manager* deve

usar a operação <*discard-changes*> para remover quaisquer alterações não desejadas, se a operação <*commit*> não for usada. Isto irá limpar o repositório <*candidate/*> sem activar as possíveis alterações que ela possa conter, e prevenir qualquer NETCONF *Manager* de usar esto repositório global e fazer alterações indesejadas. Os NETCONF *Agent* que suportam a capacidade de **:candidate**, em geral também não suportam a capacidade de **:writable-running**, uma vez que ao permitir os dois tipos de acesso ao repositório <*running/*>, contraria a finalidade de utilizar este repositório de rascunho para aplicar e validar as alterações antes de as submeter.

- <*startup/*> Este repositório está disponível, se a capacidade de **:startup** for suportado pelo NETCONF *Agent*. Ela representa a configuração a usar na próxima reinicialização do dispositivo. Se estiver presente, o servidor não irá salvar automaticamente as alterações do repositório <*running/*> no repositório de armazenamento não-volátil. Em vez disso, uma operação <*copy-config*> é necessária para substituir o conteúdo do repositório <*startup/*> pela configuração actual. Se não estiver presente, o servidor irá actualizar automaticamente a sua memória não volátil a qualquer momento, quando a configuração actual for modificada. Em ambos os casos, o servidor é obrigado a manter a memória não volátil da configuração actual, e ser capaz de restaurar uma configuração em funcionamento após uma reinicialização.

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 efectuadas pelo NETCONF *Manager* sobre o NETCONF *Agent* e são realizadas sobre a configuração do repositório actual do dispositivo, designada de *running.* Outras configurações são possíveis (Secção 2.4.3).

As operações básicas permitem que uma tarefa de configuração seja acompanhada do início ao fim. Ao suportar operações explícitas para a criação, edição ou remoção de objectos de dados, o NETCONF permite que o NETCONF *Agent* informe claramente que tipo de erro ocorreu. Isto facilita o tratamento de erros tanto no NETCONF *Agent*, que pode mais facilmente recuperar de uma falha e retornar ao estado anterior, como no NETCONF *Manager* que vai saber exactamente o que aconteceu e que medidas correctivas deve tomar para contornar o problema ou evitar que aconteça novamente.

No documento do NETCONF [48], 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 na tabela 2.4.

Os possíveis parâmetros que as operações podem receber são mapeados em elementos XML:

- *Session-id*: Número de identificação da sessão NETCONF que se deseja 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);

Tabela 2.4: Operações do protocolo NETCONF

Operação	**Uso**	**Descrição**
close-session	:base	Termina a própria sessão, libertando todos os recursos associados a ela.
commit	:base AND :candidate	Submete as conFigurações presentes no repositório <*candidate/*> para o repositório <*running/*>
copy-config	:base	Copia inteiramente uma nova configuração ou sobrescreve a actual, se permitido.
create-subscription	:notification	Cria uma subscrição de uma notificação NETCONF.
delete-config	:base	Apaga a configuração indicada. A configuração *running* não pode ser apagada. Por isso, depende de capacidades adicionais, como a : candidate.
discard-changes	:base AND :candidate	Apaga todas as alterações de configuração no repositório <*candidate/*>, restaurando a configuração do repositório <*running/*>
edit-config	:base	Edita parcialmente ou totalmente a configuração da base de dado especificada. Define operações para actualizar, criar, apagar ou fazer o merge da configuração actual com os novos dados que estão sendo enviados encapsulados no elemento <*edit-config*>.
get	:base	Obtém a configuração running e os dados de estado do dispositivo.
get-config	:base	Obtém toda ou parte de uma configuração running.
kill-session	:base	Força o término de uma sessão NETCONF através da indicação do número de identficação da sessão que se deseja encerrar.
lock	:base	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.
unlock	:base	Liberta a configuração para que outras sessões a usem.
validate	:base AND :validate	Valida todo o conteudo de configuração de umo repositório.

- *Config*: Hierarquia de dados de configuração definido pelo MD do dispositivo. Identifica o ponto de substituição da configuração actual pelos novos dados;
- *Target*: Nome do ficheiro de configuração que será o alvo de uma edição, cópia, remoção ou *lock*/*unlock*, conforme a operação. Pode ser o repositório padrão *running* ou outra, conforme o suporte a capacidades adicionais (*candidate*, por exemplo);
- *Source*: Nome do ficheiro 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 ficheiro de configuração *target* será realizada. Pode ser um dos seguintes tipos:

  - *Merge*: A configuração actual é fundida com os novos dados de configuração. Esta é a forma padrão;
  - *Replace*: Repõe completamente um ficheiro de configuração por um totalmente novo;
  - *None*: O ficheiro de configuração não é afectado pelos novos dados de configuração.

- *Test-Option*[1]: Define se o agente deve validar ou não a nova configuração recebida. Válido somente se for definida a capacidade validate;

- *Error-Option*[2]: 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*> (Figura 2.27), ou dados encapsulados na *tag* <*data*>, todos embutidos na *tag* <*rpc-reply*> (Figura 2.26). Uma resposta negativa é dada em virtude da falha de processamento por algum motivo (e.g. mensagem mal-formada ou dados fornecidos incorrectos), encapsulada na mensagem <*rpc-error*> (Figura 2.24).

Na Tabela 2.5 são descritos os parâmetros que devem estar presentes na mensagem e as respostas esperadas para cada operação.

Tabela 2.5: Operações básicas e parâmetros.

Operação	**Sess id**	**Filtro**	**config**	**target**	**source**	**Def Op**	**Resp. Positiva**	**Resp. Negativa**
<**get**>	Não	Opci	Não	Não	Não	Não	<data>	<rpc-error>
<**get-config**>	Não	Opci	Não	Não	Opci	Não	<data>	<rpc-error>
<**edit-config**> [2]	Não	Opci	Opci	Opci	Não	Opci	<data>	<rpc-error>
<**copy-config**>	Não	Não	Não	Opci	Opci	Não	<ok>	<rpc-error>
<**delete-config**>	Não	Não	Não	Obrig	Não	Não	<ok>	<rpc-error>
<**lock**>	Não	Não	Não	Obrig	Não	Não	<ok>	<rpc-error>
<**unlock**>	Não	Não	Não	Obrig	Não	Não	<ok>	<rpc-error>
<**close-session**>	Não	Não	Não	Não	Não	Não	<ok>	<rpc-error>
<**kill-session**>	Obrig	Não	Não	Não	Não	Não	<ok>	<rpc-error>

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

[2]Para a operação <*edit-config*> também é definido o atributo *operation* para qualquer elemento da confi-

Notificação de eventos

Depois da submissão do NETCONF, foi proposto ao grupo de trabalho NETCONF uma forma de enviar mensagens assíncronas ou notificação de eventos compatível com o protocolo NETCONF. A proposta actual está na sua versão 08 [75]. A proposta define o procedimento no qual um cliente NETCONF subscreve a recepção de eventos do seu interesse. A sua implementação é definida como uma nova capacidade suportada pelas entidades NETCONF *Manager* e NETCONF *Agent*, que é identificada da seguinte forma: ***urn:ietf:params:xml:ns:netconf:notification:1.0***.

Subscrição de Notificações

De forma a se mostrar interessado em receber notificações relativas à ocorrência de num evento, o NETCONF *Manager* envia ao servidor a mensagem de <create-subscription>. Nela é informada a classe de evento de interesse. No caso do NETCONF *Agent* poder atender o pedido, este envia um <*rpc-reply*> contendo o *id* da subscrição no elemento <data>. A partir deste momento, sempre que ocorrer um evento para o qual uma subscrição foi requisitada, 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, geralmente 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 2.29. De maneira a simplificar a Figura 2.29, não é mostrado os elementos <*rpc*> e <*rpc-reply*> que encapsulam respectivamente, os pedidos e as respostas após a troca de mensagens <*hello*>.

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 acção específica;
- *Data dump*: Contém informação do sistema, sua configuração e seu estado;

guração. Este atributo pode estar presente em qualquer ponto do ficheiro 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 ficheiro de configuração.

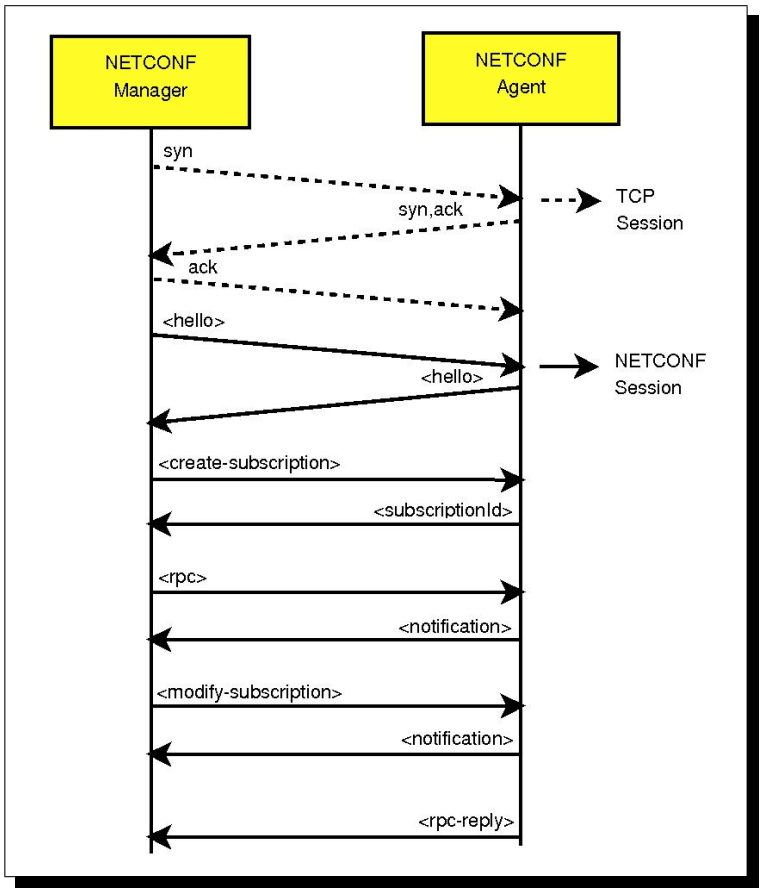

Figura 2.29: Serviço de Eventos.

- *Maintenance*: Indica o início, o meio e/ou o fim de uma acção 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 acção 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.

Capacidades

Capacidades são informações relativas às características dos elementos de gestão NETCONF que permitem estender o protocolo na medida em que permitem ao NETCONF *Manager* ajustar seu comportamento em relação ao NETCONF *Agent* de modo a obter um melhor aproveitamento no processo de configuração. 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 NETCONF *Manager* e o NETCONF *Agent* 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 ficheiro de configuração, no qual alterações não impactam na configuração actual, *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 actualização, 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 afecta a operação base. A partir das mensagens básicas <*rpc*> e <*rpc-reply*> do modelo TCP 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 transacções atómicas, onde a configuração é realizada por completo ou o dispositivo retorna ao estado anterior. A capacidade <*candidate*> permite que se tenha mais de um repositório de configuração. Este outro repositório é diferenciado pelo seu nome, <*candidate*>, diferente do nome do repositório de configuração actual, <*running*>, facilitando a nomeação dos repositórios de configuração.

2.4.4 Transporte da Informação

O protocolo NETCONF é baseado no paradigma de comunicação RPC. Um NETCONF *Manager* envia uma ou mais operações RPC de *request*, e o NETCONF *Agent* responde, igualmente com respostas RPC. Garantindo que um qualquer protocolo de transporte suporte chamadas RPC e respeite alguns pré-requisitos básicos, este pode ser usado para transportar mensagens NETCONF. Os pré-requisitos são:

1. **Operação orientada a ligação**: O NETCONF é orientado à ligação e requer, portanto, uma ligação persistente entre o cliente e o servidor. Esta ligação deve garantir uma comunicação fiável para os dados, além da sua entrega em sequência. As ligações devem manter-se durante um período relativamente extenso, 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 ligação. Assim que a ligação é encerrada, os recursos alocados a esta, devem ser libertados no servidor de forma a 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 ligações NETCONF devem ser autenticadas e isto é da responsabilidade do protocolo de transporte. Além do mais, o NETCONF faz sempre uso do mecanismo já utilizado pelo dispositivo a gerir. Por exemplo, se este usa *Remote Authentication Dial In User Service* (RADIUS) [69] 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.

Por exemplo, através de um protocolo de transporte que suporte ligação, como o HTTP sobre TCP, ou o uso do *HyperText Transfer Protocol Secure* (HTTPS) é possível estabelecer uma transacção orientada à ligação que seja segura. No caso do HTTPS, é o próprio protocolo que oferece a infra-estrutura de segurança. O NETCONF apenas utiliza o que é disponibilizado para ele.

O grupo de trabalho do IETF definiu quatro propostas referentes ao protocolo de transporte das mensagens NETCONF: SSH [76], BEEP [35], TLS [37] e SOAP [51].

O transporte via SSH é definido como obrigatório. Depois que o serviço de ligação 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]]>]]>.

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 NETCONF *Manager* ou NETCONF *Agent* inicie a sessão NETCONF.

No transporte via TLS, o elemento que actua como como NETCONF *Manager* também deve actuar como cliente TLS. Este deve-se conectar ao NETCONF *Agent* que escuta passivamente os pedidos de ligação TLS na porta TCP 6513. O NETCONF *Manager* envia a mensagem TLS *ClientHello* para iniciar o *handshake* TLS. Quando o *handshake* TLS tiver terminado como sucesso, o NETCONF *Manager* e o NETCONF *Agent* podem iniciar a troca de dados NETCONF.

Por último, é definido o mapeamento para SOAP. A grande vantagem deste mapeamento é que este é geralmente 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 possível, no entanto este sofre dos mesmos defeitos e qualidades do mapeamento do NETCONF directamente 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 para o servidor SOAP como por exemplo, o usuário e a 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 ligações 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 2.20. Além destas e outras recomendações, a proposta [51] fornece um *Web Service Definition Language* (WSDL) e uma amostra de um serviço WSDL. Na proposta do protocolo NETCONF é fornecido um *schema* XML. A união destes três, num único ficheiro WSDL, fornece os meios para se construir uma implementação do protocolo NETCONF usando SOAP.

