UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Edison Alessandro Xavier

APLICAÇÃO DE INTELIGÊNCIA COMPUTACIONAL NA GERÊNCIA DE REDES ATRAVÉS DA AUTOMATIZAÇÃO DO USO DE AGENTES MÓVEIS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Carlos Becker Westphall
Orientador

Florianópolis, Fevereiro de 2003

APLICAÇÃO DE INTELIGÊNCIA COMPUTACIONAL NA GERÊNCIA DE REDES ATRAVÉS DA AUTOMATIZAÇÃO DO USO DE AGENTES MÓVEIS

Edison Alessandro Xavier

Essa dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Fernando Alvaro Ostuni Gauthier, Dr.
Coordenador do CPGCC

Banca Examinadora

Prof. Carlos Becker Westphall, Dr.
Orientador

Profa. Carla Merkle Westphall, Dra.
Membro da Banca

Prof. Ricardo Felipe Custódio, Dr.
Membro da Banca

Prof. Roberto Willrich, Dr.
Membro da Banca

“Os limites podem ser transpostos por aqueles
que ousam olhar para fora”

Agradeço toda a paciência e colaboração do Professor Carlos Westphall, que sempre acreditou no meu trabalho. Ao Fernando Koch, agradeço pelas lutas incansáveis com nossos artigos. À minha mãe, Eunice, e minhas irmãs, Vanessa e Priscila, pela torcida calorosa. Por fim, agradeço à Carolyna, pelo carinho e companheirismo ilimitados.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A gerência de redes de computadores é uma atividade computacional que precisa ser implementada de maneira a obter os dados ou enviar ações ao ambiente gerenciado da melhor forma possível. Dentre as formas de implementação da coleta dos dados destacamos os agentes móveis, que possuem características interessantes: podem variar a quantidade de agentes envolvidos ou o momento de entrega dos dados ao gerente. Essa variação abre um ponto de decisão: qual a melhor configuração de agentes móveis, dada uma gerência? Esse trabalho inicia investigando as variações das configurações de agentes móveis, obtendo métricas que podem ser classificadas e avaliadas, dando subsídios para a decisão de qual a melhor configuração. Por fim, apresenta soluções de decisão baseadas em paradigmas de inteligência computacional, notadamente redes neurais, lógica *fuzzy* e regras de inferência.

Abstract

The network management is a computational activity that needs to be implemented in way to get data or make necessary actions to the managed environment in the best possible manner. Amongst the forms of implementation of data collects we detach the mobile agents, who possess interesting properties: they can use one or multiple agents or still they can vary the moment that the data colected is sending to the manager. These properties opens a decision point: which the best configuration to make one known management? This work initiates investigating the variations of mobile agents, getting a metric that can be classified and be evaluated, giving subsidies for the decision of which is the best configuration. Finally, it presents solutions of decision based in paradigms of computational intelligence, like as neural networks, fuzzy logic and inference rules.

1. Introdução

A gerência de redes de computadores possui um grande desafio: precisa ser rápida e eficiente e ao mesmo tempo coerente com sua finalidade. Rápida e eficiente porque as suas coletas e observações acerca do ambiente gerenciado precisam oferecer subsídios necessários para a aplicação da gerência com certa agilidade e objetividade. Precisa ainda ser coerente, consumindo poucos recursos deste ambiente gerenciado sob a pena de ser a gerência em si a grande causadora dos problemas neste ambiente.

Sob esses argumentos, precisamos observar os modos pelos quais a gerência é implementada. A gerência cliente-servidor, basicamente representada por implementações SNMP [RFC1157][STALLINGS 1998] ou CMIP [ISO-OSI 9595], possui características gerais de coleta de dados e decisão centralizados, sujeitos a problemas de escalabilidade e de alto consumo de banda no gerente.

Os agentes móveis possuem características convenientes, sob o ponto de vista de um agente ou de um grupo de agentes (comunidade de agentes) [OHARE 1996], e que podem ser aproveitadas na gerência de redes. Podemos, por exemplo, compor comunidades com um ou vários agentes; podemos ainda solicitar que um agente colete um dado e o envie imediatamente ao gerente ou, de outra forma, que esse mesmo agente faça a coleta em vários elementos gerenciados, enviando esses dados coletados somente ao final de sua jornada.

Dadas as características citadas, podemos conceber várias comunidades de agentes para a mesma gerência. Com isso, cada comunidade possui características diferentes na forma de obtenção da gerência, e que podem ser avaliadas oferecendo ao gerente humano possibilidades de escolha, de acordo com suas conveniências.

A avaliação das comunidades de agentes móveis é uma opção muito interessante ao gerente humano, desde que ele tenha essa possibilidade sem a necessidade de implementar e experimentar todas elas. Caso contrário, a gerência em si já teria sido efetuada, além das experimentações desnecessárias, que acabariam por ocasionar um consumo de banda desnecessário e prejudicial. Essa avaliação tem como resultado algumas métricas obtidas para cada configuração.

Temos ainda a classificação dessas configurações, através de alguns critérios estabelecidos pelo gerente humano, levando em consideração as métricas obtidas nas

avaliações. Esses critérios são representações das inferências feitas pelo gerente humano, por exemplo, sob algum paradigma de inteligência computacional.

Por fim temos a escolha da configuração mais adequada à gerência, considerando as classificações obtidas. Esse processo, que também pode ser implementado sob algum paradigma de inteligência computacional, simula o raciocínio humano na tomada de decisão, baseado nas classificações encontradas para cada uma das comunidades de agentes móveis consideradas.

Temos então as propostas de três modelos:

- Modelo de avaliação das comunidades de agentes móveis, sob algumas métricas;
- Modelo de classificação dessas comunidades entre si, sob determinados critérios, tendo como base, as métricas encontradas anteriormente; e
- Modelo de decisão da comunidade mais adequada à gerência, dadas algumas regras, definidas pelo gerente humano.

Com esses modelos, temos de uma forma automatizada, a decisão de qual a melhor comunidade de agentes móveis a ser aplicada em determinada gerência de rede de computadores. Além disso, essa decisão é obtida sem custo para o ambiente , pois é obtida em cima de modelos matemáticos, sem nenhuma implementação, de acordo com os objetivos da própria gerência de redes.

1.1 Estado-da-arte

Existem estudos que tratam a aplicabilidade direta do paradigma de agentes móveis enquanto solução na implementação de gerência de redes [BOHORIS 2000][KAZI 1999][THOTTAN 1998], tendo como foco a atividade de gerência em si. Em [BIESZCZAD 1998] temos argumentações sobre a utilização dos agentes móveis em cada uma das áreas funcionais da gerência de redes, propostas pela [ISO-OSI 10165], e descritas em [YEMINI 1993]. Em [BALDI 1997] temos a avaliação da gerência de redes em cada um dos paradigmas que envolvem mobilidade de código.

A implementação de gerência de redes, através de outros paradigmas e suas comparações, também é alvo de vários estudos. Existem estudos comparando o comportamento de agentes móveis com SNMP. O trabalho de [RUBINSTEIN 2001] faz comparações com SNMP em cima de implementações e modelos aproximados. Existem

trabalhos como os de [ARANTES 2002], que propõe modelos matemáticos para cálculos de tempo gasto em gerências, por agentes móveis e SNMP, em cima de estudos de casos. Nestes trabalhos percebemos a dificuldade em conceber modelos de avaliação genéricos, pois as implementações das plataformas de agentes móveis não seguem nenhum padrão. Existem propostas em [FIPA] que sugerem padrões de comunicação entre comunidades e agentes, e que estão disponíveis em documentos como o XC00082B (FIPA Network Management and Provisioning Specification), que especifica padrões utilizados no provisionamento de recursos e na gerência de redes, utilizando agentes, ou o XC00084D (FIPA Agent Message Transport Protocol for HTTP Specification) que discorre sobre o transporte de mensagens entre agentes usando HTTP. Esses padrões, apesar de estarem se tornando referência, ainda são pouco utilizados por plataformas de agentes móveis.

