Projeto e implementação de uma aplicação de gerenciamento de redes baseada em agentes móveis

UFRJ

Projeto submetido para a obtenção do título de

Engenheiro Eletrônico

ao Departamento de Eletrônica

da Escola de Engenharia da UFRJ

por

Pedro Braconnot Velloso

Projeto e Implementação de uma Aplicação de Gerenciamento de Redes Baseada em Agentes Móveis

Autor:

Pedro Braconnot Velloso

Orientadores:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Marcelo Gonçalves Rubinstein, D.Sc.

Examinadores:

Prof. Sérgio Lima Netto, Ph.D.

Prof. Luiz Wagner Pereira Biscainho, D.Sc.

Departamento de Eletrônica - Escola de Engenharia - UFRJ

Rio de Janeiro, RJ - Brasil

Setembro de 2001

Agradecimentos

À minha família, principalmente meus pais por todo o amor, orientação e apoio ao longo da minha vida.

A toda a equipe do GTA, em particular aos professores Otto e Rubi, por toda a orientação e conselhos, Leão, pela ajuda no aprendizado da linguagem Java e ao meu colega do mestrado Eric, pelas inúmeras contribuições para o desenvolvimento deste trabalho.

Aos professores do Departamento de Eletrônica, principalmente aos professores Luiz Wagner, por todos os conhecimentos adquiridos e por estar sempre disponível para dar conselhos e ajudar a superar as dificuldades enfrentadas, e Sergio Lima Netto pela presença na banca examinadora e contribuição neste trabalho.

Resumo do Projeto Final apresentado ao DEL/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletrônico.

Projeto e implementação de uma aplicação de gerenciamento de redes baseada em agentes móveis

Pedro Braconnot Velloso

Outubro de 2001

Orientadores:	Otto Carlos Muniz Bandeira Duarte
	Marcelo Gonçalves Rubinstein
Departamento:	Engenharia Eletrônica

O gerenciamento de redes de computadores é uma tarefa que está se tornando cada vez mais complexa devido ao grande crescimento da Internet. Os sistemas de gerenciamento convencionais se baseam no modelo de gerenciamento centralizado do tipo cliente-servidor que tem se mostrado ineficiente para redes com um grande número de elementos gerenciados. O paradigma de agentes móveis permite descentralizar as tarefas de gerenciamento devido à migração do código (agente móvel), que percorre os elementos gerenciados.

Este trabalho propõe e implementa uma aplicação de gerenciamento de redes baseada em agentes móveis com interfaces gráficas amigáveis que permitem ao usuário criar, editar e executar tarefas de gerenciamento. A aplicação possibilita definir as características básicas de gerenciamento, como a operação a ser executada, as variáveis de interesse e o itinerário a ser percorrido, bem como características adicionais que definem como a tarefa deverá ser executada pelos agentes móveis. Foi também desenvolvida uma plataforma simplificada de agentes móveis, na linguagem Java, com característica de mobilidade. Tanto a aplicação quanto a plataforma se mostraram eficientes em testes realizados.

Palavras Chave

Redes de computadores

Internet

Gerenciamento de redes

Agentes móveis

Lista de Acrônimos

ASN.1: *Abstract Syntax Notation One*;

CMIP: *Common Management Information Protocol*;

FIPA: *Foundation for Intelligent Physical Agents*;

ISO: *International Organization for Standardization*;

IP: *Internet Protocol*;

JDK: *Java Development Kit*:

MIB: *Management Information Base*;

PDU: unidade de dados do protocolo (*Protocol Data Unit*);

RMI: *Remote Method Invocation*;

RPC: chamada de procedimento remoto (*Remote Procedure Call*);

SNMP: *Simple Network Management Protocol*;

TCP: *Transfer Control Protocol*;

UDP: *User Datagram Protocol.*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

I.1 Motivação

O gerenciamento de redes de computadores é uma tarefa de grande importância, que tem se tornado cada vez mais complexa, principalmente, devido ao extraordinário crescimento da *Internet*. Por este motivo, surgiu a necessidade de desenvolver ferramentas que automatizassem este processo, possibilitando o gerenciamento de uma maneira mais fácil e eficiente. Com este objetivo, foi desenvolvido o protocolo SNMP (*Simple Network Management Protocol*) [1], que foi projetado para ser simples e possuir capacidade de gerenciar redes heterogêneas incorporando componentes de diferentes fabricantes.

As aplicações de gerenciamento convencionais utilizam o paradigma cliente/servidor, onde uma estação de gerenciamento (cliente) envia pedidos à estação a ser monitorada (servidor). Esta estação executa os pedidos e retorna os resultados para estação de gerenciamento. Assim, para cada pedido, trafegam pelo menos duas mensagens pela rede. O problema deste modelo é sua característica centralizada, que implica uma sobrecarga de processamento na estação de gerenciamento e um enorme gasto da banda passante da rede para a troca de mensagens com os elementos gerenciados.

Uma das propostas para solucionar este problema é a utilização de agentes móveis.

I.1 Motivação

Os agentes móveis são programas capazes de se locomover através de uma rede de computadores, podendo ser executados em diferentes nós da rede. Eles são constituídos de três partes: o código, que é o programa a ser executado; os parâmetros do programa e o estado, que permite ao agente começar a ser executado em um nó da rede, migrar para outro nó e terminar sua execução.

Neste novo paradigma, basta a estação de gerenciamento enviar um agente que percorrerá um determinado itinerário, realizando os pedidos localmente em cada nó da rede. Neste caso, o número de mensagens transmitidas será igual ao número de pedidos mais um. Portanto, à medida que o número de pedidos aumenta, a quantidade de mensagens transmitidas no modelo cliente/servidor é maior que no modelo baseado em agentes móveis. A ineficiência desta característica centralizada torna-se mais aparente à medida que o número de elementos gerenciados aumenta, como mostra a Figura I.1.

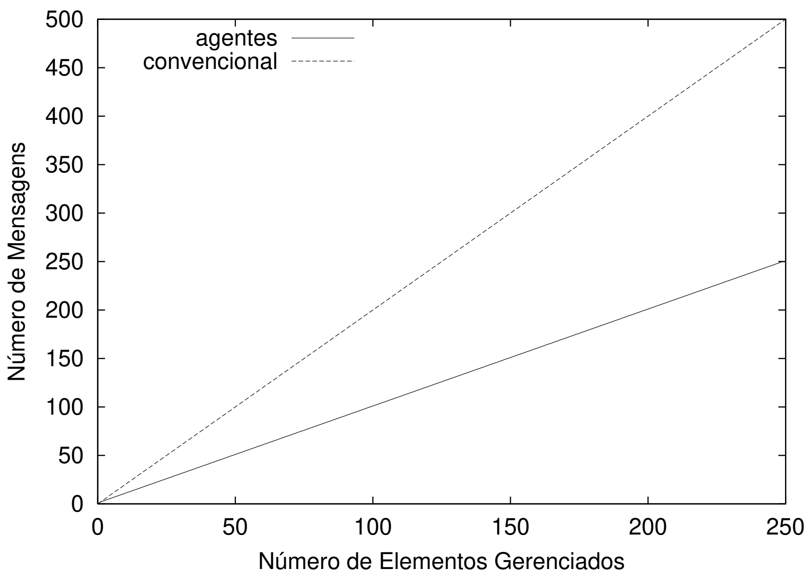

Figura I.1: Modelo convencional *versus* agentes móveis.

I.2 A aplicação

Este trabalho tem como objetivo especificar e implementar uma aplicação para o gerenciamento de redes utilizando o paradigma de agentes móveis, visando a reduzir o tempo gasto e o número de mensagens trocadas na realização desta tarefa.

Uma decisão importante na fase de implementação foi a de não utilizar infra-estruturas de agentes móveis, pois são programas muito “pesados”, que prejudicariam o desempenho do sistema. Desta forma, foi decidido criar um ambiente de mobilidade específico para esta aplicação. Portanto, o projeto foi divido em duas partes: a aplicação de gerenciamento e a plataforma simplificada de agentes móveis.

A aplicação de gerenciamento é composta por três elementos básicos: interfaces gráficas, agentes fixos e agentes móveis. As interfaces gráficas permitem ao usuário definir tarefas e analisar os dados obtidos de forma amigável. Um agente fixo desempenha o papel de um servidor de agentes, sendo responsável por criar e controlar os agentes móveis para viabilizar a execução das tarefas definidas pelo usuário. Outro agente fixo é o agente receptor, que é responsável por receber e tratar os dados coletados pelos agentes, podendo tomar decisões. Os agentes móveis devem recolher e atualizar as informações nos nós da rede, enviando o resultado para o agente receptor.

A plataforma de agentes móveis é o ambiente de mobilidade que permite a execução, a migração e a troca de mensagens entre os agentes. A parte da comunicação dos servidores foi implementada utilizando dois mecanismos de comunicação, *socket* e RMI (*Remote Method Invocation*), com o objetivo de comparar os dois desempenhos.

Este projeto foi implementado na linguagem de programação Java devido à portabilidade e modularidade da linguagem. Desta forma esta será uma aplicação multiplataforma, podendo ser executada em qualquer sistema operacional.

Este trabalho está organizado da seguinte forma. No Capítulo II são apresentados os conceitos relacionados ao gerenciamento. No Capítulo III são definidos o conceito de agente e suas características. No Capítulo IV são descritos o modelo de gerenciamento e a arquitetura da aplicação. No Capítulo V são apresentados detalhes referentes à implementação. O Capítulo VI mostra os resultados dos testes realizados com a aplicação. Por fim, no Capítulo VII são apresentadas as conclusões deste trabalho.

Capítulo II

O gerenciamento de redes

O principal objetivo da gerência de redes é garantir a melhor utilização dos recursos fisicamente distribuídos na rede e mantê-los sempre em operação, aumentando, assim, a confiabilidade da rede. Desta maneira, um sistema de gerenciamento deve ser capaz de monitorar os recursos, prover segurança, detectar falhas e, se possível, corrigi-las da maneira mais eficiente possível.

De acordo com o modelo de gerenciamento definido pela ISO (*International Organization for Standardization*), o gerenciamento é classificado em cinco áreas funcionais [2]:

- Configuração: esta área se preocupa em monitorar e controlar as configurações da rede de modo a mantê-la sempre em funcionamento;
- Falhas: tem por objetivo monitorar os elementos da rede a fim de detectar, isolar e corrigir falhas;
- Desempenho: busca monitorar a taxa de utilização da rede e a taxa de errosi, para garantir eficiência na utilização da rede e impedir congestionamentos;
- Segurança: tenta proteger os recursos da rede através de políticas de acesso e técnicas de autenticação, fornecendo a segurança necessária para o bom funcionamento da rede;

- Contabilidade: tem a função de coletar e processar dados referentes à utilização dos recursos pelos usuários, de maneira a viabilizar a verificação de quotas, a definição de custos e a taxação dos usuários.

A gerência de uma rede consiste basicamente em coletar informações sobre a rede, analisá-las e tomar providências, sendo que essas providências podem ser preventivas (gerenciamento preventivo) ou corretivas (gerenciamento reativo). Com o crescimento da *Internet*, a tarefa de gerenciamento foi se tornando cada vez mais complexa, aumentando muito o volume de informação utilizado. Desta maneira, surgiu a necessidade de criar ferramentas para auxiliar e automatizar o trabalho de gerenciamento.

A maioria das ferramentas propostas para auxiliar o gerenciamento de redes utilizam o protocolo SNMP (*Simple Network Management Protocol*), que foi projetado para ser simples e possuir capacidade de gerenciar redes heterogêneas, que incorporam componentes de diferentes fabricantes.

A tarefa de gerenciamento é baseada em três elementos: as estações de gerenciamento, os agentes e os elementos da rede, como mostra a Figura II.1. As estações de gerenciamento possuem gerentes, que são responsáveis por monitorar e controlar os elementos da rede. Os agentes são entidades autônomas capazes de prestar um serviço a outras entidades. Eles são responsáveis pela coleta de informações dos objetos gerenciados e pela execução de comandos nos elementos da rede. O gerente se comunica com os agentes através de um protocolo simples, requisitando informações ou enviando comandos. As informações relevantes ao gerenciamento, tanto nas estações de gerenciamento como nos elementos da rede, são armazenadas em uma MIB (*Management Information Base*).

A MIB é uma base de dados com formato definido pela SMI (*Structure of Management Information*), descrita na linguagem ASN.1 (*Abstract Syntax Notation One*). Os objetos armazenados na MIB, também chamados de variáveis, são predefinidos dentro de uma estrutura de árvore (Figura II.2) onde cada fabricante pode especificar novas variáveis para os seus equipamentos, além dos objetos padrões. Para referenciar uma variável

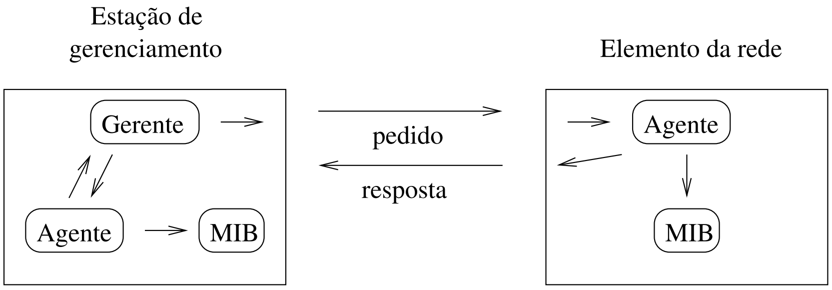

Figura II.1: Modelo de gerenciamento.

basta especificar a posição dela na árvore, podendo ser utilizado o nome das variáveis ou apenas o número correspondente, como na Tabela II.1.

