UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

THEO FERREIRA FRANCO

Uma Arquitetura baseada em Políticas para o provimento de QoS utilizando princípios de *Autonomic Computing*

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profa. Dra. Maria Janilce Bosquoli Almeida
Orientador

Prof. Dr. Lisandro Zambenedetti Granville
Co-orientador

Porto Alegre, março de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Franco, Theo Ferreira

Uma Arquitetura baseada em Políticas para o provimento de QoS utilizando princípios de *Autonomic Computing* / Theo Ferreira Franco. – Porto Alegre: PPGC da UFRGS, 2008.

71 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Orientador: Maria Janilce Bosquoli Almeida; Co-orientador: Lisandro Zambenedetti Granville.

1. Gerenciamento de redes. 2. Autonomic computing. 3. Redes *peer-to-peer*. 4. Gerenciamento de redes baseado em políticas. 5. Computação em grade. I. Almeida, Maria Janilce Bosquoli. II. Granville, Lisandro Zambenedetti. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Prof[a]. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenadora do PPGC: Prof[a]. Luciana Porcher Nedel
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

“Science is the poetry of reality.”
— RICHARD DAWKINS

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AC	Autonomic Computing
AF	Assured Forwarding
AM	Autonomic Manager
API	Application Programming Interface
BE	Best Effort
CIM	Core Information Model
COPS	Common Open Policy Service
COPS-PR	COPS for Policy Provisioning
CoS	Class of Service
DiffServ	Differentiated Services
DMTF	Distributed Management Task Force
DoS	Denial of Service
EF	Expedited Forwarding
GT4	Globus Toolkit 4
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
JMS	Java Message Service
JMX	Java Management Extensions
JSDL	Job Submission Description Language
MDS	Monitoring and Discovery Services
MR	Managed Resource
NAS	Network-Attached Storage
NAT	Network Address Translation
P2P	Peer-to-Peer
PBNM	Policy-Based Network Management
PCIM	Policy Core Information Model

PCIMe	PCIM Extensions
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PHB	Per-Hop Behavior
PMAC	Policy Management for Autonomic Computing
QoS	Quality of Service
RFT	Reliable File Transfer Service
SGBD	Sistema de Gerenciamento de Banco de Dados
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TI	Tecnologia da Informação
VoIP	Voz sobre IP
VPN	Virtual Private Network
WSDL	Web Services Description Language
WS-Notification	Web Services Notification

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Sistemas corporativos modernos cada vez mais dependentes da rede e a integração de serviços entorno do modelo TCP/IP elevam a exigência de Qualidade de Serviço da infra-estrutura de TI. Neste cenário, o dinamismo das redes atuais em conjunto com os novos requisitos de QoS exigem que a infra-estrutura de TI seja mais autônoma e confiável. Para tratar esta questão, o modelo de Gerenciamento de Redes Baseado em Políticas, proposto pelo IETF, vem se consolidando como uma abordagem para controlar o comportamento da rede através do controle das configurações dos seus dispositivos. Porém, o foco deste modelo é o gerenciamento de políticas internas a um domínio administrativo. Esta característica faz com que o modelo possua algumas limitações, tais como a incapacidade de estabelecer qualquer tipo de coordenação entre diferentes PDPs e a impossibilidade de reagir a eventos externos. Visando agregar autonomia ao modelo de gerenciamento baseado em políticas, este trabalho propõe uma arquitetura em camadas que empregue os conceitos de *Autonomic Computing* relacionados a: i) adaptação dinâmica dos recursos gerenciados em resposta às mudanças no ambiente, ii) integração com sistemas de gerenciamento de outros domínios através do recebimento de notificações destes, iii) capacidade de planejar ações de gerenciamento e iv) promoção de ações de gerenciamento que envolvam mais de um domínio administrativo, estabelecendo uma espécie de coordenação entre PDPs. Para a implementação destes conceitos, a arquitetura prevê o uso de uma camada *peer-to-peer* (P2P) sobre a plataforma de políticas. Desta forma, a partir de uma notificação recebida, a camada P2P planeja ações visando adaptar o comportamento da rede aos eventos ocorridos na infra-estrutura de TI. As ações planejadas traduzem-se em inclusões ou remoções de políticas da plataforma de políticas responsável por gerenciar a configuração dos dispositivos de rede. Para notificações que envolvam recursos de mais de um domínio administrativo, os *peers* de gerenciamento agem de forma coordenada para implantar as devidas ações em cada domínio. A arquitetura proposta foi projetada com foco em prover QoS em uma rede com suporte à DiffServ, embora acredite-se que a sua estrutura seja genérica o bastante para ser aplicada a outros contextos. Como estudo de caso, foi analisado o emprego da arquitetura em resposta a eventos gerados por uma grade computacional. Foi elaborado ainda um protótipo da arquitetura utilizando o Globus Toolkit 4 como fonte de eventos.

Palavras-chave: Gerenciamento de redes, autonomic computing, redes *peer-to-peer*, gerenciamento de redes baseado em políticas, computação em grade.

A Policy-Based Architecture for QoS Provisioning using Autonomic Computing Principles

ABSTRACT

Modern corporative systems becoming more dependent of the network and the integration of services around the TCP/IP model increase the requirement of Quality of Service (QoS) of the IT infrastructure. In this scene, the dynamism of current networks together with the new requirements of QoS demands a more autonomous and reliable IT infrastructure. To address this issue, the model of Police Based Network Management, proposed by IETF, has been consolidated as an approach to control the behavior of the network through the control of the configurations of its devices. However, the focus of this model is the management of the policies internal to an administrative domain. This feature brings some limitations to the model, such as the incapacity to establish any kind of coordination between different PDPs and the impossibility to react to external events. Aiming at to add autonomy to the model of Policy Based Network Management, this work proposes a layered architecture based on the concepts of Autonomic Computing related to: i) the dynamic adaptation of the managed resources in response to changes in the environment, ii) integration with management systems of other domains through the reception of notifications of these systems, iii) ability of planning the management actions and iv) execution of multi-domain management actions, establishing a kind of coordination between PDPs. To implement these concepts, the architecture was designed with a peer-to-peer layer above the policy platform. Thus, from a received notification, the P2P layer plans actions aiming to adapt the network behavior in response to the events occurred in the IT infrastructure. The planned actions are, actually, inclusions or removals of policies in the policy platform responsible for the management of the network devices configuration. For notifications related with resources of more than one administrative domain, the management peers act in a coordinated way in order to establish the suitable actions in each domain. The proposed architecture was designed with focus in providing QoS in a network with support to DiffServ, although we believe that its structure is generic enough to be applied to other contexts. As case study, it was analyzed the use of the architecture in response to events generated by a computational grid. Additionally, a prototype of the architecture was build making use of Globus Toolkit 4 as an event source.

Keywords: Network Management, Autonomic Computing, Peer-to-Peer Networks, Policy Based Network Management, Grid Computing.

1 INTRODUÇÃO

Na medida em que o ambiente de rede torna-se cada vez mais heterogêneo, dinâmico e interconectado, especialmente devido à popularização de novas tecnologias *wireless*, seu gerenciamento também torna-se mais complexo. Além disso, sistemas corporativos cada vez mais dependentes da rede e a integração de serviços entorno do modelo TCP/IP, como Voz sobre IP (VoIP) e videoconferência, elevam a exigência de Qualidade de Serviço (*Quality of Service* - QoS) da infra-estrutura de Tecnologia da Informação (TI). Neste cenário, o dinamismo das redes atuais em conjunto com os novos requisitos de QoS exigem que a infra-estrutura de gerenciamento de TI seja mais autônoma e confiável, reduzindo a intervenção humana na operação da rede. Redes que não possuem gerenciamento consistente geralmente apresentam problemas de desempenho e baixa disponibilidade, o que freqüentemente acarreta em perdas financeiras às corporações.

Para tratar esta questão, o *framework* de Gerenciamento de Redes Baseado em Políticas (*Policy-Based Network Management* - PBNM) (MOORE et al., 2001) (WESTERINEN et al., 2001) vem se consolidando como uma abordagem útil para estabelecer um conjunto de regras visando controlar o comportamento da rede através do controle das configurações dos seus dispositivos. Os principais benefícios da utilização de políticas são a escalabilidade e a flexibilidade no gerenciamento de sistemas distribuídos (DAMIANOU, 2002). A escalabilidade é alcançada através da aplicação de uma mesma política a um grande conjunto de dispositivos e objetos, enquanto a flexibilidade é alcançada pela abstração proporcionada através do uso de políticas. Políticas podem ser modificadas dinamicamente, tornando possível alterar o comportamento do ambiente gerenciado, sem modificar sua implementação ou interromper sua operação.

O gerenciamento baseado em políticas tem sido adotado de forma abrangente, principalmente em domínios como QoS e segurança. Os usos típicos de políticas incluem: i) configuração de dispositivos, como mecanismo de gerenciamento de filas, estratégias de descarte, listas de filtros de acesso; ii) classificação de tráfego; iii) reserva de recursos e controle de admissão baseados em parâmetros, como identidade de usuários, tipo de aplicação; e iv) estabelecimento de Service Level Agreement (SLA) entre domínios adjacentes.

A arquitetura de gerenciamento baseado em políticas tradicional, proposta pelo Internet Engineering Task Force (IETF) (MOORE et al., 2001), possui foco na definição e gerenciamento de políticas intra-domínio para um conjunto heterogêneo de dispositivos. O termo "políticas intra-domínio" refere-se às políticas relacionadas a recursos pertencentes a um mesmo domínio administrativo. Esta característica faz com que este modelo possua algumas limitações, tais como a incapacidade de comunicação e, conseqüentemente, incapacidade de estabelecer algum tipo de coordenação entre diferentes Policy Decision Points (PDP). Isto porque os protocolos propostos pelo grupo de trabalho sobre

políticas do IETF, aplicados ao gerenciamento de políticas intra-domínio, não podem ser utilizados diretamente para o gerenciamento de políticas inter-domínio (ZHENG; GREIS, 2004). Outra limitação do modelo é a impossibilidade de reagir a eventos externos, não permitindo, por exemplo, que um Intrusion Detection System (IDS) envie uma notificação ao sistema de gerenciamento de rede para isolar um determinado segmento. Além disso, a implementação de sistemas de gerenciamento baseados em políticas são tipicamente dependentes da tecnologia destes dispositivos. Isto tem limitado a capacidade de especificar de forma consistente políticas fim-a-fim, diluindo o real valor do gerenciamento baseado em políticas (VATSAVAI; CHAKRAVARTHY; MOHANIA, 2006).

Para transpor estas limitações, este trabalho propõe o uso de uma rede *overlay peer-to-peer* (P2P) que permita o estabelecimento de um comportamento multi-domínio, porém respeitando as políticas locais. Além disso, visando dar autonomia ao sistema de gerenciamento, este trabalho propõe ainda o emprego dos conceitos de Autonomic Computing (AC) (KEPHART, 2005) na implementação desta rede P2P.

AC estabelece um modelo de infra-estrutura de gerenciamento que permite a adaptação dinâmica dos recursos gerenciados em resposta às mudanças no ambiente (KONSTANTINOU, 2003). O objetivo deste modelo é manter o conjunto de recursos gerenciados, como um todo, em um estado consistente. Além disso, um sistema baseado no modelo de AC deve requerer administração mínima e, em sua maioria, no nível de gerenciamento de políticas.

O modelo de AC define conceitualmente um Gerente Autonômico formado por um laço de controle, o qual faz uso de monitores para identificar o estado dos recursos gerenciados e também receber eventos de outros Gerentes Autonômicos. Seguindo esta idéia, neste trabalho foi planejada uma rede *overlay* P2P como uma camada de gerenciamento sensível a eventos gerados por outros sistemas de gerenciamento, tais como gerentes de servidores de aplicações, de armazenamento e de grades computacionais. Esta rede P2P faz uso da plataforma de políticas como camada de suporte.

A motivação para o uso de redes P2P como a camada inteligente da solução está no fato que, ao contrário dos sistemas de gerenciamento tradicionais, as redes P2P são projetadas para permitir a interação entre usuários localizados em diferentes domínios (GRANVILLE et al., 2005). Na medida em que cresce a adoção da Internet como uma rede de trânsito, as ações de gerenciamento que transpassam os limites dos domínios administrativos tornam-se importantes. Por exemplo, para uma vídeo-conferência de alta-qualidade, uma certa largura de banda deve estar corretamente alocada em todos domínios administrativos no caminho entre os usuários finais. Em cada domínio, operadores de rede, com os devidos direitos, configuram os dispositivos do seu domínio de forma apropriada, cada qual utilizando um conjunto de ferramentas para realizar a reserva de banda. Mesmo que as políticas administrativas de cada domínio permitam esta reserva de banda, os operadores de rede destes domínios devem efetuar ações separadamente de forma a configurar os dispositivos internos, sem garantia de atomicidade. Uma rede *overlay* P2P que interaja com os sistemas de gerenciamento baseados em políticas de cada domínio, permite analisar se a ação requerida é permitida pelas políticas atuais e, caso positivo, de forma transacional, implantar políticas em cada domínio, automatizando o processo de configuração da vídeo-conferência.

Outra importante característica das redes P2P é a possibilidade de replicar um serviço em diversos *peers*, criando um serviço único que é suportado por uma rede P2P escalável e auto-organizada. Além disso, redes P2P possuem baixo custo de desenvolvimento, já que plataformas como JXTA (SUN MICROSYSTEMS, 2007a) permitem a publicação

de serviços exigindo poucas configurações. Estas características fornecem um suporte multi-domínio, flexível, confiável e de baixo custo de desenvolvimento.

Como estudo de caso, focou-se no provimento de QoS para computação em grade. Segundo Foster (FOSTER; KESSELMAN, 2003), o objetivo de uma grade computacional é o de agregar um grande conjunto de recursos compartilhados - tais como, computação, comunicação e armazenamento - para construir um ambiente de alto desempenho para aplicações intensivas em dados ou intensivas em computação. Ainda de acordo com Foster (FOSTER, 2002), os três pontos principais de uma grade são: i) os recursos computacionais de uma grade não são gerenciados de forma centralizada; ii) uso de padrões e protocolos abertos e interfaces de propósito geral; iii) requerem QoS não trivial. Um sistema que pretenda gerenciar os recursos utilizados por uma grade computacional deve suportar o monitoramento do estado da grade e, a partir deste monitoramento, tomar ações para garantir a QoS requerida para as aplicações da grade.

Acredita-se que os conceitos de AC relacionados ao monitoramento, análise, planejamento e execução, aplicados a uma rede *overlay* P2P, com o suporte do *framework* de gerenciamento baseado em políticas, podem servir de base para a construção de um sistema de gerenciamento distribuído atendendo a estes requisitos. Como forma de validar a arquitetura de gerenciamento proposta, neste trabalho foi elaborado um protótipo voltado à configuração de QoS em uma rede com suporte à Differentiated Services (DiffServ) (BLAKE et al., 1998), em resposta a eventos recebidos de um *toolkit* para computação em grade.

Este trabalho está organizado da seguinte forma:

- No Capítulo 2 são apresentadas as principais tecnologias envolvidas no trabalho. Durante a descrição de cada tecnologia busca-se identificar a sua relação com a proposta, além de citar trabalhos relevantes sobre o tema.

- Já o Capítulo 3 descreve a arquitetura proposta. Para isto, são descritas as camadas que compõem a solução e a comunicação entre elas. É discutido ainda a questão da execução de ações de gerenciamento que envolvam mais de um domínio administrativo. Por fim, é analisado o emprego da arquitetura proposta na recepção de notificações geradas por uma grade computacional.

- A prototipação do estudo de caso proposto é apresentada no Capítulo 4. Em especial, são descritas a integração com o *toolkit* da grade computacional e a implementação dos *peers* de gerenciamento de rede.

- As conclusões, contribuições deste trabalho e os possíveis trabalhos futuros são apresentados no Capítulo 5.

2 TRABALHOS RELACIONADOS

Este capítulo apresenta as principais tecnologias envolvidas no trabalho. Para cada tecnologia são apresentadas abordagens relevantes e que tenham relação com o trabalho proposto. Além disso, é apresentado um breve relato sobre como cada tecnologia é empregada na arquitetura.

2.1 Gerenciamento de Redes Baseado em Políticas

Políticas são utilizadas essencialmente como um mecanismo para codificar os objetivos corporativos, de forma a adaptar o uso dos recursos da rede a estes objetivos. Os objetivos corporativos devem ser expressos pelo administrador da rede na forma de regras do tipo *Se-Então*. Cabe a plataforma de políticas utilizada implantar efetivamente as regras estabelecidas.

Uma arquitetura de gerenciamento de redes baseado em políticas típica é mostrada na Figura 2.1.

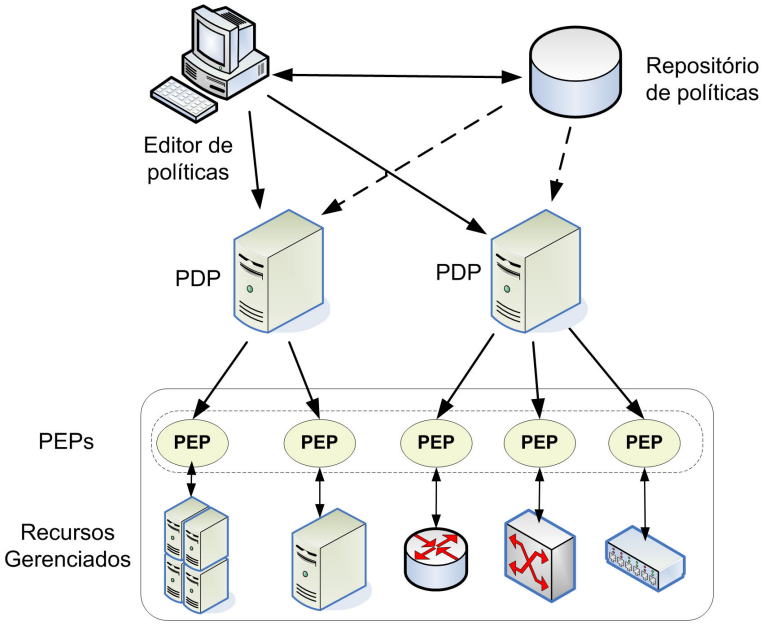

Figura 2.1: Arquitetura de gerenciamento baseado em políticas do IETF

De forma simplificada, a arquitetura proposta pelo IETF (MOORE et al., 2001), a qual tem sido o *framework* mais utilizado nas abordagens atuais, especialmente no domínio de gerenciamento de QoS, é composta por quatro elementos principais: o Policy Decision

Point (PDP), o Policy Enforcement Point (PEP), um repositório e um editor de políticas.

O PDP é o sistema responsável pelo gerenciamento de políticas, podendo também ser chamado de servidor de políticas. O PDP analisa os eventos enviados pelos PEPs controlados por ele, de acordo com o conjunto de regras armazenadas no repositório de políticas. A partir do evento recebido, o PDP reavalia as políticas adotadas e envia as ações a serem tomadas pelos PEPs. Esta comunicação é feita utilizando um protocolo padronizado entre PDP e PEP, tal como o Common Open Policy Service (COPS) (BOYLE et al., 2000), COPS for Policy Provisioning (COPS-PR) (CHAN et al., 2001), Command Line Interface/Telnet, Simple Network Management Protocol (SNMP), ou até NETCONF e SOAP (FRANCO et al., 2006).

Já o PEP é o elemento no qual a política é aplicada. A aplicação das políticas requer características de *hardware* e *software* específicas no elemento de rede, tal como filtragem de pacotes, reserva de banda, prioridade de tráfego e múltiplas filas de saída. Os *frameworks* atuais requerem este conhecimento detalhado dos elementos para determinar a possibilidade de implantação da política.

O editor de políticas provê uma interface ao administrador de rede, de forma a permitir que este insira as políticas desejadas de forma gráfica ou utilizando uma linguagem padrão. Esta ferramenta é responsável também por converter, se necessário, a forma de representação da política para aquela utilizada pelo repositório de políticas e também recuperar uma política do repositório para edição. Este elemento deve ainda, verificar a validade da política inserida, assim como, opcionalmente, verificar possíveis conflitos entre a nova política e as políticas existentes. Para que uma política seja aplicada, a ferramenta deve sinalizar aos PDPs, indicando que estes devem consultar, junto ao repositório de políticas, as informações referentes à política em questão (MARQUEZAN et al., 2005).

Segundo (PONNAPPAN et al., 2002), existem dois modelos principais utilizados no gerenciamento baseado em políticas: *outsourcing* e *provisioning*. No modelo *outsourcing*, quando há ocorrência de um evento no PEP, este delega ao PDP a tarefa de tomar uma decisão de que ação executar (e se deve ser executada alguma ação) na ocorrência deste evento. Por exemplo, este é um caso típico de controle de admissão gerado a partir de uma requisição RSVP por reserva de recursos. Baseado nas políticas relativas aos objetivos corporativos localizados no repositório, contendo a informação de que tipo de usuário/aplicação possui o privilégio de reserva de recursos, o PDP decide se a reserva deve ser efetuada ou não.

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

No trabalho proposto, o gerenciamento baseado em políticas tem um papel fundamental, já que é o *framework* de gerenciamento baseado em políticas que fornece o suporte para a aplicação das políticas na arquitetura. O suporte a políticas foi projetado como

uma camada, a qual possui comunicação com a camada física, composta pelos dispositivos de rede, e a camada P2P, composta pelos *peers* de gerenciamento de rede e o gerente de recursos. Entre a camada P2P e a camada de políticas deve haver apenas interface padronizada, tal como acontece entre o editor de políticas e a plataforma de políticas. Desta forma, na arquitetura proposta, uma camada não tem conhecimento, nem acesso a estrutura interna de outra camada. Assim, a camada de políticas pode implementar tanto o modelo *outsourcing*, quanto o modelo *provisioning*, sem prejuízo para o sistema de gerenciamento em geral.

2.2 Gerenciamento de QoS baseado em políticas

Quando a Internet surgiu, não houve a real preocupação de oferecer QoS na entrega de pacotes. Desta forma, toda Internet foi construída sobre um sistema de *best effort*. Porém, os serviços tradicionais da rede agora competem com aplicações críticas compartilhando os mesmos recursos.

Segundo (HP, 1999), QoS pode ser caracterizado segundo os critérios:

- Atraso: latência introduzida ao fluxo da aplicação através da rede devido ao tempo de propagação, tempo de processamento e eventuais congestionamentos;
- *Jitter*: distorção dos tempos de chegada entre os pacotes comparados aos tempos de transmissão dos pacotes, ou seja, variação do tempo de entrega dos pacotes;
- Perda: perda de um pacote transmitido, usualmente causada por um descarte na rede devido ao congestionamento.

Além disso, o uso da rede por aplicações como VoIP, faz com que seja necessário criar mecanismos de classificação do tráfego, já que os diferentes tipos de tráfego possuem requisitos de atraso, *jitter* e perda diferentes.

Duas abordagens podem ser aplicadas para permitir que aplicações com maiores restrições de qualidade compartilhem a rede: superestimar os recursos da rede ou adotar técnicas de QoS, tais como DiffServ (BLAKE et al., 1998). Superestimar os recursos da rede, porém, além de ser uma solução custosa, não garante que os requisitos de QoS serão atendidos, uma vez que não há a implantação de precedência ou priorização no encaminhamento de pacotes. Já o uso de técnicas de QoS permite priorizar tráfegos essenciais da corporação em detrimento de fluxos menos sensíveis a atraso e perda.

A técnica Diffserv utiliza a marcação de pacotes como forma de indicar o tipo de tratamento (classe de serviço) a ser dado a um conjunto de fluxos de dados. A esta classificação e posterior tratamento diferenciado é dado o nome de *Per-Hop Behavior* (PHB). Alguns PHBs têm sido padronizados (por exemplo, *Expedited Forwarding*), já outros podem ser definidos livremente pelos administradores. Técnicas de escalonamento de filas, *traffic shaping* e controle de admissão têm sido utilizadas pelos elementos da rede para prover os comportamentos suportados. Porém, arquiteturas que implementam QoS podem efetivamente garantir qualidade de serviço somente se estiverem adequadamente configuradas e monitoradas (GRANVILLE; TAROUCO, 2001).

Neste cenário, além do gerenciamento dos tradicionais elementos de rede (roteadores, *switches*, serviços, etc.), os administradores devem também gerenciar aspectos de QoS. Assim, a tarefa de gerenciamento de redes de maior porte torna-se mais complexa. O *framework* de gerenciamento baseado em políticas, descrito na Seção 2.1, é uma das

abordagens que busca minimizar os custos de configuração de QoS. Isto porque, no gerenciamento baseado em políticas, as definições sobre QoS, tal como marcação de tráfego, escalonamento, estratégia de descarte, *traffic shaping*, são feitas no nível de políticas, sendo estas políticas aplicadas de forma automatizada aos dispositivos de rede.