Falhas no SOAP

No caso de falha no processamento do pedido, uma falha SOAP deve ser construída conforme definido pelo W3C como forma de encapsular a mensagem <rpc-error> definida no protocolo NETCONF. Na Figura 2.24 é apresentada uma mensagem de erro devido à mensagem do pedido não possuir o atributo obrigatório *message-id*. A seguir é apresentado na Figura 2.30 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>
```

Figura 2.30: Resposta SOAP ao pedido <rpc> com erro.

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 corpo da mensagem SOAP.

Transporte de Notificações

A notificação de eventos é mapeada para os protocolos de transporte já definidos para o NETCONF: SSH, BEEP, TLS e SOAP. A grande questão é que as notificação não são

confirmadas. Ao acontecer o evento, o NETCONF *Agent* 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 um pedido (do NETCONF *Manager*) existe sempre uma resposta (do NETCONF *Agent*).

2.5 Sumário

A PBNM não é mais que uma evolução nos princípios de gestão de redes, ao favorecer a visão global dos sistemas, de modo a controlar eficazmente as infraestruturas de uma forma integrada, onde a definição de políticas na rede corresponde a um nível superior de abstracção, pois, além de afastar o detalhe técnico não nuclear, abre simultaneamente a porta a várias vantagens indirectas, nomeadamente aumentos de produtividade, reduções significativas no tempo de integração, redução do erro associado ao factor humano e possibilita a atomização das operações de configuração.

O processo actual de convergência das redes de telecomunicações, têm implicações directas no planeamento, gestão e manutenção das infra-estruturas de rede, as quais devem transportar simultaneamente e de forma eficiente, serviços de dados, de voz e multimédia. Dentro deste novo conceito de redes, um importante elemento a ser gerido é a QoS. A gestão de QoS é uma tarefa bastante complexa e envolve conceitos administrativos, tais como: distribuição dos recursos disponíveis, configuração de perfis de cliente, diferenciação das aplicações e contabilização da utilização. A PBNM tem demonstrado ser uma estratégia eficiente para simplificar a administração de sistemas complexos, normalmente caracterizados pela heterogeneidade dos dispositivos e pela variedade de serviços oferecidos.

Como resposta à falta de um modelo único para a gestão de redes e sistemas, surgiu uma tecnologia, resultado da iniciativa de várias companhias do sector das tecnologias de informação, o WBEM. À medida que o WBEM amadureceu, foram-lhe atribuídos três componentes chave: o CIM como forma de descrição de informação de gestão; uma codificação dessa informação no formato XML; e um protocolo universal para transporte desse formato, o HTTP. A combinação de tecnologias feita pelo WBEM (o CIM, XML e HTTP) abre grandes possibilidades no que diz respeito aos aspectos de unificação, interoperabilidade, troca e partilha de informação de gestão. Para além de existirem várias ferramentas para tirar partido deles, de uma forma geral, elas existem independentemente das linguagens de programação, plataformas ou sistemas operativos onde operam.

As principais ferramentas de gestão de redes disponíveis, CLI e SNMP, não atendem aos requisitos de configurações das redes actuais. Por outro lado, há uma significativa disseminação e uma clara aplicação de tecnologias baseadas em XML no domínio das Tecnologias de Informação (TI) e Internet. Com o objectivo de aplicar estas novas tecnologias na configuração de redes foi definido, no âmbito do IETF, um novo protocolo de gestão de configuração denominado NETCONF. O protocolo NETCONF recorre a chamadas de procedimentos remotos RPC para a comunicação entre a entidade gestora e a entidade gerida, o NETCONF *Manager* e o NETCONF *Agent* respectivamente.

Uma decisão importante da especificação foi a separação do protocolo relativamente ao MD. Na proposta é explicado que o NETCONF deve ser independente da linguagem de definição de dados e do MD utilizado para descrever a configuração e o estado dos dados.

Para garantir a interoperabilidade do NETCONF e a capacidade de manipular dados de forma normalizada o IETF criou um grupo de trabalho designado de NETMOD, com o objectivo de desenvolver um MD manipulado pelo NETCONF e que seja aceite pela indústria. A linguagem proposta pelo NETMOD é o YANG, no entanto, esta ainda se encontra a fase de refinamento, de acordo com os sucessivos *drafts* quem tem sido publicados.

Capítulo 3

IP Multimedia Subsystem

Em 1998 foi criado um grupo de trabalho, designado por 3GPP, e composto por cinco grandes organismos de normalização a nível mundial; o ETSI da Europa, a *Association of Radio Industries and Businesses* (ARIB) [1]/*Telecommunication Technology Committee* (TTC) [30] do Japão, a *China Communications Standards Association* (CCSA) [3] da China, a *American National Standard* (ATIS) [2] da América do Norte e a *Telecommunication Technology Association* (TTA) [32] da Coreia do Sul. Os objectivos do consorcio prendiam-se com a criação de um standard a nível mundial para a normalização da tecnologia de suporte à Terceira Geração (3G) de redes móveis. O standard criado foi designado de *Universal Mobile Telecommunications System* (UMTS), resultado da evolução do *Groupe Special Mobile* (GSM) e *General packet radio service* (GPRS).

Um dos principais objectivos das redes móveis da 3G é a integração entre dois dos mais importantes paradigmas da comunicação dos últimos anos: as comunicações móveis e a Internet. Nesse contexto, a arquitectura IMS é o elemento fundamental para permitir o acesso ubíquo à Internet através das redes móveis. Numa definição mais formal, diz-se que o IMS é uma arquitectura global baseada em redes IP que possibilita a oferta de serviços multimédia aos utilizadores através da Internet [42].

3.1 Introdução

O IMS foi introduzido na *Release* 5 do UMTS e desde então tem sido actualizado através de lançamentos sucessivos de normas, encontrando-se actualmente na *Release* 10. Contudo, a *Release* 7 é considerada por muitos como a primeira especificação estável e completa. Paralelamente à especificação IMS, e após o lançamento da *Release* 6, o grupo de trabalho TISPAN lançou uma arquitectura de telecomunicações baseada no IMS e denominada NGN-R1 que tentava expandir o IMS para o uso em redes fixas da próxima geração. Como essa iniciativa iria criar duas normalizações distintas, foi decidido fundir as normas criadas pela 3GPP e TISPAN na *Release* 7 do IMS.

O 3GPP continuou sempre responsável pelas especificações do IMS, no entanto recebe contribuições de vários outros órgãos de normalização, dos quais se podem destacar: Grupo de trabalho *Focus on NGN* (FGNGN) pertencente ao ITU-T, TISPAN pertencente ao ETSI, *Open Mobile Alliance* (OMA) e IETF. A Figura 3.1, ilustra as áreas de contribuição dos vários organismos de normalização.

Em linhas gerais, pode-se considerar que o 3GPP assenta a normalização do core do IMS,

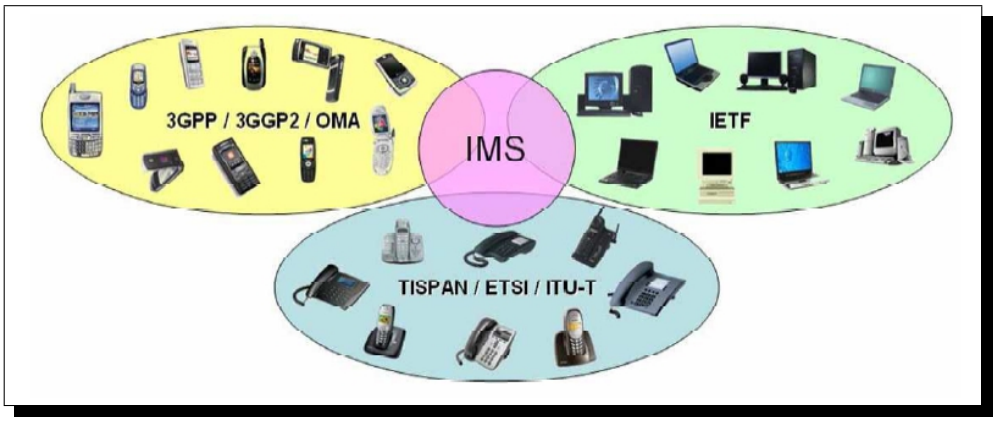

Figura 3.1: Áreas de contribuição para a arquitectura IMS.

baseado essencialmente no protocolo *Session Initiation Protocol* (SIP) [43], que por sua vez tinha sido normalizado pelo IETF. O acesso e a integração da rede fixa são normalizados pelo TISPAN, enquanto o próprio 3GPP é responsável pela normalização e integração do acesso móvel. Os outros grupos de normalização, como a OMA, actuam principalmente nas normalizações das aplicações IMS.

O IMS é uma arquitectura para redes de telecomunicações de próxima geração. Esta arquitectura é caracterizada por ser baseada inteiramente em IP, e por garantir a convergência fixo-móvel através da independência da tecnologia de acesso. O IMS é actualmente a arquitectura de referência para as redes de próxima geração, permitindo a integração entre as Redes Inteligentes (IN) tradicionais e as tecnologias emergentes como o *Voice Over IP* (VOIP) e os serviços de Internet.

A arquitectura IMS confere aos recursos da rede a inteligência necessária para uma melhor gestão na camada de transporte, oferecendo uma adaptação às diferentes necessidades de serviços e aplicações, de modo transparente, permitindo aos operadores de telecomunicações efectuar o desenvolvimento dos serviços sem ter em conta os pormenores de funcionamento e gestão da própria infra-estrutura de comunicações. Permite-lhes assim um desenvolvimento dos serviços de uma forma mais fácil e rápida, melhorando o tempo de implementação de novos serviços. Simultaneamente permite o controlo de todos os acessos aos serviços, uma vez que é a plataforma de IMS que medeia todos os pedidos de serviço, apenas permitindo o acesso a provedores de serviço registados. Em contraste com os serviços comuns de Internet, as soluções IMS pretendem aumentar a sua competitividade propondo aumentar a segurança, a qualidade de serviço e a flexibilidade da infra-estrutura de taxação [58].

3.2 Arquitectura

A arquitectura IMS prevê três camadas distintas: a camada de transporte, a camada de controlo e a camada de aplicação. Estas camadas estão representadas na Figura 3.2.

A camada de transporte tem a função de homogeneizar os diversos protocolos, fluxos multimédia e interfaces da rede de acesso para um ambiente IP, normalizado para todos os tipos de utilizadores. Esta camada comporta-se como uma camada de transporte de todas os fluxos multimédia sobre a rede IP. Na camada de controlo encontramos os equipamentos

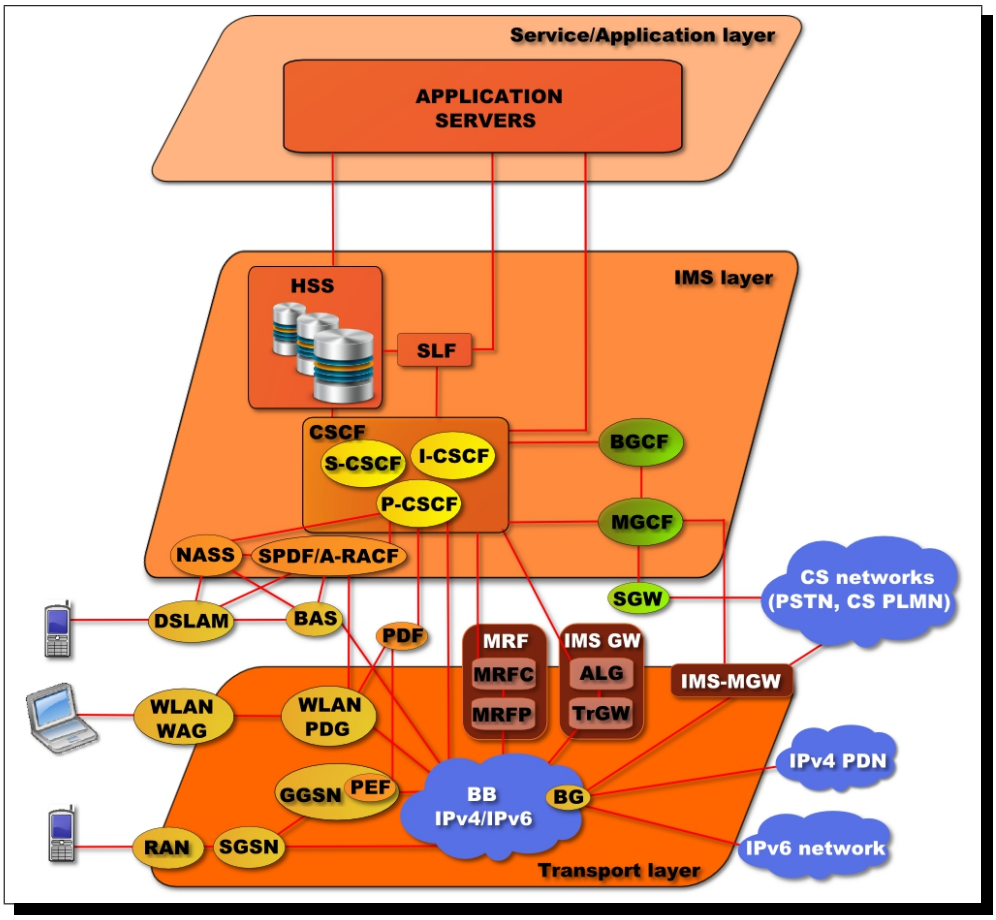

Figura 3.2: Camadas da arquitectura IMS.

que realizam o controlo dos serviços, ou seja, os elementos que através do protocolos SIP e *Diameter* [36] trocam informações com as outras duas camadas para garantir os serviços requisitados pelos utilizadores. Finalmente na camada de aplicação estão os equipamentos responsáveis pela lógica dos serviços, os *Application Server* (AS). Esta camada permite que os serviços sejam introduzidos na rede sem alterar a infra-estrutura dos outros dois planos.

A arquitectura IMS contempla a convergência das camadas de controlo e aplicação, específicas de cada rede, tornando o controlo de acesso e as aplicações transparentes à tecnologia de acesso. Esta característica proporciona um ganho em escala na aquisição e dimensionamento das plataformas da rede que podem ter as suas funcionalidades e recursos acedidos por diversos tipos de utilizadores. Com as camadas de controlo e aplicação unificadas para todos os tipos de utilizadores, é possível reduzir os investimentos e o tempo de implementação de novos serviços, pois pode-se fazer uso da infra-estrutura existente, adicionando-se à arquitectura apenas novos AS. O motivo para esta simplicidade na implantação de novos serviços é que todo o controlo das sessões é baseado em SIP, protocolo que se tornou o padrão da indústria para o controlo das sessões multimédia e dos processos de telecomunicações, permitindo a perfeita integração entre os serviços tradicionais de telecomunicações e novos serviços baseados no ambiente de Internet.

Os principais elementos da arquitectura IMS e as suas interfaces estão ilustrados na Figura

3.3.

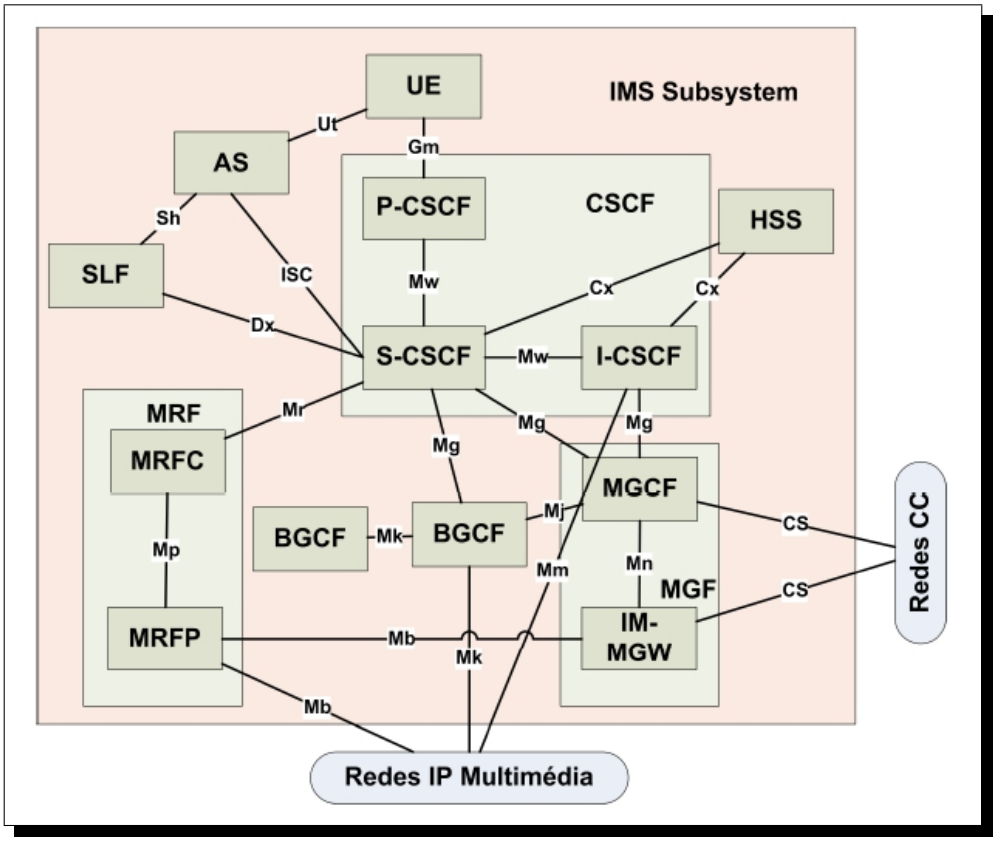

Figura 3.3: Elementos e interfaces da arquitectura IMS.

Os elementos que compõem a arquitectura podem ser agrupados nas seguintes categorias:

- Elementos de controlo IMS: *Call Session Control Funcion* (CSCF);
- Elementos de base de dados: *Home Subscriber Server* (HSS) e *Subscription locator Funcion* (SLF);
- Elementos de controlo de interoperabilidade: *Signalling Gateway* (SGW), *Media Gateway Control Function* (MGCF) e *Breakout Gateway Control Function* (BGCF);
- Elementos de serviços IMS: SIP-AS;
- Elementos de serviços externos e serviços de interoperabilidade: *Open Service Access-Service Capability Server* (OSA-SCS) e *IP Multimedia-Switching Service Function* (IM-SSF);
- Elementos de recursos: *Multimedia Resource Funcion Controller* (MRFC) e *Multimedia Resource Funcion Processor* (MRFP);
- Elementos de interoperabilidade de mídia: *IP Multimedia Subsystem-Media Gateway* (IMS-MGW);

- Elementos GPRS: *Serving GPRS Support Node* (SGSN) e *Gateway GPRS Support Node* (GGSN).

Em seguida será realizada uma breve descrição, dos elementos da arquitectura IMS e suas respectivas funções.

- **Call Session Control Funcion** - O CSCF é um servidor SIP, que processa a sinalização SIP no IMS. Este elemento estabelece, suporta, monitoriza e finaliza as sessões multimédia e gere as interacções do utilizador com o serviço. Dependendo das funcionalidades fornecidas, o CSCF é categorizado em quatro tipos de distintos: *Proxy-CSCF* (P-CSCF), *Interrogating-CSCF* (I-CSCF) e *Serving-CSCF* (S-CSCF).

- **Proxy-Call Session Control Funcion** - O P-CSCF é o primeiro ponto de contacto do equipamento do utilizador com a rede IMS. Tem a função de tratar todos os pedidos originados ou terminadas nos utilizadores IMS e direcciona-los para o elemento adequado. O P-CSCF com o qual o equipamento do utilizador troca as mensagens está sempre na rede em que o utilizador se encontra, esteja o utilizador na sua rede, ou não. Para garantir a integridade, o retorno dos pedidos dá-se sempre pelo mesmo P-CSCF que iniciou o pedido. Este elemento realiza a compressão e descompressão do protocolo SIP, para reduzir o tempo de resposta dos pedidos, pois algumas mensagens SIP são extensas, visto que o SIP é um protocolo baseado em texto. O P-CSCF desempenha funções relacionadas com a segurança, uma vez que actua no processo de registo do equipamento do utilizador. Esta entidade exerce ainda funções de policiamento, através de análise do conteúdo do *payload* do protocolo *Session Description Protocol* (SDP) para garantia de Qualidade de Serviço (QdS).

- **Interrogating-Call Session Control Funcion** - Numa rede IMS pode existir mais do que um I-CSCF, sendo este, o elemento que actua como um SIP *proxy*. O I-CSCF é a entidade que procura no HSS a informação sobre o qual o S-CSCF que atende o utilizador. Este elemento é responsável por encaminhar as consultas e ou respostas SIP da própria rede ou de redes de outras operadoras para o S-CSCF, como por exemplo as mensagens SIP do registo recebidas do P-CSCF que são encaminhadas para o S-CSCF que esta servindo o utilizador. Desta forma, o I-CSCF é o principal ponto de contacto da rede para todas as ligações IMS destinadas a um utilizador ou, para utilizadores em *roaming* que actualmente se encontram dentro da área de serviço da operadora. Opcionalmente, o I-CSCF pode cifrar partes da mensagem SIP que contenha informações sensíveis sobre o domínio, como número de servidores no domínio, os seus nomes de *Domain Name System* (DNS) ou as suas capacidades.

- **Serving-Call Session Control Funcion** - O S-CSCF é o elemento central do plano de controlo, podendo ser considerado o cérebro do IMS, que realiza o controlo do estado das sessões. A principal função deste elemento, que está sempre localizado na rede do utilizador IMS, é realizar o processo de registo do utilizador IMS. O S-CSCF tem as funcionalidades de SIP *server* e SIP *register*, e por ele passa obrigatoriamente toda a sinalização dos terminais IMS. Este elemento é responsável pelo encaminhamento das mensagens SIP, realizando a análise das mensagens SIP para determinar o elemento destino das mesmas. Uma das principais funções do S-CSCF é traduzir o número de telefone para *SIP-Universal Resource Identifier* (SIP-URI) e vice-versa. Outras funções

importantes do S-CSCF são relacionar à troca de informações com os diverso elementos da rede do utilizador, para verificar entre outras coisas quais os serviços e privilégios que o utilizador possui, realizando procuras no HSS. Este elemento actua ainda aplicando filtros ou políticas de utilizador, verificando que tipos de formatos multimédia são suportados pelo utilizador.

- **Home Subscriber Server** - O HSS é o principal repositório de dados de todos os utilizadores e serviços relacionados da rede IMS. Os principais dados armazenados no HSS incluem identidade do utilizador, informações de registo, parâmetros de acesso, informações de localização e informações relativas à triggers de serviços. O HSS armazena os dois tipos de identidade do utilizador, a identidade privada e a identidade pública. A identidade privada é a identidade do utilizador que é designada pela rede do utilizador e é utilizada nos processos de registo e autorização, enquanto a identidade pública é a identidade do utilizador que é de conhecimento dos outros utilizadores e é utilizada para realizar as chamadas para o utilizador em questão. O HSS deve ser transparente ao tipo de serviço fornecido ao utilizador, à aplicação, à rede, ao equipamento terminal do utilizador, à localização geográfica, entre outros.

- **Subscription Locator Function** - O SLF é uma base de dados adicional responsável por mapear as identidades publicas dos utilizadores e os respectivos endereços dos HSS que contém as informações dos utilizadores. É através deste elemento que o I-CSCF, o S-CSCF e o AS descobre o endereço de qual HSS mantém as informações do utilizador, nas redes que possuem mais de um HSS.

- **Media Gateway Control Function** - O MGCF é o elemento que realiza a interface entre o Serviço Fixo de Telefonia (SFT) e o IMS no plano de sinalização. É o equipamento responsável por executar a conversão dos diferentes protocolos, como por exemplo, de SIP para *ISDN User Part* (ISUP) e vice-versa. O MGCF realiza também a transcodificação dos *codecs* de áudio e vídeo, como por exemplo, de G.729 para G.711 e vice-versa. Normalmente as redes IMS possuem vários MGCF para realizar o interoperabilidade com diversos pontos das redes SFT.

- **Breakout Gateway Control Function** - O BGCF é a entidade responsável por determinar por qual BGCF a chamada sairá da rede IMS, quando a mesma for destinada a uma rede SFT. Caso o destino da chamada seja outra rede IMS, o BGCF encaminha a chamada para outro BGCF da outra rede. Esta decisão é tomada basicamente em função do endereço de destino da chamada e das regras de encaminhamento de tráfego definida pela operadora de origem.

- **Signalling Gateway** - O SGW é utilizado para interligar uma rede IMS a diferentes redes de sinalização, como redes de sinalização *Signaling System* (SS7) ou *Media Transfer Protocol* (MTP). Esta interligação dá-se ao nível do transporte, pois as sinalizações são convertidas para o transporte baseado em IP, ou seja, o *SignalingTransport* (SIGTRAN).

- **Application Server** - O AS é um nó da camada aplicacional. Possibilita a interacção com sessões SIP recebidas na camada adjacente IMS, gera pedidos SIP e envia informação de taxação para o respectivo módulo de *charging*. Para além dos serviços SIP,

os operadores podem oferecer serviços baseados em *Customized Applications for Mobile Network Enhanced Logic* (CAMEL), *Service Environment* (CSE) e *Open Service Architecture* (OSA) aos subscritores IMS.

- **Multimedia Resource Funcion** - Os elementos *Multimedia Resource Funcion* (MRF) são responsáveis pelo tratamento dos mídia e dos recursos dentro da rede IMS. O MRF é a entidade que executa actividades como executar anúncios, misturar os fluxos multimédia, realizar a transcodifição de codecs e analisar diversos tipos de médias. O MRF é dividido em duas entidades:

  - MRFC - O MRFC pode ser considerado como sendo o nó do plano de sinalização do MRFC, que actua como um agente SIP do utilizador, recebendo e interpretando os pedidos SIP do S-CSCF para controlar os recursos do MRFP.
  - MRFP - O MRFP pode ser considerado como o nó do plano de mídia do MRFC, que implementa todas as funções relacionadas com os diversos mídia, como executar anúncios, misturar fluxos de mídia e transcodificacão de codecs, conforme as instruções recebidas do MRFC.

- **IP Multimedia Subsystem-Media Gateway** - O IMS-MGW é o elemento de interface com o SFT no plano de mídia. De um lado do IMS-MGW existe o ambiente IMS com fluxos multimédia sobre o protocolo *Real-time Transport Protocol* (RTP), enquanto do outro lado, os componentes multimédia são tratados num ou mais canais *Pulse Code Modulation* (PCM) conforme o ambiente de comutação de circuitos. O controlo do IMS-MGW é feito pelo MGCF. A transcodificação de diferentes codecs entre os ambientes IMS e SFT é executado pelo IMS-MGW

Capítulo 4

Sistema desenvolvido

Este capítulo descreve o sistema desenvolvido no presente trabalho, seguindo a abordagem clássica do processo de desenvolvimento, ou modelo em cascata. Neste contexto, a evolução do trabalho, seguiu um conjunto de fases, que permitiram definir o que iria ser desenvolvido, definir como desenvolver e finalmente efectuar o desenvolvimento. O capítulo começa por apresentar uma análise de requisitos do software a desenvolver, com base nos requisitos especifica os modelos de informação e de dados e descreve a arquitectura do sistema desenvolvido através de diagramas de blocos e diagramas de sequência. É também descrito o ambiente de implementação e são apresentadas as ferramentas utilizadas para suportar o desenvolvimento.

4.1 Levantamento de requisitos

De acordo com os objectivos, neste trabalho pretende-se apresentar uma solução de gestão baseada em políticas, para gestão dos equipamentos de rede IMS. A arquitectura da solução consiste na interligação de um servidor de gestão, concretamente um servidor CIM, com os equipamentos de rede IMS recorrendo a uma implementação *open-source* do protocolo NETCONF.

4.1.1 Requisitos funcionais

Os requisitos funcionais são aqueles que descrevem o comportamento do sistema segundo a necessidade dos utilizadores, das tarefas ou das actividades [62]. Podem também ser consideradas necessidades ou interesses dos demais elementos que interajam com o sistema. Este tipo de requisito torna-se mais fácil de ser definido, uma vez que provém directamente de uma necessidade explícita.

F1 O sistema deve gerir o elemento *Policy and Charging Rules Function* (PCRF) de uma rede IMS com base em políticas;

F2 Deverá ser considerado o evento *UA_AfterMLMStartup* que determina que o gestor intermédio (PCRF ou *Resource and Admission Control Subsystem* (RACS)) arrancou;

F3 Deverá ser considerado o evento *UA_LinkThreshold* que determina que foi excedida a capacidade do link;

F4 Deverá ser considerado o evento *UA_ServiceThreshold* que determina que foi excedida a capacidade do serviço;

F5 Deverá ser considerado o evento *UA_OutOfMoney* que determina que o cliente não tem saldo;

F6 Deverá ser considerada a condição *UA_IPClassifier* relacionada com o endereço(s) IP do fluxo de dados;

F7 Deverá ser considerada a condição *UA_MPLSClassifier* relacionada com o classificador de pacotes de rede MPLS (label, etc);

F8 Deverá ser considerada a condição *UA_NullCondition* para a ausência de condição;

F9 Deverá ser considerada a condição *UA_Node* relacionada com determinado nó de rede;

F10 Deverá ser considerada a condição *UA_ServiceClass* relacionada com a classe de serviço;

F11 Deverá ser considerada a condição *UA_ATMClassifier* relacionada com o classificador de pacotes de rede *Asynchronous Transfer Mode* (ATM);

F12 Deverá ser considerada a condição *UA_MAC8021Classifier* relacionada com o classificador de pacotes de rede 802.21;

F13 Deverá ser considerada a condição *UA_DPIClassifier* relacionada com o classificador de inspecção profunda de pacotes;

F14 Deverá ser considerada a acção *UA_Shape* de shape dos pacotes do fluxo;

F15 Deverá ser considerada a acção *UA_Mark* de mark dos pacotes do fluxo;

F16 Deverá ser considerada a acção *UA_Forward* de reencaminhamento dos pacotes do fluxo;

F17 Deverá ser considerada a acção UA_FindLowerLayer que permite reencaminhar a decisão para camada inferior do gestor de políticas (PCRF é uma máquina executora de políticas genérica);

F18 Deverá ser considerada a acção *UA_ActionNull* para a ausência de acção;

F19 Deverá ser considerada a acção *UA_Police* de *police* nos pacotes do fluxo;

F20 Deverá ser considerada a acção *UA_Drop* de descarte de pacotes no fluxo;

F21 Deverá ser considerada a acção *UA_Rate* de *rate* dos pacotes do fluxo (rateia dos pacotes e descarta aleatoriamente alguns deles);

F22 Deverá ser considerada a acção *UA_QoSSessionDeterministicCAC* de controlo de admissão de chamada a uma sessão com QdS deterministico;

F23 Deverá ser considerada a acção *UA_QoSStaticCAC* de controlo de admissão de chamada com QdS estático;

F24 O servidor de gestão do sistema deve ser um servidor CIM;

F25 A interligação de um servidor de gestão com os equipamentos de rede IMS deve ser efectuado recorrendo ao protocolo NETCONF;

F26 O protocolo de transporte do NETCONF deve ser o SSH;

F27 O sistema deve suportar uma consola de gestão, que possua uma interface gráfica para a criação e edição de políticas.

4.1.2 Requisitos não funcionais

Requisitos não-funcionais são os requisitos relacionados ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenibilidade e tecnologias envolvidas. Outra definição para requisitos não-funcionais é o conjunto de requisitos que não estão directamente relacionados com as funções específicas do sistema, mas que em determinados momentos definem limitações deste [72]. Uma dificuldade para a consideração dos requisitos não-funcionais é que estes raramente estão relacionados com uma característica específica, ou com um componente do sistema, vinculando-se, na maior parte das vezes, ao sistema como um todo, podendo tornar um sistema inviável se um determinado requisito não-funcional não for atingido como, por exemplo, o desempenho.

NF1 O sistema deve correr no sistema operativo Linux;

NF2 A implementação *open-source* do servidor CIM deve ser o Open Pegasus;

NF3 A implementação *open-source* do protocolo NETCONF deve ser o Netopeer;

NF4 O desempenho do sistema, relativamente ao tráfego e sinalização gerados deve ser aceitável, assegurando a viabilidade da tecnologia para um cenário de gestão de uma rede de operador de telecomunicações.

NF5 O desempenho do sistema, relativamente ao consumo de memória e processamento deve ser aceitável, assegurando a viabilidade da tecnologia para um cenário de gestão de uma rede de operador de telecomunicações.

4.2 Especificação do Sistema de Gestão

Nesta secção é apresentada a arquitectura, o modelo de dados e o comportamento dinâmico do sistema de gestão.

4.2.1 Arquitectura do Sistema

Este projecto centra-se essencialmente no desenvolvimento de uma biblioteca de software que agrega todos os *providers* e classes relativas ao modelo de informação do sistema, definidos na Secção 4.2. A biblioteca deverá conter também, uma implementação de NETCONF *Manager* embebida para estabelecer a comunicação e configuração com os elementos a gerir. A biblioteca será posteriormente registada no CIMOM do servidor de gestão WBEM através dos ficheiros MOF descritos no Anexo A. Esta será designada por *NcPolicyProvider*.

A arquitectura da biblioteca *NcPolicyProvider* está ilustrada na Figura 4.1.

A arquitectura está dividia por camadas. Na primeira camada estão os *providers*, estes podem ser de dois tipos: *providers* de instância ou *providers* de associação, sendo que os

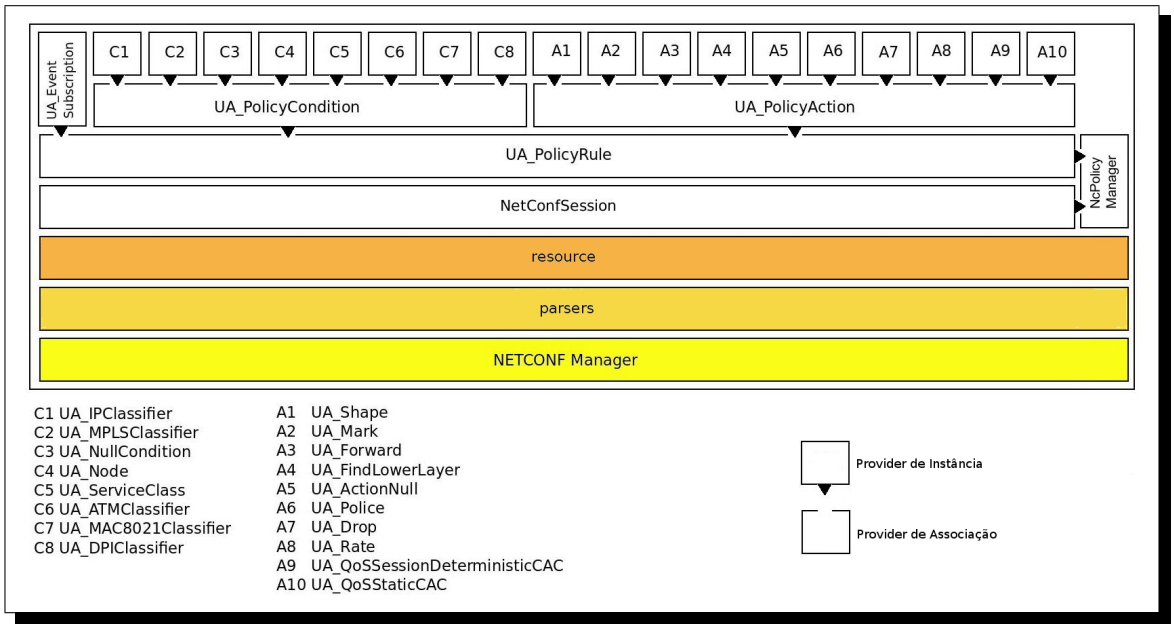

Figura 4.1: Arquitectura da biblioteca *NcPolicyProvider*.

providers de associação também são instanciados. A segunda camada, *resource*, é responsável por todo o fluxo de execução de criação e edição de políticas, fornecendo um conjunto de primitivas que serão invocadas pelos métodos intrínsecos dos *providers*. A terceira camada, *parsers*, fornece as ferramentas para a manipulação e geração do XML. A quarta e ultima camada, corresponde à implementação do NETCONF *Manager*.

Contextualizando o sistema de gestão numa perspectiva global, associada a um ambiente de produção, é apresentado na Figura 4.2 a arquitectura do sistema de gestão e o respectivo ambiente gerido.

A arquitectura de controlo de reconhecimento de serviços, que fornece aos operadores um mecanismo normalizado de controlo para QdS e tarifação, aplicável a serviços IMS, é o *Policy and Charging Control* (PCC).

O PCRF é o motor de políticas do PCC. As políticas actuam ao nível do controlo de bloqueio e do controlo de QdS. O controlo de bloqueio corresponde à capacidade de bloquear, ou de permitir pacotes IP pertencentes a um determinado fluxo IP, com base em decisões do PCRF. O PCRF pode, por exemplo, tomar decisões de bloqueio com base em eventos de sessão (*start/stop* do serviço), reportados pelo *Application Functions* (AF), através da interface *Rx*. O controlo de QdS permite que o PCRF forneça ao *Policy and Charging Enforcement Function* (PCEF) a QdS autorizada para um fluxo IP especifico [38].

O controlo de tarifação não implica apenas o controlo do acesso ao serviço mediante a gestão de crédito *online*, mas contém também importantes funcionalidades de redirecção, que são usadas por exemplo, para o aviso de tarifação e *top-up* das contas pré-pagas. A Online Charging System (OCS) poderá autorizar o acesso a serviços individuais ou a um conjunto de serviços, através da concessão de créditos para fluxos IP autorizados.

A utilização dos recursos é mapeada na forma de uma quantidade limitada de tempo, de volume de tráfego, ou de eventos tarifados. Se um utilizador não é autorizado a aceder um

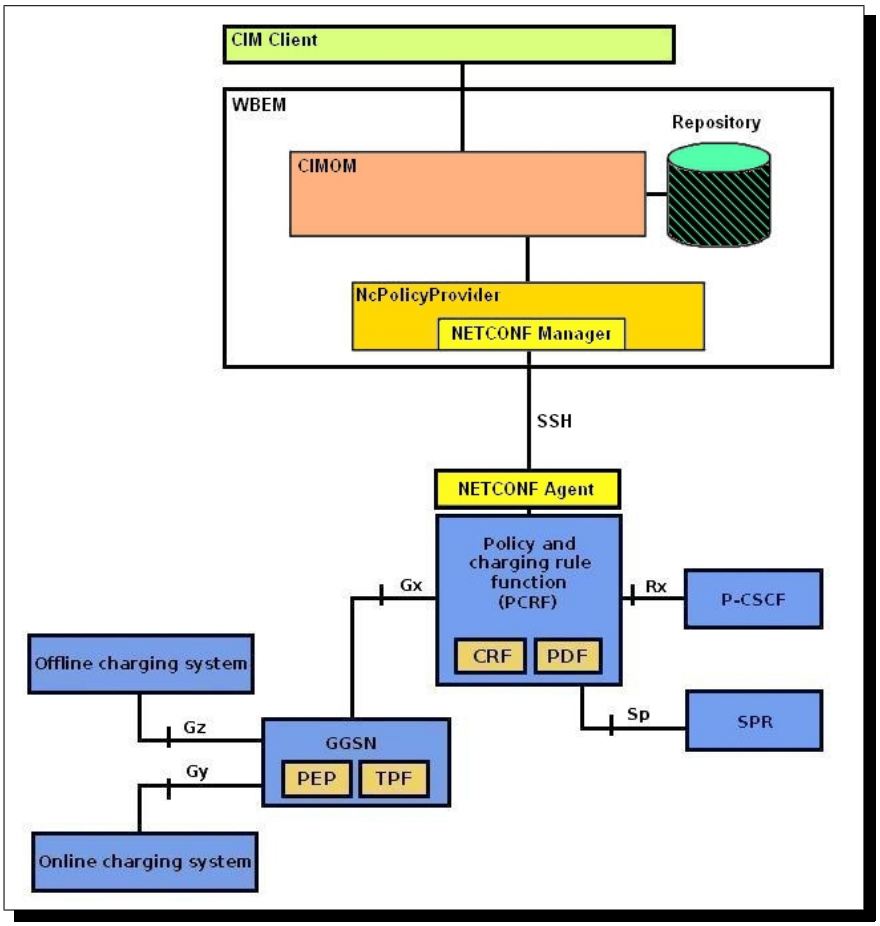

Figura 4.2: Arquitectura global do sistema de gestão.

determinado serviço, por exemplo no caso de uma conta vazia nos pré-pagos, então o OCS pode negar os pedidos de crédito, e adicionalmente, instruir o PCEF para redireccionar o serviço pedido para um destino especificado (por conta *top-up*).

O PCC também incorpora serviços baseadas em tarifação de tráfego *offline*, no entanto, esta funcionalidade não fornece qualquer meio para o controlo do acesso em si. Em vez disso, deve ser usado o controlo de políticas para restringir o acesso, e o uso de serviços específico podem ser reportados ao Offline Charging System (OFCS). O PCRF é a entidade central no PCC, responsável pelas decisões das políticas e controlo de tarifação. As decisões podem ser baseadas com base na informação proveniente de varias origens, incluindo:

- Operador de configuração no PCRF que define as políticas aplicadas a determinados serviços;
- Informação de subscrição/políticas para um determinado utilizador, recebidas do Subscription Profile Repository (SPR);
- Informação sobre o serviço recebido do AF;
- Informação da rede de acesso sobre qual a tecnologia de transporte a usar e assim por diante.

A Figura 4.2 representa o cenário de gestão, mas por uma questão de agilidade dos procedimentos de teste foi simplificado, encontrando-se agora ilustradado na 4.3.

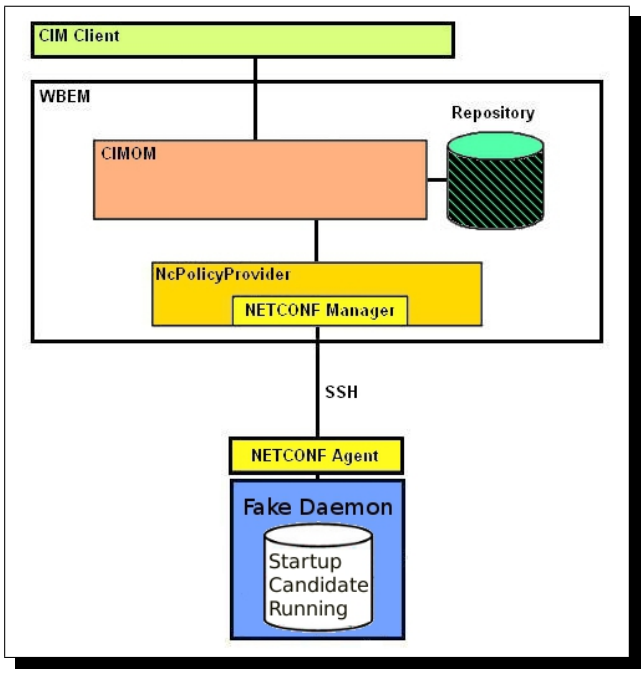

Figura 4.3: Arquitectura de teste do sistema de gestão.

No cenário de teste ilustrado Figura 4.3, o PCRF é substituído for um *fake daemon* que suporta os três repositórios de configuração, *startup*, *candidate* e *running*. Cada repositório está mapeadado num ficheiro. Como o que se pretender validar é o desempenho do sistema relativamente a sinalização e consumo de memória, o cenário de teste não impõe limitações nem condiciona os resultados.

O Netopeer implementa os repositórios através de ficheiros em disco, respeitando os fluxos de informação de acordo com a norma NETCONF. O ficheiro *startup* contém a política de arranque do serviço, o ficheiro *running* contém a política activa e o ficheiro *candidate* contem a política com as alterações que irão ser submetidas ao estado activo.

4.2.2 Modelo de Dados

O primeiro passo da especificação do sistema consiste na definição do modelo de informação do sistema (*Information Model* (IM)), organizando toda a informação relativa aos elementos que constituem as políticas: os eventos, as condições e as acções. De acordo com o requisito F24, o servidor de gestão do sistema deve ser um servidor CIM. Como o modelo de informação CIM contempla a definição de classes de políticas, as classes dos elementos que constituem as políticas do sistema serão derivadas do modelo abstracto CIM. A versão da especificação CIM considerada para a definição do modelo de informação do sistema foi a 2.22.0 [7].

Na Figura 4.4 encontra-se o diagrama de classes das condições e acções definidas nos requisitos F6 a F23.

Foram consideradas classes intermédias tanto para as condições como para as acções, *UA_Condition* e *UA_Action* respectivamente. Estas classes contém as propriedades comuns

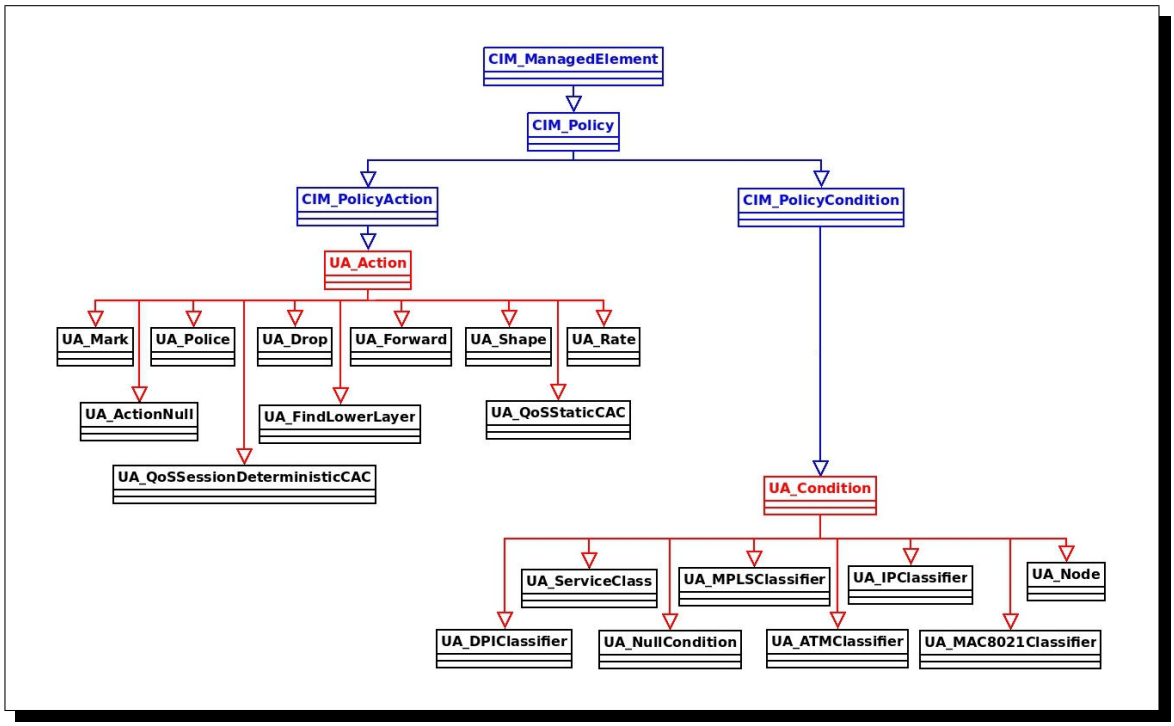

Figura 4.4: Diagrama de classes das condições e acções.

associadas aos requisitos F6 a F23 e que não se encontram definidas no modelo abstracto CIM. As classes com o prefixo *"CIM_"* dizem respeito ao modelo abstracto CIM, por sua vez as classe com o prefixo *"UA_"* são especificas desta implementação. Na Figura 4.5 encontra-se o diagrama de classes dos eventos definidos no Requisito F2 a F5.

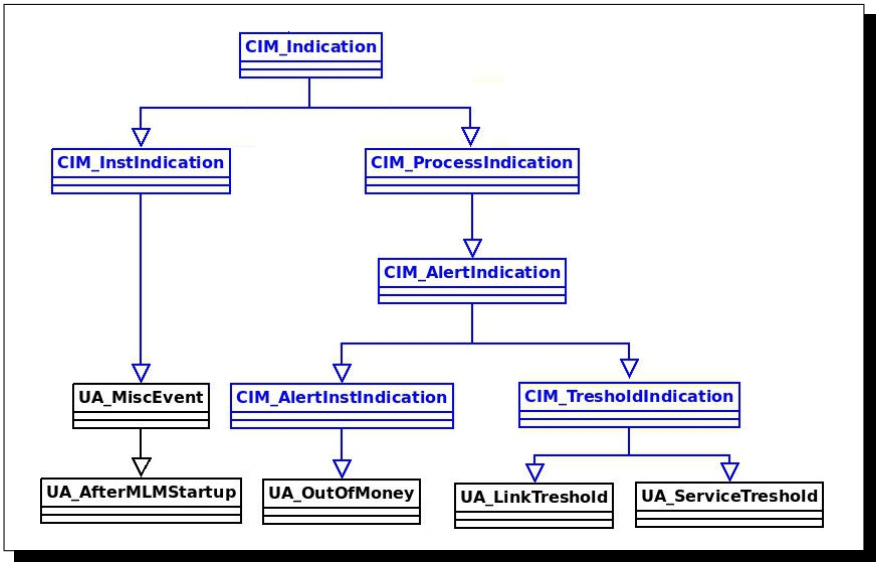

Figura 4.5: Diagrama de classes dos eventos.

De acordo com a Figura 4.5, os eventos definidos no Requisito F2 a F5 derivam das classes

CIM_InstIndication e *CIM_ProcessIndication*. Do ponto de vista da definição do sistema de informação não é possível considerar as classes dos eventos explicitamente para efeitos de inclusão na política, pois estas são instanciadas posteriormente à definição da regra, quando os eventos ocorrem. No momento de definição da regra existe somente a subscrição do tipo de evento. Desta forma, para o modelo de informação do sistema irá ser considerada uma classe de subscrição, *UA_EventSubscription*.

Na Figura 4.6 encontra-se o diagrama de classes para a subscrição de eventos.

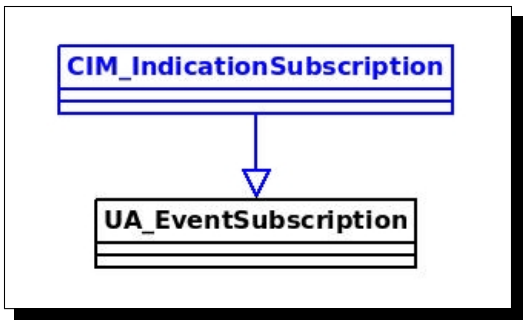

Figura 4.6: Diagrama de classes da subscrição.

A classe *UA_EventSubscription* deriva da classe *CIM_IndicationSubscription* e contém apenas uma propriedade *EventID*, que é um identificador do tipo de evento. A parte da subscrição do esquema de eventos define como é que os clientes CIM devem subscrever a recepção de indicações. Uma aplicação cliente CIM activa a subscrição criando uma instância de *CIM_IndicationSubscrition*, que por sua vez define uma associação entre uma instância *CIM_IndicationFilter* e uma instância do tipo *CIM_IndicationHandler*.

A instância *CIM_IndicationFilter* descreve o tipo de indicação desejada. A instância *CIM_IndicationHandler* descreve os elementos do sistema encarregues de tratar a indicação. A Figura 4.7 ilustra o fluxo da indicação.

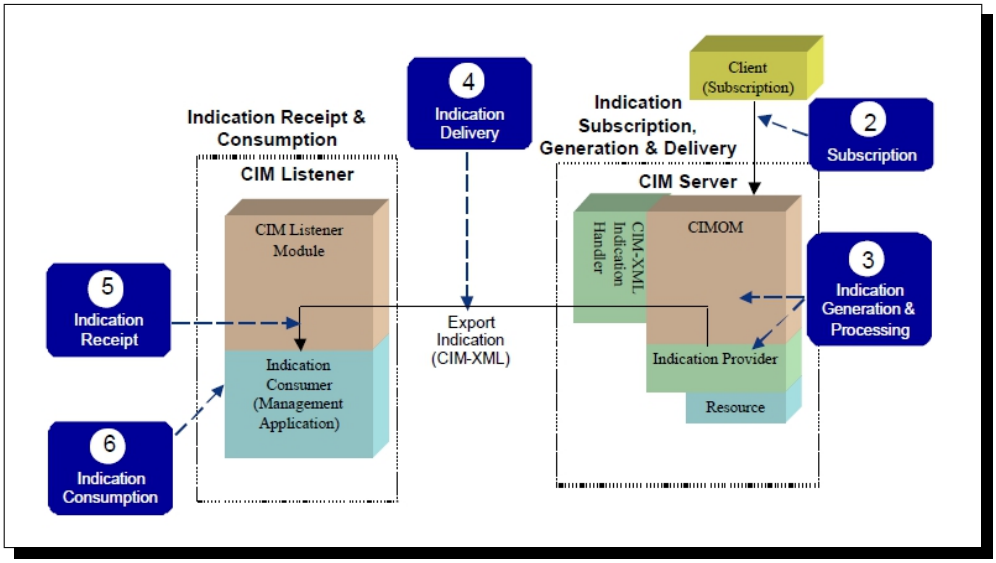

Figura 4.7: Fluxo da indicação.

As indicações são enviados se existir alguma instância do tipo *CIM_IndicationSubscription* que tenha solicitado a entrega da indicação. Ou seja, a indicação corresponde à condição definida pela instância *CIM_IndicationFilter* e é despachada somente para o destino definido pela instância *CIM_IndicationHandler*.

4.2.3 Definição dos providers

O CIM recorre aos conceitos orientados a objectos para modelar a informação: as informações de gestão são expressas usando definições de classe, de herança, instâncias, definindo propriedades e métodos. As classes CIM são definidas em ficheiros de texto usando a linguagem MOF. Uma vez definido o modelo de informação do sistema através de diagramas de classes UML, a conversão para MOF é imediata.

A abordagem seguida na implementação do sistema relativamente à definição dos *providers* consiste na criação de um *provider* de instância elementar por cada condição ou acção. Como uma política pode ser definida por uma ou mais condições, e contendo uma ou mais acções, é criado um *provider* de associação para as condições e outro para as acções.

Na Figura 4.8 encontra-se a definição da classe de associação de condições, designada a partir de agora por *UA_PolicyCondition*. Apenas a propriedade *ConditionID* é mandatória, porque do ponto de vista da instanciação tem que existir pelo menos uma propriedade mandatória, as restantes condições associadas são opcionais de modo a garantir a flexibilidade na definição da política.