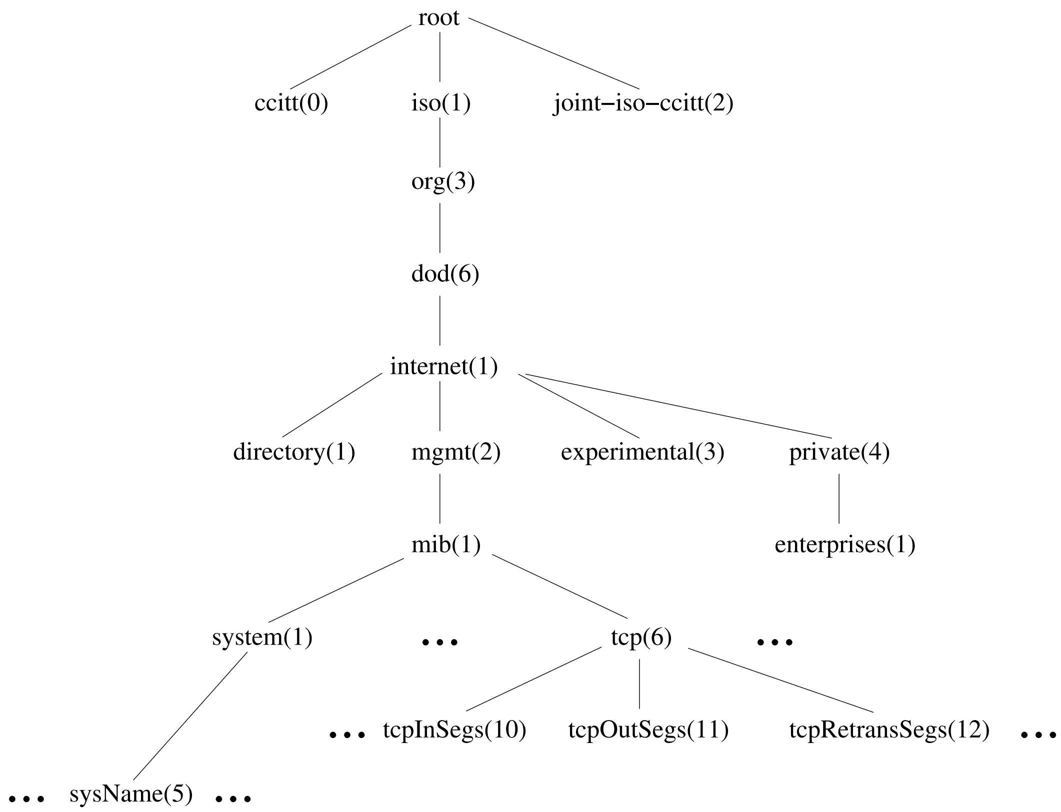

Figura II.2: Estrutura da MIB.

Tabela II.1: Variáveis da MIB-II

Nome *	Número	Descrição
system.sysName	.1.3.6.1.2.1.1.5.0	nome da máquina
tcp.tcpInSegs.0	.1.3.6.1.2.1.6.10.0	quantidade de segmentos recebidos
tcp.tcpOutSegs.0	.1.3.6.1.2.1.6.11.0	quantidade de segmentos transmitidos
tcp.tcpRetransSegs.0	.1.3.6.1.2.1.6.12.0	quantidade de segmentos retransmitidos

* todos estes nomes são precedidos por iso.org.dod.internet.mgmt.mib-2.

II.1 O protocolo SNMP

O SNMP foi projetado para viabilizar a comunicação entre gerentes e agentes, obtendo grande sucesso principalmente pela sua simplicidade. A sua primeira versão (SNMPv1) [3] é constituída por cinco primitivas:

- GetRequest: utilizada para coletar o valor de alguma variável na MIB;
- GetResponse: utilizada para enviar a resposta de um pedido;
- GetNextRequest: coleta a próxima variável da MIB, normalmente é utilizada para varrer uma parte seqüencial da MIB;
- SetRequest: atualiza uma variável da MIB;
- Trap: utilizada pelo agente para alertar da ocorrência de algum evento assíncrono. É a única mensagem que não necessita de resposta.

Utilizando-se do paradigma cliente/servidor, o gerente realiza um pedido para o agente através de uma mensagem (Figura II.3) que contém um campo para especificar a versão do protocolo, um campo especificando a comunidade e a PDU (*Protocol Data Unit*), que conterá o tipo de serviço requisitado (coleta ou atualização de variáveis) e o nome da variável. O agente responderá com outra PDU contendo a resposta.

Versão	Comunidade	PDU SNMP

Figura II.3: Formato da mensagem SNMP.

Nesta primeira versão, a parte de segurança é fornecida através do uso de comunidades, consistindo em uma senha armazenada no campo Comunidade (*CommunityString*) sem nenhuma proteção, uma vez que a senha pode ser lida e alterada por outro usuário. A segurança é a principal falha apresentada por este protocolo.

Na versão SNMPv2 [3], foram acrescentadas a primitiva GetBulkRequest que permite ao gerente fazer pedidos de leitura de uma parte específica da MIB e a InformRequest/Response, que é a implementação do Trap confirmado, onde a InformRequest é o próprio Trap e o InformResponse é a confirmação. Com o objetivo de oferecer maior segurança, foi criado o conceito de *Party*, que foi definido como um contexto de execução virtual nas entidades SNMP, onde a operação do protocolo restringe-se a um subconjunto de operações possíveis. A *Party* armazena informações de tempo (*clocks*), chaves de criptografia e políticas de acesso. O principal problema desta versão foi a incompatibilidade com a versão anterior, visto que as mensagens criptografadas não eram entendidas pelo SNMPv1. Por isso, foi lançada a versão SNMPv2c que voltou a ser baseada em comunidades, como na primeira versão. A versão SNMPv3 [3], basicamente, incorporou um conjunto de funcionalidades para prover a segurança e foi desenvolvida para ser compatível com as versões anteriores.

O grande problema do uso do SNMP deve-se ao fato de os sistemas de gerenciamentos de redes mais complexos necessitarem de uma volumosa troca de informações entre o gerente e o agente e, como conseqüência, de uma grande quantidade de mensagens enviadas pela rede.

II.2 O protocolo CMIP

O CMIP (*Common Management Information Protocol*) foi projetado para ser construído a partir do SNMP, compensando suas restrições/limitações e tornando-se, assim, um maior e mais detalhado gerente de rede. Seu projeto básico é similar ao SNMP, que utiliza as PDU's como variáveis para monitorar uma rede. O CMIP, entretanto, contém onze tipos de PDU's (comparados aos cinco do SNMP).

A grande vantagem do protocolo CMIP é o fato de suas variáveis não somente carregarem informação de um terminal para outro (como no SNMP), mas também poderem ser usadas para executar tarefas que seriam impossíveis com o SNMP [4]. Por exemplo, se um terminal em uma rede não puder alcançar seu servidor de arquivos em um predeterminado intervalo de tempo, o CMIP pode, então, notificar o usuário relacionado ao evento. No caso do SNMP, um usuário teria que manter-se explicitamente a par do número de tentativas malsucedidas de alcançar um servidor de arquivos em que um terminal incorresse. O CMIP resulta, assim, em um mais eficiente sistema de gerenciamento de redes, já que menos trabalho é requerido do usuário para manter-se atualizado sobre o estado da rede.

Outra vantagem do CMIP é a questão da segurança, visto que ele possui dispositivos de segurança que suportam a autorização, o controle de acesso e os registros de segurança.

A principal desvantagem do CMIP é o grande consumo de recursos para realizar o gerenciamento. Outra desvantagem é a grande complexidade do protocolo, principalmente, devido a serem as variáveis estruturas de dados muito complexas e sofisticadas, com muitos atributos. Além disso, o atraso no seu desenvolvimento permitiu que o SNMP conquistasse grande parte do mercado. Por isso, o CMIP não conseguiu substituir o SNMP como era esperado, pois, além de ser mais simples, o SNMP já possuía uma grande base instalada.

Capítulo III

Os agentes

O significado da palavra agente é muito amplo, pois ela é empregada em diversas áreas. Desta forma, são encontradas diferentes definições, inclusive em uma mesma área de conhecimento. No entanto, percebem-se algumas semelhanças nas diversas abordagens existentes na área da computação. Sendo assim, usando uma definição simples e que seja mais conveniente para a área de redes de computadores, diz-se que um agente é um programa capaz de realizar tarefas, de forma autônoma, em nome de um usuário ou de outro agente.

Os agentes possuem diversas propriedades e, normalmente, cada uma delas constitui uma área de pesquisa diferente. As principais propriedades [5] são:

- Autonomia: é o grau de autoridade do agente, medida através de sua interatividade. Existem três tipos de interatividade: somente com o usuário, com a aplicação e com o usuário e, por último, com o usuário, com a aplicação e com outro agente;

- Inteligência: a inteligência é o grau de raciocínio e comportamento aprendido. Existem quatro níveis de inteligências: o primeiro nível é a preferência, que é o mínimo de inteligência que um agente pode ter. Neste caso são definidas algumas preferências e é criado algum mecanismo que decida sobre estas preferências. Outro nível seria o raciocínio, que consiste em o agente ser capaz de entender o que o usuário quer fazer. O terceiro seria o planejamento, onde o agente seria capaz de planejar os passos para realização de determinada tarefa. O mais alto nível de inteligência seria o aprendizado, em que o agente seria capaz de aprender e se adaptar a novos ambientes;

- Mobilidade: é a capacidade do agente de se mover na rede. Os que não se movem são chamados agentes fixos e os que são capazes de se mover são chamados agentes móveis;
- Comunicação: é a capacidade do agente de trocar informações com outros agentes;
- Colaboração: é a capacidade de um agente trabalhar em cooperação com outros agentes a fim de solucionar um problema ou executar uma tarefa.

Existem outras características atribuídas aos agentes em outros trabalhos, que, na classificação apresentada, já estão englobadas na propriedade de inteligência, como, por exemplo, adaptabilidade, proatividade e aprendizado.

A comunicação e a colaboração são as características que possibitam aos agentes trabalharem em conjunto para resolver um problema comum, divindo o problema em pequenas partes.

III.1 Sistemas multiagentes

Um sistema multiagentes é um sistema constituído por um conjunto de agentes que colaboram para o cumprimento das tarefas que lhes foram designadas. Esses agentes podem ou não pertencer a um mesmo usuário e as funções desempenhadas por esses agentes podem ser diferentes. O grau de colaboração de um agente com outros é definido como a "sociabilidade" deste agente.

As principais aplicações de sistemas multiagentes se encontram em sistemas distribuídos. Um sistema distribuído caracteriza-se por ser composto por módulos que são executados em diferentes unidades de processamento. Existem aplicações em que, além do processamento, o controle também é distribuído, ou seja, o controle da aplicação não depende apenas de um processo. A grande vantagem do processamento distribuído é o aproveitamento do paralelismo entre as unidades de processamento para melhorar o desempenho, enquanto que a vantagem do controle distribuído é a confiabilidade, visto que o sistema não depende de nenhum processo especificamente. Desta forma, a ocorrência de alguma falha em uma das unidades de processamento, não implicará o maufuncionamento do sistema.

A sociabilidade dos agentes implica a divisão da solução do problema em tarefas que serão distribuídas entre os agentes "colaboradores". Este fato possibilita uma maior facilidade de extração do paralelismo de um certo problema. Outro benefício proporcionado pelo sistema distribuído multiagente é a tolerância a falhas, porque a falha de um agente pode ser compensada por outro agente sem prejudicar o sistema. Os agentes devem ser independentes para facilitar a reconfiguração do sistema. Para isto, os agentes devem definir uma interface com a qual eles ofereçam serviços e troquem informações com outros agentes, de maneira que o agente pode ser mudado sem que os outros agentes percebam.

Uma das aplicações em que os sistemas distribuídos multiagentes estão sendo empregado freqüentemente é o gerenciamento de redes. Os elementos da rede são monitorados por agentes que coletam e analisam as informações. Estes sistemas podem ser uma solução para o problema da grande quantidade de informação, principalmente se os agentes possuirem alguma inteligência [6].

III.2 Inteligência artificial

A inteligência é uma das principais características de um agente, por isso existem vários estudos nesta área. Uma das tecnologias empregadas para criar um agente inteligente é a Inteligência Artificial (IA).

A IA pode ser dividida em duas categorias: uma diz respeito ao método de solução de problemas e a outra, à localização espacial. A primeira categoria pode ser subdividida em outras quatro categorias: a IA Simbólica, a IA Conexionista, a IA Evolucionária e a IA Híbrida. A categoria localização espacial pode ser subdividida em outras duas: IA Monolítica e IA Distribuída. É o que mostra a Figura III.1.

A IA Simbólica utiliza regras de inferência como ferramenta básica para manipular o conhecimento. A IA Conexionista utiliza redes neurais artificiais para resolver problemas mal definidos, mas que são conhecidos por exemplos. Como exemplo destes problemas têm-se o reconhecimento de padrões, o raciocínio e a robótica. A IA Evolucionária utiliza mecanismos iguais aos encontrados na evolução biológica. A IA Híbrida mescla mecanismos de várias categorias de IA.