Considerando ainda somente o paradigma de agentes móveis, temos estudos, como [XAVIER 2002], que trabalham a comparação entre as várias comunidades de agentes móveis que uma gerência pode ter. Este trabalho mostra que todas as comunidades podem alcançar o mesmo objetivo, mas cada uma com características diferentes de obtenção desse objetivo. Esse trabalho é implementado em cima de uma plataforma de agentes móveis, o KDEMA [KDEMA].

Podemos utilizar inteligência computacional em situações que envolvam tomadas de decisão. Pesquisas que utilizam inteligência computacional em agentes móveis também podem ser encontradas, como em [KARNOUSKOS 2002], onde uma combinação *neuro-fuzzy* é utilizada para propor um sistema de mensagens inteligente e dinâmico, adaptável às condições do usuário. Em [DAS 2002] temos inteligência computacional atuando na distribuição de tarefas a agentes móveis para resolução de pesquisas complexas em bases distribuídas. As métricas analisadas foram “tempo” e “tamanho” das pesquisas. Ele apresenta três configurações de pesquisa: um agente; múltiplos agentes ou pesquisa convencional, sem agentes. Percebemos que o trabalho precisa implementar as configurações para obter decisões, ou seja, não consegue antecipar sobre a decisão mais adequada em tempo de projeto. Existem vários outros trabalhos que pesquisam a utilização de inteligência computacional nos agentes móveis, tais como [MAGEDANZ 1996], [WHITE 1998] e [HURST 1997].

Nosso foco nesse trabalho é investigar as comunidades de agentes móveis, tentando apontar sem implementações prévias, utilizando um modelo teórico de avaliação, a mais adequada dada uma gerência. Neste trabalho, aplicamos a inteligência computacional como uma ferramenta de auxílio ao gerente humano na sua escolha da melhor comunidade de agentes móveis.

1.2 Objetivos gerais e específicos

Temos como objetivo geral deste trabalho:

- Propor um modelo para automatização do uso de agentes móveis na gerência de redes de computadores que aponta, dentre as comunidades consideradas, a mais adequada; essa escolha se faz em função de uma métrica considerada (consumo de banda), e através de regras de decisão que tentam representar o raciocínio do gerente humano.

E como objetivos específicos:

- Desenvolver uma plataforma para implementação de agentes móveis;
- Propor e validar um modelo que calcule a quantidade de bytes envolvidos em cada uma das implementações nessa plataforma;
- Propor e validar um modelo de classificação, dadas algumas comunidades de agentes móveis, com relação a alguma métrica pré-definida;
- Propor e validar um modelo de decisão que apresente a comunidade mais adequada, dada uma gerência necessária;
- Utilizar inteligência computacional para auxiliar os processos de decisão dos modelos citados.

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma: nos capítulos 2 a 4 apresentamos uma revisão teórica dos conceitos e paradigmas necessários para o completo entendimento do trabalho. No capítulo 5 apresentamos as comunidades consideradas para os experimentos. No capítulo 6 propomos e validamos um modelo matemático para cálculo de consumo de banda, dada uma comunidade. Já no capítulo 7 apresentamos um modelo matemático de classificação das comunidades baseado em lógica *fuzzy* e o implementamos por redes neurais. No capítulo 8 propomos e validamos o modelo de

decisão, através de regras de inferência. Por fim, nos capítulos seguintes apresentamos as conclusões, discussão do trabalho, propostas de trabalhos futuros e alguns anexos convenientes.

2. Gerência de redes

Como pudemos perceber na Introdução, a gerência de redes é o contexto geral do trabalho. Apresentaremos em seguida uma série de conceitos acerca de gerência de redes interessantes para o bom entendimento do trabalho. Explicitamos o conceito de gerência automatizada, grande alvo de nossos modelos propostos.

2.1 Introdução

Com o advento dos computadores e sua conseqüente interligação para compartilhamento de recursos, uma nova vertente da computação foi concebida: as redes de computadores. Uma rede de computadores pode ser formada por vários elementos responsáveis pelo tráfego das informações compartilhadas, tais como hubs, switchs, roteadores, bridges, os enlaces, além dos próprios computadores. Com isso surgiu uma nova infra-estrutura, que acabou se tornando complexa, dada a variedade de equipamentos, além de uma disposição física distribuída (os equipamentos de uma mesma rede podem estar próximos ou a centenas de quilômetros de distância). Essa complexidade teve como conseqüência natural a necessidade de uma gerência desses recursos – a gerência de redes.

A necessidade da gerência de redes está intimamente ligada ao resultado dos trabalhos que utilizam essa estrutura, visto que, se algo estiver com problemas, o resultado esperado poderá não ser obtido.

A gerência de redes pode ser tão simples quanto criar um disco de inicialização para um novo usuário, tendo a certeza que ele acessará a rede de modo desejado. Mas pode ser mais complicada, como, por exemplo, ter uma tarefa diária de fazer backup dos arquivos de uma rede ou desfragmentar discos. Pode-se ainda descobrir por que alguns usuários estão com problemas de performance ao usar recursos da rede ou a necessidade de reconfiguração de um dispositivo remoto fisicamente a quilômetros de distância.

Em resumo, a gerência de redes incorpora uma lista enorme de tarefas que têm como objetivo comum fazer com que a rede esteja sempre funcionando (disponibilidade) com a máxima performance possível.

Mas essa tarefa se torna mais complexa a medida em que as redes passam a ser maiores e mais complexas. O surgimento de várias pequenas redes, que interconectadas

formam uma grande rede, formada sempre por vários tipos de dispositivos de fabricantes diversos (impressoras, computadores, hubs, routers e outros) possibilita uma infinidade de possibilidades aos usuários, mas agrega complexidade na tarefa de gerenciar essa estrutura. Alguns fabricantes desenvolvem ferramentas – baseadas ou não em software – que apresentam um mapa dos recursos da rede na tentativa de auxiliar o gerente em sua tarefa.

Desde a descoberta de um rompimento de cabo ou um dispositivo desligado até alarmes mais complexos, as ferramentas utilizadas para gerência de redes são vitais para que o gerente possa manter as características de funcionalidade desta rede.

2.2 Áreas funcionais da gerência

De uma forma geral, proposta em muitas documentações derivadas de [ISO-OSI 10165], a gerência de redes pode ser dividida em cinco áreas funcionais: configuração, tolerância à falhas, segurança, desempenho e contabilidade.

2.2.1 Gerência de configuração

A gerência de configuração é responsável por instalar, iniciar, modificar e varrer os parâmetros ou opções de configuração de hardwares e softwares de rede.

2.2.2 Gerência de tolerância a falhas

Possui uma visão histórica de erros de rede e características de alarmes ocorridos. Essa gerência apresenta ao gerente números, tipos, locais e tempo em que ocorreram falhas na rede.

2.2.3 Gerência de segurança

Permite ao gerente de redes controlar acessos aos recursos compartilhados, desde arquivos ou aplicações, até a própria rede como um todo. Implementa esquemas protegidos por senhas que controla o acesso diferenciado a usuários, conforme a característica de cada um. A gerência de segurança é também importante para a própria gerência de rede como um todo, já que controla os acessos a configurações em um servidor ou outros dispositivos de rede.

