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

TIAGO FIOREZE

Gerenciamento de Redes Baseado em Web Services em um Ambiente de Gerenciamento por Delegação: uma Abordagem Orientada a Serviços

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

Profa. Dra. Maria Janilce B. Almeida
Orientadora

Porto Alegre, junho de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Fioreze, Tiago

Gerenciamento de Redes Baseado em Web Services em um Ambiente de Gerenciamento por Delegação: uma Abordagem Orientada a Serviços / Tiago Fioreze. – Porto Alegre: PPGC da UFRGS, 2005.

100 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientadora: Maria Janilce B. Almeida.

1. Gerenciamento de redes distribuído. 2. MIB Script. 3. Web Services. 4. Gateways WS para SNMP. 5. Agente Web Services. I. Almeida, Maria Janilce B. II. Título.

“Our greatest glory is not in never falling,
but in rising every time we fall.""
— CONFUCIUS

AGRADECIMENTOS

Gostaria de agradecer, primeiramente, a minha família. Aos meus pais Arni e Sueli Fioreze e a minha irmã Marister Fioreze pelo apoio dado não só nesta etapa, mas em todas as outras já realizadas. Vocês são muito especiais para mim. Este trabalho não teria sido realizado se não fosse a ajuda, em todos os sentidos, dada por vocês. Agradeço de coração!

Agradeço, também, a Deus por ter me acompanhado em mais uma etapa da minha vida e por ter me dado força e determinação para seguir em frente.

Agradeço à minha orientadora, Profa. Maria Janilce Bosquiroli Almeida, por ter me auxiliado no desenvolvimento deste trabalho e pelo acompanhamento dado durante o mestrado. Um agradecimento muito especial ao Prof. Lisandro Zambenedetti Granville, que é uma excelente pessoa e um exemplo de profissional a ser seguido, pelo desenvolvimento deste trabalho e pelo prestativo apoio dado durante o mestrado. Gostaria de agradecer também aos grandes amigos e professores do grupo de Redes de Computadores, Juergen Rochol e João César Netto pelos ensinamentos e auxílio dado. Agradeço, também, a professora Liane Margarida Rockenbach Tarouco pela colaboração e competência nas aulas ministradas.

Agradecimentos aos meus colegas de mestrado Ricardo Vianna, Ricardo Neisse, Evandro Pereira, Rodrigo Sanger, Rafael Huff, Clarissa Marquezan, Karina Roggia e Weldson Lima. Um agradecimento especial a Ricardo Vianna, Weldson Lima e Rodrigo Sanger pelo auxílio dado no desenvolvimento deste trabalho.

Ao pessoal do futebol de salão nas quartas-feiras, do vôlei nas quintas-feiras e aos colegas do coral Sons do Vale e do curso de inglês Yázigi, meu agradecimento pela convivência e amizade de vocês.

Agradeço aos meus amigos da Universidade Federal de Santa Maria, Daniel Ordobás Bortolás, Gabriela Jacques da Silva, Hélio Antônio Miranda, Lucas Mello Schnorr e Rodrigo Righi e ao meus amigos de Tapera pelo incentivo e amizade.

Gostaria de agradecer também aos demais colegas, professores e funcionários do Instituto de Informática pela prestatividade e ajuda dispensada.

Em resumo, a todos que de uma forma ou de outra me apoiaram, auxiliaram e contribuiriam não só na realização desta dissertação, mas em todas as atividades realizadas durante o mestrado, meus sinceros agradecimentos. Muito obrigado a todos!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DOM	Document Object Model
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
MbD	Management by Delegation
MIB	Management Information Base
MLM	Mid-Level Manager
OID	Object Identifier
QAME	QoS-Aware Management Environment
RFC	Request for Comments
SMI	Structure of Management Information
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TLM	Top-Level Manager
UDDI	Universal Description, Discovery, and Integration
UDP	User Datagram Protocol
W3C	World Wide Web Consortium
WS	Web Services
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language

LISTA DE FIGURAS

RESUMO

Novas tecnologias têm sido investigadas como soluções alternativas ao protocolo de gerenciamento de redes padrão, o SNMP. Nesse cenário, a tecnologia de Web Services (WS) vem se tornando bastante promissora, mas imaginar que ela irá substituir de imediato o SNMP não é uma abordagem factível, pois é provável que o SNMP continue sendo utilizado nos dispositivos de rede por um bom tempo. Além disso, os WS são recentes, necessitando maiores investigações com respeito a utilização no gerenciamento de redes. As atuais investigações realizadas comparam os WS com o SNMP em casos genéricos, não levando em consideração casos específicos de gerenciamento. Nesse contexto, esta dissertação apresenta o desenvolvimento e a avaliação de duas implementações baseadas em WS, desenvolvidas no específico contexto do gerenciamento distribuído por delegação e tendo a MIB Script, definida pelo IETF, como base de desenvolvimento. A primeira implementação consiste na construção de um novo tipo de *gateway* WS para SNMP, denominado *gateway* em nível de serviço, que caracteriza-se por ser construído baseado nos serviços que um determinado módulo de MIB oferece. Isso proporciona um grau maior de abstração, na visão do gerente WS, na manipulação dos agentes SNMP e, também, diminui o número de mensagens trocadas entre o gerente WS e o *gateway* desenvolvido. A segunda implementação realiza a total substituição do SNMP no gerenciamento dos dispositivos, permitindo, com isso, um gerenciamento de redes completamente baseado em Web Services. A implementação consistiu na construção de um novo tipo de agente, denominado por agente orientado a serviços baseado em Web Services, ou apenas agente WS. O agente WS desenvolvido possui as mesmas características que o *gateway* em nível de serviço, diferindo apenas na não utilização de SNMP no caso do agente WS. O gerenciamento das duas implementações é realizado através de um módulo desenvolvido e incorporado ao ambiente de gerenciamento de redes QAME. Esse módulo manipula as duas implementações da mesma maneira, uma vez que os serviços oferecidos por elas possuem a mesma interface de acesso. A avaliação realizada consistiu na comparação entre as duas implementações desenvolvidas neste trabalho e outras duas soluções de gerenciamento investigadas: o *gateway* em nível de objeto e uma implementação da MIB Script, denominada Jasmin. Os parâmetros de avaliação utilizados nas comparações foram o tráfego gerado e o tempo de resposta percebido. Como será visto ao final, os resultados das comparações realizadas no caso específico do gerenciamento por delegação são diferentes daqueles apresentados em investigações mais genéricas. Além disso, as duas implementações desenvolvidas neste trabalho demonstraram serem promissoras, apresentando resultados bastante satisfatórios com relação aos parâmetros avaliados.

Palavras-chave: Gerenciamento de redes distribuído, MIB Script, Web Services, gateways WS para SNMP, agente Web Services.

Web Services-based Network Management in a Management by Delegation Environment: a Service-Oriented Approach

ABSTRACT

In the past years, alternative technologies for the Simple Network Management Protocol (SNMP) solution, which is the *de facto* standard protocol for network management, has been under investigation. In this scenario, recently, the Web Services (WS) technologies have been gaining more attention from the network management research community. However, the approach where SNMP is completely replaced by WS is unfeasible, since SNMP will be probably used on network devices for a long time. In addition, WS are young and require further investigation concerning its use as a network management alternative. The current SNMP versus WS are characterized by the fact that they are taken from a quite broad view, without paying closer attention to particular management contexts. This dissertation presents the development and evaluation of two WS-based implementations, created in the specific context of the distributed Management by Delegation (MbD). The Script MIB , which is a IETF specification for MbD in IP networks, is taken as basis for the development of this work. The first implementation consists of a WS to SNMP gateway called Service-level gateway. This gateway maps the Script MIB services to WS operations. From the perspective of a WS-based manager, that provides a higher level of abstraction in dealing with SNMP particularities. Also, this reduces the number of management messages exchanged between the WS-based manager and the gateway developed. The second implementation totaly replaces SNMP for device management, allowing a WS-based solely management. This second implementation consists in building up a novel kind of management agent, called WS-based service-oriented agent, or simply a WS agent. The developed agent has the same management interface found in the service-level gateway, but the WS agent does not uses SNMP to access management information. In order to access these two implementation, a management module has been developed and incorporated in the QAME Web-based management system. Such module interacts with the two implementations through the same operations, since both implementations offer the same management interface. This work has been evaluated comparing the two implementations with other two solutions previous developed: a WS to SNMP Object-level gateway, and Jasmin, an open source Script MIB implementation. The consumed network bandwidth and the perceived execution time are the parameters considered. At the end, the results from the comparison for the specific context of the management by delegation are different than those presented in the current more generic and broad investigations in the field. In addition, the two prototypes developed in this work seem to be promising, presenting satisfactory results concerning the evaluated parameters.

Keywords: Distributed network management, Script MIB, Web services, SNMP to WS gateways, WS agent.

1 INTRODUÇÃO

O gerenciamento de redes de computadores é uma tarefa crítica cujo objetivo principal é manter os serviços de comunicação disponibilizados aos usuários o maior tempo possível operantes e funcionando com desempenho satisfatório. Como redes de computadores são de natureza heterogênea, possuindo dispositivos muito variados e diferentes, o acesso a tais dispositivos para gerenciamento deve ser feito preferencialmente através do uso de um protocolo padronizado. O IETF (*Internet Engineering Task Force*) (INTERNET SOCIETY, 2005) definiu, para este fim, o SNMP (*Simple Network Management Protocol*) (CASE et al., 1990). Na verdade o SNMP não é apenas um protocolo de gerenciamento, mas uma solução completa composta, pelo menos, dos seguintes elementos: gerente, agente, protocolo e base de informações de gerenciamento.

Um gerente SNMP é um processo de software em execução numa estação de gerenciamento. Os agentes também são processos de software localizados dentro dos dispositivos gerenciados e têm por função principal receber as solicitações dos gerentes, executar uma tarefa solicitada (por exemplo, alterar uma configuração do dispositivo, ou reportar o estado interno do mesmo) e retornar o resultado de tal execução. A comunicação entre gerentes e agentes, obviamente, se dá através do protocolo SNMP. Por fim, as informações dos dispositivos que um gerente pode manipular junto aos agentes são definidas em módulos de bases de informações de gerenciamento, ou módulos MIB (*Management Information Base*). Tais módulos são definidos seguindo a SMI (*Structure Management Information*) (ROSE; MCCLOGHRIE, 1990), que é também uma especificação definida pelo IETF.

O número de gerentes e agentes empregados no gerenciamento de uma rede e suas interações acabam por definir alguns paradigmas de gerenciamento. O paradigma mais amplamente utilizado ainda hoje é o paradigma centralizado. Neste, um único gerente localizado numa estação de gerenciamento central controla todos os dispositivos da rede gerenciada acessando os agentes internos aos dispositivos. Apesar de ser amplamente utilizado, o paradigma centralizado possui deficiências importantes: o mesmo não é escalável porque a estação de gerenciamento pode se tornar sobrecarregada se o número de dispositivos a serem gerenciados for muito elevado; o gerenciamento centralizado não é flexível porque a instalação de novos serviços de gerenciamento na estação central pode necessitar que a mesma se torne temporariamente inoperante durante o processo de instalação; e ele não é confiável porque se a única estação central deixar de operar, toda a rede deixará de ser gerenciada.

Apesar de o SNMP ter se tornado ao longo dos anos o padrão de fato no gerenciamento de redes TCP/IP, o protocolo em si possui restrições importantes. Por exemplo, as informações para autenticação das solicitações dos gerentes junto aos agentes não são criptografadas, o que permite que estas possam ser facilmente descobertas por terceiros.

O IETF padronizou, mais recentemente, o SNMPv3 (HARRINGTON; PRESUHN; WIJNEN, 1999) que é uma versão do SNMP que inclui suporte mais adequado às questões de segurança, mas ainda assim a grande maioria dos dispositivos usam agentes SNMP que suportam a versão 1 original ou a versão 2 do protocolo, que não possuem segurança adequada. Por essa razão, na maioria dos casos os administradores de rede preferem utilizar o SNMP apenas como mecanismo de monitoração dos dispositivos, e não como ferramenta para configuração dos mesmos (SCHÖNWÄLDER; PRAS, 2004), o que obviamente restringe o uso do protocolo.

Devido aos problemas do SNMP, várias melhorias ou soluções alternativas têm sido investigadas. O uso de agentes móveis (GOLDSZMIDT; YEMINI, 1998) e CORBA (*Common Object Request Broker Architecture*) (OMG, 1997) são dois exemplos de tais soluções alternativas, nesse caso, definidas principalmente por investigações acadêmicas. Os próprios organismos de padronização, por sua vez, também vem investigando outras soluções de gerenciamento. Por exemplo, o IETF está atualmente trabalhando na especificação do NETCONF (ENNS, 2005) que é um protocolo baseado em XML (*eXtensible Markup Language*) (W3C, 1996) voltado principalmente à configuração de dispositivos que, como visto anteriormente, é um área onde o SNMP não se apresenta como uma solução muito adequada.

Nesse contexto de soluções alternativas, recentemente a tecnologia de Web Services (WS) (CURBERA et al., 2002) tem sido apresentada como uma proposta bastante interessante para o gerenciamento de redes. Criados principalmente no contexto de comércio eletrônico, e em processo de padronização do W3C (*World Wide Web Consortium*) (W3C, 1994), os Web Services podem ser descritos como um conjunto de componentes independentes disponibilizados na Web que utilizam protocolos bem conhecidos para suas comunicações (por exemplo, HTTP (FIELDING et al., 1999), SMTP (KLENSIN, 2001) e FTP (POSTEL; REYNOLDS, 1985)). Tais componentes podem assim ser acessados por aplicações Web ou ainda por outros Web Services. Nesse último caso, como um WS pode requisitar operações de outro WS, é possível se construir serviços sofisticados a partir da composição de serviços mais simples.

O uso de WS para gerenciamento de redes, entretanto, é uma abordagem nova e por isso necessita de investigação, antes que a tecnologia possa ser efetivamente utilizada em soluções de gerenciamento de mercado. Atualmente, por exemplo, comparações entre WS e SNMP são motivo de intensas pesquisas com resultados interessantes, ainda que preliminares. Alguns estudos já mostram que WS podem consumir menos largura de banda que SNMP quando um número elevado de informações de gerenciamento precisa ser obtido dos dispositivos gerenciados (NEISSE et al., 2004). Por outro lado, outros estudos mostram que questões de desempenho podem dificultar o uso de WS (PAVLOU et al., 2004), enquanto outros pesquisadores podem argumentar que a facilidade de uso dos Web Services é uma vantagem que poderia superar os problemas de desempenho (DREVERS; MEENT; PRAS, 2004).

Apesar de WS parecerem promissores no gerenciamento de redes, imaginar que eles serão substitutos imediatos do SNMP não é uma abordagem factível, uma vez que dispositivos com suporte a SNMP irão permanecer nas redes provavelmente ainda por um longo período. Por outro lado, sem uma introdução gradual dos WS no gerenciamento de redes não será possível se ter a disposição as vantagens que essa nova tecnologia oferece. Assim, nesta dissertação parte-se da visão de que os WS para gerenciamento de redes serão introduzidos e utilizados de forma gradual onde duas principais etapas podem ser previstas.

Em uma primeira etapa, WS e SNMP co-existirão de forma que as vantagens dos WS estarão a disposição dos softwares de gerenciamento (i.e. gerentes baseados em Web Services), mas dispositivos finais continuarão sendo gerenciados através de SNMP, já que a substituição imediata de tais dispositivos, como citado anteriormente, não é factível. Nessa primeira etapa, existe a clara necessidade de integração entre WS e SNMP. Por exemplo, uma requisição WS disparada por um gerente WS deve ser, em algum momento, traduzida para uma requisição SNMP entregue ao dispositivo final. Da mesma forma, a resposta SNMP gerada pelo dispositivo gerenciado precisa ser traduzida de volta para WS de forma que o gerente possa processá-la. Um dos pontos investigados nesta dissertação está relacionado com as abordagens para construção de *gateways* SNMP para WS, cuja função é então exatamente a de fornecer traduções entre WS e SNMP de forma a integrar tais tecnologias.

Atualmente, a utilização de *gateways* para integração de SNMP e WS vem sendo bastante pesquisada pela comunidade de gerenciamento de redes (YOON-JUNG et al., 2002) (KLIE; STRAUSS, 2004) (NEISSE et al., 2003). Tais investigações, entretanto, são caracterizadas por serem extremamente genéricas, não levando em consideração situações específicas de gerenciamento que poderiam revelar aspectos importantes da utilização de WS em gerenciamento de redes. Neste trabalho, um novo tipo de *gateway*, chamado de *gateway* em nível de serviço (a ser detalhado no decorrer do texto), é apresentado e avaliado, e sua construção, diferentemente das abordagens atuais, é orientada a situações específicas. Na pesquisa desse novo *gateway*, o gerenciamento por delegação (YEMINI; GOLDSZMIDT; YEMINI, 1991) é tomado como estudo de caso. O gerenciamento por delegação, que também será apresentado em mais detalhes à frente, visa substituir o paradigma de gerenciamento centralizado, citado anteriormente, através da habilidade de delegação de tarefas de gerenciamento entre gerentes (nesse caso, vários gerentes, e não apenas um, são utilizados no gerenciamento de uma rede). O IETF padronizou um módulo de MIB chamado MIB Script (LEVI; SCHÖNWÄLDER, 2001) cujo objetivo é justamente dar suporte ao gerenciamento por delegação na solução SNMP. A investigação apresentada nesta dissertação compara então a MIB Script com os *gateways* em nível de serviço de forma a avaliar, principalmente, o tráfego gerado e o tempo de resposta percebido de tais *gateways* no contexto específico do gerenciamento por delegação.

Em uma segunda etapa, que sucede o uso integrado de WS e SNMP, espera-se que os dispositivos gerenciados incorporem internamente os WS de gerenciamento, complementando ou substituindo o agente SNMP normalmente empregado. Neste trabalho, o termo "agente WS" é utilizado para se referenciar a um WS de gerenciamento que é encontrado dentro de um dispositivo de rede. Isso permite, ao longo do texto, se ter uma diferenciação clara entre os *gateways*, os agentes SNMP, e os agentes WS. Nesse contexto, esta dissertação também apresenta o desenvolvimento e a avaliação de um agente WS para o suporte ao gerenciamento por delegação. Nesse caso, o agente WS desenvolvido substitui um agente SNMP que suporta a MIB Script, chamado Jasmin (JASMIN, 2000). O Jasmin é uma implementação da MIB Script de código aberto devenvolvida pela Universidade de Braschweing e NEC Europe e é disponibilizada para *download* na Internet. O Jasmin, na realidade, foi utilizado não apenas nessa segunda etapa, mas também na avaliação da primeira etapa comentada anteriormente.

Além do desenvolvimento do *gateway* (primeira etapa) e do agente WS (segunda etapa) para gerenciamento por delegação, um gerente baseado em WS também foi implementado. O gerente WS desenvolvido foi incorporado ao ambiente de gerenciamento de redes QAME (*QoS-Aware Management Environment*) (GRANVILLE et al., 2001) de-

senvolvido no Instituto de Informática da UFRGS, sendo resultado de outras pesquisas em gerenciamento de redes realizada pelo Grupo de Redes do Instituto. Além de permitir o controle dos processos de gerenciamento por delegação por parte de um administrador de redes, a implementação do gerente junto ao QAME permitiu que o *gateway* em nível de serviço e o agente WS desenvolvidos pudessem ser avaliados em um ambiente mais próximo de um ambiente real.

O restante desta dissertação está organizada da seguinte forma. O capítulo 2 apresenta os trabalhos relacionados e o estado da arte em relação ao gerenciamento de redes por delegação. O capítulo 3 apresenta uma descrição geral das duas principais contribuições deste trabalho, o protótipo de um *gateway* em nível de serviço e de um agente WS orientado a serviços. Além disso, o capítulo apresenta também um mapeamento dos serviços oferecidos pela MIB Script para operações WS. No capítulo 4, detalhes de implementação dos protótipos e a interação dos mesmos com o gerente desenvolvido são apresentados. O capítulo 5 mostra as avaliações dos protótipos em relação ao tráfego gerado e ao tempo de resposta. Por fim, no capítulo 6, são apresentadas as conclusões resultantes desta dissertação e os trabalhos futuros.

2 TRABALHOS RELACIONADOS

O modelo de gerenciamento distribuído por delegação (MbD - *Management by Delegation*) é uma alternativa importante para os problemas encontrados no gerenciamento centralizado. Com a sua capacidade de distribuição de operações de gerenciamento para outras entidades participantes do sistema de gerenciamento, o MbD fornece uma maior escalabilidade em relação ao número de dispositivos gerenciados, além de uma descentralização de operações que antes eram executadas na única estação de gerenciamento típica do gerenciamento centralizado.

A abordagem do gerenciamento por delegação tornou-se concreta com a definição da MIB Script (LEVI; SCHÖNWÄLDER, 2001) pelo DISMAN (*Distributed Management Working Group*) (IETF, 2005) que é um grupo de trabalho do IETF (*Internet Engineering Task Force*) (INTERNET SOCIETY, 2005). A MIB Script fornece funcionalidades que permitem, através do uso de SNMP, que *scripts* de gerenciamento sejam transferidos, controlados remotamente e o resultado das execuções seja obtido.

Uma nova tecnologia que vem se tornando bastante promissora na área de gerenciamento de redes são os Web Services (WS). Embora os WS não tenham sido originalmente projetados para serem usados na tarefa de gerenciar dispositivos de rede, as suas características de funcionamento tais como, interoperabilidade e facilidade em atravessar *firewalls*, permitem, por exemplo, que um sistema hierárquico de gerenciamento seja construído.

A utilização de WS na área de gerenciamento vem sendo gradativamente introduzida através da utilização de *gateways* que convertem operações WS para operações SNMP e vice-versa. Isso ocorre devido aos esforços por parte da comunidade científica da área (KLIE; STRAUSS, 2004) (NEISSE et al., 2003) em unir a recente tecnologia de Web Services, caracterizada pela sua facilidade de implementação e utilização, com o bem conhecido e projetado protocolo de gerenciamento que é o SNMP.

Neste capítulo serão apresentados, inicialmente, conceitos sobre gerenciamento centralizado e distribuído, mais especificadamente sobre gerenciamento distribuído por delegação. Em seguida, uma descrição da estrutura da MIB Script desenvolvida pelo IETF é realizada. Após isso, a tecnologia Web Services será mostrada, seguida por um estudo sobre os *gateways* existentes atualmente. Por fim, o capítulo encerra definindo o problema investigado nesta dissertação.

2.1 Gerenciamento distribuído por delegação

Em redes de grande escala, o paradigma tradicional de gerenciamento centralizado alcançou seus limites (GOLDSZMIDT; YEMINI, 1995). Em tal paradigma, uma única estação de gerenciamento é responsável pela monitoração e controle de agentes de ge-

renciamento distribuídos ao longo da rede. Agentes são normalmente apenas um meio para coletar ou modificar os dados em um dispositivo gerenciado, atuando de uma forma bastante simples no processo inteiro de gerenciamento. No gerenciamento centralizado, uma estação central (gerente) recupera e processa os dados coletados dos agentes usando um protocolo de gerenciamento, tal como o SNMP (Figura 2.1).

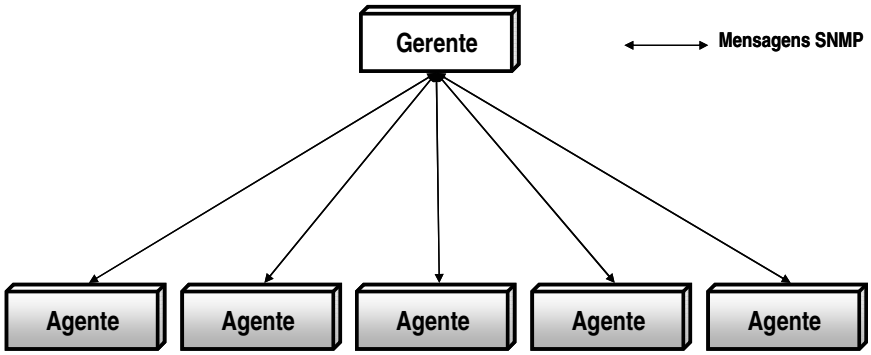

Figura 2.1: Gerenciamento centralizado

Entretanto, esse paradigma de gerenciamento não é escalável, definindo, com isso, um limite na gerenciabilidade de uma rede. Por exemplo, existe um limiar a ser cuidado sobre quantas variáveis podem ser consultadas (*polled*) pela estação central e o quão freqüente isso pode ser realizado (BEN-ARTZI; CHADNA; WARRIER, 1990). Além disso, com o crescimento da rede, o processamento de dados na estação de gerenciamento pode se tornar inviável devido à elevada quantidade de dados que precisam ser coletados e processados. Outro fator contra a utilização do gerenciamento centralizado está relacionado com a disponibilidade do processo de gerenciamento. Como apenas uma única estação é responsável pela monitoração e consulta dos agentes da rede, se tal estação falhar ou ficar sobrecarregada, a comunicação com os agentes estará comprometida, uma vez que os agentes ficarão inoperantes aguardando instruções de um gerente que não está mais disponível. Devido a isso, uma rede de dispositivos pode deixar de ser gerenciada.

O gerenciamento distribuído propôs a melhora na escalabilidade e flexibilidade dos sistemas de gerenciamento, contornando os problemas do paradigma centralizado. O paradigma de gerenciamento distribuído por delegação (MbD) propõe a delegação de funções de gerenciamento para localizações distribuídas, que pode ser realizada pela transferência e controle remoto de *scripts* de gerenciamento (YEMINI; GOLDSZMIDT; YEMINI, 1991). Isso acaba permitindo que funções de gerenciamento fiquem mais próximas das entidades gerenciadas, o que por sua vez diminui o tráfego de gerenciamento em um único ponto, distribuindo decisões de gerenciamento ao longo da rede gerenciada.

Seguindo a classificação de gerenciamento distribuído apresentada por Schönwälder et. al. (SCHÖNWÄLDER; QUITTEK; KAPPLER, 2000), as entidades de um sistema MbD podem ser classificadas como gerentes superiores (TLMs - *Top-Level Managers*), gerentes intermediários (MLMs - *Mid-Level Managers*) e agentes. TLMs são responsáveis pelo monitoramento e controle de execução de *scripts* delegados para localizações remotas, onde os MLMs são encontrados. Portanto, MLMs são os receptores de *scripts* de gerenciamento e aqueles responsáveis pela execução de tais *scripts* como forma de realizar a tarefa de gerenciamento. Para fazer isso, MLMs também interagem com agentes de gerenciamento localizados dentro de dispositivos de rede. Por fim, agentes são responsáveis pela consulta ou alteração dos dados existentes nos dispositivos de rede aos quais

eles encontram-se associados. A comunicação entre um MLM e um agente depende do protocolo de gerenciamento implementado pelo agente e o correspondente suporte a tal protocolo encontrado nos *scripts* sendo executados no MLM. A comunicação entre um TLM e um MLM, por outro lado, não depende mais dos *scripts* de gerenciamento: tal comunicação, necessária para transferir e controlar os *scripts*, é totalmente independente do conteúdo dos *scripts*.

Seguindo ainda as definições apresentadas por Schönwälder et. al., os paradigmas de gerenciamento distribuído diferem entre si em relação a quantidade de TLMs, MLMs e agentes que fazem parte do processo de gerenciamento. Schönwälder et. al. definiram *G* como o número total de TLMs e MLMs, e *N* o número total de elementos no sistema de gerenciamento, isto é, a soma de TLMs e MLMs e do número de agentes. Com essas definições, o paradigma de gerenciamento pode ser dividido em gerenciamento: fracamente distribuído, fortemente distribuído e cooperativo.

O gerenciamento fracamente distribuído caracteriza-se pela quantidade de MLMs e TLMs ser próxima a 1 e muito menor que a quantidade total de elementos participantes do sistema de gerenciamento. De uma forma relacional, tem-se que: $1 < G \ll N$. Esse paradigma de gerenciamento caracteriza-se, também, por não existir interação direta entre os MLMs (Figura 2.2).

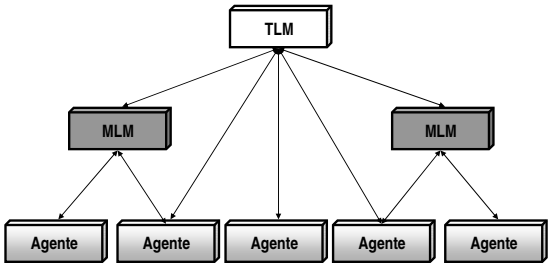

Figura 2.2: Gerenciamento fracamente distribuído

No gerenciamento fortemente distribuído, a quantidade de MLMs e TLMs é muito maior que 1 e é próxima da quantidade total de elementos participantes do sistema de gerenciamento, o que pode ser descrito por: $1 \ll G < N$. Além disso, diferentemente do gerenciamento fracamente distribuído, o gerenciamento fortemente distribuído caracteriza-se pela interação direta entre os MLMs para a realização das tarefas de gerenciamento (Figura 2.3).

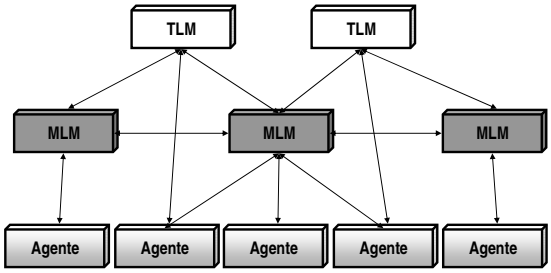

Figura 2.3: Gerenciamento fortemente distribuído

Por fim, o gerenciamento cooperativo caracteriza-se pela quantidade de gerentes (MLMs e TLMs) ser aproximadamente igual a quantidade total de elementos participantes do sis-

tema de gerenciamento, podendo ser descrito por: $G \approx N$. Salienta-se que no gerenciamento cooperativo (Figura 2.4) não existe mais o papel de TLMs e MLMs: os gerentes cooperam entre si para gerenciar os agentes.

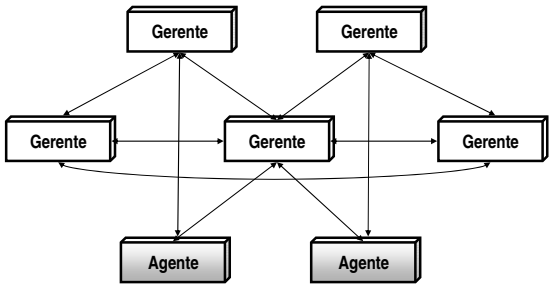

Figura 2.4: Gerenciamento cooperativo

Várias outras vantagens e características do modelo MbD são amplamente cobertos pela literatura sobre gerenciamento de redes (MARTIN-FLATIN, 2002) (KAHANI; BEADLE, 1997).

2.2 MIB Script

O IETF tem trabalhado na definição de um mecanismo baseado em SNMP que dê suporte a comunicação entre TLMs e MLMs. Desse esforço, o grupo de trabalho DISMAN definiu a MIB Script (LEVI; SCHÖNWÄLDER, 2001), que é um módulo de uma base de informações de gerenciamento SNMP (MIB) suportada pelos MLMs que permite aos TLMs transferir, controlar e obter resultados dos *scripts* de gerenciamento através do protocolo SNMP. A MIB Script não impõe a utilização de nenhuma linguagem específica a ser usada nos *scripts* de gerenciamento: ela é somente um mecanismo para controlar os *scripts*, não definindo e nem verificando o conteúdo existente nos mesmos.

A transferência dos *scripts* de gerenciamento pode ser feita através de dois métodos: **push** e **pull**. No primeiro, o TLM envia o *script* ao MLM através de sucessivas interações entre ele e o MLM. No segundo método, o TLM simplesmente informa ao MLM a localização do *script* e este fica responsável pela obtenção do *script*. Uma vez que o *script* de gerenciamento esteja localizado no MLM, ambientes de execução (e.g. Java ou Perl) são necessários para que se inicie a execução do mesmo. Logo, um MLM é composto por uma implementação da MIB Script e por um conjunto de ambientes de execução responsáveis por executar os *scripts* de gerenciamento transferidos.

2.2.1 A estrutura da MIB Script

A MIB Script é composta por seis tabelas (Figura 2.5), contendo, cada uma das tabelas, objetos que, quando manipulados de uma forma específica, realizam uma determinada ação de gerenciamento.

A tabela `smLangTable` lista todos os ambientes de execução suportados por uma implementação da MIB Script. Por exemplo, o Jasmin atualmente suporta dois ambientes de execução: Java e Tcl. A tabela em si é composta por um objeto que funciona como chave primária da tabela (`smLangIndex`) e por objetos que descrevem um ambiente de execução. Um ambiente de execução é descrito pelo seu nome (`smLangLanguage`), pela sua versão (`smLangVersion`), pelo seu fabricante (`smLangVendor`), pela revisão da versão (`smLangRevision`) e por uma descrição textual do ambiente (`smLangDescr`).

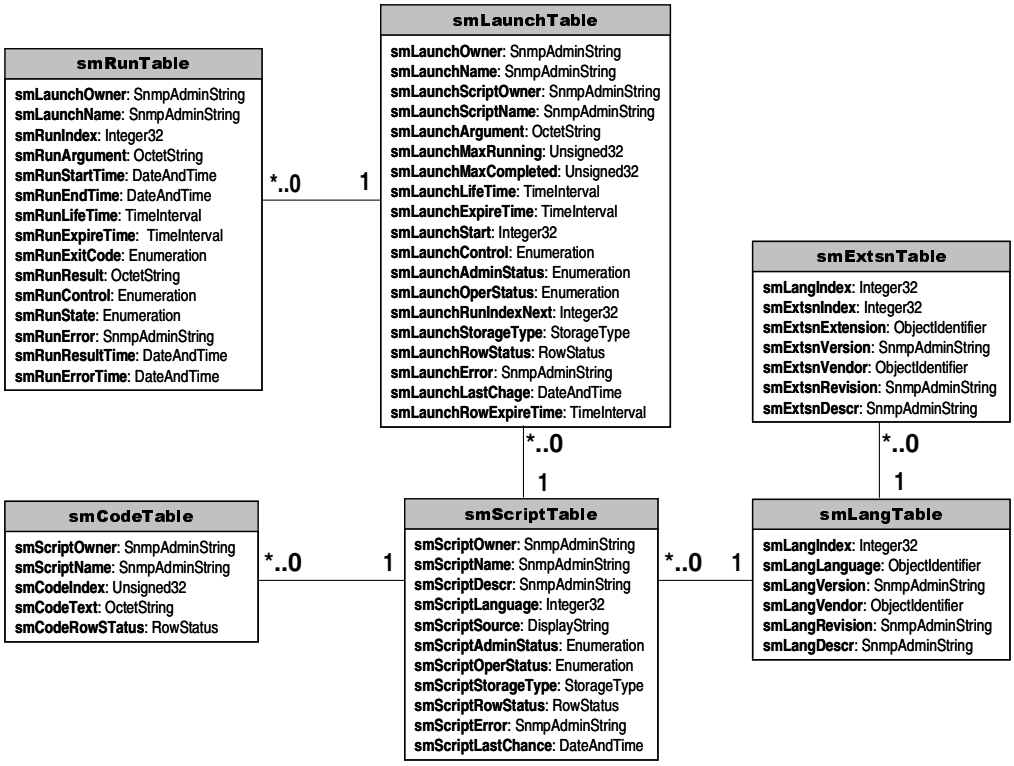

Figura 2.5: Modelo de informação da estrutura da MIB Script

A tabela `smExtsnTable` é uma extensão da tabela `smLangTable` apresentada anteriormente. Ela descreve todas as extensões existentes para um determinado ambiente de execução. Por exemplo, o ambiente de execução Java possui, dentre outras, as extensões 'policyMgmt' e 'snmpmonitor' que são extensões que permitem o gerenciamento de redes baseado em políticas e o monitoramento através de SNMP para o ambiente Java. A tabela em si é composta por dois objetos que funcionam como uma chave primária composta: `smLangIndex` e `smExtsnIndex`. O objeto `smLangIndex` é uma chave estrangeira que referencia o objeto `smLangIndex` definido na tabela `smLangTable` e o objeto `smExtsnIndex` é a chave primária da tabela `smExtsnTable`. Os demais objetos descrevem as extensões existentes para um determinado ambiente de execução. Uma extensão é descrita por um identificador de extensão (`smExtsnExtension`), pela versão da extensão (`smExtsnVersion`), pelo seu fabricante (`smExtsnVendor`), pela revisão da versão (`smExtsnRevision`) e por uma descrição textual da extensão (`smExtsnDescr`).