A arquitetura proposta neste trabalho possibilita atingir um nível superior de automatização da configuração de QoS. Isto porque adiciona dinamismo ao conjunto de políticas implantadas pela plataforma de políticas. O Capítulo 3 descreve a aplicação da arquitetura proposta para configuração de QoS em redes com suporte a DiffServ, em resposta a eventos externos.

2.3 Autonomic Computing

AC foi proposta (KEPHART; CHESS, 2003) visando possibilitar que o ambiente corporativo de TI opere de forma inteligente e dinâmica, baseando suas decisões em políticas e nos requisitos dos serviços oferecidos. O foco desta abordagem está na redução de custo e complexidade da gerência da infra-estrutura de TI.

Um sistema autônomo deve possuir as seguintes características:

i) *Auto-configuração*: capacidade do sistema de adaptar-se automaticamente a mudanças no ambiente, de forma que a própria arquitetura e/ou os componentes desta arquitetura possam alterar-se para conduzir o sistema a um determinado estado;

ii) *Auto-proteção*: funcionalidades que possibilitam ao sistema antecipadamente detectar defeitos e protegê-lo contra ataques;

iii) *Auto-recuperação*: possibilidade de descobrir, diagnosticar e reagir a falhas. O principal objetivo da auto-recuperação é maximizar a disponibilidade, sustentabilidade e tolerância à falhas;

iv) *Auto-otimização*: monitorar e eficientemente maximizar a alocação de recursos satisfazendo os requisitos de diferentes usuários. Alocação de recursos e gerenciamento de carga são dois aspectos importantes para esta característica.

Para possuir estas características, o gerenciamento de tal sistema deve possuir nível de automação elevado e interagir com outros sistemas de gerenciamento (KONSTANTINOU, 2003). As atuais arquiteturas de gerenciamento, inclusive as baseadas em políticas, não provêem tais características fundamentais para um sistema ser considerado autônomo. No ambiente de rede, por exemplo, o resultado disto está no fato que as redes atuais continuam dependendo de procedimentos operacionais efetuados por administradores de forma empírica. O alto custo associado ao gerenciamento manual é uma significante barreira à escalabilidade do investimento em tecnologia.

Na verdade, AC fornece apenas um conjunto de princípios e um modelo conceitual. Acredita-se que sistemas com arquitetura inspirada nos princípios e no modelo conceitual de AC terão fortes características de autonomia. Algumas abordagens utilizam políticas em conjunto com AC como base para suas arquiteturas. A estas plataformas deu-se o nome de Policy Management for Autonomic Computing (PMAC). Dentre elas, uma das mais relevantes é a proposta (I.B.M., 2007), a qual define um *framework* para autogerenciamento de sistemas de TI.

Esta arquitetura de AC apresenta dois componentes principais:

- Autonomic Manager (AM), o qual monitora os recursos computacionais, analisa o estado destes recursos, planeja ações para estes recursos e executa as ações planejadas. Um AM possui sensores e atuadores para obter informações e enviar comandos

aos recursos gerenciados. Possui ainda sensores e atuadores para interagir com outros AM.

- Managed Resource (MR), elemento do sistema gerenciado pelo AM.

A Figura 2.2a apresenta os componentes que integram um AM. Já a Figura 2.2b ilustra um exemplo de interação entre AMs.

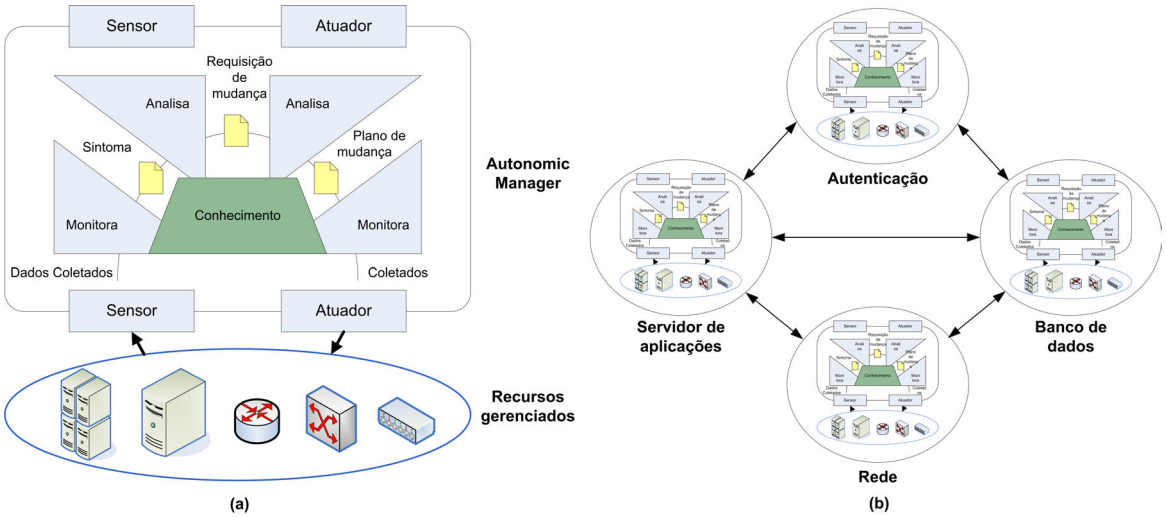

Figura 2.2: a) Funções típicas de um AM; b) Exemplo de interação entre AMs.

A comunicação entre um AM e um MR é feita através da interface de gerenciamento do MR, o qual expõe dois tipos de acesso, sensores e atuadores. Os sensores são utilizados pelo AM para leitura do estado interno do MR, enquanto os atuadores são utilizados pelo AM para invocar ações no MR.

Na abordagem descrita em (I.B.M., 2007), as ações planejadas pelo AM são baseados em políticas. Desta forma, o papel do AM é similar ao papel do PDP no *framework* de políticas (WESTERINEN et al., 2001). De fato, um AM inclui a funcionalidade de um PDP, porém suporta características adicionais, como: monitoramento de estado, correlação entre eventos e notificações. Além disso, existe uma forte semelhança entre a estrutura de um MR e de um PEP. O PEP, porém, não prevê a existência de uma interface para leitura do seu estado interno.

A abordagem proposta neste trabalho difere da arquitetura PMAC. A arquitetura PMAC substitui o *framework* tradicional de gerenciamento baseado em políticas por uma estrutura semelhante, porém com funcionalidades adicionais. Já na arquitetura proposta neste trabalho, não há alteração significativa no *framework* de políticas. Funcionalidades adicionais são agregadas na rede *overlay* P2P, a qual é organizada como camada acima da camada de políticas. Desta forma, é possível agregar novas funcionalidades à soluções existentes baseadas no *framework* de políticas.

2.4 Redes P2P aplicadas ao gerenciamento de redes

Atualmente existem inúmeras redes P2P, utilizadas para os mais diversos fins. Os tipos de aplicação P2P mais usuais são: compartilhamento de arquivos (e.g., eDonkey, BitTorrent, Gnutella e Kad), processamento distribuído (e.g., Seti@Home e Folding@Home) e comunicação e colaboração (e.g., Jabber, Icq, Msn, Skype e Microsoft Office Groove).

Segundo (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), as redes P2P podem ser definidas como:

> Sistemas distribuídos compostos por nodos interconectados capazes de se auto-organizar em topologias de rede com o propósito de compartilharem recursos, tais como conteúdo, ciclos de CPU, armazenamento e largura de banda, capazes de se adaptarem a falhas e acomodar populações transientes de nodos enquanto mantém conectividade e desempenho aceitáveis, sem requerer a intermediação ou suporte de um servidor ou autoridade global centralizada.

Os nodos de *software* que compõem as redes P2P - chamados *peers* - estão dispostos em *hosts* geralmente localizados em diversos domínios administrativos. Cada *peer* comunica-se com outros *peers*, diretamente ou indiretamente, formando sistemas robustos sobre a Internet (GRANVILLE et al., 2005).

O modelo de comunicação P2P claramente difere do modelo cliente-servidor. No modelo cliente-servidor, os clientes acessam os recursos armazenados no servidor. Já no modelo P2P, cada *peer* é um potencial repositório, o qual pode compartilhar seus recursos com outros *peers*. Além disso, os protocolos utilizados nas redes P2P fazem com que estas mantenham-se operacionais mesmo com a entrada e saída de nodos, tornando este modelo naturalmente mais dinâmico e flexível. Outro exemplo de flexibilidade do modelo P2P está na forma de roteamento das mensagens. Enquanto no modelo cliente-servidor as mensagens são roteadas pelo nível de rede (Figura 2.3a), as mensagens P2P são roteadas pelo nível de aplicação por *peers* que operam como roteadores (Figura 2.3b), independente da topologia física da rede.

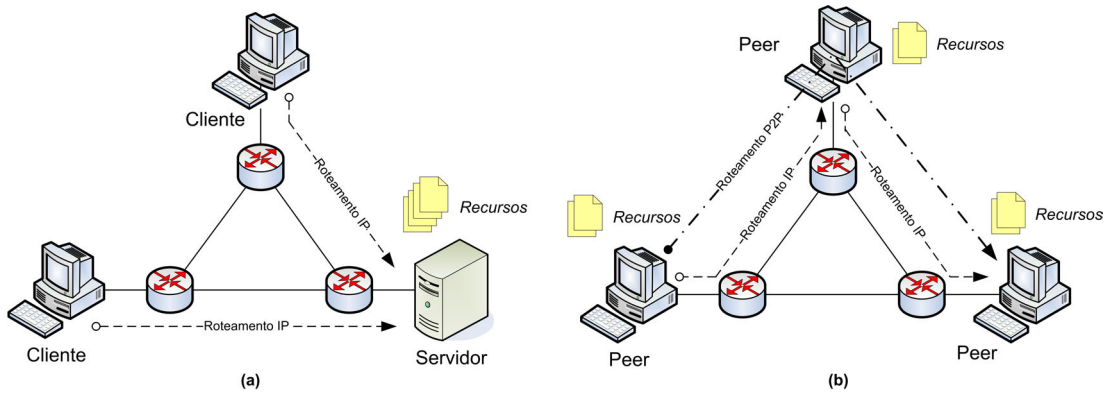

Figura 2.3: a) Roteamento cliente servidor e b) Roteamento P2P

Diversas características não são inerentes ao modelo P2P, mas sim proporcionadas especificamente por implementações deste modelo. Por exemplo, a plataforma JXTA possui uma característica importante, que é a possibilidade de organizar *peers* que fornecem o mesmo conjunto de serviços em grupos. Isto permite que um mesmo serviço possa ser fornecido por um grupo de *peers*. Fazendo uma relação com o modelo cliente-servidor, seria como se houvesse um *cluster* de servidores fornecendo um mesmo serviço, de forma transparente para o cliente. Esta característica permite balancear a carga e aumentar a confiabilidade, além de tornar o sistema escalável, na medida que a expansão da sua capacidade depende apenas da inserção de novos *peers*.

Outras motivações para a criação de grupos são (BROOKSHIER et al., 2002):

- Criar um ambiente seguro - Grupos criam um domínio local ao qual podem ser aplicadas políticas de segurança específicas. Isto permite que os grupos restrinjam acesso a conteúdos protegidos aos membros autenticados.

- Criar escopos - Grupos permitem a criação de domínios especializados com propósitos definidos, dividindo a rede em regiões abstratas. Esta divisão permite otimizar buscas por conteúdo ou serviços, na medida em que estas podem ser feitas apenas em grupos relevantes, e não em toda rede.

Algumas abordagens visam empregar estas características do modelo P2P para a construção de sistemas de gerenciamento de rede distribuídos. (STATE; FESTOR, 2003) apresentaram uma das primeiras iniciativas neste sentido. Os autores propuseram um *framework* que utiliza uma rede P2P para a integração dos componentes da infra-estrutura de gerenciamento distribuída, visando a monitoração e configuração de dispositivos móveis. A arquitetura proposta é totalmente baseada em Java, sendo a estrutura interna dos componentes implementada utilizando Java Management Extensions (JMX) (SUN MICROSYSTEMS, 2007b) e a estrutura de comunicação P2P entre estes componentes implementada sobre a plataforma JXTA.

(STATE; FESTOR, 2003) acreditam que o gerenciamento de ambientes altamente dinâmicos, como as redes *wireless*, pode ser melhor realizado por uma arquitetura que utilize o conceito de P2P, ao invés do modelo tradicional de gerente/agente. Assim, propõem uma solução inovadora, na qual ao invés de agentes, como no modelo tradicional, os dispositivos gerenciados possuem *peers* como interface de gerenciamento. Os *peers* que desejem ser gerenciados é que anunciam a sua interface de gerenciamento para os *peers* remotos com função de gerentes. A partir daí, estes *peers* ficam disponíveis para receberem comandos de gerenciamento utilizando protocolos encapsulados, tais como SNMP. Embora os autores proponham a substituição do modelo gerente/agente pelo modelo P2P, estes não apresentam novas funcionalidades, nem para o gerenciamento de redes *wireless*, nem para o gerenciamento de redes em geral, que utilizem a potencialidade do modelo P2P de forma a justificar esta mudança.

Já (GRANVILLE et al., 2005) buscam no modelo P2P uma forma de proporcionar facilidades adicionais para o gerenciamento de redes, facilidades estas em geral não providas pelas ferramentas de gerenciamento tradicionais. Os autores apresentam uma ferramenta de gerenciamento, chamada ManP2P, a qual provê as seguintes funcionalidades para enriquecer a comunicação e compartilhamento de informações entre os administradores de rede: i) o compartilhamento de arquivos de configuração de dispositivos de rede, os quais podem ser buscados segundo um conjunto de parâmetros; ii) compartilhamento de mapas de rede, inclusive com a apresentação dos alarmes emitidos por cada dispositivo apresentado no mapa; iii) *instant messaging*; e iv) criação de times de operadores, aos quais podem ser delegados privilégios administrativos.

O uso de uma rede P2P para a realização de testes e monitoramento de rede é apresentado por (BINZENHOFER et al., 2006). O objetivo desta abordagem é a construção de uma rede composta por diversos agentes de rede distribuídos, organizados através de tabelas *hash* distribuídas, baseadas no algoritmo Kademlia (MAYMOUNKOV; MAZIERES, 2002). Estes agentes possuem um conjunto de módulos para teste e monitoração. Cada módulo possui um conjunto de instruções necessárias para a realização de um teste ou ação de monitoramento específica. Estes módulos podem ser carregados sob demanda. Desta forma, qualquer *peer* é capaz de realizar qualquer tipo de teste ou ação de monitoramento após adquirir os módulos necessários para tal.

Os testes apresentados por (BINZENHOFER et al., 2006) incluem: avaliar o estado da interface de rede; verificar se a interface de rede está configurada com um endereço IP válido; testar se ao menos um dos servidores DNS configurados estão operacionais; testar conectividade (utilizando *ping*) com *hosts* conhecidos; avaliar o uso de outros agentes como proxy DNS; efetuar a varredura de portas (*portscan*) de forma distribuída, na qual agentes são escolhidos de forma aleatória para verificação da disponibilidade de serviços; e avaliar a vazão (*throughput*) entre dois agentes previamente configurados.

Os autores do artigo argumentam que a arquitetura apresentada pode facilitar a implementação do conceito de AC devido à característica de auto-organização da estrutura. Argumentam ainda que esta arquitetura complementaria as soluções atuais de gerenciamento de redes e controle de falhas. Para isto, os autores apresentam uma solução com função limitada, semelhante às *traps* SNMP. Contudo, não descrevem como esta funcionalidade poderia ser integrada à arquitetura centralizada das ferramentas atuais de gerenciamento. Não descrevem ainda como a estrutura poderia ser estendida para dar suporte a outras funcionalidades, como distribuição de *scripts* de configuração para os dispositivos gerenciados, por exemplo.

A solução proposta neste trabalho, apresentada no Capítulo 3, faz uso de uma rede P2P como uma rede *overlay*. Esta rede opera como uma camada, a qual tem a função de receber eventos externos, planejar ações a partir destes eventos e repassar estas ações, na forma de políticas para a camada de gerenciamento de redes baseado em políticas. Estes *peers* de gerenciamento fazem parte de um grupo de *peers*. Desta forma, este grupo de *peers* de gerenciamento pode ser implementado de forma a exigir que, para que um *peer* ingresse no grupo, este deve autenticar-se utilizando certificação digital, por exemplo (outros aspectos de segurança em redes P2P são discutidos em (YEAGER; WILLIAMS, 2002)). Além disso, este grupo de *peers* possui um conjunto de serviços disponíveis. Estes serviços são utilizados para a comunicação entre os próprios *peers*, de mesmo ou de outros domínios. Por exemplo, no estudo de caso apresentado na Seção 3.5, quando a fonte de eventos, neste caso o *toolkit* de grade, envia uma notificação informando que um usuário está iniciando a execução de uma aplicação utilizando a grade, o *peer* de gerenciamento que recebeu a notificação deve identificar os *peers* responsáveis pelos recursos utilizados pela aplicação e notificar estes *peers*. A busca pelos *peers* responsáveis pelos recursos utilizados pela aplicação da grade utiliza a estrutura P2P, assim como a interface de comunicação entre os *peers*.

As funcionalidades implementadas pelos *peers* de gerenciamento poderiam ser implantadas sem utilizar uma plataforma P2P. Esta decisão, porém, acarretaria em alto custo de desenvolvimento para disponibilizar facilidades já providas pela plataformas P2P. Em especial, a auto-organização da rede P2P, a descoberta de recursos, a autenticação inter-domínio e garantia de confidencialidade, providos pelas plataformas P2P, como o JXTA, seriam prejudicadas.

2.5 Suporte de rede para computação em grade

O gerenciamento da operação de uma grade computacional passa pelo gerenciamento da infra-estrutura de rede que provê o suporte de comunicação. O gerenciamento da infra-estrutura de rede é essencial, já que os usuários da grade acessam recursos compartilhados através da rede. Assim, problemas na rede, como congestionamento, atraso e indisponibilidade, comprometem diretamente a percepção de qualidade dos usuários da grade. A correta configuração da rede, em especial a configuração do suporte à QoS, provendo pri-

orização de fluxos e/ou reserva de banda aos fluxos críticos dos usuários da grade é um fator essencial para prover qualidade à sua operação.

Os *toolkits* para grade atuais (GLOBUS PROJECT, 2007a) (CONDOR, 2007) (OURGRID COMMUNITY, 2007) não interagem nem com as arquiteturas de suporte à QoS, como DiffServ e IntServ, nem com os sistemas de gerenciamento de rede. Isto leva a uma situação na qual os administradores da grade e da rede são obrigados a interagir toda vez que uma configuração de rede é necessária para dar o suporte requirido à operação da grade.

Propostas têm surgido no sentido de oferecer suporte de QoS automatizado para a operação das grades computacionais. (PRIMET; CHANUSSOT, 2004) apresentam um serviço, chamado QoSINUS, e uma interface de programação de aplicativos (*Application Programming Interface* - API), justamente com o objetivo de prover QoS fim-a-fim para os fluxos da grade em redes com suporte à DiffServ. Esta abordagem consiste em:

- Durante a inicialização da aplicação, antes de iniciar a transmissão de dados, é feita uma chamada ao serviço QoSINUS - utilizando a API fornecida - descrevendo os requisitos de QoS em termos de atraso, consumo de banda e percentual de perda de pacotes;
- O serviço QoSINUS associa a identificação do fluxo a uma classe de serviço (*Class of Service* - CoS) DiffServ específica;
- A partir deste momento, cada pacote dos fluxos da aplicação é marcado com uma CoS registrada.

O serviço QoSINUS possui ainda algoritmos que reavaliam periodicamente a CoS designada com base no monitoramento dos fluxos.

Apesar de cumprir o objetivo ao qual se propõe, isto é, prover QoS fim-a-fim para fluxos das aplicações da grade, esta abordagem possui alguns pontos negativos. Primeiramente, a solução não é transparente para as aplicações, uma vez que estas precisam ser construídas utilizando a API QoSINUS. As aplicações precisam ainda conhecer a localização do serviço QoSINUS, já que este serviço atua como um *proxy* entre as aplicações da grade e o restante da rede. Além disso, qualquer modificação do serviço ou atualização da API requer a recompilação das aplicações da grade. Por fim, os roteadores de borda necessitam prover o serviço QoSINUS ou, ao menos, ter acesso a ele, dado que o serviço QoSINUS é o responsável pela marcação de pacotes de acordo com a plataforma DiffServ.

(NEISSE et al., 2005) propõem uma solução de tradução de políticas na qual as políticas de gerenciamento de rede são criadas a partir das políticas de gerenciamento da grade computacional. A solução proposta traduz as políticas da grade em políticas de rede através de um mapeamento feito pelos administradores de rede em cada domínio administrativo que compõe a grade. Este mapeamento é definido em uma linguagem similar ao Java. As políticas de rede geradas pelo mecanismo de tradução são então aplicadas à plataforma de políticas, a qual segue o modelo proposto pelo IETF (MOORE et al., 2001), sendo, por fim, traduzidas em ações de configuração aplicadas pelos PDPs aos PEPs. A Figura 2.4 ilustra a arquitetura proposta por (NEISSE et al., 2005).

Esta abordagem, embora seja eficaz, requer um grande número de configurações a serem efetuadas pelo administrador de rede, uma vez que o administrador deve prever um conjunto de comportamentos da rede relacionados a um conjunto, possivelmente grande e dinâmico, de políticas da grade computacional.

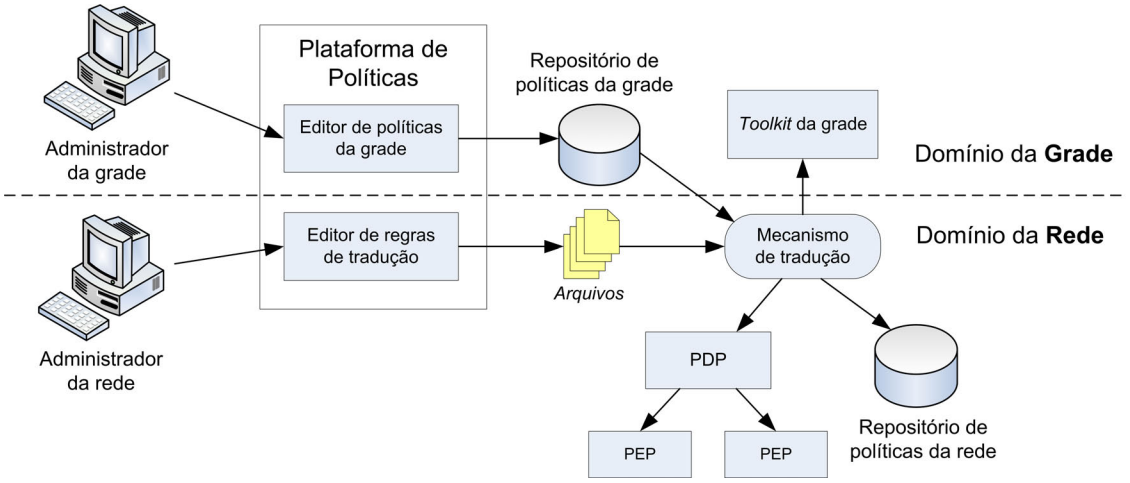

Figura 2.4: Arquitetura de (NEISSE et al., 2005)

(YANG; TODD, 2002) e (SANDER et al., 2001) utilizam políticas para a configuração da rede como suporte à computação em grade. A solução de (YANG; TODD, 2002) especifica uma arquitetura dividida em duas camadas: uma camada de gerenciamento baseado em políticas, tal como a solução apresentada neste trabalho, e uma camada que faz uso do conceito de *active networks*, como um *middleware* para as políticas. Já (SANDER et al., 2001) propõe uma arquitetura baseada em políticas para a configuração de QoS nos diferentes domínios administrativos membros de uma grade computacional. Nesta abordagem, as políticas são definidas em uma linguagem de baixo nível, de forma similar às políticas de rede definidas pelo IETF (MOORE et al., 2001). (SANDER et al., 2001) define um protocolo de sinalização inter-domínio, o qual seqüencialmente configura os membros do caminho fim-a-fim dos fluxos da grade. Embora estas abordagens sejam baseadas em políticas, estas não provêem qualquer suporte à integração das suas arquiteturas com os *toolkits* para computação em grade. Desta forma, isto exige que o administrador da grade, para realizar o gerenciamento de QoS desta, deva interagir com o administrador de rede em cada domínio, de forma a garantir que a rede de suporte esteja devidamente configurada para a operação da grade, além de utilizar o *toolkit* da grade para as demais configurações.