2.2.4 Gerência de desempenho

Produz estatística histórica e instantânea sobre a operação da rede: como os pacotes estão sendo transmitidos naquele momento; o número de usuários ativos em um

servidor ou compartilhamento específico; a utilização de determinado ramo de comunicação. Essas informações possibilitam ao gerente de rede detectar segmentos de rede que possuem situações de problema em potencial (pró-atividade).

O gerenciamento de desempenho geralmente permite investigar individualmente dispositivos de rede por informações específicas: quantos arquivos estão em uso ou a quantidade de usuários ativos em determinado servidor de arquivos; ou quantas tarefas de impressão estão na fila de um servidor de arquivos; ou quantos pacotes estão trafegando em um enlace. Isso é subsidio para identificar necessidades de expansão de recursos.

2.2.5 Gerência de contabilidade

Utilizada para verificar a utilização de recursos, no sentido de estar medindo custos gastos por determinados usuários ou departamentos. Informa, por exemplo, tempos de atividade/inatividade de usuários ou bytes transmitidos ou recebidos por determinado nó.

2.3 Tipos de gerência

2.3.1 Gerência reativa

A gerência reativa é uma espécie de gerência passiva, onde os problemas acontecem para, em seguida, serem identificados e solucionados. É geralmente feita por reclamações, tendo como conseqüência natural a queda de performance e produtividade do ambiente.

2.3.2 Gerência pró-ativa

A gerência pró-ativa, em contraste com a reativa, é uma gerência em que as tendências de problemas são identificadas, com uma antecedência tal que medidas de profilaxia podem ser tomadas antes que qualquer problema real aconteça, garantindo a produtividade do ambiente.

É auxiliada por valores-padrão (*baselines*), a partir do qual a gerência pró-ativa ativa os respectivos alarmes. A obtenção dos valores-padrão pode ser feita de duas formas:

- Através de um estudo prévio do ambiente a ser gerenciado, com medições dos valores para o funcionamento dessa rede considerado normal, e em seguida, através de métodos estatísticos e matemáticos, definições dos

valores-padrão, que passam a compor a gerência. E normalmente aplicado a ambientes pouco dinâmicos, ou seja, que não prevêem alterações significativas nas características do ambiente;

- Através do método anterior, mas com a reavaliação ativa dos valores-padrão, que são automaticamente atualizados para novas realidades do ambiente. Pode-se nesse caso, utilizar, por exemplo, recálculos solicitados pelo gerente, ou em períodos fixos de tempo, ou mesmo, considerando a pró-atividade, quando for detectada automaticamente alguma característica do ambiente que interfira nesses cálculos.

2.3.3 Gerência automatizada

A gerência automatizada é aquela que possui recursos de iniciativa automática, agindo no lugar do gerente. No caso da gerência reativa automática, essas iniciativas serão corretivas, mas automáticas. No caso da gerência pró-ativa automática, as iniciativas serão preventivas ou corretivas, e automáticas.

A gerência automatizada envolve outro elemento na gerência: capacidade de ação, decidindo a melhor atitude a ser tomada conforme a situação encontrada. Mas para se tomar atitudes, é inevitável pensar em inteligência. Na verdade, ao ser citada a inteligência, o objetivo é mostrar que deve existir algo mais que regras simples de inferência, baseadas em uma atitude por alarme ocorrido.

Por exemplo, suponha que a gerência detectou que, em um ambiente com dois servidores redundantes, um servidor está com o serviço HTTP lento e que o outro com esse mesmo serviço muito pesado. Suponha ainda que exista um alarme para baixo tráfego no barramento do servidor lento. Seriam gerados alarmes para o gerente humano, que teria que interpretar a ocorrência simultânea dos erros para entender que a saída do barramento no *backbone* está com problemas. Essa interpretação, feita pelo gerente humano, é na verdade a utilização simultânea de várias formas de raciocínio, levando em consideração regras de inferência que se auto-ponderam, além de conhecimentos anteriores.

Na gerência automática, então, além das ações propriamente ditas, existe a base de conhecimentos (regras de circunstâncias e ações), que utilizam vários paradigmas computacionais de simulação de inteligência para interpretar da melhor maneira possível o contexto.

2.4 Conclusão

Percebemos a abrangência e a importância da gerência de redes de computadores no contexto atual. Os conceitos de gerência pró-ativa e automatizada são altamente desejáveis em ambientes atuais como a Internet. Destacamos a gerência de contabilidade aliada à gerência automatizada como grandes alvos de nosso trabalho.

3. Inteligência computacional

Nos objetivos propostos para esse trabalho, verificamos a necessidade da utilização de inteligência computacional para resolução de problemas, simulando decisões de um gerente humano.

Para alcançarmos esses objetivos, precisamos utilizar vários paradigmas de inteligência computacional, seja na fase de modelagem ou na fase de implementação dos modelos teóricos propostos.

Apresentamos a seguir conceitos de inteligência computacional, necessários a compreensão de sua utilização nos modelos propostos. Damos especial ênfase às abordagens por lógica *fuzzy* e redes neurais, que são apresentadas com maiores detalhes.

3.1 Introdução

Inteligência computacional é um paradigma de computação que apresenta a habilidade de aprendizado ou de lidar com novas situações, em que o sistema é provido de habilidades de reação, como generalização, descobrimento, associação e abstração. Sistemas de inteligência computacional geralmente são híbridos de paradigmas como redes neurais artificiais, lógica *fuzzy* e computação evolucionária, mesclados com elementos de conhecimento. Podemos encontrar vários exemplos de sistemas híbridos em [BARRETO 1999], expostos de uma forma bastante acessível. Podemos ainda encontrar explicações detalhadas sobre vários paradigmas de inteligência computacional em [EBERHART 1996]. Utilizamos as duas referências citadas para expor, de forma resumida, as idéias que seguem.

Uma rede neural artificial é um paradigma de análise modelado à semelhança da estrutura de neurônios do cérebro. Ele simula uma estrutura computacional paralela, altamente interconectada, formada por vários elementos de processamentos relativamente simples.

Lógica *fuzzy* é a lógica das reações aproximadas. A lógica *fuzzy* trabalha em cima de operações como igualdade, contido, complemento, interseção e união; é uma generalização da lógica convencional (dois valores).

Computação evolucionária compreende máquinas aprendendo paradigmas de otimização e classificação baseados em mecanismos de evolução como genética biológica e seleção natural. O campo da computação evolucionária inclui algoritmos

genéticos, programação evolucionária, programação genética e estratégias evolucionárias.

3.2 Abordagens da inteligência computacional

A inteligência computacional, apesar de seu objetivo comum, que é apresentar soluções para problemas propostos, difere muito conforme a abordagem adotada. Essa diferença é apresentada em todos os momentos como implementação, forma de armazenamento, manipulação e apresentação de dados e conhecimentos. Dentre as abordagens, temos: inteligência artificial simbólica (IAS), conexionista (IAC), evolutiva (IAE) e híbrida (IAH), que é uma apresentação de solução envolvendo simultaneamente mais de uma abordagem.

3.2.1 Inteligência artificial simbólica (IAS)

A inteligência artificial simbólica utiliza-se de representação de conhecimentos, através de símbolos e manipulação dos mesmos através de regras de inferência, para chegar à solução de um problema. O alfabeto (conjunto de símbolos representativos possíveis) deve poder representar os conhecimentos do problema, que através de submissão às regras de inferência (transformação de uma situação simbólica anterior em uma posterior) deve apresentar uma solução adequada. As regras de inferência são conhecimentos especializados do problema, transcritos por um especialista, que possuem condições ou não para sua inferência, tendo como entrada determinado padrão de ocorrências de símbolos, e como saída, outro padrão de ocorrência de símbolos, que passará a ser entrada para uma próxima inferência.