A tabela `smScriptTable` é responsável por listar todos os *scripts* de gerenciamento que estão atualmente presentes em uma implementação da MIB Script. Ela contém objetos que permitem um gerente instalar um *script* através do método **pull**. Nesse método, o gerente fornece, para uma implementação da MIB Script, a localização remota (URL) de um *script*, e ela fica responsável em buscá-lo através de um protocolo de transferência de arquivos (e.g. HTTP ou FTP). A tabela também permite remover um *script* e consultar o seu estado atual. Os objetos que fazem parte da tabela consistem de dois objetos que funcionam como uma chave primária composta: `smScriptOwner` e `smScriptName`. A chave primária `smScriptOwner` define quem é o proprietário do *script* de gerenciamento e `smScriptName` informa o nome do *script*. Outros quatro objetos servem para descrever um *script* de gerenciamento, sendo eles:

- `smScriptDescr`: objeto que fornece uma descrição textual do *script*;
- `smScriptLanguage`: objeto que está associado a uma entrada na tabela `smLangTable` e que informa qual o ambiente de execução suportado pelo *script*;
- `smScriptSource`: objeto que fornece a localização remota do *script*;
- `smScriptStorageType`: objeto que possibilita o armazenamento de um *script* em um meio não-volátil.

Os últimos três objetos da tabela `smScriptTable` são objetos do tipo "operacionais". Com eles é possível alterar o estado (‘enabled’, ‘disabled’ ou ‘editing’) de um *script* de gerenciamento através da manipulação do objeto `smScriptAdminStatus`; obter informações sobre o atual estado de um *script* (`smScriptOperStatus`) e inserir ou remover entradas na tabela `smScriptTable` (`smScriptRowStatus`). A manipulação desse último objeto possibilita o início de um *download* de um *script* de gerenciamento ou a remoção do mesmo no dispositivo alvo.

A tabela `smCodeTable` permite que um *script* seja instalado no dispositivo alvo através do método **push**, através de sucessivas operações SET do SNMP. Uma implementação da MIB Script pode optar por não suportar esse tipo de método, uma vez que ele é opcional (SCHÖNWÄLDER; QUITTEK; KAPPLER, 2000). A tabela possui três objetos que funcionam como chave primária: `smScriptOwner`, `smScriptName` e `smCodeIndex`. Os dois primeiros objetos são chaves estrangeiras que referenciam os objetos `smScriptOwner` e `smScriptName` definidos na tabela `smScriptTable`. O objeto `smCodeIndex` é a chave primária da tabela `smCodeTable`. O objeto `smCodeText` é utilizado pelo gerente para enviar fragmentos de código de programação de um *script* de gerenciamento para o dispositivo alvo. Salienta-se que o código tem que ser suportado por algum ambiente de execução instalado no dispositivo alvo. Através da manipulação do objeto `smCodeRowStatus` é possível iniciar a transferência de um *script* pelo método **push**.

A tabela `smLaunchTable` possibilita a criação de perfis de execução para *scripts* de gerenciamento e, também, iniciar a execução dos mesmos. Ela possui como principais funcionalidades:

- Listar os *scripts* de gerenciamento que estão prontos para serem executados;
- Passar argumentos de execução aos *scripts*;
- Criar um perfil de execução para um *script*;
- Iniciar a execução de um ou mais *scripts*.

Por ser muito extensa, apenas os principais objetos da tabela `smLaunchTable` serão explicados aqui. Maiores informações referentes a tabela `smLaunchTable` podem ser consultadas na RFC 3165 (LEVI; SCHÖNWÄLDER, 2001). A tabela possui dois objetos que funcionam como uma chave primária composta: `smLaunchOwner` e `smLaunchName`. Os objetos `smLaunchScriptName` e `smLaunchScriptOwner` associam uma entrada existente na tabela `smLaunchTable` a uma entrada na tabela `smScriptTable`. Tais objetos contêm o nome de um *script* de gerenciamento e seu respectivo proprietário, respectivamente. Os objetos `smLaunchMaxRunning` e `smLaunchLifeTime` possibilitam configurar a execução dos *scripts*, permitindo, respectivamente, definir o número máximo

de instâncias de um mesmo *script* que poderá executar de forma concorrente, e o tempo total de execução de um *script*. O objeto `smLaunchArgument` permite que argumentos sejam passados a um *script* de gerenciamento.

Uma vez que todos os objetos necessários a execução de um ou mais *scripts* forem configurados corretamente pelo gerente, a execução pode ter início através da manipulação do objeto `smLaunchStart`. Logo após a manipulação desse objeto, uma nova entrada na tabela `smRunTable`, que será apresentada a seguir, é adicionada. Os objetos `smLaunchMaxCompleted` e `smLaunchExpireTime` servem para informar, respectivamente, o número máximo de *scripts* com a execução encerrada e o tempo que cada uma dessas execuções estará disponível para consulta na tabela `smRunTable`.

A tabela `smRunTable` permite consultar e controlar a execução dos *scripts* de gerenciamento, e, também, obter o resultado da execução dos mesmos. A tabela possui três objetos que funcionam como chave primária: `smLaunchOwner`, `smLaunchName` e `smRunIndex`. Os dois primeiros são chaves estrangeiras que estão relacionadas a tabela `smLaunchTable`. O objeto `smRunIndex` é a chave primária pertencente a tabela `smRunTable`. O valor do objeto `smRunArgument` é obtido, inicialmente, através do valor definido no objeto `smLaunchArgument`. O tempo inicial, final e máximo de execução de um *script* de gerenciamento podem ser consultados através dos objetos `smRunStartTime`, `smRunEndTime` e `smRunLifeTime`, respectivamente. Objetos relacionados ao término da execução dos *scripts* são:

- `smRunResultTime`: o tempo que o resultado parcial (se ainda em execução) ou final (término da execução) foi obtido;
- `smRunExitCode`: o código que o *script* de gerenciamento retornou ao terminar a sua execução;
- `smRunResult`: o resultado obtido pela execução do *script*;
- `smRunError`: o erro ocorrido durante a execução (caso tenha acontecido);
- `smRunErrorTime`: o tempo em que o erro ocorreu.

O estado de execução de um *script* pode ser consultado através da leitura do objeto `smRunState`, e pode ser alterado através da manipulação do objeto `smRunControl`. Por fim, o tempo de vida de uma entrada na tabela `smRunTable` é definida pelo objeto `smRunExpireTime`, sendo seu valor obtido através do objeto `smLaunchExpireTime` definido na tabela `smLaunchTable`. Porém, esse valor poder ser alterado através da manipulação do próprio objeto `smRunExpireTime`.

Maiores informações sobre a estrutura da MIB Script podem ser encontradas na RFC 3165 (LEVI; SCHÖNWÄLDER, 2001).

2.3 Web Services

A tecnologia de Web Services (WS) (CURBERA et al., 2002) tem se mostrado bastante promissora na área de gerenciamento de redes. Embora ela tenha sido desenvolvida originalmente para suportar processos de comércio eletrônico, ela pode também ser utilizada como uma ferramenta de integração em diversos sistemas, tais como redes de suporte a *grids* (GOTH, 2002), inteligência artificial (PREECE; DECKER, 2002) e, recentemente, gerenciamento de redes.

Os Web Services podem ser descritos como um conjunto de componentes independentes disponibilizados na Internet utilizando protocolos Web (e.g HTTP, SMTP e FTP) e que recebem invocações de serviços de clientes (ROY; RAMANUJAN, 2001). Os clientes de um WS podem ser aplicações de usuários finais ou mesmo outros WS. Nesse último caso, um WS pode requisitar operações de outro WS, permitindo construir uma hierarquia de requisições. Além disso, através de mecanismos conhecidos como orquestração e coreografia de WS, é possível a criação de WS bastantes complexos baseados na invocação de outros mais simples (CURBERA et al., 2003). A arquitetura geral de WS (Figura 2.6) é composta por, pelo menos, três elementos principais: Registro, Provedor de serviços e Cliente.

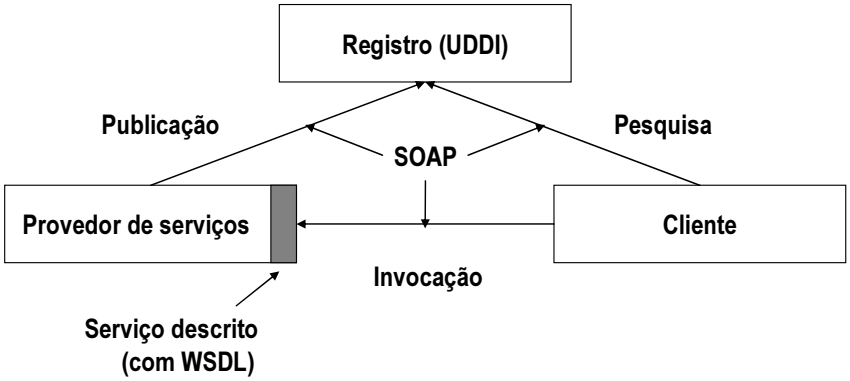

Figura 2.6: Arquitetura geral de Web Services

Para implementar os elementos dessa arquitetura, as principais tecnologias utilizadas, atualmente, são: UDDI (*Universal Description, Discovery, and Integration*) (BELLWOOD; CLÉMENT; RIEGEN, 2003), que tem a função de atuar como o registro dos WS; WSDL (*Web Services Description Language*) (CHRISTENSEN et al., 2001), um padrão para descrição dos WS; e SOAP (*Simple Object Access Protocol*) (GUDGIN et al., 2003), um protocolo baseado em XML utilizado para comunicação entre os elementos da arquitetura WS. Analisando mais detalhadamente, UDDI é um serviço de diretório emergente que trabalha como um repositório de dados para registrar e armazenar descrições de WS e informações de localização. O próprio registro UDDI é implementado como um WS, cujos serviços oferecidos são o cadastro e a pesquisa no repositório de dados. Dessa forma, clientes exploram o repositório UDDI, pesquisando por operações e WS disponíveis. Uma vez encontrado o WS apropriado, o cliente contata esse WS, invocando suas operações. Uma das informações que podem ser encontradas no registro UDDI é a localização do arquivo de descrição do WS. De posse desse arquivo, o cliente pode aprender os detalhes dos serviços oferecidos pelo WS e criar formas de acesso aos mesmos em tempo de execução. WSDL é usada para este propósito, ou seja, descrever as informações necessárias para a realização de acesso a um determinado serviço. Todas as comunicações entre cliente e UDDI, WS e UDDI, e cliente e WS são executadas através do protocolo SOAP.

A flexibilidade, a concepção para computação distribuída e a facilidade de utilização oferecida pelos WS parecem ser mais atrativas que soluções de gerenciamento mais antigas, tais como SNMP, CMIP e CORBA. As características dos WS demandam pesquisas para saber se os WS estariam aptos em melhorar ou substituir as atuais tecnologias e soluções de gerenciamento. Assim, várias investigações com respeito a utilização de WS

na área do gerenciamento de redes estão em desenvolvimento (PAVLOU et al., 2004), (DREVERS; MEENT; PRAS, 2004) e (KLIE; STRAUSS, 2004).

Embora a tecnologia WS se mostre fascinante a primeira vista, a introdução de WS no gerenciamento de redes deve ser cautelosa e bem analisada. Questões de desempenho nos elementos gerenciados e o consumo de banda da rede com o tráfego imposto pela utilização de WS poderiam impedir uma solução de gerenciamento efetiva. Também, não é adequado acreditar que o suporte a WS será encontrado em todos os elementos do processo de gerenciamento de rede. Por exemplo, dispositivos baseados em SNMP certamente ainda estarão presentes nas redes futuras. Dessa forma, é plausível acreditar que o cenário de gerenciamento baseado em WS mais comum será aquele onde as tecnologias estabelecidas de gerenciamento (como o SNMP) coexistirão com WS. Na seção a seguir será visto que isso é possível através da utilização de *gateways*.

2.4 *Gateways* WS para SNMP

Para usar a arquitetura de WS no gerenciamento de redes, processos de tradução devem ser introduzidos. Esses processos são necessários para traduzir as informações de gerenciamento obtidas através dos protocolos estabelecidos em informações oferecidas via WS. Uma maneira comum de implementar esses processos de tradução é utilizando *gateways* de protocolos nos sistemas de gerenciamento.

Yoon-Jung Oh et. al. (YOON-JUNG et al., 2002) definem *gateways* XML para SNMP e três métodos de tradução interativa: baseadas em DOM (*Document Object Model*) (APPARAO et al., 1998), em HTTP e em SOAP. Nas traduções baseadas em DOM, um gerente com suporte a XML chama uma interface DOM residente no *gateway*. Tais chamadas são traduzidas em operações SNMP entre o *gateway* e o dispositivo alvo. Na tradução baseada em HTTP, o *gateway* recebe expressões XPath (*XML Path Language*) e XQuery (*XML Query Language*) codificadas por um gerente com suporte a XML. Essas expressões são então traduzidas para requisições SNMP. Esse método de tradução permite que a filtragem de informação possa ser executada diretamente no *gateway*, reduzindo o conjunto de informações de gerenciamento entre o gerente com suporte a XML e o *gateway*, embora uma sobrecarga de processamento seja introduzida. Finalmente, na tradução baseada em SOAP, o *gateway* oferece serviços mais sofisticados, que são acessados pelo gerente com suporte a XML. Nesses serviços, o gerente pode pesquisar informações usando XPath ou prosseguir com consultas complexas através de expressões XQuery.

Strauss e Klie (KLIE; STRAUSS, 2004) propuseram um *gateway* XML para SNMP similar ao método de tradução de Yoon-Jung Oh et. al. O *gateway* aceita mensagens HTTP com expressões XPath na URL. As expressões são então verificadas e traduzidas para mensagens SNMP. DOM é usado para acessar os documentos XML dentro dos *gateways*, reduzindo os dados transferidos entre o gerente e o *gateway*. Em operações de escrita, mensagens POST são traduzidas para requisições SNMP `SetRequest`. *Traps* SNMP são suportadas dentro do *gateway* através de um buffer para *traps* que é acessado pelo gerente. Nesse processo, mensagens POST são enviadas pelo *gateway* para receptores (*listeners*) HTTP nos gerentes baseados em XML.

Em um trabalho desenvolvido no Grupo de Redes do Instituto de Informática da UFRGS, foi implementado um sistema (NEISSE et al., 2003) que, dado um arquivo SMI (*Structure of Management Information*) (ROSE; MCCLOGHRIE, 1990) de uma MIB, cria automaticamente *gateways* XML para SNMP. Os *gateways* criados consultam informações nos dispositivos e geram documentos XML, que são enviados de volta para o

gerente, onde são analisados através de parsers. Como no trabalho anterior de Strauss e Klie, a tradução é executada com a ajuda da ferramenta `smidump` (KLIE; STRAUSS, 2004), a qual gera uma versão XML das MIBs.

Como verificado, *gateways* são criados para acessar dispositivos baseados em SNMP e exportar informações de gerenciamento em documentos XML (WS reais, baseados em SOAP, dificilmente são usados). Adicionalmente, a descrição de WS através de WSDL e seu registro em UDDI não são abordados nesses trabalhos. Na próxima seção, serão apresentadas duas abordagens para *gateways* SNMP, que melhor exploram as facilidades introduzidas pela arquitetura de WS.

2.4.1 *Gateways* WS para SNMP em nível de protocolo

O *gateway* WS para SNMP em nível de protocolo (SCHÖNWÄLDER; PRAS; MARTIN-FLATIN, 2003) fornece operações que são mapeamentos diretos das primitivas SNMP. Um gerente baseado em WS requisita informações de gerenciamento acessando o *gateway* WS para SNMP através de mensagens SOAP. Já os servidores que hospedam os *gateways*, recebem do gerente, a identificação da operação a ser acessada (e.g. Get ou Set) e uma lista dos parâmetros relacionados ao SNMP (o endereço do dispositivo alvo, uma comunidade SNMP válida e OIDs SNMP). Com essas informações, a operação apropriada é invocada dentro do *gateway* WS e o dispositivo alvo é acessado via SNMP.

Em um *gateway* desenvolvido pelo Grupo de Redes da UFRGS (NEISSE et al., 2004) foram implementadas as operações Get, GetNext e Set, que geram, para cada requisição do gerente, exatamente uma requisição SNMP do *gateway* para o dispositivo alvo, e exatamente uma resposta do dispositivo alvo para o *gateway*. Após as informações SNMP serem obtidas do dispositivo, o *gateway* monta uma mensagem SOAP com tais informações e envia essa mensagem de volta para o gerente. A Figura 2.7 mostra um exemplo de interações em um cenário onde um *gateway* em nível de protocolo está presente e um gerente WS precisa saber os nomes dos *scripts* de gerenciamento disponibilizados no dispositivo alvo.

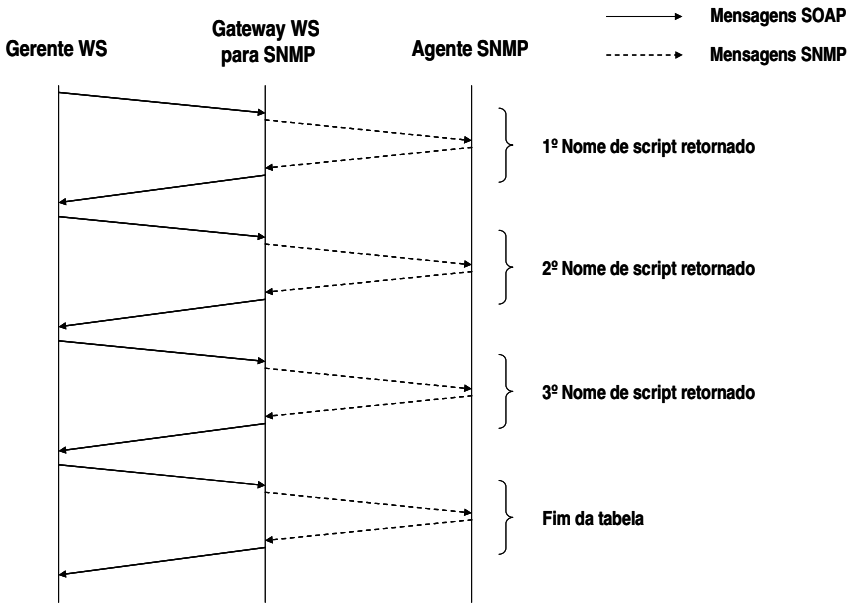

Figura 2.7: Exemplo do funcionamento de um *gateway* em nível de protocolo

Nesse exemplo, considera-se que existam três *scripts* de gerenciamento no dispositivo

alvo. Para descobrir o total de *scripts* existente, o gerente precisa percorrer as instâncias de uma tabela até que uma resposta diferente da esperada pelo gerente seja retornada pelo agente, indicando com isso que o final da tabela foi alcançado. Logo, nesse exemplo, quatro interações são necessárias para se descobrir que existem três *scripts* de gerenciamento no dispositivo alvo.

Em um uso simples desse *gateway*, suas operações (Get, GetNext e Set) foram descritas em WSDL e registradas em um repositório UDDI. Um gerente baseado em WS, procurando por WS de gerenciamento, pode pesquisar no UDDI e descobrir a localização dos *gateways* disponíveis. É importante notar que, com esta abordagem, o gerente baseado em WS ainda deve estar ciente dos OIDs SNMP para, corretamente, requisitar as instâncias dos objetos ao *gateway*. A vantagem dessa abordagem, entretanto, é o fato de que, toda vez que novos objetos passam a ser suportados por um agente SNMP, o *gateway* WS para SNMP não precisará ser alterado. A desvantagem, por outro lado, vem do fato do gerente ser obrigado a conhecer os objetos SNMP suportados em cada dispositivo gerenciado, mesmo na presença de UDDI. Além disso, o gerente ainda precisa lidar com OIDs SNMP, pois tais *gateways* não possibilitam operações de uso mais fácil do que aquelas encontradas atualmente usando SNMP.

2.4.2 *Gateways* WS para SNMP em nível de objeto

Um *gateway* WS para SNMP em nível de objeto (NEISSE et al., 2004), diferentemente do apresentado anteriormente, "conhece" os objetos de MIB suportados pelo dispositivo alvo, e apresenta tais objetos como operações de WS. Por exemplo, uma operação `GetIfTable` é uma operação que obtém a tabela completa de interfaces, enquanto que `SetAdminStatus` é uma operação que muda, no dispositivo alvo, o estado administrativo de uma das interfaces de rede disponíveis.

Uma vantagem do *gateway* em nível de objeto é que ele não apenas consulta e expõe as informações como um WS, mas também possui somente as operações permitidas para serem executadas sobre o dispositivo alvo. Uma outra vantagem está relacionada a forma que a base de informações de gerenciamento é consultada. No caso do *gateway* em nível de objeto, o próprio *gateway*, e não mais o gerente, é responsável por controlar as interações com o agente SNMP para obter todas as instâncias de uma tabela, construir as mensagens de resposta SOAP e enviá-las para o gerente WS. Esse controle de interação, que é realizado pelo gerente no caso de utilização do *gateway* em nível de protocolo, é então movido para o *gateway* em nível de objeto, introduzindo com isso um certo nível de controle no *gateway*.

Comparando com a Figura 2.7, a Figura 2.8 mostra que o número de mensagens trocadas entre o gerente WS e o *gateway* é menor quando se utiliza o *gateway* em nível de objeto. Na Figura 2.8-a, ocorre a solicitação para que todos os nomes dos *scripts* de gerenciamento sejam retornados. A Figura 2.8 adicionalmente mostra as interações necessárias para ordenar, no agente SNMP, o *download* (de um servidor que não está presente na figura) de um *script* de gerenciamento (Figura 2.8-b) e, também, solicitar o estado atual do *download* (Figura 2.8-c). O exemplo da Figura 2.8 lida com três objetos diferentes definidos na MIB Script: a lista de *scripts* de gerenciamento disponíveis (`smScriptName`), um objeto para informar a localização (`smScriptSource`) de um *script* de gerenciamento, a partir do qual o agente SNMP deve realizar o *download*, e um objeto que informa o estado atual do *download* (`smScriptAdminStatus`).

A abordagem apresentada pelo *gateway* em nível de objeto perde flexibilidade quando o agente SNMP do dispositivo alvo é alterado (tanto para incluir como para remover

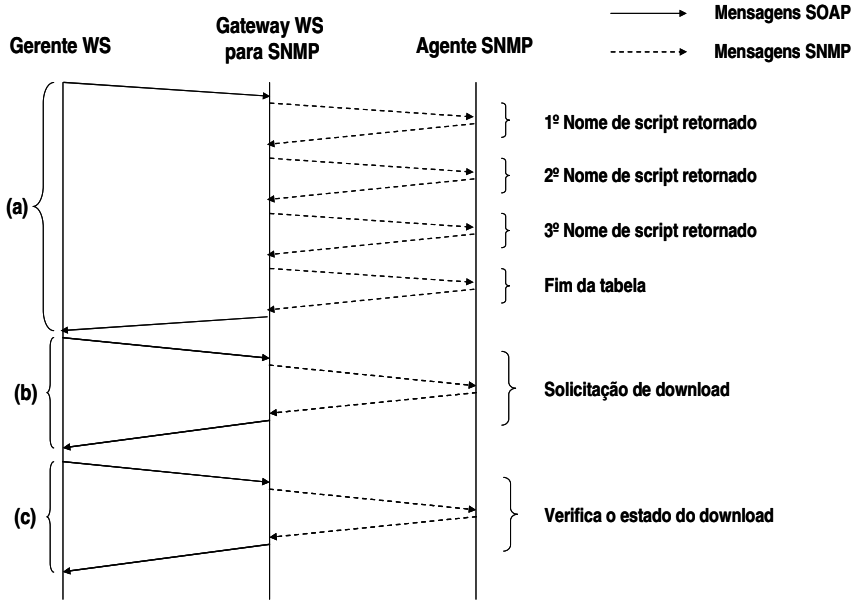

Figura 2.8: Exemplo do funcionamento de um *gateway* em nível de objeto

objetos). Nesse caso, o WS associado precisa, de fato, ser refeito para refletir as mudanças do agente SNMP. Portanto, necessita-se de uma maneira eficiente de criação de *gateways* WS para SNMP em nível de objeto. No trabalho desenvolvido por Neisse et. al. (NEISSE et al., 2004), foi desenvolvido um sistema que, dado um arquivo de MIB SMI, cria um novo WS. A Figura 2.9 apresenta a arquitetura que suporta a criação de novos *gateways* WS para SNMP em nível de objeto.

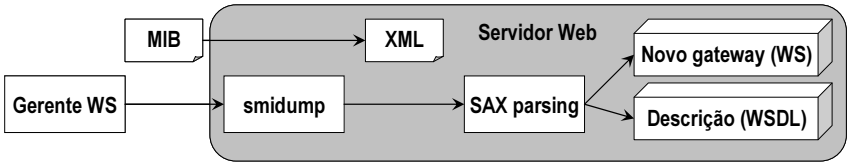

Figura 2.9: Passos para a criação de um novo *gateway* WS para SNMP em nível de objeto

Internamente no servidor, a ferramenta `smidump` verifica a MIB passada e, na ausência de inconsistências, gera um XML temporário (que é a versão XML da MIB). No próximo passo, é feito um *parsing* do documento XML temporário para construir o novo *gateway*. Cada nó da árvore da MIB original é transformado em operações WS. Nessas operações, está incluído código para contactar, via SNMP, o dispositivo alvo. O dispositivo alvo e sua *string* de comunidade são tratados dentro do código como parâmetros, cujos valores serão posteriormente passados, quando a operação for invocada pelo gerente. O WS recém criado é armazenado em um diretório padrão no servidor Web e disponibilizado para ser invocado logo após sua criação. O arquivo de MIB original recebido também é armazenado em outro diretório padrão, para fins de documentação, bem como o documento XML intermediário gerado pela ferramenta `smidump`. Ao mesmo tempo em que a etapa de *parsing* cria o código do novo WS, ela também cria o documento WSDL que descreve o WS criado. Além disso, o gerente baseado em WS que requisitou a criação de um novo serviço, opcionalmente, informa a URL do repositório UDDI onde espera que o WS criado seja registrado.

2.5 Definição do problema

Como mencionado na introdução, embora a tecnologia Web Services apresente-se como uma interessante alternativa para o gerenciamento de dispositivos de rede, ela precisa ser bem investigada para que a utilização da mesma no gerenciamento de redes possa tornar-se viável. É visto também que uma substituição completa do SNMP em prol da utilização de WS não é factível nas redes existentes atualmente. A utilização de *gateways* que convertem operações WS para SNMP e vice-versa apresenta-se como uma alternativa mais aceitável.

Atualmente, a utilizaçao de *gateways* para integração de SNMP e WS vem sendo bastante pesquisada pela comunidade de gerenciamento de redes. Tais investigações, entretanto, são caracterizadas por serem extremamente genéricas, não levando em consideração casos específicos no gerenciamento que poderiam revelar aspectos importantes da utilização de WS no gerenciamento de redes. Sendo assim, este trabalho objetiva realizar uma investigação mais específica sobre a utilização de *gateways* WS para SNMP no gerenciamento de redes.

Além dessa investigação, este trabalho acredita que protocolos antigos de gerenciamento, tal como o SNMP, possam ser substituídos por novas tecnologias, como, por exemplo, a tecnologia de Web Services. Com uma introdução gradual dos Web Services no gerenciamento de redes, acredita-se que ela possa apresentar-se como uma tecnologia apta a substituir tais protocolos em um futuro próximo. Logo, este trabalho apresenta dois protótipos baseados na tecnologia de Web Services, sendo eles: um *gateway* WS para SNMP em nível de serviço e um agente orientado a serviços baseado em Web Services. Ambos construídos no âmbito do gerenciamento por delegação.

No próximo capítulo serão mostrados os dois protótipos desenvolvidos, sendo apresentadas as suas características e funcionalidades. No capítulo 5 será apresentada a avaliação dessas duas contribuições com outras duas tecnologias pesquisadas: o *gateway* em nível de objeto e o Jasmin.

3 GERENCIAMENTO DE REDES ORIENTADO A SERVIÇOS UTILIZANDO WEB SERVICES

Conforme mencionado na introdução, a utilização de Web Services no gerenciamento de redes tende a acontecer de uma maneira gradual. Em uma primeira etapa, tanto WS quanto SNMP continuarão sendo usados através da utilização de *gateways*, cuja função principal é a de fornecer traduções entre WS e SNMP de forma a integrar essas duas tecnologias. Conforme visto no capítulo anterior, os *gateways* investigados apresentam diferenças na maneira que eles realizam a tradução de operações WS para SNMP, requisitadas pelo gerente. No caso do *gateway* em nível de protocolo, a tradução de operações é feita através do mapeamento direto das primitivas SNMP para operações WS. Nesse tipo de *gateway*, o gerente deve informar a operação a ser realizada e, também, os parâmetros relacionados ao SNMP. No caso do *gateway* em nível de objeto, os objetos de um módulo de MIB são mapeados para operações SNMP, não sendo mais necessário o envio de OIDs SNMP como é no caso do *gateway* em nível protocolo.

O problema de ambos os *gateways* apresentados anteriormente é que os mesmos ainda obrigam o gerente a saber detalhes de quais objetos de uma MIB precisam ser manipulados para que uma determinada ação seja realizada (e.g. o *download* de um *script* de gerenciamento). Partindo de uma visão diferente, o *gateway* em nível de serviço, a primeira contribuição deste trabalho, é construído tomando por base os serviços (funcionalidades) que um determinado módulo de MIB oferece. Isso permite um grau maior de abstração, por parte do gerente, dos detalhes de manipulação dos objetos de uma MIB. Porém, uma vez que ele não é baseado na estrutura de um módulo de MIB, ele não pode ser construído automaticamente por um "gerador" de *gateways*, como é no caso do *gateway* em nível de objeto.

Em uma segunda etapa, o agente SNMP normalmente empregado em um dispositivo de rede gerenciado é substituído pelos WS. Essa substituição do SNMP permite que o termo "agente WS" seja utilizado para referenciar um WS de gerenciamento, que é encontrado dentro de um dispositivo de rede. Nesse contexto, a segunda contribuição desta dissertação é o desenvolvimento de um agente WS para o suporte ao gerenciamento por delegação. O agente WS desenvolvido neste trabalho é similar ao *gateway* em nível de serviço no que se refere a interface dos serviços oferecidos. A diferença substancial entre um e outro é a maneira que os serviços são implementados. No caso da implementação dos serviços de um agente WS, a existência de um *gateway* não é mais necessária, uma vez que traduções de operações SNMP para WS e vice-versa não ocorrem, devido ao fato que não existe mais integração de tecnologias diferentes, como é no caso dos *gateways* apresentados neste trabalho.

Neste capítulo serão apresentadas as duas contribuições deste trabalho, tendo o ge-

renciamento por delegação como estudo de caso. Inicialmente, o *gateway* em nível de serviço será apresentado, sendo descrito, em seguida, um agente orientado a serviços baseado em Web Services. Por fim, um mapeamento das funcionalidades da MIB Script para operações WS será apresentada, sendo feita, nessa apresentação, uma comparação do acesso a essas funcionalidades entre um gerente SNMP e um gerente WS.

3.1 *Gateway* WS para SNMP em nível de serviço

O *gateway* WS para SNMP em nível de serviço fornece operações que são baseadas nos serviços que um determinado módulo de MIB oferece. Portanto, é válido dizer que um *gateway* em nível de serviço é específico para cada módulo de MIB, uma vez que ele é criado conforme as características de funcionamento de cada módulo. Isso difere, por exemplo, do *gateway* em nível de protocolo apresentado no capítulo anterior, que é um *gateway* genérico, no sentido que ele é construído para permitir o acesso a qualquer módulo de MIB, uma vez que ele fornece operações WS que são mapeamentos diretos para solicitações SNMP.

A forma que o *gateway* em nível de serviço é acessado pelo gerente é similar aos demais *gateways* apresentados. Um gerente baseado em WS solicita informações de gerenciamento, acessando o *gateway* através de mensagens SOAP que trafegam pela Web usando protocolos, tais como HTTP ou SMTP. Entretanto, o funcionamento de um *gateway* WS para SNMP em nível de serviço difere dos demais *gateways* vistos anteriormente. Em um *gateway* em nível de serviço, as operações oferecidas são agrupamentos de objetos de um módulo de uma MIB, que, quando manipulados de uma forma conjunta, realizam uma determinada ação.

Para melhor entender o funcionamento de um *gateway* em nível de serviço, um exemplo comparativo de manipulação em um módulo de MIB será apresentado. Primeiramente, através da manipulação direta dos objetos do mesmo por um gerente SNMP, e após, através da utilização das operações de um *gateway* em nível de serviço por um gerente WS. A MIB Script apresentada na seção 2.2 será utilizada como exemplo.

A MIB Script permite, dentre outras operações, o *download*, a partir do dispositivo gerenciado, de um *script* de gerenciamento localizado em um repositório remoto qualquer, e uma posterior execução do mesmo. Para fins de apresentação do caso de manipulação direta dos objetos pelo gerente SNMP, o cenário apresentado na Figura 3.1 será tomado como exemplo.

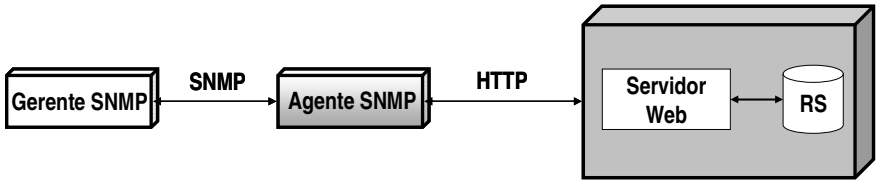

Figura 3.1: Cenário da manipulação direta dos objetos da MIB Script por um gerente SNMP

Nesse cenário, tem-se o gerente SNMP que irá solicitar ao agente SNMP a transferência, através do método **pull**, de um *script* de gerenciamento localizado em um repositório de *scripts* (RS). A transferência é feita utilizando-se o protocolo HTTP. Após o *script* ter sido transferido, o gerente solicitará a execução do mesmo. Para a realização das ações

descritas anteriormente, o gerente necessitaria realizar os passos demonstrados na Figura 3.2.