A IA Monolítica é empregada em sistemas simples sem modularidade e a IA Distribuída é empregada em casos onde o sistema possua diversos módulos que operam em conjunto para resolver um determinado problema, como é o caso dos sistemas multi-agentes.

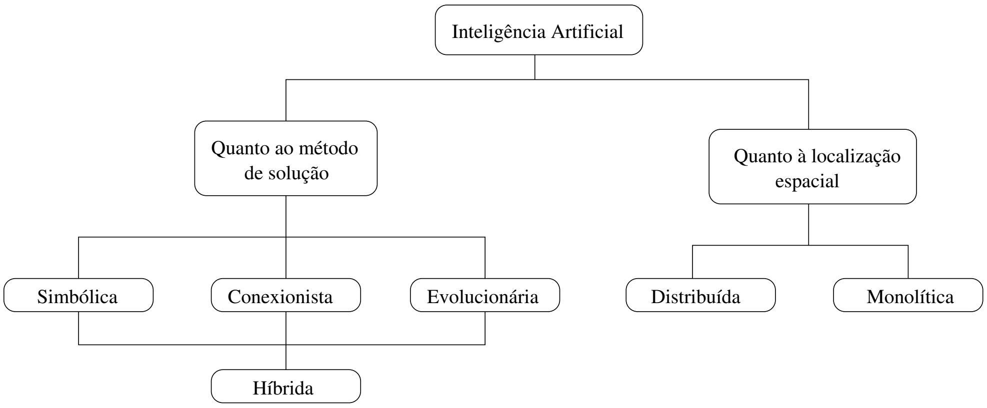

Figura III.1: Classificação da Inteligência Artificial.

No caso de gerenciamento de redes, particularmente, seria interessente desenvolver agentes que, a partir de um conjunto de variáveis, fossem capazes de reconhecer um padrão de comportamento da rede e que pudessem associar estes comportamentos a determinados procedimentos a serem executados. Desta forma, além de o agente poder notificar a ocorrência de congestionamentos, problemas de falhas e ataques, ele poderia tomar medidas para conter ou, até mesmo, para prevenir alguma dessas situações. Isto seria possível utilizando as técnicas de inteligência artificial, como por exemplo, usuando redes neurais.

III.3 Agentes móveis

O conceito de agentes móveis [7] é baseado no paradigma da programação remota (*Remote Programming* - RP), que consiste basicamente em enviar um procedimento para ser executado em outro hospedeiro. A RP surgiu como uma alternativa à chamada de procedimento remoto (*Remote Procedure Call* - RPC).

Os agentes móveis são programas capazes de se locomover através de uma rede de computadores. Eles são constituídos de três partes: o código, que é o programa a ser executado; os parâmetros do programa (identificação do agente, a origem, o destino, o dono etc.); e o estado, que permite ao agente começar a ser a executado em um nó da rede, migrar para outro nó e continuar sua execução.

Para um computador executar tarefas em outra máquina, utilizando o RPC (Figura III.2), é preciso que exista um conjunto de procedimentos, com seus respectivos argumentos e resultados, previamente estabelecidos. Desta forma, para cada pedido de execução é enviada uma resposta com os resultados. Para tarefas simples este método parece bem razoável, mas quanto maior for a complexidade da tarefa, na qual pode ser necessária a execução de "n" procedimentos maior será o número de mensagens trocadas entre os dois computadores.

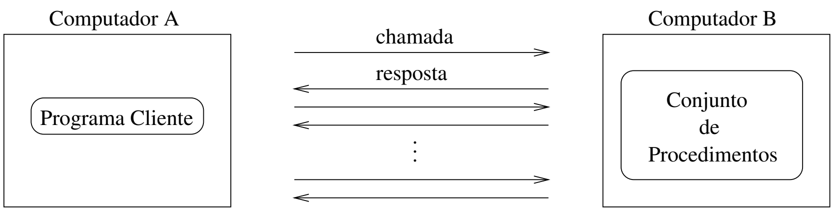

Figura III.2: Realização da tarefa remota utilizando o RPC.

Por outro lado, para executar uma tarefa em outro computador utilizando agentes móveis (Figura III.3), basta enviár o agente para o nó de destino onde executará a tarefa para depois retornar ao nó de origem com o resultado. Por esta razão, a complexidade da tarefa não influi no número de mensagens trocadas entre os computadores.

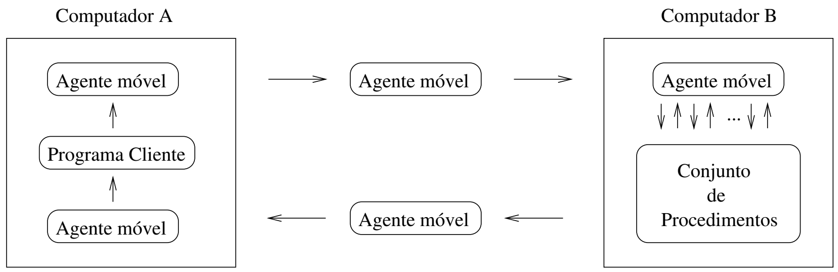

Figura III.3: Realizando uma tarefa remotamente utilizando agentes móveis.

As principais vantagens [8] oferecidas pelos agentes móveis são a flexibilidade e o processamento assíncrono. Em relação à flexibilidade, os procedimentos não precisam ser preestabelecidos, permitindo que os agentes sejam criados independentemente da máquina na qual serão executados. A mobilidade de código pode proporcionar também um processamento assíncrono, ou seja: o código é independente do processo que o gerou, desta forma; a máquina que enviou um agente para realizar uma determinada tarefa em outro computador pode encerrar a conexão, ou até mesmo ser desligada, sem prejudicar a realização da tarefa.

III.3.1 Aplicações

O estudo da tecnologia de agentes móveis é bastante recente e inúmeras pesquisas estão sendo realizadas no sentido de verificar o desempenho de certas aplicações utilizando agentes móveis em diversas áreas, tais como:

- Comércio eletrônico: agentes móveis podem ser usados para procurar os produtos desejados pelo menor preço, podem negociar com outros agentes ou apenas trocar informações sobre um determinado produto ou lugar. Uma pessoa poderia programar seu agente móvel para ele planejar sua viagem, comprando as passagens e fazendo as reservas nos hotéis, agendar encontros com outros amigos ou clientes através de seus respectivos agentes, descobrir dentre as peças que estão passando na *Broadway* qual ela gostaria de ver e comprar um ingresso;

- Segurança de redes: Queiroz [9] desenvolveu e implementou um sistema de detecção de intrusão baseada em agentes autônomos, no qual agentes locais (Sentinelas) são responsáveis por detectar e identificar o tipo da invasão, definindo o melhor agente para resolver o problema. Em seguida, o Sentinela manda um pedido para um agente central (Quartel General - QG) que recebe os pedidos e envia os agentes (Destacamentos) que combaterão o problema;

- Balanceamento de carga: os agentes móveis são utilizados para recolher informações sobre utilização de memória e de processamento de computadores em uma rede heterogênea. Essas informações são utilizadas por um outro programa para otimizar o balanceamento de carga;

- Gerenciamento de redes: devido à grande heterogeneidade das redes de telecomunicações, os agentes móveis são de grande utilidade, pois não dependem do sistema em que serão executados, permitindo recolher informações da rede de maneira mais eficiente. Eles permitem também fazer reconfigurações ou atualizações de programas.

III.3.2 As plataformas de agentes móveis

As plataformas de agentes móveis são ambientes que permitem a construção, o uso e a manutenção de agentes móveis. Os principais serviços [10] que estas plataformas devem prover são:

- Migração: permite a locomoção de um agente de um "lugar" para outro, levando seus atributos e o seu estado;

- Execução: permite a execução dos agentes;

- Comunicação: permite a troca de informações entre agentes ou chamar procedimentos de outros agentes;
- Segurança: cria níveis de permissão para proteger o hospedeiro de ataques, não permite que um agente de um usuário destrua um agente de outro, garante a privacidade do usuário e garante a autenticação para que um agente de um usuário não se passe por um agente de outro;
- Tolerância a falhas: permite que a ocorrência de falhas não prejudique o desempenho do sistema.

As plataformas e os agentes devem ser independentes de qualquer arquitetura ou sistema operacional, por isso, a grande maioria das infra-estruturas disponíveis [11] na *Internet* é implementada em linguagens interpretadas como a TCL/TK e Java. Dentre essas duas linguagens, Java tem se destacado como a mais empregada, principalmente por ser uma linguagem simples que permite grande reaproveitamento de código.

O Telescript foi a primeira infra-estrutura comercial de agentes móveis, e por isso, tornou-se uma das plataformas mais conhecidas, mas não obteve grande sucesso por ser baseada em uma linguagem proprietária [7].

Mesmo as plataformas que foram implementadas em linguagens não interpretadas oferecem suporte a agentes desenvolvidos em outras linguagens, como é o caso da TACOMA [12], da D'Agents [13] (antiga TCL) e da ARA [14].

A plataforma Aglets [15], implementada em Java, é uma das mais utilizadas por ser considerada uma plataforma de uso simples, por possuir vários exemplos disponíveis e por ser muito bem documentada. Seus dois principais problemas eram o código fonte, que não era aberto, e a desistência, por parte da IBM, de dar continuidade ao projeto. Entretanto, estes problemas já foram soluciondos com o lançamento de uma nova versão em junho de 2000 e, posteriormente, com a disponibilização do código fonte.

Existem muitas outras plataformas baseadas em Java que oferecem suporte a sistemas de agentes móveis, disponíveis na *Internet.* Algumas delas são gratuitas e possuem o código disponível, como a Mole [16], a Ajanta [17], a JATLite [18] e a Hive [19], enquanto que outras são pagas, como a JDMK [20] e a Concordia [21], e possuem apenas uma versão limitada disponível gratuitamente.

Um problema que tem sido observado é a falta de padronização destas plataformas, causando incompatibilidade entre elas. Por este motivo, a FIPA (*Foundation for Intelligent Physical Agents*) [22] definiu um conjunto de padrões para uma infra-estrutura de agentes móveis com o objetivo de permitir que um agente possa ser executado em qualquer plataforma. As plataformas FIPA-OS [23] e JADE [24], já obedecem aos padrões da FIPA.

Capítulo IV

O projeto da aplicação e da plataforma

Neste capítulo, é apresentada a arquitetura proposta para a aplicação de gerenciamento de redes baseada em agentes móveis e seus componentes. A aplicação desenvolvida tem como objetivo proporcionar um ambiente gráfico no qual o usuário pode criar tarefas de gerenciamento que serão executadas pelos agentes móveis de forma transparente. Além disso, através da interface gráfica, ele pode definir diversas características da tarefa, inclusive como ela será executada.

Uma plataforma simplificada de agentes móveis, que implementa toda a parte de execução, comunicação e mobilidade dos agentes, foi desenvolvida especificamente para esta aplicação, de forma a torná-la mais eficiente.

IV.1 Gerenciamento baseado em agentes móveis

Como foi descrito no capítulo II, o paradigma cliente/servidor tem-se mostrado pouco eficiente no gerenciamento de redes, com isso, uma das propostas para otimizar o gerenciamento é utilizar agentes móveis para minimizar o tempo de resposta e diminuir o número de mensagens.

A figura IV.1 mostra esse esquema de gerenciamento, proposto em [25], e implementado neste trabalho.

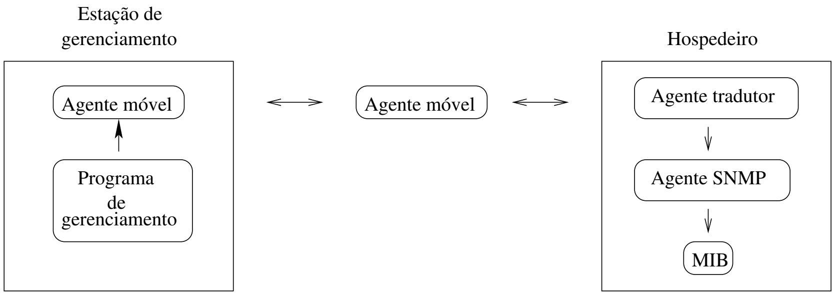

Figura IV.1: Modelo de gerenciamento baseado em agentes móveis.

Os agentes móveis são responsáveis por coletar as informações pedidas pelo gerente em todos os nós da rede. Por questões de segurança, nas infra-estruturas disponíveis, os agente móveis não possuem permissão para acessar todos os recursos disponibilizados por uma máquina, por isso, em cada elemento da rede é colocado um agente fixo, denominado agente tradutor, que desempenhará o papel de uma interface entre o agente móvel e o agente SNMP. A figura IV.2 mostra o fluxograma dos estados de um agente móvel.

