
Universidade de Aveiro Departamento de Electrónica e de Telecomunicações
2002

Rui Pedro Sanches de Castro Lopes

Gestão Distribuída em SNMP

Universidade de Aveiro Departamento de Electrónica e de Telecomunicações
2002

Rui Pedro Sanches de Castro Lopes

Gestão Distribuída em SNMP

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Electrotécnica, realizada sob a orientação científica do Prof. Dr. José Luís Oliveira, Professor Associado do Departamento de Engenharia Electrónica e de Telecomunicações da Universidade de Aveiro

o júri

presidente

Prof. Dr. Joaquim José Borges Gouveia
professor catedrático da Universidade de Aveiro

Prof. Dr. Fernando Pedro Lopes Boavida Fernandes
professor associado da Universidade de Coimbra

Prof. Dr. Paulo da Costa Luís da Fonseca Pinto
professor associado da Universidade Nova de Lisboa

Prof. Dr. Joaquim Arnaldo de Carvalho Martins
professor associado com agregação da Universidade de Aveiro

Prof. Dr. José Luís Guimarães Oliveira
professor associado da Universidade de Aveiro

agradecimentos

O culminar de um trabalho desta dimensão é sempre uma excelente oportunidade para homenagear e agradecer a todas as pessoas que, de uma forma ou de outra, contribuiram para que a vida profissional, a aprendizagem, a camaradagem e a troca de experiências sejam sempre rodeadas da melhor convivência e de menores dificuldades.

Não posso agradecer ao meu orientador científico, Professor José Luís Oliveira, a sempre presente e exigente crítica construtiva que em muito ajudou para melhorar e completar este trabalho, sem referir a amizade, a eterna disponibilidade e a experiência que é trabalhar em comum. O rigor científico, a visão e a competência estiveram sempre associados à boa disposição, o que em muito facilitou a elaboração deste trabalho.

Aos meus colegas e amigos do Instituto Politécnico de Bragança e em particular ao Eng. José Rufino e ao Eng. Pedro Rodrigues devo um especial agradecimento não só pelo incentivo mas também pela sobrecarga que a minha dedicação à elaboração deste trabalho por vezes lhes causou e que sempre aceitaram de bom grado.

À Escola Superior de Tecnologia e de Gestão que, apesar das dificuldades, permitiu uma maior entrega e dedicação durante certas fases da evolução do trabalho.

Ao Ministério da Educação (Programa PRODEP) pelos apoios de vária ordem que me foram concedidos.

À minha família e à Sara agradeço o carinho, paciência, compreensão e encorajamento que me prestaram, imprescindíveis para a conclusão desta tese.

Por último, agradeço a muitos outros amigos, que deixo no anonimato por não querer correr o risco de esquecer algum.

A todos, um sincero obrigado.

resumo

A utilização intensiva de redes de comunicação heterogéneas e em crescimento acelerado é um desafio constante. Apesar do aumento de tamanho e de complexidade, as redes de comunicação de dados tornaram-se num factor crítico de sucesso de muitas organizações. Durante a última década foram desenvolvidas e normalizadas arquitecturas de gestão de redes para a criação de um ambiente aberto que permite controlar e optimizar de forma eficiente o seu funcionamento.

Muitas destas arquitecturas seguem uma estrutura centralizada que, apesar de aceitável há uns anos, revela-se insuficiente para lidar com o aumento de dimensão e de tráfego. Durante os últimos anos, surgiram várias propostas, modelos e arquitecturas de distribuição de gestão que procuram eliminar os problemas da centralização. O cenário tradicional em que uma única estação de gestão é responsável pela globalidade de sistemas geridos é modificado de forma a abarcar a tecnologia que permite delegar esta responsabilidade para a própria rede.

Este cenário propõe aumentar a robustez do sistema de gestão pela introdução de redundância e independência relativamente a ligações intermitentes. Além da robustez, a escalabilidade em termos de carga de processamento e de tráfego de gestão, principalmente para gestão “em banda”, é um objectivo essencial para redes modernas. Por último, a flexibilidade é também aumentada pela delegação dinâmica de código e de processos.

Dados a dependência actual dos serviços de comunicação de dados, o aumento de complexidade e o aumento de dimensão das redes actuais, a distribuição de gestão é um processo não só solicitado como inevitável.

abstract

The intensive operation of constantly growing, heterogeneous networks is a challenging task. Besides the increase in size and complexity, data networks have become a critical factor for the success of many organizations. During the last decade, many network management architectures where developed and standardized, aiming at the definition of an open environment to control and optimize its operation.

Many of these architectures follow the centralized model which, although acceptable a few years ago, are no longer able to deal with the size and traffic growth. During the last years, several distribution models and architectures where proposed aiming at eliminating the centralization problems. The traditional scenario where a single station is responsible for all the managed systems is changed in order to accept the technology which allows delegating this responsibility to the network.

This scenario suggests increasing the robustness of the network management system by introducing redundancy and independence in intermittent connections. Moreover, scalability in terms of network and processing load, particularly for in-band management, is a top level goal for modern networks. At last, flexibility is also increased through the dynamic delegation of code and processes.

According to the nowadays dependency of data communications services and the increase on size and complexity, the distribution of network management tasks is not only required but also inevitable.

Índice

Índice de Figuras

Índice de Tabelas

Every piece of work, if taken seriously, becomes a mountain of worry.

Sage, em "Groo: o errante"

1 Introdução

A instalação inicial de uma rede de comunicação de dados requer um estudo aprofundado e adaptado às condições de operação. Geralmente, este processo tem início com o levantamento de necessidades de comunicação dentro de cada grupo e entre os grupos existentes na organização. De acordo com a dimensão pretendida, a rede é estruturada em secções ou domínios de forma a permitir mais facilmente adaptar e optimizar a rede a cada cenário. De acordo com o estudo prévio são definidas políticas e perfis de utilização que irão ditar os requisitos em termos de velocidade e de qualidade de comunicação.

A troca de informação será sempre um possível alvo de ataques, pelo que deverão existir desde cedo políticas de segurança adequadas de forma a evitar, tanto quanto possível, desagrados ou prejuízos devidos a ataques à segurança.

Os requisitos serão, posteriormente, comparados com as possibilidades do equipamento a adquirir tendo em vista o melhor compromisso entre a qualidade e o preço. O mesmo processo é seguido para a instalação de cablagem obedecendo as normas mais recentes para a cablagem estruturada.

Após desenhar a solução global em termos de infra-estrutura, escolher o equipamento e instalar o meio físico de comunicação, será necessário ligar o equipamento, devidamente protegido contra sobrecargas, sobretensões, falhas de energia e acessos indevidos por parte de indivíduos não autorizados.

Num mundo ideal, a tarefa de instalação e operação de uma rede de comunicação de dados terminaria aqui. No entanto, há um conjunto de factores e de modos de actuar que fazem com que a utilização da rede seja acompanhada de riscos e de falhas e que solicitam um acompanhamento contínuo.

A comunicação pode ser interrompida devido a diversos motivos, nomeadamente a avarias no equipamento, problemas de ligação ou outros. A rede pode crescer o que pode trazer

implicações nas políticas inicialmente estabelecidas de qualidade e segurança. Aplicações podem mudar e, consequentemente, provocarem alterações a nível de tráfego.

A operacionalidade da rede depende assim de inúmeros factores dos quais destacamos [Simões00]:

- monitorar constantemente o funcionamento dos sistemas de comunicação;
- realizar actualizações ao equipamento e ao software sempre que houver a necessidade para tal;
- manter registos estatísticos do comportamento da rede de forma a permitir antecipar tendências de crescimento;
- planear atempadamente o crescimento da rede;
- manter e configurar o sistema;
- detectar e prever problemas de segurança, incluindo a definição de planos de recuperação de dados e a sensibilização dos utilizadores para uso das normas de segurança adequadas.

Por outras palavras, é necessário efectuar a gestão da rede de forma a controlar o seu funcionamento com o objectivo de maximizar a sua eficiência e produtividade. Esta actividade, apesar de julgada secundária por muitas instituições, é fundamental e pode absorver uma fatia considerável do orçamento disponível para a instalação de uma rede de comunicação de dados.

1.1 Enquadramento

A obtenção e transporte de informação de funcionamento da rede constituem alguns dos processos fundamentais de gestão de redes. Devido às diferenças intrínsecas a este tipo de ambiente provocadas tipicamente pela diversidade de aplicações, de serviços e de material é importante dispôr de um ambiente aberto que permita homogenizar a forma de obter e de interpretar a informação de gestão. Durante as últimas décadas surgiram modelos e protocolos que procuram lidar com a diversidade de tecnologia e com a dispersão geográfica que muitas vezes se fazem sentir. De facto, os fabricantes na procura pela supremacia comercial, definem as suas próprias ferramentas de gestão com particularidades próprias, o que dificulta a definição de mecanismos uniformes para gerir os diversos elementos. Por outro lado, o acesso local a todo o equipamento de rede pode não ser possível devido, por exemplo, à dispersão geográfica da infraestrutura. Por este motivo, é necessário um mecanismo que permita não só consultar informação mas também configurar remotamente o equipamento.

Entre os modelos com maior sucesso encontra-se o modelo SNMP que procura definir um ambiente de gestão que deverá ser suportado por todos os elementos a gerir independentemente das suas diferenças e do fabricante, orientando-se para a interoperabilidade de gestão. O SNMP prevê um conjunto de processos instalados directamente no equipamento de rede com a finalidade de recolha de informação e de recepção de comandos de configuração. Estes processos ficaram conhecidos como agentes.

Para enviar os comandos e processar a informação foi definida uma outra entidade – a estação de gestão – uma aplicação responsável por um (tipicamente alargado) conjunto de agentes de gestão. Pode-se distinguir um agente de uma aplicação de gestão pela necessidade de

comunicação: o agente não necessita comunicar para desempenhar o seu papel, enquanto que a aplicação de gestão depende da informação recolhida em outras entidades.

O sucesso inicial desta abordagem, associada à sua simplicidade, despoletaram uma ampla disseminação do modelo constituindo ainda hoje a base de grande maioria das aplicações de gestão de redes.

No entanto, o modelo centralizado, ou mais correctamente de distribuição fraca uma vez que admite a existência de mais que uma estação central, não consegue lidar de forma eficiente com o carácter dinâmico das redes modernas. O progressivo aumento de quantidade de tráfego, de número de postos, de tipo e quantidade de aplicações, da distribuição dos sistemas e da diversidade tecnológica introduziram novas metas para a escalabilidade, flexibilidade, desempenho e robustez do sistema de gestão. Este facto causou um forte movimento de investigação em torno de modelos e técnicas que potencialmente permitem distribuir as operações de gestão.

Neste sentido foram propostas algumas abordagens que procuram aplicar tecnologias de sistemas distribuídos como a arquitectura CORBA [JIDM], agentes móveis [Pham98] ou Java [JMX] e outras abordagens que sugerem a definição de arquitecturas especialmente desenhadas para a gestão distribuída – gestão por delegação (MbD – *Management by Delegation*) [Yemini91].

No entanto, apesar de mostrarem o seu valor para a distribuição de gestão, estas abordagens sofrem do inconveniente de nem sempre apresentarem um caminho claro de integração ou mesmo interoperabilidade com o modelo de gestão mais conhecido e divulgado – o SNMP. O passado tem demonstrado que para que uma tecnologia de gestão possa ser prontamente aceite e utilizada, terá de ser compatível com o SNMP.

Neste contexto, surge o trabalho realizado pelo IETF por intermédio do grupo de trabalho DISMAN – *Distributed Management* [DISMAN]. No âmbito do modelo SNMP, o grupo procurou definir um conjunto de ferramentas que permitissem distribuir as tarefas de gestão por um número indefinido de gestores intermédios.

1.2 Objectivos

O grande objectivo deste trabalho é estudar e aplicar tecnologias de distribuição de gestão em ambientes SNMP. Neste contexto, foram focadas duas áreas complementares: a) distribuição de operações utilizando ferramentas definidas pelo grupo de trabalho DISMAN e b) a sua integração com plataformas de agentes móveis.

A associação das duas áreas permite definir arquitecturas de gestão cobrindo todos os graus de distribuição, desde o centralismo, passando pela distribuição fraca e forte e mesmo permitindo a gestão cooperativa [Martin98].

Não se encontra no âmbito deste trabalho apresentar uma arquitectura específica adaptada a um caso particular. O resultado deve ser visto como uma camada intermédia (middleware) de ferramentas standard e abertas que permitem, de acordo com as políticas de gestão definidas pelo utilizador, desenhar arquitecturas de gestão distribuída.

O primeiro objectivo específico, ou seja, a distribuição de operações utilizando ferramentas DISMAN, implica conhecer a documentação, avaliar a sua extensão e complexidade de forma a construir uma ideia do grau de dificuldade que a adopção deste tipo de ferramentas tipicamente introduz. É necessário dispor de implementações para avaliar em ambiente controlado o tipo de operações disponíveis e as suas interdependências.

O segundo objectivo, ou seja, a integração de ferramentas DISMAN com plataformas de agentes móveis, introduz problemas de interoperabilidade entre as plataformas de agentes móveis e o SNMP. Os agentes móveis potenciam formas flexíveis e escaláveis de gestão. No entanto, a sua introdução reveste-se de alguns cuidados devido ao facto de terem de conviver com ambientes de gestão instalados. Por outro lado, a gestão de agentes móveis é também um factor fundamental, pelo que é necessário dispôr de ferramentas que permitam controlar o seu próprio funcionamento. Não é objectivo deste trabalho apresentar arquitecturas ou metodologias de funcionamento com base em agentes móveis. É importante, por outro lado, adoptar uma arquitectura de integração aberta e que permita abarcar diferentes plataformas de agentes móveis de forma a não limitar ou reduzir a solução a apenas uma ou outra favorita. Tal como as linguagens de programação, não há apenas uma plataforma de agentes.

Trabalho realizado pelo OMG (*Object Management Group*) sugere utilizar interfaces CORBA para uniformizar a gestão de plataformas de agentes móveis – MAF (*Mobile Agent Facility*). Esta tese propõe associar a esta tecnologia um sistema de informação adequado para a adaptar ao modelo SNMP.

1.3 Organização do documento

Este documento encontra-se estruturado em oito capítulos sendo o primeiro esta Introdução (Figura 1.1).

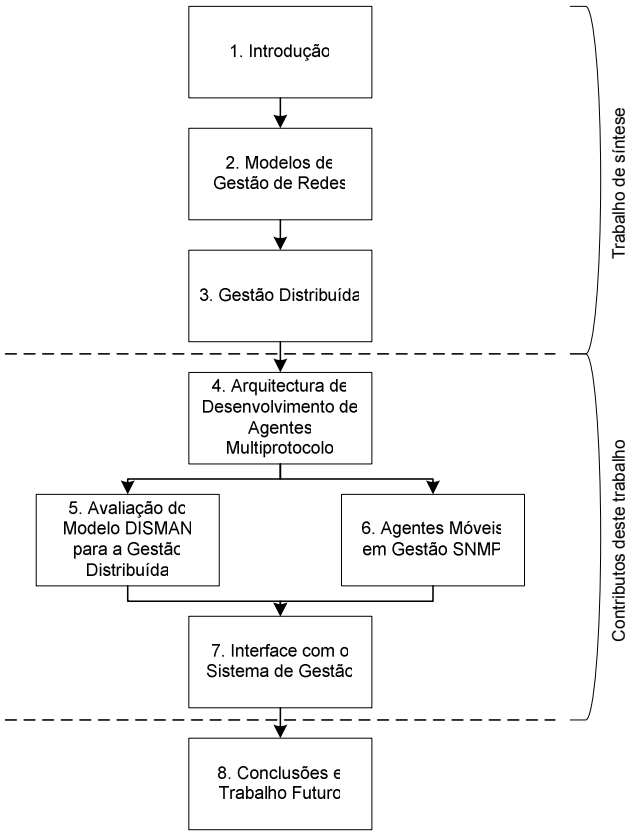


Figura 1.1 – Organização do Documento.

O capítulo 2 é dedicado a uma revisão geral de modelos e tecnologia de gestão de redes, cobrindo o modelo de gestão Internet, o modelo de gestão OSI, o modelo DMTF (*Desktop Management Task Force*), a gestão com base em Java e a gestão baseada em CORBA. Apesar de ser feita uma abordagem superficial providencia um resumo importante para a introdução e compreensão de conceitos utilizados ao longo da tese.

O capítulo 3 inicia-se com uma revisão de tecnologias de sistemas distribuídos, nomeadamente, *sockets*, RPCs, RMI, CORBA e Agentes Móveis passando de seguida para uma exposição das ferramentas DISMAN. Este é uma síntese de algumas tecnologias de distribuição de objectos quer no âmbito dos sistemas distribuídos como no âmbito do SNMP. Pretende-se avaliar as funcionalidades genéricas de distribuição de tarefas para uma eventual aplicação em cenários de gestão de redes e comparar estas tecnologias com tecnologia específica de distribuição de operações de gestão em SNMP.

O capítulo 4 descreve o primeiro contributo deste trabalho e que consiste numa camada de software (middleware) para o desenvolvimento de agentes de gestão multiprotocolo. Tendo em vista a interoperabilidade é fundamental prever uma multiplicidade de métodos de acesso tais como SNMP, HTTP (HTML ou WML), CORBA, RMI e/ou outros. Ao mesmo tempo, reúne ferramentas para facilitar o desenvolvimento de agentes de gestão.

Neste contexto, os agentes são construídos recorrendo à herança de classes e especialização de funções de forma a moldar a personalidade específica de cada agente. A funcionalidade intrínseca dos agentes, nomeadamente a manipulação de mensagens do tipo *walk* (`get-next`), a criação e eliminação dinâmica de linhas em tabelas, a indexação de objectos de gestão entre outras encontra-se já prevista no próprio software. Outro aspecto importante é a possibilidade de utilização de diferentes protocolos de transporte de informação de gestão e métodos de acesso sem haver necessidade de editar ou recompilar o código. Este é um contributo importante do trabalho e resultou no desenvolvimento e disponibilização de uma API (*Application Programming Interface*) em domínio público.

O capítulo 5 apresenta detalhes de implementação, utilização e avaliação de diversas ferramentas DISMAN. Neste âmbito foram desenvolvidos vários módulos, nomeadamente, a Expression MIB, a Schedule MIB e a Event MIB para obter uma ideia clara da complexidade envolvida nesta operação, na possibilidade real de distribuição de operações de gestão e dos possíveis obstáculos à sua utilização.

O capítulo 6 incide sobre a integração da gestão de plataformas de agentes móveis com o modelo SNMP. Dado que neste momento apenas é conhecida uma abordagem de interoperabilidade entre plataformas de agentes móveis, designada como MAF (*Mobile Agent Facilities*) e dado que não há qualquer requisito inicial relativo à escolha da plataforma de agentes móveis, foi utilizada esta norma para efectuar a gestão de agentes móveis. Neste sentido, é proposta uma MIB específica para converter as interfaces MAF em comandos SNMP e vice-versa. Esta MIB, designada como MAF-MIB, deu corpo a um agente podendo também ser considerado como um procurador (*proxy*) ou *gateway*. Cada mensagem recebida é convertida em invocações sobre a plataforma de agentes móveis sendo o resultado colocado à disposição do gestor em várias tabelas do agente.

O utilizador é um elemento fundamental na gestão de redes. É importante este dispor de ferramentas que lhe permitam interagir eficazmente com a própria rede de forma a poder construir modelos de funcionamento, consultar e alterar o estado do equipamento e das aplicações. O capítulo 7 descreve uma aplicação de gestão modular capaz de carregar em tempo de execução uma funcionalidade que inicialmente não se encontrava disponível. Esta

abordagem permite salvaguardar o investimento em soluções de gestão bem como manter sempre actualizada a aplicação pela adição ou substituição de módulos.

Neste capítulo é também apresentada uma alternativa baseada em interface Web disponibilizada directamente pelo agente. Este pode disponibilizar acesso directo à informação de gestão via Internet em duas formas de acesso, nomeadamente, por navegadores de Internet (*browsers*) e terminais WAP.

Por último, o capítulo 8 apresenta as conclusões e o resumo dos resultados obtidos. Este capítulo apresenta também ideias para possível trabalho futuro.

Network Management is the process of controlling a complex data network to maximize its efficiency and productivity.

Leinwand Conroy, "Network Management, a practical perspective".

2 Modelos de Gestão de Redes

As redes de comunicação são actualmente o sustentáculo de um modelo social e económico fazendo parte integrante da nossa forma de vida. A sua importância tem vindo a crescer nas últimas décadas e prevê-se que o seu papel seja cada vez mais central nesta sociedade de informação.

Na comunicação social, bem como em conversas informais do dia-a-dia, surgem com crescente frequência referências a dispositivos e tecnologias que permitem aceder a conteúdos e informação em qualquer lado e em qualquer altura. A Internet alarga continuamente o alcance para além do PC de secretária em direcção aos telefones móveis, agendas electrónicas e dispositivos domésticos. Esta expansão deve-se, principalmente, às tecnologias abertas que suportam a Internet assim como o conteúdo baseado em linguagens de marcas, como o HTML.

Neste contexto, a gestão de redes é uma tarefa de enorme complexidade, cujo sucesso ou insucesso tem um enorme impacto nos utilizadores e depende do conhecimento preciso dos parâmetros de funcionamento do equipamento que a constitui. Por equipamento entenda-se não só o material activo e passivo (hardware) mas também o software, como sejam os sistemas operativos ou as aplicações. É importante que esta informação seja o mais completa e diversificada possível, pois é com base nela que as decisões são tomadas.

Por este motivo, cada elemento de rede é modelado segundo a informação de gestão relevante para descrever o seu estado de funcionamento. Esta informação terá necessariamente de ser acessível a dispositivos externos de forma a estes poderem construir um modelo de funcionamento do elemento.

Os modelos de gestão estão, assim, orientados para a consulta, transporte e modificação de informação, acedida usando um modelo de base de dados virtual. A razão do termo "virtual"

deve-se a que a base de dados não armazena, na realidade, os parâmetros de funcionamento, obtendo-os por intermédio de operações de instrumentação sobre o equipamento de rede. De acordo com esta arquitectura, um modelo de gestão é geralmente constituído por uma ou mais entidades responsáveis pela instrumentação, vulgarmente conhecidos por agentes, um mecanismo de comunicação e por um posto de monitorização e controlo onde o modelo de funcionamento é criado, ou seja, a estação de gestão (Figura 2.1).

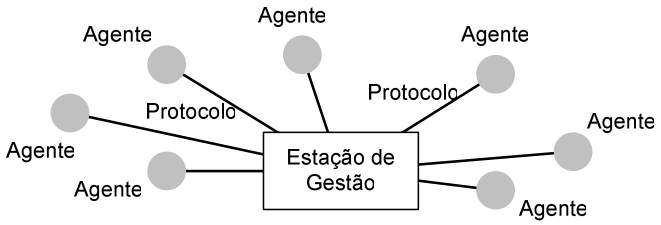


Figura 2.1 – Modelo genérico da arquitectura de gestão de redes.

Têm vindo a surgir, ao longo dos últimos anos, várias instâncias do modelo anterior como resposta a requisitos de interoperabilidade entre equipamento de diferentes fabricantes sendo o modelo SNMP um dos mais conhecidos e utilizados.

2.1 SNMP

A arquitectura de gestão baseada no SNMP (*Simple Network Management Protocol*) surgiu no fim da década de 80 como uma solução provisória vocacionada essencialmente para sistemas TCP/IP. Esta sobreviveu até aos dias de hoje passando por três grandes evoluções (Figura 2.2).

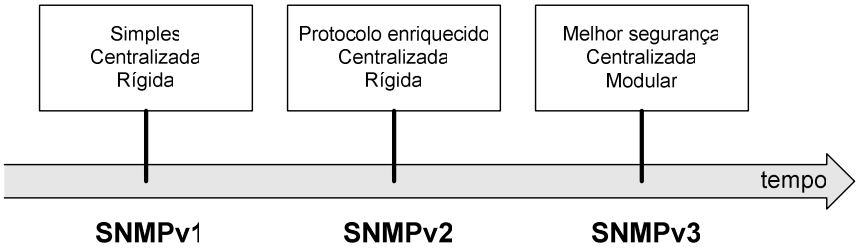


Figura 2.2 – Evolução da arquitectura SNMP.

A primeira versão da arquitectura SNMP encontra-se descrita em apenas três documentos o que possibilita uma rápida compreensão e facilita o desenvolvimento de aplicações de gestão [RFC1155][RFC1156][RFC1157]. Esta simplicidade despertou a atenção do mercado que depressa a adoptou para os seus produtos. Actualmente, ainda é a versão mais comum em todo o mundo.

De acordo com o modelo genérico de gestão de redes, a arquitectura SNMP prevê um formato de representação de informação denominado SMI (*Structure of Management Information*) [RFC1155]. O funcionamento dos elementos de redes é, portanto, descrito neste formato, em módulos que agrupam informação relacionada. Cada módulo é designado por MIB (*Management Information Base*) [RFC1156].

O protocolo de comunicação associado à arquitectura e que lhe atribui o nome, consiste num protocolo simples que permite inspeccionar e modificar a informação de gestão de um

elemento remoto de rede (agente), bem como o transporte de notificações geradas por estes (traps) [RFC1157]. O protocolo especifica os seguintes tipos de operações:

- Operação de leitura (`get`) – permite consultar um determinado parâmetro de um agente.
- Operação de actualização (`set`) – permite modificar um determinado parâmetro de um agente.
- Operação transversal (`get-next`) – permite consultar parâmetros de um determinado agente sem necessidade de o conhecer previamente.
- Operação de notificação (`trap`) – permite que cada agente notifique a ocorrência de eventos extraordinários.

Os mecanismos de segurança associados à arquitectura SNMPv1 existem mas são muito limitados, definindo apenas uma política de autenticação e de controlo de acesso. Para o efeito, introduz o conceito de comunidade, identificada por um nome, único em cada agente, e define uma relação entre um agente e um conjunto de estações gestoras. As estações gestoras pertencentes a uma comunidade necessitam apenas indicar o nome da comunidade para todas as operações de consulta ou actualização.

Com o aumento do número de utilizadores, foi sentida a necessidade de acrescentar mais funcionalidade, particularmente ao nível de protocolo e de segurança. A versão 2 teve um sucesso relativo no primeiro requisito com a adição de novos tipos de mensagem [RFC1905]:

- Operação de informação (`inform`) – permite a troca de informação entre estações gestoras.
- Operação de consulta múltipla (`get-bulk`) – é semelhante à operação transversal (`get-next`) mas torna possível a especificação de múltiplos parâmetros.
- `get` não atómico – a falha de consulta a uma variável não impede o comando de prosseguir com outras consultas.

No entanto, o modelo de segurança do SNMPv2 não conseguiu estabelecer medidas consensuais e robustas, o que levou a que a arquitectura fosse muitas vezes considerada inadequada para operações de configuração.

O SNMPv3, proposto mais recentemente [RFC2570], assenta no protocolo definido para a versão 2 e acrescenta melhores mecanismos de segurança, particularmente em termos de autenticação, privacidade e controlo de acesso. Outra característica do SNMPv3 é a sua modularidade, que torna possível a utilização de protocolos existentes e futuros em determinadas funções sem a necessidade de actualizar ou de emitir novos RFCs para toda a arquitectura.

O acréscimo de funcionalidade em relação às versões anteriores resultou, em primeira instância, num aumento do número de documentos. Estes descrevem os diversos aspectos da arquitectura, nomeadamente, processamento de mensagens, processamento de PDUs (*Protocol Data Unit*), modelo de informação, módulos MIB e documentos genéricos (Figura 2.3) [RFC1905 a RFC2580].

A série de documentos tem início com uma introdução à arquitectura SNMPv3 e a sua relação com as versões anteriores [RFC2570]. Como complemento, descreve resumidamente cada documento e apresenta a sua relação com os demais.

A adopção de novas versões implica a coexistência com versões anteriores podendo dar lugar a incompatibilidades na comunicação, no acesso à informação ou no modelo de segurança. O documento [RFC2576] procura esclarecer quais as possíveis colisões e qual a forma de as resolver.

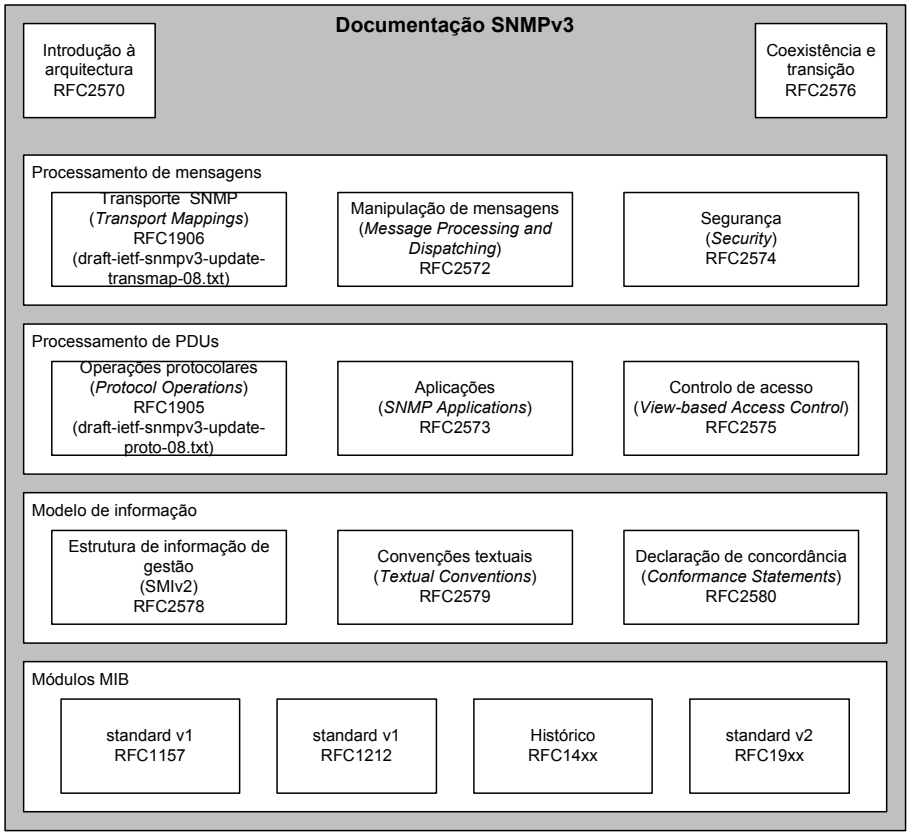


Figura 2.3 – Conjunto de documentos do SNMPv3.

As mensagens SNMPv3 podem ser encapsuladas em diferentes protocolos de rede, nomeadamente UDP (*User Datagram Protocol*), CLTS (*OSI Connectionless-mode Transport Service*), DDP (*AppleTalk*) ou IPX (*Packet Exchange Protocol*) codificadas de acordo com as BER (*Basic Encoding Rules*). O documento [RFC1906] esclarece quais os algoritmos associados a cada protocolo.

A manipulação de mensagens refere-se à forma como as mensagens devem ser processadas e encaminhadas. O documento [RFC2572] descreve um mecanismo que, de acordo com o formato da mensagem, identifica o módulo associado para a poder interpretar. De igual forma, prossegue com a delegação de segurança para os módulos adequados e, caso haja sucesso, o encaminhamento da mensagem para a aplicação SNMPv3. Central a todo este processo encontra-se um módulo conhecido como despachante.

O documento [RFC2574] apresenta o modelo de segurança com base em nomes de utilizadores e como este pode prevenir diversos ataques. Este mecanismo é particularmente importante para proteger mensagens de configuração de agentes.

O protocolo SNMPv3 define vários tipos de operações para transportar informação. Estes são descritos em [RFC1905].

De acordo com a função desempenhada, uma aplicação SNMPv3 pode ter um papel de agente, de gestor ou de procurador (*proxy*). Estes tipos são descritos pormenorizadamente em [RFC2573].

O controlo de acesso é um aspecto relacionado com a segurança e que permite designar direitos de acesso à informação de gestão. Este controlo pode ser efectuado tanto na geração de notificações como na consulta de informação [RFC2575].

A informação de gestão é, como referido anteriormente, definida de acordo com uma estrutura – SMI. A arquitectura SNMPv3 utiliza a versão SMIv2, onde especifica a sintaxe e semântica para a definição de tipos de dados. A documentação define os tipos de dados básicos em [RFC2578] e algumas convenções, ou seja, diferentes formas de interpretar os tipos básicos em [RFC2579].

O desenvolvimento de aplicações SNMPv3, nomeadamente com a função de agentes, poderá assumir diferentes graus, de acordo com o patamar de funcionalidade requerida. Os módulos designam estes patamares como declarações de concordância e apresentam-se definidos numa notação própria [RFC2580].

Existem muitos outros documentos, relacionados com a arquitectura SNMPv3, que especificam módulos MIB, ou seja, agrupam tipos de dados de acordo com determinada funcionalidade. Assim, é possível encontrar informação que modela interfaces de rede, unidades de alimentação ininterruptas, sondas de tráfego, encaminhadores, estações de trabalho, entre outras, numa listagem bastante extensa.

Apesar da extensão da documentação, a versão 3 do SNMP apresenta a mesma arquitectura genérica das versões anteriores, ou seja, é constituída por uma estação de gestão e por um ou mais agentes. Cada agente possui um armazém virtual de informação de gestão, denominado MIB estruturado em vários objectos, cada um dos quais caracterizado por um nome, uma sintaxe e uma codificação e definidos numa linguagem abstracta, a *Abstract Syntax Notation One* (ASN.1) [ISO8824].

Cada objecto é inequivocamente identificado por um nome (OID – *Object Identification*), que consiste numa sequência de números inteiros separados por pontos ('.') em que cada um destes números identifica um nó de uma árvore hierárquica. A árvore consiste numa raiz ligada a um número de nós. Por sua vez, cada um destes nós poderá estar ligado a outros nós, também numerados, constituindo assim uma sub-árvore. O processo pode continuar até um número indeterminado de camadas. Por exemplo, a sequência de nós raiz-1-3-6-1-2-1 designa o identificador .1.3.6.1.2.1, ou seja, .iso.org.dod.internet.mgmt.mib-2 (Figura 2.4). O nó correspondente à raíz é desnecessário pelo que não é representado. De referir que a numeração é atribuída com base num processo administrativo [RFC1066].

A sintaxe de um determinado tipo de objecto define a estrutura de dados correspondente. Poderá ser algum dos tipos básicos definidos pelo ASN.1, ou seja, `INTEGER`, `OCTET STRING`, `OBJECT IDENTIFIER`, `SEQUENCE`, `SEQUENCE OF`, tipos definidos ao nível de aplicação, como sejam, `Integer32`, `IpAddress`, `Counter32`, `Gauge32`, `Unsigned32`, `TimeTicks`, `Opaque`, ou `Counter64`, alguma convenção (por exemplo, `DateAndTime`, `DisplayString`, `PhysAddress`, `TruthValue`, ...) ou o construtor `BITS` [RFC2578].

Por último, a codificação está relacionada com a forma como o objecto é representado quando transmitido para a rede. Este é codificado segundo um conjunto de regras, as *Basic Encoding Rules* (BER) [ISO8825], definidas para a ASN.1, de modo a permitir a transmissão inequívoca de informação.

Estas características de informação permitem dar a conhecer o formato dos parâmetros de funcionamento do equipamento de rede o que, só por si, não é suficiente. É necessário um mecanismo de transporte de informação que permita a comunicação entre os agentes e a estação de gestão bem como entre estações de gestão.

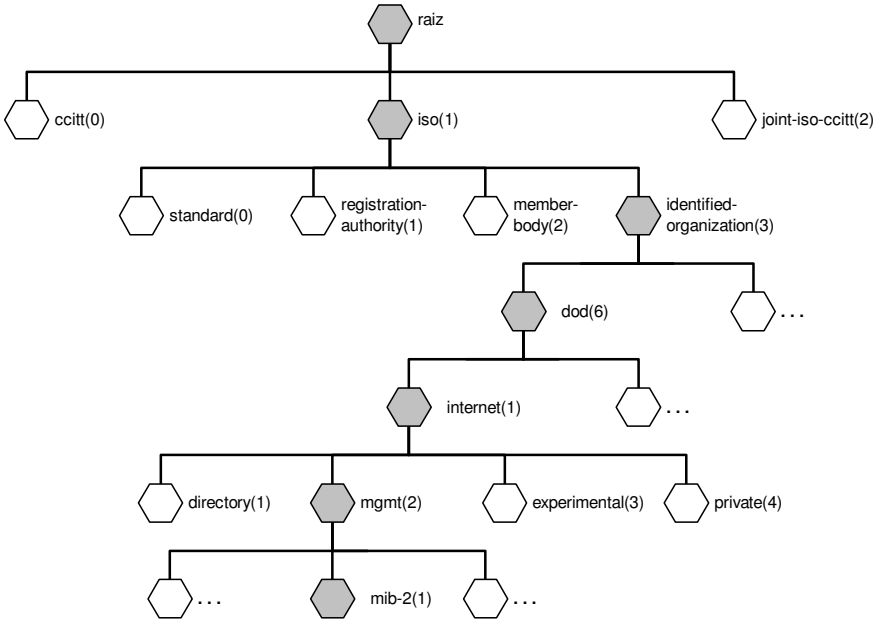


Figura 2.4 – Árvore de identificadores de objectos.

Apesar de manter a mesma arquitectura genérica das versões anteriores o SNMPv3 apresenta um modelo extensível, baseado no SNMPv2, mas que o suplementa ao nível do formato da mensagem, do modelo de segurança e do modelo de controlo de acesso.

A documentação define uma entidade SNMP como sendo constituída por um mecanismo SNMP e uma ou mais aplicações. O mecanismo SNMP providencia os serviços relacionados com a recepção, transmissão, autenticação e manutenção da privacidade de mensagens, bem como o controlo de acesso à informação de gestão. Para o efeito, é estruturado em quatro módulos: despachante, processamento de mensagens, sistema de segurança e controlo de acesso (Figura 2.5).

O despachante coordena a comunicação entre subsistemas e distingue os módulos pertencentes ao mesmo subsistema, por intermédio do despachante de mensagens. Determina, com base no PDU, que aplicação deve ser invocada (*Despachante de PDUs*) e coordena as correspondências com a camada de transporte (*Transporte*).

A concentração da tarefa de processamento de mensagens num único subsistema permite suportar diferentes formatos de mensagens com a substituição de um módulo. Além disso, torna possível a coexistência de vários formatos num único modelo, sendo invocado o processador adequado sempre que necessário [RFC2572]. Este processo opõe-se, com vantagens óbvias, à imutabilidade do formato da mensagem do SNMPv2, que obriga à definição de um novo conjunto de documentos caso seja necessária a sua alteração.

Uma filosofia idêntica é seguida para o módulo de segurança. A mensagem deve ser autenticada, de forma a não haver execução de comandos por parte de entidades não autorizadas, protegida quanto à leitura indevida e validada segundo informação temporal. Este

último aspecto visa proteger a mensagem quanto à duplicação e/ou o atraso propositado de comandos. A duplicação e atraso de um comando de reinicialização (*reboot*), por exemplo, pode ser desastroso quando efectuado de forma indevida.

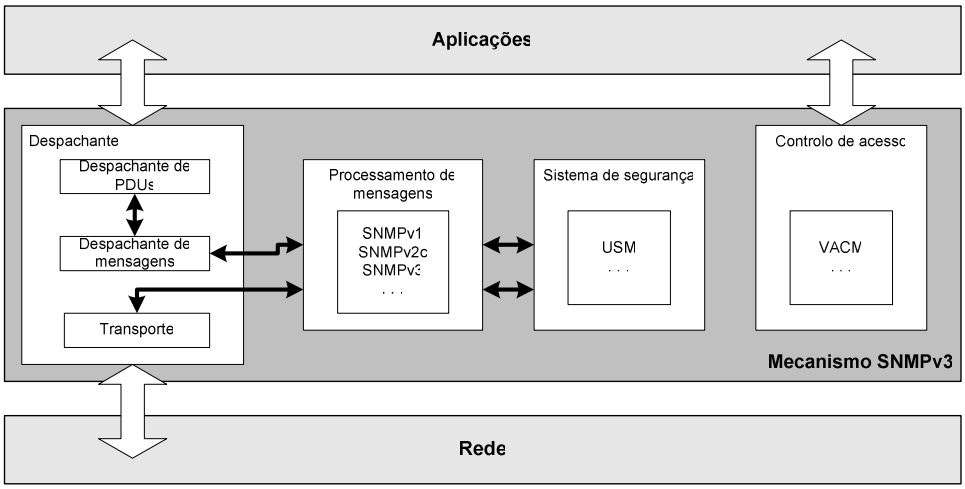


Figura 2.5 – Mecanismo SNMPv3.

Um dos pontos polémicos na definição do SNMPv2 foi, precisamente, que modelo usar para a segurança das mensagens. O SNMPv3 apresenta um subsistema responsável pelas tarefas de segurança tornando mais simples a substituição de um modelo de segurança e conseguindo simultaneamente a coexistência de várias soluções.

Num cenário real de funcionamento, quando uma mensagem é emitida, o despachante verifica a versão e tipo de protocolo seleccionado. De acordo com esta informação, o despachante invoca o processador de mensagens adequado que, por sua vez, inclui informação acerca do modelo de segurança a utilizar. A mensagem é, de seguida, entregue ao módulo de segurança para depois ser enviada. A recepção de uma mensagem segue o processo inverso: o despachante verifica a versão e tipo de protocolo indicado pela mensagem e invoca o processador de mensagens respectivo. Segue-se a etapa de autenticação, de descodificação e de verificação de atrasos. Se este processo terminar com sucesso, a mensagem regressa ao despachante que a encaminha para a aplicação adequada.

A aplicação, dependendo do tipo de comando, pode necessitar de serviços de controlo de acesso a objectos de gestão. Estes são fornecidos pelo subsistema de controlo de acesso pertencente ao mecanismo SNMPv3. O subsistema de controlo de acesso permite a coexistência de vários módulos distintos.

O agrupamento de aplicações define o papel que a entidade SNMP assume no contexto do sistema de gestão. A estação de gestão central (NMS – *Network Management Station*), por exemplo, pode ser composta por um gerador de comandos e um receptor de notificações. Por outro lado, o agente SNMP tradicional encontra-se estruturado em receptor de comandos e gerador de notificações (Figura 2.6). Para efeitos de coexistência com outro tipo de protocolos ou sistemas, encontra-se definida uma aplicação com função de procurador (*Proxy*) que tem o papel de adaptar a gestão não SNMP para gestão SNMP [RFC2573]. Esta aplicação permite integrar equipamento não compatível em sistemas de gestão SNMP de forma completamente transparente.

Podem ser utilizadas outras combinações de aplicações SNMP de forma a construir agentes ou gestores de gestão mais versáteis, optimizando o seu papel no sistema.

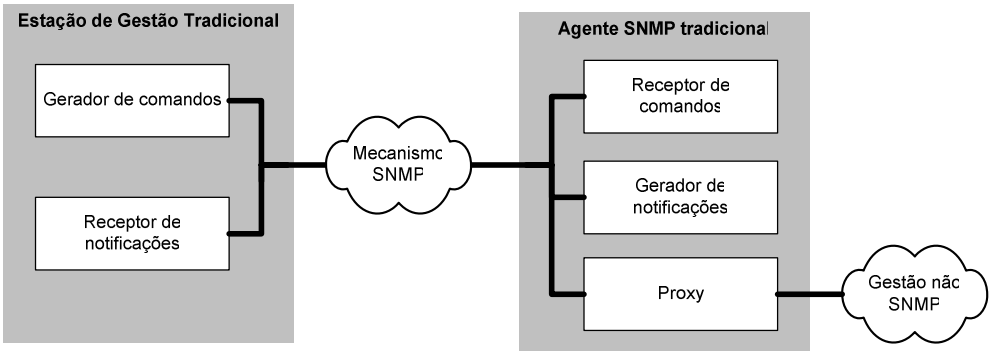


Figura 2.6 – Aplicações SNMP.

2.1.1 Identificação de informação de gestão

Os agentes possuem, tipicamente, mecanismos de instrumentação associados, ou seja, mecanismos de aquisição de informação e controlo de componentes de rede. A informação será, à partida, manipulada pelo receptor de comandos.

De uma forma geral, um conjunto de informação é acedida de acordo com um determinado contexto que, por sua vez, representa uma entidade física (como uma ponte ou um servidor), uma entidade lógica (um serviço) ou um conjunto de entidades. Cada contexto é válido apenas no domínio de uma entidade SNMPv3. Não é possível que um contexto se distribua por várias entidades SNMPv3.

Para identificar cada objecto são necessários quatro parâmetros: o identificador do mecanismo SNMPv3 (`snmpEngineID`), o nome do contexto (`contextName`), o identificador de objecto (OID – ex. `ifDescr`) e o identificador de instância (ex. '1'). No caso do SNMPv1, a consulta é efectuada apenas indicando o identificador de objecto e o identificador de instância, o que resulta numa maior simplicidade de acesso mas também numa menor flexibilidade.

2.1.2 Segurança

O SNMPv3 admite, como foi já referido, vários modelos de segurança. Estes são identificados por um campo no cabeçalho de uma mensagem SNMPv3. O grupo de trabalho definiu um modelo de segurança baseado em nomes de utilizadores, denominado *User-based Security Model* (USM) [RFC2574].

O USM define o conceito de "chefe" (*principal*), que representa um utilizador e é identificado por um nome (`userName`). O "chefe" tem a responsabilidade de armazenar chaves e outro tipo de informação de segurança, como o tipo de algoritmo criptográfico utilizado. As chaves são sequências de octetos utilizadas pelos protocolos de autenticação e privacidade.

É da responsabilidade do modelo de segurança garantir a autenticação, privacidade e manter o condicionalismo temporal das mensagens. A autenticação de mensagens é conseguida por intermédio de um Código de Autenticação de Mensagem (MAC – *Message Authentication Code*), único para cada mensagem [Blumenthal97]. Estes códigos funcionam como as "impressões digitais" da mensagem e são enviados juntamente com esta. A identidade do emissor é verificada com o auxílio do MAC e das chaves indicadas pelo "chefe".

A privacidade das mensagens é assegurada por intermédio de algoritmos de criptografia baseados nas chaves indicadas pelo "chefe" e pelo MAC.

O condicionalismo temporal tem a responsabilidade de detectar mensagens duplicadas e propositadamente atrasadas. Para o conseguir cada mensagem transporta uma marca temporal que pode ser examinada pelo receptor. Se a diferença entre a marca e o valor de relógio local exceder um certo limite a mensagem é rejeitada. O problema que se levanta reside na necessidade de sincronização entre os relógios do emissor e do receptor [RFC2574]. O cabeçalho de uma mensagem USM contém campos com informação acerca do número de vezes que um mecanismo SNMPv3 foi reinicializado (`engineBoots`) e com o número de segundos decorridos desde a última reinicialização (`engineTime`). A transmissão de uma mensagem não é instantânea, pelo que é definida uma "janela de oportunidade" (150 segundos) que define o limite de atraso de uma mensagem:

- Se o `engineBoots` indicado na mensagem for maior do que a noção do receptor, a mensagem é aceite e a noção local de `engineBoots` e `engineTime` são actualizados.
- Se o `engineBoots` indicado na mensagem for menor que a noção do receptor, a mensagem é rejeitada, uma vez que apresenta sinais de atraso.
- Se o `engineBoots` for igual à noção do receptor, o `engineTime` é comparado com a noção local respectiva. Se a diferença for inferior a 150s ("janela de oportunidade") esta será aceite, caso contrário, será rejeitada. Sempre que a mensagem é aceite, a noção local das marcas temporais são actualizados.

O mecanismo tal como foi apresentado permite a duplicação de mensagens ocorridas dentro da "janela de oportunidade", de modo que surge a necessidade de o complementar com um método adicional de protecção. Este método assenta num campo de identificação único de cada mensagem (`msgID`): se aparecerem dois `msgID` iguais dentro da mesma "janela de oportunidade", a mensagem é rejeitada.

2.1.3 Implementações

O programador, na tarefa de desenvolvimento de aplicações ou de agentes, recorre a bibliotecas de funções que implementam o mecanismo SNMP. Estas bibliotecas reunem a funcionalidade apresentada anteriormente sob uma interface bem definida que o programador terá de conhecer para a utilizar correctamente. No momento da escrita deste documento encontram-se já várias implementações em domínio público compatíveis com SNMPv3:

- Westhawk SNMP toolkit – linguagem Java (http://www.westhawk.co.uk/).
- NET-SNMP. Várias ferramentas incluindo: um agente extensível, biblioteca de funções SNMP, ferramentas de gestão SNMP entre outros – linguagem C (http://net-snmp.sourceforge.net/).
- OpenSNMP – linguagem C++ (http://sourceforge.net/projects/opensnmp/).
- ModularSnmp – linguagem Java (http://www.teleinfo.uqam.ca/snmp/).
- GxSNMP – linguagem C (http://www.gxsnmp.org/).

A adesão dos fabricantes também se faz sentir, existindo também um conjunto significativo de implementações:

- AdventNet SNMP v1, v2c, v3 API – linguagem Java (http://www.adventnet.com/).

- Ace*Comm WinSNMP – várias linguagens para ambiente Windows (http://www.winsnmp.com/).
- MG-SOFT WinSNMP – várias linguagens para ambiente Windows (http://www.winsnmp.org/).
- SNMP++ – linguagem C++ (http://www.agentpp.com/).

2.1.4 Lacunas associadas ao SNMPv3

O modelo apresentado ao longo dos pontos anteriores é extremamente dependente da estação central e do meio de comunicação, i.e. a gestão depende também do bom funcionamento da rede. Se a comunicação falta, por algum motivo, todo o sistema de gestão falha.

Por outro lado, a estrutura SNMP apresenta apenas soluções de baixo nível, como a descrição abstracta da informação de gestão (SMI e MIB) e a forma como aquela é acedida (protocolo de gestão – SNMP). Não define qualquer tarefa de nível mais elevado, como interpretação e correlação de dados bem como o tipo de medidas de correcção a tomar. Esta responsabilidade é tipicamente relegada de um modo não formal para a estação de gestão.

A arquitectura apresenta igualmente uma lacuna relacionada com a falta de extensibilidade e escalabilidade. Esta falta resulta da incapacidade de um único sistema centralizado em processar grandes quantidades de informação e da dificuldade em realizar uma monitorização periódica sobre pontos distribuídos por redes de grande dimensão.

Outro problema está relacionado com a quantidade de informação que o sistema gestor tem de manipular. Em 1998 existiam já mais de 100 módulos MIB num total aproximado de 10.000 objectos e este número tem continuado a crescer [Postel98].

Alguns autores consideraram já estes problemas nos anos decorrentes, resultando em soluções ad-hoc e parciais, tipicamente baseadas na distribuição de tarefas de gestão [Goldszmidt98, Oliveira95]. Mais recentemente, o IETF constituiu um grupo específico para considerar estas falhas, denominado DISMAN [DISMAN]. O trabalho produzido por este grupo será detalhado na secção 3.2 no contexto de gestão distribuída.

2.2 OSI

A ISO (*International Standards Organisation*), com a colaboração do ITU-T (*International Telecommunications Union Telecommunication Standardisation Sector*), desenvolveu um trabalho pioneiro na normalização de redes locais de comunicação de dados. Um dos resultados deste trabalho foi o modelo de referência de redes OSI (*Open Systems Interconnection*).

As directivas de gestão foram estabelecidas por [ISO7498-4], mais tarde complementadas por [ISO10040]. Um resumo dos documentos normativos enquadrados no trabalho de gestão OSI pode ser encontrado em [Langsford94]. Os documentos propõem uma divisão das tarefas de gestão nas seguintes áreas funcionais:

- Gestão de falhas – agrupa os mecanismos que permitem detectar, isolar e corrigir funcionamentos anómalos.
- Contabilização e gestão de recursos – contém mecanismos que permitem estabelecer custos de utilização de recursos da rede.

- Gestão de configuração – agrupa os mecanismos de controlo, identificação e consulta de dados necessários à operação dos dispositivos.
- Gestão de desempenho – tem a responsabilidade de avaliar o comportamento e eficiência da comunicação.
- Gestão de segurança – define mecanismos de segurança genérica de sistemas de comunicação.

A estrutura de informação de gestão é definida seguindo uma metodologia orientada ao objecto. Consegue-se, desta forma, utilizar os conceitos de classe, objecto, herança, encapsulamento, método e atributo na definição dos objectos de gestão. A ISO estabeleceu um conjunto de formalismos, conhecidos por GDMO (*Guidelines for the Definition of Managed Objects*) [ISO10165] com a finalidade de especificar os objectos de gestão. Este consiste numa linguagem de especificação de classes de objectos, do seu comportamento, atributos e herança.

Os objectos de gestão são representações lógicas de entidades físicas ou recursos associados a um sistema ou subsistema. Estes são organizados por classes, que definem as características comuns de um conjunto de objectos. Cada objecto reflecte continuamente o estado de uma determinada entidade, por intermédio dos valores armazenados nos seus atributos. Por outras palavras, os objectos de gestão definem uma fronteira lógica entre a entidade em causa e os mecanismos de gestão (Figura 2.7).

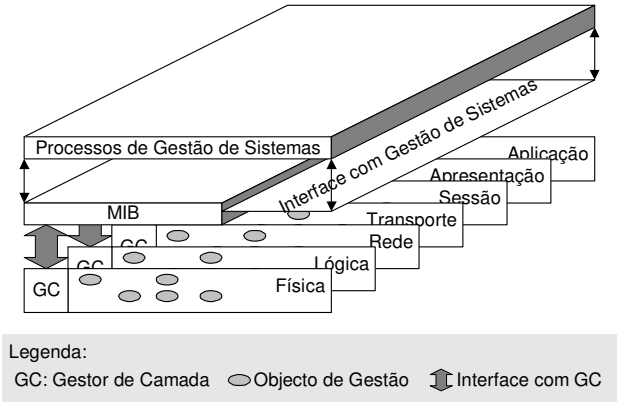


Figura 2.7 – Modelo da informação de gestão do modelo OSI.

A identificação de cada objecto de gestão é efectuado por intermédio de um identificador de objecto (OID – *Object Identifier*). Este especifica todos os aspectos inseridos no modelo de gestão OSI (tais como classes, objectos, atributos, operações, notificações e documentos).

O protocolo de gestão é o CMIP (*Common Management Information Protocol*) [ISO9596], que se encontra definido para a camada de Aplicação do modelo de referência OSI.

2.3 Modelo de gestão DMTF

A *Distributed Management Task Force* (DMTF – http://www.dmtf.org/), uma organização constituída por diversas empresas e associações industriais, tem como função principal propor normas e soluções de gestão de sistemas e de redes. Impulsionada por vários nomes sonantes da indústria (BMC Software, Cisco, Compaq, Dell, HP, IBM/Tivoli, Intel, Microsoft, NEC, Novell, Sun, Symantec, entre outros), depressa conseguiu reunir um grande número de associados.

O principal mote é propor um caminho claro para a interoperabilidade entre as diferentes soluções de gestão. No cerne desta filosofia encontra-se um modelo de informação genérico (CIM – *Common Information Model*), orientado ao objecto e com a possibilidade de definir correspondências com os modelos existentes.

2.3.1 Modelo de informação

O CIM consiste num formalismo que permite definir modelos ou representações de informação para unificar as normas existentes, como o SNMP, CMIP, DMI (*Desktop Management Interface* – ver secção 2.3.2) [CIM]. No contexto de gestão de redes, o modelo de informação requer a definição de um conjunto de tipos de dados e respectivas associações de forma a representar os vários aspectos de um sistema de gestão. Mais especificamente, o modelo de informação é usado para representar objectos reais segundo um paradigma orientado ao objecto, usando conceitos de classes e instâncias. Do ponto de vista lógico, o CIM é semelhante a uma base de dados orientada ao objecto.

O facto de ser OO introduz as seguintes características:

- Abstracção – características e propriedades comuns são agrupadas em contruções denominadas classes, reduzindo, portanto, a complexidade do problema em causa;
- Herança – a herança de objectos permite especializar determinadas funções embora mantendo o comportamento mais genérico definido nos objectos hierarquicamente superiores;
- Possibilidade de estabelecer dependências, relacões e associações entre objectos;
- Métodos comuns – a utilização de métodos comuns permite definir comportamentos semelhantes ao longo da estrutura de herança independentemente do grau de especialização, resultando, portanto, noutra forma de abstracção.

O elemento mais abrangente do CIM são os esquemas (*schemas*). Um esquema tem um identificador associado a uma única autoridade (*owner*) e agrupa classes pertencentes à mesma autoridade. Os nomes de classes terão de ser únicos dentro do esquema respectivo. Este é precedido pelo nome do esquema e pelo caracter '_': <nome do esquema>_<nome da classe>.

A classe é a definição básica de uma unidade de gestão. Esta descreve um armazém de funções e de campos de informação individuais denominados propriedades ou atributos. Cada atributo (ou propriedade) descreve aspectos particulares do componente descrito pela classe. Uma classe define um modelo para um objecto de gestão. Com base no modelo é possível definir objectos específicos criando instâncias da classe. Como exemplo, uma hipotética classe `Disco` poderia ter instâncias `disco_C` e `disco_D`. As classes podem ser organizadas segundo uma árvore em relações de herança simples.

As propriedades são os campos individuais descritos numa classe. Estas armazenam informação individual e particular a essa classe ou objecto. Normalmente, as propriedades não são acedidas directamente, mas por intermédio de métodos específicos, normalmente contendo a designação `get` e `set` seguida pelo nome da propriedade.

O CIM encontra-se estruturado em três níveis. O primeiro nível contém o esquema base, que consiste no conjunto de classes base, as suas propriedades e as suas associações (Figura 2.8) [CIMCore].

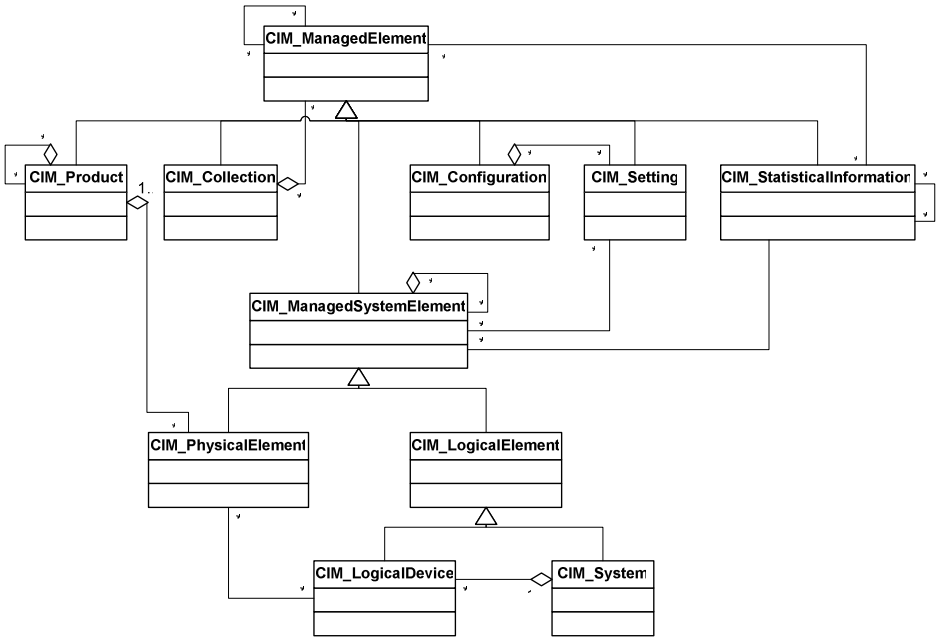


Figura 2.8 – A base da hierarquia de classes CIM.

Resumidamente:

- `CIM_ManagedElement` é a base da hierarquia e funciona como referência para as associações que incidem sobre todas as classes da hierarquia. Qualquer componente de um sistema é candidato a ser descrito por esta classe.
- `CIM_ManagedSystemElement` representa sistemas, componentes de sistemas, serviços, aplicações e redes.
- As classes representativas de elementos físicos e lógicos são subclasses de `CIM_ManagedSystemElement`. `CIM_PhysicalElement` representa qualquer componente físico como, por exemplo, uma placa de rede. `CIM_LogicalElement` representa qualquer componente que não é um sistema nem um `CIM_PhysicalElement`, como um ficheiro.
- `CIM_Product` representa contractos entre produtores e consumidores e reune informação sobre como foi adquirido determinado produto, como é efectuada a assistência técnica e onde se encontra instalado.
- `CIM_Setting` define os parâmetros específicos a serem aplicados a um ou mais `CIM_ManagedSystemElement` e encontra-se associada à configuração do equipamento.
- A `CIM_StatisticalInformation` é uma classe abstracta e representa dados estatísticos para um `CIM_ManagedElement`.
- `CIM_Collection` agrupa objectos do tipo `CIM_ManagedElement`.
- As associações (linhas com um losango – normalmente conhecidas como relações do tipo "posse") representam as possíveis ligações entre `CIM_ManagedElement` enquanto que as dependências descrevem ligações que obrigatoriamente têm de existir (linhas

sem terminação especial – normalmente conhecidas como relações do tipo "utilização").

O segundo nível define um esquema comum que, juntamente com o esquema base, providencia detalhes sobre o sistema de gestão. Trata-se de um conjunto de classes específicas mas independente da plataforma que designam sistemas, dispositivos, redes, utilizadores e aplicações. Estas áreas encontram-se interrelacionados por herança e associações entre objectos e especializam as classes e conceitos essenciais definidos no esquema base.

O terceiro nível contém extensões que representam plataformas, aplicações ou serviços específicos (Windows, UNIX).

2.3.2 Gestão de Sistemas – DMI

Independente da CIM, vocacionada para a integração e unificação de soluções de gestão, o DMTF possui a norma DMI (*Desktop Management Interface*) que define um modelo de gestão orientado para sistemas computacionais como as estações de trabalho [DMI].

Esta norma define uma camada de abstracção entre o software de gestão e os componentes de um sistema computacional, providenciando uma interface independente do sistema operativo, da estação de trabalho e mesmo do protocolo de gestão.

Por regra, o utilizador é forçado a lidar com um conjunto crescente de dispositivos e componentes que, por sua vez, introduzem mais informação de gestão e mecanismos próprios de monitorização e controlo. Tal como outros modelos de gestão, a DMI estabelece uma interface comum entre os distintos elementos de gestão e as aplicações de gestão (Figura 2.9).

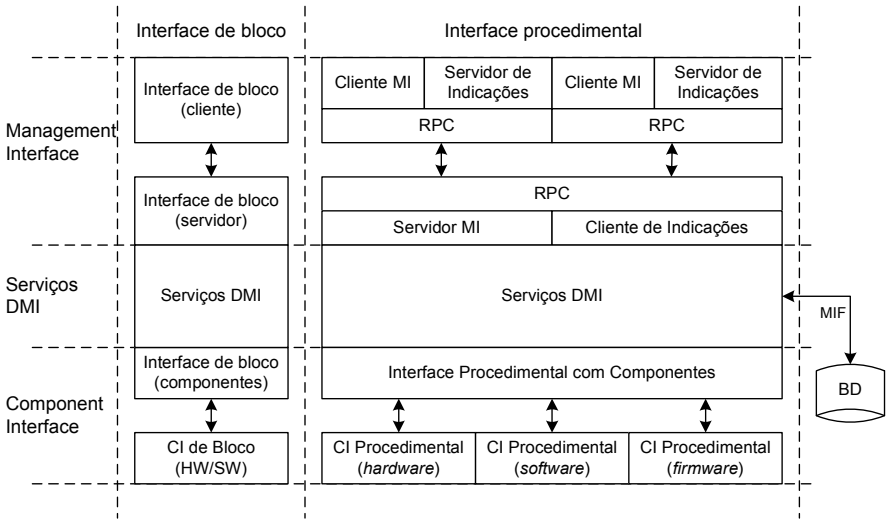


Figura 2.9 – Arquitectura DMI [DMI].

A arquitectura inclui quatro unidades elementares:

- Serviços DMI – conjunto de serviços que actuam como intermediários entre as aplicações de gestão e os componentes.
- MIF (*Management Information Format*) – define os atributos, guardados em base de dados, de estações de trabalho (PCs) e servidores.

- MI (*Management Interface*) – permite que as aplicações compatíveis com DMI tenham acesso aos serviços DMI e, consequentemente, aos componentes de gestão.
- CI (*Component Interface*) – permite que os componentes tenham acesso aos serviços DMI.

Inicialmente, a norma previa apenas interfaces de bloco (MI e CI), com apenas um procedimento (`DmiInvoke`), em que é passado um conjunto de estruturas de dados concatenadas. Na versão actual da norma (versão 2 – [DMI]) foram introduzidas interfaces com vários procedimentos facilitando o desenvolvimento de aplicações e reduzindo a possibilidade de ocorrência de erros. Em termos práticos, as interfaces são acedidas por intermédio de invocação remota de métodos (DCE/RPC, ONC/RCP ou TI/RPC) ou por protocolos de gestão do tipo SNMP ou CMIP, desde que devidamente adaptados [DMI2SNMP].

Cada componente é descrito numa linguagem conhecida como MIF. Quando um componente é instalado no sistema, o ficheiro MIF associado é dado a conhecer à entidade servidora, para ser adicionado à base de dados.

A entidade servidora apresenta uma API de ligação para os componentes (*Service Provider API for Components*). De forma a possibilitar bidireccionalidade, os componentes também apresentam uma interface que poderá ser utilizada pela entidade servidora. O conjunto destas duas APIs é conhecida por CI e é utilizada pelos componentes para dar acesso à informação de gestão e para, no fundo, serem geridos. A CI e a MIF isolam a complexidade inerente do sistema facilitando o trabalho de desenvolvimento dos fabricantes de equipamento.

Por outro lado, a entidade servidora apresenta interfaces para as aplicações de gestão, colectivamente denominadas *Service Provider API for Management Aplications*. O inverso também acontece, sendo colectivamente denominadas *Management Provider API*. Ambas as APIs constituem uma forma de comunicação entre a aplicação gestora e o equipamento sendo genericamente conhecidas por MI. Esta interface genérica providencia uma forma comum de acesso a informação de gestão independentemente dos mecanismos intrínsecos.

Resumidamente, a entidade servidora consiste num bloco de software em execução contínua num sistema computacional e que medeia as interacções entre a MI e a CI, desempenhando funções em benefício de ambos.

Em termos de segurança, o DMI baseia-se nos mecanismos disponibilizados pelo sistema operativo e pela pilha RPC para controlo de acesso (por exemplo, a ficheiros ou à base de dados com informação MIF) e privacidade (mecanismos de criptografia dos RPCs). Com estas duas assunções, o DMI introduz mecanismos de:

- Controlo de acesso de aplicações de gestão remotas sobre informação DMI;
- Segurança na instrumentação de componentes;
- Segurança na base de dados MIF;
- Segurança nas aplicações de gestão locais;
- Geração de eventos sobre operações relacionadas com a segurança;
- Registo (*logging*) de operações relacionadas com segurança;
- Autorização com base em papéis (*role-based authorization model*);

- Políticas de autorização configuráveis remotamente;
- Implementação de interfaces de autenticação independentes do sistema operativo.

2.4 Java Dynamic Management Kit

O JDMK (*Java Dynamic Management Kit*), proposto pela Sun Microsystems [JDMK], introduz um paradigma de gestão que, aproveitando a flexibilidade da linguagem Java, se encontra adaptado para gerir recursos (aplicações, dispositivos, serviços, políticas, ...) frequentemente voláteis, ou seja, que surgem e desaparecem à medida que são instalados, criados, activados ou eliminados.

Houve a preocupação em manter o paradigma compatível com os modelos de gestão existentes, de forma a ser possível gerir um sistema JDMK por SNMP, por exemplo, ou agentes SNMP por JDMK.

Tal como no paradigma genérico de um sistema de gestão de redes, os conceitos de aplicação de gestão, agentes e protocolo de comunicação mantêm-se inalterados, embora sejam enriquecidos com mecanismos que permitem lidar dinamicamente com as alterações que ocorrem.

Na base deste paradigma encontram-se as *Java Management Extensions* (JMX), que definem a arquitectura, os padrões de desenho, o conjunto de classes (API) e os serviços para desenvolver aplicações de gestão de redes na linguagem de programação Java [JMX].

A arquitectura é estruturada em três níveis (Figura 2.10):

- Instrumentação – Esta primeira camada especifica o método de acesso aos recursos. Estes podem ser aplicações, serviços, dispositivos ou utilizadores, entre outros.
- Agentes – Os agentes JMX monitoram e controlam directamente um ou mais recursos e respondem perante estações de gestão remotas.
- Serviços distribuídos – Esta camada especifica a forma de implementação de aplicações de gestão.

