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

EVANDRO DELLA VECCHIA PEREIRA PEREIRA

Uma Arquitetura de Correlação de Notificações Baseada em Políticas e Web Services

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

Prof. Dr. Lisandro Zambenedetti Granville
Orientador

Porto Alegre, junho de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Pereira, Evandro Della Vecchia Pereira

Uma Arquitetura de Correlação de Notificações Baseada em Políticas e Web Services / Evandro Della Vecchia Pereira Pereira. – Porto Alegre: PPGC da UFRGS, 2005.

61 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Lisandro Zambenedetti Granville.

1. Web Services. 2. Correlação. 3. Evento. 4. Notificação. 5. Gerenciamento de redes baseado em políticas. 6. Traps. 7. SNMP. I. Granville, Lisandro Zambenedetti. II. Título.

“Nunca ande pelo caminho traçado, pois ele conduz somente até onde os outros foram.”
— ALEXANDRE GRAHAM BELL

AGRADECIMENTOS

Primeiramente gostaria de agradecer aos meus pais, que me incentivaram sempre a continuar estudando, mesmo que para isso eu deixasse de visitá-los com tanta freqüência, de jogar uma partida de tênis com meu pai ou bater um papo com minha mãe. Uma boa lição aprendi com eles: "A melhor herança que se deixa é o estudo", e isso eles fizeram muito bem. Obrigado pai e mãe!!!

Agradeço à minha irmã, que sempre dizia uma frase de apoio, como "Vai lá! Acaba com o teu trabalho de uma vez e fica livre! Vamos pra festa!", sempre com alto astral. Agradeço à Ji pela paciência, por ter me aturado em muitas ocasiões em que eu tinha prazos a cumprir e não tinha mais tempo para nada. Valeu mesmo!!!

Ao pessoal do Labcom e da Sala 228, com quem eu podia bater um papo nos intervalos, valeu! Mesmo nos momentos de estresse, sempre tinha uma brincadeira para descontrair. Um desses momentos foi o SBRC 2004, onde mesmo não tendo muito tempo para dormir, não dispensávamos uma festa. Agradecimento especial aos que me acompanharam por mais tempo e tiveram que me aturar: Neisse ("o dos meu"), Vaguetti (samurai) e Michelle (Mity).

Tenho um agradecimento especial para fazer ao meu orientador, o Prof. Lisandro. Muitas vezes achei que ele fosse detalhista demais, mas depois vi que ele realmente estava me orientando e não apenas "dando uma olhada" no trabalho. Aprendi muito com ele e creio que com a orientação dele o caminho se tornou mais curto e com poucas pedras. Por isso tudo, meu muito obrigado!!!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
BEEP	*Blocks Extensible Exchange Protocol*
CBR	*Case-based Reasoning*
CLI	*Command Line Interface*
COPS	*Common Open Policy Service*
COPS-PR	*COPS for Policy Provisioning*
CORBA	*Common Object Request Broker Architecture*
CRC	*Cyclic Redundancy Check*
DCOM	*Distributed Component Object Model*
EDI	*Electronic Data Interchange*
FTP	*File Transfer Protocol*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HTTP Secure*
IOS	*Internetworking Operating System*
IETF	*The Internet Engineering Task Force*
IP	*Internet Protocol*
LAN	*Local Area Network*
MAN	*Metropolitan Area Network*
MIB	*Management Information Base*
MBR	*Model-based Reasoning*
OASIS	*Organization for the Advancement of Structured Information Standards*
PCIM	*Policy Core Information Model*
PCIMe	*PCIM extension*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
PHP	*Hypertext Preprocessor*

RBR	*Rule-based Reasoning*
RSVP	*Resource Reservation Protocol*
SaaS	*Software as a Service*
SAML	*Security Assertion Markup Language*
SMTP	*Simple Mail Transfer Protocol*
SNMP	*Simple Network Management Protocol*
SOAP	*Simple Object Access Protocol*
SSL	*Secure Sockets Layer*
STG	*State Transition Graphs*
UDDI	*Universal Description, Discovery and Integration*
WAN	*Wide Area Network*
WS	*Web Services*
WSDL	*Web Services Description Language*
XML	*Extensible Markup Language*
XML-RPC	*XML - Remote Procedure Calling*
W3C	*World Wide Web Consortium*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Notificações enviadas por dispositivos são essenciais para ajudar no gerenciamento de redes de computadores, pois podem alertar sobre, por exemplo, situações anormais ou indesejadas. Porém, em muitos casos, múltiplas notificações relacionadas ao mesmo problema podem ser recebidas por uma estação gerente. Isso faz com que a visualização das notificações se torne confusa. Uma forma possível de diminuir a quantidade de notificações recebidas é através da sua correlação.

Atualmente, os Web Services têm sido um importante tema de pesquisa na área de gerenciamento de redes de computadores. Contudo, não há pesquisas propriamente relacionadas à notificação de eventos usando Web Services. O protocolo SNMP, que é a solução mais aceita e utilizada, possui suporte à notificação de eventos através de suas mensagens *trap*. Porém, esse suporte é limitado e, raramente, consegue cruzar domínios administrativos diferentes.

Unindo a necessidade de correlação com a necessidade de cruzar domínios administrativos diferentes, uma arquitetura de correlação de notificações baseada em Web Services e políticas é apresentada. As políticas são utilizadas no trabalho, como mecanismo para definição das regras de correlação de notificações.

A arquitetura proposta e sua implementação são apresentadas, permitindo a investigação do uso de Web Services como ferramenta no gerenciamento de redes, considerando o caso específico de suporte a notificações. Este estudo complementa as investigações em andamento do Grupo de Pesquisa de Redes da Universidade Federal do Rio Grande do Sul, mostrando aspectos dos Web Services no gerenciamento de redes, que eram desconhecidos no campo de notificações, além de mostrar o gerenciamento baseado em políticas aplicado a notificações, assunto também inexplorado até o momento.

Palavras-chave: Web Services, correlação, evento, notificação, gerenciamento de redes baseado em políticas, traps, SNMP.

A Notification Correlation Architecture Based on Policies and Web Services

ABSTRACT

Notifications sent by devices are essential to help in the management of computer network, because they can alert some abnormal situation. However, in many cases, notifications related to the same problem can be received by a management station. This makes the notification visualization confuse. A possible way to decrease the amount of received notifications is through notification correlation.

Currently, Web Services for network management have been an intensive field of investigation. However, investigations carried up do not properly address the issue related to event notifications. The Simple Network Management Protocol (SNMP), which is the widely deployed and accepted solution, has an interesting support for event notification through its trap messages. However, this support is limited and rarely cross different administrative domains.

With the necessity of notifications correlation and the necessity of crossing different administrative domains, a notification correlation architecture based on Web Services and policies is proposed. Policies are used as a mean to define the correlation rules.

In this work, the proposed architecture and its implementation are presented, allowing the investigation of Web Services as network management tools, considering the specific case of notification support. This study complements previous investigations of the Network Management Research Group at Federal University of Rio Grande do Sul, showing some Web Services aspects not known in the notification field, besides showing the policy-based network management (PBNM) applied to notifications, a subject that is also not explored until this moment.

Palavras-chave: Web Services, correlation, event, trap, notification, policy-based network management ,PBNM, SNMP.

1 INTRODUÇÃO

Notificações enviadas por dispositivos de rede são essenciais para alertar administradores sobre possíveis problemas ou falhas. Porém, em muitos casos, múltiplas notificações (também conhecidas como alarmes), relacionadas ao mesmo problema, podem ser recebidas por uma estação gerente. Dessa forma, o *software* de visualização do gerente pode ficar sobrecarregado para o operador do sistema, mostrando muitas notificações relacionadas ao mesmo problema. Uma forma possível de diminuir a quantidade de notificações recebidas é através da correlação.

Uma maneira de controlar o comportamento da rede, no contexto de gerenciamento, é a utilização do gerenciamento baseado em políticas (PBNM), para a definição das regras de correlação. Para que o PBNM seja aplicado, uma linguagem de políticas deve ser utilizada. Muitas linguagens têm sido propostas na última década. Basicamente, os modelos de política adotados nessas linguagens permitem a definição de políticas baseadas em eventos-condições-ações ou baseadas em condições-ações.

Atualmente, há um grande interesse da comunidade de gerenciamento de redes nas pesquisas relacionadas com o uso de Web Services (WS) como ferramenta de gerenciamento. Segundo Schönwälder et al. (SCHÖNWÄLDER; PRAS; MARTIN-FLATIN, 2003), os Web Services representam uma revolução no gerenciamento de redes, porque seu uso sugere a substituição do *framework* SNMP (CASE et al., 1990) que possui, atualmente, larga aceitação e utilização. Contudo, mesmo sendo revolucionário, o uso de Web Services no gerenciamento de redes pode não ser viável em sistemas reais que possuem dispositivos dependentes apenas do SNMP. Assim, acredita-se que uma solução mista, combinando SNMP com WS, poderia ser a mais apropriada para a obtenção das vantagens do uso de WS no gerenciamento de redes, e continuar usando dispositivos compatíveis apenas com o SNMP. Os WS são implementados através do uso do protocolo SOAP (CONSORTIUM, 2004), que geralmente utiliza o protocolo HTTP como transporte. Essa ligação SOAP/HTTP é facilmente realizada devido ao grande número de ferramentas de desenvolvimento e suporte disponíveis atualmente.

Na literatura atual, há uma quantidade considerável de investigações sobre o uso de WS, utilizando como protocolo de comunicação o HTTP. Porém, esses WS implicam uma comunicação síncrona, onde o cliente invoca uma operação WS através de uma API, que é executada em um servidor localizado em um computador remoto. Geralmente, o cliente fica bloqueado, aguardando por uma resposta do servidor. Observando o tráfego na rede, pode-se verificar que uma mensagem SOAP primeiramente é enviada do cliente ao servidor, para realizar a chamada à operação WS, e uma mensagem final é enviada do servidor ao cliente, contendo os resultados da operação. Porém, a natureza síncrona da ligação SOAP/HTTP não é apropriada para mensagens de notificação SNMP (*traps*), e muito pouca (se alguma) investigação nesse tema tem sido realizada.

Este trabalho tem como objetivo principal a definição de uma arquitetura de notificação distribuída, capaz de correlacionar notificações relacionadas ao mesmo problema. A arquitetura é composta por agentes SNMP, *gateways* de notificação SNMP/WS, encaminhadores (ou gerentes intermediários) de notificações baseados em políticas e gerente superior. Como sugere o nome, os *gateways* de notificação traduzem notificações SNMP para chamadas WS assíncronas, enquanto os encaminhadores correlacionam e encaminham notificações, baseados em políticas definidas pelo administrador da rede. O gerente superior tem a função de configurar os gerentes intermediários e *gateways*, e receber as correlações para poder disponibilizá-las ao *software* gerente.

Para a criação das regras de correlação, foi escolhida a utilização de uma linguagem hipotética, com comandos semelhantes a linguagens de programação, tais como C e PHP. Essa linguagem hipotética segue o modelo eventos-condições-ações, onde os eventos são as *traps* SNMP enviadas, primeiramente pelos agentes, e, após passarem pelos *gateways*, traduzidas para Web Services. As condições são comparações feitas com os campos da *trap* recebida e condições temporais (ex.: se o horário do recebimento da trap estiver entre 10h e 11h). As ações são basicamente: a atribuição de valores para variáveis da correlação a ser gerada e a definição do gerente que irá receber a *trap* (com ou sem correlação).

A solução proposta foi avaliada com a implementação do protótipo, e medidas de consumo de tráfego e tempo de retardo de transmissão das diferentes tecnologias foram realizadas. As contribuições principais deste trabalho são: a definição e implementação de uma arquitetura de correlação de notificações baseada em políticas e Web Services e a observação da ligação de SOAP com outro protocolo além do HTTP, no contexto de gerenciamento de redes.

O restante deste trabalho está organizado como segue: O capítulo 2 apresenta o conceito de correlação de eventos (no caso deste trabalho, as *traps* são os eventos), suas técnicas de correlação e o conceito de gerenciamento baseado em políticas. O capítulo 3 apresenta o conceito de Web Services, sua aplicação no gerenciamento de redes e a contextualização do problema abordado. O capítulo 4 apresenta a arquitetura de correlação de notificações proposta, enquanto o capítulo 5 apresenta o protótipo da implementação do sistema. O capítulo 6 mostra a avaliação realizada e os resultados do uso da arquitetura proposta em uma rede de testes, e o capítulo 7 encerra o trabalho com conclusões e trabalhos futuros.

2 CORRELAÇÃO DE NOTIFICAÇÕES E GERENCIAMENTO BASEADO EM POLÍTICAS

Uma notificação (também conhecida como alarme) é a informação transmitida por um evento, relacionado a situações consideradas anormais. Notificações de eventos enviadas por dispositivos de rede são essenciais para alertar administradores sobre problemas relacionados à rede. Porém, em muitos casos, múltiplas notificações relacionadas ao mesmo problema, podem ser recebidas por uma estação gerente. Isto pode ocorrer, por exemplo, porque o intervalo definido para o envio de uma notificação é muito pequeno. Dessa forma, o *software* de visualização do gerente pode ficar sobrecarregado mostrando muitas notificações relacionadas ao mesmo problema. Outro cenário que aumenta a quantidade de notificações é quando uma única falha gera várias notificações diferentes. Por exemplo, se um roteador está inoperante e um servidor alcançado através deste roteador está funcionando normalmente, os agentes de monitoração de ambos podem enviar notificações indicando que os dois estão inoperantes ou inacessíveis. A lista abaixo mostra algumas situações comuns relacionadas a possíveis sobrecargas de notificações (CASTRO; NOGUEIRA, 1998):

- Um agente pode enviar várias notificações relacionadas à mesma falha;
- Uma falha intermitente pode produzir uma nova notificação cada vez que a falha é detectada;
- Um dispositivo contendo falha pode fazer com que seja gerado uma notificação cada vez que o dispositivo é requisitado (ex.: uma consulta);
- Uma única falha pode ser detectada por diferentes agentes;
- Uma falha em um dispositivo pode afetar outros dispositivos da mesma rede.

Neste cenário, uma redução no número de mensagens, através da correlação de notificações, é necessária para que os processos de análise e resolução de problemas sejam realizados de forma mais rápida e eficiente. Algumas técnicas podem ser usadas para a correlação de notificações. Nas seções seguintes são abordadas as cinco técnicas comumente utilizadas por sistemas comerciais (ZUPAN; MEDHI, 2003) (LEWIS, 2000), para correlação de eventos[1]; e o gerenciamento baseado em políticas, que é utilizado na arquitetura proposta no capítulo 4.

[1]Neste trabalho, as notificações enviadas pelos agentes são consideradas eventos.

2.1 Raciocínio Baseado em Regras (RBR)

A técnica RBR é considerada a mais comum para correlação de eventos, mesmo quando usada em conjunto com outra técnica. Um sistema RBR consiste de três elementos básicos: memória, repositório de regras e algoritmo de raciocínio (conforme Figura 2.1). A memória consiste de fatos. O repositório de regras representa o conhecimento sobre em quais fatos deve haver inferência e quais ações devem ser aplicadas sobre eles. O algoritmo de raciocínio é o mecanismo que realiza a inferência, verificando a memória e o repositório de regras periodicamente, para confirmar se alguma correlação pode ser feita.

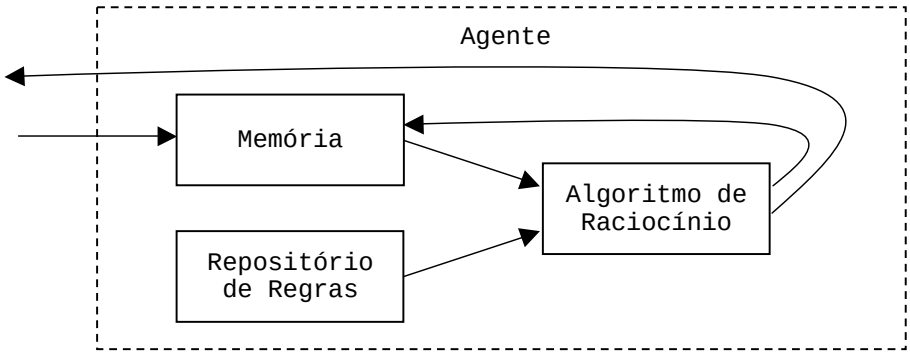

Figura 2.1: Estrutura básica da técnica de correlação RBR

Uma maneira simples de compreender como o algoritmo opera é associá-lo com a inferência clássica *Modus Ponens*, da lógica elementar. A Figura 2.2 mostra um exemplo.

A - um fato na memória

If A then B - uma regra no repositório de regras

Portanto, B - uma inferência feita pelo algoritmo de raciocínio

Figura 2.2: Exemplo da inferência clássica *Modus Ponens*

Como pode ser visto no exemplo, no momento em que o fato A foi detectado na memória, a regra que utiliza como condição a ocorrência desse fato realiza a chamada da ação B.

A grande vantagem da utilização dessa técnica é que as regras são intuitivas. Por outro lado, a grande desvantagem é que não é fácil definir um conjunto de regras que expresse todas situações possíveis (comportamento) na rede. Outro ponto a ser analisado é a questão do conflito de regras, que é responsabilidade de quem as define. Um sistema para evitar o conflito de regras seria muito complicado de ser implementado e, se implementado, exigiria muito poder computacional, visto que teria que realizar a avaliação de diversos fatos e regras (divididas em uma ou mais condições e uma ou mais ações).

Alguns exemplos de ferramentas que utilizam a técnica RBR são: BMC Patrol (ferramenta de gerenciamento de eventos, que provê uma interface gráfica com os recursos do sistema em um ambiente distribuído) (SOFTWARE, 2004), IBM Tivoli

(pacote com várias ferramentas, tendo uma delas, a finalidade de correlacionar eventos) (IBM, 2004), entre outros.

2.2 Raciocínio Baseado em Modelos (MBR)

A técnica MBR representa cada componente do sistema como um modelo, onde o modelo é a representação de uma entidade física (ex.: *hub*, roteador, porta, etc.) ou entidade lógica (ex.: LAN, MAN, WAN, serviço, etc.). Um modelo que representa uma entidade física realiza comunicação direta com a entidade (ex.: através de SNMP). Uma descrição de um modelo inclui três categorias de informação: atributo (ex.: endereço IP), relação com outro modelo (ex.: "A está conectado com B") e comportamento (ex.: "se eu sou um modelo de um servidor X e não obtenho resposta do servidor X após três tentativas, então realizo a troca do valor do meu atributo de alarme"). Como pode ser observado, a categoria de informação comportamento é semelhante à técnica RBR explicada anteriormente. Nessa técnica, a correlação de eventos é o resultado da colaboração entre os modelos, ou seja, é o resultado dos comportamentos coletivos de todos modelos.