```
1  [Association]
2  class UA_PolicyCondition
3  {
4      [key] uint32            ConditionID;
5
6      UA_IPClassifier         REF IPClassifier;
7      UA_MPLSClassifier       REF MPLSClassifier;
8      UA_NullCondition        REF NullCondition;
9      UA_Node                 REF Node;
10     UA_ServiceClass         REF ServiceClass;
11     UA_ATMClassifier        REF ATMClassifier;
12     UA_MAC8021Classifier    REF MAC8021Classifier;
13     UA_DPIClassifier        REF DPIClassifier;
14 };
```

Figura 4.8: Definição da classe de associação de condições.

Na Figura 4.9 encontra-se a definição da classe de associação de acções, designada a partir de agora por *UA_PolicyAction*. Pela mesma razão da classe *UA_PolicyCondition* também aqui apenas a propriedade *ActionID* é mandatória.

Um política consiste na associação de uma instância de subscrição, de uma associação de condições e de uma associação de acções. Na Figura 4.10 encontra-se a definição da classe de política do sistema, designada a partir de agora por *UA_PolicyRule*.

De acordo com o Requisito F25, a comunicação deve ser efectuada recorrendo ao protocolo NETCONF. O Requisito F26 especifica que o protocolo de transporte do NETCONF deve ser o SSH. É necessário então definir uma classe com as propriedades para o estabelecimento de uma sessão NETCONF em que o protocolo de transporte é o SSH. Esta classe quando

```
[Association]
class UA_PolicyAction
{
    [key] uint32                    ActionID;

    UA_Shape                        REF Shape;
    UA_Mark                         REF Mark;
    UA_Forward                      REF Forward;
    UA_FindLowerLayer               REF FindLowerLayer;
    UA_ActionNull                   REF ActionNull;
    UA_Police                       REF Police;
    UA_Drop                         REF Drop;
    UA_Rate                         REF Rate;
    UA_QoSSessionDeterministicCAC   REF QoSSessionDet;
    UA_QoSStaticCAC                 REF QoSStaticCAC;
};
```

Figura 4.9: Definição da classe de associação de acções.

```
[Association]
class UA_PolicyRule
{
    [key] uint32            RuleID;

    UA_EventSubscription    REF EventSubscription;
    UA_PolicyCondition      REF Condition;
    UA_PolicyAction         REF Action;
};
```

Figura 4.10: Definição da classe de política.

instanciada será também responsável por definir o tipo de operação, leitura ou criação de política.

Na Figura 4.11 encontra-se a definição da classe de sessão NETCONF e a definição do tipo de operação a executar, designada a partir de agora por *NetConfSession*.

```
class NetConfSession
{
    [key] uint32 SessionID;
    [key] string AgentHost;
    [key] uint32 AgentPort;
    [key] string User;
    [key] string Password;
    [key] boolean CreatePolicy;
};
```

Figura 4.11: Definição da classe de sessão NETCONF.

Finalmente é necessário fazer a associação entre a política e a sessão NETCONF para a sua criação ou edição.

Na Figura 4.12 encontra-se a definição da classe de gestão que associa a política a uma sessão NETCONF, designada a partir de agora por *NcPolicyManager*.