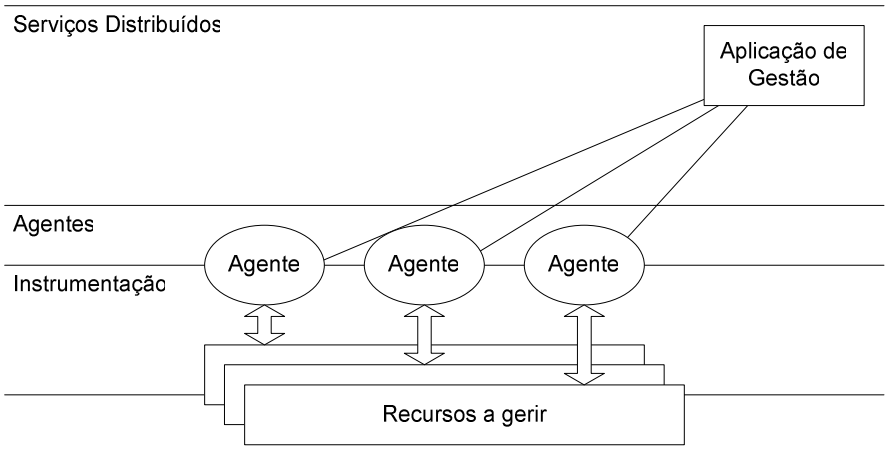


Figura 2.10 – Arquitectura JMX.

No cerne deste conceito encontram-se componentes denominados MBeans e que definem as unidades de serviços de gestão. Por outras palavras, os MBeans implementam serviços específicos de gestão, que podem ser agrupados de forma a construir agentes específicos.

Devido ao carácter dinâmico que os agentes devem apresentar, foram previstas duas formas de actualizar os serviços de gestão (Figura 2.11).

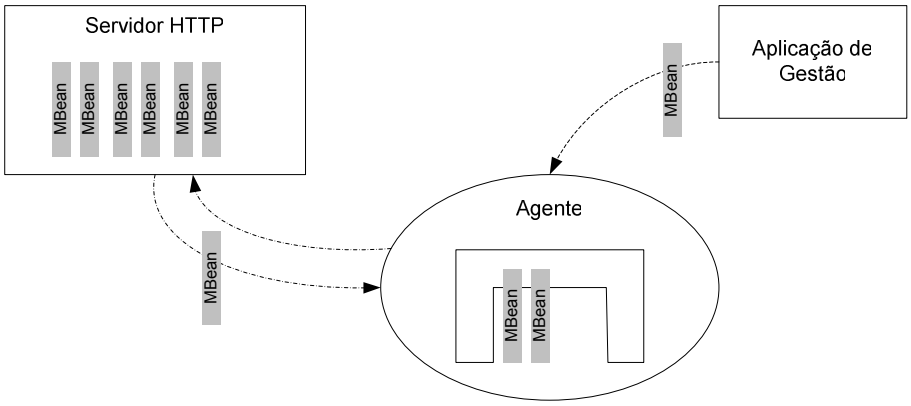


Figura 2.11 – Actualização de serviços de gestão.

Por um lado, a aplicação de gestão pode adicionar ou substituir um MBean num determinado agente enquanto que, por outro lado, o agente pode, autonomamente, requisitar novos MBeans de um repositório. Este encontra-se geralmente associado a um servidor HTTP, seguindo um paradigma semelhante à distribuição de *applets* Java. Estas abordagens resolvem alguns problemas de actualização de software e permitem optimizar recursos utilizando apenas os MBeans necessários.

Esta abordagem permite, utilizando MBeans adequados, construir hierarquias de agentes e, como tal, providenciar uma forma de distribuição de serviços de gestão.

Os MBeans possuem mecanismos derivados da própria linguagem Java, que permitem anunciar em tempo de execução quais as funcionalidades que possuem, ou seja, quais os métodos que poderão ser invocados e quais os atributos que os caracterizam. Estes são anunciados por uma classe específica (`javax.management.MBeanInfo`) ou descobertos por intermédio do mecanismo de introspecção existente na própria linguagem.

O desenvolvimento de um MBean passa, de forma semelhante ao processo utilizado para definir componentes em Java – JavaBeans, pela definição de classes de metadados (*MBean Metadata classes*). Este conjunto de classes dispõe de métodos que descrevem pormenorizadamente os seus atributos, métodos, construtores e tipos de notificações.

Esta informação permite representar graficamente o MBean e, como tal, dar a possibilidade ao utilizador de construir um determinado agente apenas com auxílio do rato. Por outro lado, torna possível aos agentes e às aplicações de gestão invocarem, em tempo de execução, serviços dos MBeans sem anteriormente os conhecerem e, desta forma, aumentar o grau de flexibilidade de todo o sistema de gestão.

O modelo de notificações é baseado no modelo de eventos da linguagem Java, ou seja, para que um determinado objecto receba notificações terá de implementar a interface `javax.management.NotificationListener` e registar-se num objecto do tipo `javax.management.NotificationBroadcaster`. Sempre que uma notificação for enviada, o

método `handleNotification` é invocado com uma referência para o objecto que encapsula a notificação.

A arquitectura JMX pode ser associada a outras tecnologias de gestão, nomeadamente o SNMP ou CMIP, usando conversores (*proxys*). Adicionalmente, a arquitectura JMX introduz o conceito de adaptadores de protocolo, objectos que quando associados aos agentes JMX dão a possibilidade de receber e interpretar mensagens com outro formato protocolar (por exemplo, SNMP), e de conectores (específicos para aplicações de gestão JMX). Estes dão a possibilidade a entidades remotas de contactarem, invocarem, instanciarem, registarem e receber notificações de MBeans.

2.5 Joint Inter-Domain Management

Algumas organizações de normalização têm feito um esforço de integração das arquitecturas e protocolos de gestão clássicos, como o SNMP e o CMIP, sob uma camada comum, nomeadamente, utilizando CORBA (*Common Object Request Broker Architecture*). Esta opção possibilita construir aplicações de gestão com recurso a invocação de funções, eliminando a complexidade adicionada por um protocolo e uma arquitectura de gestão. A tarefa de desenvolvimento de aplicações é simplificada e possibilita integrar diferentes modelos de gestão. Adicionalmente, a independência intrínseca da CORBA relativamente a tecnologias de rede e a linguagens de programação possibilita adaptar facilmente a arquitectura a novos protocolos ou a diferentes tipos de equipamento, bem como tirar partido das melhores características de cada linguagem de programação e da especificidade do hardware.

O grupo *Joint Inter-Domain Management* (JIDM) patrocinado pelo *The Open Group*, um consórcio internacional vocacionado para a integração de tecnologia e informação (http://www.opengroup.org/), e pelo *Network Management Forum* (TMN) iniciou a sua actividade para responder às necessidades de interoperabilidade entre as tecnologias e modelos de gestão. Para tal, decidiu propor uma abordagem que não coloca em causa nenhum dos modelos tradicionais.

O desenvolvimento de aplicações e de agentes de gestão assenta, essencialmente, nos seguintes pontos [JIDM]:

- definição de algoritmos de tradução entre os modelos de informação OSI ou SNMP e especificações IDL (*Interface Definition Language*),
- definição de interfaces CORBA suportando todas as interacções possíveis em CMIP e em SNMP.

Esta arquitectura permite construir adaptadores (*gateways*) entre diferentes modelos de gestão bem como desenvolver sistemas de gestão inteiramente CORBA (Figura 2.12).

Em termos práticos, uma aplicação de gestão desenvolvida em torno de um conjunto de objectos CORBA pode gerir objectos OSI ou SNMP se estes apresentarem uma interface IDL. Geralmente, os objectos de gestão encontram-se definidos em GDMO, para o caso OSI, e SMI, para o modelo Internet. Neste sentido, é necessário um mecanismo que possibilite efectuar a conversão de modelos.

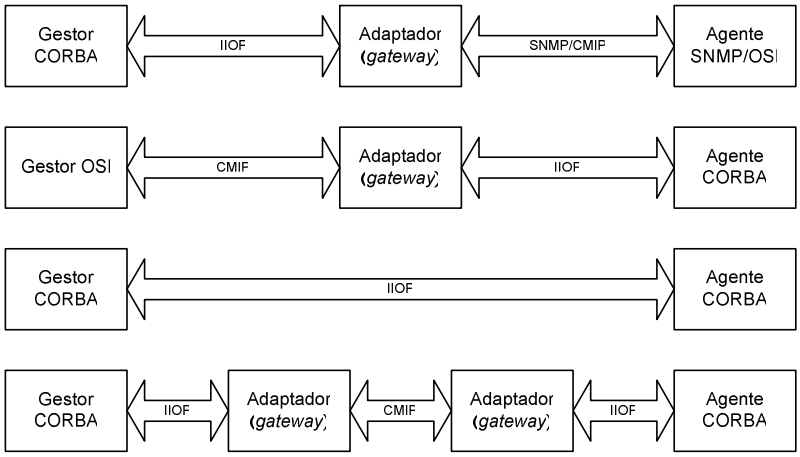


Figura 2.12 – Gestão JIDM.

A tradução do modelo de informação (*Specification Translation*) descreve algoritmos para a tradução estática dos modelos GDMO e SNMP [JIDM_ST]. O modelo de informação é transformado em interfaces IDL que posteriormente são definidas sob a forma de aplicações de gestão ou agentes CORBA (Figura 2.13).

Seguindo um exemplo para o modelo SNMP, o desenvolvimento de aplicações é efectuado com o auxílio de ferramentas proprietárias contendo muitas vezes compiladores de MIBs. Estes geram código (C/C++/Java/...) com base nos objectos de gestão definidos no ficheiro MIB, específico para determinada API SNMP.

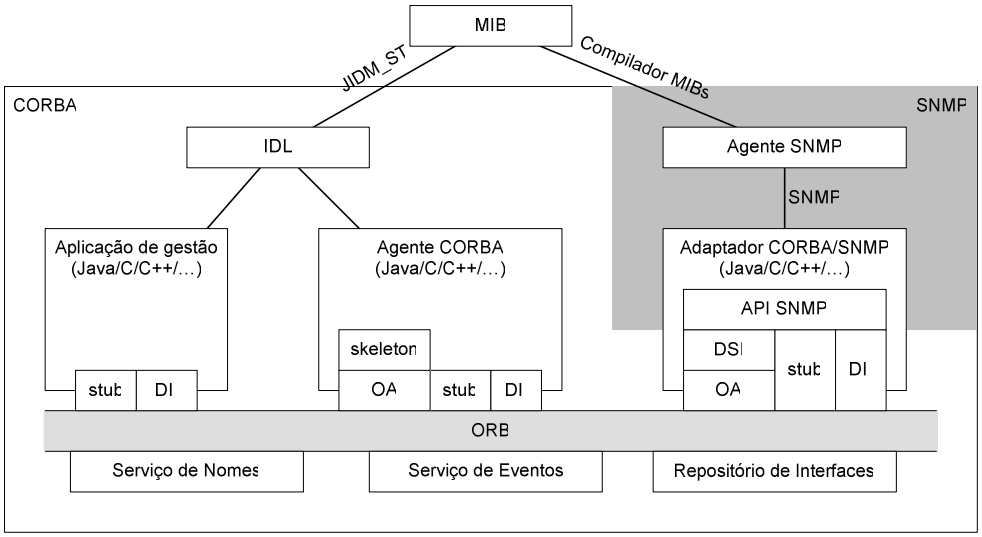


Figura 2.13 – Conversão SMI para IDL.

Por outro lado, os ficheiros MIB podem ser traduzidos para interfaces IDL. Estas interfaces permitem desenvolver agentes CORBA que efectuam a instrumentação definida nos módulos MIB originais. As mesmas interfaces são utilizadas para desenvolver os adaptadores (*stubs*) do lado do gestor. Os mecanismos de DII (*Dynamic Interface Invocation*) e DSI (*Dynamic Skeleton Interface*) permitem adicionar funcionalidade às aplicações de gestão e aos agentes, respectivamente.

A interoperabilidade requer também a conversão de mensagens de forma a tornar transparente o transporte de informação entre modelos. A conversão de protocolo (*Interaction Translation*) [JIDM_IT] descreve o processo de correspondência entre as mensagens de um determinado domínio e uma ou mais mensagens de outro domínio. Um adaptador (*gateway*) converte, por exemplo, uma mensagem CMIP em invocação de funções IDL. Para tal, terá de identificar os objectos relevantes e, sobre eles, invocar o(s) método(s) apropriados. Os resultados serão então associados e formatados de modo a construir a resposta CMIP.

Os adaptadores poderão utilizar alguns serviços CORBA, nomeadamente o serviço de nomes (*Name Service*) para obter referências, o serviço de ciclo de vida (*Lifecycle Service*) para criar instâncias de novos objectos e o serviço de eventos (*Event Service*) para as notificações.

O adaptador CMIP/CORBA terá de (Figura 2.14):

- receber pedidos `set/get/action` e efectuar a sua tradução em invocações sobre objectos IDL;
- receber eventos gerados por objectos IDL e traduzi-los em pedidos `event-report` posteriormente enviados para os sistemas registados,
- receber invocação de métodos IDL e reencaminhá-los como pedidos CMIP para um agente OSI,
- receber pedidos `event-report` e enviá-los como eventos CORBA para os objectos registados,
- receber pedidos `create/delete` e efectuar a sua tradução em invocações IDL,
- ser alvo de invocações para criar e eliminar objectos e reencaminhá-los sob a forma de pedidos CMIP do tipo `create/delete`para um determinado agente OSI.

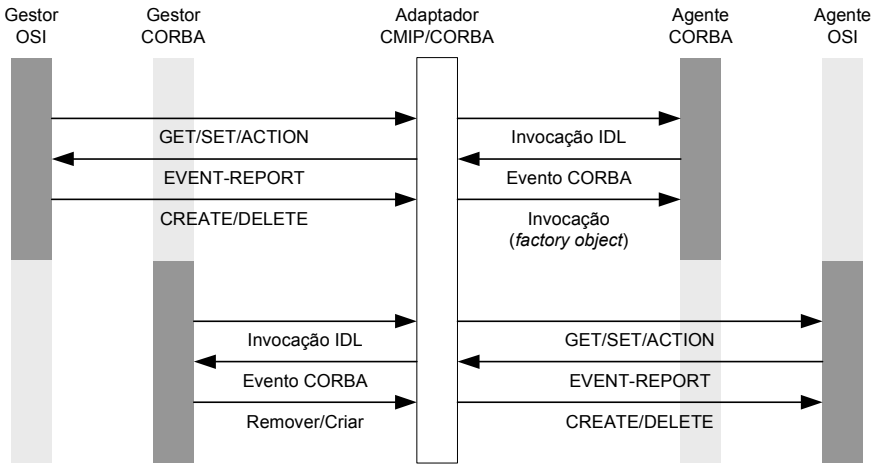


Figura 2.14 – Funções de um adaptador CORBA/CMIP.

2.6 Resumo e avaliação

Os modelos de gestão aqui apresentados revelam alguns princípios comuns. No entanto, apesar das semelhanças, as suas diferenças tornam a interoperabilidade difícil ou mesmo impossível.

As maiores diferenças residem a nível da descrição de objectos de gestão e a nível do protocolo de transporte de informação. Relativamente à primeira, encontram-se modelos sofisticados de representação de dados recorrendo a soluções orientadas ao objecto ou às mais recentes tecnologias de sistemas distribuídos, substancialmente diferentes de outros modelos que actualmente gozam de uma maior divulgação.

A nível de protocolo de transporte de informação há soluções que privilegiam a utilização de um protocolo específico, caso do SNMP, enquanto que outras fazem uso de técnicas de programação de sistemas distribuídos, como é o caso do JDMK e do JIDM (Figura 2.15).

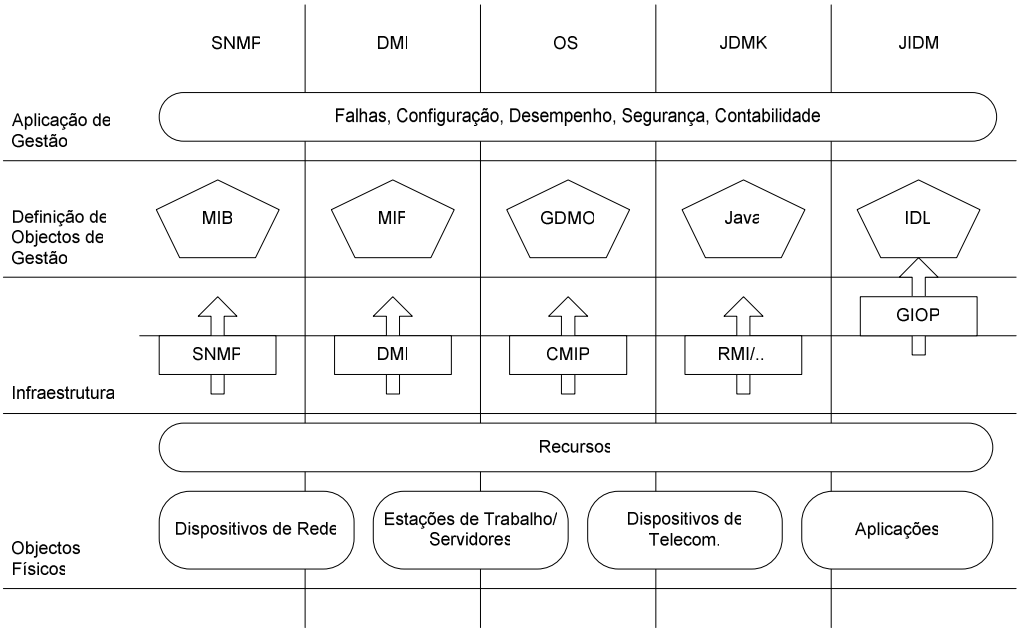


Figura 2.15 – Modelos de gestão.

Os problemas que mais se fazem sentir no SNMP relativamente às outras tecnologias é a falta de um modelo de informação orientado ao objecto e manter um baixo nível de abstracção sobre a informação de gestão. Outra crítica associada ao SNMP é a falta de "vocação" para modificação de informação, devido à ausência de mecanismos de segurança credíveis. Esta falha, apesar de ainda ser utilizada na argumentação dos seus opositores, foi eliminada com a última versão disponibilizada pelo IETF.

O modelo de gestão OSI, apesar de inicialmente mais completo que o SNMP, é bastante complexo e padece de massa crítica em termos de número de utilizadores. Por este motivo, nunca atingiu ampla divulgação no seio da gestão.

Os modelos baseados em CORBA apresentam diversos benefícios herdados desta tecnologia. Neste sentido, oferecem:

- Maior facilidade de desenvolvimento e integração de sistemas de gestão pelo facto de ser mais fácil de o fazer a nível de API do que a nível protocolar;
- Extensibilidade da arquitectura de gestão em termos de facilidade de adaptação a novas características e tecnologias de rede (ex. novos protocolos ou novo equipamento activo);

- Independência das linguagens de programação, o que permite tirar partido das melhores características de cada linguagem e da especificidade do hardware.

Por outro lado, possuem mecanismos de comunicação mais pesados tanto em termos de recursos de rede como de processamento.

A DMI apresenta um bom apoio por parte dos principais fabricantes de equipamento e de software de gestão embora não tenha ainda encontrado ampla divulgação. Falta ainda esclarecer qual o seu papel em termos futuros no seio de redes essencialmente geridas por SNMP.

A JDMK não segue qualquer norma de gestão e existe receio por parte da comunidade de gestão em adoptar soluções proprietárias, uma vez que podem comprometer o futuro da gestão da rede em causa.

2.7 Conclusões

A necessidade de gerir equipamento interligado por uma rede de comunicação resultou no aparecimento de tecnologias afins, orientadas por diferentes objectivos. O SNMP, por exemplo, surgiu com o objectivo de ser simples e rápido de implementar. O modelo de gestão OSI surgiu como um modelo bastante completo e poderoso mas excessivamente complexo para possibilitar a vulgarização de implementações. Ao longo do tempo, soluções de recurso baseadas num ou outro modelo procuraram obviar algumas limitações originais. Outras soluções procuraram romper com os modelos "clássicos" e apresentar uma abordagem completamente diferente, embora incompatível.

A sucessão e a coexistência de modelos que se tem vindo a sentir na área da gestão de redes ao longo dos últimos anos reflecte uma intensa actividade em torno da problemática da gestão de redes – controlar uma rede de dados de forma a maximizar a sua eficiência e produtividade. Cada modelo introduz um certo grau de inovação e procura eliminar as lacunas dos anteriores. No entanto, o passado demonstra que a simplicidade é um dos factores dominante na aceitação de determinada tecnologia. Outros factores são a compatibilidade com sistemas já instalados e a "abertura" das normas face a um mercado excessivamente proprietário.

You move to an area and you multiply and you multiply until every natural resource is consumed. The only way you could survive is to spread to another area.

Agent Smith, "The Matrix"

3 Gestão Distribuída

As redes de comunicação de dados são intrinsecamente heterogéneas, povoadas por equipamento de diversos fabricantes, diversas funções e diferentes sistemas operativos. A "simples" tarefa de expandir uma rede com, por exemplo, novas impressoras ou outros componentes apresenta problemas de compatibilidade que obrigam à adição de código em todos os nós que pretendam aceder aos serviços disponibilizados pelo novo equipamento.

Em termos de gestão de redes as variáveis envolvidas na instalação de um conjunto de agentes são demasiadas, pelo que procurar um sistema robusto e, simultaneamente, simples é tarefa complicada. Por exemplo, poderá ser instalado em todos os nós um agente com funções de instrumentação de recursos computacionais mas, devido à diversidade instalada, terão de ser previstas implementações diferentes (encaminhadores, sistemas Windows 2000, Windows NT, Windows 98/95, HP-UX, Solaris, Linux, etc.). Embora baseados no mesmo protocolo terão de ser adaptados a cada plataforma. O problema pode ser alargado com a introdução de mais instrumentação, nomeadamente, sondas de tráfego, instrumentação de encaminhamento, de impressão ou outras.

O cenário actual de um sistema de gestão inclui grande quantidade de elementos associados a uma ainda maior quantidade de informação de gestão. Este facto aliado ao paradigma tradicionalmente centralizado pode, sob certas condições, provocar a inacessibilidade da estação de gestão devido, por exemplo, a sobrecarga de processamento, sobrecarga de tráfego ou eventual falha. Nesta situação, todo o sistema de gestão se revela inútil, pelo que é necessário descobrir soluções que possibilitem descentralizar a carga e definir um sistema que não dependa de apenas um ponto. Nesta situação, poderá haver situações em que partes do sistema se encontram inacessíveis enquanto que outras partes ainda se mantêm em funcionamento, levando à sobrevivência global do sistema.

3.1 Tecnologia de sistemas distribuídos

Um sistema distribuído é constituído por vários componentes que, por sua vez, podem ainda ser subdivididos em partes. Os componentes são autónomos, ou seja, possuem controlo total sobre as suas partes sem no entanto possuirem controlo total sobre outros componentes.

Este facto leva à paralelização de operações eliminando problemas de:

- escalabilidade – cada componente é responsável, de forma autónoma, pela execução de determinadas operações;
- eficiência – o tempo de processamento pode ser reduzido pela distribuição da carga por vários componentes ou pela proximidade relativamente à fonte da informação;
- robustez – ao contrário dos sistemas centralizados, alguns componentes podem falhar sem afectar de forma catastrófica o funcionamento do sistema.

Para que um sistema distribuído funcione adequadamente é fundamental existir um mecanismo de comunicação entre componentes. As tecnologias actuais de sistemas distribuídos incluem mecanismos deste tipo que, adicionalmente, permitem atenuar ou mesmo eliminar as diferenças a nível de plataforma. Entre estes, encontra-se a instalação de máquinas virtuais de execução (exemplo do Java), a normalização de objectos e protocolos (invocação remota de procedimentos ou de métodos) ou a introdução de sistemas que permitem transparência a nível de localização de código (agentes móveis).

Estas tecnologias são potencialmente úteis em sistemas de gestão de redes, fornecendo mecanismos de comunicação eficientes, optimização de processamento, resolvendo problemas de escalabilidade, entre outros.

3.1.1 Sockets

Os *sockets* de Berkeley [Sechrest86] constituem um mecanismo de comunicação entre processos (IPC) independente da sua localização, ao contrário dos mecanismos de IPC tradicionais (*pipes*, mensagens, sinais, memória partilhada, etc.) os quais exigem que os processos intervenientes se localizem na mesma máquina [Stevens98]. Um *socket* designa o par endereço/porto, indicando inequivocamente a sua localização em termos de rede (endereço) e de espaço de execução (porto).

Tipicamente, quando dois processos comunicam entre si com base no mecanismo de *sockets*, um desempenha o papel de cliente e outro o de servidor. Um servidor disponibiliza um determinado serviço e tem um modo de operação predominantemente passivo, aguardando que um ou mais clientes o contactem. A distinção servidor/cliente entre um par de processos não é rígida uma vez que, em qualquer altura, podem inverter funções caso pretendam efectuar uma troca de dados em sentido inverso.

Um servidor pode ser sequencial se atende um novo pedido apenas quando tiver satisfeito o pedido actual, ao passo que um servidor concorrente gera um processo filho para atender cada cliente específico, mantendo-se o pai permanentemente à escuta de novos pedidos.

A comunicação baseada em *sockets* pode ser efectuada de duas formas:

- Será orientado à conexão se contemplar três fases: estabelecimento da conexão, transferência dos dados, terminação da conexão. Neste caso, o cliente certifica-se de que o servidor está apto a receber os seus dados e a entrega destes é confirmada (TCP).

- Não orientado à conexão quando apenas tem lugar a fase de transferência dos dados, confiando-se nos "melhores esforços" da rede para que aqueles cheguem ao seu destino (UDP).

O desenvolvimento de aplicações recorre à invocação de primitivas para definir os instantes de comunicação, como seja a reserva de recursos, o estabelecimento de ligação, a troca de informação e a terminação de comunicação.

A utilização de *sockets* requer que a aplicação efectue todo o controlo de conexão e de sessão, tarefas que podem ser complexas. Além disso, não existem primitivas para resolver problemas de representação de dados, pelo que poderá não ser possível efectuar a troca de informação com programas desenvolvidos em outras linguagens ou em execução noutras plataformas sem que haja um módulo específico para lidar com este problema.

Este mecanismo encontra-se na base de praticamente todos os sistemas de comunicações vocacionados para a Internet, incluíndo o SNMP. Este apresenta uma arquitectura baseada em servidores sequenciais (agentes) e na transferência de dados não orientada à conexão (UDP). Os agentes encontram-se, geralmente, à escuta de pedidos no porto 161.

3.1.2 RPCs

O sistema de RPCs (*Remote Procedure Calls*) é, basicamente, um mecanismo que permite a uma aplicação em execução num determinado computador, a utilização de procedimentos que são executados noutro computador, interligado fisicamente com o primeiro. A execução desse procedimento remoto processa-se como se ele fosse local, através da passagem de parâmetros e da recolha dos valores de retorno [Bloomer92].

Um sistema RPC inclui vulgarmente um conjunto de funções para definição do protocolo de comunicação, um compilador de protocolo e funções responsáveis pela gestão da comunicação.

A principal vantagem deste tipo de sistema é o facto de proporcionar o desenvolvimento de aplicações distribuídas por um conjunto de computadores sem que seja necessário considerar os aspectos respeitantes à utilização da rede e outros relacionados com a representação de dados entre sistemas.

Os RPC baseiam-se no modelo cliente/servidor, em que o programa que faz o pedido de execução do procedimento remoto tem o papel de cliente e o programa que proporciona a execução desse procedimento o papel de servidor.

Na invocação local de procedimentos o processo chamador executa o procedimento no mesmo espaço de endereçamento, ou seja no mesmo espaço de memória física (Figura 3.1).

Figura 3.1 – Invocação local de procedimentos.

O mecanismo RPC enquadra-se no modelo de computação cliente-servidor e permite que um cliente invoque procedimentos executados pelo servidor numa outra máquina. Os argumentos são fornecidos ao servidor, o qual executa o procedimento devolvendo os eventuais resultados ao cliente. Este mecanismo de execução é síncrono, uma vez que o cliente aguarda pelos resultados da execução remota do procedimento antes de prosseguir.

Sob o ponto de vista do programador, as diferenças entre a execução local e a execução remota são mínimas, uma vez que a API do mecanismo de RPCs se encarrega dos detalhes relativos à comunicação entre o cliente e o servidor. Esta é feita através de funções dedicadas, designadas por rotinas de adaptação (*stubs*) e que implementam o protocolo RPC, definindo a forma como as mensagens são construídas e trocadas (Figura 3.2).

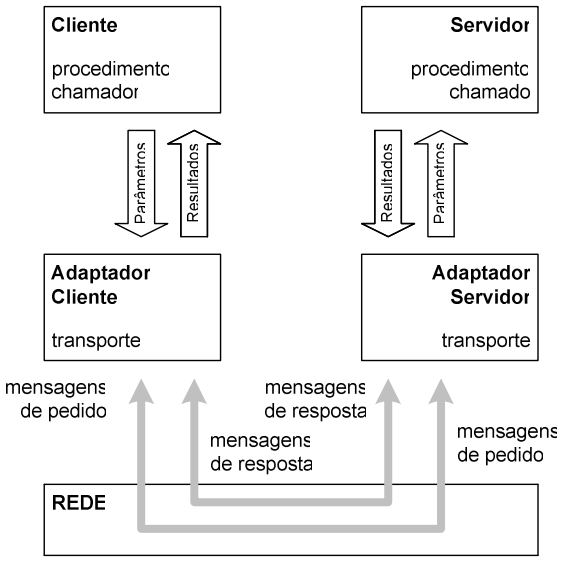


Figura 3.2 – Invocação remota de procedimentos.

A especificação da interface é efectuada numa linguagem semelhante à C, conhecida por RPCL. Com base nesta especificação, o compilador RPC gera, de forma automática, as rotinas de adaptação do cliente e do servidor.

No cliente, as rotinas de adaptação convertem as chamadas locais de um procedimento num conjunto de chamadas às funções da biblioteca RPC que se encarregam de localizar o servidor e de lhe enviar o pedido. No servidor, as rotinas de adaptação aguardam a chegada de pedidos, transformam-nos em chamadas locais ao procedimento pretendido e devolvem ao cliente os resultados produzidos. Os adaptadores utilizam filtros que convertem os tipos de dados usados em cada sistema em tipos reconhecíveis por ambos, resolvendo eventuais incompatibilidades na sua representação.

O mecanismo RPC usa serviços de transporte fornecidos pelo TCP e pelo UDP, sendo da responsabilidade do programador a escolha entre os dois.

Os servidores dão-se a conhecer a uma rede de clientes através de um serviço rudimentar de nomes. Este serviço fornece aos clientes a referência apropriada para que estes contactem directamente o servidor pretendido. No caso do mecanismo de RPCs da Sun existe um servidor de nomes designado por *portmap* que associa serviços a portos (Figura 3.3).

Quando o servidor arranca é escolhido (ou é atribuído pelo sistema operativo) um porto onde irá receber os pedidos dos clientes; esse porto é registado no servidor *portmap*, juntamente com o número e versão do programa definidos na especificação RPCL (definição da interface). Antes de invocar o procedimento remoto o cliente contacta o *portmap* da máquina hospedeira do servidor pretendido, solicitando-lhe o porto de contacto desse servidor. Por fim, o cliente

abre um canal de comunicação com o servidor através do qual envia o pedido de execução do procedimento remoto (e os respectivos parâmetros) e recebe os resultados.

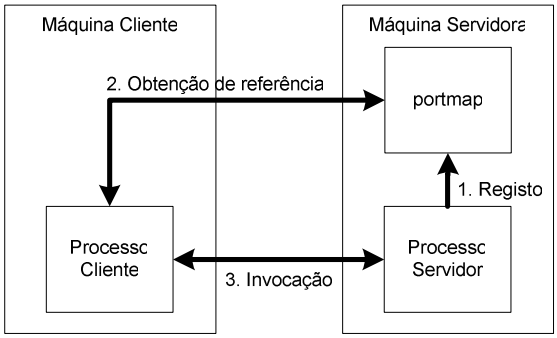


Figura 3.3 – Distribuição do cliente/servidor e do serviço de nomes.

Como foi já referido, uma invocação remota a procedimentos pode ser efectuada entre máquinas de arquitecturas diferentes. Tipo de dados como números inteiros ou reais de vírgula flutuante podem assumir diferentes representações em cada uma das máquinas. Por exemplo, algumas máquinas armazenam inteiros (`int`) com o byte menos significativo primeiro, enquanto que outras colocam de início o byte mais significativo. Problemas semelhantes ocorrem com a representação de números reais. A solução para este problema consiste em adoptar uma norma para a troca de dados.

A norma utilizada por RPCs é denominada representação externa de dados (XDR – *External Data Representation*). Esta consiste essencialmente num conjunto de funções e macros que permitem realizar a conversão entre representações específicas da máquina e a representação normalizada correspondente. O caso inverso também é contemplado. Contém primitivas para tipos de dados simples, como inteiros (`int`), reais (`float`) ou cadeias de caracteres (`strings`), bem como para estruturas de dados mais complexas, como registos (`struct`), conjuntos de elementos (`arrays`) e estruturas baseadas em ponteiros (ex: listas ligadas).

O desenvolvimento de uma aplicação baseada em RPCs ocorre em duas fases, que compreendem a especificação do protocolo de comunicação entre o cliente e o servidor (interface) e a codificação do cliente e do servidor (Figura 3.4).

O ficheiro de linguagem é compilado com o utilitário *rpcgen* para gerar ficheiros que serão posteriormente compilados conjuntamente com o restante código.

No âmbito da gestão de redes, a utilização de RPCs vem, por um lado, simplificar o desenvolvimento de aplicações de gestão devido à substituição do protocolo de comunicações pelo paradigma da invocação de métodos e, por outro lado, resolver problemas de uniformização de informação. Os RPCs são actualmente utilizados em vários sistemas de gestão, nomeadamente em DMI e em ferramentas de gestão bem conhecidas como o HP OpenView.

Apesar deste sistema efectuar o controlo de conexão, de sessão e de resolver problemas derivados da representação de dados ainda se revela insuficiente para algumas aplicações. A sua utilização em linguagens orientadas ao objecto, como o C++, apesar de possível, nem sempre segue os melhores paradigmas de programação. A sua maior limitação prende-se com o facto de não possuir um serviço de nomes flexível e que permita registar processos servidores fora da máquina local. Como consequência, os clientes terão sempre de conhecer o endereço do servidor.

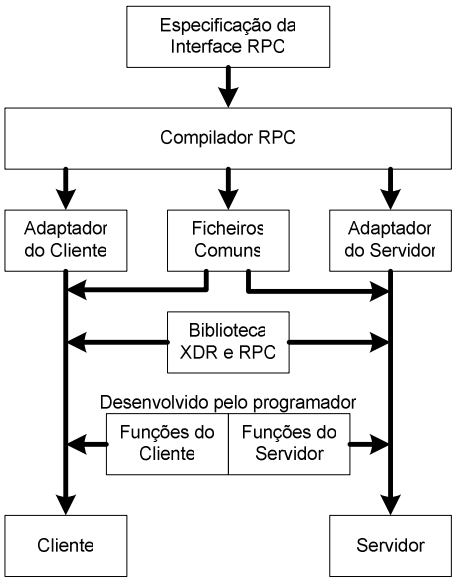


Figura 3.4 – Desenvolvimento de aplicações.

3.1.3 RMI

Uma outra instância de tecnologia de sistemas distribuídos, um pouco à semelhança da filosofia seguida nas RPCs, é o RMI (*Remote Method Invocation*), específico da linguagem Java.

A linguagem de programação Java tem vindo a modificar a forma como as aplicações são desenvolvidas e executadas na Internet. A linguagem foi desenvolvida tendo em vista o suporte de aplicações em rede, compostas por uma variedade de sistemas e arquitecturas. Para o conseguir, o compilador gera código objecto específico de uma máquina virtual, criando um ambiente homogéneo. Este código é executado em qualquer sistema que tenha instalado o respectivo interpretador. As aplicações são portáveis entre plataformas, desde que estas suportem a máquina virtual Java [Lindholm96], e podem ser acedidas por *browsers* de WWW. Esta abordagem tem, no entanto, alguns inconvenientes relacionados com a perda de desempenho associada à necessidade de interpretação.

Com a linguagem foram definidos dois tipos de programas: *applets* – programas destinados a serem armazenados num servidor HTTP e executados num *browser*, e aplicações – programas armazenados em disco e executados localmente. A diferença entre as duas abordagens consiste, essencialmente, na segurança administrada pelo ambiente de execução (*sand box*), sendo o do *browser* mais restritivo em termos de acesso a recursos locais.

Muito resumidamente, a Java é uma linguagem orientada ao objecto, apresentando características semelhantes às do C++. As principais diferenças relativamente a este reside na ausência de aritmética de ponteiros, na eliminação de algumas características como a sobrecarga de operadores (*operator overloading*) e a herança múltipla. Em adição foi criado um mecanismo de gestão dinâmica de memória (*garbage collection*) que vem simplificar o desenvolvimento.

A Java contém uma extensa biblioteca de funções de suporte aos protocolos da Internet como o TCP, UDP, HTTP ou FTP, cuja utilização simplifica o desenvolvimento de aplicações cliente/servidor. Adicionalmente, possui mecanismos de invocação remota de métodos com capacidade de serialização de objectos, o que vem tornar possível a passagem de objectos por

valor entre programas executados em máquinas distintas. Estes mecanismos associados à tecnologia de *class loading* possibilitam também transferir objectos inicialmente desconhecidos de uma das partes.

Neste contexto e à semelhança das RPCs, surge um sistema de invocação transparente de métodos em objectos remotos denominado RMI. As aplicações que fazem uso de RMI são constituídas por dois programas distintos: o servidor, com a responsabilidade de criar objectos remotos, publicar as suas referências de forma a poderem ser remotamente acedidas e aguardar invocações, e o cliente que tipicamente obtém referências para os objectos remotos e realiza as invocações.

Este tipo de aplicações necessita de, em primeiro lugar, localizar os objectos remotos. O processo pode ser realizado por intermédio de um serviço de registo específico da arquitectura RMI (`rmiregistry`) ou simplesmente comunicado pelo servidor por intermédio de passagem de parâmetros. Na posse das referências, o cliente pode passar à fase de comunicação. Todos os detalhes deste processo são encapsulados ao ponto de a comunicação entre objectos remotos ser realizada de forma em tudo semelhante a invocações locais, incluindo a passagem por valor de objectos. Este processo requer a instanciação no destino de um objecto semelhante ao da origem e a transmissão do seu estado, ou seja, do valor das variáveis membro do objecto. Precisamente devido a este último facto o RMI possui os mecanismos necessários para a disponibilização do código dos objectos trocados bem como para a transmissão do seu estado.

A invocação é, na realidade, efectuada sobre um objecto local chamado adaptador (*stub*). Este objecto consiste na representação local do objecto remoto e, para todos os efeitos, apresenta a mesma interface de acesso. É da sua responsabilidade converter a invocação em unidades protocolares que são enviadas ao objecto servidor ou, mais correctamente, a um objecto que faz a descodificação das unidades protocolares para invocação de métodos (Figura 3.5).

Este mecanismo permite reduzir a complexidade, do ponto de vista do programador, ao realizar invocações remotas restringindo-as à máquina local. A invocação não é efectuada directamente sobre o objecto servidor, mas sobre um objecto que a irá converter em mensagens protocolares. Do lado do servidor será efectuado o processo inverso, ou seja, as mensagens protocolares recebidas serão convertidas na invocação de métodos sobre o objecto servidor. A resposta seguirá no sentido inverso.

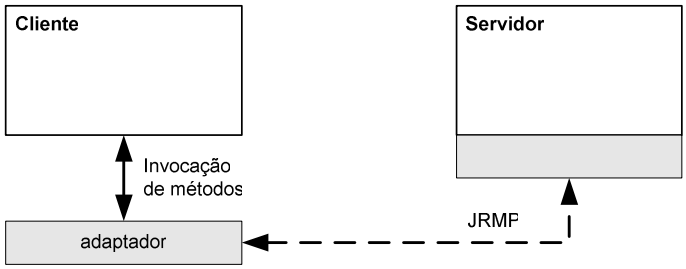


Figura 3.5 – Comunicação entre os objectos cliente e servidor.

A comunicação entre os dois objectos é efectuada por intermédio de um protocolo específico, denominado JRMP (*Java Remote Method Protocol*) ou IIOP, sendo desta forma compatível com objectos CORBA.

Há algumas questões que a figura anterior não esclarece. Como é que o cliente identifica o objecto servidor? Como é que o cliente obtém o objecto adaptador que lhe permite contactar o servidor?

O processo de registo de objectos remotos consiste na associação de um nome a uma referência de um objecto remoto (Figura 3.6).

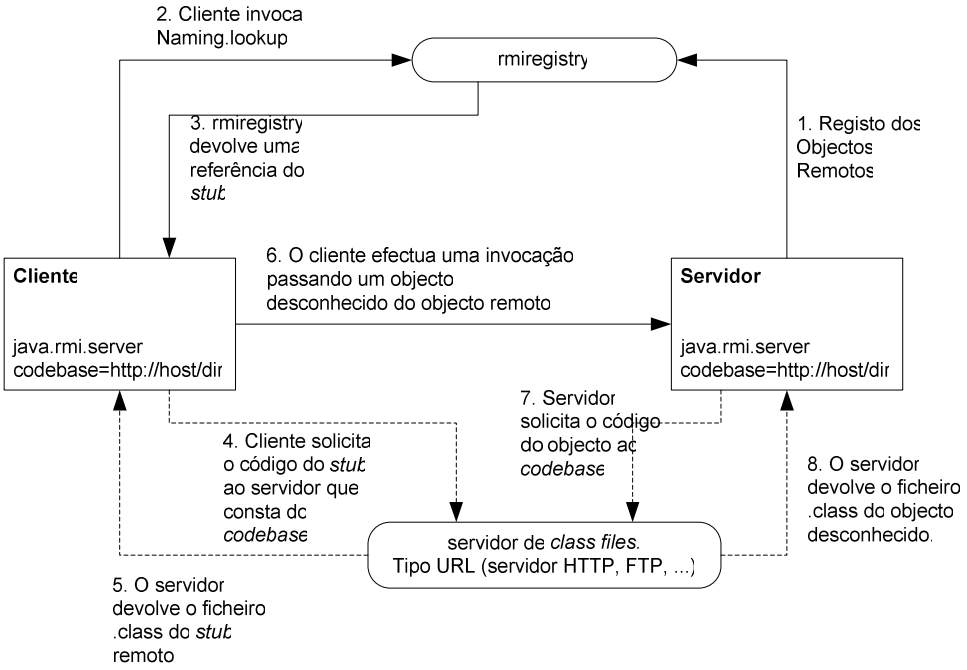


Figura 3.6 – Processo de invocação remota de métodos em RMI.

Em primeiro lugar (passo 1), após criar os objectos remotos, o servidor regista-os no serviço de nomes (`rmiregistry`). O servidor possui uma propriedade que indica ao `rmiregistry` a localização do servidor de classes (`java.rmi.server.codebase = http://host/dir`). Para que o cliente possa realizar a invocação é necessário obter uma referência do objecto destino, processo efectuado por intermédio da invocação à função de classe `Naming.lookup` (passos 2 e 3). Esta função devolve uma referência ao servidor de classes para que o cliente possa criar um objecto do tipo adaptador do objecto remoto pretendido (passos 4 e 5). Após construir o adaptador, o cliente efectua a invocação. Supondo que é passado por valor um objecto desconhecido do objecto remoto, com o auxílio da propriedade de localização do servidor de classes (`java.rmi.server.codebase = http://host/dir`) do cliente, o servidor obtém os ficheiros necessários para poder construir um objecto daquele tipo e duplicar os valores das variáveis membro do objecto. Apenas as variáveis do tipo `static` ou `transient` não são duplicadas.

As aplicações que fazem uso do mecanismo RMI são constituídas, tal como todas as aplicações Java, por interfaces e classes. As primeiras apenas definem métodos enquanto que as segundas definem a sua implementação e, eventualmente, a de outros métodos não constantes da interface. O que distingue um objecto remoto de um local é o procedimento seguido durante o desenvolvimento das interfaces e das classes.

Em primeiro lugar, é necessário definir a interface remota. Esta interface é, obrigatoriamente, derivada de `java.rmi.Remote` e declara os métodos do objecto remoto acessíveis remotamente. Todos os métodos definidos na interface remota terão de lançar a excepção `java.rmi.RemoteException`, de forma a lidar com problemas ocorridos na comunicação e com excepções lançadas pelo objecto remoto.

Um objecto que implemente esta interface torna-se um objecto remoto e será processado de forma distinta pela máquina virtual. A passagem de objectos remotos entre máquinas virtuais distintas é efectuada por referência sob a forma de um adaptador, como visto anteriormente.

Após escrever as classes correspondentes aos objectos remotos, será necessário escrever as classes dos objectos clientes. Este passo poderá ser realizado em qualquer instante após a definição das interfaces remotas, inclusivamente após os objectos servidores se encontrarem instalados e aguardando invocações.

Concluído o primeiro passo de desenvolvimento do código da aplicação, o segundo passo é a compilação do código e geração dos adaptadores. A compilação é efectuada por intermédio de um compilador Java, por exemplo o `javac` do J2SDK [J2SDK] ou o `jikes` da IBM [Jikes]. Após compilação, será necessário gerar os adaptadores por intermédio de um compilador específico – `rmic`.

Para que objectos (não remotos) possam ser passados por valor será necessário disponibilizar o seu código na rede. Este passo é necessário devido ao facto de os objectos servidores não conhecerem, em altura de compilação, os objectos que vão receber como parâmetros da invocação. Para poderem criar instâncias desses objectos, terão de aceder ao seu código por intermédio de um mecanismo de *class loading*. As classes essenciais deverão ser colocadas num servidor do tipo URL (`file://`, `http://` ou `ftp://`), compatível com o mecanismo de *class loading* standard. O endereço deste servidor será anunciado pelo servidor ou pelo cliente sob a forma de uma propriedade do tipo `java.rmi.server.codebase = http://host/dir`.

Finalmente, resta apenas iniciar a aplicação, incluindo o `rmiregistry`, o servidor e o cliente. Para definir a propriedade de localização do servidor de classes será necessário invocar o cliente ou o servidor da seguinte forma:

```
$ java –Djava.rmi.server.codebase=http://host/dir pt.ua.det.Server
```

A opção -D especifica uma propriedade de sistema para a aplicação em causa.

À semelhança das RPCs, a invocação remota de métodos é também utilizada em aplicações de gestão de redes associados à linguagem Java, nomeadamente JDMK. Além da transparência, em termos de comunicação e de representação de dados, é também conseguida flexibilidade de actualização de código sem haver necessidade de nova compilação.

3.1.4 CORBA

A *Common Object Request Broker Architecture* (CORBA) [Corba00] surge como uma resposta à falta de interoperabilidade entre diferentes produtos, sejam eles software ou hardware, tornando possível o acesso à informação de forma transparente, sem haver necessidade de conhecer a sua localização, a plataforma ou protocolo de transporte.

À semelhança do sistema RPCs, a CORBA efectua invocação remota de funções seguindo um paradigma orientado ao objecto. A arquitectura define objectos servidores, objectos clientes e um *Object Request Broker* (ORB) que estabelece as relações entre eles. O ORB intercepta a invocação e assume a responsabilidade de encontrar o objecto visado, efectuar a passagem de

parâmetros, invocar o método e devolver os resultados. Neste contexto, o objecto cliente pode, de forma transparente, invocar um método definido pelo objecto servidor, que pode residir na própria máquina ou numa máquina remota.

A CORBA assenta num procurador de objectos (ORB – *Object Request Broker*) e num conjunto de serviços que lhes estão associados. O ORB providencia a transparência relativamente à localização e linguagem em que os objectos estão implementados. Este recebe os pedidos de operação e encaminha-os para os objectos respectivos, incluindo a passagem de parâmetros e a devolução dos resultados da invocação. Os objectos clientes não necessitam de conhecer a forma como a comunicação se processa ou mesmo como os objectos remotos se encontram armazenados (Figura 3.7).

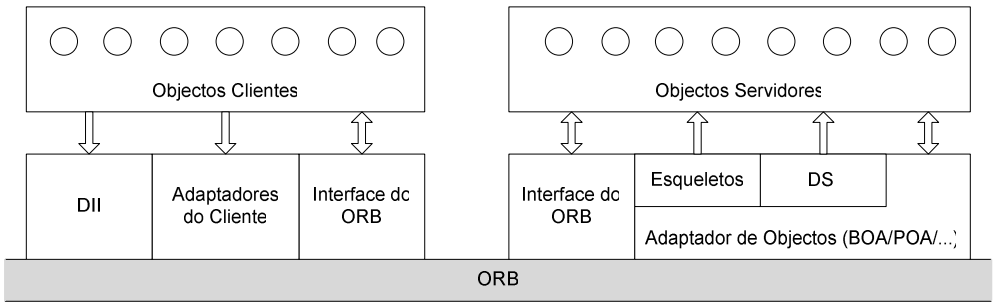


Figura 3.7 – Arquitectura CORBA.

A invocação por intermédio do ORB segue os padrões usados nas RPCs que disfarça uma invocação remota como a chamada a um procedimento local. O compilador de RPCs é usado para traduzir a especificação dos procedimentos como um par de *stubs* (cliente e servidor). A grande diferença entre as RPCs e a CORBA reside no facto de as primeiras funcionarem numa base procedimental e a CORBA definir uma base OO além de permitir a realização de invocações entre módulos desenvolvidos em diversas linguagens de forma transparente. Outra diferença de fundo é a panóplia de serviços associada à arquitectura CORBA, inexistente nas RPCs.

A troca de mensagens é um processo essencial para o desenvolvimento de uma aplicação distribuída baseada em objectos. As mensagens são trocadas sob a forma de invocações de métodos pertencentes a outros objectos e consequente recepção de resultados. Em CORBA, cada mensagem é descrita mediante a construção de uma interface que define o tipo de pedidos que o objecto está disposto a receber. Desde que o objecto se comporte como indicado pela interface não é necessário conhecer os pormenores de implementação. A interface é definida segundo uma especificação independente da linguagem, a *Interface Definition Language* (IDL).

De igual forma, é necessário um protocolo utilizado no processo de invocação de métodos. Este protocolo actua entre as interfaces, pelo que se torna necessário definir a ligação entre as interfaces e os objectos cliente e servidor. A ligação passa pela geração de correspondências entre as implementações dos objectos e o ORB: os adaptadores (*stubs*) e os esqueletos (*skeletons*). Os adaptadores constituem a terminação dos clientes, enquanto que os esqueletos realizam a correspondência entre objectos servidores e o ORB.

Em resumo e de forma simplificada, a troca de mensagens entre dois objectos por intermédio de um ORB segue o seguinte procedimento (Figura 3.8):

1. O cliente invoca um método por intermédio do adaptador.

2. O adaptador invoca o ORB.
3. O ORB passa o pedido à implementação por intermédio do esqueleto.
4. A implementação devolve o resultado ou uma excepção ao cliente por intermédio do ORB.

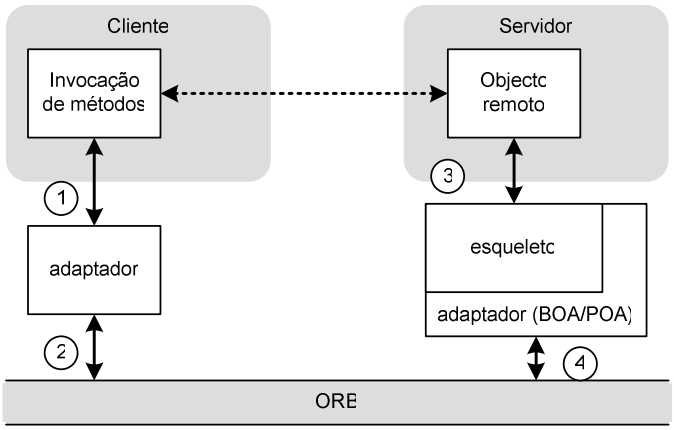


Figura 3.8 – Invocação de métodos por intermédio do ORB.

A interoperabilidade sugere que diferentes implementações possam comunicar. Para o efeito é necessário um protocolo comum. O OMG define um protocolo genérico – o *General Inter-ORB Protocol* (GIOP) – que deve ser implementado por todas os ORBs de modo a assegurar a interoperabilidade. Por outro lado, deve ser definida uma correspondência entre o GIOP e o protocolo de transporte utilizado (TCP ou IPX, por exemplo). Uma destas correspondências definida pelo OMG é o *Internet Inter-ORB Protocol* (IIOP), usado em redes TCP/IP (Figura 3.9).

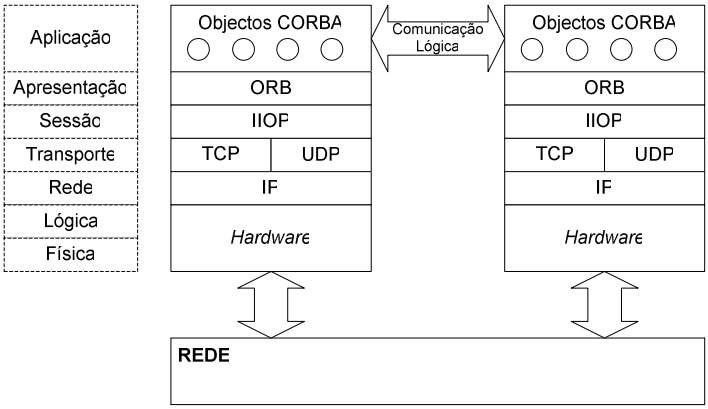


Figura 3.9 – Pilha protocolar ORB/IIOP.

A título de comparação com o modelo OSI, a CORBA está conceptualmente situada na camada de aplicação. Tal como nos RPCs, o protocolo de comunicação é definido na especificação das interfaces dos objectos, isolando os detalhes de comunicação do programador.

Estão disponíveis, tanto em domínio público como sob a forma de produtos comerciais, várias plataformas de desenvolvimento de aplicações CORBA. Independentemente da plataforma, os passos a seguir são análogos aos dos sistemas baseados em RPCs (Figura 3.10).

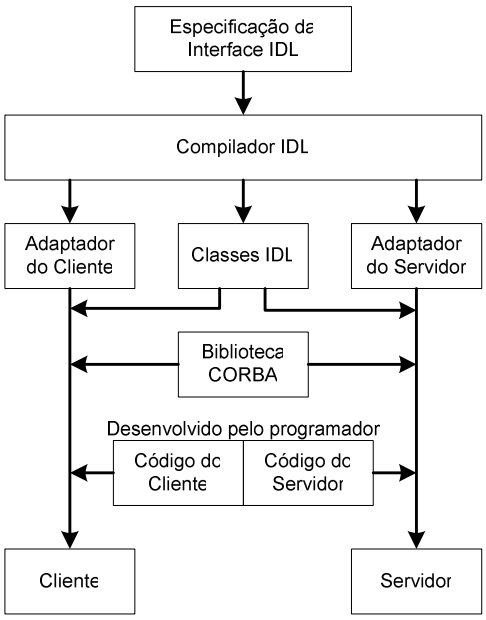


Figura 3.10 – Desenvolvimento de aplicações CORBA.

As vantagens óbvias deste tipo de arquitectura é providenciar uma camada de abstracção sobre recursos (físicos ou lógicos), linguagens de programação e tecnologia de rede. O ORB, juntamente com os serviços associados, providencia mecanismos normalizados para a invocação transparente de serviços diminuindo, portanto, o esforço de uniformização de tecnologia normalmente a cargo do utilizador.

A utilização de um ORB e de serviços normalizados vem beneficiar o desenvolvimento de sistemas e de aplicações de gestão, do qual é exemplo o JIDM.

3.1.5 Agentes Móveis

Os avanços tecnológicos modificam constantemente a forma como as empresas e organizações utilizam os serviços de informação. A passo com a evolução surgem novos paradigmas que possibilitam lidar com mais informação, diferentes tipos de equipamento, recursos frequentemente saturados, entre outros. Neste sentido, há correntes que sugerem que o paradigma dos agentes móveis podem dar resposta a muitas destas exigências.

O termo "agente móvel" é vulgarmente aplicado a programas com a possibilidade de se deslocar entre os nós de uma rede e de assumir o comportamento de agente, ou seja, de actuar de forma autónoma em representação de utilizadores ou de outras entidades [Pham98].

Os agentes móveis podem ser criados em qualquer ponto da rede, interromper temporariamente a execução, deslocar o código e o estado e retomar a execução num outro ponto da rede. Estes baseiam-se em conceitos como o de lugar (*place*), de agência (*agency* ou *agent system*) e de região (*region*). Um lugar, entre outras funções, estabelece uma abstracção (do ponto de vista do agente móvel) sobre a plataforma que o alberga. Esta função é importante, principalmente do ponto de vista de segurança, uma vez que permite restringir o acesso a recursos e serviços específicos apenas a agentes ali localizados. Uma agência cria um ambiente independente do sistema operativo, o que facilita tarefas como a migração de agentes, e alberga diversos lugares e agentes. Tipicamente, existe uma agência por computador.

Existem actualmente diversas plataformas de agentes móveis, desenvolvidas por diferentes organizações ou empresas, em várias linguagens. A plataforma Planet, por exemplo, suporta a linguagem C e Assembly e usa *strong migration* [Kato99], ou seja, o agente transporta o seu estado de execução (o conteúdo da memória e o estado da pilha são transportados para o destino, sendo o código traduzido por um compilador de forma a ser convertido para a plataforma receptora) e continua a sua execução precisamente a partir do ponto onde tinha sido interrompida. A plataforma Grasshopper utiliza a linguagem de programação Java e *weak migration* – o agente apenas mantém o estado interno das variáveis. Após a migração, a execução prossegue sempre a partir do mesmo ponto [Grasshopper]. AgentTCL, um outro exemplo, utiliza a linguagem TCL [Kotz99].

Independentemente da tecnologia que o suporta, um agente móvel é constituído por duas partes: o código, que descreve o comportamento do agente, e o estado ou seja, o valor das variáveis internas. Ambos são mantidos durante a migração, o que significa que o agente não só sabe o que fazer como sabe também o que fez.

Os agentes móveis possuem um ciclo de vida composto por estados e transições bem definidas (Figura 3.11).

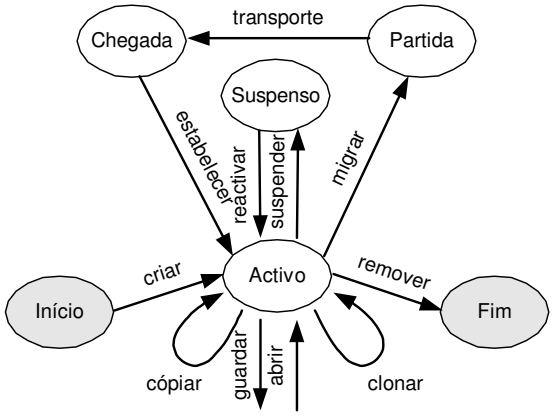


Figura 3.11 – Ciclo de vida de um agente móvel.

A gestão de agentes móveis actua sobre diversas fases do ciclo de vida:

- Criar – tarefas de inicialização tais como criar as estruturas de dados e iniciar a execução do agente. Cada agente é baseado num conjunto de classes que têm de ser enviadas à agência. Este conjunto de classes é localizado por uma cadeia de caracteres (*string*) – o *code base*.
- Remover – actividades de fim de execução.
- Suspender – interromper temporariamente a actividade do agente.
- Reactivar – reiniciar a execução do agente.
- Clonar – criar outra instância do agente no mesmo lugar.
- Copiar – criar outra instância do agente em outra localização.
- Migrar – transferir o agente. Esta acção requer o conhecimento do destino e qual o protocolo de transporte (*sockets*, SSL, RPC ou outros).

- Guardar – armazenar permanentemente a informação interna do agente. Esta informação permite reiniciar um agente que tenha sido destruído inadvertidamente por alguma falha involuntária.
- Abrir – recupera o agente móvel do dispositivo de armazenamento permanente.

Apesar de não estar directamente relacionado com o ciclo de vida do agente, um sistema de agentes móveis necessita de um mecanismo de registo – a região – de forma a tornar possível a pesquisa de agentes, lugares e agências. Este mecanismo assegura a identificação unívoca no interior do domínio definido.

Por razões de segurança, nomeadamente autenticação e controlo de acesso, uma agência identifica a autoridade que enviou o agente. É com base nesta que recursos como o sistema operativo, discos, processador, memória ou outros são protegidos contra actos ilícitos de agentes maliciosos.

O desenvolvimento de agentes móveis em ambientes orientados ao objecto, como em plataformas baseadas na linguagem Java, efectua-se geralmente por especialização (derivação) de determinadas classes que implementam mecanismos básicos do ciclo de vida do agente. De uma forma geral, estas classes disponibilizam um método que ao ser invocado despoleta a migração. A classe base encapsula os procedimentos de comunicação com a plataforma de agentes, permitindo, por intermédio de invocação de métodos, dar início a operações como migrar e comunicar com outros agentes.

As plataformas desenvolvidas por determinado fabricante apresentam um conjunto de métodos próprios, que diferem dos métodos equivalentes em plataformas de outros fabricantes. Associado a este facto, o ambiente de execução de agentes móveis é fechado em termos de interoperabilidade. Por este motivo, a gestão uniforme de diferentes plataformas não é viável, pelo que cada fabricante apresenta uma ferramenta de gestão própria.

A interoperabilidade entre plataformas de agentes assenta em dois campos. Um deles, em desenvolvimento pela *Foundation for Intelligent Physical Agents* [FIPA], visa normalizar a comunicação entre agentes bem como a linguagem utilizada. Além da tecnologia de comunicação há também a preocupação de definir protocolos de negociação em áreas específicas, tais como multimédia, viagens, serviços de rede, produção, entre outros. O segundo campo assegura a interoperabilidade entre ambientes de execução e encontra-se em desenvolvimento pelo OMG sob a forma de um documento formal – *Mobile Agents Facilities* [MAF]. Esta especificação tem como objectivo permitir que um agente móvel possa deslocar-se entre agências com perfil semelhante (partilhando a mesma linguagem, tipo de agência, tipo de autenticação e método de seriação) através de um conjunto de interfaces CORBA.

A especificação MAF consiste no primeiro documento de normalização de operações em sistemas de agentes móveis visando, assim, a interoperabilidade entre sistemas de diversos construtores. Define um conjunto de conceitos e interfaces (em IDL – *Interface Definition Language*), procurando a simplicidade de forma a facilitar o desenvolvimento futuro de sistemas de agentes móveis. As duas interfaces constituem a base para todas as operações sobre a agência e a região (Figura 3.12).

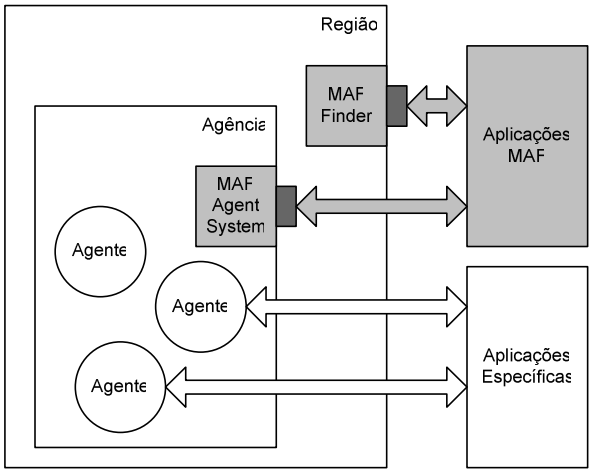


Figura 3.12 – Arquitectura MAF.

A interface `MAFFinder`, ligada à região, consiste num ponto de acesso ao serviço de directoria para agências, lugares e agentes. Os seus métodos permitem funções de pesquisa e catalogação:

Interface MAFFinder

Funcionalidade	Métodos

Registo	`void register_agent(…)`
	`void register_agent_system(…)`
	`void register_place(…)`
	`void unregister_agent(…)`
	`void unregister_agent_system(…)`
	`void unregister_place(…)`

Pesquisa	`Locations lookup_agent(…)`
	`Locations lookup_agent_system(…)`
	`Locations lookup_place(…)`

A interface `MAFAgentSystem` define métodos e objectos que suportam tarefas como consultar o nome de uma agência ou receber um agente, entre outras:

Interface MAFAgentSystem

Funcionalidade	Métodos

Ciclo de vida	`Name create_agent(…)`
	`void receive_agent(…)`
	`void resume_agent(…)`
	`void suspend_agent(…)`
	`void terminate_agent(…)`
	`OctetStrings fetch_class(…)`
	`void terminate_agent_system(…)`

Informação	`Location find_nearby_agent_system_of_profile(…)`
	`AgentStatus get_agent_status(…)`
	`AgentSystemInfo get_agent_system_info(…)`
	`AuthInfo get_authinfo(…)`
	`MAFFinder get_MAFFinder(…)`
	`NameList list_all_agents()`
	`NameList list_all_agents_of_authority(…)`
	`Locations list_all_places()`

Associadas a estas interfaces encontram-se definidas várias estruturas de dados:

- `Name` – com três atributos (`authority`, `identity` e `agent_system_type`) que asseguram a individualidade do agente.

- `ClassName` – define a sintaxe para nomes das classes que, quando instanciadas, dão origem ao agente. Esta informação é utilizada nos métodos `create_agent()` e `receive_agent()`, por exemplo.
- `Location` – consiste numa sequência de caracteres contendo: a) um URI (*Universal Resource Identifier*) com um identificador CORBA [RFC1630] ou b) um URL (*Universal Resource Locator*) contendo um endereço da Internet [RFC1738]. A vantagem do identificador CORBA é a independência do protocolo enquanto que, por outro lado, os URLs são mais adequados à Internet.

Do ponto de vista de gestão do ciclo de vida de agentes, estas interfaces permitem que uma única aplicação possa monitorar e controlar plataformas distintas desde que estas as implementem. Introduz, no entanto, a desvantagem de ignorar as funcionalidades específicas de cada plataforma [Lopes00b].

3.2 DISMAN

O grupo DISMAN (*DIStributed MANagement charter*), do IETF, foi criado com o objectivo de definir objectos de gestão, sob a forma de vários módulos MIB, que permitam distribuir tarefas de gestão por um conjunto de pontos específicos ao longo da rede. Essencialmente, os objectos DISMAN possibilitam controlar e monitorar tarefas tipicamente associadas à estação central. Por outras palavras, tornam possível "gerir a gestão" de equipamento de rede.

Este conceito, simultaneamente simples e poderoso, permite distribuir tarefas específicas de gestão por um conjunto de aplicações SNMP, de forma a melhorar a escalabilidade e permitir o funcionamento do sistema mesmo em situações de interrupção de conectividade.

Um agente SNMP que implemente os objectos de gestão DISMAN apresenta um duplo papel. De um lado, utilizando a nomenclatura definida para as aplicações SNMPv3, terá a responsabilidade de responder a comandos (*Command Responder*) e emitir notificações (*Notification Originator*), pelo que apresenta a funcionalidade de agente. Por outro lado, terá a capacidade para emitir comandos (*Command Generator*) e receber notificações (*Notification Receiver*), funcionando como gestor. A informação consultada no lado de agente é obtida com base no resultado das operações efectuadas pelo lado de gestor.

Esta aplicação é identificada, no contexto DISMAN, como um gestor distribuído (DM – *Distributed Manager*). Em termos práticos, um DM é uma entidade SNMP que recebe pedidos de outros gestores e efectua operações de gestão em agentes ou outros gestores. Este modelo de distribuição permite criar "ilhas" hierárquicas, aumentando a robustez e a tolerância a falhas do sistema (Figura 3.13). Mesmo em situações de indisponibilidade da estação de gestão central é possível que o DM lide localmente com situações críticas.

Para melhor compreender o modelo DISMAN é necessário estar ciente da funcionalidade básica de uma estação de gestão. Para esta aplicação, o modelo SNMP prevê a capacidade de gerar comandos e receber notificações. Estes mecanismos permitem que a estação de gestão obtenha e modifique informação de funcionamento da rede quer por iniciativa própria, através de técnicas de convite/resposta, quer assincronamente, através da recepção de notificações.

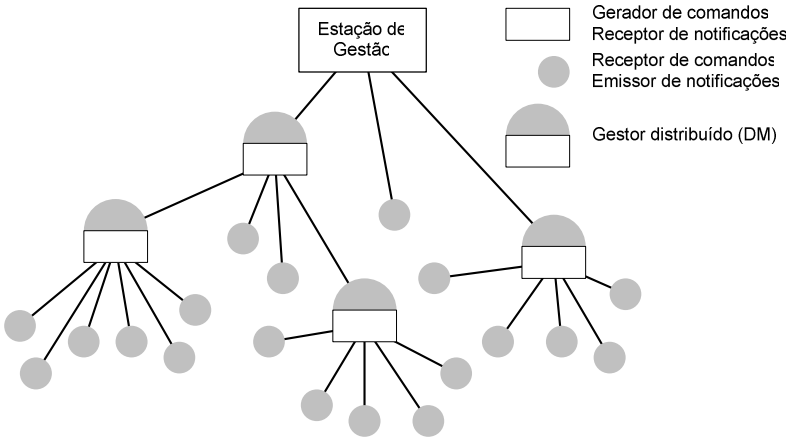


Figura 3.13 – Arquitectura DISMAN.

A funcionalidade da aplicação é complementada com ferramentas ou módulos de processamento e manipulação de informação, podendo, na sua forma mais simples, resumir-se à simples apresentação de informação (*MIB Browser*). Em termos práticos, é importante dotar a aplicação de um conjunto mais extenso de funcionalidades, como a possibilidade de analisar condições, realizar cálculos matemáticos sobre a informação recolhida, calendarizar e executar operações de gestão, armazenar notificações e processar alarmes (Figura 3.14).