Na Figura 2.3, é exibido um exemplo da técnica de correlação MBR. O exemplo mostra uma sala de aula com quatro alunos (A, B, C e D) e um professor (R), sendo que cada aluno é um modelo de computação real (i.e., um espelhamento de imagem em *software*). Os alunos A e B representam sistemas de servidores NT e os alunos C e D representam sistemas de estações de trabalho UNIX. Esses modelos possuem alguma forma de comunicação com os sistemas reais (fora da sala de aula). Os modelos verificam se os sistemas reais estão ativos a cada dez minutos e também coletam informações. A sala de aula é um modelo de sub-rede, R é um modelo do roteador da sub-rede e R se comunica com o roteador. Uma situação de falha detectada poderia ser: o aluno A tenta entrar em contato com seu servidor NT e não obtém resposta; tenta mais algumas vezes e não obtém sucesso. Então o aluno A envia uma mensagem para o professor, pedindo que ele entre em contato com o roteador. Se o professor responder que o roteador está operando, o aluno A infere que há uma falha no seu servidor NT e envia um alarme. Caso contrário, o aluno A constata que o servidor NT deve estar funcionando corretamente, mas o roteador não.

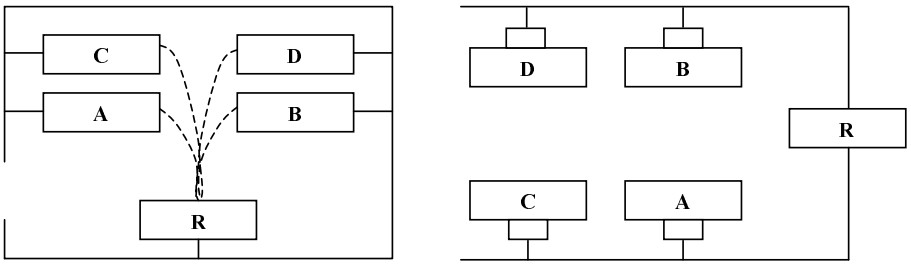

Figura 2.3: Exemplo da técnica de correlação MBR

O melhor exemplo de ferramenta, que utiliza a abordagem MBR, é o Spectrum (APRISMA, 2004). O Spectrum contém modelos (classes na terminologia de orientação a objeto) para aproximadamente mil tipos de entidades físicas e lógicas.

2.3 Grafo de Transição de Estados (STG)

A técnica STG é pouco utilizada em *softwares* comerciais pois, além de ser mais complexa, a implementação desta técnica exige mais recursos computacionais. Outro fator negativo é que esta técnica não é tão intuitiva como as mostradas anteriormente, o que dificulta a utilização por parte de usuários com pouco conhecimento. Os conceitos-chave para a técnica STG são: um *token*, um estado, um arco, uma transição de um *token* de um estado para outro estado, através de um arco, e uma ação que é ativada quando o *token* entra em um estado.

Considerando o cenário apresentado na seção anterior (um servidor NT parou de responder, mas o problema era que o roteador tinha falhado), a Figura 2.4 mostra como representar essa situação utilizando a técnica STG.

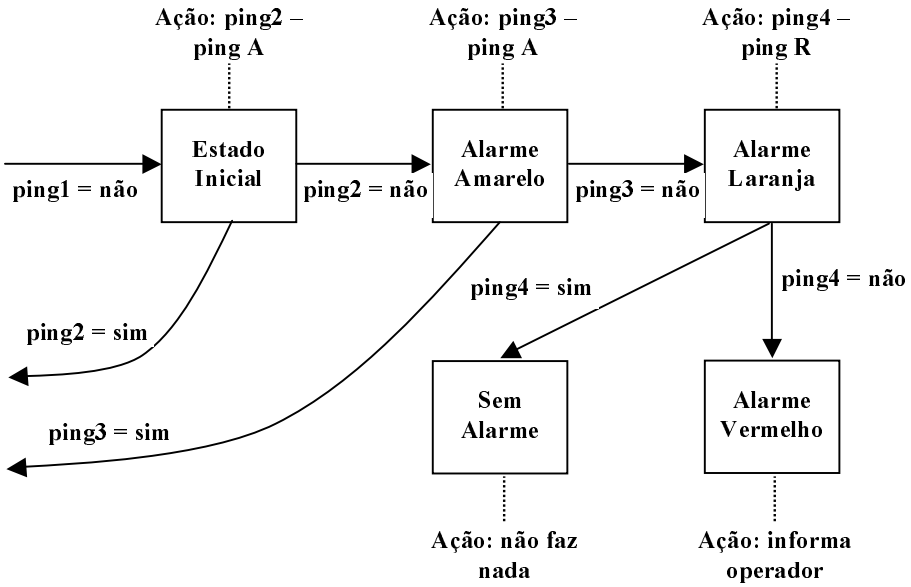

Figura 2.4: Exemplo da técnica de correlação STG

Deve-se salientar que a Figura 2.4 mostra um único domínio de interesse - um servidor NT. Porém, um sistema pode conter milhares de tipos de componentes. Dessa forma, um grande número de grafos de transição de estados podem ser necessários para representar o sistema. Felizmente, desde que um STG genérico possa ser aplicado a componentes do mesmo tipo, não há necessidade de haver STGs separados para esses componentes.

O melhor exemplo de ferramenta para essa técnica é o SeaGate NerveCenter (VERITAS, 1998), que geralmente é utilizado em conjunto com o HP Open-View (HEWLETT-PACKARD, 2004), embora possa ser utilizado separadamente, comunicando-se diretamente com dispositivos gerenciados através de SNMP (conforme mostra Figura 2.5).

Como pode ser visto na Figura 2.5, o NerveCenter utiliza a técnica RBR para selecionar eventos significativos do fluxo de eventos do OpenView, e apenas esses eventos são passados ao conjunto de STGs para realização da função de mapeamento evento/alarme. Em literatura comercial, encontra-se o NerveCenter como um sistema RBR, o que é um engano. O NerveCenter utiliza dois tipos de representação: regras (RBR) e STGs.

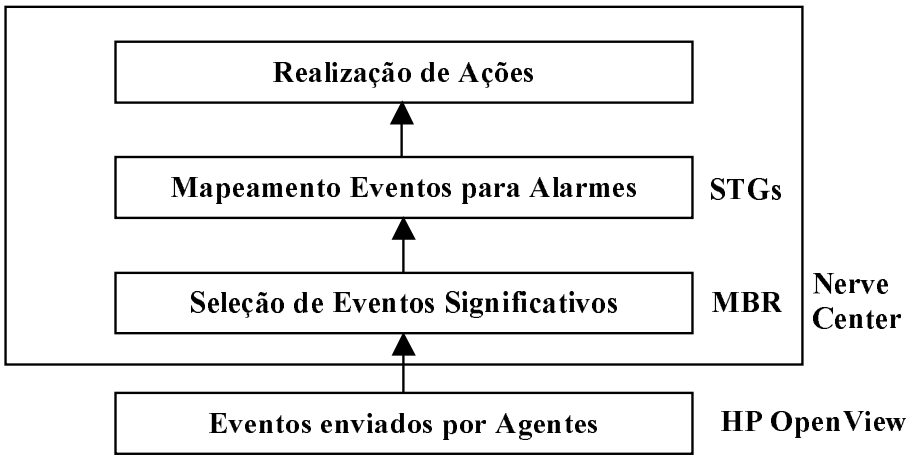

Figura 2.5: Interação das ferramentas OpenView e NerveCenter

2.4 Codebook

A técnica *codebook* pode ser considerada a mais complexa das técnicas apresentadas neste trabalho. O fato de ela trabalhar com matrizes não agrada muitos administradores, que preferem algo mais simples para poder realizar suas correlações. Os principais conceitos dessa técnica de correlação são: matriz de correlação, codificação, *codebook* e decodificação. A correlação é feita da seguinte forma: considerando, por exemplo, um domínio de interesse, composto por quatro tipos de eventos (e1, e2, e3, e4) e dois tipos de alarmes (A1, A2). Para que a correlação seja feita, deve-se saber quais são os conjuntos de eventos que geram determinados alarmes. Essa informação é organizada em uma matriz de correlação, conforme mostrada na Figura 2.6.

	A1	A2
e1	1	1
e2	0	0
e3	0	1
e4	0	0

	A1	A2	A3	A4	A5
e1	1	0	0	1	0
e2	1	1	1	1	0
e3	1	1	0	1	0
e4	1	0	1	0	1
e5	1	0	1	1	1
e6	1	1	1	0	0
e7	1	0	1	0	0
e8	1	0	0	1	1
e9	0	1	0	0	1
e10	0	1	1	1	0
e11	0	0	0	1	1
e12	0	1	0	1	0
e13	0	1	0	1	1
e14	0	0	0	0	0
e15	0	0	1	0	1
e16	0	1	1	0	0
e17	0	1	0	1	1
e18	0	1	1	1	0
e19	0	1	1	0	1
e20	0	0	0	0	1

Uma matriz de correlação

	A1	A2	A3	A4	A5	A6
e1	1	0	0	1	0	1
e2	1	1	1	1	0	0
e4	1	0	1	0	1	0

Um primeiro *codebook*

	A1	A2	A3	A4	A5	A6
e1	1	0	0	1	0	1
e3	1	1	0	1	0	0
e4	1	0	1	0	1	0
e6	1	1	1	0	0	1
e9	0	1	0	0	1	1
e18	0	1	1	1	0	0

Um segundo *codebook*

Figura 2.6: Matriz de correlação utilizada na técnica codebook

Como pode ser visto na Figura 2.6, a ocorrência do evento e1 indica o alarme A1, e a ocorrência dos eventos e1 e e3 indicam a ocorrência do alarme A2. A codificação transforma a matriz em uma matriz reduzida, que é chamada de *codebook*.

Depois de criado o *codebook*, o sistema pode realizar o mapeamento evento/alarme. Esse processo é chamado decodificação. Obviamente, o mapeamento poderia ser feito utilizando diretamente a matriz de correlação original, porém, o desempenho seria muito pior, devido ao grande número de comparações a serem realizadas.

O melhor exemplo de ferramenta que utiliza esta técnica de correlação é o InCharge (SMARTS, 2004), desenvolvido pelo System Management Arts (SMARTS). O InCharge está geralmente integrado com o HP OpenView ou IBM NetView. Esta ferramenta contém uma biblioteca genérica de classes de rede. Com essas classes, pode-se derivar classes de domínio específico, com a adição de atributos apropriados e declarações de métodos, para a criação de um modelo preciso do domínio. Por fim, pode-se adicionar definições de eventos ao modelo, tornando esta técnica parecida com a MBR descrita anteriormente.

2.5 Raciocínio Baseado em Casos (CBR)

Ao contrário das técnicas apresentas anteriormente, a CBR tem como idéia básica a rechamada, a adaptação e a execução de episódios de problemas antigos (já resolvidos), na tentativa de lidar com um problema novo. Episódios antigos de problemas resolvidos são representados como casos em uma biblioteca de casos. Quando confrontadas com um novo problema, um sistema CBR recupera um caso similar e tenta adaptar o caso na tentativa de resolver o problema atual. A estrutura geral da arquitetura CBR é mostrada na Figura 2.7.

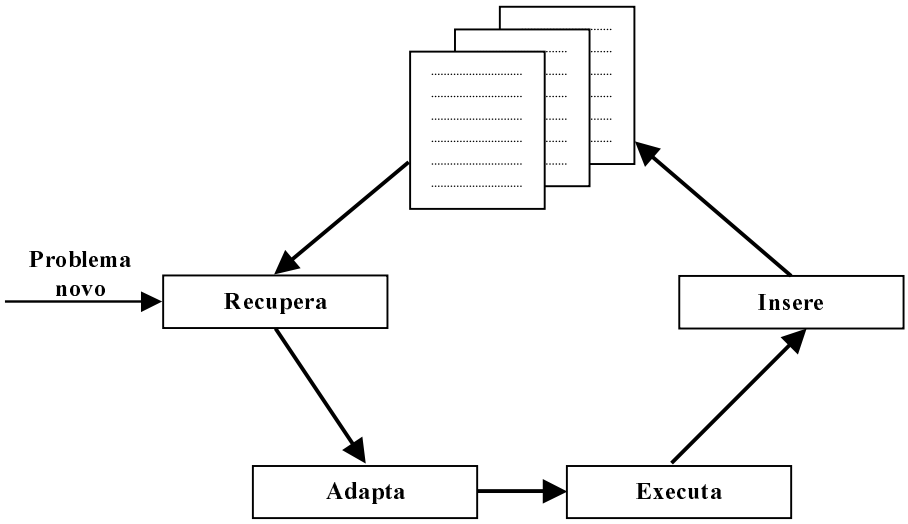

Figura 2.7: Estrutura da arquitetura CBR

O desafio da técnica CBR é o desenvolvimento de uma métrica de similaridade. Um caso que possui o maior número de equivalências com o problema atual não garante ser o melhor entre os casos antigos. Alguns campos no caso podem ser irrelevantes e "enganar" o sistema. Por isso, critérios de relevância são necessários para indicar que tipos de informação devem ser considerados em um determinado problema.

Um exemplo de ferramenta CBR é a SpectroRx da Aprisma. O SpectroRx é uma ferramenta adicional do SPECTRUM. Como descrito anteriormente, o SPECTRUM realiza a correlação de eventos utilizando a técnica MBR. Quando uma falha é

identificada, persiste o problema de localizar uma solução para a falha. Experiências antigas com falhas semelhantes são importantes, e este é o tipo de conhecimento que o SpectroRx disponibiliza ao desenvolvedor.

2.6 Gerenciamento Baseado em Políticas (PBNM)

No contexto de gerenciamento de redes, políticas são uma maneira de controlar o comportamento da rede. Na prática, o gerenciamento baseado em políticas também tem a intenção de reduzir a complexidade e diversidade das interfaces de gerenciamento atuais (ex.: SNMP, TELNET/CLI, etc.). Isso é possível porque arquiteturas PBNM permitem ao administrador da rede definir os comportamentos da rede através de políticas, que são traduzidas (pela arquitetura) para ações de configuração nos dispositivos da rede. Dessa maneira, administradores não precisam se preocupar mais com a diversidade das interfaces de gerenciamento, sendo que essa variedade é tratada pela arquitetura PBNM.

A definição de políticas requer o uso de linguagens de políticas. Muitas linguagens têm sido propostas na última década (ex.: Ponder (DAMIANOU et al., 2001) e PDL (LOBO; BHATIA; NAQVI, 1999)). Basicamente, os modelos de política adotados nessas linguagens permitem a definição de políticas baseadas em eventos-condições-ações ou baseadas em condições-ações. Embora o modelo eventos-condições-ações pareça ser mais completo, o IETF (*Internet Engineering Task Force*) tem trabalhado na definição de arquiteturas, protocolos e modelos de informação para modelos condições-ações[2].

A linguagem de políticas por si só não é suficiente para a aplicação de PBNM: uma arquitetura PBNM também é necessária. Uma das arquiteturas PBNM mais aceitas é a definida pelo IETF, que é composta por quatro módulos-chave: ferramenta de políticas, repositório de políticas, pontos de decisão de políticas (PDP) e pontos de aplicação de políticas (PEP) (Figura 2.8). A ferramenta de políticas é usada pelo administrador da rede para editar as políticas armazenadas no repositório de políticas. Elas são armazenadas para que sejam reusadas, tanto na aplicação de políticas, quanto na definição de uma nova política. As traduções de políticas, para ações de configuração em dispositivos, são realizadas pelos PDPs. Finalmente, os PEPs são os elementos nos dispositivos gerenciados que, efetivamente, aplicam a política (ex.: disciplinas de filas e interfaces são PEPs comuns em roteadores).

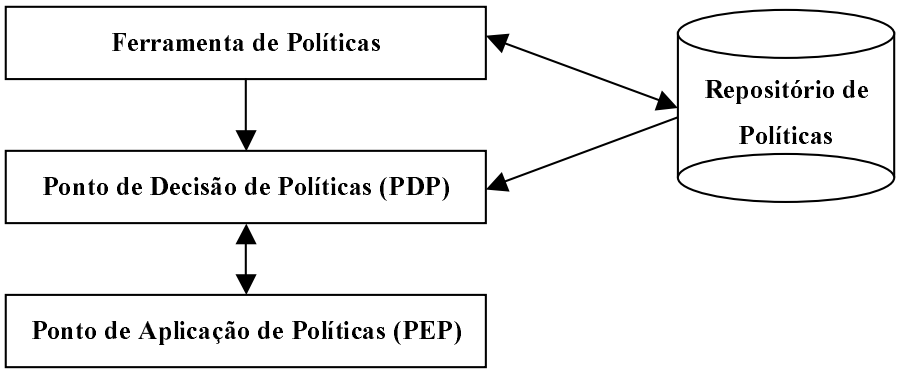

Figura 2.8: Arquitetura PBNM definida pelo IETF

Essa maneira de operação, chamada *provisioning*, é apropriada para redes que

[2]Freqüentemente é dito que eventos estão implicitamente no modelo condições-ações, conceito que muitos pesquisadores discordam, no entanto.

suportam *Diffserv*. A mesma arquitetura do IETF, no entanto, pode operar de acordo com o modelo *outsourcing*, apropriado para redes que suportam *IntServ* e usam o protocolo RSVP (BRADEN et al., 1997) como um mecanismo de sinalização para alocação de recursos de rede. O IETF está definindo o protocolo COPS (DURHAM et al., 2000)) para suporte ao modelo *outsourcing*, bem como o protocolo COPS-PR (CHAN et al., 2001) para suporte ao modelo *provisioning*. Além disso, o IETF tem trabalhado em modelos de informação tais como o PCIM e PCIMe (MOORE, 2003).

Uma política pode ser definida de duas maneiras (WESTERINEN et al., 2001):

- Um objetivo definido, curso ou método de ação para guiar e determinar decisões atuais e futuras. Políticas são implementadas ou executadas em um contexto particular (como políticas definidas em uma unidade organizacional);
- Um conjunto de regras para administrar, gerenciar e controlar o acesso a recursos da rede.

No contexto da arquitetura de correlação de eventos definida no capítulo 4, a segunda maneira será explorada. Com as políticas definidas pelo(s) administrador(es) da rede, a arquitetura é capaz de gerenciar os eventos, correlacionando-os e enviando-os para um encaminhador (gerente intermediário) pré-definido.

Uma política pode ser usada para modificar o comportamento de um sistema. Separando as políticas dos gerentes que as interpretam, pode-se modificá-las para que haja mudança do comportamento e da estratégia do sistema de gerenciamento, sem ter que atualizar o código de implementação do gerente. A arquitetura proposta pode alterar o comportamento do sistema, desabilitando políticas ou trocando políticas antigas por novas, sem ter que reinicializar o sistema (LUPU; SLOMAN, 1999).