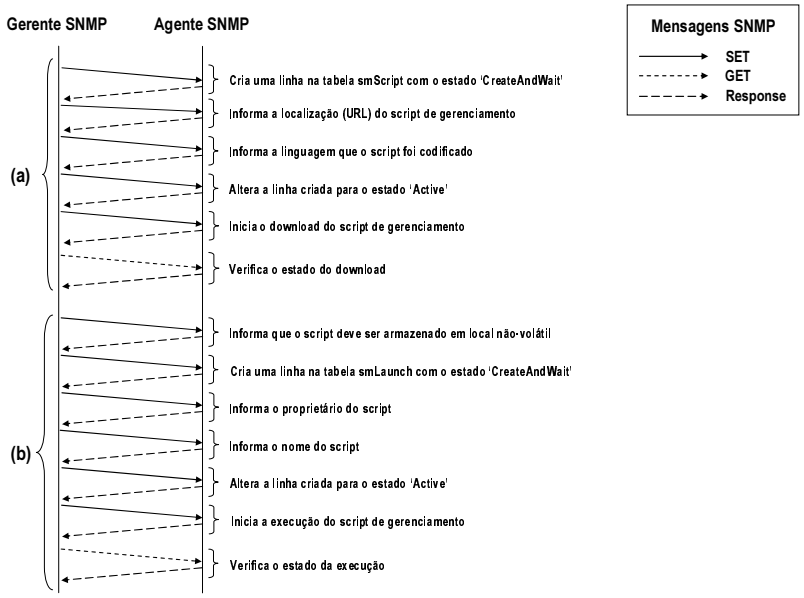

Figura 3.2: Interação entre o gerente e o agente SNMP para o *download* e execução de um *script* de gerenciamento

Numa primeira etapa, para a realização do *download* de um *script* de gerenciamento (Figura 3.2-a), o gerente SNMP precisa informar ao agente SNMP que o *download* de um *script* será realizado, atribuindo ao objeto `smScriptRowStatus` o valor 'CreateAndWait'. Antes de o *download* iniciar, os objetos `smScriptSource` e `smScriptLanguage` devem ser alterados para informar a localização remota do *script* e a linguagem que ele está codificado, respectivamente. Uma vez que tais objetos forem alterados, o gerente ordena o início do *download* atribuindo ao objeto `smScriptRowStatus` o valor 'Active'. Após esse passo, o *download* tem início. Para saber o estado atual do *download* do *script*, o gerente deve ler o valor do objeto `smScriptOperStatus`, podendo somente iniciar a utilização do *script* se o valor do objeto for 'enabled', o que informa que o *download* encerrou sem problemas. Qualquer outro valor que não for 'enabled', informa que ocorreram erros na realização do *download*. Uma vez que o gerente tenha lido o valor 'enabled' no objeto `smScriptOperStatus`, ele tem a opção de armazenar o *script* em um local não-volátil.

Terminados os passos referentes ao *download*, na segunda etapa o gerente ordena a execução do *script* de gerenciamento (Figura 3.2-b). Para isso, ele informa ao agente SNMP que a execução de um *script* ocorrerá, atribuindo ao objeto `smLaunchRowStatus` o valor 'CreateAndWait'. A partir desse momento, parâmetros de execução podem ser passados para o *script*. No exemplo apresentado na Figura 3.2-b, apenas o nome do *script* e o seu respectivo proprietário foram fornecidos pelo gerente, como parâmetros de execução. Antes de ser ordenada a execução do *script*, o objeto `smLaunchRowStatus` deve ser modificado para 'active'. Feito isso, o início da execução do *script* ocorre com o objeto `smLaunchStart` sendo alterado. Para verificar o estado da execução do *script*, o gerente deve ler o valor do objeto `smRunState`, que pode possuir os valores: 'executing', 'suspended' ou 'terminated'. Quando o valor do objeto for 'terminated', isso indicará o final da execução do *script*.

Agora, o cenário apresentado na Figura 3.3 será tomado como exemplo para demonstrar a utilização do *gateway* em nível de serviço. A figura apresenta basicamente os mesmos elementos da Figura 3.1, com exceção do gerente SNMP que foi substituído por um gerente baseado em WS, e também, com o acréscimo de um novo elemento, o *gateway* WS para SNMP em nível de serviço.

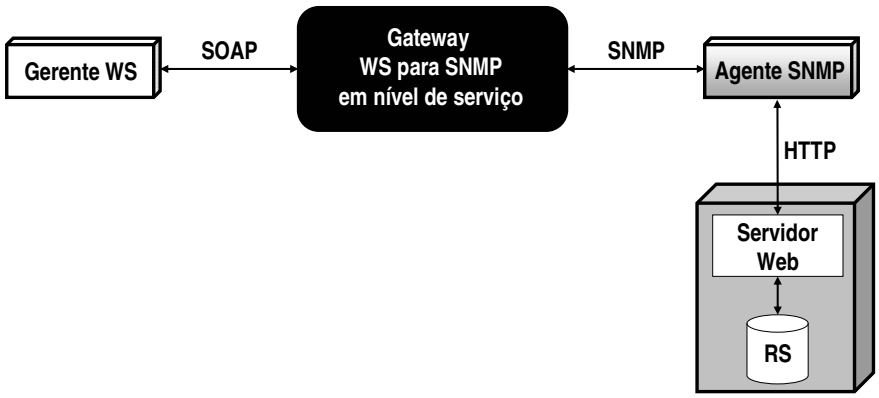

Figura 3.3: Cenário da interação entre gerente WS, *gateway* em nível de serviço e agente SNMP

No caso do *gateway* em nível de serviço, todas as funcionalidades da MIB Script são disponibilizadas pelo *gateway* através de operações WS, previamente definidas e implementadas pelo desenvolvedor do *gateway*. Os detalhes de manipulação dos objetos de um módulo de uma MIB são abstraídos da visão do gerente, sendo tais detalhes somente necessários no momento do desenvolvimento do *gateway*, e não mais no momento de utilização dos mesmo, como é no caso dos *gateways* em nível de protocolo e de objeto. Logo, para realizar a mesma operação de *download* e execução de um *script* descrita anteriormente, o gerente necessitaria realizar os passos apresentados na Figura 3.4.

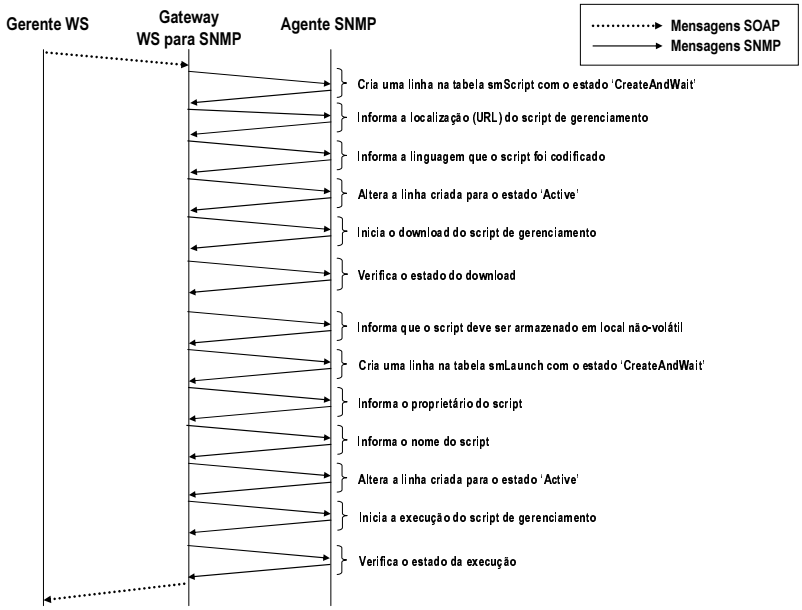

Figura 3.4: Exemplo do funcionamento de um *gateway* em nível de serviço

Se comparado aos passos realizados no primeiro exemplo (Figura 3.2), nesse segundo

exemplo (Figura 3.4) o gerente precisaria somente invocar uma única operação WS fornecida pelo *gateway* em nível de serviço que poderia ser denominada, por exemplo, `DownloadAndRunScript($url,$language,$owner,$script_name)`. Essa operação solicitada pelo gerente faz com que o *gateway* em nível de serviço ordene o agente SNMP a realização do *download* e a execução de um *script* de gerenciamento, sendo que após o término da execução, o resultado é retornado. Nota-se que não é mais necessário ao gerente saber quais objetos da MIB Script precisam ser manipulados para realizar tal ação, como é no caso do primeiro exemplo. Vale salientar também que, dependendo da forma com que a operação WS é implementada pelo desenvolvedor do *gateway*, ela pode agir de uma maneira bloqueante (como é no caso do segundo exemplo). Em uma situação bloqueante, o gerente fica bloqueado ao invocar uma operação WS, aguardando o término da mesma. Por outro lado, se uma operação WS for não-bloqueante, após a invocação, o gerente WS não fica bloqueado aguardando o término da operação WS invocada. Porém, para que ele possa saber o resultado da operação WS solicitada, ele necessitaria invocar uma outra operação, denominada, por exemplo, `GetExecutionResults()`.

As principais vantagens apresentadas pelo *gateway* em nível de serviço são a simplicidade de utilização, que está diretamente relacionada a outra vantagem, que é a abstração dos detalhes de manipulação de uma MIB. Como todos os detalhes de manipulação de objetos de uma MIB estão definidos na implementação de uma operação WS fornecida pelo *gateway*, o gerente WS precisa somente invocar tal operação WS com os parâmetros requeridos por ela, para a realização de uma determinada ação. Isso acaba simplificando a maneira que os dispositivos de rede são gerenciados.

Como desvantagens existentes, têm-se a necessidade de construção manual do *gateway* para cada módulo de MIB que se deseja manipular e a falta de flexibilidade na manipulação dos objetos da MIB, uma vez que os objetos são acessados indiretamente através da invocação de operações WS e de seus respectivos parâmetros. Dependendo da forma com que o *gateway* é desenvolvido, ele pode apresentar um grau de flexibilidade bastante satisfatório. Isso pode ocorrer, por exemplo, se existir uma grande variedade de operações WS para manipular os objetos de uma MIB. Entretanto, se o número de operações WS para realizar determinada ação for muito elevado, o *gateway* em nível de serviço acaba perdendo a característica de simplicidade de manipulação.

3.2 Um agente orientado a serviços baseado em Web Services

Como visto na seção anterior, para que um agente SNMP existente em um dispositivo gerenciado possa ser manipulado por um gerente WS, são necessárias conversões de operações WS para SNMP e vice-versa, sendo tais conversões realizadas pelos *gateways*. Essa maneira de gerenciamento acaba impondo um tempo de resposta maior por parte dos dispositivos gerenciados, pois todas as operações de gerenciamento invocadas pelo gerente precisam ser traduzidas pelo *gateway* (FIOREZE et al., 2005).

Embora uma solução inicial seja a remoção dos *gateways* e a total utilização do SNMP para o gerenciamento de dispositivos, essa forma de gerenciamento não é muito adequada devido às limitações existentes na utilização do SNMP. Por exemplo, uma vez que o tráfego SNMP é normalmente bloqueado por *firewalls*, isso poderia impedir o gerenciamento de dispositivos externos à rede a qual o gerente SNMP está localizado. Uma solução mais promissora seria a total utilização de WS no gerenciamento de dispositivos, sendo necessário portanto a construção de gerentes e agentes totalmente baseados em WS.

Como segunda contribuição desta dissertação, um agente WS que disponibiliza os

serviços oferecidos pela MIB Script foi desenvolvido. As interfaces para as operações oferecidas pelo agente WS são similares as interfaces oferecidas pelo *gateway* em nível de serviço. Porém, como o protocolo SNMP foi substituído pela tecnologia WS, não são mais necessárias traduções de operações, como ocorria nos *gateways* apresentados. O agente WS disponibiliza os serviços oferecidos pela MIB Script através de operações WS, sendo estas invocadas pelo gerente WS.

Seguindo o exemplo apresentado na seção 3.1 onde um *script* de gerenciamento era executado após o seu *download*, no caso da tecnologia WS ser totalmente utilizada na realização de tais ações, tem-se o seguinte cenário (Figura 3.5).

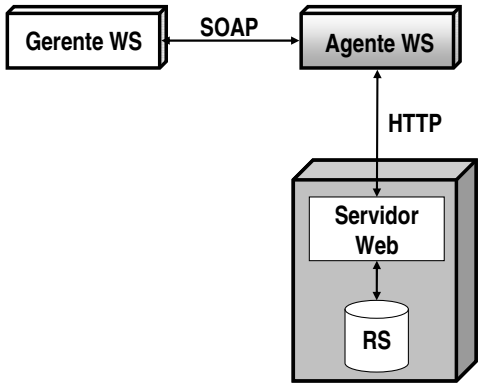

Figura 3.5: Gerenciamento de redes baseado em Web Services

Nesse cenário, a arquitetura SNMP é completamente substituída pela tecnologia de Web Services. Logo, para o gerente WS solicitar ao agente WS a realização do *download* de um *script* de gerenciamento e sua posterior execução, ele necessitaria realizar os passos apresentados na Figura 3.6.

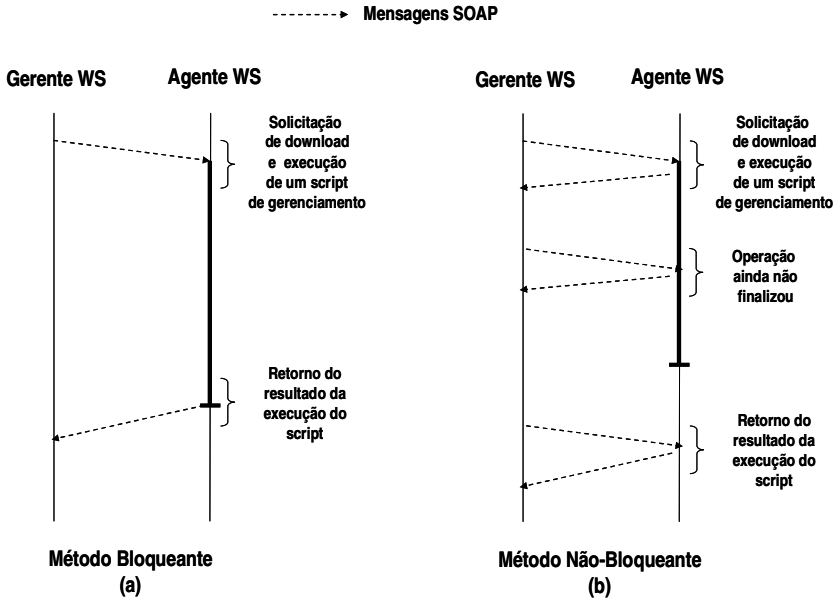

Figura 3.6: Exemplo do funcionamento de um agente orientado a serviços baseado em Web Services

Na Figura 3.6, a invocação, pelo gerente WS, de uma operação WS, que poder ser chamada de `DownloadAndRunScript($url,$language,$owner,$script_name)`, é

similar à invocação realizada pelo gerente WS apresentada nos passos descritos no *gateway* em nível de serviço. Como mencionada na seção 3.1, em uma situação bloqueante (Figura 3.6-a) o gerente WS bloquearia esperando o término da operação invocada para saber o resultado. Entretanto, ficar bloqueado aguardando o término de uma operação, deixaria o gerente WS impossibilitado de realizar outras ações de gerenciamento. Logo, para evitar esse tipo de situação, é importante que o agente WS desenvolvido possibilite o gerente WS não ficar bloqueado. Atuando de uma maneira não-bloqueante (Figura 3.6-b), o agente WS atenderia a solicitação feita pelo gerente, e permitiria que ele não ficasse esperando pelo término da mesma. No exemplo apresentado, o gerente solicitaria o *download* e a execução de um *script*, e voltaria a realizar outras ações de gerenciamento. De tempos em tempos, ele pode consultar o agente para saber se o *script* terminou a sua execução. Isso irá acontecer até o momento que *script* terminar a sua execução e o resultado da execução estiver disponível para o gerente consultar.

A implementação interna das operações WS são diferentes das existentes nos *gateways* em nível de serviço. Neste último, o desenvolvedor do *gateway* necessita saber quais conjuntos de objetos de um módulo de MIB precisam ser combinados para que uma determinada ação seja realizada. Já no caso do agente WS isso não é mais necessário, pois o SNMP não é mais utilizado. O desenvolvedor do agente WS tem a liberdade de implementar as operações WS como ele julgar mais conveniente. Salienta-se aqui, que, tanto no desenvolvimento do *gateway* em nível de serviço quanto no do agente WS, o desenvolvedor pode definir, conforme lhe convir, a quantidade de serviços de um determinado módulo de uma MIB.

3.3 Mapeamento dos serviços da MIB Script para operações Web Services

Como visto nas seções anteriores (3.1 e 3.2), o *gateway* em nível de serviço e o agente WS são construídos baseados nos serviços que um determinado módulo de MIB fornece através da manipulação dos seus objetos. Nesta seção, a estrutura da MIB Script será tomada como base para demonstrar uma possível forma de mapeamento dos serviços da MIB Script para operações WS. Salienta-se que outras maneiras de mapeamento são possíveis, pois elas são baseadas no julgamento que o desenvolvedor do *gateway* em nível de serviço ou do agente WS tiver no momento do desenvolvimento dos mesmos.

Os mapeamentos que serão vistos a seguir apresentam uma comparação entre os passos que um gerente SNMP e um gerente WS levariam para acessar um determinado serviço de gerenciamento fornecido por um agente SNMP e um agente WS, respectivamente. A MIB Script foi utilizada como caso de estudo e o mapeamento dos seus serviços será demonstrado da seguinte forma. A organização será em subseções, sendo apresentada em cada subseção, primeiramente, uma descrição do serviço oferecido pela MIB Script. Após isso, será mostrado como o serviço, implementado em PHP (PHP, 2001) no caso desta dissertação, seria acessado por um gerente SNMP e, posteriormente, por um gerente WS.

Analisando-se a MIB Script é possível descrever alguns dos principais serviços oferecidos pela mesma, sendo eles:

3.3.1 Consulta dos ambientes de execução suportados por uma implementação da MIB Script

Um gerente pode consultar um ou todos os ambientes de execução suportados por uma implementação da MIB Script. Caso o gerente desejasse consultar todos os ambientes, os seguintes passos são necessários:

- Gerente SNMP: passos necessários:

```
// snmpwalk (hostname, community, object)
$runtimes = snmpwalk ($agentIP, "secret", "smLangTable");
```

- Gerente WS: passos necessários:

```
//function GetLanguages($language="all")
$arguments = array('language'=>null);
$runtimes = manager->call("GetLanguages", $arguments);
```

3.3.2 *Download* de um *script* de gerenciamento

Uma das formas de transferência de *scripts* de gerenciamento é através do *download* dos mesmos (método **pull**). Caso o gerente desejasse que uma implementação da MIB Script realizasse o *download* de um *script*, os seguintes passos seriam necessários:

- Gerente SNMP: passos necessários:

```
//snmpset (hostname, community, object, type, value);
snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 5);
snmpset ($agentIP, "secret", "smScriptSource.$index", "s",
         "http://scriptrepository/ping.java");
snmpset ($agentIP, "secret", "smScriptDescr.$index", "s",
         "Script que verifica o estado dos roteadores da rede");
snmpset ($agentIP, "secret", "smScriptLanguage.$index", "i", 1);
snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 1);
snmpset ($agentIP, "secret", "smScriptAdminStatus.$index", "i", 1);
$operStatus="4";
while($operStatus != "1")
{
      $operStatus = snmpget($agentIP, "secret", "smScriptOperStatus.$index");
}
snmpset ($agentIP, "secret", "smScriptStorageType.$index", "i", 3);
```

- Gerente WS: passos necessários:

```
//function DownloadScript($URL, $Script_Name, $Language,
                          $Description, $Volatile_Storage_Type=false, $block=true)
$arguments =
array('URL'=>"http://scriptrepository/ping.java",
      'Script_Name'=>"ping_routers",
      'Language'=>1,
      'Description'=>"Script que verifica o estado dos roteadores da rede",
      'Volatile_Storage_Type'=>false);
$Script_ID = $manager->call("DownloadScript", $arguments);
```

3.3.3 *Upload* de um *script* de gerenciamento realizado pelo gerente

Outra forma de transferência de *scripts* de gerenciamento é através do *upload* dos mesmos pelo gerente (método **push**). Para transferir um *script* de gerenciamento para uma implementação da MIB Script, os seguintes passos seriam necessários:

- Gerente SNMP: passos necessários:

```
//snmpset (hostname, community, object, type, value);
snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 5);
snmpset ($agentIP, "secret", "smScriptSource.$index", "s", "");
snmpset ($agentIP, "secret", "smScriptLanguage.$index", "2", 5);
snmpset ($agentIP, "secret", "smScriptStorageType.$index", "i", 2);
snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 1);
snmpset ($agentIP, "secret", "smScriptAdminStatus.$index", "i", 3);
snmpset ($agentIP, "secret", "smCodeTable.$index", "s",
         "#!\bin\sh\n\necho -n \"Um exemplo de pushing de um script via SNMP\"");
snmpset ($agentIP, "secret", "smScriptAdminStatus.$index", "i", 1);
$operStatus="4";
while($operStatus != "1")
{
      $operStatus = snmpget($agentIP, "secret", "smScriptOperStatus.$index");
}
snmpset ($agentIP, "secret", "smScriptStorageType.$index", "i", 3);
```

- Gerente WS: passos necessários:

```
//function PushScript($Code, $Script_Name, $Language,
                      $Description, $Volatile_Storage_Type=false)
$arguments =
array('Code'=>"#!/bin/sh\n\necho -n \"Um exemplo de pushing de um script via WS\"",
      'Script_Name'=>"exemplo.sh",
      'Language'=>2,
      'Description'=>"Exemplo de pushing de um script utilizando uma operação WS",
      'Volatile_Storage_Type'=>false);
$Script_ID = $manager->call("PushScript", $arguments);
```

3.3.4 Remoção de um *script* de gerenciamento no dispositivo alvo

Uma vez que um *script* de gerenciamento está localizado em um dispositivo alvo, o gerente pode solicitar a sua remoção, caso o *script* não seja mais necessário. Para realizar essa ação, os seguintes passos seriam necessários:

- Gerente SNMP: passos necessários:

```
//snmpset (hostname, community, object, type, value);
snmpset ($agentIP, "secret", "smScriptAdminStatus.$index", "i", 2);
$operStatus="1";
while($operStatus != "2")
{
      $operStatus = snmpget($agentIP, "secret", "smScriptOperStatus.$index");
}
snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 6);
```

- Gerente WS: passos necessários:

```
//function RemoveScript($Script_ID)
$arguments = array('Script_ID'=>4);
$status = $manager->call("RemoveScript", $arguments);
```

3.3.5 Consulta de informações de *scripts* de gerenciamento localizado no dispositivo alvo

Informações referentes a *scripts* de gerenciamento localizado em um dispositivo alvo podem ser consultadas pelo gerente, através dos passos a seguir:

- Gerente SNMP: passos necessários:

```
// snmpwalk (hostname, community, object)
$information = snmpwalk ($agentIP, "secret", "smScriptTable");
```

- Gerente WS: passos necessários:

```
//function GetScriptInformation($Script_Name)
$arguments = array('Script_Name'=>"exemplo.sh");
$information = $manager->call("GetScriptInformation", $arguments);
```

3.3.6 Criação de perfis de execução de *scripts*

Perfis de execução definem o comportamento que um *script* de gerenciamento terá durante e após o término da sua execução. Os passos necessários para se criar um perfil são apresentados a seguir:

- Gerente SNMP: passos necessários:

```
//snmpset (hostname, community, object, type, value);
snmpset ($agentIP, "secret", "smLaunchRowStatus.$index", "i", 5);
snmpset ($agentIP, "secret", "smLaunchScriptOwner.$index", "s", "nobody");
snmpset ($agentIP, "secret", "smLaunchScriptName.$index", "s", "ping.java");
snmpset ($agentIP, "secret", "smLaunchArgument.$index", "i", 10);
snmpset ($agentIP, "secret", "smLaunchLifeTime.$index", "i", 60);
snmpset ($agentIP, "secret", "smLaunchRowExpireTime.$index", "i", 0);
snmpset ($agentIP, "secret", "smLaunchMaxRunning.$index", "i", 5);
snmpset ($agentIP, "secret", "smLaunchRowStatus.$index", "i", 1);
snmpset ($agentIP, "secret", "smLaunchAdminStatus.$index", "i", 1);
$operStatus="2";
while($operStatus != "1")
{
      $operStatus = snmpget($agentIP, "secret", "smLaunchOperStatus.$index");
}
```

- Gerente WS: passos necessários:

```
\\function CreateExecutionProfile($Script_ID, $Permission, $Owner,
                                  $Group, $Argument, $Max_Life_Time="-1",
                                  $Max_Running_Concurrently="1",
                                  $Script_Expire_Time="-1")
$arguments =
array('Script_ID'=>12,
      'Permission'=>'755',
      'Owner'=>'nobody',
      'Group'=>'nobody',
      'Argument'=>'10',
      'Max_Life_Time'=>'60',
      'Max_Running_Concurrently'=>'5',
      'Script_Expire_Time'=>null);
$profile_ID = $manager->call("CreateExecutionProfile", $arguments);
```

3.3.7 Remoção de um perfil de execução

Uma vez criados, perfis de execução também podem ser removidos pelo gerente. Os passos necessários para a remoção de um perfil seriam os seguintes:

- Gerente SNMP: passos necessários:

```
//snmpset (hostname, community, object, type, value);
snmpset ($agentIP, "secret", "smLaunchAdminStatus.$index", "i", 2);
$operStatus="1";
while($operStatus != "2")
{
      $operStatus = snmpget($agentIP, "secret", "smLaunchOperStatus.$index");
}
snmpset ($agentIP, "secret", "smLaunchRowStatus.$index", "i", 6);
```

- Gerente WS: passos necessários:

```
//function RemoveExecutionProfile($Profile_ID)
$arguments = array('Profile_ID'=>3);
$status = $manager->call("RemoveExecutionProfile", $arguments);
```

3.3.8 Início da execução de um *script* de gerenciamento

Uma vez que um *script* de gerenciamento tenha sido transferido e exista um perfil de execução associado a ele, ele pode ser executado pelo gerente. Os passos necessários para a execução de um *script* podem ser vistos a seguir:

- Gerente SNMP: passos necessários:

```
//snmpset (hostname, community, object, type, value);
snmpset ($agentIP, "secret", "smLaunchStart.$index", "i", 0);
```

- Gerente WS: passos necessários:

```
//function RunScript($Script_ID, $Profile_ID, $block=false)
$arguments =
array('Script_ID'=>"12",
      'Profile_ID'=>"3",
      'block'=>null);
$status = $manager->call("RunScript", $arguments);
```

3.3.9 Consulta do resultado da execução de um *script* de gerenciamento

Normalmente, *scripts* de gerenciamento geram resultados durante e após a sua execução. Os passos necessários para se consultar o resultado da execução de um *script* seriam os seguintes:

- Gerente SNMP: passos necessários:

```
$runState="1";
while($runState != "7")
{
      $runState = snmpget($agentIP, "secret", "smRunState.$index");
}
$result = snmpget($agentIP, "secret", "smRunResult.$index");
```

- Gerente WS: passos necessários:

```
//function GetExecutionResults($Profile_ID, $Execution_ID)
$arguments =
array('Profile_ID'=>"12",
      'Execution_ID'=>"3");
$result = $manager->call("GetExecutionResults", $arguments);
```

Os mapeamentos apresentados nesta seção demonstram algumas das principais ações de gerenciamento que um gerente (SNMP ou WS) solicitaria a um agente (SNMP ou WS) para que, *scripts* de gerenciamento sejam transferidos, executados e o resultado das execuções fossem obtidos. No capítulo a seguir, maiores detalhes sobre a implementação do *gateway* em nível de serviço e do agente WS serão apresentados.

4 IMPLEMENTAÇÃO DOS PROTÓTIPOS

No capítulo anterior, as duas contribuições deste trabalho foram introduzidas, sendo apresentada uma descrição geral de cada uma delas. Apresentou-se também um mapeamento dos serviços oferecidos pela MIB Script para operações WS. Neste capítulo serão apresentados maiores detalhes da implementação dos dois protótipos desenvolvidos neste trabalho: o protótipo de um *gateway* em nível de serviço e o protótipo de um agente orientado a serviços baseado em Web Services. No final, será apresentada a interação do gerente Web Services, incorporado ao ambiente de gerenciamento de redes QAME (*QoS-Aware Management Environment*) (GRANVILLE et al., 2001) com os dois protótipos desenvolvidos.

Inicialmente, o protótipo de um *gateway* em nível de serviço será apresentado, sendo mostrados os detalhes de desenvolvimento e as tecnologias utilizadas para sua construção. Os detalhes de implementação dos serviços oferecidos pelo *gateway* em nível de serviço não serão apresentados neste capítulo, uma vez que eles podem ser inferidos através do mapeamento das funcionalidades da MIB Script para operações WS apresentadas na seção 3.3 do capítulo anterior.

Por fim, o protótipo de um agente orientado a serviços baseado em Web Services será mostrado, sendo apresentada uma descrição geral do protótipo e das tecnologias utilizadas para o seu desenvolvimento. É apresentada também a estrutura de dados utilizada pelo agente WS para realizar o armazenamento das informações de gerenciamento. Em seguida, os serviços disponibilizados pelo agente WS serão apresentados.

Finalizando o capítulo, o gerente Web Services será apresentado. Primeiramente, a interface Web oferecida pelo gerente será mostrada, seguido por um exemplo de interação entre o gerente WS e o protótipo do agente WS desenvolvido.

4.1 Protótipo de um *gateway* WS para SNMP em nível de serviço

Nesta seção, será apresentado como foi realizado o desenvolvimento do protótipo de um *gateway* em nível de serviço e, também, quais foram as tecnologias utilizadas no seu desenvolvimento.

4.1.1 Desenvolvimento do *gateway*

Como visto, um *gateway* WS para SNMP em nível de serviço é desenvolvido utilizando-se a tecnologia de Web Services, onde suas operações WS são baseadas nos serviços que um determinado módulo de MIB oferece através da manipulação dos seus objetos. Entretanto, o desenvolvimento de um *gateway* em nível de serviço para um mesmo módulo de MIB pode diferir entre diferentes implementações. Isso ocorre porque a construção do *ga-*

teway depende da análise feita do módulo de MIB e da quantidade de serviços (operações WS) que o desenvolvedor do *gateway* deseja disponibilizar. As tecnologias utilizadas no desenvolvimento também podem variar. Atualmente, existe uma grande quantidade de tecnologias que permite a construção de Web Services, como são o caso, por exemplo, das linguagens Java (SUN MICROSYSTEMS, 1994) e PHP (PHP, 2001).

No caso do protótipo de *gateway* em nível de serviço desenvolvido neste trabalho, as seguintes tecnologias foram utilizadas:

- **PHP**: Linguagem de *script* utilizada no desenvolvimento do gerente WS e do *gateway* em nível de serviço;
- **NuSOAP** (AYALA, 2003): Implementação SOAP usada na construção de requisições e respostas realizadas entre o gerente WS e o *gateway* em nível de serviço;
- **Apache** (APACHE, 1999): Servidor HTTP utilizado para o transporte das mensagens SOAP trocadas entre o gerente WS e o *gateway* em nível de serviço;
- **NET-SNMP** (SOURCEFORGE TEAM, 1992): Implementação do protocolo SNMP usado pelo *gateway* em nível de serviço na comunicação com o Jasmin, localizado no agente SNMP.

Além disso, como o *gateway* em nível de serviço desenvolvido fornece as funcionalidades da MIB Script, também foi utilizada a tecnologia Jasmin (JASMIN, 2000) que é uma implementação da MIB Script devenvolvida pela Universidade de Braschweing e pelo laboratório NEC Europe. Como requisitos mínimos para o seu funcionamento, o Jasmin exige a instalação de uma plataforma antiga do Java, o JDK 1.1.X (sendo X=5 ou superior) e também de uma implementação antiga do SNMP, o UCD-SNMP 4.2.6.

O *gateway* em nível de serviço foi codificado na linguagem de *script* PHP por ela ser uma linguagem de fácil utilização e que apresenta uma vasta documentação disponível na Internet, facilitando, com isso, a etapa de implementação dos protótipos apresentados neste capítulo. As operações WS disponibilizadas pelo *gateway* foram construídas utilizando-se a biblioteca NuSOAP, que foi escolhida para a realização deste trabalho por ser uma das mais populares e bem estruturadas implementações do protocolo SOAP existente atualmente para PHP.

A utilização do Apache como servidor HTTP deve-se ao fato dele ser software livre e também por ser amplamente utilizado. Optou-se pelo SOAP por ele ser o protocolo mais utilizado na troca de mensagens XML entre os elementos pertencentes a arquitetura de WS e, também, por ele ter a capacidade de operar sobre serviços de transporte como SMTP, FTP e HTTP. O HTTP foi escolhido por ele ser um protocolo bastante utilizado na transferência de informações via Internet, sendo o seu fluxo de pacotes geralmente liberado nas regras de um *firewall*, possibilitando, com isso, que as mensagens SOAP transportadas via HTTP sejam trocadas sem maiores restrições de acesso por parte dos *firewalls* existentes na Internet.

Por fim, a utilização de uma implementação SNMP se faz necessária para que o *gateway* WS para SNMP em nível de serviço possa comunicar-se com os agentes SNMP. Utilizou-se a implementação UCD-SNMP por ser um requisito necessário do Jasmin. O Jasmin foi utilizado neste trabalho por ele ser uma implementação da MIB Script bastante conhecida por parte da comunidade científica na área de redes.

Tendo essas tecnologias a sua disposição, um *gateway* em nível de serviço está capacitado para realizar a tradução de operações WS para SNMP e vice-versa, permitindo disponibilizar, dessa forma, uma variedade de serviços ao gerente WS.

O ponto chave na construção de um *gateway* em nível de serviço está na forma com que as operações WS disponibilizadas por ele são implementadas. Cada operação WS deve ser capaz de atender as requisições do gerente WS e traduzí-las em uma série de comandos SNMP, que, caso fossem ordenados por um gerente SNMP, realizariam uma determinada ação no agente SNMP. Por exemplo, no caso da MIB Script, se os comandos SNMP mostrados na Figura 4.1 fossem realizados por um gerente SNMP, o *download* de um *script* de gerenciamento pelo agente SNMP seria realizado.

```
snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 5);

snmpset ($agentIP, "secret", "smScriptSource.$index", "s",
         "http://scriptrepository/ping.java");

snmpset ($agentIP, "secret", "smScriptDescr.$index", "s",
         "Script que verifica o estado dos roteadores da rede");

snmpset ($agentIP, "secret", "smScriptLanguage.$index", "i", 1);

snmpset ($agentIP, "secret", "smScriptRowStatus.$index", "i", 1);

snmpset ($agentIP, "secret", "smScriptAdminStatus.$index", "i", 1);

$operStatus="4";
while($operStatus != "1")
{
  $operStatus = snmpget($agentIP, "secret", "smScriptOperStatus.$index");
}

snmpset ($agentIP, "secret", "smScriptStorageType.$index", "i", 3);
```

Figura 4.1: *Download* de um *script* de gerenciamento via comandos SNMP

Logo, para que o gerente WS possa efetuar a mesma ação realizada pelo gerente SNMP, é necessário que o *gateway* em nível de serviço disponibilize uma operação WS que realize os mesmos comandos do gerente SNMP (Figura 4.1). Uma operação WS, fornecida pelo *gateway*, que realiza tais comandos pode ser visto na Figura 4.2.