Vale ressaltar que a IAS apresenta formas de análise de resultados intermediários, necessário para se analisar o caminho do raciocínio simulado pelas regras de inferência. Isso pode ser necessário quando o cliente do resultado precisar saber como se deu o raciocínio. Para isso basta apresentar as inferências intermediárias, com suas entradas, regra aplicada e saída obtida.

3.2.2 Inteligência artificial conexionista (IAC)

A inteligência artificial conexionista é baseada em elementos simples de processamento (*neurons*) e conexões entre eles, formando uma grande rede paralela de processamento. Essas conexões possuem valores independentes de intensidade, chamados pesos. Cada rede possui caminhos de entrada (excitação da rede), caminhos

intermediários (conexões entre os *neurons*) e caminhos de saída. Ao se aplicar valores nas entradas, estas excitam os *neurons* de entrada que passam a excitar outros *neurons*, através das conexões. O grau de excitação ao *neuron* subseqüente varia conforme peso da conexão entre o *neuron* excitador e o excitado.

É aplicada a problemas mal definidos, onde falta o conhecimento claro de como fazer, impossibilitando a construção de regras de conhecimento direcionadoras do raciocínio. Aplicada também onde a quantidade de regras de inferência é alta, tornando a computabilidade do problema por IAS muito custosa. Apesar de sua vasta gama de aplicações, deve-se evitar sua utilização em situações que necessitam de representação de resultados intermediários, ou caminho do raciocínio, pois a IAC não possui regras específicas e sim um processamento paralelo de valores, chegando imediatamente ao resultado.

3.2.3 Inteligência artificial evolucionária (IAE)

A inteligência artificial evolutiva é inspirada em regras de sobrevivência de populações. Considerando que indivíduos possuem propriedades (similares a cromossomos dos seres vivos), podemos colocar um problema como sendo a regra para sobrevivência dos indivíduos. A habilidade que cada indivíduo tem com relação à regra é chamada aptidão. O algoritmo evolucionário básico é:

a) Inicie a população de indivíduos;
b) Calcule a aptidão de cada indivíduo;
c) Reproduza a população, gerando novos descendentes;
d) Aplique regras de evolução como: mutações ou mortes;
e) Enumere sobreviventes;
f) Volte ao passo b).

As mutações consistem em alterações aleatórias das características dos indivíduos. As mortes consistem em eliminar indivíduos com menor aptidão.

Os algoritmos evolucionários são adequados para se evitar resoluções tendenciosas, causadas por máximos ou mínimos locais, que podem apresentar soluções satisfatórias, mas não a mais satisfatória.

3.2.4 Inteligência artificial híbrida (IAH)

A inteligência artificial híbrida contempla soluções com duas ou mais abordagens sendo utilizadas à mesma resolução de problema. Temos, por exemplo, soluções *fuzzy* que podem utilizar IAS e IAC ou, por exemplo, soluções para treinamento de redes neurais, utilizando IAC e IAE.

3.3 Redes neurais artificiais

As redes neurais artificiais consistem em um método de solucionar problemas de inteligência artificial, construindo um sistema que tenha circuitos que simulem o cérebro humano, inclusive seu comportamento, ou seja, aprendendo, errando e fazendo descobertas. São, mais que isso, técnicas computacionais que apresentam um modelo inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência

3.3.1 O Neurônio Artificial

Assim como o sistema nervoso é composto por bilhões de células nervosas, a rede neural artificial também seria formada por unidades que nada mais são que pequenos módulos que simulam o funcionamento de um neurônio. Estes módulos devem funcionar de acordo com os elementos em que foram inspirados, recebendo e retransmitindo informações.

3.3.2 A Rede Neural Artificial

A rede neural artificial é um sistema de neurônios (*perceptrons*) ligados por conexões sinápticas,divididos em neurônios de entrada, que recebem estímulos do meio externo, neurônios internos ou *hidden* (ocultos) e neurônios de saída, que se comunicam com o exterior. As formas de arranjo dos neurônios em camadas é denominado *multilayer perceptron*. O *multilayer perceptron* foi concebido para resolver problemas mais complexos, os quais não poderiam ser resolvidos pelo modelo de neurônio básico. Um único *perceptron* ou uma combinação das saídas de alguns *perceptrons* poderiam realizar uma operação XOR, porém, seria incapaz de aprendê-la. Para isto são necessárias mais conexões, os quais só existem em uma rede de *perceptrons* dispostos em camadas. Os neurônios internos são de suma importância na rede neural, pois se provou que sem estes se torna impossível a resolução de problemas linearmente não separáveis. Em outras palavras pode-se dizer que uma rede é composta por várias

unidades de processamento, cujo funcionamento é bastante simples. Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma rede neural artificial vem das interações entre as unidades de processamento da rede.

A maioria dos modelos de redes neurais possui alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados. Em outras palavras, elas aprendem através de exemplos. Arquiteturas neurais são tipicamente organizadas em camadas, com unidades que podem estar conectadas às unidades da camada posterior.

A rede neural passa por um processo de treinamento a partir dos casos reais conhecidos, adquirindo, a partir daí, a sistemática necessária para executar adequadamente o processo desejado dos dados fornecidos. Sendo assim, a rede neural é capaz de extrair regras básicas a partir de dados reais, diferindo da computação programada, onde é necessário um conjunto de regras rígidas pré-fixadas e algoritmos.

Usualmente as camadas são classificadas em três grupos:

- Camada de Entrada: onde os padrões são apresentados à rede;
- Camadas Intermediárias ou Ocultas: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características;
- Camada de Saída: onde o resultado final é concluído e apresentado.

Redes neurais são também classificadas de acordo com a arquitetura em que foram implementadas, topologias, características de seus nós, regras de treinamento, e tipos de modelos.

3.3.3 Processos de Aprendizado de uma Rede Neural Artificial

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e com isso melhorar seu desempenho. Isso é feito através de um processo iterativo de ajustes aplicado a seus pesos, o treinamento. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de

aprendizado específicos para determinados modelos de redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados.

A rede neural se baseia nos dados para extrair um modelo geral. Portanto, a fase de aprendizado deve ser rigorosa e verdadeira, a fim de se evitar modelos espúrios. Todo o conhecimento de uma rede neural está armazenado nas sinapses, ou seja, nos pesos atribuídos às conexões entre os neurônios. De 50 a 90% do total de dados deve ser separado para o treinamento da rede neural, dados estes escolhidos aleatoriamente, a fim de que a rede "aprenda" as regras e não "decore" exemplos. O restante dos dados só é apresentado à rede neural na fase de testes a fim de que ela possa "deduzir" corretamente o inter-relacionamento entre os dados.

3.3.4 Desenvolvimento de Aplicações

Ilustra os passos necessários para o desenvolvimento de aplicações utilizando redes neurais artificiais:

3.3.4.1 Coleta de dados e separação em conjuntos

Os dois primeiros passos do processo de desenvolvimento de redes neurais artificiais são a coleta de dados relativos ao problema e a sua separação em um conjunto de treinamento e um conjunto de testes. Esta tarefa requer uma análise cuidadosa sobre o problema para minimizar ambigüidades e erros nos dados. Além disso, os dados coletados devem ser significativos e cobrir amplamente o domínio do problema; não devem cobrir apenas as operações normais ou rotineiras, mas também as exceções e as condições nos limites do domínio do problema.

Normalmente, os dados coletados são separados em duas categorias: dados de treinamento, que serão utilizados para o treinamento da rede e dados de teste, que serão utilizados para verificar sua performance sob condições reais de utilização. Além dessa divisão, pode-se usar também uma subdivisão do conjunto de treinamento, criando um conjunto de validação, utilizado para verificar a eficiência da rede quanto a sua capacidade de generalização durante o treinamento, e podendo ser empregado como critério de parada do treinamento.