3 WEB SERVICES (WS)

Um conceito que recentemente tem conquistado a atenção de grandes líderes da indústria é o Software como Serviço (SaaS). O SaaS pode ser descrito como um modelo de software que é desenvolvido, gerenciado, atualizado e recebe suporte sob demanda. São colocados em servidores internos ou externos à organização e lembram o funcionamento de um *mainframe* e seus terminais. As atualizações são feitas apenas no servidor, evitando custo e tempo maiores (caso a atualização fosse feita em cada usuário) (GRESCHLER; MANGAN, 2002a) (GRESCHLER; MANGAN, 2002b). Esse conceito é utilizado nos Web Services, explicados nas seções seguintes.

3.1 Conceito de WS

Web Services são componentes independentes de aplicações disponibilizados na Web de tal maneira que outras aplicações Web possam achá-los e utilizá-los. Variando de simples a complexos, os Web Services trazem a promessa de flexibilidade e de computação distribuída na Internet.

Os Web Services, considerados sucessores de CORBA e DCOM, são um dos principais passos na evolução da Web. Eles permitem que objetos ativos sejam colocados em Web *sites* para fornecer serviços distribuídos a clientes. Os Web Services têm sido utilizados em *e-commerce*, no gerenciamento de informações distribuídas, já que sistemas de bancos de dados distribuídos têm dificuldades com compatibilidade de plataformas e *softwares* (ABITEBOUL; BENJELLOUN; MILO, 2002). Também podem ser descritos como aplicações capazes de executar transações na Internet e podem ser acessados tanto pelos clientes (ex.: *browsers*) como por outros Web Services (SAHAI et al., 2001).

Os Web Services podem ser vistos como integradores de computadores, dispositivos, bancos de dados e redes em um único sistema virtual, onde os usuários podem trabalhar utilizando *browsers* (VAUGHAN-NICHOLS, 2002). Uma das suas principais utilidades é a chamada de procedimento remoto: usuários ativam determinadas tarefas em um dispositivo/máquina remoto.

Uma definição mais completa para Web Services seria: um serviço disponível na Internet que utiliza um sistema de mensagens XML e não depende de sistema operacional nem de linguagem específica. Na Figura 3.1, indica-se como é feita a comunicação de um Web Service básico em alto nível (sem detalhes).

Quando dois Web Services se conectam, eles devem concordar com a utilização de um determinado protocolo de troca de documentos (SOAP é o padrão) e os formatos apropriados dos documentos. A troca de mensagens entre Web Services pode ser assíncrona. Serviços que enviam uma solicitação não precisam ficar bloqueados, aguardando por uma mensagem de resposta. O primeiro exemplo na figura 3.2 mostra

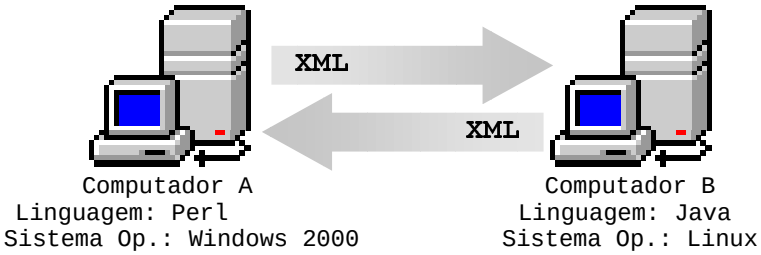

Figura 3.1: Web Service básico

uma solicitação única com múltiplas respostas. No segundo exemplo, pode ser visto um cenário contendo um *broker* (explicado na seção seguinte), onde cada solicitação é enviada ao *broker*, mas as respostas são recebidas diretamente dos provedores dos serviços (SAHAI et al., 2001).

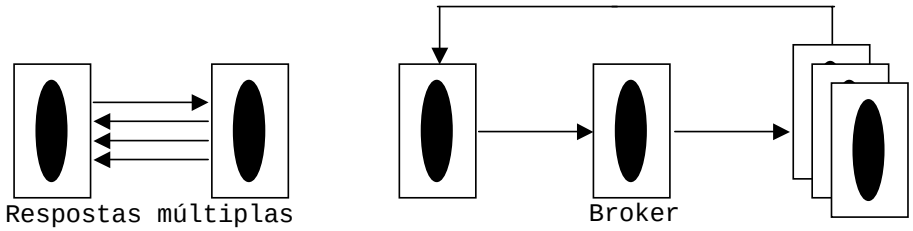

Figura 3.2: Mensagens assíncronas entre Web Services

3.2 Arquitetura dos WS

Um Web Service típico é constituído das seguintes entidades:

- ***Service Provider***: cria os Web Services e os publica, através de registros realizados com os *Service brokers*;
- ***Service Broker***: mantém o registro de serviços publicados;
- ***Service Requester***: procura serviços no registro dos *Service Brokers*. Após encontrar o serviço, é estabelecida uma comunicação entre a aplicação requerente e o *Service Provider*, para utilização dos serviços.

A figura 3.3 mostra como é realizada a comunicação entre as entidades citadas.

3.3 Tecnologias Utilizadas nos WS

O núcleo dos Web Services é formado por três tecnologias (CURBERA et al., 2002):

- ***Web Services Description Language* (WSDL)**: linguagem que os programadores podem utilizar para descrever as interfaces dos Web Services. O WSDL utiliza XML para descrever definições de tipos de dados, operações suportadas pelo serviço, formatos de mensagens de entrada e saída, endereço de rede, associações de protocolos, entre outros;

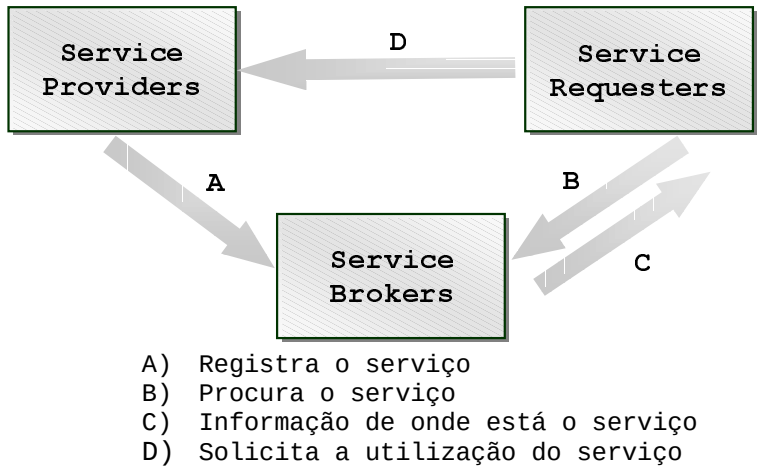

Figura 3.3: Arquitetura dos Web Services

- ***Simple Object Access Protocol* (SOAP)**: provê a comunicação entre os Web Services e as aplicações clientes. SOAP é um protocolo simples e "leve", que permite a troca de dados formatados com XML, na Web. Aplicações cliente utilizam mecanismos SOAP para acessar Web Services. O SOAP está se tornando o padrão para mensagens XML, sob supervisão do W3C. A especificação permite o transporte de mensagens XML em protocolos de mais alto nível, tais como HTTP, FTP e SMTP. O HTTP é o mais utilizado porque evita mais a possibilidade de ser filtrado em *firewalls*, e há diversos *toolkits* que o utilizam na implementação de Web Services. O SOAP é independente de protocolo de comunicação, de sistema operacional e de linguagem de aplicação (ABITEBOUL; BENJELLOUN; MILO, 2002). Abaixo, é dado um exemplo de mensagem SOAP (SAHAI et al., 2001).

```
<SOAP ENV:Envelope xmlns:SOAP ENV="http://schemas.
xmlsoap.org/soap/envelope/" SOAP ENV:encodingStyle=
"http://schemas.xmlsoap.org /soap/encoding/">
   <SOAP ENV:Header>
      <Id>1</Id>
   </SOAP ENV:Header>
   <SOAP ENV:Body>
      <PurchaseOrder>
         <Item count = 100>Postit stick notes</Item>
         <Item count = 200>Stapler</Item>
      </PurchaseOrder>
   </SOAP ENV:Body>
</SOAP ENV:Envelope>
```

- ***Universal Description, Discovery and Integration* (UDDI)**: permite aos Web Services registrarem suas características, para que outras aplicações possam saber as funcionalidades desses Web Services. O UDDI é um Web Service baseado em XML e SOAP. Os Web Services tipicamente registram dois tipos de informação utilizando o UDDI: *tModel* e *businessEntity*. Os *tModel's* descrevem o comportamento dos Web Services. Os *businessEntity's* descrevem a implementação do serviço e referem-se a múltiplos *tModel's*. Na figura 3.4, é mostrado como

aplicações cliente procuram os Web Services necessários em registros UDDI e os solicitam aos hosts que possuem os Web Services armazenados.

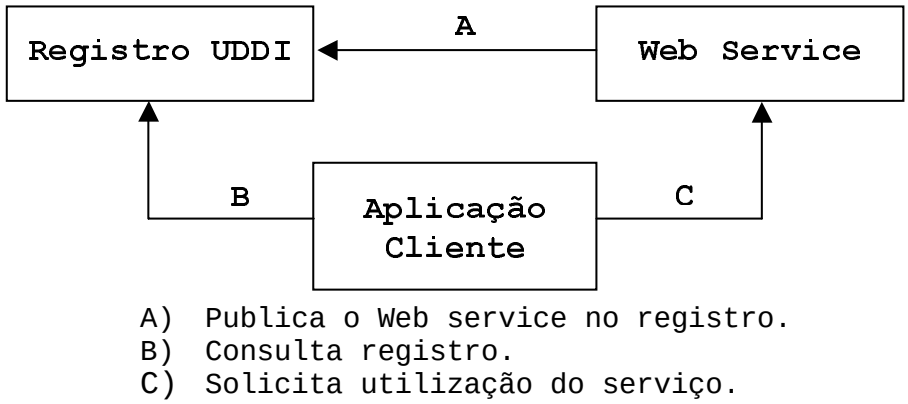

Figura 3.4: Aplicações cliente procurando Web Services, através do registro UDDI

O SOAP não é o único mecanismo para transportar mensagens XML; ele foi citado por ser o mais utilizado e estar se tornando padrão. Na figura 3.5, é mostrada a pilha WS, onde podem ser vistas outras formas de se transmitir mensagens XML. Como pode ser visto, o XML pode ser transmitido diretamente sobre o protocolo de transporte, sem a utilização de SOAP ou XML-RPC (HAROLD, 2003). Na figura, é mencionado o BEEP, que é um *framework* do IETF. Em particular, o BEEP está ligado diretamente ao TCP e inclui algumas características que o fazem melhor que o HTTP. Algumas características são: um protocolo inicial de *handshake*, autenticação, segurança e tratamento de erros. A principal vantagem sobre o HTTP é que o BEEP requer 30 bytes de overhead para cada mensagem, o que é bem melhor que o HTTP, e, além disso, o BEEP suporta múltiplos canais de dados sobre a mesma conexão.

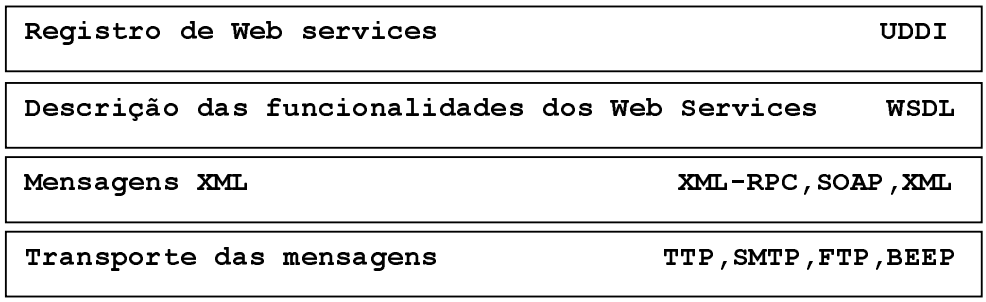

Figura 3.5: Pilha dos Web Services

A seqüência de passos de uma solicitação de serviço é mostrada na figura 3.6.

3.4 Desempenho dos WS

Uma das preocupações com os Web Services é o desempenho, pois XML é baseado em texto, o que aumenta a quantidade de dados a serem processados. O encapsulamento em HTTP (ou outro protocolo de alto nível) deixa esse processo mais lento e, se for utilizado um protocolo para adicionar segurança, como o SSL, o desempenho diminui ainda mais. Todos esses fatores podem tornar os Web Services impraticáveis para aplicações que utilizam redes com pouca vazão, como conexões que utilizam modems (redes *dial-up*) (VAUGHAN-NICHOLS, 2002).

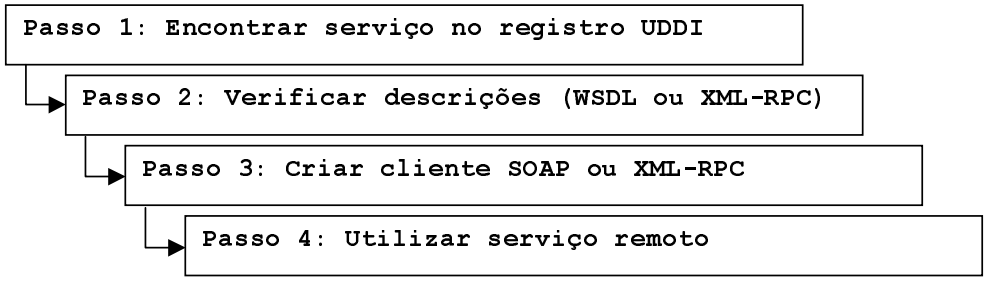

Figura 3.6: Passos para solicitação de um serviço

3.5 Vantagens e Desvantagens do Uso de WS

Segundo pesquisas realizadas em julho de 2003 (BLANK, 2003), as implementações de Web Services reduziram os custos de desenvolvimento e manutenção significativamente. Cerca de 25% das empresas entrevistadas disseram que sem os Web Services não seria possível a implementação de seus projetos. As quatro principais razões para a utilização de Web Services são:

- **Simplicidade**: comparado às interfaces de programação EDI (utilizado para troca de informações eletrônicas entre organizações em um formato estruturado), o WSDL é muito mais fácil de ser entendido. Mesmo quem não é programador pode utilizar ferramentas de criação de Web Services para criar soluções úteis;

- **Interoperabilidade**: a razão mais citada para a utilização de Web Services é a interoperabilidade de plataforma e tecnologia. Com ela, filiais podem comunicar-se facilmente com a matriz ou entre si, sem a necessidade de todas implementarem a mesma tecnologia;

- **Abstração**: refere-se ao processo de esconder todos os dados de um objeto, exceto os mais relevantes, com o objetivo de reduzir a complexidade e aumentar a eficiência de processamento. As empresas entrevistadas argumentam que a abstração provida pelos Web Services:

  1. **Facilita a integração entre organizações**: com a integração facilitada, é necessário apenas que ambas as partes concordem com a interface, sem se preocuparem com as diferentes tecnologias envolvidas;
  2. **Reduz a complexidade**: um Web Service pode disparar um processo complexo que cause impacto em vários sistemas e dados, sem que o usuário que o utilizou precise preocupar-se com a complexidade envolvida;
  3. **Torna possível aplicações *plug-and-play***: se uma empresa utiliza a interface WSDL, ela pode trocar um subsistema sem causar impacto em todo o sistema. Se a interface for feita com o devido cuidado, a migração de um sistema pode ocorrer de forma que o mesmo permaneça estável.

- **Reuso**: um dos grandes benefícios dos Web Services ocorre quando integrações recorrentes são feitas, ou seja, não é perdido tempo em programação de novos serviços. Algumas empresas, que levavam semanas em projetos de integração, levam dias com a utilização de Web Services.

Quando se fala em desenvolvimento de Web Services, sabe-se que existem vários desafios para essa nova arquitetura. Porém, a segurança é vista como a principal preocupação. Os Web Services são orientados a mensagens e não têm a garantia de uma conexão direta entre o provedor do serviço e o cliente que solicitou o serviço. Portanto, muitas comunicações orientadas à conexão, consideradas tradicionais, têm os mesmos desafios de segurança: são inapropriados ou insuficientes para a arquitetura de segurança dos Web Services. As práticas e padrões de segurança apropriados para um uso interno de uma organização podem ser completamente inapropriados quando usuários externos são introduzidos.

Se a comunicação entre a origem e o destino for privada, o problema de segurança pode ser resolvido com a utilização de protocolos orientados à conexão, que garantem segurança, como por exemplo o HTTPS. Como os Web Services são orientados a mensagens, as comunicações podem passar por muitos dispositivos intermediários antes de chegarem ao destino. Portanto, a utilização de um protocolo orientado à conexão não pode garantir confidencialidade entre a origem e o destino.

A integridade, assegurando que as comunicações não são alteradas entre a origem e o destino, também é resolvida no nível de conexão, utilizando CRCs e *checksums*, ou protocolos seguros orientados à conexão, como o SSL. Os protocolos seguros orientados à conexão, que garantem a integridade das comunicações, têm o mesmo problema em assegurar a integridade quando as mensagens passam por vários intermediários antes de chegarem ao destino. Além disso, apresentam outro problema: se uma mensagem é armazenada por um determinado período, ela deveria ser verificada no futuro para saber se foi modificada, mas mecanismos orientados à conexão não conseguem fornecer essa garantia.

Como os Web Services têm muitas similaridades com aplicações Web, algumas técnicas de segurança de aplicações Web podem ser utilizadas diretamente pelos Web Services. Como já foi mencionado, o HTTPS pode ser utilizado, assim como o HTTP, juntamente com algum esquema de autenticação com usuário e senha. Além disso, a utilização dos cabeçalhos HTTP e de *cookies* podem ajudar no desafio das comunicações seguras e informações de seções entre a origem e o destino (STANTON, 2003).

Como os Web Services geralmente operam sobre XML, um conjunto completamente novo de padrões está em desenvolvimento para prover segurança para documentos XML. O padrão de criptografia XML define como inserir dados criptografados em documentos XML e o padrão de assinatura digital XML define como codificar uma assinatura digital em um documento XML. Um novo padrão, o SAML, que está sob supervisão da OASIS (OASIS, 2003), permite a propagação e validação das questões de segurança na arquitetura dos Web Services.

Com diferentes padrões definindo diferentes componentes para garantir segurança nos Web Services, fez-se necessário uma especificação. Essa iniciativa foi tomada pelas empresas Microsoft, IBM e VeriSign, atualmente coordenada pela OASIS. Essa especificação tenta prover um modelo de fácil entendimento para integridade de mensagens, autenticação e confidencialidade, especificando como incluir itens como assinatura digital e criptografia em documentos SOAP. É esperado que essa especificação seja mais utilizada que outros padrões, como o SAML, em questões de segurança (STANTON, 2003).

3.6 Web Services aplicados no Gerenciamento de Redes e Contextualização do Problema

Como mencionado na Introdução, provavelmente o mecanismo de notificação mais utilizado atualmente seja o envio de *traps* SNMP. No entanto, esse mecanismo possui muitas restrições (desvantagens) conhecidas, tais como ausência de confirmação de recebimento e limitação no tamanho da informação carregada pelas mensagens de notificação. Uma completa substituição das *traps* SNMP não é possível, porque isso envolveria a atualização de IOS de dispositivos ou até mesmo uma completa substituição dos dispositivos, o que não é viável na maioria das redes em operação.