A proposta apresentada nesta dissertação possibilita automatizar a configuração dos dispositivos de rede de acordo com os requisitos de QoS das aplicações de grade. Esta automatização é feita através do envio de notificações por parte do *toolkit* da grade para o sistema de gerenciamento de redes, sendo a camada P2P responsável por traduzir estas notificações em políticas a serem aplicadas nas plataformas de políticas dos domínios abrangidos pela grade. A definição de como os fluxos da grade devem ser tratados pela rede, isto é, qual CoS deve ser atribuída aos pacotes destes fluxos, é obtida a partir das configurações do *toolkit* da grade. Desta forma, o administrador da grade responsabiliza-se apenas pelas configurações do *toolkit* da grade. Estas configurações são refletidas na rede de suporte de forma automatizada, seguindo o conceito de *auto-configuração* de AC. A Seção 3.5 apresenta em mais detalhes o emprego da arquitetura ao contexto de Computação em Grade.

3 PROPOSTA

A arquitetura proposta, fundamentada no modelo de gerenciamento de redes baseado em políticas, é focada especialmente em permitir que a rede adapte seu comportamento em resposta a eventos gerados por outros sistemas de gerenciamento e/ou monitoramento. Desta forma, a arquitetura proposta identifica-se fortemente com o modelo de gerenciamento de AC, o qual preconiza que seus AMs, cada qual aplicado a um contexto de TI, possuam interfaces capazes de receber e enviar notificações a AMs aplicados a outros contextos. Por exemplo, através da proposta descrita nesta dissertação, um IDS ao enviar uma notificação informando um ataque de *Denial of Service* (DoS) permite que o sistema de gerenciamento da rede implante uma política isolando os segmentos de rede dos *hosts* atacantes.

As seções a seguir descrevem a arquitetura proposta. A Seção 3.1 descreve as camadas que compõe a solução e os seus componentes. Já a Seção 3.2, descreve em mais detalhes as interfaces de comunicação entre as camadas e as mensagens trocadas entre estas. A Seção 3.3 discute a questão da execução de ações de gerenciamento que envolvam mais de um domínio administrativo. A Seção 3.4 visa evidenciar as etapas executadas a partir da recepção de uma notificação. Por fim, na Seção 3.5, é analisado o emprego da arquitetura proposta na recepção de notificações geradas por uma grade computacional.

3.1 Camadas

Para prover a funcionalidade de adaptação do comportamento da rede em resposta a eventos externos, foi projetada uma arquitetura em camadas que combina a flexibilidade das redes P2P com a eficiência do modelo de gerenciamento de redes baseado em políticas. A divisão da arquitetura em camadas visa modularizar as funções desempenhadas, permitindo que a complexidade de cada camada permaneça oculta para o restante do sistema. A Figura 3.1 apresenta as camadas que compõem a arquitetura.

3.1.1 Camada Fontes de Eventos

A camada de Fontes de Eventos representa os componentes de outros sistemas de gerenciamento e/ou monitoração que enviam notificações aos *peers* de gerenciamento de rede. Considera-se neste texto fonte de eventos como uma representação abstrata de um elemento externo responsável por enviar eventos para serem tratados pelo sistema de gerenciamento. Alguns exemplos de fontes de eventos potenciais seriam: i) o sistema de gerenciamento de um Network-Attached Storage (NAS); ii) um *toolkit* de computação em grade; iii) um sistema de autenticação; iv) um IDS; v) um Sistema de Gerenciamento de Banco de Dados (SGBD). Na verdade, uma fonte de eventos pode ser pensada como sendo

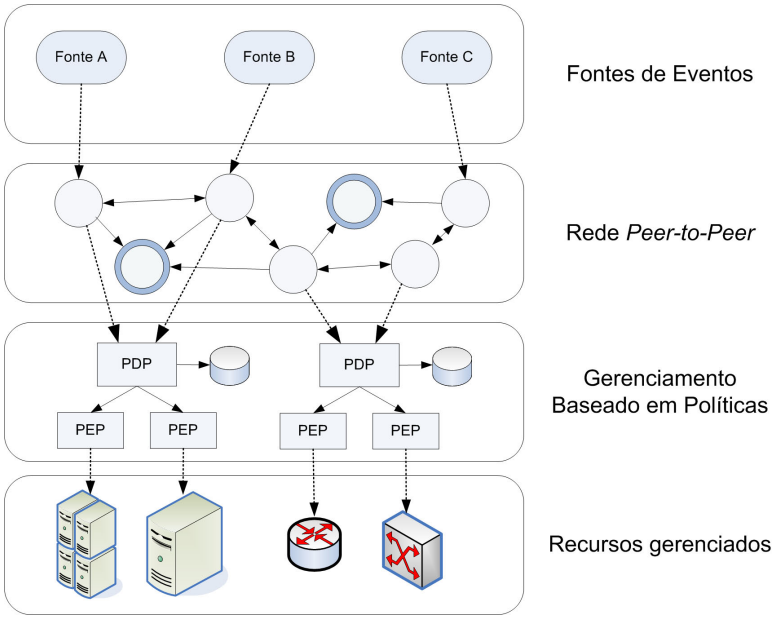

Figura 3.1: Camadas da arquitetura

qualquer elemento cujo estado (ou estado dos recursos por ele monitorado) é relevante para o sistema de gerenciamento de redes. Assim, uma aplicação que envia amostras (*probes*) para monitorar o estado dos enlaces de rede poderia também ser utilizado como fonte de eventos da solução, já que este poderia notificar o sistema de gerenciamento quando certos limitares fossem ultrapassados.

Segundo o modelo de AC, as fontes de eventos representam os atuadores de outros AMs. Assim, o AM responsável pelo gerenciamento de rede deve ser capaz de monitorar a recepção destas notificações enviadas por AMs aplicados a outros contextos, para então encaminhá-las a atividade de análise. Este papel de monitoramento das notificações recebidas é de responsabilidade da camada P2P.

3.1.2 Camada P2P

Além de monitorar as notificações geradas pelas fontes de eventos, a camada P2P é responsável pelas seguintes atividades previstas pelos conceitos de AC:

1. Análise - Verificar se a notificação recebida é relevante para o contexto sob gerenciamento, neste caso a rede. Além disso, a camada P2P deve identificar os recursos relacionados com a notificação e verificar o estado desses recursos para embasar o planejamento das ações a serem tomadas.

2. Planejamento - A partir das informações recebidas da fase de análise, os *peers* de gerenciamento de rede planejam as políticas a serem aplicadas (ou removidas). Este planejamento pode ser realizado de forma distribuída, isto é, pode envolver um ou mais *peers* de gerenciamento. Esta situação ocorre especialmente quando a notificação recebida envolve recursos localizados em diferentes domínios, no qual o PDP responsável pelo recurso não pode ser diretamente acessado pelo *peer* que recebe a notificação. Então, este *peer* necessita buscar um *peer* com acesso ao PDP.

3. Execução - Aplica as ações planejadas (inserção ou remoção de políticas) no PDP da camada de Gerenciamento de Redes Baseada em Políticas. Usualmente, um *peer*

de gerenciamento pode ter acesso apenas aos PDPs localizados no mesmo domínio administrativo. Como os PDPs não fazem parte da rede P2P, os *peers* de gerenciamento devem ser configurados (diretamente ou através de algum tipo de serviço de localização) com a localização dos PDPs. Alternativamente, a arquitetura poderia ser construída de forma aos *peers* de gerenciamento acessarem diretamente os repositórios de políticas da camada de gerenciamento baseado em políticas.

Acredita-se que o modelo P2P é mais apropriado para a implementação das atividades descritas acima em relação ao modelo cliente-servidor tradicional, utilizado no *framework* SNMP, por exemplo. Isto porque o modelo P2P possui características que possibilitam às plataformas P2P proverem:

- Melhor conectividade entre componentes localizados em diferentes domínios administrativos. O roteamento de mensagens no nível de aplicação permite que as mensagens P2P trafeguem pela Internet sem serem bloqueadas por dispositivos como *firewalls* ou NATs, característica essencial para um sistema que pretende prover gerenciamento de redes inter-domínio.

- Auto-organização da rede P2P. A conectividade existente entre os *peers* da rede permite que as plataformas P2P possuam protocolos de descoberta para o estabelecimento de caminhos virtuais entre os *peers*. Para isto, esses protocolos fazem uso de tabelas *hash* distribuídas baseadas em algoritmos como Kademlia (MAYMOUNKOV; MAZIERES, 2002) e Chord (STOICA et al., 2001).

- Protocolos de busca embarcados. Baseado nos mesmos protocolos de descoberta utilizados na auto-organização da rede, grande parte das plataformas P2P permite às aplicações realizarem buscas de recursos, como arquivos compartilhados, utilizando mecanismos (protocolos e APIs) disponibilizadas pelas próprias plataformas.

Desta forma, as ações de gerenciamento previstas pelo modelo de AC, as quais requerem forte integração entre os elementos gerentes, são mais naturalmente implementadas sobre plataformas P2P. Para dar suporte as atividades previstas pelo modelo de AC, a arquitetura proposta prevê dois tipos de *peers*: *peer de gerenciamento de rede* e *gerente de recursos*.

Os *peers* de gerenciamento de rede são os componentes da camada que coordenam a execução das atividades de monitoramento, análise, planejamento e execução apresentadas anteriormente. Isto significa que os *peers* de gerenciamento devem possuir uma interface para monitorar as notificações emitidas pelas fontes de evento. A partir daí, o *peer* de gerenciamento que recebe a notificação passa a coordenar as ações esperadas da camada P2P em resposta a notificação recebida. Estas ações, as quais são descritas em detalhes na Seção 3.4, envolvem a análise dos recursos envolvidos na notificação, ou seja, a verificação do estado destes recursos. Esta avaliação do estado dos recursos é feita com o auxílio dos *peers* com a função de gerente de recursos.

Um gerente de recursos é um *peer* integrante da rede P2P com a função de armazenar o estado de alguns recursos pré-definidos. A presença deste elemento na arquitetura pode ou não ser relevante dependendo do contexto de aplicação da solução. Por exemplo, no estudo de caso apresentado na Seção 3.5, este elemento é utilizado como um corretor (*broker*) de largura de banda para o gerenciamento de QoS em redes com suporte à DiffServ. Ele registra a quantidade de largura de banda alocada pelos usuários e aplicações

em cada CoS de cada enlace WAN, de forma similar a abordagem proposta em (MAGANA; SERRAT, 2005). Desta forma, o gerente de recursos pode controlar a quantidade de banda reservada a cada CoS e enlace. Isto evita, por exemplo, que uma determinada operação privilegiada cause *overflow* em uma determinada CoS e enlace.

Desta forma, os *peers* de gerenciamento de rede, em conjunto com os gerentes de recursos formam a camada "inteligente" da arquitetura, já que é esta camada que tem por função interpretar as notificações recebidas de outros gerentes, propondo ações a partir destas interpretações. A efetiva aplicação destas ações fica a cargo da camada de Gerenciamento Baseado em Políticas.

3.1.3 Camada Gerenciamento Baseado em Políticas

O gerenciamento de redes baseado em políticas, baseado no *framework* recomendado pelo IETF, é um conceito atualmente aceito e largamente reconhecido pela comunidade de gerenciamento de redes. Políticas têm sido aplicadas principalmente ao contexto de segurança e QoS. A principal funcionalidade provida pelo gerenciamento baseado em políticas aplicado à QoS é o de dar ao administrador de redes a possibilidade de restringir QoS aos fluxos de determinados usuários, aplicações e/ou *hosts* e sobre determinadas condições através da elaboração de regras em alto nível. Além disso, permite ao administrador incluir nestas regras a restrição de determinados serviços a determinados períodos de tempo a fim de evitar sobrecarga na rede.

Da mesma forma, na proposta apresentada nesta dissertação, a camada de Gerenciamento Baseado em Políticas implementa uma estrutura que permite o estabelecimento destas regras. A Figura 3.2 ilustra como a estrutura de gerenciamento de políticas está inserida na arquitetura proposta.

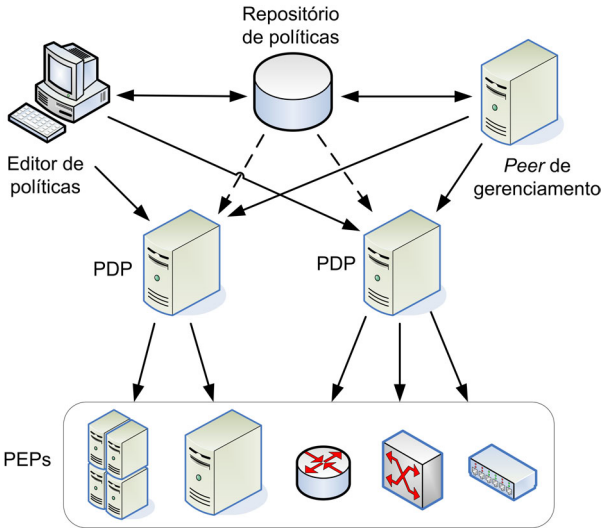

Figura 3.2: *Peer* de gerenciamento e plataforma de gerenciamento de políticas

Conforme demonstra a Figura 3.2, além de definidas pelo administrador de rede, as políticas que regulam o comportamento da rede podem ser definidas também pela camada P2P. Dentro do escopo proposto, a camada P2P possui gerência apenas sobre as políticas referentes ao gerenciamento do QoS, em especial sobre a classificação de pacotes nos dispositivos de rede com suporte a Diffserv. O papel da camada P2P resume-se a transpor para as políticas de QoS o comportamento derivado a partir das notificações recebidas e do estado do ambiente gerenciado. Na prática, isto significa que a camada P2P irá, a partir

das suas fases de análise e planejamento, definir que um determinado conjunto de fluxos deverá ser tratado pela rede com suporte a Diffserv como sendo de uma determinada CoS. Para isto, a camada P2P elabora as políticas a serem aplicadas e as repassa para a camada de políticas para a efetiva implantação. Novas notificações recebidas podem fazer com que as fases de análise e planejamento da camada P2P determinem que políticas aplicadas anteriormente devem ser agora removidas.

Na medida em que a arquitetura está organizada em camadas, é importante que o protocolo de troca de mensagens entre as camadas seja bem definido (a comunicação entre as camadas é descrita na Seção 3.2). Assim, a estrutura interna de cada camada permanece oculta para as demais camadas. Na arquitetura proposta buscou-se não alterar a estrutura da camada de políticas, isto para que se possam utilizar soluções já existentes de gerenciamento de redes baseado em políticas. Além disso, permite que as camadas da arquitetura sejam implementadas utilizando diferentes tecnologias, cada qual mais apropriada para a sua aplicação. Na camada Fonte de Eventos, o fator mais importante é permitir que se mantenha um canal de comunicação permanente entre a fonte de eventos e a camada P2P para que as notificações sejam enviadas assim que geradas. Já na camada P2P, o fator preponderante a ser considerado é a conectividade entre os elementos da camada, já que os elementos desta camada podem estar localizados em pontos diversos da Internet. Na camada de políticas, no entanto, o fator mais importante é a eficiência na aplicação das políticas. Isto porque, dependendo do modelo de distribuição de políticas adotado, o número de requisições de decisão do PEP para o PDP pode ser considerável. Assim, é necessário que esse processo seja eficiente o bastante para não introduzir atrasos significativos nas comunicações.

Apesar da independência entre as camadas, algumas relações são necessárias para que a camada P2P possa elaborar políticas de alto nível a serem aplicadas na plataforma de políticas. No contexto de gerenciamento de redes baseado em políticas, o termo "políticas em alto nível" significa que o administrador pode elaborar políticas abstraindo os dispositivos existentes em sua rede real. Isto é, podem ser criadas políticas de mais baixo nível que agrupam dispositivos com características semelhantes. Por exemplo, roteadores de um determinado modelo, ou então com função semelhante, como roteadores e *switches* do núcleo ou de borda podem ser agrupados em classes. A partir daí, o administrador pode criar políticas que se apliquem a estes agrupamentos, os quais possuem uma nomenclatura mais significativa, não tendo que descrever nominalmente cada dispositivo para o qual se deseja que a regra seja aplicada.

Desta forma, para que a camada P2P possa implementar uma política com a seguinte regra:

```
SE enderecoIPOrigem = 192.168.12.17 E
enderecoIPDestino = 192.168.24.8 E protocolo = RTP
ENTAO classeServiço = AssuredForwarding
OBJETIVO RoteadoresBordaDiffserv
```

Esta regra determina que o tráfego entre os dois *hosts* especificados deve ser tratado como sendo da classe de serviço "AssuredForwarding" pelos roteadores de borda com suporte à Diffserv, é preciso que na camada de políticas exista uma regra que determine o significado da classe "RoteadoresBordaDiffserv". Assim, é necessário que um conjunto de classes, utilizando palavras-chave pré-definidas, seja definido na camada de políticas dos domínios administrativos que implementam a arquitetura proposta.

O escopo proposto nesta dissertação para a geração de políticas pela camada P2P é o de gerenciamento de QoS em redes com suporte à Diffserv. Uma extensão possível da arquitetura proposta seria a ampliação dos tipos de políticas aplicadas pela camada P2P, como: geração de políticas de segurança a partir do recebimento de notificações de domínios confiáveis e suporte à outras técnicas de QoS em redes IP, como MPLS (AWDUCHE et al., 1999).

3.1.4 Camada Elementos Gerenciados

A camada Elementos Gerenciados representa os dispositivos de rede aos quais as políticas são destinadas. Na arquitetura proposta, estes dispositivos comunicam-se exclusivamente com os PEPs, os quais são agentes de *software* geralmente localizadas no próprio dispositivo, de forma semelhante aos agentes SNMP.

3.2 Comunicação entre camadas

Uma característica importante da modularização de um sistema em camadas é a possibilidade de substituir a implementação das camadas sem prejuízo para o sistema como um todo. Para que esta modularização seja efetiva, porém, é necessária uma definição rígida dos protocolos utilizados para a troca de mensagens entre as camadas. Assim, esta seção visa descrever como a arquitetura proposta trata a interação entre as camadas. Em especial, como forma de subsidiar o projeto de implementação da arquitetura, busca-se identificar quais modelos de troca de mensagens se adaptam melhor a cada interface entre camadas.

3.2.1 Comunicação Fonte de Eventos - Peers de Gerenciamento de Rede

Na medida em que as fontes de eventos devem enviar notificações à camada P2P, estas devem estabelecer algum tipo de comunicação com a camada P2P, em particular com os *peers* de gerenciamento de rede. Nesta proposta sugere-se que a troca de mensagens entre as fontes de eventos e os *peers* se dê através do modelo Publicar/Assinar (ou *Publish/Subscribe*) (EUGSTER et al., 2003). Publicar/Assinar é um paradigma assíncrono de troca de mensagens no qual os emissores não são previamente programados para enviar suas mensagens para determinados receptores. Ao invés disso, as mensagens são publicadas para um determinado tópico, gerenciado pelo sistema de troca de mensagens. Conforme ilustrado na Figura 3.3, este sistema é um intermediário na troca de mensagens, e opera independentemente de existirem emissores ou receptores. Os "assinantes" (receptores) podem registrar interesse em receber mensagens de um ou mais tópicos. Assim, neste modelo nem o emissor nem o receptor precisam ter conhecimento um do outro, já que a troca de mensagens é gerenciada por um terceiro sistema, o qual permite a criação de tópicos e os disponibiliza para assinatura.

Este desacoplamento trás vantagens na comunicação entre as fontes de eventos e os *peers* de gerenciamento de rede. Uma delas é a fácil adaptação às mudanças dos elementos das camadas. Isto porque a entrada e saída de elementos, tanto da camada de fontes de eventos quanto da camada de *peers* de gerenciamento, não requer a reconfiguração dos demais componentes. Por exemplo, considerando uma fonte de eventos localizada em um dispositivo móvel. Esta fonte de eventos poderá estar ligada a uma determinada rede em um momento e em um segundo momento estar ligada a outra rede (*roaming*) com outro endereço. Desde que a fonte de eventos mantenha conectividade com o sistema de troca de mensagens, esta poderá seguir publicando mensagens a um determinado tópico. De

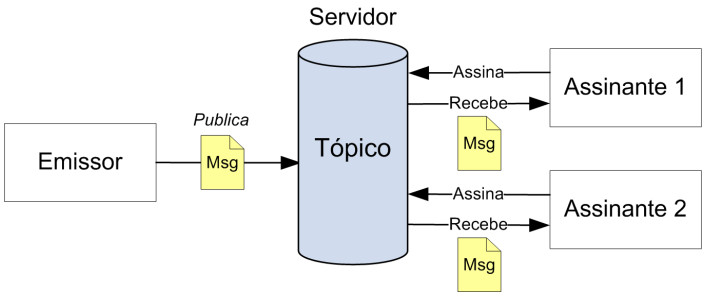

Figura 3.3: Paradigma de troca de mensagens Publicar/Assinar

forma transparente, os assinantes deste tópico continuarão recebendo mensagens normalmente, independente da localização dos emissores. Outra característica importante é que o emissor e o receptor não precisam estar disponíveis ao mesmo tempo. Isto significa que no momento do envio da mensagem pode não haver assinantes disponíveis. Porém, na medida que os assinantes ficarem disponíveis, estes poderão receber as mensagens enviadas. É possível ainda estabelecer um determinado limite de tempo para aguardar a recepção das mensagens. Na arquitetura proposta, para a comunicação entre fonte de eventos e *peers* de gerenciamento, esta configuração seria útil, já que na maioria dos casos o recebimento tardio das mensagens pode não ser relevante. Além disso, o sistema de troca de mensagens poderia ser configurado para enviar a notificação a apenas um dos assinantes, de forma aleatória. Isto evita que os *peers* trabalhem de forma replicada, além de balancear o trabalho de processamento das notificações.

Um sistema de troca de mensagens seguindo o modelo Publicar/Assinar pode ser implementado de diversas formas. A abordagem mais usual é implementar este sistema como um terceiro elemento da comunicação, de forma independente dos emissores e assinantes (Figura 3.4a). Desta forma, existe um servidor de troca de mensagens, o qual é responsável por receber, armazenar e repassar as mensagens dos emissores, além de registrar os assinantes interessados em receber determinados tópicos. Outra abordagem, é delegar aos próprios elementos da comunicação a função de gerenciar a entrega de mensagens. Conforme ilustra a Figura 3.4b, nesta abordagem todos os nodos são iguais, podendo atuar como emissores e/ou assinantes. Assim, a própria rede é responsável pela entrega das mensagens aos assinantes, não existindo nodos atuando como servidores de forma dedicada. As funções que na primeira abordagem são desempenhadas pelo servidor, como persistência, controle de transações e segurança, são agora incluídas de forma embarcada em cada nodo.

(EUGSTER et al., 2003) apresenta algumas soluções para troca de mensagens que utilizam o modelo Publicar/Assinar. Dentre eles está o Java Message Service (JMS), o qual é uma API para aplicações baseadas em Java, o qual permite a implementação de ambas as abordagens, com e sem servidor. Outro padrão de interação comumente utilizado para comunicações entre objetos é o Web Services Notification (WS-Notification) (OASIS, 2006a). O WS-Notification consiste em um conjunto de especificações que visa definir um padrão para troca de mensagens seguindo o modelo Publicar/Assinar aplicado à *web-services*. Este padrão trás consigo as vantagens de se utilizar XML e *web-services*, como: i) independência de plataforma e de linguagem de programação utilizada nos componentes e ii) possibilidade de uso de diversos protocolos de transporte.