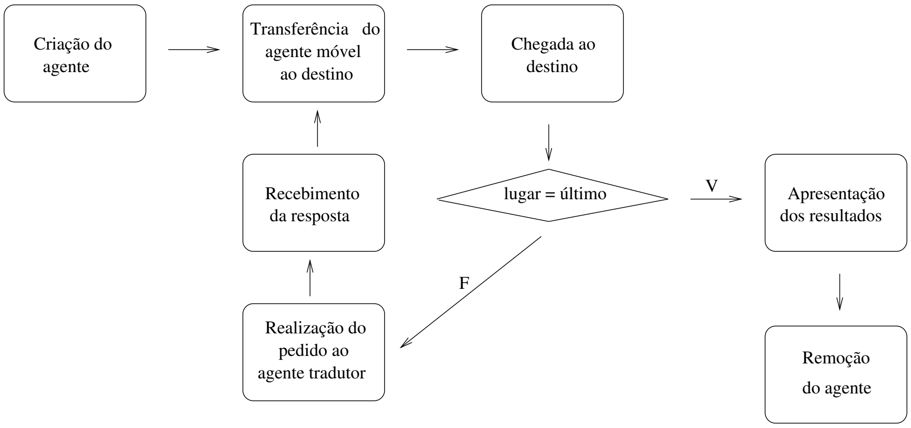

Figura IV.2: Diagrama de estados do agente móvel.

Quando o gerente precisa de alguma informação ou fazer alguma atualização em uma variável da MIB, basta definir um itinerário e enviar um agente móvel para realizar a tarefa. Em cada hospedeiro, o agente móvel passará o pedido para o agente tradutor, que fará o pedido para o agente SNMP, que por sua vez realizará a consulta ou a atualização na MIB e devolverá o resultado para o agente tradutor. O agente móvel receberá o resultado e migrará para o próximo nó, até percorrer todo o seu itinerário.

IV.2 A tarefa de gerenciamento

O que define uma tarefa são suas características. Estas podem ser divididas em duas categorias, básicas e complementares. As características básicas de uma tarefa são aquelas que não podem ser definidas pelo programa, mas apenas pelo usuário e são essenciais para a sua execução. As características básicas são, na verdade, os atributos da tarefa:

- Nome da tarefa;
- Operação a ser executada, podendo ser uma atualização ou uma consulta de uma variável da MIB;
- Variáveis de interesse para a tarefa;
- Itinerário: a relação das máquinas que serão percorridas pelos agentes.

As características complementares apenas adicionam informações à tarefa e, caso não sejam definidas, o programa assume um valor padrão para elas. Estas características são:

- Falha: representa o que o agente deve fazer em caso de falha ao tentar migrar para um determinado nó. Ele pode seguir para o próximo nó (valor padrão) ou retornar à origem, abortando a tarefa;
- Entrega dos resultados: especifica a maneira como os resultados devem ser entregues ao gerente. Os resultados podem ser enviados pelo agente móvel em um nó especificado no itinerário (valor padrão) ou o agente móvel pode retornar ao nó de origem carregando os resultados e entregá-los ao gerente;

- Freqüência: determina a freqüência com que os resultados serão entregues. O valor padrão é um, ou seja, os resultados são entregues a cada nó visitado;

- Medidas de tempo: define as medidas em relação ao tempo que deverão ser tomadas. Pode ser medido o tempo total para a execução de uma tarefa e o tempo de execução de uma operação pelos agentes móveis. O valor padrão é não medir tempo algum;

- Número de agentes: especifica o número de agentes que irão executar a tarefa. O valor padrão é um agente. Quando é especificado mais de um agente para executar a tarefa, o itinerário é dividido de forma que a diferença do número de estações entre os novos itinerários não seja maior que um; isto quer dizer que um agente visitará ou o mesmo número ou, no máximo, uma estação a mais que outros agentes;

- Número de execuções: determina o número de vezes que a tarefa será executada. O valor padrão é uma vez;

- Intervalo: determina o intervalo de tempo, em segundos, entre duas execuções de uma tarefa, caso esta seja executada mais de uma vez. O valor padrão é zero.

IV.3 A arquitetura proposta

A arquitetura da aplicação é dividida em camadas hierárquicas com quatro níveis: usuário, controle, agente e comunicação, como pode ser visto na Figura IV.3. O primeiro nível tem a função de prover a interface com o usuário. O nível de controle envolve tanto o controle das tarefas que estão sendo realizadas quanto o controle dos agentes móveis criados para executar as tarefas. O nível do agente representa os agentes de uma forma geral, incluindo os agentes móveis e os agentes tradutores (fixos). O último nível lida com toda a parte de comunicação da aplicação.

As camadas inferiores disponibilizam métodos para as camadas superiores. Desta maneira, cada camada interage apenas com as camadas adjacentes. A única exceção ocorre quando o servidor recebe mensagens dos agentes, pois ele entrega os resultados

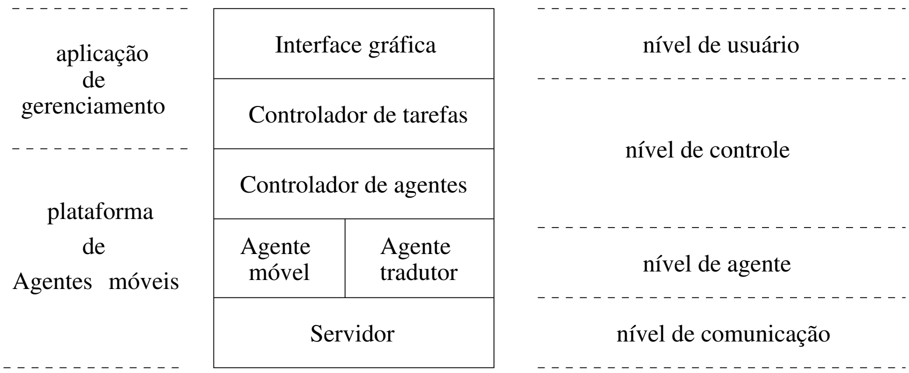

Figura IV.3: Modelo em camadas da aplicação.

diretamente ao controlador de tarefas.

IV.4 Os componentes do sistema

Cada camada da arquitetura proposta é composta por entidades que são responsáveis por prover os serviços prestados às camadas superiores.

- Interface gráfica: a interface gráfica propicia um ambiente amigável no qual o usuário pode criar, editar ou executar uma tarefa de gerenciamento. O usuário também pode visualizar os resultados das operações realizadas pelos agentes;
- Controlador de tarefas: o controlador de tarefas tem a função de criar e executar as tarefas de gerenciamento. Ele também é responsável por escolher que tipo de agente será executado para cada tarefa;
- Controlador de agentes: o controlador de agentes tem a função de executar os agentes e armazenar todas as informações necessárias para controlar todos os agentes em execução em uma mesma máquina;
- Agente móvel: os agentes móveis são responsáveis por executar as operações de gerenciamento em cada nó da rede e entregar os resultados à estação de gerenciamento;

- Agente tradutor: o agente tradutor permite a comunicação do agente móvel com o agente SNMP. Desta forma, ele deve entregar os pedidos dos agentes móveis para os agentes SNMP e, em seguida, retornar o resultado para o agente móvel. O agente móvel poderia se comunicar diretamente com o agente SNMP, mas foi escolhido colocar um agente intermediário para não sobrecarregar o código do agente móvel e por questões de segurança;

- Servidor: o servidor é o responsável por estabelecer a conexão entre as máquinas da rede. Neste trabalho, o servidor foi dividido em dois processos. Um é responsável por enviar e receber as mensagens e o outro por enviar e receber os agentes móveis.

Capítulo V

Implementação

Neste capítulo, serão apresentados os detalhes da implementação, que envolvem a escolha da linguagem, a escolha do mecanismo de comunicação a ser utilizado na plataforma de agentes móveis e as classes do programa.

V.1 Linguagem de programação

Como já foi mencionado anteriormente, Java foi a linguagem escolhida para implementação da aplicação. As principais características que levaram a esta escolha foram a portabilidade, a simplicidade e o alto grau de reaproveitamento de código desta linguagem.

Java é uma linguagem de programação orientada ao objeto com a sintaxe muito similar a da linguagem C++. Na definição de Java houve um grande esforço em diminuir a grande complexidade existente em C++, tornando-a uma linguagem mais simples, porém bastante poderosa [26]. Este foi um dos principais motivos da grande popularidade que Java possui. Outra característica importante é o fato de Java ser uma linguagem interpretada, ou seja, ao se compilar um programa em Java não é gerado um arquivo em código de máquina, mas sim um arquivo em *byte codes*.

A máquina virtual Java (*Java Virtual Machine*) é um interpretador Java responsável por executar os arquivos compilados (*byte codes*). Assim toda máquina que tiver uma máquina virtual instalada será capaz de executar um programa em Java. Portanto, uma de suas principais vantagens é a portabilidade, pois um programa em Java é executado em qualquer máquina virtual. Entretanto, deve-se observar que a portabilidade do programa está diretamente relacionada com a portabilidade da máquina virtual Java.

Outra grande vantagem de Java é o alto grau de reaproveitamento de código, pois, além de ela ser uma linguagem orientada ao objeto, existe uma grande quantidade de classes já prontas e disponíveis na *Internet*.

O maior problema enfrentado pelas aplicações implementadas em Java é o baixo desempenho em relação as linguagens que não são interpretadas, o que torna as aplicações "pesadas".

Neste trabalho foi utilizado o *Java Development Kit*[27] (JSDK 1.3.1) que possui um compilador e uma máquina virtual Java.

V.1.1 Serialização

A serialização é um mecanismo importante, que permite armazenar o estado de um objeto na forma de *bytes* e depois recuperá-lo. O estado de um objeto contém, basicamente, informações sobre a classe e os valores de suas variáveis. Este objeto serializado pode ser armazenado tanto em um arquivo quanto em um *socket*. O primeiro caso possibilita que um programa, após ser interrompido, continue a execução a partir de onde foi interrompido pela última vez. A armazenagem em um *socket* será vista na próxima seção. Para que um objeto seja serializado, a classe da qual ele foi instanciado deve implementar a interface *Serializable*.

V.2 Mobilidade

A mobilidade é a principal característica dos agentes móveis, e como foi decidido não se utilizar nenhuma das plataformas de agentes móveis disponíveis na *Internet*, foi necessária a implementação de servidores que viabilizassem a mobilidade destes agentes e a troca de mensagens entre os agentes e a estação de gerenciamento.

Os agentes móveis são objetos e por isso devem ser serializados antes de serem enviados para outra estação. Assim um agente móvel pode interromper sua execução em uma máquina, armazenando seu estado e, ao chegar no destino começar sua execução com seu estado anterior recuperado. Dois mecanismos de comunicação podem ser usados na implementação dos servidores: *sockets* ou *Remote Method Invocation* - RMI. Foi decidido implementar os dois mecanismos para se verificar qual dos dois teria o melhor desempenho.

V.2.1 *Socket*

O *socket* é um mecanismo de comunicação, inicialmente desenvolvido para o sistema operacional Unix, que permite a comunicação entre duas estações remotas e que possui um alto grau de abstração. Desta maneira, o usuário não precisa se preocupar com detalhes de protocolos de comunicação. A Figura V.1 mostra uma maneira simplificada de estender esta abstração.

O programador não precisa saber detalhes da implementação dos protocolos das camadas inferiores para estabelecer uma conexão com outro computador. Cada linguagem deve possuir uma biblioteca que implemente o *socket*, disponibilizando para o programador funções que permitam a abertura e o fechamento do *socket*, a aceitação de pedidos de conexão e a escolha do protocolo de transporte a ser utilizando entre outras funções.

Em Java, particularmente, o *socket* é implementado como se fosse um descritor de arquivo. No lado do cliente, a aplicação deve escolher a estação para a qual deseja-se estabelecer uma conexão, a porta a ser utilizada e o protocolo de transporte a ser utilizado. Depois que a conexão é estabelecida, basta escrever ou ler do *socket* como se

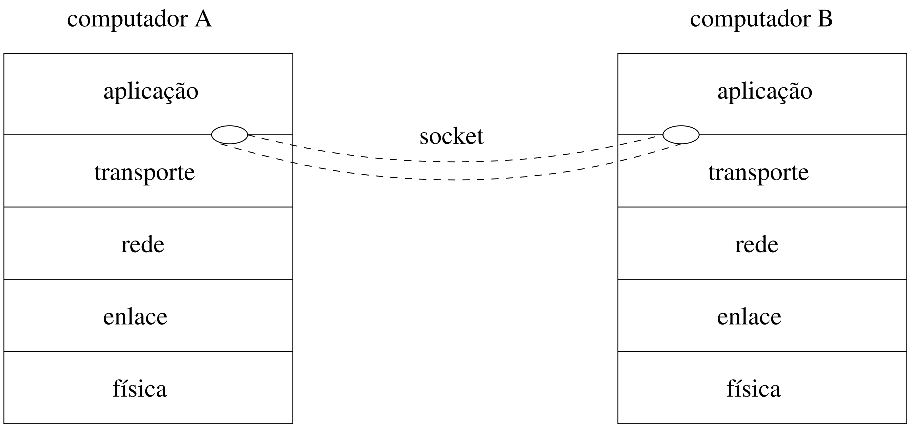

Figura V.1: *Socket* - comunicação entre dois computadores.

estivesse escrevendo ou lendo de um arquivo comum, para enviar e receber informações. No caso do *socket*, é preciso serializar o objeto antes de enviá-lo.

V.2.2 RMI

O RMI é a implementação em JAVA do RPC (chamada de procedimento remoto - *Remote Procedure Call*). O RPC permite que um programa possa executar procedimentos que estejam em outro computador da rede. Desta maneira, o RMI possibilita que um objeto sediado em uma estação qualquer da rede possa invocar métodos em uma outra máquina virtual Java de uma outra estação da rede.