Embora os WS sejam apontados como uma arquitetura revolucionária (SCHÖNWÄLDER; PRAS; MARTIN-FLATIN, 2003) para o gerenciamento de redes (poderia então substituir as *traps* SNMP), acredita-se que uma solução mista, combinando SNMP com WS, poderia ser a mais apropriada para a obtenção das vantagens do uso de WS no gerenciamento de redes, e continuar usando dispositivos compatíveis apenas com o SNMP.

No contexto de gerenciamento de redes, G. Pavlou et al. (PAVLOU et al., 2004) define que os WS possuem grande semelhança com CORBA e suportam interações de chamadas remotas do tipo solicitação (*request*), resposta (*response*) e erro (*error*).

Outras investigações têm sido realizadas com a intenção de observar aspectos relacionados ao desempenho dos WS no gerenciamento de redes. Jeroen van Sloten et al. (SLOTEN; PRAS; SINDEREN, 2004) investigam como *gateways* SNMP/WS podem reduzir o consumo de banda. R. Neisse et al. e T. Fioreze et al. realizam avaliações de *gateways* SNMP/WS no mapeamento de mensagens originais SNMP para operações WS (NEISSE et al., 2004) (FIOREZE et al., 2005). A. Pras et al. comparam o desempenho do gerenciamento utilizando SNMP com o desempenho utilizando Web Services (PRAS et al., 2004). H. Choi et al. (CHOI et al., 2004) investigam a configuração de dispositivos baseada em XML.

No geral, é possível afirmar que as investigações atuais observam o tráfego dos WS e o desempenho associado, quando há a tradução e integração (utilizando algum enfoque) de operações/informações SNMP, em um ambiente de gerenciamento de rede que permite a utilização de WS. Se separarmos o SNMP em mensagens *request/reply* e mensagens assíncronas (*traps*), as mensagens *request/reply* resultarão nas mais usadas para o gerenciamento do que as *traps*. Assim, não é surpresa perceber que investigações de WS e SNMP são relacionadas a mensagens SNMP *request/reply* e, até onde os autores deste trabalho têm conhecimento, investigações específicas relacionadas a notificações não foram realizadas até hoje.

As próximas seções apresentam a solução proposta e um protótipo da implementação para a realização da correlação de eventos baseada em WS, sem excluir os dispositivos que suportam (apenas) o SNMP. Foram utilizados elementos SNMP dessa arquitetura para avaliar os WS como uma ferramenta de notificação.

4 SOLUÇÃO PROPOSTA

Neste capítulo, serão apresentados: a arquitetura de correlação de eventos proposta, a linguagem de políticas utilizada para a correlação dos eventos, o que são e como funcionam os *gateways*, os gerentes intermediários (GIs) e o gerente superior (GS).

4.1 A Arquitetura de Correlação de Notificações

O objetivo da arquitetura proposta (Figura 4.1) é correlacionar eventos de forma hierárquica, para que haja a diminuição de notificações, geradas por dispositivos, que chegam ao gerente da rede. Para que isso ocorra, os eventos (*traps* SNMP) recebidos pelos *gateways* são convertidos para mensagens SOAP/SMTP e enviadas a um gerente intermediário (GI). Cada GI correlaciona os eventos e, possivelmente, envia a correlação a um próximo GI. Esse processo se repete até que os eventos correlacionados cheguem ao gerente superior (GS).

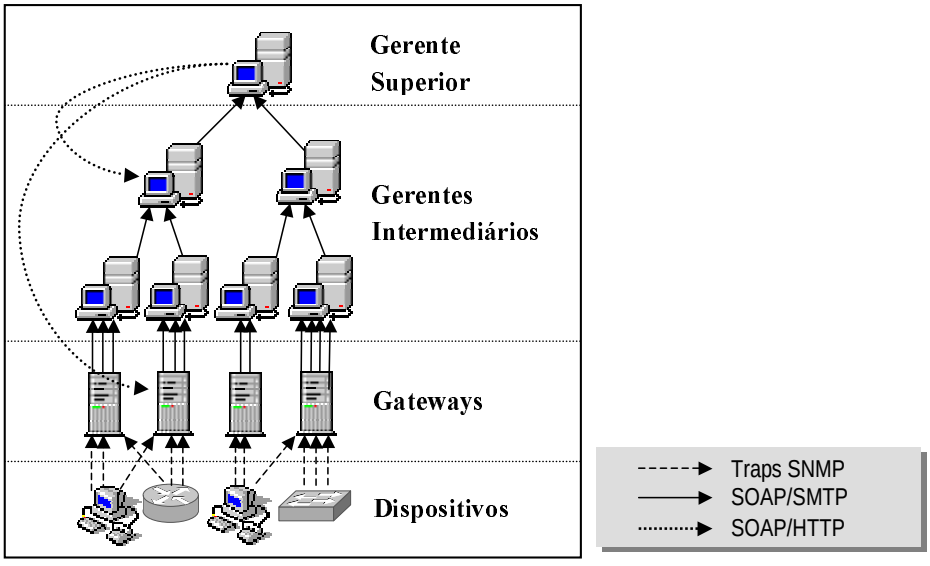

Figura 4.1: Arquitetura de correlação de eventos.

A seguir, é explicado cada componente da arquitetura mostrada na Figura 4.1:

- **Agentes SNMP**: responsáveis pela geração de eventos que, no caso específico do problema abordado, são *traps* SNMP. Os agentes SNMP devem ser configurados para que enviem suas *traps* a um determinado *gateway* (endereço IP);

- ***Gateways***: convertem as mensagens SNMP para chamadas Web Services. Esses Web Services são baseados em SOAP e utilizam como protocolo de transporte o SMTP, por se tratarem de eventos de natureza assíncrona;

- **Gerentes Intermediários**: recebem os eventos já convertidos em mensagens SOAP/SMTP e, baseados em políticas previamente cadastradas pelo gerente superior, realizam a correlação dos eventos e os enviam a um gerente pré-configurado (padrão) ou a um gerente definido na política utilizada pela correlação (pode ser um outro gerente intermediário ou o gerente superior);

- **Gerente Superior**: recebe os eventos correlacionados e os armazena em um repositório, para que *softwares* de gerência possam consultá-los. Além disso, realiza as configurações nos gerentes intermediários e *gateways*, tais como: definição de gerente-destino, envio de políticas do repositório global a um repositório local, abordagem utilizada (otimista ou pessimista, explicadas na seção 4.4), entre outras. Mais detalhes dessas configurações serão abordados neste capítulo.

A arquitetura provê uma transmissão garantida (utilização de protocolo com garantia de entrega) dos alarmes, exceto na transmissão agente/*gateway* (que geralmente é feita em redes locais, onde a perda é considerada pequena). A utilização dos protocolos HTTP e SMTP para transporte das chamadas WS permite que a arquitetura possa ser utilizada em domínios administrativos diferentes, já que esses protocolos não costumam ser interceptados em *firewalls*.

Como pode ser visto na Figura 4.1, o SMTP é utilizado para transportar as notificações, enquanto o HTTP é utilizado para ações de configuração. As ações de configuração também poderiam utilizar a ligação SOAP/SMTP, mas esse tipo de ação requer mensagens do tipo *request/reply*, já que o administrador precisa saber se as ações foram realizadas no ato e com sucesso.

Analisando ainda a Figura 4.1, pode ser observado que, para cada *trap* SNMP recebida pelo *gateway*, uma chamada WS é realizada ao gerente intermediário de primeiro nível. A partir desse gerente, com a realização de correlações, o número de chamadas WS diminui, até chegar ao gerente superior, com todas as correlações possíveis realizadas. Para que essas correlações sejam realizadas, a técnica escolhida foi a RBR (baseada em regras), por ser a mais intuitiva, do ponto de vista do administrador. Para representação das regras, foi escolhida a arquitetura de gerenciamento baseado em políticas do IETF. A linguagem de políticas será explicada na próxima seção.

4.2 A Linguagem de Políticas

Para a definição de políticas, uma linguagem de políticas deve ser usada. Algumas linguagens de políticas já foram propostas, tais como as já citadas: PDL (LOBO; BHATIA; NAQVI, 1999) e Ponder (DAMIANOU et al., 2001). Porém, nenhuma delas foi escolhida, devido ao fato de não ser necessário uma linguagem muito complexa ou extensa para validação da arquitetura. Determinou-se a utilização de uma linguagem hipotética[1], semelhante a linguagens de programação conhecidas, como C e PHP. Essa

[1] A linguagem de política definida para essa dissertação é chamada de hipotética por não ter uma definição formal. A linguagem é utilizada apenas na implementação da interface gráfica e as políticas são convertidas para XML, como explicado no capítulo 5.

linguagem hipotética segue o modelo eventos-condições-ações, onde os eventos são as *traps* SNMP, as condições são comparações com campos das *traps* ou com períodos de tempo da chegada das *traps*, e as ações são definições de valores para campos de eventos a serem gerados pela correlação (ex.: gerente destino, valor do campo *trapType*, etc.) ou atribuição de valores a variáveis específicas (ex.: *message*, *manager*, etc.). Essas variáveis específicas ficam a critério de quem implementa a arquitetura, dependendo de suas necessidades.

Na verdade, essa linguagem hipotética tem como principal objetivo facilitar a edição de políticas pelo administrador. Após a política ter sido criada ou editada, essa é armazenada no repositório de políticas global (explicado nas seções seguintes). Depois de armazenada, ela pode ser enviada a gerentes intermediários. Quando enviada a um gerente, a política é convertida para XML e uma chamada WS é feita para a transferência. A Figura 4.2 mostra um exemplo de uma política descrita na linguagem hipotética:

```
Política: 1
Nome: Web Server Reboot
Período: 120s

Regra: 1
Nome: WebServerReboot
if (trapType == coldStart or trapType == warmStart) and (hostName == Hubble or hostName == Noc)
    message = 'WARNING: Web Server rebooted!'

Regra: 2
Nome: ManagerDefinition
if (timeOfDay >= 10:00 and timeOfDay <= 14:00)
    manager = 'gerente@metropoa.tche.br'
else
    manager = 'gerente@labcom.inf.ufrgs.br'
```

Figura 4.2: Exemplo de política descrita na linguagem hipotética.

Como pode ser visto na figura, a política possui:

- **Um identificador**: possibilita identificar uma política de forma única. No exemplo, o identificador é 1;
- **Um nome**: mesmo havendo uma identificação numérica (única), uma identificação com o nome da política facilita o entendimento pelos administradores. No exemplo, o nome utilizado foi "Web Server Reboot";
- **Um período de tempo**: especifica, em segundos, em quanto tempo após a chegada da primeira *trap* deve haver a correlação. No exemplo, o tempo especificado foi 120 segundos;
- **Regras**: cada regra contém um identificador, um nome, condições e ações. No exemplo, os identificadores são 1 e 2, os nomes das regras são "WebServerReboot" e "ManagerDefinition". A regra 1 verifica se o tipo da trap é *coldStart* ou *warmStart* e se o nome do dispositivo gerenciado é "Hubble" ou "Noc". Se a condição for verdadeira, a variável "*message*" recebe o valor "WARNING: Web Server rebooted!". A regra 2 define o endereço de e-mail do gerente destino. No exemplo,

o gerente destino definido foi “gerente@metropoa.tche.br”, no caso do horário da correlação ficar entre entre 10h e 14h, e, em outro horário, o gerente definido foi “gerente@labcom.inf.ufrgs.br”. Cabe salientar que o horário utilizado na condição da regra 2 é o horário da chegada da primeira *trap* mais o período definido pela política. Ou seja, se a primeira *trap* chega às 13h59min, a correlação é realizada às 14h01min (esse último é o horário utilizado para aplicar as ações da política).

A forma de representar a linguagem ou as variáveis a serem utilizadas, pode ser definida conforme a implementação adotada. O importante é seguir a arquitetura proposta para solucionar o problema previamente relatado. Nas seções seguintes, são mostrados os componentes da arquitetura.

4.3 Gateways

Os gateways são responsáveis pela tradução das mensagens SNMP para chamadas WS. Como mencionado anteriormente, mensagens SNMP possuem muitos pontos fracos e os WS poderiam suprir muitos deles. Mas, como muitos dispositivos não suportam WS, *gateways* são necessários para efetuar a conversão SNMP/WS. A Figura 4.3 mostra os elementos que compõem um *gateway*.

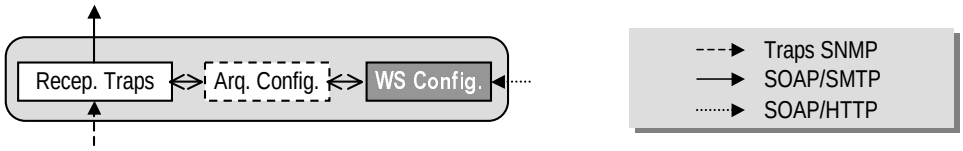

Figura 4.3: Elementos de um gateway.

A máquina que contém um *gateway* deve possuir um servidor Web e um servidor de e-mail instalados. O servidor Web deve estar instalado para que as configurações enviadas pelo Gerente Superior, através de chamadas WS, possam ser atendidas. O servidor de e-mail deve estar instalado para que mensagens SMTP possam ser enviadas a um Gerente Intermediário. O componente **WS Configuração** é o responsável por receber as configurações enviadas pelo GS e armazená-las no **Arquivo de Configuração**. O elemento **Receptor de Traps**, por sua vez, é o responsável por receber as *traps* SNMP enviadas pelos agentes, convertê-las em chamadas WS e enviá-las para um GI, previamente configurado no Arquivo de Configuração. O Arquivo de Configuração, opcionalmente, pode conter configurações como tipos de *traps* que devem ser descartadas, horários em que traps são descartadas sem a realização de conversão, etc.

4.4 Gerentes Intermediários

Os Gerentes Intermediários (GIs) são os responsáveis pela recepção de *traps*, correlação e envio para um gerente de nível superior. As *traps* recebidas já estão em formato XML, pois já passaram por algum *gateway*. Um GI pode receber *traps* de um GI de um nível inferior ou diretamente de um *gateway*, e envia a correlação para um GI de nível superior ou para o GS. A Figura 4.4 mostra os elementos que compõem um Gerente Intermediário.

Cada GI possui um elemento **WS Eventos**, que é responsável por receber os eventos, retirar os cabeçalhos que for necessário e enviá-los ao elemento **Controlador**. O

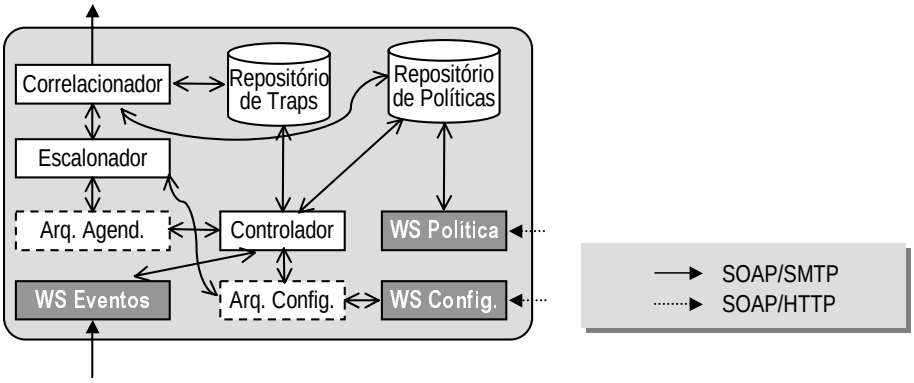

Figura 4.4: Elementos de um gerente intermediário.

Controlador trata a notificação recém recebida e procura, no **Repositório de Políticas** local, as políticas que poderiam tratar tal notificação. Todas as políticas relacionadas à notificação são avaliadas para que seja verificado se as ações associadas poderiam ser executadas. Se nenhuma política relacionada ao evento recebido é encontrada no repositório, então a notificação é encaminhada a um determinado GI pré-configurado (abordagem otimista) ou é descartada (abordagem pessimista). A seleção da abordagem a ser utilizada em cada GI é baseada na sua configuração interna (**Arquivo de Configuração**).

Geralmente, uma política recém avaliada necessita ser novamente avaliada pouco tempo depois, devido ao aspecto temporal descrito na política. Dessa forma, eventos temporais precisam ser notificados. Por exemplo, se uma regra de correlação de uma política determina que, se um evento X for seguido por um evento Y em menos de 30 segundos, a ação Z deve ser executada. Quando a notificação do evento X avaliar a política, deve haver uma maneira de reavaliar esta política depois de 30 segundos, caso nenhuma notificação de evento Y for recebida. Ou seja, deve haver um evento-relógio notificando o *timeout* de 30 segundos. Esses eventos relacionados com tempo são utilizados na arquitetura proposta através do agendamento de política. O **Arquivo de Agendamento** armazena informações de agendamento, e o seu conteúdo é gerenciado pelo elemento Controlador.

Uma vez que a notificação é recebida e associada a uma política, essa notificação é armazenada no **Repositório de Traps**, o qual possui uma tabela de *traps* e as políticas associadas (que estão sob avaliação). Caso não haja nenhuma *trap* no repositório, associada com a mesma política, o Controlador insere no Arquivo de Agendamento o horário que deve ser feita a correlação (ex.: 30 segundos após o horário corrente). Se não for a primeira, a *trap* é simplesmente armazenada. O elemento **Escalonador** verifica o Arquivo de Agendamento em determinados períodos de tempo (o intervalo entre as verificações deve estar definido no Arquivo de Configuração) e compara os horários marcados para realização de correlações com o horário corrente. Quando for horário para realização de uma correlação, o elemento **Correlacionador** realiza a correlação. Dessa forma, as ações da política são executadas e todas *traps* armazenadas, relacionadas à política avaliada, são removidas do Repositório de *Traps*. A Figura 4.5 mostra uma linha de tempo, onde o "relógio" é zerado sempre que uma *trap* é recebida e não há *traps* no repositório associadas à mesma política. Seguindo o exemplo anterior, após 30 segundos a correlação é realizada. Como pode ser visto na figura, foram realizadas três correlações, a primeira com três eventos, a segunda com cinco e a terceira com três. Essa linha de

tempo representa as correlações de apenas uma política, portanto, pode-se imaginar que existem N linhas de tempo, onde N é o número de políticas armazenadas no repositório de políticas local.

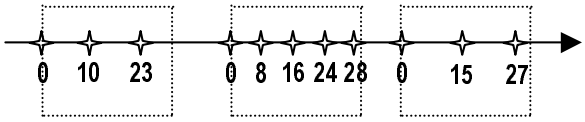

Figura 4.5: Linha de tempo mostrando correlações de eventos.