No contexto da arquitetura proposta nesta dissertação, o padrão WS-Notification parece ajustar-se melhor como solução para troca de mensagens entre as fontes de eventos

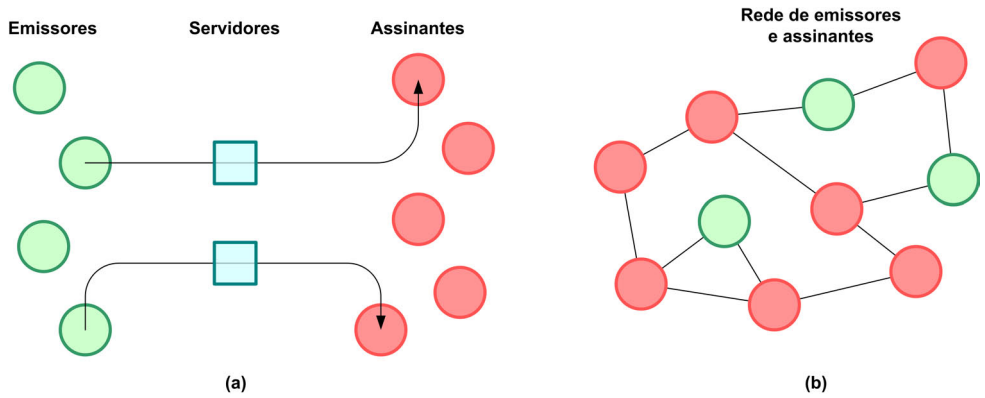

Figura 3.4: Topologias do modelo Publicar/Assinar: a) Com servidor b) Sem servidor

e os *peers* de gerenciamento. Isto, devido a característica que deseja-se implantar na arquitetura de independência de tecnologia e isolamento entre as camadas. O padrão WS-Notification é um padrão aberto aprovado pelo consórcio OASIS (OASIS, 2006a), o que dá maior garantia da continuidade da tecnologia. Além disso, é atualmente suportado nativamente por diversas plataformas, entre elas o *toolkit* para grades computacionais Globus Toolkit 4 (GT4) (GLOBUS PROJECT, 2007a), o que facilita o desenvolvimento de aplicações que façam uso deste padrão.

Cabe salientar que, independente do padrão de troca de mensagens adotado, as soluções apresentadas anteriormente visam padronizar apenas o formato do envólucro das mensagens e não o conteúdo das mensagens em si. Desta forma, o formato das mensagens deve ser definido por cada aplicação que se utiliza do sistema de troca de mensagens. Em relação ao formato das notificações enviadas pelas fontes de eventos aos *peers* de gerenciamento, acredita-se que estas sejam mais facilmente representadas como mensagens baseadas em uma linguagem de marcação, cujo esquema utilizado dependerá do contexto da fonte de eventos. Um exemplo disto, seria um sistema de armazenamento distribuído no papel de fonte de eventos. Neste caso, a fonte de eventos informaria ao sistema de gerenciamento de redes a programação de um *backup* com replicação para que este configure os dispositivos de rede para o uso de *multicast* utilizando um formato especificado. Já um sistema de agendamento de teleconferências, que necessite da reserva de banda entre dois pontos, enviaria notificação em um formato ligeiramente diferente. Desta forma, cada *peer* de gerenciamento deve suportar um conjunto limitado de tipos de fontes de eventos, não só pelas especificidades do formato das mensagens de notificação, mas também porque a análise e planejamento das ações a serem tomadas são, em geral, dependentes do tipo de fonte de eventos.

Como ponto de extensão possível da solução, uma linguagem padronizada para as diferentes fontes de eventos poderia ser elaborada, sob o custo de as fontes de eventos terem que adaptar seu formato de notificação para este formato padronizado, ao invés de os *peers* de gerenciamento terem que interpretar diferentes formatos de mensagens. O uso de *gateways* para conversão entre os formatos das mensagens também seria uma alternativa possível.

3.2.2 Comunicação Peers de Gerenciamento - Plataforma de Políticas

A comunicação entre a camada de *peers* de gerenciamento e a plataforma de políticas possui características diferentes da comunicação entre as fontes de eventos e os *peers* de gerenciamento. Isto porque, no caso da comunicação entre as fontes de eventos e a rede

P2P, ambos lados da comunicação são ambientes possivelmente dinâmicos. Neste contexto, o estabelecimento de uma comunicação ponto-a-ponto ou cliente-servidor ficaria prejudicado, devido a dificuldade da fonte de eventos localizar um *peer* de gerenciamento. Por outro lado, a comunicação entre os *peers* de gerenciamento e a plataforma de políticas possui características diferentes na medida em que os componentes da plataforma de políticas, como o PDP, são geralmente estáveis.

Neste contexto, uma abordagem possível para esta comunicação é seguir o modelo cliente-servidor tradicional. A abordagem adotada na arquitetura proposta é de os *peers* de gerenciamento enviarem mensagens com as políticas a serem aplicadas ao PDP responsável pelos recursos envolvidos e este então armazenar estas políticas no repositório. Desta forma, o PDP comporta-se como servidor, na medida que este é a parte estável da comunicação e fica a espera de mensagens. Já os *peers* atuam como clientes e devem ser configurados (diretamente ou através de algum tipo de serviço de localização) com a localização dos PDPs.

Esta conexão cliente-servidor pode se dar utilizando diversos protocolos. Mais uma vez o uso de *web-services* é uma opção a ser considerada. Optando-se pelo uso de *web-services*, deve ser avaliado ainda o protocolo a ser utilizado, tais como SOAP e XML-RPC (WINER, 2007) (avaliações de desempenho de SOAP e XML-RPC podem ser encontradas em (HEAD et al., 2005) e (ALLMAN, 2003), respectivamente). O uso de *web-services* trás consigo a vantagem do seu fácil trânsito sobre a Internet. Na medida em que os *peers* de gerenciamento e os PDPs localizam-se no mesmo domínio administrativo esta característica perde a relevância. Em relação a segurança, contudo, o uso de *web-services*, em particular de mensagens SOAP associadas ao padrão WS-Security (OASIS, 2006b), permite o uso de técnicas como criptografia e assinatura digital (RAHAMAN; SCHAAD; RITS, 2006). A especificação WS-Security descreve como incluir informações sobre assinatura e criptografia às mensagens SOAP, assim como incluir elementos de segurança como certificados X.509 e *tickets* Kerberos.

As mensagens enviadas pelos *peers* de gerenciamento aos PDPs consistem em comandos de inclusão ou remoção de políticas. Na arquitetura proposta, estas mensagens são estruturadas em XML conforme o Policy Core Information Model Extensions (PCIMe) (MOORE, 2003), parte do padrão Core Information Model (CIM). Este modelo, o qual é uma atualização da RFC-3060 (MOORE et al., 2001), é um modelo criado pelo IETF Policy Framework Workgroup (IETF, 2007) em conjunto com o Distributed Management Task Force (DMTF) (DMTF, 2007) para estruturar informações relativas à politicas. A Seção 3.4 descreve como a arquitetura proposta trata a criação das mensagens contendo as políticas a serem aplicadas ou removidas.

3.3 Gerenciamento entre domínios

Uma importante característica do modelo de gerenciamento proposto é a possibilidade de planejar políticas a serem implantadas em diferentes domínios administrativos. Isto torna possível definir um comportamento durante todo o caminho percorrido por um fluxo de dados. Além disso, a arquitetura proposta permite que este comportamento fim-a-fim seja definido de forma transacional. Na prática, isto significa que configurações da rede, como reserva de banda, podem ser implementadas de forma que se aborte a operação caso um dos roteadores da rota não permita a reserva requerida.

Para que seja possível implantar esta característica é necessário que os elementos da arquitetura possuam conectividade mesmo se localizados em diferentes domínios admi-

nistrativos. A escolha do protocolo de comunicação utilizado para a comunicação entre estes elementos é fundamental para fornecer esta conectividade. O protocolo SNMP, utilizado vastamente nas soluções de gerenciamento de rede, possui limitações neste sentido. As mensagens SNMP raramente cruzam os limites dos domínios administrativos devido aos *firewalls* instalados nos *gateways* da rede. Estes *firewalls* bloqueiam o tráfego SNMP de forma a evitar qualquer tentativa de inspeção externa não autorizada.

Já na arquitetura proposta, é a camada P2P a responsável por prover conectividade entre domínios. As plataformas P2P, em geral, são projetadas para operar em *hosts* localizados em pontos diversos da Internet, buscando evitar as restrições impostas pelos *firewalls*. Para isto utilizam técnicas como: portas dinâmicas, protocolos de tunelamento (*tunneling*) e encapsulamento de tráfego sobre HTTP (e.g. JXTA-SOAP (DISTRIBUTED SYSTEMS GROUP, 2006)).

Por questões de segurança, mas também por indisponibilidade de endereços IP válidos, grande parte das corporações se utilizam da técnica de Network Address Translation (NAT) (EGEVANG; FRANCIS, 1994). De forma similar aos *firewalls*, o uso de NAT também dificulta as ações de gerenciamento que envolvam domínios que utilizem esta técnica. Isto porque não há uma forma direta de um gerente externo ao domínio saber qual endereço válido deve utilizar para ter acesso a um dispositivo interno. Assim, independente de se utilizar um modelo de gerenciamento baseado em gerente-agente (como o SNMP) ou baseado em P2P, um tratamento especial é requerido quando ao menos uma das partes envolvidas na comunicação está localizada em um domínio protegido por NAT, já que a técnica de NAT rompe a lógica da camada de rede.

Em relação às redes P2P, a forma mais confiável de transpor o NAT é o uso de *relays*, também chamados de *gateways* ou roteadores em algumas plataformas (outras técnicas de transposição de NAT aplicadas às redes P2P são discutidas em (FORD; KEGEL; SRISURESH, 2005)). Este método consiste em inserir um *peer* com função de retransmissor. A este *peer* é atribuído um endereço IP real. Desta forma, na visão da rede, esta comunicação P2P passa a ser vista como duas comunicações cliente-servidor.

A Figura 3.5 ilustra um cenário no qual o *peer* identificado como Peer1 encontra-se em um domínio protegido por NAT, portanto não visível ao restante da rede P2P. Assim, para estabelecer comunicação com o Peer3, este deve utilizar um *peer* com função de *relay*, neste caso Peer2. A desvantagem desta solução é que Peer1 deve conhecer o endereço IP do Peer2 e estar configurado para utilizá-lo como *relay*.

Desta forma, a rede P2P presente na arquitetura proposta fornece o suporte de comunicação necessário para a execução de ações de gerenciamento envolvendo diversos domínios administrativos. A conectividade mantém-se inclusive em ambientes protegidos, neste caso podendo ser necessária alguma configuração adicional. Porém, a conectividade apenas permite a comunicação entre os elementos da arquitetura. É a ação coordenada entre os *peers* de gerenciamento que irá prover a configuração fim-a-fim do comportamento da rede. As ações de gerenciamento da solução são descritas na Seção 3.4.

3.4 Processo de notificação

Esta seção descreve o processo disparado a partir da recepção de uma notificação de uma fonte de eventos. Conforme já descrito anteriormente, os *peers* de gerenciamento de rede são os responsáveis por monitorar as notificações enviadas pelas fontes de eventos. Segundo o modelo Publicar/Assinar, apresentado na Seção 3.2.1, a fonte de eventos comporta-se como emissor das mensagens e os *peers* de gerenciamento como assinantes.

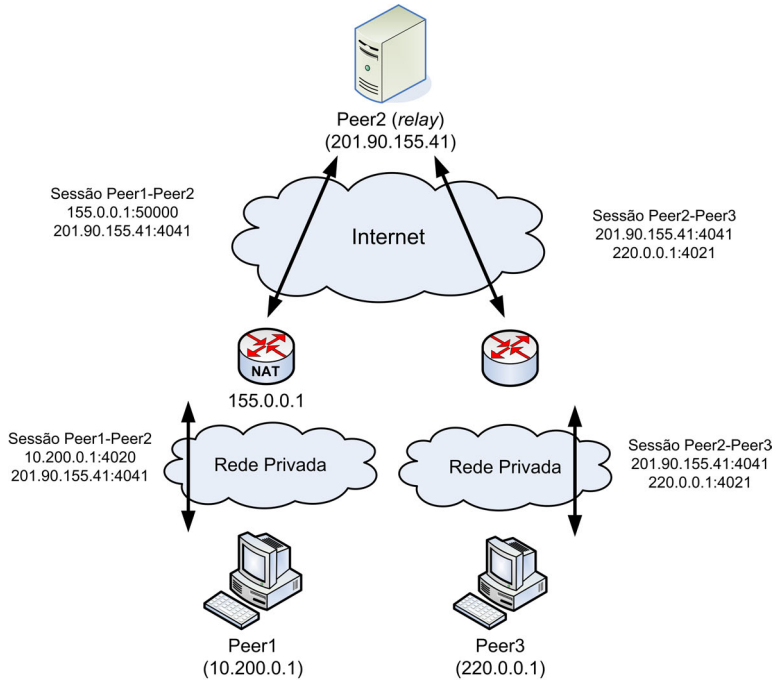

Figura 3.5: Transposição de NAT utilizando *relay*

O primeiro conjunto de ações executadas pelo *peer* de gerenciamento após ter recebido a notificação são apresentados na Figura 3.6.

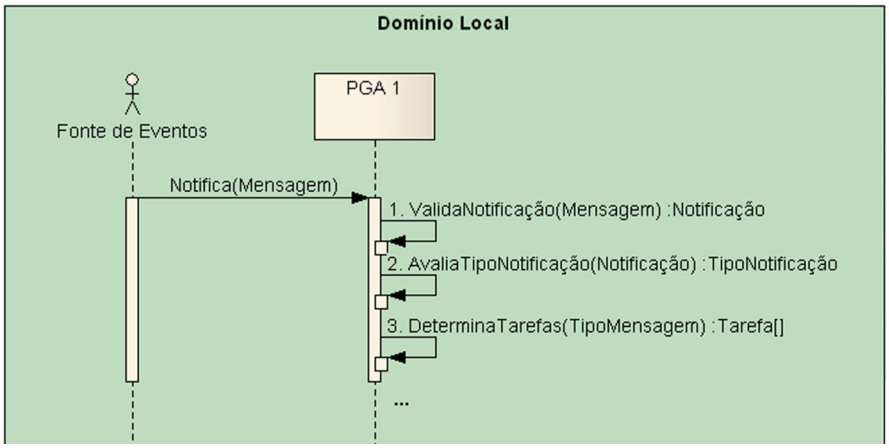

Figura 3.6: Valida a notificação e determina as tarefas a serem executadas

Conforme ilustrado pela Figura 3.6, após receber mensagem da fonte de eventos o *peer* de gerenciamento (representado como *Peer* de Gerenciamento Autonômico 1 - PGA 1) executa os seguintes passos:

1. Valida a mensagem de notificação recebida.

2. Identifica o tipo de notificação. O tipo de notificação está relacionado ao evento que gerou a notificação. Por exemplo, no contexto de grades computacionais, o *toolkit* da grade, o qual atua como fonte de eventos, deve enviar notificações aos *peers* de gerenciamento relatando a inicialização ou finalização das aplicações da grade.

3. O próximo passo para o *peer* é determinar quais as tarefas a serem realizadas para o tipo de notificação recebida. Cada *peer* deve possuir uma base de conhecimento que indique, para cada tipo de notificação suportada, quais as tarefas a serem realizadas. Esta forma de representar o conhecimento do *peer* de gerenciamento é inspirada nos modelos de raciocínio de agentes artificiais, da área de Inteligência Artificial (RUSSELL; NORVIG, 1995). Caso o tipo de notificação não seja suportado pelo *peer* de gerenciamento, este deve retornar uma exceção à fonte de eventos informando que o tipo de notificação enviado não é suportado pelo *peer* de gerenciamento.

Na arquitetura projetada, a base de conhecimento, descrita no passo 3, é obtida através de um arquivo de configuração localizado externamente ao *peer* de gerenciamento. Assim, é possível alterar os procedimentos realizados pelos *peers* sem requerer a reprogramação destes. O protótipo implementado, descrito no Capítulo 4, demonstra esta característica.

Uma extensão possível da arquitetura seria fazer com que o *peer* de gerenciamento de rede reavalie periodicamente as ações realizadas para cada tipo de notificação recebida. Esta reavaliação poderia basear-se em dados históricos de certos parâmetros da rede, como percentual de perda de pacotes, atraso e *jitter*, coletados a partir de monitores inseridos em pontos da rede. Esta característica faria com que os *peers* de gerenciamento pudessem ser considerados verdadeiramente como agentes inteligentes, na medida em que passariam a adaptar o seu comportamento às condições do ambiente.

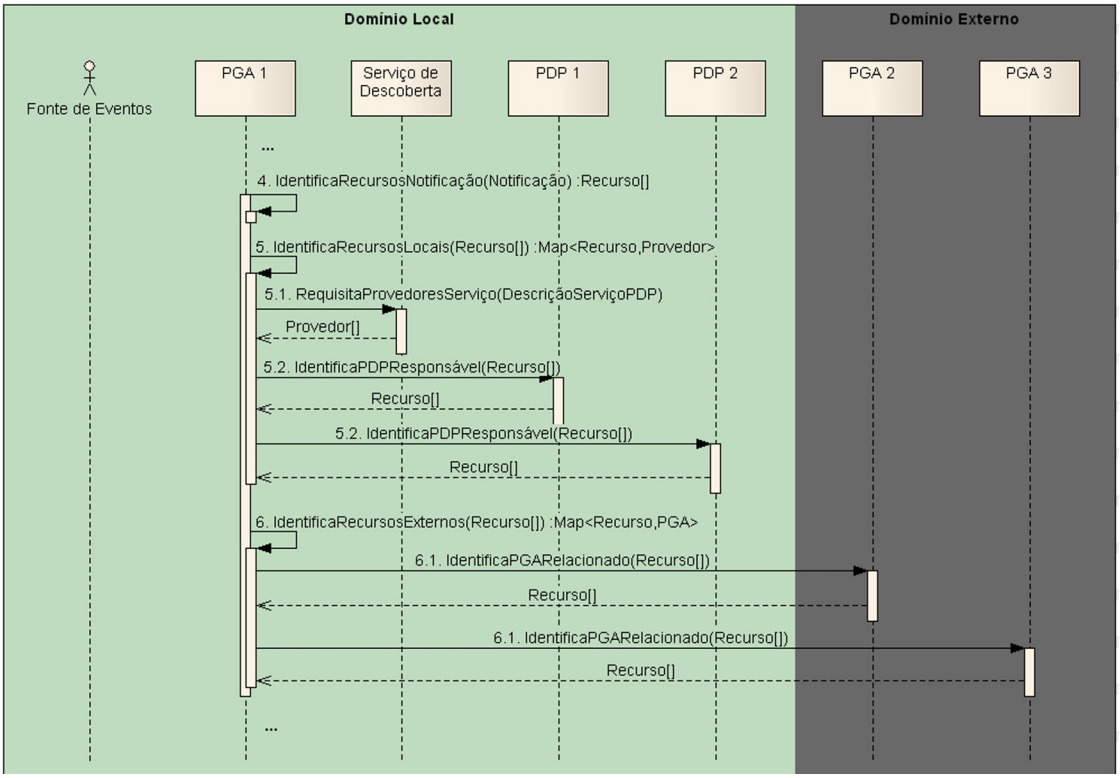

Figura 3.7: Identificação dos recursos envolvidos

As próximas tarefas apresentadas nesta seção representam o comportamento padrão de um *peer* de gerenciamento. A Figura 3.7 representa as tarefas de identificação dos recursos envolvidos na notificação e localização dos "responsáveis" por este recurso, isto é, identificação dos componentes da arquitetura que possuem acesso à configuração do

dispositivo de rede que disponibiliza o recurso. Estas tarefas são implementadas através dos seguintes passos:

4. Identifica os recursos descritos na notificação, como *hosts* e dispositivos de rede, por exemplo.

5. Verifica quais dos recursos presentes na notificação são vinculados a PDPs locais, ou seja, cujo *peer* de gerenciamento possui acesso diretamente. Este passo é implementado através dos seguintes sub-passos:

   5.1 A primeira etapa para identificação dos PDPs responsáveis por cada recurso é identificar os PDPs existentes no ambiente de rede.

   5.2 A partir daí, o *peer* de gerenciamento deve requisitar para cada PDP quais dos recursos encontrados são de sua responsabilidade. A Figura 3.7 ilustra a existência de dois PDPs, PDP 1 e PDP 2, no mesmo domínio que o *peer* de gerenciamento. Assim, é possível que o *peer* de gerenciamento mapeie os recursos locais com os respectivos PDPs responsáveis.

6. Para os recursos que o *peer* não localizou PDP local responsável, este deve buscar outros *peers* que possuam acesso aos PDPs responsáveis por estes recursos. Isto é feito para que o *peer* que recebe a notificação possa aplicar políticas inclusive para PDPs não acessíveis diretamente, utilizando os outros *peers* como uma espécie de *proxy*. Este passo é implementado pela seguinte tarefa:

   6.1 É efetuada uma busca entre os demais *peers* a fim de identificar os *peers* que possuem acesso aos PDPs responsáveis pelos recursos externos. Na medida em que os *peers* de gerenciamento estão organizados em uma rede P2P, esta busca é feita com os protocolos padrões para busca de recursos disponíveis nas plataformas P2P. Isto significa que esta busca é efetuada de modo idêntico a uma busca de arquivo em uma rede P2P de compartilhamento de arquivos. No entanto, ao invés de passar a identificação do arquivo, é passada a identificação do recurso (e.g. endereço IP) como parâmetro da busca. A partir daí, o *peer* que recebeu a notificação consegue criar um mapeamento entre os recursos externos e o *peer* responsável por este recurso. A Figura 3.7 ilustra a busca em dois *peers* de gerenciamento, PGA 2 e PGA 3.

Neste ponto, o *peer* de gerenciamento já identificou quais recursos cujos PDPs responsáveis podem ser acessados diretamente e quais recursos devem ser acessados por intermédio de outros *peers*. Conforme ilustra a Figura 3.8, as tarefas seguintes são:

7. Consulta o gerente de recursos requisitando informações sobre os recursos requeridos. As informações necessárias para embasar a decisão do *peer* sobre que ações tomar são fortemente dependentes do contexto sob gerenciamento. Por exemplo, no estudo de caso apresentado na Seção 3.5, o *peer* de gerenciamento consulta o gerente de recursos sobre a largura de banda disponível em um determinado enlace para uma determinada CoS.

8. A partir das informações obtidas do gerente de recursos, o *peer* de gerenciamento irá planejar as políticas a serem aplicadas. Assim como no passo 3, neste momento

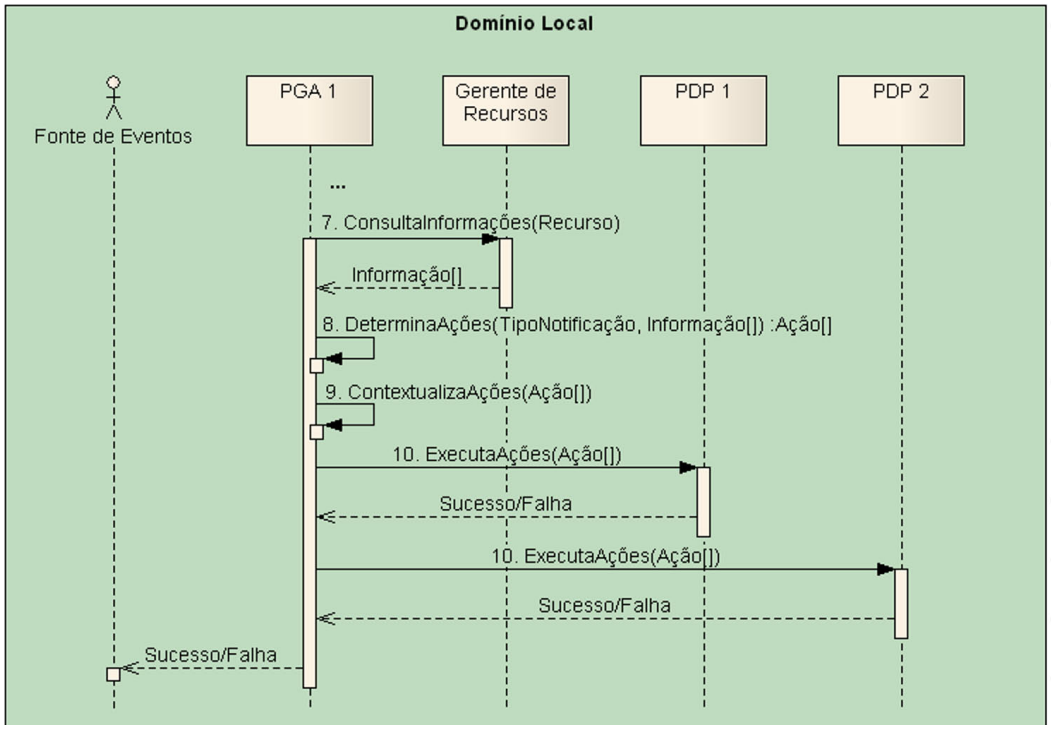

Figura 3.8: Planejamento e execução das ações de gerenciamento

o *peer* deve consultar a sua base de conhecimento, a qual indica que políticas devem ser aplicadas ou removidas em cada situação. Por exemplo, no estudo de caso apresentado na Seção 3.5, o *peer* verifica as políticas a serem aplicadas de acordo com: i) o tipo de notificação recebida (como inicialização de um serviço na grade computacional); ii) se a banda requerida está disponível segundo o gerente de recursos. Desta forma, de modo geral, a base de conhecimento referente às políticas é um mapeamento entre o tipo de notificação recebida, a disponibilidade de recursos segundo o gerente de recursos e as ações a serem executadas (inclusão/remoção de políticas). Cada política deve conter basicamente: a identificação da política, a condição que deve ser satisfeita para a aplicação das ações previstas na política e as ações que devem ser aplicadas no ambiente gerenciado caso as condições previstas sejam satisfeitas.

