J.AgentX: Uma Ferramenta Para Extensão Dinâmica de Agentes SNMP

Eduardo Lourenço, Pedro Pereira, Paulo Simões
Luís Silva, Fernando Boavida

CISUC - Dep. Eng. Informática
University of Coimbra, Polo II
P-3030 Coimbra, Portugal
psimoes@dei.uc.pt

Sumário. *Neste artigo será apresentada a ferramenta J.AgentX, a primeira implementação em Java da norma AgentX para extensão dinâmica de agentes SNMP.*
Para além das características mais interessantes desta ferramenta para desenvolvimento rápido de novos serviços de gestão serão descritas duas experiências concretas da sua aplicação: a instalação de serviços de gestão baseados em código móvel e o desenvolvimento de proxies.

Palavras-chave: *Gestão de Redes, SNMP, Extensão de Agentes SNMP*

1. INTRODUÇÃO

O *Simple Network Management Protocol* (SNMP) [1] é muitas vezes descrito como uma tecnologia pouco atraente, incapaz de responder aos requisitos de gestão actuais. No entanto, apesar das suas conhecidas limitações e do sucesso relativo de tecnologias competidoras, como o CORBA [2] e a gestão através da Web [3], o SNMP é ainda o protocolo mais difundido na gestão de sistemas e redes. Desta forma, mesmo os novos serviços de gestão necessitam de assegurar suporte SNMP para manter compatibilidade com as plataformas de gestão mais populares, tais como o HP Openview [4] e o Tivoli [5].

A arquitectura base do SNMP assenta numa estrutura rígida e monolítica pecando por estar limitada em termos de flexibilidade, adaptabilidade, descentralização e funcionalidade de alto nível, resultando quase sempre num compromisso: *funcionalidade* vs. *esforço de implementação*. Este compromisso resulta da necessidade, em muitas situações, de fornecer serviços de gestão baseados em SNMP sem que, no entanto, seja desejável levar a cabo o esforço de implementar um agente SNMP completo. Assim, muitas vezes, os agentes acabam por se revelar inoperantes, não satisfazendo as necessidades de gestão específicas da aplicação. Para solucionar estas limitações foram sendo introduzidas técnicas de extensão dinâmica de agentes, de modo a que o esforço de desenvolvimento seja focado nas funcionalidades de gestão e não nos detalhes protocolares do seu interface.

A extensão dinâmica de agentes SNMP permite separar o interface SNMP dos serviços de gestão fornecidos. O seu funcionamento poderá assim resumir-se a um "agente vazio" que apenas recebe os pedidos SNMP das aplicações de gestão e os redirecciona para os subagentes nele registados. Cada subagente é responsável por um conjunto específico de objectos SNMP e sabe como transformar esses pedidos em operações de gestão. O ideal seria que cada aplicação ou serviço de rede disponibilizasse dinamicamente o seu subagente para poder ser gerido via SNMP. Desta

forma, o agente SNMP deixa de ser monolítico já que se torna possível adicionar e remover arbitrariamente submódulos da MIB (*Management Information Base*) durante a execução do serviço, mesmo quando estes submódulos não sejam conhecidos *a priori*.

Este mecanismo introduz outras potenciais vantagens, tais como o desenvolvimento facilitado de serviços de gestão baseados em SNMP – é mais simples desenvolver um subagente que um agente SNMP completo – e maior portabilidade, uma vez que a maior parte dos subagentes é independente do sistema em que está inserido, ao contrário dos agentes SNMP monolíticos.

Tradicionalmente, a extensão de agentes foca-se no problema principal (ultrapassar as limitações dos agentes monolíticos) e tem negligenciado as outras vantagens potenciais. No nosso trabalho tivemos uma aproximação diferente: considerou-se que os *standards* actuais referentes à extensão de agentes já são satisfatórios e investiu-se mais na facilidade de desenvolvimento, na simplicidade, na portabilidade, na flexibilidade e na pesquisa de novos campos de aplicação.

O resultado desta abordagem levou ao desenvolvimento do `J.AgentX` [6], um pacote que constitui a primeira implementação em Java do standard AgentX [7]. Ao contrário de outras ferramentas, esta é totalmente dedicada à extensão de agentes e disponibiliza um interface de programação de alto nível que permite o desenvolvimento rápido de novos serviços. Para que seja possível explorar novas áreas de aplicação foram ainda introduzidos diversos melhoramentos ao nível da comunicação, com suporte simultâneo de diferentes mecanismos de comunicação.

Este artigo apresenta o pacote `J.AgentX` e propõe algumas formas alternativas de aplicação da extensão dinâmica de agentes SNMP. Neste panorama, foram feitas duas experiências específicas: uma com código móvel e outra usando subagentes *proxy*.

O resto do artigo está organizado da seguinte forma: a Secção 2 é dedicada à tecnologia de extensão de agentes e a Secção 3 apresenta o pacote `J.AgentX`. A Secção 4 propõe potenciais aplicações da ferramenta `J.AgentX` para descentralizar a gestão e a Secção 5 conclui o artigo.

2. EXTENSÃO DINÂMICA DE AGENTES SNMP

A extensão dinâmica de agentes SNMP é normalmente baseada em duas entidades: um agente *master* e vários subagentes. O agente *master* actua como um *front-end* SNMP, recebendo os pedidos SNMP do exterior e redireccionando-os para os respectivos subagentes, e reencaminhando para o exterior as respostas produzidas pelos subagentes. Cada um destes subagentes é especializado e fornece uma funcionalidade de gestão específica, implementando os objectos da MIB que representam essa funcionalidade. O subagente regista esses objectos no *master*. Posteriormente, os pedidos relativos a esses objectos chegam ao subagente, que os processa, devolvendo ao *master* os resultados correspondentes. Para as aplicações exteriores este processo é transparente, pois o que elas *vêem* é uma única entidade que fornece serviços SNMP - o agente *master*. A interacção entre o agente *master* e os subagentes é baseada num protocolo predefinido de extensão de agentes. A Figura 1 mostra esta arquitectura.