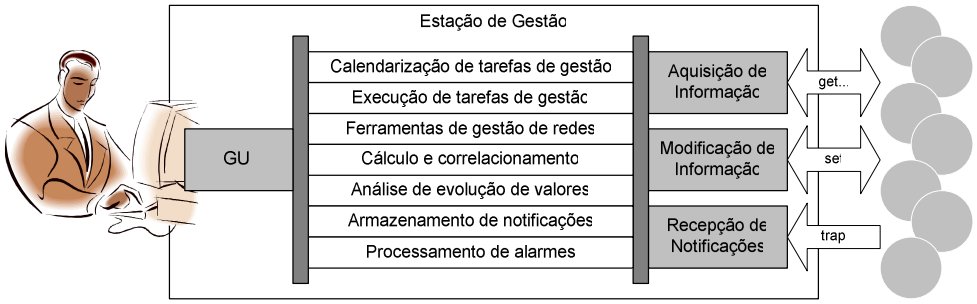


Figura 3.14 – Funcionalidade de uma estação de gestão.

O utilizador interage com um sistema deste tipo geralmente por intermédio de uma interface gráfica (GUI). Substituindo o modo de acesso e mantendo o restante conjunto de funções, é possível criar um "gestor intermédio" que, embora não dispondo de GUI, pode ser monitorado e controlado remotamente por SNMP (Figura 3.15).

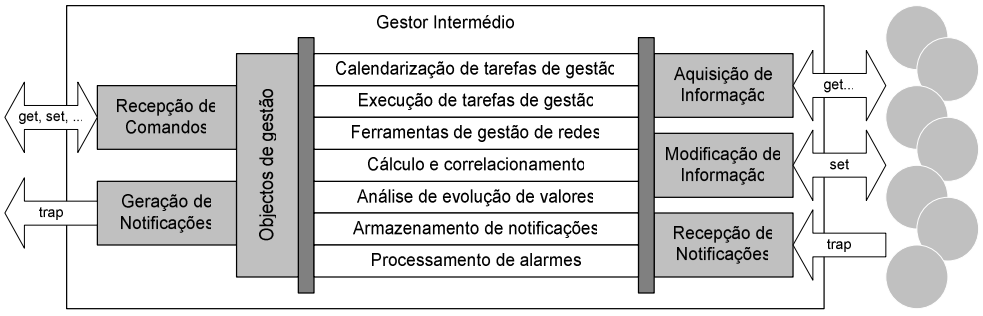


Figura 3.15 – Funcionalidade de gestor intermédio.

Para o efeito, é necessário dispor de um conjunto de objectos de gestão que permitam relacionar os comandos recebidos com a funcionalidade de gestão pretendida. É precisamente sobre esta problemática que se debruça o grupo DISMAN.

Um dos serviços indispensáveis em aplicações de gestão é a possibilidade de despoletar tarefas periódicas ou rotineiras de forma automática, por exemplo, analisar o espaço livre em disco ou realizar testes de conectividade sobre um determinado troço de rede. Um módulo de calendarização, suficientemente genérico para poder ser utilizado em diversas situações, constitui um dos serviços definidos pelo grupo DISMAN – Schedule MIB [RFC3231].

As tarefas apresentadas acima são apenas algumas das muitas que podem ser definidas pelo administrador de rede. O serviço de execução deve ser suficientemente flexível de forma a suportar a diversidade de tarefas, se possível sob a forma de um interpretador de sequências de comandos básicos – Script MIB [RFC3165].

Certas ferramentas de gestão têm a importância suficiente para serem consideradas como serviços autónomos. É o caso do teste de conectividade (`ping`), tradução de nomes para endereços e vice-versa (`lookup`) e levantamento de caminhos (`traceroute`). No contexto DISMAN, cada uma destas operações constitui um serviço distinto acedido, portanto, por MIBs distintas [RFC2925].

Neste contexto, existem também serviços vocacionados para a consulta de informação. Se, por exemplo, o espaço disponível em disco for inferior a 1/10 do espaço total poderá ser motivo para notificar o administrador. Tal como no caso anterior, este tipo de operação apresenta dois níveis de responsabilidade. Por um lado, há a necessidade de realizar o cálculo:

$$\text{resultado} = \text{livre} \leq \frac{\text{total}}{10}$$

e, por outro, gerar notificações se a expressão anterior se revelar verdadeira. De forma a manter os dois módulos genéricos, o grupo DISMAN define-os como serviços independentes, baseados na Expression MIB [RFC2982] e Event MIB [RFC2981].

Na gestão SNMP tradicional, os problemas são detectados recorrendo a técnicas de convite/resposta, geração de notificações ou uma combinação das duas. No entanto, estas abordagens introduzem alguns problemas, nomeadamente a falta de escalabilidade, quando é necessário monitorar um grande número de variáveis e o de aumento de complexidade no correlacionamento de informação proveniente de várias fontes, quer sob a forma de convite/resposta quer sob a forma de notificações. Por outras palavras, mesmo após uma análise concreta da informação recebida da rede, nem sempre se consegue obter uma visão global da falha ocorrida.

Para lidar com perdas de notificações existe um serviço de armazenamento temporário baseado na Notification MIB [RFC3014]. Como complemento, e porque nem todas as notificações correspondem a condições de falha (alarme), o grupo encontra-se a definir um conjunto de objectos para reconhecimento e processamento de alarmes [Chisholm02].

No momento da escrita deste trabalho, o grupo DISMAN encontra-se bastante dinâmico, pelo que as especificações ainda não se encontram estabilizadas, assim como o número de documentos. A abordagem das secções seguintes é feita neste pressuposto e usando os documentos mais recentes.

3.2.1 Calendarização de tarefas

Por vezes é útil executar operações periódicas ou em determinados instantes. Na sua essência, a Schedule MIB apresenta um conjunto de objectos que possibilitam calendarizar operações de escrita SNMP (`set`) sobre objectos de gestão locais do tipo `INTEGER` [RFC3231].

A Schedule MIB contém um repositório onde armazena os instantes definidos pelo utilizador, o objecto sobre o qual vai actuar e o valor associado. Quando activo, observa continuamente o relógio, que terá de manter informação de hora e de data, e caso corresponda a alguma entrada definida despoleta uma operação `set` (Figura 3.16).

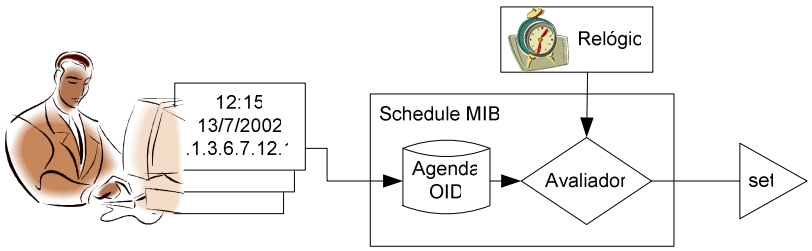


Figura 3.16 – Diagrama conceptual da Schedule MIB.

A operação é efectuada apenas sobre a máquina local, não sendo possível associar um endereço de forma a modificar valores num agente remoto.

Os instantes armazenados podem ser de três tipos: periódicos, de calendário e únicos. Para instantes periódicos, a operação é repetida com uma determinada frequência, definida por um inteiro que representa o número de segundos entre acções. Os instantes de calendário são repetidos sempre que ocorre a hora e a data definida na agenda. Os instantes únicos são semelhantes aos de calendário mas acontecem apenas na primeira ocasião possível sendo, de seguida desactivados.

3.2.2 *Scripts* de gestão

A Script MIB define um ambiente de execução de operações de gestão (*scripts*) que permitem construir operações mais complexas [RFC3165]. Os comandos encontram-se definidos numa linguagem de programação que pode ser interpretada, como o TCL, ou código nativo, desde que suportado pela implementação.

Uma implementação deste módulo MIB encontra-se constituído por duas secções: a entidade SNMP que implementa a MIB e o sistema de execução (*runtime system*), que executa os *scripts*. O sistema de execução pode ser considerado uma aplicação SNMP tal como definido na arquitectura SNMPv3, sendo este o recurso gerido pela entidade SNMP.

O envio de *scripts* para a Script MIB pode seguir dois mecanismos. Pelo *pull-model* a estação de gestão indica a sua localização, comunicando um URL ao agente, sendo este o responsável por recuperar o código. De acordo com o *push-model* o código é enviado para o agente por intermédio de comandos `set`.

A execução de *scripts* pode ser monitorada e controlada remotamente pela aplicação de gestão através de operações `get` e `set`. É também possível especificar diversos parâmetros tais como os argumentos a fornecer, o tempo máximo autorizado de execução ou número máximo de instâncias concorrentes.

3.2.3 Operações remotas

A Remote Operations MIB permite executar remotamente algumas operações essenciais para a gestão de redes, como o *ping*, *traceroute* e *lookup* [RFC2925]. Define uma forma normalizada de realizar os testes e gerar notificações de acordo com os resultados das operações remotas.

Estas operações permitem ter uma visão do estado da rede directamente no ponto remoto. Por exemplo, a operação de *lookup* permite resolver nomes para endereços e vice-versa directamente na máquina que aloja o agente e assim fornecer uma ideia do funcionamento desta operação no local. O mesmo se passa para as outras operações.

Para cada operação encontra-se definido um módulo MIB: Ping MIB, Traceroute MIB e Lookup MIB.

Todas as MIB contêm uma tabela de controlo onde se colocam os parâmetros da operação e se inicia a sua execução. Os resultados de cada operação podem ser posteriormente consultados por SNMP.

3.2.4 Eventos e notificações

A Event MIB é a sucessora da Manager-to-Manager MIB do SNMPv2 [RFC1451] e permite monitorar objectos MIB quer locais quer remotos [RFC2981]. Permite também iniciar acções de acordo com políticas predefinidas.

O utilizador define um conjunto de condições que coloca no repositório adequado. As condições são associadas a políticas que especificam em que situações é que as acções devem ser despoletadas. Por último, as acções são enumeradas pelo utilizador e armazenadas (Figura 3.17).

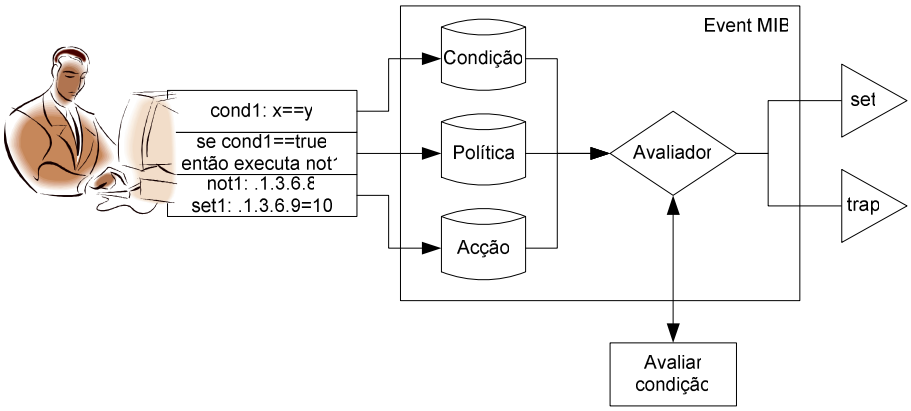


Figura 3.17 – Diagrama conceptual da Event MIB.

As condições podem ser definidas com base em operações do tipo:

- existe – verdadeira quando o OID indicado existe, quando muda o valor associado ou deixa de existir;
- booleana – compara (diferente, igual, menor, menor ou igual, maior, maior ou igual) um determinado escalar com o valor associado a um OID;
- de limiar – verdadeira quando o valor associado a um OID atravessa um determinado limiar no sentido ascendente, descendente ou ambos.

O valor é obtido por consulta e, para o efeito, é necessário indicar o OID associado, o endereço e o porto do agente. Estes parâmetros são indicados num módulo MIB independente – Target MIB [RFC2573] – mas do qual depende o funcionamento da Event MIB.

A política definida pelo utilizador indica o tipo de acção a despoletar. Esta pode corresponder ao envio de um comando `set`, à geração de uma notificação ou ambos.

Por fim, a acção pode ser uma operação `set` ou uma notificação (`trap`). No primeiro caso é necessário indicar o OID, valor, endereço e porto do destinatário. As notificações necessitam apenas do OID associado.

Como exemplo de operação, esta MIB permite associar condições a acções do tipo:

```
se (existe <um objecto>) {
  enviar notificação x;
  realizar set sobre <outro objecto>;
}
```

ou

```
se (o valor de <algum objecto> cresce além de <determinado limiar>) {
  enviar notificação y;
}
```

3.2.5 Definição de expressões

A Expression MIB foi criada com o propósito de tornar possível a definição de objectos que não foram inicialmente considerados durante a definição de outros módulos MIB [RFC2982]. Por outras palavras, permite especificar expressões baseadas em objectos de gestão existentes. Permite também encadear expressões, ou seja, definir expressões cujos parâmetros dependam do resultado de outras expressões.

Uma expressão é composta por operadores, funções e valores. Os valores podem ser constantes ou variáveis estando, neste último caso, associados a OIDs específicos. A expressão é definida por uma cadeia de caracteres onde se especificam todos os seus componentes. As variáveis são declaradas por um número precedido do símbolo '$' (por exemplo '$12'), número este que indica o índice para o OID associado (Figura 3.18).

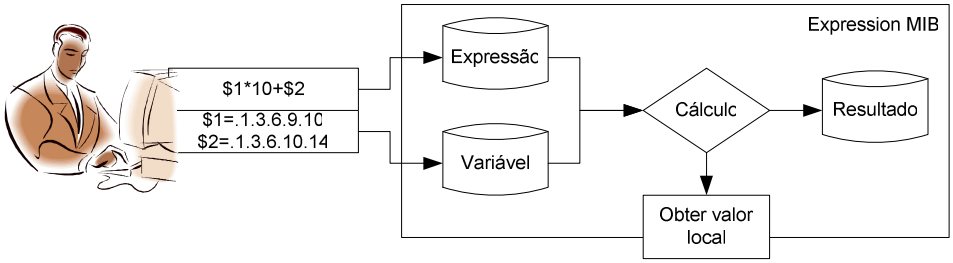


Figura 3.18 – Diagrama conceptual da Expression MIB.

A obtenção de variáveis está limitada ao agente local, não sendo possível obter valores em agentes remotos. Este facto torna difícil definir expressões que relacionem informação proveniente de vários agentes, o que poderá em certas situações ser bastante limitativo.

Uma expressão pode ser avaliada como uma taxa ou um valor lógico e, consequentemente, disparar um evento, resultando numa notificação SNMP. Sem a Expression MIB este tipo de

operações encontra-se limitado aos objectos existentes nas MIBs pré-definidas. Em conjunto com a Event MIB definem um mecanismo poderoso de processamento de informação de gestão.

3.2.6 Salvaguarda de notificações

A Notification Log MIB define um mecanismo de salvaguarda de notificações (*logging*) de forma a poder recuperar eventos perdidos [RFC3014]. Embora mais adequada a produtores de notificações também pode ser implementada pelos consumidores.

Este serviço encontra-se estruturado em três secções. A secção de configuração permite definir o tipo e quantidade de notificações armazenadas. A secção de estatística fornece indicações acerca da actividade exercida em termos de número de registos bem como de eliminação de notificações. Finalmente, a secção de registo (*log*) armazena cópias de notificações.

3.2.7 Alarmes

A descoberta de problemas segundo o modelo SNMP é baseada em operações de convite e consequente recepção de resposta (*pooling*) complementadas pela recepção de notificações. Esta abordagem levanta alguns problemas de escalabilidade relacionadas com o número de variáveis envolvidas, não se revelando prático consultar um grande número de variáveis num conjunto alargado de agentes. Podem também surgir dificuldades no relacionamento dos valores recebidos com o problema detectado, ou seja, poderá existir a noção de que um determinado problema ocorreu, mas não se saber com precisão o que aconteceu.

Um alarme, por definição, consiste na indicação persistente de uma falha, ou seja, tem a capacidade de assinalar a ocorrência de situações extraordinárias e o tipo de ocorrência. Os alarmes podem alternar de estado, nomeadamente, entre activo e inactivo, mediante as condições de falha. Cada transição de estado poderá resultar numa notificação, de forma a assinalar a ocorrência de um problema ou a sua eliminação.

A Alarm MIB especifica um método uniforme para a definição e armazenamento de alarmes [Chisholm02]. Permite relacionar alarmes e distinguir se uma determinada notificação transporta apenas informação sobre um determinado evento ou se corresponde a uma falha. Neste último caso a falha é perfeitamente identificada.

De acordo com este modelo os agentes e as aplicações de gestão encontram uma forma de entendimento sobre que tipo de problemas devem ser notificados, como serão notificados e o que poderá acontecer enquanto o problema perdurar.

As tabelas que constituem este módulo MIB mantêm uma descrição dos alarmes e listas de alarmes. Além disso, definem todos os estados que cada alarme pode assumir bem como o OID associado à notificação que é enviada sempre que ocorrer uma transição de estado.

É também possível adicionar extensões à Alarm MIB como, por exemplo, alarmes específicos ITU definidos no mesmo documento, ou outros.

Esta MIB relaciona-se com outras de forma a poder processar e identificar notificações, bem como lidar com eventuais perdas (Figura 3.19).

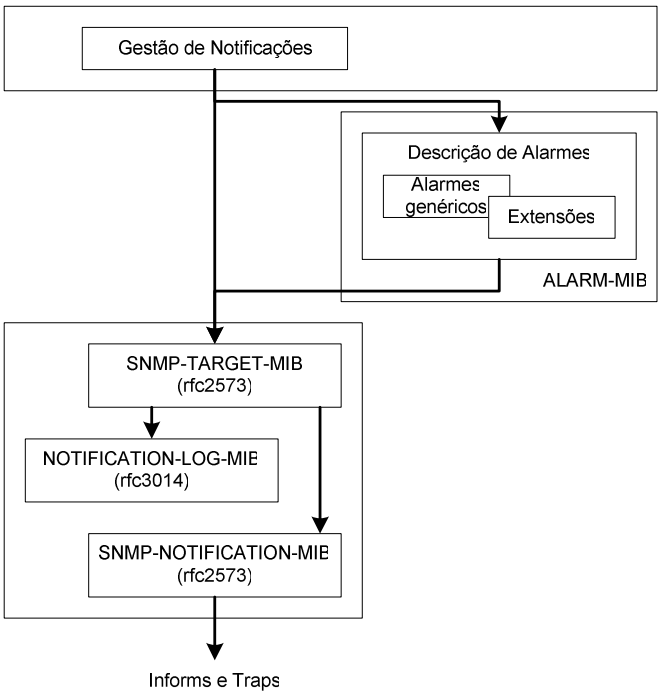


Figura 3.19 – Arquitectura da gestão de alarmes DISMAN.

As instâncias desta MIB podem ser instaladas directamente no lado do agente ou no DM, podendo hierarquizar a ocorrência de alarmes.

3.3 Conclusões

A gestão de redes, apesar de intrinsecamente distribuída, apresenta um modelo centralizado baseado numa estação sobre a qual recai a responsabilidade de aquisição e processamento de informação e num conjunto de elementos denominados agentes que desempenham o papel de interface entre as particularidades dos componentes de rede e a estação de gestão.

A comunicação entre os elementos de um sistema de gestão de redes é efectuada recorrendo a mecanismos de comunicação entre processos. O SNMP, protocolo definido para o modelo de gestão Internet, baseia-se em *sockets*, sendo estes responsáveis pelo encapsulamento dos detalhes elementares da comunicação.

Apesar de resolver os problemas básicos de comunicação, os *sockets* não abrangem outras dificuldades, como a uniformização da representação de dados. Estas são resolvidas por intermédio de outras técnicas como as RPCs ou sua congénere para a linguagem Java – RMIs. A sua utilização em modelos SNMP é normativamente inadequada, o que não acontece em outros sistemas de gestão de redes como o DMI e o JDMK, que exploram as capacidades da tecnologia.

A CORBA alarga e aplica o conceito RPCs a aplicações desenvolvidas em diferentes linguagens. Este facto torna possível a passagem transparente de mensagens entre aplicações independentemente da linguagem base. As suas características fazem com que seja um meio adequado para providenciar interoperabilidade entre diferentes modelos de gestão, como o proposto pelo JIDM.

Mais recentemente, foi introduzido o paradigma de agentes móveis. Esta tecnologia permite associar os dados ao código e enviá-los para outras máquinas, pelo que a aplicação pode ser iniciada em qualquer ponto da rede, interromper a sua execução e continuar a sua execução noutro ponto da rede.

Os sistemas centralizados podem beneficiar deste tipo de tecnologia como meio para aumentar a sua robustez, flexibilidade e eficiência. Complementarmente, no âmbito normativo da gestão Internet, o modelo DISMAN apresenta soluções de distribuição de tarefas por um conjunto de gestores intermédios (ou gestores distribuídos), criando uma hierarquia de gestão que permite dividir a responsabilidade de aquisição e processamento de informação.

Ora hei de vos iluminar um pouco dizendo que a palavra brasas está aqui por mil escudos, que esta era a moeda de Portugal, a qual moeda foi logo depoismente trocada por o euro, que era da União, e daqui podereis ver como se formou o nome da nossa moeda de hoje, que é o deseuro, como haveis de saber.

João Aguiar, "Diálogo das Compensadas"

4 Arquitectura de Desenvolvimento de Agentes Multiprotocolo

As redes actuais são constituídas por uma grande diversidade de produtos e de tecnologia. É frequente encontrar na literatura e na comunicação social referências ao aparecimento de novos paradigmas de comunicação assim como à adopção de nova tecnologia pelos utilizadores.

Os fabricantes procuram responder às solicitações do mercado desenvolvendo produtos preparados para integrar este tipo de redes, como telemóveis, PDAs, estações de trabalho, consolas de jogos, frigoríficos, máquinas de lavar, relógios e potencialmente qualquer ferramenta de produção ou de lazer.

Os componentes de rede, quando enquadrados num ambiente funcional, desempenham um papel determinado pelo fabricante e pelo administrador da rede. A sua actividade depende de regras, que, caso não sejam seguidas, poderão afectar o desempenho da rede. Por este motivo, estes componentes geralmente preveêm mecanismos que lhes permitem exportar informação relativa ao estado de funcionamento para que situações erróneas possam ser rapidamente detectadas e corrigidas.

Os mecanismos geralmente incluem sinalização visual, como os indicadores de conexão nas placas de rede, ou sinalização sonora. Estes nem sempre são práticos devido à necessidade de alguém se deslocar próximo do componente. O mesmo se passa para a modificação de estado, que terá de ser efectuada por intervenção directa.

A unidade de instrumentação de um sistema de gestão, designado por agente, consiste num programa associado aos componentes de rede. Basicamente, o agente exporta a interface de gestão para possibilitar o acesso remoto à informação de funcionamento. Adicionalmente, permite que esta seja modificada através de configuração remota.

O agente apresenta duas faces: comunicação com o sistema de gestão e associação ao componente de rede para monitorização e controlo. Para que um agente se possa adaptar à mudança será necessário prever mecanismos modulares em cada uma das vertentes. O mecanismo de comunicação deverá incorporar protocolos de comunicação diferentes de forma a suportar alterações no meio de comunicação sem afectar o contacto com o agente e, consequentemente, o funcionamento do sistema de gestão.

Em termos de instrumentação, a alteração de componentes de rede poderá fazer com que o agente associado tenha de ser modificado. O acréscimo de funcionalidade implica, muitas vezes, que o componente apresente mais possibilidades de configuração assim como mais estados de funcionamento, pelo que o agente terá de ser actualizado de acordo. Para minimizar o esforço de actualização será necessário encapsular a funcionalidade básica de agente, sendo necessário modificar apenas a forma como vai funcionar a instrumentação.

Neste capítulo é apresentada uma arquitectura modular para o desenvolvimento de agentes com suporte para vários protocolos. Esta arquitectura prevê mecanismos para o desenvolvimento de agentes de gestão, nomeadamente a organização da informação de gestão, comunicação e utilitários. Entre estes últimos encontram-se um MIB Parser, um módulo de persistência baseado em XML e um módulo de identificação uniforme de parâmetros e recursos SNMP recorrendo a URI (*Uniform Resource Identifiers*). A arquitectura foi implementada sob a forma de uma API (*Application Programming Interface*) e encontra-se disponível na Internet para livre utilização sob a licensa GPL (http://www.fsf.org/licenses/gpl.html).

4.1 Requisitos e objectivos

A arquitectura é implementada sob a forma de uma API, doravante designada por Agent API [Lopes02a] dado que é especificamente vocacionada para o desenvolvimento de agentes de gestão.

A Agent API consiste num conjunto de classes que encapsulam as operações comuns dos agentes SNMP, independentemente da instrumentação e dos detalhes definidos nos módulos MIB. O desenvolvimento de agentes é efectuado por especialização (derivação) das classes que constituem a API, providenciando as particularidades de cada agente.

Esta biblioteca de classes foi desenvolvida com o objectivo de servir de base para a implementação de agentes de gestão genéricos, sendo utilizada no contexto do trabalho apresentado nesta tese para o desenvolvimento de todos os módulos de gestão.

Os requisitos impostos no desenvolvimento desta API foram:

- agrupar mecanismos de organização e gestão de informação interna do agente, nomeadamente, a ordenação de objectos por OID, gestão de índices de tabelas de objectos e a criação e eliminação dinâmica de objectos;
- conter mecanismos de comunicação que poderão recorrer a protocolos distintos em utilização exclusiva ou simultânea, nomeadamente, SNMP, AgentX, HTTP, RMI ou CORBA;

- ter à sua responsabilidade as operações de `set`, `get`, `get-next`, `get-bulk` e `trap`, a gestão das referências para os objectos que constituem a MIB e a persistência de informação;
- suportar o mais recente mecanismo de transacções SNMP, ou seja, implementar o conceito de `RowStatus` [RFC2579].
- ser modular, de forma a não limitar a evolução futura dos agentes e poder integrar, no futuro, outros mecanismos;
- ser extensível, para conseguir, com relativa facilidade, acompanhar a efervescência tecnológica no campo de gestão de redes;
- ser eficiente, para não sobrecarregar desnecessariamente a plataforma (geralmente, limitada em termos de recursos) e conseguir responder satisfatoriamente aos comandos recebidos.

Em termos gerais, a arquitectura apresenta uma estrutura modular que lhe permite a evolução independente de cada bloco (Figura 4.1).

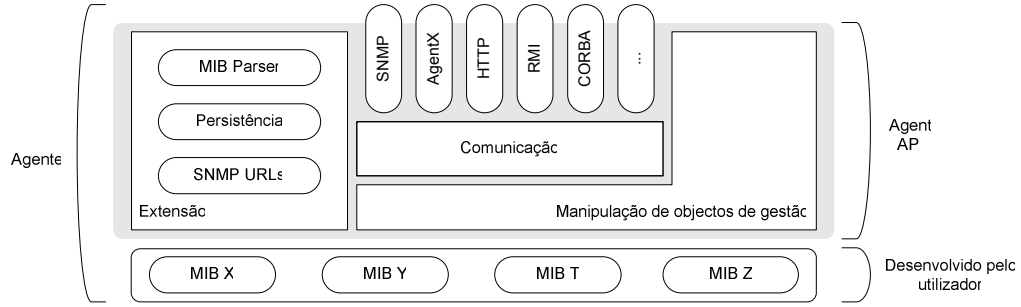


Figura 4.1 – Estrutura global do agente e da Agent API.

O *agente* é constituído pelo código *desenvolvido pelo utilizador* e pelos serviços da *Agent API*. O programa resultante é executado numa Java Virtual Machine, o que lhe dá independência da plataforma.

O módulo de *Manipulação de objectos de gestão* é a base da Agent API. A sua responsabilidade é fornecer um conjunto de serviços de controlo do ciclo de vida dos objectos de gestão definidos nas MIBs, como a indexação dos objectos de gestão por OID para optimizar a pesquisa de informação no agente e a facilitar as operações *walk* (`get-next`, `get-bulk`). Outros serviços preveêm a criação e remoção dinâmica de objectos, organização de índices de tabelas e controlo transaccional SNMP recorrendo a colunas do tipo `RowStatus` [RFC2579].

O módulo de *Comunicação* encarrega-se de efectuar a correspondência entre os comandos protocolares recebidos pelos mecanismos específicos (*SNMP*, *AgentX*, *HTTP*, ...) e as operações da plataforma com a finalidade de consultar ou modificar parâmetros de funcionamento. É este módulo que isola as especificidades de cada protocolo do funcionamento intrínseco do agente. De mencionar que os módulos de comunicação são carregados em tempo de execução de acordo com os requisitos do utilizador, sem ser necessário compilar o agente. Por exemplo, um agente usando exclusivamente SNMP irá necessitar apenas do módulo SNMP. Outros agentes poderão necessitar de outros protocolos, como HTTP, SSL, RMI, CORBA, etc..

Os módulos descritos são essenciais para o funcionamento de agentes baseados na Agent API. No entanto, é possível adicionar ferramentas que, apesar de não serem fundamentais, podem ser valiosas para a funcionalidade pretendida do agente. Cada ferramenta é vista como uma *Extensão* à API e providencia serviços que podem ser utilizados pelas MIBs integradas no agente ou mesmo pelos diferentes mecanismos de comunicação. Nesta categoria enquadram-se, por exemplo, um MIB Parser, serviços de persistência ou serviços de localização de recursos SNMP.

De acordo com o modelo anterior e com base no conjunto de módulos apresentados, o utilizador define o comportamento do agente recorrendo à derivação de classes e à passagem de mensagens entre objectos de forma a implementar o comportamento do agente como descrito nas MIBs específicas.

Nas secções seguintes descreve-se cada um dos módulos com maior detalhe, incluindo as opções tomadas e a sua estrutura interna.

4.2 Manipulação de objectos de gestão

O módulo de manipulação de objectos de gestão controla o ciclo de vida dos objectos de gestão (Figura 4.1). Como ciclo de vida entende-se a indexação, criação, eliminação e acesso ao objecto que representa determinado parâmetro de funcionamento. Por exemplo, o parâmetro *sysDescr*, definido na SNMPv2-MIB [RFC1907] é definido como `read-only` e devolve uma descrição textual da entidade. Este objecto possui uma posição bem definida na árvore de OIDs, não se pode criar nem eliminar por intermédio de comandos SNMP e o seu valor não pode ser alterado. O módulo de manipulação é responsável por cumprir os termos definidos pelos RFCs para cada objecto de gestão.

Na base do seu funcionamento encontra-se uma estrutura de dados em árvore binária. Esta efectua a indexação de objectos por OID de forma a permitir a sua ordenação natural. Esta estrutura responde a operações de pesquisa (base para as operações de *walk*) com um custo de log(N), onde N é o número de objectos que o agente possui [Cormen01]. A estrutura permite também adicionar e remover objectos em tempo de execução mantendo sempre a ordenação baseada em OID.

Para ilustrar o funcionamento deste módulo, vamos supor que um pedido de `get-next` é recebido para o objecto indexado com o OID .1.3.6.1.6.2. Se a árvore possui os objectos com os OIDs .1.3.6.1.2.1.0 e .1.3.6.1.94.2.0, o objecto devolvido será .1.3.6.1.94.2.0 uma vez que tem um OID posterior.

O serviço mais complexo disponibilizado por este módulo consiste na implementação do mecanismo de transacções definido para as aplicações SNMP. Este permite iniciar operações apenas quando todos os parâmetros necessários se encontram disponíveis.

O mecanismo transaccional SNMP assenta em estruturas tabulares designadas por tabelas abstractas (*conceptual tables*). Estas são constituídas por colecções ordenadas de objectos que podem conter zero ou mais linhas. Cada linha pode conter um ou mais objectos que correspondem aos parâmetros da operação. Apenas quando as colunas essenciais da linha se encontram preenchidas é que se torna possível activar a operação.

As operações podem encontrar-se em quatro estados distintos e dado que as operações estão relacionadas com as linhas de uma tabela abstracta, também estas estarão num desses quatro estados. Uma operação pode:

- não existir (`notExisting`) – a linha associada ainda não foi criada,
- existir e não possuir todos os parâmetros necessários (`notReady`) – quando a linha já foi criada mas ainda não possui todos os parâmetros que a operação requer,
- possuir todos os parâmetros necessários mas não se encontrar activa (`notInService`) – a linha possui já todos os valores necessários mas ainda não foi despoletada,
- estar em funcionamento (`active`) – a linha encontra-se já activada.

O estado é monitorado ou modificado por intermédio de um objecto coluna específico – o objecto de `RowStatus`. (Figura 4.2).

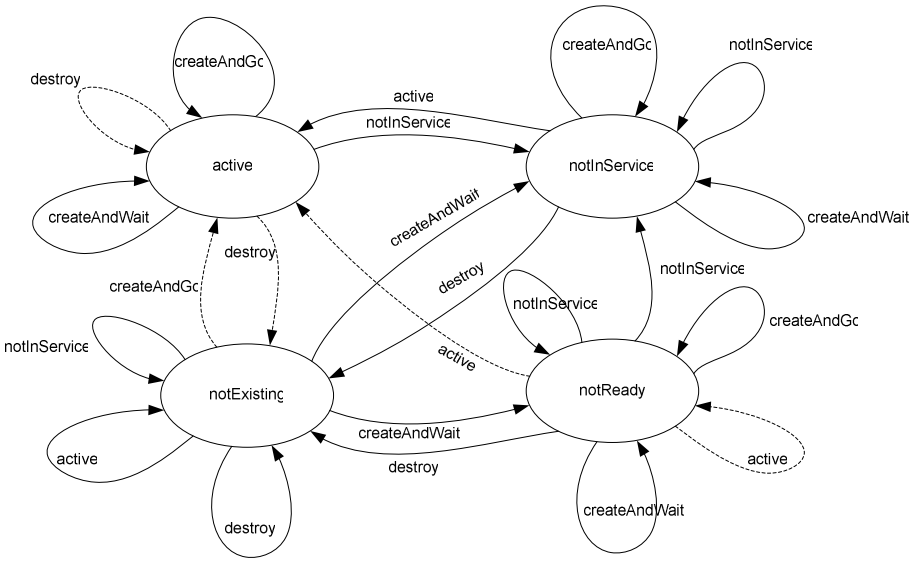


Figura 4.2 – Diagrama de estados de `RowStatus`.

A aplicação de gestão pode emitir cinco comandos diferentes: `active`, `notInService`, `createAndGo`, `createAndWait` e `destroy` [RFC2579]. De notar que poderá haver transição de estado dependendo do valor de qualquer objecto coluna. Por exemplo, a transição de `notReady` para `notInService` quando a estação de gestão envia o valor em falta.

Para simplificar a máquina de estados algumas transições podem ser decompostas em duas ou mais. A entrada `createAndGo` que poderá despoletar a transição de `notExisting` para `active`, pode ser substituída pelas entradas `createAndWait` seguida de `active`, passando pelo estado `notInService`. O mesmo processo pode ser seguido quando o estado se encontra em `notReady`, `notInService` e `active`. A entrada `destroy`, que poderá provocar a transição de estado de `active` para `notExisting`, pode ser decomposta nas entradas `notInService` seguida de `destroy`. A mesma decomposição pode ser feita quando o estado se encontra em `notReady`.

Além da estrutura de objectos, o módulo de manipulação de objectos de gestão lida com valores e com os tipos de dados associados. Tal como definidos no modelo de informação, os objectos SNMP podem ser instâncias dos seguintes tipos de dados nativos: `INTEGER`, `OCTET STRING`, `OBJECT IDENTIFIER`, `SEQUENCE`, `SEQUENCE OF`. Estes, por sua vez, estão na base da definição de `Integer32`, `IpAddress`, `Counter32`, `Gauge32`, `Unsigned32`, `TimeTicks`, `Opaque` e `Counter64`, definidos na SMIv2 [RFC2578].

A Agent API contém classes que encapsulam as características de cada tipo de dados, incluindo algumas especializações (Figura 4.3).

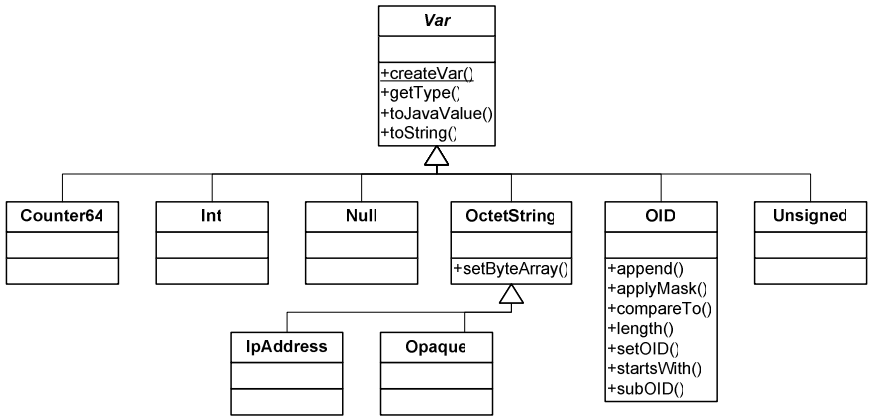


Figura 4.3 – Tipos de dados SNMP.

As MIBs, por sua vez, podem definir novos tipos de dados, como `DateAndTime`, `DisplayString`, `PhysAddress`, `TruthValue` ou `BITS`. Estes, denominados convenções textuais, têm um nome diferente, uma sintaxe semelhante e uma semântica mais rigorosa. Para lidar com eles foram previstas classes que permitem reconhecer a respectiva convenção (Figura 4.4).

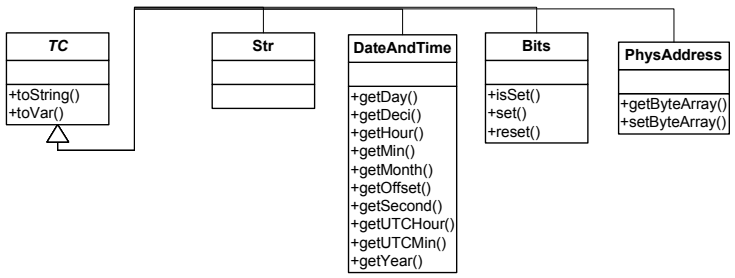


Figura 4.4 – Convenções textuais SNMP.

O seguinte código apresenta um exemplo de utilização das classes de tipos de dados SNMP e da sua interpretação de acordo com uma convenção textual.

```
...
// varBind traz o PDU SNMP com um OctetString que representa
// DateAndTime
OctetString aVar = (OctetString)varBind.getVar();
...
// Imprimir em formato recomendado pela convenção textual
System.out.println(new DateAndTime(aVar).toString());
...
```

Em situações inversas, ou seja, quando é necessário converter um formato definido por uma convenção textual para um tipo básico de SNMP será necessário criar, em primeiro lugar, um objecto do tipo `TC`.

```
...
// Criar um objecto que representa uma cadeia de bits
```

```
Bits b = new Bits("0001011010010101001");
...
// v será um OctetString
Var v = b.toVar();
...
```

A manipulação de informação de gestão é efectuada por intermédio de um conjunto de classes que têm a capacidade de criar, adicionar, ordenar, ler e escrever informação, remover e destruir objectos de gestão. Os objectos de gestão que no ambiente SNMP têm a função de providenciar um ponto de acesso à informação, por exemplo *sysDescr*, *sysContact* ou *ifTable*, são representados na Agent API como uma hierarquia de classes Java.

Na base da hierarquia encontra-se a classe `AgentObject`, caracterizada pelas operações genéricas de consulta de informação (`get`) e identificação do objecto de gestão (`getOIDObject`). Esta classe não possui métodos para escrever informação (`set`) nem para obter referências para objectos seguintes (`get-next`), da responsabilidade da interface `WritableAgentObject` e da estrutura em árvore binária, respectivamente (Figura 4.5).

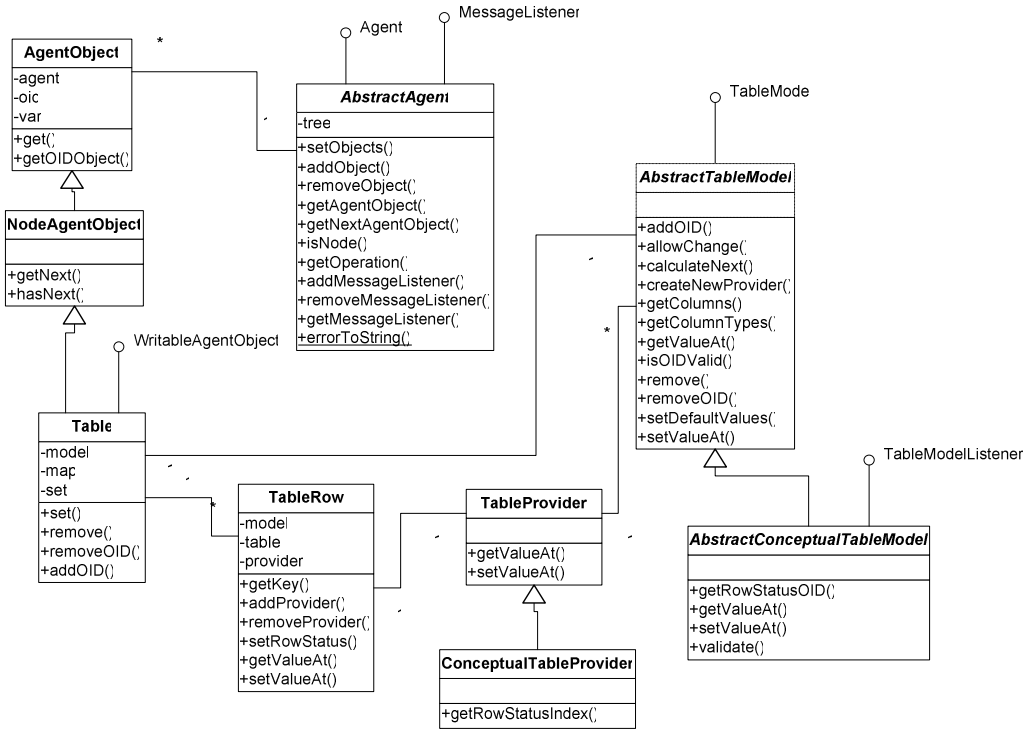


Figura 4.5 – Classes principais do módulo de manipulação de informação de gestão.

De acordo com a sua localização na estrutura definida na MIB, os objectos de gestão podem ser:

- instâncias – possuem um valor associado e encontram-se na extremidade dos ramos da árvore. São derivados directamente da classe `AgentObject`;
- nós – contêm outros objectos, pelo que mantêm uma árvore binária interna. Disponibilizam métodos para obter referências para os objectos internos (`getNext`) e são derivados da classe `NodeAgentObject`;

- tabelas – tal como os nós, contêm outros objectos e disponibilizam métodos para obter referências para os objectos internos (`getNext`). Os valores, número e tipo de dados das colunas são obtidos e validados por intermédio de uma classe do tipo `TableModel` ou `AbstractConceptualTableModel` no caso de serem tabelas conceptuais. As classes `TableProvider` e `TableRow` contêm os valores ou os mecanismos de instrumentação para cada linha da tabela.

A classe `AbstractAgent` é responsável por gerir as referências para todos os objectos, encapsula a tabela binária de indexação e disponibiliza métodos para adicionar, remover ou consultar objectos do tipo `AgentObject`. Adicionalmente, providencia também uma interface com o módulo de comunicações, detalhado na secção seguinte.

Resumidamente, quando uma mensagem é recebida por um objecto do tipo `AbstractAgent`, este localiza-o (`get` ou `set`) ou localiza o objecto imediatamente a seguir (`get-next` ou `get-bulk`). Após obter uma referência para o objecto em causa invoca o método adequado para consultar ou modificar o valor associado (Figura 4.6).

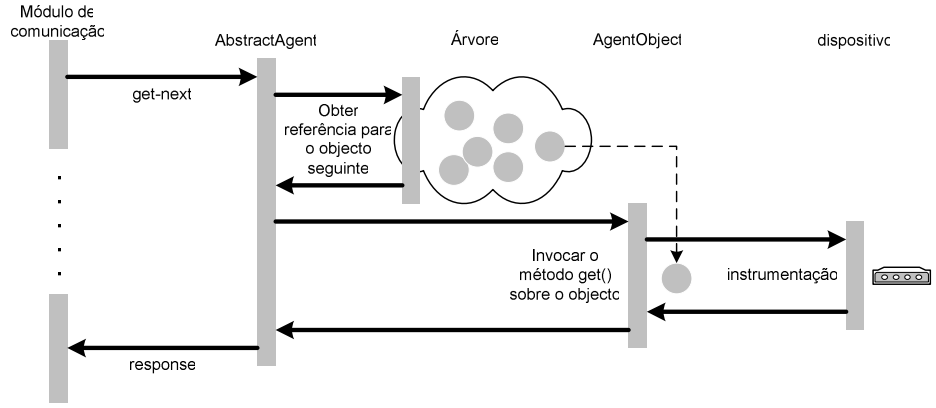


Figura 4.6 – Resposta a um get-next.

Para criar um agente específico será necessário derivar a classe `AbstractAgent` e criar instâncias dos objectos iniciais no método `setObjects`. Estes serão então adicionados à árvore binária interna e ordenados automaticamente.

Um exemplo ilustrativo do código necessário para criar um agente de gestão pode ser visto no Anexo A. Este contém um objecto `read-only` (*sysUpTime*), outro `read-write` (*sysContact*) e uma tabela abstracta (*snmpTargetAddrTable*)

4.3 Módulos de comunicação

O módulo de manipulação de objectos de gestão é complementado com o módulo de comunicação para poder receber e processar comandos recebidos da rede assim como enviar notificações (Figura 4.1).

As mensagens podem fluir na direcção do agente, como por exemplo os comandos emitidos pela aplicação de gestão, ou em sentido contrário, como por exemplo as notificações. Devido a este facto o módulo de comunicação terá de ser bidireccional, ou seja, deverá reencaminhar os comandos da rede para o interior do agente e enviar notificações do agente para a rede (Figura 4.7).

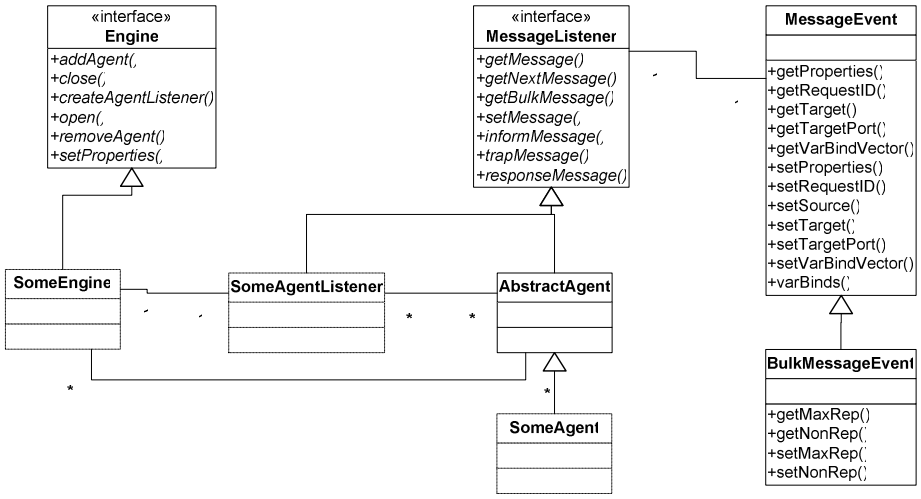


Figura 4.7 – Estrutura de classes do módulo de comunicação.

Entre o módulo de comunicação, os mecanismos de comunicação específicos e o módulo de manipulação de objectos de gestão fluem, única e exclusivamente, objectos do tipo `MessageEvent`. Esta classe encapsula a informação necessária à comunicação. De notar que os mecanismos de comunicação referem-se aos componentes responsáveis por cada protocolo específico, nomeadamente SNMP, AgentX, HTTP, RMI e CORBA.

Para que as mensagens sejam trocadas entre os diversos intervenientes é necessário efectuar uma troca prévia de referências. Este passo permite registar o agente num determinado mecanismo de comunicação para poder receber mensagens do exterior assim como registar o mecanismo de comunicação no agente para enviar notificações para o exterior (Figura 4.8).

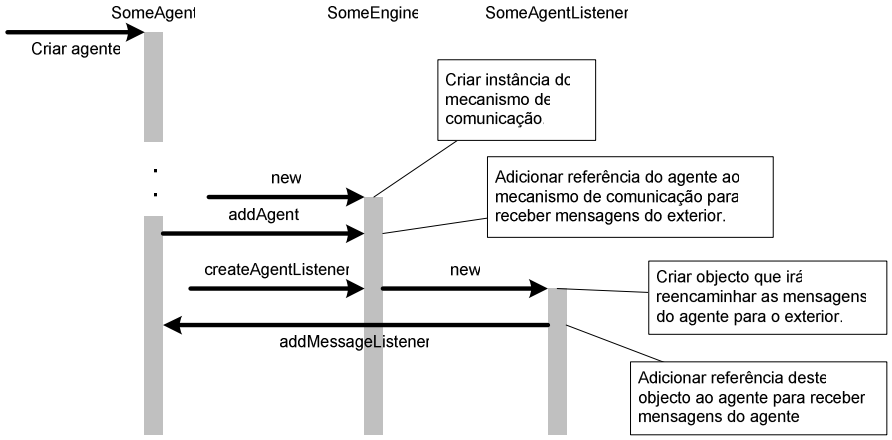


Figura 4.8 – Processo de registo de mecanismos de comunicação.

Quando um mecanismo de comunicação, por exemplo `SomeEngine`, é instanciado este recebe uma referência para o agente por intermédio do método `addAgent(Agent)`. O agente fica registado e poderá receber mensagens que o mecanismo lhe envie – tipicamente as recebidas da rede. Ao ser invocado o método `createAgentListener` o mecanismo cria uma instância derivada de `MessageListener` cujo papel será receber mensagens do agente. Será necessário

registar esta nova instância no agente por intermédio do método `addMessageListener`. Após a operação de registo, as mensagens fluem entre o agente e cada uma das instâncias, nomeadamente `SomeEngine` para as mensagens recebidas e `SomeAgentListener` para as mensagens enviadas (Figura 4.9). Nesta figura as setas preenchidas a branco indicam mensagens protocolares enquanto as setas preenchidas a preto assinalam a invocação de métodos entre objectos.

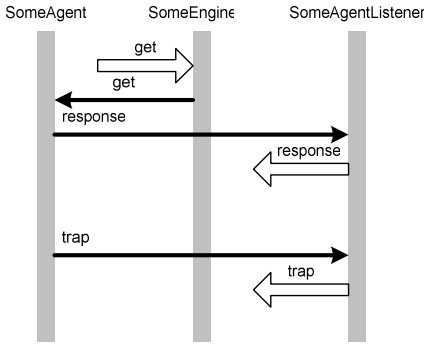


Figura 4.9 – Fluxo de mensagens entre o agente e o mecanismo de comunicação.

Apesar de o exemplo anterior mencionar um mecanismo de comunicações abstracto (`SomeEngine`), a Agent API prevê a utilização de um conjunto específico de protocolos, nomeadamente SNMP, AgentX e HTTP. Estes serão descritos mais pormenorizadamente nas secções seguintes.

4.3.1 Mecanismo de comunicação SNMP

O mecanismo de comunicação SNMP utiliza os serviços do módulo de comunicação para traduzir os comandos SNMP em objectos do tipo `MessageEvent`. Estes são depois enviados para o módulo de manipulação de objectos de gestão onde serão executados. As notificações seguem o caminho inverso – o módulo de manipulação de objectos de gestão gera objectos do tipo `MessageEvent` que envia para o módulo de comunicação. Este, por sua vez, submete-os ao mecanismo de SNMP onde serão convertidos para notificações SNMP (Figura 4.1).

A estrutura da Agent API é baseada no funcionamento do modelo SNMP e nos comandos que este define, pelo que a passagem de mensagens entre os módulos de manipulação e de comunicação assenta nas operações `getMessage`, `getNextMessage`, `getBulkMessage`, `setMessage`, `informMessage`, `trapMessage` e `responseMessage` da interface `MessageListener`. Apesar de se encontrarem previstos todos os comandos SNMP nem todos são necessários do lado do agente. Tipicamente os comandos do tipo `inform` ou `trap` fazem mais sentido do lado da aplicação de gestão, pelo que se encontram latentes na Agent API. Em caso de necessidade, esta capacidade poderá ser adicionada por eventuais sub-classes.

Os comandos de consulta, nomeadamente `get`, `get-next` e `get-bulk` podem ser resumidos a dois únicos comandos – `calculateNext` e `getMessage` (Figura 4.10).

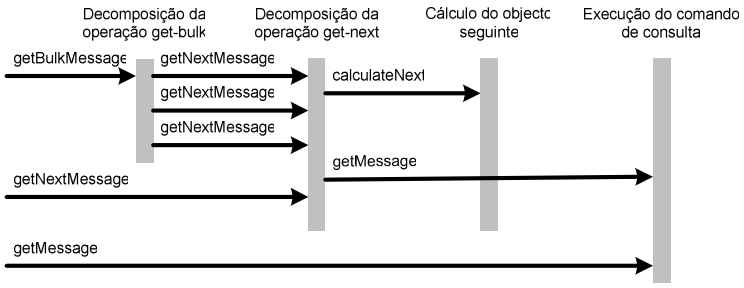


Figura 4.10 – Decomposição de comandos de consulta SNMP.

O comando `get-bulk` é decomposto em várias operações de `get-next` de acordo com os parâmetros de não-repetição e de max-repetições. O comando `get-next`, por sua vez, consiste em dois passos: cálculo do objecto seguinte com base no OID e emissão do comando `get`. O comando de escrita (`set`) encontra uma correspondência directa.

O mecanismo de comunicação SNMP integrado na Agent API assenta na JoeSNMP API [JoeSNMP], uma biblioteca de funções que implementam o protocolo SNMPv2c. As características específicas dos PDUs, nomeadamente, os parâmetros de segurança, o endereço e porta do remetente, o identificador de pedido entre outros são armazenados no objecto `MessageEvent` para serem utilizados na construção da resposta.

O último passo realizado pelo mecanismo SNMP trata da conversão de tipos de dados. Também aqui, a correspondência de tipo de dados é directa bastando apenas instanciar os objectos apresentados na Figura 4.3 de acordo com os tipos de dados recebidos. Em termos práticos, o mecanismo SNMP corresponde a apenas uma classe que implementa a interface `Engine`.

4.3.2 Mecanismo de comunicação AgentX

A norma AgentX define um mecanismo de extensão de agentes SNMP, ou seja, introduz a possibilidade de adicionar novas MIBs a agentes já existentes. O seu princípio de funcionamento assenta nos conceitos de mestre, um agente com capacidade para receber comandos SNMP, subagente, que contém mecanismos de instrumentação e num protocolo de comunicação entre o mestre e os subagentes. Cada mestre tipicamente suporta um ou mais subagentes. O mecanismo de AgentX da Agent API permite associar subagentes baseados na Agent API a agentes SNMP existentes (Figura 4.1).

A comunicação entre o mestre e os subagentes é efectuada por um protocolo específico baseado em *sockets* de domínio Unix ou de domínio Internet (TCP) o que lhe dá independência a nível da localização. O protocolo AgentX define vários comandos sob a forma de PDUs (*Protocol Data Units*) para efectivar a comunicação (Tabela 4.1).

Neste conjunto de PDUs é de assinalar o processo de configuração baseada em comandos `set`. Este segue as estapas de *test*, *commit*, *undo* e *cleanup*:

- *test* – o subagente efectua as operações relacionadas com a alteração ou criação de valores que o comando `set` define sem no entanto aplicar as alterações;
- *commit* – indica ao subagente que deve aplicar as alterações assinaladas no comando *test*;

- *undo* – no caso de ocorrer algum erro em qualquer das fases, este comando é emitido pelo mestre para indicar ao subagente que deve desfazer as alterações aplicadas pelo comando *commit*;
- *cleanup* – comando emitido pelo mestre para indicar ao subagente que pode libertar os recursos utilizados durante a operação de `set`.

Tabela 4.1 – Conjunto de PDUs do protocolo AgentX.

PDU	Descrição
agentx-Open-PDU	Gerado pelo subagente para estabelecer uma ligação ao mestre.
agentx-Close-PDU	Gerado pelo mestre ou pelo subagente para terminar a ligação.
agentx-Register-PDU	Gerado pelo subagente para registar uma secção da árvore de OIDs.
agentx-Unregister-PDU	Gerado pelo subagente para remover uma secção da árvore de OIDs.
agentx-Get-PDU	Comando get.
agentx-GetNext-PDU	Comando get-next.
agentx-GetBulk-PDU	Comando get-bulk.
agentx-TestSet-PDU	Usado para processar comandos set.
agentx-CommitSet-PDU	Usado para processar comandos set.
agentx-UndoSet-PDU	Usado para processar comandos set.
agentx-CleanupSet-PDU	Usado para processar comandos set.
agentx-Notify-PDU	Gerado pelo subagente para despoletar uma notificação no mestre.
agentx-Ping-PDU	Gerado pelo subagente para monitorar o estado do mestre.
agentx-IndexAllocate-PDU	Gerado pelo subagente para reservar um valor de índice.
agentx-IndexDeallocate-PDU	Gerado pelo subagente para remover o valor de índice.
agentx-AddAgentCaps-PDU	Gerado pelo subagente para informar o mestre das suas capacidades.
agentx-RemoveAgentCaps-PDU	Gerado pelo subagente para informar o mestre de uma alteração nas suas capacidades.
agentx-Response-PDU	Comando de resposta.

O mecanismo de AgentX da Agent API efectua este controlo automaticamente de forma a enviar ao módulo de manipulação de informação apenas uma mensagem de `setMessage` após a correcta transição de estados. Esta opção uniformiza o procedimento de comunicação com os outros mecanismos possibilitando a sua utilização de forma transparente com os agentes baseados na Agent API.

O protocolo AgentX é implementado na Agent API com o auxílio de uma API disponível em domínio público denominada JAX – Java AgentX Toolkit [JAX]. Esta API encapsula os PDUs e os tipos de dados definidos pelo protocolo permitindo efectuar a conversão de mensagens recorrendo à invocação de métodos. Em termos práticos o mecanismo AgentX da Agent API corresponde apenas à implementação de uma classe derivada da interface `Engine`.

Além da conversão entre os PDUs e os comandos `getMessage`, `getNextMessage`, `getBulkMessage`, `setMessage`, `informMessage`, `trapMessage` e `responseMessage`, relativamente simples devido à proximidade funcional do protocolo AgentX e do modelo SNMP, é necessário efectuar a conversão do tipo de dados. Este passo, tal como no caso do mecanismo SNMP descrito na secção anterior, baseia-se na criação de objectos específicos apresentados

na Figura 4.3 com base nos dados recebidos. A título ilustrativo apresenta-se de seguida o código usado neste processo:

```
AgentXVarBind vb = getAgentXVarBind();
switch(vb.getType()) {
  case AgentXVarBind.INTEGER: return new Int(vb.intValue());
  case AgentXVarBind.COUNTER32: return new Counter(vb.longValue());
  case AgentXVarBind.OCTETSTRING: return new OctetString(vb.bytesValue());
  ...
}
```

Uma limitação do modelo AgentX é a impossibilidade de aceder à informação de gestão do mestre ou de outros subagentes a partir de um subagente particular. O modelo foi pensado de forma a ser o mestre a controlar os subagentes não tendo sido prevista a possibilidade inversa, ou seja, de os subagentes acederem à informação de gestão do mestre ou de outros subagentes. Esta opção pode revelar-se uma limitação em algumas situações como, por exemplo, a definição de subagentes baseados na Expression MIB. De facto, a Expression MIB foi definida com o objectivo de obter informação sobre o agente local. Se estiver implementada como um subagente então só poderá obter informação sobre ele próprio o que não faz sentido.

As MIBs que efectuam a instrumentação sobre o componente de rede a monitorar e não sobre outras MIBs, como por exemplo a HOST-RESOURCES-MIB, são ideais para serem implementadas como subagentes uma vez que permitem definir agentes SNMP modulares e extensíveis de acordo com as opções definidas pelo utilizador. O agente SNMP que equipa o sistema operativo Windows assim como o agente SNMP NET-SNMP [NetSNMP] seguem esta abordagem.

4.3.3 Mecanismo de comunicação HTTP

O protocolo de transferência de hypertexto (HTTP – *Hypertext Transfer Protocol*) é o principal mecanismo de transporte utilizado na World Wide Web. O seu princípio de funcionamento, orientado pelo paradigma cliente/servidor, é relativamente simples: o cliente estabelece uma ligação a um determinado servidor, envia um pedido e aguarda a resposta correspondente; o servidor, após a recepção do pedido, efectua o seu processamento, envia a resposta e termina a ligação.

Os pedidos efectuados em HTTP podem ser do tipo `GET`, utilizado para obter um recurso do servidor – tipicamente qualquer tipo de dados ou um serviço específico, ou do tipo `POST` quando é necessário enviar informação para o servidor [RFC2616].

Um dos recursos frequentemente associado ao HTTP consiste em documentos escritos na linguagem HTML (*Hypertext Markup Language*). Esta permite especificar o conteúdo e o formato de informação assim como definir ligações entre palavras chave e outros documentos relacionados. Os documentos HTML são visualizados em clientes HTTP, os denominados navegadores da Internet, que são tipicamente leves, existem para várias plataformas (de PCs a telemóveis), vários sistemas operativos (Unix ou Windows) e vários tipos de interfaces gráficas (baseadas em texto ou em janelas).

As vantagens da utilização de um mecanismo baseado em HTTP e HTML para aceder à informação de gestão são diversas [Mullaney96]:

- dispensa a utilização de software de gestão específico para monitorar ou controlar dispositivos de rede;

- uma vez que a interface é exportada directamente pelo agente os problemas de actualização de software derivados da instalação de novos agentes não se colocam. A estação de gestão não necessita saber ou conhecer cada uma das MIBs de todos os agentes reconhecidos. Serão estes a apresentar a sua funcionalidade sob a forma de documentos HTML;
- a utilização de navegadores de Internet como o Internet Explorer, Netscape Navigator, Mozilla, lynx ou outros introduz a independência de plataforma e de localização do sistema de gestão;
- a utilização de ligações entre os documentos permite integrar a informação de gestão com a documentação de apoio. Este facto permite definir sistemas de ajuda sensíveis ao contexto bem como associar documentação genérica sobre o sistema.

Existem, no entanto, alguns aspectos negativos da utilização do HTTP como por exemplo a latência mais elevada devido às fases de estabelecimento de ligação do TCP e a eventual sobrecarga que recai sobre os agentes devido à exportação de interface.

A abordagem seguida na Agent API é dar ao utilizador a possibilidade de escolha sobre os protocolos a utilizar para contactar um agente de gestão. Neste âmbito encontra-se previsto um mecanismo de comunicação HTTP com a responsabilidade de receber e responder a comandos HTTP com base em informação HTML (Figura 4.1).

Os protocolos HTTP e o SNMP são consideravelmente diferentes tanto ao nível dos comandos trocados como ao nível da informação. Devido a este facto, o processamento que o mecanismo HTTP terá de desempenhar é substancialmente superior e a sua estrutura mais complexa do que nos casos anteriores, nomeadamente nos mecanismos SNMP e AgentX.

À semelhança de um sistema SNMP as operações de monitorização baseadas num navegador de Internet resultam na recuperação de um conjunto de dados de gestão. Esta, no entanto, encontra-se codificada sob a forma de um documento HTML o que faz surgir a primeira questão. Conceptualmente, como deve ser constituído cada documento HTML de resposta ao cliente?

O agente possui a informação organizada numa estrutura em árvore, organizada por OIDs. Esta é composta por vários nós, tabelas e escalares onde cada objecto é referenciado por um nome constituído pelos identificadores parciais de cada ramificação (Figura 4.11). Por exemplo, considerando que o nó raiz possui o identificador 1, o nome do objecto na extremidade inferior direita seria 1.3.2.3.

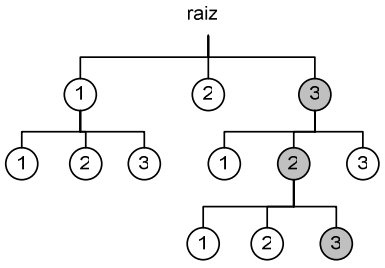


Figura 4.11 – Estrutura de informação do agente.

Este conceito será utilizado para designar os objectos que compõem cada página. Por exemplo, assumindo que o utilizador selecciona o nó raiz, a página gerada irá conter os objectos 1.1, 1.2 e 1.3 (Figura 4.12).

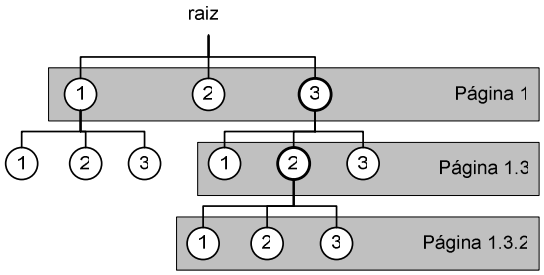


Figura 4.12 – Páginas geradas com base em opções do utilizador.

Após seleccionar o objecto 1.3, a página será substituída pela página 1.3, contendo os objectos 1.3.1, 1.3.2 e 1.3.3. Finalmente, seleccionando o objecto 1.3.2, a página será substituída pela 1.3.2, com os objectos 1.3.2.1, 1.3.2.2 e 1.3.2.3.

A segunda questão, ainda relacionada com a apresentação do documento HTML ao utilizador, refere-se ao tipo de informação a apresentar. Cada objecto de gestão poderá ser um nó, um escalar ou uma tabela o que irá afectar o seu formato de representação em HTML. Se um objecto corresponde a um nó será representado por uma ligação para a página associada. Se o objecto é um escalar será apresentado o seu valor actual. Se o objecto corresponde a uma tabela os valores serão organizados de acordo com uma representação em tabela.

De acordo com as restrições de acesso dos objectos SMI, cada objecto poderá ser também de leitura e/ou de escrita. Para os objectos apenas de leitura será apresentado o seu valor. Os objectos que possam ser modificados irão ser representados por formulários de forma a ser possível ao utilizador modificar ou definir o seu valor.

Além do valor correspondente, cada objecto poderá também ser associado a outra informação como por exemplo o seu nome, OID, descrição ou sintaxe. O agente só por si não exporta essa informação pelo que será necessário interpretar ficheiros MIB correspondentes tal como acontece nas aplicações de gestão. A Agent API contém um módulo de extensão que alberga utilitários genéricos. Um destes, denominado MIB Parser será utilizado pelo mecanismo HTTP para interpretar a informação definida em ficheiros MIB e, consequentemente, associar ao valor informação relacionada. O MIB Parser será detalhado na secção 4.4.

Basicamente, o mecanismo HTTP é composto por duas secções. Uma é responsável por efectuar a conversão de informação e a geração de páginas HTML enquanto que a outra recebe e envia mensagens HTTP (Figura 4.13).

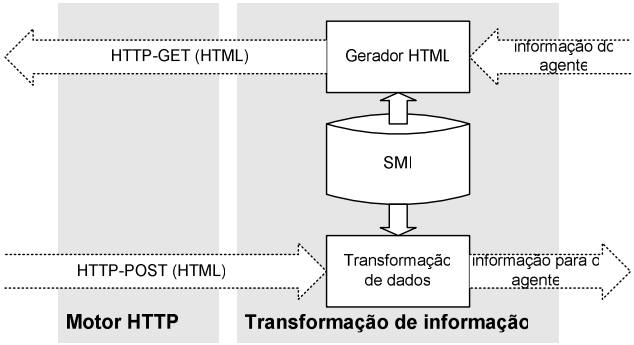


Figura 4.13 – Conversão de informação SMI para HTML.

Sobre esta arquitectura elementar foram introduzidas alterações para potenciar o mecanismo HTTP de forma a suportar diferentes tipos de clientes. Estas alterações foram aplicadas principalmente a nível da conversão de informação para tornar possível gerar tipos de dados variados como HTML, WML ou VXML (*Voice XML*). Esta abordagem possibilita aceder à informação de gestão a partir de diferentes terminais como por exemplo os navegadores de Internet, telefones móveis com capacidade WAP e mesmo telefones sem capacidade visual de apresentação de informação. Esta última pode recorrer a documentos VXML para reconhecer a voz do utilizador e a marcação de números (Figura 4.14).

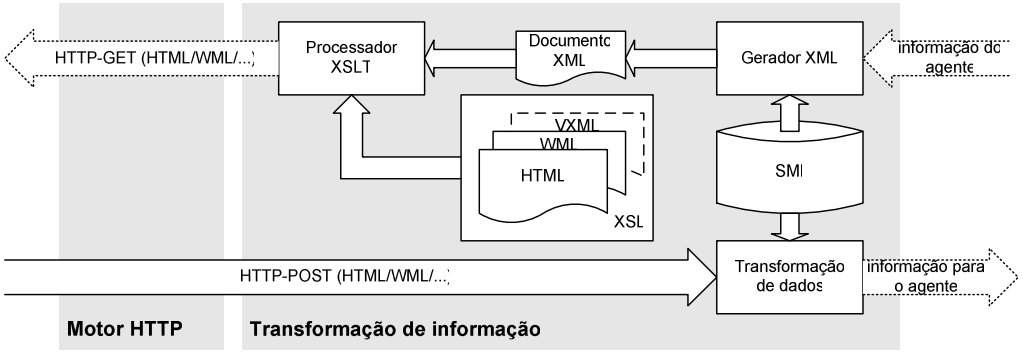


Figura 4.14 – Arquitectura de conversão de informação SMI com base em XML.