9. As ações presentes na base de conhecimento do agente são na verdade modelos (*templates*) de políticas a serem enviadas ao PDP do domínio. Assim, a próxima tarefa do *peer* é contextualizar estas políticas, isto é, substituir as variáveis presentes nas regras pelos valores relacionados a notificação.

10. O *peer* de gerenciamento, então, envia mensagem aos PDPs contendo as políticas a serem aplicadas. Cada PDP, antes de efetivamente aplicar as novas políticas, deve efetuar a análise das políticas atualmente implantadas no ambiente gerenciado. Esta análise, se implantada com a devida profundidade, pode evitar a inclusão de políticas em duplicidade ou conflitantes. A detecção e resolução de conflitos entre políticas não estão sendo tratadas nesta dissertação. (LUPU; SLOMAN, 1999) e (CHARALAMBIDES et al., 2005) apresentam propostas sobre o tema. Na arquitetura proposta, esta tarefa de validação das políticas a serem inseridas é de responsabilidade do PDP, o qual deve retornar ao *peer* uma mensagem de confirmação do su-

cesso ou não da aplicação das políticas planejadas. Em caso de sucesso, o PDP deve enviar as políticas para o repositório de políticas e aplicá-las nos PEPs de acordo com o modelo de distribuição adotado, *outsourcing* ou *configuration* (CHAN et al., 2001).

Por fim, o *peer* de gerenciamento de rede retorna mensagem de sucesso ou falha para a fonte de eventos. Esta mensagem informa se o *peer* de gerenciamento obteve sucesso na adaptação do ambiente gerenciado em resposta a notificação recebida.

3.5 Estudo de Caso

As grades computacionais tipicamente operam sobre complexas conexões através de diversos domínios administrativos, cada qual com suas características particulares. As aplicações da grade geralmente requerem meios de comunicação controlados, com suporte à QoS. Assim, é necessário um serviço de grade que permita ao seu administrador especificar requisitos de QoS envolvendo múltiplos domínios.

Neste trabalho, elaborou-se um protótipo da arquitetura apresentada no Capítulo 3 aplicada ao contexto de grades computacionais. Nesta abordagem, a arquitetura proposta ajusta as políticas referentes aos dispositivos de rede de acordo com as notificações enviadas pelo *toolkit* de grade. A plataforma para grade considerada foi o Globus Toolkit 4 (GT4) (GLOBUS PROJECT, 2007a). Esta escolha se deve ao grande uso deste *toolkit* e do seu suporte à *web-services*, o que facilita a construção de novos serviços para a comunicação com outros sistemas. Assume-se ainda que a rede utilizada para a interconexão dos nodos da grade possui suporte à DiffServ.

Esta seção descreve a adaptação da arquitetura para o estudo de caso analisado. Detalhes sobre a implementação deste estudo de caso são apresentados no Capítulo 4.

3.5.1 Operação

Neste estudo de caso, o papel de fonte de eventos é executado por um serviço do *toolkit* da grade. Assim, antes de qualquer participação do sistema de gerenciamento da rede, ocorre a interação do usuário com o *toolkit* da grade. Nesta etapa, usuário submete a sua aplicação para execução da grade. Além disso, fornece informações sobre a aplicação, como topologia, recursos necessários e tempo de execução previsto. O *toolkit* registra estas informações e atribui uma CoS a esta submissão. A CoS atribuída pode depender de diversos fatores, todos estes do domínio da grade.

Até esse ponto, o *toolkit* da grade colheu informações sobre as necessidades da aplicação do usuário e está pronto para iniciar a aplicação. O *toolkit* da grade, então, envia uma notificação ao *peer* de gerenciamento. Esta notificação é feita conforme a estrutura descrita na Seção 3.2.1. A partir daí, cabe ao sistema de gerenciamento da rede encontrar a melhor combinação entre as necessidades do usuário e a disponibilidade de recursos da infra-estrutura de rede.

A notificação informando a inicialização de uma aplicação da grade, faz com que o sistema de gerenciamento de rede crie políticas para que os roteadores de borda classifiquem os fluxos desta aplicação com a CoS requerida. É prevista ainda a notificação informando que um usuário está cancelando ou interrompendo a execução de uma aplicação da grade. O sistema de gerenciamento da rede deve, por sua vez, remover as políticas de classificação de pacotes que haviam sido inseridas para a aplicação específica.

O *peer* de gerenciamento que recebe a notificação passa a coordenar as ações para

adaptar as políticas da rede em resposta a notificação recebida. O gerente de recursos, por sua vez, opera como corretor (*broker*) de largura de banda. A seção a seguir descreve em mais detalhes a utilização do gerente de recursos como corretor de banda.

3.5.2 Gerente de Recursos

A utilização de um corretor (*broker*) para alocar dinamicamente a largura de banda da rede tem sido apresentada em diversas propostas envolvendo a arquitetura DiffServ. Com o seu uso, busca-se garantir que os enlaces intra-domínio e inter-domínios DiffServ sejam alocados de forma apropriada, evitando congestionamentos especialmente para serviços críticos.

A primeira proposta envolvendo corretores em redes DiffServ foi apresentada em (NICHOLS; JACOBSON; ZHANG, 1999). Esta introduz o uso de um corretor em cada domínio administrativo. Este corretor é responsável por configurar as políticas de diferenciação de serviços dos roteadores de borda. De forma semelhante, (TEITELBAUM et al., 1999) propõem uma arquitetura de diferenciação de serviços com a utilização de corretores de banda aplicada ao projeto Internet2. (HWANG et al., 2004) apresenta uma arquitetura similar, porém apresenta ainda a definição de um protocolo de comunicação entre corretores de diferentes domínios DiffServ e a implementação desta arquitetura. (MAGANA; SERRAT, 2005) introduz o uso de corretores de banda como suporte à operação de grades computacionais.

Nestas propostas, os usuários finais (fontes) efetuam uma requisição ao corretor informando a banda que será utilizada pela aplicação, o destino do fluxo e a CoS desejada. O corretor então avalia a disponibilidade de recursos, trocando informações inclusive com corretores de outros domínios envolvidos. Caso exista disponibilidade em todo o caminho percorrido pelo fluxo, os roteadores de borda são configurados para classificar os pacotes dos fluxos com a CoS requerida.

Neste estudo de caso buscou-se implementar uma solução semelhante a estas abordagens, porém utilizando agora a arquitetura proposta nesta dissertação. Assim, o papel de corretor de largura de banda é executado pelo gerente de recursos. A Figura 3.9 ilustra um cenário de uso do gerente de recursos. Neste cenário, a aplicação da grade está distribuída entre dois *hosts* localizados em domínios DiffServ distantes. A figura destaca a existência de um domínio de trânsito entre o domínio da origem e do destino. Destaca ainda a presença de um gerente de recursos (representado como GR) em cada domínio. O gerente de recursos apóia as decisões sobre as políticas a serem criadas pelo *peer* de gerenciamento. Estas políticas são aplicadas à plataforma de gerenciamento baseado em políticas, a qual, por fim, as aplicam nos roteadores de borda com suporte à DiffServ.

A visão do domínio do gerente de recursos é composta apenas pelos roteadores de borda e o seus respectivos *gateways*. Desta forma, o gerente de recursos registra a largura de banda utilizada pelos serviços gerenciados (aplicações da grade) somente nos enlaces WAN. A relação de enlaces WAN é obtida através de requisições periódicas a um PDP do domínio. A forma que o PDP busca esta relação não deve influenciar a operação do sistema. Este pode utilizar, por exemplo, uma linguagem semelhante ao SQL como a proposta em (CECCON et al., 2003).

A quantidade de largura de banda é alocada por usuário e aplicação em cada enlace WAN. Esta alocação é classificada ainda conforme a CoS a qual o fluxo foi classificado. A Tabela 3.1 dá um exemplo de informações armazenadas por um gerente de recursos.

A partir destas informações armazenadas pelo gerente de recursos, o *peer* de gerenciamento define se a CoS definida pelo *toolkit* da grade pode ser atendida pelo sistema de

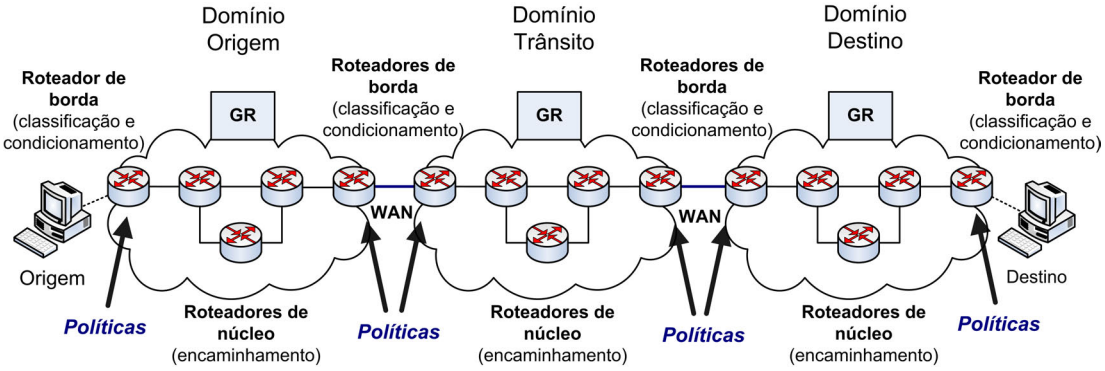

Figura 3.9: Cenário envolvendo três domínios adjacentes

gerenciamento da rede.

3.5.3 Comunicação inter-domínio

Para que o fluxo da grade seja classificado com a CoS requerida em todo o caminho percorrido é necessário que exista:

- Um gerente de recursos e ao menos um *peer* de gerenciamento em cada domínio DiffServ percorrido pelo fluxo;
- Disponibilidade de largura de banda para a CoS requerida em todos enlaces WAN.

A comunicação entre diferentes domínios se dá através dos *peers* de gerenciamento. Conforme descrito na Seção 3.4, o *peer* que recebe a notificação da fonte de eventos determina as ações a serem efetuadas. Neste estudo de caso, o *peer* deve, após validar a notificação recebida, determinar quais dos nodos da aplicação estão localizados no mesmo domínio administrativo do *peer* e quais estão localizados em domínios externos.

Para um fluxo com origem em um *host* localizado no domínio local e com destino a um *host* externo, o *peer* de gerenciamento deve consultar o PDP do domínio de origem afim de identificar para qual *gateway* aquele fluxo será encaminhado. O *peer* deve então consultar o gerente de recursos para verificar a disponibilidade de banda para a CoS e horário previsto para a execução da aplicação. A partir daí, o *peer* efetua uma busca para descobrir qual dos demais *peers* de gerenciamento é responsável por este *gateway* (roteador de borda). Caso a busca retorne ao menos um *peer* relacionado ao *gateway*, significa que o domínio administrativo adjacente também implementa a plataforma de gerenciamento proposta. Assim, basta ao *peer* de gerenciamento enviar uma mensagem ao *peer* do domínio adjacente repassando às informações sobre a aplicação da grade. O *peer* do domínio adjacente efetua um conjunto de ações idêntico. Por fim, este retorna uma mensagem ao *peer* do domínio de origem informando o sucesso ou não da reserva de banda para a CoS e horário requerido. Em caso de sucesso, o *peer* do domínio de origem

Tabela 3.1: Informações sobre uso de banda armazenada pelo gerente de recursos

Enlace WAN	IDUsuário	IDAplicação	CoS	Banda (bit/s)	Horário início	Horário fim
192.25.0.1/24 - 192.25.0.2/24	johnL	service01	EF	1200	20070901T050000	20070901T170000
192.25.0.1/24 - 192.25.0.2/24	ucla	bioAnalisys	AF	2400	20070903T230000	20070904T050000
192.25.1.6/24 - 192.25.1.7/24	ufrgs	forecast	AF	14500	20070915T220000	20070917T053000

pode efetivamente elaborar as políticas para os roteadores de borda e retornar mensagem de sucesso à fonte de eventos.

Esta forma de interação entre os *peers* de gerenciamento é especialmente útil para operações que envolvam reserva de banda. Isto porque neste tipo de operação é desejável que esta seja efetuada de forma transacional. Caso algum dos domínios percorridos pelo fluxo não permita a reserva de banda na CoS requerida, toda a operação falha. Um fluxo que envolva apenas *hosts* dentro do próprio domínio possui tratamento semelhante, porém sem a necessidade de envolver outros *peers* no processo.

Para um fluxo com origem em um *host* localizado em um domínio externo e com destino também a um *host* localizado fora do domínio do *peer* de gerenciamento, não cabe a este *peer* providenciar a reserva de banda para o fluxo. Desta forma, o *peer* deve efetuar uma busca para localizar ao menos um *peer* responsável pelo *host* origem do fluxo. O *peer* que recebeu a notificação, então, a repassa a um dos *peers* deste outro domínio, o qual fica responsável pela reserva de banda.

Há o caso ainda no qual a aplicação da grade não é distribuída (é executada apenas por um *host*). Neste caso, não há ações a serem executadas pelo *peer* de gerenciamento. Assim, este retorna uma mensagem de sucesso à fonte de eventos.

4 PROTOTIPAÇÃO

Com a implementação deste protótipo buscou-se avaliar especialmente:

- Se o GT4 é capaz de fornecer as informações para o funcionamento do sistema de gerenciamento proposto;
- A viabilidade e a melhor forma de interação entre o GT4 e os *peers* de gerenciamento;
- A comunicação entre os *peers* de gerenciamento, em particular, a técnica de fazer buscas entre os *peers*;

Este capítulo visa descrever o protótipo implementado e o resultado destas avaliações.

4.1 Componentes implementados

Como forma de reduzir a complexidade do protótipo e o seu custo de desenvolvimento, escolheu-se construir apenas os componentes mais críticos da arquitetura. A avaliação de quais componentes seriam mais críticos foi focada nos pontos da arquitetura em que haviam poucos ou nenhum trabalho com implementação semelhante. Dentre as camadas da arquitetura, ilustradas pela Figura 3.1, a camada de políticas é a que possui maior número de propostas, como (MARQUEZAN et al., 2005), por exemplo. Desta forma, o protótipo implementado foi focado na camada P2P e, principalmente, na sua interação com o GT4. A descrição da implementação dos *peers* de gerenciamento de rede é feita na Seção 4.4.

Outro ponto importante é a avaliação dos serviços disponibilizados pelo GT4. Para que o GT4 possa desempenhar corretamente o papel de fonte de eventos da arquitetura proposta é preciso que esta possua: i) um serviço que forneça informações sobre os usuários e as aplicações da grade, como topologia, largura de banda necessária e tempo de execução previsto; e ii) uma interface que permita notificar os *peers* de gerenciamento da ocorrência de eventos. A Figura 4.1 apresenta um diagrama em blocos dos componentes que compõe o GT4 agrupados nas categorias: segurança, gerência de dados, gerência de execução, serviços de informação e bibliotecas de suporte.

Os componentes apresentados na Figura 4.1 relativos aos serviços de informação são comumente chamados de Monitoring and Discovery Services (MDS). Em particular, os componentes deste tipo de serviço baseados em *web-services* são chamados de WS-MDS (ou MDS4). Os WS-MDS são geralmente instalados em cada *host* ou *cluster* que compõe a grade e têm por função monitorar os recursos utilizados pela grade destes *hosts* ou *clusters* . A obtenção destes dados de monitoramento é feita através de serviços intitulados

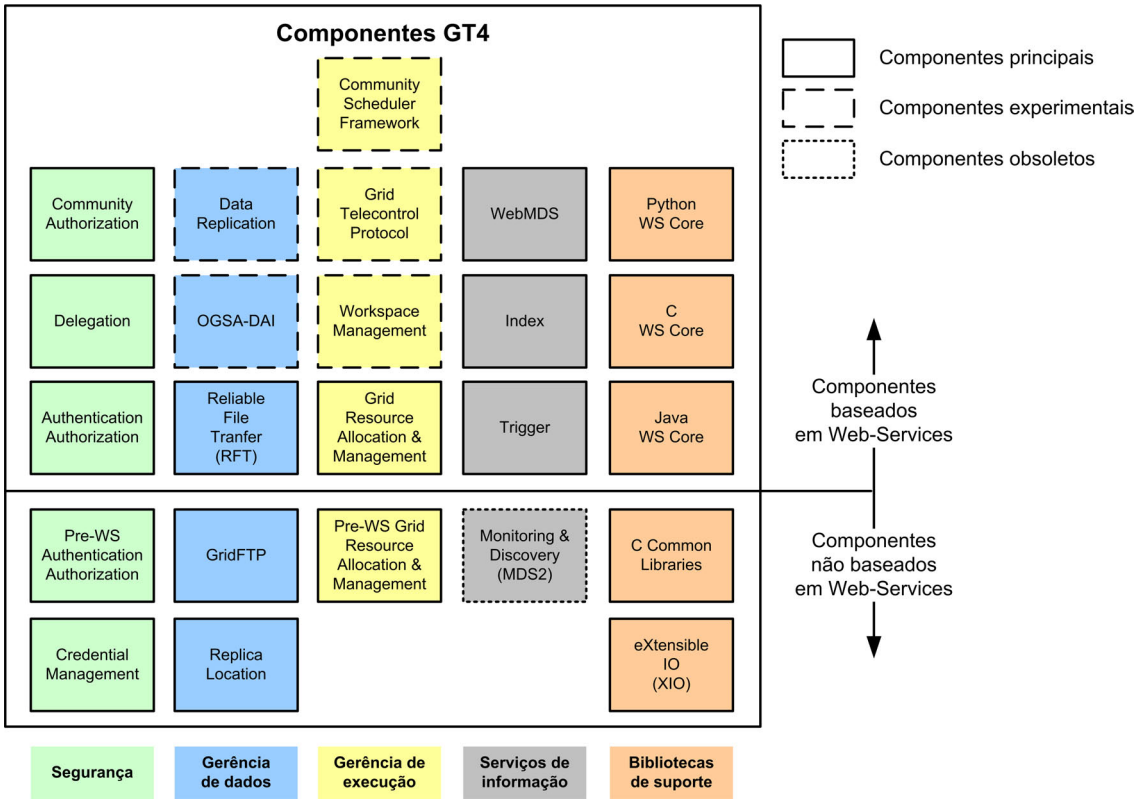

Figura 4.1: Componentes do Globus Toolkit 4

"fontes de informação". Desta forma, os componentes que compõem o WS-MDS seriam adequados para fornecer as informações requeridas pelo sistema de gerenciamento de rede. Porém, a versão atual do WS-MDS fornece as seguintes fontes de informação:

- WS GRAM: Publica informações sobre o escalonador local do serviço de submissão de *jobs* do GT4, incluindo: *jobs* na fila, número de CPUs disponíveis, número de *jobs* atendidos e algumas estatísticas sobre a memória.

- Hawkeye Information Provider: Provedor de informações sobre o escalonador de *jobs* Condor, quando este é utilizado de forma integrada ao GT4. Publica informações como: nome e identificação do *host*, informações sobre o processador, memória, sistema operacional e sistema de arquivos.

- Ganglia Information Provider: Responsável por publicar informações sobre os recursos que utilizam o sistema de monitoramento Ganglia (UNIVERSITY OF CALIFORNIA, 2007a), tais como: nome e identificação do *host*, carga do processador, nome e versão do sistema operacional e do sistema de arquivos.

- Reliable File Transfer Service (RFT): Disponibiliza informações sobre o componente de transferência de arquivos do GT4, como: estado do servidor, estado das transferências efetuadas e em curso e número de transferências ativas.

Como nenhum destes serviços provê as informações requeridas, foi necessário desenvolver uma nova fonte de informação, integrada ao WS-MDS. A fonte de informação desenvolvida foi chamada de InformationService e descrita em detalhes na Seção 4.3.2.

Conforme descrito anteriormente, além do serviço de informação, é necessário que o *toolkit* da grade possua uma interface que permita notificar os *peers* de gerenciamento da ocorrência de eventos. Em especial, os *peers* de gerenciamento de rede deveriam ser notificados toda vez que um usuário da grade submete sua aplicação para ser executada, cancela ou interrompe esta execução. Um serviço de notificação deste tipo poderia ser implementado diretamente no GT4. O serviço Trigger, provido por padrão no GT4, permite agregar dados de diversas fontes de informação e compará-los contra um conjunto de regras definido em um arquivo de configuração. Quando uma condição é satisfeita, o serviço executa uma ação também definida no arquivo de configuração. Esta ação é, em geral, um *script* escrito em alguma das linguagens suportadas pelo GT4.

Poderia-se assim definir no serviço Trigger uma regra que notificasse os *peers* de gerenciamento (assinantes) a cada nova aplicação submetida à grade, cancelamento ou interrupção de aplicações. Porém, como o objetivo do protótipo é o de apenas simular submissões de aplicações à grade e não o de submeter aplicações reais, desenvolveu-se um serviço de notificação que desempenha este papel. O serviço implementado, ao invés de utilizar informações obtidas a partir de escalonadores para "disparar" ações de notificação aos assinantes, é disparado remotamente de forma manual. Assim, o serviço implementado, chamado de NotificationService, possui uma interface *web-service*, seguindo o padrão WS-Notification, para receber assinaturas dos *peers* de gerenciamento e outra interface também *web-service* para simular a recepção de eventos de inicialização, cancelamento ou interrupção de aplicação da grade. A implementação deste serviço é descrita em mais detalhes na Seção 4.3.1.

Desta forma, resumindo a função de cada componente implementado, o NotificationService é o responsável por notificar os *peers* de gerenciamento do evento ocorrido. O *peer* que recebeu a notificação a avalia e, se necessário, busca mais informações sobre o evento da grade no InformationService. Este *peer* notifica os demais *peers* relacionados, se houverem, e formula as políticas a serem aplicadas ou removidas.

4.2 Definição de tecnologias

A primeira decisão a ser tomada para a implementação dos componentes do protótipo foi a escolha da plataforma P2P a ser utilizada no protótipo. O requisito principal para esta escolha é a capacidade da plataforma de implementar as funcionalidades previstas. O JXTA atende estes requisitos, já que possui: suporte nativo a busca de recursos entre *peers*; os *peers* são identificados por uma chave gerada de 160 bits, fazendo com que o *peer* possa mudar de endereço IP mantendo o número de identificação e a conectividade com a rede; a rede P2P se auto-organiza através de um protocolo de descoberta; e permite a inclusão de grupos de *peers*. Além disso, a plataforma JXTA é a plataforma P2P de uso geral com maior número de usuários, projetos implementados e documentação disponível, fator também importante para o ciclo de vida da plataforma. É ainda projetada para ser independente de linguagem de programação, de ambiente de desenvolvimento e de plataforma de distribuição (GONG, 2001).

Outras plataformas P2P de uso geral possíveis de serem utilizadas são Chord (STOICA et al., 2001) e Chimera (UNIVERSITY OF CALIFORNIA, 2007b). Ambas foram implementadas em C e nasceram de projetos acadêmicos. A plataforma Chord, segundo os próprios autores, não possui ainda uma versão estável disponível. Já Chimera apesar de, segundo os autores, ser utilizada também em projetos comerciais, apresenta documentação incipiente.

Além da plataforma JXTA, foram utilizadas ainda bibliotecas adicionais, as quais agregam funcionalidades à plataforma. Assim, foi utilizada a biblioteca JXTA-SOAP (DISTRIBUTED SYSTEMS GROUP, 2006). Esta biblioteca permite que a comunicação entre os *peers* da rede se dê através do protocolo SOAP. O uso de SOAP, especialmente sobre HTTP ou HTTPS, reduz a probabilidade do tráfego envolvendo *peers* distantes ser bloqueado por *firewalls* localizados entre os domínios. Foi utilizada ainda parte da estrutura utilizada pela ferramenta de gerenciamento ManP2P (GRANVILLE et al., 2005), descrita na Seção 2.4. Em particular, foram reutilizados os métodos utilitários chamados na inicialização dos *peers* e as interfaces definidas na ferramenta.