2.1 Protocolos para Extensão de Agentes

Existem várias soluções proprietárias para extensão de agentes que se apresentam como uma melhoria aos protocolos abertos existentes. Estas soluções reclamam melhor desempenho, maior flexibilidade e optimização para *hardware* e sistemas operativos específicos. No entanto, as vantagens efectivas são em regra diminutas e raramente justificam a falta de portabilidade no desenvolvimento de subagentes.

Só os protocolos abertos de extensão de agentes conseguem ser independentes de hardware ou de sistemas operativos, proporcionando interoperabilidade entre agentes *master* e subagentes de diferentes vendedores. Existem actualmente três destes protocolos: SMUX [8], DPI [9] e AgentX [7].

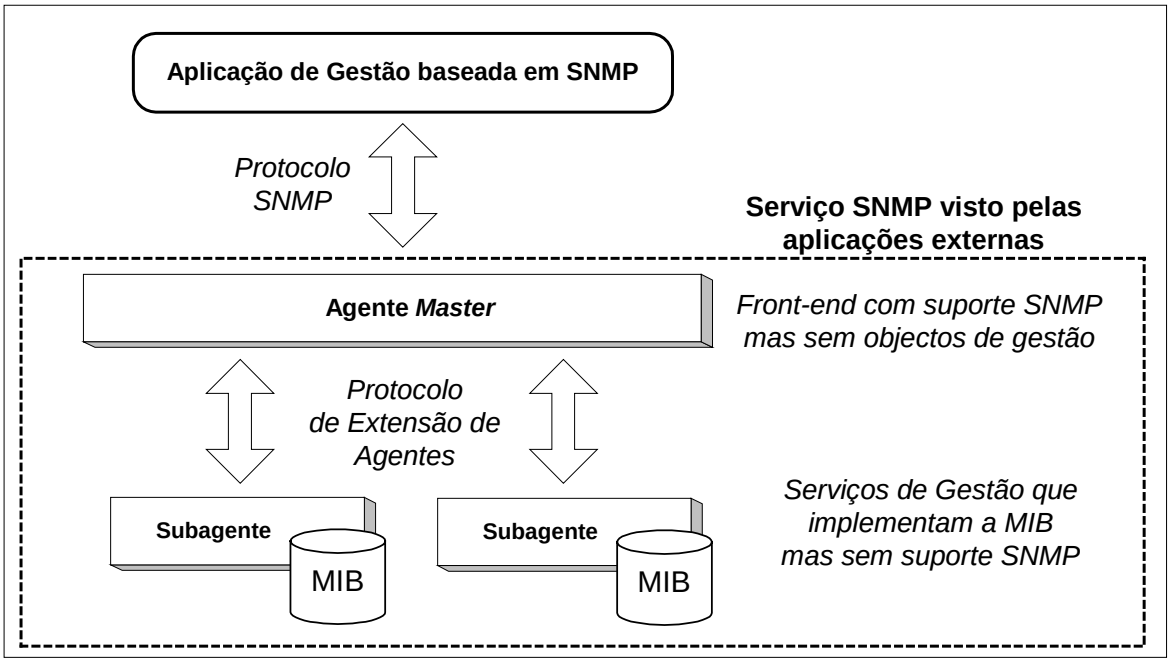

Figura 1 - Arquitectura Típica de Agentes SNMP Extensíveis

O SMUX (*SNMP Multiplexer Protocol*) é quase tão antigo como o próprio SNMP. No SMUX, a comunicação com os subagentes (ou *peers*, de acordo a terminologia do SMUX) é parcialmente baseada em mensagens SNMP, resultando em problemas de coordenação, sincronização e performance. No entanto, apesar destas limitações, o SMUX é ainda a base de alguns produtos comerciais.

O DPI (*SNMP Distributed Protocol Interface*) dá um passo em frente, definindo um protocolo específico para a interacção entre o agente *master* e os subagentes. Neste protocolo os subagentes já não precisam de "falar" SNMP sendo assim mais simples e pequenos.

O AgentX (*Agent eXtensibility Protocol*) foi proposto em 1998 como uma evolução do DPI apresentando melhorias em alguns aspectos técnicos e suportando diferentes versões do SNMP. Recentemente, este protocolo atingiu o estatuto de especificação *standard* do IETF e é esperado que substitua gradualmente as restantes soluções proprietárias e abertas.

2.2 Arquitectura e implementações do AgentX

A arquitectura do AgentX não é muito diferente da apresentada na Secção 2. Inclui um agente *master* a servir de interface com as aplicações SNMP mas com pouco ou nenhum acesso à informação de gestão, e propõe subagentes que implementam os serviços de gestão e a respectiva informação de gestão mas que não têm contacto directo com as aplicações SNMP. O protocolo AgentX fornece mecanismos para registo de subagentes, para administração da infra-estrutura e para operações SNMP (incluindo operações *Set* atómicas envolvendo vários subagentes). A comunicação usada é *connection-oriented* e o protocolo de transporte é o TCP, embora outros possam ser usados.

O protocolo também define uma MIB AgentX [10] especificamente para gerir a própria infraestrutura. Esta é um solução elegante para a gestão remota das entidades do AgentX (ex: observar subagentes, verificar erros internos, fechar subagentes) usando SNMP.

Existem neste momento cinco implementações conhecidas do AgentX: três de domínio público do mundo académico e dois produtos comerciais.

A primeira implementação do AgentX de domínio público foi integrada em fins de 1998 no pacote CMU SNMP [11]. Trata-se de uma implementação parcial que suporta SNMPv1 e SNMPv2c e é capaz de aceitar ligações via TCP ou *sockets* UNIX. Esta implementação inclui um agente *master* - uma evolução do agente SNMP da CMU - e algumas bibliotecas para desenvolvimento de subagentes.

O pacote UCD-SNMP v4.1 [12] implementa o AgentX de uma forma incompleta, já que apesar de conter um agente SNMP extensível que suporta SNMPv1, SNMPv2c e SNMPv3, não referencia qualquer suporte para o desenvolvimento de subagentes.

O `J.AgentX` [6] é, quanto julgamos saber, a primeira implementação em Java do protocolo. Ela inclui um agente *master*, várias bibliotecas para o desenvolvimento de subagentes e uma implementação da MIB AgentX [10]. O `J.AgentX` será descrito em mais detalhe na Secção 3.

