UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Aujor Tadeu Cavalca Andrade

DISTRIBUIÇÃO DE POLÍTICAS DE GERENCIAMENTO DE REDES COM ÊNFASE EM QUALIDADE DE SERVIÇO

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Carlos Becker Westphall

Florianópolis, maio de 2005.

DISTRIBUIÇÃO DE POLÍTICAS DE GERENCIAMENTO DE REDES COM ÊNFASE EM QUALIDADE DE SERVIÇO

Aujor Tadeu Cavalca Andrade

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação área de concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Dr. Raul Sidnei Wazlawick
Coordenador do PGCC

Banca Examinadora

_____

Prof. Dr. Carlos Becker Westphall
Presidente

_____

Prof. Dr. Mario Antonio Ribeiro Dantas

_____

Prof. Dr. Vitório Bruno Mazzola

_____

Prof. Dr. Joni da Silva Fraga

“Quando não se tem rumo não adianta ventos a favor...”.

(Sócrates)

Dedico este trabalho à meu pai Aujor, minha mãe Ivani, meu irmão Eduardo, minha irmã Melissa, pelo amor com que me criaram, pelo caráter que me ensinaram e por o privilegio de poder estudar. Também a Andressa pelo amor e compreensão. Neste trabalho cada um deles ajudou de forma direta e indireta. Esta dedicação é parte do meu eterno agradecimento às pessoas que mais amo.

Agradecimentos

Agradece a Deus pela minha família e pelo privilégio de poder estudar. Também ao meu pai Aujor, Eduardo e Melissa pelo apóia, compreensão e em especial a minha amada mãe Ivani por seu incentivo, orientação, exemplo e amor.

Também Andressa pelas vezes que deixamos de nos ver para realização deste trabalho, pelo seu amor, carinho e compreensão.

Ao prof. Carlos B. Westphall pelo acolhimento no LRG, orientação e apóio na dissertação em nos artigos. Também aos amigos do LRG Valério, Marcos, Cleber, Two, Parra, Silvia, Carla, André e Fabiano pela apóia, troca de idéias e amizade. Ao meu amigo Luiz pelas conversas e ajuda fundamental.

A secretária do PGCC em especial a Verinha pela sua simpatia e eficiência. Aos professores do programa pelo conhecimento e aos membros da banca pela participação.

Por fim, agradeço a todos os amigos que me ajudaram nesta jornada. Muito obrigado!

Sumário

Lista de Figuras

Lista de Siglas

APIs	*Application Program Interface*
CIM	*Common Information Model.*
COPS	*Common Open Police Service*
CORBA	*Common Object Request Broker Architecture*
DBMS	*Database Management System*
DoD	*Department of Defense of United States.*
DEN	*Directory Enabled Networks.*
DMTF	*Distributed Management Task Force.*
FTP	*File Transfer Protocol*.
HTTP	*Hyper Text Transfer Protocol*
IETF	*Internet Engineering Task Force*
LDAP	*Lightweight Directory Access Protocol*
LPDP	*Local Policy Decision Point*
MIB	*Management Information Base*
MOF	*Management Object Format*
MIF	*Management Information Format*
OOPS	*Open Outsourcing Policy Service*.
PBNM	*Policy-based Network Management*
PDP	*Policy Decision Points*
PEP	*Policy Enforcement Point*
PRC	*Provisioning Class*
PRI	*Instance Provisioning*
QoS	*Quality of Services*
RSVP	*Resource Reservation Protoco.*
RAP	*Resource Alocation Protocol*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SGML	*Structured General Markup Language*

SLA	*Service Level Agreement*
SLO	*Service Level Object*
SMI	*Structure of Management Information*
SNMP	*Simple Network Management Protocol*
SOAP	*Simple Object Access Protocol.*
SPBNM	*System Policy-based Network Management*
SPPI	*Structure of Policy Provisioning Information.*
UDDI	*Universal Description and Integration*
URI	*Uniform Resource Locator*
XML	*Extensible Markup Language*.
WSDL	*Web Service Description Language*
W3C	*World Wide Web Consortium*

Resumo

O gerenciamento redes tem com finalidade conservar o funcionamento da rede como um todo, desde os serviços apresentados aos usuários à transmissão dos dados. Conforme a evolução das redes, o gerenciamento assumiu papel cada vez mais destacado, onde os administradores de rede aprendem que adicionar largura de banda não é capaz de resolver problemas relacionados ao tráfego. Logo, reservar ou diferenciar recursos não é garantir qualidade de serviços. Neste contexto desenvolver novas formas de gerenciamento de rede é fundamentalmente importante.

O gerenciamento baseado em políticas surgiu da necessidade de melhorar a gerência de recursos e dispositivos, auxiliar a garantir qualidade de serviço (QoS – *Quality of Service*), segurança e outros atributos necessários à otimização da rede. A sua funcionalidade proporciona maior eficiência no gerenciamento, permitindo administrar redes à medida que elas se tornem distribuídas, controlar dispositivos e recursos a grandes distâncias, tipos de requisições com diferentes latências e propriedades de segurança.

Esta dissertação tem como objetivo propor a extensão do uso do gerenciamento baseado em políticas de rede. Para tanto, foi criado uma abordagem para distribuição de políticas de gerenciamento com ênfase na qualidade de serviço. A qual comunica os dispositivos gerenciados ao ponto de decisão, requerendo ações corretivas para dispositivos, reportar mudanças nas configurações, registrar os recursos gerenciados e aplicar as políticas definidas para o gerenciamento da rede. Neste contexto é definido o formato da mensagem em XML, tendo como meio difusor das políticas o HTTP entre os participantes do sistema distribuído.

Como contribuição este trabalho permitiu maior flexibilidade na distribuição de políticas para os dispositivos e o compartilhamento de um conjunto lógico de políticas para o gerenciamento de recursos e aplicações.

Palavras-chave: Gerenciamento por políticas, gerência de redes e PBNM.

Abstract

The technological development has amplified the availability of the communication service propiciating a non-precedent growth in the creation of computer networks. The services and resources shared through a common mean of transmission were one of the main propeller factors. To turn this into an efficient structure, the network management is of fundamental importance.

The network management has as its objective to conserve the functioning of the network as a whole, from the services presented to the users until data transmission. Conform network evolution, its management assumed an even more important rolr, where the network administrators learn that the addition of bandwidth is not capable of solving traffic related problems nor guarantee the quality of service to the applications /services as not only to reserve or difference the resources.

The policy-based management arised from the necessity of improvement in the management of devices and resources , helping guarantee the quality of service (QoS – *Quality of Service*), security and other attributes necessary for network optimization. Its functionality proportionate a bigger efficiency in the management, allowing network administration as they become more distributed, control of devices and resources at great distances, types of requisitions with different latencies and security proprieties.

This dissertation has as its objective to propose the extension of network policy-based management usage. For this, the creation of a policies distribution protocol for the communication among the policy decision point and policy enforcement point is intended, a protocol capable of requiring the corrective actions for devices, reporting changes in configurations, registering the managed resources and applying the policies defined for the network management. In this context the message format is defined in XML having the HTTP as a mean of policy diffusion among the participants of the management distributed system.

Through the defined patterns by the IETF, this work will allow more flexibility in the distribution policy for the devices and sharing of a logic policy group for the resources and applications management.

Key-words: Policy management, network management and PBNM.

1. Introdução

1.1 Desafio

O desenvolvimento tecnológico ampliou a disponibilidade de serviço de comunicação propiciando um crescimento sem precedentes na criação de redes de computadores. Os serviços e recursos compartilhados através de meio comum de transmissão foram os principais fatores propulsores. Para tornar essa estrutura eficiente o gerenciamento de redes é fundamentalmente importante.

O gerenciamento redes tem com finalidade conservar o funcionamento da rede como um todo, desde os serviços apresentados aos usuários à transmissão dos dados. Conforme evolução das redes o gerenciamento assumiu papel cada vez mais destacado, onde os administradores de rede aprendem que adicionar largura de banda não é capaz de resolver problemas relacionados ao tráfego e garantir qualidade de serviço (QoS) as aplicações/serviços não é somente reservar ou diferenciar recursos.

Quando há solicitação de um serviço, é essencial conferir os recursos necessários e verificar a disponibilidade para a execução do mesmo. Neste contexto, especificar parâmetros de comportamento para cada elemento, suas características e funcionalidades de forma a ajudar inteligentemente a administração de recurso da rede, por meio de um conjunto de regras que trata dos procedimentos de aplicação e usuários de modo adaptável a novas condições é a proposta do gerenciamento de redes baseadas em políticas – PBNM (*Policy-Based Network Management*).

1.2 Estado da Arte

O gerenciamento baseado em políticas vem sendo pesquisado pela comunidade científica, onde se destacam benefícios no gerenciamento, configuração e desempenho dos recursos. Também na criação de uma linguagem para especificação de políticas entre outras otimizações.

A criação de um grupo de trabalho pela IETF (*Internet Engineering Task Force*) para a pesquisa específica de gerenciamento baseado em políticas (Grupo de Políticas) vem ganhando grande destaque, impulsionando o desenvolvimento e aperfeiçoamento da gerência de redes. Um dos principais grupos de pesquisa de gerenciamento em sistemas distribuídos é o do *Imperial College* em Londres liderado por *Morris Sloman* [Sloman, 1994] um dos grandes contribuintes na distribuição de conceitos e aplicações de política.

Existem vários trabalhos em andamento sobre gerenciamento baseado em políticas, onde a empenho na padronização de uma linguagem de especificação sintática e semântica para construção de políticas, [Strassner, 1999]. Outra corrente estuda a definição dos mecanismos usados para armazenamento e distribuição das políticas de gerenciamento. As soluções mais utilizadas para distribuição de políticas são o COPS (*Common Open Police Service*) [Boyle, 1999], SNMP (*Simples Network Management Protocol*) [RFC 1157] e HTTP (*Hyper Text Transfer Protocol*) [RFC 2597]. Para o armazenamento de políticas no repositório LDAP (*Lightweight Directory Acess Protocol*) [RFC 2370].

Um dos principais esforços do grupo de políticas esta sendo em pesquisas relacionados à qualidade de serviço [Flegkas, 2002] e na alocação de recurso com segurança na rede [Sloman, 2002], onde há crescente necessidade do mercado. Grandes empresas participam deste grupo, nos quais as especificações rapidamente viram soluções de mercado.

O gerenciamento baseado em políticas é um novo campo de gerenciamento de redes, o qual fornece o caminho específico para a escolha de recursos de rede, QoS e segurança considerando previamente as condições da rede e definições do conjunto de políticas.

1.3 Motivação

O gerenciamento baseado em políticas surgiu da necessidade de melhorar a gerência de recursos e dispositivos, auxiliar a garantir qualidade de serviço, segurança e outros atributos necessários à otimização da rede. A sua funcionalidade proporciona maior eficiência no gerenciamento, permitindo administrar redes à medida que elas se

tornam mais distribuídas, controlar dispositivos e recursos a grandes distâncias, tipos de requisições com diferentes latências e propriedades de segurança.

Uma rede baseada em políticas especifica a relação entre as aplicações, recursos e usuários através de um conjunto de regras (políticas). Esse conjunto de regras é definido pelo administrador da rede. As políticas estão baseadas no conhecimento das atividades da empresa e na alocação de recursos pelos usuários da rede. Parafraseando *IEEE Network Magazine*: “as políticas podem ser usadas para se obter uma melhor escala no gerenciamento da rede através da descrição de atributos comuns de classes de objetos, tipicamente associados com o papel que representam, tais como dispositivos de rede, serviços de *software* e usuários, em vez de definir individualmente os atributos destes elementos”.

Segundo definições a utilização da gerência baseada em políticas, propõe melhoramento na gerência de dispositivos, recursos e minimiza a complexidade da gerência [RFC 3198]. A concepção de um conjunto de regras para controle de recursos, permite diferenciar usuários e aplicações classificando-os por grupo, efetuando um gerenciamento correspondente às políticas (regras). Outros benefícios da gerência por políticas estão em promover uma ampla facilidade rumo a escalabilidade e melhorar a adaptabilidade da rede nas mais variadas condições. Favorece também, requisição multimídia com diferentes características de QoS e na complexidade das aplicações de tempo real.

Atualmente a comunidade enfrenta diversos desafios na utilização do PBNM, onde entre estas estão à padronização da arquitetura de gerenciamento, protocolos de difusão de políticas, armazenamento das políticas e os métodos utilizados para a construção da regras das políticas. Nesta dissertação, o intuito é a distribuição de políticas utilizando XML (*Extensible Markup Language*)/ SOAP (*Simple Object Acess Protocol*), onde as funcionalidades destas tecnologias possibilitam “distribuição” ampla, flexível

1.4 Proposta

Este dissertação tem como objetivo propor a extensão do uso de gerenciamento baseado em políticas de rede. Para tanto, foi desenvolvido uma abordagem para

distribuição de políticas com ênfase em qualidade de serviço, onde entre a comunicação do ponto de decisão (PDP) e o ponto de aplicação (PEP), a proposta é capaz de enviar parâmetros da rede, reportar condições distintas e distribuir as políticas definidas para o gerenciamento da rede. Neste contexto é definida a mensagem XML/SOAP como meio distribuir de políticas entre os participantes de um sistema distribuído de gerenciamento.

A utilização do XML/SOAP deseja facilitar a distribuição de políticas que opera na rede, implementando uma estrutura que atua com transparência de propriedade, flexibilizando recursos e otimizando dispositivos. Na construção desta extensão PBNM, a definição do XML/SOAP que atuará como estrutura de comunicação, atualização e eliminação de políticas, também possibilitará verificação das métricas para determinados dispositivos e quais são as políticas definidas para o mesmo.

Algumas extensões para o framework PBNM têm sido propostas, as soluções de maior destaque trabalham o uso de políticas com COPS [Flegkas, 2003], SNMP [RFC 1157] e a implementação da comunicação entre redes heterogêneas, por meio de um ponto de decisão de política (*bandwidth broker)* [By, 2003].

Através dos padrões definidos pela IETF este trabalho permite a distribuição de políticas para dispositivos que compartilham o uso de um conjunto lógico de políticas em XML/SOAP para o gerenciamento de recursos e aplicações. A concepção do protocolo visa identificar sua aplicabilidade e flexibilidade em diversas condições, ampliando sua utilização e verificando sua funcionalidade.

1.5 Objetivos e Contribuições

Os principais objetivos desta dissertação são:

- Criar um mecanismo para distribuição de políticas utilizando XML/SOAP que permita maior flexibilidade na difusão de política em diferentes e distribuídos domínios administrativos;
- Desenvolver a estrutura de mensagens em padrão XML para requisição e resposta entre o ponto de aplicação da política (PEP) e ponte de decisão de políticas (PDP);

- Evidenciar a utilização do protocolo SOAP no encapsulamento das políticas;
- Aplicar a abordagem através da definição do gerenciamento baseado em políticas com ênfase na qualidade de serviço;
- Uma gerência mais flexível e distribuída, facilitando adaptações e distribuição de políticas pela rede;
- Avaliar a implementação com relação as suas características e funcionalidade;

Algumas contribuições desta dissertação são:

- No desenvolvimento de uma implementação para inserir padrões XML/SOAP em recurso e aplicações no gerenciamento baseado em políticas;
- Flexibilizar a aplicabilidade da padronização do Framework de políticas;
- Definição e contextualização de uma abordagem para distribuição de políticas baseado em recomendações IETF;
- Contribuir para desenvolvimento e aperfeiçoamento na distribuição de políticas de gerenciamento para redes;

1.6 Organização da Dissertação

O desenvolvimento desta dissertação será apresentado em seis capítulos, dispostos a seguir:

- **Capítulo 1 – Introdução**: Apresentação e contexto geral desta dissertação;
- **Capítulo 2 – Gerenciamento de Redes Baseado em Políticas – PBNM**: Trata dos principais conceitos da arquitetura, componentes e a aplicação de políticas no gerenciamento de redes.
- **Capítulo 3 – Repositório de Políticas**: Descreve os aspectos do armazenamento de políticas e das padronizações relacionadas ao repositório.
- **Capítulo 4 – Protocolo de Distribuição de Políticas:** Relata os protocolos de distribuição de políticas no gerenciamento de rede e a importância dos mesmos neste mecanismo.

- **Capítulo 5 – Trabalhos Correlatos e Contribuições:**
  Relaciona os principais trabalhos que permitiram a contextualização e concepção da abordagem desenvolvida. Também as principias motivações e contribuições.
- **Capítulo 6 – Proposta para Distribuição de Políticas utilizando SOAP/XML:** Propõe um modelo de requisição e respostas entre aplicação e a gerencia da rede, possibilitando flexibilidade utilizando XML e portabilidade com o SOAP no gerenciamento baseado em políticas.
- **Capítulo 6 – Conclusão e Trabalhos Futuros:** Apresenta as conclusões da dissertação e continuidade possíveis do mesmo.

2. Gerenciamento de Redes Baseado em Políticas – PBNM

2.1 Introdução

A gerência de rede tradicional trabalha com um conjunto de informações de rede que cresce constantemente, tanto em diversidade quanto em volume. Os administradores tratam esse volume de informações recebidas por redes de computadores heterogêneas, primando por o melhor desempenho de recursos e aplicações. Isso acaba tornando a gerência uma tarefa bastante complexa. Além disso, o tráfego das informações é diretamente proporcional ao tamanho da rede, como a Internet incentiva a interconectividade dos usuários, as redes tendem a se tornar cada vez mais distribuída, e logo, com mais informações a serem gerenciadas.

Uma solução que visa atenuar os problemas da gerência tradicional de maneira relativamente simples baseada em regras é o gerenciamento baseado em políticas. A emergente tecnologia de gerenciamento de redes baseado em políticas oferece um caminho para superar muitas limitações, auxiliando as plataformas de gerenciamento tradicionais. Seu uso tem como objetivo diminuir a complexidade, quantidades de problemas do gerenciamento, permitir automatização de tarefas na gerência e adaptabilidade às novas condições do ambiente.

Este capítulo tem como objetivo geral introduzir conceitos de políticas, da arquitetura baseada em política e seus componentes necessários para o entendimento da estrutura, distribuição e organização do sistema PBNM. O capítulo é dividido em duas seções principais 2.2 e 2.3. Na seção 2.2 serão abordados conceitos de políticas e sua utilização - 2.2.1 e também sucintamente a origem de políticas no gerenciamento - 2.2.2. Em seguida a estrutura e recomendações para criação de políticas – 2.2.3. Logo após as políticas de regras 2.2.4 e finalizando com um exemplo de política envolvendo qualidade de serviço 2.2.5. A seção 2.3. é focada na padronização e arquitetura PBNM. Em 2.3.1 apresenta as padronizações de políticas IETF. A arquitetura de gerenciamento baseada em políticas é discutida em 2.3.2. Em 2.3.3 os componentes do framework de políticas da IETF, bem como sua escalabilidade, confiabilidade e suas limitações. Os requisitos do

sistema PBNM em 2.4, seus benefícios e principais produtos PBNM na seção 2.5. As conclusões do capítulo apresentado no 2.6.

2.2 Políticas em Sistemas de Gerenciamento de Redes

2.2.1 Políticas

Na sociedade e organizações, políticas são freqüentemente regras, contratos, acordos e procedimentos. Políticas basicamente são regras que descrevem comportamentos que devem acontecer quando condições específicas existirem na rede. A regra é formada por uma ou mais combinações de regras, isso sugere que uma política pode ser formada por outra política.

O gerenciamento baseado em políticas permite formalizar e declarar o comportamento do sistema. Também objetiva orientar, limitar ou aplicar o comportamento definido em entidades autônomas. Em [Sloman, 2002] políticas são definidas como “regras que governam a escolhas do comportamento de um sistema”, enquanto [Flegkas, 2001] descreve-as como o ”caminho para orientar o comportamento de uma rede ou sistema distribuído através diretrizes de alto nível informativo”. As técnicas de políticas diferem de regras de negócios por serem focadas no gerenciamento do sistema e expressas em linguagens entendida e aplicadas pelo sistema. A gerência de rede baseada em Políticas (PBNM) aplica as políticas ajustadas ao princípio do gerenciamento de redes de computadores.

Na literatura, as políticas são divididas freqüentemente em duas categorias principais [Stevens, 1999a]:

- Políticas da Autorização: que permitem ou proíbem uma entidade autônoma de realizar uma ação;
- Políticas de Obrigação: definem os deveres, papéis e responsabilidades de uma entidade autônoma.

Na grande maioria dos sistemas de gerência baseados em políticas, as políticas de obrigação são usadas para estimular o comportamento dos agentes autônomos.

Entretanto, ambas as formas de políticas são úteis aos sistemas de gerenciamento e a política de obrigação é a primeira usada no gerenciamento de nível de serviço. A representação do fluxo da política entre o gerente e os objetos gerenciados é apresentado na Figura 1.

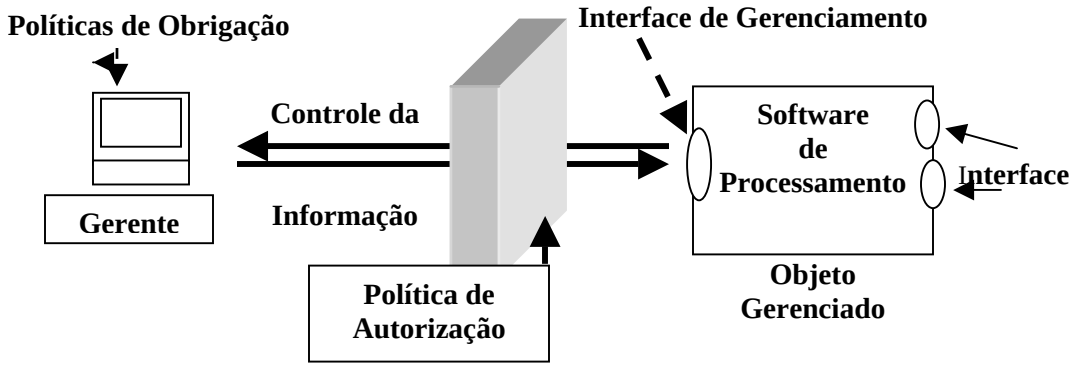

Figura 1: Sistema Baseado em Políticas por Sloman. Fonte: [Sloman, 1994].

Políticas podem também ser abstratas ou concretas [Flegkas, 2001]:

- Política Abstrata: específica um objetivo ou restrição que precisa ser executada sem especificar como ela é executada;
- Política Concreta: específica um processo que precisa explicitamente ser seguido. Políticas concretas são tipicamente fáceis para executar, por que não possuem detalhes duvidosos ou não especificados;

Além disso, a maneira como o resultado da política concreta é medido permite ser especificado também, de modo que a entidade autônoma saiba que encontrou com sua obrigação. Por outro lado, políticas abstratas têm detalhes ausentes, e assim a entidade autônoma requer bastante conhecimento e inteligência para trabalhar com ela.

A hierarquia em política também é considerada, pois o objetivo de uma abstração de alto nível é decompor em pequenas partes mais específicas até que a política se torne concreta e implementável. Esse processo é referido como políticas de refinamento, tradução ou transformação em literatura. A hierarquia de políticas é fundamental para o gerenciamento baseando em políticas, pois permite que várias políticas simples possam ser originadas de políticas complexas.

Desde então políticas descrevem o comportamento do sistema, onde se deve ter um equilíbrio entre os níveis de política abstração e políticas concretas. Aumentando o nível de abstração de um modelo, aumenta o poder da especificação e permite-o aplicar para mais entidades gerenciadas, mais reduz detalhes específicos sobre como o objetivo vai ser executado ou medido. A escolha de política de abstração é crucial, ver Tabela 1 característica de políticas de abstração versões concretas.

	Vantagens	**Desvantagens**
Políticas de Abstração	- Pode ser aplicada por mais entidades ou para o sistema inteiro; - Fácil de ler e compreender; - Fácil de escrever; - Pode ser aplicada para mais geral propriedade do sistema;	- Não pode ser específica sobre o objetivo que deve ser executado;
Políticas Concretas	- Pode ser específica sobre o que precisa fazer; - Pode aplicar o subconjunto de entidades dentro do sistema;	- Dificuldade de ler; - Complexa para escrever; - Não pode ser aplicada facilmente por entidades heterogenias;

Tabela 1: Comparação entre política abstratas e concretas Origem [Stevens, 1999a].

2.2.2 Histórico de Políticas

Formas de políticas são usadas para locar recursos computacionais para vários e distintos clientes [Moffett, 1990]. O Departamento de Defesa Americano – DoD, também usou políticas para restringir acesso a dados sensíveis e sistemas como forma de segurança [Moffett, 1991]. Eles dividiram a base de dados por níveis de autorização que possuíam atributos de identificação seguras, permitindo o processo de gerenciamento da segurança ser simplificado.

No ambiente acadêmico, o conceito de políticas e domínios tem sido popularizado por *Morris Sloman* e sua equipe. Ele é um dos principais membros do grupo de gerenciamento em sistemas distribuídos do *Imperial College* em Londres e também participa de inúmeros projetos relacionados a políticas desde 1980. Baseado neste trabalho o DoD e o padrão de gerência OSI, pesquisas foram continuadas em Conic

[Robinson, 1988], Domino [Moffett, 1994], projetos construídos para capacitar controle de acesso a listas (ACL) seguras em gerenciamento de sistemas.

Inicialmente, ênfase foi no gerenciamento baseado em domínios, o qual a entidade gerenciada poderia ser dividida em áreas relacionadas para simplificar a especificações de políticas de gerenciamento para o controle de acesso. Eventualmente, este trabalho conduziu em mais áreas específicas assim como políticas de obrigação, QoS e gerenciamento de configuração acrescentando a mais tradicional política de autorização. A IETF também tem usado conceitos de Políticas para roteamente em Inter-Domínios [Flegkas, 2002]. Estes conceitos passaram então a ser usados em áreas como Serviços Diferenciados [RFC 2475], Serviços Integrados [RFC 2210] e Segurança IP (Ipsec) [RFC 3586].

O protocolo de reserva de recurso (RSVP) foi desenvolvido pelo Instituto de Ciências da Informação da Universidade de Califórnia do Sul, e a arquitetura escolhida foi muito influenciada pelo desenvolvimento do framework de políticas há alguns anos atrás. As pesquisas da IBM em RSVP para roteamento através de rótulos conduziram à especificação do protocolo OOPS, o qual foi estendido e padronizado como COPS [RFC 2748] pelo grupo de trabalho de protocolos de alocação de recursos (RAP) criado em 1997.

Política tem também um forte conceito de DEN, o qual oferece um melhorar gerenciamento de redes através do uso de diretórios de armazenamento da informação. Uma parceria entre a Cisco e Microsoft conduziram a formalização do DEN em um grupo de trabalho Ad-Hoc, o qual criaram uma padronização original de políticas baseadas em redes. Essa padronização foi mais tarde transferida para DMTF e IETF, os quais têm colaborado juntos produzindo padronizações similares para sistemas baseado em políticas. O DMTF criou o CIM [RFC 3006], um modelo de informação para guardar informações de políticas e performance do gerenciamento da rede. A IETF ficou responsável pelos protocolos de difusão de política e acesso ao repositório de políticas. Entretanto, a IETF não fornece qualquer orientação de uma arquitetura apropriada para sistema de gerenciamento de rede baseado em políticas - SPBNM.

Em 1998, o grupo de trabalho de framework de políticas da IETF ficou responsável por criar um framework que pudesse ser usado por vários outros grupos de

trabalho. Eles estendem o trabalho da DMTF-CIM e aproveitam muitos conceitos da arquitetura de políticas.

2.2.3 Estrutura de Políticas

Uma política pode ser vista como administrar um conjunto de pontos e comportamento da rede. O grupo de trabalho de políticas da IETF definiu a política como ”agregação de regras, onde cada regra inclui um grupo de condições que corresponde a um grupo de ações” [Moore, 2001].

Modelo de política:

“Se (grupo de condições), então (grupo de ações) pode ser feito”.

A “Expressão” é um grupo de instruções que especificam a garantia ou negação do serviço e outros parâmetros usados para prover uma diferenciação na qualidade em um ou mais serviços. Se a condição ou condições são verdadeiras, então a ação ou grupo de ações correspondente é executado.

A “Condição” e “Ação” têm dois principais componentes: operação e operador. Os componentes de operação que podem ser constantes ou variáveis. Eles são examinados para saber se as condições ou o conjunto de condições são verdades ou falsas. Então, é importante estabelecer o grau de valor que o operador pode receber claramente.

Políticas podem ser simples (elas são expressas em simples instruções como “condição-ação”) ou composta (para executar funções mais elaboradas). Dentro de uma composição, a execução ordena ações que deve ser específicas. No caso de não ter uma ordem estabelecida, então as ações são executadas seqüencialmente.

As políticas podem ser implementadas em diferentes linguagens de programação. A estrutura da política fica independente da linguagem usada, mas é conveniente seguir as recomendações em qualquer linguagem usada [Sloman, 1994]:

- Precisão: política deve considerar um nível detalhado de informação para ser entendida por itens de rede, não devendo existir qualquer ambigüidade sobre suas interpretações.

- Consistência: se muitas políticas são definidas para um item da rede, as políticas devem estar cientes entre eles. Não deve ter qualquer ambigüidade ou contradição em suas interpretações
- Compatibilidade: o grupo de política para ser implementada em um dispositivo deve ser designado levando em conta a capacidade de suporte dos nós na rede.

2.2.4 Políticas de Regras

O conceito de regras é essencial para o framework de políticas. Algumas pesquisas estão relacionadas a políticas de regras, por exemplo, [Lupu, 1999]. O estabelecimento de uma regra é definido por uma métrica, por exemplo, em caso de roteamento temos métricas: atraso e perda de pacotes.

Sistemas de gerência de alta escala usam dezenas ou milhares de políticas, no qual somente um pequeno subconjunto executa o desejado comportamento na rede em um momento específico. Nesse sentido, o agrupamento de políticas em regra é básico, assim como no processo de seleção de política por um PDP para aplicar no processo de resolução de conflitos nos quais várias políticas podem ser aplicadas para o mesmo evento.

Outra vantagem pode ser vista no estado de implementação onde os objetos não são gerenciados individualmente, mais eles são agrupados em regras dependendo suas funções. Existem muitas regras conforme a necessidade de um sistema, por exemplo, políticas de motivação, configuração, falhas, funcionalidade, segurança, serviços, controle admissão, etc**.**

As regras formam uma estrutura hierárquica no qual cada política é um membro de no mínimo uma regra. Uma política pode pertencer a várias regras. Políticas são adicionadas inicialmente a uma regra na edição do processo. A Relação de hierarquia entre regras supõe que, mudança na política de “regra pai” afeta os objetos que estão em um domínio e respectivos subdomínios, mais mudança na política de “regra filho” somente afeta o membro do domínio.

Devido ao fato de uma regra definir o particular funcionamento da entidade, os nós da rede podem ser associados a funções executando o comportamento determinado

pela política. Quando a mudança no comportamento da rede o administrador das políticas executa atualizações das políticas pertencentes àquela função. Então, o framework de política verifica quais configurações necessitam ser atualizadas para todos os recursos pertencentes à função.