```
[Association]
class NcPolicyManager
{
    [key] NetConfSession    REF Session;
    [key] UA_PolicyRule     REF Policy;
};
```

Figura 4.12: Definição da classe de associação de gestão.

No Anexo A encontra-se os dois ficheiros MOF, *policy.mof* e *repository.mof*, resultantes do processo de especificação, que servem de base para a criação dos *providers*.

4.2.4 Definição do Modelo de Dados da política

O modelo de dados da política define o mapeamento do modelo de informação da política numa sintaxe que seja interpretável pelo sistema.

De acordo com o Requisito F25, a comunicação deve ser efectuada recorrendo ao protocolo NETCONF. Tal como mencionado anteriormente na Secção 2.4, a especificação NETCONF não define o Modelo de Dados, no entanto, recentemente o YANG tem sido proposto como um sério candidato. Por estas razões, e pela sua simplicidade e flexibilidade, o YANG [61] foi adoptado como modelo de dados do sistema.

No Anexo B encontra-se definido o Modelo de Dados da política do sistema, baseado na especificação CIM 2.22.0 [7].

4.3 Comportamento Dinâmico

Esta secção descreve o comportamento dinâmico do sistema de gestão para os processos de criação e edição de política.

4.3.1 Processo de criação de política

O processo de criação de um nova política, que irá sobrepor a política *running* no elemento a gerir, consiste na criação, a partir do cliente CIM, de uma instância *UA_PolicyRule* e respectivas referências, de acordo com a política pretendida.

Em seguida é necessário criar uma instância do *provider NetConfSession* com a propriedade ***CreatePolicy=true***.

Finalmente é criada uma instância do *provider NcPolicyManager* que associa a política presente na instância *UA_PolicyRule* com uma sessão NETCONF configurada para criação na instância *NetConfSession*.

A Figura 4.13 ilustra a criação de instância do *provider NetConfSession* para a criação de um nova política.

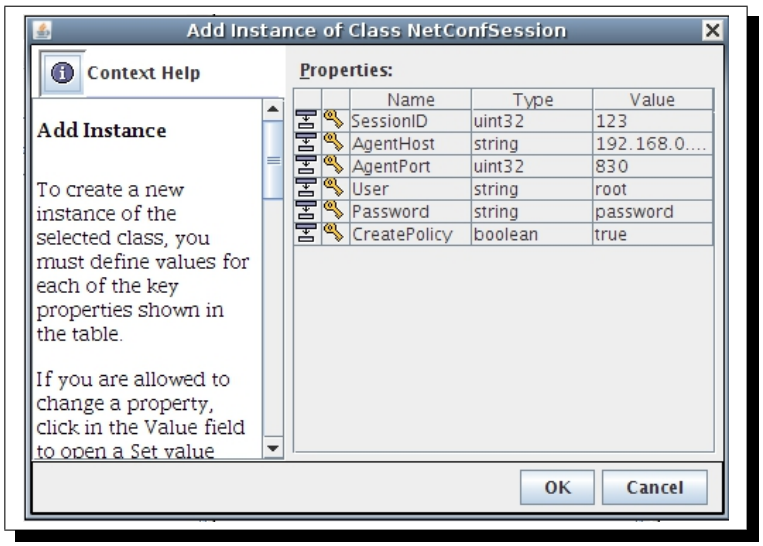

Figura 4.13: Instância do *provider NetConfSession* para criação de política.

Depois de criadas as instâncias *UA_PolicyRule* e *NetConfSession*, é necessário criar uma instância do *provider NcPolicyManager* com as referências das instâncias criadas anteriormente.

A Figura 4.14 ilustra a criação de instância do *provider NcPolicyManager*.

O método *create_instance* despoleta o processo de submissão da nova política, que irá substituir a política *running* no elemento a gerir. A operação NETCONF invocada é o *EDIT-CONFIG*. No lado do NETCONF *Agent* é efectuado o *replace* com as alterações submetidas. Como o NETCONF *Agent* do elemento gerido suporta a capacidade **:candidate**, significa que a política é primeiro transferida para o repositório *candidate* do lado do NETCONF *Agent*.

O processo de submissão da na nova política do ponto de vista NETCONF, implica a sequência de operações, *LOCK*, *EDIT-CONFIG*, *COMMIT* e *UNLOCK*. Após o *COMMIT* a política passa ao estado activo.

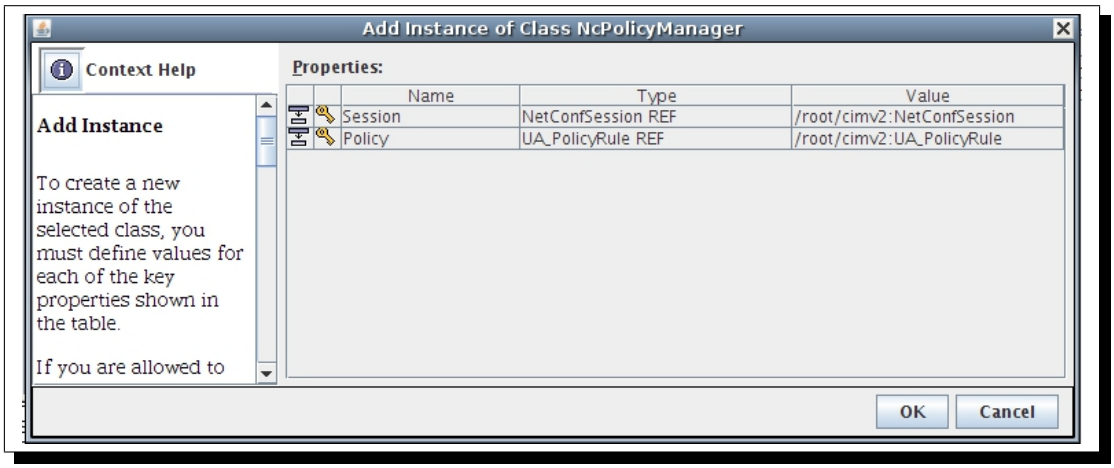

Figura 4.14: Criaçao de instância do *provider NcPolicyManager*.

A Figura 4.15 ilustra o mecanismo de criação de nova política activa, despoletado pela criação de instância do *provider NcPolicyManager*, através de um diagrama de sequência.

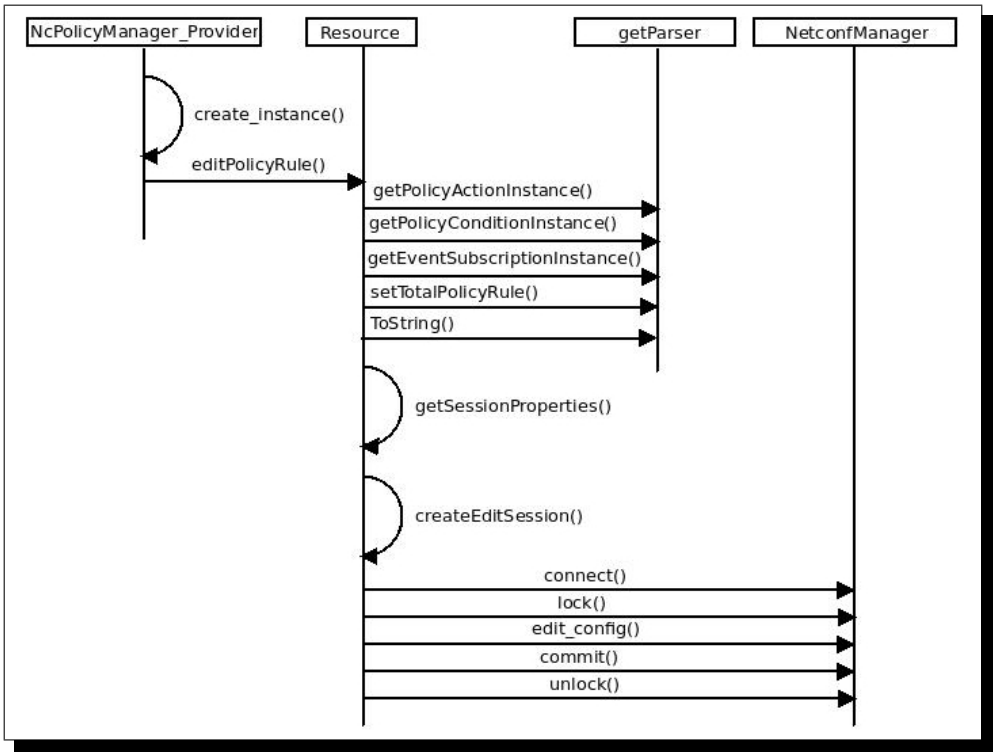

Figura 4.15: Diagrama de criação da instância do *provider NcPolicyManager*.

4.3.2 Processo de edição política

O processo de edição política pressupõe a sequência de três passos. Primeiro é necessário obter a configuração *running* do elemento gerido, depois é necessário editar a política no editor e finalmente submeter as alterações efectuadas. A solicitação por parte do servidor de gestão, da configuração associada a base de dados *running* do elemento gerido, acontece quando o *provider NetConfSession* é instanciado com a propriedade ***CreatePolicy=false***;

A Figura 4.16 ilustra a criação de instância do *provider NetConfSession* para a obtenção da política actualmente activa no elemento gerido.

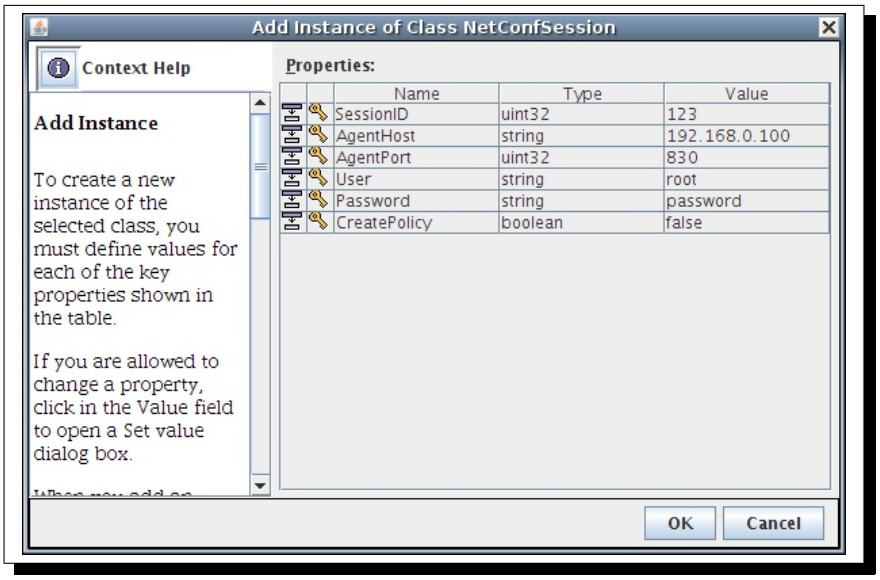

Figura 4.16: Instância do *provider NetConfSession* para edição de política.

Ao ser criada uma instância, o método *create_instance* irá pedir ao *resource* para criar uma sessão NETCONF, que por sua vez vai solicitar ao NETCONF *Manager* um pedido *GET-CONFIG*. Se tudo correr bem, o NETCONF *Manager* recebe a configuração enviada pelo NETCONF *Agent* e invoca o *parser*. O *parser*, com base na configuração recebida pelo NETCONF *Manager*, instância todos os *providers* que definem a política.

A Figura 4.17 ilustra o mecanismo de obtenção da política activa, despoletado pela criação de instância do *provider NetConfSession*, através de um diagrama de sequência.

Depois de obtida e editada política, o passo seguinte é a submissão das alterações efectuadas. No caso da edição, a política não é submetida na sua totalidade, apenas a componente da política (subscrição, condição ou acção), associada à alteração em causa. No lado do NETCONF *Agent* é efectuado o *merge* com as alterações submetidas. O processo de submissão da política alterada do ponto de vista NETCONF é igual ao processo de criação de política.

O processo de submissão ocorre quando a instância do *provider* de subscrição, ou um qualquer *provider* de condição ou acção, associados uma sessão NETCONF, são editados. O método modify_instance do *provider* editado, desencadeia o processo.

A edição de um *provider* tem de ser feita obrigatoriamente na própria instância e não através de um *provider* de associação que contenha uma referência para o *provider* que se pretende editar. A edição de uma referência dentro de um *provider* de associação não desencadeia a invocação do método modify_instance do *provider* referenciado.

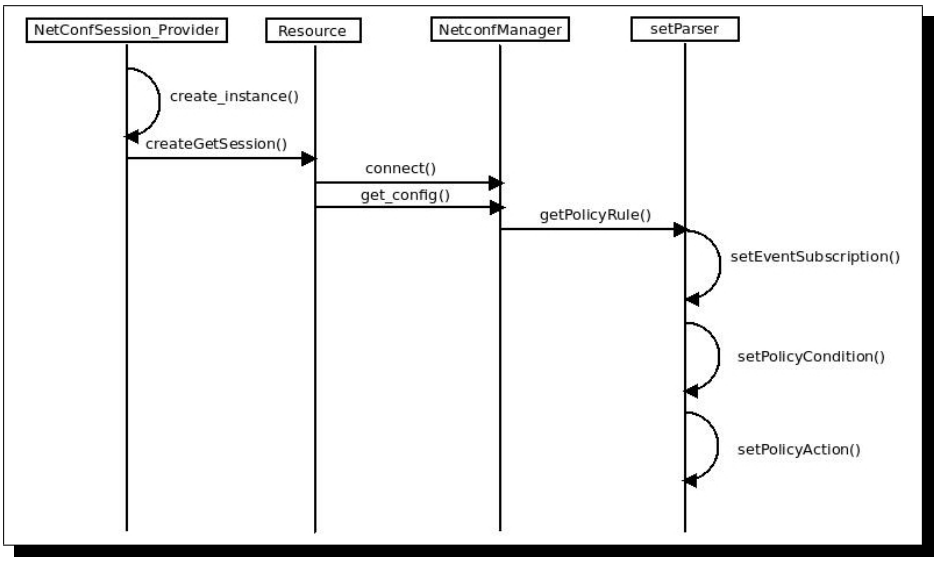

Figura 4.17: Diagrama de sequência da criação de instância do *provider NetConfSession*.

A Figura 4.18 ilustra o mecanismo de submissão da política editada, despoletado pela edição da instância do *provider UA_Drop*, através de um diagrama de sequência.

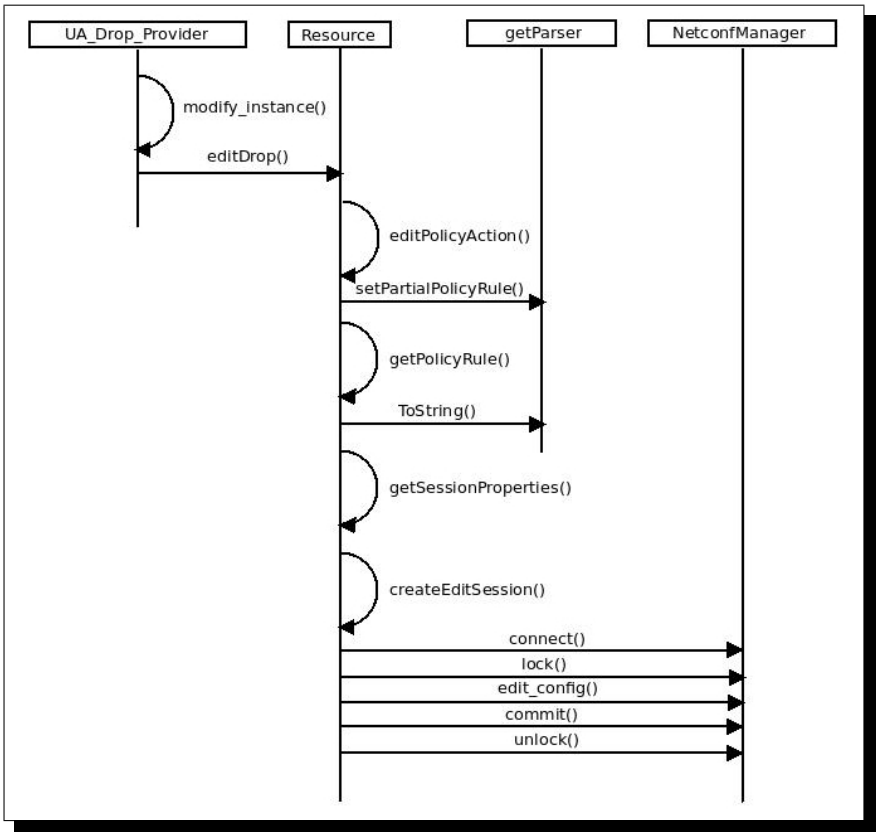

Figura 4.18: Diagrama de edição da instância do *provider UA_Drop*.

4.4 Implementação

O desenvolvimento do componente descrito nas secções anteriores foi efectuado em C++ com recurso ao ambiente de desenvolvimento Eclipse CDT [15]. O sistema proposto consiste na integração de duas implementações existentes de protocolos de gestão, WBEM e NETCONF, através do desenvolvimento de uma biblioteca de *providers* que contém internamente um NETCONF *Manager* responsável pela integração de tecnologias.

A implementação WBEM usada foi o Open Pegasus [24], pois é *open-source* e até à data de conclusão deste projecto trata-se de uma iniciativa activa e com desenvolvimentos recentes, ao contrário do OpenWBEM [25]. O Open Pegasus é altamente configurável, fornece alguma documentação e dispõe de uma *mailing list* com bastantes subscritores devido a popularidade da iniciativa.

A implementação NETCONF usada foi o Netopeer [22], pois é *open-source*, é escrita em C, o protocolo de transporte é o SSH e corresponde a uma implementação bastante completa da norma. Relativamente ao suporte, o seu autor Radek Krejci, revelou-se extremamente disponível.

De modo a simplificar os mecanismos associados à estrutura, instalação e registo de *providers* recorreu-se a uma ferramenta de desenvolvimento de *providers*, o CIMPLE [27], que permite a geração do esqueleto dos *providers* a partir do ficheiro MOF, assim como automatizar o processo de instalação e registo. No Anexo C encontra-se a Makefile da biblioteca NcPolicyProvider que permite, com um único comando (*make build*), compilar, instalar e registar os *providers*.

Na Secção 4.2 foi definido o modelo de dados YANG da política, no entanto para a transmissão da política sobre SSH, esta tem de estar descrita num formato baseado em XML. As ferramentas Yuma Tools [20] foram utilizadas para converter o modulo YANG da política numa estrutura XML para o transporte das políticas. O *parsing* XML foi efectuado recorrendo as bibliotecas *xerces-c* [33];

4.4.1 Implementação WBEM

O Open Pegasus corresponde a uma implementação WBEM *open-source*, desenvolvida de acordo com as normas definidas pelo DMTF, e publicada pelo Open Group [23]. O Open Group é um consórcio independente de tecnologias e fabricantes, cujo principal objectivo é o desenvolvimento de iniciativas que visam essencialmente o acesso integrado à informação dentro e entre organizações baseado em normas, de forma a promover a interoperabilidade global entre equipamentos.

Foi projectado para ser portátil e altamente modular. É codificado em C++ para que converta eficazmente os conceitos OO dos objectos CIM para um modelo de programação, garantindo o desempenho e a eficiência de uma linguagem compilada. O Pegasus foi projectado para ser nativamente portátil e actualmente compila e corre na maioria dos sistemas operativos, UNIX (R), Linux, OpenVMS e Microsoft Windows.

No Anexo D encontra-se o manual de instalação do Open Pegasus.

4.4.2 Implementação NETCONF

O sistema Netopeer é uma implementação *open-source* do protocolo NETCONF sobre SSH, destinado à configuração remota e gestão segura dos dispositivos de rede.

O sistema Netopeer resultou do projecto Liberouter [26], devido à necessidade em possuir um sistema de gestão que permitisse uma configuração remota e segura dos seus dispositivos de rede especializada, FlowMon. A descrição deste sistema foi apresentada na Conferência CESNET 2008 [4].

O projecto Netopeer esta alojado no Google Code, representa o núcleo do sistema Netopeer original usado para a configuração da sonda FlowMon. Este contém o sistema de execução do protocolo NETCONF e um serviço de configuração simples de dispositivos para demonstrar a ligação entre o núcleo do sistema Netopeer e o serviço de configuração.

O núcleo do sistema é composto por aplicações cliente-servidor, proporcionando operações NETCONF em repositórios de configuração colocadas no lado do servidor.

No Anexo F encontra-se o manual de instalação do Netopeer.

4.4.3 Ferramentas de suporte ao desenvolvimento

CIMPLE

- O CIMPLE é um ambiente *open-source* para o desenvolvimento de *providers* CIM que são compatíveis com várias implementações de servidores CIM, incluindo: OpenGroup CMPI Specification Version; OpenPegasus C++ Provider Interface; OpenWBEM C++ Provider Interface; Microsoft WMI Provider Interface;
- Permite que os *providers* desenvolvido sejam compatíveis com várias implementações de servidores CIM e simultaneamente com vários sistemas operativos. Ao contrário das interfaces dos *providers* tradicionais, o CIMPLE traduz as classes CIM em classes C++ concretas. Estas reduzem substancialmente a complexidade do código e melhoram a as segurança dos tipos.
- Oferece quatro grandes simplificações sobre o desenvolvimento de *providers* em tecnologias convencionais: classes concretas; geração do esqueleto do *provider*; geração de *stubs* de método extrínsecos; redução de operação do *provider*; registo automático do *provider*.

No Anexo E encontra-se o manual de instalação do CIMPLE.

Yuma Tools

Yuma Tools corresponde a um conjunto de ferramentas para o desenvolvimento de clientes e servidores NETCONF sobre SSH baseadas em YANG. O servidor netconfd inclui um sistema central automatizado do protocolo NETCONF, baseado directamente em módulos YANG. As ferramentas de desenvolvimento *yangdump* e *yangdiff* também estão incluídos, para recolher e processar os módulos YANG.

4.5 Sumário

Um processo de desenvolvimento de software corresponde a um conjunto de tarefas, que visam a obtenção de um produto de software. Sendo este um trabalho de investigação, o resultado não é um produto, mas sim um prova de conceito, no entanto, por uma questão de método, os conceitos associados são aplicáveis, pelo menos na abordagem clássica, ou processo em cascata.

A primeira tarefa consiste no levantamento de requisitos. Neste caso, e de uma forma geral, pretende-se a criação de um sistema para a gestão dos elementos de uma rede IMS, com base em políticas e assente na integração de duas tecnologias: WBEM e NETCONF.

A tarefa seguinte, é a especificação do sistema. Nesta tarefa foi definido o modelo de informação do sistema, ou seja toda a informação que permite descrever o cenário e as opções de gestão, em termos de eventos, condições e acções, como elementos de uma política. O modelo de informação foi derivado do modelo CIM, e com base nas acções, condições o acções foram definidos os *providers*. O modelo de informação tem que ser mapeado num modelo de dados, interpretável e processável pelo sistema de gestão. Foi definido o modelo de dados YANG para o NETCONF, na perspectiva de usufruir das vantagens da linguagem YANG e seguir a tendência actual da sua adopção.

Depois da especificação, a tarefa seguinte é a arquitectura do sistema. Nesta tarefa foi definida a estrutura da biblioteca desenvolvida e descrito o processo de criação e edição de uma política. Foi também contextualizado o sistema de gestão num cenário real e demonstrado o cenário de teste.

Finalmente na parte da implementação, foi descrito o ambiente de desenvolvimento, sendo o desenvolvimento da biblioteca de *providers* implementada em C++ no ambiente de desenvolvimento Eclipse CDT. Foram também apresentadas implementações *open-source* dos protocolos de gestão, OpenPegasus e Netopeer, para WBEM e NETCONF respectivamente. Foi tambem apresentado o CIMPLE, como ferramenta de suporte ao desenvolvimento da biblioteca, permitindo a criação do esqueleto dos *providers* a partir dos ficheiros MOF, e facilitando a instalação e registo dos *providers*, tudo na mesma *Makefile* do projecto.

Capítulo 5

Análise do sistema

O presente capítulo descreve o cenário, e apresenta os casos de teste que serviram para proceder à análise do sistema proposto. São apresentados e discutidos os resultados obtidos, relativamente ao tráfego e consumo de memória para as duas tecnologias, e é efectuada a análise global do resultados. Por fim, o sistema é submetido a análise, nas vertentes de *profiling* e resposta a testes de carga.

5.1 Plataforma de testes

De forma a avaliar o desempenho do sistema de acordo com os critérios estipulados, foi definido o cenário de teste ilustrado na Figura 5.1. Os elementos do sistema de gestão foram distribuídos por três maquinas. Na maquina onde corre o servidor CIM foi instalado o analisador de tráfego Wireshark [66].

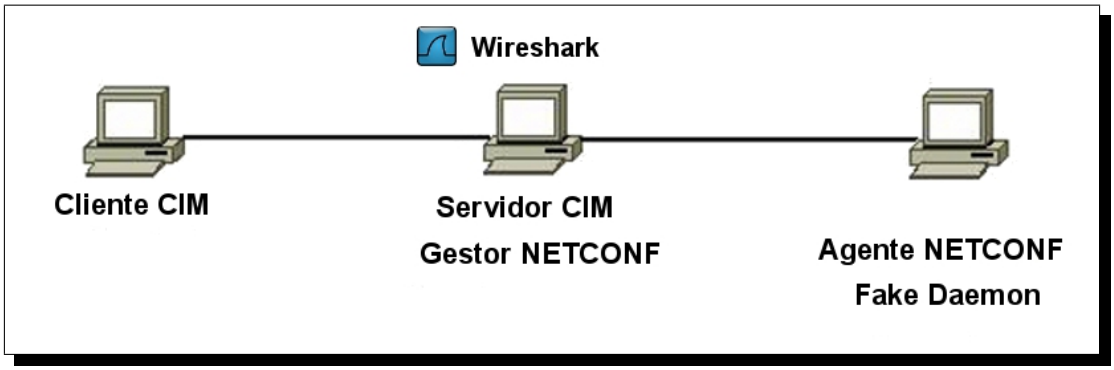

Figura 5.1: Cenário de teste.

Foram utilizados dois clientes CIM: o *WBEM Services* [31] e o *CimNavigator* [5]. O recurso a dois clientes CIM deve-se ao facto de o *WBEM Services* não criar os *providers* de associação correctamente, criando referências nulas. O *WBEM Services* foi usado sempre que possível, pois é *open-source* e é mais fácil de utilizar. O *CimNavigator*, por sua vez é licenciado, mas a sua versão não registada é completamente funcional, apresentando apenas uns *popups* a solicitar o seu registo, aumentado a sua cadência de aparecimento com o decorrer da utilização.

Foram considerados três casos de teste para a análise do sistema:

1. Ler a política *running* do *fake-daemon*. Depois de obtida a política, percorrer com o cliente CIM todas as classes e subclasses que compõem a política, *UA_PolicyRule*;

2. Editar a propriedade "*Caption*" da classe *UA_ActionNull* e submeter as alterações;

3. Criar uma política de raíz com todas as acções e condições possíveis.

Para cada um dos casos em execução foram efectuadas capturas de tráfego e analisados os consumos de memória para cada componente. Na apresentação dos resultados considerou-se que uma mensagem corresponde ao pedido é a respectiva resposta.

5.2 WBEM

Após a execução dos três casos de teste, o tráfego WBEM obtido pelas capturas foi filtrado, analisado e quantificado. A Tabela 5.1 contém para cada teste, a quantidade de tráfego discriminada por cada componente, bem como o seu peso relativo.

Tabela 5.1: Tráfego WBEM.

Operações WBEM								
	Estabelecimento Sessão		**Leitura Política**		**Edição Política**		**Criação Política**	
Nº Pacotes	8		417		10		601	
Nº Mensagens	1		102		2		45	
Tempo (s)	0,036		54,425		0,544		1053,197	
Componente	**bytes**	**%**	**bytes**	**%**	**bytes**	**%**	**bytes**	**%**
CIM-XML	0	0	211210	69,68	4255	73,07	106355	61,33
HTTP	809	60,51	64367	21,24	908	15,59	27399	15,8
TCP	256	19,15	13344	4,4	320	5,5	19232	11,09
IP	160	11,97	8340	2,75	200	3,43	12020	6,93
Ethernet	112	8,38	5838	1,93	140	2,4	8414	4,85
Total	1337	100	303099	100	5823	100	173420	100

Na Figura 5.2 encontra-se ilustrado graficamente a contribuição de cada componente de tráfego por operação.

O estabelecimento de sessão corresponde ao processo de ligação e autenticação do cliente *WBEM Services* ao servidor WBEM. Os valores obtidos correspondem apenas à criação da sessão, excluindo a enumeração de *qualifiers* e instâncias do servidor.

Iniciando a análise dos resultados pela operação de leitura da política, foram necessários 417 pacotes, 102 mensagens trocadas e 54,425 segundos para se poder visualizar a totalidade da política no cliente CIM. Este valores devem-se à natureza da tecnologia WBEM, em que a política é composta por uma série de classes e associações, tal como, definidas na Secção 4.2 do Capitulo 4. Para se poder visualizar a política na sua totalidade, é necessário percorrer com o cliente CIM cada uma das classes que compõem a política de forma independente. Visualizar uma classe com o cliente CIM implica pelo menos três pedidos ao servidor CIM: *GetClass*, *EnumerateInstanceNames* e *GetInstance*. Da análise das mensagens, verifica-se que por vezes existem pedidos que se repetem, o que totaliza 102 mensagens necessárias para a obtenção da totalidade da política. Relativamente ao tempo, o facto de a visualização da política implicar

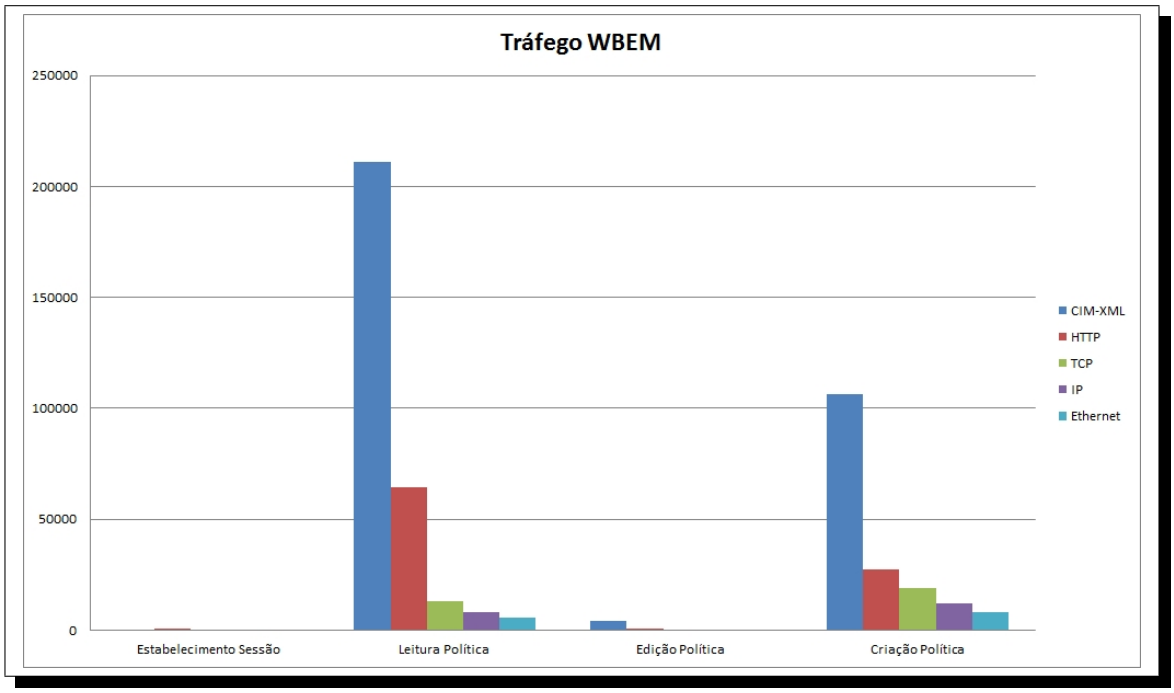

Figura 5.2: Gráfico do tráfego WBEM.

a visualização independente de cada uma das classes que a compõe, irá condicionar o tempo, que dependerá da velocidade do administrador de rede a abrir cada umas das classes, no caso do cliente *WBEM Services*, não concebido para este tipo de tarefa. Para além da grande troca de mensagens entre o cliente e o servidor CIM, o tráfego gerado é acentuado pelo transporte HTTP, que apresenta um peso superior a 20% de todo o tráfego gerado.

A edição da política, é o teste mais comedido relativamente as necessidades tráfego, apenas 10 pacotes e 2 mensagens trocadas. Neste caso a grande fragmentação da política em classes, pode ser uma vantagem, pois apenas é necessário obter com o cliente CIM, a classe pertencente à política que contém as propriedade que se pretende editar, neste caso a propriedade "*Caption*" da classe *UA_ActionNull*.

Para a criação da política, foram necessários 601 pacotes, 45 mensagens e 1053,197 segundos. Pela mesma razão da grande quantidade de tempo e tráfego necessário para leitura da política, também a criação da política implica a criação de cada classe separadamente. O facto de este teste apresentar mais pacotes e paradoxalmente menos informação, um pouco mais de metade relativamente à leitura, deve-se ao facto de a criação de classes implicar menor troca de mensagens entre o cliente e o servidor CIM, 45 *versus* 102. Relativamente ao tempo, este disparou significativamente, pois o processo de criação é bastante moroso, e neste caso foi criada um política com todas as suas possíveis componentes, acções e condições. Para este operação justifica-se claramente a necessidade de um cliente CIM, cuja interface gráfica se encontre concebida para este tipo de tarefa, camuflando a granularidade da operação em causa.

No que diz respeito as necessidades de memória do WBEM, estas foram analisadas com a execução dos testes de leitura e edição. A Figura 5.3 ilustra as necessidades de memória e os picos de processamento registados.

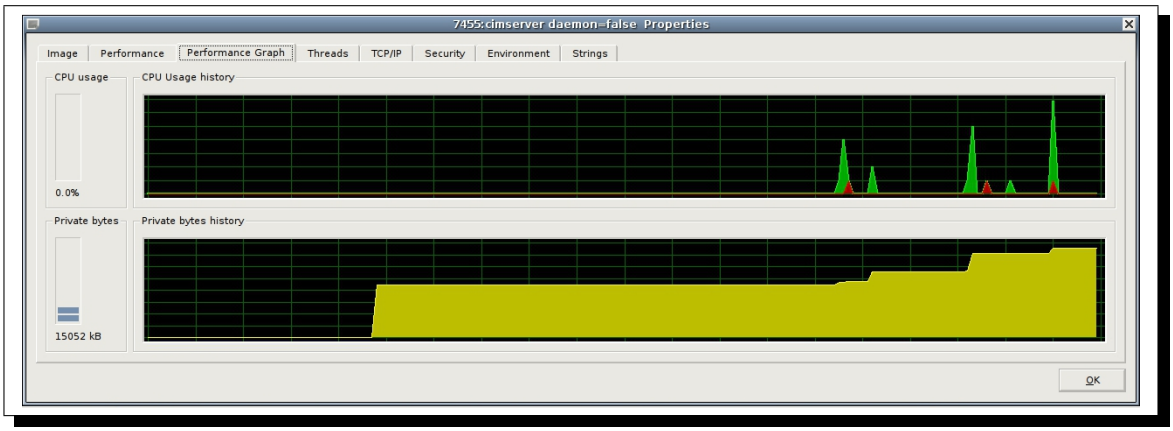

Figura 5.3: Consumo de memória do WBEM.

Os três picos de processamento mais significativos sinalizam os instantes de *login* do cliente CIM no servidor de gestão, a leitura da política e a edição da política. O segundo e terceiro pico são mais intensos devido ao processamento interno do *provider*, ao efectuar o *parsing* da informação entre as duas tecnologias. Na Tabela 5.2 encontram-se quantificados os eventos responsáveis pelos incrementos de memória observados na Figura 5.3.

Tabela 5.2: Patamares de memória.

Evento	**memória Ocupada**
Arranque do Serviço	8924kB
Login do cliente CIM	9456kB
criação de instância *NetConfSession*	11076kB
leitura da política	14152kB
edição da política	15052kB

5.3 NETCONF

Após a execução dos três casos de teste, o tráfego NETCONF obtido pelas capturas foi filtrado, analisado e quantificado. A Tabela 5.3 contém para cada teste, a quantidade de tráfego discriminada por cada componente, bem como o seu peso relativo.

Tabela 5.3: Tráfego NETCONF.

Operações NETCONF								
	Estabelecimento Sessão		**Leitura Política**		**Edição Política**		**Criação Política**	
Nº Pacotes	41		23		19		27	
Nº Mensagens	4		1		4		4	
Tempo (s)	5,485		0,276		0,047		0,096	
Componente	**bytes**	**%**	**bytes**	**%**	**bytes**	**%**	**bytes**	**%**
XML	0	0	16184	80,17	9047	66,78	18692	76,97
RPC	0	0	242	1,2	968	7,14	968	3,99
SSH	7438	73,92	2244	11,12	2279	16,82	2842	11,7
TCP	1312	12,93	736	3,65	608	4,49	864	3,56
IP	820	8,08	460	2,28	380	2,8	540	2,22
Ethernet	574	5,66	322	1,6	266	1,96	378	1,56
Total	10144	100	20188	100	13548	100	24284	100

Na Figura 5.4 encontra-se ilustrado graficamente a contribuição de cada componente de tráfego por operação.

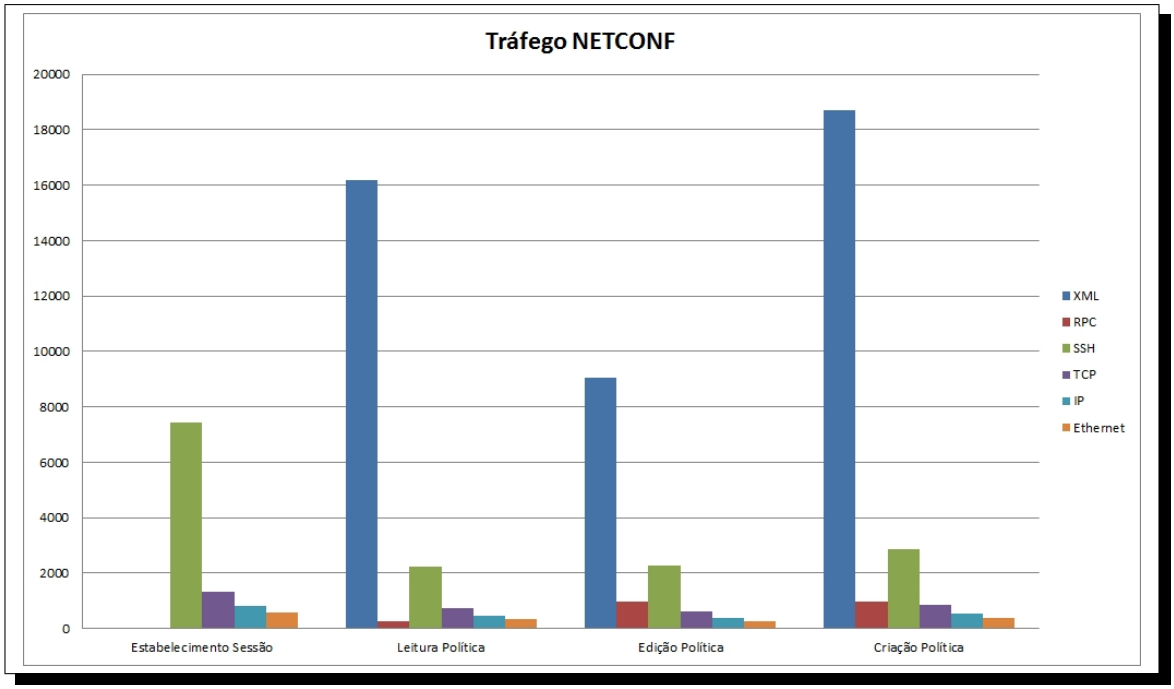

Figura 5.4: Gráfico do tráfego NETCONF.

O estabelecimento da sessão corresponde ao estabelecimento de uma ligação SSH com a

respectiva troca de chaves entre o cliente NETCONF *Manager* e o servidor NETCONF *Agent*.

A leitura da política por parte do sistema proposto, implica do ponto de vista NETCONF a obtenção da política do repositório *running* através da operação *GET-CONFIG*. Esta operação é bastante simples, pois requer apenas uma mensagem, retornando na resposta, a política de forma integral. A política em si, representa mais de 80% do tráfego trocado.

No caso da edição da política, as alterações são aplicadas através de NETCONF, com o envio da componente da política que contém a alteração. Do ponto de vista XML, as componentes da política correspondem aos seguintes elementos: <*UA_EventSubscription*>, <*UA_EventSubscription*> ou <*UA_EventSubscription*>. É expectável assim, que este seja o teste que gere menos tráfego, no entanto, do ponto de vista de operações NETCONF, a edição da política pressupõe várias operações NETCONF. Como o *fake_daemon*, que simula o dispositivo a gerir, suporta a capacidade *candidate*, a submissão das alterações implica que estas sejam aplicadas inicialmente no repositório *candidate*, através da operação *EDIT-CONFIG* e posteriormente seja executada a operação *COMMMIT*, para submeter as alterações para o repositório *running*, colocando assim a política em execução. Para evitar concorrência na configuração é necessário tornar o processo de edição numa transição atómica, para isso as operações *EDIT-CONFIG* e *COMMIT* foram antecedidas pela operação *LOCK* e sucedidas pela operação *UNLOCK*. Resulta assim um total de quatro mensagens trocadas, responsáveis pelo aumento do *overhead* de tráfego relativamente ao processo de leitura. O processo de criação de política é idêntico ao processo de edição de política, pois segue a mesma sequência de operações, *LOCK*, *EDIT-CONFIG*, *COMMIT* e *UNLOCK*, com a diferença de a política ser enviada na sua totalidade. Este facto justifica o aumento da quantidade de tráfego obtida.

Relativamente ao consumo de memória do NETCONF, foram analisados os processos SSHD (servidor SSH) e o *fake_daemon*. O comportamento obtido encontra-se ilustrado na Figura 5.5 e na Figura 5.6 respectivamente.

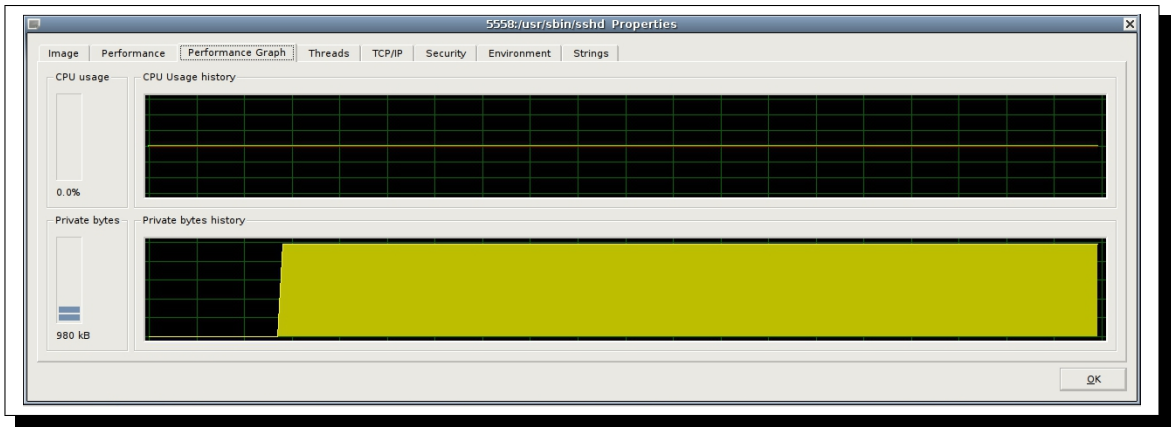

Figura 5.5: Consumo de memória do SSH.

De acordo com a Figura 5.5 o consumo de memória do servidor SSH manteve-se constante nos 580kB durante o processo de leitura e edição de política.

O *fake_daemon* apresentou o mesmo comportamento do servidor SSH mantendo o consumo de memória constante nos 1212kB durante o processo de leitura e edição de política.

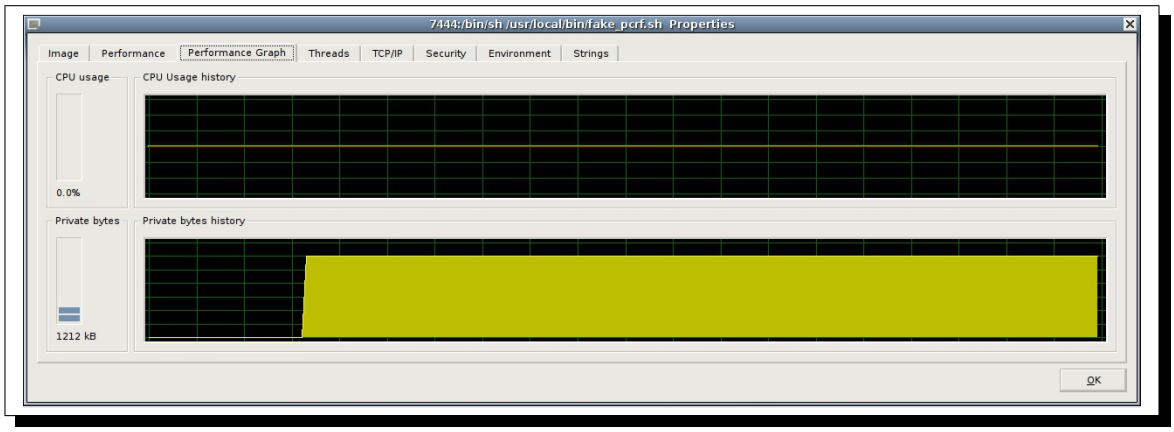

Figura 5.6: Consumo de memória do fake_daemon.

5.4 Análise de resultados

De uma forma qualitativa, os resultados obtidos para as duas tecnologias que constituem o sistema gestão, WBEM e NETCONF, correspondem ao esperado. Era expectável que o WBEM gera-se mais tráfego devido a sua natureza.

É importante ressalvar que num ambiente de produção, o overhead do WBEM será ainda maior devido à activação da encriptação *Secure Sockets Layer* (SSL), que não foram considerados para facilitar o desenvolvimento.

Relativamente à interpretação dos resultados, estes podem ser analisados de duas perspectivas distintas, as duas tecnologias podem ser concorrentes ou complementares. Na perspectiva concorrente, os resultados práticos valiam a melhor eficiência do protocolo NETCONF relativamente ao WBEM, pois requer menos mensagens e menos *overhead* de transporte. Relativamente à memória, também o NETCONF ganha vantagem apresentando menor consumo de memória. Contudo o WBEM também apresenta alguns pontos fortes relativamente ao NETCONF, nomeadamente um modelo de dados normalizado, e as enumeras implementações do protocolo existentes.

Por seu lado na perspectiva cooperante, pode-se alegar que, apesar de serem os dois protocolos de gestão, estes apresentam características próprias, com naturezas distintas. Os resultados obtidos podem corroborar esta perspectiva, pois o WBEM é mais exigente em termos de processamento e memória, tornando mais dispendioso a criação de dispositivos a gerir, sejam equipamentos de rede ou de outra natureza, capazes de correr servidores CIM embebidos. Ao delegar o WBEM para um nível de abstracção superior, responsabilizando a comunicação com os dispositivos a gerir, a protocolos de gestão como o NETCONF, SNMP ou qualquer outro, pode-se obter um sistema de gestão multi-dispositivo com uma interface normalizada, independente das tecnologias de gestão subjacentes, associadas aos dispositivos geridos.

Efectivamente a solução apresentada surge na expectativa de conciliar as as vantagens de cada uma das tecnologias, por uma lado, o modelo de dados aberto e rico do CIM, e por outro lado, a eficiência do NETCONF.

5.5 Análise dinâmica

De forma a avaliar a robustez e escalabilidade do sistema, este foi sujeito a uma análise dinâmica, recorrendo a ferramentas de *profiling* e submetido a testes de carga. No que diz respeito aos testes de carga, estes consistiram no ataque de pedidos CIM ao sistema de forma sequencial e em simultâneo.

5.5.1 *Profiling*

O *profiling* consiste nas actividades de análise dinâmica do código, como a análise de execução de uma determinada aplicação, de forma a detectar problemas com a utilização de recursos (memória, processador). Essa análise é efectuada recorrendo a ferramentas que permitem: a análise da memória utilizada, a análise do tempo despendido e do número de invocações de cada método e a análise da sequência de invocação dos métodos ou classes. Para a análise do sistema recorreu-se à ferramenta de *profiling* Valgrind [73]. O Valgrind é um software livre que auxilia o trabalho de depuração de software. Possui ferramentas que detectam erros decorrentes do uso incorrecto da memória dinâmica, como por exemplo os *memory leaks*, as alocações e as desalocações incorrectas bem como os acessos em áreas inválidas. A característica distinta desta ferramenta reside no facto de esta desenvolver uma uma máquina virtual para simular o acesso à memória da aplicação em teste, eliminando a necessidade de uso de outras bibliotecas auxiliares ou mudanças drásticas no código.

Callgrind

O *Callgrind* recorre a instrumentação de execução através da plataforma *Valgrind* para efectuar a simulação de cache e gerar os diagramas de invocações da aplicação. Desta forma, até as bibliotecas partilhadas e os *plugins* abertos dinamicamente podem ser depurados. Os ficheiros de dados gerados pelo *Callgrind* podem ser abertos usando o *KCachegrind* [18] de forma a permitir a pesquisa dos resultados de desempenho e a visualização dos diagramas.

Durante a execução dos testes, o serviço do *OpenPegasus* foi arrrancado como ilustrado na Figura 5.7 para a geração dos ficheiros de dados do *Callgrind*. Foram executas as operações de leitura e edição de política para a geração do ficheiro de dados, de forma a simular a utilização normal da aplicação.