Existem ainda alguns produtos não académicos que implementam o protocolo AgentX. A Compaq indica suporte para AgentX na seu Tru64 UNIX versão 5.0 [13]. Este suporte inclui um agente *master*, uma biblioteca para desenvolvimento de subagentes e a MIB do AgentX. O agente SNMP Envoy [14], da WindRiver, também reclama compatibilidade com o AgentX para o seu protocolo proprietário (Envoy +X).

3. O PACOTE J.AGENTX

O CMU-SNMP e o UCD-SNMP são dois pacotes tradicionais e generalistas do SNMP, implementados em linguagem C, orientados para UNIX, com um largo espectro de objectivos (desenvolvimento de agentes, de aplicações e de serviços) e que, ao longo dos anos, foram acumulando várias funcionalidades. Neste contexto, o suporte de AgentX é apenas mais uma dessas funcionalidades.

O `J.AgentX` foi desenhado de raíz com uma visão muito mais focalizada no AgentX e no desenvolvimento de serviços, resultando assim em soluções mais pequenas, simples e elegantes. Como já foi mencionado, os principais factores tomados em consideração foram:

- simplicidade – o pacote deve ser o mais simples possível;
- portabilidade – tanto o pacote como os serviços desenvolvidos devem ser realmente portáveis. Por esta razão, o `J.AgentX` é 100% Java e independente do sistema operativo;
- interoperabilidade – o `J.AgentX` deve oferecer total interoperabilidade com as restantes implementações do AgentX;
- flexibilidade – o pacote deve ser capaz de se ajustar a diferentes ambientes e diferentes campos de aplicação. Por exemplo, o pacote deve permitir a possibilidade de usar o AgentX com novos protocolos de comunicação e/ou suportar o uso simultâneo de diferentes protocolos;
- fácil desenvolvimento de serviços – os detalhes técnicos do SNMP e do AgentX devem ser camuflados atrás de APIs de programação de alto nível, levando o programador a preocupar-se essencialmente com os serviços a desenvolver e não com detalhes protocolares.

O resultado foi um pacote muito compacto com dois componentes independentes mas complementares (Figura 2):

- um agente *master* pequeno e leve;
- e um API-AgentX para desenvolvimento de subagentes AgentX.

3.1 O Agente master do `J.AgentX`

O agente *master* é uma aplicação que funciona como *front-end* SNMP com inserção dinâmica de MIBs usando o protocolo AgentX. Consiste num programa desenvolvido em Java que se executa habitualmente como um *daemon*.

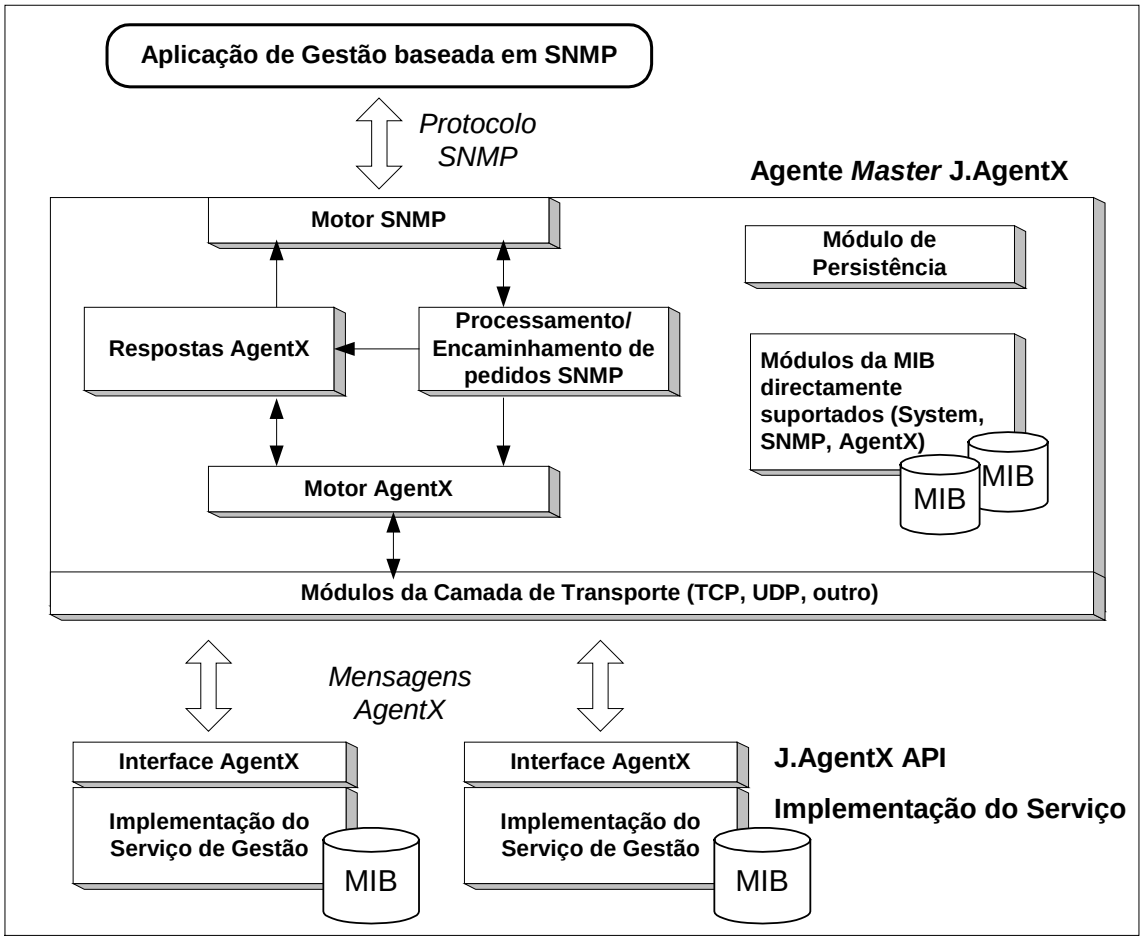

Figura 2 - Arquitectura do `J.AgentX`

Ainda que segundo a especificação do AgentX o agente *master* não deva implementar nenhum subcomponente da MIB, existe alguma informação de gestão que apenas a ele diz respeito. Assim sendo, o *master* implementa directamente os módulos *System* e SNMP da *standard* MIB-II [15] e também a própria MIB do AgentX. Esta solução não compromete a sua compacidade, portabilidade ou interoperabilidade e é bastante mais simples que manter essa informação em subagentes externos.