O uso inteligente de agentes para o monitoramento rede pode também permitir uma rápida atualização na base de conhecimento para a criação dinâmica de regras e prognóstico de comportamentos da rede (tráfego, falhas, etc.) [Barba, 2002].

2.2.5 Exemplo de Políticas

O processo de refinamento ou transição pode ser considerado com a transformação de um modelo de política abstrata de alto nível em um equivalente modelo de política concreta de baixo nível:

Tadeu tem acesso para a alta qualidade para videoconferência.

A Qualidade para videoconferência deve ter maior prioridade que o tráfego de melhor esforço.

Pode ser representado pelas seguintes regras em uma linguagem hipotética, no qual a expressão contém ambos dados e comportamento.

Tadeu. Serviço = [AltaQualVoD]

AltQualVoD.prioridade > MelhorEsforço.prioridade

Quando as regra é traduzida ou refinada, a rede pode usar serviço diferenciado para fornecer QoS. Usando alguma informação do ambiente, assim como:

Tadeu.IPAdress = 150.160.1.3

AltQualVoD.Port = 1024

AltQualVoD. DiffServCodePoint = GarantirTransferencia11

GarantirTransferencia11.DSCP = 001111b

GarantirTransferencia11.Sincronização = Fila de Prioridade

RoteadorBorda = [150.160.1.1, 150.160.5.1]

RoteatorCentro = [150.160.2.1, 150.160.3.1, 150.160.4.1]

O resultado de políticas concretas pode ser refinado para:

Para todos [RoteadorBorda]

Se PacoteUDP.DestIPAddress = 150.160.0.1 e PacoteUDP.Port = 1024
Então PacoteUDP.DSCP = GarantirTransferencia11.DSCP

Para todos [RoteatorCentro]

Se PacoteIP.DSCP = GarantirTransferencia11.DSCP
Então Fila1. EntranaFila (Pacotes IP)

Tem também muitos outros exemplos de políticas em recursos computáveis e não-computáveis. Por exemplo, o funcionamento de *switching* e roteadores de uma rede, considerando-os com entidades controladas por políticas: decisões são feitas baseadas no conhecimento que é avaliado (assim como origem e destino de endereço IP) e no grupo de regras sobre como cada pacote determinado (como rotas, LSP, filas, etc.) A Figura 2 trás um exemplo de políticas para videoconferência.

Política: Videoconferência
Se (Horario >= 8am) **e** (Horario >= 11am) **e**
(IPDestino = 150.162.6.125) – Servidor de Videoconferência
então
Velocidade de Transmissão = 200 Kbps
Perda de Pacotes = 10%
Delay = 100 ms

Figura 2: Exemplo de Política de Videoconferência.

2.3 Arquitetura PBNM

2.3.1 Padronizações de Políticas IETF

Os trabalhos iniciais da IETF foram feitos pelo grupo de trabalho de protocolo de alocação de recursos - RAP. Eles definiram o COPS [RFC 2748] como um protocolo que permite a troca de políticas de informações entre Ponto de Decisão de Políticas (PDP) e o Ponto de Aplicação de Políticas (PEP). Eles definem o PDP como ponto onde as políticas são criadas, enquanto o PEP é o ponto onde as políticas de decisão são realmente aplicadas [RFC 2753]. O COPS é um protocolo adaptável, pois permite vários tipos diferentes de clientes e determina a estrutura e armazenamento da informação da política a ser trocada.

São dois os modelos de políticas definidos pela IETF: O modo *Outsourcing* usando o protocolo COPS-RSVP [RFC 2749] e o modo *Provisioning* usando o protocolo COPS-PR [RFC 3159].

Para *outsourcing*, o PEP solicita decisões criadas pelo PDP e aguarda a decisão ser enviada. Esta é ideal para situações RSVP, onde o controle de admissão precisa da decisão para ser transferida (*outsourced*) e verificar se o usuário tem autorização para fazer a reservas.

Para o modo *Provisioning*, o PDP cria decisões e então informa o PEP da configuração que precisa ter. O PEP pode enviar notificação de solicitação para o PDP no caso mudança de estado. O modo *provisioning* usa conceitos de PIB para criar interfaces de PEP que podem ser modificadas. Este é muito semelhante à MIB que é acessado através do SNMP. A estrutura do PIBs são descritas por SPPI [RFC 1441], a qual é uma subdivisão da SMI [RFC 1441] usado em SNMP.

A vantagem chave do COPS que é mais escalável e confiável quando comparado com SNMP. Ele é responsável por o PEP contatar inicialmente o PDP e informar de qualquer alteração na configuração. Assim, o PDP somente precisa reagir ao evento específico da rede, tomar decisões e então contatar o PEP como solicitado. O protocolo COPS usa TCP preferencialmente do que o UDP, assegurando que a comunicação entre o

PDP e o PEP seja confiável. Além disso, as definições de variáveis PIB são de alto nível de abstração quando comparados com variáveis MIB [RFC 2748]. Mais detalhes da seção 3.3.

Um PDP pode ter um número de clientes PEPs e pode ser conectado a mais de um PDP para diferentes tipos de política. O estado de um dispositivo de rede é entendido por ambos PDP e o PEP, o PEP tem a responsabilidade de manter a sincronização de estados entre eles [Flegkas, 2003].

O PEP pode até executar algumas formas de tradução para dispositivos, não-políticos (não apto a política), enviando informações. A configuração de dispositivos não-políticos pode ser através de outros meios, assim como *Simple Network Management Protocol* (SNMP) ou Interface de Linha de Comandos (CLI). Estes comandos são aplicados para atualizar os objetivos das políticas. O PEP também é responsável por monitorar os dispositivos não-políticos de forma que possam interoperar com o sistema gerenciamento PBNM.

O Grupo de trabalho do Framework de Políticas estendeu esse trabalho para estabelecer uma arquitetura básica para sistemas de políticas. Eles estenderam a definição de um PDP como ”uma entidade lógica que faz políticas de decisões para si mesma ou para outros elementos da rede que requisitam tais decisões” [RFC 3198].

A arquitetura de funcionamento foi descrita conforme Internet *Draft* [Stevens, 1999a], como parte do grupo de trabalho framework de políticas. Este arquitetura básica é mostrada na Figura 3**,** onde há o relacionamento entre elas para diferentes políticas e dispositivos. Esta geralmente é a arquitetura para sistemas de PBNM mais aceita na maioria das literaturas [Verma, 2002].

Esta arquitetura diferencia as funções do repositório de políticas e da ferramenta de gerenciamento de políticas. O repositório de políticas é uma entidade a qual fornece “freqüente armazenamento e retorno as políticas de regras”. A ferramenta de gerenciamento de Políticas existe para “capacitar uma entidade” a definir e atualizar políticas de regras e opcionalmente monitorar sua estratégia. [Flegkas, 2002]. Estas ferramentas permitem interação através de um painel de controle, interface gráfica com o usuário (GUI) ou interface para programação de aplicativo de forma que políticas podem ser definidas e distribuídas para seus consumidores ou PDP para execução.

O grupo de trabalho de políticas reaproveita o trabalho de padronizada de políticas do (DEN) grupo de trabalho Ad-Hoc e DMTF, como base para a *Policy Core Information Model* (PCIM) [Snir et al, 2003] e extensões do modelo chamado PCIMe [RFC 3460]. Estes três padrões especificam o uso de "*Se <condição>Então <ação> regras*" as quais são armazenadas no repositório. Para estes também a assunto em forma de *Drafts* no qual a definições de LDAP para acesso ao repositório de políticas [Strassner, 2002] e modelo de informação de QoS.

Outro ponto interessante, o Framework de políticas IETF sugere o uso de Acordo de nível de serviço (SLA), o qual é uma coleção de objetivos de nível de serviço (SLOs) que podem ser mapeados para políticas de baixo nível [Stevens, 1999b]. Elas assumiram que políticas são derivadas de objetivos de negócios de uma organização que operam na rede. Os usuários administrativos são responsáveis por desenvolver políticas abstratas que implementam o SLOs (ou outras lógicas de negócios), então mapeiam estas políticas para políticas de regras descrevendo o comportamento da entidade na rede. Este mapeamento da primeira camada para a outra não é formalmente definido pela IETF.

Eventualmente, estas regras são traduzidas em comandos específicos para dispositivos da rede. Assim como, na tradução de políticas no qual ocorre uma abstração descrita em alto nível e os dispositivos são configurados em baixo nível. Essa tradução ocorre em um número de estágios, através da ferramenta de gerenciamento de políticas, PDP e o PEP. A tradução é necessária, pois parte dos dispositivos de rede são incapazes de entender o nível mais abstrato do conceito de política. Entretanto, a estrutura de políticas em um repositório de políticas distinto é provavelmente diferente da estrutura de políticas usadas pelo protocolo COPS, exigindo passo intermediário na tradução. Por exemplo, COPS-PR usa *Policy Information Base* (PIB), o qual tem modelo de informação inteiramente diferente.

A arquitetura IETF também identifica a necessidade com relação à detecção global de conflitos e detecção local de conflitos [Stevens, 1999a]. Desde então políticas são baseadas em formalismos, especificações declarativas de "Se (condição) Então (regras)", isto tornam possíveis que as regras sejam contraditórias.

Detecção global de conflitos é executada pela ferramenta de gerenciamento de políticas e assegura que essas políticas são estabelecidas dentro do repositório de políticas

corretamente. No entanto, uma vez a política traduzida para o dispositivo através de comandos específicos, a propriedade para localizar conflitos desde a solicitação de políticas impossíveis a conflitos existentes em grupos de políticas, são detectados pelos PDPs e PEPs usando detecção local de conflitos.

2.3.2 Arquitetura de Gerenciamento Baseado em Políticas da IETF

O gerenciamento de rede baseado em políticas tem motivado amplas pesquisas na última década dentro de grupo um de trabalho IETF. O grupo de trabalho de framework de política da IETF criou o protocolo chamado COPS, o qual originalmente é compatível com a integração de serviço. Hoje em dia é usado como a proposta geral de protocolo de políticas e tem diferentes extensões assim como *Provisioning Extension* (COPS-PR) usado para suportar configurações gerenciamento.

Os grupos da IETF e DMTF trabalham juntos em criar diferentes padronizações relacionadas a políticas: A IETF é encarregada da arquitetura de definição e a DMTF define a padronização do modelo de informação.

A Microsoft e Cisco *Systems* criaram uma infra-estrutura comum de diretório capaz de unificar o gerenciamento dos produtos, chamado DEN em 1998. Essa iniciativa considera o modelo de informação e o esquema de diretório para integrar a rede com serviços de diretórios.

Assim em lugar de gerenciarem dispositivos individualmente, é possível definir regras de gerência para controlar a rede e seus recursos de forma centralizada. A dois grupos de trabalho na IETF que estudam gerenciamento baseado em políticas são:

I. Grupo de trabalho do framework de políticas, o qual escreve vários Drafts e RFCs para representar, gerenciar, dividir e reusar todas as políticas a caminha da escalabilidade, interoperabilidade e independência de hardward e software usada pela plataforma;
II. Grupo de trabalho de protocolo de alocação de recurso (RAP), de quem inicialmente foi o propósito de estabelecer a escalabilidade do modelo de controle de política para RSVP. O grupo RAP definiu um framework de políticas para o controle de admissão, consistindo de interface de

gerenciamento de políticas, repositório para armazenamento, pontos decisão (PDP) para distribuição de políticas e pontos de aplicações (PEP) para configurar a política decisão em nós da rede;

2.3.3 Componentes do Framework de Políticas

Os métodos propostos pelo grupo de trabalho de alocação de recursos (RAP) da IETF para gerência redes baseada em políticas foram os primeiros desenvolvidos em uma perspectiva de integrar serviços, onde o ideal era usar um framework de políticas para gerência e controle de admissão. Essa proposta é independente de qualquer modelo de serviço, em pouco tempo foi possível acreditar em seu uso em ambientes com serviço diferenciados e com outras tecnologias.

A Figura 3 demonstra os principais elementos do framework e os protocolos mais utilizados para notificação de monitoramento e configuração de dispositivos.

- Ferramenta de Gerenciamento de Políticas: Fornece a interface para fazer a administração específica da rede e armazenar as políticas em um repositório. Também atua conforme um monitor do estado, gerenciando redes por meio das políticas.

- Repositório de Políticas: É um armazém que usado pelo ponto de decisão para recuperar políticas. As políticas são armazenadas em um alto nível, independentes dos dispositivos da rede e conforme o modelo de informação, o modelo indicado pela IETF é o CIM, mas repositórios como banco de base de dados, diretórios ou *web servers* são também amplamente utilizados. O uso de um protocolo de acesso é requerido com a necessidade de se integrar aos outros componentes. A IETF sugere o uso de LDAP [RFC 2251], mas outras soluções possíveis são o HTTP, SNMP, SSH e outros.

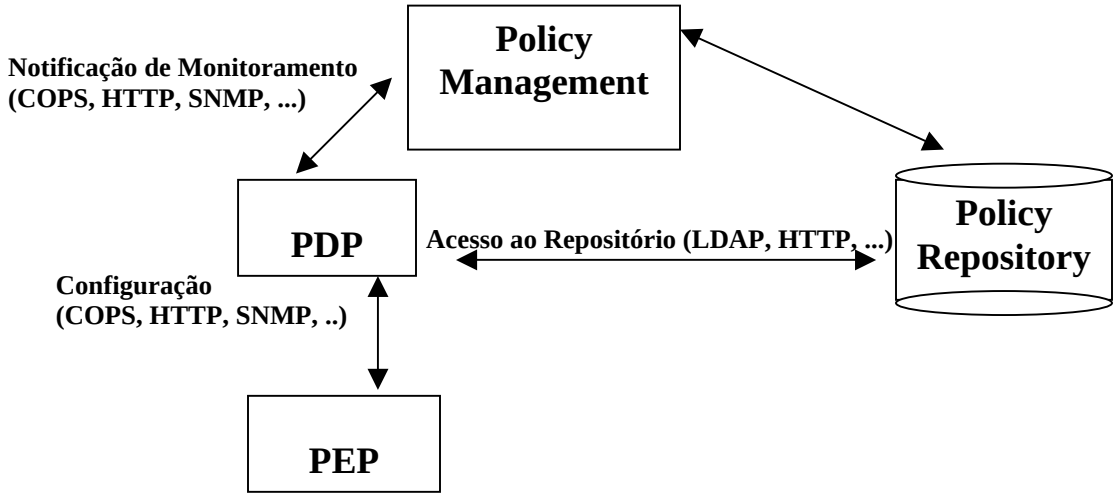

Figura 3: Framework de Políticas da IETF.

- Policy Decision Points (PDP) - Eles são pontos onde todas as decisões que devem ser aplicadas na rede são criadas. Os PDPs processam as políticas e informações sobre o estado da rede para decidir quais políticas são necessárias aplicar. Estas políticas são enviadas conforme configuração dos dados para PEP ou PEPs correspondentes. A arquitetura considera a possibilidade de ter um componente PDP para um ou vários PEPs e estes com todas regras físicas dos dispositivos. O PDP é responsável por localizar o grupo de regras e aplicar a um PEP, recuperando do repositório e transformando-as dentro de um formato sintático que pode ser entendida por dispositivos e distribuído ao PEP. Neste sentido, o PDP atua conforme monitor do estado da rede, verificando se todas as solicitações de condição foram satisfeitas pela política de aplicação. Se o PDP justificar transformações, então deve carregar no repositório de políticas novamente.

- Policy Enforcement Point (PEP) - Eles são itens envolvidos na execução de políticas que o PDP ordena. Alguns exemplos desses PEPs são roteadores, *firewalls*, *proxies* etc. Cada PEP recebe a política na forma de configurações específicas de ações sendo responsável pela estabilidade no dispositivo. O PEP pode também informar ao PDP quando qualquer situação desconhecida for existente.

Quando um PEP envia a mensagem requerendo uma política de decisão, consulta sua base local de configuração para identificar a política de itens que serão avaliadas. Então o PEP passa essa requisição do item avaliado para o PDP local (quando houver

mais de um PDP) e recebe o resultado parcial. Depois, do PEP ter passado todas as políticas de itens e os resultados ao PDP, retorna a política final de decisão para o PEP.

O framework IETF estabelece COPS para transferência de políticas de decisões para o PEPs e para transferir requisições de PEPs para o servidor de políticas. O modelo é aberto para outros mecanismos assim como HTTP, FTP ou SNMP.

2.3.3.1 Escalabilidade da Arquitetura IETF

A arquitetura IETF permite certa escalabilidade através do uso de múltiplos PDPs. Em Strassner et al., [Strassner, 2003] sugere que um PDP deveria ser usado para cada ou domínio de política. Em outras palavras, cada PDP pode gerenciar um particular subgrupo da rede, ou eles podem ser usados para funções individuais da rede (assim com tipos particulares de políticas, por exemplo, para QoS, segurança, etc).

O protocolo COPS tem a capacidade para se comunicar com um ou mais clientes. Um PDP diferente pode ser usado para cada cliente, ou um simples PDP pode suportar um número maior de clientes. Isso permite alguma flexibilidade em termos de onde a política decisão pode contribuir para diferentes domínios. Outra alternativa, a rede pode ser dividida em um número de sub-redes e cada PDP pode gerenciar uma parte da rede [Hamada, 2000].

No entanto, isto aumenta a complexidade do gerenciamento de PEPs, por que eles devem ser configurados com endereço IP dos PDPs de forma que os PEPs possam receber políticas. Se a localização de PDP mudar, então um grande número de dispositivos necessita de atualização com uma nova lista. Contudo, o número de conexões ativas deve crescer imensamente, conforme cada PEP requeira canal de comunicação via conexão TCP/IP a um PDP.

Além disso, se os PDPs são independentes, então a falta de interatividade entre eles seria preocupante, desde o resultado completo não ser com o planejado, cada PDP adotar a aplicação oposta ou conflitar requisitos de um PEP particular.

Dependendo o tipo de política que é armazenado dentro do repositório de políticas, decisões irão freqüentemente precisar ser criadas tendo um completo conhecimento de

todos os dispositivos que fazem parte da rede. Isto encoraja a centralização do PDP, desde a determinação do dispositivo a decisão.

Tendo um grande número de PDPs no interior da rede aumentará também a dificuldade de gerenciar os PEPs, por que eles irão precisar ser informados onde localizar os PDPs para cada tipo de política que o PEP entenda.

Quanto a distribuição das políticas, tem basicamente duas formas [Stevens, 1999a]:

- No modelo *push* ferramenta de gerenciamento baixa a política do repositório de políticas e envia (*upload*) para o PDP. O *upload* para o PDP pode ser executado usando FTP, HTTP, SNMP ou outros protocolos.
- No modelo *pull* a ferramenta de gerenciamento somente informa o PDP sobre a localização URL (*Uniform Resource Locator*) de uma política que precisa ser desenvolvida para componentes PEP. No modelo *pull*, cada PEP é responsável pela comunicação com um PDP e restabelecer esta configuração. O PEP deve também determina mudanças na comunicação e evento para um PDP, quando for feita uma nova política de decisão. Também nesse caso, FTP, HTTP, SNMP e protocolos proprietários podem ser usados. IETF particularmente prefere o uso do LDAP.

O modelo *pull* é considerado ser mais escalável, pois reduz complexidade da operação [Stevens, 1999b]. O PEP recebe mais funções ativas no gerenciamento de atividades, reduzindo a possibilidade de ter uma configuração incorreta. Além disso, é responsabilidade do PDP e PEP assegurar que eles tenham a informação correta quando eles são reiniciados. Desta forma, se o dispositivo é reiniciado por qualquer razão, o operador da rede não tem manualmente com reaplicar as mudanças na configuração.

Quanto à configuração dos dispositivos temos: *outsourcing* e a *provisioning* [Sloman, 2002]:

- Na configuração *outsourcing*, um PEP inicia a comunicação com PDP perguntado pela política decisória em um determinado tempo, conforme evento interno PDP que requer a decisão. Particularmente, *outsourcing* é interessante em um domínio IntServ/RSVP, onde RSVP requer mensagem

para recursos de roteamento. O protocolo de principal importância do *outsourcing* é o COPS o qual compreende este propósito específico.

- Na configuração *provisioning*, o PDP inicia a comunicação indicando ao PEP como ele deve se comportar. *Provisioning*, ao contrário de *outsourcing*, é mais adequado para domínios DiffServ. Visto que o protocolo de reserva não é esperado, as rotas têm de ser configuradas apropriadamente para suportar os requerimentos QoS de agregados DiffServ. Nesse caso, um PDP configura a rota DiffServ capaz para completar o comportamento esperado. Através COPS-PR tem-se começado a definir o suportar de acesso *provisioning*. Vários outros protocolos podem ser usados também. Tipicamente protocolos usados para configuração de dispositivos (e.g telnet, SSH e HTTP) podem ser usados em operações *provisioning*.

Sob a perspectiva de aplicação, a escolha do protocolo de configuração a ser usado para comunicação PDP/PEP depende das capacidades do PEP e suporte ao protocolo. Se o dispositivo que guarda a política PEP suporta COPS, então o PDP deverá suportar COPS também. Logo, se o PEP pode somente ser configurado via HTTP, então o HTTP deverá ser implementado com suporte em consideração. Por fim, as decisões sobre um protocolo a ser usado na comunicação PDP/PEP são baseadas em uma avaliação de protocolos suportados nos dispositivo de aplicação.

A IETF também sugere que o módulo *local Policy Decision Point* (LPDP) pode ser incluindo em um PEP para reduzir o número de decisões que precisem ser *push* (*outsourced)* para o PDP. O LPDP pode reproduzir o comportamento do PDP através de *caching* de decisões já criadas.

2.3.3.2 Confiabilidade da Arquitetura da IETF

O protocolo COPS permite ao PDP determinar o PEP para backup, que pode ser encontrado para replicação [Flegkas, 2001]. Da mesma maneira é possível ao PEP locar

outros PDPs quando ocorrer alguma falha. Entretanto, políticas de decisões importantes talvez sejam perdidas se o PEP não for reconfigurado rapidamente.

Não existe recomendação na padronização de como o repositório de política deve ser replicado e de que forma qualquer replicação deveria ocorrer, se necessária.

2.3.3.3 Limitações da Arquitetura da IETF

Existem várias dificuldades com a arquitetura de políticas definida pela IETF. De modo geral, a arquitetura é também genérica e ampla. Algumas limitações são:

- Não fornece qualquer direção para os desenvolvedores de como o sistema de PBNM deve ser construído;
- Não padroniza uma linguagem determinada para políticas, assim cada implementação pode especificar suas condições e ações em caminhos privados. Isto limita a interoperabilidade de sistemas PBNM, pois as empresas são livres para implementar componentes do sistema de políticas de qualquer maneira que achar adequado [Sun, 2000];
- Não indica métodos nos quais conflitos podem ser detectados ou políticas serem válidas, isto é responsabilidade da ferramenta de gerenciamento de políticas;
- A estrutura de política no repositório não é padronizada por uma área de domínio particular, assim como QoS ou segurança. Dessa forma, a dificuldade para sistemas de políticas diferentes terem interoperabilidade usando o mesmo repositório de políticas, a tradução dos processos que são requeridos e traduzir as políticas de uma forma que outros mais possam utilizar;
- Tem muitas funções que precisam ser acarretas a “Ferramenta de gerenciamento de Políticas”. Estes componentes são sobrecarregados à medida que a arquitetura de políticas se desenvolva para suportar operações mais complexas;

Atualmente a arquitetura favorece especialmente políticas de estados que raramente mudam. Isto é ideal em algumas redes (assim como projetos de redes), no qual o sistema de política somente se propõe a assistir a configuração e performance de estados da rede. Nesta rede, não tem a necessidade de mudança na configuração em curto ou médio espaço de tempo e também não tem a necessidade de monitoramento para assegurar que regras são entidades encontradas.

Enquanto a arquitetura de política não estipula a tecnologia para implementação do repositório de políticas, tem aparecido com grande ênfase DEN e o uso do protocolo de diretórios LDAP.

Stone [Stone, 2001] sugestiona que o repositório de políticas pode ser um servidor de diretório ou banco de dados. Também especifica LDAPv3 ou superior como o protocolo mais desejável para interação como o repositório. Isto é reforçado pelo fato que LDAP para o modelo de dados PCIM é o protocolo mais recomendado para acesso e armazenamento ao repositório de políticas.

Entretanto, ambos [Stevens el al., 1999b] e [Stevens el al., 1999a] identificam claramente algumas limitações da implementação LDAP, incluindo a falta de notificação de mudança referente à integridade e transação da integridade. Entretanto, muitos diretórios com acesso LDAP não têm implementado simples mecanismo de notificação de mudanças, criando dificuldade para o PDP e PEP tornem-se informados de novas ou mudança de políticas. Por esta razão, a IETF sugere que um mecanismo diferenciado (não especificado) pode ser usado para fazer parte da ferramenta de gerenciamento de política notificando o PDP das mudanças na política.

A ferramenta de gerenciamento de políticas é um componente de vital funcionalidade. Ela deve fornecer a interface gráfica (GUI) para o operador, permite especificar política de entradas, verificar a sintaxe das políticas, performance de detecção global de conflitos e gerenciamento das alterações no repositório de políticas. Além disso, não há uma definição de qual seleção das regras de gerenciamento devem ser suportadas. Então, cada empresa pode criar uma definição proprietária de regras específicas de gerenciamento e como eles devem ser traduzidas, limitando a interoperabilidade de sistemas PBNM. Desde então não são determinados mecanismos

para coordenação da ferramenta de gerenciamento, pode-se somente presumir que a ferramenta de gerenciamento de política é fortemente centralizada.

2.4 Pesquisas e Requisitos de *Policy-based Network Management*

É importante perceber o fato que pesquisas sobre sistema PBNM têm demonstrado muitos benefícios atualmente. Entretanto, ainda não existe qualquer sistema comercial de PBNM capaz de oferecer interoperabilidade como modelo de gerenciamento por políticas [RFC 3460].

Na realidade a grande maioria dos sistemas comerciais disponíveis possui um controle de interface, que permite um administrador definir políticas as quais são estatisticamente configuradas nos dispositivos. Então, a construção de um complexo e útil sistema PBNM é totalmente desafiante e eminente. O sistema deve possuir escalabilidade, robustez e confiabilidade. Em termos de escalabilidade, sistema PBNM deve ser capaz de suportar um grande número de dispositivos [RFC 3460], e potencialmente milhares de usuários e serviços e principalmente confiável para que políticas sejam aplicadas corretamente para usuários e aplicações.

O fundamental sobre a compreensão e abrangência de soluções PBNM é que complementam a gerência tradicional, acrescentando o conceito de políticas definidas pelo administrador. Apesar disso, a existência de pré-requisitos para seu funcionamento:

- A rede gerenciada deve ser completamente conhecida pelo seu administrador;
- PBNM deve-se moldar às características da arquitetura existente;
- Os pontos de atuação e decisão de políticas (PEPs e PDPs) já devem estar definidos;

Outra característica PBNM:

- Quando implantada a política, se o comportamento não for esperado o PBNM não é responsável por reportar ao gerente sobre a condição;

2.5 Benefícios e Mercado do Gerenciamento de PBNM

Para o propósito do gerenciamento de redes, políticas permitem ao administrador especificar como a rede é configurada e monitorada através uma linguagem descritiva (ou por outros meios equivalentes). Isto é vantajoso por várias de razões [Sloman, 2002]:

- Reduz a complexidade e quantidade de problemas de gerenciamento, por que políticas podem ser reusadas através de várias entidades gerenciadas. A IETF usa funções como método de associação de objetivos com políticas. Funções podem ser combinadas para efetuar diferentes grupos de políticas de particular objetivos. Em comparação, Sloman originalmente usa conceitos de domínios e funções para agrupamentos lógicos e associações. O uso de políticas também pode permite abstração de forma que especificações podem ser simplificadas e entendidas. Abstração também permite aos dispositivos que políticas podem ser escritas sem preocupação com detalhes de implementação de baixo nível [Verma, 2002].

Este assunto melhora a interoperabilidade:

- Permite a automatização de várias tarefas no gerenciamento [Chadha, 2002], de acordo com o grupo de solicitações preparadas nas políticas. Isto ajuda na execução de grandes números de tarefas;
- Permite que o comportamento do sistema de gerenciamento e seus processos mudem requisitos conforme o tempo [Trimintzios, 2002]. A infra-estrutura pode permitir a introdução de novos serviços e adaptar suas redes rapidamente;
- Inserir capacidade no sistema de gerenciamento de adaptar seu comportamento dinamicamente para diferentes condições do ambientes [Hamada, 1999];
- Permite o sistema alcançar objetivos de alto nível, assim como acordo de nível de serviço [Czezowski et al., 2000].

Outros potenciais benefícios incluem:

- A capacidade de gerenciamento e configuração à distância. Além disso, a política do sistema pode ficar *online* durante a mudança da política, melhorando disponibilidade e adaptabilidade da rede;
- Potencial para aumentar a escalabilidade e reduzir os custos operacionais [Hamada, 2000], a medida que a diminuição na complexidade do gerenciamento também influência na diminuição de administradores de rede;
- Melhora a segurança através do uso de políticas de autorização e especificações explícitas de segurança nos componentes dentro da rede através da política de obrigação.

Os sistemas PBNM têm ganhado grande destaque no mercado de gerenciamento de redes. As principais empresas desenvolvedoras de soluções concentram-se em redes baseadas em IP acrescentado funcionalidades na forma de gerência tradicional.

Os produtos de maior destaque são: *PolicyXpert* [Hp, 2001] da Hewlett-Packard, o *EPICenter da Extreme Networks* [Extreme, 2001] e o QPM (*QoS Policy Manager*) [QPM, 2001] da Cisco. Estas soluções são amplamente comercializadas, entretanto não seguem as recomendações da IETF ocasionando produtos sem qualquer integração. Como resultado, usuários e aplicações de diferentes sistemas PBNM enfrentaram sérios problemas com interoperabilidade.

2.6 Conclusão do Capítulo

Neste capítulo foi contextualizado como o gerenciamento de redes baseado em políticas pode contribuir para otimizar o gerenciamento tradicional. Também ressalta a importância na construção de políticas eficientes e suas recomendações, para refletir em ações na rede gerenciada.

Na seção de arquitetura de políticas, foram apresentadas as principais padronizações do grupo de políticas da IETF. Onde é fato que nem todas as recomendações da IETF para políticas são seguidas pelas empresas desenvolvedoras de sistemas de gerenciamento. Além disso, foram detalhados os componentes do framework

de políticas, suas funções e características no gerenciamento de redes. Discorrido também sobre escabilidade, confiabilidade e limitações do framework da IETF. E para finalizar levantam-se os principais requisitos para inserir a arquitetura de políticas no gerenciamento tradicional, principais contribuições do sistema PBNM e produtos de maior comercialização.

Este capítulo possibilitou a compreensão de como é formada a estrutura de políticas de gerenciamento, mostrando de forma objetiva os principais conceitos, componentes e características de uma tecnologia que pode trazer grandes benefícios ao gerenciamento de redes como um todo.