Depois de determinados estes conjuntos, eles são geralmente colocados em ordem aleatória para prevenção de tendências associadas à ordem de apresentação dos dados. Além disso, pode ser necessário pré-processar estes dados, através de normalizações,

escalonamentos e conversões de formato para torná-los mais apropriados à sua utilização na rede.

3.3.4.2 Configuração da rede

O terceiro passo é a definição da configuração da rede, que pode ser dividido em três etapas:

- Seleção do paradigma neural apropriado à aplicação;
- Determinação da topologia da rede a ser utilizada - o número de camadas, o número de unidades em cada camada, etc;
- Determinação de parâmetros do algoritmo de treinamento e funções de ativação. Este passo tem um grande impacto na performance do sistema resultante.

Existem metodologias, “dicas” e “truques” na condução destas tarefas. Normalmente estas escolhas são feitas de forma empírica. A definição da configuração de redes neurais é ainda considerada uma arte, que requer grande experiência dos projetistas.

3.3.4.3 Treinamento

O quarto passo é o treinamento da rede. Nesta fase, seguindo o algoritmo de treinamento escolhido, serão ajustados os pesos das conexões. É importante considerar, nesta fase, alguns aspectos tais como a inicialização da rede, o modo de treinamento e o tempo de treinamento.

Uma boa escolha dos valores iniciais dos pesos da rede pode diminuir o tempo necessário para o treinamento. Normalmente, os valores iniciais dos pesos da rede são números aleatórios uniformemente distribuídos, em um intervalo definido. A escolha errada destes pesos pode levar a uma saturação prematura.

O treinamento deve ser interrompido quando a rede apresentar uma boa capacidade de generalização e quando a taxa de erro for suficientemente pequena, ou seja, menor que um erro admissível. Assim, deve-se encontrar um ponto ótimo de parada com erro mínimo e capacidade de generalização máxima.

3.3.4.4 Teste

O quinto passo é o teste da rede. Durante esta fase o conjunto de teste é utilizado para determinar a performance da rede com dados que não foram previamente utilizados. A performance da rede, medida nesta fase, é uma boa indicação de sua performance real.

Devem ser considerados ainda outros testes como análise do comportamento da rede utilizando entradas especiais e análise dos pesos atuais da rede, pois se existirem valores muito pequenos, as conexões associadas podem ser consideradas insignificantes e assim serem eliminadas. De modo inverso, valores substantivamente maiores que os outros poderiam indicar que houve *over-training* da rede.

3.3.4.5 Integração

Finalmente, com a rede treinada e avaliada, ela pode ser integrada em um sistema do ambiente operacional da aplicação. Para maior eficiência da solução, este sistema deverá conter facilidades de utilização como interface conveniente e facilidades de aquisição de dados através de planilhas eletrônicas, interfaces com unidades de processamento de sinais, ou arquivos padronizados. Uma boa documentação do sistema e o treinamento de usuários são necessários para o sucesso do mesmo.

Além disso, o sistema deve periodicamente monitorar sua performance e fazer a manutenção da rede quando for necessário ou indicar aos projetistas a necessidade de retreinamento. Outras melhorias poderão ainda ser sugeridas quando os usuários forem se tornando mais familiares com o sistema; estas sugestões poderão ser muito úteis em novas versões ou em novos produtos.

3.4 Lógica fuzzy

Lógica *fuzzy* é a aplicação da Teoria dos *Fuzzy Sets* (FST) (Conjuntos Difusos ou Nebulosos) aos princípios da Lógica Clássica. Tradicionalmente, a lógica é formulada matematicamente em termos da dualidade falso/verdadeiro também conhecida como lógica bivalente ou Aristotélica. A aplicação da FST altera a fundamentação da lógica bivalente por considerar o conceito de verdade parcial. Isto porque ser um membro de um *fuzzy* set é uma questão de grau ou gradação, portanto a veracidade ou falsidade de uma observação também é. Dessa forma, uma declaração do tipo “se o tempo é bom” se torna parcialmente verdadeira na lógica *fuzzy*, o que na maioria das situações espelha melhor a realidade do que a lógica bivalente.

3.4.1 Raciocínio *Fuzzy*

Se o clima estiver “frio” nós usaremos roupas grossas. Se estiver “muito frio”, usaremos também um cachecol. Não é necessário, e é algumas vezes prejudicial, esperarmos que a temperatura caia abaixo de exatamente 70^{o}C para usar um agasalho. Considerando outras decisões, tais quais dirigir um automóvel, votar em um candidato, ajustar a água quente do chuveiro, escolher a melhor maçã ou comprar uma TV, pode-se concluir que a maioria de nossas decisões é mais baseada em raciocínio aproximado do que em relações exatas.

Na lógica *fuzzy*, as condições são compostas usando proposições universalmente compreensíveis (funções de pertinência), as quais são definidas por heurísticas ou funções de crença. Dessa maneira, a informação *crisp* (tal qual R$10 é muito caro para o estudante ou muito barato para um empresário bem sucedido) não se perde, mas é mapeada em uma função de crença (muito caro). Esta representação transforma a lógica bivalente (Falso-Verdadeiro) em uma forma multivalente, na qual as possíveis combinações de soluções são abundantes.

A lógica *fuzzy* pode não ser apropriada para todos os problemas, especialmente quando uma tolerância na precisão não é admitida. Se um ovo de crocodilo deve ser mantido à temperatura de 36°C para resultar em uma fêmea, e se um grau centígrado a mais causa mudança de sexo, então a precisão requerida para uma incubadora pode ser muito estreita para a aplicação prática da lógica *fuzzy* no controle desse dispositivo. Note-se que a inadequação nesse caso não é causada pela natureza aproximada das soluções *fuzzy*, mas pela natureza do problema, que não comporta a lógica multivalente.

A distinção citada é importante porque a inferência *fuzzy* é um método geral, que também inclui a obtenção de soluções com base na lógica *crisp*.

3.4.2 Conjuntos *Crisp* versus *Fuzzy*

Considere uma tarefa simples de classificação. Suponha que há interesse em formar um conjunto de objetos identificados como cadeiras, em uma sala onde estão seis objetos (três cadeiras, uma mesa, uma mesinha e um abajur). De acordo com a teoria clássica de conjuntos, um objeto é ou não é uma cadeira — lógica bivalente: verdadeiro ou falso. Usando-se a convenção 0 e 1, o conjunto de objetos identificados como cadeiras é representado como {1,1,1,0,0,0}, também chamado de *crisp set*.

Considere-se um problema similar. Suponha que se deseje classificar objetos que possam funcionar como uma cadeira. Dos seis objetos, apenas o abajur seria uma escolha ruim. Além das três cadeiras, outros objetos poderiam ser usados com essa função. Quando a possibilidade entra em cena, pode-se formar um conjunto *fuzzy*, p.ex. {1,1,1,0.6,0.3,0}, o qual contém elementos que refletem a possibilidade de um objeto ser membro do conjunto.

Para o exemplo, selecionou-se 0.6 para a mesa e 0.3 para a mesinha. O valor 0.6 significa que a mesa tem 60% de possibilidade de funcionar como uma cadeira. Esta representação claramente amplia a abrangência da lógica bivalente, tornando-a multivalente. O elemento básico da lógica *fuzzy*, que é o conceito de função de pertinência, é baseado na representação de conjuntos *fuzzy*.