Na prática, a conversão de informação atravessa duas fases. Em primeiro lugar a informação SMI associada aos valores do agente são convertidos para XML. Este documento é posteriormente transformado com base em regras definidas em ficheiros XSL para um formato diferente, como por exemplo HTML, WML ou VXML.

O *Network Management Research Group* (NMRG) do IRTF (*Internet Research Task Force*) abordou, no passado, um método de representação de informação SMI [RFC1155][RFC2578] em XML [XML98]. Apesar de, neste momento, o trabalho estar pendente devido à evolução das normas SMIng [Strauss01], a representação em XML constitui não só um excelente formato para troca de informação como permite modificar o seu formato de representação [Schoenwaelder00a]. O trabalho desenvolvido por este grupo assenta na definição de um documento de definição de tipos de dados (DTD) para permitir, usando aplicações de reconhecimento de código XML, a leitura ou edição de documentos SMI. Este DTD serviu como base para a conversão de informação do mecanismo HTTP da Agent API.

O *Gerador XML* constrói um documento XML baseado nas definições SMI. Adicionalmente, complementa esta informação com informação do agente de forma a incluir os valores associados a cada objecto. Por exemplo, a definição do objecto `sysDescr` da SNMPv2-MIB [RFC1907] em ASN.1 é a seguinte:

```
sysDescr OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A textual description of the entity.  This value should
            include the full name and version identification of the
            system's hardware type, software operating-system, and
            networking software."
    ::= { system 1 }
```

O código XML gerado é um pouco mais extenso. Este facto torna-o mais difícil de interpretar pelo utilizador embora seja directamente aceite por uma aplicação XML:

```
<?xml version="1.0"?>
<smi>
 <scalar name="sysDescr" oid=".1.3.6.1.2.1.1.1" status="mandatory">
  <syntax>
   <type module="RFC1213-MIB" name="DisplayString(SIZE(0..255))"/>
   <typedef optname="DisplayString" basetype="OCTET STRING">
    <parent module="RFC1213-MIB" name="system"/>
   </typedef>
  </syntax>
  <access>read-only</access>
  <description>
    A textual description of the entity.  This value should include the full name and
version identification of the system's hardware type, software operating-system, and
networking software.   It is mandatory that this only contain printable ASCII
characters.
  </description>
  <value oid='.1.3.6.1.2.1.1.1'>Test Agent Simulator</value>
</scalar>
</smi>
```

Este documento é entregue de seguida ao *Processador XSLT* que irá, com base em documentos XSL, gerar código em outros formatos, como HTML ou WML.

Apesar de o SMI definir objectos de gestão, não representa explicitamente os valores que lhe estão associados. O objectivo do NMRG com a definição de um DTD foi, à imagem do SMI, representar a estrutura de uma MIB em XML. Por este motivo não foi prevista uma etiqueta para representação de valores. No âmbito do trabalho apresentado nesta secção foi decidido acrescentar a etiqueta `<value>` de forma a possibilitar ao utilizador consultar em XML os valores de determinados objectos de gestão. Esta opção não invalida a estrutura definida pelo NMRG uma vez que a utilização da etiqueta não é obrigatória. Por outro lado, permite representar não só a estrutura SMI mas também o seu estado.

O mecanismo HTTP, ao ser utilizado com *browsers* de Internet, codifica a própria interface gráfica numa linguagem própria. Esta interface será melhor detalhada no capítulo 7 no contexto das aplicações de gestão.

Ao nível da segurança, as diferenças entre o SNMP e o HTTP são consideráveis. A autenticação em HTTP é efectuada por intermédio de uma página inicial de autenticação. Esta irá solicitar ao utilizador um nome e a palavra chave correspondente. A privacidade é conseguida por intermédio de protocolos do tipo HTTPS para cifrar a comunicação. Após ser autenticado com sucesso, o utilizador pode monitorar e controlar o estado do agente por intermédio de páginas HTML ou WML.

Um efeito secundário da abordagem apresentada nesta secção é a possibilidade de utilizar o mecanismo HTTP como *proxy* para agentes SNMP existentes (Figura 4.15).

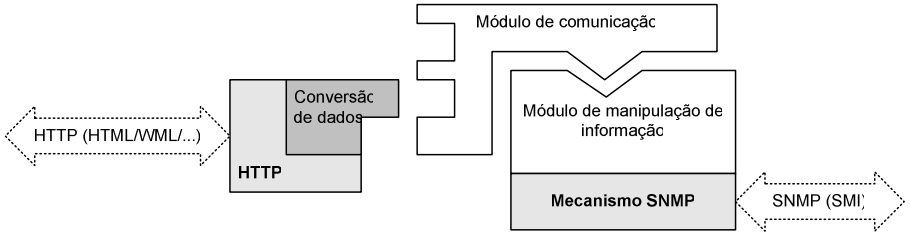


Figura 4.15 – Conversor (*proxy*) HTTP para SNMP.

O conversor pode, desta forma, apresentar páginas de Web para vários agentes SNMP. A diferença mais significativa é a operação de autenticação. Quando integrado no agente, o utilizador autentica-se por intermédio de um par <nome de utilizador, palavra chave>. No

conversor será necessário acrescentar informação relativa ao endereço do agente, ou seja, <endereço/nome do utilizador, palavra chave>. Esta abordagem permite manter o código XML e os documentos XSL inalterados além de disponibilizar a informação necessária para o conversor contactar agentes SNMP específicos.

Outra característica deste modelo prende-se com a necessidade de armazenar diferentes definições SMI (MIBs) para diferentes agentes. Neste sentido, será necessário construir uma lista que relacione os módulos correspondentes a cada agente. Esta lista irá permitir gerar páginas adaptadas a cada agente.

A necessidade de efectuar conversão de informação no agente levanta algumas questões relacionadas com desempenho. É necessário ter em conta que, inicialmente, é efectuada uma transformação de SMI para XML e, de seguida, este código é transformado pelo processador XSLT. O atraso acumulado (SMI->XML->HTML/WML/...) depende do tamanho do código SMI e, consequentemente, do documento XML. Para minimizar este inconveniente será necessário minimizar o código SMI processado. Neste sentido, o sistema processa apenas os objectos requisitados pelo cliente, página a página, ao invés de toda a estrutura.

Um outro aspecto associado ao processo de transformação XSLT é a adaptação ao tamanho do écran do cliente. Em pequenos écrans, tais como telefones móveis, a página deverá conter apenas a informação essencial, tal como o nome do nó, o OID e o valor correspondente. Para clientes HTTP vulgares, a limitação de espaço de visualização já não se faz sentir, pelo que se torna possível apresentar informação mais completa, como as descrições dos objectos SMI e os privilégios de acesso.

Para finalizar, o motor HTTP é baseado no servidor Jetty [Jetty] enquanto que a geração de documentos XML com a correspondente transformação são baseados nas ferramentas de interpretação XML [Xerces] e no processador XSLT Xalan [Xalan]. A aplicação é compilada recorrendo à ferramenta de compilação multiplataforma (Java make) [Ant].

4.3.4 Criação de novos mecanismos de comunicação

A Agent API contém mecanismos de comunicação para os protocolos SNMP, AgentX e HTTP. Futuramente poderá integrar, entre outros, mecanismos baseados em RMI e em CORBA. No entanto, o programador poderá definir novos protocolos ou métodos de acesso à informação de gestão (Figura 4.1).

Para o efeito necessita apenas de escrever uma classe derivada da classe `Engine`. Esta classe deve implementar os métodos apresentados na Tabela 4.2.

Supondo que o programador pretende definir um mecanismo de comunicação baseado em JMX do qual resulta uma classe denominada `JMXEngine`. Para iniciar o agente com este novo mecanismo de comunicações terá de o invocar da seguinte forma:

```
$ java -Dagentapi.engine=JMXEngine\
-Dagentapi.engine.config.file=engine.properties\
Agente
```

A propriedade `agentapi.engine` especifica o nome da classe que deve ser usada como mecanismo de comunicação. Para utilizar mais que um mecanismo de comunicação será necessário especificar várias classes separadas por vírgulas. A propriedade `agentapi.engine.config.file` indica o ficheiro com as propriedades específicas dos mecanismos de comunicação. Estas serão definidas por intermédio do método `setProperties`. Esta abordagem evita modificar ou recompilar o código do agente.

Tabela 4.2 – Métodos que os mecanismos de comunicação terão de implementar.

Método	Descrição
`void addAgent(Agent)`	Para registar os agentes de forma a poderem receber mensagens da rede.
`void close()`	Termina a sessão de comunicação.
`MessageListener createAgentListener()`	Constrói e devolve um objecto para receber mensagens geradas no agente. Estas serão posteriormente enviadas para a rede.
`void open()`	Prepara o mecanismo para receber ligações.
`void removeAgent(Agent)`	Remove o agente do mecanismo. A partir deste momento deixa de ser possível receber mensagens.
`void setProperties(Properties)`	Define as propriedades específicas do mecanismo de comunicação, por exemplo, porta de escuta, nome do utilizador, etc.

4.4 Módulos de extensão

A Agent API, além dos módulos essenciais para o funcionamento e desenvolvimento de agentes de gestão prevê também um módulo de extensão onde enquadra componentes relacionados com tarefas de gestão (Figura 4.1).

Os componentes não são fundamentais para o funcionamento do agente e, como tal, podem ser removidos sem afectar o seu funcionamento. No entanto, providencia serviços genéricos que podem ser úteis pelo que poderão ser utilizados pelo agente ou pelos mecanismos de comunicação. Um exemplo é o MIB Parser, utilizado pelo mecanismo de comunicação HTTP. Outros componentes definem mecanismos de persistência de informação de gestão e referências com base em URLs para recursos SNMP.

4.4.1 MIB Parser

O componente MIB Parser permite ler e interpretar definições de módulos MIB, geralmente descritos em SMI. Este irá permitir que as aplicações baseadas na Agent API possam, de forma uniforme, construir um modelo da MIB, ou seja, os objectos existentes, a sua sintaxe e outras características desse módulo.

Do ponto de vista do programador, esta ferramenta vem acelerar o processo de desenvolvimento de aplicações pois fornece acesso imediato à informação contida num módulo MIB. De outra forma seria necessário desenvolver um módulo específico, com a descrição de todas as particularidades do módulo, para integrar na aplicação (Figura 4.1).

O interpretador SMI da Agent API constrói uma estrutura dinâmica de objectos Java com base na descrição em ficheiro dos módulos MIB. A Figura 4.16 ilustra a utilização do interpretador para construir uma representação gráfica da estrutura de objectos de gestão definidos num módulo MIB. Cada nó, de acordo com a sua natureza, é associado a um ícone para mais facilmente visualizar o seu papel:

- Módulo MIB
- Nó apresentando os seus descendentes (aberto)
- Nó escondendo os seus descendentes (fechado)

- Objecto com permissões do tipo leitura e escrita (`read-write`)
- Objecto com permissões do tipo leitura (`read-only`)
- Objecto do tipo tabela, ou seja, que contém um objecto do tipo índice
- Objecto índice
- Objecto coluna com responsabilidade de indexação de tabelas definido localmente
- Objecto coluna com responsabilidade de indexação de tabelas definido anteriormente
- Objecto com permissões do tipo leitura e criação (`read-create`)
- Convenções textuais
- Notificação

Nem todos os módulos MIB definem todos os objectos anteriormente descritos, pelo que alguns ícones podem não surgir na árvore de objectos.

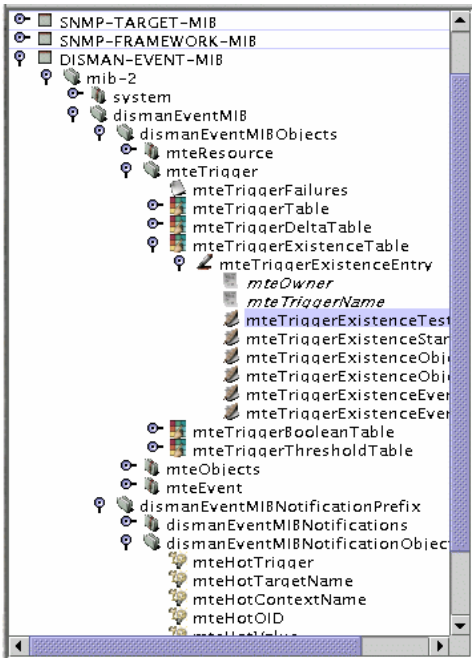


Figura 4.16 – Árvore SMIv2 correspondente à *Event MIB*.

A estrutura da MIB obedece a uma sintaxe e a uma semântica bem definidas. A primeira secção descreve os objectos importados de outros módulos (`MibImports`). De seguida surge uma secção relacionada com a organização responsável pela definição da MIB (`MibIdentity`) e o conjunto de nós, tabelas e outros objectos de gestão. Além destes, o módulo MIB define notificações (`MibTrap` e `NotificationType`). Por fim, a MIB apresenta na secção de *compliance*, ou seja, uma lista dos objectos que terão obrigatoriamente de ser considerados para a implementação e os que poderão não ser implementados. As MIB podem também definir novos tipos de dados, as convenções textuais. O diagrama de classes representa todas estas secções mantendo a informação comum nas classes de nível hierárquico mais elevado (Figura 4.17).

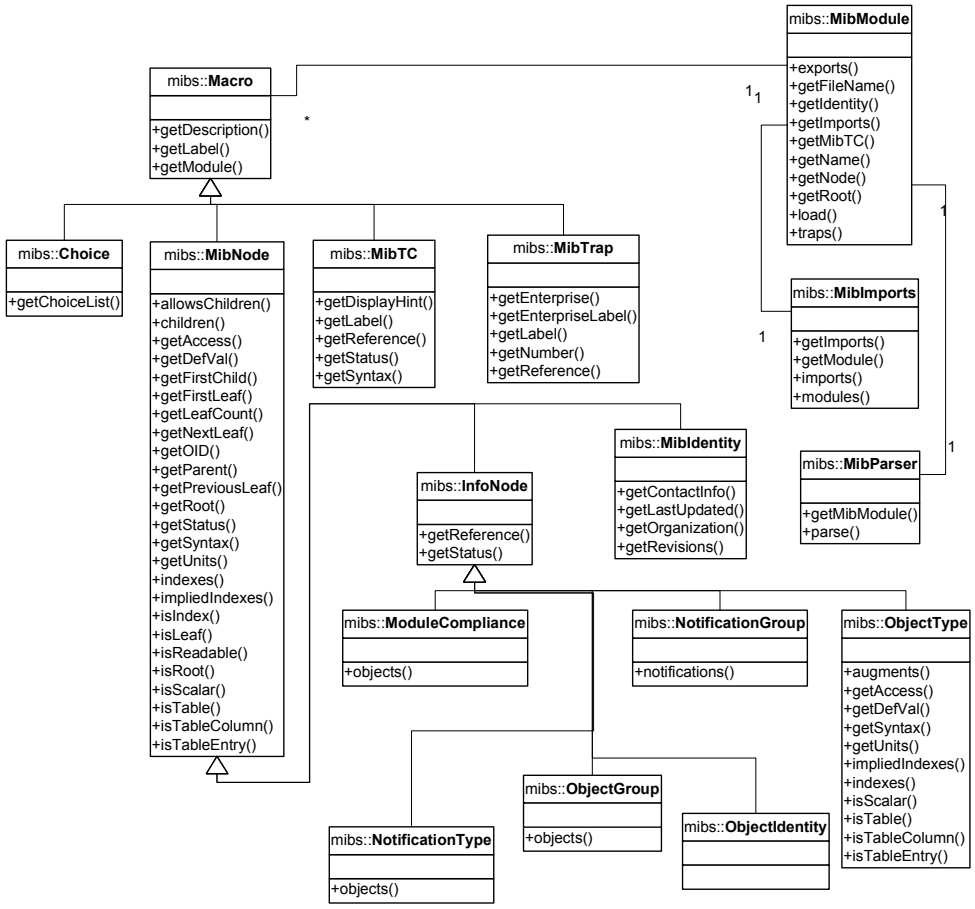


Figura 4.17 – Diagrama de classes do MIB Parser.

O elemento chave do mecanismo é a classe `MibParser` cuja função é gerar a estrutura de objectos de acordo com a descrição encontrada no ficheiro MIB. Cada ficheiro é representado por um objecto do tipo `MibModule` que, por sua vez, contém mais `MibModule`s para módulos associados às primitivas SMI do tipo `IMPORT`. A interpretação do exemplo abaixo dará origem à criação de um novo objecto do tipo `MibModule` para cada módulo referenciado (*SNMPv2-SMI*, *SNMPv2-TC*, *SNMPv2-CONF*, *SNMPv2-MIB*, *SNMP-TARGET-MIB* e *SNMP-FRAMEWORK-MIB*).

```
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, Unsigned32,
    NOTIFICATION-TYPE, Counter32,
    Gauge32, mib-2, zeroDotZero         FROM SNMPv2-SMI
    TEXTUAL-CONVENTION, RowStatus,
    TruthValue                  FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP,
    NOTIFICATION-GROUP              FROM SNMPv2-CONF
    sysUpTime                   FROM SNMPv2-MIB
    SnmpTagValue                FROM SNMP-TARGET-MIB
    SnmpAdminString             FROM SNMP-FRAMEWORK-MIB;
```

Os objectos descritos nos módulos MIB são representados por objectos do tipo `MibNode`. Estes podem ser:

- Nós – representam ramificações da árvore. Contêm apenas informação acerca do seu nome, OID e dos seus descendentes.
- Folhas – representam objectos que podem ser colunas de uma tabela ou escalares. Além do seu nome e OID contêm informação relativa à sintaxe (tipo SMI), unidades (`UNITS`), acesso (`read-write`, `read-only`, `read-create`, `not-accessible`) e estado entre outros.
- Índices – representam os objectos índice de uma tabela.
- Tabelas – representam nós que possuem índices.

Além da estrutura apresentada, o interpretador SMIv2 mantém um registo do tipo de dados básico correspondente a um determinado objecto. Por exemplo, se um objecto é descrito no módulo MIB como sendo do tipo `TDomain` [RFC2579], o seu tipo básico é `OBJECT IDENTIFIER`.

Para utilizar o mecanismo de interpretação de módulos MIB é necessário criar um objecto do tipo `MibModule` da seguinte forma:

```
...
// Carregar o módulo MIB
MibModule expressionMib = MibModule.load("DISMAN-EXPRESSION-MIB");
...
// Consultar a estrutura de nós
MibNode r = expressionMib.getRoot();
for(Enumeration i=r.children(); i.hasNextElement; ) {
  MibNode child = (MibNode)i.nextElement();
  // Efectuar processamento sobre o nó
  ...
}
...
```

4.4.2 Persistência de informação de gestão

O aumento de complexidade e quantidade de módulos MIB, bem como de objectos que estes definem, sobrecarrega os agentes SNMP com mais e melhores funcionalidades, nomeadamente a necessidade de disponibilizar uma forma de persistência de informação. É indesejável, tanto do ponto de vista de utilização como do ponto de vista de ocupação de largura de banda, que sempre que um determinado agente reinicia o seu funcionamento seja necessário reconfigurar os valores previamente definidos. Uma implementação robusta não deverá forçar a aplicação de gestão a efectuar reconfigurações cada vez que o agente é reinicializado. Desta forma, é importante que os agentes sejam capazes de armazenar os valores dos objectos que implementam, independentemente do meio disponível para o efeito – disco local, sistema de ficheiros de rede ou outro (Figura 4.1).

Tipicamente, uma característica do modelo de informação do SNMP é a necessidade de criar ou modificar vários objectos para a definição de uma única tarefa. Esta situação é semelhante à programação em *Assembly* em que são necessárias várias instruções para definir uma única instrução de nível mais elevado.

Já vários autores abordaram esta problemática, sendo a maioria das soluções baseada em APIs de nível superior que providenciam a agregação e correlação de operações SNMP [Oliveira94][Goldszmidt98][Lopes00a]. Existem, no entanto, situações em que persistência a um nível mais baixo é desejada, de forma a que os agentes SNMP, com recursos físicos adequados, possam armazenar e recuperar informação de funcionamento, sem que dependam de contacto com a aplicação de gestão.

Os módulos MIB, geralmente, não apresentam regras ou normas para o formato dos dados nem para o tipo de suporte à persistência de informação, que poderá variar de agente para agente. A linguagem XML poderá ser utilizada para definir de forma explícita como os objectos serão configurados após inicialização do agente (Figura 4.18).

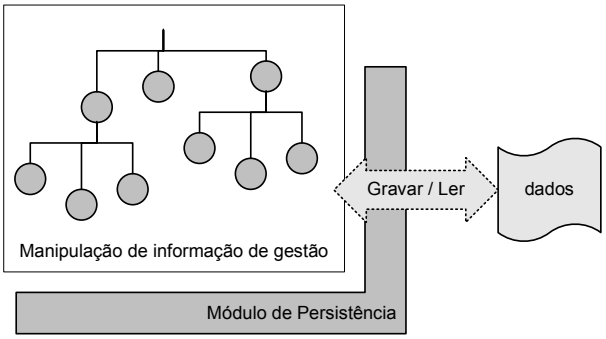


Figura 4.18 – Persistência de agentes SNMP.

Uma abordagem ao problema da persistência é definir uma correspondência entre os comandos SNMP, utilizados para ler e escrever informação em agentes, e uma sintaxe baseada em XML para fixar a informação em ficheiro. As orientações definidas na SMI descrevem a estrutura e os esquemas para a definição de informação de gestão, embora nada acrescentem sobre as mensagens SNMP, descritas em outros documentos [RFC2578].

O SNMP define oito mensagens SNMP e os respectivos PDUs [RFC1905]: `get-request`, `get-next-request`, `get-bulk-request`, `response`, `set-request`, `inform-request`, `snmpV2-trap` e `report`, apesar de a utilização e semântica deste último não estejam definidas. Dependendo do papel desempenhado pela entidade SNMP (aplicação de gestão, agente, procurador, etc.), poderão ser utilizados diferentes conjuntos de mensagens (Figura 4.19).

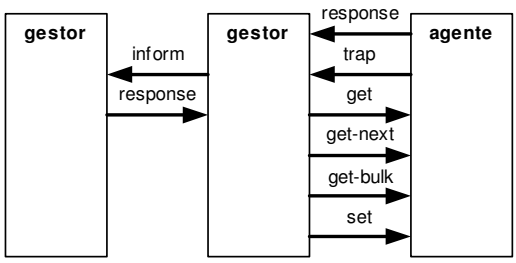


Figura 4.19 – Troca de mensagens entre entidades SNMP.

Todos os PDUs apresentam a mesma estrutura lógica à excepção de `get-bulk-request` que, apesar de ser semelhante, difere em dois campos (Figura 4.20).

O campo *request-id* é utilizado por aplicações SNMP para distinguir pedidos e responder de acordo. Um valor diferente de zero no campo *error-status* indica a ocorrência de algum erro e, neste caso, *error-index* assinala o número de ordem da variável onde este aconteceu.

O comando `get-bulk-request` é utilizado para requisitar vários parâmetros com apenas uma mensagem. Neste caso, o campo de *non-repeaters* (N) indica o número de variáveis que deverão devolver um único valor. Estes são obtidos por intermédio de operações *get-next*. Para cada variável assinalada além de N, serão obtidos *max-repetitions* (M) valores obtidos de igual forma por intermédio de operações `get-next`. Isto significa que a resposta a um comando deste tipo

poderá conter até N+(M*R) valores, onde R é o número restante de variáveis, ou seja, o número total de variáveis menos N.

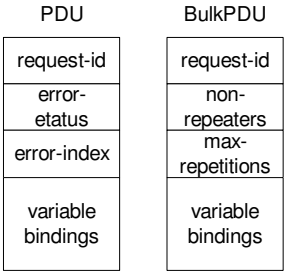


Figura 4.20 – PDUs definidos para mensagens SNMP.

Resumidamente, existem vários tipos de comandos SNMP com um número inconstante de variáveis. Os comandos podem ser agrupados para constituir uma operação ou tarefa. Será esta a abordagem seguida para resolver o problema da persistência de informação de gestão: descrever grupos de comandos SNMP em ficheiro.

A linguagem XML permite descrever informação estruturada por intermédio de etiquetas específicas [XML98]. O agrupamento de etiquetas define um documento que pode ser utilizado para troca de informação independentemente da plataforma, linguagem de programação ou objectivo da aplicação. Estas características tornam-na ideal para representar sequências de comandos SNMP e as respectivas variáveis e valores.

O documento XML deve iniciar-se com a etiqueta <snmp> e os respectivos parâmetros:

```
<snmp version="3" user="senior" authProtocol="MD5" authPassword="senior"
  privPassword="senior">
...
</snmp>
```

No âmbito do SNMP, as variáveis são referênciadas por OID ou por um nome, definidos no ficheiro MIB. Para efectuar a correspondência entre os dois será necessário fazer uma referência ao ficheiro correspondente. Para o efeito é criada uma etiqueta `<mib>`:

```
<mib name="RFC1213-MIB" location="file:/usr/local/mibs"/>
```

Seguindo um conceito semelhante ao das variáveis em programação, a etiqueta `<property>` permite definir parâmetros válidos no âmbito do documento, de forma a facilitar a reutilização de código XML. O nome da propriedade precedida pelo símbolo '`$`' é substituído pelo valor definido na etiqueta `<property>` correspondente:

```
<property name="trapObject" value=".1.3.6.1.2.4.5.6"/>
<property name="otherObject" value=".1.3.6.1.2.4.5.7"/>
<property name="destination" value="192.168.0.1:162"/>
...
<varBind name="someOID" oid="$trapObject"/>
...
```

O agrupamento de comandos é efectuado por intermédio da etiqueta `<task>`:

```
<task name="myTrap">
...
</task>
```

Cada tarefa pode conter vários comandos que identifica sob um nome comum. Os comandos podem ser:

```
<get>…</get>
<getNext>…</getNext>
<getBulk nonrep="N" maxrep="M">…</getBulk>
<set>…</set>
<inform>…</inform>
<response>…</response>
```

A lista de variáveis é definida como uma sequência de etiquetas `<varBind>` com três parâmetros: `name`, `oid` e `value`. Este último poderá ser omitido em caso de não haver nenhum valor definido.

```
<trap version="2" destination="$destination">
  <!-implicit VarBinds
  <varBind name="sysUpTime" oid=".1.3.6.1.2.1.1.3.0"/>
  <varBind name="snmpTrapOID" oid=".1.3.6.1.6.3.1.1.4.1"/>
  -->
  <varBind name="someOID" oid="$trapObject"/>
</trap>
```

Por comodidade, quando um comando transporta apenas uma única variável, pode ser representada de forma resumida, substituindo, portanto, a forma mais extensa.

```
<trap version="2" destination="$destination" name="someOID" oid="$otherObject"/>
```

Finalmente, é também importante prever um mecanismo que permita reutilizar tarefas definidas em outros documentos.

```
<runTask name="someTask" document="file:/usr/local/operation/someOperation.xml"/>
```

O documento DTD (*Data Type Definition*) que descreve a sintaxe dos ficheiros XML de persistência pode ser consultado no Anexo B.

Como exemplo de definição de uma expressão, considere-se o cálculo da taxa de utilização de ligação. Este valor, apresentado em percentagem, será o rácio da soma do número total de bits enviados e recebidos por unidade de tempo e a largura de banda disponível:

$$\text{utilização}\,(\%) = \frac{\text{total bits enviados} + \text{total bits recebidos}}{\text{largura de banda}}$$

Em termos de objectos MIB, será necessário introduzir a constante 8 (considerando um octeto composto por 8 bits) e multiplicar o resultado por 100 para que o valor resultante seja um inteiro:

$$\text{utilização} = \frac{(\text{ifInOctets} + \text{ifOutOctets}) \times 8 / \text{sysUpTime}}{\text{ifSpeed}} \times 100 = \frac{(\text{ifInOctets} + \text{ifOutOctets}) \times 800}{\text{sysUpTime} \times \text{ifSpeed}}$$

Esta expressão é complementada por uma outra que verifica se os objectos correspondem a ligações físicas, ou seja, hardware. Para o efeito, é utilizado o valor de *ifConnectorPresent*, que representa verdade por `true` ('1') e falso por `false` ('2'):

$$\text{ifConnectorPresent} == \text{true}$$

O resultado da expressão condicional associado ao objecto *expObjectConditional* é utilizado para validar o valor obtido para *ifInOctets* e, consequentemente, o resultado final. Por outras palavras, ao invalidar um parâmetro, a expressão não será calculada.

A definição destas expressões num módulo Expression MIB [RFC2982] requer a criação de 31 objectos [Lopes00c]. Através da utilização de macros SNMP em XML é possível, por exemplo, definir uma tarefa para a expressão condicional composta por dois comandos `set` e outra tarefa para a definição da expressão de taxa composta por cinco comandos `set`. As variáveis de cada comando contêm a informação necessária para criar ou modificar o objecto correspondente. O exemplo seguinte ilustra o procedimento.

```
<snmp community="public" writeCommunity="private">
  <mib name="DISMAN-EXPRESSION-MIB" location="file:/usr/local/mibs"/>
  <mib name="RFC1213-MIB" location="file:/usr/local/mibs"/>

  <task name="hard_limit">
    <set>
```

```
      <varbind name="expExpression"
        oid="expExpression.2.'me'.4.'hard'" value="$1==1"/>
      <varbind name="expExpressionValueType"
        oid="expExpressionValueType.2.'me'.4.'hard'" value="unsigned32"/>
      <varbind name="expExpressionRowStatus"
        oid="expExpressionRowStatus.2.'me'.4.'hard'" value="active"/>
    </set>

    <set>
      <varbind name="expObjectID" oid="expObjectID.2.'me'.4.'hard'.1"
        value="ifConnectorPresent"/>
      <varbind name="expObjectWildcard"
        oid="expObjectWildcard.2.'me'.4.'hard'.1" value="true"/>
      <varbind name="expObjectSampleType"
        oid="expObjectSampleType.2.'me'.4.'hard'.1" value="absoluteValue"/>
      <varbind name="expObjectRowStatus"
    oid="expObjectRowStatus.2.'me'.4.'hard'.1" value="active"/>
    </set>
  </task>

  <task name="line_utilization">
    <set>
      <varbind name="expExpression"
oid="expExpression.2.'me'.4.'util'" value="($1+$2)*800/$4/$3"/>
      <varbind name="expExpressionValueType"
oid="expExpressionValueType.2.'me'.4.'util'" value="integer32"/>
      <varbind name="expExpressionDeltaInterval"
oid="expExpressionDeltaInterval.2.'me'.4.'util'" value="6"/>
      <varbind name="expExpressionRowStatus"
oid="expExpressionRowStatus.2.'me'.4.'util'" value="active"/>
    </set>

    <set>
      <varbind name="expObjectID1"
oid="expObjectID.2.'me'.4.'util'.1"  value="ifInOctets"/>
      <varbind name="expObjectWildcard1"
oid="expObjectWildcard.2.'me'.4.'util'.1"  value="true"/>
      <varbind name="expObjectSampleType1"
oid="expObjectSampleType.2.'me'.4.'util'.1"  value="deltaValue"/>
      <varbind name="expObjectConditional1"
oid="expObjectConditional.2.'me'.4.'util'.1"
value=" expValueUnsigned32Val.4.'hard'.0.0"/>
      <varbind name="expObjectConditionalWildcard1"
oid="expObjectConditionalWildcard.2.'me'.4.'util'.1"  value="true"/>
      <varbind name="expObjectDiscontinuityID1"
oid="expObjectDiscontinuityID.2.'me'.4.'util'.1"
value="ifCounterDiscontinuityTime"/>
      <varbind name="expObjectDiscontinuityIDWildcard1"
oid="expObjectDiscontinuityIDWildcard.2.'me'.4.'util'.1"  value="true"/>
      <varbind name="expObjectRowStatus1"
oid="expObjectRowStatus.2.'me'.4.'util'.1"  value="active"/>
    </set>

    <set>
      <varbind name="expObjectID2"
oid="expObjectID.2.'me'.4.'util'.2"  value="ifOutOctets"/>
      <varbind name="expObjectWildcard2"
oid="expObjectWildcard.2.'me'.4.'util'.2"  value="true"/>
      <varbind name="expObjectSampleType2"
oid="expObjectSampleType.2.'me'.4.'util'.2"  value="deltaValue"/>
      <varbind name="expObjectRowStatus2"
oid="expObjectRowStatus.2.'me'.4.'util'.2"  value="active"/>
    </set>

    <set>
      <varbind name="expObjectID3"
oid="expObjectID.2.'me'.4.'util'.3"  value="ifSpeed"/>
      <varbind name="expObjectWildcard3"
oid="expObjectWildcard.2.'me'.4.'util'.3"  value="true"/>
      <varbind name="expObjectSampleType3"
oid="expObjectSampleType.2.'me'.4.'util'.3"  value="absoluteValue"/>
      <varbind name="expObjectRowStatus3"
oid="expObjectRowStatus.2.'me'.4.'util'.3"  value="active"/>
    </set>

    <set>
      <varbind name="expObjectID4"
oid="expObjectID.2.'me'.4.'util'.4"  value="sysUpTime.0"/>
```

```
      <varbind name="expObjectWildcard4"
oid="expObjectWildcard.2.'me'.4.'util'.4"  value="false"/>
      <varbind name="expObjectSampleType4"
oid="expObjectSampleType.2.'me'.4.'util'.4"  value="deltaValue"/>
      <varbind name="expObjectRowStatus4"
oid="expObjectRowStatus.2.'me'.4.'util'.4"  value="active"/>
    </set>

  </task>
</snmp>
```

Os documentos XML apresentam uma forma adequada para descrever informação estruturada como, por exemplo, SMI, ou operações SNMP. Esta última abordagem permite agrupar comandos e instruções em tarefas podendo, portanto, ser definida uma biblioteca de funções de alto nível.

A arquitectura do módulo de persistência resulta num conjunto de classes que representam os conceitos definidos anteriormente e que tornam a sua utilização modular (Figura 4.21).

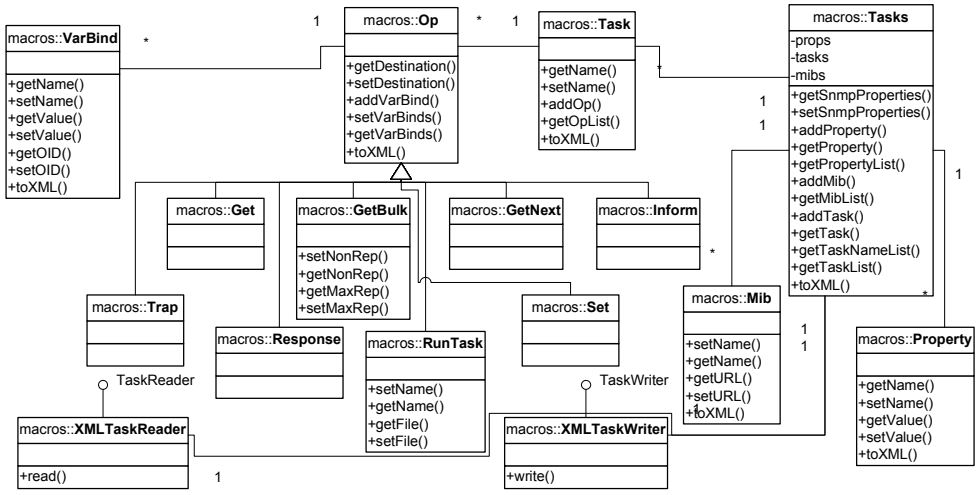


Figura 4.21 – Diagrama de classes para persistência de informação e macros SNMP.

A estrutura apresentada no DTD para a persistência de informação de gestão (Anexo B) define o elemento `<task>` para agrupar operações. Cada operação representa uma mensagem SNMP e constitui uma classe derivada de `Op`. A classe `RunTask`, apesar de também ser derivada de `Op`, não representa uma mensagem SNMP. A função desta operação é invocar tarefas (`<task>`) definidas em outros documentos.

Cada comando pode conter vários objectos do tipo `VarBind` que associam OIDs aos respectivos valores. De lembrar que em operações de consulta o campo correspondente ao valor encontra-se normalmente vazio.

O agrupamento de tarefas (`<task>`) assim como a definição de propriedades (`<property>`) e de especificação de módulos MIB (`<mib>`) é da responsabilidade da classe `Tasks`. Toda a estrutura pode ser guardada ou criada por intermédio de classes que implementam as interfaces `TaskReader` e `TaskWriter`, respectivamente. No esquema apresentado, objectos do tipo `XMLTaskReader` e `XMLTaskWriter` transformam a estrutura de tarefas em XML ou vice-versa, construindo o mecanismo de persistência.

Para a execução de macros SNMP, a classe `SnmpMacrosWriter` (não representada na figura), que implementa `TaskWriter`, recebe um objecto do tipo `Tasks` e gera mensagens SNMP. Pode ser construído um conversor de XML para mensagens SNMP da seguinte forma:

```
...
// Parâmetros associados ao mecanismo SNMP
SnmpProperties props = new SnmpProperties();
props.setCommunity("public");
props.setVersion(SnmpConstants.SNMPv2c);

TaskReader reader = new XMLTaskReader("/usr/local/macros/macros.xml");
SnmpMacrosWriter writer = new SnmpMacrosWriter(props);
writer.setHost("192.168.0.1");      // Destino
writer.write(reader.read());        // Ler XML e gerar mensagens SNMP
...
```

4.4.3 Esquema de identificação uniforme de recursos SNMP

À semelhança da identificação de recursos na Internet, foi adicionada à Agent API a funcionalidade para interpretar e identificar recursos SNMP com base numa sintaxe do tipo URL. Apesar de não ser muito utilizada actualmente no âmbito da gestão SNMP o conceito é suficientemente poderoso e flexível para que possa trazer vantagens em termos de API e de utilização em aplicações de gestão de redes.

A informação, os serviços ou, genericamente, qualquer dos recursos que diariamente se acrescentam à Internet tem subjacente um espaço de nomes, parcialmente baseado no DNS. A identificação de recursos é efectuada por linhas de texto denominadas URI (*Uniform Resource Identifiers*) [RFC2396]. Esta cadeia de caracteres concentra a informação necessária para consultar ou, eventualmente, modificar a informação contida num determinado recurso.

Os recursos podem ser físicos, como um processador, memória ou dispositivos de armazenamento de massa, ou lógicos, como um servidor de páginas de Internet ou um agente de gestão de redes. Os recursos são localizados por intermédio de referências que assinalam a sua localização e natureza. Por exemplo, a caixa de correio electrónico de um determinado utilizador é assinalada segundo o esquema mailto:<nome>@<endereço>.

Apesar das diferenças intrínsecas a cada recurso, a sua referenciação é regida por conceitos comuns como o nome ou o endereço. Este conjunto de características permite utilizar uma sintaxe uniforme para os referenciar independentemente da natureza dos recursos.

A uniformidade de representação de referências permite utilizar identificadores para recursos diferentes num mesmo contexto. No contexto da Internet, por exemplo, os recursos FTP, HTTP ou NEWS são acessíveis por intermédio de uma ferramenta comum – o navegador de Internet. O recurso é especificado no campo de endereço por intermédio do URI e de acordo com a sua semântica é invocada a ferramenta adequada para o seu processamento e apresentação.

O conceito URI foi definido pelo grupo de redes do IETF como a especificação de referências universais para recursos físicos ou lógicos [RFC2396]. A sua sintaxe é suficientemente genérica para que possa ser aplicado na identificação de um grande conjunto de recursos, nomeadamente páginas de Internet, endereços de correio electrónico, grupos de discussão e livros, entre muitos outros. No âmbito do IANA existem actualmente esquemas de URL para 36 serviços [URISchemes]. Apesar desta adesão continua por definir uma proposta que contemple o SNMP.

Sintaxe genérica dos URI

No seu nível mais abstracto um URI tem o seguinte formato:
```
[esquema:]parte-dependente-do-esquema[#fragmento]
```

Os símbolos '[' e ']' indicam secções opcionais enquanto que os símbolos ':' e '#' delimitam as várias secções. Os URI dizem-se absolutos, quando é indicado o seu esquema, e relativos no caso contrário. Neste último caso, o URI relativo é construído com base num URI absoluto, de forma a completar a informação em falta.

Os URI podem também ser classificados como hierárquicos. Neste caso, a parte dependente do esquema começa com o símbolo '/' e tem o seguinte formato:
```
[esquema:][//autoridade][caminho][?consulta][#fragmento]
```

A autoridade representa o topo hierárquico de um esquema de identificação, como [utilizador@]endereço[:porto] (de notar que o símbolo '/' precede a autoridade no formato acima). O caminho contém dados que identificam um recurso único no contexto da autoridade em causa. A secção consulta contém informação que irá ser entregue e interpretada pelo recurso. O fragmento consiste na informação adicional a ser interpretada do lado do utilizador após a operação de consulta ser realizada com sucesso. Teoricamente não faz parte do URI uma vez que não é comunicado ao recurso, mas é frequentemente associado, para controlo de marcadores em páginas html, por exemplo.

Os URI urn:isbn:096139210x, que assinala um livro, e mailto:pemz@mail.hosting.pt, que especifica um endereço de correio electrónico, são não hierárquicos ou opacos, de acordo com a nomenclatura definida nas normas.

O URI http://www.ics.uci.edu/pub/ietf/uri/#Related é absoluto e hierárquico, uma vez assinala o esquema http e a parte dependente do esquema começa com o símbolo '/'. A autoridade representa um nome de Internet (www.ics.uci.edu) e o caminho assinala o recurso /pub/ietf/uri/, único no âmbito desse servidor. O fragmento, utilizado neste caso pelo navegador de Internet para deslizar a página verticalmente, não é comunicado ao servidor e contém a informação "Related".

A Tabela 4.3 apresenta alguns exemplos de URI já normalizados.

Tabela 4.3 – Exemplos de URIs.

Modelo	URI
HTTP	http://www.det.ua.pt
FTP	ftp://jprs@ftp.univ.pt/private/projectX/
XMLORG	urn:xmlorg:objects:schema:xmlschema:xcatalog
NFS	nfs://server/a/b
LDAP	ldap://ldap.itd.umich.edu/o=University%20of%20Michigan,c=US
MAIL	mailto:rlopes@ipb.pt

URL para SNMP

De acordo com a especificação da versão SNMPv3 a funcionalidade de gestão associada às entidades SNMP é definida por agrupamento de aplicações. Estas podem ser de quatro tipos: geradoras de comandos, geradoras de notificações, receptoras de comandos e receptoras de notificações. Como tal, um agente de gestão SNMP é constituído por um gerador de notificações e um receptor de comandos enquanto que as aplicações de gestão são constituídas por um receptor de notificações e um gerador de comandos.

As entidades SNMP são identificadas univocamente por um número designado por `snmpEngineID`. Cada entidade poderá conter diversos contextos, únicos em cada entidade. Para identificar cada objecto são necessários quatro parâmetros: o identificador do mecanismo SNMPv3 (`snmpEngineID`), o nome do contexto (`contextName`), o identificador de objecto (OID – ex. *ifDescr*) e o identificador de instância (ex. '1'). No caso do SNMPv1, a consulta é efectuada apenas indicando o identificador de objecto e o identificador de instância, o que resulta numa maior simplicidade de acesso mas também numa menor flexibilidade devido à inexistência de contextos: não admite duplicação de OIDs num único agente.

A comunicação entre entidades SNMP rege-se por um conjunto de parâmetros que asseguram a correcta identificação dos intervenientes, a segurança da comunicação e o processamento das mensagens. Um dos aspectos fundamentais é a segurança. De acordo com a versão do protocolo podem ser utilizados vários modelos de segurança. Para as versões SNMPv1 e SNMPv2c a privacidade é inexistente e a autenticação é efectuada com base num nome de comunidade. A versão SNMPv3 preve já a possibilidade para cifrar mensagens, o que garante a privacidade da comunicação, e modelos de autenticação com base em nomes de utilizadores e correspondentes palavras chave.

Resumidamente, para que a comunicação entre entidades SNMP seja possível, é necessário indicar o protocolo a ser utilizado. Este consiste actualmente numa de três versões: SNMPv1, SNMPv2c ou SNMPv3. Associado ao protocolo encontra-se, obrigatoriamente, o endereço do destinatário bem como os parâmetros relacionados com a segurança. Estes últimos poderão consistir numa cadeia de caracteres representativa de uma comunidade para as versões SNMPv1 e SNMPv2c ou em três parâmetros para o SNMPv3:

- Modelo de segurança – SNMPv1, SNMPv2c, USM (*User Security Model*).
- Nome de utilizador – cadeia de caracteres.
- Nível de segurança – sem autenticação nem privacidade (`noAuthNoPriv`), com autenticação e sem privacidade (`authNoPriv`), com autenticação e privacidade (`authPriv`).

Adicionalmente, será necessário passar ao módulo de segurança as chaves associadas aos algoritmos de autenticação e de privacidade.

O protocolo poderá também receber indicação do tempo que deverá esperar por uma resposta (*timeout*) e do número de tentativas que deverá efectuar antes de desistir de contactar o destinatário (*retry count*)

Finalmente, para aceder ao recurso pretendido, ou seja, à informação de gestão, será necessário assinalar o contexto, o OID e a instância.

Com base nestas parâmetrizações desenvolveu-se uma proposta de URL para SNMP que apresenta toda a informação necessária para a comunicação entre entidades SNMP num formato compatível com o conceito URI.

De acordo com a especificação, os URI têm restrições em termos de símbolos ou caracteres que, devido à sua extensão não são apresentados neste documento. Para mais detalhes poderá ser consultado o documento [RFC2396].

Genericamente, a sintaxe proposta é a seguinte:

```
snmpurl          = esquema "://" [segurança "@"] [end_porto] ["/"
                   [recurso] ["?" [operação] ["?" [versão] ["?"
```

```
                [contexto]]]]] ["#" parâmetros]
esquema         = "snmp"
segurança       = [comunidade] |
                  [utilizador [":" autenticação [":" privacidade]]]
comunidade      = comunidade da secção 3.2.5 de [RFC1157]
utilizador      = nome de utilizador da secção 2.1 de [RFC2574]
autenticação    = elemento_a ?("&" elemento_a)
elemento_a      = "auth=" protocolo_a | "pass=" chave
privacidade     = elemento_p ?("&" elemento_p)
elemento_p      = "priv=" protocolo_p | "pass=" chave
protocolo_a     = protocolo de autenticação da secção 1.4.2 de [RFC2574]
protocolo_p     = protocolo de privacidade da secção 1.4.3 de [RFC2574]
chave           = cadeia de caracteres
end_porto       = endereço [":" porto]
endereço        = endereço IP ou nome associado
porto           = número inteiro
recurso         = OID ["/" instância]
OID             = sequência de inteiros separados por '.' |
                  nome associado
instância       = sequência de inteiros separados por '.'
operação        = "op=" nome_op [op_params]
nome_op         = nome da operação SNMP. Actualmente
                  ("get" | "getNext" | "set" | "trap" | "response" |
                  "getBulk" | "inform" | "trap2")
op_params       = op_param *("&" op_param)
op_param        = ("value=" valor | "maxrep=" valor | "nonrep=" valor)
valor           = cadeia de caracteres
versão          = "v1" | "v2c" | "v3"
contexto        = contexto da secção 3.3.1 de [RFC2571]
parâmetros      = parâmetro *("&" parametro)
parâmetro       = "timeout=" inteiro | "retries=" inteiro
```

Este formalismo constitui um modelo universal de representação de informação de gestão que pode ser utilizado quer na Agent API quer em aplicações desenvolvidas sobre outras API. Apresentam-se e discutem-se de seguida alguns exemplos de representações nas diferentes versões do SNMP.

Para SNMPv1 pode ser, por exemplo:

snmp://private@sw1.estig.ipb.pt/sysContact/0?op=set&value=Rui%20Lopes

snmp://public@nms.estig.ipb.pt:161/sysUpTime?op=getNext

Para SNMPv2c poderão ser do tipo:

snmp://private@sw1.estig.ipb.pt/sysContact/0?op=set&value=Rui%20Lopes?v2c

snmp://public@nms.estig.ipb.pt:161/sysUpTime?op=getNext?v2c

E para SNMPv3:

snmp://rlopes@sw1.estig.ipb.pt/sysContact/0?op=set&value=Rui%20Lopes?v3

snmp://guest@nms.estig.ipb.pt:161/sysUpTime?op=getNext?v3?router

O prefixo '%' é utilizado para representar caracteres especiais. A combinação "%20" corresponde ao espaço.

De notar que apesar de ser possível indicar chaves e parâmetros de segurança directamente no URL este procedimento não é recomendado devido às falhas que poderão resultar da utilização de sequências de caracteres "em claro". Este inconveniente pode ser resolvido através de interacção com o utilizador (a aplicação apresenta uma janela específica para a introdução de informação de segurança) ou, no caso de não haver interacção com o utilizador, armazena a informação de segurança de forma ilegível (cifrada) em ficheiros associados de configuração. Esta última opção é actualmente utilizada em algumas aplicações SNMPv3 no papel de agentes [NetSNMP].

A aplicação destes conceitos à Agent API traduz-se no desenvolvimento de classes e mecanismos que interpretam os SNMP URL e extraem a informação adequada, nomeadamente, versão, endereço, porto, parâmetros de segurança e OID, entre outros.

4.5 Conclusões

O desenvolvimento de agentes de gestão requer, por um lado, que o agente possa comunicar e que, por outro lado, possa aceder à informação de gestão dos componentes de rede. A crescente diversidade, a introdução de nova tecnologia e o aumento de dimensão das redes de comunicação vem dificultar a tarefa do programador, uma vez que pode ser necessário adaptar o agente a novos componentes e a novos mecanismos de comunicação.

Para facilitar o desenvolvimento de agentes, foi criada uma interface de programação de aplicações, denominada Agent API, que agrupa e encapsula os mecanismos essenciais de funcionamento do agente. Para criar um agente, é apenas necessário invocar serviços sobre a API e detalhar o mecanismo de instrumentação.

No âmbito da Agent API foi desenvolvido um módulo central de manipulação de informação de gestão responsável pela organização e controlo do ciclo de vida dos objectos de gestão. A modularidade é essencial neste processo. As capacidades de comunicação são da responsabilidade de outro módulo que efectua a interface com mecanismos específicos. Neste contexto foram descritos os mecanismos SNMP, AgentX e HTTP este último com a possibilidade de gerar diferentes formatos de representação com base na transformação de documentos XML com XSL.

Neste capítulo foram também descritos módulos que apesar de não serem essenciais para o funcionamento dos agentes de gestão providenciam serviços genéricos e que funcionam como extensões à Agent API. A persistência de informação é um exemplo e permite que um agente se auto-configure após uma reinicialização, seja esta provocada ou acidental. Foi descrita uma abordagem à persistência com base no armazenamento em ficheiro de sequências de comandos SNMP descritos em XML.

Como complemento, foi também adicionado um módulo de interpretação de informação SMI, de forma a permitir ao programador incluir informação dinâmica proveniente das especificações MIB. Este suporta SMIv1 e SMIv2 e cria uma representação em memória dos ficheiros MIB. Por último, a localização de recursos SNMP com base em URLs é um mecanismo que permite agregar numa única linha de texto todos os parâmetros necessários à comunicação SNMP independentemente da versão e da API. Este módulo foi disponibilizado com o objectivo de uniformizar a comunicação entre a Agent API e os módulos de comunicação SNMP mas pode ser também utilizado para armazenar internamente os parâmetros de comunicação com outros agentes ou outras aplicações de gestão.

No seu todo, a Agent API fornece um conjunto de serviços comuns e essenciais que possibilita reduzir o tempo e complexidade no desenvolvimento de agentes, além de se encontrar preparada para lidar com a diversidade instalada na rede.

With network sizes well beyond the ability of people to manage them directly, automated, distributed management is vital. An important aspect of such management is the ability of a system to monitor itself or for some other system to monitor it.

Em "rfc2981".

5 Avaliação do Modelo DISMAN para Gestão Distribuída

Os problemas levantados na gestão tradicional SNMP pela utilização de técnicas de convite/resposta ou de geração de notificações, nomeadamente, de escalabilidade, de aumento de complexidade no correlacionamento de informação ou na dificuldade de operação em situações sem conectividade, levaram ao aparecimento de técnicas que permitem delegar responsabilidade de gestão sobre um conjunto de pontos distribuídos ao longo da rede. A dependência dos sistemas geridos num único ponto de responsabilidade pode levar, entre outros inconvenientes, à ocorrência de situações de sobrecarga de processamento.

Apesar de as soluções para evitar a sobrecarga aparentarem alguma simplicidade, nomeadamente o aumento da capacidade do sistema ou a diminuição da carga, já a sua aplicação prática apresenta algumas dificuldades. Nem sempre é possível aumentar a capacidade do sistema devido a esta se encontrar limitada por factores físicos, financeiros ou por decisões estratégicas. Supondo, no entanto, que o aumento de capacidade do sistema é possível, o problema resolve-se provisoriamente até que ocorra nova sobrecarga. Em suma, apenas se está a adiar o problema.

Por outro lado, a redução de carga é possível através da redução da extensão do caso a tratar, ou seja, quanto mais pequeno for o sistema, menor carga este representa. No caso de sistemas de gestão de redes, esta solução implica reduzir o número de agentes, o que não é desejável devido ao facto de se perder resolução e controlo sobre o equipamento de rede. A alternativa passa pela criação de sectores autónomos sob a responsabilidade de uma estação de gestão dedicada. Dentro do IETF, o grupo de trabalho DISMAN debruça-se sobre esta problemática procurando definir mecanismos autónomos de gestão [DISMAN] que tenham a capacidade de

lidar com um conjunto de agentes, permitindo definir estruturas hierárquicas de responsabilidade de gestão (ver secção 3.2).

A arquitectura DISMAN apresenta-se actualmente como a principal arquitectura de distribuição de gestão em ambientes SNMP e, neste contexto, é importante efectuar uma avaliação do impacto causado pela sua introdução. Será este um método adequado para efectuar gestão distribuída em SNMP? Existirá alguma outra metodologia mais favorável? Que impacto tem a arquitectura DISMAN nos sistemas existentes? O aumento de complexidade será aceitável? Será que a gestão destes sistemas não se revela mais complexa que a gestão da própria rede? Qual o impacto nos princípios de simplicidade do SNMP e da não interferência?

O modelo SNMP ganhou popularidade devido essencialmente à sua simplicidade, sendo este um dos princípios associados à sua criação. É também importante que o sistema de gestão não interfira com as aplicações nem com o restante sistema de produção. A introdução da arquitectura DISMAN não deverá colocar em causa nenhum destes princípios, com o risco de prejudicar a simplicidade e transparência do modelo de gestão.

Neste capítulo apresentamos os objectivos, as opções de desenho e as decisões de implementação que resultaram no desenvolvimento de um conjunto de ferramentas DISMAN, nomeadamente a Schedule MIB, a Expression MIB e a Event MIB, essenciais para proceder a uma avaliação prática da arquitectura.

5.1 Metodologia

Para avaliar o modelo definido pelo grupo de trabalho DISMAN foi necessário definir, em primeiro lugar, uma metodologia de trabalho de forma a identificar os objectivos do estudo, os passos a seguir, o ambiente de avaliação e os recursos necessários.

Os grandes objectivos da avaliação do modelo DISMAN assentaram em três vertentes:

1. Dificuldade de aprendizagem e compreensão das normas.
2. Adequabilidade do modelo para realizar gestão distribuída em SNMP.
   - 2.1. Capacidade de descentralização efectiva de serviços de gestão.
   - 2.2. Compatibilidade com SNMP.
   - 2.3. Disponibilidade de implementações.
   - 2.4. Sobrecarga de agentes DISMAN.
3. Utilização do modelo em ambientes de gestão existentes.
   - 3.1. Aumento de complexidade com a introdução do modelo.
   - 3.2. Manutenção e gestão de agentes DISMAN.
   - 3.3. Adaptação da estação de gestão aos agentes DISMAN.

A complexidade das normas pode agravar a sua aprendizagem e, consequentemente, levar à utilização menos correcta do modelo em ambientes de gestão. Por outro lado, pode também originar agentes mais complexos devido ao maior número de serviços.

Para avaliar a adequabilidade do modelo é necessário verificar se a arquitectura DISMAN apresenta os serviços suficientes para descentralizar as tarefas tipicamente associadas à estação de gestão central. Este aspecto foi já considerado anteriormente (secção 3.2) e pode-se concluir que, apesar de o modelo DISMAN apresentar um conjunto mais reduzido de serviços

relativamente a uma estação de gestão, estes são significativos do ponto de vista de distribuição de gestão. Adicionalmente, o modelo é inteiramente compatível com a arquitectura SNMP, nomeadamente SNMPv3, pelo que a sua introdução não provoca ruptura com os agentes existentes.

A abordagem seguida pelo grupo de trabalho encontra-se segmentada em três ramos complementares: o ramo de acção, de reacção e de delegação. Relativamente ao primeiro, na sua forma mais básica um gestor distribuído necessita de um mecanismo que lhe permita iniciar acções por iniciativa própria, periodicamente e/ou em instantes predefinidos. Neste sentido, o gestor distribuído terá de implementar a Schedule MIB [RFC3231] e a Script MIB [RFC3165].

Para desencadear reacções é necessário sondar a rede de forma a reconhecer situações anómalas de funcionamento e actuar de acordo, ou seja, iniciar tarefas dependendo de parâmetros de funcionamento de rede. No centro desta abordagem encontra-se a Event MIB [RFC2981], complementada com a Expression MIB de forma a permitir efectuar cálculos sobre informação de gestão [RFC2982].

O terceiro ramo enfoca a delegação de funções de gestão, concentrado no conjunto Remote Operations MIB [RFC2925].

O principal elemento do modelo é o agente DISMAN (ou DM, de acordo com a nomenclatura definida pelo grupo) que reúne todos ou parte dos módulos definidos. No início deste estudo, não existiam agentes DISMAN disponíveis. Como consequência desta falta de implementações e também para auxiliar na avaliação do modelo DISMAN, decidiu-se desenvolver alguns dos módulos DISMAN, nomeadamente a Schedule MIB, a Event MIB e a Expression MIB. Ainda durante a escrita deste documento, foram apenas identificadas as seguintes:

- Jasmin: The Java 'disman' Client Package (http://www.ibr.cs.tu-bs.de/projects/jasmin/disman.html). Contém implementações da Script MIB e da Schedule MIB.
- Implementação da Event MIB no IOS 12.2 da CISCO (http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121newft/121t/121t3/dtevent.htm).
- SNMP Research: CIAgent (http://www.snmp.com/products/ciagent.html). Contém implementações da Schedule MIB e da Script MIB.

Optou-se pela linguagem Java para o desenvolvimento devido a um conjunto de factores:

- A instrumentação realizada pelo DM passa geralmente por SNMP, pelo que não se encontra dependente de características específicas de hardware.
- O DM encontra-se vocacionado para sistemas computacionalmente mais poderosos que geralmente serão estações de trabalho ou servidores, equipados com os sistemas operativos Windows ou UNIX.
- Pode ser executado em várias plataformas, o que permite instalar o DM em virtualmente qualquer sistema operativo.
- A gestão de memória automática, a orientação ao objecto e uma API rica tornam o desenvolvimento mais simples.

Os módulos desenvolvidos e descritos nesta tese, nomeadamente a Schedule MIB, Expression MIB e a Event MIB, foram baseados na Agent API, de forma a optimizar a reutilização de código e usufruirem de serviços comuns, como a persistência de informação e a utilização de vários protocolos.

Este capítulo descreve essencialmente os módulos Script MIB, Schedule MIB, Event MIB e Expression MIB. Os restantes módulos DISMAN não se encontram referidos. O número de MIBs e a complexidade do modelo DISMAN são consideráveis pelo que algumas opções tiveram de ser feitas. Neste trabalho procurou-se incidir principalmente sobre os ramos de acção, com a Script MIB e a Schedule MIB e de reacção, com a Event MIB e a Expression MIB. Estes permitem definir um mecanismo sensorial elementar a ser aplicado em gestão de redes – analizar, agir.

5.2 Script MIB

O conceito de distribuição de gestão, discutido detalhadamente por Martin-Flatin em [Martin98], teve início em princípios da década de 90 com o trabalho de Yemini *et al.* quando a escalabilidade, flexibilidade e robustez foram consideradas características essenciais para desenvolvimentos futuros em gestão de redes [Yemini91]. Goldszmidt e Yemini desenvolveram a metodologia de delegação de operações de gestão para junto da informação [Goldszmidt98]. De acordo com este conceito, a execução de operações de gestão são delegadas dinamicamente para os elementos de rede e executadas localmente. Esta abordagem introduz uma mudança no conceito original em que a informação é enviada para a estação de processamento central. A metodologia passou a ser conhecida pelo nome de Gestão por Delegação (MbD – *Management by Delegation*).

O modelo MbD utiliza a mobilidade de código para deslocar operações de gestão para junto dos dados. Este processo é tanto mais eficiente quanto maior for a quantidade de informação ou quanto menor for a qualidade de ligação. A abordagem MbD permite reduzir a quantidade total de tráfego e detectar ou mesmo corrigir problemas quando a ligação à estação central é interrompida.

No âmbito da arquitectura DISMAN a Script MIB permite transferir operações de gestão para locais remotos e iniciar e controlar a sua execução, introduzindo a gestão por delegação no modelo SNMP.

De forma a ser o mais geral e, consequentemente, o mais flexível possível, a Script MIB assume quatro diferentes aspectos de intervenção [Schoenwaelder97]:

1. O protocolo utilizado para a delegação de operações de gestão pode seguir diferentes mecanismos, nomeadamente *push-model* ou *pull-model* (ver secção 3.2.2).
2. A linguagem de descrição das operações de gestão (*scripts*) poderá ser qualquer uma desde que reconhecida pela implementação.
3. As operações de gestão poderão necessitar de argumentos e devolver resultados.
4. O ambiente de execução de operações de gestão poderá suportar várias linguagens, multiplos processos e operações simultâneas.

Como resultado, a Script MIB é constituída por seis tabelas (Figura 5.1). Devido à complexidade da MIB, utilizou-se a representação em UML, sugerida em [Schoenwaelder01]. Esta representação identifica as tabelas pelas suas intâncias, ou seja, pelas linhas que as compõem. Por sua vez, estas são representadas pelo objecto do tipo *entry*.

Figura 5.1 – Estrutura UML da Script MIB.

A documentação levanta poucas restrições ao que deverá ser considerado um *script*. De igual forma e em termos genéricos, consideramos *script* qualquer tipo de código executável que possa correr no agente que implementa a Script MIB. Isto significa que os construtores são livres para escolher a linguagem de programação em que serão desenvolvidas as *scripts*. Apesar deste aspecto poder causar problemas em termos de interoperabilidade de *scripts*, esta poderá ser uma medida favorável à adopção da MIB, uma vez que não há apenas uma única "linguagem de programação favorita" [Levi99]. A MIB prevê duas tabelas, *smLangTable* e *smExtsnTable*, para anunciar a linguagem de programação implementada assim como as extensões previstas. Uma estação de gestão que pretenda efectuar a distribuição de *scripts* poderá consultar estas tabelas para verificar se o agente é compatível.

A distribuição de *scripts* é efectuada por intermédio da tabela *smScriptTable*. No caso de ser utilizado o *pull-model*, o URL do código será colocado em *smScriptSource* para posteriormente ser recuperado pelo agente. Para o *push-model*, será utilizada a tabela *smCodeTable*.

A execução de *scripts* é controlada por *smLaunchTable*. Esta tabela prevê mecanismos para especificar argumentos assim como outros atributos, nomeadamente o tempo máximo durante o qual o *script* poderá correr. Os *scripts* que estão a correr ou que completaram recentemente a sua execução são representados na tabela *smRunTable*.

Resumindo e concretizando o exemplo para a linguagem Java, a distribuição de *scripts* para um agente obedece à seguinte sequência de comandos:

1. Desenvolver o código fonte em Java.
2. Compilar e criar o arquivo de código executável (jar).
3. Verificar se o agente suporta a linguagem e respectivas extensões.

4. Definir o *owner* e o *name* da operação.
5. Colocar o código num servidor FTP ou HTTP e indicar o URL ao agente (alternativamente, o código poderá ser directamente enviado por SNMP para o agente, embora não seja a melhor opção em termos de eficiência uma vez que o SNMP não foi desenvolvido para transportar código).
6. Iniciar a execução do *script*.
7. Iniciar sessões de convite/resposta (*polling*) para obtenção dos resultados de execução.

De acordo com algumas experiências práticas de utilização, a MIB consegue com sucesso distribuir a carga de processamento por um conjunto de agentes em vários cenários, nomeadamente, extensão dinâmica de agentes SNMP, monitorização, vigilância de serviços, gestão de serviços e processamento de falhas [Schoenwaelder00b].

No entanto, subsistem algumas limitações. A mais notável ocorre a nível da comunicação entre o *script* e a aplicação de gestão responsável para monitorização e controlo do seu funcionamento e obtenção dos resultados de execução. Os *scripts* podem comunicar com a aplicação de gestão por intermédio de notificações, podem colocar valores num único campo de resultados em formato texto (*smRunResult*) e podem armazenar códigos de saída (*smRunExitCode*). O campo de resultados pode ser escrito durante ou no fim da execução sendo o valor anterior eliminado. Qualquer destes mecanismos é insuficiente quando os resultados de execução de *scripts* são extensos ou reflectem estruturas de dados complexas. Nestes casos será preferível utilizar um mecanismo de extensão de agentes, como o AgentX, para associar mais informação ao agente SNMP [Quittek99].

Dado que os *scripts* são executados remotamente, o processo de monitorização e controlo é mais complexo devido à distribuição, actualização e controlo da sua execução.

Por último, está também identificada a falta de mobilidade de *scripts* o que restringe a utilização da Script MIB em ambientes de gestão cooperativos [Schoenwaelder00b].

5.3 Schedule MIB

A operação de sistemas é muitas vezes regida pela associação de eventos cronológicos a acções. Desde o despertador que nos levanta pela manhã à caldeira de aquecimento que inicia a tranformação de combustível em conforto ou mesmo o gravador de vídeo, programado para gravar semanalmente a série de televisão que, por azar, passa a horas impróprias, os sistemas encontram-se regulados por um relógio que depende da vontade ou da necessidade do utilizador.

Os sistemas de gestão nem sempre têm a disponibilidade total do utilizador para iniciar acções, pelo que recorrem muitas vezes a eventos temporais para despoletar determinadas acções: Exemplos são o início de *backups*, o refrescamento do sistema operativo, a ligação em bloco de PCs nas salas de trabalho e muitas outras. No âmbito do modelo DISMAN, a Schedule MIB recebe a responsabilidade de iniciar acções com base em eventos cronológicos.

Conceptualmente, a Schedule MIB baseia o seu funcionamento na conversão de eventos temporais em operações SNMP do tipo `set` locais. Por outras palavras, sempre que o relógio interno detecta a ocorrência de um instante específico, a Schedule MIB despoleta a alteração de uma variável SNMP no agente local. Os instantes temporais podem ser periódicos, instantes de calendário ou instantes únicos. Os primeiros são definidos por um número inteiro

que especifica o número de segundos entre amostras enquanto que os restantes são compostos por data e hora, assinalando um momento no calendário (Figura 5.2).

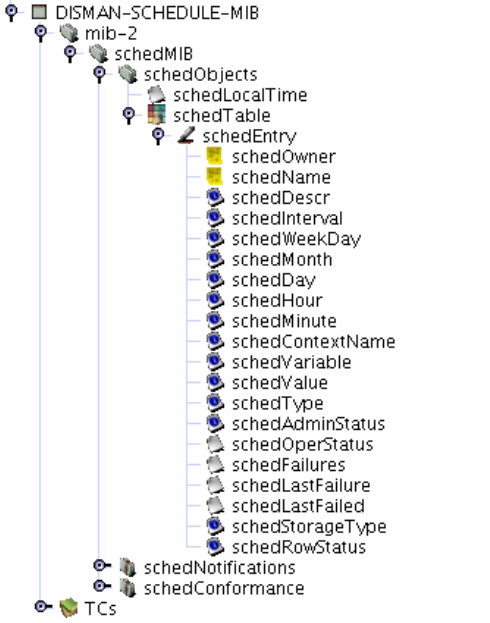


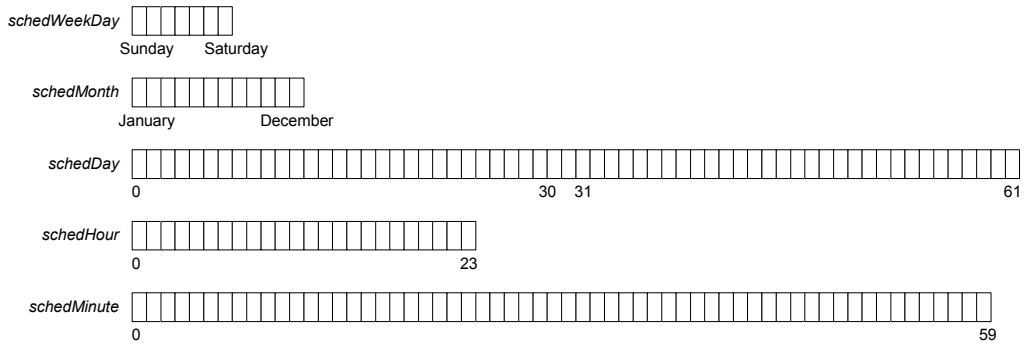
Figura 5.2 – Árvore de objectos da Schedule MIB.