3. Repositório de Políticas

3.1 Introdução

Um dos componentes de fundamental importância na arquitetura de gerenciamento baseado em políticas é o modelo utilizado para armazenamento das políticas no repositório e o protocolo de acesso ao repositório de políticas.

Como descrito no capítulo 2.2.3, as políticas são armazenadas em um alto nível, independentes dos dispositivos da rede e conforme modelo de informação. O mercado de soluções de gerenciamento baseado em políticas, a exemplo de redes de computadores com o modelo OSI, não segue as recomendações sugeridas pela IETF e DMTF para modelos de repositórios e ferramentas de políticas. As aplicações mais vendidas neste segmento utilizam outros tipos de modelos para repositórios entre os mais utilizados estão banco de dados, diretórios e *web services*.

Neste capítulo são apresentados os modelos para repositórios de políticas recomendados pelo grupo de políticas da IETF e freqüentemente empregado nas pesquisas de maior significância da área. O *Commom Information Model* e *Policy Information Base* são de forma mais geral definições e métodos para a estruturação de políticas em um repositório. Onde ambas adotam procedimentos distintos para tratar o fornecimento de políticas e transmissão aos dispositivos da rede.

O CIM é apresentado em 3.1 discorrendo sobre a finalidade de construção, principais participantes e sua forma de organização. Também se relata sobre sua forma de mapeamento dentro do CIM, estratégias e gravações de decisões de mapeamento. Os principais aspectos do Modelo Comum de Informação da DMTF são encontrados na RFC 3006, a qual foi a principal referência utilizada.

Em 3.2 será abortado o PIB, modelo que baseia as políticas usando textos comuns para todos os clientes. Esta estrutura é criada utilizando-se de classes e objetos. As características do PIB, construção, estrutura das políticas, grupos de classes e outros aspectos serão descritas ao longo desta seção. A RFC 3318 descreve os fundamentais aspectos do PIB, principal referência. Por fim este capítulo possibilita visualizar umas

das partes fundamentais do *framework* da IETF, contribuindo conceitualmente para o melhor entendimento e desenvolvimento deste trabalho.

3.2 Common Information Model - CIM

O *Common Information Model* (CIM) é uma aproximação do gerenciamento de redes e sistemas que aplicam estruturas básicas e técnicas conceituais do paradigma de orientação a objetos. Esta aproximação usa um formalismo de modelagem unificada que junto com o repertório básico de construtores orientados a objetos, suportam o desenvolvimento cooperativo de um esquema orientado a objeto através de múltiplas organizações [RFC 3006].

3.2.1 Objetivos

A proposta do CIM é de permitir que as informações usadas para executar tarefas sejam organizadas ou estruturadas para permitir acesso a grupos diferentes. Através de um modelo de informação que requer um grupo legal de tipos de expressões, sintaxe para capturar a representação e por fim uma coleção de expressões necessárias para gerencia comum de aspectos de domínio.

O CIM é um modelo baseado em orientação a objetos sobre UML. Sendo assim o modelo inclui expressões para elementos comuns que devem ser claramente apresentados para gerenciamento das aplicações (por exemplo, classes objetos, propriedades, métodos e associações) [RFC 3006].

O CIM fornece um grupo de classes com propriedades e associações que permite um melhor entendimento conceitual do *framework*, o qual possibilita organizar a informação disponível sobre o ambiente gerenciado. Isto admite que CIM será claramente entendido por qualquer programador requerido para escrever códigos que trabalhem com esquema de objeto, ou por qualquer estrutura destinada a trabalhar com informações dentro do ambiente gerenciado [RFC 3006].

3.2.2 Bases do Gerenciamento CIM

Bases do gerenciamento são o fundamental para construção de plataformas e aplicações de gerenciamento, por exemplo, configuração de dispositivos, gerenciamento de performance e mudanças no gerenciamento.

A estrutura CIM permite agir como meio de gerenciamento de ambientes, podendo ser vista com uma coleção de sistemas inter-relacionada, cada qual composto de um número discreto de elementos.

O modelo comum de informação é organizado nas seguintes camadas [RFC 3006]:

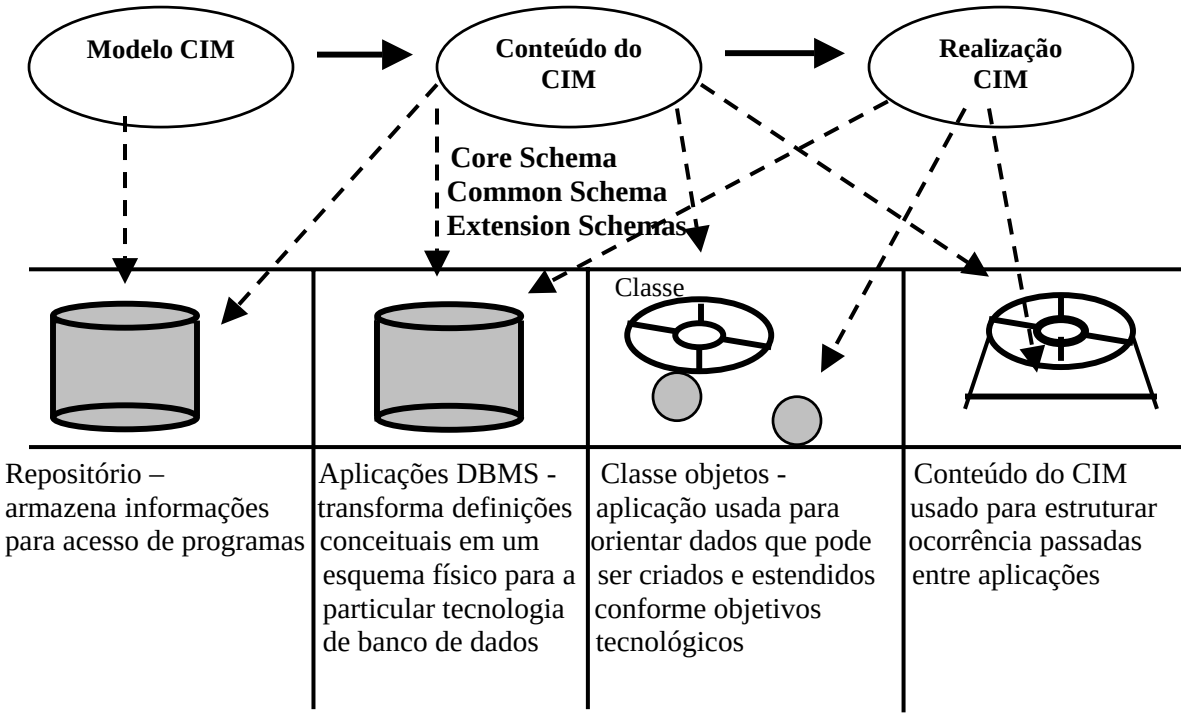

Figura 4: Aplicabilidade do CIM [RFC 3006].

- ***Core model*** – modelo responsável por capturar idéias que são aplicadas para todas as áreas de gerenciamento;
- ***Common Model*** - um modelo de informação que captura idéias que são aplicadas para todas as áreas de gerenciamento, mais independentemente de uma tecnologia particular ou implementada. As áreas comuns são sistemas,

aplicações, redes e dispositivos. O modelo de informação é especificado para prover uma base para o desenvolvimento de gerenciamento de aplicações. Esse esquema fornece um grupo base de classes para extensão dentro da área de esquemas tecnológicos específicos;

- ***Extension schemes*** - representa extensões de tecnologias específicas do *common model*. Estas extensões são definidas para ambientes, assim com sistemas operacionais (por exemplo, UNIX ou Microsoft Windows).

3.2.2.1 *Core Model*

É o modelo mais genérico de todos. O C*ore model* é um pequeno grupo de classes, associações e propriedades que prove um básico vocabulário para analisar e descrever o gerenciamento do sistema. O *core model* representa o ponto de partida para determinar e analisar como estender um esquema (*schema).*
Enquanto possível que classes adicionais sejam acrescidas no *core model* com o tempo, a maioria re-interpreta as próprias classes.

3.2.2.2 *Commom Model*

O *Common model* é um grupo básico de classe que define várias áreas independentes de tecnologias. As áreas são sistemas, aplicações, redes e dispositivos. As classes, propriedades, associações e métodos dentro de *Commom Model* são programados para fornecer uma observação da área detalhada para uso conforme base projetada do programa.

Extensões são adicionadas à abaixo do *common model* em plataformas específicas que acrescem classes concretas e implementações. Conforme o *commom model* é estendido, oferece um amplo alcance de informação.

3.2.2.3 *Extension Schema*

Os *extension schema* são extensões tecnologias específicas para o *common model*. É esperado que o *common model* se desenvolva como resultado de objetos e propriedades definidas no *extension schema*.

3.2.3 Aplicações CIM

CIM é um modelo conceitual que não é limitado a uma implementação particular. Isso permite ser usado para trocar informações de gerenciamento com variedade de perspectivas. Quatro são as principais, conforme Figura 4 [RFC 3006]:

- Conforme o repositório, a construção do modelo de definições é armazenada em um banco de dados. Esses construtores não são ocorrências de objetos ou relacionamentos, mais preferencialmente são definidos para serem usados no estabelecimento de objetos e relacionamentos. O modelo usado por CIM é armazenado em um repositório que se torna a representação do modelo. Isto é completado por mapear a construção do modelo em esquemas físicos de objetivo no repositório, então popularizaram o repositório com classes e propriedades expressas no *Core model, Common model e extension schema*.
- Para uma aplicação de sistemas de gerenciamento de banco de dados - DBMS, o CIM é mapeado em esquema físico de objetivos DBMS, por exemplo, modelo relacional. As informações armazenadas em uma base de dados consistem de atuais ocorrências dos construtores. Aplicações podem trocar informações quando eles têm acesso a um comum DBMS, que mapeiam as ocorrências em um caminho previsível.
- Para aplicação no objeto, o CIM é usado para criar um grupo de aplicações em uma particular linguagem. Aplicações podem trocar informações, enquanto que eles podem obrigar a aplicação no objeto.
- Para troca de parâmetros, o CIM expressa algumas combinações de sintaxe, é uma forma neutra usada para troca de informações de gerenciamento a

caminho de um grupo de padronização de objetos APIs (*Application Program Interface*). A troca pode ser efetuada via um grupo direto das chamadas APIs, ou pode ser acompanhada pela ocorrência de APIs orientadas que podem criar o objeto apropriado na tecnologia local de execução.

3.2.3.1 Adaptação Implementação CIM

A habilidade de trocar a informação entre aplicações da gerência é fundamental ao CIM. O mecanismo atual para trocar as informações de gerência é o formato do objeto de gerenciamento (MOF). Atualmente, nenhuma relação de programas ou os protocolos foi definido como mecanismo da troca. Conseqüentemente, um sistema sobre CIM capaz deve poder importar e exportar construções corretamente dadas no formato MOF.

Como as operações de importação e de exportação são executadas é um detalhe da execução do sistema CIM capaz. Os objetos criados no MOF devem, no mínimo, incluir todas as propriedades chaves e todas as propriedades marcadas como obrigatórias. As propriedades requeridas têm a solicitação presente qualificada e ajustadas.

3.2.3 - Mapeamentos Existentes no Modelo CIM

O modelo CIM possui suas particularidades. São três tipos de mapeamento que pode ocorrer entre os esquemas: *technique*, *recast* e *domain*. Cada um desses mapeamentos pode ser aplicado quando convertido da sintaxe de formato de informação de gerenciamento (MIF) para sintaxe formato de objeto gerenciado (MOF) [RFC 3006].

3.2.3.1 Mapeamento *Technique*

O mapeamento *technique* fornece um mapeamento que usa idéia de construtores do modelo CIM para descrever a origem da modelagem técnica (por exemplo, MIF, GDMO e SMI). Essencialmente, o modelo CIM é um modelo para a origem técnica.

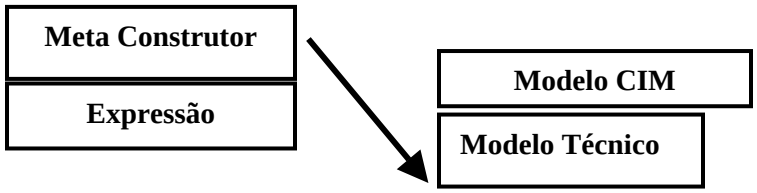

Figura 5: Exemplo mapeamento técnico [RFC 3006].

A DMTF usa o Formato de Informação de Gerenciamento (MIF) como modelo para descrever a distribuição do gerenciamento da informação em um caminho comum. Isto significa descrever um mapeamento *technique* no qual o modelo CIM é usado para descrever a sintaxe MIF.

O mapeamento apresentado aqui introduz importantes tipos que podem aparecer em um arquivo MIF e então cria classes para eles. Assim, componentes, grupos, atributos, tabela são expressas no modelo CIM conforme classes. Além disso, associações são definidas para documentar como estes são combinados.

3.2.3.2 Mapeamento Recast

Um mapeamento *recast* prove um mapeamento meta-construtor de origem dentro dos objetivos meta-construtores. A maioria dos projetos e trabalhos são para desenvolver mapeamento entre a origem do meta-modelo e o modelo CIM. Uma vez isso feito, as expressões de origem são reformadas.

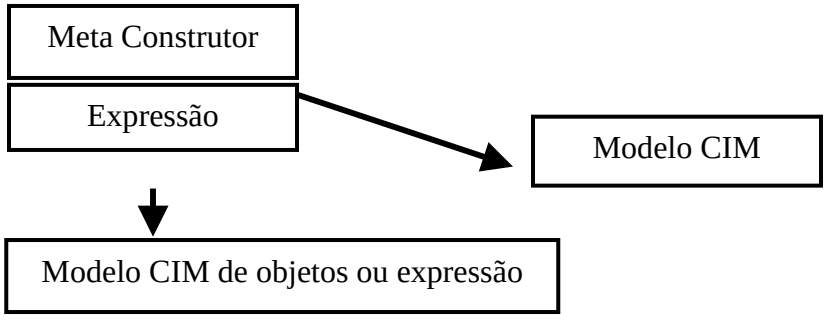

Figura 6: Mapeamento Recast [RFC 3006].

3.2.3.3 Mapeamento *Domain*

Um mapeamento *domain* recebe uma expressão de origem em uma técnica particular, mapeia esse conteúdo para um dos centros ou modelos comuns do CIM, ou extensões de esquemas do CIM. Em alguns casos, o mapeamento é na verdade uma re-expressão do conteúdo em um caminho comum, usado mais significativamente.

3.2.3.4 Mapeamento *Scratch Pads*

Em geral, quando o conteúdo do modelo é mapeado entre diferentes esquemas, informações podem ser perdidas. Para evitar este problema, "*scratch pads*" são expressos no modelo. O *scratch pads* são importantes para a troca do *core, common e extension model*, contendo conhecimento das tecnologias usadas para construir o gerenciamento de aplicações.

3.2.4 Perspectiva do Repositório

Um repositório de armazenamento define a estrutura da informação incluindo a capacidade de extrair a definição em uma forma útil para desenvolvimento de aplicações. Alguns repositórios permitem a definição de múltiplas formas de importação e exportação. As noções de importação e exportação podem ser refinadas assim que eles distinguirem entre os três tipos de mapeamento. Usando a definição de mapeamento da seção 3.3, o repositório pode ser organizado dentro de divisões (*meta, tecnique, recast e domain*). A descrição básica do repositório juntamente com a Figura 7 completa a abaixo.

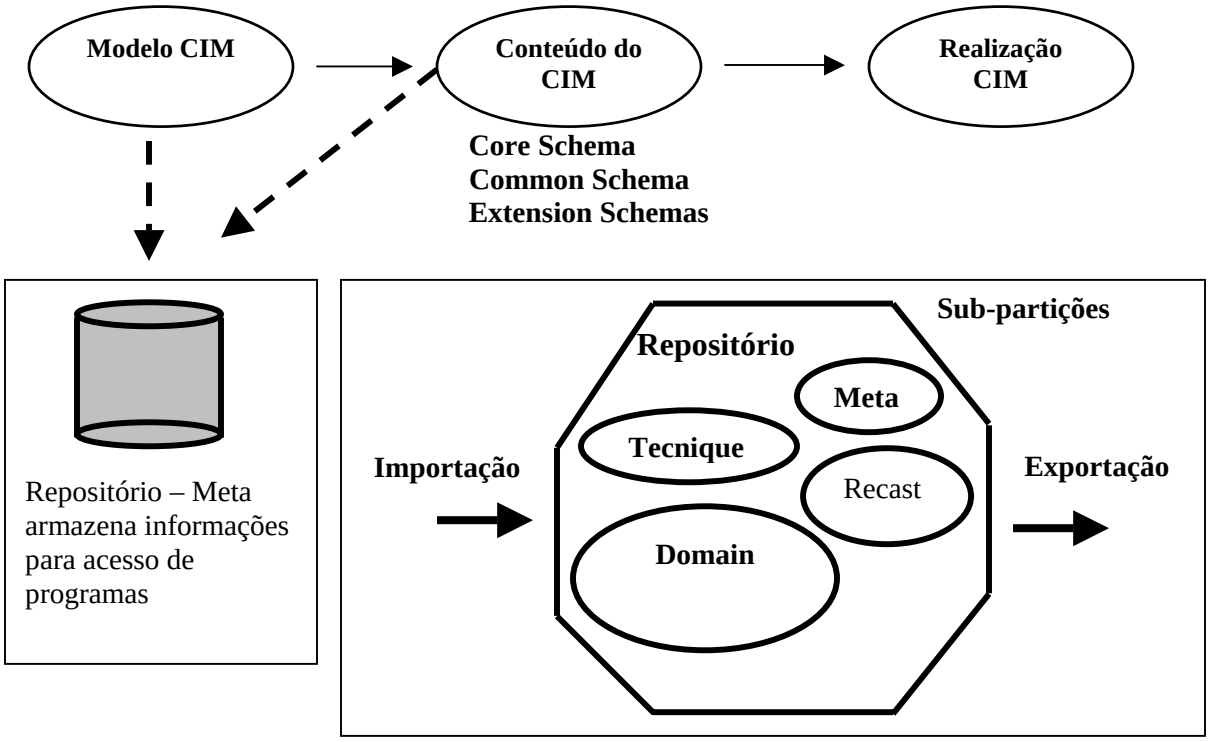

Figura 7: Divisões do repositório [RFC 3006].

As divisões do repositório têm as seguintes características [RFC 3006];

- Cada divisão é separada. A divisão *meta* se refere a definição do modelo CIM. A divisão técnica se refere as definições que são carregadas usando mapeamento *technique*. A divisão reforma refere as definições que são carregadas usando mapeamentos *recast*. A divisão de domínio se refere as definições que estão associadas com o *core* e *common* e *extension schemas*.
- As divisões *technique* e *recast* podem ser organizadas em múltiplas subdivisões para fazer a captura de cada origem de modo único. Por exemplo, supondo ser uma técnica subdividida para cada linguagem encontrada (sendo uma para MIF, GDMO, SMI, e assim por diante). Em divisão de reforma, pretende ser subdividida para cada linguagem.
- O ato de importação do conteúdo de uma origem existente pode resultar em entradas na divisão de *recast* ou *domain*.

3.2.4.1 Estratégias de Mapeamento MIF - DMTF

Assumir a definição do modelo é a linha de partida para o completo esquema CIM. O próximo passo é mapear a origem da informação de gerenciamento para o repositório. O objetivo primário é fazer o requisito de trabalho para importação de um ou mais grupos padronizados.

Quanto à possibilidade de importar cenário [RFC3006]:

1 - Para partição técnica: Criar um mapeamento *technique* para a sintaxe MIF. Este mapeamento pode ser o mesmo para todos os grupos da padronização e pode ser somente necessário atualizar a sintaxe MIF quando alterada;

2 – Para partição reforma: Criar um mapeamento *recast* de um particular grupo da padronização dentro da subdivisão de reforma. Este mapeamento permitir selecionar o completo conteúdo do grupo para ser carregado em uma subdivisão de reforma. O mesmo algoritmo pode ser usado para mapear adicionalmente grupos padronizados dentro da mesma subdivisão;

3 - Para partição domínio: Criar um mapeamento *domain* para o conteúdo de um particular grupo da padronização que envolve o conteúdo do esquema CIM;

4. Para partição domínio: Criar um mapeamento *domain* para o conteúdo de um particular grupo da padronização que não sobrepõem o CIM dentro da uma extensão de sub-esquema;

5. Para partição domínio: propõe uma extensão para o conteúdo do CIM e então executar os passos três e/ou quatro;

Quaisquer combinações de esses cinco cenários podem ser iniciadas por um grupo, o qual é responsabilizado pelo mapeamento de origem existente dentro do repositório CIM. Presumindo a existência de numerosas origens terem sido importadas usando todos os cenários. Ignorando a partição técnica, os possíveis cenários de exportações [RFC 3006]:

1 - Para partição de reforma: Criar o mapeamento *recast* para a subdivisão da partição de reforma do grupo da padronização (que é inverso da importação - 2). O

pedido do método pode usar mapeamento *recast* para traduzir o grupo padronizado da definição GDMO;

2 - Por partição de reforma: Cria o mapeamento *domain* para a subdivisão *recast*, por reconhecido modelo de gerenciamento que não foi à legítima origem para o conteúdo envolvido;

3 – Por partição de domínio: cria o mapeamento *recast* para o conteúdo completo do CIM e a técnica selecionada (para MIF, isto resulta em um grupo não padronizado);

4 – Por partição de domínio: Cria um mapeamento *domain* para o conteúdo do esquema CIM que envolve com o conteúdo do modelo de mapeamento existente;

5 – Por partição de domínio: cria o mapeamento *domain* para a entrada do conteúdo do esquema CIM de acordo com o modelo de gerenciamento existente e extensões necessárias;

3.2.4.2 Gravação das Decisões de Mapeamentos

Quando o conteúdo do modelo é mapeado entre diferentes esquemas, informações sofrem perdas ou são perdidas. Para completar, "*scratch pad*" são expressos no modelo CIM usando qualificadores, o qual atual como extensão para o modelo. Este *scratch pad* são críticos para a transmissão do *core, common e extesion model* contendo várias tecnologias usadas para construir aplicações de gerenciamento.

Para fazer entender a regra do *scratch pad* dentro do repositório, é necessário entender os cenários de importação e exportação para diferentes partições do repositório. Estes mapeamentos podem ser organizados em duas categorias: homogêneo e heterogêneo:

- A categoria homogênea: inclui o mapeamento quando a sintaxe da importação é expressa igualmente a da exportação (por exemplo, software MIF dentro e software MIF fora).
- A categoria heterogênea: inclui o mapeamento quando as sintaxes de importação são expressas diferentemente da exportação (por exemplo, MIF dentro e GDMO

fora). Para a categoria homogênea, a informação pode ser gravada pela criação de qualificadores durante a operação de importação assim o conteúdo pode ser apropriadamente exportado. Para a categoria heterogenia, os qualificadores devem ser adicionados depois que o conteúdo é carregado em uma divisão do repositório.

A figure 8, mostra o esquema de importação para o esquema Y, começando homogeneamente a exportação para o X e heterogenicamente exportação para o Z. Cada um exporta com diferentes *scratch pad.*

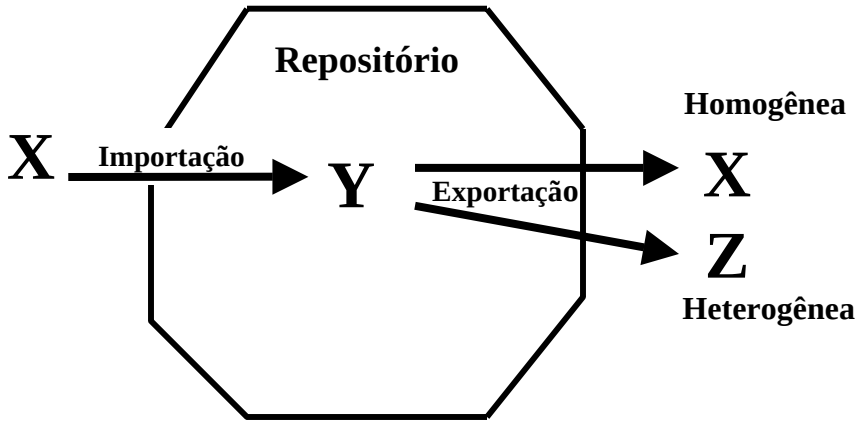

Figura 8: Exportação homogênea e heterogênea [RFC 3006].

A definição de categorias heterogêneas é atualmente baseada em conhecimento com esquemas carregados no repositório. O caminho mais geral para exportação do processo é usar múltiplos *scratch pads*. Um *scratch pads* é criado quando esquemas são carregados, outro é adicionado para identificar o mapeamento e outros que a origem importa.

A Figura 9 mostra como o *scratch pads* de qualificação são usados para cada partição (*technique, recast*) dentro do repositório CIM.

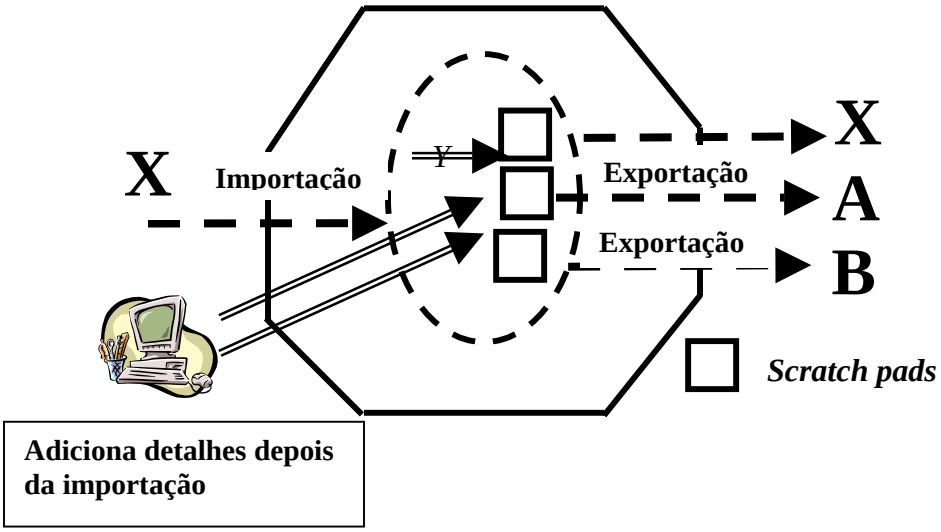

Figura 9: Aplicação de *Scratch pads*.

3.2.4.3 Contexto das Partições

Para cada partição a um contexto de utilização de *scratch pad* [RFC 3006]:

Para a partição técnica, não precisa *scratch pad* visto a homogeneidade dos mapeamentos. Então, por definição, existe um mapeamento homogêneo para cada esquema protegido por partição *technique*. Este mapeamento cria objetos CIM para a construção sintática de esquema e cria associações para eles poderem ser combinados. (Por exemplo, grupos MIF incluem atributos).

Para partição de *recast*, são múltiplos *scratch pads* para cada subdivisão, desde a exportação da requisição de cada objetivo. A ocorrência por que parte da criação do mapeamento *recast* envolve mapear a construção da origem do modelo CIM. Então para a sintaxe MIF, o mapeamento deve ser criado para componentes, grupos e atributos dentro da construção do modelo CIM com objeto, associações e propriedades.

Conforme exemplo específico, uma decisão deve ser criada se há grupo ou componentes no quadro de objetos. Com isto é possível ter dois diferentes algoritmos de mapeamento *recast*, um que mapea grupos de objetos com qualificador que preservam o componente, e um que mapea o componente dentro do objeto com qualificador que

preserva o nome do grupo com a propriedade. Então, o *scratch pads* em partições são organizados por objetivos técnicos e algoritmos empregados.

Para partição de *domain*, são dois tipos de mapeamento. O primeiro é similar a partição *recast*, onde algumas divisões do domínio são mapeadas dentro da sintaxe de outros esquema. Este mapeamento pode usar o mesmo qualificador *scratch pads* e algoritmos associados que são desenvolvidos para a partição *recast*. O segundo tipo de mapeamento facilita documentação do conteúdo envolvido entre o domínio diferente e algum outro modelo (por exemplo, grupo de softwares).

Portanto, a escolher da partição, desenvolver o mapeamento e identificar o qualificador necessário para capturar perda potenciais são fundamentais para o desenvolvimento do mapeamento da informação.

3.3 Policy Information Base

O PIB define o fornecimento de políticas através de uma estrutura específica para que possam ser transmitidas aos dispositivos da rede com o propósito de configuração.

O modelo desta estrutura é constituído por classes (*Provisioning Class - PRCs*) e objetos (*Instance Provisioning - PRIs*), estas classes residem em um depósito virtual de informações chamado *policy information base*.

3.3.1 Conceitos Gerais

As políticas para serem aplicadas dependem de vários fatores, assim com características fundamentais e imutáveis de cada interface (e.g., *ethernet* ou *frame relay*), o status da interface (e.g., *half* ou *full* duplex), ou configurações do usuário (e.g., filial ou matriz da empresa) [RFC 3318]. Preferencialmente que especifiquem as políticas claramente para cada interface e para todos os dispositivos da rede, políticas são definidas em termos de funcionalidades da interface.

Para descrever as funcionalidades da interface é utilizado o conceito de função. As funções são simples *strings* associadas à interface. Uma determinada interface pode ter várias funções simultaneamente. Classes *provisioning* têm um atributo chamado de "*RoleCombination*" o qual é ordenado em grupo de funções. As ocorrências de determinadas classes *provisioning* são aplicadas nas interfaces se e somente se o grupo de funções estiver compatível ao grupo de funções da interface.

Então, funções fornecem caminhos para a aplicação do grupo de políticas, especificando interfaces de uma maneira consistente para todos os dispositivos da rede. Isto é, define políticas aos dispositivos e identifica especificamente suas interfaces.

Além disso, se a mesma política for aplicada para várias interfaces, esta política necessita ser requerida ao dispositivo somente uma vez, de preferência pela mesma interface, enquanto são configuradas com a função-combinação. No evento que o administrador precise de uma única política para cada interface, o administrador pode configurar cada interface com uma única função.

O PEP envia todas as capacidades dos grupos de nomes (*capability set name*), função-combinação, interface controlador de política (*Policy Controlled Interfaces)*, e relacionamentos para o PDP na primeira mensagem de requisição COPS, para negociar qualquer atualização ou exclusão de políticas.
O PDP pode instalar novos objetos ou transformar os existentes nestes PRIs (*Provisioning Instance)*. Esta operação pode também ocorrer em subseqüentes mensagens de requisição, gerando mensagens de respostas sincronizadas para alterações das configurações locais.

3.3.1.1 Exemplo de Funcionamento

O funcionamento das funções pode ser mais bem entendido por um exemplo baseado na RFC 3318. Supondo que tenha um dispositivo com três interfaces, como as seguintes funções:

Interface 1: "financeiro"
Interface 2: "financeiro"
Interface 3: "gerência"

Supomos que haja PDP com duas políticas:

Pacote 1: Pacote do departamento de finanças leva código DSCP 5.
Pacote 2: Pacote da gerencia leva código DSCP 6.