Uma vez escolhida a plataforma P2P, o próximo passo é definir qual padrão de comunicação será adotado para interação entre a fonte de eventos (*toolkit* da grade, neste caso) e os *peers* de gerenciamento. A Seção 3.2.1 sugere que esta comunicação siga o modelo Publicar/Assinar. Dentre as possibilidades de implementação deste modelo está o padrão WS-Notification. Sendo este padrão suportado nativamente pelo GT4, esta foi a escolha natural para a implementação do protótipo.

4.3 Serviços implementados no GT4

Para este estudo de caso foram desenvolvidos dois novos serviços: NotificationService e InformationService. Estes serviços foram implementados de forma integrada à estrutura do GT4. Em especial, estão integrados à um conjunto de bibliotecas chamado Java WS Core (Figura 4.1). O GT4 é um *software* extremamente modular, composto por diversos componentes, os quais podem ser instalados em conjunto ou em separado. Esta modularidade que, por um lado, permite que o *toolkit* seja personalizado de acordo com as necessidades do ambiente, faz com que a sua instalação e o desenvolvimento de componentes para o *toolkit* sejam tarefas relativamente complexas. O GT4 faz uso de entorno de 150 bibliotecas na linguagem C, 100 bibliotecas Java e 150 descrições de interfaces *web-services* na linguagem Web Services Description Language (WSDL). Para o desenvolvimento e execução do protótipo, é necessária a instalação dos serviços de informação baseados em *web-services* do GT4, além das bibliotecas presentes no componente Java WS Core ((GLOBUS PROJECT, 2007b) apresenta um guia de instalação rápida do GT4).

Após a configuração do ambiente, pode-se iniciar a implementação dos serviços. As seções a seguir descrevem a metodologia adotada e os componentes implementados em cada serviço.

4.3.1 NotificationService

O roteiro adotado para a construção do NotificationService foi o seguinte:

1. Construir as classes do NotificationService;
2. Construir as classes do assinante destas notificações;
3. Implementar componente que irá atuar como produtor de notificações, simulando eventos gerados pelo GT4;
4. Criar arquivo WSDL, o qual descreve as interfaces *web-services* utilizadas pelo serviço;
5. Construir arquivo XML para o Ant (APACHE PROJECT, 2007) para a compilação e distribuição da implementação do serviço.

O NotificationService é composto por quatro classes principais, todas localizadas no pacote "org.globus.www.notificationService". A Figura 4.2 apresenta um diagrama de classes contendo estas classes e as suas classes ancestrais, pertencentes às bibliotecas do Java WS Core.

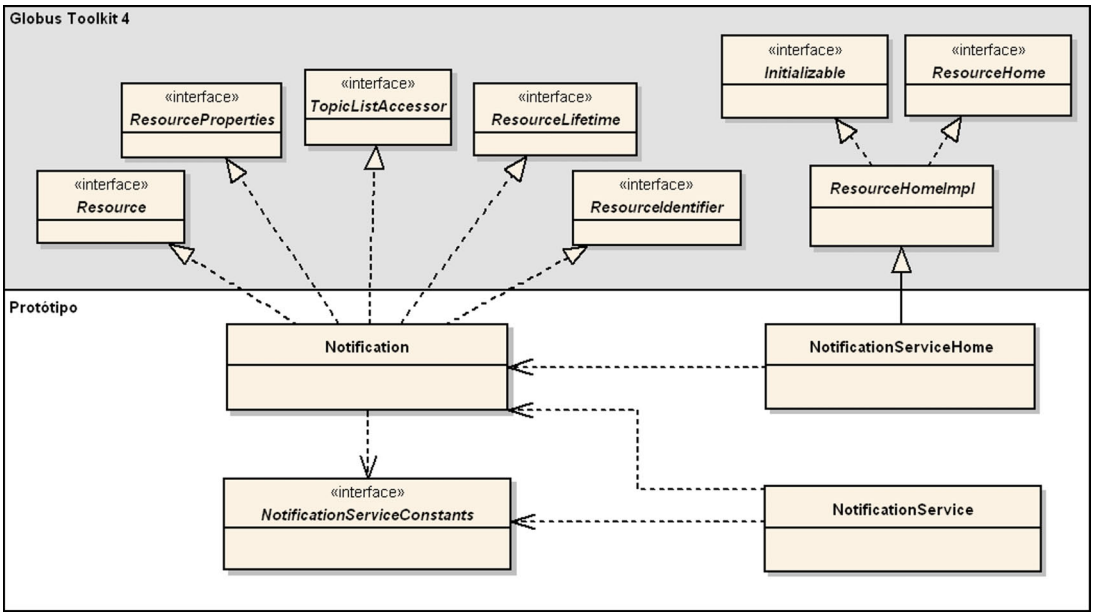

Figura 4.2: Diagrama de classes do NotificationService

A classe **`Notification`** representa a entidade notificação em si. Isto significa que cada notificação a ser emitida pelo serviço é tratada internamente como sendo uma instância da classe `Notification`. Esta classe implementa as interfaces `Resource`, `ResourceProperties`, `TopicListAccessor`, `ResourceLifetime` e `ResourceIdentifier`.

A interface `Resource` tem a função apenas de identificar a classe como sendo um recurso. Já a interface `ResourceProperties` define o método `getResourcePropertySet()`, o qual deve retornar as propriedades (atributos e valores) do recurso. Para isto, foi definido um atributo `propSet` na classe `Notification`, o qual é preenchido na inicialização dos objetos da classe `Notification` (método `initialize()`) com um conjunto de pares nome-valor. Assim, quando o GT4 chama o método `getResourcePropertySet()`, a fim de identificar as propriedades do serviço, este método retorna o valor do atributo `propSet` já preenchido. A principal propriedade definida é a própria mensagem da notificação. A mensagem armazena, em formato XML, as seguintes informações: i) a identificação do emissor da mensagem (serviço de notificação da grade); ii) o tipo de evento que gerou a mensagem; iii) o nome do usuário; iv) a identificação da aplicação e v) o estado da aplicação (iniciada ou terminada).

A indicação de que o recurso está associado a um tópico do padrão Publicar/Assinar é feita através da interface `TopicListAccessor`. Esta interface define o método `getTopicList()`, o qual retorna a lista de tópicos aos quais este recurso está presente. Ao inicializar um objeto da classe `Notification` (método `initialize()`), esta lista de tópicos é inicializada com a identificação do tópico relacionado ao NotificationService.

`ResourceLifetime` é uma interface que possui métodos para identificar o tempo de vida do recurso. Assim, é possível definir o tempo que uma notificação é válida. Após o tempo definido, se nenhum assinante receber a notificação, esta é descartada. Já a

interface `ResourceIdentifier` possui como única função definir o método `getId()`, o qual deve retornar a identificação do recurso. O protótipo utiliza um número gerado aleatoriamente como identificador. Estando a classe `Notification` com estas interfaces devidamente implementadas, está pronta para ser utilizada pelo serviço de notificação.

Outra classe importante do NotificationService é a classe **`NotificationServiceHome`**. Esta classe estende a classe do GT4 chamada `ResourceHomeImpl`. Desta forma, `ResourceHomeImpl` implementa todo o comportamento padrão da classe. `NotificationServiceHome` apenas sobrescreve os métodos aos quais deseja implementar um comportamento diferente do padrão.

É possível entender a função da classe `ResourceHomeImpl` (e por conseqüência da classe `NotificationServiceHome`) através das interfaces que esta implementa. `ResourceHomeImpl` implementa as interfaces `ResourceHome` e `Initializable`. `ResourceHome` define uma interface chamada pelo GT4 para lidar com conjuntos de recursos de mesmo tipo, em operações como: localizar ou remover um recurso no conjunto a partir de sua chave, criar e incluir um novo recurso ao conjunto e manter um *cache* dos recursos. A interface `Initializable` define o método `initialize()`, o qual é chamado para indicar à classe que o implementa que esta deve inicializar seus atributos neste momento. Assim, como `ResourceHomeImpl` implementa esta interface, o tratamento de inicialização do conjunto de recursos gerenciados pela classe é feito neste método.

Na medida em que `NotificationServiceHome` estende `ResourceHomeImpl` e está relacionada ao recurso `Notification`, ações sobre o conjunto de notificações são de responsabilidade de `NotificationServiceHome`. Conforme já dito anteriormente, **`NotificationServiceHome`** apenas sobrescreve os métodos aos quais deseja implementar um comportamento diferente do padrão. Assim, sobrescreve os métodos `create()`, `add( ResourceKey key, Resource resource)` e `remove( ResourceKey key)` para, além de realizar o comportamento padrão de cada método, promover o registro e remoção de registro do recurso no *container* WS MDS. Ao criar uma notificação (método `create()`), a classe chama o método `add(ResourceKey key, Resource resource)` a fim de adicionar a notificação ao conjunto de notificações do serviço. Porém, o método `add(ResourceKey key, Resource resource)` sobrescrito pela classe `NotificationServiceHome`, além de adicionar a notificação ao conjunto, a registra no *container* WS MDS. Como o NotificationService trata-se de uma fonte de informações, tal como outras apresentadas na Seção 4.1, após este momento, a notificação criada passa a estar "visível" para quem acessa o WS MDS. Além disso, a estrutura Publicar/Assinar do GT4 pode então enviar uma mensagem aos assinantes do serviço NotificationService, informando a criação de uma nova notificação.

A classe **`NotificationServiceConstants`**, como o próprio nome identifica, tem como única função armazenar as constantes utilizadas pelo serviço.

O ponto de entrada da interface *web-service* do NotificationService é implementada pela classe **`NotificationService`**. O método principal da classe, chamado `write( String text)`, é o método utilizado para simular um evento gerado pelo GT4. Conforme já descrito, ao invés de receber eventos do próprio GT4 informando a inicialização ou cancelamento da execução de uma aplicação da grade, no protótipo este evento é gerado manualmente. Para isto foi criada uma pequena aplicação Java chamada **`NotifyService`**.

`NotifyService` estende a classe do GT4 chamada `BaseClient`, a qual provê as funcionalidades padrão para clientes dos serviços WS MDS. `NotifyService` foi implementada de forma a ser executado por linha de comando e receber como parâmetro: o

nome do usuário, identificação da aplicação, tipo de evento e estado da aplicação. A partir destes parâmetros é gerada uma mensagem XML a ser enviada ao NotificationService. Porém, para localizar o serviço é preciso fazer uso de um localizador de *web-services*. Este localizador, implementado pela classe `WSResourcePropertiesServiceAddressingLocator`, permite obter uma conexão com o *web-services* a partir do seu endereço (armazenado em um arquivo ".epr"). A partir desta conexão, é chamado o método `write(String text)` passando como parâmetro a notificação em XML. O serviço de notificação criado recebe esta mensagem através do método `write(String text)` da classe **`NotificationService`**, já descrito anteriormente.

`NotifyService` recebe então a confirmação de que a notificação foi enviada. Uma mensagem de sucesso é impressa no console, como ilustrado pela Figura 4.3.

```
globus@kde: /notificationService> ./notify-service -e gridNotificationService.epr
ServiceDeploy johnL servico01 start
Notification Accepted.
(Message Type:  ServiceDeploy User:johnL Service:servico01 Status:  start)
```

Figura 4.3: Notificação por linha de comando

Após estes passos, o serviço NotificationService está devidamente implementado. Porém, resta ainda criar os arquivos WSDL que descrevam as interfaces *web-services* utilizadas pelo serviço para que o *container* do Java WS Core saiba da existência do novo serviço.

A definição das interfaces *web-services* do serviço é feita por um documento WSDL, o qual possui como principais elementos: *portType* e *binding*. O elemento *portType* define as operações providas pelo serviço e as mensagens envolvidas. Por exemplo, a Figura 4.4 descreve um trecho do elemento *portType*. Neste trecho, é definido que o serviço possui a operação "Subscribe", a qual permite aos clientes assinarem o serviço de notificação.

```
<portType name="NotificationServicePortType"
  wsrp:ResourceProperties="NotificationServiceResourceProperties">
  <operation name="Subscribe">
    <input message="wsntw:SubscribeRequest"
      wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-
      WS-BaseNotification/Subscribe"/>
    <output message="wsntw:SubscribeResponse"
      wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-
      WS-BaseNotification/SubscribeResponse"/>
    <fault name="TopicPathDialectUnknownFault"
      message="wsntw:TopicPathDialectUnknownFault"/>
    <fault name="SubscribeCreationFailedFault"
      message="wsntw:SubscribeCreationFailedFault"/>
    <fault name="ResourceUnknownFault"
      message="wsntw:ResourceUnknownFault"/>
    </operation>
  ... Outras operações ...
</portType>
```

Figura 4.4: Definição WSDL NotificationService - Elemento *portType*

Já o elemento *binding* define o formato da mensagem e detalhes do protocolo utilizado pelo *web-service*. A Figura 4.5 exibe um trecho do elemento *binding* do NotificationService. Neste trecho, o atributo *type* do elemento *wsdl:binding* o relaciona com o elemento *portType* exibido na Figura 4.4. A seguir, o elemento *soap:binding* define que protocolo

SOAP será implantado sobre o protocolo HTTP. Então, são relacionadas quais das operações definidas no elemento *portType* que serão disponibilizadas utilizando o protocolo SOAP. Para cada operação são definidos os tipos de dados dos parâmetros de entrada e saída.

```
<wsdl:binding name="NotificationServicePortTypeSOAPBinding"
   type="porttype:NotificationServicePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Write">
    <soap:operation
     soapAction="http://www.globus.org/notificationService/NotificationServicePortType/
     WriteRequest"/>
     <wsdl:input>
     <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>

  ... Outras operações ...
</wsdl:binding>
```

Figura 4.5: Definição WSDL - Elemento *binding*

Por fim, foi criado um *script* em XML a ser utilizado pelo Ant. Este *script* facilita a implantação do protótipo, já que automatiza a compilação, empacotamento das classes compiladas e demais arquivos em um arquivo "gar" (padrão do GT4) e copia este arquivo para o *container* do GT4.

4.3.2 InformationService

O InformationService foi criado para atender às requisições dos *peers* de gerenciamento de rede sobre informações de usuários e aplicações da grade em execução ou agendadas. Seu funcionamento simula o conhecimento do GT4 sobre a topologia das aplicações e sobre a largura de banda requerida entre os nodos da aplicação. Simula ainda a classificação de CoS do par usuário-aplicação, como se o GT4 possuísse políticas de diferenciação de serviços no nível da grade. Assim, um determinado usuário executando uma certa aplicação seriam classificados com uma CoS, como *Assured Forwarding* (AF), *Expedited Forwarding* (EF) ou *Best Effort* (BE). Os *peers* de gerenciamento buscam replicar esta diferenciação para o nível de rede.

A partir destes requisitos, o InformationService foi implementado com apenas uma interface *web-service*, a qual atende à requisições de informação sobre usuários e/ou aplicações. Assim, diferentemente do NotificationService, o serviço não requer a implementação do modelo Publicar/Assinar, já que os clientes do serviço não se relacionam com este como assinantes. Informações são requeridas ao serviço conforme as necessidades dos clientes. A Figura 4.6 apresenta as classes do serviço e as suas classes ancestrais.

Apesar de possuir apenas uma interface, o InformationService possui uma estrutura bastante semelhante ao NotificationService, agora localizada no pacote "org.globus.
www.infomationService". A classe **`Information`** é análoga a classe `Notification`, descrita na Seção 4.3.1. Representa a informação a ser enviada aos *peers* de gerenciamento de rede. Implementa as mesmas interfaces que a classe `Notification`, com exceção de `TopicListAccessor`. Isto porque a interface `TopicListAccessor` está relacionada ao modelo Publicar/Assinar, não implementado pelo InformationService. As demais interfaces possuem as mesmas funções descritas no NotificationService. `Resource-`

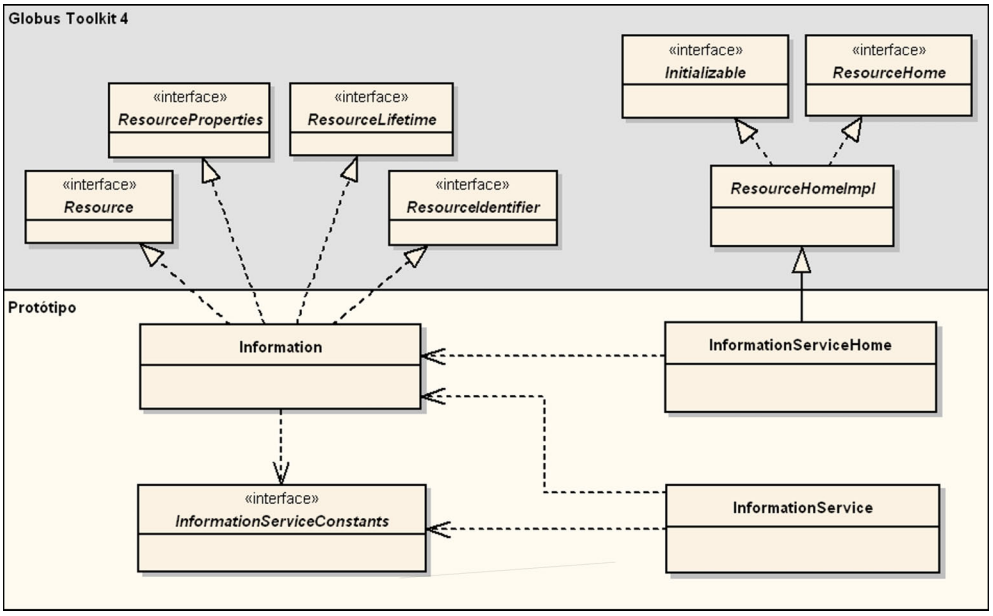

Figura 4.6: Diagrama de classes do InformationService

`Lifetime`, por exemplo, identifica o tempo de vida de uma informação prestada pelo InformationService.

As demais classes do serviço, **`InformationServiceHome`**, **`InformationServiceConstants`** e **`InformationService`**, possuem implementações praticamente idênticas ao NotificationService. Apenas a classe **`InformationService`** não implementa o método `write(String text)`, já que este é utilizado para simular a recepção de um evento gerado pelo GT4.

Desta forma, resta criar a definição WSDL sobre a interface *web-service* do serviço. Como o serviço possui apenas uma interface *web-service*, a definição WSDL possui menos operações definidas. A principal delas é apresentada na Figura 4.7 e representa a consulta de valores ao serviço.

```
<portType name="InformationServicePortType"
  wsrp:ResourceProperties="InformationServiceResourceProperties">
  <operation name="GetResourceProperty">
    <input message="GetResourcePropertyRequest"
      message="wsrpw:GetResourcePropertyRequest"
      wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/
      wsrf-WS-ResourceProperties/GetResourceProperty"/>
    <output name="GetResourcePropertyResponse"
      message="wsrpw:GetResourcePropertyResponse"
      wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/
      wsrf-WS-ResourceProperties/GetResourcePropertyResponse"/>
    <fault name="InvalidResourcePropertyQNameFault"
      message="wsrpw:InvalidResourcePropertyQNameFault"/>
    <fault name="ResourceUnknownFault"
      message="wsrpw:ResourceUnknownFault"/>
    </operation>
  ... Outras operações ...
</portType>
```

Figura 4.7: Definição WSDL InformationService - Elemento *portType*

Após criado o *script* em XML a ser utilizado pelo Ant, a implementação do serviço está finalizada.

4.4 *Peer* de gerenciamento de rede

A metodologia adotada para a criação da camada P2P do protótipo foi a seguinte:

1. Construção das classes principais para que o *peer* consiga acessar a rede P2P;
2. Identificação das entidades do domínio e criação das classes respectivas;
3. Integração com serviço de notificação da grade (NotificationService);
4. Implementar planejamento para determinar as tarefas a serem executadas para cada tipo de notificação recebida;
5. Integração com serviço de informação da grade (InformationService);
6. Implementar identificação dos recursos envolvidos na notificação e dos PDPs ou *peers* com acesso a estes recursos;
7. Criação do gerente de recursos como uma base de dados centralizada;
8. Notificar demais *peers* relacionados aos recursos;
9. Montar mensagem a ser enviada para a plataforma de políticas.

Cada um destes passos são descritos a seguir.

4.4.1 Classes principais

Conforme já descrito anteriormente, o protótipo foi desenvolvido utilizando a plataforma JXTA, além de bibliotecas como JXTA-SOAP e algumas das classes e interfaces utilizadas na ferramenta ManP2P. A Figura 4.8 apresenta as classes principais que compõe a estrutura dos *peers* de gerenciamento de rede.

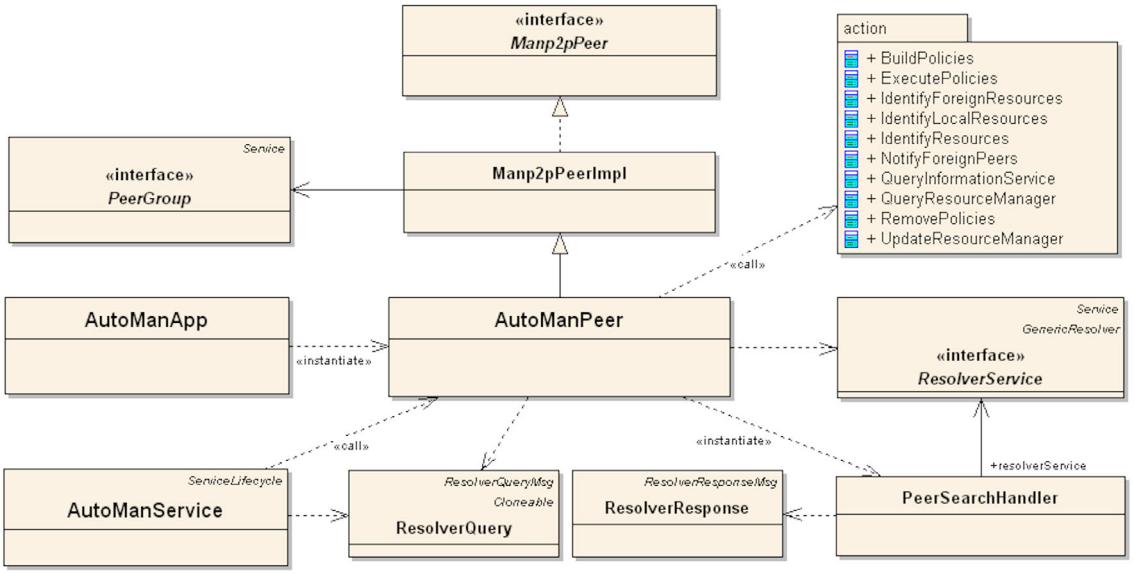

Figura 4.8: Classes principais do *peer* de gerenciamento de rede

As três classes principais, exibidas na Figura 4.8 em negrito, são: **`AutoManApp`**, **`AutoManService`** e **`AutoManPeer`**. Estas classes, em conjunto com a classe **`Peer-`**

SearchHandler e as classes contendo as ações (pacote **action**), foram desenvolvidas especificamente para o protótipo e são as responsáveis pelo comportamento esperado.

`AutoManApp` é o ponto de entrada da aplicação, já que é ela que possui o método `main(String[] args)`. Assim, é responsável por inicializar o *peer* de gerenciamento, através da instanciação de um objeto da classe `AutoManPeer` e da chamada ao método `start()` desta classe. A partir daí, a classe `AutoManPeer` passa a ser responsável por controlar as ações do *peer*.

Quando o método `start()` da classe `AutoManPeer` é chamado, este primeiramente chama o método `start()` da sua classe ancestral `Manp2pPeerImpl` (`super.start()`). `Manp2pPeerImpl` provê a inicialização do *peer* através das seguintes operações:

- Configura variáveis de ambiente a partir de arquivos de configuração, como nome e descrição do *peer*, senha para autenticação, identificação dos grupos aos quais o *peer* irá pertencer, endereço de *peers rendezvous* e de *peers relay*;
- Inicializa a plataforma JXTA;
- Autentica o *peer* em um serviço de segurança;
- Insere o *peer* nos grupos especificados;
- Conecta a um dos *rendezvous* configurados;
- Ativa os *web-services* implementados no *peer*.

Após a finalização do método `start()` da classe ancestral, `AutoManPeer` registra o seu mecanismo de busca nos grupos aos quais o *peer* faz parte. Este mecanismo de busca, implementado especificamente para o protótipo, consiste na classe `PeerSearchHandler` e é registrado em um serviço chamado **ResolverService**. Após este registro, o *peer* passa a estar disponível para ser utilizado pela classes de ações para receber consultas de outros *peers* e para efetuar consultas a outros *peers*.