Outra consideração importante no agente *master* é o suporte simultâneo de vários canais de comunicação, tais como UDP e TCP. Esta solução possibilita fornecer serviços AgentX em ambientes com requisitos de canais de comunicação especiais – tal como código móvel – sem perda de interoperabilidade.

Motor SNMP

Este módulo tem por missão ficar à escuta de pedidos SNMP vindos do exterior, proceder ao seu encaminhamento para o módulo de processamento, e devolver os resultados dos pedidos SNMP.

Processamento/Encaminhamento de Pedidos SNMP

Este módulo é responsável pelo mapeamento de mensagens SNMP em mensagens AgentX. Quando chega um pedido SNMP, este módulo distribui-o de acordo com a localização da informação de gestão pelos diversos subagentes. Se o pedido se refere apenas a objectos contidos

na MIB do agente *master*, a resposta pode ser fornecida imediatamente. Caso contrário, o pedido terá que ser encaminhado para o motor AgentX.

Respostas AgentX

Quando um pedido está associado a um ou vários subagentes, é necessário, posteriormente, recolher as respostas. Neste módulo são processadas operações como a verificação de *timeouts*, de modo a eliminar subagentes que tenham *timeouts* sucessivos; e validação de respostas. Esta última operação filtra as respostas dos subagentes para averiguar se estão lexicograficamente ordenadas na MIB ou se são válidas para o pedido inicial. Se não forem, o pedido é lançado para outro subagente.

Motor AgentX

A missão deste módulo é bastante semelhante à do motor SNMP no que se refere aos pedidos de protocolo. Por outro lado, tem a tarefa adicional de efectuar a administração do AgentX: operações de abertura de ligações, registo de objectos na MIB, etc. Este módulo pode comunicar com os subagentes tanto por TCP como por UDP, e está preparado para futuras implementações usando JavaSpaces para comunicação entre agentes móveis.

Módulo de Persistência

Este módulo foi elaborado devido à necessidade de introduzir persistência. Muitas vezes, é impossível prever todos os estados possíveis a que o sistema pode chegar aquando da comunicação com os subagentes ou com as entidades de gestão. Apesar de a sua implementação ser simples, traduz-se em trabalho bastante útil. Basicamente, quando ocorre uma excepção em qualquer dos outros módulos, este detecta que a respectiva *thread* parou e reinicializa-a com o anterior estado previamente guardado.

3.2. O API do `J.AgentX`

O API do `J.AgentX` consiste num conjunto de classes que facilitam o desenvolvimento de subagentes AgentX, fornecendo ao programador um interface de alto nível para o protocolo. Uma descrição mais detalhada deste interface está disponível em [6], assim como alguns exemplos elucidativos.

Este API está estruturado em quatro *packages.* São elas:

- agentx.syntax – contém a sintaxe usada no protocolo AgentX;
- agentx.pdu – contém as unidades protocolares do AgentX;
- agentx.subagent – contém métodos de alto nível para o desnvolvimento de subagentes;
- agentx.util – contém métodos para gerar a MIB que os subagentes irão suportar;

A Figura 3 mostra a estrutura geral do API. As setas representam as dependências entre diferentes *packages*. A *package* **`agentx.pdu`** usa classes da **`agentx.syntax`** de modo a construir as PDUs do AgentX. A *package* **`agentx.util`** faz uso também da **`agentx.syntax`** para construir os objectos da MIB. A *package* **`agentx.subagent`** usa todas as outras, já que são todas necessárias de modo a fornecer os métodos de alto nível.

Neste momento, o API do `J.AgentX` inclui:

- Ligação/Conclusão de Ligação com o agente *master*;
- Registo/Desregisto de objectos da MIB;
- Registo/Desregisto de gamas de objectos da MIB;
- Atribuição de prioridades a objectos registados;
- Associação de métodos aquando da chegada de *gets* ou *sets*;
- Associação de limites (fronteiras) a objectos da MIB de cariz numérico;
- Definição da MIB autonomamente.

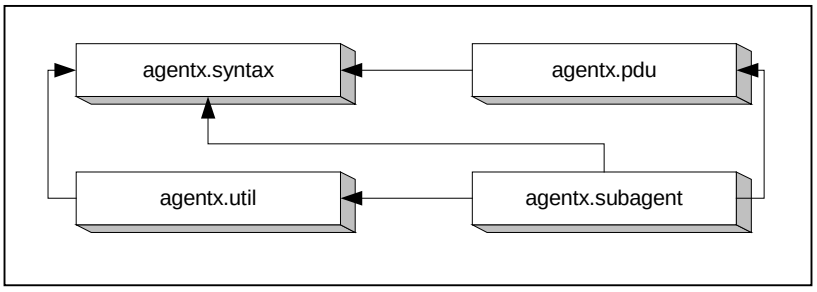

Figura 3 – Estrutura Geral do API do J.AgentX.

O desenvolvimento do API foi também ele codificado em Java. O funcionamento de *threads* e do protocolo são transparentes para o programador e toda a sintaxe e unidades protocolares do AgentX foram implementadas. A não ser que estejamos a programar subagentes complexos, não há necessidade de ter um conhecimento amplo de programação em Java, já que o API foi desenhado para um uso bastante intuitivo.

Em seguida, e para o ilustrar, serão apresentados com algum detalhe os procedimentos básicos no desenvolvimento de um subagente usando o API do `J.AgentX`.