A acção a tomar encontra-se associada ao instante. Esta tem a forma de uma operação `set` que será activada sempre que uma data/hora do calendário se verificar. A Schedule MIB armazena, em conjunto com o calendário, o OID destinatário e o valor associado à operação.

De acordo com o RFC, a MIB contém apenas um escalar (*schedLocalTime*) e uma tabela (*schedTable*). O escalar *schedLocalTime* apresenta a noção de tempo do agente no formato definido pela convenção textual *DateAndTime*, do tipo 2000-10-26,10:15:43.2.+0:0, ou seja, indicativo da data, hora e deslocamento relativamente ao UTC (*Universal Time Clock*).

Cada entrada na tabela *schedTable* especifica um único evento cronológico. Cada evento é identificado por um *schedOwner* e um *schedName* e descrito por *schedDescr*. Cada evento pode ser periódico, de calendário ou único, definido pelo objecto *schedType*.

A definição de eventos cronológicos do tipo calendário ou único baseia-se nas seguintes estruturas de dados de tipo `BIT`:



Por exemplo, um evento que ocorra sextas-feiras dia 13 às 5 horas e 24 minutos será definido de acordo com os valores apresentados na Figura 5.3. As datas são calculadas de acordo com uma conjunção entre os diferentes objectos e uma disjunção dentro do mesmo objecto.

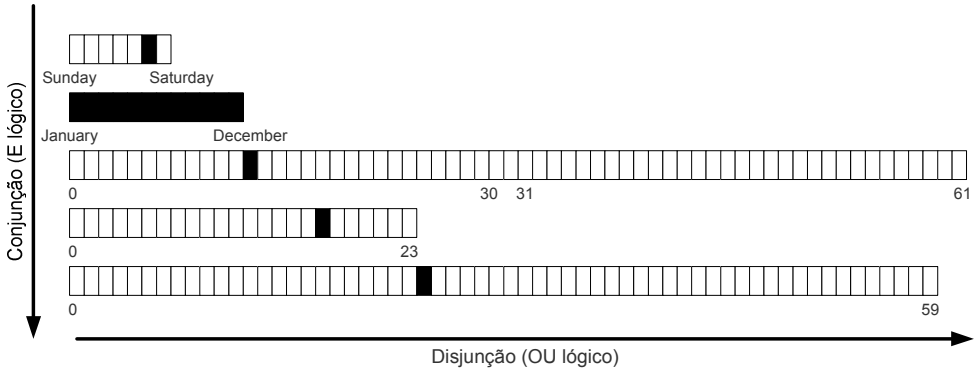


Figura 5.3 – Cálculo de activação do evento cronológico.

Sempre que for detectado que um evento deve ocorrer, a MIB refere a execução de um comando `set` sobre um objecto local (*schedVariable*) do tipo `INTEGER` (*schedValue*) com um determinado contexto (*schedContextName*). Embora este facto possa parecer limitativo, acções mais complexas podem ser efectuadas através da execução de *scripts*.

Os eventos apresentam um estado actual (*schedOperStatus*) que pode ser modificado pelo utilizador (*schedAdminStatus*) para activar ou desactivar a operação.

5.3.1 Modelo construído

A arquitectura da implementação da Schedule MIB é constituída por três grandes blocos (Figura 5.4). O agente mantém um conceito local de calendário, obtido por intermédio do objecto *schedLocalTime*. A informação necessária para a calendarização de acções é armazenada na tabela *schedTable*.

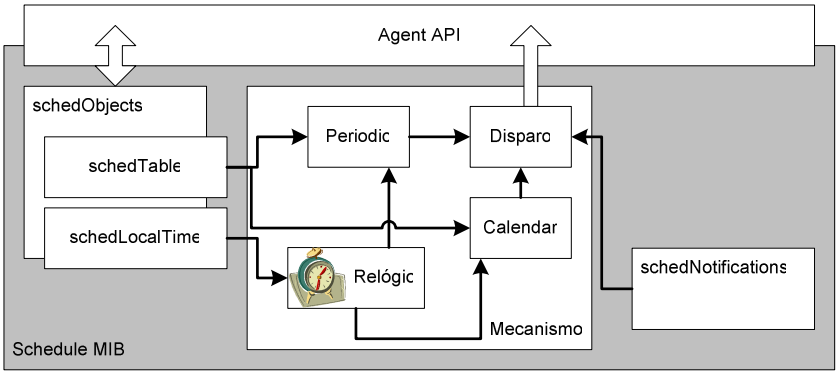


Figura 5.4 – Diagrama de blocos da implementação da Schedule MIB.

Os eventos temporais podem ser periódicos, de calendário ou únicos, identificados pelo estado de *schedType*. No caso de acções periódicas é necessário especificar a frequência de ocorrências. Para os outros dois casos é necessário especificar os instantes que irão despoletar a acção. No caso de acções marcadas como únicas (`oneshot`) a ocorrência de acções é suspensa imediatamente após o instante designado.

O objecto *schedLocalTime* é construído derivando a classe `AbstractObject` (Agent API) de forma a devolver uma cadeia de caracteres com a informação de data num formato especificado pela convenção textual *DateAndTime*: 1992-5-26,13:30:15.0,-4:0 [RFC2579].

A tabela *schedTable* é baseada nos objectos `Table`, `ConceptualTableProvider` e `AbstractConceptualTableModel`. Quando a linha correspondente é tornada `active`, o módulo de relógio inicia o seu funcionamento, notificando a secção de disparo sempre que os instantes designados sejam atingidos. Esta, por sua vez, gera um comando `set` dirigido ao objecto indicado em *schedVariable* com o valor armazenado em *schedValue*.

A Schedule MIB assenta no mecanismo anterior a possibilidade de iniciar acções com base em informação temporal. No entanto a acção terá de ser desempenhada por outro módulo, como a Script MIB. O mecanismo sensorial DISMAN, composto pela Expression MIB e pela Event MIB complementam esta abordagem com a possibilidade de análise do estado de funcionamento da rede e de efectuar algum processamento sobre os valores recolhidos.

5.4 Expression MIB

As expressões matemáticas são correntemente utilizadas para extrair conhecimento de um conjunto de valores que, à partida, não apresentam significado imediato. Por exemplo, pretende-se pavimentar uma sala com 3 m de largura por 5 m de comprimento. Sabendo que cada ladrilho ocupa uma área de 0,5 m^2, será necessário calcular o número de ladrilhos de forma a dimensionar correctamente o orçamento a disponibilizar. Os números não indicam, à partida, resposta concreta ao que se pretende saber. Será necessário efectuar alguns cálculos para avaliar a cabimentação orçamental.

Em gestão de redes, existem muitos valores que, só por si, não são suficientes para sinalizar uma falha. Por exemplo, saber que uma determinada ligação transmite 100 bps num determinado instante não é significativo. Por outro lado, saber que uma ligação se encontra ocupada a 97% pode já apresentar algum significado. O mesmo se pode dizer sobre o número de erros ocorridos durante a leitura de um determinado dispositivo de armazenamento. Saber que ocorrem 7 erros por segundo num determinado instante pode não ser significativo, enquanto que a indicação de que a taxa de erros aumenta 3% por segundo pode indicar uma falha iminente. Os exemplos são inúmeros e fazem com que a existência de um mecanismo de cálculo seja essencial para o bom desempenho de um sistema de gestão de redes.

A Expression MIB pretende dotar os agentes com capacidade autónoma para resolver determinadas expressões matemáticas. O seu objectivo é distribuir os cálculos geralmente efectuados pela estação central de gestão por um conjunto de pontos. As vantagens desta abordagem fazem-se sentir imediatamente em dois aspectos. A consequência imediata é aliviar a sobrecarga de processamento devida à centralização. Por outro lado, o tráfego na rede é também reduzido pois deixa de ser necessário transportar todos os valores, bastando o resultado da aplicação de uma determinada expressão.

Simplificadamente, uma expressão é constituída por valores, operadores e funções articulados de acordo com uma gramática que descreve os passos a seguir para efectuar o cálculo. Este pode ser efectuado a pedido do utilizador ou periodicamente e de forma autónoma, pelo que será necessário realizar a amostragem de valores de acordo com a solução pretendida.

Poderá acontecer que os recursos computacionais disponíveis sofram variações ao longo do tempo devido a um conjunto de factores imprevisíveis. Por exemplo, se o agente está instalado numa estação de trabalho vai estar a competir pelos recursos com as aplicações

executadas pelo utilizador. O número de aplicações e a sua exigência em termos de recursos poderá afectar o desempenho do agente uma vez que a capacidade de processamento e a memória são partilhados com outras aplicações. Por este motivo, será necessário ter em conta os recursos disponíveis em todos os momentos.

Resumidamente, a MIB terá de possuir objectos para monitorização de recursos, definição de expressões, definição e amostragem de variáveis e obtenção de resultados. Cada um destes tópicos ocupa um grupo específico constituído por tabelas e objectos simples (Figura 5.5). Será com base nesta estrutura que a aplicação de gestão interage com o agente durante a definição de novas expressões, eliminação de expressões, recolha de resultados e monitorização de recursos.

Figura 5.5 – Árvore de objectos da Expression MIB.

A MIB encontra-se estruturada em três grupos:

- *expResource* – este grupo está relacionado com o controlo de recursos, com especial incidência sobre parâmetros de amostragem, uma vez que esta operação pode ter algum impacto sobre os recursos do sistema.

- *expDefine* – encontra-se organizado em três tabelas que mantêm informação sobre a definição de expressões e sobre os erros que possam ocorrer durante o seu cálculo. A *expExpressionTable* define a expressão, o tipo de dados que compõe o resultado e o período de amostragem. A *expErrorTable* mantém um registo de erros, contendo o instante em que ocorreu, a operação que o causou e o seu tipo. Finalmente, a *expObjectTable* especifica as variáveis que constituem a expressão, nomeadamente, o tipo de amostragem e se é ou não *wildcarded* (os *wildcards* permitem a aplicação de uma única expressão a diversas instâncias do mesmo objecto MIB, por exemplo a todos os valores de uma tabela).

- *expValue* – este grupo contém uma única tabela com instâncias de objectos que representam o resultado de expressões. Estes são conhecidos por intermédio de operações de consulta.

Como foi referido anteriormente, as variáveis que constituem a expressão podem ser amostradas. Na amostragem absoluta (*absolute*) os objectos são lidos imediatamente antes de avaliar a expressão. Se a amostragem especificar diferenças (*delta*), o valor que é utilizado no cálculo corresponde à diferença entre amostras consecutivas. Para tal é necessário manter a amostra anterior, o que causa uma sobrecarga constante independentemente de o resultado ser consultado ou não. Finalmente, a amostragem do tipo mudança (*changed*) corresponde ao valor lógico 1 (verdadeiro) caso o objecto tenha mudado de valor desde a última amostragem ou 0 (falso) caso contrário.

Associada à amostragem, esta MIB também suporta *wildcarding*, permitindo usar uma única expressão sobre várias instâncias do mesmo objecto MIB. Estes objectos, ao contrário dos regulares, são amostrados por intermédio de uma operação `get-next`. Se a expressão contiver mais de um objecto deste tipo, todos eles terão de partilhar a mesma terminação. Por exemplo, a expressão (1) representa uma expressão constituída por duas variáveis que correspondem a OIDs *wildcarded* ($1="1.3.6.1.32.1.4" e $2="1.3.6.1.50.2.7.1.321"), uma constante (100) e duas operações (multiplicação e divisão).

100*$1/$2 (1)

Os valores são amostrados com operações `get-next` pelo que, além do OID base ($1 ou $2), recuperam também um índice. Se o OID obtido pelo `get-next` para $1 for "1.3.6.1.32.1.4.1.2.3", então o índice será "1.2.3" calculado por comparação com o original. O OID para $2 será então o correspondente à concatenação do OID definido e do índice, ou seja, "1.3.6.1.50.2.7.1.321.1.2.3".

Uma vez que o índice vai ser utilizado na leitura de todas as variáveis da expressão, a tabela *expExpressionTable* contém uma coluna que auxilia na sua recuperação (*expExpressionPrefix*). O agente terá de fazer uma operação `get-next` sobre *expExpressionPrefix* para obter o sufixo a adicionar a cada uma das variáveis da expressão. De seguida, poderá efectuar uma nova operação de `get` sobre a concatenação do OID de cada variável com o sufixo obtido. No exemplo anterior o valor para a coluna de *expExpressionPrefix* poderia ser $1 ou $2, uma vez que o sufixo é igual em ambos os casos.

Uma expressão é constituída por parâmetros, operadores e resultados. Pode-se definir uma expressão em notação BNF (Backus Naur Form) como:

```
expressão             ::= expressãoLógicaOU

expressãoLógicaOU     ::= expressãoLógicaE ["||" expressãoLógicaOU]
expressãoLógicaE      ::= expressãoOU ["&&" expressãoLógicaE]
expressãoOU           ::= expressãoOUExclusivo ["|" expressãoOU]
expressãoOUExclusivo  ::= expressãoE ["^" expressãoOUExclusivo]
expressãoE            ::= expressãoIgualdade ["&" expressãoE]
expressãoIgualdade    ::= expressãoRelacional ["=="|"!=" expressãoIgualdade]

expressãoRelacional   ::= expressãoDeslocamento ["<"|">"|"<="|">=" expressãoRelacional]

expressãoDeslocamento::= expressãoSoma ["<<"|">>" expressãoDeslocamento]

expressãoSoma         ::= expressãoProduto ["+"|"-" expressãoSoma]

expressãoProduto      ::= expressãoNegação  ["*"|"/"|"%" expressãoProduto]

expressãoNegação      ::= expressãoPrimária ["-" expressãoNegação]
```

```
expressãoPrimária    ::= variável | constante | função | "(" expressão ")"

variável             ::= "$" constante
constante            ::= digito* | "\""letra*"\"" | letra | OID
função               ::= identificador "(" {parâmetros} ")"
parâmetros           ::= expressão {"," expressão}
```

Resumindo, a MIB admite vários operadores próximos da linguagem de programação C e que obedecem à mesma prioridade, tais como:

```
( ) + - * / % & | << >>  ! && || == != > >= < <=
```

e ainda um conjunto de funções (Tabela 5.1).

Tabela 5.1 – Funções definidas na Expression MIB.

Função	# argum.	Tipo de argumento
counter32	1	integer
counter64	1	integer
arraySection	3	array, integer, integer
stringBegins	2	octetString, octetString
stringEnds	2	octetString, octetString
stringContains	2	octetString, octetString
oidBegins	2	oid, oid
oidEnds	2	oid, oid
oidContains	2	oid, oid
average	1	integer
maximum	1	integer
minimum	1	integer
sum	1	integerObject* (wildcard)
exists	1	anyTypeObject

O resultado das expressões consta da tabela *expValueTable*, em que cada linha possui apenas um objecto (coluna), identificada de acordo com o tipo de dados do resultado. O OID para cada linha é construído pela concatenação do OID para a coluna específica (descrevendo o tipo de dados respectivo), o nome da expressão e um fragmento (Figura 5.6).

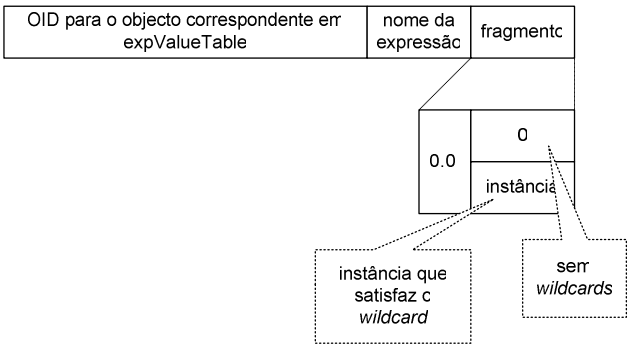


Figura 5.6 – Formato do OID para a *expValueTable*.

O nome da expressão tem a forma x.“owner”.y.“name” convertida para inteiros separados por pontos (‘.’). O inteiro x indica o número de letras de *owner* e y o número de letras de *name*.

Se a última secção do OID, o fragmento, começa com “0.0.” e termina com “0” então o resultado corresponde a um único objecto. Caso contrário estamos perante expressões com *wildcards*, em que cada resultado termina com o índice correspondente.

5.4.1 Modelo construído

O agente é constituído por quatro grandes blocos (Figura 5.7). De acordo com os objectos definidos em [RFC2982], um dos blocos contém os objectos responsáveis pela monitorização de recursos (*expResource*). O bloco *expDefine* é utilizado para a definição de expressões e o *expValue* para a apresentação de resultados. Entre os blocos *expDefine* e *expValue* encontra-se o mecanismo responsável por amostrar valores, calcular a expressão e preencher a *expValueTable*.

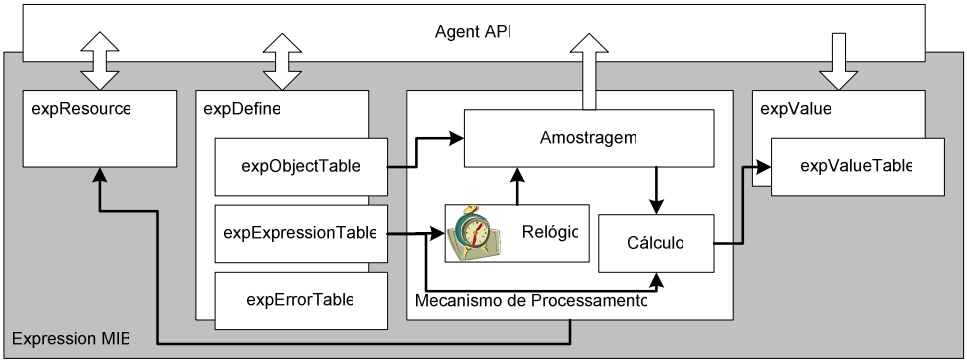


Figura 5.7 – Diagrama de blocos da implementação da Expression MIB.

O módulo *Relógio*, com base no objecto *expExpressionDeltaInterval* da *expExpressionTable*, envia um evento ao módulo *Amostragem* para indicar os instantes de amostragem. Este último, de acordo com os parâmetros definidos na tabela *expObjectTable*, nomeadamente o tipo de amostragem (*absoluteValue*, *deltaValue* ou *changedValue*) e se os valores a obter correspondem a várias instâncias (*wildcarded*), encarrega-se de fazer a amostragem dos objectos necessários para o cálculo da expressão.

O agente desenvolvido neste trabalho contém um módulo de *Cálculo* que, uma vez na posse dos valores amostrados e da expressão definida em *expExpression*, efectua o cálculo e coloca o resultado na *expValueTable*. Para calcular a expressão é necessário reconhecer os elementos da expressão (operadores, funções, constantes e variáveis), isto é, a sintaxe e a gramática da expressão. Para o efeito foi utilizado um analisador sintáctico e um analisador gramatical. Existem várias ferramentas que auxiliam na criação deste tipo de aplicações e que podem, por exemplo, gerar código em C [Manson90] ou Java [Appel98][JavaCC]. Sendo a implementação da MIB baseada em Java foi escolhida a ferramenta javacc [JavaCC], um compilador sintáctico e gramatical. Este gera código fonte correspondente ao analisador sintáctico e gramatical com base nos ficheiros de linguagem sendo posteriormente compilados e incluídos no agente.

O analisador sintáctico interpreta uma cadeia de caracteres e tenta identificar sequências conhecidas a que associa símbolos para posteriormente passar ao analisador gramatical. Este, por sua vez, agrupa os símbolos em frases e invoca acções sobre estes. Neste caso, as frases correspondem a expressões e as acções ao seu cálculo.

Resumidamente, o funcionamento de um agente que implementa a Expression MIB segue um conjunto de estados bem definido cujas transições se encontram associadas a comandos recebidos e ao estado actual (Figura 5.8).

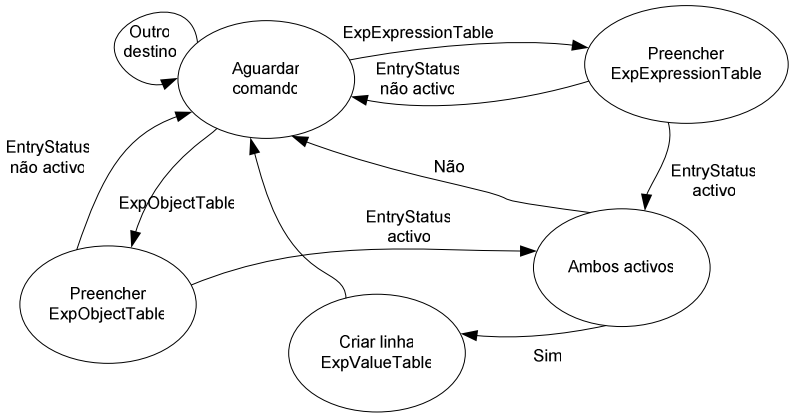


Figura 5.8 – Definição de expressões.

Inicialmente, o agente aguarda comandos. Quando recebe um comando `set` averigua se o OID associado se destina à tabela *ExpExpressionTable* ou à tabela *ExpObjectTable* e, em caso afirmativo, preenche a linha adequada. Se ambas as colunas *expExpressionEntryStatus* e *expObjectEntryStatus* estão activas, averigua a existência de erros na definição da expressão e cria uma entrada na *expValueTable*.

O objecto *expValueTable* é responsável por calcular a expressão. Se a expressão tem alguma forma de amostragem periódica é lançado um *thread* para provocar o cálculo periódico da expressão e consequente armazenamento do resultado. Caso contrário, a expressão é calculada apenas quando o resultado é consultado (operação `get` sobre a *expValueTable*).

De entre todo este processo, a etapa de cálculo é a mais complexa e exigente em termos de recursos, particularmente quando é efectuado sobre várias instâncias (*wildcarded*). Para o efeito, o módulo de cálculo:

1. Consulta a expressão (*expExpression*).
2. Cria objectos para a análise sintáctica e gramatical.
3. Verifica se a expressão incide sobre várias instâncias (*expExpressionPrefix*).
4. Constrói uma lista de objectos (variáveis) que a expressão contém.
5. Consulta o valor correspondente a cada objecto (*expObjectTable*).
6. Calcula o resultado e armazena-o na linha correspondente na tabela *expValueTable*.

A Expression MIB estabelece uma plataforma de cálculo de expressões sobre informação de gestão. Esta providencia uma plataforma de processamento ou de extracção de conhecimento que poderá ser utilizada pela aplicação de gestão para obter a informação significativa, reduzindo a necessidade de obtenção exaustiva de dados. O Anexo E descreve alguns exemplos de expressões que podem ser utilizados em cenários práticos de gestão.

Um possível consumidor para a informação resultante da aplicação de expressões são os agentes baseados na Event MIB. Estes recuperam a informação e efectuam análises de

fronteira com base em condições predefinidas. Caso a condição se verifique verdadeira então é despoletado um evento que poderá resultar numa notificação ou numa operação de escrita.

5.5 Event MIB

Por definição, um evento está associado a algo fora de rotina, distinto, anómalo. Além da sequência natural de eventos cronológicos que dependem da passagem do tempo, surgem eventos que denotam a ocorrência de uma condição especial ou de algum acontecimento inesperado. Os eventos surgem quando uma determinada condição se torna verdadeira. Por exemplo, a campainha da porta toca porque alguém pressiona o botão correspondente ou o piloto automático corrige a posição do avião porque detectou a inclinação das asas.

Alguns eventos solicitam uma acção imediata por parte do utilizador, como abrir a porta quando a campainha toca ou apanhar o pisa-papéis quando cai, outros apenas reportam a ocorrência, como o sinalizador de combustível dos automóveis, e outros passam desapercebidos pela sua pouca importância ou porque o sistema os processa automaticamente.

Os sistemas de gestão de redes actuam de forma a manter o bom funcionamento do equipamento e das aplicações que as compõem. Os eventos resultam da análise de valores que representam o estado de funcionamento dos componentes e das condições associadas. Por exemplo, se a taxa de erros do disco X crescer então produz um evento. O evento pode resultar numa notificação que sugere a substituição do disco ou numa acção que poderá consistir em efectuar uma análise profunda ao disco.

A Event MIB gera um evento sempre que determinada condição se revelar verdadeira, não se encontrando relacionado com os eventos dependentes de outras condições. O evento poderá resultar numa acção ou numa notificação, de acordo com a política definida pelo utilizador.

O módulo MIB contém quatro secções (Figura 5.9).

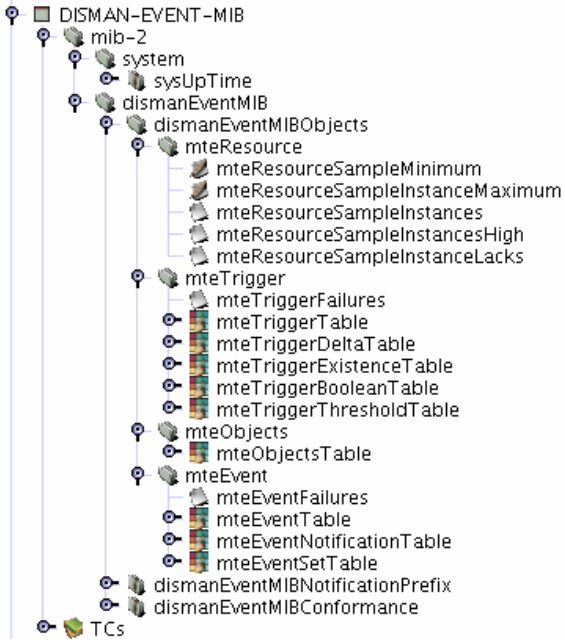


Figura 5.9 – Árvore de objectos da Event MIB.

Cada secção encontra-se relacionada com um aspecto relacionado com o evento:

- condições de disparo – definem os objectos que devem ser monitorados, o tipo de monitorização e o seu relacionamento com os eventos (*mteTrigger*);
- objectos – definem os OIDs adicionados às notificações (*mteObjects*);
- eventos – definem políticas de acção (lançar uma notificação ou modificar um valor) quando um determinado evento ocorre (*mteEvent*);
- notificações – definem as notificações genéricas associadas aos eventos e os objectos associados (*dismanEventMIBNotificationPrefix*).

A primeira secção define condições de disparo que poderão levar à ocorrência de eventos. Esta é constituída por uma tabela que define os objectos a ser monitorados e a forma como cada condição se relaciona com o evento respectivo. Esta tabela é acompanhada de mais quatro que "herdam" os objectos desta e a complementam com o tipo de teste a realizar (*boolean*, *existence*, *threshold* e *delta*) (Figura 5.10).

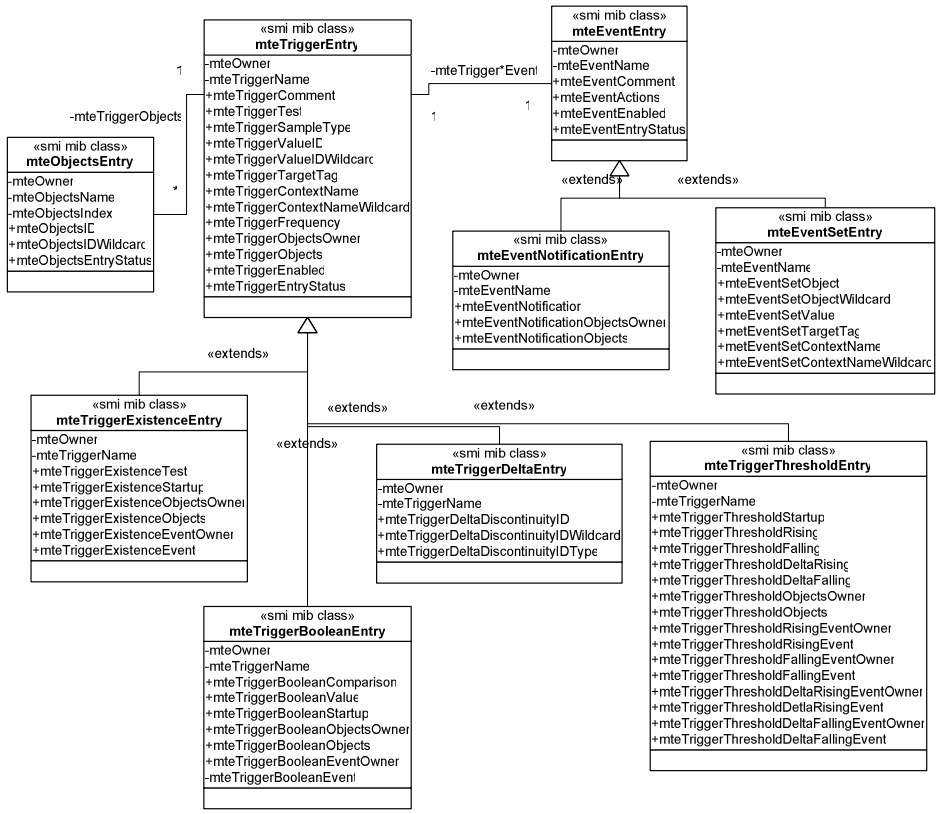


Figura 5.10 – Objectos da Event MIB.

Apesar de a figura apresentar referências para os elementos instanciáveis correspondentes às entradas da tabela, no texto optou-se por utilizar o termo tabela com o objectivo de simplificar a tarefa de compreensão do leitor. O termo *mteEventNotificationEntry*, utilizado para descrever

uma classe na figura, representa as entradas pertencentes à tabela *mteEventNotificationTable*. De notar que a única diferença entre os dois nomes é a terminação da palavra.

O modelo de informação do SNMP não é orientado ao objecto, pelo que não é possível definir herança de objectos. No entanto, é utilizado um artifício para a simular com base na indexação de tabelas. A tabela "pai" (*mteTriggerTable*) é indexada pelo duplo <proprietário, nome da condição>. Esta tabela contém a informação comum, ou seja, os atributos mais genéricos, a todas as condições pelo que tende a ser mais extensa que as tabelas "filho".

As tabelas "filho" utilizam a mesma indexação para relacionar a informação com a existente na tabela "pai" e assim complementar a informação genérica com informação mais específica.

A tabela *mteObjectsTable*, definida na segunda secção, lista os OIDs que irão ser adicionados às notificações. Cada notificação pode ser associada a vários objectos, pelo que existe mais um elemento de indexação – *mteObjectsIndex*.

A terceira secção (*mteEventTable*, *mteEventNotificationTable* e *mteEventSetTable*) define a política a seguir quando um evento é despoletado. A acção a tomar poderá passar por enviar uma notificação, realizar uma operação de `set` ou ambas.

No conjunto de tabelas apresentadas não há qualquer menção ao endereço e ao porto do agente a ser monitorado, informação indispensável para conseguir aceder a informação remota. Para definir os agentes alvo a Event MIB depende dos serviços definidos no módulo SNMP-TARGET-MIB [RFC2573].

Este módulo contém duas tabelas para descrever a informação relativa ao transporte de informação e aos parâmetros de segurança. O transporte de informação é efectuado por um protocolo identificado pelo seu tipo. Este, por sua vez, necessita do endereço do destinatário, tempo de espera e número de tentativas a realizar no caso da entrega falhar. Em termos de segurança, será necessário assinalar o modelo a utilizar, o nome associado à entidade SNMP e o nível de segurança (Figura 5.11).

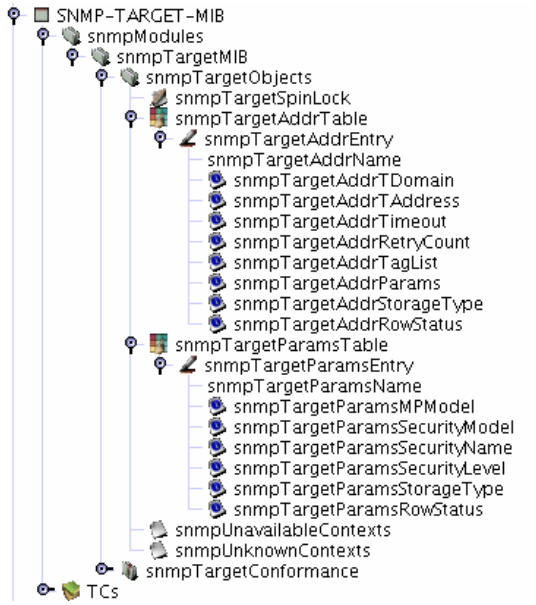


Figura 5.11 – Árvore de objectos da SNMP-TARGET-MIB.

As entradas nas tabelas são agrupadas através de um mecanismo de etiquetagem (*tag list*). Cada linha da tabela contém uma lista de etiquetas (*snmpTargetAddrTagList*) que refere as situações em que esta linha deve ser invocada. Por exemplo, se a lista de etiquetas for "router tráfego análise", qualquer das etiquetas aí presentes selecciona a entrada respectiva. O caracter espaço funciona como separador de etiquetas. Este mecanismo permite seleccionar várias entradas e, consequentemente, contactar vários agentes. No contexto da Event MIB, a etiqueta associada a cada condição encontra-se referida em *mteTriggerTargetTag*.

O objecto *snmpTargetSpinLock* é utilizado para resolver acessos simultâneos à lista de etiquetas (operações atómicas). Sendo um objecto do tipo `TestAndIncr`, cada acesso de escrita só terá sucesso se o valor a modificar for igual ao transportado. Após sucesso, o valor será automaticamente incrementado. O processo de actualização da lista de etiquetas passa por:

1. Consultar os valores *snmpTargetSpinLock* e *snmpTargetAddrTagList*.
2. Gerar a nova lista de etiquetas.
3. Modificar o valor de *snmpTargetSpinLock* para o valor obtido em 1 e *snmpTargetAddrTagList* para o valor obtido em 2. Se o primeiro falhar, regressar ao ponto 1.

5.5.1 Modelo construído

Os agentes baseados na Event MIB são constituídos essencialmente por cinco módulos (Figura 5.12).

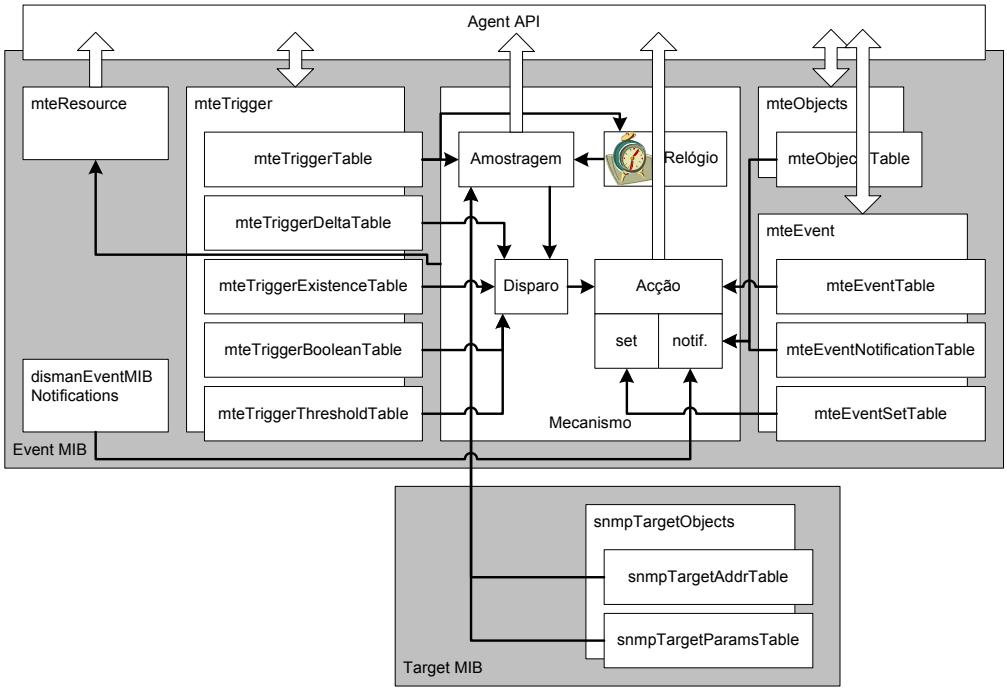


Figura 5.12 – Diagrama de blocos da implementação da Event MIB.

O módulo *mteTrigger*, com as suas cinco tabelas, lista os objectos a serem monitorados, o tipo de amostragem e as condições necessárias para disparar um evento. Este módulo consiste num repositório de informação proveniente, em última instância, do utilizador. Devido ao

facto de os valores poderem ser obtidos de agentes remotos, o módulo de *Amostragem* recorre às tabelas *snmpTargetAddrTable* e *snmpTargetParamsTable* para obter o endereço respectivo.

À semelhança do que acontece com a Expression MIB, a amostragem pode ser efectuada sobre várias instâncias de um objecto SNMP (*wildcarded*) e pode corresponder ao seu valor absoluto (`absoluteValue`) ou à diferença entre amostras consecutivas (`deltaValue`). O módulo *Relógio* é responsável por indicar os instantes de amostragem com base no valor obtido de *mteTriggerFrequency*.

No caso de um determinado valor amostrado obedecer à condição indicada, este facto é sinalizado ao módulo *Acção* que, por indicação da tabela *mteEventTable*, gera um comando de escrita (`set`) e/ou uma notificação (`trap/inform`). Os parâmetros para a acção são obtidos das tabelas *mteEventSetTable* ou *mteEventNotificationTable*, respectivamente.

Todas as tabelas presentes na implementação da Event MIB são objectos do tipo `Table` (Agent API). Os valores que estas contêm e que irão apresentar ao utilizador provêem de classes específicas, derivadas de `TableProvider`, de `AbstractConceptualTableModel` ou de `AbstractTableModel` (ver exemplo de Tabela Abstracta, no Anexo A).

Cada linha criada pelo utilizador provoca a instanciação de todos os objectos relacionados, nomeadamente, os objectos `TableProvider` e os objectos pertencentes ao mecanismo embora respectivos apenas ao evento criado.

A complexidade desta MIB deve-se, principalmente, ao elevado número de objectos que as tabelas contêm bem como ao relacionamento entre tabelas. É também de prever uma sobrecarga considerável sobre a plataforma de execução devido à necessidade de *threads* distintos para cada evento.

5.6 Avaliação da arquitectura DISMAN

Esta secção apresenta um conjunto de ideias vocacionadas para a avaliação da arquitectura DISMAN. Podem-se enumerar os grandes tópicos de avaliação como sendo a adequabilidade para realizar gestão distribuída em SNMP, a integração do modelo em ambientes de gestão existentes e o grau de dificuldade que existe na compreenção dos conceitos que as normas introduzem. A gestão distribuída em SNMP requer total compatibilidade com o modelo SNMP assim como a capacidade efectiva para descentralizar a gestão.

O cenário típico de utilização da arquitectura DISMAN prevê a distribuição de operações de gestão por um conjunto de gestores intermédios (DM). Os objectivos principais da distribuição são diminuir as trocas protocolares com a estação de gestão central, aliviar a carga de processamento que normalmente aí se encontra concentrada e aumentar a robustez do sistema pela introdução de redundância e pela diminuição de dependência da conectividade.

Dado que o modelo SNMP se encontra bastante divulgado em todo o mundo é imperativo que o modelo DISMAN seja de fácil integração em redes e em sistemas existentes sem interromper ou prejudicar as aplicações e os serviços instalados. Uma vez que é necessário introduzir novos agentes com a funcionalidade de gestores intermédios poderá daqui resultar um impacto negativo sobre a rede?

Finalmente, a extensão documental pode também ser uma barreira à aceitação do modelo, pelo que é importante ter uma ideia do esforço necessário em apreender e em passar à prática os documentos definidos pelo grupo de trabalho.

5.6.1 Modelo de distribuição

A distribuição de gestão pode ser classificada quanto ao grau de acordo com quatro designações [Strauss00] (Figura 5.13). A classificação depende do número de aplicações de gestão, representadas por um rectângulo preenchido na figura, do número de gestores intermédios, representados por um rectângulo branco e pelo número de agentes, representados por um círculo branco.

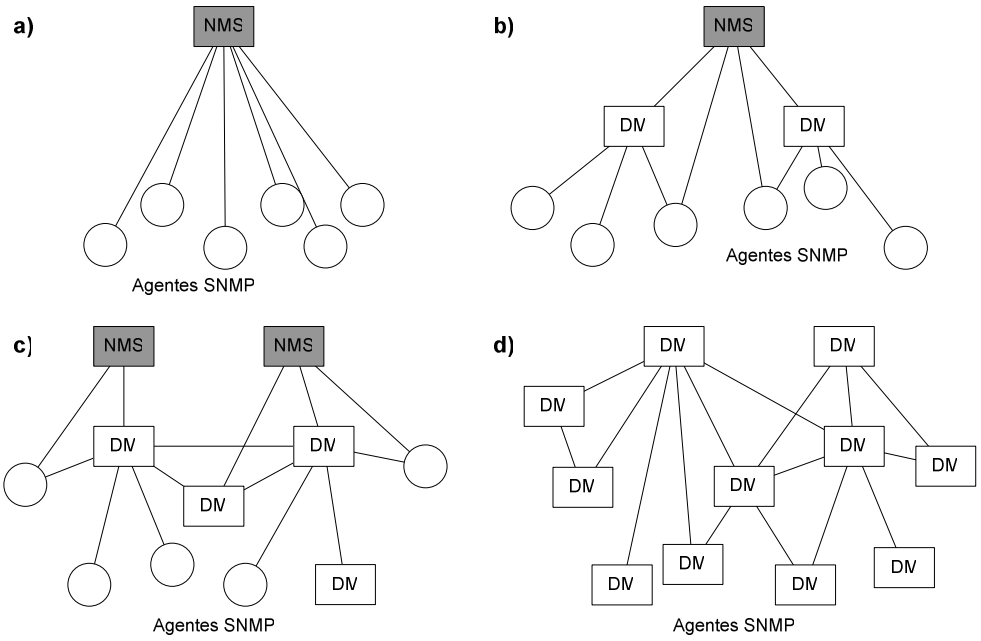


Figura 5.13 – Modelos de distribuição de gestão (adaptado de [Strauss00])
a) gestão centralizada, b) distribuição fraca, c) distribuição forte, d) gestão cooperativa.

Considerando m o número total de gestores, ou seja, a soma do número de aplicações de gestão (NMS) com o número de gestores intermédios (DM), e n o número total de elementos do sistema de gestão (a soma do número total de gestores com o número total de agentes) podem-se definir as seguintes classes [Schoenwaelder00b]:

a) $m = 1$: gestão centralizada;

b) $1 < m << n$: distribuição fraca;

c) $1 << m < n$: distribuição forte;

d) $m \approx n$: gestão cooperativa.

Em a) existe apenas um único gestor, pelo que o correspondente sistema é forçosamente centralizado. b) e c) definem os casos intermédios em que o número relativo de gestores (superior a 1) e de agentes (superior a 1) balanceia o sistema entre a distribuição fraca e distribuição forte. O caso d) representa um cenário em que todos os elementos do sistema têm responsabilidade de gestão e cooperam entre eles para atingir o fim proposto.

A capacidade de comunicação entre os elementos do sistema de gestão é também um factor importante na classificação do sistema quanto ao grau de distribuição. Assim, mantendo o número de gestores e variando apenas a possibilidade de comunicação entre eles posiciona o

sistema entre a distribuição fraca e a distribuição forte. Estas ideias encontram-se sumariadas na Tabela 5.2.

Tabela 5.2 – Modelos de distribuição de gestão.

Modelo	Número de DMs	Comunicação entre gestores	Delegação
Gestão centralizada	nulo	inexistente	inexistente
Distribuição fraca	baixo	inexistente	esporádica
Distribuição forte	médio	esporádica	esporádica
Gestão cooperativa	alto	frequente	frequente

O modelo de gestão tradicional encontra-se enquadrado no modelo de gestão centralizado, onde a estação de gestão é responsável por todo o conjunto de agentes.

Os modelos de gestão cooperativa são mais comuns em sistemas de gestão baseados em agentes móveis ou em agentes inteligentes, em que o grau de delegação e mesmo o número de agentes é elevado.

Os módulos DISMAN são suficientemente flexíveis para serem utilizados independentemente ou em associação com outros módulos, mesmo não sendo DISMAN. Este facto abre a possibilidade de definir gestores com diferentes capacidades de comunicação assim como com diferentes mecanismos de instrumentação. Se todos os agentes de um sistema de gestão forem DMs e efectuarem também instrumentação, então teremos um cenário semelhante ao da gestão cooperativa. No entanto, alguns dos módulos encontram-se limitados em termos de comunicação, pelo que a classificação da arquitectura DISMAN quanto à distribuição terá de ser baseada nestes dois critérios.

Os módulos DISMAN, nomeadamente, Schedule, Script, Alarm, Event, NotificationLog, RemoteOps e Expression, podem ser agrupados num DM ou associados às MIBs intrínsecas do agente SNMP (Figura 5.14).

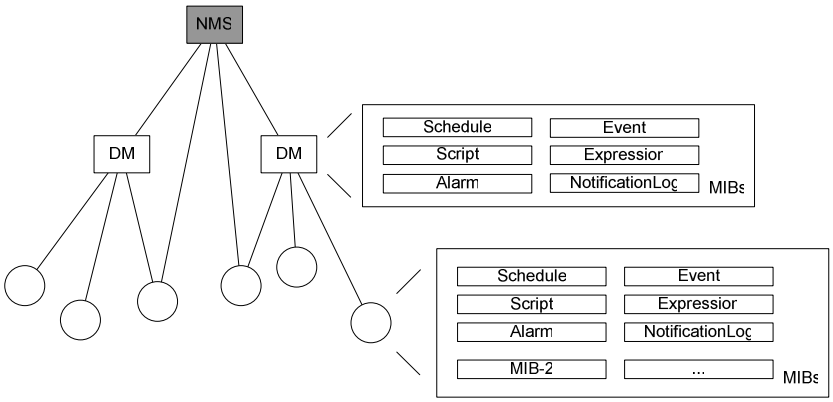


Figura 5.14 – Distribuição de módulos em sistemas SNMP.

Apesar de ser permitida qualquer combinação de módulos, o seu agrupamento rege-se por comportamentos afins, o que faz com que alguns módulos sejam mais compatíveis com uns relativamente a outros. Por exemplo, a Expression MIB fornece a informação já processada à Event MIB enquando que a Schedule MIB inicia acções sobre a Script MIB. Já associações como a Expression MIB e a Schedule MIB não fazem tanto sentido, uma vez que a função da primeira é realizar cálculos enquanto que a segunda é iniciar uma acção com base em eventos periódicos.

Relativamente ao posicionamento dos módulos, alguns apresentam-se mais adaptados ao lado do agente SNMP enquanto que outros se enquadram melhor no lado do DM. Por exemplo, a Expression MIB não pode obter valores de agentes remotos, o que limita a possibilidade de correlacionamento de dados bem como a diversidade de expressões. Este facto faz com que faça mais sentido existir do lado do agente SNMP do que do lado do DM. Aí poderá aplicar expressões sobre a informação local, o que tem vantagens óbvias como a salvaguarda de largura de banda e a distribuição da carga de processamento (Figura 5.15).

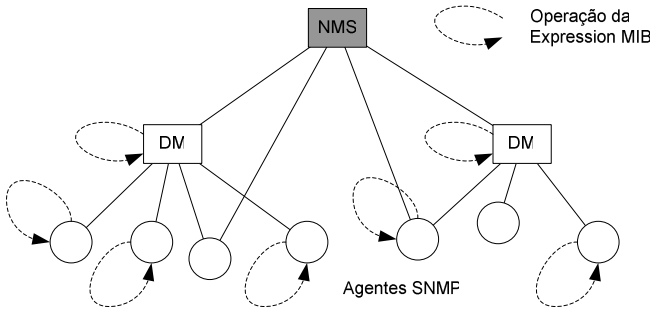


Figura 5.15 – Cenário típico de operação da Expression MIB.

Por este motivo, a utilização da Expression MIB em sistemas DISMAN reduz o grau para distribuição fraca, uma vez que a conectividade entre gestores é inexistente e o número de DMs com Expression MIB é tipicamente baixo.

As motivações que estão por trás desta decisão no âmbito do grupo de trabalho não são claras. A Event MIB tem a possibilidade de obter valores de agentes remotos e para o efeito recorre ao módulo SNMP-TARGET-MIB e ao armazenamento local de credenciais de segurança. Já a Expression MIB não contempla esta possibilidade o que pode dificultar a sua adopção. A sua integração nos agentes existentes só poderá ser efectuada recorrendo à recompilação de código enquanto que a utilização de mecanismos de extensão de agentes, como o AgentX, não é uma opção pois este não permite que subagentes efectuem a consulta de valores.

Uma possível solução para a obtenção de valores remotos e que constitui uma proposta concreta deste trabalho é efectuar uma alteração à definição da Expression MIB com base na Event MIB de forma a uniformizar os dois módulos. Esta alteração permite definir expressões em que as variáveis correspondem a etiquetas (*tags*). As credenciais de segurança necessárias para a obtenção de valores são consultadas na SNMP-TARGET-MIB:

```
x = Expression(target1, oid2, ... targetn)
```

Para adicionar esta funcionalidade à Expression MIB é necessário modificar a tabela responsável por identificar os parâmetros intervenientes na expressão, ou seja, *expObjectTable*. Será necessário adicionar os seguintes objectos:

- *expObjectTargetTag* – designação do sistema remoto a contactar. Funciona em conjunto com a SNMP-TARGET-MIB.
- *expObjectContextName* – designação do contexto utilizado para obter o parâmetro.
- *expObjectContextNameWildcard* – assinala se o nome de contexto deverá ser truncado de forma a admitir qualquer terminação.

A principal vantagem desta abordagem é a total compatibilidade com o modelo SNMPv3 e em particular o módulo Event MIB. No entanto, é necessário que o agente possua mais um módulo MIB, o SNMP-TARGET-MIB.

Uma segunda abordagem para este problema passa pela utilização de SNMP URLs em cada variável da expressão (ver secção 4.4.3). Esta permitem utilizar uma única linha de texto para localizar e configurar os parâmetros necessários à obtenção remota de valores SNMP. Alguns exemplos de URLs são:

```
snmp://rlopes@sw1.estig.ipb.pt/sysContact/0?op=set&value=Rui?v3
snmp://guest@nms.estig.ipb.pt:161/sysUpTime?op=getNext?v3?router
```

Estes apontadores permitem definer expressões do tipo:

```
x = Expression(url1, oid2, ... urln)
```

O impacto no agente e em particular na Expression MIB resume-se à alteração do tipo de dados da coluna de variáveis de OID para URL. Este define o endereço, o objecto, o contexto e os outros parâmetros necessários à comunicação numa única linha de texto. Ao contrário da abordagem anterior, não será necessário implementar mais módulos MIB e a única alteração ocorre ao nível do objecto *expObjectID*.

Independentemente da abordagem seguida, se a Expression MIB puder obter valores remotos então será possível utilizar valores de outros agentes e de outros DMs na mesma expressão (Figura 5.16).

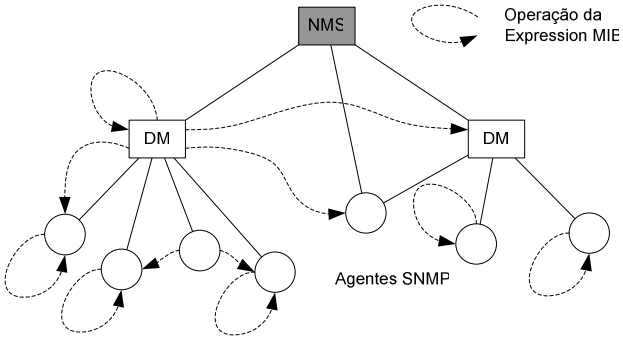


Figura 5.16 – Operação modificada da Expression MIB.

Outra falha semelhante encontra-se na Schedule MIB. Tal como proposta pelo grupo, não é possível modificar valores remotos, encontrando-se o módulo limitado ao agente local. Pode-se argumentar que esta lacuna é eliminada pela sua associação à Script MIB. Em certas situações não é necessária uma ferramenta tão elaborada, o que leva ao lugar comum de a solução servir para "matar uma mosca a tiros de canhão".

A extensão da Schedule MIB para permitir iniciar tarefas remotamente não se revela uma tarefa que aumente exageradamente a sua complexidade e, no entanto, aumenta bastante a sua flexibilidade em termos de cenários de utilização.

Em termos gerais, a arquitectura DISMAN é adequada para distribuir tarefas de gestão SNMP, uma vez que é inteiramente compatível com o ambiente e apresenta um conjunto bastante significativo de operações de gestão em funcionamento autónomo. Deveria, no

entanto, prever a possibilidade de cálculo de expressões com valores remotos assim como despoletar acções remotamente, o que viria a aumentar a sua flexibilidade de utilização.

A disponibilidade de implementações completas das especificações DISMAN é ainda reduzida, provavelmente devido à relativamente recente adopção das normas pelo IETF. É, no entanto, possível encontrar e utilizar alguns módulos inclusivamente integrados em equipamento de comunicação, como encaminhadores. De qualquer forma, a não existência de implementações completas e economicamente equilibradas pode resultar numa baixa aceitação por parte da comunidade.

5.6.2 Complexidade das normas

O número de documentos definidos pelo grupo de trabalho DISMAN é já bastante extenso, totalizando de momento dez módulos MIB definidos em seis RFCs e dois *draft*. Este é já um número considerável de documentos (de lembrar que o modelo SNMP começou com apenas três) e com um certo grau de complexidade devido à diversidade de serviços que apresentam.

Em termos absolutos e para efeitos de comparação, pode-se traduzir a extensão dos documentos numa contagem do número de linhas (Figura 5.17). Independentemente da sua dimensão, os documentos podem ser interpretados de duas formas distintas. O utilizador de um sistema de gestão DISMAN necessita conhecer o funcionamento de cada módulo, a articulação entre módulos e a interacção com o restante sistema. Para tal terá de fazer uma interpretação geral da arquitectura, coadjuvada por ferramentas gráficas que lhe dão uma ideia global da estrutura das MIBs, da responsabilidade de cada tabela e do funcionamento de cada objecto. Mesmo recorrendo a ferramentas gráficas de gestão a leitura da documentação é recomendável.

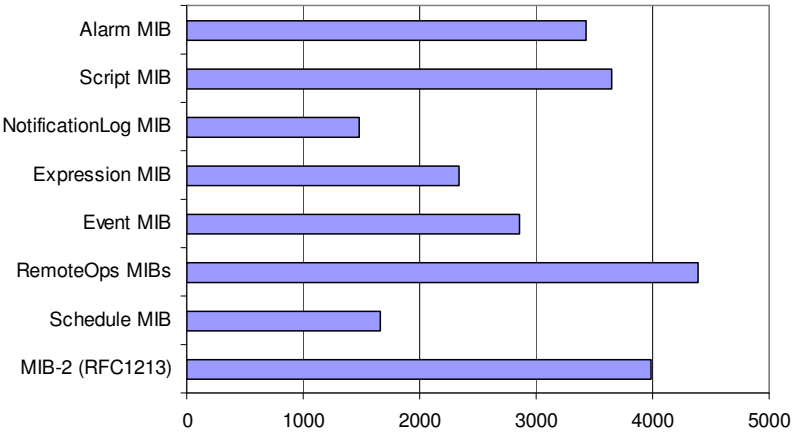


Figura 5.17 – Número de linhas dos RFCs relativos à arquitectura DISMAN.

A interpretação geral da documentação DISMAN é relativamente simples para utilizadores com conhecimento razoável do modelo de gestão SNMP. Os programadores ou genéricamente as entidades responsáveis por desenvolver protótipos ou produtos baseados nas especificações DISMAN, têm de efectuar uma interpretação pormenorizada da documentação. Os documentos são muito ricos em detalhes, apresentando grande concentração de pormenores, o que facilita a planificação do desenvolvimento mas requer mais atenção no momento de leitura.

Neste tipo de abordagem, os documentos DISMAN não bastam por si. Existem muitas dependências e referências a outros documentos relacionados com SNMPv3, nomeadamente, outros módulos MIB, modelos de segurança e outros mecanismos específicos como o transaccional. Estas dependências vêm aumentar o número de documentos e a sua densidade, consequentemente resultando numa maior complexidade global.

Resumidamente, do ponto de vista do utilizador a documentação é relativamente acessível e bastante completa levando a uma fácil adopção da arquitectura. Do ponto de vista do programador, a tarefa de aprendizagem é bastante complexa devido às dependências e relações entre módulos, assim como à funcionalidade e flexibilidade das ferramentas, o que pode explicar o reduzido número de implementações da arquitectura DISMAN.

5.6.3 Impacto do modelo em ambientes de gestão existentes

A introdução do modelo DISMAN em ambientes de gestão existentes influencia diversos factores que abrangem, entre outros, as trocas protocolares e os recursos computacionais. O grande objectivo da introdução da arquitectura DISMAN em ambientes de gestão é poder usufruir dos mecanismos de distribuição de forma a reduzir, por um lado, as trocas protocolares e, por outro, a concentração de capacidade computacional. Há ainda um outro aspecto associado que pode ser considerado um efeito secundário da distribuição e que consiste no aumento de robustez em possíveis interrupções de comunicação. Este último caso permite lidar localmente com situações em que a estação central não se encontra disponível.

As trocas protocolares poderão ser reduzidas em dois sentidos. Por um lado, o número de mensagens é reduzido devido ao processamento local de informação. Neste aspecto, a Expression MIB apresenta um papel importante, embora possa ser coadjuvada por outros módulos, nomeadamente, Alarm MIB, Script MIB e Schedule MIB.

Por outro lado, a utilização de DMs permite criar ilhas de tráfego local. Em arquitecturas clássicas (ou não distribuídas) a estação de gestão efectua a monitorização e controlo de todos os agentes instalados na rede (Figura 5.18).

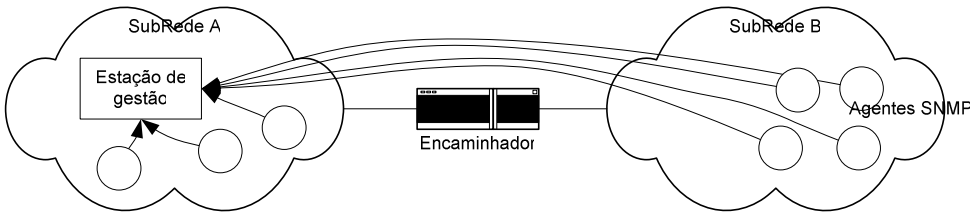


Figura 5.18 – Monitorização de agentes em sub-redes distintas.

Neste caso, haverá tráfego que atravessa as duas sub-redes ocupando os recursos de comunicação de ambos os lados bem como os partilhados.

A instalação de um DM nas sub-redes remotas traz a vantagem de diminuir as trocas protocolares com a estação de gestão central (Figura 5.19).

Poder-se-á argumentar que esta medida traz também o inconveniente de aumentar o número de agentes nas sub-redes remotas, o que obrigará a central de gestão a realizar um maior esforço de configuração. De lembrar que a configuração do DM é efectuada esporadicamente o que, associada à persistência local de informação (segundo modelo proposto na Agent API – secção 4.4.2), reduz este esforço ao mínimo. Após este passo, o tráfego entre sub-redes será filtrado pelo DM através dos serviços aí instalados.

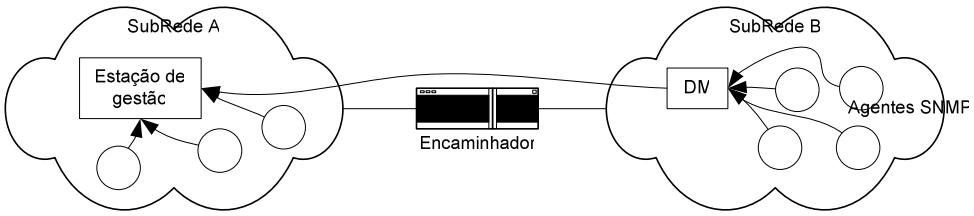


Figura 5.19 – Criação de ilhas de tráfego local.

Nos casos apresentados anteriormente, caso a comunicação entre sub-redes seja interrompida, cada sub-rede continuará a ter pelo menos um elemento de gestão. No caso da sub-rede 'A' será a própria estação de gestão e no caso da sub-rede 'B' será o DM. Apesar da interrupção, o sistema de gestão continua operacional, ou seja, verifica-se um melhor comportamento em situações de perda de comunicação. Adicionalmente, é também mais autónomo, uma vez que algumas funções inicialmente da inteira responsabilidade da estação de gestão são agora repartidas.

Se, por um lado, o impacto da introdução da arquitectura DISMAN é positivo, por outro lado há aspectos que resultam prejudicados, nomeadamente um aumento de requisitos de hardware e uma maior despesa com software.

Foi referido que a descentralização de processamento alivia a estação de gestão central devido a um certo número de operações ser realizado remotamente. Para este efeito o hospedeiro do módulo DISMAN terá de possuir os recursos suficientes, em termos de memória e de processamento, para as executar. No caso de o DM ser instalado em estações de trabalho ou servidores, os recursos são fornecidos pela máquina não sendo necessário, à partida, adquirir hardware especializado. Neste caso poderá, contudo, haver algum impacto sobre as aplicações ou sobre os serviços anteriormente instalados.

No caso de o DM ser alojado em equipamento específico de rede, por exemplo, encaminhadores ou comutadores, poderá ser necessário efectuar uma actualização do seu sistema operativo e, eventualmente, do seu hardware embora a crescente miniaturização que actualmente se faz sentir possa aliviar este problema.

Finalmente, as estações de gestão já instaladas não suportam os módulos DISMAN. A interpretação da especificação SMI pode possibilitar manipular os DMs através de uma ferramenta genérica, como por exemplo, um *browser* de MIBs. No entanto, as particularidades das tabelas, o elevado número de objectos, as relações entre serviços e as interdependências podem fazer com que a utilização de ferramentas deste tipo seja um pesadelo. É importante existirem ferramentas gráficas para gestão da arquitectura DISMAN e que possam ser integradas nas ferramentas de gestão.

Será necessário, portanto, readaptar as estações de gestão aos novos agentes, o que poderá corresponder a novas despesas. Neste sentido, a modularidade a nível da aplicação de gestão é indispensável de forma a permitir adicionar novos módulos sem necessidade de recompilação de código.

5.7 Conclusões

Neste capítulo discutiu-se e avaliou-se o modelo IETF-DISMAN para gestão distribuída cobrindo três vertentes: dificuldade levantada ao utilizador para aprender e compreender as normas, capacidade efectiva do modelo para descentralizar operações de gestão SNMP e impacto da sua introdução em sistemas de gestão existentes.

Procurou-se restringir o foco de discussão aos módulos Script MIB, Schedule MIB, Expression MIB e Event MIB, que representam os ramos de acção e de reacção da gestão distribuída, para não alargar demasiadamente a discussão. Neste âmbito, foi realizado um estudo aprofundado das normas DISMAN após o qual se desenvolveram os módulos Schedule MIB, Event MIB e Expression MIB. Foi também utilizada uma implementação disponível em domínio público da Script MIB denominada Jasmin. Este passo procurou facilitar a resposta às questões levantadas para melhor avaliar os requisitos de um sistema DISMAN e qual o seu comportamento em ambientes de gestão existentes.

A complexidade inerente ao desenvolvimento é relativamente elevada dada a interdependência e a extensão documental. Apesar de perfeitamente compatível com o modelo SNMP, os módulos DISMAN não são de configuração imediata. Há muitos e variados factores de configuração, nomeadamente, objectos, tabelas e operações, que têm de ser manipulados pelo utilizador de forma a construir uma política adequada.

A reduzida granularidade do modelo SNMP, que deriva da necessidade de configuração de várias variáveis para definir uma única operação, torna os agentes mais complexos e difíceis de configurar. Comparando com a invocação de funções, por exemplo, a configuração de uma operação calendarizada será efectuada passando todos os parâmetros necessários à função e receber os respectivos resultados. Em SNMP será necessário criar uma entrada numa tabela e configurar todas as colunas antes que a operação possa ser activada. A manutenção de índices, a criação dinâmica de objectos e a possibilidade de falhas de comunicação transportam para o lado do agente a complexidade associada.

Devido ao facto de os módulos DISMAN requererem uma forte componente de configuração durante a definição de políticas de gestão distribuídas, estes factores fazem-se sentir com maior relevância o que resulta numa tarefa de desenvolvimento e de utilização mais pesada.

Foi detectada alguma incoerência entre os módulos MIB a nível da funcionalidade. Enquanto que alguns módulos têm a possibilidade de contactar agentes remotos, como a Event MIB e a Script MIB, os módulos Schedule MIB e Expression MIB são virtualmente míopes, restringindo o seu acesso ao agente local. Este capítulo propõe duas abordagens para ultrapassar este problema dando a possibilidade de obtenção remota de valores com base na SNMP-TARGET-MIB e nos SNMP URLs.

Não! Não vou por aí! Só vou por onde
Me levam meus próprios passos...
Se às coisas que eu pergunto (em vão) ninguém responde.
Porque me dizeis vós: - "Vem por aqui?"
Prefiro escorregar nos becos lamacentos,
Redemoinhar, aos ventos,
Como farrapos, arrastar os pés sangrentos,
A ir por aí...

José Régio, "Cântico Negro"

6 Agentes Móveis em Gestão SNMP

A arquitectura SNMP é actualmente o modelo de gestão mais divulgado em termos de equipamento e *software*. A sua utilização ao longo de mais de uma dezena de anos criou um conhecimento técnico que outras arquitecturas não atingiram. Qualquer nova ferramenta ou arquitectura de gestão terá de lidar com as normas SNMP se pretender ter algum impacto no mercado.

Os agentes móveis são programas que têm a possibilidade de se deslocar entre os nós da rede, ou seja, podem ser criados em qualquer ponto da rede, interromper temporariamente a sua execução, deslocar o código e o estado e retomar a execução noutro ponto da rede. Estes assentam em conceitos como o de agência, que lhes dá o ambiente de execução, lugar, que lhes garante uma abstracção sobre a plataforma e região, um serviço de directoria para agentes móveis.

Para que a tecnologia de agentes móveis possa contribuir de forma válida em cenários de gestão de redes terá de passar por uma fase de integração com o SNMP [Lopes00a]. Esta integração poderá ser efectivada em duas vertentes:

1. Por um lado, poder-se-á tirar partido de aplicações de gestão móveis no sentido em que os agentes móveis terão a capacidade de monitorar e controlar agentes SNMP clássicos. Esta solução consiste no enriquecimento de aplicações de gestão clássicas com mobilidade [Oliveira99]. Inevitavelmente, os agentes móveis serão maiores e mais complexos como resultado da adição do mecanismo de gestão. O código a transportar

tenderá portanto a ser excessivo. De forma a atenuar este inconveniente parte dos serviços poderão ser deslocados para a agência. Nestas situações pode ser desenvolvida uma interface entre o agente móvel e a agência que permita obter uma solução mais eficiente [Pagurek00].

2. A segunda vertente tem a ver com a própria gestão de agentes móveis. A gestão de mobilidade não implica necessariamente a utilização do SNMP, ou seja, a gestão do ambiente pode ser efectivada pelo próprio sistema, geralmente recorrendo a ferramentas e/ou técnicas proprietárias. Há, no entanto, vantagens em utilizar a arquitectura SNMP, uma vez que é uma tecnologia já difundida e permite utilizar a mesma ferramenta para a monitorização e controlo do equipamento de rede e dos próprios agentes. Para resolver este problema a solução será integrar a pilha protocolar SNMP no código do agente. Tal como apontado anteriormente, esta medida pode fazer com que o código seja excessivamente extenso. Por outro lado, a arquitectura de gestão pode ser providenciada pela agência, criando assim uma porta para todo o sistema sem sobrecarregar os agentes.

Neste capítulo serão discutidos e clarificados estes problemas e apresentadas algumas soluções para os resolver.

6.1 Agentes móveis como sistema distribuído

A mobilidade de código permite melhorar a flexibilidade e adaptabilidade das aplicações distribuídas em tempo de execução. Adicionalmente, introduz vantagens do ponto de vista de desempenho em situações em que a deslocação de código é preferível relativamente à deslocação de grandes quantidades de dados. Esta abordagem introduz um paradigma de programação mais geral relativamente aos modelos de convite/resposta utilizados, por exemplo, na invocação remota de funções, ou relativamente à utilização de memória partilhada e distribuída.

Tipicamente, são sugeridas três variações deste paradigma [Baldi97]:

- Código a Pedido – as aplicações baseadas nesta variação podem carregar e utilizar diferentes blocos de código proveniente de um servidor específico em qualquer instante. Este facto faz com que o código em que se baseiam os serviços de uma aplicação deste tipo possa mudar ao longo do tempo. Um exemplo desta variação são os *applets* normalmente integrados numa página HTML e executados num navegador de Internet.

- Execução Remota – de acordo com esta variação, os componentes no papel de clientes invocam serviços em outros componentes, por sua vez, no papel de servidores. Os clientes associam o nome, os parâmetros e o código necessário para descrever o serviço pretendido e enviam-nos para o servidor, onde será concedida autorização para aceder aos recursos da plataforma. Por outras palavras, os clientes possuem o código necessário para implementar o serviço enquanto que os servidores possuem os recursos. Estes últimos disponibilizam um único serviço, correspondente à execução do código proveniente do cliente. A passagem de objectos por valor pode ser utilizada como exemplo, à imagem do que acontece na tecnologia RMI.