Seu funcionamento se baseia no modelo cliente/servidor, no qual o servidor contém os métodos que serão chamados remotamente e o cliente invoca os métodos do servidor. O servidor deve se inscrever (realizar um *bind*) em uma base de dados que funciona como um registro e é chamada de *rmiregistry*. No ato do registro, o servidor deve informar o nome da estação, a porta e quais métodos estarão disponíveis para serem invocados remotamente.

O cliente deve fazer uma procura (*naming lookup*) no registro da estação da qual invocará o método remoto, informando a porta, o endereço IP e o nome do método. Após realizar esta operação com sucesso, ele já pode fazer a chamada ao método como se fosse um método local.

O mecanismo de comunicação utilizado pelo RMI é implementado pelo *stub*, do lado do cliente, e pelo *skeleton*, do lado do servidor. Eles são os responsáveis por abrir e fechar a conexão, enviar as informações e pela serialização dos objetos. O *stub* recebe pedidos do cliente (localmente) e os envia para o *skeleton*, que irá invocar o método do servidor e retornará os resultados para o *stub*. Para cada classe que exporta um método é associado um *stub* e um *skeleton*.

V.3 Classes

A seguir serão apresentadas todas as classes que foram criadas para este projeto. Além destas classes, também foram utilizadas classes do pacote *AdventNetSnmpV1* [28], que implementam a comunicação com os agentes SNMP. As classes foram separadas por níveis segundo a arquitetura proposta (Figura IV.3). Para cada nível, é ilustrado um diagrama de classes simplificado. A Figura V.2 mostra as convenções utilizadas em um diagrama de classes.

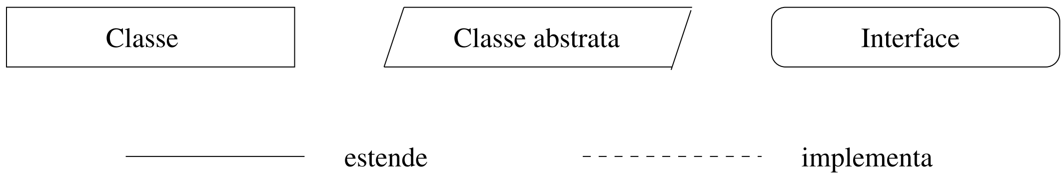

Figura V.2: Diagrama de classes - legenda.

V.3.1 Nível de usuário

As classes (Figura V.3) a seguir implementam as interfaces que possibilitam ao usuário interagir com a aplicação, criando, editando, executando tarefas e visualizando os resultados das tarefas executadas.

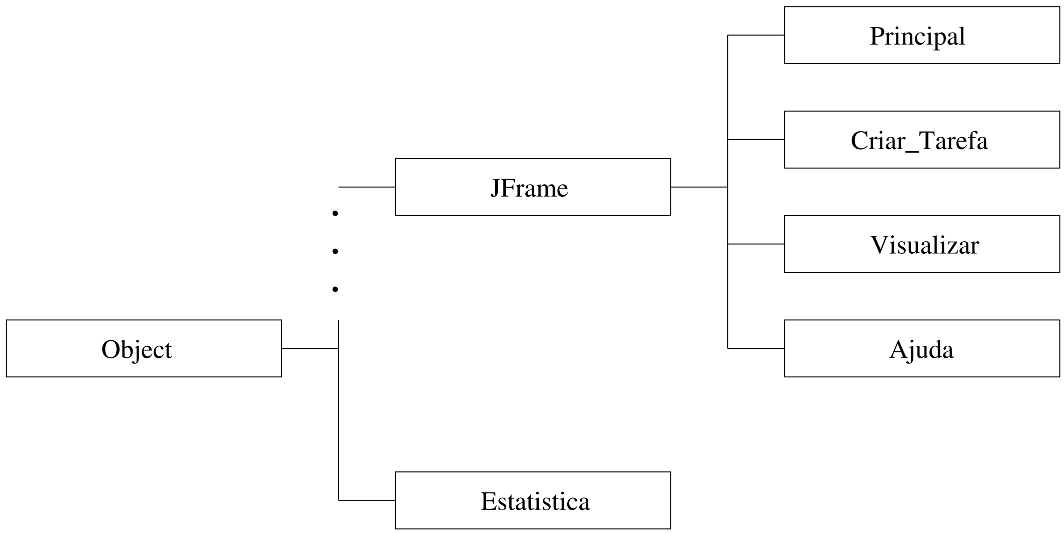

Figura V.3: Diagrama de classes - nível usuário.

Principal

É a classe responsável por conferir as configurações iniciais armazenadas no arquivo de configuração de cada máquina, iniciar a execução dos servidores e mostrar a tela principal do programa.

A tela principal possui um menu que contém as seguintes opções, como mostra a Figura V.4.

- Tarefa (Figura V.4):
  - Criar: Abre a tela que possibilita a criação de tarefas de gerenciamento;
  - Editar: Abre a tela que possibilita a edição de tarefas de gerenciamento preexistentes;
  - Executar: Abre a tela que possibilita a escolha de uma tarefa a ser executada;
  - Sair: Fecha o programa.
- Visualizar (Figura V.5):

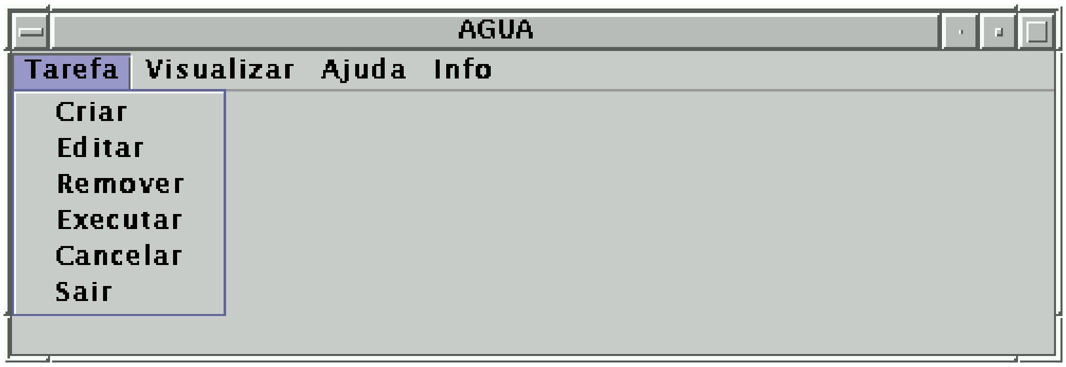

Figura V.4: Tela Principal - tarefa.

- Agentes: permite visualizar os agentes que estão sendo executados em uma mesma máquina;
- Resultados: abre a tela que possibilita a visualização dos resultados.

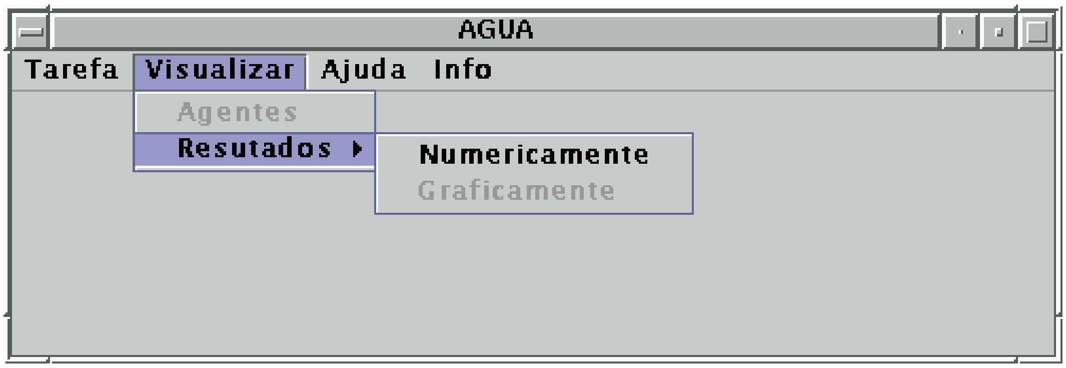

Figura V.5: Principal - Menu visualizar.

- Ajuda (Figura V.6):
  - Arquivos importantes: explica para que servem os arquivos de configuração e onde ficam os arquivos das tarefas, variáveis, itinerários e resultados;
  - Criar tarefa: explica os passos que devem ser seguidos para criar uma tarefa, bem como as características que cada tarefa pode ter;
  - Executar tarefas: explica como executar uma tarefa;
- Info: para ver informações sobre o autor do programa.

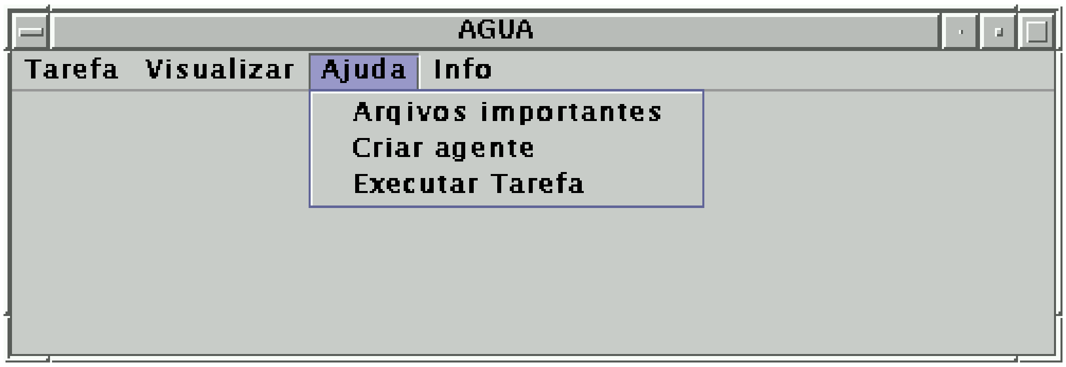

Figura V.6: Tela principal - Menu Ajuda.

Criar_Tarefa

Esta classe implementa as interfaces gráficas relacionadas com a criação de tarefas. Ela chama os métodos do Controlador_Tarefas e possui três telas:

- Características: nesta tela o usuário deve escolher as características básicas da tarefa, como o nome da tarefa, a operação a ser executada, o que o agente deve fazer em caso de falha ao tentar migrar para algum nó e as variáveis que serão consultadas ou atualizadas, como pode ser visto na Figura V.7;
- Itinerário: esta tela (Figura V.8) permite ao usuário definir o itinerário para a tarefa. Ele tanto pode escolher obter o itinerário de um arquivo, como pode editar um novo itinerário. O botão "atualizar" permite visualizar as máquinas disponíveis para que o usuário possa editar o itinerário da tarefa. Após terminada a edição, o usuário deve pressionar o botão "salvar" para que o novo itinerário seja armazenado;
- Resultados: nesta tela (Figura V.9), o usuário deve definir como os resultados serão entregues pelos agentes móveis, se será realizada alguma medida relacionada com o tempo de execução da tarefa, o número de agentes necessários para executar a tarefa, quantas vezes a tarefa será executada e o intervalo entre uma execução e outra.

Figura V.7: Tela criar tarefa - características.

Visualizar

Permite a visualização dos resultados obtidos em uma determinada tarefa. Além de mostrar os resultados, ainda mostra a média e desvio-padrão das medidas de tempo utilizando a classe Estatistica. A Figura V.10 mostra a tela de visualização do resultado do tempo total de execução da tarefa.

Ajuda

Mostra uma tela de ajuda com um caixa de texto e um botão para fechar como mostrado na Figura V.11.

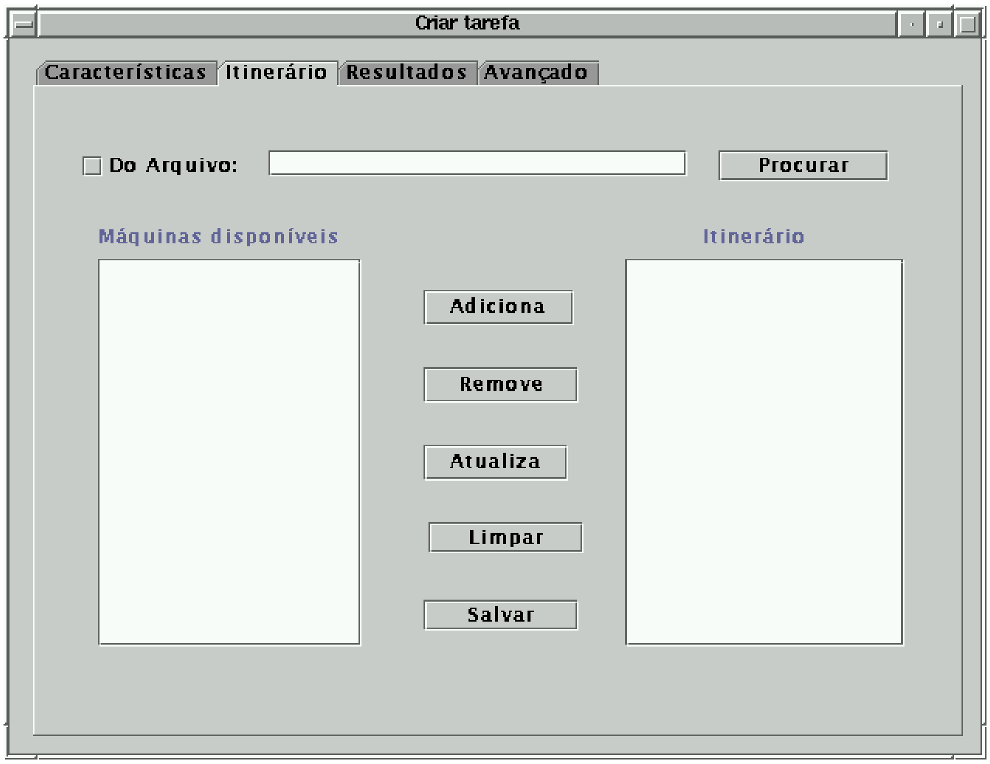

Figura V.8: Tela criar tarefa - itinerário.