4.4.2 Entidades

Uma entidade é qualquer coisa, concreta ou abstrata, incluindo associações entre entidades, abstraídos do mundo real e modelado de forma a poder ser armazenada. Desta forma, para a elaboração do protótipo foi necessário identificar as entidades envolvidas no domínio da grade. Caso os *peers* de gerenciamento fossem aplicados a outro domínio, novas entidades deveriam ser identificadas.

A Figura 4.9 apresenta as entidades identificadas para o protótipo, com os seus respectivos atributos (métodos foram suprimidos). Estas entidades compõem a estrutura utilizada pelos *peers* de gerenciamento para representar as informações obtidas a partir das notificações recebidas e das consultas ao InformationService. As classes de entidade criadas seguem o padrão de projeto *Value Object*, o qual define que as classes de entidade devem possuir apenas atributos e métodos de acesso a estes atributos.

Dentre estas classes de entidade, **VONotification** é a de mais alto nível. Isto porque é dela que derivam as demais classes de entidade. Além dos relacionamentos, possui o atributo `eventType`, o qual é uma enumeração para identificar o tipo de evento que deu origem à notificação. No protótipo foram utilizados os tipos de eventos: “Service Deploy” e “Service Undeploy”, os quais referem-se aos eventos de inicialização e cancelamento (ou interrupção) de uma aplicação da grade.

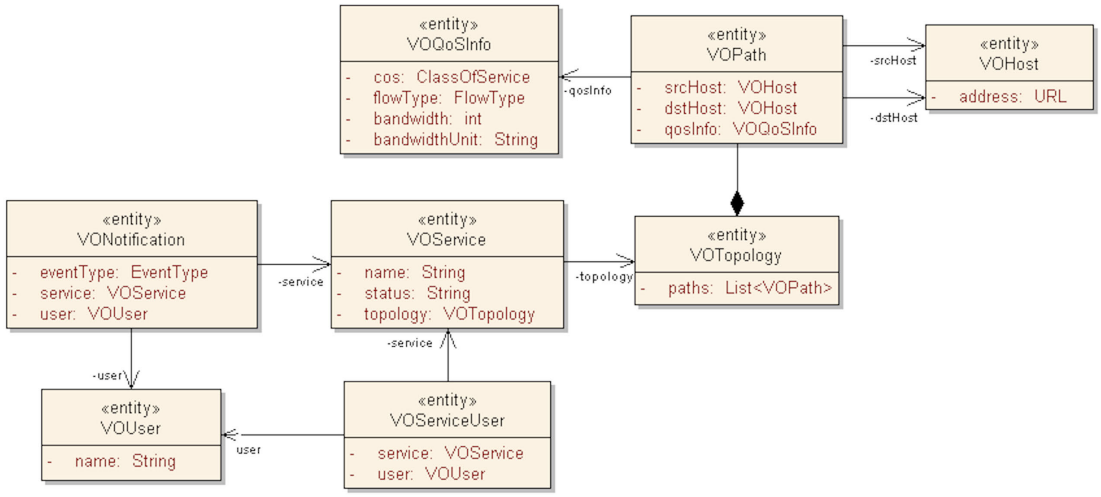

Figura 4.9: Entidades do *peer* de gerenciamento de rede

A classe **`VOUser`** tem a função de armazenar a identificação do usuário da grade relacionado ao evento. Já **`VOService`** representa a aplicação da grade. Possui como atributos o nome do serviço e o seu estado. **`VOServiceUser`** faz a relação entre usuários e serviços.

`VOService` possui relação com a classe **`VOTopology`**, o qual é uma composição de instâncias da classe **`VOPath`**. Esta estrutura visa representar a topologia da aplicação, isto é, os nodos da aplicação e a comunicação entre estes nodos, sendo os nodos representados como instâncias da classe **`VOHost`**. Para cada fluxo está associada ainda uma CoS, representada pela classe de entidade **`VOQoSInfo`**.

4.4.3 Integração ao NotificationService

Conforme descrito na Seção 4.4.1, na inicialização do *peer* são ativados também os *web-services* implementados. Estes serviços são descritos em um arquivo XML chamado "descriptor.xml". Para cada serviço deve ser fornecido o seu nome, descrição, um identificador único e, principalmente, o pacote e classe do *peer* correspondente ao serviço, neste caso "edu.ufrgs.autoMan.component.AutoManService". `AutoManService`, a qual implementa a interface `ServiceLifecycle`, compõe a interface responsável por receber as notificação enviadas pela fonte de eventos (serviço NotificationService). As notificações da grade são recebidas através do método `receiveNotification(String message)`.

A Figura 4.10 exemplifica uma mensagem de notificação recebida.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<notification eventType="ServiceDeploy">
  <user> <name>jonhL</name> </user>
  <service>
    <name>service01</name>
    <status>start</status>
  </service>
</notification>
```

Figura 4.10: Mensagem de notificação

Dentro do modelo de AC, descrito na Seção 2.3, a classe de `AutoManService` representa o sensor do sistema de gerenciamento de rede. Além disso, realiza parte da

função de análise das notificações recebidas, já que efetua a validação destas. Após validada, o conteúdo da notificação é estruturado utilizando as classes de entidade apresentadas na Seção 4.4.2 e a mensagem XML desprezada. Esta estrutura, que tem a classe `VONotification` como raiz, é então repassada à classe `AutoManPeer`.

A partir daí, a classe `AutoManPeer` avalia as tarefas a serem executadas de acordo com o planejamento efetuado pelo *peer*. Este planejamento é descrito na Seção 4.4.4.

4.4.4 Planejamento

O objetivo dos Sistemas Multi-Agentes (SMA), no ponto de vista do desenvolvimento de sistemas computacionais, é a definição de modelos genéricos de agentes, interações e organizações que possam ser instanciados dinamicamente dado um problema (HÜBNER; BORDINI; VIEIRA, 2004). Dado este ideal de metodologia de desenvolvimento de sistema, a abordagem de SMA apresenta as seguintes características, segundo (ALVARES; SICHMAN, 1997):

- Agentes são concebidos independentemente de um problema particular;
- Decomposição de tarefas para solucionar um dado problema pode ser feita pelos próprios agentes;
- Não existe um controle centralizado da resolução do problema.

É com este intuito que buscou-se inspiração no modelo de raciocínio de agentes artificiais cognitivos para elaboração da atividade de planejamento, prevista no modelo de AC. Em particular, buscou-se que os *peers* de gerenciamento planejassem as tarefas a serem executadas: i) a partir do tipo de evento recebido; ii) de forma configurável, sem necessitar de reprogramação dos *peers*; iii) extensível, de forma que novos tipos de eventos sejam suportados sem requerer alteração na estrutura dos *peers*.

O modelo de raciocínio dos *peers* de gerenciamento é composto por uma base de crenças, objetivos e planos. A base de crenças é o que o agente conhece sobre o ambiente gerenciado. A base de crenças do *peer* de gerenciamento é formada pelas informações obtidas a partir do serviço de informações da grade, ainda na fase de análise no modelo AC (Figura 4.13). Os objetivos representam o que o *peer* está tentando atingir. Neste estudo de caso, o objetivo do *peer* de gerenciamento de rede é tentar transpor para a rede o mesmo tratamento (nível de serviço) que o usuário e aplicação possui no domínio da grade computacional. Este objetivo é representado como a palavra-chave "NetworkDeploy". A partir deste objetivo são buscados os planos para promover a reserva de recursos descritos na notificação. Já a notificação informando o cancelamento ou interrupção de uma aplicação da grade está vinculada ao objetivo "NetworkUndeploy", o qual está vinculado a um plano para desalocação dos recursos relacionados. Isto significa que cada plano deve estar vinculado a um objetivo, sendo este vínculo feito a partir da palavra-chave. No protótipo, este mapeamento entre objetivos e planos é feito no arquivo "goalPlanMapping.xml".

Um plano é, por sua vez, um conjunto se ações definidas de forma seqüencial. Esta definição, exemplificada pela Figura 4.11, possui um conjunto de planos e as suas respectivas ações. Para cada plano é definido o objetivo para o qual aquele plano foi elaborado, a sua prioridade e um identificador. No protótipo implementado, a definição dos planos é feita em um arquivo XML chamado "planning.xml".

Para um mesmo objetivo podem existir vários planos, porém cada um com uma prioridade distinta. Assim, para um dado objetivo, o *peer* de gerenciamento busca o conjunto de

planos aplicáveis. O plano com maior prioridade é executado primeiramente. Caso este plano retorne falha em alguma das suas ações, o segundo plano com maior prioridade é então executado, e assim consecutivamente.

As ações do plano referem-se à classes que possam ser acessadas pelo *peer*. Estas classes implementam a interface `ManagementAction`, a qual define o método `execute( Map<String, Object> beliefBase)`. No parâmetro deste método são passados os valores das variáveis de ambiente, assim como as respostas das ações anteriores. Este parâmetro representa a base de crenças do *peer*. É dentro deste método que são implementadas efetivamente as tarefas de gerenciamento, descritas na Seção 3.4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<planning>
  <plan goal="NetworkDeploy" priority="0">
    <action implementedBy="edu.ufrgs.autoMan.action.IdentifyNotificationResources" />
    <action implementedBy="edu.ufrgs.autoMan.action.QueryInformationService" />
    <action implementedBy="edu.ufrgs.autoMan.action.IdentifyLocalResources" />
    <action implementedBy="edu.ufrgs.autoMan.action.IdentifyForeignResources" />
    <action implementedBy="edu.ufrgs.autoMan.action.QueryResourceManager" />
    <action implementedBy="edu.ufrgs.autoMan.action.NotifyForeignPeers" />
    <action implementedBy="edu.ufrgs.autoMan.action.BuildPolicies" />
    <action implementedBy="edu.ufrgs.autoMan.action.ExecutePolicies" />
    <action implementedBy="edu.ufrgs.autoMan.action.UpdateResourceManager" />
  </plan>
  <plan goal="NetworkUndeploy" priority="0">
    <action implementedBy="edu.ufrgs.autoMan.action.IdentifyNotificationResources" />
    <action implementedBy="edu.ufrgs.autoMan.action.QueryInformationService" />
    <action implementedBy="edu.ufrgs.autoMan.action.IdentifyLocalResources" />
    <action implementedBy="edu.ufrgs.autoMan.action.IdentifyForeignResources" />
    <action implementedBy="edu.ufrgs.autoMan.action.NotifyForeignPeers" />
    <action implementedBy="edu.ufrgs.autoMan.action.RemovePolicies" />
    <action implementedBy="edu.ufrgs.autoMan.action.UpdateResourceManager" />
  </plan>

  ... Outros planos ...
</planning>
```

Figura 4.11: Plano

Com esta forma de estruturação das tarefas a serem executadas pelo *peer* de gerenciamento foi possível torná-lo facilmente extensível. Por exemplo, caso o *peer* de gerenciamento passasse a prever outras ações de gerenciamento, agora para o domínio de segurança, seria necessário:

1. Criar um objetivo relativo à segurança;
2. Mapear o tipo de notificação recebida para este objetivo;
3. Adicionar planos à configuração do *peer* de gerenciamento;
4. Implementar as ações de gerenciamento de segurança.

Nota-se que nenhum destes itens requer a alteração da estrutura ou reprogramação do *peer* de gerenciamento. A adição de novas funcionalidades é feita de modo semelhante a adição de um *plugin* da aplicação. No protótipo, porém, foi configurado apenas um plano para cada um dos objetivos, "NetworkDeploy" e "NetworkUndeploy".

4.4.5 Integração ao InformationService

Como parte da função de análise, prevista no modelo de AC, os *peers* de gerenciamento devem buscar informações adicionais junto ao serviço de informação da grade sobre as notificações recebidas. Isto porque estas notificações descrevem apenas o usuário, a

aplicação e o estado da aplicação. Esta tarefa é implementada pela ação **QueryInformationService**, seguindo o plano ilustrado pela Figura 4.11. Nesta ação, o *peer* de gerenciamento envia uma requisição ao InformationService através da mensagem ilustrada pela Figura 4.12.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<informationRequest>
  <user>
    <name>jonhL</name>
  </user>
  <service>
    <name>service01</name>
  </service>
</informationRequest>
```

Figura 4.12: Requisição de informação

A mensagem de resposta do InformationService é apresentada na Figura 4.13. Esta mensagem contém a topologia da aplicação (determinada pela comunicação entre nodos), a CoS requerida e a indicação sobre o tipo de tráfego, protocolo e previsão de banda de cada fluxo utilizado pela aplicação. Estas informações servem de base para o gerente de recursos prever a alocação de banda nos enlaces WAN do domínio. O InformationService envia ainda informações sobre quando a aplicação estará efetivamente em execução (o formato utilizado para representar o período de execução é baseado no Policy Core Information Model (PCIM) (MOORE et al., 2001)).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<informationResponse>
  <user> <name>jonhL</name> </user>
  <service>
    <name>service01</name>
    <topology>
      <path protocol="RTP">
        <srcHost> <address port="5041">200.176.3.142</address> </srcHost>
        <dstHost> <address port="15095">249.10.0.21</address> </dstHost>
        <qosinfo>
          <cos>AF</cos>
          <flowType>videoStream</flowType>
        </qosinfo>
      </path>
      <path protocol="FTP">
        <srcHost> <address port="1030">200.176.3.142</address> </srcHost>
        <dstHost> <address port="21">201.21.228.67</address> </dstHost>
        <qosinfo>
          <cos>EF</cos>
          <flowType>dataIntensive</flowType>
          <bandwidth unit="kbps">1400</bandwidth>
        </qosinfo>
      </path>
      <scheduling>
        <timePeriod>
          <start>20000101T050000</start> <end>20000402T070000</end>
        </timePeriod>
        <dayOfWeekMask>Monday</dayOfWeekMask>
        <timeOfDayMask>
          <start>T130000</start> <end>T170000</end>
        </timeOfDayMask>
        <localOrUtcTime>UTC</localOrUtcTime>
      </scheduling>
  </service>
</informationResponse>
```

Figura 4.13: Resposta de informação

A resposta apresentada na Figura 4.13 representa uma aplicação com dois fluxos de dados. O primeiro é descrito como sendo uma transmissão de vídeo utilizando o protocolo RTP que deve ser tratada pela rede como sendo da classe AF. Já o segundo fluxo, é descrito como sendo uma transferência FTP que deve ser tratada como sendo da classe EF. Os elementos *cos* e *flowType* utilizam palavras-chave que devem ser interpretadas pela camada P2P.

4.4.6 Identificação de recursos

Após obter as informações junto ao serviço de informação da grade, o *peer* de gerenciamento deve identificar os recursos envolvidos. Para isto, uma simples análise nas informações obtidas a partir do InformationService permite relacionar os *hosts* utilizados pelos nodos da aplicação. Esta tarefa é executada pela ação **`IdentifyNotificationResources`**.

Então, devem ser identificados quais destes *hosts* são internos ao domínio do *peer*, ou seja, cujo *peer* de gerenciamento possui acesso direto. A ação **`IdentifyLocalResources`** é a responsável por esta tarefa. Na implementação completa da arquitetura proposta, conforme descrito na Seção 3.4, a ação `IdentifyLocalResources` deveria primeiramente identificar os PDPs do domínio. A seguir, deveria identificar quais destes PDPs são "responsáveis" pelos *hosts* (possuem acesso aos PEPs relacionados aos *hosts*). Assim, seria feito um mapeamento entre os *hosts* e os PDPs responsáveis por estes *hosts*. Porém, como o protótipo implementado não possui integração com a camada de políticas, este mapeamento é efetuado de forma simulada.

Para a identificação dos *peers* responsáveis por recursos localizados fora do domínio do *peer* de gerenciamento é utilizado o sistema de busca entre *peers*. Este sistema é configurado durante a inicialização da classe `AutoManPeer`. Para isto é instanciado um objeto da classe `PeerSearchHandler` e registrado no *container* de serviços de busca do GT4. A partir daí, este serviço passa a estar disponível para a ação **`IdentifyForeignResources`**.

Para efetuar a busca, a ação `IdentifyForeignResources` identifica na base de crenças a lista de *hosts* para os quais não foram localizados PDPs responsáveis. Assim, uma mensagem a ser enviada na consulta é montada, conforme ilustra a Figura 4.14. Esta mensagem é incluída em um objeto da classe `ResolverQuery` e enviada ao serviço de busca.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<peerSearch>
  <resource>
    <type>host</type>
    <address>249.10.0.21</address>
  <resource>

  ... Outros recursos ...
</peerSearch>
```

Figura 4.14: Busca entre *peers*

Conforme são recebidas as respostas da consulta, é montado o mapeamento entre os *hosts* e os *peers* com acesso aos PDPs responsáveis por estes *hosts*. É possível ainda que para certos *hosts* não seja identificado nem o PDP, nem o *peer* de gerenciamento relacionados. Este caso pode ocorrer quando não há PEP associado ao *host*, assim a plataforma de políticas desconhece a existência deste dispositivo. Pode ocorrer ainda quando o domínio ao qual está inserido o *host* não possui *peer* de gerenciamento disponível.

O mapeamento efetuado por `IdentifyLocalResources` será utilizado para montar as mensagens a serem enviadas para a camada de políticas pela ação `BuildPolicies` (Seção 4.4.9). Já o mapeamento criado por `IdentifyForeignResources` será utilizado para requerer aos demais *peers* relacionados que estes procedam a reserva de banda nos seus domínios administrativos. Esta notificação é efetuada pela ação `NotifyForeignPeers` (Seção 4.4.8).

4.4.7 Gerente de recursos

Neste protótipo, o gerente de recursos é um *peer* com a função de manter uma base de dados com o registro da utilização dos enlaces WAN do domínio. A sua estrutura é semelhante à ilustrada pela Figura 4.8, porém não implementando as classes relacionadas a busca entre *peers*. Além disso, na inicialização do *peer* (método `start()`) é iniciado também o banco de dados embarcado. Para isto foi utilizado o banco de dados HSQLDB (HSQLDB, 2007), por poder ser utilizado de forma embarcada, ser leve e facilmente integrável a qualquer aplicação Java.

A classe principal do gerente de recursos foi chamada de **`ResourceManPeer`** e corresponde à classe **`AutoManPeer`** do *peer* de gerenciamento de rede. A comunicação com os demais *peers* é feita através da classe **`ResourceManService`**. Esta classe disponibiliza dois métodos como interfaces *web-services*: `checkAvailability(String query)` e `updateState(String update)`. O primeiro permite que os outros *peers* consultem a disponibilidade de largura de banda de um enlace WAN específico. Baseado nesta consulta, o *peer* decidirá sobre a viabilidade de atender a requisição da grade por reserva de banda em uma CoS específica.

Neste protótipo, a banda prevista para uso da aplicação em cada enlace é dada em termos de banda média e em valores absolutos. Uma implementação mais complexa do gerente de recursos poderia estimar um padrão de volume de tráfego utilizado a partir da informação sobre o tipo de fluxo e sobre o protocolo utilizado. Assim, a estimativa de uso de banda estaria mais próxima da realidade, evitando sub-alocação ou super-alocação dos enlaces WAN. O gerente de recursos implementado não considerada estas informações para avaliação de disponibilidade.

A Figura 4.15 apresenta um exemplo de mensagem enviada pelo *peer* de gerenciamento ao gerente de recursos durante a ação **`QueryResourceManager`**, definida no plano de gerenciamento (Seção 4.4.4).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<checkAvailability>
  <link number="1">
    <src> <address port="5041">200.176.3.142</address> </src>
    <dst> <address port="10084">200.10.0.21</address> </dst>
    <protocol>FTP</protocol>
    <qosinfo>
      <cos>AF</cos>
      <flowType>dataIntensive</flowType>
      <bandwidth unit="kbps">1400</bandwidth>
    </qosinfo>
  </link>
```

... Outros enlaces WAN ...

```
</checkAvailability>
```

Figura 4.15: Mensagem para verificar disponibilidade de banda

Já a Figura 4.16 representa a resposta do gerente de recursos. No estudo de caso apresentado na Seção 3.5, o gerente de recursos consultaria periodicamente algum PDP do

domínio a fim de obter a lista de enlaces WAN do domínio. No protótipo implementado, porém, o gerente de recursos foi configurado para armazenar estaticamente esta lista. A partir dela, o gerente de recursos define o enlace WAN relacionado ao fluxo da aplicação da grade e decide a disponibilidade de recursos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<checkAvailability>
  <link number="1" available="true"/>
  <link number="2" available="false"/>
```

... Outros enlaces WAN ...

```
</checkAvailability>
```

Figura 4.16: Resposta do gerente de recursos

Ao receber a resposta do gerente de recursos, o *peer* de gerenciamento pode dar seguimento as tarefas previstas. Após definir que a requisição de banda em uma determinada CoS será efetivada, o *peer* de gerenciamento envia uma mensagem ao gerente de recursos através do método `updateState(String update)`, chamado pela ação **`UpdateResourceManager`** do plano de gerenciamento. Esta mensagem contém as mesmas informações descritas na Figura 4.15, porém agora o gerente de recursos utiliza estas informações para atualizar sua base de dados.

Quando os *peers* de gerenciamento recebem uma notificação informando o cancelamento ou interrupção da execução de uma aplicação da grade estes executam o plano com o objetivo "NetworkUndeploy", o qual deve desalocar os recursos relacionados. Neste plano, a ação `UpdateResourceManager` envia mensagem ao gerente de recursos para que este remova o registro de reserva de banda efetuado. Desta forma, o gerente de recursos cumpre seu papel no protótipo de controlar a alocação de banda nos enlaces WAN.

4.4.8 Notifica demais *peers* relacionados

Após a decisão do *peer* de gerenciamento de realmente efetuar a reserva de banda para a aplicação da grade, este *peer* deve enviar uma mensagem aos demais *peers* identificados pela ação `IdentifyForeignResources` (Seção 4.4.6). Com esta mensagem, o *peer* de gerenciamento que recebeu a notificação solicita aos demais *peers* que estes também procedam a reserva de banda no seu domínio respectivo. Esta tarefa é implementada pela ação **`NotifyForeignPeers`**.

É neste ponto que o *peer* implementa o comportamento transacional descrito na Seção 3.5.3. Após receber as respostas de sucesso ou insucesso dos demais *peers*, o *peer* avalia se deve prosseguir a implantação da reserva. Por padrão, o protótipo implementado cancela a reserva de banda caso algum dos *peers* responda negativamente à reserva de banda. Para os *peers* que enviaram mensagem confirmando a reserva de banda, é enviada uma nova mensagem, agora a cancelando.

De forma semelhante, quando o objetivo do *peer* de gerenciamento é o de desalocar os recursos relacionados à notificação recebida, a ação `NotifyForeignPeers` envia mensagem aos demais *peers* para que estes desfaçam a reserva efetuada.

4.4.9 Mensagem a ser enviada para a plataforma de políticas

As últimas ações a serem executadas no plano, estabelecido com o objetivo de transpor para a rede o mesmo nível de serviço que o usuário e a aplicação possui no domínio da grade computacional, são as referentes às políticas a serem implantadas na rede. A ação `BuildPolicies` é a responsável por montar as mensagens a serem enviadas à camada

de políticas. Estas mensagens são montadas de acordo com as informações obtidas nas ações anteriores. A Figura 4.17 mostra um exemplo de mensagem enviada pelo *peer* à camada de políticas. Esta mensagem refere-se ao primeiro fluxo indicado na Figura 4.13. O seu formato é baseado no padrão PCIMe, definido pelo DMTF.

Quando o objetivo do *peer* de gerenciamento é de remover as políticas anteriormente aplicadas, a mensagem enviada à camada de políticas é semelhante. Porém, a mensagem indica que as políticas descritas devem ser removidas, ao invés de implantadas.

Por fim, a ação `ExecutePolicies` deve localizar os PDPs identificados pela ação `IdentifyLocalResources` (Seção 4.4.6). Como para cada *host* é possível que exista mais de PDP relacionado, o *peer* escolhe aleatoriamente algum destes PDPs e envia a mensagem descrita na Figura 4.17. O PDP deve então proceder a implantação da política no repositório de políticas e nos PEPs relacionados. No protótipo implementado este processo de interação com a camada de políticas é apenas simulado.

4.5 Considerações Finais

Após a implementação deste protótipo pode-se verificar que o GT4 possui formas de agregar novas fontes de informação a sua estrutura. Isto permitiu que os serviços de notificação e de informação fossem integrados ao WS-MDS sem que fosse necessário alterar qualquer classe implementada pelo GT4.

Por outro lado, ficou claro que a versão atual do *toolkit* não fornece a estas fontes de informação, dados sobre a topologia da aplicação, nem possui qualquer estimativa sobre o volume de dados que irão trafegar entre os nodos da aplicação. Esta característica se deve ao fato do GT4 possuir fraca integração com o escalonador de trabalhos. Assim, a função do GT4 resume-se a submeter ao escalonador, presente em algum *host* ou *cluster* da grade, a aplicação a ser executada, em conjunto com a sua descrição. A partir daí, o escalonador distribui a aplicação entre os nodos da grade, de forma independente do GT4.

Uma solução que permitiria o fornecimento das informações requeridas para a arquitetura de gerenciamento proposta, seria fazer com que o escalonador disponibilizasse uma fonte de informações contendo os dados sobre as aplicações escalonadas. Assim, na medida em que o escalonador distribuísse a aplicação em um conjunto de nodos, este deveria disponibilizar dados sobre este escalonamento em uma fonte de informações. Além disso, a linguagem de descrição de trabalhos utilizada pelo GT4, chamada Job Submission Description Language (JSDL) (OPENGRIDFORUM, 2005), permite ao usuário informar as tarefas a serem executadas, a dependência entre estas tarefas e a largura de banda necessária para cada tarefa individualmente. Porém, seria necessário que a linguagem permitisse ao usuário informar a largura de banda necessária para a comunicação entre estas tarefas (nodos da aplicação).

Em relação a interação entre o GT4 e os *peers* de gerenciamento pode-se avaliar também que o sistema de notificação implementado, utilizando o suporte do GT4 ao modelo Publicar/Assinar, supriu as necessidades em termos de funcionalidade. Para o domínio de grades computacionais, nos quais os trabalhos em geral são agendados com antecedência, o tempo de resposta do sistema de gerenciamento de rede proposto não é restritivo. Porém, seria relevante analisar o tempo decorrido entre a submissão dos trabalhos e o envio da notificação aos *peers* de gerenciamento e o tempo de resposta do serviço de informações.

Com a implementação do protótipo foi possível verificar que o suporte da plataforma JXTA, em conjunto com a biblioteca JXTA-SOAP, forneceu as funcionalidades neces-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<policySet enabled="true">
  <policy commonName="NetworkDeploy">
    <policyRule conditionListType="DNF">
      <compoundPolicyCondition>
        <simplePolicyCondition>
          <variable>protocol</variable>
          <value>rtp</value>
        </simplePolicyCondition>
        <simplePolicyCondition>
          <variable>sourceAddress</variable>
          <value>200.176.3.142</value>
        </simplePolicyCondition>
        <simplePolicyCondition>
          <variable>sourcePort</variable>
          <value>5041</value>
        </simplePolicyCondition>
        <simplePolicyCondition>
          <variable>destAddress</variable>
          <value>200.176.3.142</value>
        </simplePolicyCondition>
        <simplePolicyCondition>
          <variable>destPort</variable>
          <value>15095</value>
        </simplePolicyCondition>
        <policyTimePeriodCondition>
          <timePeriod>
            <start>20000101T050000</start>
            <end>20000402T070000</end>
          </timePeriod>
          <dayOfWeekMask>Monday</dayOfWeekMask>
          <timeOfDayMask>
            <start>T130000</start>
            <end>T170000</end>
          </timeOfDayMask>
          <localOrUtcTime>UTC</localOrUtcTime>
        </policyTimePeriodCondition>
      </compoundPolicyCondition>
      <policyAction policyActionName="gridServiceQoS">
        <compoundPolicyAction>
          <simplePolicyAction>
            <variable>DSCP</variable>
            <value>AF</value>
          </simplePolicyAction>
          <simplePolicyAction>
            <variable>flowType</variable>
            <value>videoStream</value>
          </simplePolicyAction>
        </compoundPolicyAction>
      </policyAction>
    </policyRule>
  </policy>
```