- Agentes Móveis – nem sempre o termo agente é correctamente aplicado quando é feita referência a agentes móveis. Este termo apresenta significados distintos de acordo com o contexto da área de investigação onde é utilizado. Geralmente, um

> agente móvel é visto como uma unidade de execução independente, tal como um processo num sistema operativo com características de multiprocessamento. Este tem a capacidade de interromper a sua execução, migrar para outro nó e continuar a execução a partir do ponto onde tinha sido interrompida, de forma autónoma. Cada agente móvel possui o código necessário para desempenhar determinada tarefa mas não possui os recursos, podendo deslocar-se para os poder utilizar.

A tecnologia de agentes móveis é sugerida muitas vezes como uma mais valia na gestão de redes cada vez maiores e mais heterogéneas [Oliveira99] [Breugst98]. De facto, os agentes móveis permitem descentralizar o processamento e o controlo, ou seja, permitem transportar as tarefas de gestão para onde estas são necessárias, ou seja, para cada dispositivo. Como consequência, podem reduzir o tráfego dirigido à plataforma de gestão, adaptar de forma transparente a comunicação tradicionalmente síncrona para assíncrona, ser usado em ligações de baixa qualidade ou com tendência a falhas, distribuir carga e aumentar a flexibilidade dos agentes de gestão clássicos. Além disso, os agente móveis são caracterizados pela possibilidade de se deslocarem para junto dos repositórios de informação e operar localmente sobre ela, seja com o objectivo de efectuar pesquisas, processar informação ou outro tipo de operação definida pelo utilizador.

Os campos de aplicação de agentes móveis têm seguido essencialmente o mercado das telecomunicações, nomeadamente em distribuição e delegação de gestão [Goldszmidt98], serviços de rede [Krause96], optimização de tráfego e tolerância a falhas [Lopes99]. A tecnologia encontra utilidade também em pesquisa de informação [Nekrestyanov99], comércio electrónico [Gleizes99] ou mesmo interfaces com o utilizador [Green97].

6.2 Agentes móveis em gestão de redes

O conceito de delegação surge naturalmente no contexto dos agentes móveis – a estação de gestão associa o código e os dados necessários para cumprir determinada tarefa e envia-os sob a forma de agentes para serem executados nos dispositivos de rede. A estação de gestão fica então livre para realizar outras operações, aumentando o grau de paralelismo do sistema. Por outro lado, o código não fica associado directamente aos dispositivos, podendo ser alterado e reenviado pela estação em qualquer altura. Esta característica permite optimizar dinamicamente os serviços disponibilizados pelos dispositivos de rede.

Neste sentido foram já propostos vários sistemas e arquitecturas cobrindo diversas áreas. Bieszczad *et al.* [Bieszczad98] apresentam um conjunto de vantagens da utilização de agentes móveis para a gestão de redes. Estes mesmos problemas podem ser resolvidos com a utilização de paradigmas do tipo cliente/servidor, embora possam resultar em soluções mais complexas, menos eficientes ou de adopção mais difícil. Serão mais complexas quando o sistema envolve um maior número de operações, menos eficientes quando ocupam demasiados recurso e de adopção mais difícil quando a sua utilização se reveste de maiores dificuldades.

Como exemplo, pode ser utilizado um sistema de correlacionamento de informação proveniente de várias fontes. Em arquitecturas cliente/servidor, será necessário contactar individualmente as várias fontes e adquirir a informação para processamento posterior. Um agente móvel com a mesma função consiste em apenas uma instância que efectua o processamento à medida que se desloca.

A delegação de gestão pode incidir sobre qualquer das cinco áreas sugeridas em [ISO10040]. A gestão de configuração assinala o processo de configuração de todos os componentes de rede

com base em informação aí adquirida. Este processo inclui a manutenção de um inventário actualizado, o armazenamento da informação e a geração de relatórios. Como exemplo, suponha-se que a interface de rede de um determinado dispositivo causa o aparecimento de erros numa determinada secção da rede. Através de uma ferramenta de configuração, o utilizador poderá avaliar se a interface se encontra incorrectamente configurada e, caso seja este o motivo, corrigir o problema ou simplesmente desactivá-la.

Ainda no âmbito da gestão de configuração, o primeiro passo para a gestão de redes passa, geralmente, pela construção de modelos ou vistas da rede, desde o levantamento topológico à construção de vistas detalhadas de serviços disponíveis. À medida que o número de nós e/ou a complexidade aumenta, mais difícil será de implementar um sistema com base em paradigmas do tipo cliente/servidor que permita efectuar de forma eficiente este levantamento.

Até certo ponto, os agentes móveis são imunes ao aumento de dimensão ou de complexidade da rede. Com base nesta característica, uma abordagem sugerida para a construção de modelos da rede é fazer com que cada agente percorra a rede construindo uma visão própria obtida ao longo do percurso. A cada encontro com outro agente, estes partilham informação, obtendo uma visão mais completa da rede. Cada novo encontro vai aumentando o seu conhecimento até abranger a rede como um todo [Minar98]. Os agentes são conceptualmente simples, bastando terem a capacidade para se lembrarem por onde passaram e para trocar informação entre eles.

Esta informação pode ser utilizada para construir uma interface gráfica com o utilizador. Será por intermédio dela que os comandos são gerados e os avisos provenientes da rede apresentados. Geralmente, esta aplicação encontra-se instalada em nós relativamente poderosos da rede e providenciam um posto de trabalho fixo para o utilizador. Este facto faz com que o utilizador não possa afastar-se do seu lugar com o risco de deixar de receber informação da rede e de actuar sobre o equipamento.

Sahai *et al.* [Sahai98] sugerem a utilização de plataformas de gestão móveis, ou seja, sem associação directa a nós específicos e podendo acompanhar o utilizador mesmo quando este não se encontra próximo da rede. O sistema pode inclusivamente funcionar a partir de postos portáteis, geralmente com larguras de banda reduzidas e tempos de latência elevados. Devido ao facto de, por princípio, ser permitido o funcionamento através de ligações intermitentes é necessário um segundo módulo. Este é denominado servidor de gestão e desempenha o papel de procurador entre a aplicação de gestão e os agentes SNMP. Além disso serve como repositório para os eventos gerados pela rede.

A introdução de tecnologia mais recente nas redes em funcionamento pode provocar uma rotura com o sistema de gestão existente. No caso particular das redes sem fios será necessário prever mecanismos de configuração que permitam a um determinado utilizador deslocar-se ao longo da rede mantendo a conectividade independentemente da sua localização. Kramer *et al.* [Kramer99] defendem a utilização de agentes móveis, redes activas e modelos de sociedades de insectos, como colónias de formigas, para a gestão de redes sem fios. Para o efeito sugerem dois tipos de agentes. Os agentes de encaminhamento deslocam-se pela rede reunindo informação topológica e actualizando as tabelas de encaminhamento dos nós por onde passam. Estes são responsáveis por criar e reparar, de forma automática, as tabelas de encaminhamento. Os mensageiros, outro tipo de agentes, têm a responsabilidade de transportar a comunicação (transporte e encapsulamento de pacotes) e utilizam a informação contida nas tabelas de encaminhamento para se deslocarem até um destino predefinido.

Existem ainda outros mecanismos baseados na mobilidade de código aplicados à gestão de redes e, em particular, à gestão de configuração. Um exemplo de abordagem segue o paradigma das redes inteligentes e disponibiliza métodos de actualização e configuração do equipamento de rede [Silva01]. Este trabalho descreve linguagens baseadas em XML para descrever o fluxo de dados, a semântica de protocolos e as regras de classificação de pacotes. Estas linguagens, associadas a blocos de código "activo", definem um comportamento flexível e adaptativo de acordo com os pacotes que transitam pela rede. Os blocos de código têm a responsabilidade de processar o fluxo de dados de acordo com as regras especificadas pelas linguagens e podem ser instalados ou actualizados em tempo de execução.

A gestão de falhas consiste no processo de localização e correcção de problemas na rede e passa por três fases distintas. A primeira consiste em identificar a falha, onde se procura detectar que algo não está a correr como esperado. De seguida vem a fase de diagnóstico, ou seja descobrir a causa da falha. Por último surge a etapa de correcção visando, se possível, a eliminação da falha.

A identificação de falhas passa também pela construção de um modelo da rede onde se possa avaliar o tipo de serviços e dispositivos existentes na rede bem como o seu comportamento. Estes modelos, de acordo com paradigmas denominados clássicos, poderão ser construídos por intermédio de técnicas de convite/resposta e por análise de eventos. Estas técnicas são, no entanto, bastante sensíveis ao aumento de dimensão da rede e a eventuais estrangulamentos, pelo que deram lugar ao aparecimento de propostas que sugerem o seu melhoramento ou mesmo substituição por agentes móveis.

Neste sentido, será necessário responder a questões como: quantos agentes de monitorização serão necessários? Onde os colocar? Quais serão responsáveis por que agentes de gestão? As operações serão executadas centralmente ou delegadas a um conjunto de agentes? Como será efectuada a comunicação entre agentes? Existem recursos suficientes para suportar os agentes?

A sobrecarga reflecte-se em recursos como a carga de processador, ocupação de memória e tráfego gerado. Com base nestes parâmetros, Abdu *et al.* [Abdu99] definem um modelo matemático que procura minimizar a sobrecarga causada pela monitorização de acordo com os requisitos do utilizador e a capacidade do sistema. Este modelo permite definir a quantidade, o tipo e a função dos agentes a utilizar em determinadas circunstâncias de forma a optimizar a utilização de recursos.

Liotta *et al.* [Liotta99] sugerem uma abordagem hierárquica de forma a mais facilmente poder delegar a responsabilidade de monitorização sobre ramos específicos da árvore de agentes. As capacidades de clonagem dos agentes móveis são também utilizadas para equilíbrio de carga e melhor gestão de recursos.

O comportamento de cada agente, com inspiração em alguns modelos químicos e biológicos, poderá ser aproximado ao de uma formiga, apresentando individualmente a capacidade para realizar algumas funções básicas. Esta medida tem a vantagem de manter o agente simples, residindo a inteligência global na comunidade. O principal mecanismo de coordenação entre agentes é baseado na capacidade que as feromonas têm de marcar um caminho ou realçar um trilho. À semelhança da comunidade de formigas, quanto mais forte for a marca do trilho mais alta será a probabilidade de encontrar uma fonte de alimento. A localização de falhas é consequência do movimento dos agentes, uma vez que a sua passagem assinala as rotas e ligações mais ou menos favoráveis à comunicação [White98].

Hassan Ku *et al.* [Ku97] argumentam que, por outro lado, a utilização de agentes móveis com diferentes graus de inteligência permite diminuir o tempo de resposta e aumentar a

flexibilidade do sistema embora à custa de um aumento da sua complexidade e dimensão. Cada agente toma decisões de forma dinâmica, tais como decidir sobre o próximo destino, optimizar o itinerário e detectar falhas à medida que se desloca. Os agentes poderão reunir mais capacidade de processamento e um maior conhecimento individual.

A gestão de segurança visa proteger informação sensível nos dispositivos ligados à rede por intermédio de controlo de acesso. Em primeiro lugar, é necessário identificar qual é a informação sensível. De seguida é necessário identificar os pontos de acesso, torná-los seguros e mantê-los seguros. Apesar de não se encontrar trabalho representativo que mencione a gestão de segurança com base em agentes móveis, há um conjunto de medidas que podem beneficiar da sua utilização. Uma delas é a actualização remota e automática de software, contribuindo para uma mais fácil correcção de falhas de segurança devidas a aplicações desactualizadas ou com vulnerabilidades.

A gestão de taxação implica medir a utilização de recursos de rede de forma a estabelecer medidas, avaliar quotas, determinar custos e apresentar respectivas facturas aos utilizadores. Nesta área, os agentes móveis podem ser utilizados na optimização do mecanismo de aquisição de informação devido à interacção local com a fonte.

A gestão de desempenho procura resolver possíveis problemas de congestionamento, de forma a manter os recursos de rede suficientemente livres para serem utilizados em qualquer momento (i. e. acessíveis). Para o conseguir, será necessário manter um conhecimento actualizado da utilização dos dispositivos de rede e das respectivas ligações. Esta informação será analisada para levantamento de tendências de utilização e, caso estas excedam limites predefinidos, seguir-se-á uma etapa de simulação para descobrir de que forma a rede poderá ser alterada para maximizar o seu desempenho.

Um problema clássico de desempenho é a descoberta do melhor caminho ao longo de uma malha de encaminhadores de forma a minimizar o número de saltos, minimizar o custo ou equilibrar tráfego, desviando-o para ligações menos congestionadas. A resposta a este problema, mesmo em redes estáticas, nem sempre é fácil de obter. Em redes dinâmicas (por exemplo, redes sem fios [Amaro01], redes *ad hoc* [Kawaguchi99]) ou em redes em que a probabilidade de falhas é elevada, o problema é ainda mais complexo de resolver. Alguns autores defendem também aqui a utilização de modelos semelhantes ao de comunidades de formigas. Quando um pedido atinge um determinado servidor este lança uma quantidade de agentes móveis. Quando um agente chega ao destino, regressa à origem marcando o caminho percorrido. Quando todos os agentes tiverem regressado o servidor poderá efectuar cálculos de forma a descobrir qual o melhor caminho. Apesar de conceptualmente simples, este algoritmo poderá ser optimizado em termos de número de agentes e de saltos [Sum99].

Poder-se-á pensar que esta abordagem não é muito eficiente devido à sobrecarga introduzida pelo tamanho do agente, quando comparado com mensagens protocolares mais simples como, por exemplo, as geradas pelo comando `ping`. Na realidade, apesar do tamanho de cada mensagem ser diferente também o será o número de mensagens trocadas (Figura 6.1).

A Figura 6.1 apresenta um caso de descoberta de encaminhadores entre duas estações, uma no papel de servidor e outra no papel de cliente. Na parte superior da figura, a descoberta de encaminhadores (representados por um círculo cinzento) é feita com base em técnicas de convite/resposta. Terão de ser emitidos convites individualmente para cada encaminhador até ao cliente e comparar respectivos parâmetros de comunicação. Cada mensagem passa por todos os encaminhadores que se encontram a montante do alvo.

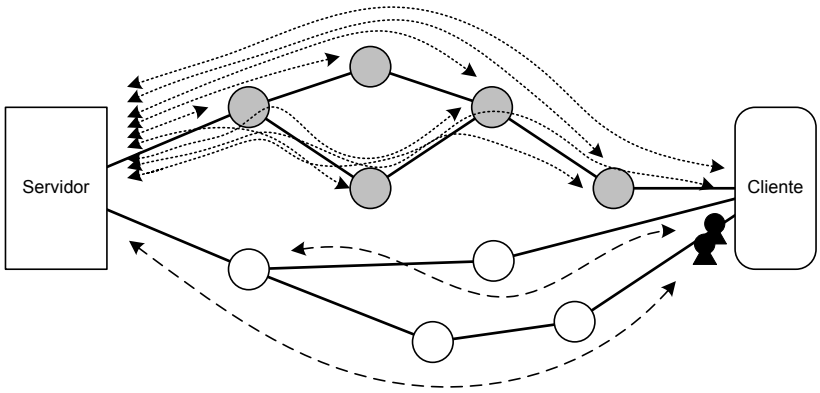


Figura 6.1 – Descoberta de nós de rede.

Na parte inferior da figura são utilizados agentes móveis que, após gerados pelo servidor, deslocam-se ao longo dos encaminhadores (representados por um círculo branco). Se em cada cruzamento for utilizada a técnica de clonagem a sobrecarga será ainda menor uma vez que o novo agente não terá de percorrer o troço de rede a montante.

Ainda relativamente a questões de desempenho são sugeridas técnicas que permitem diminuir a sobrecarga causada pela monitorização de informação de gestão. Gavalas *et al.* [Gavalas00] sustentam que os principais factores que contribuem para a sobrecarga são o tamanho do agente móvel e a utilização de métodos inadequados para o processamento e filtragem remota de informação. Relativamente a este último aspecto são sugeridas técnicas para agregar valores de várias MIBs em valores mais significativos através da aplicação de expressões matemáticas, adquirir tabelas SNMP de forma atómica e filtrar o conteúdo de tabelas.

O número e variedade de propostas para a utilização de agentes móveis na gestão de redes cresce continuamente, não tendo, no entanto, surgido ainda uma aplicação verdadeiramente revolucionária. Para melhor perceber o impacto que a utilização de agentes móveis pode ter é necessário efectuar estudos e comparações com outro tipo de tecnologia. Este passo permite avaliar em que medida e em que situações os agentes móveis poderão complementar ou mesmo substituir outros tipos de tecnologia.

Neste âmbito, interessa, antes de mais, minimizar a largura de banda utilizada para a gestão da rede. Apesar de os agentes móveis terem a possibilidade de efectuarem compressão semântica de informação, ou seja, de extraírem informação útil de grandes quantidades de dados, podem nem sempre ser a melhor opção. Existem situações em que as vantagens da utilização de agentes móveis parecem óbvias, como a pesquisa de valores em tabelas SNMP, evitando transferir toda a tabela para a estação de gestão central. Outras situações podem ser mais facilmente avaliadas se for construído um modelo de tráfego de gestão e comparado com o tráfego causado pelos agentes móveis [Baldi98]. Rubinstein e Duarte [Rubinstein99], por outro lado, apresentam simulações de tarefas de gestão de forma a comparar os efeitos de latência e de largura de banda numa ligação entre os agentes de gestão e a plataforma central.

As conclusões obtidas por ambos são semelhantes. As vantagens de utilização de agentes móveis dependem fortemente das características da rede, nomeadamente do custo, do número de nós, tipo de protocolo e do tipo de tarefa de gestão a desempenhar, ou seja, a possibilidade de compressão semântica, quantidade e frequência de interacções, complexidade da tarefa e dimensão dos pedidos. Resumindo, os autores apresentam resultados que mostram os agentes

móveis menos sensíveis à latência e a possíveis condições de estrangulamento da ligação apresentando, no entanto, uma sobrecarga relativamente ao SNMP quando a quantidade de informação trocada entre a estação e os nós de gestão é reduzida.

Apesar da quantidade de estudos existentes, o comportamento dos agentes móveis em gestão de redes ainda não é bem compreendido. Por este motivo, Simões *et al.* [Simões02] definiram um conjunto extenso de cenários concretos de gestão para obterem uma visão alargada do funcionamento geral do sistema de gestão com base em agentes móveis. Este trabalho inclui comparações e análise de distribuição de operações de gestão em cenários do tipo estático centralizado, migratório, migratório delegado, *master/worker* e estático delegado.

A tecnologia de implementação de agentes também influencia as condições em que estes devem ser utilizados, seja pela sobrecarga introduzida pelos protocolos de migração ou pelo tipo de linguagem (formato de dados, dimensão do código). Neste sentido, é comum surgirem dúvidas sobre o tipo de plataforma a utilizar.

Em termos de modelo de desenvolvimento podem também surgir dúvidas envolvendo outras soluções, tipicamente orientadas para os sistemas distribuídos tais como RMI e CORBA. Ismail *et al.* [Ismail98] apresentam um estudo que incide sobre medições efectuadas em ambiente real entre RMI, uma plataforma mínima de agentes móveis, a plataforma de agentes móveis Aglets [Aglets] e a plataforma AAA [Bellissard99]. Este estudo, restringido a aplicações de *data mining* e de optimização da transferência de documentos por intermédio de compressão de dados, permite obter valores de custo mínimo para a migração de agentes e o ponto a partir do qual é benéfica a sua utilização. A conclusão principal é a de que os agentes móveis, para estes tipos de aplicações e em certas situações, permitem poupar largura de banda quando comparados com paradigmas cliente/servidor.

6.2.1 Agentes móveis em ambientes SNMP

A maior parte das propostas apresentadas anteriormente, apesar das melhorias introduzidas, ignora o mérito dos modelos e protocolos denominados clássicos. A interoperabilidade com as tecnologias instaladas, como o SNMP, CORBA ou CMIP, contribui para um maior impacto da utilização de agentes móveis no panorama da gestão de redes.

Estas tecnologias providenciam acesso directo a serviços e informação de gestão, pelo que, em várias situações funcionarão como o sistema sensorial dos agentes móveis. Poder-se-á dizer que sem a possibilidade de recorrer a elas, os agentes móveis não terão possibilidade de interagir com os elementos de rede no sentido de adquirirem informação ou alterarem parâmetros de funcionamento. Por outro lado, a integração dos dois tipos de tecnologia permite obter modelos mais funcionais ou mais eficientes para a gestão e, simultaneamente, dotar os agentes móveis com a capacidade de lidar com sistemas de gestão existentes.

Em paralelo com a abordagem anterior, é importante manter a interoperabilidade, não só com os agentes de gestão, mas também com as aplicações de gestão. Por outras palavras, os agentes móveis deverão ser vistos como qualquer outro elemento de rede, ou seja, disponibilizarem interfaces para poderem ser geridos de forma semelhante (Figura 6.2).

Assim, os agentes móveis poderão não só contactar os agentes de gestão como também ser contactados por aplicações de gestão. Neste cenário podem ser distinguidas duas etapas de integração. Para que os agentes móveis possam interagir com a rede será necessário dotar os agentes móveis ou a sua plataforma de suporte (agência) com capacidade de enviar comandos e receber respostas no protocolo de gestão, tipicamente, SNMP. Será necessário associar uma API que encapsule o protocolo de gestão ao agente móvel, o que implica aumentar o tamanho

do código do agente ou da agência. Por outro lado, os agentes móveis deverão responder a comandos provenientes de aplicações de gestão no protocolo específico. Para o efeito, além da API adequada, será necessário manter uma estrutura de informação semelhante à implementada nos agentes de gestão típicos – uma MIB.

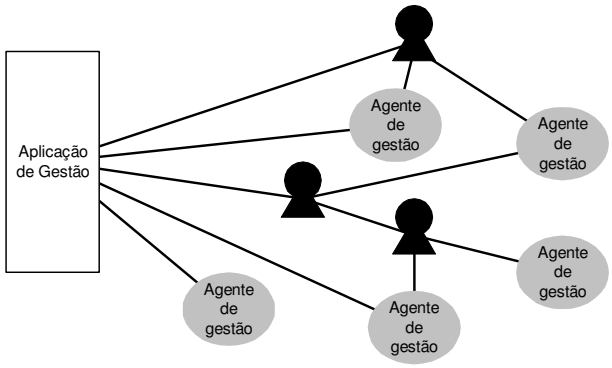


Figura 6.2 – Interoperabilidade entre agentes móveis e sistemas de gestão.

A primeira vertente de integração, ou seja, a forma como os agentes móveis poderão contactar agentes de gestão SNMP poderá passar por dotar os agentes móveis com capacidade de gestão SNMP. Um agente móvel teria assim a capacidade para gerar comandos, codificar mensagens e receber e descodificar as respostas. A maior vantagem desta abordagem é a possibilidade de contactar directamente qualquer agente SNMP. As desvantagens incluem o aumento da complexidade e do tamanho do agente móvel. Para reduzir ou mesmo eliminar estes inconvenientes, Pagurek *et al.* [Pagurek00] sugerem a utilização de um protocolo simplificado que permite ao agente móvel consultar e alterar a informação do agente de gestão. A função deste protocolo é simétrica ao DPI (*Distributed Protocol Interface*) [RFC1592] ou ao mais recente AgentX [RFC2257] e é apresentado com o nome RDPI (*Reverse Distributed Protocol Interface*). Para a vertente simétrica, ou seja, para que as aplicações de gestão possam contactar o agente móvel, os autores sugerem que este se associe a um agente SNMP existente através de DPI.

O inconveniente desta abordagem é a necessidade dos agentes SNMP implementarem os protocolos DPI e RDPI, o que nem sempre acontece. Este facto vem dificultar a adopção deste modelo, pois um grande número de agentes SNMP já instalados não incluem estes mecanismos de extensão.

Para eliminar estes inconvenientes, Simões *et al.* [Simões01] sugerem utilizar SNMP em ambas as situações, ou seja, para trocar informação de gestão com agentes SNMP e para que as aplicações de gestão possam monitorar e controlar os agentes móveis. Estes mecanismos foram criados para a plataforma de agentes móveis JAMES [Simões99], desenvolvida com o objectivo concreto de gestão de redes. Cada agência contém um módulo de gestão SNMP, o que dá a possibilidade aos agentes móveis de invocarem serviços de gestão directamente a partir da API da plataforma. Esta abordagem permite utilizar agentes móveis na gestão de elementos de rede que não suportem uma agência JAMES ou, por razões de segurança, não permitam outro modo de acesso à informação de gestão além de SNMP. Por outro lado, cada agência JAMES contém um agente SNMP específico, o que lhe permite ser gerida por uma aplicação de gestão SNMP. Este agente SNMP possui mecanismos de extensão para integrar informação proveniente de agentes móveis que aí se desloquem e, portanto, serem eles próprios controlados e monitorizados da mesma forma. O agente SNMP JAMES precisa de uma MIB específica (JAMES-MIB) para lidar com as características da plataforma [Simões01].

Outra abordagem para o problema de gestão de agências e de agentes móveis passa por traduzir a funcionalidade MAF para SNMP por intermédio de uma MIB específica – MAF-MIB [Lopes01b]. A MIB é implementada por um adaptador, quer interno quer externo à plataforma de agentes móveis, que traduz os comandos SNMP em invocações CORBA sobre interfaces MAF [Lopes01a]. As aplicações de gestão SNMP poderão, portanto, gerir qualquer plataforma de agentes móveis compatível com MAF, não ficando o sistema dependente da plataforma de agentes móveis de um determinado fabricante. Esta solução será descrita em maior detalhe na secção 6.4.

6.3 Gestão de agentes móveis

A utilização de agentes móveis para a gestão de redes poderá trazer vantagens relacionadas com a redução de tráfego, eficiência, continuação de operação em situações sem conectividade, equilíbrio de carga, entre outras [Goldszmidt98][Lopes00c]. Apesar de poderem tornar o sistema de gestão mais flexível e eficiente, a utilização de agentes móveis também introduz alguns inconvenientes, nomeadamente, a sobrecarga em termos de gestão [Breugst99], a dificuldade de utilização e, eventualmente, algumas ameaças [Kaasinen99]. Obviamente, estes factos provocam a necessidade de gerir os próprios agentes móveis.

As plataformas de agentes móveis existentes, quer em domínio público quer em produtos comerciais, apresentam arquitecturas e implementações diferentes, uma vez que se destinam a diferentes sistemas operativos, estão baseadas em linguagens de programação diferentes e possuem mecanismos de gestão incompatíveis. No entanto, estão bem identificados os requisitos comuns que todas terão de suportar. As plataformas deverão implementar mecanismos que lhes permitam criar, localizar, suspender, reactivar e terminar agentes. Terão também de prever mecanismos de apoio à mobilidade, ou seja, criação remota e migração de agentes. Por último, será necessário identificar inequivocamente cada agente e cada agência e manter mecanismos de comunicação entre agentes.

A generalidade das plataformas de agentes móveis, como Grasshopper (http://www.grasshopper.de/) ou Aglets (http://aglets.sourceforge.net/), associam uma ferramenta gráfica em cada nó (i. e., a agência) que permite ao utilizador ver quais os agentes móveis em execução nesse local (Figura 6.3).

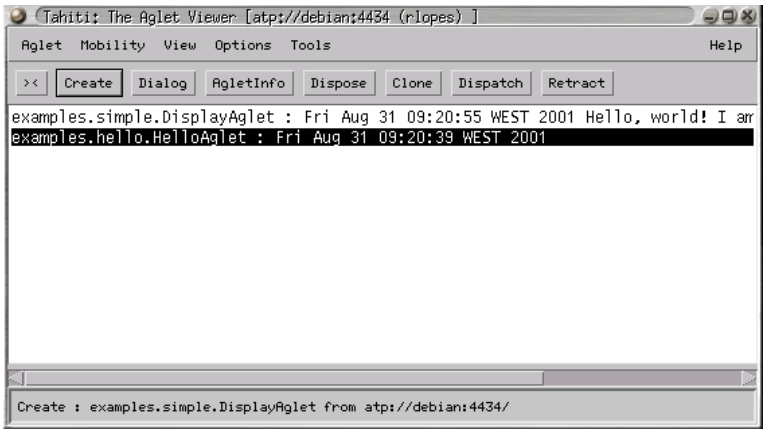


Figura 6.3 – Representação gráfica de uma agência Aglet.

Além disso, permitem visualizar outras características da plataforma, como os contextos de execução de agentes, ou seja, os lugares (Figura 6.4).

Algumas plataformas disponibilizam, inclusivamente, uma outra ferramenta gráfica associada ao serviço de directoria (i. e., a região) e que permite ao utilizador ter uma visão global do sistema em termos de agências e de agentes (Figura 6.5). Esta representação é actualizada cada vez que um determinado agente móvel alcança ou abandona um nó. Associada à representação gráfica de cada agente, lugar ou agência está um conjunto de opções, que permite ao utilizador consultar ou alterar o seu estado, no sentido de os suspender, retomar, iniciar, terminar ou eliminar.

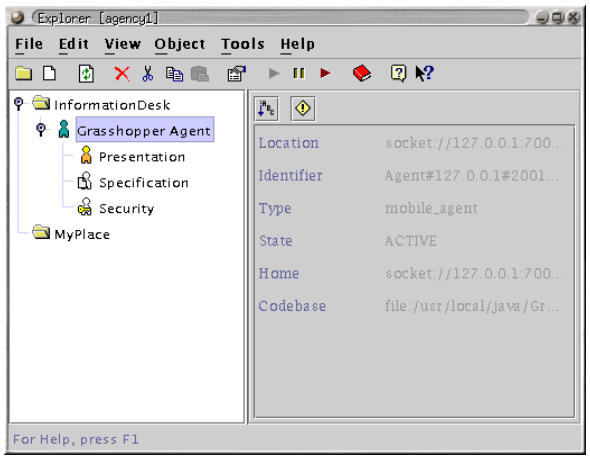


Figura 6.4 – Representação gráfica de uma agência Grasshopper.

No entanto, apesar de implementarem operações comuns, as diferenças entre plataformas tornam a interacção difícil, senão impossível, e agravam a sua adopção na prática.

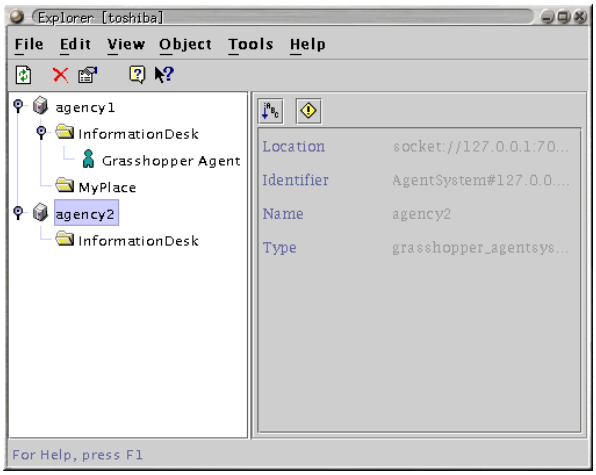


Figura 6.5 – Representação gráfica de uma região Grasshopper.

Choy *et al.* [Choy99] sugerem um sistema de gestão de agentes móveis com base no conceito de objecto de gestão, definido na especificação OSI. Os objectos de gestão são encapsulados em agentes móveis específicos, designados por agentes de objectos de gestão. Estes têm funcionalidade semelhante aos agentes tradicionais OSI e podem desempenhar funções de gestão, o que permite criar modelos distribuídos de gestão além de manter compatibilidade com soluções de gestão já instaladas.

A existência de uma estrutura que permite a mobilidade de agentes entre diferentes plataformas além de definir operações de gestão comuns, como a MAF, vem possibilitar a utilização de ferramentas de gestão únicas.

Actualmente, são conhecidas quatro plataformas compatíveis com MAF, nomeadamente, Grasshopper produto da empresa IKV++ Technologies [Grasshopper], SOMA, desenvolvido na Universidade de Bolonha com o objectivo de ser uma plataforma de agentes móveis segura e aberta [SOMA], MAP Agent System, desenvolvido no Laboratório de Multimédia e Sistemas Distribuídos das Universidades de Catania e Messina [MAP] e MobiliTools, da France Telecom [Dillenseger00]. A gestão de uma plataforma MAF é efectuada através da invocação de métodos em objectos CORBA do tipo `MAFFinder` e `MAFAgentSystem` – ver secção 3.1.5 (Figura 6.6).

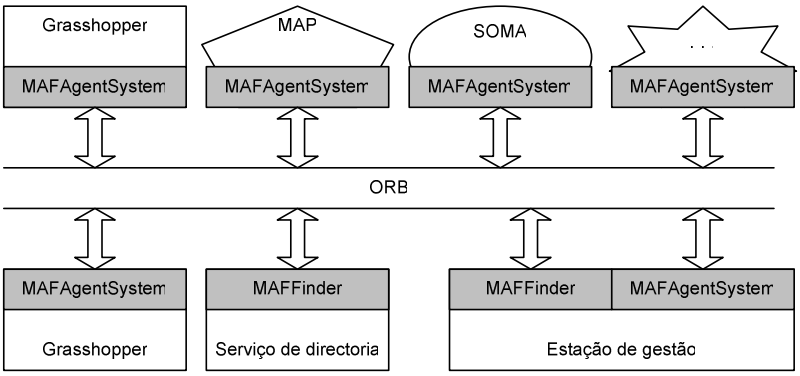


Figura 6.6 – Gestão de sistemas MAF.

O facto de se tratarem de objectos CORBA permite que as plataformas sejam implementadas em qualquer linguagem de programação, desde que esteja definida a sua correspondência com IDL. O mesmo se passa com a aplicação de gestão [Di Pietro00].

6.3.1 Gestão de agentes móveis por SNMP

Do ponto de vista do utilizador, habituado a ferramentas de gestão baseadas em SNMP, é de todo vantajoso poder visualizar o estado de funcionamento dos agentes móveis bem como poder modificar o seu comportamento em conjunto com a gestão de aplicações e de equipamento de rede. Desta forma, os próprios agentes, lugares e agências passam a "fazer parte" da rede do ponto de vista de administração, facilitando assim a integração de tecnologia.

A gestão de agentes móveis através da agência vai de encontro à arquitectura MAF, em que as próprias interfaces se encontram acessíveis nas agências e na região. Aproveitando este facto, o desenvolvimento de um conversor SNMP para MAF permite gerir qualquer plataforma de agentes móveis compatível com MAF sem a sobrecarregar com a adição de código, uma vez que o agente SNMP fica isolado da plataforma pela invocação remota de métodos da arquitectura CORBA (Figura 6.7).

Nesta arquitectura, o conversor SNMP para MAF assegura, a partir da mensagem recebida, a correcta identificação dos métodos a invocar nas interfaces MAF. Em termos de segurança, a utilização da arquitectura SNMPv3 garante a autenticação, privacidade e controlo de acesso de acordo com o *User-based Security Model* [RFC2573] e o *View-based Access Control Model*

[RFC2575]. Em resumo, o conversor realiza a invocação de métodos MAF a partir de mensagens SNMP.

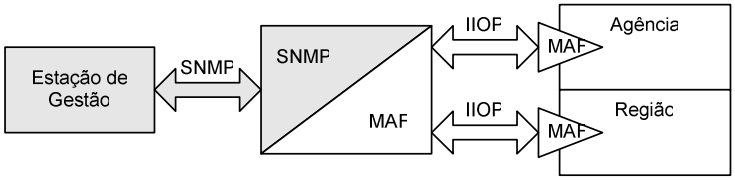


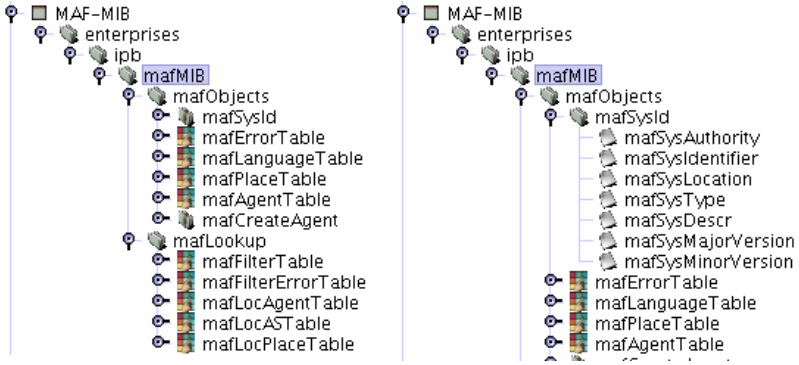
Figura 6.7 – Conversor SNMP para MAF.

6.4 MAF-MIB

Para a concretização do conversor foi fundamental conceber um módulo MIB para identificar o método MAF a invocar e os seus argumentos de forma a conseguir uma gestão coerente da plataforma de agentes móveis. Por exemplo, quando a estação de gestão (NMS) envia um comando `set`, a mensagem transporta um identificador (OID) e um valor. O identificador assinala um objecto particular da MIB que, por sua vez, se encontra associado ao método MAF apropriado. O valor é utilizado como argumento do método. Por exemplo, se a intenção do utilizador é criar um agente terá de enviar um `set` com os argumentos necessários, ou seja, a localização inicial do agente (*home*), o nome da classe (c*lass name*) e a localização do código (*code base*).

A MIB foi desenhada com base no conjunto de métodos definidos nas interfaces MAF, nos seus argumentos e resultados. Para cada argumento é definido um objecto de gestão e agrupado aos outros objectos que estão associados ao mesmo conceito. O tipo de agrupamento resulta numa tabela, caso os objectos incidam sobre várias instâncias (listagem de agentes, por exemplo), ou num grupo de objectos para operações individualizadas, como a criação de agentes móveis. Os tipos de dados utilizados na definição dos objectos de gestão são baseados aos definidos pelas interfaces MAF de forma a simplificar a tarefa de conversão entre comandos SNMP e invocações MAF.

De acordo com a especificação normativa, a MAF-MIB foi estruturada em dois grandes grupos: *mafObjects* e *mafLookup* (Figura 6.8 - a). No Anexo C a sintaxe desta MIB está apresentada na totalidade.



a) Grupos *mafObjects* e *mafLookup*. b) Detalhe do *mafSysId*.

Figura 6.8 – Estrutura global da MAF-MIB.

O grupo *mafObjects* está relacionado com as tarefas de gestão disponibilizadas pela interface `MAFAgentSystem` e possui objectos que permitem, em primeiro lugar, consultar parâmetros de funcionamento e obter características da plataforma, tais como a autoridade, identificação, tipo, descrição e versão da agência (*mafSysID*) (Figura 6.8 - b).

A especificação MAF prevê também mecanismos para identificação das linguagens suportadas pela agência (*mafLanguageTable*), registo dos lugares disponíveis (*mafPlaceTable*) e uma tabela de agentes que se encontram presentemente na agência (*mafAgentTable*). O índice para esta última tabela é composto por três campos: autoridade, identificação e tipo de plataforma. Assim assegura-se a unicidade de cada nome no universo dos agentes.

Além da operação de consulta, é possível ainda actuar activamente sobre os agentes, nomeadamente podem-se suspender, terminar ou reactivar a sua execução (*mafAgentRequiredStatus*) usando para o efeito a *mafAgentTable*. A criação de agentes é efectuada no grupo *mafCreateAgent*. Para isto, é necessário fornecer informação sobre a autoridade sob a qual fica o agente, a sua identidade, o nome da classe que irá ser instanciada, a localização do código e os argumentos a passar ao construtor do agente (Figura 6.9).

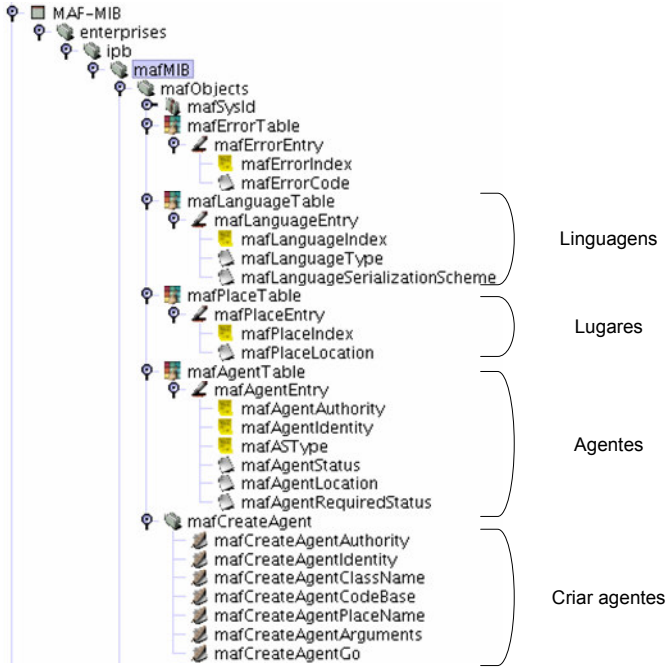


Figura 6.9 – Detalhes do grupo *mafObjects*.

Durante a migração de agentes podem ocorrer diversos erros que são descritos na tabela *mafErrorTable*. Estes poderão ser do seguinte tipo:

- Erros relacionados com ausência de serviço de nomes CORBA: `namingServiceNotFound`;
- Erros relacionados com o serviço de pesquisa da MAF: `finderNotFound`;
- Erros na criação de agentes: `classUnknown`, `argumentInvalid`, `deserializationFailed`, `mafExtendedException`;

- Erros relacionados com o estado (ciclo de vida) do agente: `agentNotFound`, `resumeFailed`, `agentAlreadyRunning`, `terminateFailed`, `suspendFailed`, `agentAlreadySuspended`;
- Outros erros não enquadrados nos anteriores: `genericError`.

O utilizador baseia-se nesta informação para descobrir o que aconteceu de errado em determinada situação ou com determinado agente.

Como exemplo de utilização da MIB, suponha-se que o utilizador necessita avaliar o estado de conectividade de vários troços da rede e que desenvolve um agente para esse propósito. Para lançar o agente irá actuar sobre o grupo *mafCreateAgent* no sentido de fornecer a informação necessária para que o agente seja criado, nomeadamente:

- *mafCreateAgentClassName*`.0` – "nms.mobile.ConnectivityEvaluation"
- *mafCreateAgentCodeBase*`.0` – "http://code.server.enterprise/nms/classes"
- *mafCreateAgentPlaceName*`.0` – "pingPlace"
- *mafCreateAgentGo*`.0` – 1

Após a criação, o agente inicia o teste de conectividade armazenando o tempo de resposta nos diversos troços. De seguida, por intermédio de consultas efectuadas sobre a região, o agente percorre as agências disponíveis continuando a operação até ter visitado todos os pontos disponíveis.

Supondo que em determinado momento o utilizador decide que o teste de conectividade não é mais relevante pode optar por terminar o agente. Para o efeito, envia um `set` à *mafAgentTable* a solicitar o estado `terminate`. Esta mensagem será traduzida para uma invocação MAF e o agente será eliminado.

O grupo *mafLookup* assenta nos serviços de pesquisa previstos na interface `MAFFinder`, nomeadamente pesquisa de agentes, lugares e agências (Figura 6.10).

Este serviço reveste-se de algumas características importantes relacionadas com a localização de entidades num sistema de agentes móveis. Para tornar o serviço mais versátil foram acrescentadas funcionalidades ao sistema sem, no entanto, sacrificar a compatibilidade com a MAF:

- Flexibilidade de pesquisa – os motores de pesquisa de Internet utilizam chaves que permitem especificar com grande flexibilidade os termos a pesquisar. O mesmo tipo de abordagem foi seguida para pesquisar agentes móveis numa região.
- Acesso concorrencial – várias operações de pesquisa podem ser iniciadas em simultâneo por várias aplicações de gestão distintas. O sistema de pesquisa terá de suportar este tipo de acesso.
- Simplicidade de utilização – o sistema de pesquisa deverá ser simples de utilizar.

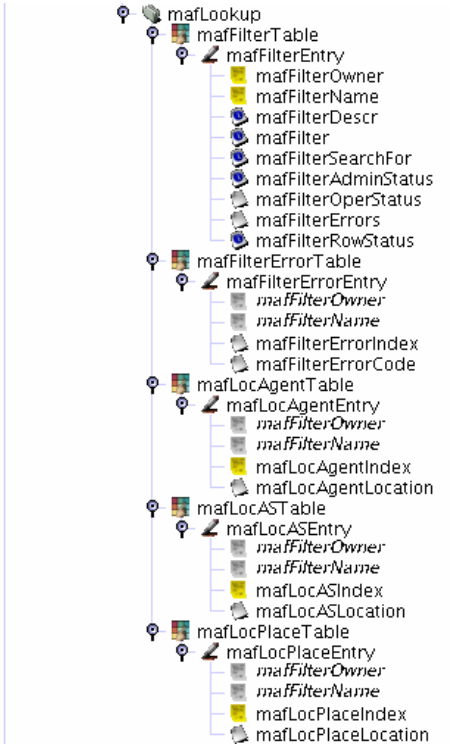


Figura 6.10 – Detalhes do grupo *mafLookup*.

Estas opções resultaram na definição de filtros de pesquisa, armazenados na tabela *mafFilterTable*. Esta mantém vários filtros que são indexados de acordo com a aplicação de gestão (*mafFilterOwner*) e por nome (*mafFilterName*). O filtro é composto por uma descrição, pela chave de pesquisa e pelo tipo de elemento a pesquisar. A chave corresponde a uma cadeia de caracteres armazenada na coluna *mafFilter* e que especifica a condição. Por exemplo, a localização de lugares é efectuada unicamente por intermédio do nome. Para agentes e agências podem ser importantes outros parâmetros como a linguagem suportada ou o método de serialização. Por estes motivos a chave de pesquisa é baseada numa sintaxe própria de forma a permitir pesquisas mais complexas (Figura 6.11).

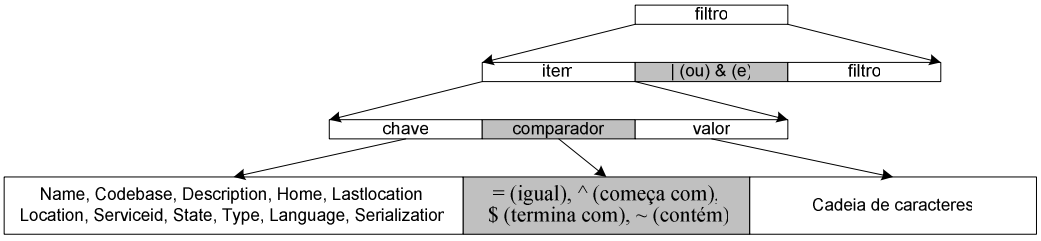


Figura 6.11 – Semântica e sintaxe da chave de pesquisa.

A representação em BNF da sintaxe e da semântica da chave de pesquisa é:

```
filtro ::= item ["|"|"&" filtro]
item ::= chave ("=" | "^" | "$" | "~") valor
chave ::= "Name" | "Codebase" | "Description" | "Home" | "LastLocation" | "Location"
        | "ServiceID" | "State" | "Type" | "Language" | "Serialization"
valor ::= string
```

O filtro "`Name=pingAgent&Codebase~myHost`" é um exemplo de pesquisa de um agente com o nome "pingAgent" e que possui as classes em "myHost".

Por indicação da coluna *mafFilterSearchFor*, a chave de pesquisa pode ser aplicada para pesquisar lugares, agentes e agências. Este é um objecto do tipo `BITS`, pelo que pode assinalar apenas um ou vários tipos.

O filtro pode ser activado (`searching`), ou seja, forçado a realizar novamente a pesquisa, ou desactivado (`stopped`) por intermédio da coluna *mafFilterAdminStatus*. De forma semelhante poderá ser consultado o seu estado por intermédio da coluna *mafFilterOperStatus*: `searching`, `stopped`, `finished`.

Os filtros são criados e eliminados por intermédio da coluna *mafFilterRowStatus*.

De acordo com o tipo de pesquisa a realizar (*mafFilterSearchFor*), os resultados são obtidos nas tabelas *mafLocPlaceTable*, *mafLocASTable* e/ou *mafLocAgentTable*, respectivamente para lugares, agências e/ou agentes. Estas encontram-se associadas ao filtro respectivo pelo índice *mafFilterOwner* e *mafFilterName* embora possam conter várias entradas para cada filtro (*mafLocPlaceIndex*, *mafLocASIndex* e/ou *mafLocAgentIndex*).

Caso algum erro ocorra, nomeadamente, devido a sintaxe desconhecida ou algum outro problema, este é assinalado na tabela *mafFilterErrorTable*.

Se o filtro for eliminado, todos os resultados dependentes deste serão, de igual forma, eliminados da(s) tabela(s) respectiva(s).

6.4.1 O conversor SNMP para MAF

A MIB apresentada anteriormente constitui a base de suporte ao conversor SNMP MAF que é apresentado nesta secção. O conversor é constituído por três grandes blocos (Figura 6.12).

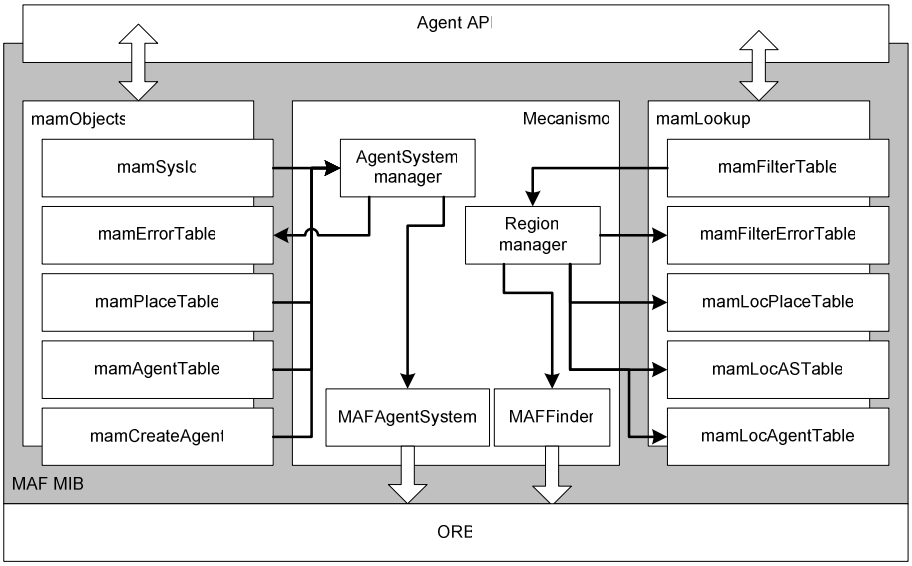


Figura 6.12 – Diagrama de blocos da implementação da MAF-MIB.

Numa posição central, encontra-se o mecanismo que define o comportamento do conversor. Este tem unicamente a função de efectuar invocações sobre interfaces remotas MAF (`MAFAgentSystem` e `MAFFinder`), povoando posteriormente as tabelas associadas com os

resultados da invocação. De igual forma, os parâmetros passados durante a invocação são obtidos a partir dos valores contidos nos objectos da MIB.

A Agent API é responsável pela comunicação com a aplicação de gestão e, consequentemente, com o utilizador, bem como pela organização de objectos e pelo serviço de persistência de informação. Por outro lado, a comunicação com a plataforma de agentes móveis é efectuada por intermédio de um ORB (*Object Request Broker*).

Nesta implementação foi utilizada a plataforma de agentes móveis Grasshopper com o módulo de compatibilidade MAF (http://www.grasshopper.de/). Foi ainda utilizado o ORB do J2SDK bem como o serviço de nomes CORBA associado (`tnameserv`). Todos os outros recursos, nomeadamente os mecanismo SNMP e HTTP, encontram-se já disponíveis na Agent API.

Cada um dos objectos definidos na MIB é responsável por determinadas acções. Assim, o *mafLanguageType* é responsável por invocar o método MAF que lhe permite averiguar quais são as linguagens suportadas pelo sistema de agentes. Outros objectos têm outras responsabilidades, como por exemplo averiguar a versão ou pesquisar agentes.

Este cenário permite utilizar um simples MIB *Browser* para monitorar e controlar qualquer sistema de agentes compatível com MAF (Figura 6.13).

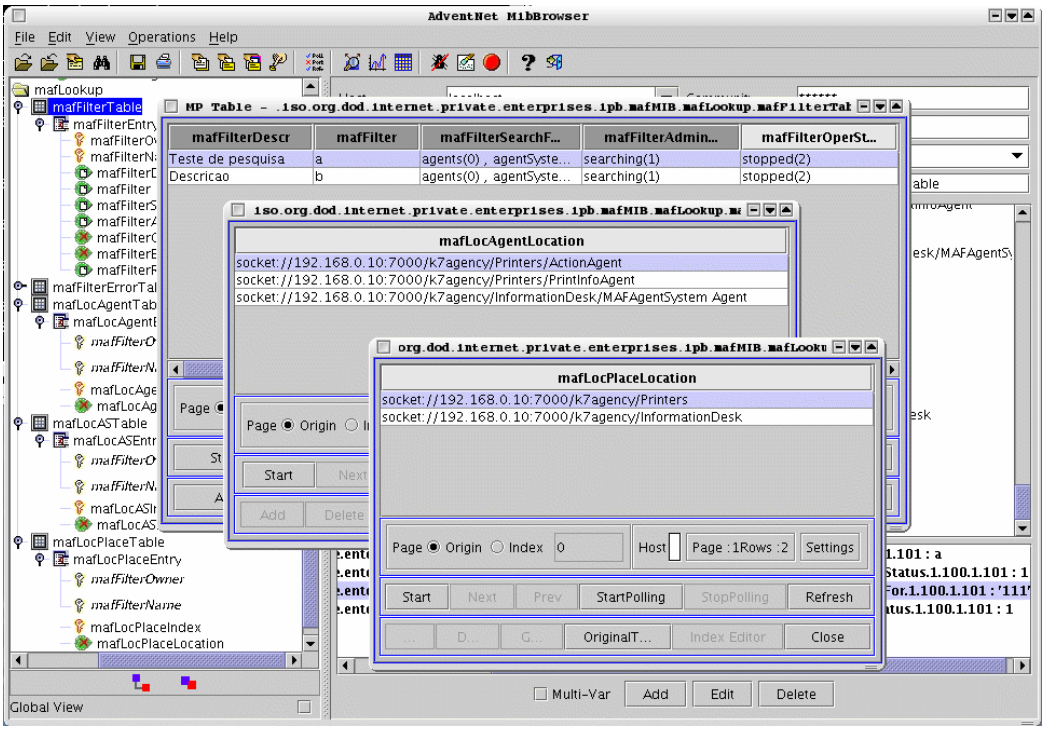


Figura 6.13 – Sessão de gestão de agentes via MAF-MIB e utilizando um MIB *browser*.

Actividades como, por exemplo, parar, reiniciar e mover agentes passam a poder ser efectuadas utilizando a mesma metáfora de gestão da rede tradicional.

De acordo com a especificação MAF é necessário um conversor para cada agência, tal como para o caso do equipamento de rede em que, tipicamente, cada elemento está associado a um agente SNMP.

As interacções entre a estação de gestão e a plataforma de agentes móveis envolvem, tipicamente, três tipos de comunicação: 1) entre o NMS/MIB Browser e o conversor é utilizado o SNMP; 2) este último utiliza o IIOP (*Internet InterORB Protocol*) para contactar a agência; 3) no interior da plataforma de agentes móveis são usados protocolos específicos (geralmente, *sockets*, SSL, RMI ou outros).

Para simplificar a gestão de plataformas MAF podem ser utilizadas outras ferramentas complementares ao *browser* de MIBs, mais adaptadas à gestão de agentes móveis. Um exemplo é apresentado no capítulo 7.

6.5 Conclusões

Este capítulo faz um apanhado genérico da gestão baseada em agentes móveis nas suas diversas áreas, nomeadamente gestão de configuração, gestão de falhas, gestão de desempenho, gestão de segurança e gestão de contabilização. Apesar do contributo que os agentes móveis possam trazer à gestão de redes, tais como a migração a pedido ou autónoma, a adaptação dinâmica de interfaces e serviços em sistemas remotos, a redução de dependência na constante disponibilidade de conectividade de rede, equilíbrio de carga e distribuição dinâmica de funções apresentam também a necessidade de serem geridos.

Tal como o equipamento de rede defeituoso pode agravar o funcionamento da rede, um agente móvel mal comportado poderá mostrar-se prejudicial. É importante poder monitorar e controlar o seu funcionamento bem como poder suspender, terminar ou destruir o agente em causa.

Neste capítulo foi também focada a problemática de integração da tecnologia de agentes móveis com modelos de gestão de redes clássicos. É importante reunir as melhores características de ambos os modelos e providenciar uma integração elegante entre tecnologias. A abordagem proposta sugere uma solução para o problema da integração da plataforma com sistemas de gestão de redes baseados em SNMP e apresenta mecanismos para gerir qualquer plataforma de agentes móveis compatíveis com MAF.

Foi definida uma MIB específica para converter os comandos SNMP em invocações sobre as interfaces MAF que, apesar de resolver o problema de gestão uniforme de plataformas de agentes móveis, não tira partido de todas as possibilidades de configuração que poderão existir, geralmente específicas de cada fabricante. No entanto, a interoperabilidade é de suma importância, pelo que a escolha entre a gestão particular e individual de possivelmente milhares de componentes é muitas vezes preterida em função da gestão global e uniforme.

Independentemente das abordagens aqui apresentadas, existe um conjunto de questões ainda por resolver. Os agentes móveis só terão plena utilidade se puder ser efectuada a sua integração com o equipamento e os serviços existentes. Será possível realizar esta integração de forma a não ser necessário introduzir novas estações centrais de gestão, mais protocolos e, de uma forma geral, forçar o utilizador a efectuar uma aprendizagem desnecessária? Tendo-se conseguido a integração, será prático representar os agentes móveis juntamente com o restante equipamento de rede no mesmo modelo gráfico? Neste caso, será que o aparecimento e desaparecimento de agentes devido à sua migração é comportável para este tipo de modelo? Que outro tipo de problemas poderão surgir com a utilização de aplicações parcialmente móveis? Que implicações, relacionadas com a segurança, estas trarão?

Espelho meu, haverá no meu reino alguém mais bela do que eu?

Em A Branca de Neve.

7 Interface com o Sistema de Gestão

A gestão de redes é o nome atribuído ao processo de controlo de redes de comunicação com a finalidade de maximizar a sua eficiência e produtividade. A arquitectura geral prevê a instalação de vários agentes com funções de instrumentação, estrategicamente colocados ao longo da rede. A informação recolhida é enviada para um posto de gestão por intermédio de um protocolo de gestão. Finalmente, o posto de gestão tem a responsabilidade de processar, interpretar e modificar a informação. Geralmente, possui ferramentas gráficas para apresentar a informação de gestão ao utilizador num formato conveniente.

O sistema de gestão terá de se adaptar às vagas tecnológicas que repetidamente surgem, bem como ao aumento da dimensão da rede de comunicações em termos de equipamento e de número de utilizadores. Como parte integrante, a interface com o sistema de gestão terá de modificar a forma de apresentação de informação de acordo com as alterações sofridas pelo sistema.

É também importante que a interface gráfica apresente o desenho e funcionalidade adequados à estrutura do sistema de gestão e aos agentes que a constituem. Por exemplo, a Schedule MIB prevê três modos de funcionamento que, se forem definidos por intermédio de uma ferramenta genérica do tipo *browser* de MIBs, poderão ser complexos de definir. Se for utilizada uma ferramenta específica, com clara definição dos modos de funcionamento e respectiva informação associada, será mais simples de configurar e mais resistente à ocorrência de erros de configuração.

Esta secção apresenta um sistema modular baseado em componentes e que visa essencialmente privilegiar a facilidade de actualização da ferramenta e a simplicidade de adição de novos módulos. Esta característica permite desenvolver sistemas de gestão capazes de lidar com a informação de gestão de forma coerente e global, ao contrário de outras ferramentas mais simples. O paradigma do *browser* de MIBs não é, de facto, o mais adequado para a gestão

de agentes complexos uma vez que não é suficientemente poderoso para apresentar as características das recentes e cada vez mais complexas MIBs como as que foram desenvolvidas durante o trabalho apresentado nesta tese, nomeadamente, a Schedule MIB, Event MIB e MAF-MIB.

7.1 Modelo

Tal como no sistema de gestão, a modularidade a nível de posto central é importante. Esta permite mais facilmente adaptar o sistema às alterações causadas pela introdução de nova tecnologia, ao aumento de dimensão da rede ou do número de utilizadores. A utilização de técnicas adequadas de programação, por exemplo recorrendo a paradigmas baseados em componentes, permite instalar módulos inicialmente inexistentes sem haver necessidade de compilar ou mesmo interromper o funcionamento da estação de gestão.

Cada componente encapsula ferramentas gráficas que complementam o posto central. Sempre que necessário, o componente pode ser instanciado de forma a aumentar a funcionalidade do posto central de gestão (Figura 7.1).

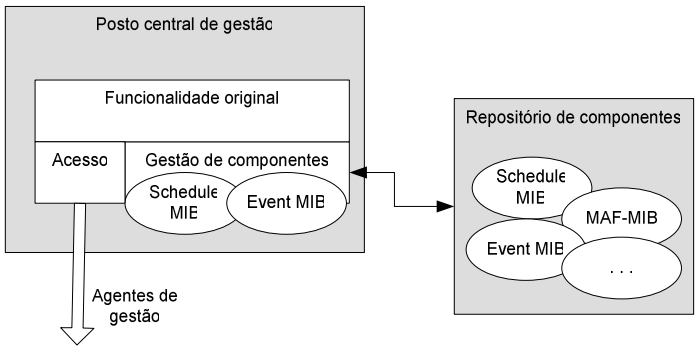


Figura 7.1 – Estação de gestão modular.

Os componentes podem ser instalados em qualquer momento no repositório de componentes. Este poderá ser uma base de dados, um servidor específico ou uma unidade de armazenamento de massa, como um disco. A estação de gestão, sempre que necessário ou a pedido do utilizador, poderá requisitar um componente do repositório e assim complementar a sua funcionalidade original.

O gestor de componentes tem a função de instanciar, analizar a funcionalidade acrescida em cada componente e apresentar uma interface comum de serviços à aplicação. A análise é efectuada em tempo de execução evitando desta forma ter de interromper o funcionamento da estação de gestão para invocar um novo serviço.

Além de possibilitar a instalação de novos componentes, a modularidade torna possível a actualização de apenas alguns componentes e isola serviços em unidades autónomas. Este processo permite definir colecções de componentes adaptados a situações de gestão específicas. Por exemplo, se uma rede contém apenas agentes do tipo MIB-II não é necessário definir uma aplicação contendo mais módulos, bastando apenas o módulo correspondente. A instanciação apenas dos componentes necessários resulta numa poupança de recursos computacionais, principalmente de memória dinâmica (RAM), optimizando o desempenho global da aplicação.

Do ponto de vista comercial, esta abordagem tem também algumas implicações, podendo o cliente adquirir apenas os módulos julgados indispensáveis e actualizar a ferramenta posteriormente ao nível das necessidades.

O posto central de gestão deverá apresentar ao utilizador a lista de componentes armazenados para que este possa, em qualquer altura, aperceber-se da funcionalidade instalada e invocar o serviço pretendido. Após a invocação de um serviço, por exemplo a configuração da Event MIB, o utilizador poderá terminar o serviço e regressar ao modo anterior de funcionamento. Para detectar estas opções, a aplicação terá de se dar a conhecer a cada componente, assim como os serviços que possui (Figura 7.2).

Figura 7.2 – Processo de instanciação de componentes.

Para validar esta arquitectura bem como para definir um conjunto de ferramentas gráficas específicas para a gestão dos módulos apresentados anteriormente, optou-se por desenvolver uma aplicação gráfica de gestão. Esta é constituída por uma ferramenta genérica construída em torno de um Editor de Macros SNMP, descrito à frente (Figura 7.3). Por macros SNMP consideram-se sequências de comandos SNMP, uma concretização do conceito de comandos de grupo (*batch files*).

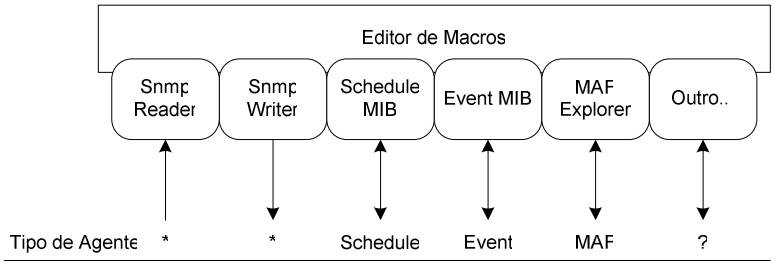


Figura 7.3 – Modelo simplificado do sistema gráfico.

Através dos mecanismos dinâmicos anteriormente referidos, a funcionalidade do editor de macros é complementada por componentes para a gestão dos módulos Schedule MIB, Event MIB e MAF-MIB. Cada um destes módulos pode funcionar de forma independente uma vez que possui capacidade autónoma para contactar (unicamente) os agentes respectivos, facto assinalado pelas setas verticais bidireccionais.

Além dos módulos referidos, encontram-se os módulos Snmp Reader e Snmp Writer. Estes possuem a capacidade de ler e escrever valores em agentes SNMP. É por intermédio destes módulos que o editor de macros contacta os agentes SNMP uma vez que não possui essa funcionalidade só por si. A separação destas duas funcionalidades permite obter módulos mais simples e estanques. Uma ferramenta pode apenas ler informação (monitorização) enquanto que outra pode apenas configurar agentes (no caso de configuração múltipla, por exemplo).

Uma vez que todos os módulos se encontram integrados no editor de macros torna-se possível utilizar este último como porta para contactar os agentes. O utilizador pode definir os valores por intermédio dos módulos MAF, Schedule, Event ou outros, passar estes valores para a aplicação principal e terminar o seu funcionamento. A aplicação principal pode então armazenar a informação ou enviá-la para o agente respectivo.

Optou-se por utilizar a linguagem Java para o desenvolvimento o que permite recorrer às ferramentas de introspecção para descobrir e instalar os serviços disponíveis em cada componente.

7.2 Editor de Macros

O Editor de Macros é uma ferramenta genérica que permite construir sequências de comandos SNMP. Esta utiliza o mecanismo de persistência definido na secção 4.4.2 mas o produtor de informação será o utilizador e não o agente. Para o efeito, apresenta um ambiente gráfico de trabalho de forma a permitir ao utilizador definir comandos SNMP. No contexto desta tese consideram-se macros SNMP (ou simplesmente macros) como grupos de tarefas definidas pelo utilizador que por sua vez agrupam sequências de operações SNMP elementares. Considera-se que cada sequência constitui uma tarefa, pelo que um macro pode conter várias tarefas.

Basicamente, esta ferramenta consiste num editor gráfico de código XML. O utilizador poderá definir macros e armazená-los em ficheiros em formato XML sem ser necessário estar ciente da linguagem ou do procedimento intrínseco. Este código será interpretado e executado posteriormente resultando em última instância na reprodução de comandos SNMP.

A sua natureza genérica permite consultar e modificar informação em qualquer módulo MIB. À semelhança de um *browser* de MIBs, permite manipular objectos de gestão, embora as semelhanças terminem aí. O Editor de Macros é principalmente orientado para executar e consultar sequências de comandos básicos SNMP enquanto que um *browser* de MIBs articula-se melhor com cada objecto de gestão. Por outras palavras, o Editor de Macros é baseado em comandos SNMP enquanto que o *browser* de MIBs é baseado em objectos de gestão.

7.2.1 Modelo desenvolvido

A definição de macros SNMP é uma tarefa laboriosa devido ao possivelmente elevado número de operações envolvidas. A definição de um evento na Event MIB, por exemplo, necessita de criar ou modificar dezenas de objectos de gestão, pelo que recorrer a um *browser* de MIBs não será a opção mais confortável. No caso do Editor de Macros, toda a sequência de operações terá também de ser descrita, pelo que não se ganha em eficiência pela sua utilização. No entanto, após criar a primeira sequência, esta pode ser guardada e utilizada como base para a criação de novos eventos, facilitando o trabalho seguinte.

A janela de trabalho do Editor de Macros encontra-se dividida em três áreas (Figura 7.4).

Do lado esquerdo, encontra-se a área de comandos, ou seja, as unidades elementares que surgem na sequência. Além dos sete comandos básicos SNMP (`Get`, `GetNext`, `GetBulk`, `Set`, `Trap`, `Inform` e `Reponse`) existem mais quatro comandos:

- `Task` – Inicia uma nova tarefa. Um macro pode conter várias tarefas que, por sua vez, podem conter vários comandos SNMP elementares.
- `Mib` – Representa a adição de mais um módulo MIB ao macro SNMP. Este comando resulta na leitura de um ficheiro SMI e na interpretação dos objectos aí definidos.
- `Property` – Uma propriedade corresponde à associação de uma variável a uma etiqueta, à semelhança das variáveis em ficheiros de código fonte. A definição de propriedades permite utilizar a mesma sequência de comandos sobre agentes distintos ou com valores distintos.

- `RunTask` – À semelhança da primitiva `#include` na linguagem de programação C, este comando permite invocar tarefas definidas em outros ficheiros (ou macros).