Estatistica

Esta classe possui um método que calcula a média de um vetor de medidas e um outro método que calcula o seu desvio-padrão.

V.3.2 Nível de controle

As classes (Figura V.12) a seguir implementam a parte do controlador de tarefas e do controlador de agentes, ou seja, toda a parte de controle da aplicação.

ControlaAgentes

Possui métodos que permitem a execução propriamente dita dos agentes. Para executar um agente móvel, ele invoca o método "run" deste agente. Após iniciar a execução, o

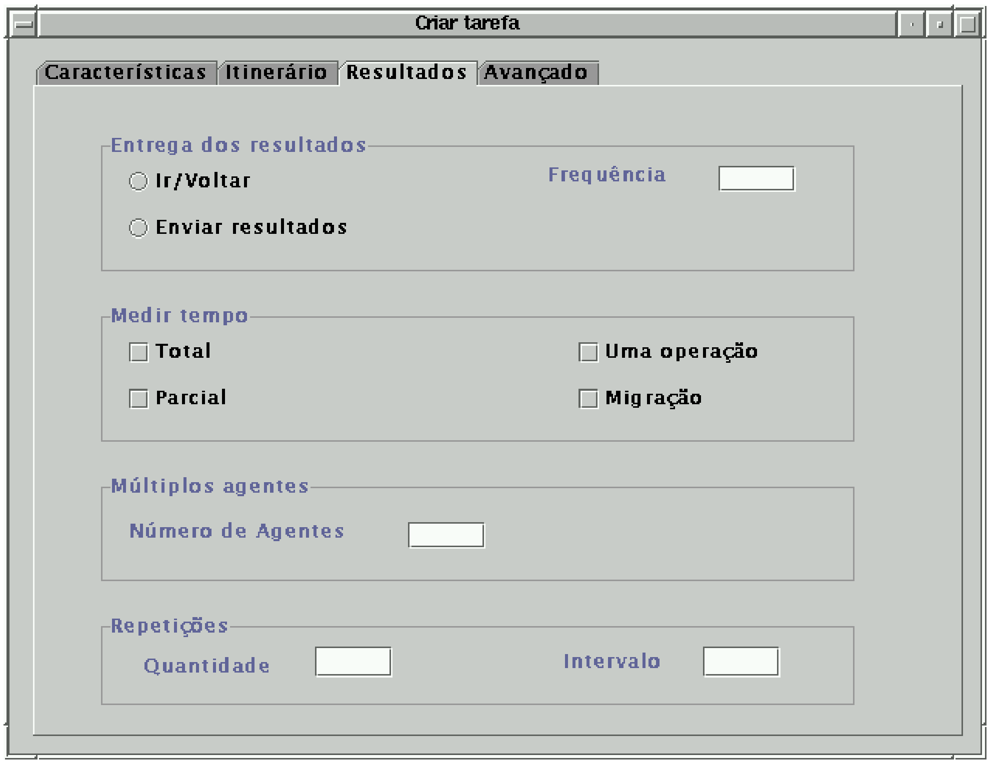

Figura V.9: Tela Resultados.

agente é armazenado em uma lista que auxilia no controle de todos os agentes em execução em uma determinada máquina.

ListaAgentes

Implementa métodos para armazenar, excluir e procurar agentes em uma lista. Esta é a lista dos agentes móveis que estão sendo executados no exato momento em uma determinada máquina.

Elemento

São os elementos armazenados na lista dos agentes. Possuem informações do agente, da tarefa que o agente está executando, a *thread* do agente e dados de controle da lista.

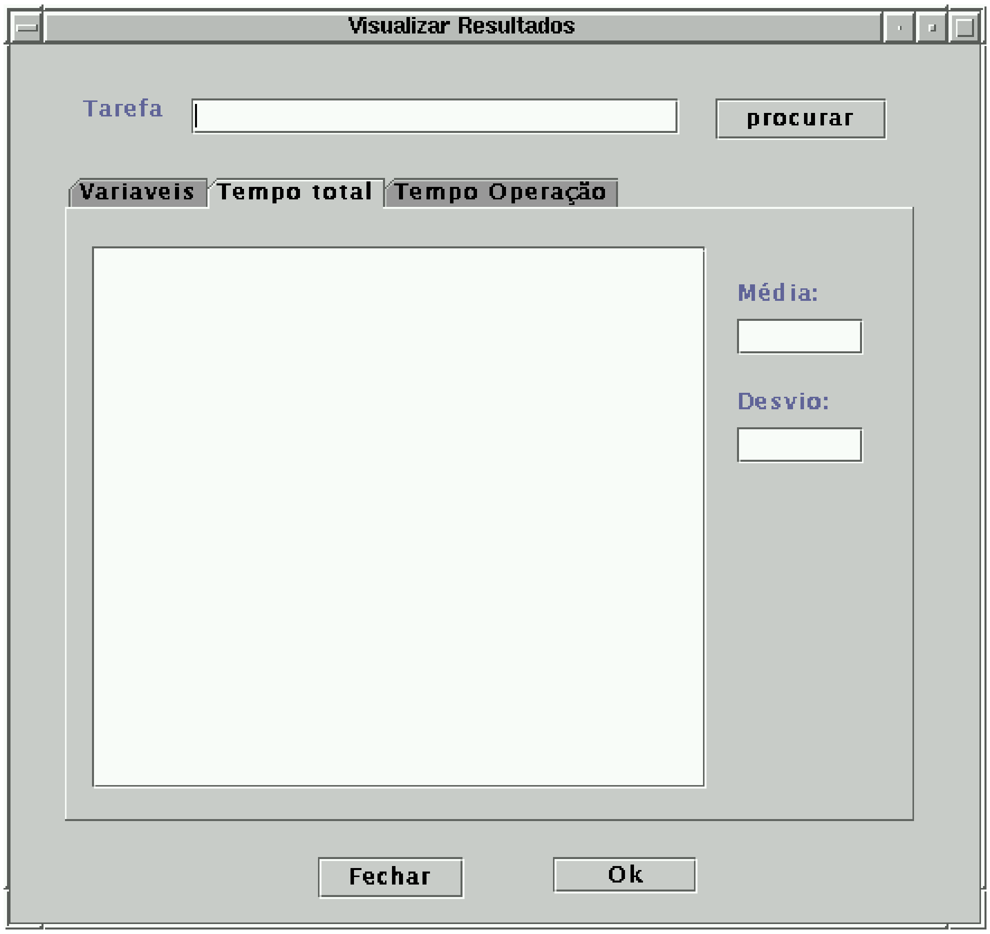

Figura V.10: Tela visualizar - tempo total.

Controlador_Tarefas

Esta classe implementa métodos que permitem criar, executar, armazenar e cancelar tarefas. É responsável também por armazenar os resultados enviados pelos agentes.

ListaTarefa

Implementa uma lista de tarefas que estão sendo executadas, utilizada pelo Controlador de Tarefas. Possui métodos de inclusão, exclusão e procura de tarefas.

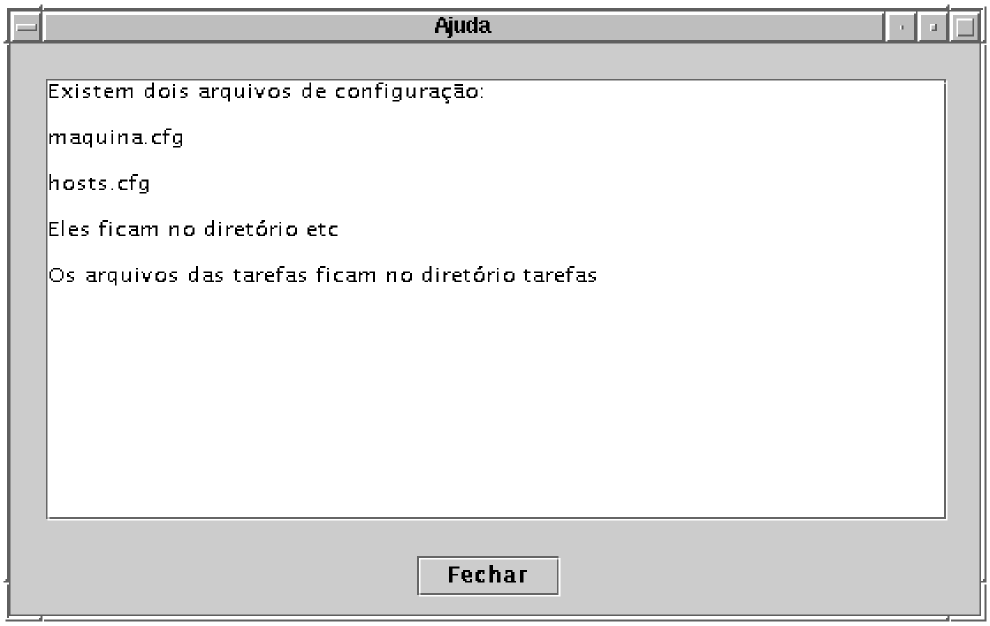

Figura V.11: Menu Ajuda - Arquivos importantes.

ElemenTar

Análogo ao Elemento, é a classe que armazena, além das tarefas propriamente ditas, informações extras para controle. Possui métodos que permitem saber se a tarefa já foi encerrada.

Tarefa

Esta classe estende a classe Arquivo e é utilizada para armazenar as características de uma determinada tarefa em um arquivo.

Variavel

Esta classe estende a classe Arquivo e é utilizada para armazenar as variáveis de interesse de cada tarefa em um arquivo.

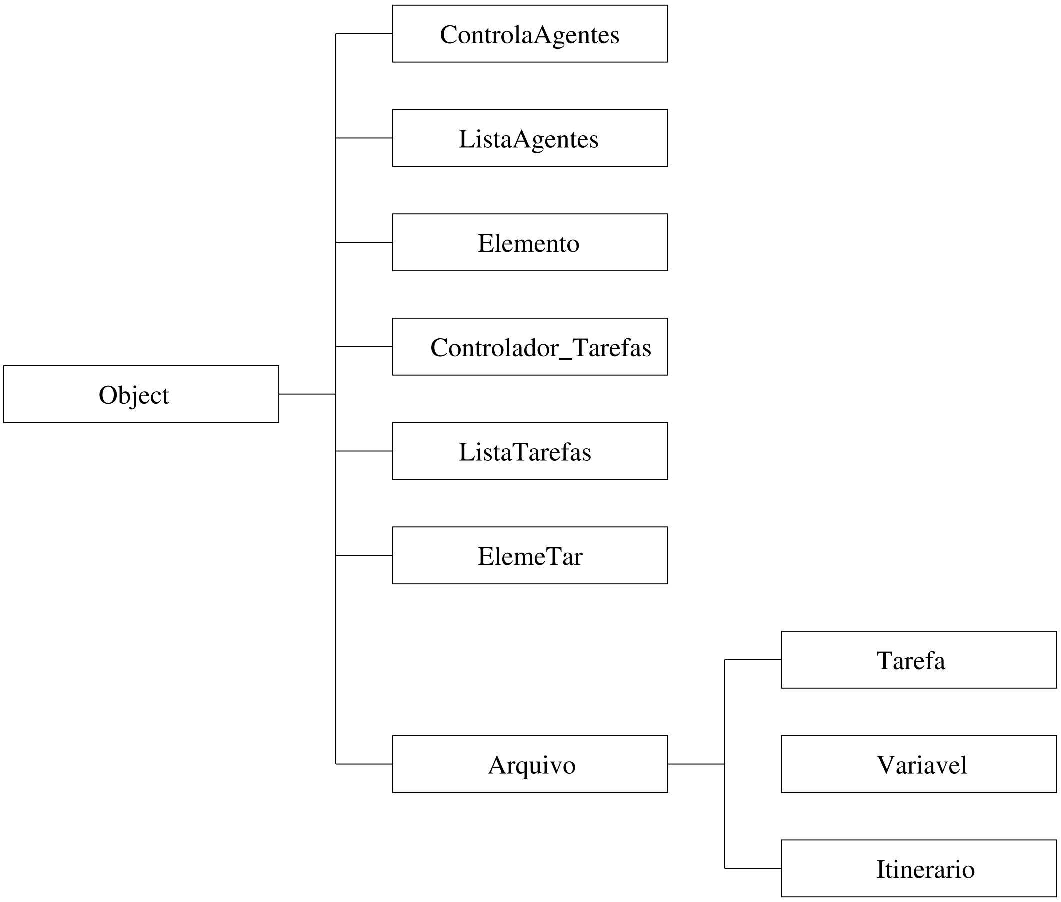

Figura V.12: Diagrama de classes - nível de controle.

Itinerario

Esta classe estende a classe Arquivo e é utilizada para armazenar o itinerário de cada tarefa em um arquivo.

V.3.3 Nível de agente

Estas classes (Figura V.13) implementam os agentes móveis e os fixos, além de outras classes referentes ao nível de agente.