... Outras políticas ...

```
</policySet>
```

Figura 4.17: Mensagem do *peer* para o PDP

sárias para a implementação da camada P2P. Em especial, o mecanismo de busca entre *peers* pode ser implementado fazendo uso do protocolo de busca do JXTA. Da mesma forma, seria relevante avaliar os tempos gastos com a comunicação entre *peers*, além da avaliação do tempo total decorrido entre a recepção da notificação e efetivo envio das políticas a serem aplicadas na plataforma de políticas.

5 CONCLUSÃO

O trabalho descrito nesta dissertação pode ser visto como um projeto de arquitetura, cujo foco está no desenvolvimento de um sistema de gerenciamento de redes complementar ao gerenciamento de redes baseado em políticas tradicional. O requisito principal desta arquitetura é o de prover maior dinamismo ao sistema de gerenciamento de redes, permitindo que este interaja com outros sistemas de gerenciamento/monitoramento. Como parte do projeto da arquitetura, foi realizada a prototipação desta com o intuito de avaliar a viabilidade da sua implementação tendo como fonte de eventos um *toolkit* de computação em grade.

Para atender a este requisito, o projeto da arquitetura buscou inspiração no modelo de AC, o qual prevê que o ambiente de TI possua uma infra-estrutura de gerenciamento que permita a adaptação dinâmica dos recursos gerenciados em resposta às mudanças no ambiente. Isto significa que em um ambiente corporativo que empregue o modelo de gerenciamento autonômico, os sistemas de gerenciamento de domínios distintos devem interagir de forma que o ambiente gerenciado permaneça em um estado consistente. Assim, foi projetada uma arquitetura em camadas, na qual a estrutura da camada P2P inspira-se no modelo de AC.

Com o projeto da arquitetura e a avaliação do estudo de caso foi possível identificar que o uso de uma rede *overlay*, neste caso uma rede P2P, sobre a plataforma de políticas permite o desenvolvimento de funcionalidades que não poderiam ser implantadas caso não houvesse interação entre as políticas de cada domínio. Por exemplo, a implementação de um comportamento transacional que dependa das políticas vigentes em mais de um domínio pode ser implantado somente se houver interação entre as plataformas de políticas de cada domínio ou se houver uma camada sobre a plataforma de políticas que realize esta coordenação. Genericamente, todo procedimento que dependa de políticas de mais de um domínio comporta-se de forma semelhante. A arquitetura proposta, através da camada P2P, fornece suporte para estes casos.

Outra característica importante da arquitetura projetada, comprovada pela implementação do protótipo, é o fraco acoplamento entre os componentes da arquitetura provido pelo uso de padrões abertos. O uso de interfaces *web-services* para a comunicação entre a fonte de eventos e a camada P2P permitiu, por exemplo, que os *peers* de gerenciamento de rede, implementados como uma rede JXTA, pudessem registrar-se como assinantes no serviço de notificação da grade, além de possibilitar a consulta às suas fontes de informação. Isto indica que o projeto de comunicação entre a camada P2P e a plataforma de políticas apresentado, utilizando a mesma tecnologia de *web-services*, é uma boa opção.

Além do uso de padrões abertos, outro ponto relevante é a escolha dos padrões adotados para as mensagens XML trocadas. Na medida em que a arquitetura proposta pretende fornecer suporte a interoperabilidade entre sistemas de gerenciamento de domínios dife-

rentes e de forma modular, a padronização das mensagens trocadas entre os sistemas e entre as camadas da arquitetura é fundamental. A arquitetura proposta indica o uso do padrão PCIMe, parte do padrão CIM, para as mensagens XML trocadas entre a camada P2P e a plataforma de políticas. Porém, a comunicação entre a camada de fonte de eventos e a camada P2P, o que constitui a comunicação entre sistemas de gerenciamento, carece de um padrão possível de ser utilizado. O próprio CIM possui um grupo de especificações voltadas ao tratamento de eventos. Porém, estas especificações são bastante complexas e específicas, já que possuem um grande número de atributos obrigatórios relacionados a contextos particulares. Desta forma, o protótipo implementado utilizou um formato próprio para a comunicação entre estas camadas.

O estudo de caso foi focado no provimento de QoS para computação em grade utilizando a arquitetura proposta. O estudo de caso e a sua prototipação permitiu avaliar a integração de um *toolkit* de grade com a camada P2P da arquitetura. Assim, foi possível concluir que para a efetiva implantação da arquitetura proposta seria necessário que o *toolkit* da grade fornecesse informações sobre a topologia da aplicação e sobre o escalonamento dos trabalhos, atualmente não fornecidas pelo GT4. Por outro lado, verificou-se que a estrutura do GT4 permite facilmente agregar novas fontes de informação ao seu *container* de *web-services*. Desta forma, parece ser possível a extensão do GT4 para prover as informações requeridas sem um alto custo de desenvolvimento e sem alterar a sua estrutura interna.

Outra avaliação obtida a partir da elaboração do protótipo foi em relação a comunicação entre os *peers* da camada P2P. A plataforma utilizada para o desenvolvimento dos *peers*, utilizando JXTA em conjunto com o JXTA-SOAP, mostrou-se satisfatória na tarefa de prover: i) a comunicação entre *peers* de gerenciamento, ii) a localização e comunicação dos *peers* com o gerente de recursos e iii) a execução de buscas entre os *peers*. Apesar de ser necessária uma análise apurada sobre o desempenho da rede P2P, pode-se perceber que em todos os testes efetuados a rede organizou-se automaticamente.

5.1 Contribuições

As contribuições deste trabalho são:

- Apresentar um esforço inicial na tarefa de prover a interação entre sistemas computacionais e o sistema de gerenciamento de rede;
- Projeto de uma arquitetura aplicada ao gerenciamento de rede, mas que poderia facilmente ser adotada para o gerenciamento de outros aspectos de TI, como segurança por exemplo;
- Análise da aplicação da arquitetura à configuração de QoS em uma rede com suporte à DiffServ em resposta a eventos recebidos de um *toolkit* para computação em grade;
- Elaboração de um protótipo integrado ao GT4.

5.2 Trabalhos Futuros

Os seguintes trabalhos futuros e pontos de extensão da arquitetura proposta foram identificados:

- **Expansão das ações de gerenciamento**
  A arquitetura projetada está focada em prover a auto-configuração dos dispositivos de rede em resposta a eventos externos. Deve ser avaliada a utilidade dos demais aspectos de AC - auto-proteção, auto-recuperação e auto-otimização - ao gerenciamento de rede e a aplicabilidade da arquitetura a estes aspectos.

- **Reavaliação periódica das ações executadas**
  Fazer com que os *peers* de gerenciamento de rede reavaliem periodicamente as ações realizadas para cada tipo de notificação recebida. Esta reavaliação poderia basear-se em dados históricos de certos parâmetros da rede, coletados a partir de monitores inseridos na rede. Esta característica faria com que os *peers* de gerenciamento pudessem ser considerados verdadeiramente como agentes inteligentes, na medida em que passariam a adaptar o seu comportamento às condições do ambiente.

- **Avaliação de desempenho do protótipo**
  Analisar o tempo decorrido entre a submissão dos trabalhos e o envio da notificação aos *peers* de gerenciamento e o tempo de resposta do serviço de informações. Analisar ainda os tempos gastos com a comunicação entre *peers* e o tempo total decorrido entre a recepção da notificação e o efetivo envio das políticas a serem aplicadas na plataforma de políticas.

- **Padrão XML para notificação de eventos**
  Extensão da especificação do padrão CIM relativa a eventos para atender a comunicação entre a fonte de eventos e a camada P2P;

- **Complemento do protótipo**
  Integração do protótipo a uma plataforma de políticas, como o QAME (GRANVILLE; TAROUCO, 2001);

- **Segurança**
  Avaliar técnicas de autenticação a serem aplicadas para restringir o acesso aos *peers* de gerenciamento à apenas fontes de eventos autorizadas. Da mesma forma, deve-se restringir o acesso ao grupo de *peers* de gerenciamento aos *peers* autenticados.

REFERÊNCIAS

ALLMAN, M. An evaluation of XML-RPC. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.30, n.4, p.2–11, 2003.

ALVARES, L. O.; SICHMAN, J. S. Introdução aos sistemas multiagentes. **Jornada de Atualização em Informática**, Porto Alegre, v.1, p.1–38, 1997.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A survey of peer-to-peer content distribution technologies. **ACM Comput. Surv.**, NewYork, NY, USA, v.36, n.4, p.335– 371, 2004.

APACHE PROJECT. **Apache Ant**. Disponível em <http://ant.apache.org>. Acesso em: 03 mar. 2008.

AWDUCHE,D. et al. **Requirements for Traffic Engineering Over MPLS**: RFC 2702. [S.l.]: Internet Engineering Task Force, Network Working Group, 1999.

BINZENHOFER, A. et al. A P2P-Based Framework for Distributed Network Management. In: INTERNACIONAL WORKSHOP OF THE EURO-NGI NETWORK OF EXCELENCE, 2., 2005, Villa Vigoni, Italy. **Wireless Systems and Network Architecture in Next Generation Internet**: revised selected papers. Berlin: Springer, 2006. p.198–210. (Lecture Notes in Computer Science v.3883).

BLAKE, S. et al. **An Architecture for Differentiated Service**: RFC 2475. [S.l.]: Internet Engineering Task Force, Network Working Group, 1998.

BOYLE, J. el al. **The COPS (Common Open Policy Service) Protocol**: RFC 2748. [S.l.]: Internet Engineering Task Force, Network Working Group, 2000.

BROOKSHIER, D. et al. **JXTA**: Java P2P Programming. [S.l.]: Sams Publishing, 2002. 432p.

CECCON,M.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. Definition and Visualization of Dynamic Domains in Network Management Environments. In: **Information Networking**: Networking Technologies for Enhanced Internet Services. Berlin: Springer, 2003. p. 828-838.

CHAN, K. et al. **COPS Usage for Policy Provisioning (COPS-PR)**: RFC 3084. [S.l.]: Internet Engineering Task Force, Network Working Group, 2001.

CHARALAMBIDES, M. et al. Policy Conflict Analysis for Quality of Service Management. In: IEEE INTERNATIONALWORKSHOPON POLICIES FOR DISTRI-

BUTED SYSTEMS AND NETWORKS, POLICY, 6., 2005. **Proceedings...** Washington, D.C.: IEEE Computer Society, 2005. p.99–108.

CONDOR. **Condor Project**. Disponível em <http://www.cs.wisc.edu/condor>. Acesso em: 03 mar. 2008.

DAMIANOU, N. **A Policy Framework for Management of Distributed Systems**. 2002. Tese (Doutorado em Ciência da Computação) – Imperial College of Science, Technology and Medicine, London.

DISTRIBUTED SYSTEMS GROUP. **JXTA-SOAP Project**. Disponível em <http://soap.jxta.org>. Acesso em: 03 mar. 2008.

DMTF: Distributed Management Task Force. Disponível em <http://www.dmtf.org>. Acesso em: 03 mar. 2008.

EGEVANG, K.; FRANCIS, P. **The IP Network Address Translator (NAT)**: RFC 1631. [S.l.]: Internet Engineering Task Force, Network Working Group, 1994.

EUGSTER, P.T. et al. The many faces of publish/subscribe. **ACM Comput. Surv.**, New York, NY, USA, v.35, n.2, p.114– 131, 2003.

FORD, B.; KEGEL, D.; SRISURESH, P. Peer-to-Peer Communication Across Network Address Translators. In: USENIX TECHNICAL CONFERENCE, 2005. **Proceedings...** [S.l.: s.n.], 2005.

FOSTER, I. What is the grid? A three point checklist. In: **GridToday**. [S.l.]: Tabor Communications Inc., 2002.

FOSTER, I.; KESSELMAN, C. **The Grid 2**: blueprint for a new computing infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

FRANCO, T.; LIMA, W.; SILVESTRIN, G.; PEREIRA, R.; ALMEIDA, M. J.; TAROUCO, L.; GRANVILLE, L.; BELLER, A.; JAMHOUR, E.; FONSECA, M. Substituting COPS-PR: an evaluation of NETCONF and SOAP for policy provisioning. In: IEEE INTERNATIONAL WORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 7., 2006. **Proceedings...** [S.l.: s.n.], 2006.

GLOBUS PROJECT. **Globus Toolkit Quick Start**. Disponível em: <http://www.globus.org/toolkit/docs/4.0/>. Acesso em: 03 mar. 2008.

GONG, L. JXTA: a network programming environment. In: INTERNET COMPUTING, IEEE, 2001. **Anais...** [S.l.: s.n.], 2001. v.5, p.1089–7801.

GRANVILLE, L. Z.; ROSA, D. M. da; PANISSON, A.; MELCHIORS, C.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. Managing Computer Networks Using Peer-to-Peer Technologies. **IEEE COMMUNICATIONS MAGAZINE**, New York, v.43, n.10, p.62-68, Oct. 2005.

GRANVILLE, L. Z.; TAROUCO, L M. R. QAME : QoS-aware management environment. In: ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 25., 2001. **Proceedings...** Los Alamitos: IEEE, 2001. p.269-274.

HÜBNER, J.; BORDINI, R.; VIEIRA, R. Introdução ao Desenvolvimento de Sistemas Multiagentes com Jason. In: ESCOLA DE INFORMÁTICA DA SBC – REGIONAL PARANÁ, ERI, 12., 2004, Guarapuava, PR. **Inteligência Artificial**: anais. Guarapuava: UNICENTRO, 2004. p.51-58.

HEAD, M. R. ET al. A Benchmark Suite for SOAP-based Communication in Grid Web Services. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2005. **Proceedings…** Washington, DC: IEEE Computer Society, 2005. p.19.

HP. **A Primer on Policy-based Network Management**. Fort Collins, USA: Hewlett-Packard Company, 1999.

HSQLDB. **HSQL DataBase Engine**. Disponível em <http://www.hsqldb.org>. Acesso em: 03 mar. 2008.

HWANG, J. et al. An implementation study of a dynamic inter-domain bandwidth management platform in DiffServ networks. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, 2004. **Proceedings...** [S.l.: s.n.], 2004.

I.B.M. **Autonomic computing**: creating self-managing computing systems. Disponível em <http://www.ibm.com/autonomic/>. Acesso em: 03 mar. 2008.

IETF. **Policy Framework Workgroup**. Disponível em: <http://www.ietf.org/html.charters/OLD/policycharter.html>. Acesso em: 03 mar. 2008.

KEPHART, J. O. Research challenges of autonomic computing. In: SOFTWARE ENGINEERING, 27., 2005. **Proceedings...** [S.l.: s.n.], 2005.

KEPHART, J. O.; CHESS, D. M. The Vision of Autonomic Computing. **Computer**, Los Alamitos, CA, USA, v.36, n.1, p.41–50, 2003.

KONSTANTINOU, A. V. **Towards autonomic networks**. 2003. Tese (Doutorado em Ciência da Computação) – Columbia University. Adviser-YechiamYemini.

LUPU, E. C.; SLOMAN, M. Conflicts in Policy-Based Distributed Systems Management. **IEEE Trans. Softw. Eng.**, Piscataway, NJ, USA, v.25, n.6, p.852–869, 1999.

MAGANA,E.; SERRAT, J. QoS Aware Policy-Based Management Architecture for Service Grids. In: WORKSHOP ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, WETICE, 14., 2005, Linköping, Sweden. **Proceedings…** Los Alamitos, CA: IEEE Computer Society, 2005. p.290–294.

MARQUEZAN, C.; MACHADO, I.; RODRIGUES, L.; GRANVILLE, L. Z.; VIANNA, R.; ALVES, R. Gerenciamento Baseado em Políticas para Redes de Dimensões Nacionais no Ambiente QAME. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 23., 2005, Fortaleza. **Anais...** Fortaleza: SBC, 2005. p.1265-1272.

MAYMOUNKOV, P.; MAZIERES, D. Kademlia: a peer-to-peer information system based on the xor metric. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, IPTPS, 1., 2002. **Proceedings...** Berlin: Springer-Verlag, 2002. p.53–65.

MOORE, B. **Policy Core Information Model (PCIM) Extensions**: RFC 3460. [S.l.]: Internet Engineering Task Force, Network Working Group, 2003.

MOORE, B. et al. **Policy Core Information Model (PCIM)**: RFC 3060. [S.l.]: Internet Engineering Task Force, Network Working Group, 2001.

NEISSE, R.; ALMEIDA, M. J.; GRANVILLE, L. Z.; TAROUCO, L. M. R. Policies translation for integrated management of grids and networks. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2005. **Proceedings...** [S.l.: s.n.], 2005.

NICHOLS, K.; JACOBSON, V.; ZHANG, L. **A Two-bit Differentiated Services Architecture for the Internet**: RFC 2638. [S.l.]: Internet Engineering Task Force, Network Working Group, 1999.

OASIS. **Web Services Notification (WSN) TC**. Disponível em: <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn>. Acesso em: 03 mar. 2008.

OASIS. **WS-Security Core Specification 1.1**. Disponível em: <http://www.oasis-open.org/specs/>. Acesso em: 03 mar. 2008.

OPENGRIDFORUM. **Job Submission Description Language (JSDL) Specification, Version 1.0**. Disponível em: <https://forge.gridforum.org/sf/projects/jsdl-wg>. Acesso em: 03 mar. 2008.

OURGRID COMMUNITY. **OurGrid**. Disponível em: <http://www.ourgrid.org>. Acesso em: 03 mar. 2008.

PONNAPPAN, A. et al. Apolicy based QoS management system for the IntServ/DiffServ based Internet. In: INTERNATIONAL WORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 3., 2002. **Proceedings...** [S.l.] IEEE, 2002. v.249, p.159–168.

PRIMET, P.; CHANUSSOT, F. End to end network quality of service in grid environments: the QOSINUS approach. In: INTERNATIONAL WORKSHOP ON NETWORKS FOR GRID APPLICATIONS, 1., 2004. **Proceedings...** [S.l.: s.n.], 2004.

GLOBUS. **Globus Toolkit**. Disponível em: <http://www.globus.org/toolkit/>. Acesso em: 03 mar. 2008.

RAHAMAN, M. A.; SCHAAD, A.; RITS, M. Towards secure SOAP message exchange in a SOA. In: ACM WORKSHOP ON SECURE WEB SERVICES, SWS, 3., 2006. **Proceedings…** New York: ACM Press, 2006. p.77– 84.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence**: a modern approach. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.

SANDER, V. et al. End-to-end provision of policy information for network QoS. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 10., 2001. **Proceedings...** [S.l.: s.n.], 2001.

STATE, R.; FESTOR, O. A management platform over a peer to peer service infrastructure. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, v.1, 2003. **Proceedings...** [S.l.], 2003. p.124–131.

STOICA, I. et al. Chord: a scalable peer-to-peer lookup service for internet applications. In: CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS, SIGCOMM, 2001. **Proceegins…** New York: ACM Press, 2001. p.149–160.

SUN MICROSYSTEMS. **JXTA Technology**. Disponível em: <http://www.sun.com/software/jxta>. Acesso em: 03 mar. 2008.

SUN MICROSYSTEMS. **Java Management Extensions (JMX) Technology**. Disponível em: <http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/>. Acesso em: 03 mar. 2008.

TEITELBAUM, B. et al. **Internet2 QBone**: building a testbed for differentiated services. **IEEE NETWORK MAGAZINE**, [S.l.], v.13, n.5, p.8-16, Sep./Oct. 1999.

UNIVERSITY OF CALIFORNIA. **Ganglia Monitoring System**. Disponível em: <http://ganglia.sourceforge.net>. Acesso em: 03 mar. 2008.

UNIVERSITY OF CALIFORNIA. **Chimera**. Disponível em: <http://current.cs.ucsb.edu/projects/chimera/>. Acesso em: 03 mar. 2008.

VATSAVAI, R.; CHAKRAVARTHY, S.; MOHANIA, M. Access Control Inference And Feedback For Policy Managers. In: IEEE INTERNATIONALWORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 7., 2006. **Proceedings...** [S.l.: s.n.], 2006.

WESTERINEN, A. et al. **Terminology for policy-based management**: RFC 3198. [S.l.]: Internet Engineering Task Force, Network Working Group, 2001.

WINER, D. **XML-RPC Specification**. Disponível em: <http://www.xmlrpc.com/spec>. Acesso em: 03 mar. 2008.

YANG, K.; TODD, C. Policy-based active grid management architecture. In: IEEE INTERNATIONAL CONFERENCE ON NETWORKS, 10., 2002. **Proceedings...** [S.l.: s.n.], 2002.

YEAGER, W.; WILLIAMS, J. Secure Peer-to-Peer Networking: the JXTA example. **IT Professional**, Los Alamitos, CA, USA, v.04, n.2, p.53–57, 2002.

ZHENG, H.; GREIS, M. Ongoing research on QoS policy control schemes in mobile networks. **Mob. Netw.Appl.**, Hingham, MA, USA, v.9, n.3, p.235–242, 2004.