Para obter política, o PEP reporta ao PDP que têm algumas interfaces com função-combinação “financeiro” e outras com função-combinação “gerencia”. Em resposta, o PDP baixa (*download)* a política P1 associada com função-combinação “financeiro” e a segunda política P2 associada com função-combinação “gerencia” e instala.

Agora supondo que as pessoas do financeiro ligadas a Interface dois sejam promovidas para a gerencia e assim o sistema administrador adiciona a função “gerencia” a Interface 2. O PEP agora reporta para o PDP que tem três funções-combinações:

interfaces com função-combinação “financeiro”, com função-combinação “gerencia” e com função-combinação “financeiro+gerente”. Em resposta, o PDP requisita (*download)* uma terceira política adicional com a nova função combinação “financeiro+gerente”.

Como o PDP determina a política para essa nova função-combinação, é inteiramente responsabilidade do administrador da rede. Isto pode ser realizado através de algoritmos, função ou protocolos.

O PDP requerido para determinar que política aplicar a esta nova função-combinação baixa *(download)* uma terceira política para PEP da função-combinação “financeiro+gerencia”, mesmo se esta política é igual uma já baixada. O PEP não permite a construção de políticas para novas funções-combinação de políticas existentes.

3.3.1.2 Gerenciamento da Função-Combinação do PDP

O PEP notifica o PDP da função-combinação determinada para cada interface e capacitam os grupos de nomes (identificação dos grupos) em uma requisição de configuração COPS (objetos da *frwkIfRoleComboTable*).

A primeira requisição enviada ao PDP deve ser a requisição “*full state*”. A requisição “*full state*” de um PEP inclui as notificações e instalações da tabela de objetos PRIs enviadas em uma mensagem prévia. Uma requisição é especificada como requisição “*full state*” por preparar o atributo (*frwkPibIncarnationFullState)* do objeto (*frwkPibIncarnation* PRI) para envio da requisição.

Todos os objetos existentes (*frwkIfRoleCombo)* devem ser enviados para o PDP na primeira requisição de configuração identificadora. Se a função-combinação não determina valores específicos, por padrão (*default)* a função-combinação deve ser enviada ao PDP e a todos os PEPs ativos (*ifIndices).*

Para resposta desta requisição de configuração, caso necessário, o PDP pode enviar políticas ao PEP em resposta a decisão ou enviar uma decisão de cancelamento. O PEP então envia uma mensagem requerendo nova informação para decisão. A qualquer momento, o PDP pode atualizar determinada função-combinação para uma interface específica. O PDP envia atualizações PRIs para a tabela (*frwkIfRoleComboTable).*

Quando associações da função-combinação são atualizadas pelo PDP o PEP deve enviar requisição "*full state*" de atualização para todo contexto aberto. Um contexto é um objeto do módulo PIB identificado por um único e particular tipo de cliente COPS. Isto acontece mesmo que requisição de estado PEP mude devido a evento interno ou se o estado for mudado por um PDP.

Completando, o PDP não deve esperar receber informações de atualização de função-combinação até que recebe a confirmação de atualização. Se o PDP não receber a atualização de alguma requisição identificadora, o PEP não deve enviar atualizações para seus objetos (*frwkIfRoleCombo).*

3.3.2 Múltiplos Objetos PIB

O COPS suporta a configuração de políticas de múltiplos, individuais e independentes objetos PIB. O objetivo é facilitar a configuração da política nos dispositivos ativos pela decisão do PDP [RFC 3318].

O contexto COPS pode ser definido como uma independente requisição de estados COPS para uma particular categoria (*client-type*). Um contexto pode ser dividido em contexto *outsourcing* e contexto de configuração [RFC 3318]:

- Contexto de configuração é um objeto da PIB funcionando e controlado pelo PDP, o qual contém informações de instalação do dispositivo. Esta informação de configuração dita o comportamento do dispositivo como especificado pelo PDP.
- Contexto *outsourcing* é um objeto PIB que funciona de apoio ao PEP, requisitando ações para o PDP. A ação de a requisição é interpretada no domínio do cliente.

O contexto de configuração pertence a um grupo de contextos para um específico tipo de cliente, os quais somente um contexto de configuração pode ser ativo. Entretanto no contexto *outsourcing* múltiplos podem ser ativos simultaneamente.

Com o protocolo COPS, cada um destes estados é identificado por um único cliente. A criação ou eliminação destes objetos PIB pode ser controlado por PDP como descrido no COPS ou pelo funcionamento de um evento PEP.

O PEP deve abrir “requisições de estados” para configuração e para determinado tipo do cliente. A ”requisição de estados” do PEP pode ser iniciada pelo PDP ou gerado pelo o PEP por *outsourcing* devido ao evento local, o qual é completamente especificado por dados transmitidos na requisição.

Embora muitos objetos PIB possam ser configurados em um dispositivo, somente um contexto do grupo contexto de configuração pode ser ativo a qualquer tempo e somente o ativo é selecionado pelo PDP.

O *framework* PIB suporta o atributo (*frwkPibIncarnationActive)* na tabela (*frwkPibIncarnationTable)* para permitir ao PDP marcar os objetos PIB como início ativo em uma mensagem de política COPS, e similarmente, reportar o estado (ativo ou não) do objeto PIB para o PDP em um mensagens requisição COPS. Quando o PEP instala um atributo (*frwkPibIncarnationActive)* que é “verdadeiro” em um objeto PIB o qual pertence ao grupo “contexto de configuração”, o PEP deve assegurar, re-ajustar o atributo se necessário. Quando o atributo (*frwkPibIncarnationActive)* for “falso” em todos as outras instalações pertence a este grupo será falso.

Para troca de contexto, o PDP teria o grupo de atributos (*frwkPibIncarnationActive)* “verdade” no contexto necessário para tornar o contexto ativo. O PDP teria grupos destes atributos em um contexto “falso” somente para desativar contexto PEP. Se um contexto é inativo, o PEP não deve ter políticas a aplicar em qualquer contexto de configuração.

3.3.3 Capacidade do Dispositivo de Configuração e Reportagem

Cada dispositivo de rede fornece um serviço baseado em políticas particulares inerentes a sua capacidade. Estas capacidades podem ser específicas de hardware, por exemplo, classificação de interfaces por hardware, ou pode ser configurações estáticas, por exemplo, filas de prioridades. Essa capacidades são organizadas em “grupo de

capacidades" e determinadas por um único nome (*frwkCapabilitySetName*) e associadas ao grupo função-combinação.

Deste modo, todas as funções-combinação podem ser associadas a um grupo de interfaces. Estas capacidades são comunicadas para o PDP quando a política é requisitada para o PEP. Verificada a capacidade, o PDP pode enviar o PRIs relevantes para o dispositivo determinado.

A especificação de capacidades PRCs podem ser definidas em outras PIBs. Essas capacidades de objetos são agrupadas via tabela (*frwkCapabilitySetTable).* Se o PEP deseja enviar capacidades de informações para o PDP, o PIB deve indicar qual a capacidade PEP pode enviar ao PDP ou através de notificações (SPPI - *Structure of Policy Provisioning Information*). Se um PIB não tem capacidade para comunicar o PDP, não envia objetos para a tabela (*frwkCapabilitySetTable*).

3.3.4 Reportagem de Limitações do Dispositivo

Para facilitar a política de instalação, é importante entender as limitados dos dispositivos em relação às capacidades determinadas. As limitações podem ser classes baseadas: por instalação suportada, por notificação, por número limitado de classes de objetos criadas ou atributos base [RFC 3318].

Um PDP pode evitar certos problemas de instalação de forma pró-ativa, por levar em conta a limitação do dispositivo antes de instalar a política, preferencialmente do que a forma reativa durante a instalação. Os critérios relacionados são capacidades, limitações e comunicação ao PDP quando a política é requisitada.

A reportagem das limitações dos dispositivos pode ser acompanhada por valores guias usados por um PDP para determinar valores aceitáveis para atributos identificados.

3.3.5 Componentes do Framework PIB

O *Framework* PIB define quatro grupos de PRCs (*Provisioning Class*) [RFC 3318].

3.3.5.1 Grupos de Classes Base PIB

Este grupo contêm PRCs destinadas a descrever o suporte ao PEP, limitações de atributos e configurações [RFC 3318].

- **Tabela PIB:** Este PRC contém exatamente uma ordem (correspondente a uma PRI) por contexto. Isto identifica o PDP que foi o último baixar (*download)* a políticas para o dispositivo e também contém um identificador para a versão da política atual. Este identifica tanto a sintaxe como os valores significantes somente para os PDPs. Isto é planejado para ser um mecanismo segundo o qual um PDP aceita a conexão do PEP podendo facilmente identificar a política descrita. Este PRC define um *flag* através do quais os contextos instalados são divididos em um grupo de contexto, onde o contexto de configuração permite somente um contexto ativo e o contexto *outsourcing* no qual são todos ativos. A descrição PRC também define um atributo para indicar qual contexto de configuração é ativo no presente momento no grupo "contexto de configuração". As descrições do objeto são específicas para a particular categoria.

- **Tabelas de Limitações dos Componentes:** Alguns dispositivos não são capazes de implementar completamente os valores para todos os atributos. A principio, cada PRC suporta um grupo de erros que o PEP pode reportar ao PDP no evento que a política especificada não é implementada.

  Isto pode ser desejável ao PDP para ser informado das limitações do dispositivo antes de aplicar política de instalação. Enquanto o erro pode indicar o valor do atributo inaceitável para o PEP, isto não ajuda a certificar quais valores podem ser aceitos pelo PDP. Para suavizar estes limites, o PEP pode reportar algumas limitações de atributos e/ou classes guiando valores para atributos na tabela de limitação de componentes.

- **Tabela identificação de dispositivos:** Este PRC contém um simples PRI que contém informação específica dos dispositivos que são usados para facilitar

eficientemente a instalação da política pelo PDP. O objeto deste PRC é informado para o PDP em uma mensagem de requisição COPS de forma que o PDP pode levar em conta certas características dos dispositivos durante a instalação da política.

3.3.5.2 Grupos de Capacidades dos Dispositivos

Esse grupo contém o PRCs que descreve a característica da interface do dispositivo e a função-combinação determinada por eles [RFC 3318].

- **Tabela Grupo de Capacidades:** a capacidade de suporte do PEP é descritas por ordem neste PRC (*frwkCapabilitySetTable*). Cada ordem, ou objeto desta classe, associa um único nome de capacidade ao grupo de capacidades da entidade do PEP. Um único nome é usado para formar um grupo de capacidades. A referência da capacidade pode especificar objetos em tabelas de capacidade de qualquer PIB. O PEP notifica o PDP desses grupos de capacidades e então o PDP configura a interface, por função-combinação. O único nome (*frwkCapabilitySetName*) não pode ser confundido com o objeto *IfType* da Interface Grupo MIB [RFC 2863].

- **Interface e Tabelas Função-Combinação:** A tabela grupo de capacidades descreve a entidade PEP (por exemplo, interfaces) por capacidades e por determinar ao grupo de capacidades um nome único (*frwkCapabilitySetName*). É possível que o comportamento da interface seja determinado utilizando específica função-combinação para o grupo de capacidades. Isto permite as interfaces com o mesmo grupo de capacidade ser escolhidas por diferentes políticas, baseadas na atual função determinadas para ele.

  No PDP, configurações são feitas em termo destas interfaces de capacidades, grupos de nomes e determinadas função-combinação para elas. Então, cada ordem dessa classe é um ***<interfaces Index, interface nome do grupo de capacidade, função-combinação***>, que indicam a função que foi determinada por um grupo particular de capacidade (como identificado por *frwkRoleComboCapSetName*) e por uma particular interface.

Note que o único critério para esse PRC ter todos os atributos são *frwkRoleComboCapSetName* possuir múltiplas funções-combinações, que são associadas com o IfIndex, este PRC responde a perguntas de “quais interfaces tem uma específica função-combinação” e “que função-combinação uma interface faz parte ?”.

3.3.5.3 Grupo de Classificador

Este grupo contém o IP e elementos classificadores de rótulos internos. O grupo de tabelas consiste de um tabela filtro base que contém o indicador *InstanceId* e o *flag* negociador para o filtro. Este tabela *(frwkBaseFilterTable)* é estendido para formar a tabela de filtro IP e a tabela rótulo interno. Filtros podem também ser definidos e usados para estender a tabela filtro base.

As classes estendidas não têm uma separação de valor indicado. Objetos de classe estendidas tem o mesmo índice como relação a suas classes de objetos base. A herança é executada usando as palavras *extendes* como definidas em SPPI.

3.3.6 Considerações sobre Segurança

O PIB é usado para configurações utilizando COPS, sendo que qualquer dispositivo pode ser configurado ou atualizado. Isto é relatado para controlar o acesso a informações sensíveis e a capacidades de configuração dos dispositivos.

Abaixo temos os números de classes *provisioning* definidas no PIB, onde a condição de acesso (PIB-ACCESS) a instalação e notificação da instalação. Estas são [RFC 3318]:

- *frwkPibIncarnationTable*: acesso malicioso deste PRC, pode levar o PEP ao uso de contexto de políticas incorretos;
- *frwkReferenceTable*: acesso malicioso deste PRC, pode levar o PEP a interpretar a instalação da política de uma maneira incorreta;
- *frwkErrorTable*: acesso malicioso deste PRC, pode levar o PEP a supor incorretamente que o PDP não pode processar esta mensagem;
- *FrwkCapabilitySetTable, frwkRoleComboTable* e *frwkIfRoleComboTable*: acesso

malicioso destes PRCs, pode levar o PEP a aplicar políticas para interfaces erradas;

- *FrwkBaseFilterTable, frwkIpFilterTable, frwk802FilterTable e frwkILabelFilterTable*: acesso malicioso desta PRCs, pode levar a classificação do tráfego para o PEP em direção a políticas incorretas serem aplicadas;
- *frwk802MarkerTable, frwkILabelMarkerTable*: acesso malicioso deste PRCs pode causar sinalização do tráfego para o PEP em direção a políticas serem aplicadas;

Do mesmo modo objetos podem ser considerados sensíveis ou vulneráveis em qualquer ambiente de rede. O suporte a “instalação” ou “atualização” de decisões enviadas sobre COPS em um ambiente não seguro sem adequada proteção podem ter uma malefícios nas operações da rede.

Existem números de classes *provisioning* neste PIB, que podem conter informações sensíveis à perspectiva de negócios, em que eles podem representar um contrato de serviços a clientes ou filtros que o servidor escolhe para aplicar aos clientes no tráfego de entrada ou saída. Isto pode ser importante para controle “notificação” de acesso para estes PRCs e possibilita igualar códigos e valores destes PRIs quando enviam sobre a rede via COPS.

O uso de IPSEC entre o PDP e PEP, é descritos em COPS, provendo a proteção necessária a ameaças de segurança. Então, mesmo a rede sendo segura, este não controla para quem é permitido a instalação ou atualização dos PRIs no PIB.

3.3.7 Conclusão do Capítulo

Para melhor utilização dos modelos apresentados, devem-se levar em conta as características da aplicação, funcionalidade e desempenho. Também para a escolha do repositório é necessário avaliar suas características para haver conformidade entre aplicação e o repositório.

No modelo CIM (seção 3.2) sugerido pela DMTF com repositório são organizadas e estruturadas as informações para permitir a distribuição entre seus grupos. Também os objetos utilizados, os tipos de expressões e sintaxe para capturar a representação são fundamentais para a gerencia do domínio.

Outro ponto relevante é o modelo CIM ser baseado em orientação a objeto, onde são definidos claramente as aplicações de gerenciamento através de classes, objetos, propriedades, métodos e associações. Por fim, a estrutura do modelo CIM possibilita o melhor entendimento conceitual sobre o ambiente gerenciado e facilita aos programadores a identificação da informação no *framework*.

Na seção 3.3 foi discorrido *Policy Information Base*, outro modelo sugerido pela DMTF com repositório de políticas. O PIB define a estruturação para transmissão de políticas de gerenciamento entre os dispositivos gerenciados. Este modelo é constituído por classes e objetos sendo também baseado em orientação a objetos.

O PIB suporta múltiplos objetos, e todos eles reportam suas capacidades e limitações a seus PDPs. Ao longo da seção foi descrito a relação entre a interface e sua função-combinacão para configuração e gerenciamento do dispositivo.

A percepção dos repositórios de política é fundamental para a estrutura do framework de políticas, onde são armazenadas todas as políticas de configuração dos dispositivos gerenciados. Compreender sua organização e construção facilita a transmissão das políticas, gerenciamento e aplicabilidade.

No trabalho o repositório foi abstraído devido ao foco ser a distribuição de política e não o armazenamento, todavia este capítulo elucidou os principais repositórios.

4. Distribuição de Políticas

4.1 Introdução

Este capítulo tem o objetivo de relatar protocolos de distribuição de políticas e os principais aspectos da comunicação aplicando tecnologias e padrões conhecidos. Neste contexto, temos o cenário tecnológico para poder entender a padronização IETF “COPS” e as tecnologias que permitem sustentar a proposta da construção de um protocolo de distribuição de políticas de gerenciamento utilizando estrutura XML e SOAP.

Na seção 4.2 o protocolo de comunicação padronizado pela IETF será abordado. O *Common Open Policy Services* – COPS é um protocolo descrito na RFC 2748, onde é apresenta toda a funcionalidade e aplicabilidade na interação entre os dispositivos (PEP) e o servidor de políticas (PDP). Ao longo da seção serão descritos fatores decisivos como: comunicação entre PEP e PDP, sincronização das mensagens, aspectos de segurança e operações possíveis com o protocolo COPS.

O *Web services* será contextualizado na seção 4.3. Onde se apresenta sua estrutura de serviço, funcionalidade que propõe e características que visam dinamizar a comunicação em sistemas distribuídos.

Na seção 4.4 é descrito o padrão W3C para representar conteúdo de documentos bem definidos o XML. Suas principais funcionalidades e otimizações oferecidas.

Na seção 4.5 temos as principais características do protocolo SOAP e sua função de permite que um documento XML possa ser usado para trocar informações entre sistemas em um ambiente descentralizado e distribuído. Completando, SOAP é parte imprescindível na proposta por sua portabilidade. Seguindo na seção, descrevemos as partes fundamentais do SOAP, atributos, tamanho da mensagem, complexidade do empacotamento, interoperabilidade e por final um modelo de requisição/resposta SOAP. Este capítulo pretende contemplar as principais tecnologias utilizadas na concepção da proposta bem como na sua implementação.

4.2 Common Open Policy Service - COPS

4.2.1 Conceitos Gerais

O protocolo *Common Open Policy Service* – COPS é um modelo de requisição/respostas que pode ser usado para trocar políticas de informação entre um servidor de políticas PDP e seus clientes PEP

Um dos principais objetivos deste protocolo de políticas de controle são a simplicidade e extensibilidade. As fundamentais características do protocolo COPS são [RFC 2748]:

1. O protocolo emprega um modelo cliente/servidor, onde o PEP envia mensagens de requisições, atualização e exclusão para um PDP. O PDP retorna a decisão novamente para o PEP;
2. O protocolo usa TCP como protocolo de transporte pela sua confiabilidade nas trocas de mensagens de política entre clientes e o servidor;
3. O protocolo é extensível com isso alcança objetos devidamente identificados e suporta informações específicas de diversos clientes sem requerer modificação do protocolo COPS;
4. O COPS prove segurança na autenticação, proteção repetições e integridade das mensagens. COPS pode também utilizar protocolos existentes para segurança assim como IPSEC [IPSSEC] para autenticação e a segurança da comunicação entre o PEP e o PDP;
5. O protocolo é determinado em dois principais aspectos:
   (1) Determina requisição/resposta, sendo dividido entre cliente e servidor; e
   (2) Determina diferentes eventos de requisição/resposta para as associações;

6. Completando, o protocolo é determinado para permitir a distribuição de políticas no servidor "*push*" (empurrar) enviando informações de

configuração para o cliente e removendo-as quando necessário desde que os mesmo pertençam ao mesmo domínio;

4.2.1.1 Modelo Básico de Funcionamento do COPS

O COPS é usado para comunicar política de gerenciamento entre PEP e o PDP, resguardando as particularidades e tipos dos clientes. Na Figura 10 é demonstrada a comunicação entre um dispositivo (PEP) e seu servidor de políticas (PDP). Entretanto, a escolha em determinadas situações de PDP local pode ser bastante interessante caso ausência do PDP responsável pelo domínio. O PDP local tem a função de fazer localmente as decisões através de um conjunto de políticas para o dispositivo específico.

Figura 10: Distribuição de políticas COPS.

O PEP é responsável por iniciar a conexão TCP ao PDP. O PEP utiliza a conexão TCP para enviar requisições e para receber respostas do PDP.

A Comunicação entre o PEP e PDP é principalmente no formato de requisição/resposta, de qualquer forma o PDP pode ocasionalmente enviar uma atualização da decisão para o PEP forçando-a transformar as requisições previamente expressas. O PEP também tem a capacidade para reportar ao PDP que teve completo sucesso na execução da decisão, útil na contabilidade e monitoração da finalidade. O PEP é responsável por notificar o PDP quando uma requisição determinada sofre mudanças. Também por apagar qualquer determinação não aplicável devido a eventos de clientes ou decisões emitidas pelo servidor.

Quando o PEP envia a requisição de configuração, aguarda do PDP o envio do nome da unidade de configuração de dados através da mensagem de resposta para o PEP, conforme desejada pela requisição de configuração. Depois de nomeada a unidade de dados de configuração e instalada com sucesso no PEP, o mesmo pode enviar uma mensagem informando para o PDP da confirmação da instalação. O servidor pode então

atualizar ou remover a informação de configuração através de novas mensagens de decisão. Quando o PDP envia uma mensagem de decisão para remover a informação de configuração do PEP, o mesmo apaga a configuração específica e enviar uma mensagem de estado para o PDP como confirmação.

O protocolo de política foi projetado para comunicar a identificação própria do objeto o qual contêm os dados necessários para a identificação da requisição, tipo de requisição, referenciando previamente a requisição instalada, re-estabelecendo a política de decisões, reportando erros, provendo integridade da mensagem e transferindo informações do cliente específico [RFC 2748].

Para distinguir entre diferentes tipos de cliente é especificada a identificação em cada mensagem. Diferentes tipos de clientes podem ter distintos dados específicos e podem requerer diferentes tipos de políticas de decisão. Isto é expresso em cada novo tipo de cliente.

O contexto de cada tipo de requisição corresponde ao tipo de evento que o desencadeia. No COPS a identificação de objetos, os tipos de requisição e a mensagem (se aplicável) desencadeiam políticas de eventos através de tipo de mensagem e do campos de requisição [RFC 2748].

O COPS identifica três tipos de evento *outsourcing*:

1- A chegada da mensagem de entrada;
2- A distribuição no local do recurso;
3- O encaminhamento da mensagem de saída;

Há quatro tipos de requisições usadas por tipos de clientes que desejam receber informações de configurações do PDP. Isso permite ao PEP lançar uma requisição de configuração para um específico dispositivo nomeado ou módulo que requerer informações de configurações para ser instalado.

O PEP pode também ter a capacidade para fazer política de decisão local através do LPDP (local PDP), no entanto, o PDP do domínio fica informado dos pontos de decisão todo tempo. Portanto, o PDP deve conceder acesso para todas as informações

relacionadas à construção da política de decisão. Para facilitar este funcionamento, o PEP deve enviar a informação de requisição para o PDP remoto via LPDP. O PEP deve então agir de acordo com a decisão do PDP.

A tolerância à falha é uma capacidade exigida do protocolo COPS, particularmente devido ao fato de associarem-se com a segurança e gerenciamentos de serviços em redes distribuídas. A tolerância à falha pode ser alcançada por ter ambos PEP e PDP remotos constantemente verificando suas conexões através de mensagens.

Quando uma falha é detectada, o PEP deve tentar re-conectar ao PDP ou tentar a conexão para um backup/alternativo do PDP local. Quando não a conexão, o PEP deve reverter para decisão local. Somente uma conexão é restabelecida, o PEP é esperado para notificar o PDP de qualquer evento novo ou removido passado pelo local de controle de admissão depois da conexão perdida.

O PDP remoto pode requisitar que todos PEPs internos declarados sejam sincronizados (todos previamente localizados). Depois que há falha e antes da nova conexão é completamente funcional a interrupção do serviço para poder ser minimizado o PEP cachê, revisando o comunicado da resposta e continuando o uso das políticas por qualquer quantidade de tempo.

4.2.2 Descrição do Protocolo

O protocolo COPS obedece a seguinte descrição para o formato da mensagem e troca de objetos entre o PEP e o PDP [RFC 2758].

4.2.2.1 Cabeçalho Comum

Cada mensagem COPS consiste de um cabeçalho e seguindo por um número de tipos de objetos em bits.

0	1	2	3
Version	**Flags**	**Op Code**	**Client-type**
Message Length			

Figura 11: Cabeçalho COPS.

- ***Flags* (**4 bits**)**: define o valor *flag* (todos outros *flags* deve começar com 0);
  Este *flag* é usado quando a mensagem é solicitada por outra mensagem COPS.
  Este *flag* não é para ser usado (valor = 0) ao menos que seja especificado.

- ***Op Code*** (8 bits): Operações COPS.

  1 = Request (REQ)
  2 = Decision (DEC)
  3 = Report State (RPT)
  4 = Delete Request State (DRQ)
  5 = Synchronize State Req (SSQ)
  6 = Client-Open (OPN)
  7 = Client-Accept (CAT)
  8 = Client-Close (CC)
  9 = Keep-Alive (KA)
  10 = Synchronize Complete (SSC)

- ***Client-type*** (16 bits): O *client-type* identifica a política do cliente. Interpretação de todos os objetos encapsulados é relativo ao *client-type*. *Client-type* que usa o maior número de bit significante no campo *client-type* são iniciativa definidas.

- ***Message longth*** (32 bits): Tamanho da mensagem em octetos, a qual inclui o cabeçalho COPS padronizado e todo encapsulamento de objetos. A mensagem deve ser ordenada em intervalos de octetos.

4.2.3 Comunicação do COPS

O protocolo COPS usa conexão TCP entre o PEP e o PDP. Uma implementação de servidor PDP deve ouvir no número de porta conhecida (por exemplo, COPS = 3288).

O PEP é responsável por iniciar a conexão TCP para o PDP. A localização do PDP pode também ser configurada, ou obtida através de mecanismo de localização de serviço (fora do escopo da RFC 2748).

Um PEP pode suportar múltiplos tipos de clientes, este pode enviar múltiplas mensagens, especificando cada tipo particular de cliente sobre um ou mais conexões TCP. Da mesma forma, um PDP residente em um determinado endereço e número de porta suporta (número limitado) tipo de clientes. Determinar o tipo de cliente suportado é capacidade do PDP e também aceitar ou rejeitar cada tipo de cliente independentemente. Caso o cliente seja rejeitado, o PDP pode redirecionar o PEP para um endereço alternativo de PDP e determinar a porta TCP para um tipo de cliente através do COPS. Diferentes números de portas TCP podem ser usados para redirecionar o PEP para outro PDP implementado, executado em um mesmo servidor.

O protocolo COPS não fornece meios para suportar múltiplos tipos de clientes em apenas um servidor PDP. De modo que são deixados para a arquitetura da plataforma do servidor determinado. É possível a um único PEP poder abrir múltiplas conexões a PDPs. Isto acontece quando são PDPs fisicamente diferentes, suportando diferentes tipos de clientes conforme a Figura 12.

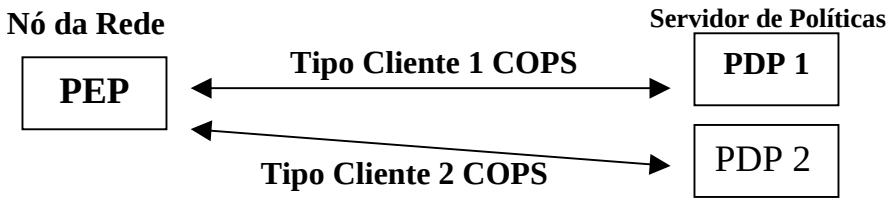

Figura 12: Comunicação de um PEP com dois PDPs utilizando COPS.

Quando uma conexão TCP é destruída ou perdida, o PDP aguarda a requisição de estado “normal” para retomar as trocas de mensagens com o PEP. De modo que o PEP detecta a perda da conexão devido ao tempo expirado e este deve enviar uma mensagem de cliente perdido para cada tipo de cliente contendo a indicação de ”falha na comunicação”.

Completando, o PEP deve continuamente tentar o contato com PDP primário, caso sem sucesso, qualquer PDP conhecido pode realizar o backup. Especificamente o PEP deve fazer tentativas a todos os PDP relevantes com o qual tenha existido

configuração até que possa estabelecer a conexão. Se a PEP em comunicação com PDP de backup, este é responsável por redirecionar o PEP novamente para o PDP primário.

4.2.4 Sincronização COPS

Quando desconectando de um PDP, o PEP deve retornar a última decisão local criada. Uma vez a conexão restabelecida, o PEP aguarda a notificação do PDP de qualquer evento que tenha passado no local para controle de admissão. Completando, o PDP requisita que todos os PEPs determinados sejam re-sincronizados (todos requisições previamente instaladas) por envio de uma mensagem de sincronização [RFC 2748].

Depois que à falha e antes de uma nova conexão é totalmente funcional a interrupção do serviço, podendo ser minimizado se o cachê PEP previamente comunicar as decisões e continuar o uso delas. As especificações de regras para tais comportamentos são definidas com apropriadas especificações dos tipos de clientes COPS.

Um PEP que determina troca de cachês com um PDP desconectado deve comunicar este fato a qualquer PDP com o qual é capaz de se re-conectar depois. Isto é complementado por incluir o endereço e porta TCP do último PDP para o qual o PEP envia as mensagens.

O último endereço PDP somente será incluído se caso o PEP esteja em sincronismo como: o serviço interrompido e o PDP ainda conter o completo estado do PEP. O PDP pode ainda não escolher todos PEPs para sincronização de estados.

É de responsabilidade do PEP a atualização dos PDPs de todas as transformações de estados que ocorrem durante a interrupção do serviço, incluindo qualquer estado comunicado ao PDP anteriormente que possa ser excluído depois da perda da conexão. Este deve ser explicitamente excluído depois do restabelecimento da comunicação. Se PDP requisita sincronização, o PEP deve transmitir todos os estados atuais. Se o PEP travar e perder todos estados cachês de um tipo de cliente, simplesmente não será incluído como último endereço de PDP.

4.2.5 Inicialização do PEP

Depois de a conexão ser estabilizada entre o PEP e o PDP e negociada a segurança (se requerida), o PEP envia uma ou mais mensagens ao cliente, uma para cada tipo de cliente suportado pelo PEP. A mensagem ao cliente deve conter o endereço do último PDP [RFC 2748].