Observa-se pela Figura 4.2 que a invocação da operação `DownloadScript()` pelo gerente WS, realizaria os mesmos comandos SNMP que seriam realizados por um gerente SNMP (Figura 4.1), fazendo com que, dessa forma, o *download* de um *script* de gerenciamento fosse efetuado.

O *gateway* em nível de serviço construído neste trabalho fornece uma variedade de serviços que refletem as funcionalidades da MIB Script. Não serão apresentados, entretanto, nesta dissertação, os detalhes de construção de todas as operações WS oferecidas pelo *gateway*, uma vez que o desenvolvimento de cada operação WS segue a mesma linha de raciocínio apresentada no exemplo anterior, diferindo apenas quanto à forma que cada uma delas é implementada.

Vale salientar que um *gateway* em nível de serviço que disponibilize uma quantidade de serviços bastante elevada, oferecendo operações muito específicas, acaba atuanto como um *gateway* em nível de objeto. Como visto na seção 2.4.2, um *gateway* em nível de objeto é baseado na estrutura de um módulo de MIB, apresentado seus objetos como operações WS específicas a cada objeto. Como, geralmente, um módulo de MIB possui muitos objetos, a quantidade de operações WS oferecida por um *gateway* em nível de objeto é bastante elevada. Logo, recomenda-se que um *gateway* em nível de serviço não ofereça serviços muitos específicos e em uma quantidade muito grande, pois, dessa forma,

```
$server->register('DownloadScript');
function DownloadScript($community, $url, $descr, $language, $owner, $name,
                        $storage_type)
{
  snmpset ($agentIP, $community,
           "smScriptRowStatus"."\"$owner\"".".".".\"$name\"", 'i', 5);

  snmpset ($agentIP, $community,
           "smScriptSource"."\"$owner\"".".".".\"$name\"", 's', $url);

  snmpset ($agentIP, $community,
           "smScriptDescr"."\"$owner\"".".".".\"$name\"", 's', $descr);

  snmpset ($agentIP, $community,
           "smScriptLanguage"."\"$owner\"".".".".\"$name\"", 'i', 1);

  snmpset ($agentIP, $community,
           "smScriptRowStatus"."\"$owner\"".".".".\"$name\"", 'i', 1);

  snmpset ($agentIP, $community,
           "smScriptAdminStatus"."\"$owner\"".".".".\"$name\"", 'i', 1);

  $SCRIPT_STATUS="INTEGER: 4";
  while ($SCRIPT_STATUS == "INTEGER: 4"){
      $SCRIPT_STATUS = snmpget ($agentIP, $community,
                        "smScriptOperStatus"."\"$owner\"".".".".\"$name\"");
  }
  snmpset ($agentIP, $community,
           "smScriptStorageType"."\"$owner\"".".".".\"$name\"", 'i', $storage_type);
}
```

Figura 4.2: Protótipo de operação WS que realiza o *download* de um *script* de gerenciamento

ele terá um funcionamento bastante similar ao *gateway* em nível de objeto. Por outro lado, se a quantidade de serviços for muito pequena, isso acaba tornando o *gateway* em nível de serviço muito inflexível, não possibilitando ao gerente WS realizar os mesmos grupos de comandos que um gerente SNMP realizaria na manipulação de um agente SNMP para a realização de determinada ação de gerenciamento.

4.2 Protótipo de um agente WS orientado a serviços

Inicialmente, será apresentado nesta seção, uma descrição geral do desenvolvimento do protótipo de um agente WS orientado a serviços, bem como as tecnologias que foram utilizadas na sua construção. Em seguida, a estrutura de dados utilizada pelo agente WS para armazenar dados de gerenciamento é mostrada. Por fim, os detalhes de implementação dos serviços oferecidos pelo agente WS são apresentados.

4.2.1 Descrição geral

Seguindo a mesma idéia de um *gateway* em nível de serviço, o agente WS também foi desenvolvido utilizando-se a tecnologia de Web Services e a sua construção também se baseia nas funcionalidades que um determinado módulo de MIB possui. Da mesma forma que o *gateway* em nível de serviço, a implementação de um agente WS pode variar dependendo da análise feita de um determinado módulo de MIB, da quantidade de serviços (operações WS) que o desenvolvedor do agente WS deseja disponibilizar e das tecnologias empregadas na sua construção. Entretanto, diferentemente do *gateway* em nível de serviço, a tecnologia SNMP não é mais utilizada, existindo, com isso, um sistema de gerenciamento totalmente baseado em Web Services.

O protótipo do agente WS construído neste trabalho visa fornecer as funcionalidades

que um determinado módulo de MIB oferece, mas não obriga que o dispositivo gerenciado precise ter necessariamente um agente SNMP instalado, como é o caso mais comum. Ao invés disso, o dispositivo é gerenciado apenas usando recursos do seu próprio sistema operacional e através de um servidor Web, que é um recurso comumente encontrado em dispositivos de rede, sendo de fácil instalação e configuração. Logo, como boa parte dos dispositivos de rede possui um sistema operacional instalado, a utilização de um servidor Web com suporte a PHP é o requisito mínimo para se ter um dispositivo de rede gerenciado através do agente WS desenvolvido neste trabalho. No caso de ser necessário o armazenamento de informações de gerenciamento no dispositivo gerenciado, uma base de dados se torna um outro requisito.

No caso do protótipo do agente WS desenvolvido neste trabalho, as funcionalidades que a MIB Script oferece são disponibilizadas pelo agente WS através de operações WS que interagem diretamente com o sistema operacional do dispositivo gerenciado. Neste trabalho, somente dispositivos com sistema operacional baseado em UNIX são suportados pelo agente WS. Além de um sistema operacional baseado em UNIX, as mesmas tecnologias utilizadas no desenvolvimento do protótipo de um *gateway* WS para SNMP em nível de serviço, com exceção dos requisitos do Jasmin, são utilizadas na construção do protótipo de um agente WS orientado a serviços, acrescidas da utilização da tecnologia MySQL (MYSQL, 1995).

As justificativas de utilização das tecnologias: PHP, NuSOAP e Apache são as mesmas apresentadas no caso do *gateway* em nível de serviço. A base de dados MySQL foi utilizada devido a necessidade de armazenamento em um local não-volátil das estruturas de dados utilizadas pelo agente WS e das informações de gerenciamento. Esse armazenamento de informações de gerenciamento se faz necessário devido a própria definição da MIB Script (LEVI; SCHÖNWÄLDER, 2001) que possibilita que informações referentes a transferência, execução e obtenção dos resultados de execução dos *scripts* de gerenciamento possam ser consultadas por um gerente de rede muito tempo depois do término da execução de um *script*, por exemplo.

Um cenário demonstrando a interação entre o agente WS desenvolvido neste trabalho e o gerente WS incorporado ao QAME pode ser visualizado na Figura 4.3.

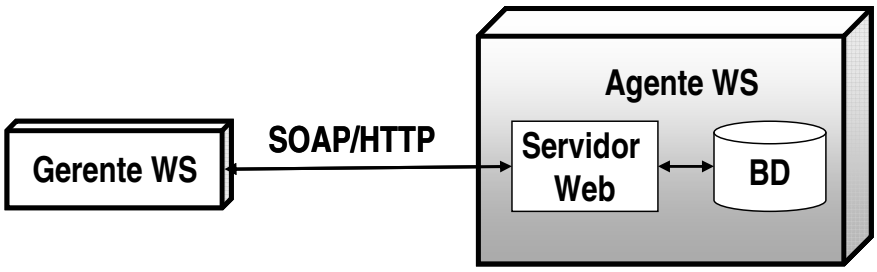

Figura 4.3: Cenário de interação entre um gerente e um agente WS

O gerente WS se comunica com o agente WS através de requisições SOAP transportadas via HTTP. No agente WS, existe um servidor Web que atende as requisições do gerente, processa-as, interagindo se necessário com a base de dados (BD) instalada, e devolve o resultado das requisições para o gerente WS, novamente através de SOAP via HTTP. Como mencionado anteriormente, no caso deste trabalho, utilizou-se a tecnologia Apache como o servidor Web e a tecnologia MySQL como base de dados. Logo, os requisitos mínimos para o funcionamento do protótipo são resumidos a seguir:

- Sistema operacional baseado em UNIX;
- Servidor Web Apache com suporte a PHP;
- Base de dados MySQL.

Os requisitos esperados, pelo agente WS, do sistema operacional são a disponibilidade dos comandos: `wget`, `sudo` e `time`. O comando `wget` é utilizado pelo agente WS para realizar a transferência de *scripts* de gerenciamento pelo método **pull** (*download* de um *script* pelo agente WS). O comando `sudo` é utilizado para realizar a execução de comandos que exigam privilégios de administrador (usuário *root* no caso de sistemas UNIX) [1]. Por fim, o comando `time` fornece informações sobre a execução de um *script*.

Quanto ao servidor Web Apache, a sua instalação não exige nenhuma opção especial habilitada. Por outro lado, o módulo PHP precisa ter suporte ao MySQL e a XML. Uma vez instalado, o servidor Web Apache precisa ser configurado para que o diretório raiz, onde estão localizados os documentos Web disponibilizados pelo Apache, reflita a localização do agente WS, permitindo com isso o acesso pelo gerente WS.

A base de dados MySQL utilizada também não exige nenhuma opção especial habilitada na sua instalação. Porém, uma vez que o agente WS utiliza o MySQL para armazenar informações de gerenciamento, é necessário que a estrutura de dados utilizada pelo agente WS seja instalada pelo MySQL. Além disso, para fins de segurança, recomenda-se a criação de um usuário MySQL que tenha somente permissão de acesso e modificação na base de dados do agente WS.

Apresentada a descrição geral do protótipo de um agente WS orientado a serviços, nas subseções a seguir serão apresentados detalhes sobre o seu desenvolvimento, mais especificadamente sobre a estrutura da base de dados utilizada e os serviços disponibilizados pelo protótipo.

4.2.2 A estrutura da base de dados

Apresentada a descrição geral de um sistema de gerenciamento baseado em Web Services, nesta subseção serão apresentados maiores detalhes do desenvolvimento do protótipo de um agente WS desenvolvido neste trabalho, mais especificadamente sobre a estrutura da base de dados. Inicialmente, o modelo de informação da estrutura será mostrado, sendo explicada em seguida cada uma das tabelas pertencentes a base de dados.

A estrutura da base de dados utilizada pelo protótipo de um agente WS desenvolvido neste trabalho, é baseada na estrutura da MIB Script descrita na RFC 3165 (LEVI; SCHÖNWÄLDER, 2001). Entretanto, tal estrutura não é seguida rigidamente, ela serve apenas como um modelo de construção. Da mesma forma que a MIB Script, a estrutura da base de dados desenvolvida neste trabalho é composta por seis tabelas (Figura 4.4), que serão descritas a seguir.

Tabela Script:

A tabela `Script` tem por função descrever um *script* de gerenciamento que foi transferido ao agente WS através do método **push** ou **pull**. Todo o *script* transferido possui

[1] Por questões de segurança, não é recomendável a concessão de privilégios de administrador para outros usuários diferentes do usuário *root*. Entretanto, neste trabalho, não foram consideradas tais questões de segurança na construção do protótipo de um agente WS por não ser esse o foco do trabalho.

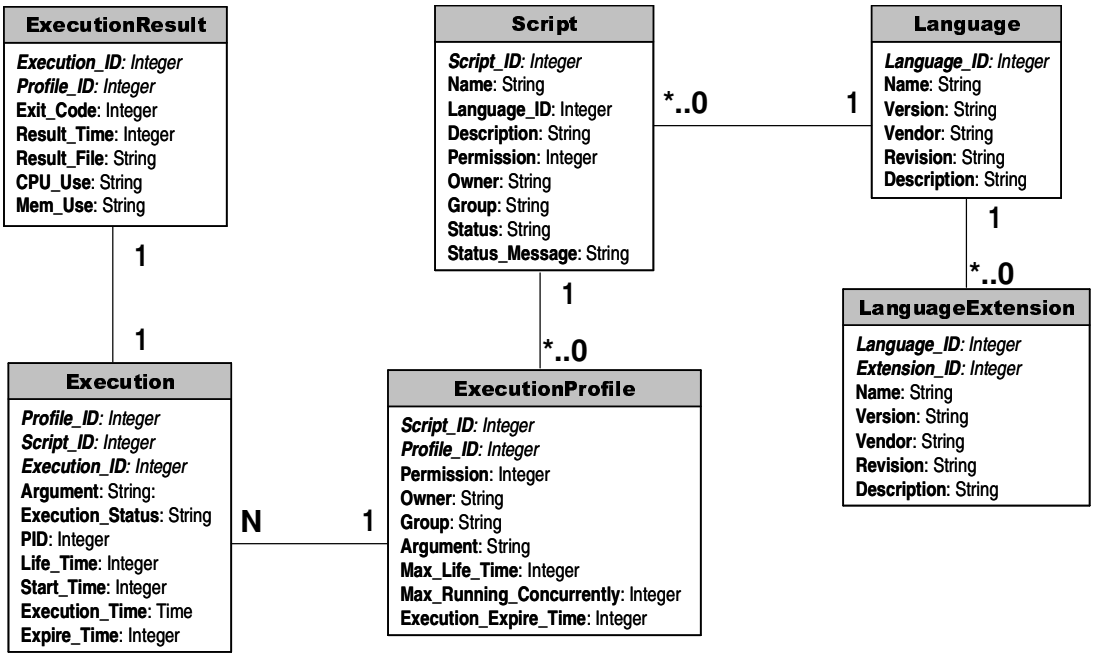

Figura 4.4: Modelo de informação da estrutura do agente WS

um identificador `Script_ID` que irá identificá-lo nas demais tabelas que serão apresentadas no decorrer desta subseção. Um *script* é descrito por um nome (`Name`), por um identificador de ambiente de execução (`Language_ID`), por uma descrição textual (`Description`) e por permissões de acesso (`Permission`, `Owner` e `Group`). Essas informações são fornecidas pelo gerente WS. O estado atual de transferência e a descrição textual do estado são fornecidos pelos atributos `Status` e `Status_Message`, respectivamente. Um *script* pode estar em um dos seguintes estados:

- *downloading*: indica que o *script* está em processo de transferência pelo método **pull**;
- *pushing*: indica que o *script* está em processo de transferência pelo método **push**;
- *downloaded*: indica que o *script* foi transferido com sucesso pelo método **pull**;
- *pushed*: indica que o *script* foi transferido com sucesso pelo método **push**;
- *transfer error*: indica que erros ocorreram durante a transferência. A descrição do erro é fornecida em `Status_Message`.

A tabela `Script` é demonstrada na Figura 4.5, onde é apresentado um exemplo de *scripts* de gerenciamento atualmente sendo tratados pelo agente WS.

Pela Figura 4.5 pode-se verificar que existem cinco *scripts* de gerenciamento atualmente registrados na tabela `Script`. Cada *script* possui seu identificador, um nome, um identificador de ambiente de execução, uma descrição textual, suas permissões e seus proprietários. Por padrão, o usuário (`Owner`) e o grupo (`Group`) serão o usuário e o grupo com permissão de execução do servidor Web localizado no agente WS. Verifica-se também pela tabela que dois *scritps*: '*Ping*' e '*Who.sh*', foram transferidos com sucesso através dos métodos **pull** e **push**, respectivamente. Outros dois: '*Route.sh*' e '*Sleep*' estão

Script_ID	Name	Language_ID	Description	Permission	Owner	Group	Status	Status_Message
1	Ping	1	Programa em Java que verifica o estado atual dos roteadores	644	nobody	nobody	Downloaded	Script has been successfully downloaded
2	Who.sh	3	Shell Script que verifica usuários logados	644	nobody	nobody	Pushed	Script has been successfully pushed
3	Route.sh	2	Shell Script que verifica a tabela de rotas	644	nobody	nobody	Pushing	Script is being pushed
4	Sleep	1	Programa em Java que dorme por um tempo determinado	644	nobody	nobody	Downloading	Script is being downloaded
5	Remove.sh	3	Shell Script que remove arquivos temporários	644	nobody	nobody	Transfer Error	Script URL has not been specified

Figura 4.5: Exemplo de informações contidas na tabela `Script`

em processo de transferência. Por fim, um dos *scripts*: '*Remove.sh*' não obteve sucesso na sua transferência, sendo a descrição do erro apresentada no atributo `Status_Message`.

Tabela Language:

Todos os ambientes de execução que são suportados pelo agente WS são listados na tabela `Language`. Um ambiente de execução pode estar associado a vários *scripts* de gerenciamento existentes na tabela `Script`. A tabela `Language` é formada por um identificador de ambiente de execução (`Language_ID`), sendo cada ambiente descrito por um nome (`Name`), uma versão (`Version`), um fabricante (`Vendor`), uma revisão da versão (`Revision`) e uma descrição textual do ambiente (`Description`).

A tabela `Language` é demonstrada na Figura 4.6, onde é apresentado um exemplo de ambientes de execução suportados pelo agente WS.

Language_ID	Name	Version	Vendor	Revision	Description
1	Java	1.4.2	Sun Microsystems Inc.	01-b01	Java runtime engine
2	Bourne Again Shell	2.05	Free Software Foundation	-	Bourne Again Shell runtime engine
3	Tcl	8.3	Sun Microsystems, Inc.	-	Tcl runtime engine

Figura 4.6: Exemplo de informações contidas na tabela `Language`

A figura 4.6 mostra três ambientes de execução que são suportados pelo agente WS: '*Java*', '*Bourne Again Shell*' e '*Tcl*'. Cada ambiente possui seu identificador e demais atributos que descrevem o ambiente instalado no agente WS.

Tabela LanguageExtension:

As extensões de um ambiente de execução suportadas pelo agente WS são listadas na tabela `LanguageExtension`. Uma extensão está associada a um ambiente de execução

através da chave estrangeira `Language_ID`. A tabela `LanguageExtension` é formada por um identificador de extensão (`Extension_ID`), sendo cada extensão de um ambiente descrito por um nome (`Name`), uma versão (`Version`), um fabricante (`Vendor`), uma revisão da versão (`Revision`) e uma descrição textual da extensão (`Description`).

A tabela `LanguageExtension` é demonstrada na Figura 4.7, onde é apresentado um exemplo de extensões existentes para os ambientes de execução suportados pelo agente WS.

Extension_ID	Language_ID	Name	Version	Vendor	Revision	Description
1	1	jmgmt	-	-	-	SNMP extension for Java
2	1	snmpmonitor	-	-	-	SNMP monitoring extension for Java
3	1	policyMgmt	-	-	-	Policy management extension for Java
4	3	Tnm	-	-	-	SNMP extension for Tcl

Figura 4.7: Exemplo de informações contidas na tabela `LanguageExtension`

A figura 4.7 apresenta quatro extensões existentes no agente WS: '*1*', '*2*', '*3*' e '*4*'. As três primeiras, '*jmgmt*', '*snmpmonitor*' e '*policyMgmt*', são extensões para o ambiente de execução '*Java*'. A última, '*Tnm*', é uma extensão para o ambiente '*Tcl*'. Cada extensão possui seu identificador e demais atributos que a descreve.

Tabela ExecutionProfile:

A criação de perfis de execução, para os *scripts* de gerenciamento transferidos, é realizada na tabela `ExecutionProfile`. Uma entrada na tabela `ExecutionProfile` está associada a uma entrada na tabela `Script` através da chave estrangeira `Script_ID` e, também, está associada a `N` entradas na tabela `Execution`. Os valores dos atributos `Permission`, `Owner` e `Group` indicam quais são as permissões de execução do *script* e quem são os seus proprietários. Tais valores são obtidos através dos valores fornecidos aos atributos, de mesmo nome, existentes na tabela `Script`. Salienta-se que esses valores podem ser alterados através das operações WS fornecidas pelo agente WS (subseção 4.2.3). Os argumentos para a execução de um *script* são passados através da atribuição de valores ao atributo `Argument`. O tempo máximo de execução de um *script* (`Max_Life_Time`) é definido em segundos. Como mencionado anteriormente, uma entrada na tabela `Script` pode estar associada a `N` entradas na tabela `Execution`. Esse valor `N` é determinado através do valor contido no atributo `Max_Running_Concurrently`, que determina qual é o número máximo de execuções em paralelo de *scripts* com o mesmo perfil de execução. Por fim, o atributo `Execution_Expire_Time` define o tempo em segundos que o resultado da execução de um *script* estará disponível na tabela `Execution`.

Profile_ID	Script_ID	Permission	Owner	Group	Argument	Max_Life_ Time	Max_Running_ Concurrently	Execution_Expire_ Time
1	1	744	nobody	nobody	"router1.inf.ufrgs.br"	10	1	600
2	1	744	tfioreze	labcom	"router2.inf.ufrgs.br"	10	2	-1
3	4	755	tfioreze	labcom	10	-1	1	-1
4	5	744	root	nobody	-	-1	1	-1

Figura 4.8: Exemplo de informações contidas na tabela `ExecutionProfile`

A tabela `ExecutionProfile` é demonstrada na Figura 4.8, onde é apresentado um exemplo de perfis de execução atualmente definidos no agente WS. Nessa figura, quatro perfis de execução estão registrados na tabela `ExecutionProfile`, identificados pelos valores do atributo `Profile_ID`: '*1*', '*2*', '*3*' e '*4*'. O perfil de execução '*1*' está associado ao *script* '*Ping*' existente na tabela `Script`, definindo que o usuário e grupo '*nobody*' são seus proprietários, com permissão de execução ('*7*') apenas para o usuário '*nobody*'. É definido também que o argumento '*"router1.inf.ufrgs.br"*' será passado ao *script* e que o mesmo terá um tempo de vida de 10s. O número máximo de *scripts* que irão executar em paralelo é o valor padrão ('*1*'), o que é entendido pelo agente WS que apenas uma execução associada ao perfil de execução '*1*' irá ocorrer. Define-se também que o resultado do *script* '*Ping*' estará disponível em uma entrada na tabela `Execution` durante o tempo máximo de 600s.

O perfil de execução '*2*' está associado ao mesmo *script* '*Ping*', o que implica em um novo perfil de execução para o mesmo *script*. Os proprietários do *script* são o usuário '*tfioreze*' e o grupo de usuários '*labcom*'. Entretanto, apenas o usuário '*tfioreze*' possui permissão de execução ('*7*'). O argumento passado é '*"router2.inf.ufrgs.br"*'. Definiu-se também que o número máximo de *scripts* em execução, associado ao perfil de execução '*2*', será '*2*'. Logo, ter-se-á dois *scripts* em execução associados ao perfil de execução '*2*', o que implica em duas entradas na tabela `Execution`. Cada execução terá um tempo máximo de 10s. O valor padrão '*-1*' foi atribuído ao atributo `Execution_Expire_Time`, o que implica que o resultado estará disponível na tabela `Execution` até o momento que o *script* '*Ping*' permanecer na tabela `Script`.

O perfil de execução '*3*' está associado ao *script* '*Sleep*', definindo que o proprietário do mesmo é o usuário '*tfioreze*' e o grupo de usuários '*labcom*'. Nesse perfil, tanto o usuário '*tfioreze*' quanto membros do grupo '*labcom*', tem poder de execução. O argumento passado é '*10*', que, no caso do *script* '*Sleep*', se refere a quantos segundos ele irá "dormir". O valor '*-1*' atribuído ao atributo `Life_Time` determina que não há tempo máximo de execução, o que significa que o *script* irá executar sem limite de tempo no agente WS. Tanto o atributo `Max_Running_Concurrently` quanto `Execution_Expire_Time` estão com os valores padrão ('*1*').

O último perfil de execução, o perfil '*4*', está associado ao *script* '*Remove.sh*' e possui o usuário '*root*' e o grupo de usuários '*nobody*' como proprietário. Porém, apenas o usuário '*root*' possui permissão de execução. Os demais atributos não foram alterados, ficando com os seus valores padrão.

Tabela Execution:

A tabela `Execution` lista todos os *scripts* que estão atualmente em execução no dispositivo gerenciado pelo agente WS, apresentando informações referente a execução dos mesmos. Uma entrada na tabela `Execution` está associada a uma entrada na tabela `ExecutionProfile` através da chave estrangeira `Profile_ID` e a uma entrada na tabela `ExecutionResult` através da sua própria chave primária `Execution_ID`. Através da tabela `Execution` é possível consultar os argumentos que foram passados ao *script* em execução (`Argument`) e o seu atual estado de execução, que pode ser um dos seguintes:

- *Sleeping*: O processo está parado por algum tempo determinado ou aguardando algum evento ser realizado (um sinal, por exemplo);
- *Uninterruptible Sleep*: O processo está parado e não pode ser interrompido. Nor-

malmente, devido a alguma requisição feita ao sistema operacional (leitura de algum arquivo em disco);

- *Running*: O processo está atualmente em execução ou pronto para ser executado;
- *Terminated*: A execução do processo foi interrompida por um depurador ou pelo próprio sistema operacional;
- *Defunct*: Um processo filho que encerrou a sua execução, mas não pode liberar os recursos alocados pelo sistema operacional porquê ele ainda está ligado ao processo que o criou;
- *Finished*: Um processo que encerrou normalmente a sua execução.

A tabela também fornece o identificador do processo em execução (`PID`), o tempo máximo de execução do *script*, definido em segundos, e obtido através do valor existente no atributo `Max_Life_Time` definido na tabela `ExecutionProfile`, o *timestamp* do início da execução (`Start_Time`) e o tempo total que o *script* levou para executar (`Execution_Time`), definido no formato de HH:MM:SS. Todos esses valores são persistentes até que o tempo máximo de consulta de informações seja alcançado (`Expire_Time`) ou que o *script* seja removido da tabela `Script`. O tempo máximo de consulta de informações é definido em segundos e seu valor é obtido através do valor existente no atributo `Execution_Expire_Time`, definido na tabela `ExecutionProfile`.

A tabela `Execution` é demonstrada na Figura 4.9, onde é apresentado um exemplo de *scripts* de gerenciamento em execução no dispositivo gerenciado pelo agente WS.

Execution_ID	Profile_ID	Script_ID	Argument	Execution_ Status	PID	Life_ Time	Start_ Time	Execution_ Time	Expire_ Time
1	1	1	"router1.inf.ufrgs.br"	Running	14235	10	1108759969	00:00:00	600
2	2	1	"router2.inf.ufrgs.br"	Running	14232	10	1108759949	00:00:00	-1
3	2	1	"router2.inf.ufrgs.br"	Running	14233	10	1108759950	00:00:00	-1
4	3	4	10	Sleeping	14242	-1	1108758800	00:00:00	-1
5	4	5	-	Finished	14237	-1	1108755512	00:35:23	-1

Figura 4.9: Exemplo de informações contidas na tabela `Execution`

Na figura 4.9, cinco execuções de *scripts* de gerenciamento estão registradas na tabela `Execution`, sendo eles identificados pelos valores do atributo `Execution_ID`: '*1*', '*2*', '*3*', '*4*' e '*5*'.

A execução '*1*' se refere ao perfil de execução '*1*' descrito anteriormente. Verifica-se pela figura que o *script* '*Ping*' (`Script_ID` '*1*') teve o valor '*"router1.inf.ufrgs.br"*' passado como argumento e que está ainda em execução, com o seu identificador de processo igual a '*14235*'. A execução teve início com o valor de *timestamp* igual a '*1108759969*' e terá um tempo de duração de 10s. O tempo total de execução está em '*00:00:00*' pois o *script* ainda não encerrou a sua execução. Uma vez terminada, essa entrada na tabela `Execution` irá durar o tempo de 600s.

As execuções '*2*' e '*3*' se referem ao perfil de execução '*2*'. Verifica-se pela figura que essas execuções possuem os mesmos valores nos atributos `Argument`, `Life_Time` e `Expire_Time`. Isso é devido ao fato dessas execuções estarem associadas ao mesmo perfil de execução (`Profile_ID` '*2*'). Verifica-se também que o estado atual das execuções é '*Running*', que indica estarem ainda executando.

No caso da execução '*4*', o perfil de execução utilizado foi o '*3*', que está associado ao *script* '*Sleep*' (`Script_ID` '*4*'). Esse *script* teve o valor '*10*' passado como argumento e atualmente está em execução com PID igual a '*14242*'. Embora esteja em execução, seu estado atual é '*Sleeping*' indicando com isso que o processo não está atualmente usando o processador, talvez por um escalonamento de processos realizados pelo sistema operacional ou por uma própria instrução existente no código do programa '*Sleep*'. O processo iniciou com o valor de *timestamp* igual a '*1108758800*' e terminará com o encerramento normal da execução do programa, uma vez que não foram passados valores para o atributo `Life_Time`.

A última execução é a execução de número '*5*' que está associada ao perfil de execução '*4*', referente ao *script* `Remove.sh` (`Script_ID` '*5*'). A tabela mostra que nenhum argumento foi passado e que o *script* já terminou a sua execução. Mostra também que o PID '*14237*' foi o identificador que ele teve durante o tempo de execução e que a execução teve início com o valor de *timestamp* igual a '*1108755512*'. Uma vez que o *script* encerrou a sua execução, o valor do atributo `Execution_Time` foi preenchido com o tempo total de execução do *script*: '*00:35:23*'.

Tabela ExecutionResult:

Os resultados da execução de um *script* de gerenciamento podem ser obtidos através da consulta a tabela `ExecutionResult`. Um resultado está associado a uma entrada na tabela `Execution` através das chaves estrangeiras: `Execution_ID` e `Profile_ID`. Uma entrada é adicionada na tabela `ExecutionResult` quando um *script* encerra a sua execução, sendo possível consultar o código de retorno da execução (`Exit_Code`), o valor de *timestamp* (`Result_Time`) referente a última alteração no arquivo definido em `Result_File`, sendo esse o arquivo que contém os resultados da execução do *script*. É possível também consultar a porcentagem de utilização da CPU (`CPU_Use`) e a quantidade média em Kilobytes de utilização de memória (`Mem_Use`) que o *script* teve.

A tabela `ExecutionResult` é demonstrada na Figura 4.10, onde é apresentado um exemplo de resultados obtidos pela execução de *scripts* de gerenciamento que executaram no dispositivo gerenciado pelo agente WS.

Execution_ID	Profile_ID	Exit_Code	Result_Time	Result_File	CPU_Use	Mem_Use
5	4	0	1108757264	1108755512_4	25%	20
1	1	127	1108759979	1108759969_1	11%	0

Figura 4.10: Exemplo de informações contidas na tabela `ExecutionResult`

Na figura 4.10, dois resultados de execuções de *scripts* de gerenciamento estão registrados na tabela `ExecutionResult`, sendo eles identificados pelos pares de atributo: `Execution_ID` '*5*' - `Profile_ID` '*4*' e `Execution_ID` '*1*' - `Profile_ID` '*1*'.

A primeira entrada na tabela `Execution` mostra que o *script* identificado pelos valores de atributo '*5*' (`Execution_ID`) e '*4*' (`Profile_ID`) encerrou normalmente a sua execução, uma vez que o código de retorno teve o valor '*0*'. É mostrado também que o valor de *timestamp* da última alteração no conteúdo do arquivo '*1108755512_4*' foi '*1108757264*'. Por fim. o *script* teve uma porcentagem de utilização de CPU de '*25%*' e uma quantidade média de memória utilizada de 20 Kilobytes. O resultado da execução está contido dentro do arquivo '*1108755512_4*'.

A segunda entrada na tabela `Execution` mostra que o *script* identificado pelos valores de atributo '*1*' (`Execution_ID`) e '*1*' (`Profile_ID`) encerrou sua execução com erro, uma vez que o código de retorno teve o valor '*127*'. É mostrado também que o valor de *timestamp* da última alteração no conteúdo do arquivo '*1108759969_1*' foi '*1108759979*'. Analisando-se a tabela `Execution`, verifica-se que o *script* iniciou a sua execução com o valor de timestamp em '*1108759969*' e que o tempo máximo de duração do *script* foi configurado para 10s, o que totalizaria '*1108759979*', que é o valor da última alteração no arquivo '*1108759969_1*'. Logo, pode-se deduzir que o *script* estava em execução, quando foi encerrado pelo agente WS, retornando com isso o valor '*127*'. Esse valor de retorno diferente de zero caracteriza um fim abrupto da execução de um programa. Verifica-se também que o *script* teve uma porcentagem de utilização de CPU em torno de '*11%*' e a quantidade média de memória utilizada foi de 0 Kilobytes. Por fim, o resultado da execução e a descrição do erro estão contidos no arquivo '*1108759969_1*'.

Essas são as tabelas utilizadas pelo agente WS para manter as informações de gerenciamento armazenadas de forma estruturada em um meio não-volátil de armazenamento. A seguir serão apresentados os serviços oferecidos pelo agente WS.

4.2.3 Os serviços oferecidos

A interface dos serviços oferecidos por um agente WS é similar as interface oferecidas por um *gateway* em nível de serviço. Os serviços oferecidos pelo protótipo do agente WS podem ser separados em 5 grupos, contendo cada grupo, operações WS relacionadas. As operações podem ser divididas em funções unitárias e compostas.

***Funções unitárias*:**

O agente WS fornece várias funções unitárias. Funções unitárias são funções que não agrupam outras funções, como são os casos das funções compostas que serão apresentadas no decorrer desta subseção. As funções unitárias são divididas em grupos, sendo os mesmos apresentados a seguir.

Ambientes de Execução e Extensões:

As operações existentes neste grupo permitem que os ambientes de execução e respectivas extensões (caso existam) sejam consultadas pelo gerente WS. Os ambientes de execução suportados pelo agente WS são definidos em um arquivo de configuração.

A API dos serviços oferecidos neste grupo é a seguinte:

- function GetLanguages($Name)

A função `GetLanguages()` retorna informações existentes na tabela `Language` (subseção 4.2.2), referentes aos ambientes de execução que são suportados pelo agente WS. Ela possui um único argumento (`$Name`) que é opcional. Caso valores não sejam passados pelo gerente WS ao argumento, todos ambientes de execução suportados pelo agente WS, com suas respectivas informações, serão retornados.

- function GetLanguageExtensions($Language_ID)

A função `GetLanguageExtensions()` retorna informações sobre as extensões, existentes na tabela `LanguageExtension` (subseção 4.2.2), para um determinado ambiente de execução suportado pelo agente WS. Ela possui um único argumento (`$Language_ID`) que é opcional. Caso o argumento não seja passado pelo gerente WS, todos os ambientes de execução suportados pelo agente WS com suas respectivas extensões serão retornados.