```
1  $valgrind --tool=callgrind cimserver daemon=false
```

Figura 5.7: Início do *profiling* com a ferramenta *Callgrind*.

Com base no ficheiro de dados obtido, foi gerado o mapa da invocações do sistema através da ferramenta *KCachegrind*. O diagrama encontra-se ilustrado na Figura 5.8.

Tal como descrito anteriormente, este sistema consiste na integração de duas tecnologias de gestão, e como tal, uma parte significativa do processamento será destinada ao *parsing* da informação em XML recebida e enviada através de NETCONF. Na Figura 5.8 está ilustrado o mapa da invocações do sistema, representando as invocações a métodos da biblioteca de manipulação XML como a área mais significativa em termos de tempo de execução, seguindo-se os métodos da biblioteca que implementa o NETCONF *Manager*. Como é expectavel o tempo de execução dos métodos da biblioteca que implementa o NETCONF *Manager*

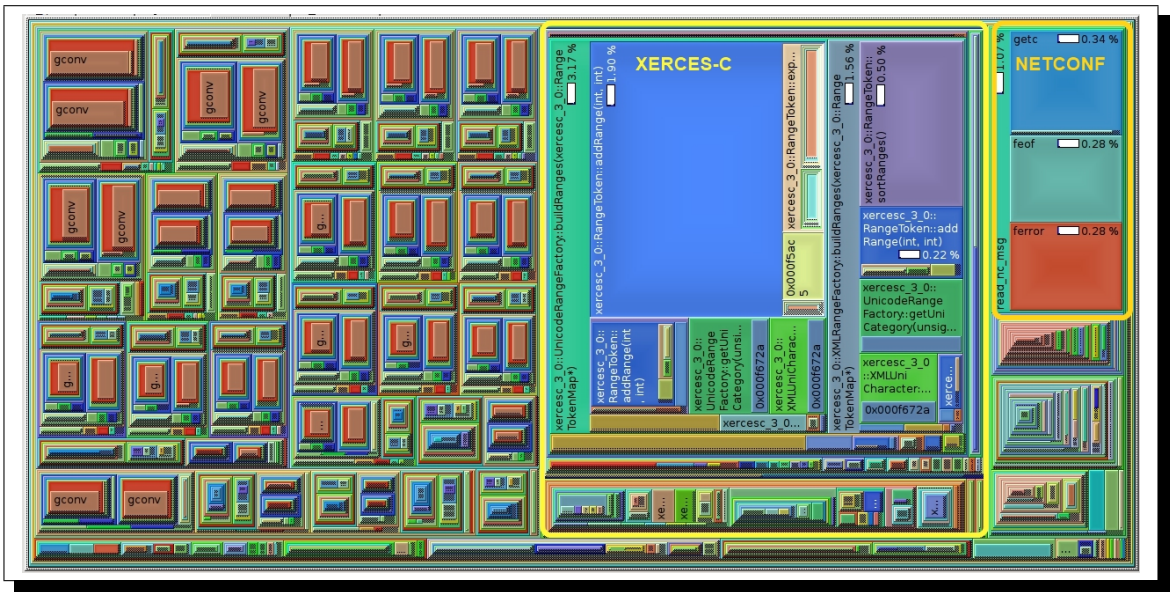

Figura 5.8: Mapa da invocações do sistema.

é significativo devido aos tempos comunicação do NETCONF *Manager* com o NETCONF *Agent*.

O *KCachegrind* permite também descriminar e visualizar os diagramas de invocações para os vários fluxos de execução, neste caso interessa-nos analisar as operações de leitura e edição de política. Para o fluxo de leitura de política, foi obtido o diagrama de invocações ilustrado na Figura 5.9.

O diagrama da Figura 5.9 apresenta, para cada método, o custo associado em termos de tempo de execução relativamente ao método que lhe deu origem, bem como a quantidade de vezes que o método em questão foi invocado. Os métodos mais invocados são os métodos intrínsecos que correspondem as operações WBEM, pois tratam-se de métodos *template*, genéricos para todas as classes.

Relativamente ao fluxo de leitura de política, na Figura 5.10 encontra-se o diagrama de invocações que descrevem esta operação. É uma operação mais complexa, pois requer mais invocações e necessita de mais tempo para ser executada.

Massif

Tal como o *Callgrind*, também o *Massif* recorre a instrumentação de execução através da plataforma *Valgrind*, mas focando a análise na memória dinâmica, ou seja mede a quantidade de memória dinâmica que é alocada pela aplicação. Isto inclui simultaneamente o espaço útil e os bytes extra alocados para as finalidades de contabilidade e do alinhamento. Pode igualmente medir o tamanho da(s) pilha(s) da aplicação, embora não o faça por defeito. O *profiling* de memória dinâmica pode ajudar a reduzir a quantidade de memória usada pela aplicação. Em máquinas modernas com memória virtual, isto fornece os seguintes benefícios:

- Pode acelerar a aplicação - uma aplicação que consuma menos memória interagirá melhor com as várias memórias *cache* da maquina, o que evita a paginação;

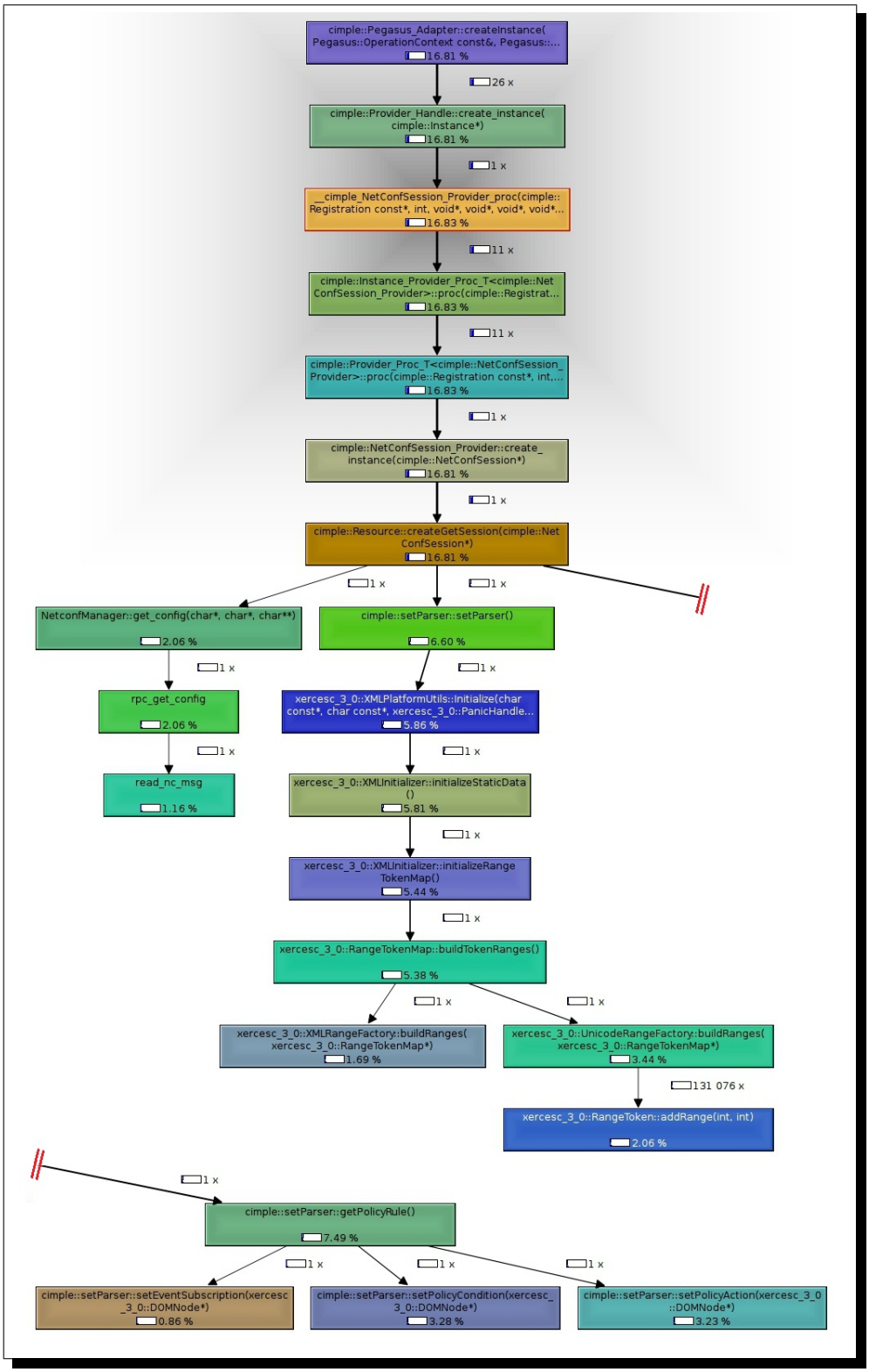

Figura 5.9: Diagrama de invocações para a leitura de política.

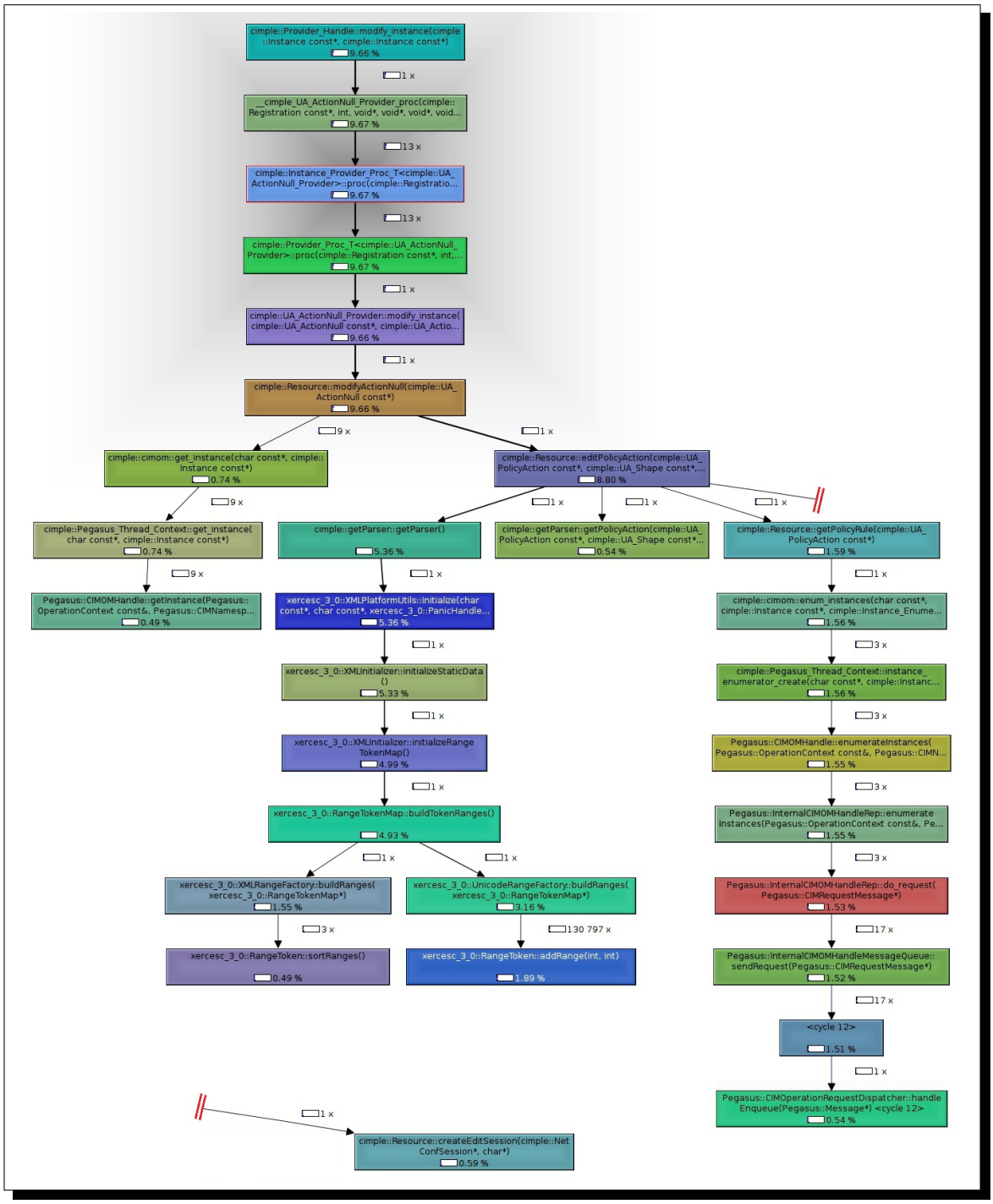

Figura 5.10: Diagrama de invocações para a edição de política.

- Se a aplicação consumir muita da memória, reduzirá a possibilidade de esgotar o espaço de memória em *swap* da maquina.

Existem determinados *space leaks* que não são detectados por analisadores de memória tradicionais, tais como *Memcheck's*. Isto porque a memória não se perdeu realmente para

sempre, ou seja um ponteiro continua a referencia-la, no entanto esta não está em uso. As aplicações que têm *memory leaks* como este, podem desnecessariamente aumentar a quantidade de memória alocada com o decorrer do tempo. O *Massif* pode ajudar a identificar estas fugas. Acima de tudo, o *Massif* não informa somente acerca da quantidade de memória dinâmica alocada a uma aplicação em execução, mas fornece simultaneamente informações detalhadas acerca das partes da aplicação que são responsáveis pela alocação de memória dinâmica.

O serviço do *OpenPegasus* foi arrancado como ilustrado na Figura 5.11 para a geração dos ficheiros de dados do *Massif*. Foram executas as operações de leitura e edição de política para a geração do ficheiro de dados.

```
1  $valgrind --tool=massif cimserver daemon=false
```

Figura 5.11: Início do *profiling* com a ferramenta *Massif*.

Com base no ficheiro de dados obtido após a execução das operações de leitura e edição de política, foi gerado o gráfico da alocação de memória identificando as 10 maiores contribuições. O gráfico foi obtido recorrendo a ferramenta *Massif Visualizer* [19]. Este encontra-se ilustrado na Figura 5.12.

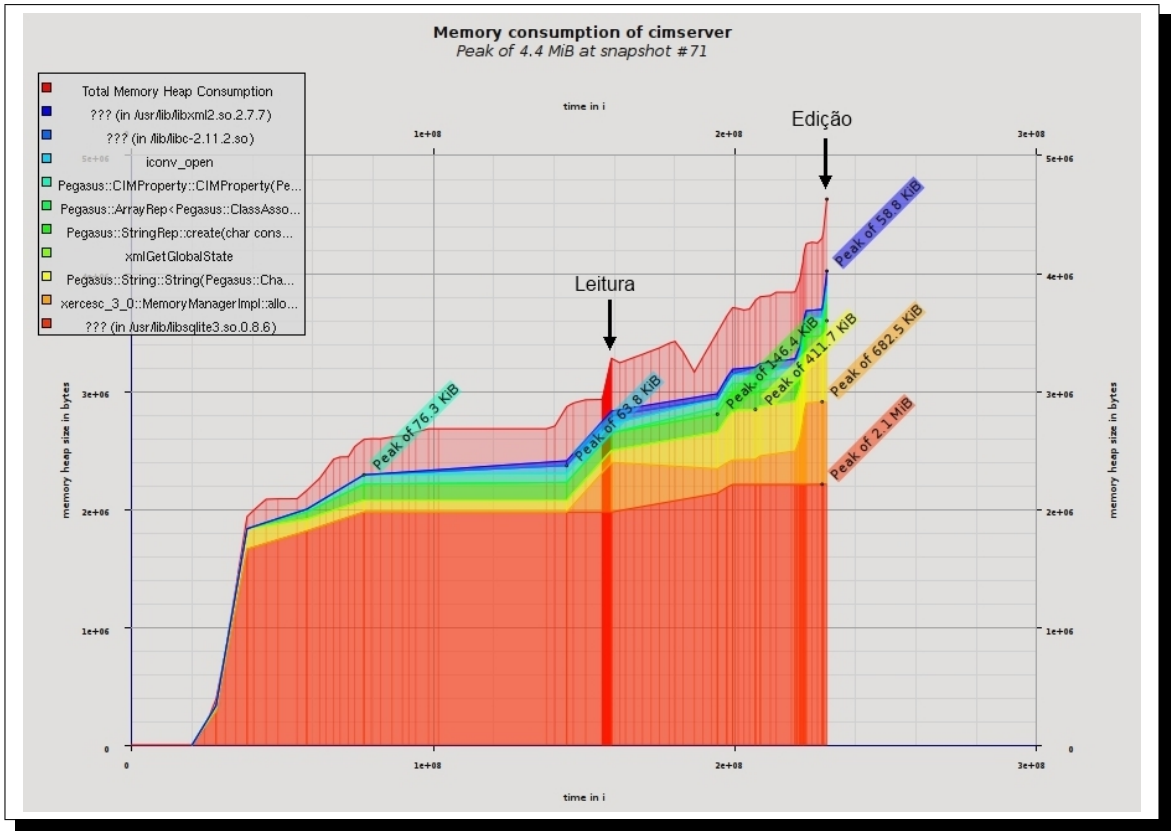

Figura 5.12: Picos de memória alocada.

O gráfico da Figura 5.12 ilustra a evolução do consumo de memória, onde se destacam as bibliotecas *sqlite* [29] relativa ao repositório CIM como o maior pico igual a 2.1MB e a biblioteca *xerces-c* com 682.5KB. O repositório CIM é implementado sobre a base de dados *sqlite*, logo é normal que a memória associada aumente com a instânciação de novas classes fruto do processo de leitura. Esta memória não é desalocada pois as instâncias são voláteis e encontram-se em memória enquanto o serviço do servidor CIM estiver em execução. O *OpenPegasus* gere a memória mantendo as instâncias durante um certo periodo de tempo e depois remove-as automaticamente. O processo de edição de política aumenta ainda mais a memória do *sqlite* devido as operações de pesquisa sobre o repositório, atinjindo nesse instante o maximo pico. Relativamente ao *xerces-c* atinje também o primeiro pico na leitura com a recepção da política por parte do NETCONF *Manager* e efectuando o seu *parsing* para a criação das instâncias das classes CIM que compõem a politíca, posteriormente com a edição o *xerces-c* atinje o seu máximo pico, sendo que nesta operação o XML é gerado de raiz. Os métodos que manipulam o XML são do tipo *Document Object Model* (DOM), onde o documento se encontra todo em memória.

5.5.2 Testes de carga

Os testes de carga permitem validar a escalabilidade do sistema face ao *stress* de utilização pretendido. Com recurso ao cliente CIM por linha de comandos (*cimcli*), que faz parte da plataforma do *Open Pegasus*, foram criados *scripts* que simulam o ataque de pedidos ao sistema. Um pedido neste contexto consiste na criação de uma instância *NetConfSession* em modo de edição de modo a obter a política *running* no *fake-daemon* através de NETCONF.

Para os pedidos consecutivos ou em série foi criado o seguinte *script*, descrito na Figura 5.13.

```
1  #!/bin/bash
2  CIMSERVER=127.0.0.1   # CIM Server Host
3  NCAGENT=127.0.0.1   # NetConf Agent Host
4
5  echo > stress_wb_s.log    # Clean File
6
7  for i in {1..3500}
8  do
9    cimcli -l $CIMSERVER ci NetConfSession SessionID=$i AgentHost=$NCAGENT
       AgentPort=830 User=<user> Password=<password> CreatePolicy=False
10   pidof cimserver | xargs pmap | grep total >> stress_wb_s.log
11 done
```

Figura 5.13: Script para pedidos em série.

O *script* da Figura 5.13 simula a execução de 3500 pedidos. Como se trata de invocações de pedidos consecutivos para a criação da mesma instância, obteve-se um comportamento linear de crescimento de consumo de memória por parte do processo do servidor CIM. Este comportamento está ilustrado no gráfico da Figura 5.14.

O gráfico da Figura 5.14 ilustra um crescimento linear da memória consumida, o que significa que a limitação para a recepção de novos pedidos por parte do servidor CIM seja a memória da maquina onde o processo está a correr. Efectivamente é isto que acontece, pois

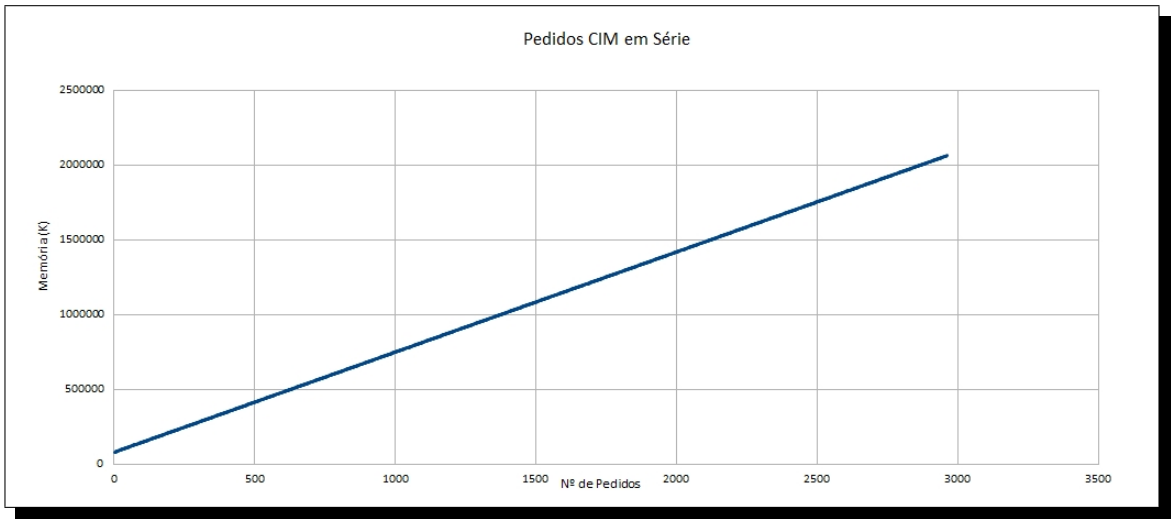

Figura 5.14: Comportamento da memória para pedidos CIM em série.

dos 3500 pedidos, o servidor CIM apenas foi capaz de processar 2961, atingindo assim o limite de memória disponível (2GB), sendo o processo terminado, tal como ilustra a mensagem da Figura 5.15.