3.4.3 Variáveis *Fuzzy*

Considere uma variável contínua X tal qual definida no Cálculo. Assuma que X pode tomar qualquer valor entre 1 e 10. Não há qualquer outra informação agregada à variável X usando-se esta representação. Por exemplo, se X denota a temperatura de uma sala, então a informação "X = 30^{o}C" significa somente que a temperatura da sala é 30^{o}C.

Por outro lado, considere a abordagem heurística na qual o universo U (intervalo de valores possíveis) de X seja dividido em diversas regiões, cada qual definida por um termo interpretável. Suponha que os intervalos 1-5, 3-8 e 7-10 sejam interpretados como as regiões de temperatura "baixa", "média", e "alta", respectivamente. Então, pode-se redefinir X como uma variável *fuzzy* que assume os valores lingüísticos: "baixa", "média", e "alta". Como se infere, as qualificações da temperatura são definições aproximadas, diferentes de valores numéricos exatos.

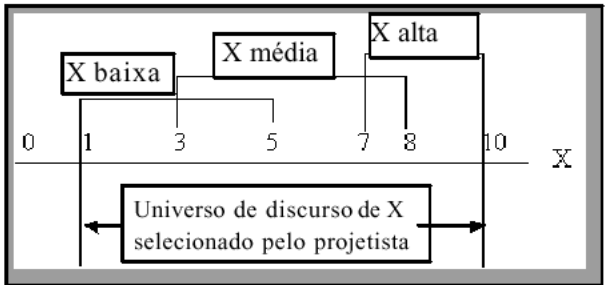

Figura 1. Definição de variáveis *Fuzzy*

A representação *fuzzy* contém mais informação sobre uma variável do que a representação numérica tradicional. Usando-se o exemplo anterior, "X=baixa" significa que a temperatura está entre 1 a 5^{o}C, e é baixa com relação a outros valores que ela pode assumir. Mais importante, a variável torna-se sensível a uma faixa de valores, em vez de a um número exato, tal qual X=3. Esta representação é conhecida por reproduzir o raciocínio humano de uma forma mais fidedigna do que se usando variáveis numéricas *crisp*.

3.4.4 Funções de pertinência

Como é definida ”baixa" no intervalo 1-5? A definição de ”baixa" ou de qualquer outro termo lingüístico depende somente do julgamento do interessado. Na lógica *fuzzy*, tal julgamento é formulado por meio de uma função de distribuição de possibilidade, geralmente com domínio em [0,1], e que é chamada de função de pertinência.

No intervalo 1-5 do exemplo, a função de pertinência ”baixa" deve assumir o valor 1 para um certo ponto de U que represente ”baixa" da melhor maneira.

3.4.5 Tipos de variáveis *Fuzzy*

A discussão a seguir abrange dois tipos de variáveis *fuzzy*:

- Variáveis de entrada (*Input*);
- Variáveis de saída ou ação (*output*).

As variáveis de entrada e de saída são também chamadas de antecedentes e conseqüentes, respectivamente, nos textos de lógica *fuzzy*. Todas as variáveis de entrada devem ser usadas no lado esquerdo de uma regra, e as de saída no lado direito. Se uma variável de saída for usada em ambos os lados (em regras diferentes), o uso no lado esquerdo significará simplesmente a repetição do antecedente original.

Por exemplo:

- SE o fluxo é alto ENTÃO a válvula está aberta (Válvula é uma variável de saída)
- SE a válvula está aberta ENTÃO a Potência é alta (Válvula é também uma variável de entrada)

As duas regras podem ser expressas em uma forma alternativa:

- SE o fluxo é alto ENTÃO a válvula está aberta
- SE o fluxo é alto ENTÃO a Potência é alta

Os dois conjuntos de regras mostrados são equivalentes.

3.4.6 Composição de regras

A composição de regras refere-se à organização das declarações do lado esquerdo usando operadores lógicos tais quais OU e E. Por exemplo, examine-se a seguinte regra *fuzzy*:

- SE X é baixo E Y é alto ENTÃO Z é médio

Aqui, as declarações à esquerda (X=baixo, e Y=alto) são unidas pelo operador E. Suponha que as funções de pertinência para X=baixo e Y=alto foram projetadas como triângulos. Quando as entradas são X=1.5 e Y=8.5, os valores correspondentes das funções de pertinência são encontradas da seguinte forma:

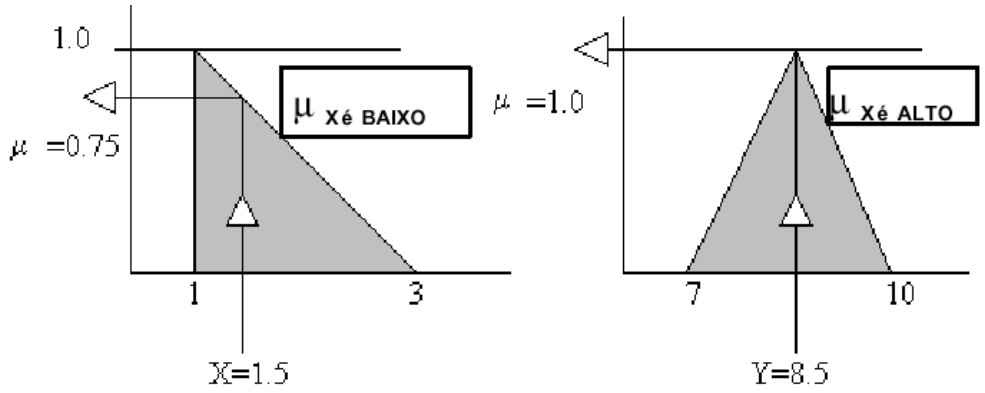

Figura 2. Exemplos de avaliação da função de pertinência

Os valores de X=1.5, e Y=8.5 geram 75% e 100% de possibilidade de X ser “baixo” e de Y ser “alto”, respectivamente. O operador E usa os valores de m =0.75 *and* m =1.0 para computar o resultado da composição (E gera o mínimo=0.75). Esta saída (valor DOF) será usada no processo de implicação pelo operador “então”. m X é BAIXO m X é ALTO

3.5 Áreas de aplicação de inteligência computacional

Apresentamos a seguir algumas áreas de aplicação de inteligência computacional, divididas entre os paradigmas apresentados.

3.5.1 Redes neurais

Existem cinco áreas para as quais os resultados de utilização de redes neurais são adequados:

- Classificação como refletida na teoria de decisão, onde uma série pré-definida de padrões reflete o padrão de entrada;
- Memória associativa, onde, para determinado padrão incompleto de entrada, a rede neural apresenta a versão completa em sua saída;
- Compressão, onde para determinado padrão de entrada, a rede apresenta um padrão de saída com menor quantidade de bits, de forma que a perda de resolução do padrão de entrada seja aceitável;
- Capacidade de generalização de estruturas de seqüências ou padrões por uma rede treinada com exemplos;
- Resolvem muito bem o problema de não-linearidade do comportamento de sistemas (sistemas que não podem ser modelados em comportamentos lineares). Além disso, o tempo de desenvolvimento de uma rede neural para sistemas de controle é tipicamente muito mais rápido que em outras técnicas tradicionais.

O número de aplicações específicas para casa uma das cinco áreas citadas aumenta a cada dia. Entre várias possibilidades, estão algumas:

- Classificação de voz;
- Composição de música;
- Detecção de explosivos em aeroportos;
- Reconhecimento de voz;
- Detecção de falhas em equipamentos;
- Conversão de texto para voz
- Reconhecimento e procura de alvos militares; e
- Processamento de imagens;

3.5.2 Computação evolucionária

Algoritmo genético é o paradigma mais comumente usado na computação evolucionária. As duas áreas principais de utilização dos algoritmos genéticos são otimização e classificação. Algumas aplicações estão listadas abaixo:

- Diagnóstico de múltiplas falhas;
- Determinação de melhor caminho;
- Problemas de otimização.