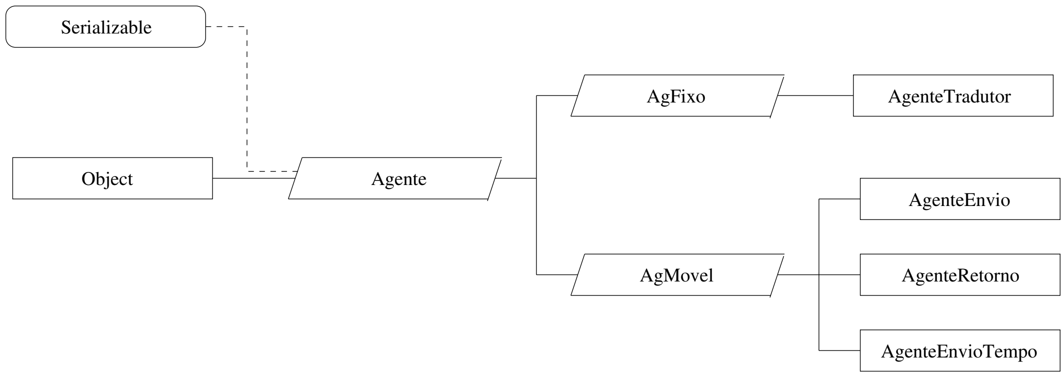

Figura V.13: Diagrama de classes - nível de Agente.

Agente

Classe abstrata que serve como base para todos os agentes.

AgFixo

Classe abstrata que estende a classe agente. Todos os agentes fixos devem estender esta classe.

AgMovel

Classe abstrata que estende a classe Agente. Contém o método "migrar" e informações que permitem a mobilidade dos agentes móveis. Todos os agentes móveis devem estender esta classe e devem possuir o método "run".

Agente_Tradutor

Classe que estende a classe AgFixo. Possibilita a comunicação entre os agentes móveis e os agentes SNMP através dos métodos de "consultar" e "atualizar".

AgenteEnvio

Agente móvel que busca as variáveis do vetor *variáveis* em cada hospedeiro do vetor *itinerário*, enviando os resultados com uma certa freqüência especificada pelo usuário. Em caso de falha, ele migra para o próximo nó.

AgenteRetorno

Agente móvel que busca as variáveis do vetor *variáveis* em cada hospedeiro do vetor *itinerário*, retornando à origem com os resultados. Em caso de falha, ele migra para o próximo nó.

AgenteEnvioTempo

É uma variação do AgenteEnvio, portanto, além de possuir todos as características do AgenteEnvio, ele também é capaz de medir o tempo de consulta das variáveis em cada nó.

V.3.4 Nível de comunicação

As classes (Figura V.14) a seguir implementam toda a parte da mobilidade e da comunicação entre os agentes e a estação de gerenciamento. A porta a ser utilizada pelos servidores deverá ser escolhida pelo usuário. No caso do *socket*, são utilizadas duas portas diferentes, uma para o envio de mensagens e outra para o envio de agentes.

Servidor

Classe que é a base dos servidores que utilizam o *socket* como mecanismo de comunicação.

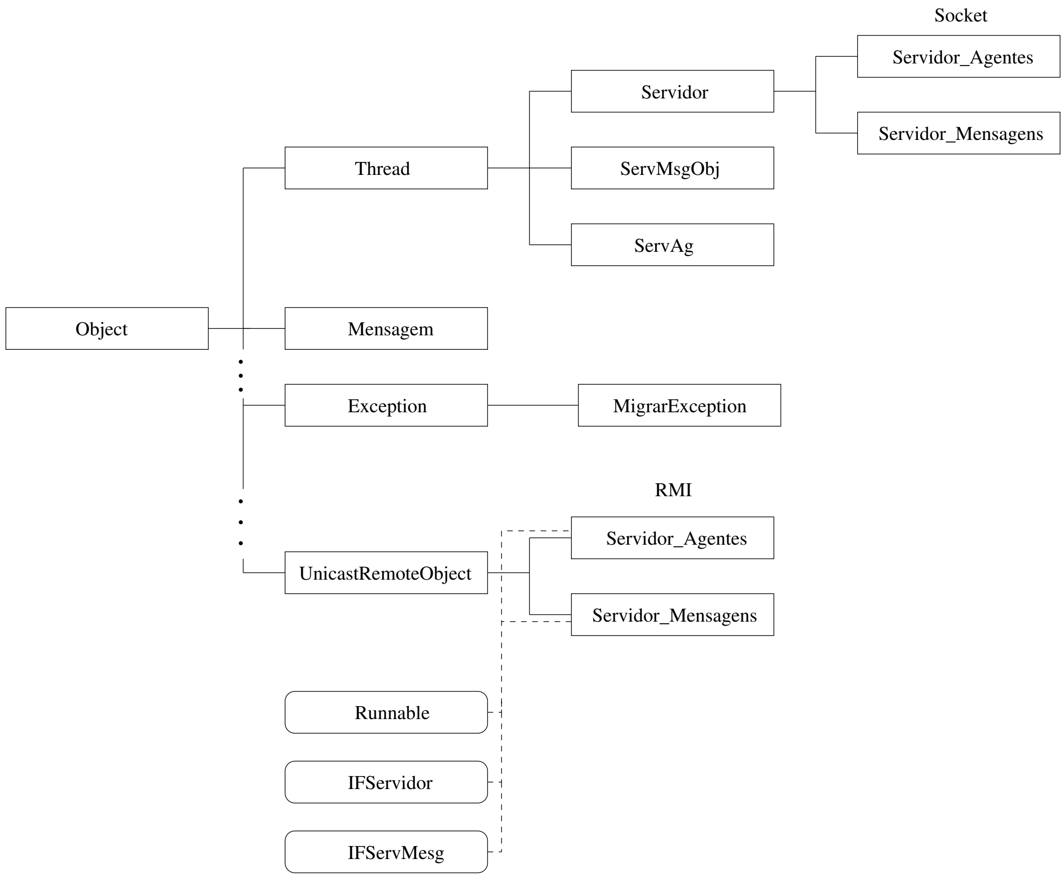

Figura V.14: Diagrama de classes - nível de comunicação.

Servidor_Agentes

Esta classe implementa um servidor concorrente que contém os métodos que permitem enviar e receber os agentes e, por isso, também possui duas versões. A primeira versão implementa o mecanismo de comunicação via *socket* e por isso, estende a classe Servidor. A outra versão implementa a comunicação por RMI e, portanto, estende a classe UnicastRemoteObject e implementa a interface IFServidor. O servidor de agentes invoca métodos do controlador de agentes para executar os agentes móveis que chegam a um determinado nó.

Servidor_Mensagens

Esta classe implementa um servidor concorrente que receberá os resultados dos agentes móveis. Também possui um método que possibilita o envio de mensagens pelos agentes. Existem duas verões desta classe. A primeira versão implementa o mecanismo de comunicação via *socket* e, por isso, estende a classe Servidor. A outra versão implementa a comunicação por RMI e, portanto, estende a classe UnicastRemoteObject e implementa a interface IFServidor.

ServMsgObj

Esta classe estende a classe Thread. Ela executa uma nova *thread* para tratar os pedidos recebidos pelo servidor de mensagens que utiliza o *socket* como mecanismo de comunicação.

ServAg

Esta classe estende a classe Thread. Ela executa uma nova *thread* para tratar os pedidos recebidos pelo servidor de agentes que implementa o mecanismo de comunicação via RMI.

MigrarException

Esta exceção é lançada pelo servidor de agentes quando o agente móvel não consegue migrar para um determinado nó.

IFServidor

É a interface implementada pela classe Servidor_Agente que contém o método que permite o envio de agentes via RMI.

IFServMesg

É a interface implementada pela classe Servidor_Mensagens que contém o método que permite o envio de mensagens via RMI.

Mensagem

Esta classe encapsula todos os dados referentes aos resultados que serão enviados, assim como métodos para envio da mensagem e leitura dos resultados. Existem dois tipos de mensagem: o primeiro é uma mensagem que contém os resultados obtidos pelos agentes móveis e o segundo é uma mensagem de controle indicando que o agente acabou de executar a tarefa para a qual ele foi designado.

Arquivo

Esta classe possui métodos que permitem criar, armazenar e ler dados de um arquivo. Além disto permite editar arquivos, como por exemplo tirar linhas em branco, apagar linhas que contenham um determinado caracter e apagar uma linha até um determinado caracter.

V.3.5 Sistema de diretórios

A aplicação contém um diretório principal ("AGUA_HOME"), que deve ser uma variável de ambiente. Este diretório principal contém os seguintes diretórios:

- classes: é dividido em outros quatro subdiretórios (rmi, socket, interface e comum) nos quais são armazenadas as respectivas classes;
- codigo: é dividido em outros quatro subdiretórios (rmi, socket, interface e comum) nos quais são armazenados os respectivos códigos fonte de cada classe;
- doc: armazena os arquivos de ajuda;

- etc: contém os arquivos de configuração;
- itinerarios: armazena os arquivos de itinerários;
- resultados: contém os arquivos de resultado de cada tarefa;
- tarefas: armazena os arquivos de tarefas;
- variaveis: armazena os arquivos de variáveis.

Além destes diretórios ainda existem três *scripts* no diretório principal. O primeiro, "run", executa a aplicação utilizando RMI. O segundo, "srun", executa a aplicação utilizando *socket*. O último, "compilar", como o próprio nome indica, compila todos os arquivos-fonte e armazena as classes nos seus respectivos diretórios. Todos os arquivos podem ser vistos em um exemplo no Apêndice A.

V.3.6 Arquivos de Configuração

Os arquivos de configuração necessários para execução desta aplicação são:

- maquina.cfg: a primeira linha deste arquivo deve conter o nome da máquina em que será executada a aplicação. A segunda linha deve conter a porta que o servidor deverá utilizar. Não é preciso especificar mais de uma porta para usar o *socket* (que, como já foi mencionado anteriormente, precisa de duas portas para funcionar), pois a segunda porta é assumida como sendo a porta seguinte à que foi definida neste arquivo.
- hosts.cfg: este arquivo deve conter os nomes de todas as máquinas da rede que se deseja gerenciar. Ele permite a edição do itinerário através da interface gráfica.

Capítulo VI

Os mecanismos de comunicação

Neste capítulo, serão apresentados os testes que foram realizados com o objetivo de comparar o desempenho da aplicação utilizando os mecanismo de comunicação *Remote Method Invocation* e *Socket* na implementação dos servidores.

VI.1 Ambiente de testes

Para a realização dos testes foram utilizados dois microcomputadores compatíveis IBM-PC. Um microcomputador com processador AMD 1.2GHz e 512MB de memória RAM desempenhou o papel de estação de gerenciamento e de estação a ser monitorada. Outro microcomputador com processador Pentium MMX 233MHz e 128MB de memória RAM desempenhou o papel de estação a ser monitorada. Ambos estão conectados a uma rede *Ethernet* (padrão de rede local) com velocidade de transmissão de 10Mb/s. Os testes foram realizadas em horários em que a rede estava subutilizada.

VI.1.1 Tarefa

Foi definida uma tarefa bem simples, com apenas um agente móvel, para a realização dos testes. Como o objetivo era avaliar o desempenho da comunicação, foi definida uma tarefa na qual houvesse o maior número possível de trocas de mensagens. Nesta tarefa, o agente deve consultar o valor de uma variável a cada nó, enviando o resultado logo em seguida. O único parâmetro variado foi o itinerário, que para cada teste assumiu os valores de 5, 10, 20, 40, 80, 160, 320 nós percorridos. Cada tarefa foi realizada 10 vezes para se poder tirar uma média e o respectivo desvio-padrão.

VI.2 Resultados

As Tabelas VI.1 e VI.2 ilustram os resultados obtidos nos testes. Em algumas medidas os desvios-padrão ficaram altos, ou seja, houve uma grande variação das medidas, chegando a 13,1% do valor da média.

Tabela VI.1: RMI - resultados das medidas.

nós	valor (ms)	desvio	desvio/valor (%)
5	654,0	85,7	13,1
10	1.259,4	49,0	3,9
20	2.522,4	147,3	5,8
40	5.287,6	295,4	5,6
80	11.823,7	1.154,5	9,8
160	23.431,0	1.403,2	6,0
320	55.029,1	4.062,7	7,4

O gráfico da Figura VI.1 mostra que não houve quase diferença entre os dois mecanismos para um número de nós inferior a 100. Isto porque, nesta região do gráfico, as diferenças entre as curvas ainda eram muito pequenas e, como houve grandes variações nos valores das medidas, as duas curvas acabaram se confundindo uma com a outra. Inclusive, notam-se trechos em que uma curva tem o desempenho superior à outra e trechos em que acontece o oposto. A partir de 100 nós já se começa a perceber uma pequena diferença, que vai aumentando conforme cresce o número de nós. Nesta faixa do gráfico,

Tabela VI.2: Socket - resultados das medidas.

nós	valor (ms)	desvio	desvio/valor (%)
5	689,0	51,2	7,4
10	1.205,4	103,9	8,6
20	2.787,5	160,4	5,8
40	5.277,0	274,3	5,2
80	11.130,2	968,3	8,7
160	25.934,4	1.248,7	4,9
320	68.349,1	2.024.4	3,9

o desempenho do RMI é superior ao do *socket*, como já era esperado, pois o RMI já possui toda a parte de serialização, abertura e fechamento de conexão implementada em linguagem nativa da máquina.