***Scripts* de Gerenciamento**:

As operações existentes neste grupo permitem ao gerente solicitar a transferência de *scripts* de gerenciamento pelos métodos **push** ou **pull**. Permite também que *scripts* sejam removidos do agente WS, o que implica no encerramento da sua execução, caso ela esteja ocorrendo, e a remoção de todas as suas entradas nas tabelas apresentadas na subseção 4.2.2. Por fim, é possível consultar informações sobre a transferência de *scripts* e, também, listar os *scripts* de gerenciamento existentes no agente WS.

A API dos serviços oferecidos neste grupo é a seguinte:

- function DownloadScript($URL, $Script_Name, $Language, $Description, $Volatile_Storage_Type=false, $block=true)

A função `DownloadScript()` solicita o agente WS a realizar o *download* de um *script* de gerenciamento (método **pull**). Os argumentos da função são a localização remota do *script* (`$URL`), o nome que o mesmo terá no agente WS (`$Script_Name`), o ambiente de execução que ele é suportado (`$Language`) e uma descrição textual do *script* (`$Description`), sendo este último argumento opcional. O argumento `$Volatile_Storage_Type` será sempre igual a '*false*' na implementação desenvolvida neste trabalho, indicando com isso que o *script* transferido ficará armazenado no agente WS até que o gerente WS decida pela sua remoção. Esse argumento está relacionado ao objeto `smScriptStorageType`, definido na MIB Script, que indica se o *script* será armazenado, ou não, em um meio de armazenamento não volátil. O último argumento (`$block`) é utilizado pelo gerente WS para indicar se a operação invocada será bloqueante ou não. Por padrão, ela atua como bloqueante. O retorno da função `DownloadScript()` será o identificador do *script* transferido (`$Script_ID`) caso a transferência tenha ocorrido com sucesso. Caso contrário, uma mensagem de erro é retornada ao gerente WS.

- function PushScript($Code, $Script_Name, $Language, $Description, $Volatile_Storage_Type=false)

A função `PushScript()` permite ao gerente WS transferir um *script* de gerenciamento, para o agente WS, através do método **push**, ou seja, o próprio gerente WS envia o código do *script* através da invocação dessa função. Conforme a especificação da MIB Script, uma implementação da MIB Script pode permitir o envio de um *script* pelo método **push** e também permitir uma posterior execução do mesmo no agente. O agente implementado neste tabalho suporta essa transferência, o que não ocorre com a implementação Jasmin. Entretanto, ele não suporta a compilação do código transferido,

funcionando somente com ambientes de execução que atuam como interpretadores. Os argumentos que são passados à operação WS são `$Code`, que é um *array* contendo linhas de código de programação, `$Script_Name`, que define o nome do *script* transferido, `$Language`, que define o ambiente de execução suportado pelo *script*, e o argumento opcional `$Description`, que permite uma descrição textual do mesmo. O argumento `$Volatile_Storage_Type` será sempre igual a '*false*', devido a explicação dada anteriormente. Essa função não disponibiliza a opção de ser bloqueante ou não, uma vez que é o próprio gerente WS que irá indicar o término da operação, quando o mesmo terminar de transferir o *script* de gerenciamento. O retorno da função `PushScript()` será o identificador do *script* transferido (`$Script_ID`) caso a transferência tenha ocorrido com sucesso. Caso contrário, uma mensagem de erro é retornada ao gerente WS

- **function RemoveScript($Script_ID)**

A função `RemoveScript()` tem por objetivo remover de um meio não-volátil de armazenamento um *script* de gerenciamento localizado no agente WS. Caso o *script* esteja em execução, o mesmo é encerrado e todas as suas informações são removidas da base de dados MySQL. O único argumento obrigatório passado é o identificador do *script* na base de dados MySQL (`$Script_ID`). Caso a remoção tenha sido efetuada com sucesso, uma mensagem confirmando a remoção é retornada ao gerente WS. Caso contrário, uma mensagem de erro é retornada.

- **function GetScriptInformation($Script_Name)**

A função `GetScriptInformation()` retorna todas as informações contidas na tabela `Script` (subseção 4.2.2). Ela possui apenas um argumento (`$Script_Name`), que é opcional. Caso o nome do *script* não seja especificado, informações referentes a todos os *scripts* existentes atualmente na tabela `Script` serão retornadas.

Perfil de Execução:

Neste grupo estão definidas operações que permitem a criação, remoção, alteração e consulta de perfis de execução dos *scripts* de gerenciamento atualmente localizados no agente WS. Um perfil de execução é um conjunto de opções de execução de um *script*, tais como argumentos do *script* e o seu tempo máximo de execução.

A API dos serviços oferecidos neste grupo é a seguinte:

- **function CreateExecutionProfile($Script_ID, $Permission, $Owner, $Group, $Argument, $Max_Life_Time="−1", $Max_Running_Concurrently="1", $Script_Expire_Time="−1")**

A função `CreateExecutionProfile()` cria um perfil de execução para um *script* de gerenciamento. O argumento `$Script_ID` é o único argumento obrigatório. É ele que associa um perfil de execução a um *script* de gerenciamento localizado no agente. Os argumentos `$Permission`, `$Owner` e `$Group` identificam quais as permissões de exe-

cução e qual usuário ou grupo de usuários de um *script*, respectivamente. As permissões padrão de um *script* são 644. O usuário e grupo padrão são os que possuem permissão de execução do servidor Apache, localizado no agente WS, sendo geralmente o usuário e grupo *nobody*. Esses valores são definidos no momento da transferência do *script* para o agente WS. Entretanto, tais valores podem ser alterados pelo gerente WS, passando-os como argumentos para a função `CreateExecutionProfile()`. *Scripts* de gerenciamento podem necessitar de argumentos no momento da execução. Esses argumentos podem ser passados através do argumento `$Argument`. O tempo máximo que um *script* irá executar é definido em segundos pelo argumento `$Max_Life_Time`, sendo seu valor padrão igual a '*-1*', o que determina a execução de um *script* sem limite de tempo. O número máximo de *scripts* que podem executar com o mesmo perfil de execução é definido em `$Max_Running_Concurrently`, sendo seu valor padrão igual a '*1*', definindo com isso que existirá apenas uma execução de um *script* associado a um determinado perfil de execução. O último argumento (`$Script_Expire_Time`) define a quantidade de tempo que informações sobre a execução de um determinado perfil de execução serão mantidas na tabela `Execution`, sendo o seu valor padrão igual a '*-1*', o que determina que tais informações ficaram disponíveis até que um *script*, associado a um determinado perfil de execução, seja removido. O valor de retorno da função `CreateExecutionProfile()`, no caso de sucesso na execução, será um identificador do perfil de execução criado. Caso contrário, mensagens de erro serão retornadas ao gerente WS informando erros na criação do perfil de execução.

- function GetExecutionProfile($Script_ID, $Profile_ID)

A função `GetExecutionProfile()` retorna todos os perfis de execução contidas na tabela `ExecutionProfile` (subseção 4.2.2). Ela possui dois argumentos: `$Script_ID` e `$Profile_ID`, sendo os dois opcionais. Caso os argumentos não sejam especificados, informações referentes a todos os perfis de execução existentes atualmente na tabela `ExecutionProfile` serão retornadas.

- function RemoveExecutionProfile($Profile_ID)

A função `RemoveExecutionProfile()` remove todos ou apenas o perfil de execução especificado em `$Profile_ID`. No caso do gerente WS não especificar qual perfil de execução deseja excluir, o agente WS irá remover todos os perfis de execução existentes atualmente na tabela `ExecutionProfile`. A função retorna uma mensagem confirmando o sucesso da(s) remoção(ões). Caso contrário, mensagens de erro são retornadas.

- function SetProfileSecurity($Profile_ID,$Permission,$Owner,$Group)

A função `SetProfileSecurity()` permite alterar as permissões de execução, o usuário e o grupo de um ou mais perfis de execução existentes na tabela `ExecutionProfile`. No caso do gerente WS não especificar qual perfil de execução (`$Profile_ID`) a ser alterado, todos os perfis de execução localizados na tabela `ExecutionProfile` terão as permissões de execução (`$Permission`) e os usuários (`$Owner` e `$Group`) alterados.

- function GetProfileSecurity($Profile_ID)

A função `GetProfileSecurity()` retorna informações sobre quais são as permissões de execução, o usuário e o grupo de um ou mais perfis de execução existentes na tabela `ExecutionProfile`. No caso do gerente WS não especificar qual perfil de execução (`$Profile_ID`) a ser consultado, o agente WS retornará informações referentes a permissões, usuário e grupo de todos perfis de execução.

- function SetProfileExecutionOptions($Profile_ID, $Max_Life_Time, $Max_Running_Concurrently, $Script_Expire_Time)

A função `SetProfileExecutionOptions()` permite alterar as opções de execução de um ou mais perfis de execução existentes na tabela `ExecutionProfile`. No caso do gerente WS não especificar qual perfil de execução (`$Profile_ID`) se deseja alterar, todos os perfis de execução terão as opções de execução (`$Max_Life_Time`, `$Max_Running_Concurrently` e `$Script_Expire_Time`) alteradas.

- function GetProfileExecutionOptions($Profile_ID)

A função `GetProfileExecutionOptions()` retorna informações sobre quais são as opções de execução de um ou mais perfis de execução existentes na tabela `Execution Profile`. No caso do gerente WS não especificar qual perfil de execução (`$Profile_ID`) se deseja consultar, o agente WS retornará informações referentes ao tempo máximo de execução, número máximo de *scripts* em execução associados ao mesmo perfil de execução e o tempo máximo que as informações ficarão disponíveis na tabela `Execution`, de todos perfis de execução.

- function SetProfileExecutionArgument($Profile_ID, $Argument)

A função `SetProfileExecutionArgument()` permite alterar o argumento de execução (`$Argument`) de um ou mais perfis de execução existentes na tabela `ExecutionPr ofile`. No caso do gerente WS não especificar qual perfil de execução (`$Profile_ID`) se deseja alterar, todos os perfis de execução terão os seus argumentos de execução alterados.

- function GetProfileExecutionArgument($Profile_ID)

A função `GetProfileExecutionArgument()` retorna informações sobre quais são os argumentos de execução existentes para um ou mais perfis de execução existentes na tabela `ExecutionProfile`. No caso do gerente WS não especificar qual perfil de execução (`$Profile_ID`) se deseja consultar, o agente WS retornará informações sobre argumentos de execução de todos os perfis de execução.

Controle de Execução

Operações de controle de execução sobre *scripts* de gerenciamento em execução são definidas neste grupo. São definidas operações que permitem o início, a suspensão, o reinicio e o término de uma ou mais execuções de *scripts* de gerenciamento.

A API dos serviços oferecidos neste grupo é a seguinte:

- function RunScript($Script_ID, $Profile_ID, $block=true)

A função `RunScript()` inicia a execução de um *script* de gerenciamento. O argumento `$Script_ID` é obrigatório, definindo qual é o *script* que terá a sua execução iniciada. Caso se deseje que todos os perfis de execução associados a um *script* tenham a sua execução iniciada, o argumento `$Profile_ID` deve ser igual a '*null*'. Por padrão, a função `RunScript` é bloqueante (`$block=true`), ou seja, o gerente WS ficará bloqueado ao invocá-la, aguardando até que a função retorne. Entretanto, ele pode optar pela função ser não-bloqueante alterando o valor de `$block` para '*false*'. A função retorna o identificador da execução `$Execution_ID` existente na tabela `Execution`, caso tenha tido sucesso. Caso contrário, mensagens de erro são retornadas ao gerente WS.

- function PauseScript($Execution_ID)

A função `PauseScript()` pausa a execução de um *script*. O argumento `$Execution_ID` é obrigatório e indica qual execução em andamento deseja-se pausar. A função consulta se o processo está em execução através do atributo `PID` associado a `$Execution_ID`, existente na tabela `Execution`. Caso esteja em execução, o sinal `SIGSTOP` é dado ao processo, indicando que ele deverá parar a sua execução. Caso o valor de `PID` não se encontre mais na tabela de processos do sistema operacional uma mensagem de erro é retornada ao gerente WS informando que o processo não está mais em execução.

- function PauseScripts($Script_ID, $Profile_ID)

A função `PauseScripts()` pausa a execução de *scripts* (`$Script_ID`) que estão associados a um ou mais perfis de execução. O único argumento obrigatório é `$Script_ID`. Se o argumento `$Profile_ID` não for especificado pelo gerente, todas as execuções de `$Script_ID` serão pausadas. Caso `$Profile_ID` seja especificado, apenas os *scripts* associados ao perfil `$Profile_ID` serão pausados. A função irá retorna um vetor informando o sucesso ou falha na pausa de cada uma das execuções de `$Script_ID`.

- function ResumeScript($Execution_ID)

A função `ResumeScript()` reinicia a execução de um *script*. O argumento `$Execution_ID` é obrigatório e indica qual execução em andamento deseja-se reiniciar. A função consulta se o processo está pausado através do valor de atributo `PID` referente ao identifi-

cador de execução `$Execution_ID`, ambos atributos da tabela `Execution`. Caso esteja pausado, o sinal `SIGCONT` é dado ao processo, indicando que ele voltará a executar. Caso o valor de `PID` não se encontre mais na tabela de processos do sistema operacional uma mensagem de erro é retornada ao gerente WS informando que o processo não está mais em execução.

- function ResumeScripts($Script_ID, $Profile_ID=null)

A função `ResumeScripts()` reinicia a execução de *scripts* (`$Script_ID`) que estão associados a um ou mais perfis de execução. O único argumento obrigatório é `$Script_ID`. Se o argumento `$Profile_ID` não for especificado, todas as execuções de `$Script_ID` serão reiniciadas. Caso `$Profile_ID` seja especificado, apenas os *scripts* associados ao perfil `$Profile_ID` serão reiniciados. A função irá retornar um vetor informando o sucesso ou falha no reinício de cada uma das execuções de `$Script_ID`.

- function StopScript($Execution_ID)

A função `StopScript()` termina a execução de um *script*. O argumento `$Execution_ID` é obrigatório e indica qual execução em andamento deseja-se terminar. A função consulta se o processo está em execução através do valor do atributo `PID` referente a `$Execution_ID`, ambos pertencentes a tabela `Execution`. Caso esteja em execução, o sinal `SIGKILL` é dado ao processo, indicando que ele deve encerrar a sua execução. Caso o valor de `PID` não se encontre mais na tabela de processos do sistema operacional uma mensagem de erro é retornada ao gerente WS informando que o processo não está mais em execução.

- function StopScripts($Script_ID, $Profile_ID=null)

A função `StopScripts()` termina a execução de *scripts* (`$Script_ID`) que estão associados a um ou mais perfis de execução. O único argumento obrigatório é `$Script_ID`. Se o argumento `$Profile_ID` não for especificado, todas as execuções de `$Script_ID` serão encerradas. Caso `$Profile_ID` seja especificado, apenas os *scripts* associados ao perfil `$Profile_ID` serão terminados. A função irá retorna um vetor informando o sucesso ou falha no encerramento de cada uma das execuções de `$Script_ID`.

Consulta de Estados e Resultados de Execuções

Neste grupo estão definidas operações WS que permitem consultar o estado da execução dos *scripts* de gerenciamento localizados no agente WS e, também, recuperar os resultados das execuções encerradas.

A API dos serviços oferecidos neste grupo é a seguinte:

- function GetExecutionStatus($Profile_ID, $Execution_ID)

A função `GetExecutionStatus()` retorna informações sobre o estado atual de exe-

cução de uma ou mais execuções (`$Execution_ID`) associadas a um perfil de execução (`$Profile_ID`). O único argumento obrigatório é `$Profile_ID`. Se o argumento `$Execution_ID` não for especificado pelo gerente WS, informações sobre o estado atual de execução de todas as execuções associadas ao perfil de execução `$Profile_ID` serão retornadas. As informações retornados são referentes aos valores dos atributos: `Script_Name`, `Execution_Status`, `Life_Time`, `Start_Time`, `Execution_Time` e `Expire_Time` existentes na tabela `$Execution`.

- function GetExecutionResults($Profile_ID, $Execution_ID)

A função `GetExecutionResults()` retorna informações sobre o resultado da execução de uma ou mais execuções (`$Execution_ID`) associadas a um perfil de execução (`$Profile_ID`). O único argumento obrigatório é `$Profile_ID`. Se o argumento `$Execution_ID` não for especificado pelo gerente WS, informações sobre o resultado de execução de todas as execuções associadas ao perfil de execução `$Profile_ID` serão retornadas. As informações retornadas são referentes aos valores dos atributos: `Exit_Code`, `Result_Time`, `CPU_Use`, `Mem_Use` existentes na tabela `$ExecutionResult`. Além disso, o conteúdo do arquivo definido em `Result_File` é retornado também. O conteúdo retornado se refere a valores enviados pelo *script* à saída padrão (*stdout*) e, também, a erros de execução, caso ocorridos, enviado à saída de erro padrão (*stderr*).

- function GetScriptID($Execution_ID)

A função `GetScriptID()` retorna o identificador de um *script* de gerenciamento (`$Script_ID`) associado a uma execução (`$Execution_ID`). Ela possui um único argumento obrigatório, `$Execution_ID`.

- function GetExecutionProfileID($Execution_ID)

A função `GetExecutionProfileID()` retorna o identificador de um perfil de execução (`$Profile_ID`) associado a uma execução (`$Execution_ID`). Ela possui um único argumento obrigatório, `$Execution_ID`.

***Funções compostas*:**

No agente WS também estão definidas funções compostas. Funções compostas são um agrupamento de funções definidas no agente WS, que quando executadas em uma determinada ordem, permitem a realização de uma ação. O agente WS desenvolvido neste trabalho fornece quatro funções compostas, sendo elas:

- function DownloadAndRunScript($arguments, $block=true)

A função `DownloadAndRunScript()` permite que o *download* de um *script* de gerenciamento e sua posterior execução sejam realizados. Essa função composta é um agru-

pamento das funções unitárias `DownloadScript()`, `CreateExecutionProfile()` e `RunScript()` apresentadas anteriormente. Os argumentos que devem ser passados a função são os mesmos argumentos que seriam passados as funções `DownloadScript()` e `CreateExecutionProfile()`. Não é necessário passar argumentos para a função `RunScript()` porque os argumentos necessários (`$Script_ID` e `$Profile_ID`) a ela, são retornados pelas funções `DownloadScript()` e `CreateExecutionProfile()`, respectivamente. A função `DownloadAndRunScript()` também permite a escolha de atuar de forma bloqueante ou não, através do argumento `$block`. Por padrão, ela atua de forma bloqueante. O retorno da função será o identificador da execução (`$Execution_ID`), no caso de sucesso. Caso algum erro ocorra, será retornada uma mensagem de erro.

- function DownloadRunAndGetResults($arguments)

A função `DownloadRunAndGetResults()` é similar a função descrita anteriormente, com apenas uma diferença. Além dela permitir o *download* de um *script* de gerenciamento e sua posterior execução, ela também retorna os resultados da execução. Essa função composta é um agrupamento das funções unitárias `DownloadScript()`, `CreateExecutionProfile()`,`RunScript()`,`GetExecutionStatus()` e `GetExecutionResults()` apresentadas anteriormente. A função é bloqueante, somente retornando quando os resultados da execução do *script* forem obtidos. Os argumentos que devem ser passados a essa função são os mesmos argumentos que seriam passados as funções `DownloadScript()` e `CreateExecutionProfile()`. Não é necessário passar argumentos para a função `RunScript()` devido às mesmas razões apresentadas na função descrita anteriormente. Também não é necessária a passagem de argumentos para `GetExecutionStatus()` e `GetExecutionResults()`, pois os argumentos necessários para as duas funções (`$Profile_ID` e `$Execution_ID`), são retornados pelas funções `CreateExecutionProfile()` e `RunScript()`, respectivamente. O retorno dessa função será os resultados das execuções de `$Script_ID` associado a `$Profile_ID`, no caso de sucesso. Caso algum erro ocorra, será retornada uma mensagem de erro.

- function PushAndRunScript($arguments, $block=true)

A função `PushAndRunScript()` permite que um *script* de gerenciamento seja enviado pelo gerente WS ao agente WS e sua posterior execução seja realizada. Essa função composta é um agrupamento das funções unitárias `PushScript()`,`CreateExecutionProfile()` e `RunScript()` apresentadas anteriormente. Os argumentos que devem ser passados a essa função são os mesmos argumentos que seriam passados às funções `PushScript()` e `CreateExecutionProfile()`. Não é necessário passar argumentos para a função `RunScript()` porque os argumentos necessários (`$Script_ID` e `$Profile_ID`) a ela, são retornados pelas funções `PushScript()` e `CreateExecutionProfile()`, respectivamente. A função `PushAndRunScript()` também permite a escolha de atuar de forma bloqueante ou não, através do argumento `$block`. Por padrão, ela atua de forma bloqueante. O retorno dessa função será o identificador da execução (`$Execution_ID`), no caso de sucesso. Caso algum erro ocorra, será retornada uma mensagem de erro.

- function PushRunAndGetResults($arguments)

A função `PushRunAndGetResults()` é similar a função `DownloadRunAndGetResults()` descrita anteriormente. A única diferença entre elas é na forma que o *script* de gerenciamento é transferido. Neste caso, pelo envio do gerente WS ao agente WS do *script*. Essa função composta é um agrupamento das funções unitárias `PushScript()`, `CreateExecutionProfile()`,`RunScript()`,`GetExecutionStatus()` e `GetExecutionResults()` apresentadas anteriormente. A função é bloqueante, somente retornando quando os resultados da execução do *script* forem obtidos. Os argumentos que devem ser passados a essa função são os mesmos argumentos que seriam passados as funções `PushScript()` e `CreateExecutionProfile()`. Não é necessário passar argumentos para a função `RunScript()` devido às mesmas razões apresentadas na função descrita anteriormente. Também não é necessário passar argumentos para `GetExecutionStatus()` e `GetExecutionResults()`, pois os argumentos necessários para as duas funções (`$Profile_ID` e `$Execution_ID`), são retornados pelas funções `CreateExecutionProfile()` e `RunScript()`, respectivamente. O retorno dessa função será os resultados das execuções de `$Script_ID` associado a `$Profile_ID`, no caso de sucesso. Caso algum erro ocorra, será retornada uma mensagem de erro.

4.3 Gerente Web Services

Tendo sido apresentadas as implementações dos protótipos de um *gateway* em nível de serviço e de um agente orientado a serviços baseado em Web Services, nesta seção será apresentado o gerente WS responsável pela interação com os dois protótipos.

O gerente WS desenvolvido faz parte do ambiente de gerenciamento de redes QAME (*QoS-Aware Management Environment*) (GRANVILLE et al., 2001) que é um sistema Web de gerenciamento de redes construído como um conjunto de *scripts* codificados em PHP. Ele suporta tarefas convencionais de gerenciamento tais como monitoração de dispositivos de rede (baseado na solução RRDTool (OETIKER, 2003)), tratamento de notificações (baseado no serviço de recepção de *trap* oferecido pela implementação NET-SNMP (SOURCEFORGE TEAM, 1992)) e a descoberta de serviços e redes. O ambiente QAME foi desenvolvido para suportar, de uma maneira fácil, a inclusão de novos módulos, quando necessário.

Essa característica permitiu que o novo módulo desenvolvido neste trabalho fosse incluído no ambiente QAME para controlar os protótipos desenvolvidos. Através desse módulo, um gerente de rede pode, por meio da interface Web oferecida pelo QAME, manipular um dispositivo de rede que funcione como um *gateway* WS para SNMP em nível de serviço ou um agente orientado a serviços baseado em Web Services. Salienta-se que a interface Web de acesso aos protótipos é a mesma, ou seja, o gerente de redes irá interagir da mesma maneira tanto com o agente WS quanto com o *gateway* em nível de serviço. Uma vez que as telas de acesso são a mesma para os dois protótipos, o que será visto a seguir são telas que demonstram um exemplo de interação do módulo desenvolvido com o protótipo de um agente WS.

No caso do QAME, antes de se solicitar a realização de tarefas de gerenciamento em um dispositivo, é necessário que o administrador saiba quais funcionalidades este dispositivo possui e quais tarefas de gerenciamento o dispositivo tem capacidade de realizar.

A Figura 4.11 mostra a interface Web fornecida pelo QAME para a consulta das propriedades (funcionalidades) de um dispositivo de rede.

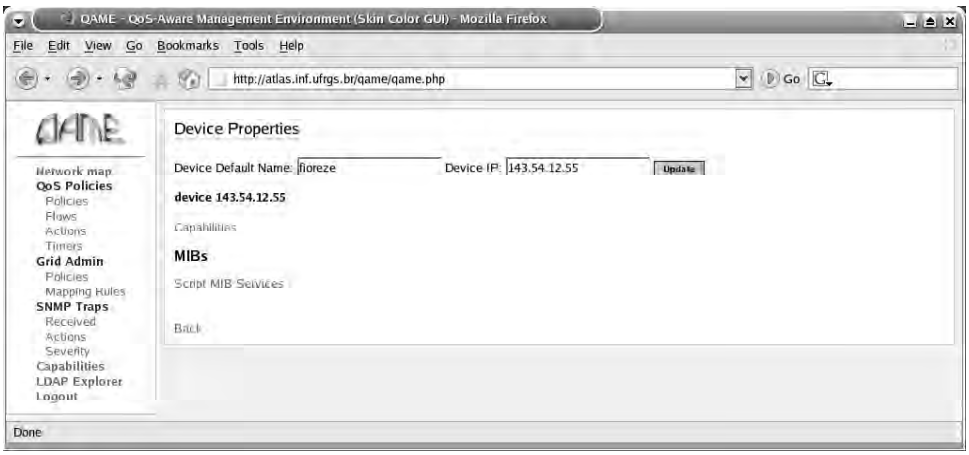

Figura 4.11: Propriedades de um dispositivo de rede

Caso o gerente deseje saber quais são as funcionalidades que um dispositivo de rede possui (Figura 4.12), ele deve acessar o *link* '*Capabilities*' existente na tela *Device Properties*. Tanto a *listbox* da esquerda quanto a *listbox* da direita mostradas na Figura 4.12 listam as propriedades reconhecidas pelo QAME. Na *listbox* à direita são listadas as capacidades suportadas pelo dispositivo de rede consultado. Verifica-se pela figura que o dispositivo consultado tem a capacidade de fornecer gerenciamento por delegação baseado em Web Services, podendo, portanto, atuar como um agente WS.

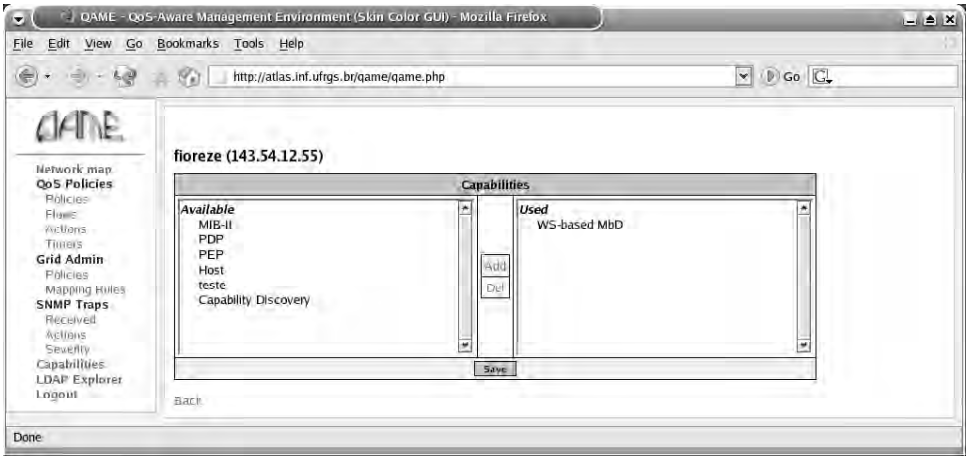

Figura 4.12: Capacidades de um dispositivo de rede

Para o gerente WS acessar as operações WS que ele pode invocar no agente WS, ele deve acessar o *link* '*Script MIB Services*' existente na tela *Device Properties* (Figura 4.11). Acessando o *link*, a lista dos serviços oferecidos é mostrada na forma de grupos de operações relacionadas (Figura 4.13). Salienta-se que a lista de serviços oferecidas é a mesma tanto para o caso de interação com um *gateway* em nível de serviço quanto para um agente WS.

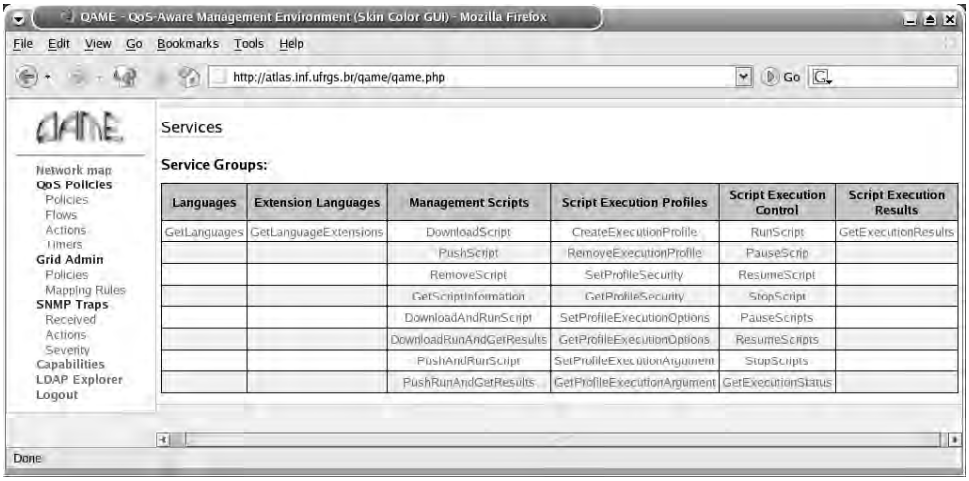

Figura 4.13: Serviços disponibilizados pelo agente WS e pelo *gateway* em nível de serviço

A interação do gerente WS com o agente WS será apresentada através de dois exemplos. Em cada um deles será demonstrado um caso onde o gerente WS deseja transferir um *script* de gerenciamento pelo método **push**, executá-lo e, após isso, obter os resultados da execução. O *script* de gerenciamento é um simples *shell script* chamado '*ifconfig.sh*' que simplesmente retorna o estado de uma ou mais interfaces no dispositivo contendo o agente WS. A diferença entre os exemplos está na forma que o gerente WS realiza as ações desejadas. Primeiramente, será apresentado um caso onde o gerente WS utiliza funções unitárias fornecidas pelo agente WS e, por fim, um caso onde funções compostas são utilizadas.

4.3.1 Utilizando funções unitárias

Antes de o gerente WS solicitar ao agente WS a execução de um *script*, ele necessita transferí-lo, pelo método **push**, ao agente WS. Para realizar a transferência a operação `PushScript` pertencente ao grupo *Management Scripts* é invocada pelo gerente WS. Os campos que necessitam ser preenchidos são apresentados na Figura 4.14.

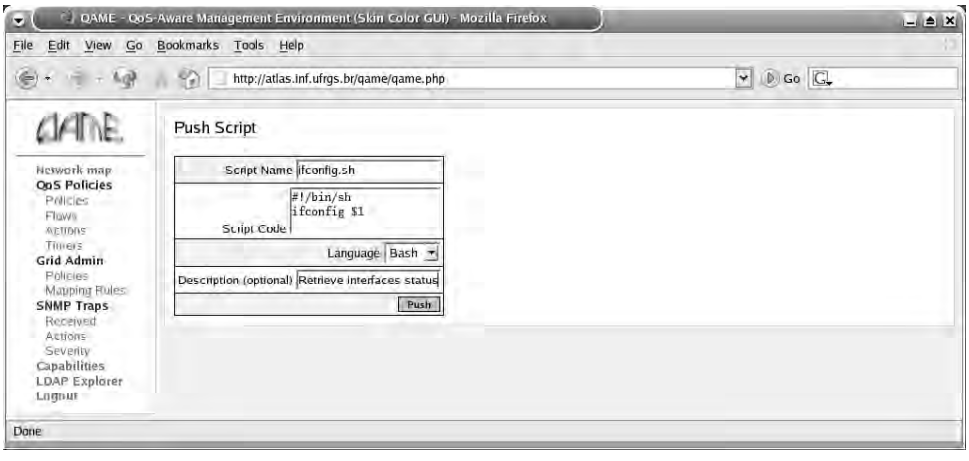

Figura 4.14: Campos da operação `Push`

No campo `Script Name` o gerente WS preenche o nome do *script* (‘*ifconfig.sh*’), nos campos `Script Code` e `Language` são colocados o código do *script* e em qual linguagem ele está codificado (‘*Bash*’), respectivamente. O campo `Description` é opcional e serve para dar uma breve descrição sobre o *script*. Ao ser pressionado o botão `Push`, o processo de transferência do *script* inicia. No caso de sucesso no processo de transferência, o gerente WS recebe o identificador do *script* no agente WS que, por exemplo, é ‘*96*’.

Uma vez que o *script* tenha sido transferido com sucesso, o gerente pode criar um perfil de execução para o mesmo, através da operação `CreateExecutionProfile`. Ao ser invocada, essa operação apresenta todos os *scripts* de gerenciamento que estão localizados no dispositivo de rede que atua como agente WS (Figura 4.15).

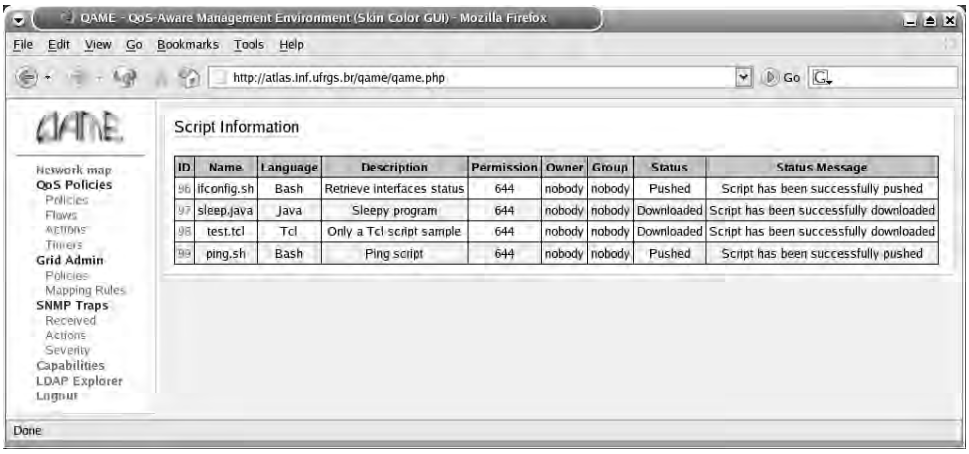

Figura 4.15: Exemplo de *scripts* de gerenciamento localizados no dispositivo

Verifica-se pela Figura 4.15 que *script* ‘*ifconfig.sh*’ está na relação de *scripts* enviada pelo agente WS. Para criar o perfil de execução para o *script*, o gerente deve acessar o valor ‘*96*’ do atributo `ID`. Uma vez acessado, uma nova tela será apresentada ao gerente, onde na qual ele necessita preencher os campos referentes a um perfil de execução (Figura 4.16). Salienta-se que o campo `Script ID` já vem com o seu valor preenchido.