Em alguns casos, outros paradigmas de computação evolucionária podem ser usados.

3.5.3 Lógica *fuzzy*

Lógica *fuzzy* é aplicada em uma grande gama de situações em áreas de engenharia. Outras áreas de aplicação incluem medicina, gerenciamento, análise de decisão e ciência de computação. Como em redes neurais, várias aplicações surgem a cada dia. Duas das maiores áreas de aplicação são controles *fuzzy* e sistemas especialistas *fuzzy*.

Aplicações em controle *fuzzy* incluem:

- Sistemas de controle de trânsito;
- Controle de trens e metrôs;
- Aplicações domésticas;
- Câmeras de vídeo;
- Controles automobilísticos (transmissão, freio, etc).

Exemplos de sistemas especialistas *fuzzy* incluem:

- Diagnósticos médicos;
- Navegação automatizada;
- Agendamentos;
- Diagnósticos automobilísticos;
- Seleção de estratégias de negócios.

3.6 A inteligência computacional e a gerência de redes

Como observado anteriormente, qualquer dos paradigmas de inteligência computacional possui suas áreas ótimas de aplicação, para resolução de problemas. Existem, inclusive, problemas cuja natureza pode ser tratada por mais de uma solução de inteligência computacional, de uma forma híbrida ou como soluções individuais.

Considerando a existência de um ambiente computacional provido de estruturas genéricas de inteligência computacional, onde pudéssemos entrar com problemas obtendo soluções, temos:

- Ao se contemplar todos os paradigmas de inteligência computacional para resolução de problemas, podemos ter como resultado um ambiente pesado do ponto de vista de código, necessitando de muitos recursos para seu processamento; além disso, esse ambiente será complexo, dadas todas as variações de utilização;
- De outro lado, ao se contemplar apenas um paradigma nesse ambiente computacional, teríamos um código mais leve, mas em contrapartida, teríamos problemas cuja resolução estaria prejudicada, por não ser o paradigma adequado.

Focando mais a gerência de redes, encontramos problemas que estão classificados como aplicações em todos os paradigmas de inteligência computacional, de forma que deve existir uma avaliação prévia, para melhor definir os paradigmas de inteligência computacional a serem utilizados.

3.7 Conclusão

Percebemos que as abordagens de inteligência computacional têm, cada qual, suas utilizações ótimas. Precisamos avaliar a abordagem mais adequada para cada uma das situações que sugerem a utilização de inteligência computacional, de acordo com vários fatores tais como disponibilidade de “como fazer”, dos dados de entrada e saída, necessidade de resultados intermediários, entre outros.

Na proposta ou implementação dos modelos (objetivo específico deste trabalho) utilizamos basicamente IAS e IAC, através de regras de inferência, lógica *fuzzy* e redes neurais.

4. Paradigmas de computação

Uma das grandes frentes de pesquisa na área de computação distribuída é a que estuda as suas possibilidades de implementação. Entre os paradigmas existentes citamos: cliente-servidor, avaliação remota, código por demanda e agentes móveis. Cada um possui suas características, com conseqüentes aplicações ótimas, como será exposto em seguida.

4.1 Introdução

Existe uma classificação para os paradigmas, baseada nas características:

- Onde estão os dados da computação (dados);
- Onde a computação é efetivamente feita (recursos); e
- Onde está a lógica responsável pela computação (código).

Essa classificação é importante para posicionar o modelo utilizado no trabalho (agentes móveis) com relação a outros modelos de computação disponíveis, apontando algumas vantagens e desvantagens de cada um. Além disso, esse capítulo apresenta, de uma forma mais detalhada, o paradigma de agentes móveis, mostrando características e conceitos necessários para o entendimento dos capítulos seguintes.

4.2 Cliente-servidor

No modelo cliente-servidor, código, dados e recursos estão do lado do servidor e o cliente do processamento do outro. As solicitações são enviadas pelo cliente ao servidor através de primitivas conhecidas por ambos, e o processamento ocorre no servidor, que consulta dados locais e utiliza recursos locais, e que pode retornar ou não ao cliente uma primitiva com resultados do processamento. Nesse modelo existe uma interação muito grande entre cliente e servidor, principalmente se as primitivas são de granularidade muito fina. Existe ainda a desvantagem de se ter operações do cliente limitadas às primitivas implementadas no servidor; para novas tarefas, deve-se reimplementar o servidor. O cliente deve conhecer todas as primitivas disponíveis pelo servidor; caso contrário estará correndo o risco de exigir processamentos que poderiam ser feitos com primitivas mais adequadas. Em resumo, só trafegam primitivas e, nem sempre, dados do servidor para o cliente.

4.3 Avaliação remota

Na avaliação remota os dados estão do lado do cliente, mas os recursos e o código estão do lado do servidor. Os dados são, então, enviados pelo cliente ao servidor, que executa o processamento solicitado e retorna o resultado ao cliente. Esse modelo apresenta todas as desvantagens do modelo anterior, além de maior consumo de largura de banda do meio de comunicação (os dados sempre são enviados do cliente para o servidor e o resultado retornado).

4.4 Código por demanda

No código por demanda o cliente tem os dados e o recurso para o processamento, mas não sabe como faze-lo. O servidor possui uma serie de conhecimentos (código) à disposição do cliente. O cliente então solicita o código necessário ao seu processamento para o servidor. O código é enviado ao cliente que pode, então, executar o seu processamento local, pois tem os três elementos necessários para tal. Nesse modelo existem várias características interessantes tais como:

- O cliente possui mais poder de processamento para sua finalidade, já que, quando precisa de código para determinada tarefa, pode descartar outros que possui e solicitar o adequado ao novo processamento ao servidor;
- São admitidos clientes com menor poder de processamento, pelos mesmos motivos acima citados;
- Quando da alteração de código para determinada tarefa, não existe a necessidade de atualização nos clientes, pois quando a próxima solicitação dessa tarefa for efetuada, o novo código virá atualizado;
- Só trafegam códigos, o que acaba gerando pouca interação do cliente com o servidor, propício para operações desconectadas ou ambientes de pouca banda passante ou alta latência.

4.5 Agentes móveis

Nesse modelo existe a migração de um código, com todo o seu contexto (estados), para um cliente. Existe então o processamento, que utiliza os recursos e alguns dados do cliente hospedeiro, em adição ao código hospedado e seu contexto. Ao final desse processamento, o código pode ou não migrar para outro cliente, levando consigo

resultados e contexto do processamento efetuado anteriormente. Ao final de todas as visitas o código pode retornar ao servidor com os resultados colhidos em sua jornada.

Esse modelo permite a liberdade de planejamento de várias alternativas computacionais, tais como:

- Definir os clientes a serem visitados previamente, inclusive com a possibilidade de alterar esses clientes em tempo de viagem;
- Trabalhar de forma desconectada do servidor (somente existe interação na saída e/ou na chegada do código);
- Redução de tempo para completar a tarefa, considerando que existam características de gargalo entre clientes e servidor; nesse caso o código só passa pelo gargalo por duas vezes;
- Pouco consumo de banda (ao contrário do cliente-servidor, onde existe uma excessiva quantidade de interações entre servidor e clientes);
- Pouca ou nenhuma sensibilidade à latência do gargalo do trecho a ser vencido.

Considerando essas alternativas, o paradigma é muito promissor, e uma das aplicações mais adequadas seria a gerência de redes. Entre as vantagens do paradigma, comparando aos outros, citamos:

- Economia de largura de banda;
- Redução no tempo da tarefa;
- Redução no impacto da latência;
- Operação desconectada;
- Balanceamento de carga;
- Desenvolvimento dinâmico.