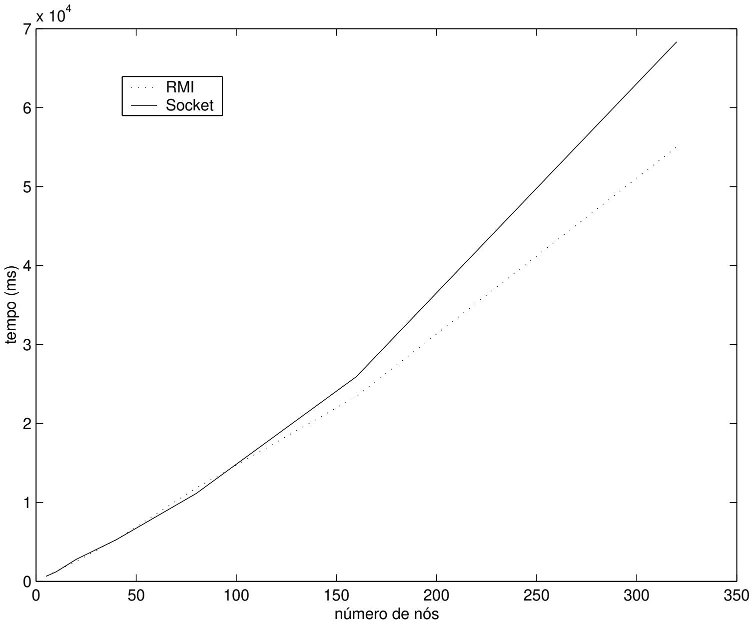

Figura VI.1: Gráfico do desempenho de RMI X *socket*

Capítulo VII

Conclusão

Este trabalho teve como objetivo desenvolver e implementar uma aplicação de gerenciamento utilizando a tecnologia de agentes móveis que permitisse ao usuário realizar, com facilidade, as tarefas de gerenciamento. Assim, foi implementada uma interface gráfica (com menus, botões, caixas de diálogo etc.) que proporcionam ao usuário a criação de tarefas de uma maneira simples, permitindo definir as características básicas de uma tarefa (operação, itinerário e variáveis), além de muitas outras características complementares que fornecem a possibilidade de determinar como a tarefa será executada.

O projeto foi desenvolvido em Java devido à simplicidade, à portabilidade e ao alto grau de reaproveitamento de código que a linguagem proporciona. Ele é dividido em duas partes principais: a aplicação de gerenciamento e a plataforma simplificada de agentes móveis. A arquitetura proposta é composta de camadas hierárquicas independentes, o que possibilita um alto grau de modularidade.

A aplicação de gerenciamento implementada permite realizar medidas relacionadas ao tempo de execução das tarefas, o que é de grande utilidade para estudos de desempenho, tendo em vista que esta é uma nova área e, por isso, ainda está sendo muito investigada.

A plataforma de agentes móveis implementada é uma infra-estrutura simplificada que possibilita a execução, a migração e a comunicação dos agentes. Foram implementadas duas versões da plataforma. Na primeira, a comunicação dos servidores é realizada via *socket* e na segunda, a comunicação entre os servidores é realizada via RMI (*Remote Method Invocation*).

A principal contribuição deste trabalho é o fato de ter sido desenvolvida uma aplicação que não somente contém toda uma infra-estrutura básica para realização de tarefas de gerenciamento utilizando agentes móveis como também provê muitas outras funcionalidades. Além disto, a aplicação é inteiramente dedicada, pois a parte referente à infra-estrutura dos agentes móveis foi implementada especificamente para esta aplicação, não sendo utilizada nenhuma outra plataforma de agentes móveis.

Outro ponto importante a ser ressaltado é a modularidade da aplicação, que permite que novas classes e novos agentes sejam criados e adicionados ao programa principal. Assim, esta aplicação pode ser utilizada para novos estudos sobre o uso de agentes móveis no gerenciamento de redes. Um exemplo seria a criação de agentes inteligentes.

A partir das medidas realizadas é possível concluir que o desemenho da aplicação utilizando o mecanismo RMI é superior ao seu desempenho utilizando *socket*. No entanto, a diferença foi desprezível para um pequeno número de nós, somente sendo percebida para um grande número de nós.

A maior limitação do desempenho da aplicação é a linguagem Java, que, por ser uma linguagem interpretada, faz com que os programas se tornem "pesados". Entretanto, com processadores cada vez mais velozes, este problema fica minimizado a ponto de não ser notado, como ocorreu no computador no qual foram realizadas as simulações (AMD 1.2GHz).

A versão final do programa possui dois arquivos de configuração, cinco arquivos de armazenagem de dados, 38 novas classes e mais de 4.200 linhas de programa, estando disponível no endereço "http://www.gta.ufrj.br/p̃velloso/pf".

Como trabalhos futuros podem-se acrescentar novas funcionalidades à aplicação, como por exemplo a visualização dos resultados através de gráficos, o que possibilitaria monitorar variáveis (número de pacotes perdidos, transmitidos, recebidos etc.) ao longo do tempo de uma maneira muito mais eficiente. Outra possibilidade seria o desenvolvimento de agentes inteligentes que, de alguma forma, fossem capazes de reconhecer o comportamento da rede e notificar os eventos ou, até mesmo, tomar alguma decisão.

Referências Bibliográficas

[1] J. Case, M. Fedor, M. Schoffstall e J. Davin, "Simple management network protocol", *Internet RFC 1157*, maio de 1990.

[2] W. Stallings, *Network Management.* Los Alamitos, CA: IEEE Computer Society Press, 1993.

[3] W. Stallings, *SNMP, SNMPv2, SNMPv3, RMON 1 and 2.* Adison-Wesley, third ed., 1999.

[4] R. P. Avalle, "Gerenciamento de redes". http://www.gta.ufrj.br/grad/99_1/ rodrigo/ger_redes.htm, agosto de 1999.

[5] M. G. Rubinstein e O. C. M. B. Duarte, "Análise da eficiência de agentes móveis no gerenciamento de redes", in *Seminário Integrado de Software e Hardware*, julho de 1999.

[6] C. de Paula Bianchini e A. F. do Prado, "Ferramenta para monitoramento de dispositivos utilizando computação pervasiva e agentes inteligentes", in *Simpósio Brasileiro de Redes de Computadores*, maio de 2001.

[7] W. T. Cockayne e M. Zyda, *Mobile Agents.* Manning Publications Co., first ed., 1998.

[8] D. B.Lange e M. Oshima, "Seven good reasons for mobile agents", *Communications of the ACM*, vol. 42, no. 3, pp. 88–89, março de 1999.

[9] J. D. de Queiroz, "Micael: Uma arquitetura para detecção de atividades intrusas baseada em agentes móveis de software", Tese de Mestrado, Universidade Federal do Rio de Janeiro, março de 2001.

[10] M. G. Rubinstein e O. C. M. B. Duarte, "Infra-estrutura GTA/AG: Um ambiente de desenvolvimento de sistemas baseados em agentes", Relatório Técnico, GTA-98-01, Grupo de Teleinformática e Automação, janeiro de 1998.

[11] P. B. Velloso, "Agentes móveis". http://www.gta.ufrj.br/grad/00_1/pvelloso/ index.html, agosto de 2000.

[12] Universidade Cornell, http://www.tacoma.cs.uit.no/, *TACOMA*.

[13] Dartmouth College, http://agent.cs.dartmouth.edu/software/agent2.0/, *D'Agents*.

[14] Universidade de Kaiserslautern, http://uni-kl.de/AGNehmer/Projekte/ARA/indexe.html, *ARA*.

[15] IBM Corporation, http://www.trl.ibm.com/aglets, *Aglets*.

[16] University of Stuttgart, http://mole.informatik.uni-stuttgart.de, *The Home of Mole*, 1995.

[17] Universidade de Minnesota, http://www.cs.umn.edu/Ajanta/home.html, *AJANTA*.

[18] Universidade de Stanford, http://java.stanford.edu/, *JATLite*.

[19] Media Lab - MIT, http://www.hivecell.net/, *Hive*.

[20] Sun, http://www.sun.com/software/java-dynamic/ ;*sessionid*ASHQDGYAAHUMDAMUVFZE4GUBS1JHEUDO, *JDMK*.

[21] Mitsubishi, http://www.concordiaagents.com/, *Concordia*.

[22] Foundation for Intelligent Physical Agents, http://www.fipa.org/, *The Foundation for Intelligent Physical Agents*.

[23] Nortel Networks, http://www.nortelnetworks.com/products/announcements/fipa/, *FIPA-OS*.

[24] *Java Agent DEvelopment framework*. http://www.sharon.cselt.it/projects/jade.

[25] M. G. Rubinstein, *Avaliação do Desempenho de Agentes Móveis no Gerenciamento de Redes*. Tese de Doutorado, Universidade Federal do Rio de Janeiro, março de 2001.

[26] D. Flanagan, *Java in a Nutshell*. O'Reilly, third ed., 1999.

[27] Sun Corporation, http://java.sun.com/j2se/1.3/download-linux.html/, *Java 2 SDK Plataform, Standard edition 1,3.0*.

[28] AdventNet, http://www.adventnet.com/products/index.html#snmp, *AdventNetSNMPv1*.

Apêndice A

Arquivos

Este apêndice mostra o conteúdo dos arquivos da aplicação. No caso dos arquivos de configuração e de armazenagem de dados são apresentados exemplos.

A.1 Configuração

Os arquivos de configuração necessários para execução desta aplicação são apresentados a seguir.

A.1.1 maquina.cfg

```
A.gta.ufrj.br   /* nome da máquina
1099            /* porta
```

A.gta.ufrj.br /* nome da máquina
B.gta.ufrj.br /* nome da máquina
C.gta.ufrj.br /* nome da máquina
D.gta.ufrj.br /* nome da máquina
E.gta.ufrj.br /* nome da máquina

A.1.2 hosts.cfg

A.2 *Scripts*

Os *scripts* são pequenos programas escritos em linguagem de *shell* utilizados para executar e compilar a aplicação.

A.2.1 run

#!/bin/sh

cd classes/interface

userdir=$AGUA_HOME

echo $userdir

java -cp .:/lab/users/Otto-1/pvelloso/graduacao/pf/aplicacao/classes/rmi: /lab/users/Otto-1/pvelloso/graduacao/pf/aplicacao/classes/comum: /usr/local/AdventNetSnmpV1/classes/: /usr/local/AdventNetSnmpV1/swingall.jar Principal $1 $userdir

A.2.2 compilar

#!/bin/sh

cd codigo/rmi

userdir=$AGUA_HOME

echo $userdir

javac -classpath .:userdir/codigo/socket: userdir/codigo/interface: userdir/codigo/comum: /usr/local/AdventNetSnmpV1/classes/: /usr/local/AdventNetSnmpV1/swingall.jar *.java

rmic Servidor_Mensagens rmic Servidor_Agente

mv -f *.class ../../classes/rmi

cd ../interface

javac -classpath .:userdir/codigo/socket: userdir/codigo/rmi: userdir/classes/rmi: userdir/codigo/comum: /usr/local/AdventNetSnmpV1/classes/: /usr/local/AdventNetSnmpV1/swingall.jar *.java

mv -f *.class ../../classes/interface

cd ../comum

javac -classpath .:userdir/codigo/socket: userdir/codigo/rmi: userdir/classes/rmi: userdir/codigo/interface: /usr/local/AdventNetSnmpV1/classes/: /usr/local/AdventNetSnmpV1/swingall.jar *.java

mv -f *.class ../../classes/comum

cd ../socket

javac -classpath .:userdir/codigo/interface: userdir/codigo/rmi: userdir/codigo/comum: /usr/local/AdventNetSnmpV1/classes/: /usr/local/AdventNetSnmpV1/swingall.jar *.java

mv -f *.class ../../classes/socket

cd ../..

echo 'Pronto'

A.3 Armazenagem de dados

São os arquivos que armazenam informações sobre as tarefas a serem executadas, as variáveis a serem obtidas, os itinerários a serem seguidos pelos agentes e os resultados finais.

A.3.1 tarefa.tsk

teste	/* nome da tarefa
teste	/* nome do arquivo do itinerário
teste	/* nome do aqruivo de variáveis
get	/* operação
proximo	/* em caso de falha
enviar	/* entrega de resultados
4	/* freqüência de entrega de resultados
2	/* número de arquivos de resultados
Ttotal	/* pegar tempo total
NaoMedirTempo	/* pegar tempo parcial
NaoMedirTempo	/* pegar tempo de uma operação
1	/* número de agentes
nao	/* mudança de itinerário
nao	/* operação condicional
10	/* número de repetições
0	/* intervalo

A.3.2 variaveis.var

.1.3.6.1.2.1.1.5.0 /* representa o nome da máquina

.1.3.6.1.2.1.6.10.0 /* representa a quantidade de segmentos recebidos

.1.3.6.1.2.1.6.11.0 /* representa a quantidade de segmentos transmitidos

.1.3.6.1.2.1.6.12.0 /* representa a quantidade de segmentos retransmitidos

A.3.3 itinerario.itn

A.gta.ufrj.br

B.gta.ufrj.br

A.3.4 resultados.txt

A.gta.ufrj.br

1000267

1279000

3035

B.gta.ufrj.br

1020522

2176040

3511