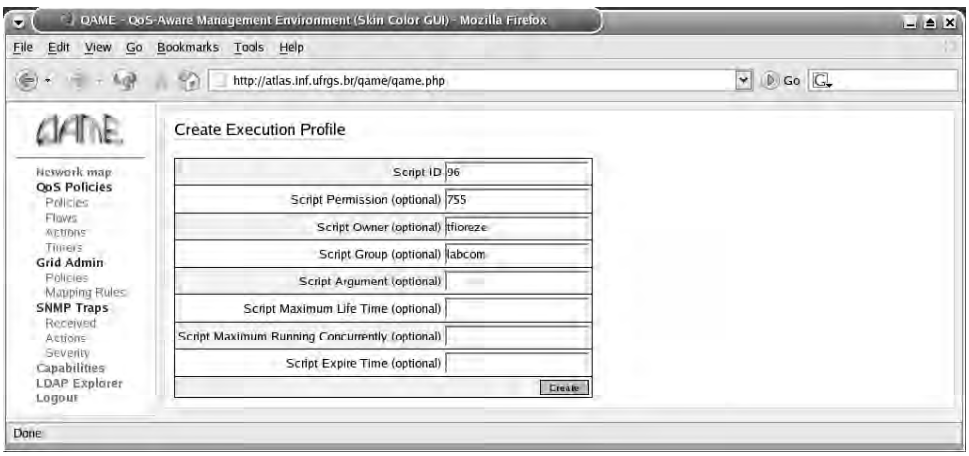

Figura 4.16: Campos da operação `CreateExecutionProfile`

Ao pressionar o botão `Create`, o perfil de execução criado pelo gerente WS para o *script* '*ifconfig.sh*' será armazenado no agente WS e o identificador de perfil '*85*', por exemplo, é retornado, confirmando, dessa forma, o sucesso da operação. Uma vez que o perfil de execução tenha sido criado para o *script* '*ifconfig.sh*', a execução do *script* pode ser iniciada através da operação `RunScript`. Ao ser invocada, essa operação lista todos os perfis de execução existentes (Figura 4.17).

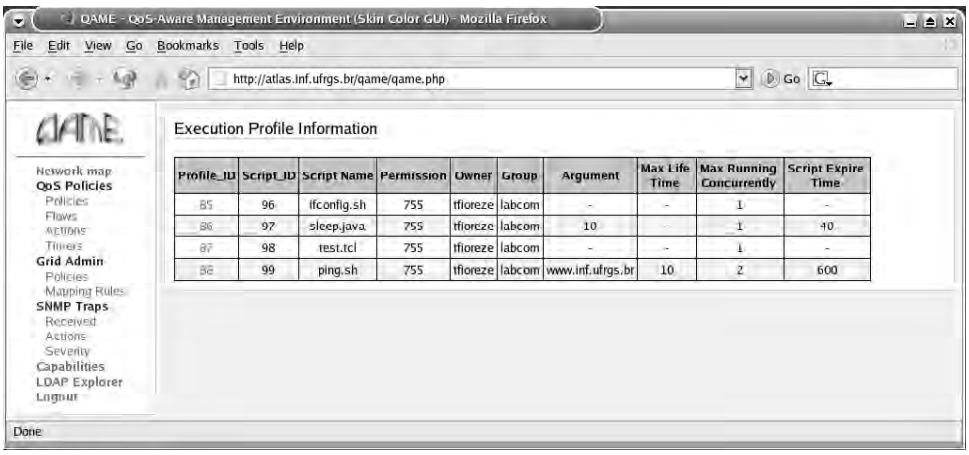

Figura 4.17: Exemplo de perfis de execução

Para iniciar a execução do *script*, o gerente deve acessar o valor '*85*' do atributo `Profile_ID`. Uma vez acessado, o gerente pode decidir se a operação `RunScript` será bloqueante ou não. Isso é feita através da seleção da *checkbox* `Blocking` mostrada na Figura 4.18.

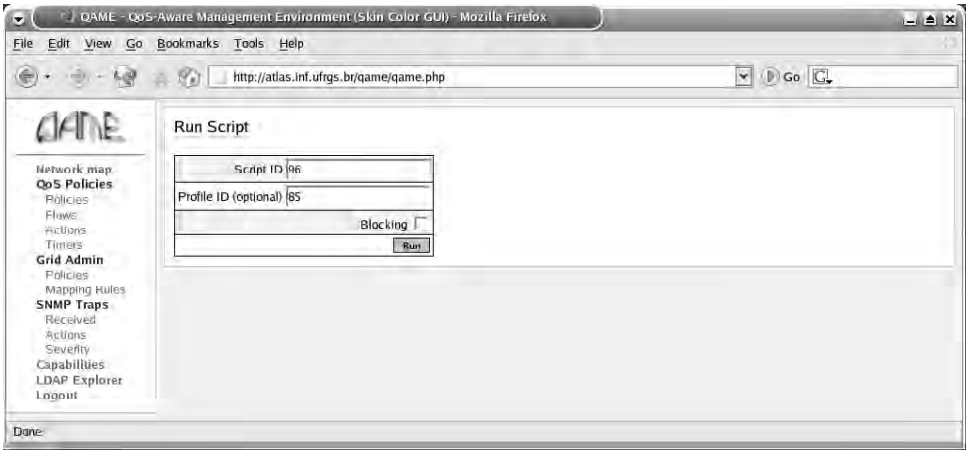

Figura 4.18: Campos da operação `RunScript`

Ao pressionar o botão `Run`, a execução do *script* pelo agente WS tem início e caso o gerente WS não tenha selecionado a opção `Blocking`, o agente WS retornará de imediato o identificador de execução. Caso contrário, o identificador será retornado somente após o término da execução do *script*. Pelo exemplo apresentado na Figura 4.18, o gerente optou por não ficar aguardando o retorno da operação e recebeu, por exemplo, o identificador de execução '*91*' do agente WS assim que a execução do *script* '*ifconfig.sh*' teve início.

Para monitorar o andamento da execução do *script* no agente WS, o gerente WS utiliza a operação `GetExecutionStatus`. Ao invocar essa operação, todos os estados de execução dos *scripts* são listados (Figura 4.19).

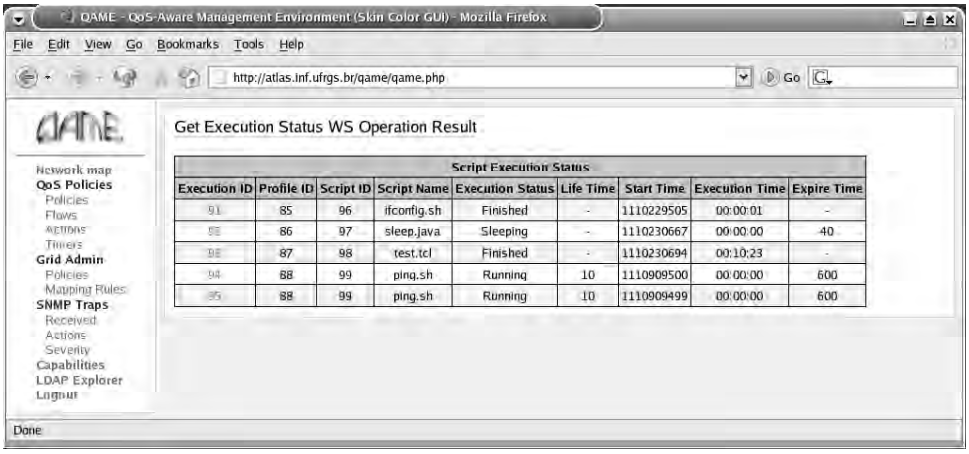

Figura 4.19: Exemplo de estados de execução

Verifica-se pela figura que o *script* '*ifconfig.sh*' não está mais executando, estando o seu estado de execução como '*Finished*'. Uma vez que o *script* tenha encerrado a sua execução, o gerente WS pode consultar o resultado da execução, através do acesso ao valor '*91*' do atributo `Execution_ID`. Uma vez acessado, o resultado da execução do *script* é apresentado (Figura 4.20).

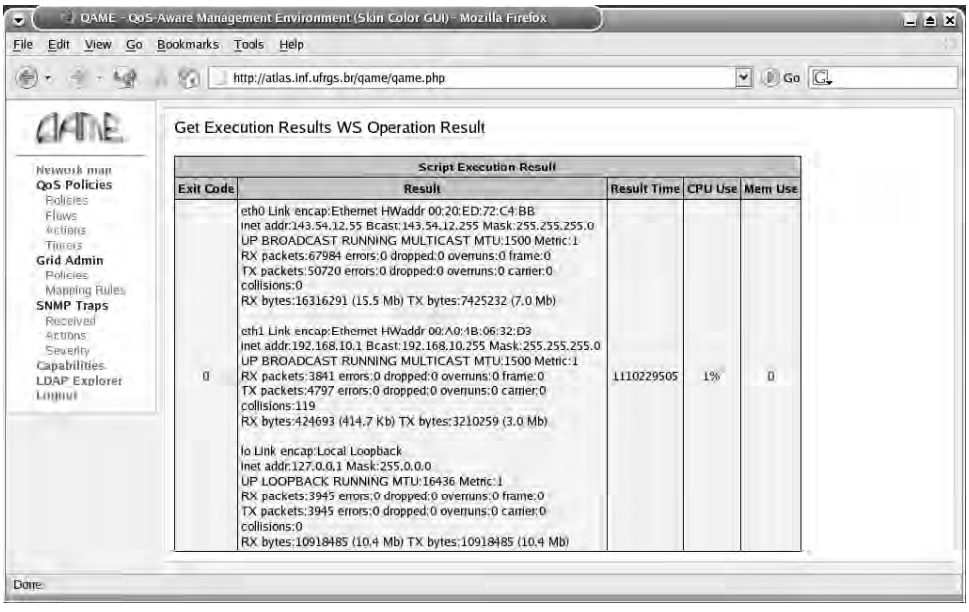

Figura 4.20: Exemplo de resultados de execução

4.3.2 Utilizando funções compostas

Uma maneira mais simples do gerente WS realizar o mesmo procedimento apresentado anteriormente no agente WS, é através da utilização de funções compostas. Funções

compostas são conjuntos de funções unitárias agrupadas em uma única função e que seguem uma ordem de execução para realizar uma determinada ação. Por exemplo, caso o gerente WS desejasse realizar o mesmo procedimento descrito anteriormente, ele poderia fazer isso invocando a operação composta `PushRunAndGetResults`.

A função `PushRunAndGetResults` é um agrupamento das funções unitárias `PushScript()`, `CreateExecutionProfile()`, `RunScript()`, `GetExecutionStatus()` e `GetExecutionResults()`. Os campos da função são basicamente os mesmos campos das funções unitárias: `PushScript` e `CreateExecutionProfile` (Figura 4.21).

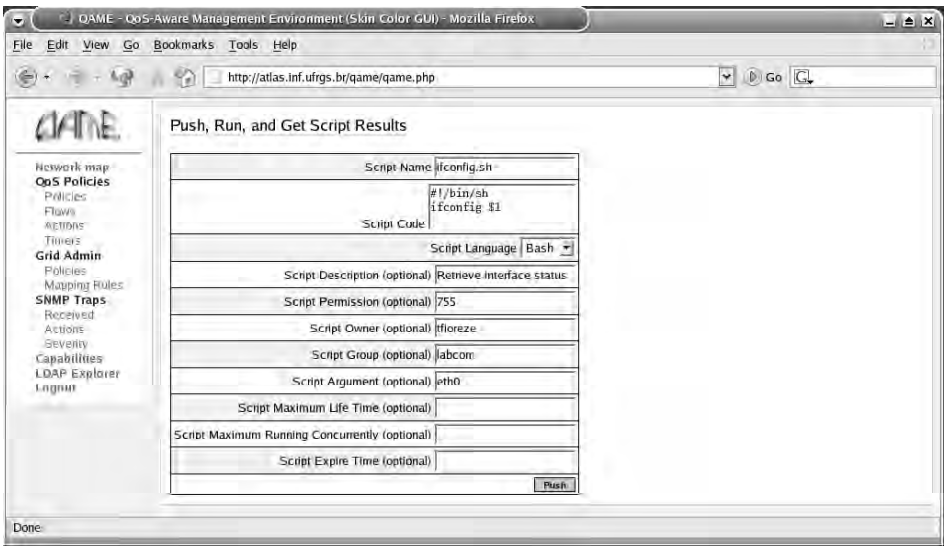

Figura 4.21: Campos da operação `PushRunAndGetResults`

Ao pressionar o botão `Push`, o primeiro passo da função `PushRunAndGetResults` será a transferência do *script* '*ifconfig.sh*' pelo método **push**. Uma vez que o *script* tenha sido transferido com sucesso, a função realiza a criação do perfil de execução baseada nos valores dos campos preenchidos e inicia a execução do *script*. Tendo sido encerrada a execução do *script* '*ifconfig.sh*', o resultado da execução é retornado pelo agente WS ao gerente WS (Figura 4.22).

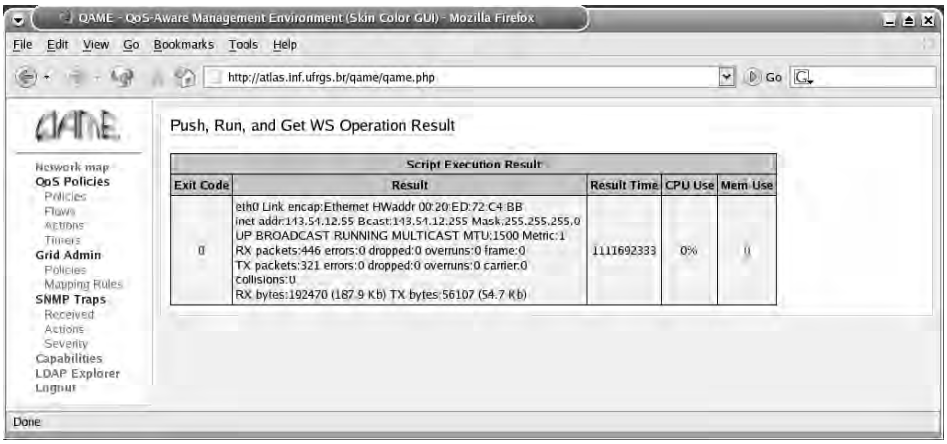

Figura 4.22: Exemplo de resultado da operação `PushRunAndGetResults`

5 RESULTADOS

No capítulo anterior, os dois protótipos desenvolvidos neste trabalho e o ambiente de gerenciamento QAME foram apresentados. Foi apresentada também a forma que o gerente (QAME) e os protótipos interagem para realizarem uma ação de gerenciamento. Uma vez encerrada a etapa de desenvolvimento dos protótipos, faz-se necessária uma avaliação dos mesmos para verificar questões de desempenho.

Neste capítulo, o *gateway* em nível de serviço e o agente orientado a serviços baseado em Web Services são comparados com outras duas soluções apresentadas anteriormente nesta dissertação: o *gateway* em nível de objeto e o gerenciamento baseado em SNMP. A comparação foi feita no âmbito do gerenciamento por delegação e teve como parâmetros de avaliação o tráfego total gerado e o tempo de resposta percebido.

Inicialmente, os cenários de avaliação utilizados para a análise dos protótipos serão apresentados. Em seguida, os requisitos de hardware e software das máquinas utilizadas nos cenários de avaliação são descritos. Após isso, a metodologia de avaliação das quatro soluções de gerenciamento de dispositivos é apresentada. Por fim, o capítulo é encerrado com a apresentação dos resultados obtidos.

5.1 Cenários de avaliação

O *gateway* em nível de serviço e o agente orientado a serviços baseado em Web Services foram comparados com o *gateway* em nível de objeto e o gerenciamento baseado em SNMP, sendo utilizado, neste último, o Jasmin como agente SNMP. O *gateway* em nível de protocolo não foi utilizado na comparação, por ter apresentado um comportamento pior que o *gateway* em nível de objeto em um trabalho anterior (NEISSE et al., 2004) desenvolvido pelo Grupo de Redes do Instituto de Informática da UFRGS.

Para realizar a comparação das quatro soluções de gerenciamento, três cenários de avaliação foram construídos. Em cada cenário existe a figura de um TLM e um MLM (seção 2.1). O TLM é representado pelo ambiente de gerenciamento de redes QAME, que incorpora três módulos específicos para cada um dos MLMs utilizados na avaliação. O número de MLMs utilizados foi três e cada um deles é baseado em uma única solução de gerenciamento, sendo elas: SNMP, *gateways* Web Services para SNMP e Web Services. A metodologia de avaliação utilizada nos três cenários consistiu em o TLM solicitar ao MLM a transferência, pelo método **pull**, de um *script* de gerenciamento localizado em um repositório de *scripts* e, após isso, o TLM solicita ao MLM a execução do *script*. Mais detalhes serão apresentados na seção 5.3.

Cada um dos cenários de avaliação possui particularidades em relação a comunicação entre o TLM e o MLM, e, também, em relação aos elementos que compõem cada MLM. Em comum, todos os cenários hospedam um servidor Web e um repositório de *scripts*

de gerenciamento nos MLMs. Como o objetivo deste trabalho foi concentrar a avaliação apenas no desempenho da comunicação TLM-MLM, essa hospedagem tem por objetivo minimizar a influência de tráfego externo à avaliação, principalmente em relação ao tempo de resposta. Embora esse tipo de cenário não condiza com a realidade do gerenciamento de redes, ele proporciona uma avaliação mais justa para as 4 soluções comparadas neste trabalho, uma vez que não existem fatores externos (e.g. tráfego concorrente) interferindo nos cenários de avaliação.

O primeiro cenário de avaliação é apresentado na Figura 5.1. Nele, estão presentes o módulo do QAME (TLM) e o dispositivo gerenciado (MLM), sendo a comunicação entre eles realizada através de SNMP sobre UDP (*User Datagram Protocol*).

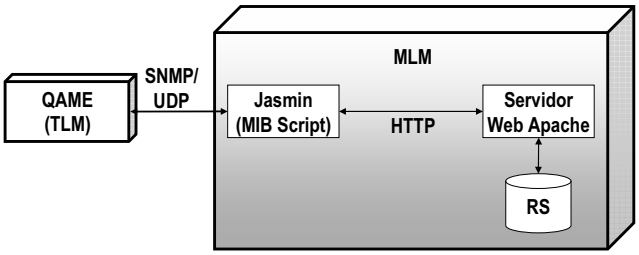

Figura 5.1: Cenário de avaliação utilizando SNMP

O MLM é baseado em SNMP, sendo composto por um agente SNMP (Jasmin), um servidor Web Apache e um repositório de *scripts* de gerenciamento (RS). O Jasmin é responsável pela transferência e execução do *script* de gerenciamento informado pelo TLM. O processo de transferência é realizado contactando-se o servidor Web Apache, através de HTTP, que ao receber a solicitação de *download* do Jasmin, obtém o *script* desejado no repositório de *scripts* e o envia ao Jasmin. Uma vez que o Jasmin receba o *script*, ele prepara a execução do mesmo e, após isso, o executa.

No cenário de avaliação apresentado na Figura 5.2 estão presentes o módulo do QAME (TLM) e o dispositivo gerenciado (MLM), sendo a comunicação entre eles realizada através de SOAP sobre HTTP.

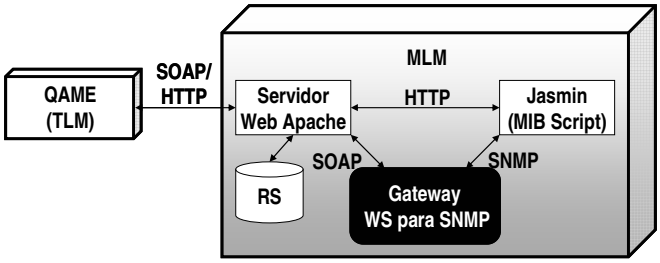

Figura 5.2: Cenário de avaliação utilizando *gateways* WS para SNMP

O MLM é baseado em *gateways* WS para SNMP, sendo ele composto por dois *gateways* WS para SNMP: o *gateway* em nível de objeto e o *gateway* em nível de serviço, desenvolvido neste trabalho. Salienta-se que mesmo que os dois *gateways* façam parte do mesmo MLM, eles são avaliados em momentos diferentes. Além dos *gateways*, também fazem parte do MLM o Jasmin, um servidor Web Apache e um repositório de *scripts* de gerenciamento (RS). Embora a configuração normal seria tanto o agente SNMP quanto os *gateways* WS para SNMP estarem em localizações diferentes, eles foram dispostos na

mesma entidade (MLM) para, como mencionado anteriormente, minimizar a influência de tráfego externo à avaliação.

Os *gateways* WS para SNMP têm responsabilidade de traduzir as solicitações Web Services do TLM em mensagens SNMP enviadas ao Jasmin. As solicitações feitas pelo TLM se referem a transferência e execução do *script* de gerenciamento utilizado na avaliação. Em cada *gateway* WS para SNMP avaliado, a solicitação de transferência enviada pelo TLM é recebida pelo *gateway*, que a traduz em mensagens SNMP enviadas ao Jasmin. Este, ao receber as mensagens enviadas pelo *gateway* avaliado, inicia a transferência do *script* entrando em contato com o servidor Web Apache através de HTTP. Então, o servidor Web Apache obtém o *script* no repositório de *scripts* e o envia para o Jasmin, que inicia a execução do *script* assim que ele for totalmente transferido e estiver apto a ser executado.

No último cenário de avaliação representado pela Figura 5.3, a comunicação entre o TLM e o MLM é feita também através de SOAP sobre HTTP, mas, ao contrário do cenário apresentado anteriormente, o MLM é baseado em Web Services apenas. Além disso, o MLM é composto pelo agente WS desenvolvido neste trabalho, um servidor Web Apache, um repositório de *scripts* de gerenciamento (RS) e uma base de dados MySQL. A base de dados MySQL é utilizada pelo agente WS para o armazenamento de informações de gerenciamento. O agente WS é responsável pela transferência do *script* de gerenciamento e pela sua execução. O processo de transferência e execução do *script* '*wait.jar*' é similar ao processo descrito no caso do MLM baseado em SNMP.

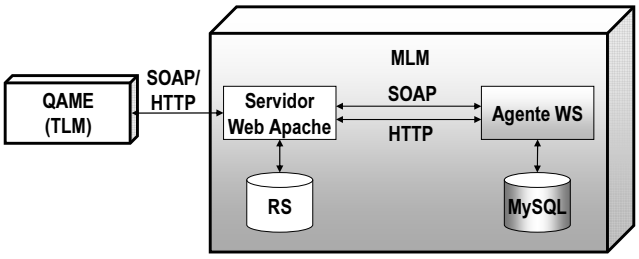

Figura 5.3: Cenário de avaliação utilizando Web Services

Salienta-se que tanto o TLM quanto o MLM estão localizados em máquinas diferentes. Na seção a seguir serão apresentadas a configuração de cada máquina e, também, como elas estão interligadas fisicamente.

5.2 Configuração das máquinas

Para a realização das avaliações do trabalho, duas máquinas foram utilizadas: uma que executa o sistema QAME (TLM) e outra que executa os MLMs descritos na seção 5.1. As máquinas foram conectadas uma a outra utilizando-se placas de redes de 10Mbps e um cabo de rede cruzado. Salienta-se que além do tráfego de dados gerado entre o TLM e MLM, não existe nenhum outro tráfego concorrente. A Figura 5.4 resume os requisitos de hardware e software de cada máquina utilizada nas avaliações.

Os requisitos de software são particulares a cada cenário de avaliação apresentado, sendo apenas o servidor Web Apache comum a todos. O cenário de avaliação mostrado na Figura 5.1 requer como requisitos mínimos de software a utilização do NET-SNMP e do PHP para o TLM e do JDK 1.1.8 e do UCD-SNMP 4.2.6 para o MLM baseado em SNMP. Esses requisitos do MLM são necessários para o funcionamento do Jasmin.

		TLM	MLM - SNMP	MLM – WS para SNMP	MLM - WS
Hardware	**Processador**	Intel Pentium IV 2.40 GHz	AMD K6-II 550 MHz	AMD K6-II 550 MHz	AMD K6-II 550 MHz
	Memória RAM	256 MB	128 MB	128 MB	128 MB
Software	**Sistema Operacional**	Red Hat 9 (2.4.20-31..9)	Suse 6.4 (2.2.14)	Suse 6.4 (2.2.14)	Suse 6.4 (2.2.14)
	Servidor Web Apache	2.0.50	2.0.50	2.0.50	2.0.50
	Base de Dados MySQL	-	-	-	3.23.58
	Java	-	JDK 1.1.8	JDK 1.1.8	JDK 1.1.8
	PHP	4.3.8	-	4.3.8	4.3.8
	SOAP	NuSOAP 0.6.3	-	NuSOAP 0.6.3	NuSOAP 0.6.3
	SNMP	Net-SNMP 5.1.1	UCD-SNMP 4.2.6	UCD-SNMP 4.2.6	-

Figura 5.4: Configuração do *Top-Level* (TLM) e dos *Mid-Level* (MLMs) *Managers*

No caso dos ambientes apresentados nas Figuras 5.2 e 5.3, os requisitos mínimos para o TLM são o PHP e a biblioteca NuSOAP, sendo esses requisitos necessários também aos MLMs baseados em Web Services e em *gateways* WS para SNMP. Além desses requisitos, os MLMs baseados em *gateways* WS para SNMP e em Web Services utilizam o JDK 1.1.8, presente no MLM baseado em SNMP, para que o *script* '*wait.jar*' seja executado pelo mesmo ambiente de execução em todos os MLMs. No caso particular do MLM baseado em Web Services, acrescenta-se o MySQL como requisito de software e retira-se o UCD-SNMP. Salienta-se que o UCD-SNMP foi utilizado no MLM baseado em *gateways* WS para SNMP por uma razão de reaproveitamento de recursos, uma vez que o Jasmin também utiliza UCD-SNMP. Entretanto, caso os *gateways* e o Jasmin estivessem em máquinas diferentes, o MLM em questão poderia utilizar outra implementação do SNMP (e.g. NET-SNMP).

5.3 Metodologia de avaliação

A metodologia de avaliação utilizada foi a mesma para os três cenários de avaliação apresentados na seção 5.1. Na metodologia, um *script* codificado em Java, denominado '*wait.jar*', foi utilizado, consistindo o seu funcionamento em iniciar a sua execução, "dormir" por um tempo determinado (definido através da passagem de argumentos) e, por fim, encerrar a sua execução. Esse programa não realiza nenhuma tarefa de gerenciamento, mas é suficiente para verificar o comportamento das quatro soluções avaliadas neste trabalho em relação aos parâmetros de avaliação tráfego gerado e tempo de resposta percebido. A metodologia de avaliação consistiu nos seguintes passos:

- Primeiramente, o TLM solicita ao MLM a transferência do *script* '*wait.jar*' através do método de transferência **pull**;
- Até que o MLM termine o processo de transferência do *script* '*wait.jar*' localizado no repositório de *scripts*, O TLM fica bloqueado. Isso ocorre, pois o processo de transferência é bloqueante;
- Uma vez que a transferência tenha encerrado, o TLM bloqueia novamente até que o MLM inicie o ambiente de execução (Java) e deixe o *script* '*wait.jar*' apto a ser executado;
- Sendo possível a execução do *script*, o TLM solicita ao MLM o início da execução do *script* '*wait.jar*' passando, como argumento, o número de segundos que ele deve

"dormir" até o encerramento da sua execução (os tempos 0, 5, 10 e 15 segundos foram utilizados nos testes);

- O TLM bloqueia novamente até que a execução do *script* se encerre;
- Finalmente, o TLM obtém o resultado da execução do *script* '*wait.jar*', no MLM, que deve ser igual a '1', indicando com isso um encerramento normal da sua execução.

Esse procedimento foi repetido 30 vezes para cada um dos 4 tempos de execução (tempos 0, 5, 10 e 15 segundos) passados ao *script* '*wait.jar*', o que totalizou 120 amostras para cada solução avaliada. As amostras foram analisadas considerando um intervalo de confiança de 95%.

Nas amostras coletadas, dois parâmetros foram avaliados: tráfego total gerado e tempo de resposta percebido. O parâmetro tráfego total gerado se refere a quantidade total de bytes gerada pela comunicação entre o TLM e o MLM. Já o parâmetro tempo de resposta percebido se refere ao tempo que o TLM leva para perceber que a execução de um *script* de gerenciamento encerrou-se no MLM. O tempo de resposta percebido é calculado através do registro do tempo da primeira mensagem (SNMP ou SOAP/HTTP) partindo do TLM em direção ao MLM (primeiro passo descrito anteriormente) e pelo registro do tempo da última mensagem partindo do MLM em direção ao TLM (último passo descrito anteriormente). Além da avaliação do SNMP e do SOAP/HTTP, foi analisada também a utilização de compressão nas mensagens SOAP/HTTP. Nesta dissertação foram considerados os parâmetros tráfego gerado e tempo de resposta percebido porque tais parâmetros são comumente utilizados nas pesquisas genéricas comparando WS com SNMP. Assim, considerando-se esses mesmos parâmetros é possível colocar os resultados obtidos nesta dissertação em perspectiva com os resultados das pesquisas genéricas, permitindo dessa forma uma análise por comparação mais clara.

5.4 Tráfego gerado

Intuitivamente, pode-se pensar que a quantidade de bytes gerada pelo SNMP para transferir, executar e consultar o resultado da execução do *script* '*wait.jar*', no caso do cenário de avaliação utilizando SNMP (Figura 5.1), é menor que a quantidade de bytes gerada pelos cenários de avaliação utilizando Web Services (Figura 5.3) e *gateways* WS para SNMP (Figura 5.2). Esse pensamento é devido ao fato de que uma mensagem SNMP possui um tamanho menor que uma mensagem SOAP, uma vez que SOAP é baseado em XML, que é orientado a texto, enquanto que SNMP é um protocolo codificado binariamente. Além disso, como o SNMP confia no protocolo de transporte UDP, não existe uma resposta de confirmação de recebimento de uma mensagem SNMP, o que acaba gerando uma quantidade menor de dados trafegados entre o TLM e o MLM. Por outro lado, quando se usa SOAP/HTTP, o protocolo de transporte TCP é utilizado, fazendo com que respostas de confirmação de recebimento de mensagens SOAP/HTTP sejam retornadas, gerando, dessa forma, uma quantidade maior de bytes.

Embora sejam afirmações verdadeiras, existem casos onde a quantidade de bytes gerada pelo SNMP/UDP é maior que a gerada pelo SOAP/HTTP/TCP. Um exemplo disso ocorre quando o módulo do TLM realiza uma quantidade elevada de consultas de informações de gerenciamento no MLM baseado em SNMP em um espaço de tempo muito curto, gerando, dessa maneira, uma quantidade de bytes trafegados bastante elevada. Esse

tipo de situação caracteriza-se por não haver nenhum controle na consulta de informações (*polling control*). Por outro lado, no caso do TLM responsável por comunicar-se com o *gateway* em nível de objeto, as mensagens SOAP/HTTP partem em um espaço de tempo maior, devido ao procedimento utilizado pelo TCP na entrega das mensagens (*three-way handshake*) e, também, pelo tempo de processamento das bibliotecas (NuSOAP no caso deste trabalho), gerando, assim, uma quantidade menor de bytes trafegados.

Logo, para uma comparação mais justa entre o cenário de avaliação utilizando SNMP e o *gateway* em nível de objeto, foi utilizado neste trabalho o controle de *polling* no módulo do TLM responsável pela comunicação com o MLM baseado em SNMP. Medições de rede realizadas no caso de utilização do *gateway* em nível de objeto mostraram que cada mensagem SOAP/HTTP partia do TLM em intervalos de 0,5 segundo. Assim, esse valor foi utilizado como o intervalo de tempo para o envio de mensagens SNMP partindo do TLM em direção ao MLM baseado em SNMP. No caso do *gateway* em nível de serviço e do agente Web Services não foi necessária a utilização do controle de *polling* devido ao próprio funcionamento dessas duas implementações (Seções 3.1 e 3.2, respectivamente), que possuem como característica principal, a redução da quantidade de mensagens enviadas pelo TLM.

Na primeira avaliação realizada (Figura 5.5), não foi utilizado controle de *polling* no módulo do TLM responsável por gerenciar o MLM baseado em SNMP. Além disso, não foi utilizada compressão de dados nas mensagens trocadas entre o módulo do TLM e os MLMs baseados em *gateways* WS para SNMP e Web Services.

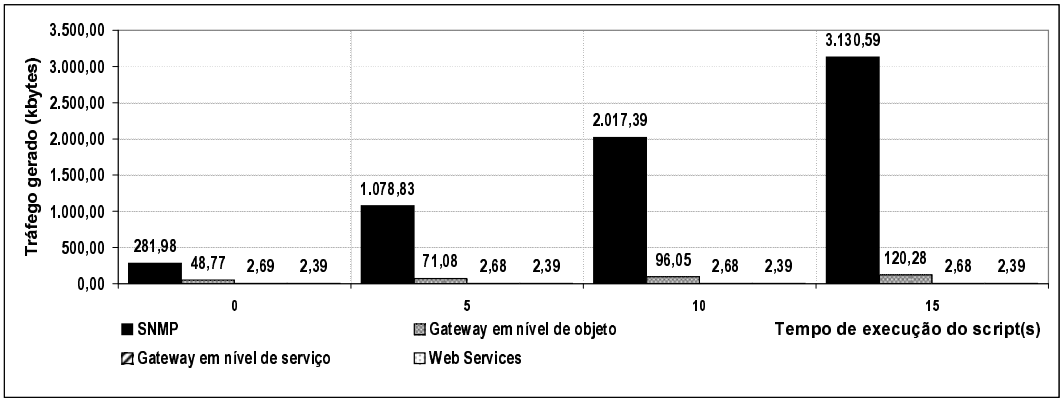

Figura 5.5: Tráfego gerado: TLM sem controle de *polling*

Verifica-se pela Figura 5.5, que sem controle de *polling* no módulo do TLM que comunica-se com o MLM baseado em SNMP, o tráfego gerado pelo SNMP cresce consideravelmente conforme o tempo de execução do *script* '*wait.jar*' aumenta. Isso ocorre porque o módulo do TLM envia uma quantidade bastante elevada de requisições SNMP em um intervalo muito curto de tempo em, principalmente, dois passos descritos anteriormente: quando ele deseja saber o término da transferência e o encerramento da execução do *script* '*wait.jar*'. No caso do *gateway* em nível de objeto, o módulo do TLM responsável por comunicar-se com ele realiza as mesmas consultas feitas pelo módulo do TLM comentado anteriormente. Entretanto, as consultas são feitas utilizando SOAP sobre HTTP que, devido a utilização do TCP e da biblioteca NuSOAP, partem em intervalos de tempo maiores, gerando com isso menos tráfego. O tráfego gerado no caso do *gateway* em nível de serviço e do agente WS se mantém constante independente do tempo de execução do *script* '*wait.jar*'. Isso ocorre devido ao funcionamento dessas duas tecnologias.