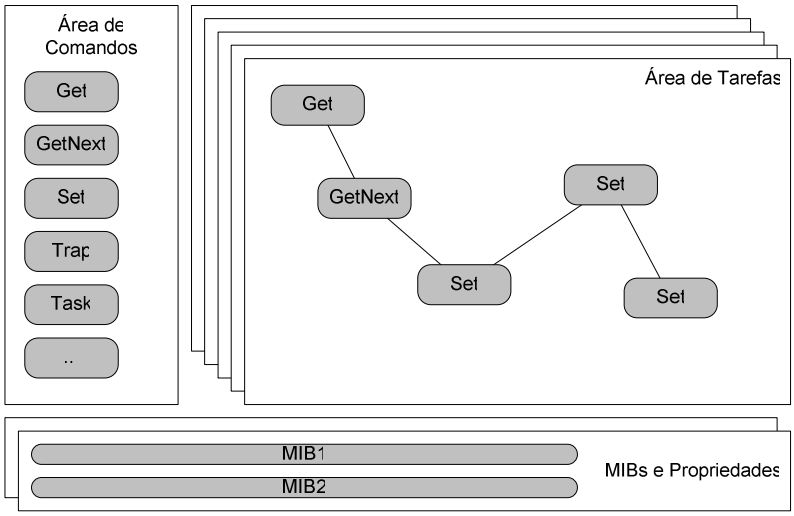


Figura 7.4 – Estrutura do Editor de Macros SNMP.

Do lado direito do editor encontra-se a janela de tarefas. As tarefas correspondem a páginas, agrupadas num "livro" e identificadas por etiquetas ou "separadores". Ao arrastar o comando `Task` do lado esquerdo da janela de aplicação para o lado direito surge uma nova tarefa em branco (nova etiqueta) que pode ser preenchida com uma nova sequência de comandos.

Os comandos são numerados sequencialmente de acordo com a ordem de criação. Por exemplo, se a janela se encontra em branco e um comando é arrastado para a tarefa correspondente, será atribuído o primeiro número da sequência – '0'. O segundo comando terá o segundo número ('1') e apresenta-se ligado ao primeiro por uma linha recta. O processo continua até o utilizador considerar a tarefa terminada.

Os comandos podem ser reordenados arrastando o símbolo correspondente para a linha recta que une dois comandos. Ao cruzar a linha, esta será interrompida e os comandos reordenados.

Na parte inferior da janela de aplicação encontra-se a terceira secção. Aqui podem ser indicados os módulos MIB e as propriedades associadas ao macro SNMP. Esta informação é válida para todas as tarefas definidas no editor.

Os módulos MIB incluídos no macro são seleccionados a partir de ficheiros em formato SMI. Ao arrastar o comando `Mib` da área esquerda da janela de aplicação para a parte inferior surge uma nova entrada na tabela de MIBs. Cada entrada na tabela descreve o ficheiro respectivo e o módulo seleccionado. Este passo é geralmente efectuado no início uma vez que os objectos de gestão existentes nos módulos MIB serão utilizados como parâmetros para os comandos SNMP, embora seja possível alterar esta informação em qualquer momento. Os módulos MIB definem um conjunto de dependências (`IMPORTS`) que leva a que estes sejam carregados simultaneamente. No entanto, apenas o módulo escolhido pelo utilizador irá figurar no macro. Por este motivo, a secção de MIBs apresenta a possibilidade de seleccionar o ficheiro e o módulo respectivo de entre todos os que se encontram relacionados.

Na Figura 7.5 é apresentada a forma final do editor, ilustrando a operação de selecção de MIBs por intermédio da janela de diálogo adequada.

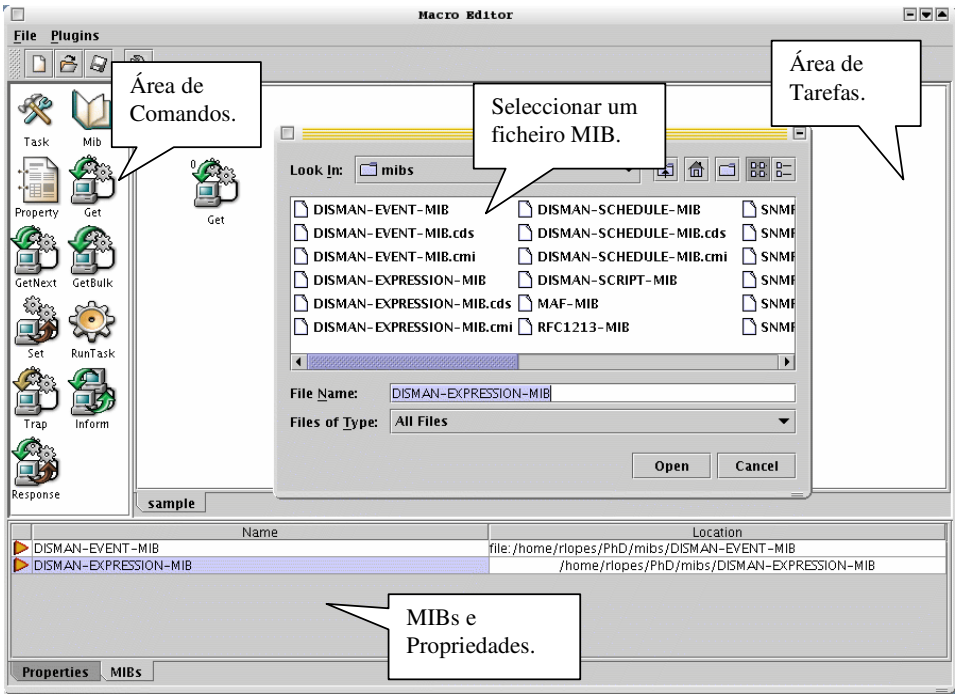


Figura 7.5 – Definição de um módulo MIB no Editor de Macros.

Após adicionar um comando é possível aceder à janela de detalhes que lhe está associada para alterar ou definir os parâmetros associados. Ao seleccionar duplamente um comando, surge uma janela de configuração que varia de acordo com o objecto. Por exemplo, um comando `Set` permite designar o endereço, porto, objectos e valores enquanto que o comando `Get` permite designar endereço, porto e os objectos a consultar (Figura 7.6).

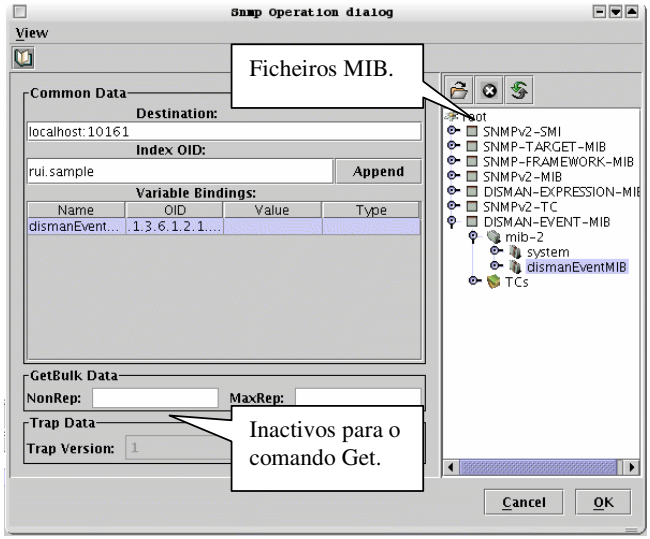


Figura 7.6 – Definição da informação associada ao comando `Get`.

Do lado direito desta janela de diálogo encontra-se uma árvore de objectos MIB que podem ser arrastados para a tabela à sua esquerda para indicar os objectos visados pela operação. São também visíveis outros parâmetros associados a diferentes operações, como `GetBulk` e `Trap`, disponíveis apenas para os comandos correspondentes. Após validar a informação através do botão 'OK' o utilizador associa a informação ao comando, podendo esta ser posteriormente modificada. As tarefas podem então ser armazenadas em disco em formato XML.

Como exemplo, o seguinte macro declara dois módulos MIB e uma tarefa ("sample") constituída por dois comandos – `get` e `set`:

```
<snmp version="3" user="" authProtocol="NoAuth" authPassword=""
    privPassword="" community="public" writeCommunity="private">
  <mib name="DISMAN-EVENT-MIB"
    location="file:/home/rlopes/PhD/mibs/DISMAN-EVENT-MIB"/>
  <mib name="DISMAN-EXPRESSION-MIB"
    location="file:/home/rlopes/PhD/mibs/DISMAN-EXPRESSION-MIB"/>
  <task name="sample">
    <get destination="localhost:10161">
      <varBind name="dismanEventMIB"
        oid=".1.3.6.1.2.1.88.10.114.117.105.46.115.97.109.112.108.101"/>
    </get>
    <set destination="localhost:10161">
      <varBind name="mteTriggerValueID"
 oid=".1.3.6.1.2.1.88.1.2.2.1.6.10.114.117.105.46.115.97.109.112.108.101"
       value=".1.3.6" type="OBJECT IDENTIFIER"/>
    </set>
  </task>
</snmp>
```

Observando simplesmente a dimensão de cada um dos OIDs representados e tendo em conta a facilidade com que estes são especificados através da árvore de MIBs é fácil perceber a importância de uma ferramenta deste tipo para simplificar operações de gestão.

7.2.2 Componentes

O Editor prevê ainda a adição de componentes com a finalidade de complementar ou especializar a sua funcionalidade, tal como sugerido anteriormente. Na barra de menus surge a opção 'Plugins' onde é visualizada uma lista dos componentes disponíveis (Figura 7.7).

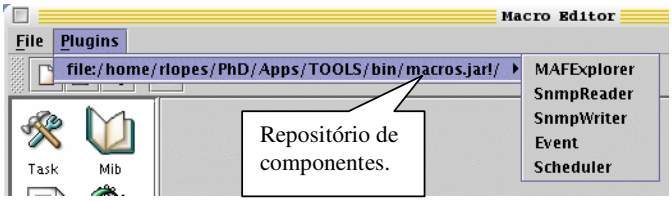


Figura 7.7 – Listagem de componentes associados ao Editor de Macros.

A indicação que surge imediatamente após seleccionar o menu 'Plugins' designa o repositório de componentes, neste caso um ficheiro em formato de arquivo de código Java, assinalado pelo URL `file:/home/rlopes/PhD/TOOLS/bin/macros.jar`. Este repositório contém cinco componentes, nomeadamente, MAFExplorer, SnmpReader, SnmpWriter, Event e Scheduler.

A integração de módulos recorre ao mecanismo de introspecção da linguagem Java associado a um leitor de ficheiros JAR (*Java Archive*). A aplicação, neste caso o Editor de Macros, lê o(s) ficheiro(s) JAR indicado(s) pelo utilizador e apresenta uma listagem dos módulos encontrados. A listagem é recuperada a partir da secção MANIFEST, parte integrante dos arquivos JAR, que contém a descrição das características principais do repositório, nomeadamente, a assinatura digital e o nome das classes que constituem componentes, entre outras.

Quando o utilizador selecciona um determinado módulo, a classe obtida pelo leitor de arquivos JAR é instanciada recorrendo a um *Class Loader* específico. Os métodos são invocados por intermédio do mecanismo de introspecção que permite, em tempo de execução, obter uma descrição completa dos métodos, argumentos, parâmetros e variáveis de qualquer classe Java. Esta abordagem permite manipular módulos inseridos posteriormente e sem que haja conhecimento anterior da sua funcionalidade.

Um dos componentes mais importantes da aplicação é o módulo de leitura de objectos de gestão de um agente SNMP e que pode funcionar autonomamente ou pode igualmente fornecer este serviço básico a todos os outros componentes. Resumidamente, a ferramenta constrói um macro SNMP de acordo com a operação de leitura realizada (Figura 7.8).

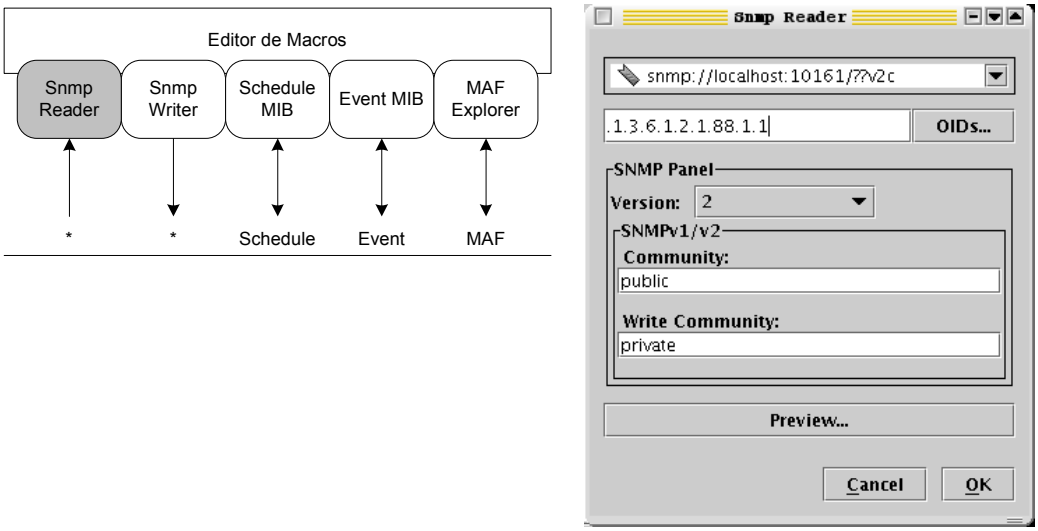


Figura 7.8 – Janela de diálogo da ferramenta de leitura de objectos remotos (`get` v1, v2c e v3).

A janela apresenta, em primeiro lugar, a barra de endereçamento de agentes. À semelhança dos navegadores de Internet a barra aceita endereços do tipo URL ainda que com algumas modificações. Mais precisamente, o endereçamento de agentes é efectuado com base em URIs (ver secção 4.4.3), designando o protocolo, o utilizador, o endereço, porto e OID.

De seguida é apresentado um campo para designar o OID raíz a partir do qual será efectuada a operação de consulta. Este poderá ser indicado directamente pelo utilizador ou seleccionado numa árvore de objectos. Todos os OIDs hierarquicamente inferiores serão obtidos através de operações do tipo `get-next`.

Os parâmetros de configuração do protocolo de gestão são ajustados numa secção que modifica o seu aspecto gráfico de acordo com a versão do protocolo escolhida. Para SNMPv3 surge um painel que permite especificar o nome de utilizador, palavra chave e os mecanismos de segurança associados. Para a versão SNMPv1 e SNMPv2c é apenas possível designar a comunidade de leitura e de escrita.

Antes de validar, o utilizador pode obter uma ideia do resultado ao seleccionar o botão 'Preview...' (Figura 7.9).

Nesta situação surge uma nova janela com a apresentação da informação obtida do agente em em formato XML. Ao validar a operação em 'OK', o macro SNMP será visualizado no Editor.

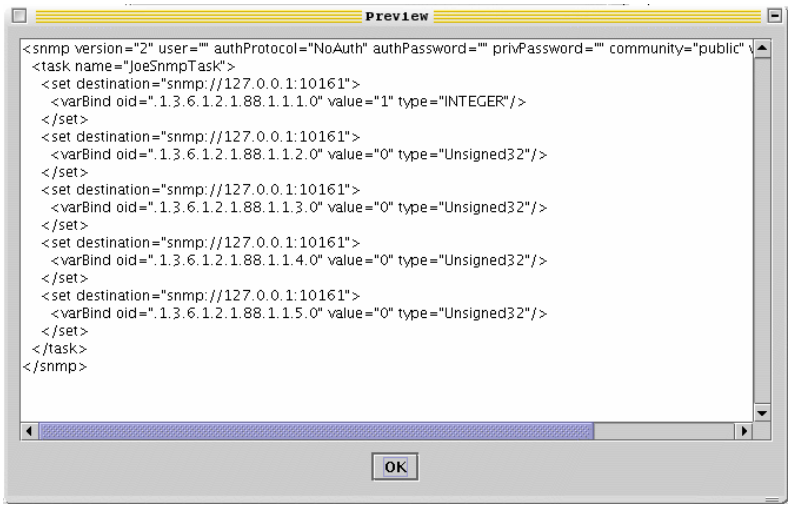


Figura 7.9 – Visualização do resultado da consulta em XML.

O módulo SnmpWriter efectua as operações complementares às do SnmpReader, ou seja, recebe do Editor de Macros a informação do macro em formato XML e executa sequencialmente as instruções aí descritas (Figura 7.10).

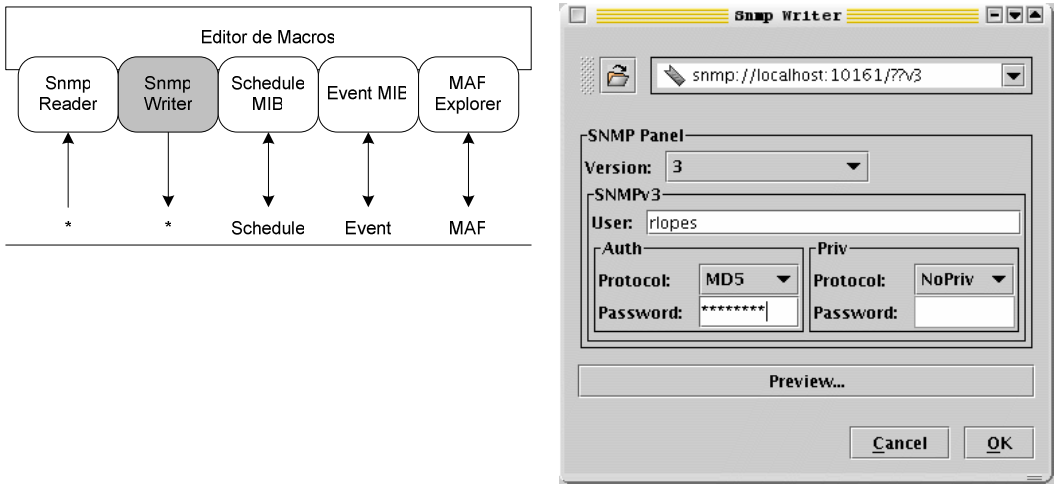


Figura 7.10 – Janela de diálogo da ferramenta de escrita de objectos remotos.

Este módulo permite efectuar operações de configuração sobre agentes SNMP genéricos, independentemente das MIBs implementadas.

7.3 Calendarização de tarefas

O módulo de calendarização de tarefas, denominado Scheduler no menu 'Plugins' do Editor de Macros, é uma ferramenta gráfica de configuração de agentes DISMAN que recorrem à Schedule MIB (Figura 7.11).

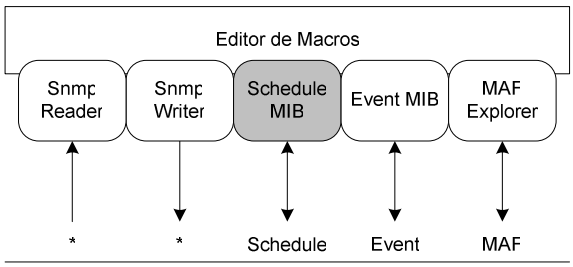


Figura 7.11 – Módulo de calendarização de operações sobre a Schedule MIB.

O objectivo deste componente é simplificar a tarefa de definição de operações de calendarização. A interface gráfica dispõe das áreas e dos controlos necessários para definir operações periódicas, de calendário ou únicas, tal como definido pelo grupo DISMAN. Este módulo prevê a possibilidade de ler valores de um agente bem como a de configurar, facto apresentado pela seta bidireccional na figura anterior. Além disso, permite averiguar a data e hora do agente e obter informação actualizada do funcionamento do agente (Figura 7.12).

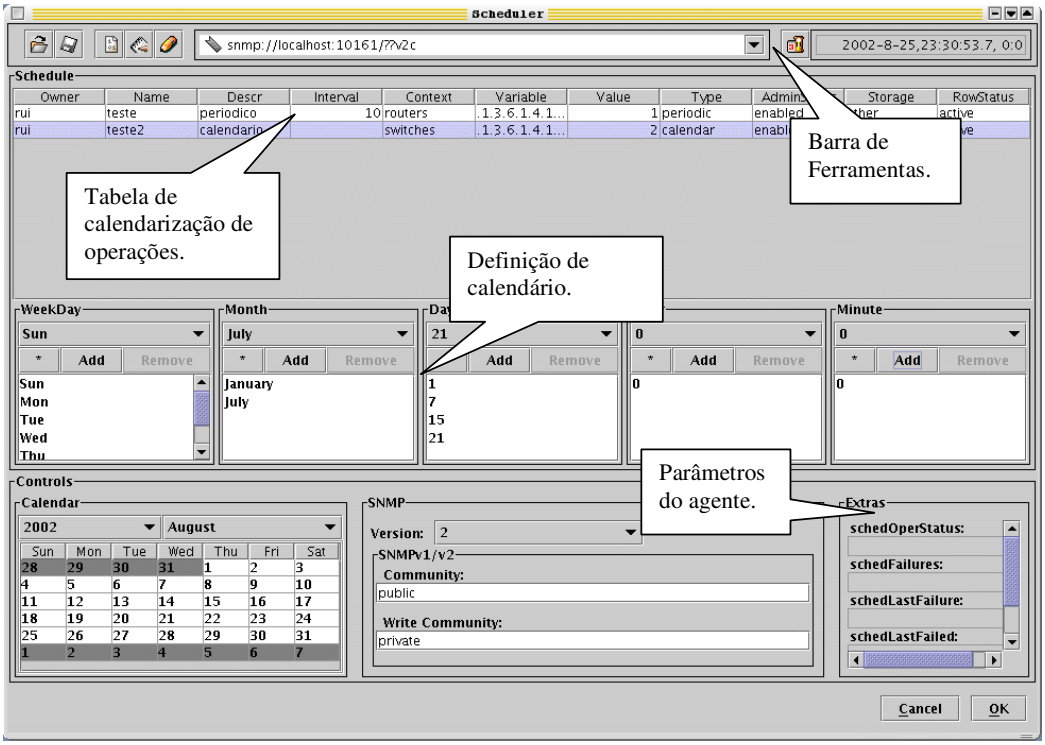


Figura 7.12 – Definição de um operações de calendário.

A janela apresenta três secções ordenadas verticalmente (barra de ferramentas, calendarização e controlos). A barra de ferramentas contém vários grupos de opções. O primeiro grupo na barra de ferramentas, com duas opções, permite ler e escrever informação de calendário em formato XML num dispositivo de armazenamento. O grupo de botões seguinte encontra-se relacionado com operações SNMP. A primeira opção efectua a leitura de informação de um agente Schedule MIB enquanto que as duas opções seguintes efectuam o oposto, ou seja, enviam a informação definida no componente para um agente SNMP. O endereço de agente é especificado na barra de endereçamento no formato URL para SNMP [Lopes02b]. O último

grupo apresenta a informação temporal do agente SNMP correspondente ao objecto *schedLocalTime*.

A segunda secção, directamente relacionada com a tabela *schedTable* no módulo MIB, permite calendarizar operações. Cada entrada na tabela corresponde a uma operação que poderá ser periódica, de calendário ou única. Os controlos gráficos de WeekDay, Month, Day, Hour e Minute são necessários para definir os instantes de calendário.

A última secção apresenta um calendário com função informativa, um painel de configuração de parâmetros SNMP e um painel de informação obtida do agente SNMP relativa ao funcionamento de cada entrada de calendário.

A definição de uma operação é efectuada pelo utilizador através do preenchimento da tabela correspondente. Em primeiro lugar deverá ser fornecido um nome que identifica o proprietário da operação, seguido do nome que identifica a própria operação. Entre outros será necessário indicar o período, a variável a modificar e o valor associado. Após ser definida, a operação é enviada para o agente onde inicia funcionamento.

Para as operações calendarizadas a coluna correspondente ao período é ignorada prevalecendo as entradas correspondentes ao calendário dispostas por baixo da tabela. Para que esta informação seja válida é necessário indicar que o tipo de operação é do tipo calendário ou único (coluna 'Type', seleccionar 'calendar' ou 'singleShot').

Este tipo de operações são executadas nos instantes de calendário indicados de acordo com operações booleanas de tipo 'E' dentro da mesma categoria (mesmo dia, mês ou hora) e de tipo 'OU' em categorias diferentes. No exemplo da figura, a operação 'rui.teste2' será executada nos dias 1, 7, 15 e 21 dos meses de Janeiro e Julho às 0 horas, independentemente do dia de semana. Quando estes instantes ocorrerem, o objecto indicado na coluna 'Variable' será modificado para o valor '2'.

Após seleccionar o botão de 'OK', a informação é transferida para o Editor de Macros, criando uma tarefa (*task*) para cada entrada na tabela de calendarização, neste caso 'rui.teste' e 'rui.teste2' (Figura 7.13).

As sequências de operações resultantes são inteiramente constituídas por comandos do tipo `set` uma vez que a definição de operações de calendário depende apenas da criação de objectos numa tabela SNMP, não sendo necessário adicionar qualquer operação de leitura.

A utilização desta ferramenta encapsula grande parte da complexidade de funcionamento da Schedule MIB, pelo que não será necessário para o utilizador aperceber-se de todos os detalhes descritos na documentação. Basta um conhecimento geral do mecanismo e algum treino na utilização da ferramenta gráfica para que seja possível definir operações de calendário em SNMP. A aplicação pode ser fechada ou a ligação interrompida sem que o funcionamento das operações calendarizadas seja afectado, uma vez que estas estão a ser executadas no agente e que se encontra instalado remotamente.

De notar que a dificuldade associada à definição de longas sequências de operações é desta forma bastante reduzida. A necessidade de conhecimento de XML por parte do utilizador é também eliminada, podendo o código ser gerado automaticamente pela aplicação caso haja necessidade. Apenas a título de curiosidade, o ficheiro XML gerado pela aplicação para a definição das duas operações contém 22 operações de escrita em duas tarefas (13+9), totalizando 110 linhas de código (ver Anexo D).

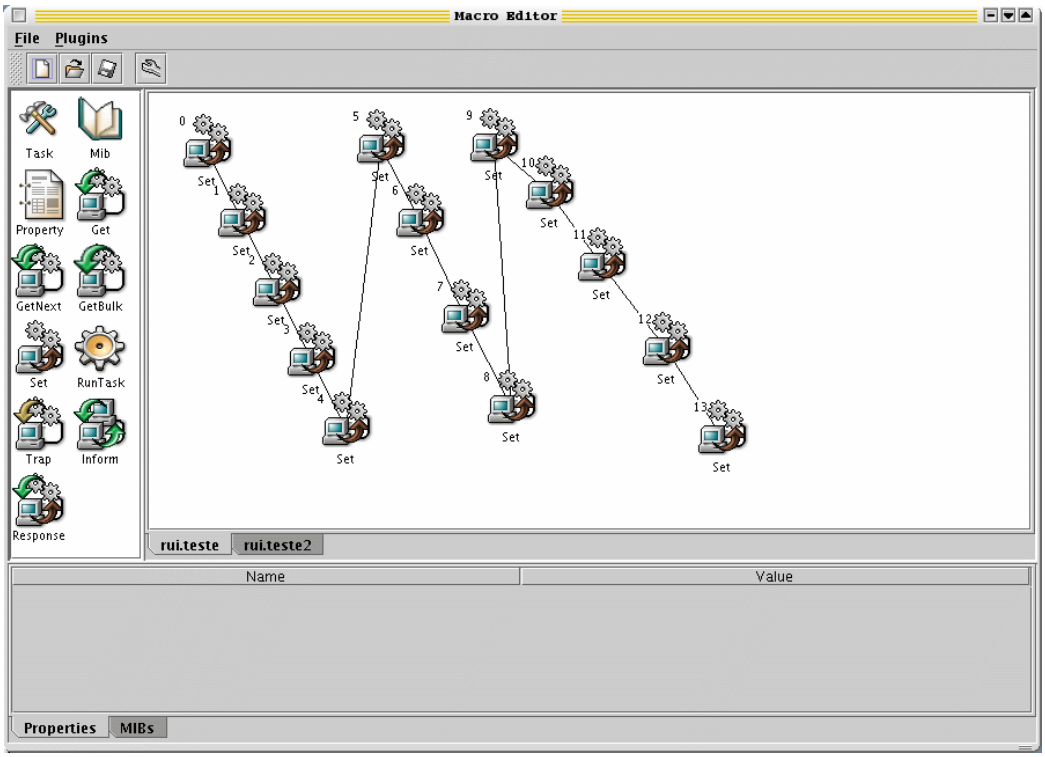


Figura 7.13 – Macro SNMP correspondente à definição de duas operações de calendário.

7.4 Definição de eventos

O componente Event foi desenvolvido para auxiliar na configuração de agentes DISMAN que implementam a Event MIB (Figura 7.14).

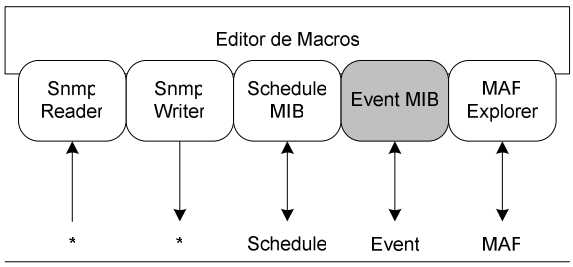


Figura 7.14 – Módulo de definição de eventos sobre a Event MIB.

Este módulo permite definir, por etapas, todos os parâmetros necessários para reconhecer condições excepcionais em agentes de gestão remotos e iniciar uma acção consequente. Cada evento inicia acções de acordo com condições predefinidas do tipo `existe`, `booleana` ou de `limiar` (ver secção 3.2.4). As condições são analisadas de acordo com valores consultados periodicamente em outros agentes. Caso a condição seja verdadeira, o evento pode provocar a geração de uma notificação ou de uma operação do tipo `set`.

Para providenciar ao utilizador uma visão global do agente é apresentada uma janela com o nome e número de entradas em todas as tabelas da Event MIB. Esta ferramenta é importante pois permite obter um acesso directo a toda a informação relevante (Figura 7.15).

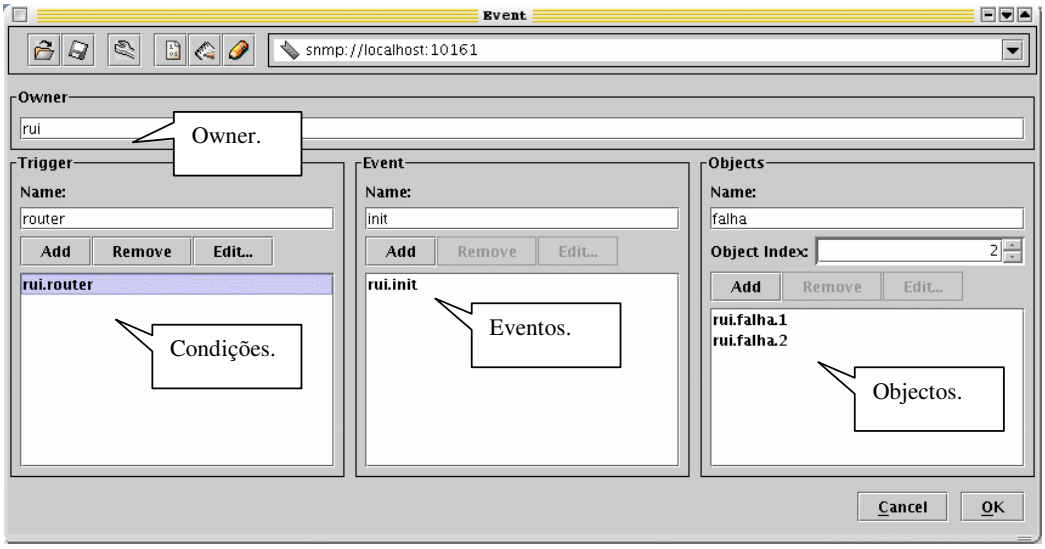


Figura 7.15 – Visão global da informação da Event MIB.

De notar que o elemento de indexação global é o *owner* enquanto que cada tabela pode apresentar nomes diferentes. Neste caso, o *owner* é 'rui' enquanto que a condição tem o nome 'router' e o evento 'init'. Um evento pode ser representado da seguinte forma:

```
/* Se determinada condição for verdadeita, iniciar um evento.
 */
IF <condição> THEN
  Call <evento>
END

/* O evento poderá lançar notificações ou iniciar uma operação, ambos definidos
 * pelo utilizador.
 */
BEGIN <evento>
  Call <notificação>
  Call <operação>
END

/* A notificação contém valores e OIDs, pelo que é necessário definir os parâmetros
 * a serem associados à notificação.
 */
BEGIN <notificação>
  Trap <parâmetros>
END

/* A operação é do tipo set e contém valores e OID.
 */
BEGIN <operação>
  Set <parâmetros>
END
```

A definição de eventos segue a sequência inversa, ou seja, em primeiro lugar é necessário definir os parâmetros e só depois as eventuais operações do tipo `set` e/ou notificações. Estas são depois agrupadas e associadas a um evento. Só de seguida é possível associar o evento às condições. Por outras palavras, as condições dependem de eventos que por sua vez dependem de acções que dependem de parâmetros.

Os parâmetros para as notificações correspondem OIDs que são definidos pelo utilizador e armazenados na tabela *mteObjectsTable*. Estes são indexados por *owner* ('rui'), nome ('falha') e índice ('1'). O índice serve para tornar possível a associação de vários OIDs a uma única notificação, estando esta última indexada apenas por *owner* e nome (Figura 7.16).

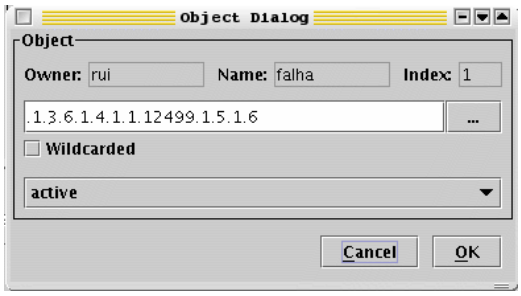


Figura 7.16 – Definição de notificações.

Após definir os objectos a incluir nas notificações, o utilizador deverá definir as notificações, as operações do tipo `set` e os eventos que as agrupam. Os parâmetros para as operações `set` são definidos na janela de diálogo de evento. Esta permite também agrupar a notificação e a operação, uma vez que o evento pode conter apenas uma notificação e/ou uma operação. O evento encontra-se indexado por *owner* ('rui') e por nome ('init'), pelo que corresponde a uma entrada nas tabelas *mteEventTable* e *mteEventNotificationTable* e/ou *mteEventSetTable* (Figura 7.17).

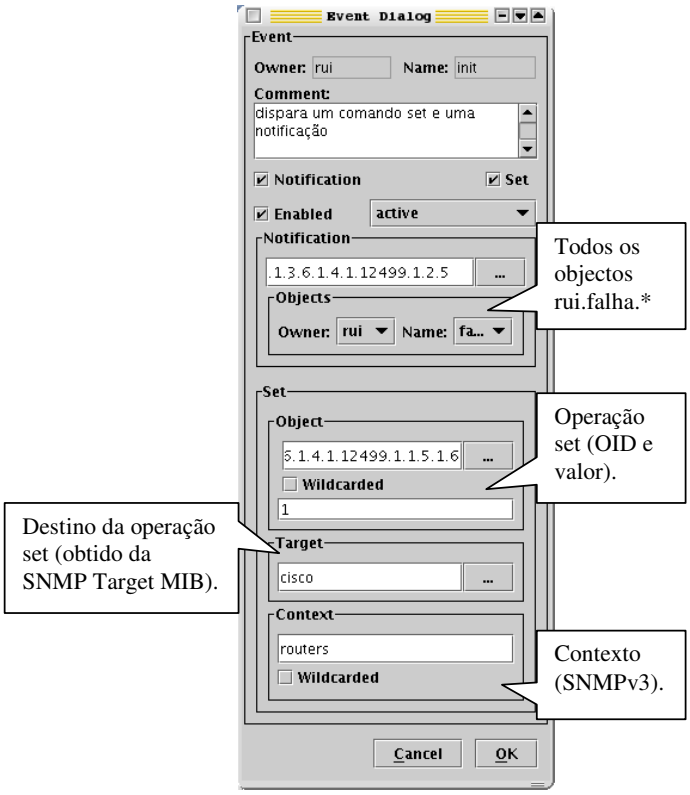


Figura 7.17 – Associação de notificação ou de operação `set` ao evento.

No exemplo acima, o evento 'rui.init' é constituído por uma notificação que transporta todos os objectos 'rui.falha.*' e por uma operação `set` efectuada sobre o agente identificado pela etiqueta 'cisco' na SNMP-TARGET-MIB com o contexto SNMPv3 'routers'.

Finalmente, é necessário definir a condição de disparo e associá-la ao evento. Para o efeito é necessário definir o valor a amostrar bem como a frequência de amostragem. Para cada

amostra é efectuada uma verificação que pode ser do tipo existe ('Existence'), booleana ('Boolean') ou de limiar ('Threshold'). Caso alguma se verifique verdadeira, é iniciado um evento (Figura 7.18).

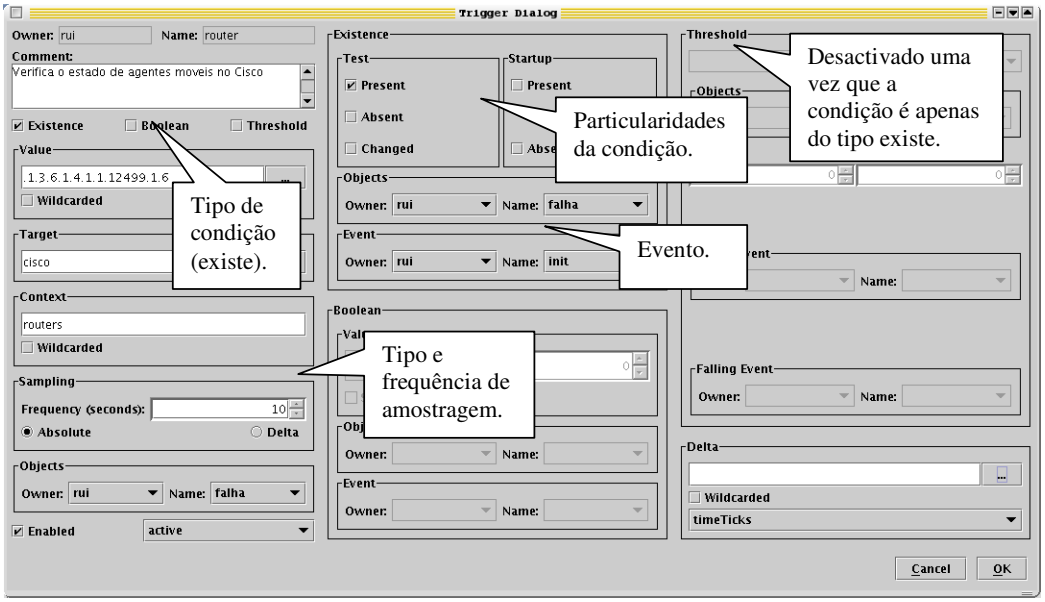


Figura 7.18 – Definição da condição de disparo de eventos.

Tal como nos módulos anteriores, ao seleccionar o botão 'OK' a informação é enviada para o Editor de Macros (Figura 7.19).

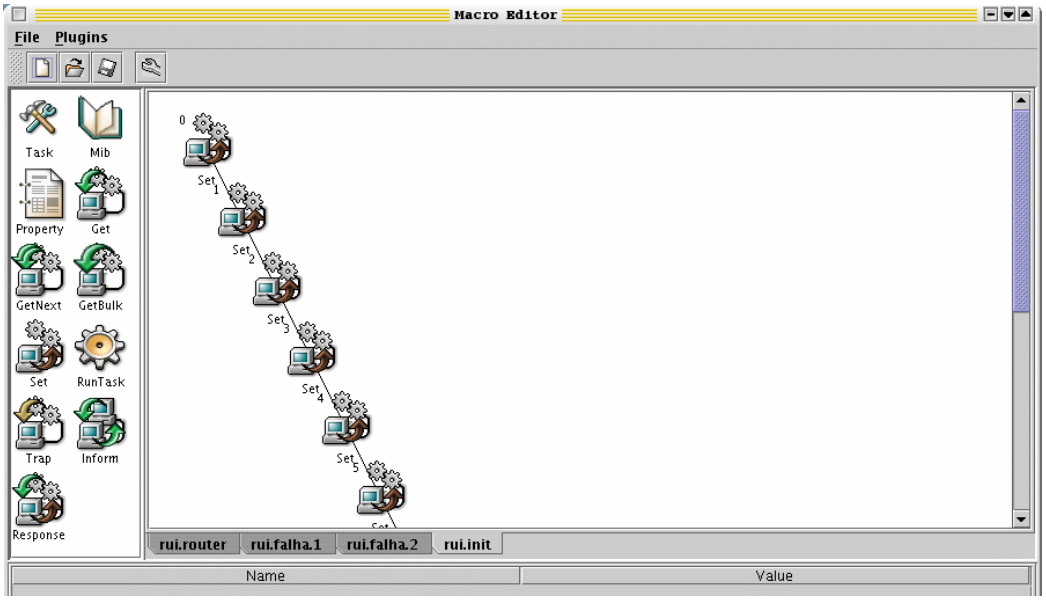


Figura 7.19 – Eventos definidos no Editor de Macros.

Neste caso, cada tarefa contém a informação associada a cada tabela. O macro resultante contém quatro tarefas num total de 28+3+3+12 operações do tipo `set` e 148 linhas de código XML.

7.5 Gestão de agentes móveis

O último módulo desenvolvido para o Editor de Macros é uma ferramenta de monitorização e controlo de agentes móveis por intermédio de SNMP e MAF (Figura 7.20).

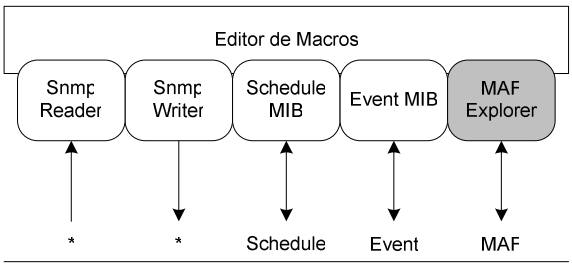


Figura 7.20 – Módulo de gestão de agentes móveis.

Esta ferramenta utiliza diferentes esquemas ou mecanismos de acesso a uma determinada agência prevendo inicialmente o acesso por SNMP e por MAF. Basicamente, utiliza o paradigma de explorador para apresentar a hierarquia de agências, lugares e agentes móveis. Esta informação é obtida do serviço de directoria por intermédio da interface de pesquisa `MAFFinder` (Figura 7.21).

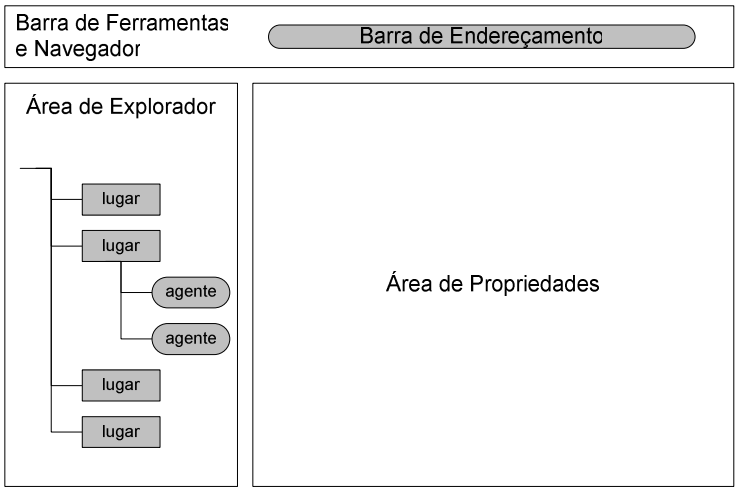


Figura 7.21 – Modelo do MAFExplorer.

De acordo com o endereço inserido na barra de endereçamento, o MAFExplorer recorre a CORBA (para endereços do tipo maf://locahost:1050/) ou a SNMP (para endereços do tipo snmp://localhost:10161/) para contactar a agência. No caso de utilizar MAF, o contacto será efectuado sobre o serviço de nomes CORBA de forma a obter referências para as interfaces `MAFFinder` e `MAFAgentSystem`.

O explorador representa graficamente as agências, lugares e agentes (Figura 7.22).

Quando o utilizador selecciona um dos objectos representativos de agências, lugares ou agentes, a área da direita mostra as propriedades relacionadas com esse objecto. No exemplo anterior, o agente SleepyAgent encontra-se seleccionado, pelo que as suas propriedades são apresentadas. O utilizador poderá mudar o estado do agente alterando a propriedade 'status' e consequentemente, suspender ou remover o agente.

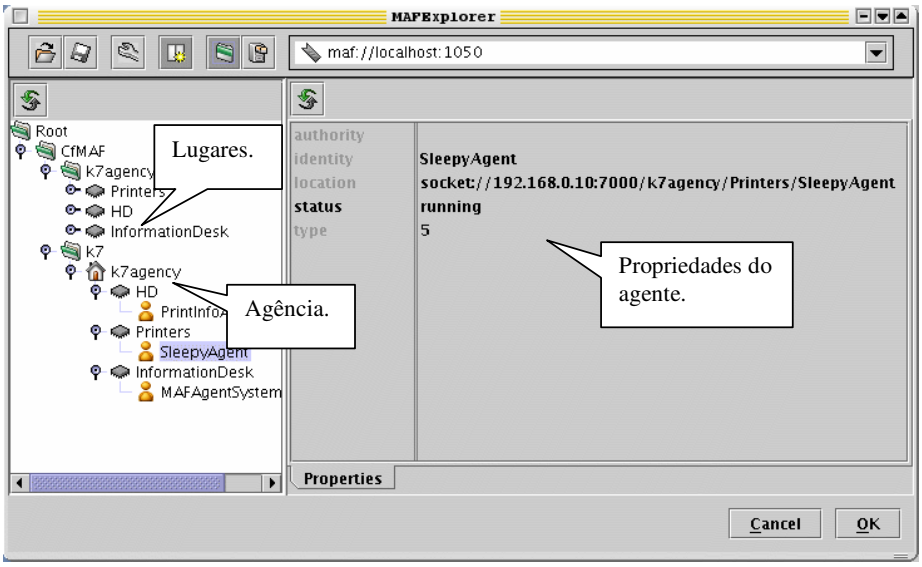


Figura 7.22 – MAFExplorer com as propriedades do agente SleepyAgent.

Na árvore do explorador são imediatamente visíveis os lugares 'Printers', 'HD' e 'InformationDesk' sob a agência 'k7agency' bem como os agentes que cada um contém.

7.6 Interface HTML e WML

Foi referido anteriormente que os agentes desenvolvidos em torno da Agent API (capítulo 4) possuem a capacidade de acesso directo por intermédio de um *browser* de Internet ou de um terminal WAP. Esta reside na possibilidade de os próprios agentes exportarem uma interface gráfica própria, dispensando a utilização de ferramentas específicas. Com características genéricas, esta interface mantém os inconvenientes de falta de adequabilidade que os *browsers* de MIB têm embora permitam consultar e modificar o estado do agente por intermédio de ferramentas vulgares como os navegadores de Internet ou os próprios telefones móveis (Figura 7.23).

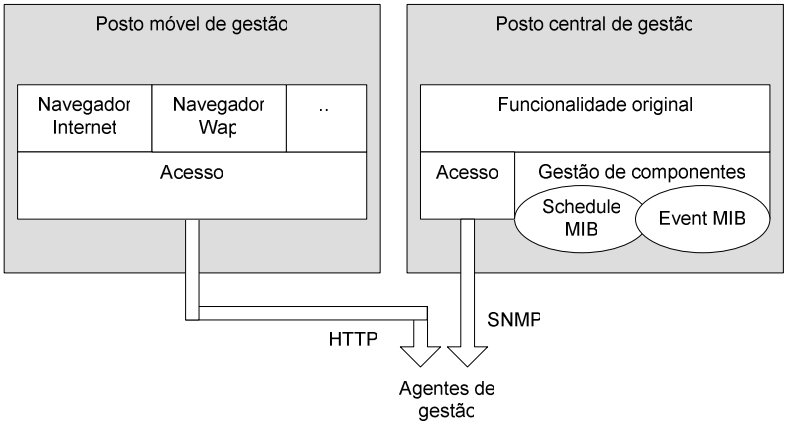
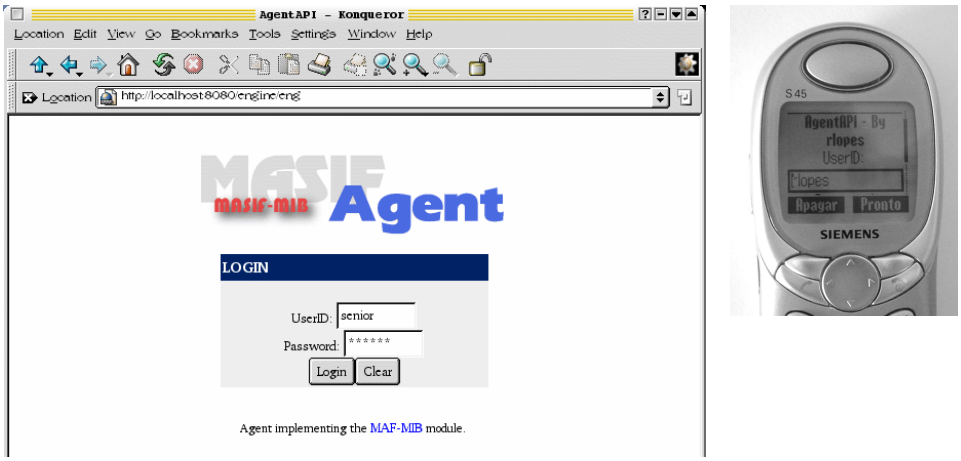


Figura 7.23 – Interface de acesso móvel aos agentes de gestão.

Apesar desta abordagem não se enquadrar no modelo modular apresentado anteriormente, trata-se de mais uma forma de o utilizador interagir com o sistema de gestão, pelo que não podia deixar de ser referida neste capítulo.

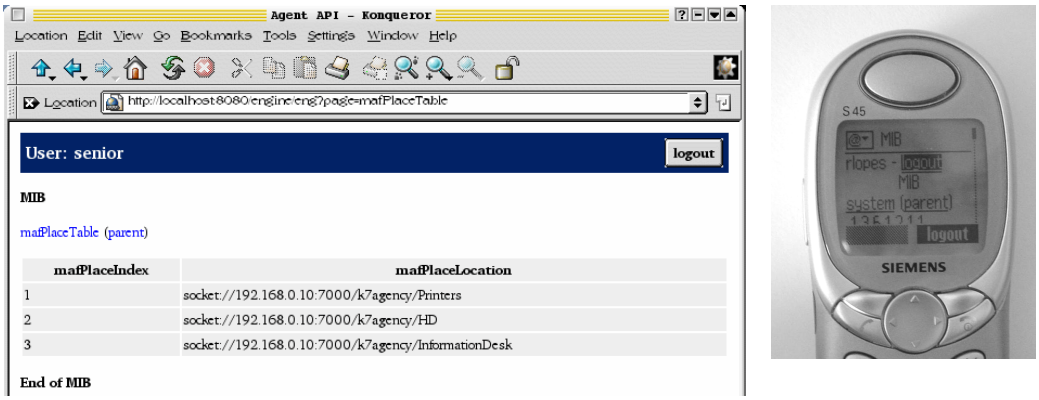
A interface gráfica é especificada pelo próprio agente por intermédio de linguagens do tipo HTML ou WML sendo esta interpretada pelo cliente de forma a apresentar ao utilizador um conjunto de páginas com o estado do agente. A primeira etapa do processo consiste em contactar o agente e autenticar o utilizador (Figura 7.24).



a) Acesso por HTTP. b) Acesso por WAP.

Figura 7.24 – Acesso directo ao agente.

Neste caso, são apresentadas páginas que solicitam ao utilizador o seu nome e a palavra chave associada. Caso este seja correctamente autenticado o agente passará a apresentar um conjunto de páginas com informação específica de gestão (Figura 7.25).



a) Acesso por HTTP. b) Acesso por WAP.

Figura 7.25 – Acesso à informação de gestão.

O utilizador pode então navegar pela informação de gestão a partir de qualquer lado e de qualquer plataforma sem necessidade de recorrer a aplicações específicas.

7.7 Conclusões

Os sistemas de gestão de redes não se resumem unicamente a processos de aquisição de dados ou de identificação de problemas. É importante providenciar ferramentas que permitam ao utilizador obter modelos ou representações gráficas sugestivas do estado de funcionamento do equipamento e da rede como um todo. Estas ferramentas, como parte integrante da rede, são também afectadas pela rápida evolução tecnológica e pela contínua introdução de soluções de comunicação. É importante que possa sobreviver não só à heterogeneidade como ao carácter dinâmico da rede.

Neste sentido a modularidade é muito importante. A actualização independente de módulos permite adicionar ou modificar funcionalidade enquanto que a remoção de módulos elimina problemas de consumo excessivo de recursos.

Por outro lado, o aumento de mobilidade de aplicações e utilizadores torna indesejável a dependência sobre uma única aplicação de gestão em execução num único local da rede. O utilizador deve poder, em qualquer altura e em qualquer local, consultar informação de gestão e modificar o estado de funcionamento do equipamento de rede. Neste sentido, são muitas vezes referidas as técnicas e aplicações utilizadas na Internet, como os *browsers* e os terminais WAP.

Neste capítulo descreveu-se uma aplicação gráfica modular constituída por um Editor de Macros no papel de aplicação principal e por um conjunto de componentes ou módulos que a complementam com funcionalidade específica. Cada um dos módulos apresentados têm a função de lidar com a informação exclusiva definida na Schedule MIB, Event MIB e MAF-MIB, tendo sido apresentados dois módulos adicionais para a leitura e escrita de informação de gestão em agentes genéricos.

O enriquecimento de aplicações com módulos estanques reúne várias vantagens quando comparadas com aplicações monolíticas. Por um lado, permitem actualizar apenas parte do seu funcionamento ou acrescentar mais funcionalidade, de acordo com a evolução tecnológica, sem necessitar de ser recompilada ou substituída na íntegra. Por outro lado, a utilização racional de módulos permite poupar recursos computacionais como a memória ou o espaço de armazenamento.

O capítulo termina com a apresentação da interface HTML ou WML exportada pelo agente. Apesar desta abordagem sobrecarregar o agente quando usada em conjunção com SNMP, pode significar o acesso imediato e ubíquo à informação de gestão.

Este balanceamento permite, por um lado, obter toda a funcionalidade de forma coerente em estações de trabalho específicas e o acesso mais genérico ao agente por parte de estações de trabalho generalistas.

A culture of overachievers, we make things happen – and happen fast. While rushing along, though, the days seem to get shorter and shorter. If only time would hold still, just a little bit, to let us savor life's simplest moments… Without further ado – and without feeling guilty – we learn to unwind, exhale, and, yes, stop and smell the roses.

Em "The Art of Doing Nothing".

8 Conclusões e Trabalho Futuro

As arquitecturas de gestão de redes que surgiram em finais da década de oitenta visaram a lacuna mais sentida na altura, ou seja, a interoperabilidade de soluções de gestão independentemente do modelo e da marca do equipamento de rede. Foi importante disponibilizar com urgência ferramentas que permitissem monitorar e controlar o equipamento e as aplicações de rede de forma uniforme mesmo com o prejuízo de funcionalidade. Foi neste contexto que surgiu o modelo de gestão SNMP.

Apesar da forte adopção que desde início caracterizou o modelo de gestão SNMP, nunca houve uma total satisfação devido às lacunas de segurança e à sua arquitectura centralizada. Os problemas de segurança foram resolvidos mais recentemente com a especificação do modelo SNMPv3 mas a arquitectura manteve-se dependente de um único posto de gestão.

Durante a década de noventa houve um grande avanço na engenharia de software, o que resultou no desenvolvimento de soluções baseadas em tecnologia de sistemas distribuídos que permite realizar a gestão de outras formas e seguindo outros modelos. Podem-se referir vários exemplos, como a linguagem Java, com a uniformização conseguida com base em máquinas virtuais, a arquitectura de distribuição de objectos CORBA, os agentes inteligentes, a mobilidade de código, entre muitos outros.

Como resultado, tem surgido uma verdadeira avalanche de soluções que procuram resolver os problemas de centralização e que prometem melhorar a escalabilidade, robustez e flexibilidade, em falta no modelo SNMP. No entanto, a introdução deste tipo de tecnologia pode prejudicar

a interoperabilidade. Dado que a grande maioria de sistemas instalados segue o modelo SNMP, a adopção de tecnologia será tanto mais fácil quanto maior for o grau de compatibilidade com este.

O grupo DISMAN vai de encontro a este requisito ao sugerir um conjunto elementar de operações e funções de distribuição de gestão no âmbito do SNMP. No entanto, herda também algumas lacunas do próprio modelo, nomeadamente, inexistência de mobilidade de processos e inadequabilidade para definir modelos cooperativos.

Além dos problemas técnicos ou derivados da adopção de uma determinada solução de gestão, existem também alguns problemas sociais, associados ao utilizador enquanto gestor da rede. A depedência de uma única aplicação e de uma única plataforma impede o utilizador de se deslocar com frequência para longe da estação de gestão sem perder o conhecimento do estado de funcionamento da rede.

A constante modificação da rede, devido à introdução de novo equipamento, à instalação de novas aplicações e de novos agentes de gestão torna difícil também a gestão coerente da rede, devido às aplicações terem de ser actualizadas para se compatibilizarem com as modificações introduzidas.

Neste trabalho apontaram-se algumas soluções e fizeram-se algumas propostas que procuraram ir de encontro às falhas apresentadas acima, nomeadamente o acesso multiprotocolo à informação de gestão, a distribuição de gestão em SNMP, a introdução de mobilidade de processos no modelo de gestão e a introdução de modularidade na interface com o utilizador.

O acesso multiprotocolo à informação de gestão tem por objectivos permitir utilizar diferentes mecanismos de comunicação e diferentes aplicações. Neste sentido, os agentes de gestão são fundamentais para o funcionamento de todo o sistema. Estes têm a responsabilidade de responder aos comandos recebidos com base nos parâmetros obtidos junto do sistema em análise. Dada a diversidade de plataformas, equipamento e aplicações, o desenvolvimento de agentes de gestão é uma tarefa complexa e muitas vezes negligenciada durante as fases de planeamento do sistema. Estes desempenham também um papel fundamental no acesso à informação e encontram-se associados ao protocolo de transporte utilizado.

A solução proposta para estes problemas assenta numa interface de programação, denominada Agent API, que disponibiliza uma arquitectura modular para o desenvolvimento de agentes de gestão. No cerne da Agent API encontra-se um mecanismo de manipulação de objectos de gestão com a responsabilidade de indexar, criar, ordenar e destruir dinamicamente os objectos de gestão. Em associação a este mecanismo encontra-se uma camada de abstracção sobre os mecanismos de comunicação particulares para, de forma transparente e sem necessitar recompilar o agente, utilizar diferentes protocolos e métodos de acesso como o SNMP, AgentX, HTTP, CORBA, RMI ou outros. Esta possibilidade permite efectuar a gestão por SNMP, associar os agentes a um *master agent* por AgentX e consultar os resultados via Internet e via WAP.

Foram também integrados na Agent API alguns mecanismos de extensão, como um mecanismo de persistência de informação de gestão com base em XML, um interpretador de SMI (SMIv1 e SMIv2) e um interpretador de URLs para recursos SNMP. Este último permite, numa única linha de texto e independentemente da versão SNMP, uniformizar a passagem de parâmetros entre as aplicações SNMP.

Os SNMP URLs definem um mecanismo de referenciação que permite univocamente identificar qualquer objecto de gestão e que pode ser usado em diversos e distintos cenários.

As aplicações do tipo MIB browser, por exemplo, podem utilizá-los para construir operações de gestão, indicando o objecto e todos os parâmetros necessários para contactar o agente que o contém. Podem também utilizá-los para substituir referências OID em determinadas MIBs de forma a providenciar acesso remoto à informação de gestão. A sua sintaxe uniforme permite também alargar o espaço de referência das aplicações de forma a aceitarem URLs para diferentes recursos com diferentes protocolos, por exemplo, FTP, HTTP, SNMP, LDAP, etc.

Relativamente à distribuição de gestão em SNMP, foram estudados dois ramos da distribuição, nomeadamente o de acção e o de reacção, sobre o modelo DISMAN. Para o efeito foram implementadas, com base na Agent API, a Schedule MIB, a Expression MIB e a Event MIB. Juntamente com uma implementação disponível em domínio público da Script MIB este conjunto de ferramentas permitiu extrair algumas ilações sobre o funcionamento do modelo DISMAN para a distribuição de gestão. Neste âmbito foram detectadas algumas dificuldades relacionadas com o acesso à informação de gestão. A Schedule MIB e a Expression MIB não definem mecanismos de acesso remoto à informação de gestão apesar de a Script MIB e a Event MIB o preverem, o que pode dificultar a sua adopção e o que limita a sua funcionalidade.

Foram propostas algumas alterações ao modelo de forma a permitir o acesso remoto à informação de gestão com base em SNMP URLs. Esta alteração dá à Schedule MIB a possibilidade de enviar comandos para agentes remotos e à Expression MIB a possibilidade de obter valores remotos como parâmetros das expressões.

O modelo DISMAN apesar de definir um modelo de execução remota de código com a Script MIB não prevê mecanismos de mobilidade de processos que, por sua vez, estão na base de alguns modelos de distribuição forte e mesmo cooperativos.

Ao transportar o código e o estado de execução, um agente móvel pode migrar transportando as ferramentas e o conhecimento adquirido, pelo que pode trazer uma mais valia para o sistema de gestão. Por outro lado, a introdução de agentes móveis como tecnologia de gestão introduz também a necessidade de gestão desta nova ferramenta. No âmbito deste trabalho foi considerada principalmente a problemática da gestão dos próprios agentes móveis com o objectivo de integrarem uma ferramenta de gestão de redes.

No contexto do modelo SNMP faz sentido que a gestão de agentes móveis seja também efectuada por SNMP. Neste sentido será necessário adicionar agentes SNMP à plataforma de agentes móveis de forma a responder a comandos SNMP com base no estado de funcionamento da plataforma.

Nesta tese foi proposto um mecanismo de integração de plataformas de agentes móveis para complementar o modelo DISMAN com a possibilidade de mobilidade de processos. Para o efeito foi definida uma MIB que permite criar, suspender, eliminar, monitorizar e pesquisar os agentes móveis, os lugares e as agências definidos no modelo de gestão. Esta MIB, denominada MAF-MIB, recorre às interfaces MAF, o que mantém o carácter genérico da abordagem e permite ser utilizado virtualmente para gerir todas as plataformas compatíveis com as *Mobile Agent Facilities*.

Por outras palavras, esta abordagem permite associar plataformas de agentes móveis ao conceito de DM (*Distributed Manager*) de forma a complementar a sua funcionalidade com agentes móveis. Neste sentido, um DM poderá ser constituído pela Schedule MIB para iniciar acções com base em eventos temporais, pela Script MIB para executar localmente funções de gestão, pela MAF-MIB para criar, iniciar e lançar agentes móveis, pela Expression MIB para

filtrar informação de gestão e pela Event MIB como mecanismo principal de análise de informação.

Do ponto de vista do utilizador, ferramentas simples, como *browser* de MIBs, não são adequadas para realizar a configuração de agentes DISMAN devido ao elevado número de operações realizadas e à diversidade de tipos de dados. Por outro lado, a maior constante em redes de comunicação de dados e consequentemente em sistemas de gestão de redes é precisamente a mudança. Por este motivo, é fundamental providenciar uma interface com o utilizador modular e extensível para a monitorização e controlo de agentes desconhecidos à partida.

Neste trabalho, foi apresentada uma aplicação gráfica modular constituída por um Editor de Macros no papel de aplicação principal e por um conjunto de componentes ou módulos que a complementam com funcionalidade específica. O enriquecimentos de aplicações com módulos estanques reúne várias vantagens quando comparadas com aplicações monolíticas. Por um lado, permitem actualizar apenas parte do seu funcionamento ou acrescentar mais funcionalidade, de acordo com a evolução tecnológica, sem necessitar de ser recompilada ou substituída na íntegra. Por outro lado, a utilização racional de módulos permite poupar recursos computacionais como a memória ou o espaço de armazenamento. Caso o utilizador esteja afastado da aplicação de gestão, o agente multiprotocolo providencia uma interface gráfica acessível via Internet ou via telemóvel.

Este trabalho apresenta um conjunto de soluções e de propostas com o principal objectivo de definir mecanismos flexíveis, poderosos e compatíveis com o modelo SNMP para realizar gestão distribuída. As soluções incidiram sobre os três principais níveis de gestão distribuída, nomeadamente os agentes, os gestores intermédios e as estações de gestão, aqui vistas mais como a interface com o utlizador.

8.1 Divulgação

Um aspecto fundamental em qualquer trabalho de investigação e desenvolvimento é a divulgação de resultados. A Internet constitui um meio privilegiado de divulgação do ponto de vista económico, de disponibilidade e de acesso, permitindo a consulta ubíqua e praticamente instantânea. Por estes motivos desenvolveu-se um sítio de Internet inteiramente dedicado ao trabalho apresentado ao longo desta tese[1].

O desenvolvimento foi guiado por um conjunto de requisitos que ditaram não só o seu aspecto final mas também a sua estrutura (Figura 8.1):

- Multilíngue – a Internet é intrinsecamente povoada por várias línguas e culturas. Para facilitar o acesso à informação e aumentar o número de acessos, as páginas de divulgação são apresentadas em várias línguas. No momento podem ser consultadas duas traduções: português – enriquecendo assim o espaço nacional na Internet – e inglês.
- Simples – ligações imediatas, colocadas em sítios visíveis.
- Sóbrio – poucas e significativas imagens, de forma a optimizar a velocidade de transferência e a não causar "distracções" sobre o texto apresentado.

[1] Disponível em http://nms.estig.ipb.pt/, alojado num servidor Apache e recorrendo a Java Server Pages.

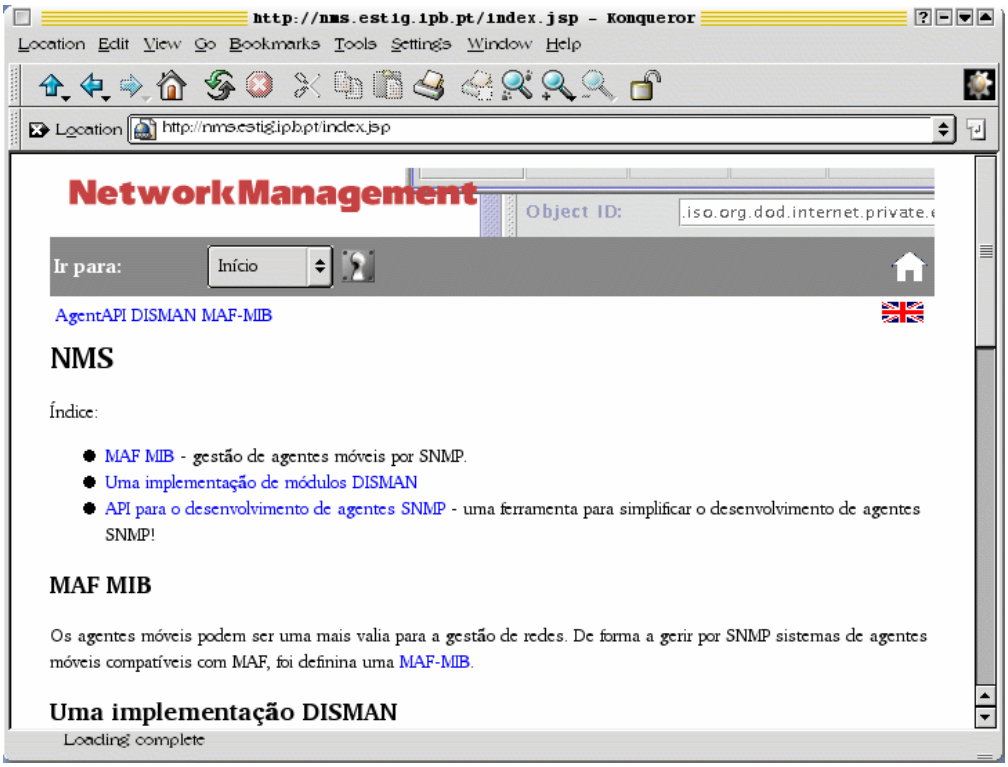


Figura 8.1 – Página principal de divulgação na Internet.

Desde que se começou a fazer estatística de acessos, nomeadamente a partir do período de 1 de Março de 2002, notou-se uma quantidade razoável de acessos que resultaram na transferência de várias páginas e ficheiros (Tabela 8.1). A Agent API, só por si, foi acedida 1310 vezes durante o ano de 2002, o que resulta em cerca de 4 *downloads* por dia.

Tabela 8.1 – Estatística global de acessos.

Descrição	Total	Últimos 7 dias
Pedidos atendidos	46 911	1 565
Número médio de pedidos atendidos por dia	147	223
Pedidos de páginas atendidos	6 980	102
Número médio de pedidos de páginas atendidos por dia	21	14
Ficheiros diferentes solicitados	765	108
Clientes diferentes atendidos	2 129	68
Tráfego total	2,088 gigabytes	51,857 megabytes
Tráfego médio transferido por dia	6,728 megabytes	7,408 megabytes

A disponibilização do sítio de Internet permite validar os modelos e as ferramentas aí apresentadas assim como recolher opiniões e comentários de outros utilizadores. Estes são valiosos como guias para futuros desenvolvimentos. Neste sentido, os contactos recebidos até ao momento revelaram alguns erros de programação (vulgo *bugs*) com consequente correcção o que elevou a qualidade global das ferramentas desenvolvidas. Foram também solicitadas

funcionalidades diferentes que permitem definir algumas linhas de intervenção para trabalho futuro.