Exemplos do API do J.AGENTX

Este exemplo genérico pretende ilustrar o funcionamento geral do API do `J.AgentX` no que diz respeito ao desenvolvimento de Subagentes. Começa por iniciar uma sessão com o agente *master* e em seguida regista o conjunto de objectos da MIB pelos quais ele é responsável. A leitura e escrita do único objecto registado é feita através de métodos definidos pelo programador.

```
/* Classe SubagenteContador
 * Este Subagente implementa um objecto que serve como contador do número
 * de vezes que é lido. É possível modificar o valor do contador através
 * de operações de Set do SNMP */

import agentx.subagent.*;
import agentx.syntax.*;
import agentx.pdu.AgentXProtocolException;

public class SubagenteContador{
// A variável que serve como contador
  private static int contador = 0;

// Método invocado a este objecto quando uma operação SNMP de Get é feita
  public static int getContador(){
    return ++ contador;   }

// Método invocado a este objecto quando uma operação SNMP de Set é feita
  public static void setContador(Integer novo_valor){
    contador = novo_valor.intValue();  }

  public static void main(String[] args){
// Instancia o Subagente
    AgentX_SubAgent subagente = new AgentX_SubAgent();

// Declara o objecto "Varbind"
    AgentX_Varbind vb;

// Declara a variável "booleana" para depois conter o resultado do
// início da sessão com o Agente Master
```

```
   boolean sessao_comecada;

// Inicia uma sessão com o Agente Master na máquina local e no port 705.
// "Timeout" para as respostas é 3 segundos. A Descrição do Subagente é
// "Subagente Contador" e serão mostradas mensagens consoante os
// acontecimentos no Subagente
   sessao_comecada = subagente.startSession("localhost", 705, 3, "Subagente  Contador", true);

// Se a sessão se iniciou ...
   if (sessao_comecada){

// Regista a sub-árvore da "MIB" "enterprises.uc-pt.11.2", para a
// comunidade "public" e com prioridade 127. A Prioridade vai de 1 a 255,
// a maior é 1, neste caso será 127
     subagente.register("1.3.6.1.4.1.1331.11.1", "public", 127);

// Depois de registar o "object identifier" o subagente é responsável por
// responder aos pedidos direccionados para as seguintes "Varbinds"
     try{
// Cria um "object identifier" em enterprises.uc-pt.11.2.1.0 e associa-o à
// variável contador
       vb = new AgentX_Varbind(AgentX_Varbind.INTEGER, "1.3.6.1.4.1.1331.11.1.1.0", contador);

// Adiciona o "Varbind" à "MIB"
       subagent.addVarbind(vb);

// Associa o método para ser executado quando uma operação de Get ou Get-Next
for feita
       subagente.assignMethodGetToVarbind(vb, false, "SubagenteContador", "getContador", new Class[]{}, new
Object[]{}, null);

// Associa o método para ser executado quando uma operação de Set for feita
       subagente.assignMethodSetToVarbind(vb, "SubagenteContador", "setContador", new Class[]
{java.lang.Integer.class}, null);

// Apanha a excepção se o "object identifier" não for válido
     } catch (AgentXProtocolException e){
       System.out.println("Erro: " + e);
     }
   }
  }
}
```

4. APLICAÇÕES DO J.AGENTX

Tradicionalmente, os mecanismos de extensão de agentes são usados para desenvolver agentes SNMP flexíveis que suportem a inserção dinâmica de serviços de gestão. Este tipo de aplicação que fragmenta a funcionalidade através de vários subagentes é, naturalmente, o primeiro campo de aplicações para esta tecnologia.

No entanto, a extensão de agentes pode ter também um papel importante na introdução de topologias de gestão mais flexíveis e distribuídas. Nesta secção, serão apresentadas duas experiências nas quais o `J.AgentX` é usado de forma pouco convencional. Na primeira experiência a funcionalidade de gestão foi inserida em agentes móveis, resultando em serviços que modificaram dinamicamente a sua localização de acordo com as circunstâncias. O `J.AgentX` foi assim usado no sentido de fornecer um interface SNMP para esses serviços

móveis. Na segunda experiência, usou-se o `J.AgentX` para desenvolver *proxies* no intuito de integrar serviços de gestão locais já existentes e baseados em SNMP.

4.1 Serviços SNMP com Capacidade de Mobilidade

A tecnologia de agentes móveis é uma das mais recentes e entusiasmantes abordagens em computação distribuída. Um agente móvel (AM) pode ser definido como um pequeno programa capaz de migrar entre diferentes máquinas durante a sua execução mantendo o seu estado anterior. Com este modelo, as tarefas de processamento podem ser distribuídas de forma dinâmica pela rede e colocadas nos locais mais apropriados, reduzindo, desta forma, o tráfego da rede e aumentando significativamente a escalabilidade e flexibilidade das aplicações.

Na nossa opinião, os AM têm bastante utilidade no estabelecimento de serviços remotos de gestão (constituídos por um ou vários agentes móveis). Aproveitando as características que esta tecnologia nos oferece – como a coordenação inter-agentes com localização transparente, mobilidade, robustez, actualização remota – é possível desenvolver serviços de gestão intermédios mais potentes e flexíveis do que com outros paradigmas de computação distribuída. Contudo, estes novos serviços continuam a necessitar de fornecer interfaces clássicos de gestão, tais como o SNMP.

O projecto JAMES [16] explora a tecnologia de agentes móveis no desenvolvimento de aplicações para a gestão de redes e telecomunicações. Uma componente chave deste projecto refere-se ao desenvolvimento de uma plataforma de AM para suporte de serviços distribuídos de gestão. Neste contexto o `J.AgentX` desempenha um papel importante na integração entre as arquitecturas do JAMES e do SNMP [17].