Como elas foram construídas com base nos serviços que a MIB Script oferece, um dos serviços implementado neste trabalho, denominado `DownloadRunAndGetResults()` (seção 4.2.3), permite que um TLM transfira, execute e receba o resultado da execução de um *script* com a invocação de apenas uma operação WS. Com essa operação WS sendo disponibilizada pelo *gateway* em nível de serviço e pelo agente WS, o tráfego gerado pela comunicação entre o TLM e o MLM se mantém praticamente constante e se resume no envio de duas mensagens: uma mensagem enviada pelo TLM invocando a operação WS e uma mensagem enviada pelo MLM contendo a resposta da operação WS solicitada.

Na segunda avaliação realizada (Figura 5.6), utilizou-se controle de *polling* no módulo do TLM responsável por gerenciar o MLM baseado em SNMP, mas continuou-se não utilizando compressão de dados nas mensagems trocadas entre o módulo do TLM e os MLMs baseados em *Gateways* WS para SNMP e Web Services.

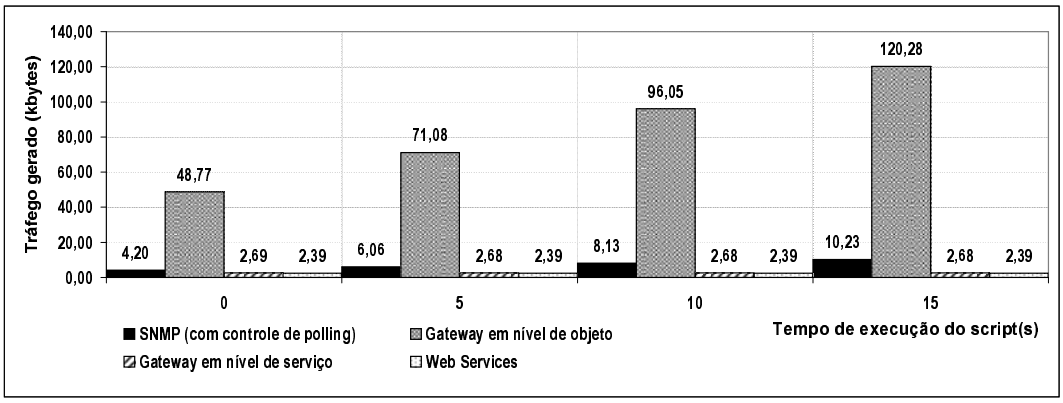

Figura 5.6: Tráfego gerado: TLM com controle de *polling*

Verifica-se pela Figura 5.6 que ao se controlar o intervalo de envio das mensagens SNMP do módulo do TLM responsável pela comunicação com o MLM baseado em SNMP, o tráfego gerado pelo SNMP se reduz drasticamente, sendo consideravelmente menor que o tráfego gerado pelo *gateway* em nível de objeto. Isso acontece porque tanto as mensagens SNMP enviadas do TLM em direção ao MLM baseado em SNMP quanto as mensagens SOAP/HTTP enviadas em direção ao *gateway* em nível de objeto partem em intervalos iguais, devido ao controle de *polling*. Além disso, como o tamanho (em bytes) das mensagens SNMP são menores que as mensagens SOAP sobre HTTP e pelo fato do SNMP utilizar UDP como protocolo de transporte, o tráfego gerado pelo SNMP é menor que o tráfego gerado pelo SOAP/HTTP. O tráfego gerado pelo *gateway* em nível de serviço e pelo agente WS se mantém constante pela mesma razão apresentada anteriormente.

Na terceira e última avaliação (Figura 5.7), referente ao parâmetro tráfego gerado, utilizou-se controle de *polling* no módulo do TLM responsável por gerenciar o MLM baseado em SNMP e compressão de dados nas mensagems SOAP/HTTP trocadas entre o módulo do TLM e os MLMs baseados em *gateways* WS para SNMP e Web Services. Pela figura verifica-se que a utilização de compressão das mensagens SOAP sobre HTTP diminui o tráfego gerado pelos MLMs baseados em *gateways* WS para SNMP e Web Services por uma razão de certa forma óbvia: o tamanho das mensagens SOAP/HTTP com compressão é menor que aquelas sem compressão, gerando dessa forma menos tráfego.

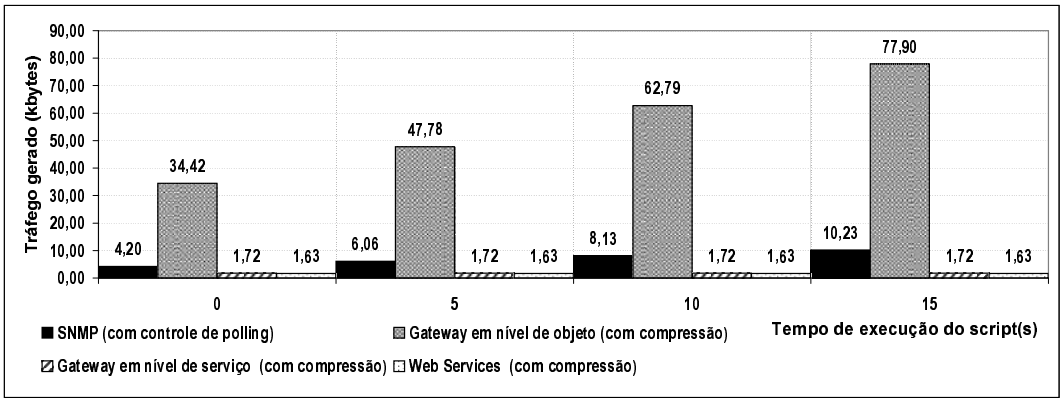

Figura 5.7: Tráfego gerado: TLM com controle de *polling* e mensagens SOAP/HTTP com compressão

5.5 Tempo de resposta percebido

O parâmetro tempo de resposta percebido se refere ao tempo que o TLM leva para perceber que o MLM encerrou as tarefas de transferência, execução e retorno da execução do *script* '*wait.jar*'. Obviamente, o tempo percebido pelo TLM é maior que o tempo levado pelo MLM na realização dessas tarefas. Além disso, uma vez que cada cenário de avaliação é composto por diferentes tecnologias, contendo cada uma delas diferenças de funcionamento, é natural pensar que o tempo de resposta percebido por cada TLM pode ser diferente.

A avaliação do parâmetro tempo de resposta percebido seguiu a mesma ordem apresentada na avaliação do tráfego total gerado. Na primeira avaliação (Figura 5.8), não se utilizou controle de *polling* no módulo do TLM responsável por gerenciar o MLM baseado em SNMP e, também, não foi utilizada compressão de dados nas mensagems trocadas entre o módulo do TLM e os MLMs baseados em *gateways* WS para SNMP e Web Services.

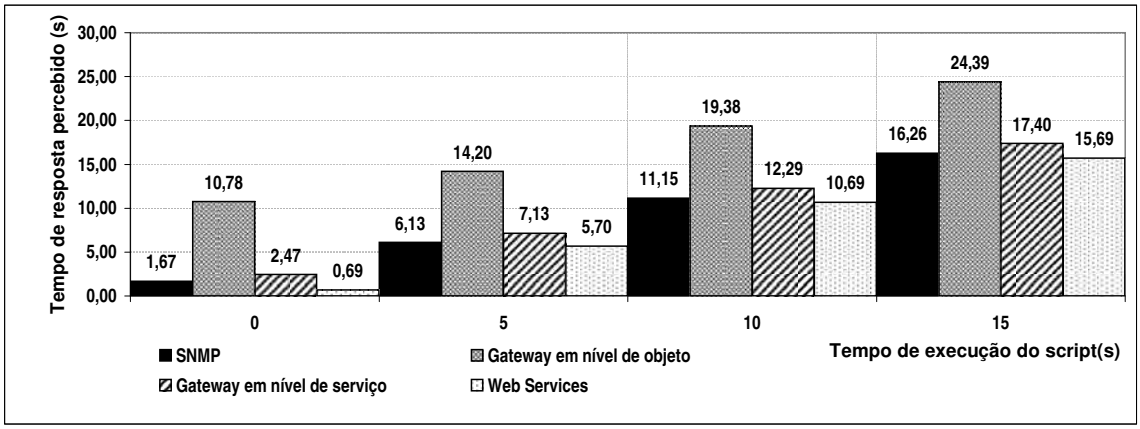

Figura 5.8: Tempo de resposta percebido: TLM sem controle de *polling*

Verifica-se pela Figura 5.8 que o cenário de avaliação utilizando Web Services foi o que apresentou o melhor tempo de resposta percebido. Isso ocorre devido a implementação do agente WS desenvolvido neste trabalho que permite que o número de passos neces-

sários ao TLM para a realização de determinada tarefa seja consideravelmente pequena. O TLM responsável pela comunicação com o agente WS necessitou apenas invocar a operação WS `DownloadRunAndGetResults()` para solicitar ao agente WS a realização dos passos descritos na seção 5.3. Com essa transferência de controle ao MLM, o tempo de resposta percebido pelo TLM fica mais próximo do término real da execução das tarefas realizadas pelo MLM, devido ao fato que o número de mensagens trocadas entre o TLM e o MLM é menor, o que minimiza a influência no parâmetro tempo de resposta percebido dos fatores: atraso de propagação e manipulação na pilha de protocolos.

O cenário de avaliação utilizando SNMP apresentou um tempo de resposta percebido próximo ao apresentado pelo cenário utilizando Web Services. O tempo foi um pouco superior devido ao fato que ao contrário do cenário de avaliação utilizando Web Services, onde o MLM realiza os passos descritos na seção 5.3, no cenário de avaliação utilizando SNMP, o TLM é responsável pela realização dos passos. Isso acaba aumentando o tempo percebido pelo TLM uma vez que a quantidade de mensagens SNMP trocadas entre o módulo do QAME e o Jasmin são maiores e sofrem com o atraso na transmissão pelo cabo cruzado ligando essas duas entidades e com a manipulação das mensagens na pilha de protocolos. Outro fator que contribui com a elevada quantidade de mensagens trocadas entre o TLM e o MLM é o fato de não se estar utilizando controle de *polling* no TLM.

O TLM responsável pela comunicação com o *gateway* em nível de serviço teve um tempo de resposta percebido muito próximo ao apresentado pelo TLM responsável pela comunicação com Jasmin. Embora o número de mensagens trocadas entre o TLM e o *gateway* em nível de serviço seja igual ao número trocado entre o TLM e o agente WS, o *gateway* em nível de serviço sofre com o atraso imposto pelo Jasmin na realização dos passos. Além disso, existe o atraso na tradução das mensagens Web Services para operações SNMP e vice-versa. Devido a essas razões, o TLM responsável pela comunicação com o *gateway* em nível de serviço leva um pouco mais de tempo para perceber o tempo de resposta.

O TLM responsável pela comunicação com o *gateway* em nível de objeto foi o que apresentou o pior desempenho dentre as soluções avaliadas. Isso ocorre pois além de o *gateway* em nível de objeto sofrer com o atraso imposto pelo Jasmin, o TLM é responsável pela realização dos passos. Com isso, o número de mensagens SOAP/HTTP trocadas entre o TLM e o MLM é maior, fazendo com que cada mensagem trocada, sofra com o atraso na transmissão e com a manipulação na pilha de protocolos. Outro fator que aumenta o número de mensagens trocadas entre o TLM e o *gateway* em nível de objeto é a utilização do TCP como protocolo de transporte. Todos esses fatores contribuem para um tempo de resposta, percebido pelo TLM, mais alto.

Na segunda avaliação (Figura 5.9), o controle de *polling* foi utilizado no módulo do TLM responsável por gerenciar o MLM baseado em SNMP. Porém, as mensagems trocadas entre o módulo do TLM e os MLMs baseados em *gateways* WS para SNMP e Web Services continuaram sem compressão de dados. Verifica-se pela figura que o TLM responsável pela comunicação com o *gateway* em nível de serviço continua apresentando o melhor tempo de resposta percebido. Verifica-se, também, que o TLM responsável pela comunicação com o Jasmin teve um desempenho um pouco pior ao apresentado na Figura 5.8. Isso é devido ao fato de se ter controle de *polling* no TLM, o que faz com que o número de mensagens trocadas entre ele e o Jasmin diminua e seja trocada em intervalos regulares de tempo. Essa regularidade no envio das mensagens aumenta o tempo de resposta percebido pois é possível que a execução tenha encerrado no Jasmin em um determinado tempo, mas o TLM não pode enviar uma nova mensagem consultando o estado

da execução até que o tempo imposto no intervalo de envio seja alcançado. Por exemplo, se a execução de um *script* encerrou no tempo `X` segundos e a última mensagem enviada pelo TLM foi no tempo `X-1` segundos. Um nova mensagem consultando o estado da execução só poderia ser enviada novamente no tempo `X+ip` segundos, sendo `ip` o intervalo de *polling* definido no TLM. Dependendo do valor de `ip` imposto ao TLM, pode-se ter um tempo de resposta percebido próximo ao término da execução do *script* no Jasmin (`ip` com um valor baixo) ou um tempo de resposta percebido muito além (`ip` com um valor grande). Os *gateways* em nível de serviço e de objeto apresentam os mesmos resultados apresentados anteriormente devido as mesmas razões explicadas no caso da Figura 5.8.

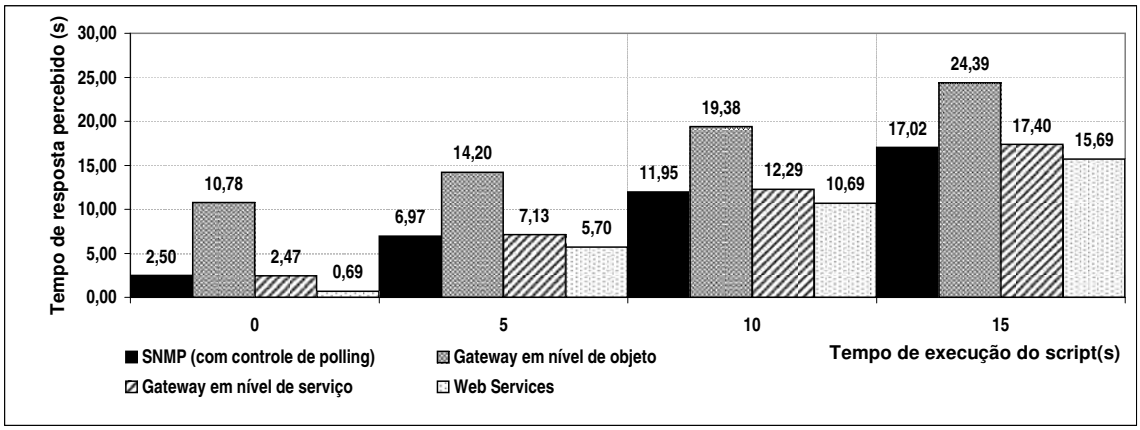

Figura 5.9: Tempo de resposta percebido: TLM com controle de *polling*

Encerrando as avaliações realizadas neste trabalho, na terceira avaliação (Figura 5.10) foi utilizado controle de *polling* no módulo do TLM responsável por gerenciar o MLM baseado em SNMP e compressão de dados nas mensagems SOAP/HTTP trocadas entre o módulo do TLM e os MLMs baseados em *Gateways* WS para SNMP e Web Services.

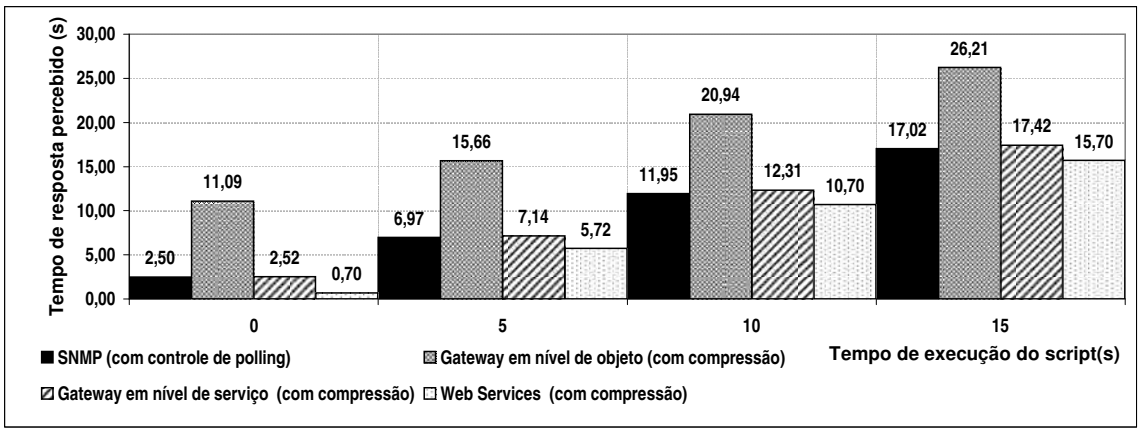

Figura 5.10: Tempo de resposta percebido: TLM com controle de *polling* e mensagens SOAP/HTTP com compressão

Verifica-se pela Figura 5.10 que a utilização de compressão nas mensagens SOAP/HTTP eleveram um pouco o tempo de resposta percebido pelo módulo do TLM no caso do

agente WS e dos *gateways* em nível de serviço e de objeto. Isso ocorre porque a compressão das mensagens SOAP/HTTP exigem um certo tempo de processamento no TLM e no MLM, demorando um pouco mais até serem enviadas à rede.

6 CONCLUSÕES

Novas tecnologias vêm sendo utilizadas no gerenciamento de redes como soluções alternativas ao protocolo padrão de gerenciamento de redes TCP/IP, o SNMP. A tecnologia investigada nessa dissertação foi a tecnologia de Web Services (WS) que, embora não tenha sido desenvolvida originalmente para ser utilizada na área de gerenciamento, vem se tornando bastante promissora. A substituição imediata do SNMP nos dispositivos gerenciados em prol da utilização dos WS não é uma abordagem factível, pois é muito provável que o SNMP permanecerá sendo utilizado por um bom tempo. Além disso, a tecnologia de Web Services é recente, necessitando maiores investigações com respeito a sua utilização na área de gerenciamento de redes.

Uma abordagem mais factível é a introdução e utilização conjunta dos Web Services com SNMP na realização de tarefas de gerenciamento, como é o caso dos *gateways* WS para SNMP apresentados neste trabalho. Essa introdução gradual dos WS na realização de tarefas específicas, comumente feitas por intermédio do SNMP, se faz necessária para uma melhor avaliação dos WS no âmbito do gerenciamento de redes. Seguindo essa abordagem, duas implementações baseadas em Web Services foram desenvolvidas no contexto específico do gerenciamento distribuído por delegação. A MIB Script, cujo objetivo é dar suporte ao gerenciamento por delegação na solução SNMP, foi utilizada como base no desenvolvimento das duas implementações.

A primeira implementação, visando a co-existência do SNMP com a tecnologia de Web Services, foi a construção de um novo tipo de *gateway* WS para SNMP, denominado *gateway* em nível de serviço. Esse *gateway* tem por característica principal criar um grau de abstração maior, pela visão do gerente, na manipulação dos objetos de um módulo de MIB, tornando, dessa forma, mais simples o gerenciamento dos agentes manipulados. Outra característica desse *gateway* é diminuir o número de mensagens trocadas entre ele e o gerente WS, reduzindo, com isso, o tráfego gerado pela comunicação dos dois.

A segunda implementação desenvolvida neste trabalho visou a total substituição do SNMP no gerenciamento dos dispositivos, permitindo dessa forma que um gerenciamento de redes completamente baseado em Web Services seja realizado. A implementação baseou-se na construção de um novo tipo de agente, denominado por agente orientado a serviços baseado em Web Serviços ou agente WS. Tanto o agente WS quanto o *gateway* em nível de serviço desenvolvidos fornecem o mesmo grupo de serviços e com a mesma interface para as operações, permitindo dessa forma que o gerente WS não precise acessá-los de maneira diferente.

Implementou-se também um novo módulo, incorporado ao ambiente de gerenciamento QAME, que permitiu a manipulação via interface Web das duas implementações desenvolvidas neste trabalho. O módulo desenvolvido é o mesmo para as duas implementações, uma vez que as operações oferecidas pelas mesmas possuem a mesma interface

de acesso, permitindo com isso que tanto o *gateway* WS para SNMP em nível de serviço quanto o agente orientado a serviços baseado em Web Services sejam manipulados da mesma maneira.

Ao final das implementações, foram realizadas comparações entre as duas implementações desenvolvidas neste trabalho e outras duas soluções de gerenciamento investigadas: o *gateway* em nível de objeto e uma implementação da MIB Script, denominada Jasmin. Os parâmetros avaliados nas comparações foram o tráfego gerado pela comunicação entre o TLM e o MLM e o tempo de resposta percebido pelo TLM.

A utilização da tecnologia de WS no gerenciamento de redes pode se tornar inadequada se cuidados especiais não forem tomados. A grande quantidade de tráfego gerado e o elevado tempo de resposta percebido no *gateway* em nível de objeto são provas disso. Embora inicialmente o *gateway* em nível de objeto seja uma interessante maneira de integrar WS e SNMP, ele não pode ser considerado adequado para todos os casos onde existam a integração de WS e SNMP, como é o caso do gerenciamento por delegação estudado neste trabalho.

O SNMP apresentou o segundo melhor tempo percebido dentre as soluções avaliadas. Isso se deve ao fato, principalmente, da utilização do protocolo SNMP sobre UDP. Como o UDP não trabalha com confirmação de entrega de mensagens, isso faz com que o TLM gere novas mensagens e as envie em um intervalo de tempo curto, permitindo, com isso, uma monitoração mais apurada do tempo real de término da execução de um *script* de gerenciamento. Entretanto, se nenhum cuidado for tomado quanto a taxa de envio de mensagens pelo TLM à rede (*polling control*), o tráfego gerado pelo ambiente SNMP é extremamente elevado, sendo muitas vezes pior que o *gateway* em nível de objeto, comentando anteriormente.

O *gateway* em nível de serviço apresentou um baixo tráfego gerado, conforme a metodologia empregada neste trabalho. Isso se deve ao fato da sua característica de funcionamento, que possibilita que um número pequeno de mensagens necessitem ser trocadas entre ele e o TLM na realização de uma determinada ação de gerenciamento. O tempo de resposta percebido foi um pouco superior ao apresentado pelo ambiente SNMP, devido ao fato do *gateway* sofrer com o atraso nas traduções das mensagens WS para operações SNMP e vice-versa, e também, por ele depender do tempo que o agente SNMP leva para perceber o término da execução de um *script*, para então poder informar ao TLM.

O agente WS apresentou os melhores resultados nos dois parâmetros avaliados. O pouco tráfego gerado possui a mesma justificativa apresentada no caso do *gateway* em nível de serviço. No caso do tempo de resposta percebido, o TLM leva menos tempo para perceber o término da execução de um *script*. Isso ocorre, principalmente, porque o número de passos necessários para a realização de determinada ação é muito menor, se comparado com a quantidade de passos normalmente necessária para realizar a mesma ação no caso do ambiente SNMP. Com um número de passos menor, uma quantidade menor de mensagens necessitam ser trocadas entre o TLM e o MLM, diminuindo com isso o tempo necessário no processamento das mensagens na pilha de protocolos.

O objetivo principal desta dissertação foi alcançado, uma vez que os resultados apresentados mostraram que em situações específicas, como foi o caso do gerenciamento por delegação estudado neste trabalho, a utilização do SNMP ou da tecnologia de Web Services pode diferir dos resultados apresentados em investigações mais abrangentes. Por exemplo, em um estudo realizado pelo Grupo de Redes da UFRGS, o *gateway* em nível de objeto apresentava-se como uma solução mais apropriada na consulta de uma quantidade bastante elevada de objetos de um agente SNMP ao invés de realizar consultas

através do gerente SNMP. Entretanto, no caso deste trabalho, salvo o caso de não ser utilizado controle de *polling* no gerente SNMP, o *gateway* em nível de objeto foi o que apresentou os piores resultados, não sendo, portanto, muito apropriado para a utilização em um ambiente de gerenciamento por delegação.

Como trabalhos futuros, novos parâmetros de avaliação podem ser investigados, permitindo com isso que novas questões de desempenho sejam verificadas. Quanto as implementações realizadas, metódos que forneçam autenticação e autorização de usuários na manipulação do agente WS e do *gateway* em nível de serviço são necessários. Além disso, é importante a utilização de métodos que possibilitem a privacidade das informações trocadas entre o gerente WS e as implementações desenvolvidas, melhorando com isso questões de segurança referentes ao processo de gerenciamento desenvolvido neste trabalho.

Por fim, acredita-se que a tecnologia de Web Services possui um grande potencial na sua utilização no gerenciamento de redes, permitindo, por exemplo, que WS mais complexos e com um número maior de serviços ofertados possam ser construídos a partir de outros WS mais simples. Essa característica pode ser utilizada para integrar serviços de gerenciamento de redes, ou, de uma maneira mais interessante, criar novos serviços de gerenciamento que não poderiam ser facilmente construídos, atualmente, sem um grande esforço de codificação. Integrar WS mais simples para formar WS mais complexos e com um número maior de serviços de gerenciamento é uma interessante abordagem a ser seguida. Acredita-se que isso permita, por exemplo, que um gerenciamento orientado a redes tome lugar do atual gerenciamento baseado em dispositivos.

REFERÊNCIAS

APACHE. **The APACHE Software Foundation**. [S.l.], 1999. Disponível em: <http://www.apache.org/>. Acesso em: 21 mar. 2005.

APPARAO, V.; BYRNE, S.; CHAMPION, M.; ISAACS, S.; JACOBS, I. **Document Object Model (DOM) Level 1 Specification**. [S.l.], 1998. Disponível em: <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>. Acesso em: 21 mar. 2005.

AYALA, D. **NuSOAP - Web Services Toolkit for PHP**. [S.l.], 2003. Disponível em: <http://dietrich.ganx4.com/nusoap/>. Acesso em: 21 mar. 2005.

BELLWOOD, T.; CLÉMENT, L.; RIEGEN, C. **UDDI Version 3.0.1**. [S.l.], 2003. Disponível em: <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>. Acesso em: 21 mar. 2005.

BEN-ARTZI, A.; CHADNA, A.; WARRIER, U. Network Management of TCP/IP Networks: present and future. **IEEE Network Magazine**, New York, NY, USA, v.4, n.4, p.35–43, July 1990.

CASE, J.; FEDOR, M.; SCHOFFSTALL, M.; DAVIN, J. **A Simple Network Management Protocol (SNMP)**: RFC 1157. [S.l.]: Internet Engineering Task Force, Network Working Group, 1999.

CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEERAWARANA, S. **Web Services Description Language (WSDL) 1.1**. [S.l.], 2001. Disponível em: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acesso em: 21 mar. 2005.

CURBERA, F.; DUFTLER, M.; KHALAF, R.; NAGY, W.; MUKHI, N.; WEERAWARANA, S. Unraveling the Web services Web: an introduction to SOAP, WSDL, and UDDI. **IEEE Internet Computing**, New York, NY, USA, v.6, n.2, p.86–93, March/April 2002.

CURBERA, F.; KHALAF, R.; MUKHI, N.; TAI, S.; WEERAWARANA, S. The Next Step in Web Services. **Communications of the ACM**, New York, NY, USA, v.46, n.10, p.29–34, Oct. 2003.

DREVERS, T.; MEENT, R. van de; PRAS, A. On the standardisation of Web Services management operations. In: OPEN EUROPEAN SUMMER SCHOOL AND IFIP WG6.3 WORKSHOP, EUNICE, 10., 2004, Tampere, Finland. **Proceedings...** [S.l.: s.n.], 2004. p.143–150.

ENNS, R. **NETCONF Configuration Protocol - draft-ietf-netconf-prot-05**. [S.l.]: IETF, 2005. Work in progress.

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. **Hypertext Transfer Protocol – HTTP/1.1**: RFC 2616. [S.l.]: Internet Engineering Task Force, Network Working Group, 1999.

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, 2005, Nice, France. **Proceedings...** [S.l.: s.n.], 2005. p.601–614.

GOLDSZMIDT, G.; YEMINI, Y. Distributed Management by Delegation. In: IFIP/IEEE INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 1995, Vancouver, Canada. **Proceedings...** [S.l.: s.n.], 1995. p.333 – 340.

GOLDSZMIDT, G.; YEMINI, Y. Delegated Agents for Network Management. **IEEE Communications Magazine**, New York, NY, USA, v.36, n.3, p.66–70, Mar. 1998.

GOTH, G. Grid Services Architecture Plan Gaining Momentum. **IEEE Internet Computing**, New York, NY, USA, v.6, n.4, p.11– 12, July/Aug. 2002.

GRANVILLE, L. Z.; CECCON, M. B.; TAROUCO, L. M. R.; ALMEIDA, M. J. B.; CARISSIMI, A. S. An Approach for Integrated Management of Networks with Quality of Service Support Using QAME. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 2001, Nancy, France. **Proceedings...** [S.l.: s.n.], 2001. p.167–178.

GUDGIN, M.; HADLEY, M.; MENDELSOHN, N.; MOREAU, J.-J.; NIELSEN, H. F. **SOAP Version 1.2 Part 1 - Messaging Framework**. [S.l.: s.n.], 2003. Disponível em: <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>. Acesso em: 21 mar. 2005.

HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. **An Architecture for Describing SNMP Management Frameworks**: RFC 2571. [S.l.]: Internet Engineering Task Force, Network Working Group, 1999.

IETF. **Distributed Management (disman) Charter**. [S.l.], 2005. Disponível em: <http://www.ietf.org/html.charters/disman-charter.html>. Acesso em: 21 mar. 2005.

INTERNET SOCIETY. **Internet Engineering Task Force**. [S.l.], 2005. Disponível em: <http://www.ietf.org>. Acesso em: 21 mar. 2005.

JASMIN. **Jasmin - a Script-MIB Implementation**. [S.l.], 2000. Disponível em: <http://www.ibr.cs.tu-bs.de/projects/jasmin/>. Acesso em: 21 mar. 2005.

KAHANI, M.; BEADLE, H. Decentralized Approaches for Network Management. **Computer Communications Review**, New York, NY, USA, v.27, n.3, p.36–47, July 1997.

KLENSIN, J. **Simple Mail Transfer Protocol**: RFC 2821. [S.l.]: Internet Engineering Task Force, Network Working Group, 2001.

KLIE, T.; STRAUSS, F. Integrating SNMP agents with XML-based management systems. **IEEE Communications Magazine**, New York, NY, USA, v.42, n.7, p.76–83, July 2004.

LEVI, D.; SCHÖNWÄLDER, J. **Definitions of Managed Objects for the Delegation of Management Scripts**: RFC 3165. [S.l.]: Internet Engineering Task Force, Network Working Group, 2001.

MARTIN-FLATIN, J. **Web-Based Management of IP Networks and Systems**. Hoboken, New Jersey: John Wiley & Sons, 2002. v.1.

MYSQL. **MySQL - The World's Most Popular open Source Database**. [S.l.], 1995. Disponível em: <http://www.mysql.com/>. Acesso em: 21 mar. 2005.

NEISSE, R.; GRANVILLE, L. Z.; BALLVÉ, D. O.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. A Dynamic SNMP to XML Proxy Solution. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.481–484.

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 Services Gateways. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2004, Seoul, South Korea. **Proceedings...** [S.l.: s.n.], 2004. p.715–728.

OETIKER, T. **Round Robin Database Tool (RRDTool)**. [S.l.], 2003. Disponível em: <http://www.rrdtool.org>. Acesso em: 21 mar. 2005.

OMG. **Object Management Group CORBA Website**. [S.l.], 1997. Disponível em: <http://www.corba.org>. Acesso em: 21 mar. 2005.

PAVLOU, G.; FLEGKAS, P.; GOUVERIS, S.; LIOTTA, A. On Management Technologies and the Potential of Web Services. **IEEE Communications Magazine**, New York, NY, USA, v.42, n.7, p.58–66, July 2004.

PHP. **PHP - The Hypertext Preprocessor**. [S.l.], 2001. Disponível em: <http://www.php.net/>. Acesso em: 21 mar. 2005.

POSTEL, J.; REYNOLDS, J. **File Transfer Protocol (FTP)**: RFC 959. [S.l.]: Internet Engineering Task Force, Network Working Group, 1985.

PREECE, A.; DECKER, S. Intelligent Web Services. **IEEE Intelligent Systems**, New York, NY, USA, v.17, n.1, p.15–17, Jan./Feb. 2002.

ROSE, M.; MCCLOGHRIE, K. **Structure and Identification of Management Information for TCP/IP-based Internets**: RFC 1155. [S.l.]: Internet Engineering Task Force, Network Working Group, 1990.

ROY, J.; RAMANUJAN, A. Understanding Web Services. **IT Professional**, New York, NY, USA, v.3, n.6, Dec. 2001.

SCHÖNWÄLDER, J.; PRAS, A. Internet Management: status and challenges. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.19–23.

SCHÖNWÄLDER, J.; PRAS, A.; MARTIN-FLATIN, J. P. On the Future of Internet Management Technologies. **IEEE Communications Magazine**, New York, NY, USA, v.41, n.10, p.90–97, Oct. 2003.

SCHÖNWÄLDER, J.; QUITTEK, J.; KAPPLER, C. Building Distributed Management Applications with the IETF Script MIB. **IEEE Journal on Selected Areas in Communications**, New York, NY, USA, v.18, n.5, p.702–714, May 2000.

SOURCEFORGE TEAM. **Net-SNMP SourceForge project**. [S.l.], 1992. Disponível em: <http://www.net-snmp.org/>. Acesso em: 21 mar. 2005.

SUN MICROSYSTEMS. **Java Techonology**. [S.l.], 1994. Disponível em: <http://java.sun.com/>. Acesso em: 21 mar. 2005.

W3C. **The World Wide Web Consortium**. [S.l.], 1994. Disponível em: <http://www.w3.org>. Acesso em: 21 mar. 2005.

W3C. **Extensible Markup Language (XML)**. [S.l.], 1996. Disponível em: <http://www.w3.org/XML>. Acesso em: 21 mar. 2005.

YEMINI, Y.; GOLDSZMIDT, G.; YEMINI, S. Network Management by Delegation. In: INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, 1991. **Proceedings...** [S.l.: s.n.], 1991. p.95–107.

YOON-JUNG, O.; HONG-TAEK, J.; MI-JUNG, C.; HONG, J. Interaction Translation Methods for XML/SNMP Gateway. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS AND MANAGEMENT, DSOM, 2002, Montreal, Canada. **Proceedings...** [S.l.: s.n.], 2002. p.54–65.

ANEXO A ARTIGO PUBLICADO

Neste anexo é apresentado o artigo "*Comparing Web Services with SNMP in a Management by Delegation Environment*" desenvolvido durante o mestrado. O artigo é resultado do segundo ano de mestrado e apresenta a utilização da tecnologia de Web Services, através de *gateways* WS para SNMP, no contexto específico do gerenciamento por delegação. Ele apresenta também o *gateway* WS para SNMP em nível de serviço, mostrado nesta dissertação, e as comparações realizadas entre os *gateways* em nível de serviço e de objeto e uma implementação da MIB Script.