```
...
:(-1) Cannot allocate memory
Segmentation fault
...
```

Figura 5.15: Mensagem de falta de memória.

Para os pedidos simultâneos ou em paralelo foi criado o seguinte *script*, descrito na Figura 5.16.

```
#!/bin/bash
CIMSERVER=127.0.0.1   # CIM Server Host
NCAGENT=127.0.0.1   # NetConf Agent Host

echo > stress_wb_p.log    # Clean File

for i in {1..70}
do
  cimcli -l $CIMSERVER ci NetConfSession SessionID=$i AgentHost=$NCAGENT
      AgentPort=830 User=<user> Password=<password> CreatePolicy=False &&
  pidof cimserver | xargs pmap | grep total >> stress_wb_p.log &
done
```

Figura 5.16: Script para pedidos em paralelo.

Neste caso em que os pedidos são executados em paralelo, a carga imposta ao sistema é muito superior, logo o numero de pedidos do *script* terá de ser muito inferior. Para esta situação foram considerados 70 pedidos CIM. Uma nota importante para a execução deste

teste foi a incremento do numero de sessões SSH simultâneas no servidor SSH do lado do NETCONF *Agent* para um valor elevado, pois por defeito o parâmetro *MaxStartups* que define o numero de sessões simultâneas, tem o valor 10. O gráfico da Figura 5.17 ilustra o comportamento do consumo de memória nesta situação.

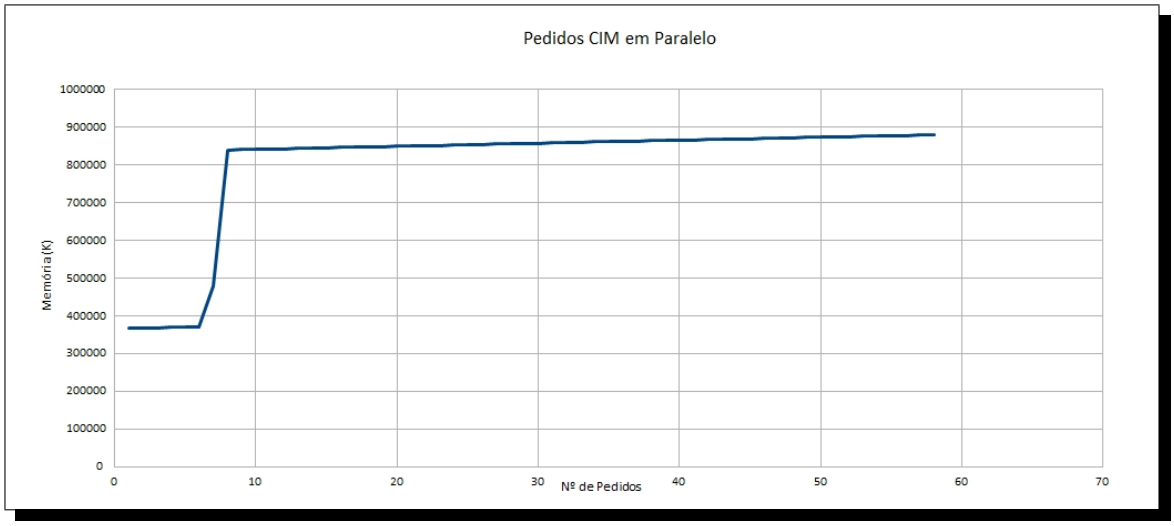

Figura 5.17: Comportamento da memória para pedidos CIM em paralelo.

Neste caso o comportamento da memória é totalmente diferente, e como se trata de muito menos pedidos CIM a limitação para o processamento de novos pedidos não será a memória da maquina, mas sim o poder de processamento, traduzindo-se na capacidade do servidor CIM em responder as múltiplas solicitações simultâneas em tempo útil. Nos testes efectuados, o sistema foi capaz de processar 58 pedidos, rejeitando os restantes por *timeout*, como ilustrado na mensagem da Figura 5.18.

```
1 ...
2 Returned Path NetConfSession.AgentHost="127.0.0.1",AgentPort=830,CreatePolicy=
      FALSE,Password="P@ssw0rd",SessionID=64,User="root@127.0.0.1"
3 Returned Path  NetConfSession.AgentHost="127.0.0.1",AgentPort=830,CreatePolicy=
      FALSE,Password="P@ssw0rd",SessionID=41,User="root@127.0.0.1"
4 Returned Path  NetConfSession.AgentHost="127.0.0.1",AgentPort=830,CreatePolicy=
      FALSE,Password="P@ssw0rd",SessionID=51,User="root@127.0.0.1"
5 cimcli Pegasus Exception: connection timed out. Cmd = ci Object = NetConfSession
6 cimcli Pegasus Exception: connection timed out. Cmd = ci Object = NetConfSession
7 cimcli Pegasus Exception: connection timed out. Cmd = ci Object = NetConfSession
8 ...
```

Figura 5.18: Output de rejeição de pedidos.

Na prática, este cenário de vários pedidos a acontecer exactamente no mesmo instante não se coloca, apenas por software é possível simular esta situação. Por outro lado, não é expectável que um sistema de gestão seja alvo de tanta concorrência.

Capítulo 6

Conclusões

O propósito maior deste trabalho, é a validação da aplicabilidade da tecnologia NETCONF ao suporte de gestão de Plataforma IMS. Concretamente, este trabalho propõe um sistema de gestão baseado em políticas para a gestão de um PCRF, elemento responsável pelo controlo da QdS e tarifação de serviços em redes IMS

O NETCONF é um protocolo de gestão normalizado dentro do IETF, e que pretende unificar o processo de configuração dos dispositivos de rede, e simultaneamente suprimir as deficiências das tecnologias actualmente disponíveis, como é o caso do SNMP. Um dos objectivos do NETCONF é unificar o processo de configuração dos dispositivos, definindo um conjunto uniforme de instruções a ser disponibilizado pelos dispositivos. Essas instruções definem mecanismos para obter, criar, editar e remover configurações, além de gerir estatísticas e informação de estado dos dispositivos. O princípio de funcionamento do NETCONF é baseado no paradigma RPC, através do qual é fornecido um conjunto de operações RPC definidas no protocolo. Resumidamente, o NETCONF *Manager* codifica um pedido RPC em XML e envia-o ao NETCONF *Agent*. O NETCONF *Agent*, ao receber uma mensagem NETCONF, processa o pedido e envia uma resposta RPC de volta ao NETCONF *Manager*. Tanto o pedido quanto a resposta estão codificados em XML, sendo as suas estruturas totalmente descritas em XML *schema*, permitindo assim ao NETCONF *Manager* e ao NETCONF *Agent* validarem as mensagens recebidas.

Apesar de o nome da dissertação apenas fazer referência ao protocolo de gestão NETCONF, um outro protocolo de gestão foi utilizado na solução proposta com servidor de gestão, concretamente o WBEM.

O WBEM é o resultado de uma iniciativa da indústria informática para criar uma tecnologia standard que permita aceder a informação de gestão num ambiente empresarial. Esta iniciativa foi patrocinada por algumas das maiores empresas do mundo na área, como são os casos da IBM, Compaq/HP, Cisco Systems e Microsoft. Uma das componentes mais importantes do WBEM é o seu modelo de informação OO que está incluído na sua definição. Esse modelo é denominado *Common Information Model* (CIM) e, como o nome indica, pretende conseguir representar de forma normalizada os mais variados componentes de um sistema computacional através de um modelo OO. A arquitectura WBEM é composta por cinco componentes básicos, o Cliente CIM e quatro componentes no lado do servidor WBEM, as aplicações de gestão, o CIMOM, o repositório de dados e os *providers*. Os *providers* correspondem a um conjunto de processos que permitem criar, modificar ou remover instâncias de objectos ou classes.

Do ponto de vista prático, o sistema de gestão proposto, consiste na interligação de uma implementação *open-source* de um servidor de gestão CIM com uma implementação NETCONF, também esta *open-source*, através do desenvolvimento de uma biblioteca de *providers*, contendo internamente uma implementação do NETCONF *Manager*.

A arquitectura da biblioteca de *providers* e consequente implementação, resulta de um processo prévio de definição do modelo de informação da política do sistema. O modelo de informação da política é um modelo OO, sendo uma política o resultado da associação de três outras classes: classe de subscrição de eventos, classe de condições e classe de acções. Por sua vez a classe de condições e classe de acções, são também classes de associação de classes que mapeiam os requisitos funcionais e que foram derivadas do modelo abstracto CIM. A classe de subscrição de eventos também deriva do modelo CIM. Da definição do modelo de informação da política e da lógica do sistema resultam dois ficheiros MOF. Com recurso a ferramenta *Cimple* gerou-se o esqueleto dos *providers* a partir do ficheiros MOF como ponto de partida, e desenvolveu-se a biblioteca.

Depois de implementado o sistema, este foi submetido a uma análise pormenorizada do tráfego gerado pelas duas tecnologias envolvidas, WBEM e NETCONF. Da análise do tráfego, quantificou-se a contribuição do *overhead* associado em cada caso de teste. Foram simultaneamente analisados os consumos de memória pelos vários componentes do sistema.

Analisando as duas tecnologias de forma independente, e iniciando a análise pelo WBEM, os resultados justificam a baixa eficiência do protocolo no que diz respeito a quantidade de tráfego adicional necessário para efectuar o transporte da política. O WBEM é claramente penalizado no seu desempenho ao utilizar XML e HTTP para codificação de transporte, situação agravada pela grande quantidade de mensagens trocadas nos casos de leitura e criação da política. Relativamente a edição de política, a troca de mensagens é significativamente menor e consequentemente também o tráfego gerado, porque apenas a classe que se pretende editar é obtida pelo cliente CIM. Por outro lado, quando se trata de gestão de configuração, estas sanções não farão muita diferença, pois a frequência de acções de configuração é reduzida, no entanto este problema pode ser brutalmente agravado como a introdução de indicações no sistema fruto de processos de monitorização, disparando assim a frequência de troca de mensagens. A reduzida eficiência do WBEM relativamente ao transporte e quantidade de mensagens apreciável é contudo uma consequência da sua complexidade e versatilidade que suportam um rico e poderoso modelo de informação CIM, que fornece uma arquitectura flexível, permitindo estender facilmente o seu modelo de informação para cobrir novos dispositivos e aplicações. Outra consequência de toda esta flexibilidade e complexidade é o consumo de recursos computacionais nomeadamente memória.

No que diz respeito ao NETCONF, estabelecendo uma análise comparativa dos resultados obtidos com os do WBEM, o NETCONF é claramente mais eficiente ao nível do transporte e consumo de memória, mesmo recorrendo também ao XML para codificação da informação. Por outro lado, o seu modelo de informação ainda se encontra em fase de adopção. Para garantir a interoperabilidade do NETCONF e a capacidade de manipular dados de forma normalizada, o IETF criou um grupo de trabalho designado de NETMOD com o objectivo de apoiar o desenvolvimento de um modelo de dados que seja adoptado pela industria designado por YANG. A linguagem YANG foi proposta como a linguagem de definição de dados, incluindo dados de configuração e de estado, manipulados pelo protocolo NETCONF, através de RPC e notificações.

Não é ambição do sistema proposto estabelecer uma comparação entre as duas tecnologias de gestão, mas sim tirar partido das melhores características de cada uma. O WBEM, é com-

putacionalmente mais exigente, e inviabiliza uma comunicação *CIM-XML* com os elementos a gerir, ficando assim atribuída ao NETCONF essa tarefa. O NETCONF é mais eficiente ao nível do transporte e menos exigente em termos de recursos computacionais, tal como fundamentam os resultados obtidos.

Os resultados obtidos permitem fundamentar a viabilidade da tecnologia NETCONF, como uma tecnologia capaz de responder ao desafios impostos pela complexidade e heterogeneidade das Redes de Próxima Geração. O NETCONF permite colmatar as deficiências de segurança e escalabilidade do SNMP, fornecendo simultaneamente eficiência e desempenho.

Relativamente à implementação do sistema de gestão, as opções tomadas sobre as implentações dos protocolos de gestão e as ferramentas de suporte ao desenvolvimento revelaram-se bastantes válidas. O OpenPegasus é de facto uma implementação bastante poderosa do protocolo WBEM, devido à sua flexibilidade e potencialidade, apesar de não ser a implementação com melhor desempenho. À data de selecção da implementação do protocolo NETCONF, o Netopeer era das poucas implementações *open-source* que implementava o protocolo SSH desenvolvida em "C". Apesar de não apresentar muita documentação o suporte por parte do seu autor, foi bastante útil. O *xerces-c* é uma biblioteca globalmente utilizada para a manipulação de XML com imensa documentação e exemplos. A contribuição do CIMPLE foi muito significativa, pois permitiu ganhar tempo na definição dos esqueletos dos *providers* e no processo de registo, permitindo focar o esforço na lógica de adaptação. O resultado de tudo isto foi uma proposta de um sistema de gestão, funcional e robusto como evidenciam os resultados dos testes a que o sistema foi submetido.

Não se tratando de um produto comercial, a análise dinâmica de *profiling* e de carga é sempre de extrema importância pois, exterioriza as vulnerabilidades do sistema, que podem não incidir apenas na qualidade da implementação, mas sim nas próprias tecnologias. No que diz respeito ao *profiling*, foi analisado o sistema recorrendo á ferramenta *Valgrind* para a análise da memória utilizada, a análise do tempo despendido e do número de invocações de cada método e a análise da sequência de invocação dos métodos. Os resultados obtidos permitem validar a implementação, pois os fluxos de invocações correspondem ao comportamento pretendido, não foram detectados *memory leaks* e os tempos de execução são perfeitamente aceitáveis apesar de não existirem métricas de comparação.

No que diz respeito à análise dinâmica na vertente de carga, o sistema foi injectado com pedidos CIM de forma sequencial e em simultâneo. Relativamente à resposta do sistema, este não evidenciou problemas de estabilidade ou escalabilidade da implementação, sendo as limitações impostas pelos recursos de *hardware*, memória no caso dos pedidos em série e capacidade de processamento no caso dos pedidos em paralelo.

Finalmente no que toca a trabalho futuro, existem alguns aspectos que podem ser melhorados, nomeadamente o processo de edição da política. A submissão da informação acontece quando uma qualquer classe da política é editada, na invocação do método *modify_instance*, mas se as alterações pretendidas afectam mais de uma classe, então a informação é submetida mais que umas vez. A questão é que todos os procedimento têm de ser mapeados nos métodos intrínsecos dos *providers*, e nesta situação será difícil ajuizar quando é que o processo de edição começa e quando acaba, de modo a submeter as alterações apenas uma vez. Outra aspecto que não trata de melhoria, pois está fora do âmbito deste trabalho, é o desenvolvimento de um cliente CIM optimizado para este sistema, ou seja que forneça uma interface gráfica que permita automatizar as operações, de forma a minorar os tempos de leitura e criação de políticas.

Referências

[1] ARIB association of radio industries and businesses.
http://www.arib.or.jp/english/, 2010-11-22.

[2] ATIS | welcome.
http://www.atis.org/, 2010-11-22.

[3] CCSA home.
http://www.ccsa.org.cn/english/, 2010-11-22.

[4] CESNET conference 2008.
http://www.ces.net/events/2008/conference/, 2010-08-23.

[5] CimNavigator^{TM}.
http://cimnavigator.com/, 2010-09-13.

[6] CiscoWorks QoS policy manager - products & services - cisco systems.
http://www.cisco.com/en/US/products/sw/cscowork/ps2064/, 2010-08-24.

[7] DMTF - cim schema: Version 2.22.0.
http://www.dmtf.org/standards/cim/cim_schema_v2220, 2010-08-24.

[8] DMTF - common information model (CIM).
http://www.dmtf.org/standards/cim/, 2010-08-24.

[9] DMTF - DMI 2.0s specification.
http://www.dmtf.org/standards/dmi/spec, 2010-08-24.

[10] DMTF - specification for CIM operations over HTTP.
http://www.dmtf.org/standards/documents/WBEM/DSP200.html, 2010-08-24.

[11] DMTF - specification for the representation of CIM in XML.
http://www.dmtf.org/standards/documents/WBEM/DSP201.html, 2010-08-24.

[12] DMTF - stackmap.
http://www.dmtf.org/standards/stackmap/, 2010-08-24.

[13] DMTF - Web-Based enterprise management (WBEM).
http://www.dmtf.org/standards/wbem/, 2010-08-24.

[14] draft-ietf-netconf-beep-10 - using the NETCONF protocol over the blocks extensible exchange protocol (BEEP).
http://tools.ietf.org/html/draft-ietf-netconf-beep-10, 2010-08-24.

[15] Eclipse CDT.
http://www.eclipse.org/cdt/, 2010-08-24.

[16] Extreme networks.
http://www.extremenetworks.com/products/policy-manager.aspx, 2010-08-24.

[17] Home | DMTF.
http://dmtf.org/, 2010-08-24.

[18] KCachegrind.
http://kcachegrind.sourceforge.net/html/Home.html, 2010-12-04.

[19] Massif visualizer - overview - KDE projects.
https://projects.kde.org/projects/kdereview/massif-visualizer, 2010-12-04.

[20] Netconf central.
http://www.netconfcentral.org/, 2010-08-24.

[21] Netmod status pages.
https://trac.tools.ietf.org/wg/netmod/, 2010-08-24.

[22] netopeer - project hosting on google code.
http://code.google.com/p/netopeer/, 2010-08-24.

[23] The open group - making standards work.
http://www.opengroup.org/, 2010-08-24.

[24] OpenPegasus.
http://www.openpegasus.org/, 2010-08-24.

[25] OpenWBEM home page.
http://openwbem.sourceforge.net/, 2010-08-24.

[26] Programmable hardware | www.liberouter.org.
http://www.liberouter.org/, 2010-08-24.

[27] SimpleWBEM.
http://simplewbem.org/, 2010-08-24.

[28] SNMPLink.org - simple network management protocol (SNMP) portal.
http://www.snmplink.org/, 2010-08-24.

[29] SQLite home page.
http://www.sqlite.org/, 2010-12-03.

[30] THE TELECOMMUNICATION TECHNOLOGY COMMITTEE.
http://www.ttc.or.jp/e/, 2010-11-22.

[31] WBEM services.
http://wbemservices.sourceforge.net/, 2010-09-13.

[32] Welcome to TTA - telecommunications technology association of korea.
http://www.tta.or.kr/English/, 2010-11-22.

[33] Xerces-C++ XML parser.
http://xerces.apache.org/xerces-c/, 2010-08-24.

[34] Management information base (mib) for the simple network management protocol (snmp), 2002.

[35] Using the NETCONF protocol over the blocks extensible exchange protocol (BEEP).
http://tools.ietf.org/html/rfc4744, 2010-08-24, December 2006.

[36] Haseeb Akhtar, Pat R Calhoun, Allan C Rubens, Erik Guttman, John Loughney, and Jari Arkko. Diameter base protocol. http://tools.ietf.org/search/rfc3588, September 2003.

[37] Mohamad Badra. NETCONF over transport layer security (TLS).
http://tools.ietf.org/html/rfc5539, 2010-08-24, May 2009.

[38] J.-J. P. Balbas, S. Rommer, and J. Stenfelt. Policy and charging control in the evolved packet system. 47(2):68–74, 2009.

[39] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0, 1996.

[40] J. Boyle, R. Cohen, D. Durham, R. Rajan, and A. Sastry. Cops usage for rsvp, 2000.

[41] J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The cops (common open policy service) protocol, 2000.

[42] Gonzalo Camarillo and Miguel-Angel Garca-Martn. *The 3G IP Multimedia Subsystem: Merging the Internet and the Cellular Worlds*. Wiley Publishing, 3 edition, 2008.

[43] Gonzalo Camarillo, Mark Handley, Jon Peterson, Jonathan Rosenberg, Alan Johnston, Henning Schulzrinne, and Robert Sparks. SIP: session initiation protocol. http://tools.ietf.org/search/rfc3261, June 2002.

[44] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. Cops usage for policy provisioning (cops-pr), 2001.

[45] Huiyang Cui, Bin Zhang, Guohui Li, Xuesong Gao, and Yan Li. Contrast analysis of NETCONF modeling languages: XML schema, relax NG and YANG. In *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, pages 322–326, 2009.

[46] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.

[47] Nicodemos Damianou, Naranker Dulay, Emil Lupu, Morris Sloman, and Nicodemos Damianou Naranker Dulay. Ponder: An object-oriented language for specifying security and management policies. 2000.

[48] R. Enns. NETCONF configuration protocol.
http://tools.ietf.org/html/rfc4741, 2010-08-24.

[49] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1997.

[50] Martin Fowler and Kendall Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition)*. Addison-Wesley Professional, 1999.

[51] Ted Goddard. Using NETCONF over the simple object access protocol (SOAP). http://tools.ietf.org/html/rfc4743, 2010-08-24, December 2006.

[52] Pankaj Goyal, Rao Mikkilineni, and Murthy Ganti. Fcaps in the business services fabric model. In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 45–51, Washington, DC, USA, 2009. IEEE Computer Society.

[53] Robert Hancock, Sven van den Bosch, Georgios Karagiannis, and John Loughney. Next steps in signaling (NSIS): framework. http://tools.ietf.org/html/rfc4080, 2010-08-24, June 2005.

[54] Sean Harnedy. *Web-Based Information Management: An Introduction to the Technology and Its Application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[55] Heinz-Gerd Hegering, Sebastian Abeck, and Bernhard Neumair. *Integrated management of networked systems: concepts, architectures, and their operational application*. Morgan Kaufmann Publishers Inc., 1998.

[56] F. Kastenholz and K. McCloghrie. The interfaces group MIB. http://tools.ietf.org/html/rfc2863, 2010-08-24.

[57] Dave Kosiur. *Understanding policy-based networking*. John Wiley & Sons, Inc., 2001.

[58] Tatiana Kovacikova, Pavol Segec, and Milan Kubina. Ims in the next generation network. In *Proceedings of the 11th Conference on 11th WSEAS International Conference on Communications - Volume 11*, pages 45–50, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).

[59] Jorge Lobo. Cim simplified policy language (cim-spl). In *Specification DSP0231 v1.0.0*. Distributed Management Task Force (DMTF), 2007.

[60] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 291–298, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[61] Ed. M. Bjorklund. Yang - a data modeling language for the network configuration protocol (netconf). http://tools.ietf.org/html/rfc6020, October 2010.

[62] Pierre Metz. *Functional Requirements Engineering with Use Cases: Revising and Unifying the Use Case Textual and Graphical Worlds*. VDM Verlag, Saarbrücken, Germany, Germany, 2009.

[63] B. Moore, editor. *Policy Core Information Model (PCIM) Extensions*. 2003.

[64] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy core information model – version 1 specification, 2001.

[65] H. Nielsen, P. Leach, and S. Lawrence. An http extension framework, 2000.

[66] Angela Orebaugh, Gilbert Ramirez, Josh Burke, and Larry Pesce. *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[67] A. Ponnappan, Lingjia Yang, R. Pillai, and P. Braun. A policy based qos management system for the intserv/diffserv based internet. In *Proc. Third Int Policies for Distributed Systems and Networks Workshop*, pages 159–168, 2002.

[68] J. Postel. Transmission control protocol.
http://tools.ietf.org/html/rfc793, 2010-08-24.

[69] C. Rigney, W. Simpson, S. Willens, and A. Rubens. Remote authentication dial in user service (RADIUS).
http://tools.ietf.org/html/rfc2138, 2010-08-24.

[70] Christian Esteve Rothenberg and Andreas Roos. A review of Policy-Based resource and admission control functions in evolving access and next generation networks. *J. Netw. Syst. Manage.*, 16(1):14–45, 2008.

[71] J. Schoenwaelder and A. Pras. On the difference between information models and data models.
http://tools.ietf.org/html/rfc3444, 2010-08-24.

[72] Ian Sommerville. *Software Engineering (7th Edition)*. Pearson Addison Wesley, 2004.

[73] Lambert M. Surhone, Miriam T. Timpledon, and Susan F. Marseken. *Valgrind: Programming Tool, Memory Debugger, Memory Leak, Performance Analysis, Debugging, Julian Seward, GNU General Public License*. Betascript Publishing, Mauritius, 2010.

[74] IBM Tivoli. Autonomic computing policy language. In *Tutorial*. IBM Corporation, 2005.

[75] Hector Trevino and Sharon Chisholm. NETCONF notification transport mappings.
http://tools.ietf.org/html/draft-trevino-netconf-notification-transport-01, 2010-08-24.

[76] Margaret Wasserman and Ted Goddard. Using the NETCONF configuration protocol over secure SHell (SSH).
http://tools.ietf.org/html/rfc4742, 2010-08-24, December 2006.

Anexos

Apêndice A

Ficheiros MOF dos Providers

A.1 repository.mof

```
#pragma include ("policy.mof")

class NetConfSession
{
  [key] uint32 SessionID;
  [key] string AgentHost;
  [key] uint32 AgentPort;
  [key] string User;
  [key] string Password;
  [key] boolean CreatePolicy;
};

[Association]
class NcPolicyManager
{
  [key] NetConfSession  REF Session;
  [key] UA_PolicyRule REF Policy;
};
```

A.2 policy.mof

```
// ===================================================================
// UA_Action
// ===================================================================
[Version ( "1.0.0" ), Description (" To be done")]
//    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
class UA_Action : CIM_PolicyAction {
        [Description ("To be done")]
  uint16 Precedence;
};

[Version ( "1.0.0" ), Description (" To be done")]
//    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
class UA_Shape : UA_Action {

};

[Version ( "1.0.0" ), Description (" To be done")]
```

```
18 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
19 class UA_Mark : UA_Action {
20
21 };
22
23 [Version ( "1.0.0" ), Description (" To be done")]
24 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
25 class UA_Forward : UA_Action {
26
27 };
28
29 [Version ( "1.0.0" ), Description (" To be done")]
30 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
31 class UA_FindLowerLayer : UA_Action {
32
33 };
34
35 [Version ( "1.0.0" ), Description (" To be done")]
36 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
37 class UA_ActionNull : UA_Action {
38
39 };
40
41 [Version ( "1.0.0" ), Description (" To be done")]
42 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
43 class UA_Police : UA_Action {
44
45 };
46
47 [Version ( "1.0.0" ), Description (" To be done")]
48 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
49 class UA_Drop : UA_Action {
50
51 };
52
53 [Version ( "1.0.0" ), Description (" To be done")]
54 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
55 class UA_Rate : UA_Action {
56
57 };
58
59 [Version ( "1.0.0" ), Description (" To be done")]
60 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
61 class UA_QoSSessionDeterministicCAC : UA_Action {
62
63 };
64
65 [Version ( "1.0.0" ), Description (" To be done")]
66 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
67 class UA_QoSStaticCAC : UA_Action {
68
69 };
70
71 // ================================================================
72 // UA_Condition
73 // ================================================================
74
```