Se não há decisões existentes no cachê do PDP anterior, este não deve ser incluindo na mensagem ao cliente. Cada mensagem ao cliente deve pelo menos conter um cabeçalho comum apontando um tipo de cliente suportado pelo PEP.

O PDP então responde com separadas mensagens ao cliente uma para cada tipo de cliente requisitado pelo PEP. Se um específico tipo de cliente não é suportado pelo PDP, a resposta determina o tipo do cliente não suportado, possibilitando sugestões conforme alternativos PDPs, endereços e portas.

De outra forma, o PDP pode enviar ao cliente uma especificação do intervalo de tempo entre as mensagens aguardadas e o PEP pode começar distribuir requisições para o PDP.

4.2.6 Operações *Outsourcing*

Em cenários *outsourcing*, quando o PEP recebe um evento que requer uma nova política de decisão, é enviada uma mensagem de requisição para o PDP. O qual específica qualificações conforme o evento para o tipo de cliente. O PDP então cria uma decisão e envia uma mensagem de retorno para o PEP. Desde que a requisição é determinada, a requisição irá ser instalada no PDP.

O único identificador (único por conexão TCP e tipo de cliente) especificado em ambos as mensagens (requisições), identifica a decisão correspondente da requisição determinada. O PEP é responsável for apagar esta requisição determinada no que a requisição não é aplicada.

O PEP pode atualizar uma requisição instalada previamente por repetir-la. O PDP é, portanto esperado para criar a nova decisão e enviar retorno de mensagem de decisão para o PEP. Da mesma forma, o servidor pode transformar uma decisão previamente

emitida em qualquer requisição atualmente instalada a qualquer tempo por distribuir uma mensagem de cancelamento da decisão.

4.2.7 Operações de Configuração

No cenário de configuração *outsourcing*, o PEP irá criar uma requisição de configuração para o PDP, a interface particular e funcionalidades podem ser especificadas no nome do cliente objeto da informação [RFC 2748].

O PDP então envia várias decisões potenciais contendo nomes de unidades e configuração de dados para o PEP. O PEP aguarda para instalar e usar as configurações localmente. Um particular nome de configuração pode ser atualizado pelo simples envio da mensagem de decisão para a mesma configuração.

Quando o PDP não desejar o uso do PEP, será enviada a mensagem de decisão especificando o nome da configuração e a decisão objeto *flags* com o comando de remover a configuração. O PEP deverá então proceder para remover a correspondente configuração e enviar uma mensagem reportando ao PDP a específica exclusão.

Nestes casos, o PEP pode notificar o PDP do estado local de uma instalação usando mensagem. A mensagem de informação é usada para indicar quando pode começar, quando ações foram ocupadas, monitoramento de estados e propósitos de contabilidade dependendo do cliente.

4.2.8 Segurança

O Protocolo COPS fornece integridade de objetos que podem executar autenticação, integridade da mensagem e prevenir repetições. Todas as implementações COPS devem suportar a integridade de objetos.

Para assegurar a comunicação entre PEP e o servidor PDP com a política correta, é necessário requer autenticação do PEP e PDP usando distribuições secretas e consistentes evidências sobre a validade da conexão. A distribuição secreta requer a configuração de chaves (identifica por uma chave ID) distribuídas entre o PEP e o PDP.

A chave é usada em conjunto com o conteúdo de uma mensagem COPS para calcular a mensagem que é parte da integridade do objeto. A integridade do objeto é então usada para validar todas as mensagens COPS enviadas sobre a conexão TCP entre o PEP e PDP.

A integridade de objetos COPS também prove seqüência numerosa para evitar ataques de repetição. O PDP escolhe a seqüência de números inicial para o PEP e o mesmo escolhe a seqüência de números inicial para o PDP. Estes números iniciais são então incrementados em cada mensagem sucessiva enviada sobre a conexão conforme direção. A seqüência inicial deve ser escolhida do mesmo modo que são aumentadas seqüencialmente e nunca repete para uma particular chave.

4. 3 *Web Services*

Os *web services* específica um grupo de padrões para a comunicação entre aplicações computacionais para a internet. Provendo uma estrutura capaz de resolver problemas de internet, fazendo uso de padrões abertos, amplamente conhecidos e de grande aceitação. Todos estes fatores possibilitam uma grande expansão dos *web services*.

A idéia de uma solução em sistemas distribuídos, capaz eliminar problemas de comunicação entre aplicações e redes distribuídas, através de um formato de informação previamente determinada são as funcionalidades do *web services*.

Os *Web services* facilitam a procura de informações na *web*, onde podemos fornecer apenas o nome do produto ou serviço e receber todas as informações necessárias dos mesmos, ao invés de visitar vários *sites* a procura. O *web services* se encarrega dessa tarefa sem precisar da interação humana. Para esta estrutura dar certo, cada componente ira receber suas informações, entender o tipo de informação recebida e como esta informação pode ser tratada. Para realizar essa tarefa os *web services* precisam fornecer além informações, informações sobre essas informações.

Isso é possível através do XML que é uma linguagem específica para manipulação dos dados. Então, encapsulado em um documento XML, além das informações enviadas, iram as informações da informação "os meta-dados". Em [Reynolds, 2001] define meta-dados como: "Dados dos dados, meta-dados, são uma das funcionalidades chaves que tornam possível à criação dos *web services*". As definições das funcionalidades do *web services* ficam armazenar em XML.

Todas estas características são altamente relevantes, porém se analisarmos o *web services* através da sua perspectiva inicial, a conseqüência é a firmação dos *web services* com tecnologia de integração em sistemas distribuídos.

As principais tecnologias proprietárias desenvolvidas para comunicação distribuída foram DCOM, CORBA e RMI. Como os mesmo princípios dos *web services*, também permitiam a comunicação entre aplicações de desenvolvidas em diferentes linguagens. No entanto, a pecavam por sua interoperabilidade. Os *web services* por usar

padrões abertos, possibilitam melhor interoperabilidade em relação às tecnologias proprietárias.

4.3.1 Estrutura *Web Services*

Em [Saganich 2001], o *web services* é uma tecnologia que poderá revolucionar a maneira como os serviços *business-to-business* e *business-to-consumer* podem ser fornecidos, pois usam uma variedade de tecnologias para permitir que duas aplicações comuniquem-se, sendo que, entretanto, nenhuma destas tecnologias seja considerada uma inovação. O que faz os *web services* diferentes de outros mecanismos similares são as tecnologias que fornecem o serviço.

Os padrões utilizados para o desenvolvimento de *web services* são basicamente as funções a seguir:

- Estrutura para procurar e gravar serviços;
- Mecanismo de transporte do serviço;
- Estrutura de parâmetros do serviço;

"Os *web services* têm em seu núcleo um mecanismo para comunicação e são baseados em três tecnologias específicas: mecanismo para registrar o serviço, mecanismo para encontrar um serviço e mecanismo para que duas partes comuniquem-se" [Saganich, 2001].

A combinação destas funcionalidades fornece a estrutura *web services* para ambientes distribuídos. Contudo, o fator decisivo para o *web services* ser atrativo é permitir a comunicação entre implementações de diferentes linguagens sendo executadas em plataformas distintas. Na figura 13, a estrutura de um *web service* e os principais componentes.

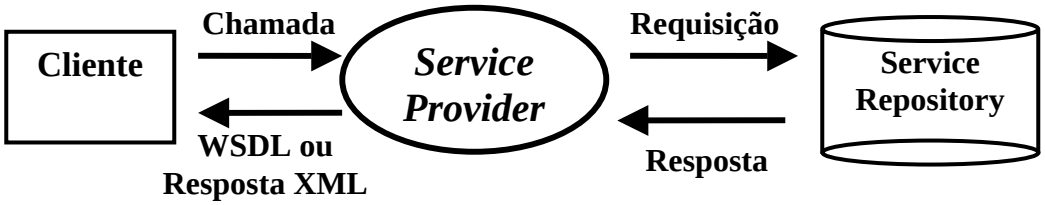

Figura 13: Estrutura *Web Services*.

- *Service provider* - O "fornecedor de serviço" desempenha papel fundamental na armação *web service,* onde abrigam diversos serviços os quais são exibidos como *web service;*
- *Service repository* - O repositório de serviços tem a função de armazenar as informações utilizadas pelos clientes na procura sobre serviços ou aplicações *web;*

O mecanismo responsável pela comunicação com o serviço é fundamental, onde é a ligação entre o cliente e as funcionalidades do serviço procuradas. Como mecanismos podem citar:

- *Web Service Description Language* – WSDL, que fornece uma técnica padrão para descrever propriedades e serviços *web* [Cristensen, 2001];
- *Universal Description and Integration* – UDDI: define regras para a construção de diretórios para serviços web [Box, 2000];
- *Simples Object Acess Protocol* – SOAP: protocolo para troca de informações entre serviços *web* fundamentado no padrão XML [Box, 2000].

A utilização de *web services* proporciona benefícios por utilizar padrões abertos de comunicação e aceitar componentes implementados em distintas linguagens. Outras vantagens são as integrações entre diferentes aplicações, redução de custo pela aplicação de protocolos de comunicação existentes.

A ampliação da utilização de *web services* proporciona uma nova forma de oferecer serviços e usa-los, onde implementações podem ser desenvolvidas em diferentes linguagens com padrões abertos. Por conseqüência a um menor custo de desenvolvimento e implantação.

4.4 XML (*Extensible Markup Language*)

O XML surgiu para adicionar funções às limitações existentes no HTML (*Hiper Text Markup Language*). O XML foi desenvolvido pelo W3C (*World Wide Web Consortium*) sendo uma linguagem de marcação que propõe flexibilidade e interoperabilidade entre as aplicações *web.*

A estrutura e o formato do XML e HTML são baseados em padrões SGML *(Sctructured General Markup Language),* os quais utilizam marcadores para demarcar o documento.

Com a evolução da internet, o HTML passou a ter papel destacado sendo utilizado por diversas aplicações em distintas áreas. Isto consolidou o HTML com a linguagem de marcação mais utilizada e abriu precedência para sua evolução. Entretanto, a necessidade por aplicações cada vez mais flexíveis foi indicando que o HTML esta chegando ao limite na estrutura de documento.

O padrão XML surgiu com a característica de representar o conteúdo do documento, deforma clara, com regras e validações bem definidas. Isto permite a separação do conteúdo do documento e qualquer apresentação visual. Como vantagem desta separação é o uso do XML em aplicações onde display visual não é importante, e o conteúdo do documento neste caso ser determinados puramente como dados. Com estas características, o XML se mostrou fácil para compreensão e um padrão para a transmissão dos dados.

O XML prove um simples caminho padronizado para delimitar e interpretar textos de dados. Através de uma coleção de regras para formação semântica *tags* que dividem um documento em partes identificando-as. Diferente da pré-determinação de coleção de regras de *tags* do HTML, XML é extensível; é uma linguagem *meta-markup* e o usuário pode definir *tags* necessárias.

Contudo, o HTML não atende as necessidades dos *web services* e aplicações diversas por extensibilidade e flexibilidade. Por conseqüência, vem sendo substituído pelo XML.

A W3C estabeleceu as fundamentais diretrizes do padrão XML:

- A internet com principal meio de emprego e aplicação;
- Documento XML com construções claras e legíveis;
- Desenvolvimento e interpretação fácil de programas em XML;
- Dar suporte a um grande número de aplicações;
- Compatibilidade com o SGML;
- Formalismo e consistência no projeto XML;

A W3C específica sistemas de definição para construção de XML: o DTD sugestão original e o XML *schema* para substituição do DTD. Há também linguagem de programação que trabalham com XML e são definidas pela W3C:

- XSLT: linguagem que tem por função principal a apresentação de documentos;
- Xquery: linguagem para manipulação de dados em XML, através de consulta semelhante ao SQL;
- Xpath: linguagem que permite acesso aos elementos e o endereçamento a partes do documento XML;

A utilização do XML pode ser evidenciada em aplicações de comercio eletrônico, *web services* entre outras. XML começa também a ser usado como formato de trocas de informações entre componentes e aplicações de larga escala.

4.5 *Simple Object Access Protocol* - SOAP

SOAP [Box, 2000] é um protocolo fundamentado na linguagem XML e concebido para troca de informações entre sistemas de forma a simplificar a comunicação em um ambiente distribuído.

O protocolo SOAP prove uma estrutura simples para expressar a o mecanismo de encapsulamento de informações e expressa uma semântica, a qual permite codificar os vários tipos de dados pertencentes aos pacotes. Há compatibilidade com grande variedade de sistemas enfatiza a utilização do SOAP para comunicação, independentemente da linguagem de programação empregadas.

4.5.1 Objetivos

Os objetivos primordiais do protocolo SOAP são possibilitar extensibilidade e maior simplicidade na comunicação entre diferentes linguagens. É um protocolo leve para troca de informação em um ambiente distribuído descentralizado como a internet.

O fato de o SOAP ser aceito no mercado recai sobre sua definição de padrões, por exemplo, a linguagem XML. A compatibilidade do SOAP com uma grande variedade de protocolos de comunicação é outro fato fundamental para sua aceitação, como exemplos de protocolos compatíveis têm SMTP, FTP e HTTP. O protocolo SOAP utilização o HTTP como transportador e encapsulando os dados em arquivos XML.

4.5.2 Funcionalidades

O SOAP é baseado em uma estrutura de informação textual transmitida pela internet. Esta estrutura textual é transformada no formato de arquivo XML, que possui um formato específico para processamento e transmissão. O protocolo padrão para transmissão de informações é o HTTP, o qual gerencia o tráfego por servidores *web* dos arquivos XML. Esta mistura da linguagem XML e do HTTP, conferem ao protocolo SOAP uma alta interoperabilidade.

A interoperabilidade nos serviços proporcionada pelo SOAP, é parte fundamental no processo de comunicação, onde o encapsulamento do arquivo XML utiliza o protocolo SOAP como base para a troca das mensagens. Isto implica que aplicações desenvolvidas em plataformas diferentes possam interagir sem a necessidade de software específico ou administradores de rede.

4.5.3 - Partes Fundamentais do SOAP

O protocolo SOAP baseasse em XML, contendo elementos de definição estrutural e elementos de definição de tipo de conteúdo.

O SOAP é um protocolo baseado em XML que consiste de três partes fundamentais: um “envelope” que detalha o conteúdo da mensagem e como processá-la; um conjunto de regras de codificação que têm a finalidade de expressar instâncias de tipos de dados definidos pela aplicação; e uma convenção para representarem chamadas e respostas de procedimentos remotos (RPC). SOAP pode ser utilizado em conjunto com uma grande variedade de protocolos.

O protocolo SOAP provê a definição de um documento XML, permitindo a comunicação entre dispositivos e objetos de qualquer linguagem e em qualquer plataforma. Uma mensagem SOAP é um documento XML que contém as seguintes partes conforme Figura 14 [W3C].

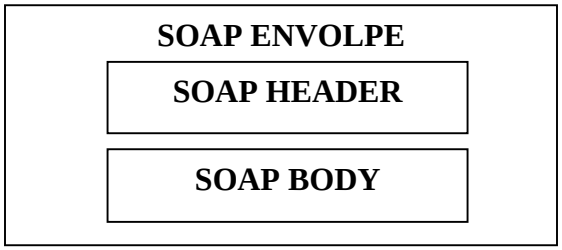

Figura 14: Formato de Mensagem SOAP.

- Envelope: bloco externo que representa a mensagem;
- Cabeçalho (*Header*): bloco genérico para adicionar funcionalidades a uma mensagem SOAP, define atributos e funcionalidade. Obrigatório ou opcional.

- O Corpo (*Body*): bloco para as informações a serem enviadas ao destinatário.

SOAP oferece um mecanismo simples e leve para trocas de informações estruturadas entre pares num ambiente distribuído descentralizado, usando XML. Como foi dito anteriormente, SOAP consiste de três partes [W3C]:

- Envelope – define um completo *framework* para expressar o quê está contido na mensagem, quem deve tratá-la;
- Regras de codificação – define o mecanismo de serialização que pode ser usado na troca de instâncias de tipos de dados definidos pela aplicação;
- Representação RPC – define uma convenção que pode ser utilizada para representarem chamadas e respostas de procedimentos remotos.

Embora estas partes juntas componham o SOAP, elas são funcionalmente ortogonais. Em particular, o envelope e as regras de codificação são definidos em *namespaces* (define as regras de codificação que o documento deve seguir) diferentes com o intuito de promover simplicidade através da modularidade.

4.5.3.1 - Atributo *EncodingStyle*

O atributo global *encodingStyle* serve para indicar as regras de serialização e desserialização usadas na mensagem SOAP. Este atributo pode aparecer em qualquer elemento e seu escopo abrange todo o conteúdo do elemento, inclusive os elementos filhos que não possuam seu próprio atributo *encodingStyle*, assim como o escopo de um *namespace*. Não há um *encodingStyle default* para as mensagens SOAP, é preciso declarar um [W3C].

4.5.3.2 - Cabeçalho (SOAP)

O “Cabeçalho” é um mecanismo oferecido pelo SOAP para estender uma mensagem de uma maneira modular e descentralizado sem necessitar de acordo prévio entre as partes. O elemento “*Header*” é definido como o primeiro elemento filho do

“Envelope”. Todos os elementos filhos do “*Header*” são chamados entradas do cabeçalho.

As regras de codificação das entradas do cabeçalho são as seguintes [W3C]:

1. Uma entrada do cabeçalho é identificada pelo nome completo do seu elemento, o qual consiste do URI do *namespace* e de um nome local. Elas devem ser ligadas a um *namespace*;
2. O atributo *encodingStyle* pode ser usado para indicar o estilo de codificação usado nas entradas do cabeçalho;
3. Os atributos SOAP *mustUnderstand* (define qual elemento do header deve aparecer para o receptor da mensagem) e *actor* (O atributo *actor* define a URI (equivalente a URL do http) à qual o *HEADER* se refere.) podem ser usados para indicar como e por quem processar a entrada;

4.5.3.3 Corpo da Mensagem. (SOAP)

O elemento “*Body*” representa a informação que se deseja enviar a uma aplicação destino. Toda a estrutura de SOAP trabalha para enviar de forma correta a informação contida no elemento “*Body*”; portanto o corpo da mensagem é a parte principal da mensagem SOAP. É nele que são transportadas as informações.

O uso mais comum do elemento SOAP é no transportar de parâmetros para as chamadas RPC e mensagens de erro. Os elementos filho de “*Body*” são chamados entradas do corpo da mensagem e são tratadas como elementos independentes.

As regras de codificação das entradas do corpo da mensagem são as seguintes [W3C]:

- Uma entrada do corpo da mensagem é identificada pelo seu nome completo, o qual consiste de uma URI de um *namespace* e de um nome local. Elas podem ser, ou não ligadas a um *namespace*;
- O atributo *encodingStyle* pode ser usado para definir o estilo de codificação usado nas entradas do corpo;
- Existe apenas uma entrada de corpo da mensagem predefinida, a entrada “*Fault*” usada para reportar erros.

4.5.3.4 *Fault*

O elemento "*Fault*" é usado para transportar informações de erro e/ou status. O elemento "*Fault*" deve aparecer como uma entrada do corpo. Apenas um elemento "Fault" deve estar presente no "*Body*".

O elemento "*Fault*" define quatro sub-elementos a seguir [W3C]:

- *faultcode* – usado para identificar o erro. Os tipos de erro predefinidos na especificação são: *VersionMismatch, MustUnderstand, Server* e *Client*;
- *faultstring* – é o texto que descreve o erro;
- *faultactor* – identifica a origem do erro.
- *detail* – contém informação de erro específica da aplicação relacionado ao processamento do elemento "*Body*". A ausência do elemento "*detail*" indica que o erro não está relacionado ao processamento do elemento "*Body*", isso serve para determinar se o corpo da mensagem foi processado ou não.

4.5.3.5 RCP utilizando SOAP

Um dos principais objetivos de SOAP são o encapsulamento e a troca de chamadas RPC inserindo extensibilidade e a flexibilidade do XML. O uso de SOAP para RPC é independente do protocolo de comunicação utilizado, no entanto a grande maioria das aplicações que implementam RPC sob SOAP usa o HTTP como protocolo de comunicação. Neste caso, a chamada RPC (*RPC call*) é diretamente mapeada para a requisição HTTP (*HTTP request*) e a resposta RPC (*RPC response*) para a resposta HTTP (*HTTP response*).

Para implantar a chamada de procedimentos remotos (RPC), as seguintes informações são necessárias [W3C]:

- O URI do objeto remoto;
- Nome do método;
- A assinatura do método (opcional);

- Os parâmetros do método;
- Dados de cabeçalho (opcional).

Mensagem SOAP empacotada numa requisição HTTP:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Body>
                <m:GetLastTradePrice xmlns:m="Some-URI">
                        <symbol>DIS</symbol>
                </m:GetLastTradePrice>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 15: Mensagem SOAP de requisição HTTP [W3C].

4.5.4 Complexidade e Interoperabilidade do SOAP

Para definir os prós e contras a de se analisar os propósitos a serem alcançados e fundamentalmente a natureza do sistema. A soma destes requisitos será determinante para aplicação do protocolo de comunicação.

As mensagens SOAP são utilizadas para encapsular os dados de uma aplicação no formato XML. O funcionamento dessas mensagens na rede leva em conta o meio de comunicação para definir o tamanho dos pacotes gerados. Nas aplicações que utilização HTTP o protocolo SOAP não degrada a performance de forma a comprometer o serviço. Entretanto em aplicações *wireless*, o tamanho das mensagens influencia diretamente no consumo da rede devendo ser cuidadosamente avaliado.

Quando ao empacotamento do SOAP, sua estrutura envolve uma maior complexidade devido a seus pacotes. No entanto o fato de ser totalmente XML permite incremento de bons mecanismos de empacotamento de arquivos XML, em contraponto ocasiona uma queda de performance em relação aos seus concorrentes.

O SOAP possui característica de facilitar a depuração de suas mensagens resultando em uma rápida correção de erros. Outro ponto de extrema importância é com relação à sua interoperabilidade, onde cada vez mais a segurança sobre os dados que trafega na rede e o que se filtrar também. De modo uma padronização SOAP [W3C] permite que os pacotes trafeguem normalmente pelos mecanismos de segurança com muita consistência. Em sistemas distribuídos a utilização do SOAP é bastante vantajosa.

4.5.5 Modelo Servidor e Cliente SOAP

O servidor SOAP trabalha com um interpretador e distribuidor de arquivos SOAP, é basicamente uma aplicação que escuta as requisições SOAP.

O cliente SOAP é uma aplicação que gera um arquivo XML contendo a informação precisa para requisitar remotamente um método em um sistema distribuído. Aplicações como servidor *web* e aplicações servidoras são exemplos de cliente SOAP. As mensagens SOAP feitas pelos clientes são transmitidas sobre HTTP. A Figura 16 mostra o funcionamento de uma mensagem XML sobre o SOAP

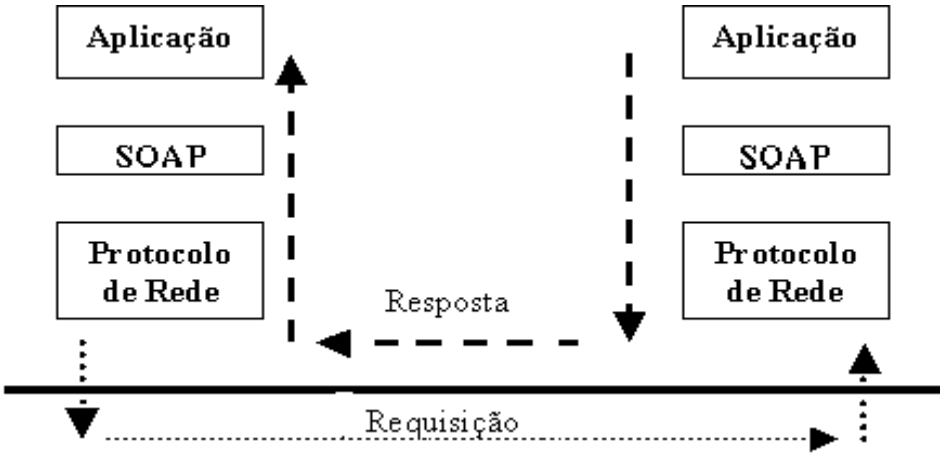

Figura 16: Mensagem XML através SOAP.

São quatro os passos básicos para o funcionamento e interação de uma aplicação SOAP.

1. O cliente solicita um serviço e a aplicação gera uma mensagem SOAP. Esta mensagem SOAP é a solicitação de disponibilidade de operação pelo serviço *web.* O protocolo SOAP utilizado para a comunicação será verificado pelo cliente para transporte ao destinatário.
2. A mensagem é enviada pela rede ao provedor de serviço. O servidor SOAP envia uma requisição para o provedor *web services*. Para a conversação da mensagem XML em comandos da linguagem a qual o foi implementado, o módulo SOAP invoca o *runtime* responsável por esta função.
3. O processamento da solicitação e o estabelecimento da resposta ficam a cargo do *web Service*.
4. No recebimento da resposta, a mensagem é processada e convertida na formatação adequada e apresentada a aplicação.

O modelo de requisição/resposta descrito permite demonstra o funcionamento do SOAP. Todavia este cenário pode ser estendido acrescentando vários receptores, onde é aplicada a técnicas de *multicast.*

4.6 Conclusão do Capítulo

Este capítulo foi sobre distribuição de política, onde foram apresentadas diversas tecnologias para difusão de políticas de gerenciamento de rede. Cada uma delas possui diferentes característica e operacionalidades, sendo essenciais seus conhecimentos para otimização da forma de distribuição de políticas de gerenciamento.

A Primeira abordada é a padronizada pelo IETF com protocolo de distribuição COPS. O entendimento do COPS foi de fundamental importância para concepção do protocolo proposto, onde a forma de descrição do protocolo permite a compreensão de sua estrutura, a comunicação entre o PDP e PEP, sincronização dos mesmos e demais operações de configurações. Quanto à segurança, a mecanismos para evitar envio de políticas a falsos dispositivos e garantir a autenticidade das entidades.

Em segui foi contextualizado o papel dos *web services* e de que forma seu desenvolvimento possibilita avanços no gerenciamento tradicional. Também a importância das especificações de padrões para a comunicação entre diferentes aplicações e para resolução de problemas de portabilidade. Para finalizar, o formato da mensagem é determinante para o sucesso e funcionalidades dos *web services*.

Na seção sobre XML são elucidadas as principais características com meio mais transparente, robusto e flexível de linguagem para a comunicação. Sua representação de conteúdo de forma clara com regra e validações são os pontos fortes. Em resumo, o XML se mostra um padrão para transmissão de dados amplamente utilizado pelo mercado e largamente contextualizado.

A seção sobre SOAP tem fundamental importância no encapsulamento das políticas, facilitando o tráfego na rede por suas características de portabilidade e adaptabilidade. Complementando, a descrição dos principais componentes, atributos e interoperabilidade permitem entender a grande aceitação do SOAP e larga utilização em ambientes distribuídos.

Por fim este capítulo permite entender as escolhas tecnológicas para criação do proposto protocolo de distribuição de políticas, pontos fortes e suas características como adaptabilidade, flexibilidade, portabilidade e capacidade de evolução e crescimento.

5. Trabalhos Correlatos

Neste capítulo é abortado em maiores detalhes a motivação para realização esta dissertação – seção 5.1. Na seção 5.2 são comentados sobre os trabalhos correlatos de maior significância para distribuição de políticas e demais correntes. Na seção 5.3 os benefícios e contribuições da distribuição políticas através SOAP/XML.

5.1 Motivação para Criação da Abordagem

A gerência de redes surgiu da necessidade de controlar os dispositivos e suas funcionalidades. O gerenciamento baseado em políticas pode ser a evolução do gerenciamento tradicional, permitindo maior eficiência no gerenciamento ao administrar dispositivo e recurso através de um conjunto de atributos e parâmetros específicos para cada participante.

O gerenciamento baseado em políticas através das políticas especifica a relação entre aplicação, recursos e usuários. Por exemplo, tipos de requisição, propriedades de segurança, largura de banda e outros. A definição das regras é determinada pelo gerente da rede, baseado nas atividades da empresa e na alocação de recursos pelos usuários da rede.

A definição de políticas segundo a *IEEE Network Magazine*: “as políticas podem ser usadas para se obter uma melhor escala no gerenciamento da rede através da descrição de atributos comuns de classes de objetos, tipicamente associados com o papel que representam, tais como dispositivos de rede, serviços de *software* e usuários, em vez de definir individualmente os atributos destes elementos”.

Neste contexto, a utilização do gerenciamento baseado em políticas minimiza a complexidade do gerenciamento e otimiza a administração dos dispositivos e recursos [RFC 3198]. As regras concebidas para o controle de recursos, permitem diferenciar aplicações e usuários classificando-os por grupo, realizando gerenciamento baseado nas políticas.

Outros benefícios da gerência por políticas estão em promover uma ampla facilidade rumo a escalabilidade e melhorar a adaptabilidade da rede nas mais variadas

condições. Favorece também, requisição multimídia com diferentes características de QoS e na complexidade das aplicações de tempo real.

É importante perceber o fato que pesquisas sobre sistema PBNM têm demonstrado muitos benefícios na gerencia de configuração e desempenho. A grande maioria dos sistemas comerciais disponíveis possui um controle de interface, que permite um administrador definir políticas as quais são estatisticamente configuradas nos dispositivos. Então, a construção de um complexo e útil sistema PBNM é totalmente desafiante e eminente. O sistema deve possuir escalabilidade, robustez e confiabilidade.

Em termos de escalabilidade, sistema PBNM deve ser capaz de suportar um grande número de dispositivos [RFC 3460], e potencialmente milhares de usuários e serviços e principalmente confiável para que políticas sejam aplicadas corretamente para usuários e aplicações. Entretanto, ainda não existe qualquer sistema comercial de PBNM capaz de oferecer interoperabilidade como modelo de gerenciamento por políticas [RFC 3460].

O fundamental sobre a compreensão e abrangência de soluções PBNM é que complementam a gerência tradicional, acrescentando o conceito de políticas definidas pelo administrador. Apesar disso, a existência de pré-requisitos para seu funcionamento:

- A rede gerenciada deve ser completamente conhecida pelo seu administrador;
- PBNM deve-se moldar às características da arquitetura existente;
- Os pontos de atuação e decisão de políticas (PEPs e PDPs) já devem estar definidos;
- Quando implantada a política, se o comportamento não for esperado o PBNM não é responsável por reportar ao gerente sobre a condição;

Atualmente a comunidade enfrenta diversos desafios na utilização do PBNM, onde entre estas estão à padronização da arquitetura de gerenciamento, protocolos de distribuição de políticas, armazenamento das políticas e os métodos utilizados para a construção da regras das políticas. Nesta dissertação, o intuito é a distribuição de políticas utilizando XML / SOAP, onde as funcionalidades destas tecnologias possibilitam “distribuição” ampla, flexível.

5.2 Trabalhos Correlatos

Nesta seção, a maiores detalhes com relação aos trabalhos sobre distribuição de políticas de gerenciamento. Estes trabalhos permitiram o entendimento conceitual e na concepção da proposta.