- Título: *Comparing Web Services with SNMP in a Management by Delegation Environment*
- Nome: IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)
- URL: http://www.ieee-im.org/
- Data: De 15 a 19 de maio de 2005
- Local: Nice-Acropolis, Exhibition Hall, Nice, França
- Páginas: 601-614
- ISBN: 0-7803-9087-3

Comparing Web Services with SNMP in a Management by Delegation Environment

T. Fioreze, L. Z. Granville, M. J. Almeida, L. R. Tarouco
Institute of Informatics, Federal University of Rio Grande do Sul
Av. Bento Gonçalves, 9500
Porto Alegre, RS, Brazil
{tfioreze|granville|janilce|liane}@inf.ufrgs.br

Abstract

The traditional management by delegation model has lead the IETF, more recently, to define its Script MIB, which allows management entities to transfer, control and retrieve results generated by management scripts. These operations, however, can also be accomplished by Web Services. Web Services has been pointed out as an interesting approach for network management in general, but in this paper we present the use of Web Services in the specific context of management by delegation. We also present a Web Services-based system prototype that allows us to compare a Script MIB implementation against SNMP to Web Service gateways for management by delegation.

Keywords

web services, SNMP gateways, management by delegation, integrated management

1. Introduction

In the last decade, several efforts were carried out to investigate and develop decentralized network management solutions that could overcome the limitations of the traditional centralized management approach. The centralized approach is often inadequate because it imposes a single point of failure, it does not scale well, and it lacks flexibility [1]. In this scenario, the management by delegation (MbD) model, from the pioneer work of Yemini and Goldszmidt [2], was conceived and is today widely accepted and recognized by the network management community. One of the main features of the management solutions that are based on the MbD model is the ability to transfer and remotely control management scripts located on remote entities, which leads to the ability to delegate management functions along the management system, therefore decentralizing the management operations. The use of scripts as a way to delegate management functions is a direct result of the efforts taken by both IETF and ISO/ITU to integrate the MbD model into their management frameworks that are based on SNMP [3] and CMIP [4], respectively.

Recently, the Web Services (WS) technology [5], proposed by software engineering research groups and by Web-related industry, and under standardization process at the World Wide Web Consortium, has been presented as a feasible alternative to overcome some of the problems investigated for years in the network management area. However, since WS technology is young, it needs to be carefully investigated before it can be effectively deployed in network management solutions. Today, for example, comparing WS

with SNMP (the *de facto* Internet management protocol) is an intensive research area with very important, although preliminary, results. Some studies already show that WS can consume less bandwidth than SNMP when a large number of management objects needs to be retrieved from a management entity [6]. On the other hand, other studies show that performance issues may prevent the WS usage [7], while one can argue that WS ease of use is an advantage that could surpass the performance aspect [8].

The current WS versus SNMP investigations are characterized by the fact that they are taken from a quite broad view, where the SNMP is considered in a general perspective, without paying closer attention to particular management situations or scenarios. We believe that such particular situations, however, can reveal aspects of the use of WS for network management that are currently hidden by the broader, more general investigations. In this paper we investigate the definition and use of SNMP to WS gateways built considering SNMP services defined through management information bases (MIBs), instead of considering direct mappings of SNMP operations to WS operations, which is the approach usually taken in the current investigations. In our view, WS can also play an important role in MbD due to their flexibility and ease of use. Thus, we present a study on the use of WS in the context of MbD, and compare WS with SNMP considering the specific case where the IETF Script MIB is used as a MbD tool. This particular situation servers as a case study for the proposed approach of building SNMP to WS gateways based on services. We have evaluated the proposed approach observing the bandwidth consumed by the WS traffic as well as the WS execution time when WS are providing the same set of services exposed by the Script MIB. We also present a prototype of a WS-based MbD system developed to allow the observation of WS against SNMP. In order to put our investigation in perspective, we observe the WS and SNMP differences through the use of two different SNMP to WS gateways: object-level (already defined in a previous work) [6] and service-level (introduced in this paper) gateways.

The remainder of this paper is organized as follows. Section 2 reviews the MbD model, the Script MIB, and the WS technology. Section 3 presents our system prototype paying more attention to the implementation of WS that support the MbD model. Section 4 shows the evaluation scenarios used to compare WS with SNMP (in the context of the Script MIB) as well as the evaluation results. Finally, we finish the paper presenting conclusions and future work in Section 5.

2. Background

In this section we summarize three topics that may be used together in a management solution. First, the MbD model is presented, followed by IETF Script MIB; then WS are reviewed from a network management perspective.

2.1 Management by Delegation

In large networks, the traditional centralized management approach reached its limits. In such an approach, a single management station is responsible for monitoring and controlling management agents distributed along the network. Agents are usually just a mean to collect or modify the data on managed devices, playing a quite simple role in the whole management process. The central management station collects and processes data from

these agents using a management protocol such as SNMP. With the network growing larger, the data processing at the management station can become pragmatically unfeasible due the great amount of data to be retrieved and processed. Distributed management has been proposed to improve the scalability and flexibility of management systems, overcoming the centralized approach problems. The distributed management by delegation (MbD) model proposes the delegation of management functions to distributed locations, which can be accomplished by transferring and remotely controlling a set of management scripts [2]. That allows pushing management functions closer to target managed entities, then decreasing the management traffic on a central point, and distributing the management decisions along the managed network. Several other advantages and features of the MbD model are widely covered by the network management literature [9] [10].

Traditionally, the entities of a MbD system are top-level managers (TLMs), mid-level managers (MLMs), and agents. TLMs are responsible for monitoring and controlling the execution of delegated scripts on the remote locations where MLMs are found. Thus, MLMs are the receivers of the management scripts and the ones responsible for executing such scripts in order to accomplish a management task. To do that, MLMs also interact with management agents located inside network devices. As pointed out before, agents are just a way to collect or modify the data of the managed devices. The communication between a MLM and an agent depends on the management protocol implemented by the agent and the corresponding support for that protocol found in the scripts being executed in the MLM. The communication between a TLM and a MLM, on the other hand, does not depend on the management scripts at all: such communication is required only to transfer and control the scripts, but is totally independent of the scripts contents.

2.2 The IETF Script MIB

The IETF has been working on the definition of an SNMP-based mechanism to support the TLM/MLM communication. The Distributed Management Working Group defined the Script MIB [11], which is an SNMP Management Information Base (MIB) implemented by MLMs that allows TLMs to transfer, control, and retrieve results from management scripts through SNMP. The Script MIB itself does not impose any specific language to be used in the scripts: it is just a mechanism to control them, but neither to define nor to check their contents. After a script transmission, however, the MLM needs to execute the script. That is done by means of execution engines (e.g. Java and TCL) available at the MLM. Thus, a MLM is composed by an implementation of the Script MIB and a set of execution engines responsible for running the management scripts transferred to the MLM. The following list summarizes the services exposed by the Script MIB (further details about the Script MIB operations can be found in RFC 3165 [11]):

- Services to describe and list the execution engines found in a MLM;
- Services to transfer scripts using both pull and push models;
- Services to schedule scripts executions;
- Services to check scripts executions status;
- Services to control (suspend, resume, and abort) the scripts executions;
- Services to retrieve the results of the scripts executions.

Although the Script MIB provides a standardized solution for MbD, because it is developed in the arms of the IETF, other technologies, for example, HTTP, CORBA RMI, XML, and so on, could be used to implement the same set of services exposed by the Script MIB. MbD using WS, as long as the authors of this paper is aware of, however, have never been investigated before, in order to observe whether WS is a feasible approach for MbD.

2.3 Web Services

Web Services (WS) technology has been gaining more and more attention in the network management area. Basically, it consists of independent application components that are published on the Web in such a way that other Web applications can find and use them [5]. The WS architecture consists of three main elements: service providers, service brokers, and service requesters. Service providers host and publish WS in order to allow them to be accessed by clients that can be either other applications or other WS. Service brokers are responsible for maintaining records about published WS, acting as a WS directory. Finally, service requesters search for WS in the service broker and, once the required WS is found, requesters bind their applications to the provider WS.

Providing interoperation among the WS architecture elements requires agreement on standards. UDDI (Universal Description, Discovery, and Integration) is a service broker specification used to register WS providers in order to make them available to WS requesters. WSDL (Web Services Description Language) is a standard to describe the WS operations and interfaces in XML documents. Finally, SOAP (Simple Object Access Protocol) is used to provide the communication among the WS architecture elements. Although UDDI, WSDL, and SOAP are probably the current most important WS standards, others can be used in a WS environment, such as WSIL (Web Services Inspection Language) and XML-RPC (XML Remote Procedure Call). Also, not all WS architecture elements need to be used in a WS-based solution: for example, the majority of the current WS-based systems does not include WS brokers.

The flexibility, distributed computing, and ease of use offered by WS technology seem to be more attractive than earlier management solutions such as SNMP, CMIP or CORBA. The WS features easily make people curious to know if WS would be able to improve or substitute the current management technologies and solutions. As indicated in the introduction, today, several investigations concerning WS for network management are under development [7] [8] [12]. SNMP versus WS is a first study required to compare the traditional, established approach (SNMP) with the brand-new one (WS).

SNMP has several applications in network management. Monitoring, configuration management, and management by delegation are some examples of applications where SNMP plays important roles, although in most of them SNMP seems not to be as successful as it was originally supposed to [13]. As pointed out before, we are particularly interested in comparing WS and SNMP in the context of management by delegation. To do that, we take the approach to develop WS that provide the same set of services exposed by the Script MIB. In the following section we present a prototype system that uses WS-based MLMs as an alternative for the traditional Script MIB-based MLM implementation.

3. Web Services-based MbD

In order to compare WS with the Script MIB we have implemented a WS-based system prototype and SNMP to WS gateways. The WS-based system is an extension of QAME (QoS-Aware Management Environment) [14], a system originally developed for QoS management research. QAME acts as a central top-level manager (TLM) that is able to delegate management scripts to three different MLMs.

The first MLM is a traditional installation that uses the Jasmin [15] system, which implements the IETF Script MIB. The second MLM is an object-level SNMP to WS gateway [6] that maps the structure of the Script MIB into WS operations. Finally, the last MLM is a new service-level SNMP to WS gateway that, although not strictly based on the Script MIB structure, provides the same services exposed by the Script MIB. In this section we will describe these three MLM implementations in detail and show how the QAME system interacts with them. Then, the next section presents the evaluation results.

3.1 MLM based on the Script MIB and Jasmin

Jasmin [15] is an implementation of the Script MIB developed by the Technical University of Braunschweig and NEC C&C Research Laboratories. Jasmin implements the Script MIB published in the RFC 2592, which was later updated by the RFC 3165. Jasmin supports the Java and TCL runtime engines, so that QAME (our TLM) can delegate Java and TCL management scripts to the Jasmin-based MLM.

Besides the TLM and MLM communication, a Script MIB-based MLM may also require the interaction with an external script repository, from where the MLM retrieves the scripts to be executed. Such interaction is needed when one is using a **pull script transfer model**, where the TLM informs the MLM via SNMP about the script to be transferred, giving the script identification and the repository address in a URL. Then, the MLM contacts the repository in order to download the script using a protocol such as HTTP or FTP. If a **push transfer model** is used instead, the script to be executed is first downloaded by the TLM, and then sent from the TLM to the target MLM, therefore requiring no MLM/repository interaction. Currently, Jasmin supports only the pull model and the script download is accomplished by HTTP, which means that this first, Script MIB-based MLM uses HTTP to retrieve management scripts from the script repository. Considering these questions, but also considering our purpose of a WS versus SNMP investigation, we designed and installed the Script MIB-based MLM according to Figure 1.

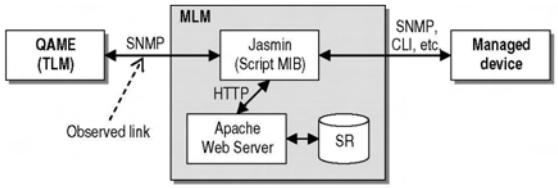

Figure 1: The Script MIB-based MLM

In this installation, the MLM not only hosts the Jasmin system, but also the script repository (SR) and an Apache Web server. Although this setup may be inadequate in real solutions because the script repository and Apache would be better placed outside the MLM, bringing the repository closer to Jasmin minimizes the script download delay via HTTP, which allows us to concentrate on the evaluation of the TLM/MLM interactions.

3.2 MLM as an Object-Level SNMP to WS Gateway

SNMP gateways are a software resource used to integrate SNMP-enabled entities into non-SNMP systems through protocol and/or information translation. Gateways from SNMP to WS can be built in several ways. In a previous work [6], we have defined two SNMP to WS gateways: protocol-level and object-level gateways. Protocol-level gateways directly map SNMP operations to WS operations. For example, the SNMP GetNext message is mapped to a `GetNext()` WS operation. Object-level gateways, on their turn, map SNMP MIB structures to WS operations. For example, the `sysUpTime` MIB-II object is mapped to a `GetSysUpTime()` WS operation. Although both protocol and object-level gateways could be used to access a Script MIB implementation, in this paper we investigate only the use of object-level gateways because our previous work showed that protocol-level gateways impose a high network bandwidth consumption [6].

Our second MLM is composed by an object-level SNMP to WS gateway whose set of operations is a mapping of the Script MIB objects. The gateway is coded in PHP [16] and accessed through the Apache Web server [17]. To build up the object-level gateway we used a tool [6] that receives an SNMP MIB as input and automatically creates the corresponding gateway. In this case, we have provided the Script MIB as the input MIB for the building tool. The operations of the created WS gateway are divided in Get's and Set's. The following list shows the two operations created for the Script MIB `smScriptSource` object.

```
function GetSmScriptSource ($index=-1, $community="public")
function SetSmScriptSource ($index, $newvalue, $community="public")
```

The Get operation requires an index, and an SNMP community. Both arguments are optional. If no index is provided, an array with all values associated with the `smScriptSource` object will be returned. The default SNMP community to be used is *public*. The Set operation requires an index and a new value as non optional arguments, while SNMP community remains optional as in the Get operation. It is important to notice that not all Script MIB objects have a Set operation: the MIB read-only objects will have only Get operations associated to them.

The installation for the MLM based on SNMP to WS gateways is presented in Figure 2. In order to reduce the delay between the SNMP to WS gateway and Jasmin, we have chosen to install both on the same device. The SNMP protocol required between TLM/MLM in the scenario presented in Figure 1 has been replaced by SOAP, but SNMP is still used internally in the MLM between the object-level gateway and Jasmin. The installation presented in Figure 2 is applied for this second scenario but also for the third scenario to be presented in the next subsection.

The Apache Web server is needed for two different purposes. First, it is used by the TLM to access the WS to SNMP gateways coded in PHP. While running, the gateways

contact the Jasmin installation via SNMP. Since one of the tasks to be executed is to download a management script from the repository, Jasmin contacts the Apache Web server again to retrieve the script via HTTP.

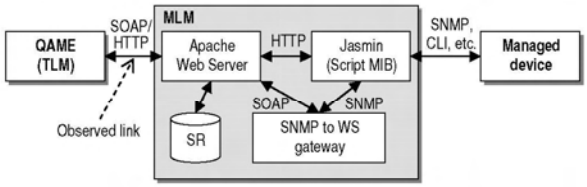

Figure 2: MLMs based on Object-Level and Service-Level SNMP to WS gateways

3.3 MLM as Service-Level SNMP to WS gateway

Although the object-level gateway presented before makes WS-based MbD possible, it is a SNMP generic gateway in the sense that the gateway can be built from any SNMP MIB provided. We believe that more specific service-oriented approaches need to be investigated as well. Thus, besides the protocol-level and object-level gateways already defined, we now define a third gateway named service-level SNMP to WS gateway. A service-level gateway is built considering the set of services provided by a MIB, but not taking strictly into account the MIB structure.

Since it is not based on a MIB structure, a service-level gateway cannot be built automatically by a building system. In fact, it needs to be built from scratch observing the services provided by a MIB. Our third MLM is a service-level gateway that provides the same services implemented by the Script MIB. We have studied the Script MIB and implemented a set of operations that provides the services of the Script MIB previously listed in section 2. The list below shows the interfaces of some operations exposed by our service-level gateway.

```
function BeginMbD ($owner, $community)
function EndMbD ($sessionID)
function ScriptDownload ($sessionID, $url, $store=1)
function ScriptRun ($sessionID, $scriptID, $args="")
function GetResults ($sessionID, $execID, $block=1)
function ScriptRemove ($sessionID, $scriptID)
function ScriptDownloadAndRun ($sessionID, $url, $args="", $block=1)
```

The `BeginMbD` operation initiates a MbD session. The operation returns a session ID to be used as an input argument when calling other operations. The `EndMbD` operation finishes a MbD session. The `ScriptDownload` operation requires a session ID and the address from where a script must be downloaded, as well as the information on how to store the script. By default, scripts are stored in the non-volatile storage. As a return value, `ScriptDownload` provides a script ID. The `ScriptRun` operation initiates the execution of a script given a session ID, a script ID, and the script arguments, and it returns an identi-

fication (`execID`) for the just initiated script execution. The `GetResults` operation, on its turn, blocks (`block` argument equal to 1) until a script execution is over and results are available. `GetResults` may also return false if the `block` argument is 0 and the execution is not over yet. In order to remove a script from the MLM, one has to call the `ScriptRemove` operation. Finally, the `ScriptDownloadAndRun` operation combines the functionalities of the previous operations and downloads, executes, and blocks until the results of the execution of a management script are available. In the evaluation to be presented in the next section we have used the `ScriptDownloadAndRun` operation extensively.

3.4 Coding TLMs in QAME

QAME [14] is an open source, Web-based network management system developed as a set of PHP scripts. It supports common management tasks such as monitoring (based on the RRDTool [18]), notification handling (based on the NET-SNMP [19] trap service), and network and service discovery. The environment was developed to support the easy inclusion of new modules when required. This feature allows us to build and study new management tasks (e.g. Policy-Based Network Management support [20]) without having to develop a whole new management system for each topic we want to investigate.

In order to control the three MLMs presented before, three QAME modules were developed. In our tests, we have used a management script named `wait.jar` that executes no action except waiting some seconds before returning. Although it has no action, `wait.jar` is suitable for our purposes because it still needs to be transferred, executed, and its results retrieved by the TLM. The number of seconds `wait.jar` must wait until its end is passed as an execution argument.

The codes that are going to be presented now show some extracts of such QAME modules. These extracts present the functions each TLM module must call in order to request a complete download of the `wait.jar` script to each MLM. A complete download corresponds not only to the script download itself, but also to the checking of the final download action status (success or fail). In the developed modules, NET-SNMP [19] has been used to access the Jasmin Script MIB implementation, while nuSOAP [21] (a PHP SOAP library) has been used to access both object-level and service-level gateways. In the first extract below the code required for the Script MIB-based MLM is presented.

```
// snmpset (hostname, community, object_id, type, value);
snmpset ($mlmIP, "secret", "smScriptRowStatus.$index", "i", "5");
snmpset ($mlmIP, "secret", "smScriptSource.$index", "s",
         "http://localhost/wait.jar");
snmpset ($mlmIP, "secret", "smScriptLanguage.$index", "i", "1");
snmpset ($mlmIP, "secret", "smScriptRowStatus.$index", "i", "1");
snmpset ($mlmIP, "secret", "smScriptAdminStatus.$index", "i", "1");
snmpset ($mlmIP, "secret", "smScriptStorageType.$index", "i", "3");
$operStatus="4";
while ($operStatus != "1") {
   apply_polling_control();
   $operStatus = snmpget ($mlmIP, "secret","smScriptOperStatus.$index");
}
```

In the case of the second QAME module, which access the object-level gateway, the TLM calls WS operations mapped from the Script MIB (e.g. `SetSmScriptLanguage`) in the exactly same order presented above. The code for this second module is not presented because the it is quite similar from the code of the first module (except that NET-SNMP calls are replaced by nuSOAP calls).

In case of the last module, where the TLM contacts the service-level gateways, the TLM requires less nuSOAP calls than in the previous cases to request a script download, as one can observe in code below. That happens because the `ScriptDownload` operation downloads, and returns the script download status all through the same call.

```
// ScriptDownload ($sessionID, $url, $store=1);
$args = new array ($sessionID, "http://localhost/wait.jar");
$result = $mlm->call ("ScriptDownload", $args)
```

First, the `$args` array is initialized with the arguments to be passed to the remote WS. The `$mlm` object is a reference to that remote WS (such reference includes the MLM network address), and the `call` function calls the remote `ScriptDownload` WS operation with the arguments stored in `$args`.

4. Evaluation

Given the three MLMs previously presented, we are now interested in observing the bandwidth and execution time associated to each of them. We considered the following steps taken by the TLM to execute a management script on each MLM (the codes presented in the end of the last section covers the first two steps).

- First, the TLM transfers to a MLM the `wait.jar` script using the pull transfer model;
- The TLM blocks until the MLM finishes the script download from the local repository;
- The TLM blocks again until `wait.jar` is ready to run;
- Then, the TLM requests the execution of `wait.jar` to the MLM passing the number of seconds `wait.jar` must wait until its end (we have used 0, 5, 10 and 15 seconds);
- The TLM blocks again until the execution of `wait.jar` is over;
- Finally, the TLM retrieves the execution results from the MLM.

The following subsections will present the test scenarios used in our evaluation. In each test scenario, we have executed the test steps described above 30 times. Each scenario is related to a particular aspect. Regarding protocols, in the case of the WS-based MLMs, we have also considered the compression of the SOAP/HTTP messages, as well as have observed those MLMs when they are implemented through two different SOAP APIs: first using nuSOAP [21], and then using PEAR::SOAP [22]. We have used nuSOAP and PEAR::SOAP because these are the two most popular PHP SOAP APIs. Also, the NET-SNMP API was used in the TLM and Script MIB-based MLM interactions. In the tests, we have used the script pull transfer model because Jasmin supports only this model. Once required, we block the execution of the TLM using a loop that reads the status of the script on the MLM. All tests were executed using two hosts: one that runs the QAME system (TLM), and another that runs our MLMs. The hosts have been connected to each

other using 10 Mbps network cards and a cross connect cable. Besides the TLM/MLM communication, there is no other traffic between the two hosts. Figure 3 summarizes the main hardware/software of each host.

	TLM	MLM
Processor	Intel Pentium IV 2.40 GHz	AMD K6-II 550 MHz
RAM	256 MB	128 MB
Operating system	Red Hat 9 (2.4.20-31.9)	Suse 6.4 (2.2.14)
Apache Web Server	2.0.50	2.0.50
PHP and Java	PHP 4.3.8	PHP 4.3.8, JDK 1.1.8 (required by Jasmin)
SOAP support	nuSOAP 0.6.3, PEAR::SOAP 0.7.5	nuSOAP 0.6.3, PEAR::SOAP 0.7.5
SNMP support	NET-SNMP 5.1.1	UCD-SNMP 4.2.6 (required by Jasmin)

Figure 3: Top level and mid level managers configuration

We considered as the bandwidth consumed just the bandwidth required solely by SNMP (for the Script MIB-based MLM) or SOAP/HTTP (for both object-level and service-level gateways) to execute all the test steps. Subjacent traffic, however, is present (but not considered), such as ARP and ICMP. In addition to bandwidth consumption, we have also measured the perceived script execution time by saving the time of the first message (SNMP or SOAP/HTTP) from the TLM to a MLM that initiates the first step, and saving the time of the last message from the MLM to the TLM that finishes the test steps. Using this environment, the following subsections present the evaluation results.

4.1 Bandwidth Consumption

Intuitively, the SNMP traffic required to transfer, execute, and retrieve the results from the `wait.jar` script on the Script MIB-based MLM is less than the traffic required to execute the same actions on the object-level MLM. This is due to the fact that a single SNMP message is smaller than the corresponding SOAP/HTTP message, since SOAP is based on XML, which is text-oriented, while SNMP is a binary encoded protocol. Besides, becouse SNMP relies on UDP, there is no UDP reply to acknowledge the delivery of an SNMP message. In the case of SOAP/HTTP, TCP is used instead, where acknowledgement messages do exist and consume further bandwidth. However, it is important to highlight that some polling control is needed in order to avoid an excessive traffic when polling a MLM. For example, in some initial tests (whose results are not presented in this paper), if this such polling control is not performed, the SNMP traffic can consume much more bandwidth than the traffic generated by the use of the nuSOAP library. This happened because the NET-SNMP implementation (based on C) is faster than the nuSOAP implementation (based on PHP). With messages being sent faster by NET-SNMP, more SNMP messages are delivered to the network, then more network resources are consumed.

For a fairer comparison, we have used polling control for SNMP messages, implemented inside the `apply_polling_control` function previously presented in a code extract. Measurement on the test network showed that successive SOAP/HTTP messages for the object-level gateway have been leaving the TLM in average intervals of 0.5 seconds. The SNMP polling control then forces the SNMP messages to leave the TLM at the

same 0.5 seconds interval. Considering such SNMP polling control and the SOAP/HTTP traffic, Figure 4 shows the network usage by different TLM/MLMs interactions.

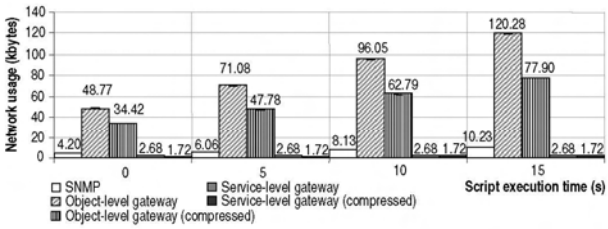

Figure 4: Network usage

According to the testing steps presented in the beginning of section 4, the TLM blocks in three moments: when it waits for the `wait.jar` script to be downloaded, when it waits for the script to be ready to run, and when it waits for the script to finish its execution. To implement such blocking, the QAME modules for the Script MIB and object-level MLMs loop until the value of the `smScriptOperStatus`, `smLaunchOperStatus` and `smRunStatus` Script MIB objects, or the `GetSmScriptOperStatus`, `GetSmLaunchOperStatus` and `GetSmRunStatus` WS operations (implemented by the object-level gateway) indicate that the script is downloaded, ready to run, and finished.

From Figure 4, SNMP consumes far less bandwidth than the object-level gateway, but it is still greater than the bandwidth consumed by the service-level gateway. In the case of the service-level MLM, no loop is required in the TLM because we are using the service-level gateway `ScriptDownloadAndRun` operation that blocks until the script results are available. In this case, the testing loop has been moved from the TLM to the MLM, then consuming less network resources. Figure 4 also shows the corresponding bandwidth consumed when the SOAP/HTTP traffic is compressed (via `zlib`). As observed, and intuitively believed, less resources are consumed when compression is used because compressed SOAP/HTTP messages are smaller than the non-compressed ones.

4.2 Execution Time

The execution time is the time perceived by the TLM spent in transferring, executing, and retrieving the results from the `wait.jar` script. The execution time perceived by the TLM is obviously greater than the actual execution time of the `wait.jar` script (Figure 5).

In the case of the Script MIB-based MLM, that uses SNMP, the perceived execution time is closer to the actual script execution time because as soon as the `wait.jar` script is downloaded or finishes, an SNMP message "discovers" that the script is ready and a reply message is quickly sent back to the TLM. In the case of the object-level MLM, the TLM takes more time to read the MLM status because the SOAP support (nuSOAP) is slower than the SNMP support (NET-SNMP), then increasing the perceived execution time. When compression is used, the perceived execution time is slightly greater because compression introduces further CPU processing so that an addition delay occurs. Finally,

with the service-level MLM, the perceived time is close to the SNMP time because the service-level MLM locally polls the Script MIB, the same way the TLM did when no intermediate gateway was used.

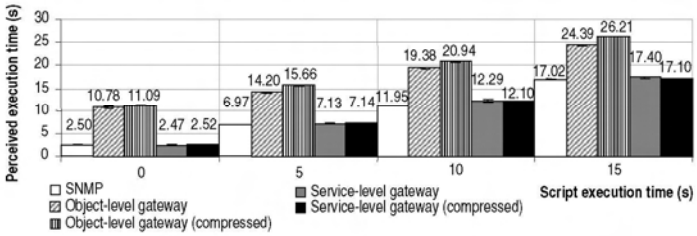

Figure 5: Perceived execution time

4.3 SOAP APIs

Since the performance of the SNMP and SOAP support has heavily influenced the results previously presented, it is feasible to think that different SOAP libraries would present different performance results as well. In a final comparison, we have replaced nuSOAP by PEAR::SOAP in the object-level and service-level gateways implementations. Figure 6 presents the results comparing nuSOAP and PEAR::SOAP regarding the perceived execution time. We omit the network usage results because both libraries present quite similar numbers, since SOAP messages are not too different in length, otherwise interoperation would be compromised.

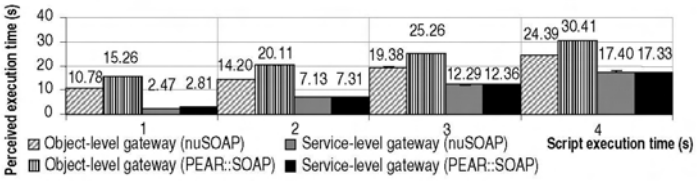

Figure 6: Perceived execution time using nuSOAP and PEAR::SOAP APIs

As can be observed, PEAR::SOAP imposes a greater perceived execution time. This is so because PEAR::SOAP has a processing delay greater than the nuSOAP processing delay, then taking more time to encode and deliver the SOAP/HTTP messages to the MLMs.

5. Conclusions and Future Work

In this paper we have evaluated the use of SNMP to WS gateways for MbD. Our goal was to investigate the use of WS in the context of the IETF Script MIB in order to check

if hidden aspects of WS for network management could be revealed. In such evaluation, besides the already defined protocol and object-level gateways, we have introduced a new one named service-level SNMP to WS gateway.

The use of WS for network management can be risky if no special cares are taken. The high bandwidth consumption and the execution time observed in our object-level gateway is an example of that. Although at first sight object-level gateways are an interesting SNMP/WS integration tool, not all object-level gateways are feasible because they depend on the environment where they are running. In the case of the Script MIB and MbD, the object-level gateways proved to be an inadequate solution because they consume more bandwidth than SNMP and take too much time to perceive when a script execution is over.

Object-level gateways are adequate when they are able to aggregate SNMP information in few SOAP messages, for example, when retrieving the contents of a MIB table at once, which is not the action often required when dealing with the Script MIB. It means that information aggregation is key to a proper SNMP/WS integration. This can be observed in the results obtained from the service-level gateway: since the polling loops imposed to a Script MIB client have been moved from the TLM to the service-level gateway, the gateway had the opportunity to aggregate and reduce the SNMP information sent to the TLM, then consuming less network resources without losing timely result delivery. Moreover, the service-level gateway added a new level of abstraction in dealing with the Script MIB, easing the development of TLMs while keeping the level of control.

In addition to protocols and gateways, we noted that SOAP support also has an impact on the performance of WS-based management solutions. For example, as observed in the use of the nuSOAP and PEAR::SOAP libraries, the execution time perceived by a TLM can be different if different SOAP implementations are used.

The results from this work help us to argue in favor of a service-oriented SNMP/WS integration, instead of information or protocol-oriented integration. The presented service-level gateway is easier to use (from the manager perspective), consumes less bandwidth than the other gateways investigated so far, and presents a response time quite close to the SNMP response time. It is important to highlight that these conclusions are based on the specific case of MbD investigated in this paper. Generalizing, however, we believe that although the SNMP framework is well known and accepted, designing WS management solutions taking the SNMP framework as a starting point is not a proper approach. Rather, we think that services must be considered in the first place and SNMP "details" should be treated only when the protocol and related information issues begin to be relevant.

A WS feature that is often cited is that more complex and richer WS can be built from the combination and reuse of other simpler WS. This feature could be used to integrate network management services, or, more interesting, to create new services that could not be easily built today except through heavy coding efforts. Integrating simpler WS to form richer network management WS is a next step in our research work. We believe that this is a way to achieve, for example, a real network-oriented management in the place of the current device-oriented management.

References

[1] J. Schoenwaelder, J. Quittek, and C. Kappler. Building Distributed Management Applications with the IETF Script MIB. *IEEE Journal on Selected Areas in Communications*, 18(5):702–714, May 2000.

[2] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In *International Symposium on Integrated Network Management*, pages 95–107, 1991.

[3] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Professional, third edition, 1998.

[4] Information technology – Open Systems Interconnection – Systems Management: Command sequencer for Systems Managenment. Int. Standard ISO 10164-21, 1998.

[5] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web services Web: an Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing, Vol. 6, Issue 2*, pages 86–93, March/April 2002.

[6] R. Neisse, R. L. Vianna, L. Z. Granville, M. J. B. Almeida, and L. M. R. Tarouco. Implementation and Bandwidth Consumption Evaluation of SNMP to Web Sevices Gateways. In *IEEE/IFIP Network Operations and Management Symposium, NOMS*, pages 715–728, Seoul, South Korea, 2004.

[7] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta. On Management Technologies and the Potential of Web Services. *IEEE Communications Magazine*, 42(7):58–66, 2004.

[8] T. Drevers, R. van de Meent, and A. Pras. On the standardisation of Web services management operations. In *Proceedings of the 10th Open European Summer School and IFIP WG6.3 Workshop, EUNICE*, pages 143–150, Tampere, Finland, 2004.

[9] JP. Martin-Flatin. *Web-Based Management of IP Networks and Systems*. Wiley, 2002.

[10] M. Kahani and H.W. Beadle. Decentralized Approaches for Network Management. *Computer Communications Review, ACM SIGCOMM*, 27(3):36–47, July 1997.

[11] D. Levi and J. Schönwälder. Definitions of Managed Objects for the Delegation of Management Scripts. RFC 3165, August 2001. IETF.

[12] T. Klie and F. Strauss. Integrating SNMP agents with XML-based management systems. *IEEE Communications Magazine*, 42(7):76–83, July 2004.

[13] J. Schoenwaelder, Aiko Pras, and J. P. Martin-Flatin. On the Future of Internet Management Technologies. *IEEE Communications Magazine*, 41(10):90–97, Oct. 2003.

[14] L. Z. Granville, M. B. Ceccon, L. M. R. Tarouco, M. J. B. Almeida, and A. S. Carissimi. An Approach for Integrated Management of Networks with Quality of Service Support Using QAME. In *IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, DSOM*, pages 167–178, Nancy, France, October 2001.

[15] TU Braunschweig and NEC C&C Europe. Jasmin: A Script-MIB Implementation. http://www.ibr.cs.tu-bs.de/projects/jasmin/, 2003.

[16] PHP: Hypertext Preprocessor. PHP. http://www.php.net/, 2004.

[17] The APACHE Software Foundation. APACHE. http://www.apache.org/, 2004.

[18] T. Oetiker. Round Robin Database Tool (RRDTool), 2003. http://www.rrdtool.org.

[19] Net-SNMP SourceForge project. NET-SNMP. http://www.net-snmp.org/, 2004.

[20] L. Z. Granville, G. A. F. S. Coelho, M. J. B. Almeida, and L. M. R. Tarouco. PoP - An Automated Policy Replacement Architecture for PBNM. In *IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY*, pages 140–146, June 2002.

[21] D. Ayala. NuSOAP - Web Services Toolkit for PHP. http://dietrich.ganx4.com/nusoap/, 2003.

[22] PEAR PHP Extension and Application Repository. PEAR::PACKAGE::SOAP. http://pear.php.net/package/SOAP, 2004.