```
75
76 [Version ( "1.0.0" ), Description (" To be done")]
77 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
78 class UA_Condition : CIM_PolicyCondition {
79   [Description ("To be done")]
80         boolean ORed;
81
82 };
83
84 [Version ( "1.0.0" ), Description (" To be done")]
85 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
86 class UA_IPClassifier : UA_Condition {
87   [Description ("To be done")]
88         uint32 IPClassifierID;
89
90   [Description ("To be done")]
91         string sourceIP;
92
93   [Description ("To be done")]
94         uint16 sourcePort;
95
96   [Description ("To be done")]
97         string DestinationIP;
98
99   [Description ("To be done")]
100        uint16 DestinationPort;
101
102  [Description ("To be done")]
103        uint16 Protocol;
104
105  [Description ("To be done")]
106        uint16 Priority;
107
108  [Description ("To be done")]
109        uint16 DSCPMask;
110 };
111
112 [Version ( "1.0.0" ), Description (" To be done")]
113 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
114 class UA_MPLSClassifier : UA_Condition {
115        [Description ("To be done")]
116        uint32 labelStackID;
117
118 };
119
120 [Version ( "1.0.0" ), Description (" To be done")]
121 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
122 class UA_NullCondition : UA_Condition {
123
124 };
125
126 [Version ( "1.0.0" ), Description (" To be done")]
127 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
128 class UA_Node : UA_Condition {
129        [Description ("To be done")]
130        string  Name;
131
```

```
132         [Description ("To be done")]
133         uint16 Type;
134
135         [Description ("To be done")]
136         boolean Configurable;
137
138         [Description ("To be done")]
139         string Community;
140
141         [Description ("To be done")]
142         string ControlEntityAddress;
143
144 };
145
146 [Version ( "1.0.0" ), Description (" To be done")]
147 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
148 class UA_ServiceClass : UA_Condition {
149         [Description ("To be done")]
150         uint32 ServiceClassID;
151
152         [Description ("To be done")]
153         string Name;
154
155         [Description ("To be done")]
156         boolean Reserved;
157
158         [Description ("To be done")]
159         uint8 Utilization;
160
161         [Description ("To be done")]
162         uint16 UtilizationTarget;
163
164         [Description ("To be done")]
165         uint16 ExtraUtilizationTarget;
166
167 };
168
169
170 [Version ( "1.0.0" ), Description (" To be done")]
171 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
172 class UA_ATMClassifier : UA_Condition {
173         [Description ("To be done")]
174         uint32 vci;
175
176         [Description ("To be done")]
177         uint32 vpi;
178
179 };
180
181 [Version ( "1.0.0" ), Description (" To be done")]
182 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
183 class UA_MAC8021Classifier : UA_Condition {
184         [Description ("To be done")]
185         uint32 svlan;
186
187         [Description ("To be done")]
188         uint32 cvlan;
```

```
189
190 };
191
192 [Version ( "1.0.0" ), Description (" To be done")]
193 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
194 class UA_DPIClassifier : UA_Condition {
195         [Description ("To be done")]
196         uint16 ApplicationType;
197
198 };
199
200 // ===================================================================
201 // Indications
202 // ===================================================================
203 [Version ( "1.0.0" ), Description (" To be done")]
204 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
205 class UA_MiscEvent : CIM_InstIndication {
206         [Description ("To be done")]
207         uint32 EventID;
208 };
209
210 [Version ( "1.0.0" ), Description (" To be done")]
211 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
212 class UA_AfterMLMStartup : UA_MiscEvent {
213         [Description ("To be done")]
214         uint32 MLMID;
215 };
216
217 [Version ( "1.0.0" ), Description (" To be done")]
218 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
219 class UA_LinkThreshold : CIM_ThresholdIndication {
220         [Description ("To be done")]
221         uint32 Value;
222 };
223
224 [Version ( "1.0.0" ), Description (" To be done")]
225 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
226 class UA_ServiceThreshold : CIM_ThresholdIndication {
227         [Description ("To be done")]
228         uint32 ServiceID;
229
230   [Description ("To be done")]
231         uint32 LinkID;
232
233   [Description ("To be done")]
234         uint32 Value;
235
236   [Description ("To be done")]
237         uint16 ValueType;
238 };
239
240 [Version ( "1.0.0" ), Description (" To be done")]
241 //    MappingStrings { "DaidalosCIMExtensions" }, provider("c++::qosprovider")]
242 class UA_OutOfMoney : CIM_AlertInstIndication {
243         [Description ("To be done")]
244         uint32 UserID;
245 };
```

```
246
247 class UA_EventSubscription : CIM_IndicationSubscription
248 {
249   [Description ("To be done")]
250   uint32 EventID;
251 };
252
253 // ==================================================================
254 // Associations
255 // ==================================================================
256
257 [Association]
258 class UA_PolicyAction
259 {
260   [key] uint32      ActionID;
261
262   UA_Shape      REF Shape;
263   UA_Mark       REF Mark;
264   UA_Forward      REF Forward;
265   UA_FindLowerLayer   REF FindLowerLayer;
266   UA_ActionNull     REF ActionNull;
267   UA_Police     REF Police;
268   UA_Drop       REF Drop;
269   UA_Rate       REF Rate;
270   UA_QoSSessionDeterministicCAC REF QoSSessionDet;
271   UA_QoSStaticCAC     REF QoSStaticCAC;
272 };
273
274 [Association]
275 class UA_PolicyCondition
276 {
277   [key] uint32      ConditionID;
278
279   UA_IPClassifier     REF IPClassifier;
280   UA_MPLSClassifier   REF MPLSClassifier;
281   UA_NullCondition    REF NullCondition;
282   UA_Node       REF Node;
283   UA_ServiceClass     REF ServiceClass;
284   UA_ATMClassifier    REF ATMClassifier;
285   UA_MAC8021Classifier    REF MAC8021Classifier;
286   UA_DPIClassifier    REF DPIClassifier;
287 };
288
289 [Association]
290 class UA_PolicyIndication
291 {
292   [key] uint32    IndicationID;
293
294   UA_AfterMLMStartup  REF AfterMLMStartup;
295   UA_LinkThreshold  REF LinkThreshold;
296   UA_ServiceThreshold REF ServiceThreshold;
297   UA_OutOfMoney   REF OutOfMoney;
298 };
299
300 [Association]
301 class UA_PolicyRule
302 {
```

```
303   [key] uint32    RuleID;
304
305   UA_EventSubscription  REF EventSubscription;
306   UA_PolicyCondition  REF Condition;
307   UA_PolicyAction   REF Action;
308 };
```

Apêndice B

Modelo de dados YANG da política

```
module UA_Policy {

  namespace
    "urn:ietf:params:xml:ns:yang:smiv2:ua:policy";

  prefix
    ua_policy;

  organization
    "IT, Instituto de Telecomunicacoes";

  contact
    "Ricardo Mendes <rmendes@hng.av.it.pt>";

  description
    "Data model for PBNM Policy.";

  revision "2010-04-17" {
    description
      "Initial version, draft 1.";
  }

  container UA_PolicyRule {
    description
      "UA_PolicyRule";

    leaf RuleID { type uint32; config true; }

    container UA_EventSubscription {
      description
        "UA_EventSubscription : CIM_IndicationSubscription";

      leaf EventID  { type uint32; config true; }

      leaf OnFatalErrorPolicy     { type uint16; config true; }
      leaf OtherOnFatalErrorPolicy    { type string; config true; }
      leaf FailureTriggerTimeInterval   { type uint64; config true; }
      leaf SubscriptionState      { type uint16; config true; }
      leaf OtherSubscriptionState   { type string; config true; }
      leaf TimeOfLastStateChange    { type string; config true; } /* datetime */
      leaf SubscriptionDuration   { type uint64; config true; }
```

```
42        leaf SubscriptionStartTime    { type string; config true; } /* datetime */
43        leaf SubscriptionTimeRemaining    { type uint64; config true; }
44        leaf RepeatNotificationPolicy   { type uint16; config true; }
45        leaf OtherRepeatNotificationPolicy  { type string; config true; }
46        leaf RepeatNotificationInterval   { type uint64; config true; }
47        leaf RepeatNotificationGap    { type uint64; config true; }
48        leaf RepeatNotificationCount    { type uint16; config true; }
49
50        container Filter {
51          leaf    InstanceID      { type string; config true; }
52          leaf    Caption       { type string; config true; }
53          leaf    Description     { type string; config true; }
54          leaf    ElementName     { type string; config true; }
55          leaf    SystemCreationClassName   { mandatory true; type string; config
                true; }
56          leaf    SystemName      { mandatory true; type string; config true; }
57          leaf    CreationClassName   { mandatory true; type string; config true; }
58          leaf    Name        { mandatory true; type string; config true; }
59          leaf    SourceNamespace     { type string; config true; }
60          leaf-list SourceNamespaces    { type string; config true; }
61          leaf    IndividualSubscriptionSupported { type boolean; config true;}
62          leaf    Query       { type string; config true; }
63          leaf    QueryLanguage     { type string; config true; }
64        }
65
66        container Handler {
67          leaf InstanceID     { type string; config true; }
68          leaf Caption      { type string; config true; }
69          leaf Description    { type string; config true; }
70          leaf ElementName    { type string; config true; }
71          leaf SystemCreationClassName  { mandatory true; type string; config true;
                }
72          leaf SystemName     { mandatory true; type string; config true; }
73          leaf CreationClassName    { mandatory true; type string; config true; }
74          leaf Name     { mandatory true; type string; config true; }
75          leaf PersistenceType    { type uint16; config true; }
76          leaf OtherPersistenceType { type string; config true; }
77          leaf Destination    { type string; config true; }
78          leaf OtherProtocol    { type string; config true; }
79          leaf Protocol     { type uint16; config true; }
80        }
81      }
82
83      container UA_PolicyCondition {
84        description
85          "UA_PolicyCondition";
86
87        leaf ConditionID  { mandatory true; type uint32; config true; }
88
89        container UA_IPClassifier {
90          description
91            "UA_IPClassifier : UA_Condition";
92
93          leaf IPClassifierID     { type uint32; config true; }
94          leaf sourceIP       { type string; config true; }
95          leaf sourcePort       { type uint16; config true; }
96          leaf DestinationIP      { type string; config true; }
```

```
97          leaf DestinationPort      { type uint16; config true; }
98          leaf Protocol       { type uint16; config true; }
99          leaf Priority       { type uint16; config true; }
100         leaf DSCPMask       { type uint16; config true; }
101
102         leaf InstanceID       { type string; config true; }
103         leaf Caption        { type string; config true; }
104         leaf Description      { type string; config true; }
105         leaf ElementName      { type string; config true; }
106         leaf CommonName       { type string; config true; }
107         leaf-list PolicyKeywords    { type string; config true; }
108
109         leaf SystemCreationClassName    { mandatory true; type string; config
              true; }
110         leaf SystemName       { mandatory true; type string; config true; }
111         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
              true; }
112         leaf PolicyRuleName     { mandatory true; type string; config true; }
113         leaf CreationClassName      { mandatory true; type string; config true; }
114         leaf PolicyConditionName    { mandatory true; type string; config true; }
115
116         leaf ORed { type boolean; config true; }
117       }
118
119       container UA_MPLSClassifier {
120         description
121           "UA_MPLSClassifier : UA_Condition";
122
123         leaf labelStackID     { type uint32; config true; }
124
125         leaf InstanceID       { type string; config true; }
126         leaf Caption        { type string; config true; }
127         leaf Description      { type string; config true; }
128         leaf ElementName      { type string; config true; }
129         leaf CommonName       { type string; config true; }
130         leaf-list PolicyKeywords    { type string; config true; }
131
132         leaf SystemCreationClassName    { mandatory true; type string; config
              true; }
133         leaf SystemName       { mandatory true; type string; config true; }
134         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
              true; }
135         leaf PolicyRuleName     { mandatory true; type string; config true; }
136         leaf CreationClassName      { mandatory true; type string; config true; }
137         leaf PolicyConditionName    { mandatory true; type string; config true; }
138
139         leaf ORed { type boolean; config true; }
140       }
141
142       container UA_NullCondition {
143         description
144           "UA_NullCondition : UA_Condition";
145
146         leaf InstanceID       { type string; config true; }
147         leaf Caption        { type string; config true; }
148         leaf Description      { type string; config true; }
149         leaf ElementName      { type string; config true; }
```

```
150         leaf CommonName       { type string; config true; }
151         leaf-list PolicyKeywords    { type string; config true; }
152
153         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
154         leaf SystemName       { mandatory true; type string; config true; }
155         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
156         leaf PolicyRuleName     { mandatory true; type string; config true; }
157         leaf CreationClassName      { mandatory true; type string; config true; }
158         leaf PolicyConditionName    { mandatory true; type string; config true; }
159
160         leaf ORed { type boolean; config true; }
161       }
162
163       container UA_Node {
164         description
165           "UA_Node : UA_Condition";
166
167         leaf Name       { type string; config true; }
168         leaf Type       { type uint16; config true; }
169         leaf Configurable     { type boolean; config true; }
170         leaf Community        { type string; config true; }
171         leaf ControlEntityAddress   { type string; config true; }
172
173         leaf InstanceID       { type string; config true; }
174         leaf Caption        { type string; config true; }
175         leaf Description      { type string; config true; }
176         leaf ElementName      { type string; config true; }
177         leaf CommonName       { type string; config true; }
178         leaf-list PolicyKeywords    { type string; config true; }
179
180         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
181         leaf SystemName       { mandatory true; type string; config true; }
182         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
183         leaf PolicyRuleName     { mandatory true; type string; config true; }
184         leaf CreationClassName      { mandatory true; type string; config true; }
185         leaf PolicyConditionName    { mandatory true; type string; config true; }
186
187         leaf ORed { type boolean; config true; }
188       }
189
190       container UA_ServiceClass {
191         description
192           "UA_ServiceClass : UA_Condition";
193
194         leaf ServiceClassID     { type int32; config true; }
195         leaf Name       { type string; config true; }
196         leaf Reserved       { type boolean; config true; }
197         leaf Utilization      { type uint8; config true; }
198         leaf UtilizationTarget      { type uint16; config true; }
199         leaf ExtraUtilizationTarget   { type uint16; config true; }
200
201         leaf InstanceID       { type string; config true; }
202         leaf Caption        { type string; config true; }
```

```
203         leaf Description      { type string; config true; }
204         leaf ElementName      { type string; config true; }
205         leaf CommonName       { type string; config true; }
206         leaf-list PolicyKeywords    { type string; config true; }
207
208         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
209         leaf SystemName       { mandatory true; type string; config true; }
210         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
211         leaf PolicyRuleName     { mandatory true; type string; config true; }
212         leaf CreationClassName      { mandatory true; type string; config true; }
213         leaf PolicyConditionName    { mandatory true; type string; config true; }
214
215         leaf ORed { type boolean; config true; }
216       }
217
218       container UA_ATMClassifier {
219         description
220           "UA_ATMClassifier : UA_Condition";
221
222         leaf vci        { type uint32; config true; }
223         leaf vpi        { type uint32; config true; }
224
225         leaf InstanceID       { type string; config true; }
226         leaf Caption        { type string; config true; }
227         leaf Description      { type string; config true; }
228         leaf ElementName      { type string; config true; }
229         leaf CommonName       { type string; config true; }
230         leaf-list PolicyKeywords    { type string; config true; }
231
232         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
233         leaf SystemName       { mandatory true; type string; config true; }
234         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
235         leaf PolicyRuleName     { mandatory true; type string; config true; }
236         leaf CreationClassName      { mandatory true; type string; config true; }
237         leaf PolicyConditionName    { mandatory true; type string; config true; }
238
239         leaf ORed { type boolean; config true; }
240       }
241
242       container UA_MAC8021Classifier {
243         description
244           "UA_MAC8021Classifier : UA_Condition";
245
246         leaf svlan        { type uint32; config true; }
247         leaf cvlan        { type uint32; config true; }
248
249         leaf InstanceID       { type string; config true; }
250         leaf Caption        { type string; config true; }
251         leaf Description      { type string; config true; }
252         leaf ElementName      { type string; config true; }
253         leaf CommonName       { type string; config true; }
254         leaf-list PolicyKeywords    { type string; config true; }
255
```

```
256         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
257         leaf SystemName       { mandatory true; type string; config true; }
258         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
259         leaf PolicyRuleName     { mandatory true; type string; config true; }
260         leaf CreationClassName      { mandatory true; type string; config true; }
261         leaf PolicyConditionName    { mandatory true; type string; config true; }
262
263         leaf ORed { type boolean; config true; }
264       }
265
266       container UA_DPIClassifier {
267         description
268           "UA_DPIClassifier : UA_Condition";
269
270         leaf ApplicationType      { type uint16; config true;}
271
272         leaf InstanceID       { type string; config true; }
273         leaf Caption        { type string; config true; }
274         leaf Description      { type string; config true; }
275         leaf ElementName      { type string; config true; }
276         leaf CommonName       { type string; config true; }
277         leaf-list PolicyKeywords    { type string; config true; }
278
279         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
280         leaf SystemName       { mandatory true; type string; config true; }
281         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
282         leaf PolicyRuleName     { mandatory true; type string; config true; }
283         leaf CreationClassName      { mandatory true; type string; config true; }
284         leaf PolicyConditionName    { mandatory true; type string; config true; }
285
286         leaf ORed { type boolean; config true; }
287       }
288     }
289
290     container UA_PolicyAction {
291       description
292         "UA_PolicyAction";
293
294       leaf ActionID { mandatory true; type uint32; config true; }
295
296       container UA_Shape {
297         description
298           "UA_Shape : UA_Action";
299
300         leaf InstanceID       { type string; config true; }
301         leaf Caption        { type string; config true; }
302         leaf Description      { type string; config true; }
303         leaf ElementName      { type string; config true; }
304         leaf CommonName       { type string; config true; }
305         leaf-list PolicyKeywords    { type string; config true; }
306
307         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
```

```
308         leaf SystemName       { mandatory true; type string; config true; }
309         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
310         leaf PolicyRuleName     { mandatory true; type string; config true; }
311         leaf CreationClassName      { mandatory true; type string; config true; }
312         leaf PolicyActionName     { mandatory true; type string; config true; }
313         leaf DoActionLogging      { mandatory true; type boolean; config true; }
314
315         leaf Precedence       { type uint16; config true; }
316       }
317
318       container UA_Mark {
319         description
320           "UA_Mark : UA_Action";
321
322         leaf InstanceID       { type string; config true; }
323         leaf Caption        { type string; config true; }
324         leaf Description      { type string; config true; }
325         leaf ElementName      { type string; config true; }
326         leaf CommonName       { type string; config true; }
327         leaf-list PolicyKeywords    { type string; config true; }
328
329         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
330         leaf SystemName       { mandatory true; type string; config true; }
331         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
332         leaf PolicyRuleName     { mandatory true; type string; config true; }
333         leaf CreationClassName      { mandatory true; type string; config true; }
334         leaf PolicyActionName     { mandatory true; type string; config true; }
335         leaf DoActionLogging      { mandatory true; type boolean; config true; }
336
337         leaf Precedence       { type uint16; config true; }
338       }
339
340       container UA_Forward {
341         description
342           "UA_Forward : UA_Action";
343
344         leaf InstanceID       { type string; config true; }
345         leaf Caption        { type string; config true; }
346         leaf Description      { type string; config true; }
347         leaf ElementName      { type string; config true; }
348         leaf CommonName       { type string; config true; }
349         leaf-list PolicyKeywords    { type string; config true; }
350
351         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
352         leaf SystemName       { mandatory true; type string; config true; }
353         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
354         leaf PolicyRuleName     { mandatory true; type string; config true; }
355         leaf CreationClassName      { mandatory true; type string; config true; }
356         leaf PolicyActionName     { mandatory true; type string; config true; }
357         leaf DoActionLogging      { mandatory true; type boolean; config true; }
358
359         leaf Precedence       { type uint16; config true; }
```

```
360       }
361
362       container UA_FindLowerLayer {
363         description
364           "UA_FindLowerLayer : UA_Action";
365
366         leaf InstanceID       { type string; config true; }
367         leaf Caption        { type string; config true; }
368         leaf Description      { type string; config true; }
369         leaf ElementName      { type string; config true; }
370         leaf CommonName       { type string; config true; }
371         leaf-list PolicyKeywords    { type string; config true; }
372
373         leaf SystemCreationClassName    { mandatory true; type string; config
              true; }
374         leaf SystemName       { mandatory true; type string; config true; }
375         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
              true; }
376         leaf PolicyRuleName     { mandatory true; type string; config true; }
377         leaf CreationClassName      { mandatory true; type string; config true; }
378         leaf PolicyActionName     { mandatory true; type string; config true; }
379         leaf DoActionLogging      { mandatory true; type boolean; config true; }
380
381         leaf Precedence       { type uint16; config true; }
382       }
383
384       container UA_ActionNull {
385         description
386           "UA_ActionNull : UA_Action";
387
388         leaf InstanceID       { type string; config true; }
389         leaf Caption        { type string; config true; }
390         leaf Description      { type string; config true; }
391         leaf ElementName      { type string; config true; }
392         leaf CommonName       { type string; config true; }
393         leaf-list PolicyKeywords    { type string; config true; }
394
395         leaf SystemCreationClassName    { mandatory true; type string; config
              true; }
396         leaf SystemName       { mandatory true; type string; config true; }
397         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
              true; }
398         leaf PolicyRuleName     { mandatory true; type string; config true; }
399         leaf CreationClassName      { mandatory true; type string; config true; }
400         leaf PolicyActionName     { mandatory true; type string; config true; }
401         leaf DoActionLogging      { mandatory true; type boolean; config true; }
402
403         leaf Precedence       { type uint16; config true; }
404       }
405
406       container UA_Police {
407         description
408           "UA_Police : UA_Action";
409
410         leaf InstanceID       { type string; config true; }
411         leaf Caption        { type string; config true; }
412         leaf Description      { type string; config true; }
```

```
413         leaf ElementName      { type string; config true; }
414         leaf CommonName       { type string; config true; }
415         leaf-list PolicyKeywords    { type string; config true; }
416
417         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
418         leaf SystemName       { mandatory true; type string; config true; }
419         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
420         leaf PolicyRuleName     { mandatory true; type string; config true; }
421         leaf CreationClassName      { mandatory true; type string; config true; }
422         leaf PolicyActionName     { mandatory true; type string; config true; }
423         leaf DoActionLogging      { mandatory true; type boolean; config true; }
424
425         leaf Precedence       { type uint16; config true; }
426       }
427
428       container UA_Drop {
429
430         description
431           "UA_Drop : UA_Action";
432
433         leaf InstanceID       { type string; config true; }
434         leaf Caption        { type string; config true; }
435         leaf Description      { type string; config true; }
436         leaf ElementName      { type string; config true; }
437         leaf CommonName       { type string; config true; }
438         leaf-list PolicyKeywords    { type string; config true; }
439
440         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
441         leaf SystemName       { mandatory true; type string; config true; }
442         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
                true; }
443         leaf PolicyRuleName     { mandatory true; type string; config true; }
444         leaf CreationClassName      { mandatory true; type string; config true; }
445         leaf PolicyActionName     { mandatory true; type string; config true; }
446         leaf DoActionLogging      { mandatory true; type boolean; config true; }
447
448         leaf Precedence       { type uint16; config true; }
449       }
450
451       container UA_Rate {
452         description
453           "UA_Rate : UA_Action";
454
455         leaf InstanceID       { type string; config true; }
456         leaf Caption        { type string; config true; }
457         leaf Description      { type string; config true; }
458         leaf ElementName      { type string; config true; }
459         leaf CommonName       { type string; config true; }
460         leaf-list PolicyKeywords    { type string; config true; }
461
462         leaf SystemCreationClassName    { mandatory true; type string; config
                true; }
463         leaf SystemName       { mandatory true; type string; config true; }
464         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
```

```
            true; }
465         leaf PolicyRuleName     { mandatory true; type string; config true; }
466         leaf CreationClassName      { mandatory true; type string; config true; }
467         leaf PolicyActionName     { mandatory true; type string; config true; }
468         leaf DoActionLogging      { mandatory true; type boolean; config true; }
469
470         leaf Precedence       { type uint16; config true; }
471       }
472
473       container UA_QoSSessionDeterministicCAC {
474         description
475           "UA_QoSSessionDeterministicCAC : UA_Action";
476
477         leaf InstanceID       { type string; config true; }
478         leaf Caption        { type string; config true; }
479         leaf Description      { type string; config true; }
480         leaf ElementName      { type string; config true; }
481         leaf CommonName       { type string; config true; }
482         leaf-list PolicyKeywords    { type string; config true; }
483
484         leaf SystemCreationClassName    { mandatory true; type string; config
              true; }
485         leaf SystemName       { mandatory true; type string; config true; }
486         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
              true; }
487         leaf PolicyRuleName     { mandatory true; type string; config true; }
488         leaf CreationClassName      { mandatory true; type string; config true; }
489         leaf PolicyActionName     { mandatory true; type string; config true; }
490         leaf DoActionLogging      { mandatory true; type boolean; config true; }
491
492         leaf Precedence       { type uint16; config true; }
493       }
494
495       container UA_QoSStaticCAC {
496         description
497           "UA_QoSStaticCAC : UA_Action";
498
499         leaf InstanceID       { type string; config true; }
500         leaf Caption        { type string; config true; }
501         leaf Description      { type string; config true; }
502         leaf ElementName      { type string; config true; }
503         leaf CommonName       { type string; config true; }
504         leaf-list PolicyKeywords    { type string; config true; }
505
506         leaf SystemCreationClassName    { mandatory true; type string; config
              true; }
507         leaf SystemName       { mandatory true; type string; config true; }
508         leaf PolicyRuleCreationClassName  { mandatory true; type string; config
              true; }
509         leaf PolicyRuleName     { mandatory true; type string; config true; }
510         leaf CreationClassName      { mandatory true; type string; config true; }
511         leaf PolicyActionName     { mandatory true; type string; config true; }
512         leaf DoActionLogging      { mandatory true; type boolean; config true; }
513
514         leaf Precedence       { type uint16; config true; }
515       }
516     }
```

```
517     }
518 }
519
520 /* EOF */
```

Apêndice C

NcPolicyProvider Makefile

```
1  TOP = ../../..
2  include $(TOP)/mak/config.mak
3
4  MODULE=1
5
6  SHARED_LIBRARY = NcPolicyManager
7
8  SOURCES = \
9  module.cpp \
10 resource.cpp \
11 getParser.cpp \
12 setParser.cpp \
13 nc_manager.cpp \
14 CIM_AlertIndication.cpp \
15 CIM_AlertInstIndication.cpp \
16 CIM_Indication.cpp \
17 CIM_IndicationFilter.cpp \
18 CIM_IndicationSubscription.cpp \
19 CIM_InstIndication.cpp \
20 CIM_ListenerDestination.cpp \
21 CIM_ManagedElement.cpp \
22 CIM_PolicyAction.cpp \
23 CIM_PolicyCondition.cpp \
24 CIM_Policy.cpp \
25 CIM_ProcessIndication.cpp \
26 CIM_ThresholdIndication.cpp \
27 CIM_IndicationFilter_Provider.cpp \
28 CIM_ListenerDestination_Provider.cpp \
29 NcPolicyManager.cpp \
30 NcPolicyManager_Provider.cpp \
31 NetConfSession.cpp \
32 NetConfSession_Provider.cpp \
33 UA_Action.cpp \
34 UA_ActionNull.cpp \
35 UA_ActionNull_Provider.cpp \
36 UA_AfterMLMStartup.cpp \
37 UA_AfterMLMStartup_Provider.cpp \
38 UA_ATMClassifier.cpp \
39 UA_ATMClassifier_Provider.cpp \
40 UA_Condition.cpp \
41 UA_DPIClassifier.cpp \
```