O elemento **WS Política** armazena políticas recém recebidas no Repositório de Políticas local. O elemento **WS Configuração** recebe as configurações enviadas pelo GS e as armazena no Arquivo de Configuração, para que os elementos Controlador, WS Eventos e Escalonador possam consultá-lo. As correlações realizadas pelo Correlacionador são enviadas a um gerente de nível superior através de uma mensagem SOAP/SMTP. O servidor SMTP é novamente usado, só que, dessa vez, para enviar uma mensagem.

Tipicamente, políticas de rede não abrangem apenas aspectos de rede. Como mencionado anteriormente, elas abrangem, também, aspectos temporais. Isso significa que políticas podem ser agendadas para serem ativadas e, então, avaliadas em períodos de tempo específicos. Dessa forma, é possível que os eventos correlacionados nos GIs sejam encaminhados a diferentes gerentes, dependendo do dia da semana e horário correntes. Um exemplo dessa abordagem (Figura 4.6) é quando um gerente A envia eventos correlacionados ao gerente B das 7h01min às 8h e envia ao gerente C das 8h01min às 9h. Assim, a abordagem de correlação baseada em políticas tem a capacidade de definir árvores de correlação dinâmicas, sendo que as árvores podem mudar seu formato no decorrer do tempo, devido às condições temporais.

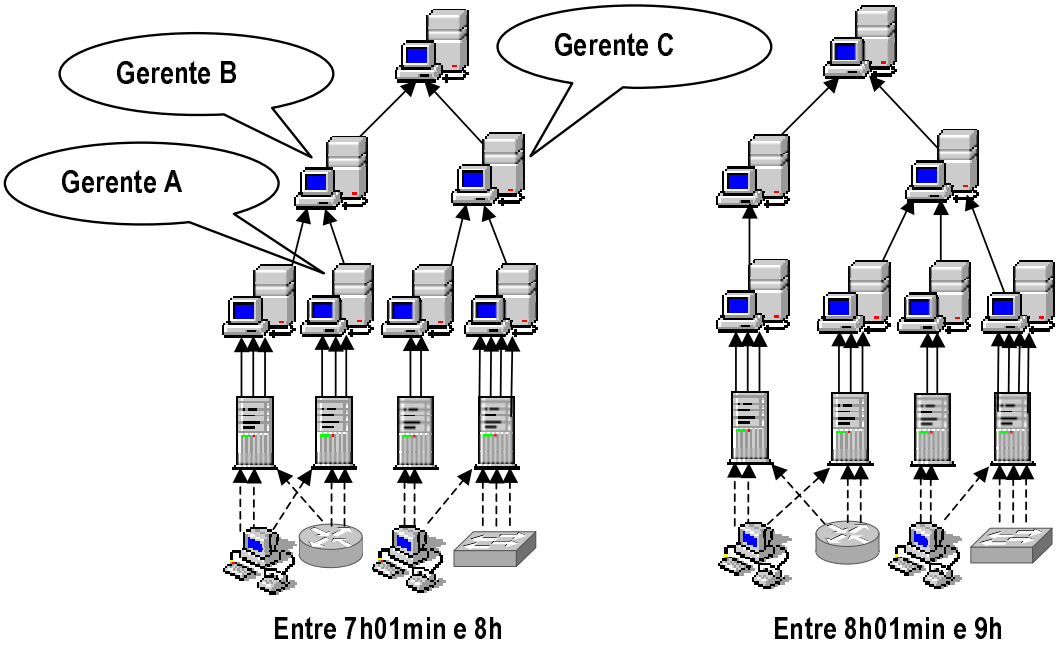

Figura 4.6: Exemplo de árvores de correlação dinâmicas.

4.5 Gerente Superior

O componente raiz da árvore de correlação é o Gerente Superior (GS). Esse é responsável pela configuração dos demais gerentes e dos *gateways*. Assim como os GIs, o GS também possui o elemento **WS Configuração** e um **Repositório de Traps**, ambos com as mesmas finalidades descritas na seção anterior. Como pode ser visto na Figura 4.7, o GS possui menos componentes que o GI, devido ao fato de que o GS não realiza correlações. O GS simplesmente recebe os eventos já correlacionados e efetua ações de configuração. As ações de configuração são enviadas com o uso de WS, utilizando como transporte o protocolo HTTP. O HTTP foi escolhido devido à natureza dessa operação, ou seja, um administrador precisa saber se as ações de configuração são realizadas com sucesso no mesmo instante e não alguns minutos depois.

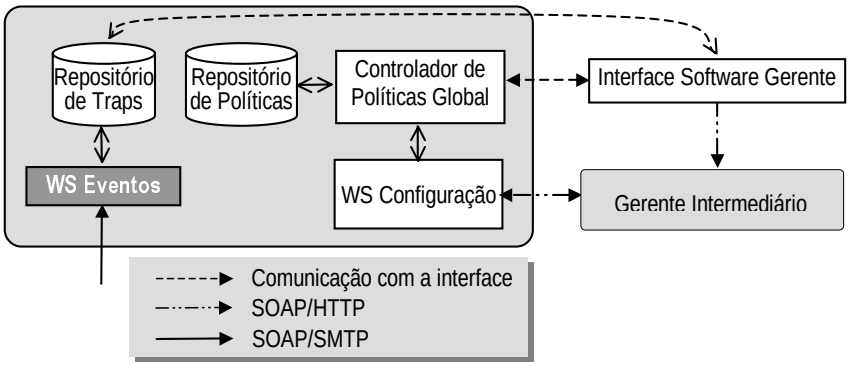

Figura 4.7: Elementos do gerente superior.

Comparando com o GI, há também um **Repositório de Políticas** no GS, porém este se diferencia do encontrado no GI pois, ao invés de conter políticas a serem usadas no GS, este repositório contém as políticas a serem distribuídas em todos GIs. O controle de inclusão, alteração e exclusão de políticas no repositório é feito pelo **Controlador de Políticas Global**. Esse elemento também controla a manipulação das políticas nos GIs, mas para isso ele utiliza o elemento **WS Configuração**, o qual efetivamente realiza a comunicação com os GIs. Para a comunicação do administrador do sistema com os gerentes, há o elemento **Interface Software Gerente**. Com esse elemento (interface), é possível a comunicação entre as solicitações do administrador e o Controlador de Políticas Global. Cabe também à interface do *software* gerente a configuração de GIs (ex.: definição de gerente destino padrão). Com a interface, é possível a verificação das *traps* já correlacionadas, recebidas pelo GS, para que seja possível analisar que tipos de falhas podem estar ocorrendo nos dispositivos ou na rede. A interface pode ser gráfica ou um simples arquivo em formato texto; quem a define é o desenvolvedor.

Seguindo a arquitetura proposta, diversos tipos de implementação podem ser realizados. No capítulo seguinte, a implementação do protótipo é mostrada para que a avaliação possa ser realizada.

5 IMPLEMENTAÇÃO DO PROTÓTIPO

Para tornar possível a avaliação da arquitetura proposta, foi realizada a implementação do protótipo. O protótipo possui todos os elementos mostrados na arquitetura, com exceção dos agentes. Tanto os *gateways* como os gerentes foram implementados em ambiente Linux. Nos dispositivos gerenciados, a única tarefa realizada foi a configuração do endereço IP para onde devem ser enviadas as *traps* SNMP. Nas seções seguintes, serão expostas as tecnologias utilizadas na implementação, a forma como foi realizada a implementação dos *gateways*, gerentes intermediários e gerente superior e, por fim, suas limitações.

5.1 Tecnologias Utilizadas

Nos dispositivos gerenciados, foram utilizados agentes SNMP, configurados para enviarem *traps* para endereços IP onde estavam instalados os *gateways*. Nas máquinas onde estavam instalados os *gateways*, foi instalado o pacote de aplicações NET-SNMP (SOURCEFORGE.NET, 2004), que possui, entre outras, aplicações para receber, tratar e gerar *traps* SNMP. Para os *gateways*, em específico, foi utilizada a aplicação *snmptrapd*, que tem como função receber traps SNMP na porta UDP 162 e, de acordo com as configurações pré-definidas, filtrá-las e tratá-las.

Os *scripts* para os *gateways* e gerentes foram implementados em PHP (GROUP, 2004a). Para a chamada de Web Services, foi utilizada a biblioteca PEAR:SOAP (GROUP, 2004b). Existem outras bibliotecas que permitem a utilização de Web Services em *scripts* PHP (ex.: nuSOAP), porém, dentre as que foram pesquisadas, apenas a PEAR:SOAP permite que, além do HTTP, o SMTP também seja utilizado como protocolo de transporte.

Para o envio das *traps* via e-mail, foi utilizado o servidor de e-mail *sendmail* (SENDMAIL, 2005) e para o envio de políticas ou configurações, foi instalado o servidor Web Apache (APACHE, 2004), para que a comunicação Web Service baseada em SOAP/HTTP pudesse ser realizada.

Os repositórios de políticas e de *traps* foram implementados através do gerenciador de banco de dados MySQL (MYSQL, 2004) e as interfaces produzidas para interagirem com o gerente superior foram implementadas em HTML.

5.2 Implementação dos Gateways

Para a recepção das *traps* SNMP, a aplicação *snmptrapd* foi executada de acordo com a Figura 5.1.

```
snmptrapd -c /etc/snmp/snmpd.conf
-F "
<Trap>
        <localDate>%04.4y/%02.2m/%02.2l</localDate>
        <localTime>%02.2h:%02.2j:%02.2k</localTime>
        <sysUpDate>%04.4Y/%02.2M/%02.2L</sysUpDate>
        <sysUpTime>%02.2H:%02.2J:%02.2K</sysUpTime>
        <agentAddr>%a</agentAddr>
        <hostName>%A</hostName>
        <PDUSourceAddress>%b</PDUSourceAddress>
        <PDUSourceHostname>%B</PDUSourceHostname>
        <enterpriseString>%N</enterpriseString>
        <trapType>%w</trapType>
        <trapDescription>%W</trapDescription>
        <trapSubType>%q</trapSubType>
        <variableBindings>%v</variableBindings>
</Trap> \n"
-Lf /gw/traps.txt
```

Figura 5.1: Linha de comando utilizada para ativar a aplicação *snmptrapd*.

Como pode ser visto na figura, as opções utilizadas foram “-c”, “-F” e “-Lf”.

A opção “-c” indica qual o arquivo de configuração a ser utilizado que, no caso apresentado, foi o “/etc/snmp/snmpd.conf” . A opção “-F” indica o formato a ser montada a *trap* recebida que, no caso apresentado, foi o formato XML, sendo a *tag* raíz <Trap> e as demais *tags* o nome do campo da *trap*. Os valores apresentados entre as *tags* são os valores de cada campo da *trap*. Para as datas, foi utilizado o formato ano/mês/dia e para os horários foi utilizado o formato hora:minuto:segundo. Os demais campos são *strings* ou valores numéricos. Por fim, a opção “-Lf” define o arquivo onde as *traps* com o formato definido por “-F” devem ser armazenadas que, no caso apresentado, foi o “/gw/traps.txt”.

O arquivo de configuração permaneceu com a configuração padrão, com a adição da seguinte linha: “traphandle default /usr/bin/php /gw/gw.php”. A diretiva “traphandle” indica que deve haver o tratamento das *traps* recebidas. O parâmetro “default"” indica que todas traps devem ser tratadas e a parte que vem após “default” indica que programa ou *script* deve ser acionado. No caso apresentado, o *script* “/gw/gw.php” é acionado.

O *script* “/gw/gw.php” lê a última linha do arquivo “/gw/traps.txt” (coloca o conteúdo na variável $trap), preenche os parâmetros necessários para a chamada do Web Service e realiza a chamada. O trecho de código que realiza o preenchimento dos parâmetros e a chamada do Web Service é mostrado na Figura 5.2.

Referente à figura, pode-se observar, como opções, que o gerente destino é “gerente@noc.metropoa.tche.br”. Um *namespace* é definido, denominado “urn:SOAP_SMTP_Server”. O *namespace* serve como referência no gerente, para saber onde (em que classe) está a função chamada. O *gateway* remetente é “gw@labcom.inf.ufrgs.br”, o servidor de e-mail está na máquina local (“localhost”) e a porta utilizada é a 25 (SMTP). Como parâmetro, é definido o conteúdo da *trap* (variável $trap previamente preenchida). Na chamada WS, é realizada a chamada à função “InsertTrap” com parâmetro e opções previamente preenchidos.

```
$soapclient = new SOAP_Client('mailto: gerente@noc.metropoa.tche.br');

$options = array(
      'namespace' => 'urn:SOAP_SMTP_Server',
      'from' => 'gw@labcom.inf.ufrgs.br',
      'host' => 'localhost',
      'port' => 25);

$params = array('Conteudo' => $trap);

$return = $soapclient->call('insertTrap',
       $params,
       $options);
```

Figura 5.2: Trecho do script gw.php, responsável pelo preenchimento de parâmetros e chamada Web Service.

5.3 Implementação do Gerente Intermediário (GI)

Por conter mais elementos que os outros componentes da arquitetura, a implementação do GI é a mais complexa. Para tornar o entendimento mais simples, a Figura 5.3 mostra novamente a estrutura do GI, com números entre parênteses em cada elemento ou ação realizada. No decorrer da explicação, referências a esses números serão feitas. Dessa forma, ficará mais fácil de associar a arquitetura exposta no capítulo 4 com a implementação do protótipo.

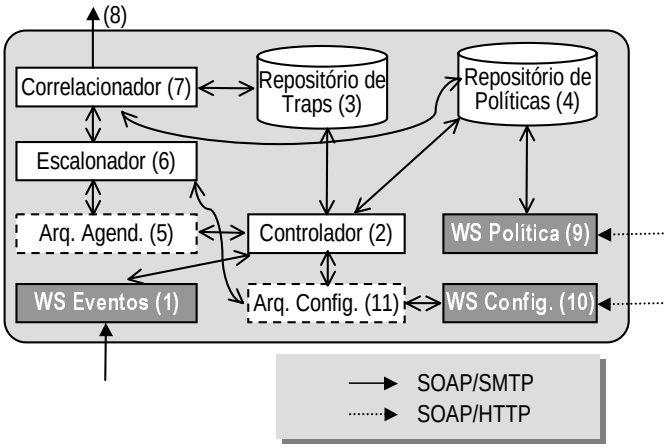

Figura 5.3: Seqüência de passos realizados pelo protótipo.

Na máquina onde há um GI instalado, deve haver um *alias* de e-mail configurado para realizar a chamada ao *script* do gerente. O arquivo que contém a lista de *aliases* é o "/etc/aliases" e a linha responsável pela chamada do *script* gerente é: gerente | "/usr/bin/php /gw/gerente.php", onde "gerente" é o *alias* e o conteúdo entre aspas é a chamada ao *script* "gerente.php". Quando um e-mail é destinado ao *alias* gerente na máquina em questão, o *script* gerente.php (1) é chamado e o conteúdo do e-mail fica armazenado na entrada padrão (*stdin*). Esse *script* lê na entrada padrão o conteúdo do e-mail e armazena em uma variável denominada $email. Uma classe denominada SOAP_SMTP_Server e um *namespace* com mesmo nome são criados e a variável $email

é submetida à classe através de uma chamada de serviço. Essa chamada de serviço retira os cabeçalhos (que são criados para tornar a comunicação possível), verifica qual método deve ser chamado (que no caso apresentado é "insertTrap") e faz com que esse método receba como entrada apenas o conteúdo enviado pelo *gateway* (apresentada na Figura 5.2 como $trap).

O método insertTrap recebe como entrada o conteúdo da variável enviada pelo *gateway* ($trap), agora denominada $xmlTrap. Um *parser* (2) é criado e a variável $xmlTrap é submetida a ele. O *parser* tem como função armazenar campos e valores das *traps* na medida em que lê cada campo no formato XML. Esse armazenamento é realizado no repositório de *traps* (3) (a tabela 5.1 mostra os campos do repositório).

Tabela 5.1: Campos do repositório de traps.

Campo	Tipo	Comentário
id	inteiro	identificador da *trap*
politica	inteiro	identificador da política
variavel	texto	nome do campo da *trap*
valor	texto	valor do campo da *trap*

Todos os registros de campos de uma mesma *trap* são armazenados com o mesmo identificador (id). O id serve para distinguir as *traps* e é gerado no início do *parser*, de acordo com os id's já armazenados no repositório (o campo politica recebe valor 0, pois ainda não se sabe qual política será usada para essa *trap*). Para cada campo armazenado, uma verificação é realizada no repositório de políticas (4) para saber se há uma condição que satisfaça o campo e o valor analisados no momento. Se houver alguma condição satisfeita, a política que contém essa condição é considerada uma política "candidata". Todos id's das políticas candidatas são colocados em um vetor. Após o término da verificação realizada pelo *parser*, um confronto de cada política candidata com a *trap* é realizado. Se houver alguma candidata com todas condições satisfazendo a *trap*, essa política é eleita e seu id é armazenado no repositório de *traps*, no campo politica. Cabe salientar que, se houver mais de uma candidata que satisfaça a *trap*, a primeira verificada é a escolhida. Esse confronto só pode ser realizado após o término da verificação feita pelo *parser*, porque só nesse momento as condições que devem satisfazer a *trap* estão montadas (uma consulta SQL é montada de acordo com cada passo realizado pelo *parser*). Se houver uma política escolhida, o id da política é gravado nos campos da *trap* em questão; caso contrário, a *trap* é enviada ao próximo gerente ou descartada, dependendo da abordagem utilizada (11).

Caso uma política seja escolhida, antes de gravar o id da política nos campos da *trap*, uma consulta é realizada ao repositório de *traps* (3) para verificar se há alguma *trap* com essa mesma política. Se não houver, deve ser iniciado um intervalo de correlação para essa política. Para isso, uma consulta ao intervalo definido para a política é realizada (4). Com o intervalo, é possível definir o horário que deve ser realizada a correlação (horário atual + intervalo). O id da política e o horário de correlação são armazenados na tabela Escalonador (5) (mostrada na tabela 5.2).

Para realizar a correlação, o *script* escalonador.php (6) executa um laço infinito, onde a cada "X" segundos é realizada uma consulta na tabela Escalonador, para saber se há alguma correlação a ser realizada com horário menor que o horário atual. O intervalo "X" é definido pelo GS (11). Se houver alguma correlação a ser realizada, esse registro é

Tabela 5.2: Tabela Escalonador.

Campo	Tipo	Comentário
horario	inteiro	horário (em segundos) a ser realizada a correlação
politica	inteiro	identificador da política

removido da tabela Escalonador e a função de correlação (7) é chamada, passando como parâmetro o id da política. A função de correlação verifica na tabela Acao (4) (tabela 5.3), se há definição de gerente destino para a política em questão. Se houver, verifica nas tabelas Condicao e Condicoes (4), quais condições temporais resultam na definição do gerente destino. Dessa forma, aplica-se as condições temporais comparando com o horário e data correntes e, então, o gerente é definido. Um e-mail contendo a correlação é montado, os registros das *traps* correlacionadas são excluídos (3) e a correlação é enviada ao próximo gerente (8).