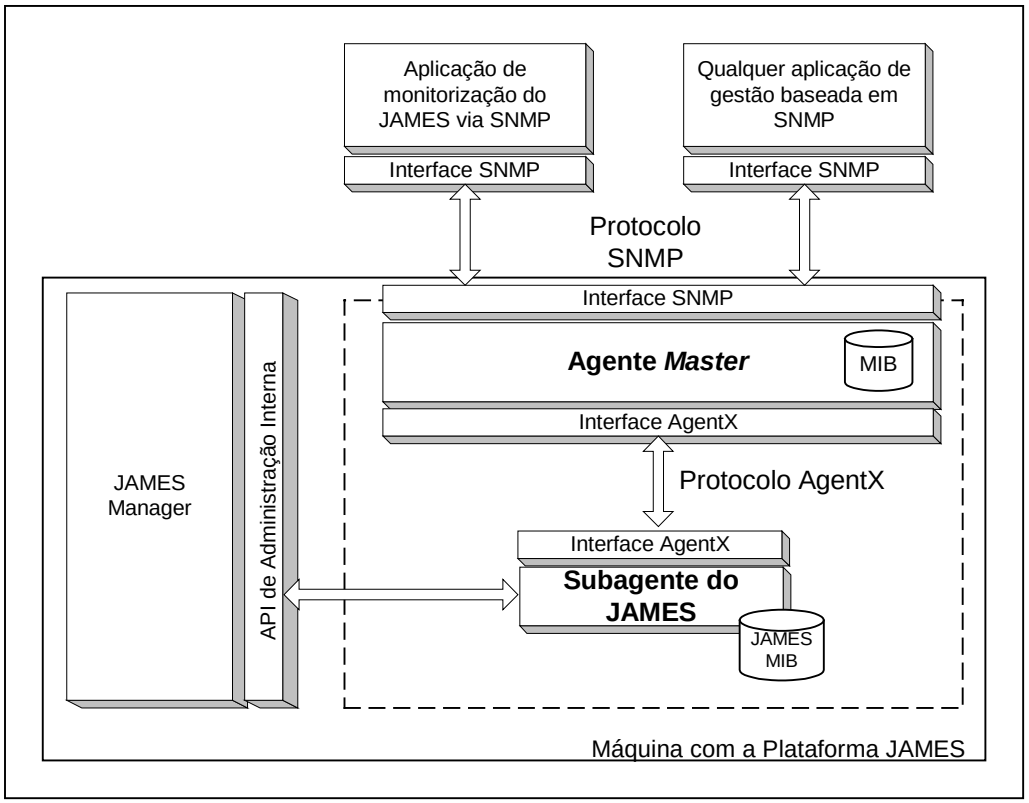

Figura 4 - O `J.AgentX` no contexto do JAMES

Neste projecto um agente *master* `J.AgentX` foi colocado na plataforma de AM para fornecer um *front-end* SNMP às aplicações típicas que tentem aceder à plataforma ou aos serviços distribuídos (desempenhados por AM). Para aumentar a flexibilidade (nomeadamente em instalações *on demand*), o próprio agente *master* consiste num AM que pode ser invocado consoante as necessidades. Contudo, e por razões óbvias, este AM estará num local bem definido em vez de se deslocar pela rede.

Este *front-end* é então usado pelos agentes móveis que implementam os serviços de gestão. Estes agentes (ou conjuntos de agentes coordenados) actuam como subagentes AgentX, não abdicando contudo da sua mobilidade, uma vez que as mensagens AgentX são enviadas usando mecanismos de comunicação inter-agentes com transparência de localização, fornecidos pela plataforma, em vez do uso de TCP. Com esta arquitectura é possível adicionar interfaces legados a serviços de gestão distribuídos baseados em AM.

A vantagem desta solução é a capacidade de fornecer um interface SNMP para a gestão da própria plataforma JAMES. Para tal, foram criados outros subagentes que traduzem operações SNMP (sobre objectos de uma MIB predefinida para a monitorização do JAMES) em operações internas de gestão da plataforma através do seu API interno. A Figura 4 ilustra esta solução (embora na figura seja omitida a ligação aos agentes móveis).

Na Figura 5 é apresentado o interface gráfico de monitorização da plataforma JAMES usando SNMP. Este permite visualizar as características mais relevantes das Agências, dos Agentes e dos Utilizadores da plataforma. É ainda possível modificar o estado dos agentes e das agências. De sublinhar que tanto as operações de recolha de informação da plataforma como as operações de alterações à plataforma são executadas através de operações SNMP, sendo assim possível executar estas operações em qualquer aplicação de gestão SNMP. Obviamente que estas operações são transparentes para o utilizador final que se serve desta aplicação.