```
42 UA_DPIClassifier_Provider.cpp \
43 UA_Drop.cpp \
44 UA_Drop_Provider.cpp \
45 UA_EventSubscription.cpp \
46 UA_EventSubscription_Provider.cpp \
47 UA_FindLowerLayer.cpp \
48 UA_FindLowerLayer_Provider.cpp \
49 UA_Forward.cpp \
50 UA_Forward_Provider.cpp \
51 UA_IPClassifier.cpp \
52 UA_IPClassifier_Provider.cpp \
53 UA_LinkThreshold.cpp \
54 UA_LinkThreshold_Provider.cpp \
55 UA_MAC8021Classifier.cpp \
56 UA_MAC8021Classifier_Provider.cpp \
57 UA_Mark.cpp \
58 UA_Mark_Provider.cpp \
59 UA_MiscEvent.cpp \
60 UA_MiscEvent_Provider.cpp \
61 UA_MPLSClassifier.cpp \
62 UA_MPLSClassifier_Provider.cpp \
63 UA_Node.cpp \
64 UA_Node_Provider.cpp \
65 UA_NullCondition.cpp \
66 UA_NullCondition_Provider.cpp \
67 UA_OutOfMoney.cpp \
68 UA_OutOfMoney_Provider.cpp \
69 UA_Police.cpp \
70 UA_Police_Provider.cpp \
71 UA_PolicyAction.cpp \
72 UA_PolicyAction_Provider.cpp \
73 UA_PolicyCondition.cpp \
74 UA_PolicyCondition_Provider.cpp \
75 UA_PolicyIndication.cpp \
76 UA_PolicyIndication_Provider.cpp \
77 UA_PolicyRule.cpp \
78 UA_PolicyRule_Provider.cpp \
79 UA_QoSSessionDeterministicCAC.cpp \
80 UA_QoSSessionDeterministicCAC_Provider.cpp \
81 UA_QoSStaticCAC.cpp \
82 UA_QoSStaticCAC_Provider.cpp \
83 UA_Rate.cpp \
84 UA_Rate_Provider.cpp \
85 UA_ServiceClass.cpp \
86 UA_ServiceClass_Provider.cpp \
87 UA_ServiceThreshold.cpp \
88 UA_ServiceThreshold_Provider.cpp \
89 UA_Shape.cpp \
90 UA_Shape_Provider.cpp \
91 repository.cpp
92
93 LIBRARIES += xerces-c pthread ssl dl xml2 ncmanager util
94 DEFINES += -DEBUG
95
96 ####CIMPLE_CMPI_MODULE=1
97 ####CIMPLE_PEGASUS_MODULE=1
98 ## Allow building providers for either CMPI or Pegasus C++ provider itf
```

```
99 ## Changing Provider itfs requires reregistering the provider.
100 ifdef CIMPLE_PROVIDER_BUILD_CMPI
101     CIMPLE_CMPI_MODULE=1
102 else
103     CIMPLE_PEGASUS_MODULE=1
104 endif
105
106 include ../common.mak
107
108
109 include $(TOP)/mak/rules.mak
110
111 PROVIDERS = \
112 CIM_IndicationFilter \
113 CIM_ListenerDestination \
114 NcPolicyManager \
115 NetConfSession \
116 UA_ActionNull \
117 UA_AfterMLMStartup \
118 UA_ATMClassifier \
119 UA_DPIClassifier \
120 UA_Drop \
121 UA_EventSubscription \
122 UA_FindLowerLayer \
123 UA_Forward \
124 UA_IPClassifier \
125 UA_LinkThreshold \
126 UA_MAC8021Classifier \
127 UA_Mark \
128 UA_MiscEvent \
129 UA_MPLSClassifier \
130 UA_Node \
131 UA_NullCondition \
132 UA_OutOfMoney \
133 UA_Police \
134 UA_PolicyAction \
135 UA_PolicyCondition \
136 UA_PolicyIndication \
137 UA_PolicyRule \
138 UA_QoSSessionDeterministicCAC \
139 UA_QoSStaticCAC \
140 UA_Rate \
141 UA_ServiceClass \
142 UA_ServiceThreshold \
143 UA_Shape
144
145 regmod:
146   $(BINDIR)/regmod -c $(TARGET)
147
148 genprov:
149   $(BINDIR)/genprov $(PROVIDERS)
150
151 genmod:
152   $(BINDIR)/genmod NcPolicyManager $(PROVIDERS)
153
154 genclass:
155   $(BINDIR)/genclass -r $(PROVIDERS)
```

```
156
157 genproj:
158   $(BINDIR)/genproj NcPolicyManager $(PROVIDERS)
159
160 regmod-dump:
161   $(BINDIR)/regmod -d -c $(TARGET)
162
163 build:
164   make clean
165   make genclass
166   make genprov
167   make genmod
168   make genproj
169   make
170   make insmod
171   make regmod
```

Apêndice D

Manual de instalação do Open Pegasus

Este anexo descreve os passos necessários para configuração, compilação e instalação do OpenPegasus.

1. **Download**

   Download do servidor CIM

```
$ wget http://www.openpegasus.org//uploads/40/21822/pegasus-2.10.0.tar.gz
```

   Descomprimir o arquivo tar;

```
$ tar -xf pegasus-2.10.0.tar.gz
```

   O processo de instalação descrito é baseados na instalação de OpenPegasus num directório local (exemplo: /home/rmendes/workspace/pegasus_src).

2. **Requisitos**
   Antes de compilar, instalar ou executar OpenPegasus, certifique-se que possui o conjunto adequado de software que o OpenPegasus depende. Actualmente OpenPegasus tem as seguintes dependências:

   - GNUMAKE - Para simplificar a compilação do OpenPegasus em múltiplas plataformas, foram padronizados um conjunto de ferramentas de compilação, incluindo: GNUMAKE. Foi testado o gnumake 3.79.1 com sucesso, tanto em ambientes Windows como em Linux. Está disponível a partir de http://www.gnu.org.
   - FLEX and BISON - Estas ferramentas foram usadas para desenvolver o compilador MOF e analisador WQL, sendo apenas necessárias apenas para o desenvolvimento de parsers, não para a compilação do OpenPegasus. Bison versão 2.3 ou posterior e flex versão 2.5.4 ou posterior são necessários.

- DOC++ - A documentação OpenPegasus é gerada por uma combinação de ficheiros de texto e ficheiros de cabeçalho si. Esta documentação está formatado com DOC++ e GAWK. Estas ferramentas são necessárias para construir a documentação da interface definida.
- ICU Internationalization libraries - Estas bibliotecas são usadas como base para catálogos para a internacionalização de mensagens.
  Consulte o site da UTI (http://icu.sourceforge.net) para obter mais informações sobre essas bibliotecas.
- OpenSSL - Se pretende usar SSL sobre o protocolo de comunicação, as bibliotecas OpenSSL são necessários (http://www.openssl.org).
- OpenSLP - Se se optar por usar o OpenSLP como escolha de implementações de SLP, então terá de ser instalado e disponível para OpenPegasus. Consulte as variaveis de compilação PEGASUS_USE_OPENSLP e PEGASUS_OPENSLP_HOME no PEP 292, e no site da OpenSLP (http://www.openslp.org).
- zlib - Se se optar por comprimir o repositório, deve-se criar a variável de compilação PEGASUS_ENABLE_COMPRESSED_REPOSITORY, para isso será necessário instalar a ferramenta de compressão gzip (GNU zip). Consulte o ficheiro readme.compressed_repository na pasta de instalação do OpenPegasus.

3. **Pré-configuração de compilação**

- Primeiro que tudo, devem ser definidas as variáveis de ambiente que controlam a compilação do OpenPegasus (A lista completa das variáveis pode ser encontrada no site do OpenPegasus no documento PEP 277 (http://www.openpegasus.org/pp/uploads/40/6160/PEP174_RecommendedReleaseOptions.htm):
  - PEGASUS_ROOT: necessário, define o caminho para a pasta que contém o código fonte do OpenPegasus (Esta variável também deve ser definida para a execução)
  - PEGASUS_HOME: necessário, define o caminho para a pasta que irá conter o resultado da compilação (i.e. binários, bibliotecas, repositórios, etc)
  - PEGASUS_PLATFORM: necessário, define a plataforma sobre a qual compilar (Linux 32 bits: LINUX_IX86_GNU, Linux 64 bits da Intel: LINUX_IA64_GNU, Linux 64 bits da AMD: LINUX_X86_64_GNU, etc)
- **Suporte ExecQuery() (opcional)**
  - PEGASUS_ENABLE_EXECQUERY: suporte do método ExceQuery()
- **Suporte a SSL (opcional)**
  - PEGASUS_HAS_SSL: define o recurso ou não para o protocolo SSL (https) entre o cliente e o servidor (true ou false)
  - OPENSSL_HOME: necessário se o SSL estiver activado, define o caminho para a pasta que contém as bibliotecas e os binários do OpenSSL (ex: /usr)
- Autenticação & autorização do utilizador (opcional)
  - PEGASUS_PAM_AUTHENTICATION: habilita o suporte para autenticação baseada em PAM (Pluggable Authentication Modules) (true ou false)

  - PEGASUS_ENABLE_USERGROUP_AUTHORIZATION: permite ao administrador restringir o acesso às operações de CIM para os membros de um conjunto de grupos designados. (true ou false)

- Debug (opcional)
  - PEGASUS_DEBUG : Activa/desactiva flag de compilação para versão debug (true ou false)
  - PEGASUS_USE_SYSLOGS : Define se as mensagens de log são enviadas para o log de sistema (true ou false)
- As variáveis podem ser definidas como:

```
$ export <VARIABLE_NAME>=<VALUE>
```

- Ou (recomendado) editar o perfil do utilizador /home/rmendes/.bashrc ou de forma global /etc/bashrc
- Simultaneamente, pode ser extendido à variável PATH, adicionando o caminho para os binários do OpenPegasus.

```
$ export PATH=$PEGASUS_HOME/bin:$PATH
```

- Exemplo do ficheiro /home/rmendes/.bashrc:

```
... ...
# OpenPegasus definição da variável de ambiente
export PEGASUS_HOME=/home/rmendes/workspace/pegasus-src
export PEGASUS_ROOT=/home/rmendes/workspace/Pegasus
export PEGASUS_PLATFORM=LINUX_IX86_GNU

# OpenPegasus suporte SSL
export OPENSSL_HOME=/usr
export PEGASUS_HAS_SSL=yes

# OpenPegasus autenticação e autorização
export PEGASUS_ENABLE_USERGROUP_AUTHORIZATION=true
export PEGASUS_PAM_AUTHENTICATION=true

# OpenPegasus suporte de metodo ExecQuery()
export PEGASUS_ENABLE_EXECQUERY=true

# OpenPegasus debug
export PEGASUS_DEBUG=true
export PEGASUS_USE_SYSLOGS=true

# PATH
export PATH=$PEGASUS_HOME/bin:$PATH
```

```
... ...
```

4. **Compilação**

   - Depois de garantidas as dependências e definição das variáveis de ambiente pode-se proceder a compilação. No directório fonte do OpenPegasus executar o seguinte:

```
$ make
$ make repository
```

   - Todos os ficheiros executáveis, bibliotecas e configuração serão criados no directório definido por PEGASUS_HOME.

5. **Funcionamento**

   - Para arrancar o serviço do OpenPegasus, executar :

```
$ cimserver
```

   - Para parar o serviço OpenPegasus, executar :

```
$ cimserver -s
```

   - Para arrancar o serviço OpenPegasus como aplicação, executar :

```
$ cimserver daemon=false
```

   - Outros comandos úteis:
     - *cimconfig* : configuração de execução do OpenPegasus
     - *cimprovider* enumeração e gestão dos providers registados

6. **Configurando o repositório CIM**
   Se for pretendido actualizar o modelo CIM antes da instalação dos providers.

   - Remover o namespace root/cimv2 existente:

```
$ rm -rf $PEGASUS_HOME/repository/root#cimv2
```

   - Fazer o download do site do DMTF da última versão do modelo CIM (arquivo zip do código do MOF, é possível optar pela versão final ou experimental) (http://www.dmtf.org/standards/cim/).
   - Descomprimir o arquivo zip descarregado (em qualquer lugar, mas não na pasta do repositório!):

```
$ unzip cimvMm-MOFs.zip
```

Na pasta onde o modelo CIM foi descomprimido deverá existir um arquivo chamado cimvMm.mof.

- Crie o namespace root/cimv2 e preencha-o com o modelo CIM (-aE para permitir o modelo experimental):

```
$ cimmof -aE -n root/cimv2 cimvMm.mof
```

Nota: O cimserver deve ser iniciado antes de fazer esta operação.

O próximo passo é necessário para estar de acordo com o perfil DMTF para o registo.

- Crie o namespace Interop e preencha-o com modelo CIM (a versão do modelo deve ser a mesma da criação do namespace root/cimv2):

```
$ cimmof -aE -n Interop cimvMm.mof
```

Nota: O cimserver deve ser iniciado antes de executar esta operação.

7. **Configuração do OpenPegasus (opcional, deve ser iniciado o cimserver previamente)**
   De modo a garantir a administração do sistema de forma segura as seguintes medidas devem ser aplicadas.

   - As chaves privadas devem ser mantidos em segredo e os certificados devem ser emitidos por entidades confiáveis, caso contrário, a criptografia de dados é inútil.
   - Autenticação e autorização dos utilizadores
     - O root pode sempre emitir pedidos CIM.
     - $ cimconfig -s enableAuthentication=true -p // habilita (recomendado) / desabilita a autenticação do utilizador (necessário reiniciar o OpenPegasus).
     - $ cimconfig -s authorizedUserGroups=<group1>,...,<groupN> -p // define a autorização do utilizador, authorizedUserGroups deve ser a lista de grupos de utilizadores, cujos membros podem emitir pedidos CIM (necessário reiniciar o OpenPegasus).
     - Por exemplo, criar um grupo chamado pegasususers : $ groupadd pegasususers. pegasususers deve ser adicionado ao authorizedUserGroups. Se o utilizador rmendes pretende emitir pedidos CIM, este dever ser adicionado ao grupo: $ usermod-G pegasususers rmendes.
   - Ligaçoes SSL
     - $ cimconfig -s enableHttpConnection=true/false -p // activa/desactiva (recomendado) ligações HTTP inseguras (necessário reiniciar o OpenPegasus).
     - $ cimconfig -s enableHttpsConnection=true/false -p // activa (recomendado) /desactiva ligações HTTPS seguras (necessário reiniciar o OpenPegasus).

  - Autenticação do servidor CIM
  - Autenticar um servidor CIM a partir de um cliente CIM.
  - A chave privada do servidor CIM e o certificado vêm por por defeito da instalação do OpenPegasus.
  - $ cimconfig -g sslCertificateFilePath // recebe o caminho para o certificado de servidor actual.
  - $ cimconfig -s sslCertificateFilePath=<newPath> -p // define um novo certificado para o servidor (necessário reiniciar o OpenPegasus).
  - $ cimconfig -g sslKeyFilePath // obtém o caminho para a chave privada do servidor.
  - $ cimconfig -s sslKeyFilePath // define uma nova chave privada para o servidor(necessário reiniciar o OpenPegasus).
  - O administrador deve obter o certificado de servidor a partir de uma entidade confiável. O cliente CIM irá utilizar este certificado para identificar o servidor CIM. Note que o cliente CIM também pode optar por confiar no servidor CIM e descartar o processo de autenticação, a seu próprio risco.
- Autenticação do cliente CIM
  - Autenticar um cliente CIM a partir de um servidor CIM.
  - $ cimconfig -g sslClientVerificationMode // obtém a política de autenticação do cliente.
  - $ cimconfig -s sslClientVerificationMode=<newPolicy> -p // define a política de autenticação do cliente (desnecessário, facultativo ou necessário (valor recomendado), necessário reiniciar o OpenPegasus).
  - Os certificados de confiança do cliente são armazenadas num sitio de confiança, que pode ser um único ficheiro ou um directório.
  - $ cimconfig -g sslTrustStore // obtém a localização dos certificados
  - $ cimconfig -s sslTrustStore=<newPath> -p // define a localização dos certificados (necessário reiniciar o OpenPegasus).
  - $ cimconfig -s sslTrustStoreUserName=<trustedUser> -p // define o nome do (único) utilizador de confiança do cliente CIM (obrigatório se os certificados estão contidos num único ficheiro, necessário reiniciar o OpenPegasus).
- Geração de chaves privadas / certificados (RSA)
  - $ openssl genrsa -out private_key.pem <private_key_modulus> // maior o módulo, mais forte é a criptografia (geralmente 512 ou 1024)
  - $ openssl req -new -key private_key.pem -out certificate_request.csr
  - $ openssl x509 -in certificate_request.csr -out certificate.pem -req -signkey private_key.pem -days <number of days> // definir o número de dias ate à data de expiração do certificado

Apêndice E

Manual de instalação do Cimple

Este anexo descreve os passos necessários para configuração, compilação e instalação do Cimple.

1. **Download**

   Download do CIMPLE

```
$ wget http://simplewbem.org/cimple-2.0.14.tar.gz
```

   Descomprimir o arquivo tar;

```
$ tar -xf cimple-2.0.14.tar.gz
```

2. **Pré-configuração de compilação**

   Uso: *./configure [OPÇÃO]...*

   Exemplos de configuração:

   - Compilar o CIMPLE de forma independente.

```
$ ./configure
```

   - Compilar o CIMPLE com suporte para o Pegasus (DIR deve conter os directórios bin e lib do Pegasus).

```
$ ./configure –with-pegasus=DIR
```

   - Compilar o CIMPLE com suporte CMPI (DIR deve conter os ficheiros de cabeçalho CMPI).

```
$ ./configure –with-cmpi=DIR
```

- Compilar o CIMPLE com suporte para o OpenWBEM (DIR deve conter os directórios bin e lib do OpenWBEM).

```
$ ./configure –with-openwbem=DIR
```

- Compilar o CIMPLE para uso com uma distribuição fonte do Pegasus usando as variáveis de ambiente do Pegasus para obter automaticamente as opções.

```
$ ./configure –with-pegasus-env
```

- Compilar o CIMPLE para uso com uma distribuição fonte Pegasus.

```
$ ./configure –prefix=$PEGASUS_HOME
–with-pegasus=$PEGASUS_HOME
–with-cmpi=$PEGASUS_ROOT/src/Pegasus/Provider/CMPI
```

- Compilar o CIMPLE para uso com uma distribuição fonte do Pegasus em plataformas de 64 bits, caso em que o directório da biblioteca Pegasus é misnamed (Ou seja, 'lib' em vez de "lib64).

```
$ ./configure –prefix=$PEGASUS_HOME
–libdir=$PEGASUS_HOME/lib
–with-pegasus=$PEGASUS_HOME
–with-pegasus-libdir=$PEGASUS_HOME/lib
–with-pegasus-includes=$PEGASUS_ROOT/src
–with-cmpi=$PEGASUS_ROOT/src/Pegasus/Provider/CMPI
```

Para obter a lista de opções:

```
$ ./configure –help
```

3. **Compilar**

```
$ make
```

4. **Instalar**

```
$ make install
```

Apêndice F

Manual de instalação do Netopeer

Este anexo descreve os passos necessários para configuração, compilação e instalação do Netopeer.

- **Download**

  Download do Netopeer

```
$ wget http://netopeer.googlecode.com/files/netopeer-1.0.1.tar.gz
```

  Descomprimir o arquivo tar;

```
$ tar -xf netopeer-1.0.1.tar.gz
```

- **Requisitos**

  Comuns

  gcc >= 3.4.3 (http://gcc.gnu.org)

  libxml2 >= 2.6.16 (http://xmlsoft.org)

  NETCONF *Manager*

  ssh (http://www.openssh.com)

  NETCONF *Agent*

  sshd (http://www.openssh.com)

  libxslt >= 1.1.11 (http://xmlsoft.org/XSLT/)

  libdbus-1 >= 1.0.0 (http://dbus.freedesktop.org)

- **Pré-configuração de compilação**

  Uso: *./configure [OPÇÃO]... [VAR=VALOR]...*

[1]OpenSSH 2.0 ou posterior implementação SSHv2 com mecanismo SSH Subsystem

[1]OpenSSH 2.0 ou posterior implementação SSHv2 com mecanismo SSH Subsystem

–with-user=USER
Define o utilizador dono dos ficheiro de configuração presentes em */etc/netopeer*;

–with-group=GROUP
Define o grupo dos ficheiro de configuração presentes em */etc/netopeer*;

–with-libxml2-path=DIR
Define o caminho para biblioteca libxml2 se ela não estiver instalada no directório por defeito do sistema;

–with-libxslt-path=DIR
Define o caminho para biblioteca libxslt se ela não estiver instalada no directório por defeito do sistema;

–with-libdbus-path=DIR
Define o caminho para biblioteca libdbus-1 se ela não estiver instalada no directório por defeito do sistema.

–with-dbus-config-path=[/etc/dbus-1]
Define o caminho para os ficheiros de configuração D-Bus se estes não estiveram instaladas no directório por defeito do sistema;

–disable-server
Desactiva a verificação de bibliotecas e ferramentas necessárias ao servidor Netopeer, inibindo a sua instalação;

–disable-client
Desactiva a verificação de bibliotecas e ferramentas necessárias ao cliente Netopeer, inibindo a sua instalação.

Para obter a lista de opções:

```
$ ./configure –help
```

- **Compilar**

  O Netopeer disponibiliza o NETCONF *Manager* com uma interface *CLI*, é necessário converter o NETCONF *Manager* num biblioteca que possa ser invocada pelo *provider*. A *Makefile* terá de ser modificada de modo gerar e instalar a biblioteca, de acordo com alterações a vermelho.

```
1  srcdir     = .
2  DESTDIR    =
3  CC     = gcc
4  CFLAGS     =  -fPIC -g -DDEBUG -Wall -O2 -I/usr/include/libxml2 -I/usr/
       include/libxml2 -I/usr/include/dbus-1.0 -I/usr/lib/dbus-1.0/include
5  CPPFLAGS   =
6  LDFLAGS    =  -L/lib -L/usr/lib -L/usr/local/lib
7  prefix     = /usr/local
8  INSTALL    = /usr/bin/install -c
9  OWNER      = root
10 GROUP      = root
11 DBUSCONF   = /etc/dbus-1
12 PREFIX     = $(DESTDIR)/$(prefix)
13 MANDIR     = $(DESTDIR)/${datarootdir}/man
```

```
14 datarootdir   = ${prefix}/share
15 COMMON_OBJ  = debug.o config.o
16 MANAGER_OBJ = nc_manager.o nc_manager_general.o nc_manager_op.o
       nc_manager_op_powerctl.o nc_misc.o $(COMMON_OBJ)
17 MANAGER_LIB_OBJ = nc_manager_general.o nc_manager_op.o
       nc_manager_op_powerctl.o nc_misc.o $(COMMON_OBJ)
18 AGENT_OBJ = nc_agent.o nc_agent_op.o nc_agent_op_shared.o
       nc_agent_op_powerctl.o nc_misc.o $(COMMON_OBJ) dbus_bindings.o
19 FAKESRV_OBJ = nc_fake_daemon.o $(COMMON_OBJ) dbus_bindings.o
20 LIBS    =  -lxml2 -lxslt -ldbus-1
21 CLIENT_PRGS     = netopeer libncmanager.so.1.0.1
22 SERVER_PRGS     = netopeer-agent netopeer-daemon-fake
23 OBJECTS         = $(SERVER_PRGS) $(CLIENT_PRGS)
24
25 .c.o:
26   $(CC) -c $(CPPFLAGS) $(CFLAGS) $<
27
28 all: $(OBJECTS)
29
30 netopeer: $(MANAGER_OBJ)
31   $(CC) $(CFLAGS) -o $@ $(MANAGER_OBJ) $(LDFLAGS) -lutil $(LIBS)
32
33 libncmanager.so.1.0.1:  $(MANAGER_LIB_OBJ)
34   $(CC) -shared -Wl,-soname,libncmanager.so.1 -o libncmanager.so.1.0.1
       $(MANAGER_LIB_OBJ) -lc
35
36 netopeer-agent: $(AGENT_OBJ)
37   $(CC) $(CFLAGS) -o $@ $(AGENT_OBJ) $(LDFLAGS) $(LIBS)
38
39 netopeer-daemon-fake: $(FAKESRV_OBJ)
40   $(CC) $(CFLAGS) -o $@ $(FAKESRV_OBJ) $(LDFLAGS) $(LIBS)
41
42 install: $(OBJECTS)
43   @test -d $(PREFIX) || $(INSTALL) -d -o root -g root -m 755 $(PREFIX)
44   @test -d $(PREFIX)/bin || $(INSTALL) -d -o root -g root -m 755 $(PREFIX)/
       bin
45   @test -d $(MANDIR) || $(INSTALL) -d -o root -g  root -m 755 $(MANDIR)
46   @test -d $(MANDIR)/man1 || $(INSTALL) -d -o root -g root -m 755 $(MANDIR)/
       man1
47   @if [ -n "$(SERVER_PRGS)" ]; then \
48     test -d $(MANDIR)/man5 || $(INSTALL) -d -o root -g root -m 755 $(MANDIR)
         /man5; \
49     test -d /etc/netopeer || $(INSTALL) -d -o $(OWNER) -g $(GROUP) -m 775 /
         etc/netopeer; \
50     test -d /etc/netopeer/xsl || $(INSTALL) -d -o $(OWNER) -g $(GROUP) -m
         775 /etc/netopeer/xsl; \
51     test -d /usr/local/include/netopeer || $(INSTALL) -d -o $(OWNER) -g
         $(GROUP) -m 775 /usr/local/include/netopeer; }
52   fi;
53   for file in $(OBJECTS); do \
54     $(INSTALL) -o $(OWNER) -g $(GROUP) -m 755 $$file $(PREFIX)/bin; \
55     if [ -f $$file.1 ]; then \
56       $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 $$file.1 $(MANDIR)/man1; \
57     fi \
58   done;
59   if [ -n "$(SERVER_PRGS)" ]; then \
60     $(INSTALL) -o $(OWNER) -g $(GROUP) -m 664 ./config/netopeer.conf /etc/
```

```
        netopeer; \
61      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 664 ./config/*_startup.xml /etc/
        netopeer; \
62      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 664 ./xsl/* /etc/netopeer/xsl; \
63      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 664 ./config/netopeer-dbus.conf $(
        DBUSCONF)/system.d; \
64      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 664 ./config/netopeer-dbus.conf $(
        DBUSCONF)/session.d; \
65      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 ./config/netopeer.conf.5 $(
        MANDIR)/man5; \
66      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 debug.h
        /usr/local/include/netopeer; }
67      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 nc_manager_op.h
        /usr/local/include/netopeer; }
68      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 nc_misc.h
        /usr/local/include/netopeer; }
69      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 nc_manager_general.h
        /usr/local/include/netopeer; }
70      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 nc_manager_op_powerctl.h
        /usr/local/include/netopeer; }
71      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 nc_protocol.h
        /usr/local/include/netopeer; }
72      $(INSTALL) -o $(OWNER) -g $(GROUP) -m 644 libncmanager.so.1.0.1
        /usr/local/lib; }
73      ln -s /usr/local/lib/libncmanager.so.1.0.1
        /usr/local/lib/libncmanager.so.1; }
74      ln -s /usr/local/lib/libncmanager.so.1.0.1
        /usr/local/lib/libncmanager.so; }
75    fi;
76
77  uninstall:
78    for file in $(OBJECTS); do \
79      /bin/rm -f  $(PREFIX)/bin/$$file ; \
80      if [ -f $$file.1 ]; then \
81        /bin/rm -f  $(MANDIR)/man1/$$file.1 ; \
82      fi \
83    done;
84    if [ -n "$(SERVER_PRGS)" ]; then \
85      /bin/rm -f $(DBUSCONF)/system.d/netopeer-dbus.conf; \
86      /bin/rm -f $(MANDIR)/man5/netopeer.conf.5; \
87      /bin/rm -rf /etc/netopeer; \
88      /bin/rm -rf /usr/local/include/netopeer; }
89      /bin/rm -f /usr/local/lib/libncmanager.*:  }
90    fi;
91
92  clean:
93    rm -f *.o *core* $(OBJECTS)
```

```
$ make
```

- **Instalar**

```
$ make install
```

- **Configuração do servidor**

  O *Netopeer Agent* é executado como um subsistema SSH. Para activar o subsistema SSH, designado por netconf, é necessário editar o ficheiro de configuração do serviço SSH. O ficheiro de configuração é o */etc/ssh/sshd_config*. Para activar o subsistema netconf é necessário adicionar as seguintes linha:

  Subsystem netconf /usr/local/bin/netopeer-agent

  em que a última parte corresponde ao caminho onde o Netopeer Agent está instalado. A sessão NETCONF é estabelecida por defeito no porto 830. É possível alterar esta porto com a opção "*-p*", fornecida ao programa cliente *netopeer* e utilizar por exemplo a porta 22, aberta por defeito pelo SSH. Para abrir a porta 830 para o SSH é necessário editar o ficheiro */etc/ssh/sshd_config* e acrescentar uma linha que especifica o número de porta que o sshd escuta.

  Port 22
  Port 830

  A comunicação entre o *Netopeer Agent* e o serviço configuração (netopeer-daemon-fake para fins de teste) é efectuada através do sistema de D-Bus. Não esquecer de arrancar o serviço *dbus* antes de usar o Netopeer.

```
$/etc/init.d/dbus start
```