O trabalho de Natarajan et al. propõe devido à alta heterogeneidade de sistemas distribuídos uma arquitetura implementada em XML para distribuir políticas. Este abordagem para distribuição de políticas contempla:

- Eventos que reportam falhas que requeiram ações corretivas;
- Eventos relativos à segurança do sistema;
- Eventos que realizam testes de desempenho e debugging a aplicação;
- Eventos que coletam e reportam rotinas de uso de dados específicos;

Em comparação com o trabalho realizado e dando ênfase na qualidade de serviços, a abordagem criada observa os seguintes eventos:

- Requerer ações corretivas para dispositivos;
- Reportar mudanças nas configurações; e
- Registrar os recursos gerenciados.

Os diferentes eventos são relativos aos distintos propostos, entretanto convergem no sentido de requisitar políticas corretivas de um repositório.

Na definição dos campos da mensagem XML Natarajan et al., concebe a idéia de dividir a mensagem em duas partes principais: Cabeçalho e Corpo. Esta divisão permite direcionar as partes de identificação e políticas da mensagem. Abordagem adotada nesta dissertação para criação dos campos da mensagem.

As diferenças com relação aos campos do cabeçalho da mensagem, estão na adição dos campos:

- Tipo de mensagem;
- ReqID;
- SNY;

No corpo da mensagem, partindo que as informações do cabeçalho são válidas, os campos foram criados para simplificar e otimizar a resposta contendo a política corretiva. Em relação a Natarajan et al. os campos do corpo da mensagem de resposta, são definidos de acordo com os eventos, contudo alguns campos foram incorporados nesta trabalho visado contemplar aspectos antes não previstos.

Complementando, arquitetura de Natarajan et al. identifica o tipo de ação e o serviço de informação envia as novas políticas em um formato de mensagem XML. Com isto, implementa políticas de gerenciamento em todos os níveis de gerencia.

A diferença mais significante entre as obras de Natarajan et al. e esta dissertação repousam na utilização do protocolo SOAP. Natarajan et al. concebe sua idéia utilizando XML como forma de distribuição das políticas. Entretanto a como a utilização do SOAP acrescentamos todas as funcionalidades deste protocolo descritas no capítulo 4.5 e implementadas no capítulo 6.

Phanse et. al [Phanse, 2004], propõe a extensão do gerenciamento baseado em políticas para configuração de Manets. Para isto, estende o protocolo COPS combinado técnicas *outsourcing* e *provinsioning* para o desenvolvimento de sistema PBNM para Manet. Os serviços estendidos são mecanismos para reconhecimento automático dos nós e negociação entre domínios distintos. Segundo os autores, eficiência, confiabilidade e robustez são os pontos fortes para a interoperabilidade entre sistemas de redes ad hoc e a integração de mecanismos padronizados PBNM. Como conclusão o protocolo prove mobilidade em redes ad hoc, limita o tráfego redundante e reconhecimento automático de domínios.

Law K. l. et al [Law, 2003], propõe uma extensão do framework de políticas inserindo agentes entre o ponto de decisão e o ponto de aplicação. Esta arquitetura foi chamada de *Unified Policy-Based Management - UPM*, tendo por objetivo o melhor controle dos dispositivos gerenciados e estender a aplicação do protocolo de distribuição. Onde os agentes monitoram os dispositivos e enviam mensagens sobre as informações

ocorrentes ao ponto de decisão (via COPS). Como resultado, foi comparado a performance do UPM ao framework de políticas da IETF.

Danciu et. al [Danciu, 2004], neste trabalho os autores propõem uma linguagem para definição de políticas denominada ProPolis, onde há um modelo de geração automática de políticas para processos específicos. As ligações entre os processos específicos e as políticas utilizam XML. Para executar a tradução entre o processo e a política, segundo os autores, é alcançada uma característica comum entre o processo e a política, a definição de um modelo padrão de tradução. O uso deste modelo permite tradução automática de processos orientados definidos pela política. Contribuição, na definição de ferramenta para tradução de processos específicos e linguagem de definição de políticas para sistemas de larga escala em ambientes heterogêneos.

Vaguetti et al. [Vaguetti, 2003], neste trabalho foi proposto à extensão da padronização do framework de políticas, incluindo elementos capazes de prover facilidades no monitoramento de QoS em um ambiente integrado baseado em políticas. Estes monitores de QoS podem ser vistos como pontos de decisão (PDP) no processo de monitoração dos pontos de aplicação (PEP).

Lymberopoulos et. al [Lymberopoulos, 2003], apresenta um framework especificamente para o gerenciamento de redes baseado em políticas, o qual suporta automaticamente desenvolvimento de políticas, permite flexibilidade na configuração dinâmica e insere conceitos de acordo de nível de especificação para garantir parâmetros da política. O enfoque principal do trabalho está na adaptação dinâmica de política, na resposta as transformações de parâmetros e reconfiguração de política de objetos. Conclusões, através do framework e das políticas de adaptação são possíveis a transformação dinamicamente de parâmetros de QoS, especificação de novos atributos e atualização de valores em tempo de execução.

Shepard [Shepard, 2000] realiza uma pesquisa sobre os principais aspectos da arquitetura de gerenciamento baseado em políticas. Analisando sua perspectiva de desenvolvimento sobre requisitos de qualidade de serviço, segurança e outras capacidades auxiliando no gerenciamento de redes.

Jha et al. [Jha, 2003], apresentou uma implementação em Java do gerenciamento de largura de banda por políticas. No trabalho é utilizado negociador de banda (*Bandwidth Broker*) sobre o protocolo padronizado para comunicação com remotos *bandwidth brokers*, onde a negociação do nível de serviço para determinar as condições e configurações da rede.

Na proposta de Devarakonda et al. [Devarakonda, 2002] foi estendido o framework com relação ao repositório de políticas, segundo os autores as alterações visam suavizar custos e a complexidade do armazenamento das políticas. No framework o repositório é gerenciado pela ferramenta de aplicação e classificado em níveis de armazenamento.

Já em Flesh et al. [Flesh, 2002], sugere uma arquitetura para integrar funcionalidades da gerência dinâmica e gerenciamento baseado em políticas. Para isso, constrói agentes com funcionalidades em tempo de execução e políticas pré-determinadas. Segundo os autores os principais objetivos são flexibilizar e automatizar a aplicação do gerenciamento.

5.3 Benefícios e Contribuições da Distribuição de Políticas através SOAP/XML

A utilização de políticas de gerenciamento de redes permite ao administrador especificar como a rede será configurada e monitorada. Desde modo é possível reduzir a complexidade do gerenciamento uma vez que as políticas podem ser reutilizadas. O uso de políticas também pode permite abstração de forma que especificações podem ser simplificadas e entendidas.

A infra-estrutura pode permitir a introdução de novos serviços e adaptar suas redes rapidamente com SOAP/XML. Permite que o comportamento do sistema de gerenciamento e seus processos mudem requisitos conforme o tempo e políticas.

A Distribuição SOAP/XML permite a automatização de várias tarefas no gerenciamento via HTTP, de acordo com o com as políticas preparadas. Isto ajuda na execução de grandes números de tarefas. Também inserindo capacidade no sistema de gerenciamento de adaptar seu comportamento dinamicamente para diferentes condições do ambientes através de políticas. Outro ponto de influencia é na capacidade de

gerenciamento e configuração à distância. Além disso, a política do sistema pode ficar *online* durante a mudança da política, melhorando disponibilidade e adaptabilidade da rede.

A utilização do XML e SOAP aumenta potencialmente a escalabilidade e reduz os custos, uma vez que se pode gerenciar de forma descentralizada. A segurança através do uso de políticas de autorização e especificações explícitas de segurança nos componentes da rede é possível melhor à segurança. Fundamentalmente importante na distribuição SOAP/XML é a capacidade de interação e cooperação com outras soluções de gerenciamento integrando infra-estruturas existentes com novas tecnologias.

Por final temos uma abordagem que permite uma gerência mais flexível, distribuída e facilmente adaptável às novas características na rede. Complementando, a criação de uma estrutura de mensagens em padrão XML e encapsulado no SOAP permitem distribuição de políticas em diferentes e distribuídos domínios administrativos.

Contribuições finais estão para inserir padrões XML/SOAP em recurso e aplicações no gerenciamento baseado em políticas. Na definição contextualização de uma abordagem para distribuição de políticas baseado no protocolo COPS e na contribuição para desenvolvimento e aperfeiçoamento na distribuição de políticas de gerenciamento para redes.

6. Distribuição de Políticas de Gerenciamento de Redes com Ênfase em Qualidade de Serviço

Este capítulo apresenta vários aspectos da distribuição estendida de políticas de gerenciamento. Na seção 6.1 discorre sobre o protocolo de distribuição padronizado pela IETF, principais soluções do mercado, descrição do cenário da proposta, papel do SOAP e os motivos da utilização do XML como forma de distribuir as políticas de gerenciamento. Em aspectos da implementação seção 6.2, mostra a definição funcional de cada componente das mensagens de requisição/resposta, seus formatos DTDs. A seção 6.3 descreve o formato e o conteúdo das regras das políticas fundamentais no processo de gerenciamento por políticas. E por final a seção 6.4, onde são explicadas as abstrações necessárias, ferramentas utilizadas, a forma utilizada para distribuição e de mais aspecto da implementação de distribuição da política de gerenciamento.

6.1 Descrição Funcional

A proposta para distribuição de políticas de gerenciamento através do SOAP/XML, visa acrescentar funcionalidades ao framework da IETF, possibilitando gerenciamento distribuído, extensibilidade do modelo, flexibilidade no gerenciamento, portabilidade e por fim a capacidade de evolução e crescimento.

A distribuição de política de gerenciamento através do protocolo COPS padronizado IETF, prevê ao menos um ponto de decisão - PDP existente em cada domínio administrativo, estabelecendo o transporte através do protocolo TCP e suportando diversos clientes específicos. Quanto ao repositório, as recomendações são CIM ou PIB e para o protocolo de acesso ao repositório LDAP.

As soluções do mercado para política de gerenciamento não seguem as recomendações estabelecidas pela IETF e DMTF, entretanto estas têm grande aceitação no mercado. Como exemplo, o *PolicyXpert* [HP, 2001] da HP e o QPM (*QoS Policy Manager*) [Cisco, 2000] da Cisco. Porém, todas estas soluções possuem deficiência de integração.

O cenário da proposta se concentra entre o ponto de decisão da política - PDP e o ponto de aplicação da política - PEP, onde há distribuição de políticas para os dispositivos. Esta distribuição de políticas é inspirada nos fundamentos do COPS, mas implementado em uma linguagem de marcação desenvolvida pela W3C o XML.

Para a escolha do protocolo de distribuição de políticas devem ser considerados fatores de extrema importância:

- Todos os PDPs do domínio devem suportar o protocolo adotado para distribuição; e
- Para diferentes domínios a ferramenta de gerenciamento de políticas precisa traduzir o protocolo utilizado.

A idéia de aplicar o XML como meio distribuidor de políticas de gerenciamento surgiu dele proporcionar suporte a integração de grande variedade de aplicações *web*, sobrepondo limitações do HTML [W3C, 2004]. A linguagem de marcação mais popular é o HTML amplamente utilizado por milhares de aplicações de vários níveis em sistemas computacionais, por exemplo, *browsers*, software de comunicação e outros.

A utilização do XML para difusão das políticas permite maior flexibilidade por ser um padrão W3C. Sua grande aceitação advém de sua portabilidade, HTTP como meio de comunicação e extensibilidade. Para possibilitar a troca de informações de gerenciamento de forma estruturada entre o PEP e o PDP, é necessária a definição do documento XML. O protocolo SOAP provê a definição de um documento XML, permitindo a comunicação entre dispositivos e objetos de qualquer linguagem e em qualquer plataforma [Box, 2001].

Por fim, definida a estrutura necessária para comunicação entre PEP e PDP, onde o conjunto de regras das políticas de gerenciamento é traduzido para o XML encapsulado no protocolo SOAP e transmitidas sobre o HTTP.

6.2 Aspectos de Implementação

Para a concepção dos campos da mensagem foi realizada uma análise dos principais requisitos necessários à transmissão. Também considerado as estrutura de mensagem COPS e de que forma pode ser mais bem utilizada sobre o SOAP/XML.

Como resultado obteve-se os requisitos de performance, eficiência, funcionalidade, flexibilidade e alcance no ambiente gerenciado.

A proposta se concentra nas seguintes ações:

- Requerer ações corretivas para dispositivos;
- Reportar mudanças nas configurações; e
- Registrar os recursos gerenciados.

Na construção dos DTDs (*Document Type Definitions)* foram definidos os tipos de elementos e atributos necessários para o documento XML válido, bem como a ordem necessária para o funcionamento. A seguir serão apresentados os componentes: cabeçalho e corpo da mensagem.

Como relação a artigos sobre o assunto, em [Natarajan, 2001] é concebido um modelo em XML para distribuição de políticas. Entretanto este modelo não considera aspectos relevantes como: campos de sincronização das mensagens, identificação da requisição, tipo de mensagem (requisição/atualização) e campos de controle para melhor gerenciamento. Contudo, um dos aspectos de maior contribuição desta proposta é a utilização do protocolo SOAP para o encapsulamento das informações No artigo [Andrade, 2004] deste autor, também pode ser encontrado informações sobre gerenciamento baseado em políticas.

6.2.1 Componentes do Cabeçalho

Descrição dos componentes formadores do cabeçalho da mensagem de Requisição/Atualização. Na Figura 17 temos a representação gráfica.

- Campo Origem: máquina que gerou a solicitação ou atualização ao ponto de decisão PDP;

- Campo Destino: responsável pelo endereço da PDP, podendo ser um endereço IP ou URL;

- Campo Tipo de Mensagem: identifica qual a finalidade da mensagem. No modelo criado existem dois tipos:

  - Mensagem de Atualização: O PEP informa a situação do dispositivo ao

PDP;

- Mensagem de Requisição: solicitação de uma política corretiva ao PDP para estabelecer o funcionamento pré-determinado pelo administrador;

- Campo Tempo: indica o instante de tempo (dd/mm/aa) e (hh/mm/ss) do envio da mensagem ao PDP. Possibilita um controle mais efetivo;

- Campo ReqID: responsável por identificar a solicitação para o PDP;

- Campo SYN (sincronização): define um número de seqüência das mensagens para controle das políticas pelo PDP.

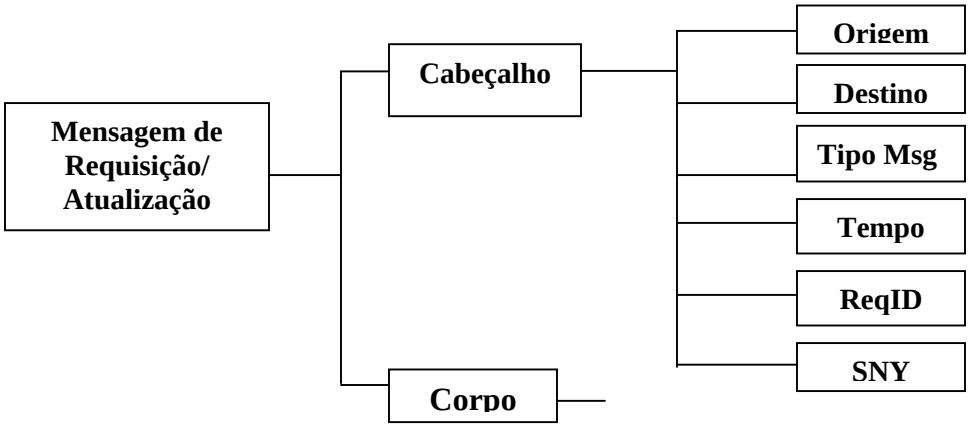

Figura 17: Componentes do Cabeçalho da Mensagem.

6.2.2 Componentes do Corpo

Descrição dos componentes formadores do corpo da mensagem de Requisição/Atualização e representação gráfica na Figura 18.

- Campo Tipo de Mensagem: classifica os acontecimentos no dispositivo PEP, onde pode haver grande variedade de ações referentes ao gerenciamento;
- Campo Prioridade: define qual mensagem terá preferência no envio pelo PDP;
- Campo Tempo: tempo da detecção da degradação da política no dispositivo;
- Campo Parâmetro: Dividido em duas partes;
    - Campo ParamentroID: são os parâmetros solicitados pela mensagem de requisição para envio de uma política de correção;
    - Campo Valor: valores definidos pela política para ajustar o dispositivo.

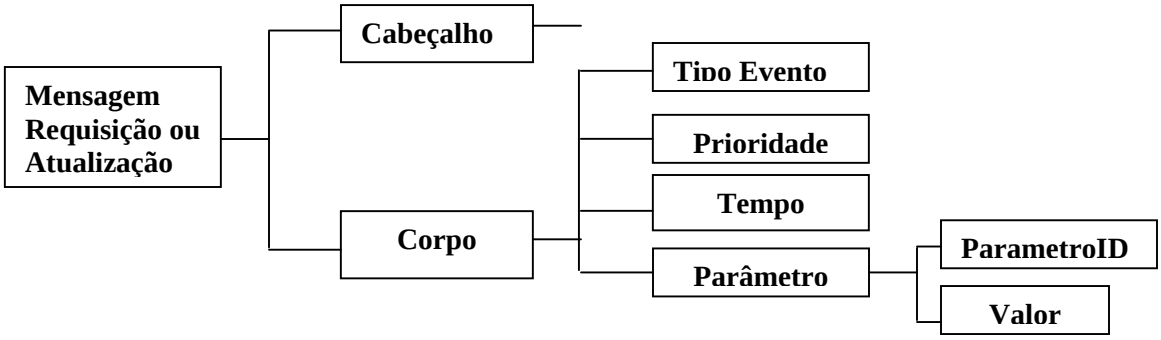

Figura 18: Componentes do Corpo da Mensagem.

Para enviar as políticas para o PEP, o PDP utiliza os dados de identificação do cabeçalho vindo na mensagem de requisição. Entretanto há alterações no corpo da mensagem para o envio da política:

- Campo Política: especifica a política para o dispositivo;
- Campo Parâmetro: idem ao campo acima, mas com as políticas corretivas.

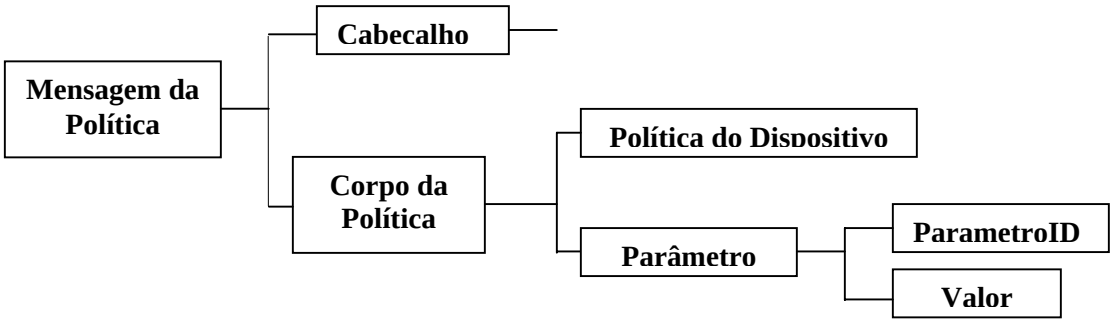

Figura 19: Componentes do Corpo da Política.

6.2.3 Formato da Mensagem de Requisição e Resposta

Na Figura 20 temos a mensagem de requisição:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Request (Cabecalho?, Corpo?)>
<!ELEMENT Cabecalho (#PCDATA)>
<!ATTLIST Cabecalho
        Origem CDATA #REQUIRED
        Destino CDATA #REQUIRED
        TipoMsg CDATA #REQUIRED
        Tempo CDATA #REQUIRED
```

```
        ReqID CDATA #REQUIRED
>
<!ELEMENT Corpo (Evento+)>
<!ELEMENT Evento (Parametro+)>
<!ATTLIST Evento
        TipoEvento CDATA #REQUIRED
        Prioridade CDATA #REQUIRED
        Tempo CDATA #REQUIRED
<!ELEMENT Parametro (#PCDATA)>
<!ATTLIST Parametro
        ParametroID CDATA #REQUIRED
        Valor CDATA #REQUIRED>
```

Figura 20: Mensagem de Requisição.

Na Figura 21 a mensagem de Resposta:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Response (Cabecalho?, Corpo?)>
<!ELEMENT Cabecalho (#PCDATA)>
<!ATTLIST Cabecalho
        Origem CDATA #REQUIRED
        Destino CDATA #REQUIRED
        TiposMsg CDATA #REQUIRED
        Tempo CDATA #REQUIRED
        ReqID CDATA #REQUIRED
>
<!ELEMENT Corpo (Politica+)>
<!ELEMENT Politica (#PCDATA)>
<!ATTLIST Politica
        NomePolitica CDATA #REQUIRED
>
<!ELEMENT PoliticaDados (#PCDATA)>
<!ATTLIST PoliticaDados
        ParametroID CDATA #REQUIRED
        Valor CDATA #REQUIRED >
```

Figura 21: Mensagem de Resposta.

6.3 Geração de Regras/Políticas

No gerenciamento de redes de computadores existem diferentes características peculiares a cada ambiente. Para construção de políticas eficientes é fundamental entender todos os requisitos necessários às aplicações e ao tráfego de dados na rede.

As políticas de gerenciamento trabalham na perspectiva de otimizar o funcionamento do gerenciamento, eficiência do administrador, e possibilitar a integração

do gerenciamento entre diferentes domínios administrativos. Com relação a políticas de gerenciamento entre diferentes domínios administrativos, há grande empenho das empresas na integração de soluções PBNM e também por parte da comunidade científica.

As políticas são construídas a partir das características de cada rede gerenciada. Os exemplos abaixo, mostram a estrutura da geração de políticas e sua aplicabilidade.

Neste exemplo de gerenciamento de desempenho foi determinado que as máquinas com os números IP (Figura 22), terão sua Largura de Banda. Com as regras desta política, as máquinas trabalharão com 60 MHz, proporcionando maior largura de banda para outras aplicações pertencentes à rede.

Política: Gerenciamento de Desempenho
Regra: Restringir largura de banda para outras aplicações.
Se (IP = 150.162.6.125) **e** (IP = 150.162.6.132)
e (IP = 150.162.8.166) - Máquinas da Rede
então
Largura de Banda = 60 MHz

Figura 22: Políticas de desempenho.

Para gerenciamento de configuração segue o exemplo de videoconferência (Figura 23). As regras da política especificam que entre 8 e 11 horas da manhã todas as solicitações para o IP 150.162.6.125 (Servidor da videoconferência) terão velocidade de transmissão de 200 Kbps, taxa de perda de pacotes de 10% e *delay* de 100ms. Com estas regras se pretende estabelecer uma comunicação melhor e restringir outras aplicações que consomem recursos do sistema.

Política: Videoconferência
Regra: Otimizar a comunicação para a videoconferência
Se (Horario >= 8am) **e** (Horario >= 11am) **e**
(IPDestino = 150.162.6.125) – Servidor de Videoconferência
então
Largura de Banda = 200 Kbps
Perda de Pacotes = 10%
Delay = 100 ms

Figura 23: Políticas de videoconferência.

Para o gerenciamento de segurança (Figura 24) o exemplo trata de acesso remoto via *telnet*. As regras desta política especificam que somente os dispositivos que tiverem os números IP abaixo poderão ter acesso remoto a porta 23 da máquina destino.

Política: Gerenciamento de Segurança
Regra: Determina quais máquinas podem fazer acesso Telnet.

Se (IP = 150.162.6.125) **e** (IP = 150.162.6.125) **e** (IP = 150.162.6.125) **e** (IP = 150.162.6.132) **então**

Porta 23 = Open (Aberta)

Figura 24: Políticas de segurança.

6.4 Distribuição e Implementação das Políticas de Gerenciamento

Para construção do protocolo para distribuição de políticas de gerenciamento, é necessária a abstração de alguns pontos do extenso framework da IETF. Com isto se pode limitar melhor o foco do trabalho. Como mencionado na descrição funcional, a área de concentração do trabalho está entre o PDP e o PEP. Onde se desenvolve toda a troca das informações e o uso efetivo da distribuição de políticas sugerido.

A proposta deste protocolo destaca–se para a distribuição de políticas "*Pull*" e para a configuração de dispositivos "*outsourcing*". A distribuição de política "*Pull*" prevê que a ferramenta de gerenciamento informe ao PDP a localização da política, e a configuração do dispositivo "*outsourcing*" define a requisição da política a partir do PEP para a PDP (Seção 2.3.3.1).

As abstrações mencionadas são: a ferramenta de gerenciamento e a monitoração dos dados dos dispositivos. A ferramenta de gerenciamento foi abstraída pelo fato do objetivo ser a distribuição de políticas de gerenciamento, onde se concentrou as funções de ferramenta de gerência e repositório de políticas no PDP. Foi adotada essa postura para dar maior ênfase na extensão proposta e diminuir muito da complexidade do framework da IETF. Completando, a monitoração pelo fato de existir vários mecanismos já amplamente utilizados e contextualizados em diversos trabalhos. Contudo podemos

citar o SNMP e o WMI e também a construção de agentes para este fim.

Os *web services* foram criadas para possibilitar automatizações de serviços através de aplicação na internet, utilizado chamadas de métodos em XML. A padronização da comunicação para *web service* permite independência de linguagens de programação e de plataformas. Segundo a W3C, a utilização do SOAP é fundamental para a comunicação entre *web services*, sendo responsável pela portabilidade.

Na Figura 25 temos a representa das trocas de mensagem entre a aplicação cliente e o *web service.*

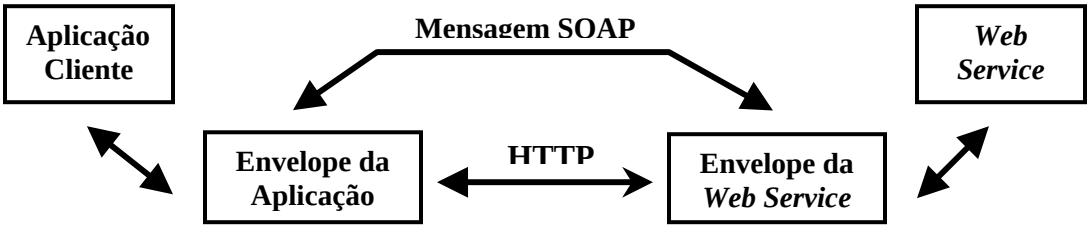

Figura 25: Comunicação *Web Service.*

Para o funcionamento do *web service* é imprescindível sua definição, como deve ser acessado e que valores retornam. Para estas definições a um arquivo em XML de acordo com a padronização WSDL, responsável pelo entendimento do serviço para que possa ser utilizado.

Os *web services* são disponíveis em servidores para que possam responder as requisições sempre que efetuadas. Nesta implementação será utilizado o servidor Tomcat, ficando ativo na porta 8080.

O Tomcat [Tomcat, 2004] é um container, servidor onde os servlets e JSP são instalados para tratar as requisições. Há uma grande variedade de container, entretanto o Tomcat por ser gratuito e bastante popular fui escolhido para implementação. A versão do Tomcat utilizado: Apache Tomcat 5.0.28.

Para implementação do protocolo SOAP, foi utilizado o Axis. O Axis é um *framework* para construção de componentes que trabalham com SOAP, como clientes, servidores, gateways e outros. Suas principais funções são [Axis, 2004]:

- Na implementação de Classes para dinamizar a comunicação e a publicação de web services;
- Na disponibilidade de web services utilizado containers JSP;

A versão do Axis é escrita em Java e permite criar [Axis, 2004]:

- Servidor stand-alone;
- Servidor no qual a plugs em servlet;
- Suporta extensão para WSDL;
- Emitir instruções que geram classes java para WSDL;
- Ferramenta de monitoração de pacotes TCP/IP;

Uma vez tendo o servidor Tomcat operacional, falta acrescentar o Axis para que as mensagens agreguem as funções do protocolo SOAP. Para isto deve-se acrescentar no diretório do tomcat o conjunto de ferramentas Axis em c:\tomcat\webapps .

Para tornar o serviços web disponível é necessário renome ar QoSWS.java para QoSWS.jws e coloca-lo em c:\tomcat\webapps\axis. Os arquivos com extensão jws são lidos pelo Axis e representam o java web service. O Axis baseia-se neste arquivo para definir o WSDL.

Também inserido no Tomcat c:\tomcat\webapps\axis\web-inf\classes os arquivos da pasta bin da aplicação java, com a função de suportar o protocolo SOAP nas trocas de mensagens. Na figura 26 temos a visualização do WSDL utilizado nesta implementação.

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://127.0.0.1:8080/axis/QoSWS.jws"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://127.0.0.1:8080/axis/QoSWS.jws"
xmlns:intf="http://127.0.0.1:8080/axis/QoSWS.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://mensagens" xmlns:tns2="http://www.w3.org/1999/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">- <wsdl:types>
- <schema targetNamespace="http://mensagens"
xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://127.0.0.1:8080/axis/QoSWS.jws" />
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
```

```
- <complexType name="Cabecalho">
- <sequence>
  <element name="destino" nillable="true" type="soapenc:string" />
  <element name="envio" nillable="true" type="xsd:dateTime" />
  <element name="origem" nillable="true" type="soapenc:string" />
  <element name="reqid" nillable="true" type="soapenc:long" />
  <element name="sny" nillable="true" type="soapenc:long" />
  <element name="tipoMensagem" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
- <complexType name="CorpoRequisicao">
- <sequence>
  <element name="parametros" nillable="true" type="impl:ArrayOf_tns2_anyType" />
  <element name="prioridade" nillable="true" type="soapenc:long" />
  <element name="tempo" nillable="true" type="xsd:dateTime" />
  <element name="tipoEvento" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
- <complexType name="EnvelopeRequisicao">
- <sequence>
  <element name="cabecalho" nillable="true" type="tns1:Cabecalho" />
  <element name="corpo" nillable="true" type="tns1:CorpoRequisicao" />
  </sequence>
  </complexType>
- <complexType name="CorpoPolitica">
- <sequence>
  <element name="parametros" nillable="true" type="impl:ArrayOf_tns2_anyType" />
  <element name="politica" nillable="true" type="soapenc:string" />
  </sequence>
  </complexType>
- <complexType name="EnvelopePolitica">
- <sequence>
  <element name="cabecalho" nillable="true" type="tns1:Cabecalho" />
  <element name="corpo" nillable="true" type="tns1:CorpoPolitica" />
  </sequence>
  </complexType>
  </schema>
- <schema targetNamespace="http://127.0.0.1:8080/axis/QoSWS.jws"
xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://mensagens" />
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
- <complexType name="ArrayOf_tns2_anyType">
- <complexContent>
- <restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="tns2:anyType[]" />
  </restriction>
  </complexContent>
  </complexType>
  </schema>
  </wsdl:types>
- <wsdl:message name="buscaPoliticaResponse">
```

```
  <wsdl:part name="buscaPoliticaReturn" type="tns1:EnvelopePolitica" />
  </wsdl:message>
- <wsdl:message name="buscaPoliticaRequest">
  <wsdl:part name="envelope" type="tns1:EnvelopeRequisicao" />
  </wsdl:message>
- <wsdl:portType name="QoSWS">
- <wsdl:operation name="buscaPolitica" parameterOrder="envelope">
  <wsdl:input message="impl:buscaPoliticaRequest" name="buscaPoliticaRequest"
/>
  <wsdl:output message="impl:buscaPoliticaResponse"
name="buscaPoliticaResponse" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="QoSWSSoapBinding" type="impl:QoSWS">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="buscaPolitica">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="buscaPoliticaRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
  </wsdl:input>
- <wsdl:output name="buscaPoliticaResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/axis/QoSWS.jws" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="QoSWSService">
- <wsdl:port binding="impl:QoSWSSoapBinding" name="QoSWS">
  <wsdlsoap:address location="http://127.0.0.1:8080/axis/QoSWS.jws" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

Figura 26: Arquivo WSDL do QoSWS.jws

A geração do WSDL é fundamentalmente importante para a implementação do SOAP, o Axis é um framework que torna este processo transparente, com conseqüência sua recomendação para construção de web services.