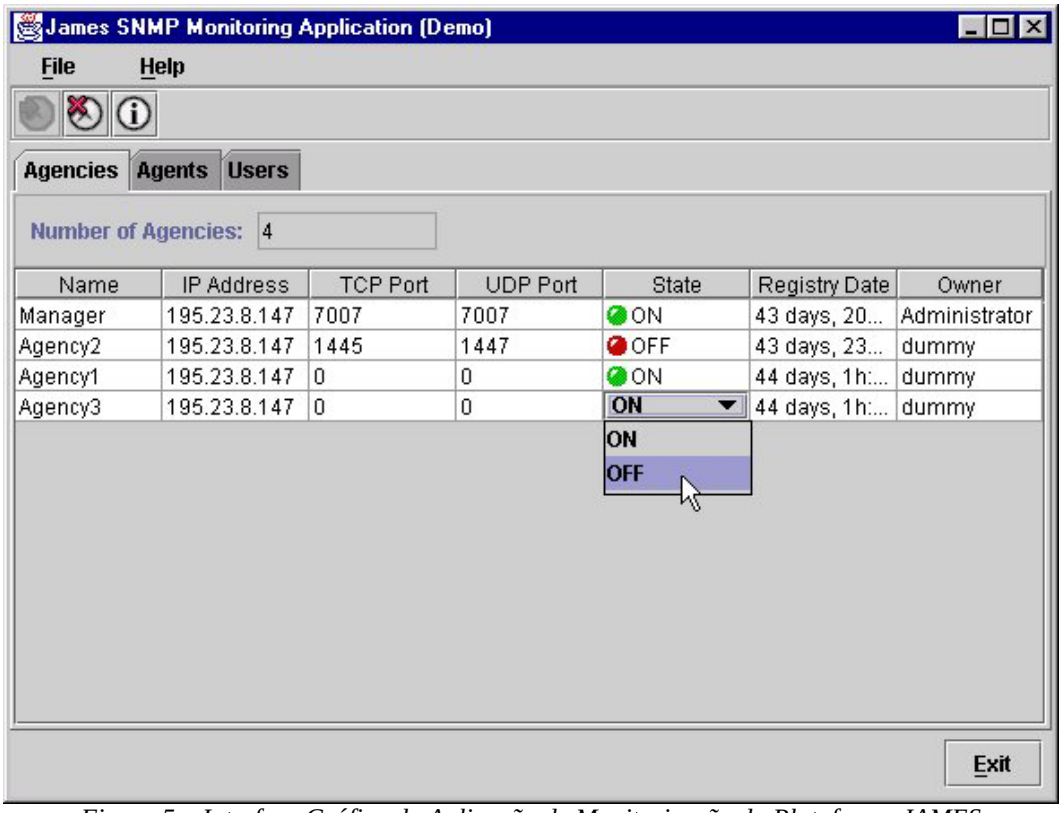

Figura 5 – Interface Gráfico da Aplicação de Monitorização da Plataforma JAMES

Esta experiência demonstra como dois simples ajustamentos em aplicações tradicionais do AgentX (a colocação de um agente *master* num AM e o uso de mecanismos de comunicação com localização transparente) são suficientes para fornecer interfaces para serviços móveis de gestão. Uma descrição mais detalhada sobre a integração da infra-estrutura JAMES com o SNMP, que se encontra fora do âmbito deste artigo, pode ser encontrada em [17-18]. Uma abordagem semelhante, que também usa extensão dinâmica de agentes SNMP para integrar AM na arquitectura do SNMP, é descrita em [19].

4.2 Integração de *Proxies*

Um *proxy* SNMP é um agente que funciona como mediador da comunicação entre aplicações SNMP e serviços de gestão não SNMP. São geralmente agentes SNMP "vazios" (i.e. sem informação de gestão própria) especificamente concebidos para representar um dispositivo legado.

Este conceito de *proxy*, no entanto, não é necessariamente restrito à representação de dispositivos não SNMP. Os serviços de gestão intermédios (isto é, o agente *proxy*) fornecem soluções descentralizadas interessantes mesmo quando estão disponíveis mecanismos de baixo nível baseados em SNMP. Neste contexto, o *proxy* pode agregar e processar antecipadamente dados de baixo nível de uma ou várias entidades monitorizadas no sentido de oferecer funcionalidade de alto nível às aplicações de gestão, optimizando assim o desempenho, o tráfego da rede e a escalabilidade. Esta aplicação menos vulgar do conceito de *proxy* é uma possível solução para distribuir alguma inteligência de gestão através da rede – e mais perto das entidades a gerir – mantendo inalteradas as aplicações de gestão SNMP centralizadas (sistemas legados).

Independentemente da natureza das entidades a gerir (suportando ou não SNMP), com o `J.AgentX` torna-se possível desenvolver *proxies* como subagentes, reduzindo os custos de desenvolvimento – não sendo já necessário incluir um motor SNMP no *proxy* – incrementando a portabilidade e melhorando a integração dos serviços. A Figura 6 apresenta uma possível configuração para um agente SNMP extensível que inclui dois subagentes actuando como *proxies*.

O primeiro é um *proxy* clássico que oferece funcionalidade SNMP em nome de um dispositivo não SNMP. Esta configuração corresponde ao uso clássico de *proxies*.

O segundo *proxy* actua como um serviço intermédio que agrega e pré-processa informação de gestão proveniente de um conjunto de dispositivos SNMP, fornecendo um interface de gestão mais integrada e com maior funcionalidade para aplicações de gestão de topo. Esta abordagem *ad-hoc* para descentralização da gestão permite distribuir alguma funcionalidade de gestão pela rede sem obrigar a ajustes nos agentes ou nas aplicações da arquitectura SNMP.

O terceiro subagente da Figura 6 não se encontra relacionado com a tecnologia *proxy*, e é apresentado apenas para ilustrar como os *proxies* podem coexistir com outros subagentes no sentido de fornecer um interface de gestão mais integrado.

5. CONCLUSÕES

Neste artigo, descreveu-se o `J.AgentX`, a primeira implementação do protocolo AgentX desenvolvida em Java. Ao contrário de anteriores implementações o `J.AgentX` é um pequeno produto focado integralmente na extensibilidade de agentes, não sendo apenas um suplemento para um pacote SNMP genérico.

Ao usar mecanismos de extensão de agentes torna-se possível desenvolver serviços mais flexíveis, com uma clara separação entre funcionamento protocolar (assegurado pelos agentes *master*) e implementação de funcionalidade de gestão (fornecida por subagentes especializados). A portabilidade e integração aperfeiçoada são outras vantagens desta tecnologia.