Tabela 5.3: Tabela Acao.

Campo	Tipo	Comentário
id	inteiro	identificador da ação
politica	inteiro	identificador da política
regra	inteiro	identificador da regra
variavel	texto	nome da variável
valor	texto	valor da variável

O repositório de políticas (4) é composto por cinco tabelas: Politica (tabela 5.4), Regra (tabela 5.5), Acao, Condicoes (tabela 5.6) e Condicao (tabela 5.7).

Tabela 5.4: Tabela Politica.

Campo	Tipo	Comentário
id	inteiro	identificador da política
nome	texto	nome da política
intervalo	inteiro	intervalo de correlação desta política

Tabela 5.5: Tabela Regra.

Campo	Tipo	Comentário
id	inteiro	identificador da regra
politica	inteiro	identificador da política
nome	texto	nome da regra

A relação entre as cinco tabelas do repositório de políticas (4) é mostrada na Figura 5.4.

O *script* receptorPolitica.php (9), localizado em um diretório onde o servidor Web tem acesso, recebe a política enviada pelo gerente superior. A política é recebida em um

Tabela 5.6: Tabela Condicoes.

Campo	Tipo	Comentário
id	inteiro	identificador das condições
politica	inteiro	identificador da política
regra	inteiro	identificador da regra
cond1	inteiro	identificador da condição 1
operador	texto	operador lógico (AND, OR)
cond2	inteiro	identificador da condição 2
interno	texto	interno separa condições dentro de parênteses, externo separa parênteses

Tabela 5.7: Tabela Condicao.

Campo	Tipo	Comentário
id	inteiro	identificador da condição
politica	inteiro	identificador da política
regra	inteiro	identificador da regra
variavel	texto	nome da variável
valor	texto	valor da variável
operador	texto	operador comparativo (=, >=, <=)

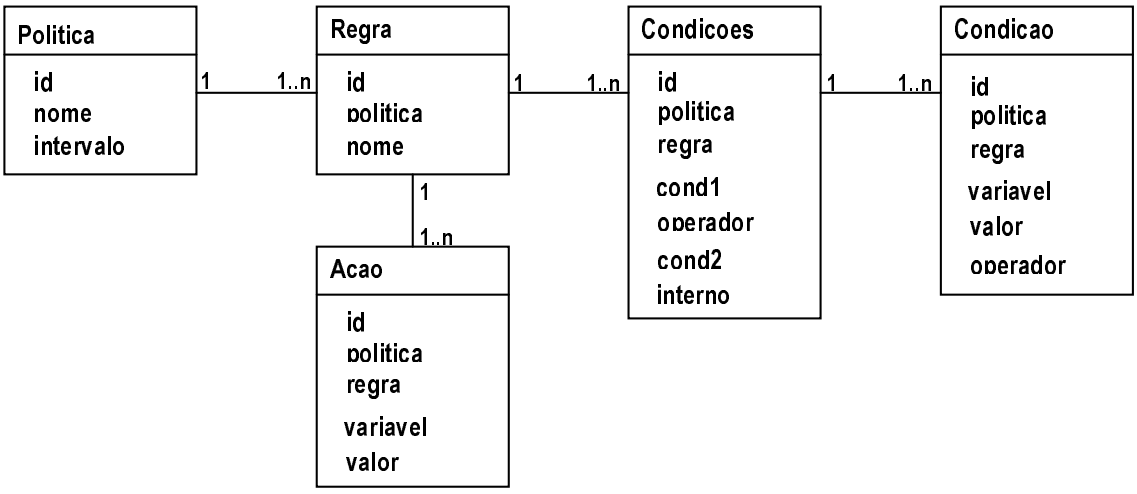

Figura 5.4: Relação entre as tabelas do repositório de políticas.

formato XML pré-definido (Figura 5.5). No *script*, há um *parser* que lê a política e, de acordo com as tags lidas, armazena o(s) registro(s) na tabela adequada do repositório de políticas. É essencial que a política seja enviada exatamente como definido na figura 5.5, para que o *parser* seja capaz de interpretar e armazenar adequadamente cada registro.

Como pode ser visto na figura, a versão do XML utilizado é a 1.0. A *tag* raiz é <Policy>. A tag <ID>, quando utilizada após <Policy>, indica o identificador da política e, quando utilizada após <Rule>, indica o identificador da regra. O mesmo acontece com a tag <Name>, que indica o nome da política ou da regra. <Interval> indica o intervalo em que *traps* que satisfazem determinada política devem ser correlacionadas. <Rule> indica uma nova regra. <Conditions> indica o começo

```
<?xml version="1.0"?>
<Policy>
   <ID>4</ID>
   <Name>Hubble Partial Reboot</Name>
   <Interval>60</Interval>
   <Rule>
      <ID>1</ID>
      <Name>Reboot</Name>
      <Conditions>
         <Group>
            <trapType>M=0</trapType>
            <operator>and</operator>
            <trapType>m=2</trapType>
         </Group>
         <operator>and</operator>
         <Group>
            <hostName>==143.54.47.81</hostName>
         </Group>
      </Conditions>
      <Actions>
         <trapType>100</trapType>
         <hostName>Hubble</hostName>
      </Actions>
   </Rule>
   <Rule>
      <ID>2</ID>
      <Name>ManagerDefinition</Name>
      <Conditions>
         <Group>
            <localTime>M=13:00:00</localTime>
            <operator>and</operator>
            <localTime>m=23:00:00</localTime>
         </Group>
         <operator>and</operator>
         <Group>
            <localDate>M=2005/01/24</localDate>
            <operator>and</operator>
            <localDate>m=2005/01/29</localDate>
         </Group>
      </Conditions>
      <Actions>
         <manager>gerente@noc.metropoa.tche.br</manager>
         <else>gerente@labcom.inf.ufrgs.br</else>
      </Actions>
   </Rule>
</Policy>
```

Figura 5.5: Formato XML de uma política.

das condições para a regra atual. <Group> indica um início de um grupo dentro das condições, ou seja, as condições que ficam dentro de parênteses. Dentro de <Group>, tem-se as *tags* contendo os mesmos nomes de campos de *traps*, e a *tag* <operator> indica um operador lógico (OR ou AND) interno (dentro de parênteses). Quando a *tag* <operator> é utilizada entre o fim de um e o início de outro <Group>, há a indicação de um operador lógico externo (fora de parênteses).

A *tag* <Actions> indica o início das ações. As ações consistem em definições de valores para campos de *traps* (ex.: trapType) ou definições de valores para variáveis (ex.: gerente-destino, mensagem de alerta, etc.). Os operadores de comparação utilizados são: == (igual), M= (maior ou igual) ou m= (menor ou igual). O caractere "M" substitui o sinal de maior e o caractere "m" substitui o sinal de menor para não haver conflito com os caracteres utilizados para abrir e fechar uma *tag*. A conversão para o sinal adequado é feita pelo *parser*. Os sinais > (maior) e < (menor) não são utilizados, pois pode-se representar os mesmos intervalos com >= e <=, e, com a utilização de menos sinais, o tamanho do *parser* fica menor.

O *script* config.php (10), também localizado em um diretório onde o servidor Web tem acesso, recebe configurações enviadas pelo gerente superior. As configurações são armazenadas na tabela Configuracao (11) (tabela 5.8)

Tabela 5.8: Tabela Configuracao.

Campo	Tipo	Comentário
gerente	texto	e-mail do gerente destino padrão
abordagem	texto	quando não houver correlação, envia direto (otimista) ou descarta (pessimista)
correlacao	inteiro	intervalo de tempo, em segundos, utilizado pelo escalonador

5.4 Implementação do Gerente Superior (GS)

Inicialmente, o GS foi implementado apenas com a utilização de *scripts* PHP, sem interface gráfica. A execução dos *scripts* era realizada por linha de comando e os arquivos contendo informações (em formato XML) eram utilizados para armazenar políticas ou configurações a serem enviadas aos GIs. Essa foi a maneira mais simples de implementar, porém, a utilização do protótipo estava ficando cada vez mais difícil e massante. Para tornar a utilização do GS mais simples, foi utilizada a linguagem HTML para criação de uma interface gráfica. O PHP continuou sendo utilizado, mas, ao invés de ler de arquivos, foi implementada uma conversão do que é digitado pelo usuário, para o formato XML. Depois de realizada a conversão, os dados são enviados através de WS baseados em SMTP, como descrito anteriormente.

O GS do protótipo possui um menu com os seguintes itens: Políticas (subitens: Inclusão, Consulta, Exclusão e Envio a GIs), Configuração e Traps Correlacionadas. O subitem Inclusão (Figura 5.6) possui os seguintes campos para preenchimento:

- **Nome Política**: nome da política a ser incluída no repositório de políticas global. Deve conter no máximo 50 caracteres.

- **Intervalo**: tempo (em segundos) que a política em questão deve aguardar desde a primeira *trap* recebida até a correlação. Entende-se como primeira *trap* recebida aquela que utiliza determinada política e, quando chega ao GI, não há nenhuma outra *trap* no repositório que utiliza essa mesma política.

- **Regra 1**: nome da Regra 1. Deve conter no máximo 50 caracteres.

- **Condições da Regra 1**: há três conjuntos de regras (cada conjunto entre parênteses). Em cada conjunto, há duas possibilidades de valor e um operador lógico entre eles (AND ou OR), e entre cada conjunto há um operador lógico. Não é necessário que todas condições sejam preenchidas.
- **Ações da Regra 1**: há quatro ações possíveis. Para cada ação, uma variável deve ser escolhida e o valor correspondente deve ser digitado.
- **Regra 2**: nome da Regra 2. Deve conter no máximo 50 caracteres.
- **Condições da Regra 2**: há dois conjuntos de regras (cada conjunto entre parênteses). Em cada conjunto, há duas possibilidades de valor e um operador lógico entre eles (AND ou OR), e entre os dois conjuntos há um operador lógico. Não é necessário que todas condições sejam preenchidas. As condições da regra 2 são as temporais (data e hora).
- **Ações da Regra 2**: há três ações possíveis. Para cada ação, uma variável deve ser escolhida e o valor correspondente deve ser digitado.

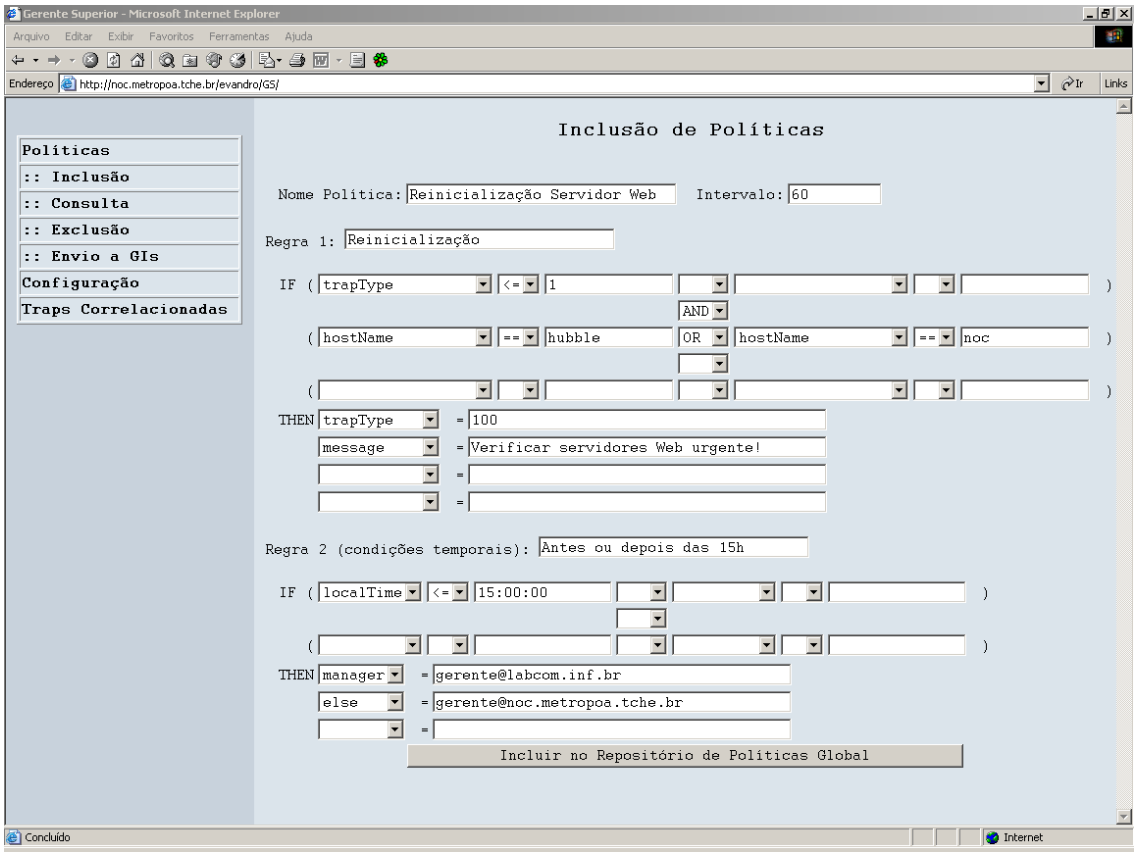

Figura 5.6: Inclusão de Políticas no Repositório Global.

No exemplo da Figura 5.6, foi criada uma política que alerta o administrador quanto à reinicialização de um dos dois servidores *Web*. Os tipos de *trap* que sinalizam essa situação são: 0 (*coldStart*) e 1 (*warmStart*). A *coldStart* indica uma reinicialização através de linha de comando, falha de energia elétrica ou travamento do equipamento, sendo que a configuração do agente ou a implementação pode ter sido alterada. A *warmStart*

indica uma reinicialização sem alteração da configuração do agente ou implementação. Os servidores Web possuem os nomes “hubble” e “noc”. Como ações da primeira regra, tem-se a geração de um tipo de *trap* 100 (tipo hipotético que pode ser utilizado em uma regra de uma política em um gerente superior, por exemplo) e de uma mensagem “Verificar servidores Web urgente!”. Na condição temporal, tem-se a ação de enviar a correlação para o gerente “gerente@labcom.inf.br” até as 15h ou para o gerente “gerente@noc.metropoa.tche.br” após às 15h.

O subitem Consulta possibilita a consulta de uma política com todos seus dados (Figura 5.7), ou uma listagem de todas políticas (Figura 5.8). Quando for escolhida a consulta de apenas uma política, pode-se consultar pelo identificador (id) ou pelo nome.

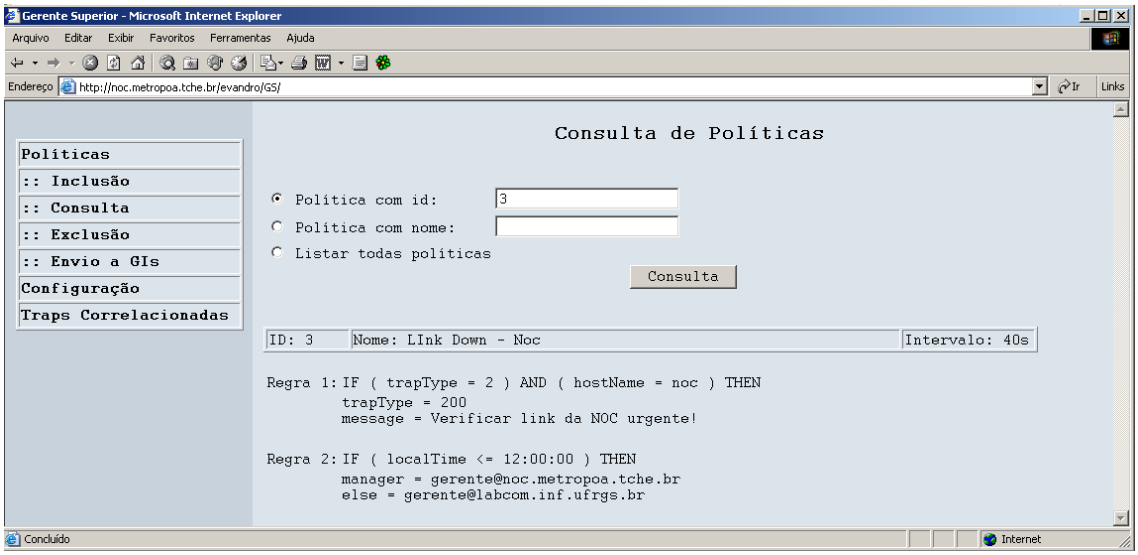

Figura 5.7: Consulta de Políticas por ID ou Nome.

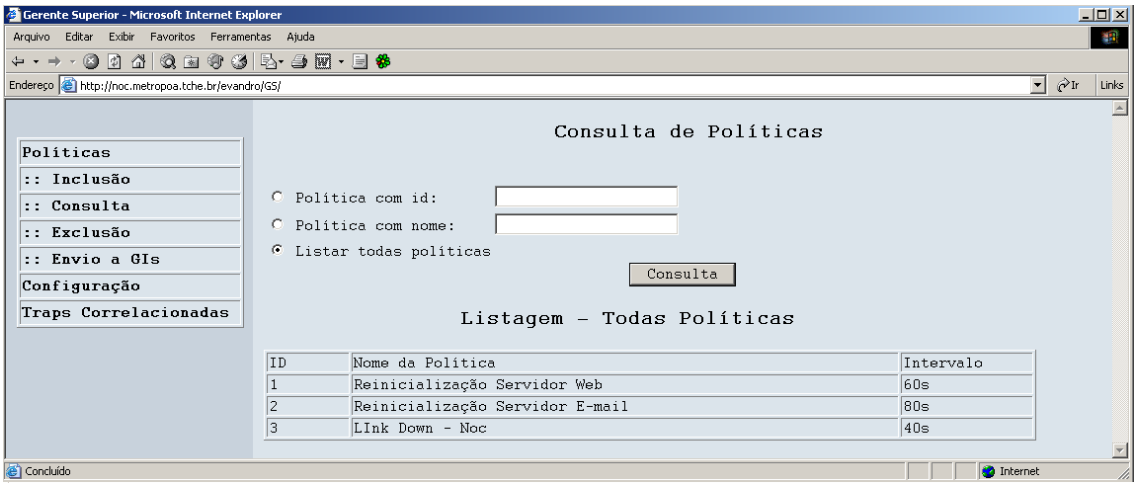

Figura 5.8: Listagem de todas políticas cadastradas.

O subitem Exclusão possibilita a exclusão de uma política no repositório global de políticas (Figura 5.9). A exclusão pode ser feita através do id ou do nome da política.

O subitem Envio a GIs possibilita o envio de políticas, previamente cadastradas no repositório global, a um gerente intermediário. A Figura 5.10 mostra o envio da política com id 1 ao GI “Labcom”.