Para o desenvolvimento da aplicação em java foi utilizado o Eclipse [Eclipse, 2004]. O eclipse é uma plataforma que foi projetada para fundamentar a integração de ambientes web e o desenvolvimento de aplicações provendo um modelo de interface amigável e eficiente para desenvolvimento. Na Figura 27 mostra a interface no eclipse.

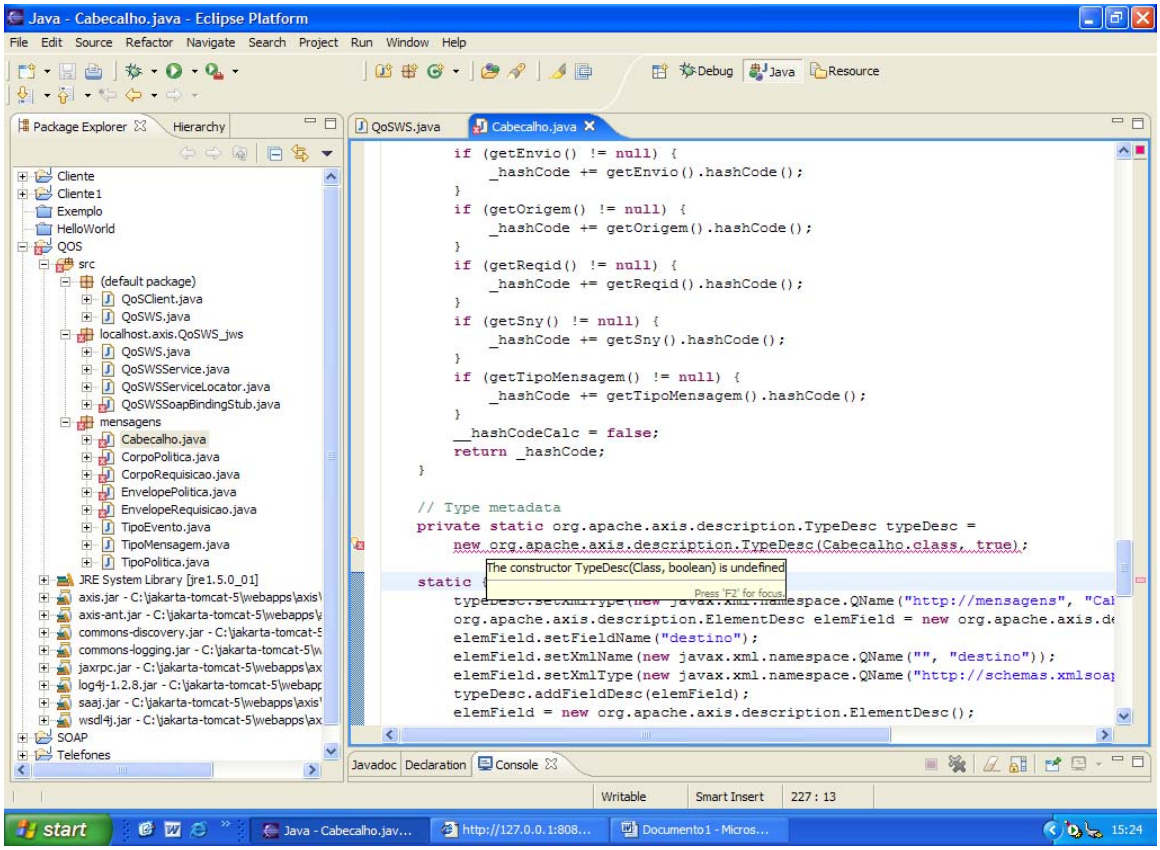

Figura 27: Ambiente de desenvolvimento Eclipse.

O cenário criado para esta implementação utiliza política de desempenho. O dispositivo constata uma condição não prevista pelo gerenciamento. O PEP cria um documento XML contendo as informações atuais, requerendo uma política corretiva. A mensagem é encapsulada utilizando o SOAP e transmitida sobre o HTTP pelo requisitante.

Na Figura 28 temos a demonstração do layout desenvolvido, ao lado esquerdo temos a aplicação no PEP como seus respectivos campos. Na direita temos o retorno do web service com a política pré-definida, embora as interfaces de aplicação apareçam juntas elas executam funções distintas. Através do Tomcat é possível simular a requisição e resposta com se estivesse em máquinas diferentes, na aplicação ao enviar a requisição os valores são enviados para o web service pela por 8080 e seu retorna também.

Nos campos temos informações da máquina gerenciada “Sirius”, que requisita a política corretivas ao PDP “Pegasus”. Nos demais campos do cabeçalho têm informações de tempo, reqid e sny. Todas possibilitam maior controle das mensagens. Nos campos de

corpo da mensagem temos os campos responsáveis pelo envio das condições do dispositivo.

Figura 28: Disposição da aplicação requisição.

O PDP é responsável pelo processamento da requisição e por gerar uma resposta. A resposta também será encapsulada pelo SOAP e utilizará os mesmos campos do cabeçalho da requisição com valores correspondentes. Na Figura 29 é demonstrada a resposta, contendo no cabeçalho informações do dispositivo e a política com novos parâmetros de desempenho.

Figura 29: Disposição da aplicação resposta.

De mais informações sobre os fontes do cliente, web service e demais códigos comentados podem ser encontrados nos Anexos.

7. Conclusão

A distribuição de políticas de gerenciamento de redes como foi proposto, permite o gerenciamento entre domínios. Uma vez que todos os domínios que utilizam COPS precisam de um PDP local, a abordagem proposta se destaca por não ter essa condição, por flexibilidade de sua arquitetura e o gerenciamento de dispositivos distribuídos. Por trabalhar na camada de aplicação é possível agregar novas funções diferentemente do padronizado que trabalha na camada de transporte.

Outro ponto de favorecimento é o meio de comunicação utilizado para distribuição das políticas o HTTP, por sua portabilidade e extensibilidade. Complementando, a capacidade de desenvolvimento e evolução do XML são uma das peças fundamentais para a flexibilidade do protocolo.

A utilização do protocolo SOAP para definir o formato XML para troca de mensagens é decisiva para a distribuição, pois esta pode proporcionar independência de plataforma e utilizar protocolo de grande credibilidade como o HTTP e SMTP. Estes fatores, juntamente com os *Web services*, contribuem para que a utilização do SOAP aumente como tecnologia de comunicação para sistemas distribuídos.

A configuração dos dispositivos mediante o recebimento das políticas não é abordado nesta dissertação. Pois o protocolo utilizado tem a função de troca de mensagens entre o PEP e o PDP. Todavia, os mecanismos para estabelecer as políticas requisitadas aos dispositivos podem ser criados (agentes) ou adaptados aos já existentes (SNMP, WMI, etc.).

No desenvolvimento da aplicação temos um conjunto de ferramentas abertas: tomcat, eclipse e axis. Todas elas possibilitaram a atingir os objetivos propostos neste trabalho, suas utilizações permitiram a aplicação portabilidade, eficiência e flexibilidade. A quantidade de política pode ser aumentada conforme a necessidade do gerenciamento e a complexidade das políticas da mesma forma.

Os objetivos desta dissertação eram na criação de uma aplicação que possibilitasse a distribuição de políticas de gerenciamento de forma descentralizada, eficiente e controlada. No desenvolvimento da aplicação foram alvos: a comunicação entre o dispositivo (PEP) e o web service (PDP), a comunicação trafegando em HTTP,

mas com informações em XML e encapsulamento no protocolo SOAP. Ao final do desenvolvimento os objetivos proposto desta dissertação foram plenamente alcançados.

As contribuições desta pesquisa estão na distribuição de políticas de gerenciamento através do SOAP/XML, onde foi acrescentado funcionamento baseado no framework da IETF, possibilitando gerenciamento distribuído, extensibilidade, flexibilidade, portabilidade e por fim capacidade de evolução.

7.1 Trabalhos Futuros

Em trabalhos futuros parâmetros tais como: desempenho, segurança e consistência deverão ser pesquisados para atestar a eficiência do protocolo. Também construir um repositório utilizando o modelo CIM ou PIB, acrescentando uma variedade de dispositivos gerenciados bem como políticas de gerenciamento mais abrangentes.

Outros tópicos para trabalhos futuros são a implementação de agentes coletores das informações gerenciadas, criando uma comunicação entre a distribuição de políticas proposta e a aplicação delas nos dispositivos.

E por fim, a construção de uma ferramenta de gerenciamento de políticas capaz de coordenar todas as operações. Desde a criação das políticas, armazenamentos no repositório, ordenar os PDPs, classificar os PEPs e configurar os agentes de monitoramento.

Referências Bibliográficas

[Andrade, 2004] Andrade A. e Westphall C. “Protocolo para Distribuição de Políticas de Gerenciamento Utilizando SOAP/XML” Simpósio Brasileiro de Redes de Computadores – SBRC – IX Workshop de Gerenciamento de Redes WSRG, pp. 27-38, Maio 2004.

[Axis, 2004] Framework Axis. Disponível em: http://ws.apache.org/axis/index.html. Acessado em 17 de julho de 2004.

[Barda, 2002] Barba A., García J., Xirinacs L., Manual para representaciones geométrica de políticas. Internal Report Entel, UPC. 2002.

[Boyle, 1999] Boyle, J., et al., *The COPS* (*Common Open Policy Service*) Protocol, February 1999, < draft-ietf-rap-cops-06.txt >

[Box, 2000] Box, D., Ehnerbuske, D. Kakyvaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer, D., Simple Object Acess Protocol (SOAP) 1.1. Disponível em: <http://www.w3.org/tr/soap>

[By, 2003] By S. Jha, M Hassan, “*Java Implementation of Policy-based bandwith management*”, Internation Journal of Network Management, vol 13, number 4, August 2003.

[Chadha, 2002] Chadha R., Lapiotis G., Wright S., "*Policy-Based Networking*," *IEEE Network*, vol. 16, pp. 8-9, 2002.

[Cristensen, 2001] Christensen E., Mederith G., Francisco Cubero F., Weerawarana S. “*Web services Description Language”* (WSDL) 1.1 Microsoft Corporation e IBM Research, 2001. Disponível em: http//:msdn.microsoft.com.

[Czezowski, 2000] Czezowski P., Hamada T., Chujo T., "*Policy-based Management for Enterprise and Carrier IP Networking*," *Fujitsu Science and Technology*, vol. 36, pp. 128-39, 2000.

[Danciu, 2004] Danciu V., Kempter B. "*From Processes to Policies Concepts for Large Scale Policy Generation*" – 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), 2004.

[Devarakonda, 2002] Devaranko, *A Policy-Based Storage Management Framework*. Third International Workshop on Policies for Distributed Systems and Networks (POLICY' 02) IEEE Computer Society, 2002.

[Eclipse, 2000] Elipse. Disponível: http://elipse.org. Acessado em 20 de junho de 2004.

[Fesh, 2002] Fesh et. al, *A Policy-Based Quality of Service Management System for IP DiffServ Networks*. IEEE Communications Magazine, Mach 2002

[Flegkas, 2001] P. Flegkas, P. Trimintzios, G. Pavlou, I. Andrikopoulos, and C. F. Cavalcanti, "*Policy-based Extensible Hierarchical Network Management*," presented at Proceedings of Workshop on Policies for Distributed Systems and Networks (Policy 2001), Bristol, U.K, 2001,

[Flegkas, 2002] Flegkas P., Trimintzios P., Pavlou G., "*A Policy- based Quality of Service Management Architecture for IP DiffServ Networks*", IEEE Network Magazine, Special issue on Policy Based Networking, vol. 16, no. 2, pp. 50-56, March/April 2002

[Flegkas, 2003] P. Flegkas et al., "Design and Implementation of a Policy-Based Resource Management Architecture", IEEE/IFIP IM, 2003.

[Hamada, 2000] T. Hamada, P. Czezowski, and T. Chujo, "Policy-based Management for Enterprise and Carrier IP Networking," Fujitsu Science and Technology, vol.36, pp. 128-39, 2000.

[Hamada, 1999] Hamada T., Blight D. C., Czezowski P., "*Active Policies in Knowledge Hyperspace: Intelligent Agents and Policy-based Networking*," *KNOM review*, vol. 2, pp. 31-41, 1999.

[HP, 2001] Hewlett-packard. HP OpenView PolicyXpert Homepage. Disponível em: <http://www.openview.hp.com/ >.

[Jha, 2003] Jha S., Hassan M., "*Development of SNMP-XML Translator and Gateway for XML-based Integrated Network Management*". International Journal of Network Management. Vol. 13, Number 4, July/August 2003.

[Law, 2003] Law K. l. et al, "*Scalable Design of a Policy-Based Management System and its Peformance*" – IEEE Communications Magazi, June 2003.

[Lupu, 1999] Lupu E. Sloman M. "*Conflicts in Policy-based Distributed Systems Management*". IEEE Transaction on Software Engineering, Vol 25. No.6 pp. 852-869 November/December 1999.

[Lymberopoulos, 2003] Lymberopoulos l., Lupu E., Sloman M. "*An Adaptive Policy-Based Framework for Network Services Management*" – Journal of Network and Systems Management, Vol. 11, No. 3, September 2003.

[Moffett, 1990] Moffett J., Sloman M., Twidle K., "*Specifying discretionary access control policy for distributed systems*," *Computer Communications*, vol. 13, pp. 571-80, 1990, http://citeseer.nj.nec.com/moffett90specifying.html.

[Moffett, 1991] Moffett J. D. and Sloman M., "The representation of policies as system objects", presented at Organizational computing systems", Atlanta, Georgia, 1991.

[Moore, 2001] Moore, E. Elleson, J. Strassner, A. Westerinen. "*Policy Core Informatin Model"* – Version 1 Specification. IETF Request For Comment (RFC) 3060. February 2001.

[Natarajan, 2001] Natarajan R, Mckee A. P., Mathur, "*A XML based Policy-Driven Information Service"*, IEEE/IFIP International Symposium on Integrated Network Management (IM'2001), May 2001.

[Phase, 2004] Phase K., Dasilva Luiz. "*Protocol Support for Policy-Based Management of Mobile Ad Hoc Networks"* – 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), 2004.

[QPM, 2000] Cisco systems. Cisco QoS Policy Manager (QPM) Homepage. Disponível em: http://www.cisco.com/ Acessado em 17 de junho de 2004.

[RFC 1102] Clark D., "*Policy Routing in Internet Protocols*," Network working group, May 1989, http://www.ietf.org/rfc/rfc1102.txt.

[Reynolds, 2001] Reynolds M., "*What Are Web Services*". Disponível em <http://www.webservicesarchitect.com> . Acessado em 25 de julho de 2004.

[RFC 1441] Case J., McCloghrie K., Rose M., Waldbusser S., "*Introduction to version 2 of the Internet-standard Network Management Framework*," SNMPv2, April 1993, http://www.ietf.org/rfc/rfc1441.txt.

[RFC 1157] Case, J. D., et al., *Simples Network Management Protocol* (SNMP), RFC 1157, May 1990.

[RFC 2210] Wroclawski J., "*The Use of RSVP with IETF Integrated Services*, "Network Working Group, September 1997, http://www.ietf.org/rfc/rfc2210.txt.

[RFC 2475] Blake S., Black D., Carlson M., Davies E., Wang Z., Weiss W., *"An Architecture for Differentiated Services,"* Network working group,December 1998, http://www.ietf.org/rfc/rfc2475.txt.

[RFC 2597] Fielding, E., et al., *Hypertext Transfer Protocol* – HTTP/1.1, RFC 2597, February 1999.

[RFC 2307] Howard, L., et al., *An Approach for Using LDAP as a Network Information Service*, RFC 2307, March 1998.

[RFC 2748] Durham D., Boyle J., Cohen R., Herzog S., Rajan R., Sastry A., "*The COPS (Common Open Policy Service) Protocol*," Network Working Group, January 2000, http://www.ietf.org/rfc/rfc2748.txt?number=2748.

[RFC 2749] Herzog S., Boyle J., Cohen R., Durham D., Rajan R., *COPS usage for RSVP*," Network Working Group, January 2000, http://www.ietf.org/rfc/rfc2749.txt

[RFC 2753] Yavatkar R., Pendarakis D., Guerin R., "*A Framework for Policy-based Admission Control*," Resource Allocation Protocol, January 2000, http://www.ietf.org/rfc/rfc2753.txt.

[RFC 3006] *Distributed Management Task Force*, "CIM Schema: Version 2.8 (Preliminary)," updated 2003, http://www.dmtf.org/standards/cim/cim_schema_v28.

[RFC 3159] McCloghrie K., Fine M., Seligson J., Chan K., Hahn S., Sahita R., *"Structure of Policy Provisioning Information* (SPPI)," Resource Allocation Protocol, August 2001, http://www.ietf.org/rfc/rfc3159.txt.

[RFC 3198] Westerinen A., Schnizlein J., Strassner J., Scherling V, Quinn B., Herzog S., Huynh A., Carlson M. Perry J., Waldbusser S., "*Terminology for Policy-Based Management*," Policy framework, November 2001, http://www.ietf.org/rfc/rfc3198.txt

[RFC 3318] Sahita R., Hahn S., Intel Labs, Nortel Networks, Cisco System "*Framework Policy Information Base*" Policy Framework, March 2003, http://www.ietf.org/rfc/rfc3318.txt.

[RFC 3460] Moore B., "*Policy Core Information Model Extensions*," Policy Framework, January 2003, http://www.ietf.org/rfc/rfc3460.txt.

[RFC 3586] Blaze M., Keromytis A., Richardson M., Sanchez L., *"IP Security Policy (IPSP) Requirements*," IP Security Policy, August 2003, http://www.ietf.org/rfc/rfc3586.txt.

[Robinson, 1988] Robinson D. C. and Sloman M., "*Domain-based access control for distributed computing systems*," *Software Engineering*, vol. 3, pp. 161-170,1988.

[Saganich, 2001] Saganich A. ”Java and *Web Services*”, Par I. Disponível em: <http://www.onjava.com>. Acessado em abril de 2004.

[Shepard, 2000] Shepard S., *“Policy-based Network: Hype and Hope”*, IEEE Internation Journal of Network Management, vol. 2, No. 1, pp 12-16, January/February 2000.

[Sloman, 1994] SLOMAN, M., “Policy Driven Management for Distributed System”, Journal of Network and System Management, Vol 2, No. 4, pp. 333-360, Plenum Publishing, December, 1994.

[Sloman, 2002] M. Sloman and E. Lupu, "*Security and Management Policy Specification*,"*IEEE Network*, pp. 10-19, 2002.

[Snir, 2003] Snir Y., Ramberg Y., Strassner J., Cohen R., Moore B., "*Policy QoS Information Model*," Policy Framework, May 2003, http://www.ietf.org/internet-drafts/draft-ietf-policy-qos-infomodel-05.txt.

[Stevens, 1999a] Stevens M., Weiss W. et al., *"Policy Framework,"* Policy Framework, 13 Sept 99, Draft-ietf-policy-framework-00.txt.

[Stevens, 1999b] Stevens M., Weiss W. J., "Policy-based Management for IP Networks," Bell Labs Technical Journal, pp. 75-94, 1999.

[Strassner, 1999] Strassner, J. et al., *Terminology for Describing Network Policy and Services*. February 1999, <draft-ietf-policy-terms-01.txt>

[Strassner, 2002] Strassner J., *"DEN-ng: Achieving Business-Driven Network Management*," presented at NOMS 2002, Florence, Italy, 2002. http://www.comsoc.org/confs/noms/2002/techincal_sessions.html.

[Strassner, 2003] Strassner J., Policy-Based Network Management: Solutions for the Next Generation: Morgan Kaufmann, 2003.

[Stone, 2001] Stone G. N., Lundy B., Xie G. G., "*Network Policy Languages: A Survey and a New Approach*," *IEEE Network*, pp. 10-21, 2001.

[Sun, 2000] Sun W., "QoS/Policy/Constraint Based Routing," Ohio State University, Survey paper 7 February 2000, http://www.cis.ohio-state.edu/~jain/cis788-99/qos_routing.

[Trimintzios, 2002] P. Trimintzios, P. Flegkas, G. Pavlou, "*A Policy- based Quality of Service Management Architecture for IP DiffServ Networks*", IEEE Network Magazine, Special issue on Policy Based Networking, vol. 16, no. 2, pp. 50-56, March/April 2002.

[Tomcat, 2004] Apache Tomcat. Disponível em: http://jakarta.apache.org/tomcat Acessado em 17 de junho de 2004.

[Vaguetti, 2003] Vaguetti L. et al., "*An Architecture for Integrated Policy-Based Managemente of QoS and Multicast-enable Networks*" Simpósio Brasileiro de Redes de Computadores, Maio 2003.

[Verma, 2002] Verma D. C., "Simplifying Network Administration Using Policy-Based Management," IEEE Network, vol. 16, pp. 20-6, 2002.

[W3C] *World Wide Web Consortium*. Disponível em: http://www.w3c.org. Acessado em julho de 2004.

Anexo A

Este anexo traz o fonte do QoSClient.java comentado. O QoSClient.java é o dispositivo sob monitoramento e o receptor da política.

// Fonte do QoSClient.java

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.rmi.RemoteException;
import java.text.SimpleDateFormat;
import java.util.Calendar;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSplitPane;
import javax.swing.JTextField;
import javax.xml.rpc.ServiceException;

import localhost.axis.QoSWS_jws.QoSWSService;
import localhost.axis.QoSWS_jws.QoSWSServiceLocator;
import mensagens.Cabecalho;
import mensagens.CorpoRequisicao;
import mensagens.EnvelopePolitica;
import mensagens.EnvelopeRequisicao;

/**
 *
 * Classe responsável por desenhar um formulário contendo campos para
 * preenchimento da requisição, e de um botão para servir de gatilho para a
 * solicitação de políticas.
 *
 * Na parte direita será mostrada a politica retornada pelo Web Service.
 *
 * @author Aujor Tadeu C. Andrade
 * @author Luiz Carlos Metzger
 *
 */

public class QoSClient extends JFrame {

    private static final long serialVersionUID = 1L;

    private Dimension dm = Toolkit.getDefaultToolkit().getScreenSize();
    private JButton btEnviar = null;
```

```
// Campos para requisição
private JTextField tfRqOrigem = new JTextField();
private JTextField tfRqDestino = new JTextField();
private JTextField tfRqTipoMSG = new JTextField();
private JTextField tfRqTempoCab = new JTextField();
private JTextField tfRqReqID = new JTextField();
private JTextField tfRqSNY = new JTextField();
private JTextField tfRqTipoEvento = new JTextField();
private JTextField tfRqPrioridade = new JTextField();
private JTextField tfRqTempoCor = new JTextField();
private JTextField tfRqBandwidth = new JTextField();
private JTextField tfRqDelay = new JTextField();
private JTextField tfRqPerdaPacotes = new JTextField();

// Campos para mostrar o retorno da requisição
private JTextField tfRtOrigem = new JTextField();
private JTextField tfRtDestino = new JTextField();
private JTextField tfRtTipoMSG = new JTextField();
private JTextField tfRtTempoCab = new JTextField();
private JTextField tfRtReqID = new JTextField();
private JTextField tfRtSNY = new JTextField();
private JTextField tfRtTipoEvento = new JTextField();
private JTextField tfRtBandwidth = new JTextField();
private JTextField tfRtDelay = new JTextField();
private JTextField tfRtPerdaPacotes = new JTextField();

/**
 * Construtor padrão da classe
 */
public QoSClient() {
    super();

    setTitle("Distribuição de Políticas");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Não permite redimensionar a tela
    //              setResizable(false);

    JSplitPane split = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
            panelRequisicao(), panelRetorno());
    split.setVisible(true);
    getContentPane().add(split);
    pack();

    // Centraliza o formulário na tela
    setBounds((dm.width - getWidth()) / 2, (dm.height - getHeight()) / 2,
     getWidth(), getHeight());
    setVisible(true);
}

/**
 * Implementação do painel de requisições
 */
private JPanel panelRequisicao() {
```

```
    JPanel pn = new JPanel();
    JPanel tp = new JPanel();
    JPanel ce = new JPanel();
    JPanel fu = new JPanel();

    // Cria Botão
    btEnviar = new JButton("Enviar");
    btEnviar.setMnemonic('E');
    btEnviar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                enviarRequisicao();
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    });

    // Painel Topo
    tp.add(new JLabel("Requisição"));
    tp.setBackground(Color.GRAY);

    // Painel Centro
    ce.setLayout(new BorderLayout());
    ce.add(cabecalhoRequisicao(), BorderLayout.NORTH);
    ce.add(corpoRequisicao(), BorderLayout.CENTER);

    // Painel fundo
    fu.add(btEnviar);

    // Painel Principal
    pn.setLayout(new BorderLayout());
    pn.add(tp, BorderLayout.NORTH);
    pn.add(ce, BorderLayout.CENTER);
    pn.add(fu, BorderLayout.SOUTH);
    return pn;
}

/**
 * Monta o painel do Cabeçalho da requisição
 */
private JPanel cabecalhoRequisicao() {

    JPanel pn = new JPanel();
    JPanel tp = new JPanel();
    JPanel ce = new JPanel();
    // Painel Topo
    tp.add(new JLabel("Cabeçalho"));
    tp.setBackground(Color.GRAY.brighter());

    // Painel Centro
    ce.setLayout(new GridLayout(3, 4));
    ce.add(new JLabel("Origem"));
    ce.add(tfRqOrigem);
    ce.add(new JLabel("Destino"));
    ce.add(tfRqDestino);
```

```
        ce.add(new JLabel("Tipo MSG"));
        ce.add(tfRqTipoMSG);
        ce.add(new JLabel("Tempo"));
        ce.add(tfRqTempoCab);
        ce.add(new JLabel("ReqID"));
        ce.add(tfRqReqID);
        ce.add(new JLabel("SNY"));
        ce.add(tfRqSNY);

        // Painel Principal
        pn.setLayout(new BorderLayout());
        pn.add(tp, BorderLayout.NORTH);
        pn.add(ce, BorderLayout.CENTER);

        return pn;

    }

    /**
     * Monta o painel do corpo da requisição
     */
    private JPanel corpoRequisicao() {

        JPanel pn = new JPanel();
        JPanel tp = new JPanel();
        JPanel ce = new JPanel();

        // Painel Topo
        tp.add(new JLabel("Corpo"));
        tp.setBackground(Color.GRAY.brighter());

        // Painel Centro
        ce.setLayout(new GridLayout(3, 4));
        ce.add(new JLabel("Tipo Evento"));
        ce.add(tfRqTipoEvento);
        ce.add(new JLabel("Prioridade"));
        ce.add(tfRqPrioridade);
        ce.add(new JLabel("Tempo"));
        ce.add(tfRqTempoCor);
        ce.add(new JLabel("Bandwidth"));
        ce.add(tfRqBandwidth);
        ce.add(new JLabel("Delay"));
        ce.add(tfRqDelay);
        ce.add(new JLabel("Perda Pacotes"));
        ce.add(tfRqPerdaPacotes);

        // Painel Principal
        pn.setLayout(new BorderLayout());
        pn.add(tp, BorderLayout.NORTH);
        pn.add(ce, BorderLayout.CENTER);
        return pn;
    }
    /**
     * Implementação do painel de retorno
     */
    private JPanel panelRetorno() {
```

```
    JPanel pn = new JPanel();
    JPanel tp = new JPanel();
    JPanel ce = new JPanel();
    JPanel fu = new JPanel();

    // Painel Topo
    tp.add(new JLabel("Retorno"));
    tp.setBackground(Color.GRAY);

    // Painel Centro
    ce.setLayout(new BorderLayout());
    ce.add(cabecalhoRetorno(), BorderLayout.NORTH);
    ce.add(corpoRetorno(), BorderLayout.CENTER);

    // Painel Principal
    pn.setLayout(new BorderLayout());
    pn.add(tp, BorderLayout.NORTH);
    pn.add(ce, BorderLayout.CENTER);
    pn.add(fu, BorderLayout.SOUTH);
    return pn;
}

/**
 * Monta o painel do cabecalho do retorno
 */
private JPanel cabecalhoRetorno() {

    JPanel pn = new JPanel();
    JPanel tp = new JPanel();
    JPanel ce = new JPanel();

    // Painel Topo
    tp.add(new JLabel("Cabeçalho"));
    tp.setBackground(Color.GRAY.brighter());

    // Painel Centro
    ce.setLayout(new GridLayout(3, 4));
    ce.add(new JLabel("Origem"));
    ce.add(tfRtOrigem);
    ce.add(new JLabel("Destino"));
    ce.add(tfRtDestino);
    ce.add(new JLabel("Tipo MSG"));
    ce.add(tfRtTipoMSG);
    ce.add(new JLabel("Tempo"));
    ce.add(tfRtTempoCab);
    ce.add(new JLabel("ReqID"));
    ce.add(tfRtReqID);
    ce.add(new JLabel("SNY"));
    ce.add(tfRtSNY);

    // Painel Principal
    pn.setLayout(new BorderLayout());
    pn.add(tp, BorderLayout.NORTH);
    pn.add(ce, BorderLayout.CENTER);
```

```
    return pn;
}
/**
 * Monta o painel do corpo do retorno
 */
private JPanel corpoRetorno() {

    JPanel pn = new JPanel();
    JPanel tp = new JPanel();
    JPanel ce = new JPanel();

    // Painel Topo
    tp.add(new JLabel("Corpo"));
    tp.setBackground(Color.GRAY.brighter());

    // Painel Centro
    ce.setLayout(new GridLayout(4, 2));
    ce.add(new JLabel("Tipo Evento"));
    ce.add(tfRtTipoEvento);
    ce.add(new JLabel("Bandwidth"));
    ce.add(tfRtBandwidth);
    ce.add(new JLabel("Delay"));
    ce.add(tfRtDelay);
    ce.add(new JLabel("Perda Pacotes"));
    ce.add(tfRtPerdaPacotes);