8.2 Perspectivas de Trabalho Futuro

O trabalho apresentado ao longo desta tese encontra-se vocacionado para a engenharia e para o desenvolvimento de um modelo de gestão distribuída em SNMP. Neste sentido, foram estudados, implementados e comentados diversos aspectos relacionados com a gestão distribuída sem, no entanto, apresentar resultados da sua utilização prática.

Há diversas vertentes que providenciam tópicos de intervenção para trabalho futuro e que passam pela avaliação da sobrecarga das entidades e das aplicações apresentadas e pela integração do modelo definido em cenários de aplicação real.

Falta ainda especificar uma interface gráfica de gestão de agentes móveis que permita lidar com o aparecimento e desaparecimento dos agentes móveis em diversos nós da rede e que permita, de forma eficaz, lidar com o carácter extremamente dinâmico da actualização de código e a interacção entre agentes.

Uma outra direcção, vocacionada para o desenvolvimento, passa por dotar a Agent API de um compilador de MIBs universal, recorrendo a uma camada de informação comum e a vários módulos de interpretação (*parsers*). Cada módulo de interpretação funciona como um *driver* e encontra-se associados a cada formato, nomeadamente SMIv1, SMIv2 e SMIng.

Finalmente, seria interessante averiguar as diferenças de complexidade global de um sistema DISMAN com outros sistemas, por exemplo JMX, para operações idênticas, nomeadamente a execução de operações remotas, a calendarização de operações, a definição de expressões e a análise de eventos.

Anexo A.
Exemplo de construção de um agente usando a Agent API

```
/* Ficheiro SysUpTime.java. Implementa o objecto sysUpTime da MIB-II
 */
import pt.ipb.agentapi.*;

public class SysUpTime extends AgentObject {

  long initial;

  public SysUpTime(String oid) {
    super(oid);
    initial = System.currentTimeMillis();
  }

  public VarBind get(String oid) throws MessageException {
    long now = System.currentTimeMillis();
    long sysUpTime = (now-initial)/10;
    return new VarBind(
      new String(getOID()), new Counter(new Long(sysUpTime).toString())
    );
  }
}
```

```
/* Ficheiro SysContact.java. Implementa o objecto sysContact da MIB-II
 */
import pt.ipb.agentapi.*;

public class SysContact extends AgentObject
     implements WritableAgentObject {
  Var value = Var.createVar("", Var.OCTETSTRING);

  public SysContact(String oid) {
    super(oid);
  }

  public VarBind get(String oid) throws MessageException {
    return new VarBind(new String(getOID()), value);
  }

  public VarBind set(VarBind varBind) throws MessageException {
    Var val = varBind.getValue();
    if(val.getType()!=Var.OCTETSTRING)
      throw new MessageException(AbstractAgent.WRONG_TYPE);
    value = Var.createVar(val.toString(), Var.OCTETSTRING);
    return new VarBind(new String(getOID()), value);
  }
}
```

```
/* Ficheiro SnmpTargetAddrTable.java. Implementa a tabela conceptual
 * snmpTargetAddrTable da SNMP-TARGET-MIB.
 */
import pt.ipb.agentapi.*;
import pt.ipb.agentapi.event.*;
import java.util.Hashtable;
import java.util.StringTokenizer;

class TargetProvider extends ControlAdapter
    implements ConceptualTableProvider {
  Var values[] = null;

  public TargetProvider() {
    values = new Var[8];
  }

  public void activate(ControlEvent e) {
    ConceptualTableRow row = (ConceptualTableRow)e.getSource();
    String key = row.getKey().toString();
    String tagList = Str.toString((OctetString)getValueAt(4));
  }

  public int length() {
    return values.length;
  }

  public void setValueAt(int i, Var v) {
    values[i] = v;
  }

  public Var getValueAt(int i) {
    return values[i];
  }

  public int getRowStatusIndex() {
    return 7;
  }

}

public class SnmpTargetAddrTable extends AbstractConceptualTableModel {
  Hashtable index = null;

  public static int OTHER = 1;
  public static int VOLATILE = 2;
  public static int NON_VOLATILE = 3;
  public static int PERMANENT = 4;
  public static int READ_ONLY = 5;

  public OID[] oids;
  public byte[] type;
  public SnmpTargetAddrTable(String mask) {
    super(mask);
    oids = new OID[8];
    type = new byte[8];
    oids[0] = new OID(".1.3.6.1.6.3.12.1.2.1.2");  // No DefVal
    oids[1] = new OID(".1.3.6.1.6.3.12.1.2.1.3");  // No DefVal
    oids[2] = new OID(".1.3.6.1.6.3.12.1.2.1.4");  // 1500
    oids[3] = new OID(".1.3.6.1.6.3.12.1.2.1.5");  // 3
    oids[4] = new OID(".1.3.6.1.6.3.12.1.2.1.6");  // ""
    oids[5] = new OID(".1.3.6.1.6.3.12.1.2.1.7");  // No DefVal
    oids[6] = new OID(".1.3.6.1.6.3.12.1.2.1.8");  // NON_VOLATILE
    oids[7] = new OID(".1.3.6.1.6.3.12.1.2.1.9");  // No DefVal
    type[0] = Var.OID;
    type[1] = Var.OCTETSTRING;
    type[2] = Var.INTEGER;
    type[3] = Var.INTEGER;
    type[4] = Var.OCTETSTRING;
    type[5] = Var.OCTETSTRING;
    type[6] = Var.INTEGER;
    type[7] = Var.INTEGER;

    index = new Hashtable();
  }

  public OID getRowStatusOID() {
    return oids[7];
```

```
  }

  public TableProvider createNewProvider(OID key) {
    return new TargetProvider();
  }

  public void index(String tagList, String key) {
    StringTokenizer st = new StringTokenizer(tagList);
    while (st.hasMoreTokens()) {
      index.put(st.nextToken(),key);
    }
  }

  public String getName(String tagValue) {
    return (String)index.get(tagValue);
  }

  public OID[] getColumns() {
    return oids;
  }

  public byte[] getColumnTypes() {
    return type;
  }

  public byte validate(ConceptualTableProvider provider) {
    for(int i=0; i<oids.length; i++) {
      if(provider.getValueAt(i)==null) {
        return AbstractAgent.NOT_READY;
      }
    }
    return AbstractAgent.NO_ERROR;
  }

  public void setDefaultValues(TableProvider provider) {
    try {
      if(provider.getValueAt(2)==null) provider.setValueAt(2, new Int(1500));
      if(provider.getValueAt(3)==null) provider.setValueAt(3, new Int(3));
      if(provider.getValueAt(4)==null) provider.setValueAt(4, new Str("").toVar());
      if(provider.getValueAt(6)==null) provider.setValueAt(6, new Int(NON_VOLATILE));
    } catch(Exception e) {}
  }

  public boolean allowChange(int colNumber, TableRow r) {
    try {
      ConceptualTableRow row = (ConceptualTableRow)r;
      if(row.getState()==AbstractAgent.ACTIVE && colNumber==0) return false;
      if(row.getState()==AbstractAgent.ACTIVE && colNumber==1) return false;
      return true;
    } catch(Exception e) {
      return false;
    }
  }

  public boolean isOIDValid(OID oid) {
    try {
      StringOID o = StringOID.createStringOID("",0,oid.toString());
    } catch(Exception e) {
      return false;
    }
    return true;
  }

}
```

```
/* Ficheiro TestAgent.java. Classe principal do agente.
 * Construção dos objectos sysUpTime, sysContact e snmpTargetAddrTable e
 * associação do(s) protocolos de comunicação ao agente.
 */
package pt.ipb.agentapi.demo;

import pt.ipb.agentapi.*;

public class TestAgent extends AbstractAgent {

  public final static String SYS_UP_TIME_INSTANCE = ".1.3.6.1.2.1.1.3.0";
  public final static String SYS_CONTACT_INSTANCE = ".1.3.6.1.2.1.1.4.0";
```

```
  public final static String IFTABLE = ".1.3.6.1.2.1.2.2";

  public final static String SNMP_TARGET_MIB = ".1.3.6.1.6.3.12";
  public final static String SNMP_TARGET_OBJECTS = new String(SNMP_TARGET_MIB+".1");
  public final static String SNMP_TARGET_ADDR_TABLE =
    new String(SNMP_TARGET_OBJECTS+".2");

  public TestAgent() {
    super();
  }

  public void setObjects() {
    SysUpTime a = new SysUpTime(SYS_UP_TIME_INSTANCE);
    addObject(a);

    SysContact b = new SysContact(SYS_CONTACT_INSTANCE);
    addObject(b);

    SnmpTargetAddrTable model = new SnmpTargetAddrTable("0.0.0.0.0.0.0.0.0.0");
    Table c = new Table(SNMP_TARGET_ADDR_TABLE, model);
    addObject(c);

    Table ct = new Table(IFTABLE, new IFTable());
    addObject(ct);

  }

  public static void main(String arg[]) {
    try {
      Agent agent = new TestAgent();
      pt.ipb.agentapi.engine.EngineFactory.start(agent);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Anexo B.
DTD para a representação de mensagens SNMP em XML

```
<?xml encoding="UTF-8"?>
<!ELEMENT snmp (mib*,property*,task*)>
<!ATTLIST snmp version (1|2|3) '3'>
<!ATTLIST snmp user CDATA #IMPLIED>
<!ATTLIST snmp authProtocol (MD5|md5|SHA|sha) 'MD5'>
<!ATTLIST snmp authPassword CDATA #IMPLIED>
<!ATTLIST snmp privPassword CDATA #IMPLIED>
<!ATTLIST snmp community CDATA #IMPLIED>
<!ATTLIST snmp writeCommunity CDATA #IMPLIED>

<!ELEMENT mib EMPTY>
<!ATTLIST mib name CDATA #REQUIRED>
<!ATTLIST mib location CDATA #REQUIRED>

<!ELEMENT property EMPTY>
<!ATTLIST property name CDATA #REQUIRED>
<!ATTLIST property value CDATA #REQUIRED>

<!ELEMENT task (trap*,get*,getNext*,getBulk*,set*,inform*,response*,runTask*)>
<!ATTLIST task name CDATA #REQUIRED>

<!ELEMENT trap (varBind)*>
<!ATTLIST trap version CDATA #IMPLIED>
<!ATTLIST trap destination CDATA #REQUIRED>
<!ATTLIST trap name CDATA #IMPLIED>
<!ATTLIST trap oid CDATA #IMPLIED>

<!ELEMENT get (varBind)*>
<!ATTLIST get destination CDATA #REQUIRED>
<!ATTLIST get name CDATA #IMPLIED>
<!ATTLIST get oid CDATA #IMPLIED>

<!ELEMENT getNext (varBind)*>
<!ATTLIST getNext destination CDATA #REQUIRED>
<!ATTLIST getNext name CDATA #IMPLIED>
<!ATTLIST getNext oid CDATA #IMPLIED>

<!ELEMENT getBulk (varBind)*>
<!ATTLIST getBulk nonrep CDATA #REQUIRED>
<!ATTLIST getBulk maxrep CDATA #REQUIRED>
<!ATTLIST getBulk destination CDATA #REQUIRED>
<!ATTLIST getBulk name CDATA #IMPLIED>
<!ATTLIST getBulk oid CDATA #IMPLIED>

<!ELEMENT set (varBind)*>
<!ATTLIST set destination CDATA #REQUIRED>
<!ATTLIST set name CDATA #IMPLIED>
<!ATTLIST set oid CDATA #IMPLIED>
<!ATTLIST set value CDATA #IMPLIED>
```

```
<!ELEMENT inform (varBind)*>
<!ATTLIST inform destination CDATA #REQUIRED>
<!ATTLIST inform name CDATA #IMPLIED>
<!ATTLIST inform oid CDATA #IMPLIED>
<!ATTLIST inform value CDATA #IMPLIED>

<!ELEMENT response (varBind)*>
<!ATTLIST response destination CDATA #REQUIRED>
<!ATTLIST response name CDATA #IMPLIED>
<!ATTLIST response oid CDATA #IMPLIED>

<!ELEMENT varBind EMPTY>
<!ATTLIST varBind oid CDATA #REQUIRED>
<!ATTLIST varBind name CDATA #REQUIRED>
<!ATTLIST varBind value CDATA #IMPLIED>

<!ELEMENT runTask EMPTY>
<!ATTLIST runTask name CDATA #REQUIRED>
<!ATTLIST runTask document CDATA #REQUIRED>
```

Anexo C.
MAF-MIB

```
MAF-MIB DEFINITIONS ::= BEGIN
IMPORTS

    enterprises
        FROM SNMPv2-SMI
    SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB
    RowStatus
        FROM SNMPv2-TC;

ipb    MODULE-IDENTITY
    LAST-UPDATED    "0009151526Z"
    ORGANIZATION    "Instituto Politecnico de Braganca / Universidade de Aveiro"
    CONTACT-INFO    "Rui Pedro Lopes
ESTiG - IPB
Campus St Apolonia
5300 Braganca
Portugal
rlopes@ipb.pt

Jose Luis Oliveira
DET-UA
Campus Universitario
3810 Aveiro
Portugal
jlo@det.ua.pt"
    DESCRIPTION
        "MIB to monitor and control a Mobile Agent System using MAF"
    REVISION    "0201150000z"
    DESCRIPTION
        "Changed mafLookup section to cope with several filters."
  ::=  {  enterprises  12499  }

mafMIB    OBJECT IDENTIFIER  ::=  {  ipb  1  }

mafObjects    OBJECT IDENTIFIER  ::=  {  mafMIB  1  }
mafLookup    OBJECT IDENTIFIER  ::=  {  mafMIB  2  }

-------------------------------------------------------------
-- System Identification Group
-- information from MAFAgentSystem.get_agent_system_info()
-- and from MAFFinder.lookup_agent_system();
-------------------------------------------------------------
mafSysId    OBJECT IDENTIFIER  ::=  {  mafObjects  1  }

mafSysAuthority OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  mandatory
              DESCRIPTION
                      "identifies the person or organization for whom
                       the agent system acts. For example, an agent system
```

```
                        with authority Bob implements Bob's security policies
                        in protecting Bob's resources."
              DEFVAL  { ''H }
              ::= { mafSysId 1 }

mafSysIdentifier OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  mandatory
              DESCRIPTION
                      "Identifies this AgentSystem."
              DEFVAL  { ''H }
              ::= { mafSysId 2 }

mafSysLocation OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  mandatory
              DESCRIPTION
                      "Address of this AgentSystem."
              DEFVAL  { ''H }
              ::= { mafSysId 3 }

mafSysType OBJECT-TYPE
              SYNTAX  INTEGER {
                    nonAgentSystem(0),
                    aglets(1),
                    MOA(2),
                    agentTCL(3),
                    grasshopper(5)
                    }
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Type of this AgentSystem."
              ::= { mafSysId 4 }

mafSysDescr OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Description of this AgentSystem."
              DEFVAL  { ''H }
              ::= { mafSysId 5 }

mafSysMajorVersion OBJECT-TYPE
              SYNTAX  INTEGER
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Major version of this AgentSystem."
              ::= { mafSysId 6 }

mafSysMinorVersion OBJECT-TYPE
              SYNTAX  INTEGER
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Minor version of this AgentSystem."
              ::= { mafSysId 7 }

-------------------------------------------------------------
-- Error table
-------------------------------------------------------------

mafErrorTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF MafErrorEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A table of maf errors."
    ::= { mafObjects 2 }

mafErrorEntry OBJECT-TYPE
    SYNTAX      MafErrorEntry
    MAX-ACCESS  not-accessible
    STATUS      current
```

```
    DESCRIPTION
     "Information about errors in MAF operation."
    INDEX       { mafErrorIndex }
    ::= { mafErrorTable 1 }

MafErrorEntry ::= SEQUENCE {
  mafErrorIndex    INTEGER,
  mafErrorCode     INTEGER
}

mafErrorIndex OBJECT-TYPE
    SYNTAX      INTEGER
    ACCESS      read-only
    STATUS      current
    DESCRIPTION
     "A unique value for every error."
    ::= { mafErrorEntry 1 }

mafErrorCode OBJECT-TYPE
    SYNTAX      INTEGER {
                  namingServiceNotFound(0),
                  finderNotFound(1),
                  classUnknown(2),
                  argumentInvalid(3),
                  deserializationFailed(4),
                  mafExtendedException(5),
                  agentNotFound(6),
                  agentAlreadyRunning(7),
                  agentAlreadySuspended(8),
                  resumeFailed(9),
                  suspendFailed(10),
                  terminateFailed(11),
                  genericError(12)
                }
    ACCESS      read-only
    STATUS      current
    DESCRIPTION
     "A unique value for every error."
    ::= { mafErrorEntry 2 }

-------------------------------------------------------------
-- Language and Serialization Table
-- information from MAFAgentSystem.get_agent_system_info()
-------------------------------------------------------------

mafLanguageTable    OBJECT-TYPE
    SYNTAX    SEQUENCE  OF  MafLanguageEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "List of supported languages."
    ::=  { mafObjects  3 }

mafLanguageEntry    OBJECT-TYPE
    SYNTAX    MafLanguageEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "A Language Entry"
    INDEX    {  mafLanguageIndex }
    ::=  { mafLanguageTable 1 }

MafLanguageEntry  ::=  SEQUENCE {
    mafLanguageIndex  INTEGER,
    mafLanguageType  INTEGER,
    mafLanguageSerializationScheme  INTEGER
    }
mafLanguageIndex    OBJECT-TYPE
    SYNTAX    INTEGER
    MAX-ACCESS    not-accessible
    STATUS    current

    DESCRIPTION
        "The Index of the AgentSystem language."

    ::=  {  mafLanguageEntry  1  }

mafLanguageType OBJECT-TYPE
```

```
        SYNTAX  INTEGER {
              languageNotSpecified(0),
              java(1),
              tcl(2),
              scheme(3),
              perl(4)
              }
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Type of supported language."
              ::= { mafLanguageEntry 2 }

mafLanguageSerializationScheme OBJECT-TYPE
        SYNTAX  INTEGER {
              serializationNotSpecified(0),
              javaObjectSerialization(1)
              }
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Serialization scheme of supported language."
              ::= { mafLanguageEntry 3 }

-------------------------------------------------------------
-- Place table
-- information from MAFAgentSystem.list_all_places()
-------------------------------------------------------------

mafPlaceTable    OBJECT-TYPE
    SYNTAX    SEQUENCE  OF  MafPlaceEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "List of current agent system places."
    ::=  { mafObjects  4 }

mafPlaceEntry    OBJECT-TYPE
    SYNTAX    MafPlaceEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "A Place Entry"
    INDEX    {  mafPlaceIndex }
    ::=  { mafPlaceTable 1 }

MafPlaceEntry  ::=  SEQUENCE {
    mafPlaceIndex  INTEGER,
        mafPlaceLocation  DisplayString (SIZE (0..255))
    }

mafPlaceIndex    OBJECT-TYPE
    SYNTAX    INTEGER
    MAX-ACCESS    read-only
    STATUS    current

    DESCRIPTION
        "The Index of the AgentSystem place."

    ::=  {  mafPlaceEntry  1  }

mafPlaceLocation OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Location of the place."
              ::= { mafPlaceEntry 2 }

-------------------------------------------------------------
-- Agent table
-- information from MAFAgentSystem.list_all_agents()
-------------------------------------------------------------

mafAgentTable    OBJECT-TYPE
    SYNTAX    SEQUENCE  OF  MafAgentEntry
    MAX-ACCESS    not-accessible
    STATUS    current
```

```
     DESCRIPTION
        "List of current agents ."
    ::=  { mafObjects  5 }

mafAgentEntry    OBJECT-TYPE
    SYNTAX    MafAgentEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "A Agent Entry"
    INDEX    {  mafAgentAuthority, mafAgentIdentity, mafASType }
    ::=  { mafAgentTable 1 }

MafAgentEntry  ::=  SEQUENCE {
    mafAgentAuthority  DisplayString,
    mafAgentIdentity  DisplayString,
    mafASType  INTEGER,
    mafAgentStatus  INTEGER,
        mafAgentLocation  DisplayString (SIZE (0..255))
    }

mafAgentAuthority    OBJECT-TYPE
    SYNTAX    DisplayString
    MAX-ACCESS    not-accessible
    STATUS    current

    DESCRIPTION
        "The Authority of the agent."

    ::=  {  mafAgentEntry  1  }

mafAgentIdentity    OBJECT-TYPE
    SYNTAX    DisplayString
    MAX-ACCESS    not-accessible
    STATUS    current

    DESCRIPTION
        "The Identity of the agent."

    ::=  {  mafAgentEntry  2  }

mafASType    OBJECT-TYPE
              SYNTAX  INTEGER {
                    nonAgentSystem(0),
                    aglets(1),
                    MOA(2),
                    agentTCL(3),
                    grasshopper(5)
                    }
    MAX-ACCESS    not-accessible
    STATUS    current

    DESCRIPTION
        "The type of the agent system."

    ::=  {  mafAgentEntry  3  }

mafAgentStatus    OBJECT-TYPE
              SYNTAX  INTEGER {
                    running(0),
                    suspended(1),
                    terminated(2)
                    }
    MAX-ACCESS    read-only
    STATUS    current

    DESCRIPTION
        "The status of the agent."

    ::=  {  mafAgentEntry  4  }

mafAgentLocation OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                    "Location of the agent."
              ::= { mafAgentEntry 5 }
```

```
mafAgentRequiredStatus OBJECT-TYPE
               SYNTAX  INTEGER {
                     suspend(1),
                     resume(2),
                     terminate(3)
                     }
               ACCESS  write-only
               STATUS  current
               DESCRIPTION
                       "Required status of this agent."
               ::= { mafAgentEntry 6 }

-------------------------------------------------------------
-- Agent Creation Group
-------------------------------------------------------------
mafCreateAgent    OBJECT IDENTIFIER  ::=  {  mafObjects  6  }

mafCreateAgentAuthority OBJECT-TYPE
               SYNTAX  DisplayString (SIZE (0..255))
               ACCESS  read-write
               STATUS  current
               DESCRIPTION
                       "The authority of the agent to be created.
                        Ignored if the user does not have naming responsibility."
               DEFVAL  { ''H }
               ::= { mafCreateAgent 1 }

mafCreateAgentIdentity OBJECT-TYPE
               SYNTAX  DisplayString (SIZE (0..255))
               ACCESS  read-write
               STATUS  current
               DESCRIPTION
                       "The identity of the agent to be created.
                        Ignored if the user does not have naming responsibility."
               DEFVAL  { ''H }
               ::= { mafCreateAgent 2 }

mafCreateAgentClassName OBJECT-TYPE
               SYNTAX  DisplayString (SIZE (0..255))
               ACCESS  read-write
               STATUS  current
               DESCRIPTION
                       "The name of the class necessary to instantiate
                        the agent."
               DEFVAL  { ''H }
               ::= { mafCreateAgent 3 }

mafCreateAgentCodeBase OBJECT-TYPE
               SYNTAX  DisplayString (SIZE (0..255))
               ACCESS  read-write
               STATUS  current
               DESCRIPTION
                       "The code base from where the classes will
                        be loaded."
               DEFVAL  { ''H }
               ::= { mafCreateAgent 4 }

mafCreateAgentPlaceName OBJECT-TYPE
               SYNTAX  DisplayString (SIZE (0..255))
               ACCESS  read-write
               STATUS  current
               DESCRIPTION
                       "The name of the place where the agent
                        will be created."
               DEFVAL  { ''H }
               ::= { mafCreateAgent 5 }

mafCreateAgentArguments OBJECT-TYPE
               SYNTAX  DisplayString (SIZE (0..255))
               ACCESS  read-write
               STATUS  current
               DESCRIPTION
                       "The arguments provided to the agent constructor."
               DEFVAL  { ''H }
               ::= { mafCreateAgent 6 }

mafCreateAgentGo OBJECT-TYPE
```

```
              SYNTAX  INTEGER {
                    waiting(0),
                    create(1)
                    }
              ACCESS  read-write
              STATUS  current
              DESCRIPTION
                      "Used to create the agent."
              DEFVAL { 0 }
              ::= { mafCreateAgent 7 }

-------------------------------------------------------------
-- Lookup group
-- information from MAFFinder interface
-------------------------------------------------------------

-------------------------------------------------------------
-- Filter table which fills the results tables
-------------------------------------------------------------

mafFilterTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF MafFilterEntry
    ACCESS  not-accessible
    STATUS  current
    DESCRIPTION
        "This table defines filters to lookup for
         Places, Agents and AgentSystems."
    ::= { mafLookup 1 }

   mafFilterEntry OBJECT-TYPE
       SYNTAX  MafFilterEntry
       ACCESS  not-accessible
       STATUS  current
       DESCRIPTION
           "An entry describing a particular search filter.

            Unless noted otherwise, writable objects of this row
            can be modified independent of the current value of
            mafFilterRowStatus, mafFilterAdminStatus and mafFilterOperStatus."
       INDEX { mafFilterOwner, mafFilterName }
       ::= { mafFilterTable 1 }

   MafFilterEntry ::= SEQUENCE {
       mafFilterOwner          SnmpAdminString,
       mafFilterName           SnmpAdminString,
       mafFilterDescr          SnmpAdminString,
       mafFilter               SnmpAdminString,
       mafFilterSearchFor      BITS,
       mafFilterAdminStatus    INTEGER,
       mafFilterOperStatus     INTEGER,
       mafFilterErrors         Counter32,
       mafFilterRowStatus      RowStatus
   }

   mafFilterOwner OBJECT-TYPE
       SYNTAX      SnmpAdminString (SIZE(0..32))
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The owner of this filtering entry. The exact semantics of
            this string are subject to the security policy defined by
            the security administrator."
       ::= { mafFilterEntry 1 }

   mafFilterName OBJECT-TYPE
       SYNTAX      SnmpAdminString (SIZE(1..32))
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The locally-unique, administratively assigned name for this
            filtering entry. This object allows a mafFilterOwner to have
            multiple entries in the mafFilterTable."
       ::= { mafFilterEntry 2 }

   mafFilterDescr OBJECT-TYPE
       SYNTAX      SnmpAdminString
       MAX-ACCESS  read-create
       STATUS      current
```

```
        DESCRIPTION
            "The human readable description of the purpose of this
             filtering entry."
        DEFVAL { ''H }
        ::= { mafFilterEntry 3 }

    mafFilter OBJECT-TYPE
        SYNTAX      SnmpAdminString
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The filter string to be used in the search."
        ::= { mafFilterEntry 4 }

    mafFilterSearchFor OBJECT-TYPE
        SYNTAX      BITS {
                        agents(0),
                        agentSystems(1),
                        places(2)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The type of objects this filtering entry should look for.
             Setting multiple bits will include several searches
             keeping results on separate tables."
        DEFVAL { {} }
        ::= { mafFilterEntry 5 }

    mafFilterAdminStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        searching(1),
                        stopped(2)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The desired state of the filter."
        DEFVAL { stopped }
        ::= { mafFilterEntry 6 }

    mafFilterOperStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        searching(1),
                        stopped(2),
                        finished(3)
                    }
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The current operational state of this filter. The state
             searching(1) indicates this entry is active and that the
             agent is searching. The stopped(2) state indicates that
             this entry is currently inactive. The finished(3)
             state indicates that the search is over and the results
             are available on the correspondent tables."
        ::= { mafFilterEntry 7 }

    mafFilterErrors OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
        "The number of errors encountered while performing this
        search."
       ::= { mafFilterEntry 8 }

    mafFilterRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The status of this filter. A control that allows
             entries to be added and removed from this table.

             Unless noted otherwise, writable objects of this row
             can be modified independent of the current value of
             mafFilterRowStatus, mafFilterAdminStatus and mafFilterOperStatus.
```

```
            In particular, it is legal to modify mafFilter
            and mafFilterSearchFor when
            mafFilterRowStatus is active and mafFilterAdminStatus and
            mafFilterOperStatus are both 'searching'.

            The miminum number of objects that need to be set during
            row creation before a row can be set to `active' are
            mafFilter and mafFilterSearchFor."
       ::= { mafFilterEntry 9 }

-------------------------------------------------------------
-- Search error table
-------------------------------------------------------------
mafFilterErrorTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF MafFilterErrorEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A table of errors occurred during search operations."
    ::= { mafLookup 2 }

mafFilterErrorEntry OBJECT-TYPE
    SYNTAX      MafFilterErrorEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "Information about errors in MAF operation."
    INDEX       { mafFilterOwner, mafFilterName }
    ::= { mafFilterErrorTable 1 }

MafFilterErrorEntry ::= SEQUENCE {
  mafFilterErrorIndex    Integer32,
  mafFilterErrorCode     INTEGER
}

mafFilterErrorIndex OBJECT-TYPE
   SYNTAX  Integer32
   ACCESS  read-only
   STATUS  current
   DESCRIPTION
      "The one-dimensioned character array index into mafFilter
       for where the error occurred. The value zero indicates
       irrelevance."
   ::= { mafFilterErrorEntry 1 }

mafFilterErrorCode OBJECT-TYPE
    SYNTAX      INTEGER {
                  namingServiceNotFound(0),
                  finderNotFound(1),
                  argumentInvalid(2),
                  mafExtendedException(3),
                  genericError(4)
                }
    ACCESS      read-only
    STATUS      current
    DESCRIPTION
     "A unique value for every error."
    ::= { mafFilterErrorEntry 2 }

-------------------------------------------------------------
-- Agents results table
-------------------------------------------------------------

mafLocAgentTable    OBJECT-TYPE
    SYNTAX    SEQUENCE  OF  MafLocAgentEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "List of found agents."
    ::=  { mafLookup  3 }

mafLocAgentEntry    OBJECT-TYPE
    SYNTAX    MafLocAgentEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "A Agent Entry"
    INDEX    { mafFilterOwner, mafFilterName, mafLocAgentIndex }
```

```
    ::=  { mafLocAgentTable 1 }

MafLocAgentEntry  ::=  SEQUENCE {
    mafLocAgentIndex  Unsigned32 (1..4294967295),
    mafLocAgentLocation  DisplayString (SIZE (0..255))
}

mafLocAgentIndex    OBJECT-TYPE
    SYNTAX    Unsigned32 (1..4294967295)
    MAX-ACCESS    not-accessible
    STATUS    current

    DESCRIPTION
        "The Index of the AgentSystem place."

    ::=  {  mafLocAgentEntry  1  }

mafLocAgentLocation OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Location of the place."
              ::= { mafLocAgentEntry 2 }

-------------------------------------------------------------
-- Agent System results table
-------------------------------------------------------------
mafLocASTable    OBJECT-TYPE
    SYNTAX    SEQUENCE  OF  MafLocASEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "List of found agent system places."
    ::=  { mafLookup 4 }

mafLocASEntry    OBJECT-TYPE
    SYNTAX    MafLocASEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "An AS Entry"
    INDEX    { mafFilterOwner, mafFilterName, mafLocASIndex }
    ::=  { mafLocASTable 1 }

MafLocASEntry  ::=  SEQUENCE {
    mafLocASIndex  Unsigned32,
    mafLocASLocation  DisplayString (SIZE (0..255))
}

mafLocASIndex    OBJECT-TYPE
    SYNTAX    Unsigned32 (1..4294967295)
    MAX-ACCESS    not-accessible
    STATUS    current

    DESCRIPTION
        "The Index of the AgentSystem place."

    ::=  {  mafLocASEntry  1  }

mafLocASLocation OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Location of the place."
              ::= { mafLocASEntry 2 }

-------------------------------------------------------------
-- Places result table
-------------------------------------------------------------
mafLocPlaceTable    OBJECT-TYPE
    SYNTAX    SEQUENCE  OF  MafLocPlaceEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "List of found agent system places."
    ::=  { mafLookup 5 }
```

```
mafLocPlaceEntry    OBJECT-TYPE
    SYNTAX    MafLocPlaceEntry
    MAX-ACCESS    not-accessible
    STATUS    current
     DESCRIPTION
        "A Place Entry"
    INDEX    {  mafFilterOwner, mafFilterName, mafLocPlaceIndex }
    ::=  { mafLocPlaceTable 1 }

MafLocPlaceEntry  ::=  SEQUENCE {
    mafLocPlaceIndex  Unsigned32,
    mafLocPlaceLocation  DisplayString (SIZE (0..255))
    }

mafLocPlaceIndex    OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current

    DESCRIPTION
        "The Index of the AgentSystem place."

    ::=  {  mafLocPlaceEntry  1  }

mafLocPlaceLocation OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  current
              DESCRIPTION
                      "Location of the place."
              ::= { mafLocPlaceEntry 2 }

END
```

Anexo D.
Exemplo de um macro para a Schedule MIB

Função: Define uma operação de repetição periódica indexada por *owner* 'rui' e nome 'teste' com período de 10 segundos e uma operação calendarizada indexada por *owner* 'rui' e nome 'teste2' que ocorre em todos os dias da semana às 0h.

```
<snmp version="v2c" community="public" writeCommunity="private">
  <task name="rui.teste">
    <set destination="snmp://localhost:10161/??v2c">
      <varBind       oid=".1.3.6.1.2.1.63.1.2.1.3.3.114.117.105.5.116.101.115.116.101"
value="112.101.114.105.111.100.105.99.111" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind       oid=".1.3.6.1.2.1.63.1.2.1.4.3.114.117.105.5.116.101.115.116.101"
value="10" type="Unsigned32"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.10.3.114.117.105.5.116.101.115.116.101"
value="114.111.117.116.101.114.115" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.11.3.114.117.105.5.116.101.115.116.101"
value=".1.3.6.1.4.1.1.12499.1.5.1.6" type="OBJECT IDENTIFIER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.12.3.114.117.105.5.116.101.115.116.101"
value="1" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.13.3.114.117.105.5.116.101.115.116.101"
value="1" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.14.3.114.117.105.5.116.101.115.116.101"
value="1" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.19.3.114.117.105.5.116.101.115.116.101"
value="1" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind      oid=".1.3.6.1.2.1.63.1.2.1.20.3.114.117.105.5.116.101.115.116.101"
value="1" type="INTEGER"/>
    </set>
  </task>
  <task name="rui.teste2">
    <set destination="snmp://localhost:10161/??v2c">
      <varBind    oid=".1.3.6.1.2.1.63.1.2.1.3.3.114.117.105.6.116.101.115.116.101.50"
value="99.97.108.101.110.100.97.114.105.111" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind    oid=".1.3.6.1.2.1.63.1.2.1.5.3.114.117.105.5.116.101.115.116.101.50"
value="127" type="OCTET STRING"/>
    </set>
```

```
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.6.3.114.117.105.5.116.101.115.116.101.50"
value="0.0" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.7.3.114.117.105.5.116.101.115.116.101.50"
value="0.0.0.0" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.8.3.114.117.105.5.116.101.115.116.101.50"
value="0.0.0" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.9.3.114.117.105.5.116.101.115.116.101.50"
value="0.0.0.0.0.0.0.0" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.10.3.114.117.105.6.116.101.115.116.101.50"
value="115.119.105.116.99.104.101.115" type="OCTET STRING"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.11.3.114.117.105.6.116.101.115.116.101.50"
value=".1.3.6.1.4.1.1.12499.1.5.1.5" type="OBJECT IDENTIFIER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.12.3.114.117.105.6.116.101.115.116.101.50"
value="2" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.13.3.114.117.105.6.116.101.115.116.101.50"
value="2" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.14.3.114.117.105.6.116.101.115.116.101.50"
value="1" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.19.3.114.117.105.6.116.101.115.116.101.50"
value="1" type="INTEGER"/>
    </set>
    <set destination="snmp://localhost:10161/??v2c">
      <varBind   oid=".1.3.6.1.2.1.63.1.2.1.20.3.114.117.105.6.116.101.115.116.101.50"
value="1" type="INTEGER"/>
    </set>
  </task>
</snmp>
```

Anexo E.
Cenários práticos de utilização da Expression MIB

Este anexo apresenta alguns exemplos da utilização de expressões em casos práticos de gestão e a sua tradução para a Expression MIB.

Exemplo 1 – Taxa de perda de rotas de encaminhamento ao nível da ligação de dados

Este exemplo aplica-se às pontes ou comutadores e calcula a taxa do número total de rotas de encaminhamento que foram ou que seriam aprendidas mas que foram eliminadas devido a falta de espaço para as armazenar. Quando este valor aumenta, indica que a tabela de encaminhamento se encontra regularmente cheia, o que pode resultar em problemas de desempenho na rede. Os objectos encontram-se definidos na Bridge MIB [RFC1493].

$$\text{taxa} = \frac{\Delta \text{dot1dTpLearnedEntryDiscards}}{\Delta \text{tempo em segundos}}$$

Para calcular a expressão é necessário consultar o dispositivo duas vezes, nos instantes t1 e t2. Se a taxa for superior a zero então existe um problema e o utilizador deverá ser notificado.

Após definir a expressão, é necessário traduzi-la para uma forma compatível com a Expression MIB. É necessário descrever a expressão em termos de objectos MIB de acordo com a sintaxe do modelo SNMP. Neste caso será:

```
expExpression.3."adm".4."lost"              = "$1"
expExpressionValueType.3."adm".4."lost"     = integer32
expExpressionDeltaInterval.3."adm".4."lost" = 5
expExpressionRowStatus.3."adm".4."lost"     = 'active'

expObjectID.3."adm".4."lost".1              = dot1dTpLearnedEntryDiscards.0
expObjectSampleType.3."adm".4."lost".1      = 'deltaValue'
expObjectRowStatus.3."adm".4."lost".1       = 'active'
```

Não é necessário normalizar a expressão, isto é, dividir pela diferença entre amostras consecutivas pelo intervalo de amostragem, uma vez que a diferença entre os instantes será sempre a mesma. Assim, a expressão é composta por um único parâmetro com amostragem do tipo `delta`. Por último, de notar que não há objectos do tipo *wildcarded*.

Exemplo 2 – Percentagem de processos activos

Este exemplo adequa-se às estações de trabalho para transmitir uma ideia dos recursos ocupados. A expressão calcula a razão entre os processos actuais e o número máximo de

processos que o sistema suporta. Os objectos encontram-se definidos na Host Resources MIB [RFC2790].

$$\text{percentagem de processos} = \frac{\text{hrSystemProcesses}}{\text{hrSystemMaxProcesses}} \times 100$$

A configuração correspondente para a Expression MIB é:

```
expExpression.3."adm".3."sys"                = "$1/$2*100"
expExpressionValueType.3."adm".3."sys"       = unsigned32
expExpressionDeltaInterval.3."adm".3."sys"   = 5
expExpressionRowStatus.3."adm".3."sys"       = 'active'

expObjectID.3."adm".3."sys".1                = hrSystemProcesses.0
expObjectSampleType.3."adm".3."sys".1        = 'absoluteValue'
expObjectConditional.3."adm".3."sys".1       = hrSystemMaxProcesses.0
expObjectRowStatus.3."adm".3."sys".1         = 'active'
expObjectID.3."adm".3."sys".2                = hrSystemMaxProcesses.0
expObjectSampleType.3."adm".3."sys".2        = 'absoluteValue'
expObjectRowStatus.3."adm".3."sys".2         = 'active'
```

O objecto condicional é responsável por invalidar a expressão se o `hrSystemMaxProcesses` não se encontrar definido para o agente em causa. Neste caso, a expressão não deve ser calculada uma vez que tenta efectuar uma divisão por zero. Basta invalidar um único parâmetro para impedir que a expressão seja calculada, pelo que não é necessário invalidar o segundo parâmetro.

Exemplo 3 – Taxa de utilização de uma interface de rede para ligações *half-duplex*

Será necessário calcular a utilização da interface, como descrito em [Leinwand96]. Os objectos necessários encontram-se definidos na MIB-II [RFC1213].

$$\text{utilização} = \frac{(\Delta\text{ifInOctets} + \Delta\text{ifOutOctets}) \times 8}{(\Delta\text{tempo em segundos}) \times \text{ifSpeed}} \times 100$$

Se a ligação for *full-duplex*, a expressão poderá resultar num valor até 200%, pelo que será necessário calcular o máximo dos valores `ifInOctets` and `ifOutOctets`. No entanto, esta abordagem esconde a direcção com o menor valor e apresenta resultados menos precisos. Será preferível calcular duas expressões, uma para cada sentido, e considerer as ligações *half-duplex*.

A tradução para objectos da Expression MIB encontra-se definida em [RFC2982] pelo que este processo não é aqui apresentado.

Exemplo 4 – Exactidão (*accuracy*) da interface

A exactidão (ou *accuracy*) de uma interface corresponde à percentagem de tráfego que não resulta em erro [Cisco]. A expressão calcula a razão entre o número de erros ocorridos e o número total de pacotes e subtrai este valor de 100 para obter a exactidão da interface.

$$\text{exactidão} = 100 - \frac{\Delta\text{ifInErrors}}{\Delta\text{ifInUcastPkts} + \Delta\text{ifInNUcastPkts}} \times 100$$

Expression 1: Interface accuracy.

Uma exactidão de 100% implica que não aconteceram erros e uma exactidão de 97% significa que em cada 100 pacotes foram perdidos 3 devido a erros. Os objectos da Expression MIB são configurados da seguinte forma:

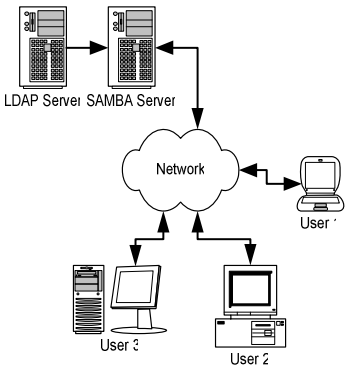
```
expExpression.3."adm".4."cond"                = "$1==1"
expExpressionValueType.3."adm".4."cond"       = unsigned32
```

```
expExpressionRowStatus.3."adm"4."cond"                = 'active'

expExpression.3."adm".4."accu"                        = "100-$1/($2+$3)*100"
expExpressionValueType.3."adm".4."accu"               = integer32
expExpressionDeltaInterval.3."adm".4."accu"           = 5
expExpressionRowStatus.3."adm"4."accu"                = 'active'

expObjectID.3."adm".4."cond".1                        = ifConnectorPresent
expObjectWildcard.3."adm".4."cond".1                  = 'true'
expObjectSampleType.3."adm".4."cond".1                = 'absoluteValue'
expObjectRowStatus.3."adm".4."cond".1                 = 'active'

expObjectID.3."adm".4."accu".1                        = ifInErrors
expObjectWildcard.3."adm".4."accu".1                  = 'true'
expObjectSampleType.3."adm".4."accu".1                = 'deltaValue'
expObjectConditional.3."adm".4."accu".1               =
expValueUnsigned32Val.3."adm".4."accu".0.0
expObjectConditionalWildcard.3."adm".4."accu".1       = 'true'
expObjectDiscontinuityID.3."adm".4."accu".1           = ifCounterDiscontinuityTime
expObjectDiscontinuityIDWildcard.3."adm".4."accu".1   = 'true'
expObjectRowStatus.3."adm".4."accu".1                 = 'active'

expObjectID.3."adm".4."accu".2                        = ifInUcastPkts
expObjectWildcard.3."adm".4."accu".2                  = 'true'
expObjectSampleType.3."adm".4."accu".2                = 'deltaValue'
expObjectRowStatus.3."adm".4."accu".2                 = 'active'

expObjectID.3."adm".4."accu".3                        = ifInNUcastPkts
expObjectWildcard.3."adm".4."accu".3                  = 'true'
expObjectSampleType.3."adm".4."accu".3                = 'deltaValue'
expObjectRowStatus.3."adm".4."accu".3                 = 'active'
```

Tal como no Exemplo 3, é utilizada uma operação de igualdade para definer o objecto condicional para validar a expressão. Esta deverá ser calculada apenas quando a interface se encontra ligada à rede (`ifConnectorPresent==1`). Os parâmetros seguintes correspondem directamente aos objectos da `ifTable`.

Exemplo 5 – Correlação de serviços

Este último exemplo incide sobre a Expression MIB modificada de forma a utilizar expressões sobre valores indicados por URLs. Esta alteração introduz a possibilidade para correlacionar serviços diferentes mas relacionados. Por exemplo, as contas de utilização são armazenadas num servidor LDAP enquanto que os ficheiros se encontram disponíveis num servidor SAMBA (http://www.samba.org/). Este último depende do servidor LDAP para obter informação sobre os utilizadores:



Do ponto de vista da gestão de redes, se algum dos services falhar então o acesso aos ficheiros por parte dos utilizadores falha também. O objectivo deste exemplo é verificar o correcto funcionamento do serviço como um todo. Para o efeito, foi definida e integrada na Expression MIB a função `checkService($1)`, onde $1 representa um URL. Esta função

devolve verdadeiro (valor 1) se o serviço responde correctamente e falso (valor 0) se o service não responde. Somando o resultado de duas funções, obtemos o resultado global.

$$total = checkService(\$1) + checkService(\$2)$$

A descrição da expressão é:

```
expExpression.3."adm".4."srvc"                = "checkService($1)+checkService($2)"
expExpressionValueType.3."adm".4."srvc"       = Unsigned32
expExpressionDeltaInterval.3."adm".4."srvc" = 20
expExpressionRowStatus.3."adm".4."srvc"       = 'active'

expObjectID.3."adm".4."srvc".1                = ldap://ldap.ipb.pt
expObjectSampleType.3."adm".4."srvc".1        = 'absoluteValue'
expObjectRowStatus.3."adm".4."srvc".1         = 'active'

expObjectID.3."adm".4."srvc".2                = smb://samba.ipb.pt
expObjectSampleType.3."adm".4."srvc".2        = 'absoluteValue'
expObjectRowStatus.3."adm".4."srvc".2         = 'active'
```

Referências

Abdu99	H. Abdu, H. Lutfiyya, M. Bauer, "A Model for Adaptive Monitoring Configuration", *actas do IFIP/IEEE 6th International Symposium on Integrated Network Management – IM'99*, Maio 1999.
Aglets	Aglets (http://www.aglets.org/).
Amaro01	J. Amaro, R. Lopes, "A Wireless MAN in Bragança – Digital City", *actas da 3ª Conferência de Telecomunicações – ConfTele2001*, Figueira da Foz, Portugal, 23-24 Abril 2001.
Ant	The ANT project (http://jakarta.apache.org/).
Appel98	A. Appel, *A Modern Compiler Implementation in Java*, Cambridge University Press, 1998, ISBN 0-521-58388-8.
Baldi97	M. Baldi, S. Gai, G. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management", *actas da 1st International Workshop on Mobile Agents – Mobile Agents'97*, Berlin, Alemanha, 7-8 de Abril 1997.
Baldi98	M. Baldi, G. Picco, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications", *actas da 20th International Conference on Software Engineering – ICSE'98*, Kyoto, Japão, Abril 1998.
Bellissard99	L. Bellissard, N. De Palma, A.Freyssinet, M. Herrmann, S. Lacourte, "Agent Infrastructure: The Agent Anytime Anywhere Platform", *relatório técnico submetido ao projecto C3DS (Control and Coordination of Complex Distributed Services)*, 1999 (http://www.newcastle.research.ec.org/c3ds/trs/papers/14.pdf).
Bieszczad98	A. Bieszczad, B. Pagurek, T. White, "Mobile Agents for Network Management", *IEEE Communications Surveys*, Setembro 1998.
Bloomer92	J. Bloomer, *Power Programming with RPC*, O'Reilly & Associates, Inc., 1992.
Blumenthal97	U. Blumenthal, B. Wijnen, "Security Features of SNMPv3", *The Simple Times, Vol. 5, N. 1*, Dezembro 1997.
Breugst98	M. Breugst, T. Magedanz, "Mobile Agents – Enabling Technology for Active Intelligent Network Implementation", *IEEE Network Magazine, Special Issue on Active and Programmable Networks*, Vol. 12 N. 3, Maio/Junho 1998.
Breugst99	M. Breugst, S. Choy, "Management of Mobile Agent Based Services", *actas da 6th International Conference on Intelligence in Services and Networks – IS&N'99*, Barcelona, Espanha, Abril 1999.

Chisholm02 — S. Chisholm, D. Romascanu, "Alarm MIB", draft-ietf-disman-alarm-mib-07.txt, June 2002.

Choy99 — S. Choy, M. Breugst, M. Datta, "Management Issues of a Mobile Agent-Based Service Environment", *Journal of Network and Systems Management*, Vol.7, Nº3, 1999.

CIM — Common Information Model (CIM) Specification v2.2, Junho 1999. (http://www.dmtf.org/).

CIMCore — Common Information Model (CIM) Core Model v2.4, Agosto 2000. (http://www.dmtf.org).

Cisco — Cisco Systems, *Performance Management: Best Practices White Paper*.

Corba00 — OMG, The Common Object Request Broker: Architecture and Specification, v.2.4, Outubro 2000. (http://www.omg.org/).

Cormen01 — T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms – Second Edition*, MIT Press, Setembro 2001.

Di Pietro00 — E. Di Pietro, A. La Corte, O. Tomarchio, A. Puliafito, "Extending the MASIF Location Service in the MAP Agent System", *actas do IEEE Symposium on Computer Communications – ISCC'2000*, Antibes, França, Julho 2000.

Dillenseger00 — B. Dillenseger, "MobiliTools: An OMG Standards-Based Toolbox for Agent Mobility and Interoperability", France Telecom, 2000.

DISMAN — DISMAN Charter (http://www.ietf.org/html.charters/disman-charter.html).

DMI — Desktop Management Interface Specification v2.0, Junho 1998. (http://www.dmtf.org/).

DMI2SNMP — DMI to SNMP Mapping Standard v1.0, Novembro 1997. (http://www.dmtf.org).

FIPA — Foundation for Intelligent Physical Agents, (http://www.fipa.org/).

Gavalas00 — D. Gavalas, D. Greenwood, M. Ghanbari, M. O'Mahony, "Advanced Network Monitoring Applications Based on Mobile/Intelligent Agent Technology", em *Computer Communications Journal, special issue on Mobile Agents for Telecommunication Applications*, Vol. 23, No 8, pp. 720-730, Abril 2000.

Gleizes99 — M. Gleizes, A. Léger, E. Athanassiou, P. Glize, "Self-Organization and Learning in MultiAgent Based Brokerage Services", *actas da 6th International Conference on Intelligence in Services and Networks – IS&N'99*, Barcelona, Espanha, Abril 1999.

Goldszmidt98 — G. Goldszmidt, Y. Yemini, "Delegated Agents for Network Management", *IEEE Communications Magazine, Vol. 36 No. 3, pgs. 66-71*, Março 1998.

Grasshopper — Grasshopper Agent Platform, (http://www.grasshopper.de/).

Green97 — S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, R. Evans, *Software Agents: A review*, Maio 1997.

Ismail98 — L. Ismail, D. Hagimont, J. Mossière, "Evaluation of the Mobile Agents Technology and Comparison Studies", *SIRAC Project (IMAG-INRIA) INRIA*, 1998.

ISO10040 — ISO/IEC 10040, Information Processing Systems – Open Systems Interconnection – Systems Management Overview. (ITU-T Recommendation X.701, Information technology - Open Systems Interconnection - Systems Management: Overview).

ISO10165 — ISO/IEC 10165, GDMO – Guidelines for Definition of Managed Objects. (ITU-T Recommendation X.720, Information technology - Open Systems Interconnection - Structure of Management Information: Management Information Model).

ISO7498-4 — ISO/IEC 7498-4, Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management Framework. (ITU-T Recommendation X.700, Management Framework Definition for Open Systems Interconnection).

ISO8824 — ISO/IEC 8824, Information Processing Systems - Open Systems Interconection - Specification of Abstract Syntax Notation One. (ITU-T Reccomendation X.680, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation).

ISO8825 — ISO/IEC 8825, Information Processing Systems - Open Systems Interconection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). (ITU-T Recommendation X.690, Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)).

ISO9596 — ISO/IEC 9596, Information technology - Open Systems Interconnection - Common Management Information Protocol - Part 1 Protocol: Specification. (ITU-T Recommendation X.711, Common Management Information protocol specification for CCITT Applications).

J2SDK — Java Platform Software Development Kit (http://www.javasoft.com).

JavaCC — Javacc (http://www.webgain.com/products/java_cc/).

JAX — Java AgentX Toolkit (http://www.ibr.cs.tu-bs.de/projects/jasmin/jax.html).

JDMK — Java Dynamic Management Kit 5.0 (http://www.sun.com/products-n-solutions/nep/software/java-dynamic/JDMK_5.0DS.pdf).

Jetty — Jetty Java HTTP Servlet Server (http://jetty.mortbay.com/).

JIDM — The Open Group, "Inter-Domain Management: Specification Translation and Interaction Translation", *JIDM Working Group*, Janeiro 2000.

JIDM_IT — The Open Group, "Book 2: Inter-Domain Management: Interaction Translation (JIDM_IT)", *JIDM Working Group*, Janeiro 2000.

JIDM_ST — The Open Group, "Book 1: Inter-Domain Management: Specification Translation (JIDM_ST)", *JIDM Working Group*, Janeiro 2000.

Jikes — (http://oss.software.ibm.com/developerworks/opensource/jikes/).

JMX — Sun Microsystems, "Java™ Management Extensions Instrumentation and Agent Specification, v1.0", (http://www.javasoft.com/).

JoeSNMP — JoeSNMP (http://www.opennms.org).

Kaasinen99 — E. Kaasinen, "Usability Challenges in Agent Based Services", *actas da 6th International Conference on Intelligence in Services and Networks – IS&N'99*, Barcelona, Espanha, Abril 1999.

Kato99 — K. Kato, K. Matsubara, Y. Someya, K. Itabashi, and Y. Moriyama, "Planet: An Open Mobile Object System for Open Network", *actas do IEEE Joint Symposium of 1st Int. Symposium on Agent Systems and Applications/Third Int. Symposium on Mobile Agents*, Outubro 1999 (http://www.osss.is.tsukuba.ac.jp/~planet/).

Kawaguchi99 — N. Kawaguchi, K. Toyama, Y. Inagaki, "MAGNET: Ad-hoc Network System Based on Mobile Agents", *actas da 1st International Wokshop on Mobile Agents for Telecommunications Applications – MATA'99*, Otawa, Canada, 6-8 Outubro 1999.

Kotz99 — D. Kotz, R. Gray, "Mobile Agents and the Future of the Internet", *ACM Operating Systems Review, 33(3)*, P. 7-13, August 1999 (http://agent.cs.dartmouth.edu/).

Kramer99 — K. Kramer, N. Minar, P. Maes, "Tutorial: Mobile Software Agents for Dynamic Routing", em *Mobile Computing and Communications Review* vol.3 no.2, Março 1999.

Krause96 — S. Krause, T. Magedanz, "Mobile Service Agents enabling Intelligence on Demand in Telecommunications", *actas da IEEE Global Telecommunications Conference – GLOBECOM'96*, 1996.

Ku97 — H. Ku, G. Luderer, B. Subbiah, "An Intelligent Mobile Agent Framework for Distributed Network Management", *actas da IEEE Global Telecommunications Conference – GLOBECOM'97*, Phoenix, USA, Novembro 1997.

Langsford94 — A. Langsford, "OSI Management Model and Standards", *Network and Distributed Systems Management*, Morris Sloman, Addison-Wesley, 1994.

Leinwand96 — A. Leinwand, K. Conroy, *Network Management. A Practical Perspective – 2ed*, Setembro 1996.

Levi99 — D. Levi, "Introduction to the Script MIB", *The Simple Times, Vol. 7, N. 2*, Novembro 1999.

Lindholm96 — T. Lindholm, F. Yellin, *The JavaTM Virtual Machine Specification (2nd Ed)*, Addison-Wesley, Abril, 1999.

Liotta99 — A. Liotta, G. Knight, G. Pavlou, "On the Efficiency of Decentralised Monitoring using Mobile Agents", *actas da IFIP/IEEE International Workshop on Distributed Systems: Operations & Management – DSOM'99*, Zuriqie, Suiça, 11-13 Outubro 1999.

Lopes00a — R. Lopes, J. Oliveira, "Managing Mobile Agents with SNMP", *actas do Simposio Español de Informatica Distribuida – SEID'2000*, Orense, Espanha, Setembro 2000.

Lopes00b — R. Lopes, J. Oliveira, "On the use of Mobility in Distributed Network Management", *actas da Hawaii International Conference on System Sciences – HICSS*, Maui, Hawaii, EUA, Janeiro 2000.

Lopes00c — R. Lopes, J. Oliveira, "Distributed Management: Implementation issues", *actas da International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet – SSGRR'2000*, l'Aquila, Roma, Itália, Agosto 2000.

Lopes01a — R. Lopes, J. Oliveira, "Distributed Management of Mobile Components", *actas da 3ª Conferência de Telecomunicações – ConfTele2001*, Figueira da Foz, Portugal, April 2001.

Lopes01b — R. Lopes, J. Oliveira, "SNMP Management of MASIF Platforms", *actas do IFIP/IEEE International Symposium on Integrated Management – IM'2001*, Seattle, Maio 2001.

Lopes02a — R. Lopes, J. Oliveira, "A multi-protocol architecture for SNMP entities", *actas da 2002 IEEE Workshop on IP Operations & Management – IPOM 2002*, Dallas, Outubro 2002.

Lopes02b — R. Lopes, J. Oliveira, "A uniform resource identifier scheme for SNMP", *actas da 2002 IEEE Workshop on IP Operations & Management – IPOM'2002*, Dallas, Outubro 2002.

Lopes99 — R. Lopes, J. Oliveira, "Software Agents in Network Management", *actas da 1st International Conference on Enterprise Information Systems – ICEIS'99*, Setúbal, Portugal, Março 1999.

MAF — Mobile Agent Facility Specification, Object Management Group, 00-01-02.pdf (ftp://ftp.omg.org/pub/docs/formal/00-01-02.pdf).

Manson90 — T. Manson, D. Brown, *lex & yacc*, O'Reilly & Associates, 1990, ISBN 0-837175-49-8.

MAP — MAP Agent System (http://sun195.iit.unict.it/MAP/).

Martin98 — J. Martin-Flatin, S. Znaty, J. Hubaux, "A Survey of Distributed Network and Systems Management Paradigms", *Technical Report SSC/1998/024, Swiss Federal Institute of Technology Lausanne*, Agosto 1998.

Minar98 — N. Minar, K. Kramer, P. Maes, "Cooperating Mobile Agents for Mapping Networks", *actas da 1st Hungarian Conference on Agent Based Computing*, 1998.

Mullaney96 — P. Mullaney, "Overview of a Web-based Agent", *The Simple Times, Vol 4, N. 3*, Julho 1996.

Nekrestyanov99 — I. Nekrestyanov, T. O'Meara, A. Patel, E. Romanova, "Building Topic-Specific Collections with Intelligent Agents", *actas da 6th International Conference on Intelligence in Services and Networks, IS&N'99*, Barcelona, Espanha, Abril 1999.

NetSNMP — The NET-SNMP Project (http://www.net-snmp.org/).

Oliveira94 — J. L. Oliveira, J. A. Martins, "A Management Architecture based on Network Topology Information", *Journal of Network and Systems Management*, Vol. 2, Nº 4, pg. 401-414, Dezembro 1994.

Oliveira95 — J. Oliveira, Arquitectura para Desenvolvimento e Integração de Aplicações de Gestão, *Tese de Doutoramento*, Universidade de Aveiro, Setembro 1995.

Oliveira99 — J. Oliveira, R. Lopes, "Distributed Management based on Mobile Agents", *actas da 1st International Workshop on Mobile Agents for Telecommunications Applications – MATA'99*, Ottawa, Canada, Outubro 1999.

Pagurek00 — B. Pagurek, Y. Wang, T. White, "Integration of Mobile agents with SNMP: Why and How", *actas da IEEE/IFIP Network Operations and Management Symposium – NOMS'2000*, Honolulu, 2000.

Pham98 — V. Pham, A. Karmouch, "Mobile Software Agents: An Overview", *IEEE Communications, Vol. 36, No. 7* (1998) pp. 26-37.

Postel98 — J. Postel, J. Reynolds, "Internet Official Protocol Standards", STD 1, Setembro 1998.

Quittek99 — J. Quittek, C. Kappler, "Practical Experiences with Script MIB Applications", *The Simple Times, Vol. 7, N. 2*, Novembro 1999.

RFC1066 — K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", *Internet Request for Comments 1066*, Agosto 1988.

RFC1155 — K. McCloghrie, M. Rose, "Structure and Identification of Management Information for TCP/IP-based Internets", *Internet Request for Comments 1155*, Maio 1990.

RFC1156 K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", *Internet Request for Comments 1156*, Maio 1990.

RFC1213 K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", *Internet Request for Comments 1213*, Março 1991.

RFC1157 J. Case, M. Fedor, M. Schoffstall, J. Davin, "A Simple Network Management Protocol (SNMP)", *Internet Request for Comments 1157*, Maio 1990.

RFC1451 J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Manager-to- Manager Management Information Base", *Internet Request for Comments 1451*, Abril 1993.

RFC1493 E. Decker, P. Langille, A. Rijsinghani, K. McCloghrie, "Definitions of Managed Objects for Bridges", *Internet Request for Comments 1493*, Julho 1993.

RFC1592 B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters, "The SNMP Distributed Protocol Interface, Version 2.0", *Internet Request for Comments 1592*, Março 1994.

RFC1630 T. Berners-Lee, "Universal Resource Identifiers in WWW", *Internet Request for Comments 1630*, Junho 1994.

RFC1738 T. Berners-Lee, L. Masinter, "Uniform Resource Locators (URL)", *Internet Request for Comments 1738*, Dezembro 1994.

RFC1905 J. Case, K. McCloghrie, M. Rose, S. Waldbusser," Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", *Internet Request for Comments 1905*, Janeiro 1996.

RFC1906 J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", *Internet Request for Comments 1906*, Janeiro 1996.

RFC1907 J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", *Internet Request for Comments 1907*, Janeiro 1996.

RFC2257 M. Daniele, B. Wijnen, D. Francisco, "Agent Extensibility Protocol Version 1", *Internet Request for Comments 2257*, Janeiro 1998.

RFC2396 T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", *Internet Request for Comments 2396*, Agosto 1998.

RFC2570 J. Case, R. Mundy, D. Partain, B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", *Internet Request for Comments 2570*, Abril 1999.

RFC2571 D. Harrington, R. Presuhn, B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", *Internet Request for Comments 2571*, Abril 1999.

RFC2572 J. Case, D. Harrington, R. Presuhn, B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", *Internet Request for Comments 2572*, Abril 1999.

RFC2573 D. Levi, P. Meyer, B. Stewart, "SNMP Applications", *Internet Request for Comments 2573*, Abril 1999.

RFC2574 U. Blumenthal, B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", *Internet Request form Comments 2574*, Abril 1999.

Comment [rpcl1]: Verificar actualização a este RFC no charter SNMPv3

Comment [rpcl2]: Verificar actualização a este RFC no charter SNMPv3.

RFC2575 B. Wijnen, R. Presuhn, K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", *Internet Request for Comments 2575*, Abril 1999.

RFC2576 R. Frye, D. Levi, S. Routhier, B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", *Internet Request for Comments 2576*, Março 2000.

RFC2578 K. McCloghrie, D. Perkins, J. Schoenwaelder (, J. Case, K. McCloghrie, M. Rose, S. Waldbusser), "Structure of Management Information Version 2 (SMIv2)", *Internet Request for Comments 2578*, Abril 1999.

RFC2579 K. McCloghrie, D. Perkins, J. Schoenwaelder (, J. Case, K. McCloghrie, M. Rose, S. Waldbusser), "Textual Conventions for SMIv2", *Internet Request for Comments 2579*, Abril 1999.

RFC2580 K. McCloghrie, D. Perkins, J. Schoenwaelder, (, J. Case, K. McCloghrie, M. Rose, S. Waldbusser), Conformance Statements for SMIv2", *Internet Request for Comments 2580*, Abril 1999.

RFC2616 R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", *Internet Request for Comments 2616*, Junho 1999.

RFC2790 S. Waldbusser, "Host Resources MIB", *Internet Request for Comments 2790*, Março 2000.

RFC2925 K. White, "Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations", *Internet Request for Comments 2925*, Setembro 2000.

RFC2981 R. Kavasseri (, B. Stewart), "Event MIB", *Internet Request for Comments 2981*, Outubro 2000.

RFC2982 R. Kavasseri (, B. Stewart), "Distributed Management Expression MIB", *Internet Request for Comments 2982*, Outubro 2000.

RFC3014 R. Kavasseri (, Bob Stewart), "Notification Log MIB", *Internet Request for Comments 3014*, Novembro 2000.

RFC3165 D. Levi, J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts", *Internet Request for Comments 3165*, Agosto 2001.

RFC3231 D. Levi, J. Schoenwaelder, "Definitions of Managed Objects for Scheduling Management Operations", *Internet Request for Comments 3231*, Janeiro 2002.

Rubinstein99 M. Rubinstein, O. Duarte, "Evaluating Tradeoffs of Mobile Agents in Network Management", *Journal of Networking and Information Systems, Vol. 2, N. 2, P. 237-252*, 1999.

Sahai98 A. Sahai, C. Morin, "Enabling a Mobile Network Manager (MNM) through mobile agents", *actas da 2nd International Workshop on Mobile Agents – MA'98*, Estugarda, Alemanha, 9-11 Setembro 1998.

Schoenwaelder00a J. Schoenwaelder, F. Strauss, "Using XML to Exchange SMI Definitions", Internet Draft draft-irtf-nmrg-smi-xml-00.txt, Junho 2000.

Schoenwaelder00b J. Schoenwaelder, J. Quittek, C. Kappler, "Building Distributed Management Applications with the IETF Script MIB", *IEEE Journal on Selected Areas in Communications, Vol. 18, N. 5*, Maio 2000.

Schoenwaelder01 J. Schoenwalder, A. Muller, "Reverse Engineering Internet MIBs", *actas do IFIP/IEEE International Symposium on Integrated Management – IM'2001*, Seattle, Maio 2001.

Schoenwaelder97 — J. Schoenwaelder, "Network Management by Delegation – From Research Prototypes Towards Standards", *Computer Networks and ISDN Systems*, 29(15): 1843-1852, Novembro 1997.

Sechrest86 — S. Sechrest, "An Introductory 4.4BSD Interprocess Communication Tutorial", *Computer Science Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkley*, 1986.

Silva01 — S. da Silva, Y. Yemini, D. Florissi, "The NetScript Active Network System", *Journal on Selected Areas in Communications, Vol. 19, N. 3*, Março 2001.

Simões00 — P. Simões, *Gestão Distribuída de Redes Baseada em Tecnologia de Agentes Móveis*, tese de doutoramento, Universidade de Coimbra, 2000.

Simões01 — P. Simões, L. Silva, F. Boavida, "Mobile Agent Infrastructures: a Solution for Management or a Problem to Manage?", *actas da 3rd Conference on Telecommunications – ConfTele'2001*, Figueira da Foz, Portugal, April 2001.

Simões02 — P. Simões, J. Rodrigues, L. Silva, F. Boavida, "Distributed Retrieval of Management Information: Is it About Mobility, Locality or Distribution?", *actas do Network Operations and Management Symposium – NOMS'2002*, Florença, Itália, 15-19 Abril 2002.

Simões99 — P. Simões, L. Silva, F. Fernandes, "Integrating SNMP into a Mobile Agent Infrastructure", *actas da 10th IEEE/IFIP Int. Workshop on Distributed Systems: Operations & Management – DSOM'99*, Zurique, Outubro 1999.

SOMA — SOMA: Secure and Open Mobile Agent (http://www-lia.deis.unibo.it/Research/SOMA/).

Stevens98 — W. Stevens, *UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI*, Prentice Hall, 1998.

Strauss00 — F. Strauss, "Distribution Models for Network Management Functions", *Relatório de Projecto*, Technische Universitat Braunschweig (http://www.ibr.cs.tu-bs.de/projects/jasmin/dismod.ps), Junho 2000.

Strauss01 — F. Strauss, J. Schoenwaelder, K. McCloghrie, "SMIng - Next Generation Structure of Management Information", *Internet Draft draft-ietf-sming-02.txt*, Julho 2001.

Sum99 — J. Sum, H. Shen, G. Young, "Analysis on a Mobile Agent based Algorithm for Network Management", *actas da International Conference on Parallel and Distributed Processing Techniques and Applications – PDPTA99*, Las Vegas, Nevada, EUA, 28 Junho a 1 de Julho 1999.

URISchemes — URI SCHEMES (http://www.iana.org/assignments/uri-schemes).

White98 — T. White, A. Bieszczad, B. Pagurek, "Distributed Fault Location in Networks Using Mobile Agents", *actas da 2nd International Workshop on Intelligent Agents in Telecommunications Applications – IATA'98, Springer Verlab Pub.*, Paris, 4-7 de Julho 1998.

Xalan — The XSLT processor Xalan (http://xml.apache.org/xalan-j/index.html).

Xerces — The XML parser Xerces (http://xml.apache.org/xerces-j/index.html).

XML98 — World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", W3C REC-xml-19980210 (http://www.w3.org/TR/1998/REC-xml-19980210), Fevereiro 1998

Yemini91 Y. Yemini, G. Goldszmidt, S. Yemini, "Network Management by Delegation", I. Krishnan and W. Zimmer (Eds.), *actas do IFIP 2nd Int. Symposium on Integrated Network Management – ISINM'91*, Washington, DC, EUA, Abril 1991.