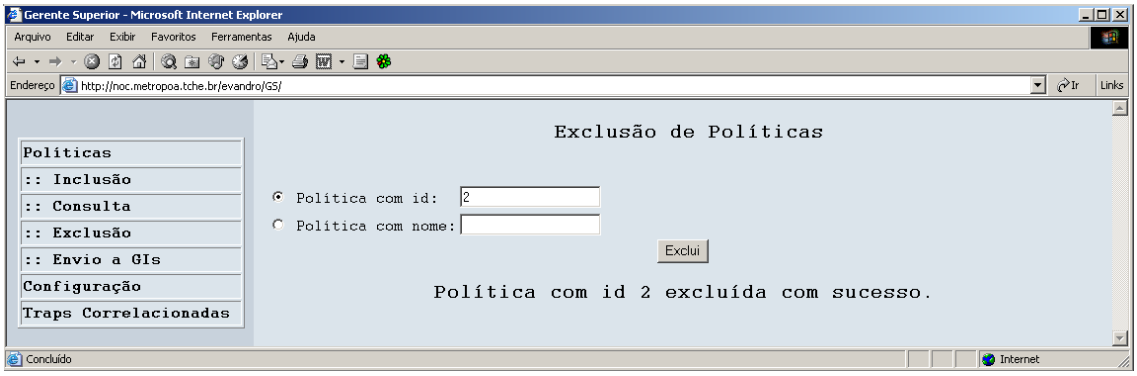

Figura 5.9: Exclusão de Políticas.

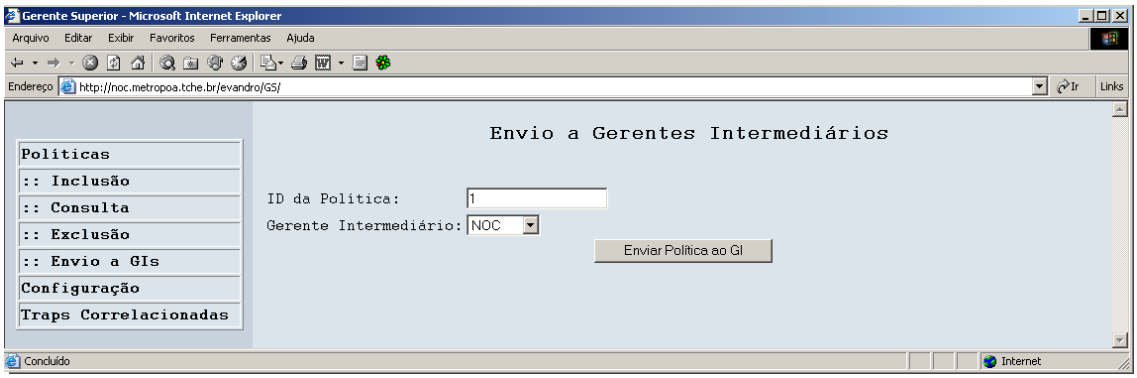

Figura 5.10: Envio de Políticas a um Gerente Intermediário.

O item Configuração permite que o administrador configure, para cada GI,:

- o e-mail do GI destino padrão, para o caso de uma política não especificar o GI destino;
- a abordagem da correlação (otimista ou pessimista, conforme já explicado);
- o intervalo do escalonador, que define de quanto em quanto tempo deve haver uma verificação na base de dados em busca de uma nova correlação. As opções são 10, 20, 30, 40, 50 ou 60 segundos.

Na Figura 5.11 é mostrado um exemplo de configuração, a ser enviado para o GI "NOC". O e-mail para onde o GI "NOC" deve enviar as correlações, no caso da política não definir um gerente destino, é "labcom@inf.ufrgs.br". A abordagem escolhida foi a "otimista", ou seja, no caso de não haver nenhuma política que satisfaça os campos da *trap* recebida, essa deve ser enviada ao GI padrão, sem realizar correlação. O intervalo escolhido para o escalonador foi de 30 segundos e, assim, a cada 30 segundos o escalonador verifica se há alguma correlação pendente a ser realizada.

O item Traps Correlacionadas mostra ao administrador quatro possibilidades para visualizar as *traps* correlacionadas, de acordo com o tempo. As possibilidades são nos últimos cinco, dez, trinta ou sessenta minutos (Figura 5.12).

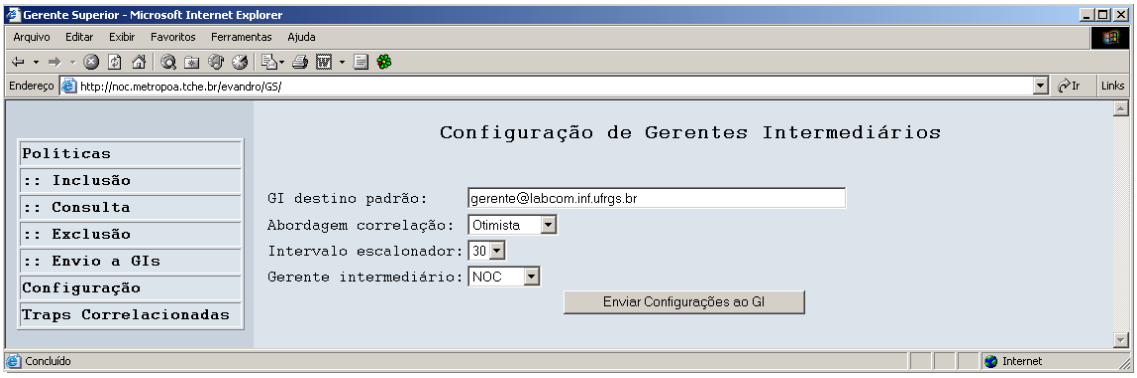

Figura 5.11: Configuração de Gerentes Intermediários.

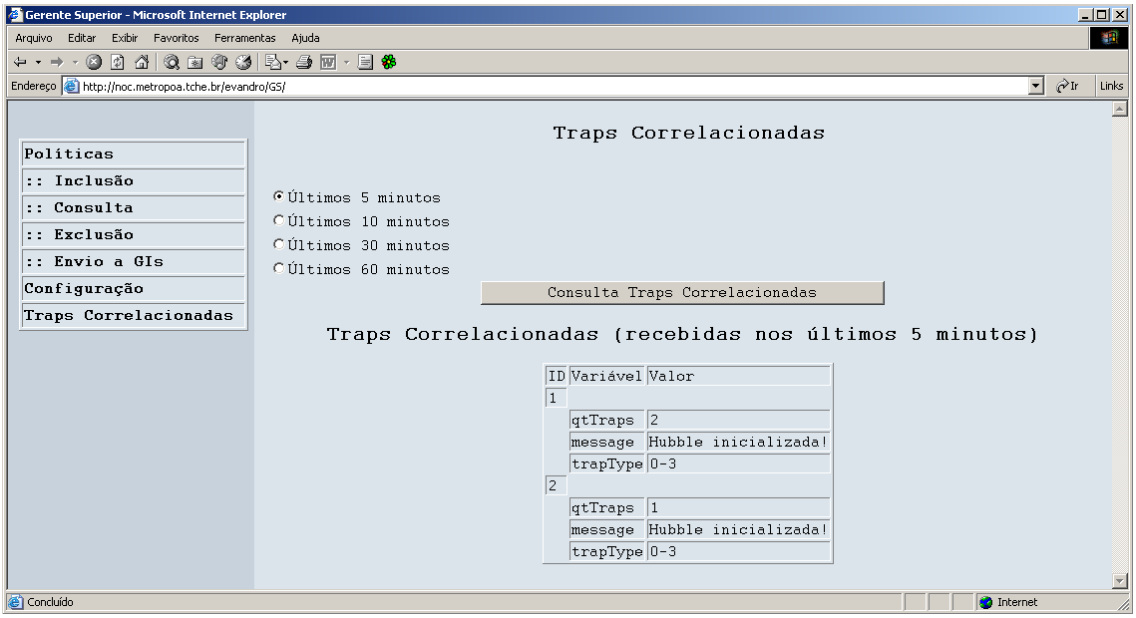

Figura 5.12: Traps Correlacionadas.

5.5 Limitações

Como mencionado anteriormente, seguindo a arquitetura proposta, diversos tipos de implementações podem ser realizadas. O protótipo mostrado possui algumas limitações, listadas abaixo, as quais não impediram a validação da arquitetura.

- Na inclusão de uma política, pode ser escolhido um ou dois campos dentro dos parênteses. Sendo escolhidos dois campos, esses devem ser o mesmo, para efeito de comparação de um determinado intervalo (campo1 >= valor1 AND campo1 <= valor2) ou a utilização do operador comparativo OR (campo1 == valor1 OR campo1 == campo2);
- São utilizados apenas dois operadores lógicos: AND e OR;
- São utilizados apenas três operadores comparativos: ==, >= e <=;
- O campo variableBindings não é utilizado pois, dependendo da MIB utilizada pelo agente, diferentes variáveis podem ser utilizadas, o que tornaria a implementação demorada;

- Uma política pode conter duas regras, a primeira com condições não temporais e a segunda com condições temporais, ou seja, a primeira não contém os campos localTime e localDate e a segunda contém apenas esses dois campos;
- O intervalo de tempo para cada verificação do escalonador pode ser de 10, 20, 30, 40, 50 ou 60 segundos.

Diversos testes foram aplicados para validação da arquitetura e, com eles, foi possível realizar medidas de tempo e tráfego na rede, consumidos por *traps* (SNMP e chamadas WS) e ações de configuração (WS utilizando HTTP ou SMTP como protocolo de transporte). Essas medidas são mostradas no próximo capítulo.

6 AVALIAÇÃO E RESULTADOS

Após o término da implementação do protótipo, testes foram realizados. No momento em que a implementação se encontrou estável, políticas foram criadas e enviadas a GIs, e *traps* foram geradas e enviadas aos *gateways*, para verificar-se o funcionamento da arquitetura. As *traps* foram geradas através do Serviço SNMP do sistema operacional Windows 2000 Server e da aplicação *snmptrap*, contida no pacote NET-SNMP (sistema operacional Linux).

Para verificar se as políticas estavam sendo avaliadas de forma correta, foram geradas *traps* de acordo com as políticas armazenadas previamente, testando diversas situações possíveis, como *traps* que poderiam ser avaliadas por mais de uma política, por uma política apenas, ou por nenhuma. Políticas com condições temporais foram criadas para verificar se os e-mails contendo correlações estavam sendo enviados aos GIs correspondentes.

Enquanto as *traps* eram enviadas, consultas foram realizadas ao repositório de *traps* para que fosse verificado todo processo de inserção e remoção dos dados. Dessa forma, foi possível validar a arquitetura e, a partir desse momento, possível realizar medições de tráfego e tempo consumidos com mensagens SNMP e chamadas Web Services, tanto para as chamadas que utilizam o HTTP, quanto às que utilizam o SMTP como protocolo de transporte. As seções seguintes mostram os ambientes de teste utilizados, bem como os resultados das medições realizadas.

6.1 Ambientes de Teste

Para a validação da arquitetura e a realização das medições, foram utilizadas três máquinas em rede, denominadas Hubble, Noc e Labcom. A configuração de cada uma está descrita na tabela 6.1.

Tabela 6.1: Máquinas utilizadas nos ambientes de teste e suas configurações.

Máquina	Processador	Memória RAM	Sistema Operacional
Hubble	Pentium III 550MHz	128MB	Windows 2000 Server
Noc	Pentium MMX 233MHz	64MB	Red Hat Linux 8.0
Labcom	Xeon 2.4GHz	1GB	Red Hat Linux 8.0

Para a validação da arquitetura, foi utilizado o "Ambiente de Teste I", conforme mostrado na Figura 6.1. Nesse ambiente, há mais de um nível de gerente intermediário e há o gerente superior, ou seja, a arquitetura está completa. Dessa forma, foi

possível realizar a validação da arquitetura e a comparação de medições de mensagens SOAP/HTTP e SOAP/SMTP. Nesse ambiente, as máquinas Noc e Labcom possuem mais de um componente da arquitetura, cada uma.

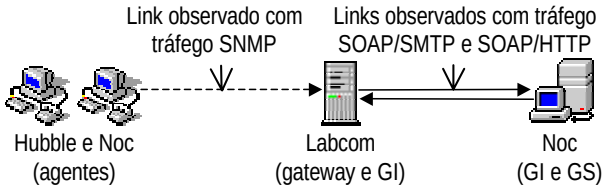

Figura 6.1: Ambiente de teste I.

Para verificar o volume de tráfego gerado pelas mensagens SNMP e WS, e o tempo de retardo das mensagens, um novo ambiente de teste foi criado (Figura 6.2), denominado "Ambiente de Teste II". Nesse ambiente, há apenas um nível de gerente intermediário e não há gerente superior. Dessa forma, cada máquina possui apenas um componente da arquitetura. Esse ambiente foi montado para que as medições de retardo de transmissão não fossem prejudicadas pelo uso de recursos do computador por outros componentes da arquitetura.

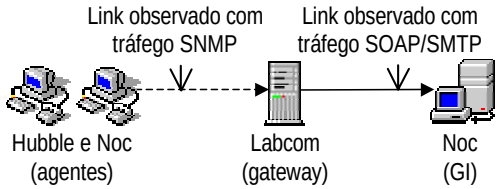

Figura 6.2: Ambiente de teste II.

6.2 Consumo de Tráfego SOAP/SMTP x SNMP

Na máquina Hubble, foi realizado o controle do envio de *traps* SNMP através de um *script* PHP simples. Esse *script* inclui quantas variáveis inteiras forem passadas por um parâmetro de entrada (linha de comando). As *traps* informavam um evento *link up* e a lista de variáveis descrevia as interfaces hipotéticas da máquina Hubble que voltavam a operar.

O tráfego SNMP inclui um cabeçalho regular e uma lista de variáveis adicionais. Inicialmente, foi gerada uma *trap* sem variáveis adicionais. O tamanho da *trap* e o volume de tráfego WS gerado para transferir a mensagem SOAP/SMTP correspondente, foram verificados. Depois, o número de interfaces foi progressivamente aumentado até 90. A Figura 6.3 mostra o volume de tráfego gerado pelas mensagens SNMP e WS.

Como pode ser observado, as mensagens SNMP geram muito menos tráfego que as mensagens SOAP/SMTP. Para mensagens com menos variáveis, a diferença desse volume de tráfego é considerável. Com o aumento do número de variáveis, o volume da diferença de tráfego também aumenta. Deve ser ressaltado que a proporção do tráfego gerado pelas mensagens SOAP/SMTP em relação às mensagens SNMP decresce na medida que a quantidade de variáveis aumenta. Com nenhuma variável adicional, o tráfego SOAP/SMTP é cerca de 52 vezes maior que o tráfego SNMP. Com 90 variáveis adicionais, a proporção cai para cerca de 5,7 vezes. Com mais de 90 variáveis, a proporção muda pouco, ou seja, não fica abaixo de 5 vezes.

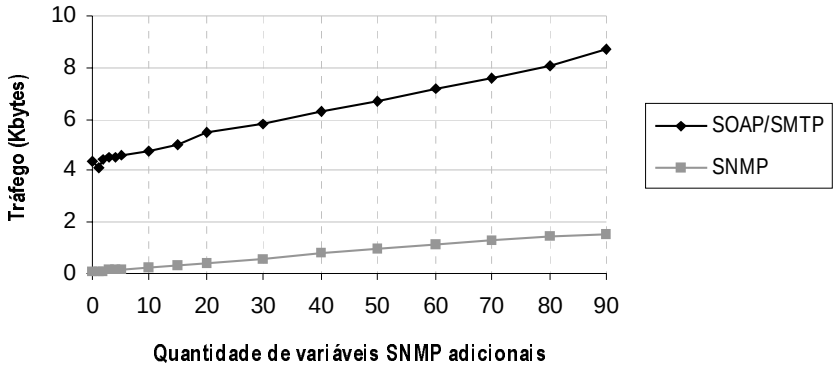

Figura 6.3: Tráfego SNMP x SOAP/SMTP.

A observação é crítica porque outras investigações (NEISSE et al., 2004) (FIOREZE et al., 2005) que comparam SNMP com WS mostraram que, mesmo que o tráfego gerado por WS seja maior que o tráfego gerado por SNMP, quando poucas variáveis estão presentes, o tráfego WS cresce menos que o tráfego SNMP quando mais variáveis estão presentes. Isso leva a uma situação em que, eventualmente, o tráfego WS, para um número muito grande de variáveis, seria menor que o tráfego SNMP associado. Mas essa observação é correta para mensagens *request/reply* (conforme mostrado nas investigações mencionadas), o que não é o caso deste trabalho.

6.3 Consumo de Tráfego SOAP/SMTP x SOAP/HTTP

As ações de configuração e transferência de políticas, realizadas pelo GS, são realizadas através de chamadas WS baseadas em HTTP, como mencionado anteriormente. O HTTP é utilizado pelo fato de possuir a característica *request/reply*, o que é geralmente requisitada por administradores de rede. Porém, essas ações também podem ser realizadas através de chamadas WS baseadas em SMTP.

Para a realização da comparação de consumo de tráfego entre as mensagens SOAP/HTTP e SOAP/SMTP (até então desconhecida na literatura), foi realizada a transferência de políticas de diferentes tamanhos entre o GS e um GI. Primeiramente, foi utilizada a implementação sem modificação alguma. Após, foram realizadas modificações na chamada WS no GS, e criado um *alias* e um *script* que recebe a chamada WS baseada em SMTP no GI.

Parte do "Ambiente de teste II" foi aproveitado, sendo somente utilizado o componente GS e um componente GI. Dessa forma, somente um *link* foi monitorado. Foram transmitidas seis políticas, com tamanhos que variam de 536 a 1122 *bytes* (conteúdo da política no formato XML apresentado na Figura 5.5, sem os cabeçalhos inseridos nas chamadas WS). A tabela 6.2 mostra o tamanho das políticas no formato XML e a quantidade de tráfego consumida pelas mensagens SOAP/HTTP e SOAP/SMTP em cada uma. A unidade de medida utilizada é *byte*.

Como pode ser visto na tabela, o tráfego consumido por mensagens SOAP/SMTP é maior que o tráfego consumido por mensagens SOAP/HTTP, quando trata-se de políticas menores. A partir do momento que se começa a transferir políticas de maior tamanho, as mensagens SOAP/HTTP aumentam um pouco menos que o tamanho das políticas, em proporção, e as mensagens SOAP/SMTP aumentam bem menos que as mensagens SOAP/HTTP. No exemplo das seis políticas apresentado, a diferença de

Tabela 6.2: Tamanho das políticas e tráfego consumido.

Política	Tamanho	Tráfego SOAP/HTTP	Tráfego SOAP/SMTP
1	536	4451	4664
2	644	4602	4844
3	796	5849	5118
4	966	6990	5698
5	994	7476	5768
6	1122	8060	5862

tamanho da política menor para a maior foi de aproximadamente 109%. As chamadas WS baseadas em HTTP aumentaram o seu tráfego em 81%, enquanto as baseadas em SMTP aumentaram apenas 25%. Para uma melhor visualização da diferença do tráfego entra as mensagens SOAP/HTTP e SOAP/SMTP, um gráfico é mostrado na Figura 6.4.