    // Painel Principal
    pn.setLayout(new BorderLayout());
    pn.add(tp, BorderLayout.NORTH);
    pn.add(ce, BorderLayout.CENTER);

    return pn;

}

/**
 * Método criado para enviar a requisição ao Web Service.
 *
 * É invocado no momento do clique do botão Enviar.
 *
 * @throws ServiceException
 * @throws RemoteException
 */
private void enviarRequisicao() throws ServiceException, RemoteException {
    System.out.println("enviar Requisicao foi clicado!");

    // Montagem do envelope para requisição
    EnvelopeRequisicao envreq = new EnvelopeRequisicao(new Cabecalho(),
            new CorpoRequisicao());
    envreq.getCabecalho().setOrigem(tfRqOrigem.getText());
    envreq.getCabecalho().setDestino(tfRqDestino.getText());
    envreq.getCabecalho().setEnvio(Calendar.getInstance());
    envreq.getCabecalho().setReqid(new Long(tfRqReqID.getText()));
    envreq.getCabecalho().setSny(new Long(tfRqSNY.getText()));
    envreq.getCabecalho().setTipoMensagem("Resposta");
```

```
        QoSWSService service = new QoSWSServiceLocator();
        mostraRetorno(service.getQoSWS().buscaPolitica(envreq));

    }

    /**
     * Mostra a política retornada pelo Web Service
     */
    private void mostraRetorno(EnvelopePolitica env) {

        // Mostra o cabecalho da politica retornada

            tfRtOrigem.setText(env.getCabecalho().getDestino());
            tfRtDestino.setText(env.getCabecalho().getOrigem());
           tfRtTipoMSG.setText(env.getCabecalho().getTipoMensagem());
           tfRtTipoEvento.setText("Desempenho");
            tfRtTempoCab.setText(new SimpleDateFormat("dd/MM/yyyy").format(env
.getCabecalho().getEnvio().getTime()));
        tfRtReqID.setText(env.getCabecalho().getReqid().toString());
        tfRtSNY.setText(env.getCabecalho().getSny().toString());

        for (int i=0;i<env.getCorpo().getParametros().length;i++) {
            System.out.println("Retorno: " + env.getCorpo().getParametros()[i]);
            tfRtBandwidth.setText( (String) env.getCorpo().getParametros()[1] );
            tfRtDelay.setText( (String) env.getCorpo().getParametros()[3] );
            tfRtPerdaPacotes.setText( (String) env.getCorpo().getParametros()[5] );
        }
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        new QoSClient();
    }
}

      /**
```

Anexo B

Este anexo traz o fonte do QoSWS.java comentado. O QoSWS.java é o dispositivo que recebe as requisições as analisa e envia a política corretiva.

// Fontes do QoSWS.java

```
import mensagens.Cabecalho;
import mensagens.CorpoPolitica;
import mensagens.EnvelopePolitica;
import mensagens.EnvelopeRequisicao;
```

```
/**
* @author Aujor Tadeu C. Andrade
*@author Luiz Carlos Metzger
 */
public class QoSWS {

    private EnvelopePolitica p1;

    /**
     * Construtor da classe
     */
    public QoSWS() {
        super();

        p1 = new EnvelopePolitica(new Cabecalho(), new CorpoPolitica());
        p1.getCorpo().setParametros(
                new Object[] { "Bandwidth", "150", "Delay", "10%", "Perda", "15%" } );
    }

    /**
     * Retorna a politica requisitada
     *
     * @return
     */
    public EnvelopePolitica buscaPolitica(EnvelopeRequisicao envelope) {

        // 1o. Exemplo de Politica
        if (buscaParametro(envelope.getCorpo().getParametros(), "trafego") <= 90
            && buscaParametro(envelope.getCorpo().getParametros(), "perda") >= 30)
            {
            p1.setCabecalho(envelope.getCabecalho());
            p1.getCabecalho().setSny(
            new Long(p1.getCabecalho().getSny().longValue() + 1));
            return p1;
            }

        p1.setCabecalho(envelope.getCabecalho());
        p1.getCabecalho().setSny(
        new Long(p1.getCabecalho().getSny().longValue() + 1));
        return p1;
    }

    /**
     * Retorna o valor do parametro
     */
    private long buscaParametro(Object[] parametros, String chave) {
    return -1;
    }
}
```

```
      /**
      Cabecalho.java
      *
      Este arquivo é gerado pelo WSDL
      by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST) WSDL2Java emitter.
      */
```

```
package mensagens;
import java.util.Calendar;
public class Cabecalho  implements java.io.Serializable {

private static final long serialVersionUID = 1L;
private java.lang.String destino;
private java.util.Calendar envio;
private java.lang.String origem;
private java.lang.Long reqid;
private java.lang.Long sny;
private java.lang.String tipoMensagem;

public Cabecalho() {
}
public Cabecalho(
		java.lang.String destino,
		java.util.Calendar envio,
		java.lang.String origem,
		java.lang.Long reqid,
		java.lang.Long sny,
		java.lang.String tipoMensagem) {
		this.destino = destino;
		this.envio = envio;
		this.origem = origem;
		this.reqid = reqid;
		this.sny = sny;
		this.tipoMensagem = tipoMensagem;
}
/**
Pega os valores do destino do Cabecalho.
*
@return destino
*/
public java.lang.String getDestino() {
		return destino;
}
/**
Pega os valores do destino do Cabecalho.
*
@param destino
*/
public void setDestino(java.lang.String destino) {
		this.destino = destino;
}
/**
Pega os valores do destino do Cabecalho.
*
@return envio
*/
public java.util.Calendar getEnvio() {
		return envio;
}
/**
Grupo de valores do envio do Cabecalho.
*
```

```
@param date
*/
public void setEnvio(Calendar envio) {
	this.envio = envio;
}
/**
Grupo de valores do envio do Cabecalho.
*
@return origem
*/
public java.lang.String getOrigem() {
	return origem;
}
/**
Grupo de valores do envio do Cabecalho.
*
@param origem
*/
public void setOrigem(java.lang.String origem) {
	this.origem = origem;
}
/**
Pega os valores do reqid do Cabeçalho.
*
@return reqid
*/
public java.lang.Long getReqid() {
	return reqid;
}
/**
Pega os valores do reqid do Cabeçalho.
*
@param reqid
*/
public void setReqid(java.lang.Long reqid) {
	this.reqid = reqid;
}
/**
Pega os valores do sny do Cabeçalho.
*
@return sny
*/
public java.lang.Long getSny() {
	return sny;
}
/**
Pega os valores do sny do Cabeçalho.
*
@param sny
*/
public void setSny(java.lang.Long sny) {
	this.sny = sny;
}
/**
Pega os valores do TipoMensagem do Cabeçalho.
*
```

```
@return tipoMensagem
*/
public java.lang.String getTipoMensagem() {
	return tipoMensagem;
}
/**
Grupo de valores do TipoMensagem do Cabeçalho.
*
@param tipoMensagem
*/
public void setTipoMensagem(java.lang.String tipoMensagem) {
	this.tipoMensagem = tipoMensagem;
}

private java.lang.Object __equalsCalc = null;
public synchronized boolean equals(java.lang.Object obj) {
	if (!(obj instanceof Cabecalho)) return false;
	Cabecalho other = (Cabecalho) obj;
	if (obj == null) return false;
	if (this == obj) return true;
	if (__equalsCalc != null) {
	return (__equalsCalc == obj);
	}
	__equalsCalc = obj;
	boolean _equals;
	_equals = true &&
	((this.destino==null && other.getDestino()==null) || (this.destino!=null &&
		this.destino.equals(other.getDestino()))) && ((this.envio==null &&
		other.getEnvio()==null) || (this.envio!=null &&
		this.envio.equals(other.getEnvio()))) &&
	((this.origem==null && other.getOrigem()==null) || (this.origem!=null &&
	this.origem.equals(other.getOrigem()))) && ((this.reqid==null &&
	other.getReqid()==null) || (this.reqid!=null && this.reqid.equals(other.getReqid()))) &&
	((this.sny==null && other.getSny()==null) || (this.sny!=null &&
	this.sny.equals(other.getSny()))) && ((this.tipoMensagem==null &&
	ther.getTipoMensagem()==null) || (this.tipoMensagem!=null &&
	this.tipoMensagem.equals(other.getTipoMensagem())));
	__equalsCalc = null;
	return _equals;
}
private boolean __hashCodeCalc = false;
public synchronized int hashCode() {
	if (__hashCodeCalc) {
	return 0;
	}
	__hashCodeCalc = true;
	int _hashCode = 1;
	if (getDestino() != null) {
	_hashCode += getDestino().hashCode();
	}
	if (getEnvio() != null) {
	_hashCode += getEnvio().hashCode();
	}
	if (getOrigem() != null) {
	_hashCode += getOrigem().hashCode();
	}
```

```
            if (getReqid() != null) {
            _hashCode += getReqid().hashCode();
            }
            if (getSny() != null) {
            _hashCode += getSny().hashCode();
            }
            if (getTipoMensagem() != null) {
            _hashCode += getTipoMensagem().hashCode();
            }
            __hashCodeCalc = false;
            return _hashCode;
}

// Tipo Metadados
private static org.apache.axis.description.TypeDesc typeDesc =
            new org.apache.axis.description.TypeDesc(Cabecalho.class, true);

static {
            typeDesc.setXmlType(new          javax.xml.namespace.QName("http://mensagens",
            "Cabecalho"));
            org.apache.axis.description.ElementDesc          elemField          =          new
            org.apache.axis.description.ElementDesc();
            elemField.setFieldName("destino");
            elemField.setXmlName(new javax.xml.namespace.QName("", "destino"));
            elemField.setXmlType(new
            javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/",
            "string"));
            typeDesc.addFieldDesc(elemField);
            elemField = new org.apache.axis.description.ElementDesc();
            elemField.setFieldName("envio");
            elemField.setXmlName(new javax.xml.namespace.QName("", "envio"));
            elemField.setXmlType(new
            javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "dateTime"));
            typeDesc.addFieldDesc(elemField);
            elemField = new org.apache.axis.description.ElementDesc();
elemField.setFieldName("origem");
            elemField.setXmlName(new javax.xml.namespace.QName("", "origem"));
            elemField.setXmlType(new
            javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/",
            "string"));
            typeDesc.addFieldDesc(elemField);
            elemField = new org.apache.axis.description.ElementDesc();
            elemField.setFieldName("reqid");
            elemField.setXmlName(new javax.xml.namespace.QName("", "reqid"));
            elemField.setXmlType(new
            javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/", "long"));
            typeDesc.addFieldDesc(elemField);
            elemField = new org.apache.axis.description.ElementDesc();
            elemField.setFieldName("sny");
            elemField.setXmlName(new javax.xml.namespace.QName("", "sny"));
            elemField.setXmlType(new
            javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/", "long"));
            typeDesc.addFieldDesc(elemField);
            elemField = new org.apache.axis.description.ElementDesc();
            elemField.setFieldName("tipoMensagem");
            elemField.setXmlName(new javax.xml.namespace.QName("", "tipoMensagem"));
```

```
                elemField.setXmlType(new
                javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/",
                "string"));
                typeDesc.addFieldDesc(elemField);
        }

        /**
        Retorna Metadados
        */
        public static org.apache.axis.description.TypeDesc getTypeDesc() {
                return typeDesc;
        }

        /**
        Adapta a Serialização
        */
        public static org.apache.axis.encoding.Serializer getSerializer(
                java.lang.String mechType,
                java.lang.Class _javaType,
                javax.xml.namespace.QName _xmlType) {
                return
                new  org.apache.axis.encoding.ser.BeanSerializer(
                _javaType, _xmlType, typeDesc);
        }
        /**
        Desadapta a Deserialização
        */
        public static org.apache.axis.encoding.Deserializer getDeserializer(
                java.lang.String mechType,
                java.lang.Class _javaType,
                javax.xml.namespace.QName _xmlType) {
                return
                new  org.apache.axis.encoding.ser.BeanDeserializer(
                _javaType, _xmlType, typeDesc);
        }
        }
/**
 * CorpoPolitica.java
 *
 * Este arquivo é gerado pelo WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST) WSDL2Java emitter.
 */
package mensagens;

public class CorpoPolitica  implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private java.lang.Object[] parametros;
    private java.lang.String politica;

    public CorpoPolitica() {
    }
    public CorpoPolitica(
          java.lang.Object[] parametros,
          java.lang.String politica) {
          this.parametros = parametros;
          this.politica = politica;
```

```
}
/**
 * Pega os valores dos parametro do CorpoPolitica.
 *
 * @return parametros
 */
public java.lang.Object[] getParametros() {
    return parametros;
}
/**
 * Pega os valores dos parametro do CorpoPolitica.
 *
 * @param parametros
 */
public void setParametros(java.lang.Object[] parametros) {
    this.parametros = parametros;
}
/**
 * Pega os valores dos parametro do CorpoPolitica.
 *
 * @return politica
 */
public java.lang.String getPolitica() {
    return politica;
}
/**
 * Pega os valores dos parametro do CorpoPolitica.
 *
 * @param politica
 */
public void setPolitica(java.lang.String politica) {
    this.politica = politica;
}
private java.lang.Object __equalsCalc = null;
public synchronized boolean equals(java.lang.Object obj) {
    if (!(obj instanceof CorpoPolitica)) return false;
    CorpoPolitica other = (CorpoPolitica) obj;
    if (obj == null) return false;
    if (this == obj) return true;
    if (__equalsCalc != null) {
        return (__equalsCalc == obj);
    }
    __equalsCalc = obj;
    boolean _equals;
    _equals = true &&
        ((this.parametros==null && other.getParametros()==null) ||
         (this.parametros!=null &&
          java.util.Arrays.equals(this.parametros, other.getParametros()))) &&
        ((this.politica==null && other.getPolitica()==null) ||
         (this.politica!=null &&
          this.politica.equals(other.getPolitica())));
    __equalsCalc = null;
    return _equals;
}

private boolean __hashCodeCalc = false;
```

```
    public synchronized int hashCode() {
        if (__hashCodeCalc) {
            return 0;
        }
        __hashCodeCalc = true;
        int _hashCode = 1;
        if (getParametros() != null) {
            for (int i=0;
                 i<java.lang.reflect.Array.getLength(getParametros());
                 i++) {
                java.lang.Object obj = java.lang.reflect.Array.get(getParametros(), i);
                if (obj != null &&
                    !obj.getClass().isArray()) {
                    _hashCode += obj.hashCode();
                }
            }
        }
        if (getPolitica() != null) {
            _hashCode += getPolitica().hashCode();
        }
        __hashCodeCalc = false;
        return _hashCode;
    }
    // Type metadata
    private static org.apache.axis.description.TypeDesc typeDesc =
        new org.apache.axis.description.TypeDesc(CorpoPolitica.class, true);

    static {
       typeDesc.setXmlType(new                javax.xml.namespace.QName("http://mensagens",
"CorpoPolitica"));
        org.apache.axis.description.ElementDesc          elemField          =          new
org.apache.axis.description.ElementDesc();
        elemField.setFieldName("parametros");
        elemField.setXmlName(new javax.xml.namespace.QName("", "parametros"));
        elemField.setXmlType(new
javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "anyType"));
        typeDesc.addFieldDesc(elemField);
        elemField = new org.apache.axis.description.ElementDesc();
        elemField.setFieldName("politica");
        elemField.setXmlName(new javax.xml.namespace.QName("", "politica"));
        elemField.setXmlType(new
javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/", "string"));
        typeDesc.addFieldDesc(elemField);
    }
    /**
     * Retorna Metadados
     */
    public static org.apache.axis.description.TypeDesc getTypeDesc() {
        return typeDesc;
    }

    /**
     * Adapta a Serialização
     */
    public static org.apache.axis.encoding.Serializer getSerializer(
           java.lang.String mechType,
```

```
            java.lang.Class _javaType,
            javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanSerializer(
            _javaType, _xmlType, typeDesc);
    }
    /**
     * Desadapta a Deserialização
     */
    public static org.apache.axis.encoding.Deserializer getDeserializer(
            java.lang.String mechType,
            java.lang.Class _javaType,
            javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanDeserializer(
            _javaType, _xmlType, typeDesc);
    }
}
```

Anexo C

Fontes da criação do Corpo da Requisição SOAP de requisição. Também são encontrados os comentários.

```
/**
 * CorpoRequisicao.java
 *
 * Este arquivo foi gerado pelo WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST) WSDL2Java emitter.
 */
package mensagens;

import java.util.Calendar;

public class CorpoRequisicao  implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private java.lang.Object[] parametros;
    private java.lang.Long prioridade;
    private java.util.Calendar tempo;
    private java.lang.String tipoEvento;

    public CorpoRequisicao() {
    }
    public CorpoRequisicao(
            java.lang.Object[] parametros,
            java.lang.Long prioridade,
            java.util.Calendar tempo,
            java.lang.String tipoEvento) {
            this.parametros = parametros;
            this.prioridade = prioridade;
            this.tempo = tempo;
            this.tipoEvento = tipoEvento;
    }
    /**
```

```
 * Pega os valores dos parametros do CorpoRequisicao.
 *
 * @return parametros
 */
public java.lang.Object[] getParametros() {
    return parametros;
}
/**
 * Grupo de valores dos parametros do CorpoRequisicao.
 *
 * @param parametros
 */
public void setParametros(java.lang.Object[] parametros) {
    this.parametros = parametros;
}
/**
 * Pega os valores das propriedades do CorpoRequisicao.
 *
 * @return prioridade
 */
public java.lang.Long getPrioridade() {
    return prioridade;
}
/**
 * Grupo de valores das propriedades do CorpoRequisicao.
 *
 * @param prioridade
 */
public void setPrioridade(java.lang.Long prioridade) {
    this.prioridade = prioridade;
}
/**
 * Pega o valor tempo do CorpoRequisicao.
 *
 * @return tempo
 */
public java.util.Calendar getTempo() {
    return tempo;
}
/**
 * Pega o valor tempo do CorpoRequisicao.
 *
 * @param tempo
 */
public void setTempo(Calendar tempo) {
    this.tempo = tempo;
}
/**
 * Pega o valor tipoEvento do CorpoRequisicao.
 *
 * @return tipoEvento
 */
public java.lang.String getTipoEvento() {
    return tipoEvento;
}
/**
```

```
 * Pega o valor tipoEvento do CorpoRequisição
 *
 * @param tipoEvento
 */
public void setTipoEvento(java.lang.String tipoEvento) {
    this.tipoEvento = tipoEvento;
}
private java.lang.Object __equalsCalc = null;
public synchronized boolean equals(java.lang.Object obj) {
    if (!(obj instanceof CorpoRequisicao)) return false;
    CorpoRequisicao other = (CorpoRequisicao) obj;
    if (obj == null) return false;
    if (this == obj) return true;
    if (__equalsCalc != null) {
        return (__equalsCalc == obj);
    }
    __equalsCalc = obj;
    boolean _equals;
    _equals = true &&
        ((this.parametros==null && other.getParametros()==null) ||
         (this.parametros!=null &&
          java.util.Arrays.equals(this.parametros, other.getParametros()))) &&
        ((this.prioridade==null && other.getPrioridade()==null) ||
         (this.prioridade!=null &&
          this.prioridade.equals(other.getPrioridade()))) &&
        ((this.tempo==null && other.getTempo()==null) ||
         (this.tempo!=null &&
          this.tempo.equals(other.getTempo()))) &&
        ((this.tipoEvento==null && other.getTipoEvento()==null) ||
         (this.tipoEvento!=null &&
          this.tipoEvento.equals(other.getTipoEvento())));
    __equalsCalc = null;
    return _equals;
}
private boolean __hashCodeCalc = false;
public synchronized int hashCode() {
    if (__hashCodeCalc) {
        return 0;
    }
    __hashCodeCalc = true;
    int _hashCode = 1;
    if (getParametros() != null) {
        for (int i=0;
             i<java.lang.reflect.Array.getLength(getParametros());
             i++) {
            java.lang.Object obj = java.lang.reflect.Array.get(getParametros(), i);
            if (obj != null &&
                !obj.getClass().isArray()) {
                _hashCode += obj.hashCode();
            }
        }
    }
    if (getPrioridade() != null) {
        _hashCode += getPrioridade().hashCode();
    }
    if (getTempo() != null) {
```

```
            _hashCode += getTempo().hashCode();
        }
        if (getTipoEvento() != null) {
            _hashCode += getTipoEvento().hashCode();
        }
        __hashCodeCalc = false;
        return _hashCode;
    }

    // Tipo dos Metadados
    private static org.apache.axis.description.TypeDesc typeDesc =
        new org.apache.axis.description.TypeDesc(CorpoRequisicao.class, true);
    static {
        typeDesc.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"CorpoRequisicao"));
        org.apache.axis.description.ElementDesc elemField = new
org.apache.axis.description.ElementDesc();
        elemField.setFieldName("parametros");
        elemField.setXmlName(new javax.xml.namespace.QName("", "parametros"));
        elemField.setXmlType(new
javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "anyType"));
        typeDesc.addFieldDesc(elemField);
        elemField = new org.apache.axis.description.ElementDesc();
        elemField.setFieldName("prioridade");
        elemField.setXmlName(new javax.xml.namespace.QName("", "prioridade"));
        elemField.setXmlType(new
javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/", "long"));
        typeDesc.addFieldDesc(elemField);
        elemField = new org.apache.axis.description.ElementDesc();
        elemField.setFieldName("tempo");
        elemField.setXmlName(new javax.xml.namespace.QName("", "tempo"));
        elemField.setXmlType(new
javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "dateTime"));
        typeDesc.addFieldDesc(elemField);
        elemField = new org.apache.axis.description.ElementDesc();
        elemField.setFieldName("tipoEvento");
        elemField.setXmlName(new javax.xml.namespace.QName("", "tipoEvento"));
        elemField.setXmlType(new
javax.xml.namespace.QName("http://schemas.xmlsoap.org/soap/encoding/", "string"));
        typeDesc.addFieldDesc(elemField);
    }
    /**
     * Retorna os Metadados
     */
    public static org.apache.axis.description.TypeDesc getTypeDesc() {
        return typeDesc;
    }
    /**
     * Adapta a Serialização
     */
    public static org.apache.axis.encoding.Serializer getSerializer(
           java.lang.String mechType,
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanSerializer(
```

```
            _javaType, _xmlType, typeDesc);
    }
    /**
     * Desadapta a Deserialização
     *
     */
    public static org.apache.axis.encoding.Deserializer getDeserializer(
           java.lang.String mechType,
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanDeserializer(
            _javaType, _xmlType, typeDesc);
    }
}
```

Anexo D

Fontes da criação do envelope SOAP da polític, também são encontrados os comentários.

```
/**
 * EnvelopePolitica.java
 *
 * Este arquivo foi gerado do WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST) WSDL2Java emitter.
 */
package mensagens;
public class EnvelopePolitica  implements java.io.Serializable {
    private static final long serialVersionUID = 1L;

    private mensagens.Cabecalho cabecalho;
    private mensagens.CorpoPolitica corpo;

    public EnvelopePolitica() {
    }

    public EnvelopePolitica(
           mensagens.Cabecalho cabecalho,
           mensagens.CorpoPolitica corpo) {
           this.cabecalho = cabecalho;
           this.corpo = corpo;
    }
    /**
     * Pega os valores do cabeçalho do EnvelopePolitica.
     *
     * @return cabecalho
     */
    public mensagens.Cabecalho getCabecalho() {
        return cabecalho;
    }
    /**
     * Pega os valores do cabeçalho do EnvelopePolitica.
     *
```

```
     * @param cabecalho
     */
    public void setCabecalho(mensagens.Cabecalho cabecalho) {
        this.cabecalho = cabecalho;
    }
    /**
     * Pega os valores do corpo do EnvelopePolitica.
     *
     * @return corpo
     */
    public mensagens.CorpoPolitica getCorpo() {
        return corpo;
    }
    /**
     * Grupo de valores do coro  do EnvelopePolitica.
     *
     * @param corpo
     */
    public void setCorpo(mensagens.CorpoPolitica corpo) {
        this.corpo = corpo;
    }
    private java.lang.Object __equalsCalc = null;
    public synchronized boolean equals(java.lang.Object obj) {
        if (!(obj instanceof EnvelopePolitica)) return false;
        EnvelopePolitica other = (EnvelopePolitica) obj;
        if (obj == null) return false;
        if (this == obj) return true;
        if (__equalsCalc != null) {
            return (__equalsCalc == obj);
        }
        __equalsCalc = obj;
        boolean _equals;
        _equals = true && ((this.cabecalho==null && other.getCabecalho()==null) ||
             (this.cabecalho!=null && this.cabecalho.equals(other.getCabecalho()))) &&
((this.corpo==null && other.getCorpo()==null) || (this.corpo!=null &&
              this.corpo.equals(other.getCorpo())));
        __equalsCalc = null;
        return _equals;
    }
    private boolean __hashCodeCalc = false;
    public synchronized int hashCode() {
        if (__hashCodeCalc) {
            return 0;
        }
        __hashCodeCalc = true;
        int _hashCode = 1;
        if (getCabecalho() != null) {
            _hashCode += getCabecalho().hashCode();
        }
        if (getCorpo() != null) {
            _hashCode += getCorpo().hashCode();
        }
        __hashCodeCalc = false;
        return _hashCode;
    }
    // Type metadata
```

```
    private static org.apache.axis.description.TypeDesc typeDesc =
        new org.apache.axis.description.TypeDesc(EnvelopePolitica.class, true);
    static {
        typeDesc.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"EnvelopePolitica"));
        org.apache.axis.description.ElementDesc elemField = new
org.apache.axis.description.ElementDesc();
        elemField.setFieldName("cabecalho");
        elemField.setXmlName(new javax.xml.namespace.QName("", "cabecalho"));
        elemField.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"Cabecalho"));
        typeDesc.addFieldDesc(elemField);
        elemField = new org.apache.axis.description.ElementDesc();
        elemField.setFieldName("corpo");
        elemField.setXmlName(new javax.xml.namespace.QName("", "corpo"));
        elemField.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"CorpoPolitica"));
        typeDesc.addFieldDesc(elemField);
    }

    /**
     * Retorna o metadados
    */
    public static org.apache.axis.description.TypeDesc getTypeDesc() {
        return typeDesc;
    }
    /**
     * Adapta a serialização
     */
    public static org.apache.axis.encoding.Serializer getSerializer(
           java.lang.String mechType,
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanSerializer(
            _javaType, _xmlType, typeDesc);
    }
    /**
     * Desadapta Deserialização
     */
    public static org.apache.axis.encoding.Deserializer getDeserializer(
           java.lang.String mechType,
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanDeserializer(
            _javaType, _xmlType, typeDesc);
    }
}
/**
```

Anexo E

Fontes da criação do envelope SOAP de requisição, também são encontrados os comentários.

* **EnvelopeRequisicao.java**

```
 *
 * Este arquivo foi gerando pelo WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST) WSDL2Java emitter.
 */
package mensagens;
public class EnvelopeRequisicao  implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private mensagens.Cabecalho cabecalho;
    private mensagens.CorpoRequisicao corpo;
    public EnvelopeRequisicao() {
    }
    public EnvelopeRequisicao(
           mensagens.Cabecalho cabecalho,
           mensagens.CorpoRequisicao corpo) {
           this.cabecalho = cabecalho;
           this.corpo = corpo;
    }
    /**
     * Pega os valores do Cabeçalho para o EnvelopeRequisicao
     *
     * @return cabecalho
     */
    public mensagens.Cabecalho getCabecalho() {
        return cabecalho;
    }
    /**
     * Grupo de valores cabeçalho para este EnvolopeRequisicao
     *
     * @param cabecalho
     */
    public void setCabecalho(mensagens.Cabecalho cabecalho) {
        this.cabecalho = cabecalho;
    }
    /**
     * Pega os valores do corpo da EnvelopeRequisicao
     *      *
     * @return corpo
     */
    public mensagens.CorpoRequisicao getCorpo() {
        return corpo;
    }
    /**
     * Pega os valores do corpo da EnvelopeRequisicao
     *
     * @param corpo
     */
    public void setCorpo(mensagens.CorpoRequisicao corpo) {
        this.corpo = corpo;
    }
    private java.lang.Object __equalsCalc = null;
    public synchronized boolean equals(java.lang.Object obj) {
```

```
        if (!(obj instanceof EnvelopeRequisicao)) return false;
        EnvelopeRequisicao other = (EnvelopeRequisicao) obj;
        if (obj == null) return false;
        if (this == obj) return true;
        if (__equalsCalc != null) {
            return (__equalsCalc == obj);
        }
        __equalsCalc = obj;
        boolean _equals;
        _equals = true &&
            ((this.cabecalho==null && other.getCabecalho()==null) ||
             (this.cabecalho!=null &&
              this.cabecalho.equals(other.getCabecalho()))) &&
            ((this.corpo==null && other.getCorpo()==null) ||
             (this.corpo!=null &&
              this.corpo.equals(other.getCorpo())));
        __equalsCalc = null;
        return _equals;
    }

    private boolean __hashCodeCalc = false;
    public synchronized int hashCode() {
        if (__hashCodeCalc) {
            return 0;
        }
        __hashCodeCalc = true;
        int _hashCode = 1;
        if (getCabecalho() != null) {
            _hashCode += getCabecalho().hashCode();
        }
        if (getCorpo() != null) {
            _hashCode += getCorpo().hashCode();
        }
        __hashCodeCalc = false;
        return _hashCode;
    }
    // Type metadata
    private static org.apache.axis.description.TypeDesc typeDesc =
        new org.apache.axis.description.TypeDesc(EnvelopeRequisicao.class, true);
    static {
        typeDesc.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"EnvelopeRequisicao"));
        org.apache.axis.description.ElementDesc elemField = new
org.apache.axis.description.ElementDesc();
        elemField.setFieldName("cabecalho");
        elemField.setXmlName(new javax.xml.namespace.QName("", "cabecalho"));
        elemField.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"Cabecalho"));
        typeDesc.addFieldDesc(elemField);
        elemField = new org.apache.axis.description.ElementDesc();
        elemField.setFieldName("corpo");
        elemField.setXmlName(new javax.xml.namespace.QName("", "corpo"));
        elemField.setXmlType(new javax.xml.namespace.QName("http://mensagens",
"CorpoRequisicao"));
        typeDesc.addFieldDesc(elemField);
    }
```

```
    /**
     * Retorna os metadados
     *
     */
    public static org.apache.axis.description.TypeDesc getTypeDesc() {
        return typeDesc;
    }
    /**
     * Adapta a serialização
     */
    public static org.apache.axis.encoding.Serializer getSerializer(
           java.lang.String mechType,
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanSerializer(
            _javaType, _xmlType, typeDesc);
    }
    /**
     * Desadapta a deserialização
         */
    public static org.apache.axis.encoding.Deserializer getDeserializer(
           java.lang.String mechType,
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType) {
        return
          new  org.apache.axis.encoding.ser.BeanDeserializer(
            _javaType, _xmlType, typeDesc);
    }
}
```