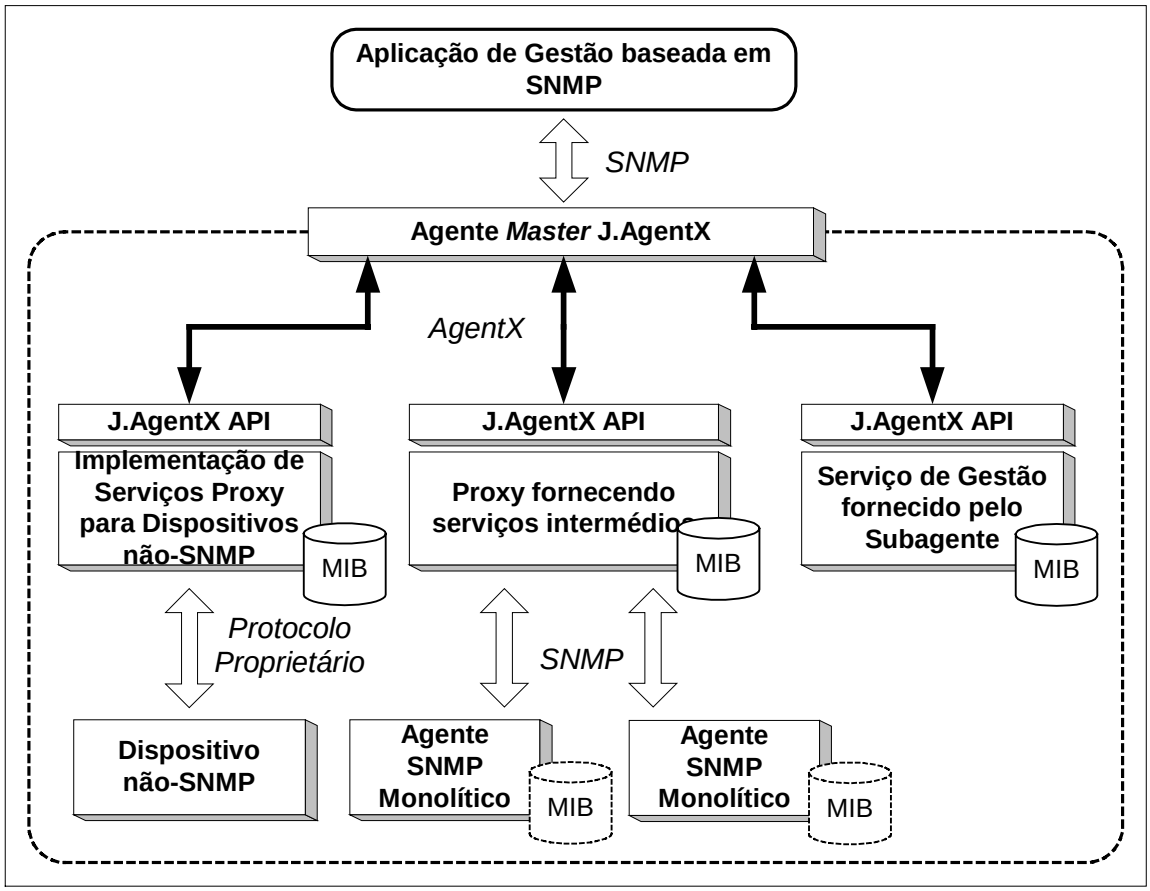

Figura 6 - Os Agentes Proxy na Arquitectura do J.AgentX

Para além deste modelo de utilização mais comum, a extensão de agentes pode ser usada para alguma descentralização *ad-hoc* no âmbito da arquitectura SNMP [18-20]. Neste artigo apresentaram-se dois exemplos demonstrativos.

No primeiro, os agentes extensíveis foram usados para interagir com uma infra-estrutura de agentes móveis. Os agentes móveis permitem introduzir serviços de gestão intermédios inovadores, enquanto a extensão de agentes actua como uma ponte entre estes serviços distribuídos e as aplicações de gestão legadas baseadas em SNMP.

No segundo exemplo o conceito de *proxy* SNMP foi inserido na infra-estrutura `J.AgentX` e estendido por forma a conter gestores intermédios que autonomamente cuidam da gestão de baixo nível de dispositivos SNMP. Este tipo de delegação *ad-hoc* proporciona melhorias de desempenho, flexibilidade e escalabilidade sem obrigar a modificações nos dispositivos geridos ou nas aplicações de gestão.

Estas duas aplicações não têm como objectivo demonstrar que o SNMP é adequado à delegação de gestão – o SNMP não o é, definitivamente – mas sim demonstrar que se pode introduzir alguma flexibilidade no seu uso, quando não é possível substitui-lo.

AGRADECIMENTOS

Agradecimentos ao Rodrigo Reis pelo desenvolvimento do *stack* SNMP usado no `J.AgentX`. A aplicação descrita na Subsecção 4.1 foi desenvolvida no contexto do projecto JAMES, parcialmente financiado pela Agência de Inovação (ADI) e aceite no programa europeu Eureka (Σ!1921).

REFERÊNCIAS

[1] M. Rose, "The Simple Book - An Introduction to Management of TCP/IP-based Internets, 2nd Edition", Prentice-Hall International Inc., 1994
[2] OMG, "The Common Object Request Broker Architecture and Specification", 1995
[3] J. P. Thompson, "Web-based Enterprise Management Architecture", IEEE Communications Magazine, 1998
[4] Hewllet-Packard Openview, http://www.openview.hp.com
[5] Tivoli, http://www.tivoli.com
[6] University of Coimbra, The J.AgentX Package, http://www.dei.uc.pt/agentx
[7] M. Daniele, B. Wijnen, D. Francisco, "Agent Extensibility (AgentX) Protocol Version 1", RFC 2257, 1998
[8] M. Rose, "SNMP MUX Protocol and MIB", RFC 1227, 1991
[9] B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters, "Simple Network Management Protocol Distributed Protocol Interface Version 2.0", RFC 1592, 1994
[10] L. Heintz, S. Gudur, M. Ellison, "Definitions of Managed Objects for Extensible SNMP Agents", RFC 2742, 2000
[11] The Carnegie Mellon University AgentX Implementation, http://www.net.cmu.edu/groups/netdev/agentx
[12] University of California-Davis, UCD-SNMP, http://ucd-snmp.ucdavis.edu/
[13] The Compaq Tru64 UNIX Product, http://www.tru64unix.compaq.com/
[14] Wind River Networking Products - Envoy SNMP Agent, http://www.wrs.com/products/html/envoy.html/
[15] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Manager Information Base for version 2 of the Simple Network Management Protocol", RFC 1450, 1993
[16] L. Silva, P. Simoes, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida, N. Stohr, "JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks", in Proceedings of IATA'99 – Third International Workshop on Intelligent Agents for Telecommunication Applications, Stockholm, 1999
[17] P. Simões, L. Silva, F. Boavida, "Integrating SNMP into a Mobile Agent Infrastructure", in Proceedings of DSOM'99 – Tenth IEEE/IFIP International Workshop on Distributed Systems: Operations and Maintenance, Zurich, 1999
[18] P. Simões, R. Reis, L. Silva, F. Boavida, "Enabling Mobile Agent Technology for Legacy Network Management Frameworks", Proceedings of SoftCOM'99 International Conference on Software in Telecommunications and Computer Networks, Split, 1999
[19] B. Pagurek, Y. Wang, T. White, "Integration of Mobile Agents with SNMP: How and Why", Proceedings of NOMS'2000 – IEEE/IFIP Network Operations and Management Symposium, Honolulu, 2000
[20] J. Quittek, C. Kappler, "Practical Experiences with Script MIB Applications", The Simple Times, Vol. 7, Number 2, 1999