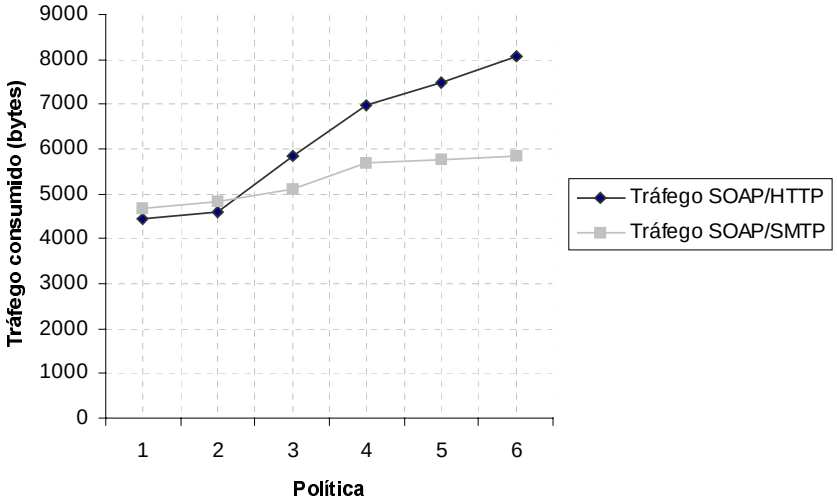

Figura 6.4: Tráfego SOAP/HTTP x SOAP/SMTP.

Em sistemas onde são utilizadas políticas mais complexas, e quando o retorno das chamadas WS não precisar ser instantâneo, WS baseados em SMTP parecem ser uma boa solução. Porém, também deve ser levado em conta o tempo de retardo de transmissão, onde o SMTP leva mais tempo que o HTTP, devido ao atraso derivado de servidores de e-mail. Um comparativo de retardo de transmissão é mostrado na seção seguinte, entre mensagens SNMP e SOAP/SMTP.

6.4 Retardo de Transmissão SOAP/SMTP x SNMP

Utilizando o mesmo intervalo de variáveis adicionais mencionado na seção 6.2, uma medição do retardo de transmissão foi realizada para comparar a diferença do tempo necessário para a transmissão de *traps* SNMP e mensagens SOAP/SMTP equivalentes. Para a realização dessa medida, os relógios das máquinas Hubble, Labcom e Noc foram sincronizados. No momento de envio de cada *trap*, o horário da máquina Hubble foi gravado. Na máquina Labcom, o arquivo temporário utilizado para gravar a *trap* recebida, foi verificado (um dos campos da *trap* é o horário local). No *script* de geração da mensagem SOAP/SMTP, comandos adicionais foram colocados antes da chamada da

API responsável por enviar a mensagem ao servidor SMTP, permitindo a gravação do horário corrente. Na máquina Noc, o mesmo procedimento foi realizado no *script* de recepção da mensagem SOAP/SMTP, após a recepção da mensagem e antes da gravação no repositório.

O que pôde ser observado é que, independente do volume de tráfego gerado (dentro do intervalo observado), não houve variação significativa na diferença do retardo de transmissão de mensagens SOAP/SMTP e SNMP. A variação da diferença verificada foi menor que 1 segundo. Todas *traps* SNMP foram transmitidas em menos de 1 segundo e todas mensagens SOAP/SMTP foram transmitidas em cerca de 49 segundos (Figura 6.5). Cabe ressaltar que nesses 49 segundos estão inclusos o tempo necessário para o estabelecimento de conexão TCP (já que o SMTP utiliza o TCP como protocolo de transporte) e o tempo de processamento de cada servidor de e-mail. No momento em que os dados da mensagem são lidos, o horário é gravado.

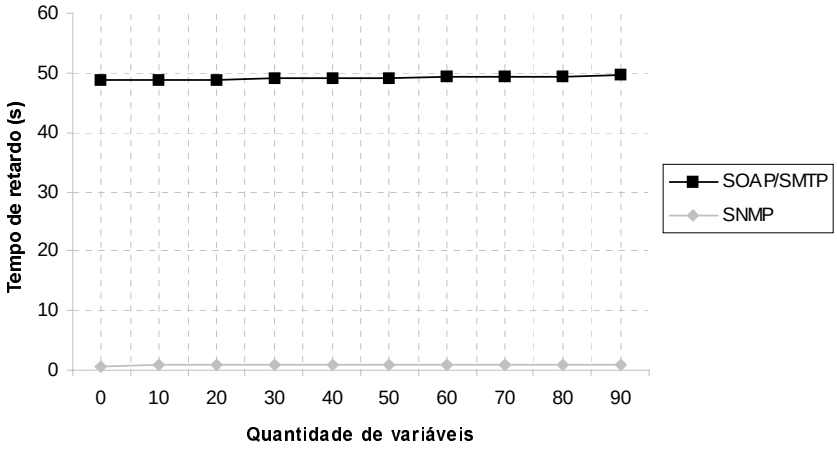

Figura 6.5: Retardo de transmissão SOAP/SMTP x SNMP.

Assim como medido no volume de tráfego gerado, a diferença no tempo de transmissão de mensagens SOAP/SMTP ficou em torno de 50 vezes maior que a transmissão de *traps* SNMP. Porém, essa diferença não mudou com o aumento do número de variáveis adicionais, ao contrário do volume de tráfego. Esse é um fator importante para ajudar na decisão do uso de uma dessas duas tecnologias, quando o tempo é considerado importante no sistema em questão.

As medidas apresentadas neste capítulo mostram algumas vantagens e desvantagens de se utilizar WS baseados em SMTP ao invés de se utilizar o mais difundido atualmente, os WS baseados em HTTP. No capítulo seguinte, são mostradas as conclusões da presente dissertação e os trabalhos futuros.

7 CONCLUSÕES E TRABALHOS FUTUROS

Um aspecto importante do uso de WS para o gerenciamento de redes é que esses são fáceis de implementar e de manter, e são adequados para a interconexão de diferentes sistemas. Outro importante aspecto é a garantia de entrega das mensagens de notificação, tornando o sistema mais confiável. SOAP sobre HTTP para o gerenciamento de redes tem sido investigado ultimamente, e pode ser considerado relativamente bem conhecido. Por outro lado, pouco se conhece sobre o gerenciamento de redes utilizando SOAP sobre SMTP. Há duas razões principais para esse fato: o suporte para SOAP sobre HTTP é bem mais utilizado que SOAP sobre SMTP, e SMTP é apropriado para mensagens assíncronas como as *traps*, mas as *traps* são bem menos utilizadas no SNMP do que outras mensagens síncronas tais como *GetRequest* e *GetResponse*.

Neste trabalho, foi apresentada uma arquitetura distribuída para correlação de notificações, onde *traps* SNMP são mapeadas para mensagens assíncronas SOAP/SMTP. Tal mapeamento é realizado por *gateways* que encaminham notificações aos GIs, responsáveis pela correlação das notificações baseada em políticas (definidas pelo administrador da rede). Essa arquitetura foi validada através da implementação do protótipo. Medições de tráfego e retardo de transmissão foram realizadas a fim de verificar vantagens e desvantagens nas tecnologias apresentadas. O fato da correlação ser baseada em políticas também é uma novidade, pois não há na literatura nenhum experimento envolvendo gerenciamento baseado em políticas (PBNM) para a correlação de notificações.

Em relação ao gerenciamento baseado em políticas, foi utilizada a técnica de raciocínio baseado em regras (RBR), por ser a mais intuitiva (do ponto de vista de um programador). O administrador da rede insere as políticas em um repositório e distribui aos gerentes escolhidos. O algoritmo de correlação utilizado no protótipo procura por políticas "candidatas" na medida que um *parser* interpreta os campos da *trap* recebida. Após ler todos campos da *trap* recebida, a política é escolhida entre as candidatas, caso alguma corresponda aos valores dos campos da *trap*. Esse é apenas um dos diversos algoritmos possíveis. O algoritmo adotado foi testado e funcionou de acordo com o esperado, pois as correlações previstas foram realizadas de forma correta.

Em relação às medidas realizadas, o volume de tráfego gerado pelas mensagens WS foi muito maior que o volume de tráfego gerado pelas mensagens SNMP. Essa é uma importante observação porque investigações anteriores focaram em mensagens SNMP síncronas, mostrando que em algumas situações o tráfego WS pode ser menor que o tráfego SNMP. Neste trabalho, contudo, mostrou-se que não somente o tráfego WS baseado em SMTP é maior que o tráfego SNMP, mas também que a diferença entre os dois cresce cada vez mais, se mais variáveis estão presentes em um objeto da MIB. No entanto, a proporção do tráfego SOAP/SMTP sobre o tráfego SNMP diminui

conforme o número de variáveis aumenta. Na comparação SOAP/SMTP x SOAP/HTTP, as mensagens SOAP/HTTP mostraram ser menos consumidoras de tráfego para políticas pequenas, enquanto as mensagens SOAP/SMTP mostraram ser menos consumidoras de tráfego em políticas médias e grandes. Em relação ao retardo de transmissão, uma grande diferença foi notada entre mensagens SNMP e SOAP/SMTP, sendo que mensagens SNMP mostraram ser cerca de 50 vezes mais rápidas que mensagens SOAP/SMTP, de acordo com o ambiente utilizado para teste.

Os fatos mostrados acima podem levar à conclusão de que notificações baseadas em WS não valem à pena. Isso é verdade se considerado apenas o aspecto de volume de tráfego gerado e o retardo de transmissão. Contudo, WS possuem facilidades de integração ausentes no SNMP e, ainda, notificações baseadas em WS podem cruzar diferentes domínios administrativos, pois WS são baseados em protocolos Web, enquanto o tráfego SNMP é geralmente restrito a redes locais. Pensando em correlação de eventos entre domínios diferentes, a solução apresentada sugere uma redução do número de notificações com a utilização de correlações, sendo que, em redes locais, mensagens SNMP ainda podem ser utilizadas e, em redes metropolitanas ou de longa distância, as mensagens SOAP/SMTP garantem a entrega nos diferentes domínios administrativos.

Trabalhos futuros nessa direção incluem investigações sobre como o tráfego gerado por chamadas WS pode ser reduzido com a utilização de métodos de compactação de dados, sem afetar muito o retardo de transmissão. Mais investigações em torno de outros protocolos, que podem servir de transporte aos Web Services (ex.: FTP, BEEP), serão realizadas com o intuito de verificar as vantagens e desvantagens de utilização de cada protocolo. Por fim, a solução proposta nesta dissertação será adaptada para a utilização do WS-Notification, que consiste em um padrão em desenvolvimento para suporte específico a notificações em WS. Para tanto, faz-se necessário investigar o uso do WS-Notification como suporte a notificações de eventos de rede.

REFERÊNCIAS

ABITEBOUL, S.; BENJELLOUN, O.; MILO, T. Web services and data integration. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, WISE, 2002. **Proceedings...** [S.l.:s.n.], 2002.

APACHE. **HTTP Server Project**. Disponível em: <http://httpd.apache.org/>. Acesso em: jun.2004.

APRISMA. **Spectrum**. Disponível em: <http://www.aprisma.com/>. Acesso em: ago.2004.

BLANK, M. Why Web Services Work: a positive impact on business - Web Services in the Real World. **Web Services Journal**, USA, v.3, p.36-39, July 2003.

BRADEN, R. et al. **Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification** : RFC 2205. [S.l.]: IETF, 1997.

CASE, J. et al. **A Simple Network Management Protocol (SNMP)**: RFC 1157. [S.l.]: IETF, 1990.

CASTRO, T. de; NOGUEIRA, J. An alarm correlation system for SDH networks. In: SBT/IEEE INTERNATIONAL TELECOMMUNICATIONS SYMPOSIUM, ITS, 1998, São Paulo, v.2, p.492-497. **Proceedings...** Piscataway, NJ: IEEE, 1998.

CHAN, K. et al. **COPS Usage for Policy Provisioning (COPS-PR)**: RFC 3084. [S.l.]: IETF, 2001.

CHOI, H.; CHOI, M.; HONG, J. Design and Implementation of XML-based Configuration Management System for Distributed Systems. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2004. **Proceedings...** [S.l:s.n.], 2004.

CONSORTIUM, W3C. **XML Protocol Working Group**. Disponível em: <http://www.w3c.org/2000/xp/Group/>. Acesso em: jul.2004.

CURBERA, F. et al. Unraveling the Web services Web: an Introduction to SOAP, WSDL, and UDDI. **IEEE Internet Computing**, [S.l.], v.6, n.2, p.86-93, Mar./Apr.2002.

DAMIANOU, N. et al. The Ponder Policy Specification Language. In: IEEE INTERNATIONAL WORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, POLICY, 2001. **Proceedings...** [S.l.:s.n.], 2001. p.18-22.

DURHAM, D. et al. **The COPS (Common Open Policy Service) Protocol**: RFC 2748. [S.l.]: IETF, 2000.

FIOREZE, T.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. Comparing Web Services with SNMP in a Management by Delegation Environment. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, **Proceedings...** [S.l.:s.n.], 2005.

GRESCHLER, D.; MANGAN, T. Networking lessons in delivering 'Software as a Service' - Part I. **International Journal of Network Management**, [S.l.], v.12, n.5, p.317-321, Sept./Oct.2002.

GRESCHLER, D.; MANGAN, T. Networking lessons in delivering 'Software as a Service' - Part II. **International Journal of Network Management**, [S.l.], v.12, n.6, p.339-345, Nov./Dec.2002.

GROUP, T. P. **PHP: hypertext preprocessor homepage**. Disponível em: <http://www.php.net/>. Acesso em: jul.2004

GROUP, T. P. **PEAR - PHP Extension and Application Repository**. Disponível em: <http://www.pear.php.net/>. Acesso em: jul.2004

HAROLD, W. **Using Extensible Markup Language-Remote Procedure Calling (XML-RPC) in Blocks Extensible Exchange Protocol (BEEP)**: RFC 3529. [S.l.]: IETF, 2003.

HEWLETT-PACKARD. **HP OpenView Management Software**. Disponível em: <http://www.managementsoftware.hp.com/>. Acesso em: ago.2004.

IBM. **Tivoli**. Disponível em: <http://www-306.ibm.com/software/tivoli/solutions/event/>. Acesso em ago.2004.

LEWIS, L. **Event Correlation in SPECTRUM and Other Commercial Products**. Aprisma White paper. Sept.2000. Disponível em: <http://www.aprisma.com/literature/white-papers/wp0551.pdf>. Acesso em: ago.2004

LOBO, J.; BHATIA, R.; NAQVI, S. A Policy Description Language. In: AAAI/IAAI NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 16., 1999. **Proceedings...** Menlo Park: AAAI Press, 1999. p.291-298.

LUPU, E. C.; SLOMAN, M. Conflicts in Policy-Based Distributed Systems Management. **IEEE Transactions on Software Engineering**, Los Alamitos, v.25, n.6, p.852-869, Nov./Dec.1999.

MOORE, B. **Policy Core Information Model (PCIM) Extensions**: RFC 3060. [S.l.]: IETF, 2001.

MYSQL. **MySQL Database Server**. Disponível em: <http://www.mysql.com/products/mysql/>. Acesso em: jul.2004.

NEISSE, R.; VIANNA, R. L.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. Implementation and Bandwidth Consumption Evaluation of SNMP to Web Sevices Gateways. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2004. **Proceedings...** [S.l.:s.n.], 2004. p.715-728.

OASIS. **Security Assertion Markup Language**. Disponível em: <http://xml.coverpages.org/saml.html>. Acesso em: ago.2003.

PAVLOU, G. et al. On Management Technologies and the Potential of Web Services. **IEEE Communications, special issue on XML-based Management of Networks and Services**, [S.l.], v.42, n.7, p.58-66, July 2004.

POSTFIX. **The Postfix Home Page**. Disponível em: <http://www.postfix.org/>. Acesso em: nov.2004.

PRAS, A. et al. Comparing the Performance of SNMP and Web Services-Based Management. **IEEE eTransactions on Network and Service Management (eTNSM)**, [S.l.], v.1, n.2, Dec.2004.

ROY, J.; RAMANUJAN, A. Understanding Web Services. **IEEE IT Professional**, [S.l.], p.69-73, Nov.2001.

SAHAI, A.; MACHIRAJU, V.; OUYANG, J.; WURSTER, K. Message Tracking in SOAP-based Web Services. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, **Proceedings...** [S.l.:s.n.], 2002.

SCHÖNWÄLDER, J.; PRAS, A.; MARTIN-FLATIN, J. P. On the Future of Internet Management Technologies. **IEEE Communications Magazine**, [S.l], v.41, n.10, p.90-97, Oct.2003.

SENDMAIL. **Sendmail Home Page**. Disponível em: <http://www.sendmail.org/>. Acesso em: nov.2004.

SLOTEN, J. v.; PRAS, A.; SINDEREN, M. J. v. On the standardisation of Web service management operations. Open European Summer School and IFIP WG6.3 Workshop. **Proceedings...** Tampere, Finland, Jun. 2004, pp. 143-150.

SMARTS. **InCharge Integrated Product Suite**. Disponível em: <http://www.smarts.com/products/>. Acesso em: ago.2004.

SOFTWARE, B. **BMC Patrol**. Disponível em: <http://www.bmc.com/>. Acesso em: ago.2004.

SOURCEFORGE.NET. **The NET-SNMP Project Home Page**. Disponível em: <http://net-snmp.sourceforge.net/>. Acesso em: nov.2004.

STANTON, D. The Differentiation of Web Services Security. **Web Services Journal**, [S.l.], v.3, p.18-20, June 2003. Disponível em: <http://www.sys-con.com/webservices/>. Acesso em: jul.2003.

VAUGHAN-NICHOLS, S. J. Web Services: beyond the hype. **Computer**, [S.l.], v.35, n.2, p.18-21, Feb.2002.

VERITAS. **Press Releases 1998**. Disponível em: <http://www.veritas.com/us/aboutus/pressroom/1998/98-02-09-0.html>. Acesso em: ago.2004.

WESTERINEN, A. et al. **Terminology for Policy-Based Management**: RFC 3198. [S.l.]: IETF, 2001.

YUAN, S.; LIN, K. Building Simple Intelligence into Web Services. In: IEEE/WIC INTERNATIONAL CONFERENCE ON WEB INTELLIGENCE, WI, 2003. **Proceedings...** [S.l.:s.n.], 2003. p.26.

ZUPAN, J.; MEDHI, D. An alarm management approach in the management of multi-layered networks. In: IEEE WORKSHOP ON IP OPERATIONS AND MANAGEMENT, IPOM, 2003. **Proceedings...** [S.l.:s.n.], 2003. p.77-84.