4.5.1 Economia de largura de banda

No modelo tradicional cliente-servidor, o servidor possui uma série de operações pré-definidas que o cliente solicita pela rede. Se o servidor não possui a operação que o cliente precisa, este terá que fazer uma seqüência de operações para realizar seu desejo ou, menos provável, o desenvolvedor do servidor precisará implementar essa operação no servidor. A primeira alternativa implica em várias operações sendo enviadas pela

rede, congestionando a operação, além de retornar ao cliente dados intermediários não desejados no objetivo real da operação.

Do outro lado está o modelo de agentes móveis, que economiza largura de banda, simplesmente pelo fato de que as operações entre agente e servidor são realizadas localmente, não poluindo a rede. Existem somente dois momentos clássicos de utilização da banda: quando o agente migra para o servidor, e quando retorna ao gerenciador, com o resultado desejado.

Não se pode deixar de citar que, nas situações em que a quantidade de operações no modelo cliente-servidor for pequena, com tamanhos de operações também pequenas, o código do agente poderá ser maior, tornando o modelo de agentes móveis menos vantajoso.

4.5.2 Redução no tempo da tarefa

Como dito anteriormente, o modelo de agentes móveis consome menor largura de banda que o modelo cliente-servidor. Quando o usuário busca economia de largura de banda, automaticamente está buscando maior velocidade no processamento de sua necessidade. Considerando que menor quantidade de interações leva um menor tempo para ser executado, temos o paradigma de agentes móveis como mais rápido.

Apesar da teoria, vale ressaltar um ponto:

- Os agentes móveis são porções de código que devem migrar para um elemento gerenciado, se instalar, processar e finalmente migrar para outro elemento gerenciado. Essa operação consome, naturalmente, muito mais recurso de máquina que solicitações simples cliente-servidor, e conseqüentemente o tempo de execução não deve ser desconsiderado;

Existem algumas variações no paradigma de agentes móveis que podem ser implementados visando maior desempenho na execução, resolvendo o problema citado anteriormente. Se o agente, por exemplo, envia resultados ao gerente após um número fixo de visitas, ocasionará duas vantagens: estará transmitindo o retorno em paralelo com outras visitas e estará diminuindo o tamanho do código ao ser migrado para o próximo elemento gerenciado (já que não precisa mais carregas as respostas, que foram previamente enviadas). Outra variação seria o envio de múltiplos agentes, que fariam visitas em paralelo.

Essas variações são o grande argumento que justifica nossa investigação, como será visto nos capítulos seguintes.

4.5.3 Redução no impacto da latência

O objetivo principal em aplicações de recuperação de informações é a redução do tempo de resposta, ou seja, o tempo entre quando a solicitação é enviada e quando a resposta chega. Se no caminho entre cliente e servidor existe algum circuito com alta latência, todas as informações que precisam trafegar por esse circuito têm sua perda de tempo.

No modelo cliente-servidor, composto por uma série de operações que sempre passam por esse circuito duas vezes (ao se transmitir para o servidor e ao retornar com a resposta ao cliente), a sensibilidade à latência é evidente e de grande impacto no tempo total gasto para completar a operação.

Já no modelo agente móvel, esse impacto praticamente não existe, pois só existem dois momentos de trânsito pelo circuito de alta latência, quando o código migra para o primeiro elemento gerenciado e quando ele retorna do último com as respostas.

Concluindo: o modelo agente móvel não é mais vantajoso por diminuir a latência do circuito, mas simplesmente por quase não senti-la em seu tempo total de processamento[1].

4.5.4 Operação desconectada

No modelo cliente-servidor, as operações enviadas e recebidas precisam do meio de transmissão sempre disponível. E sempre existe a necessidade de uma resposta a uma operação solicitada, para que a próxima operação seja enviada. A indisponibilidade surpresa do meio pode fazer com que uma tarefa cliente-servidor não seja completada, causando perda de tudo o que já tinha sido feito.

No modelo agente móvel, o código utiliza o meio de transmissão ao migrar do gerente para o primeiro elemento gerenciado. A partir desse momento, o agente não precisa mais se reportar ao gerente, a não ser no momento em que precisa enviar as respostas colhidas. Se o meio de transmissão é inconstante e estiver indisponível, o agente pode esperar pela sua próxima disponibilidade para então enviar as respostas.

[1] Se precisarmos contabilizar o tempo total da gerência, perceberemos que o tempo da latência desse circuito só será somado duas vezes, conforme apresentado em [Arantes 2002]

Essa operação desconectada pode ser muito útil quando existem no caminho da transmissão enlaces inconstantes.

Podemos, por exemplo, desenvolver aplicações em conexões discadas, onde se envia uma requisição (agente móvel), que passa a viajar pela rede executando a tarefa para a qual foi programado. Nesse entremeio, a conexão pode ser desligada e, em um segundo momento, existe uma segunda conexão para receber o resultado da tarefa, que pode estar em algum lugar específico esperando para ser descarregada pelo solicitante.

4.5.5 Balanceamento de carga

Problemas de balanceamento de carga estão evidentes na computação distribuída, onde pode existir carga de trabalho dividida de forma inadequada, causada por alguma anomalia não prevista. Mas desempenho está diretamente ligado à capacidade de divisão de trabalho de forma adequada aos recursos, de forma que nenhum recurso esteja mais carregado que outros. Balanceamento de carga pode ser dinâmico ou estático. Estático se a alocação de carga de trabalho acontece por uma análise anterior da computação exigida. Quando essa alocação está concluída, não é alterada até a finalização da computação. Atualmente a maioria das implementações de balanceamento de carga é estática nas computações distribuída e paralela. Balanceamento dinâmico pressupõe reavaliação e realocação da carga aos recursos computacionais em tempo de execução da computação.

Os agentes móveis suportam a forma mais geral e efetiva de balanceamento de carga dinâmico: executam os processos, implementados como agentes móveis, podem migrar, sob comando externo ou interno para qualquer elemento computacional compatível, no sistema distribuído.

4.5.6 Desenvolvimento dinâmico

Como discutido anteriormente, no modelo cliente-servidor, se o cliente precisa de uma operação que o servidor não dispõe, terá que fazer várias seqüências de operações para obter um resultado correspondente. No modelo de agentes móveis o código pode ser desenvolvido no cliente, enviado ao servidor (de processamento). Esse novo servidor é executado localmente e retorna com os resultados ao cliente solicitante.

Os agentes móveis suportam a forma mais geral de desenvolvimento dinâmico, onde a aplicação pode distribuir seus componentes *on-the-fly* a nós determinados e estes podem se distribuir a outros nós, conforme a necessidade.

4.6 Os paradigmas e a gerência de rede de computadores

A gerência de redes é uma atividade que, por sua natureza, deve possuir pelo menos as seguintes características:

- Deve produzir o mínimo possível de tráfego nos meios de comunicação pelos quais deve trafegar. Em caso contrário, a gerência chegaria à conclusão que é ela a causadora de congestionamento;
- Deve ser rápida suficiente para que possa detectar em tempo hábil os problemas das redes, evitando maiores problemas.

De posse dessas prerrogativas, e observando as características dos paradigmas citados, devemos verificar a melhor solução para cada situação de ambiente. Alguns dos paradigmas podem ser aplicados a implementações de gerência de redes, cada qual com suas vantagens e desvantagens. Contudo, historicamente, temos nos produtos e tecnologias de gerência de redes a utilização do modelo cliente-servidor, representado basicamente pelo SNMP ou pelo CMIP.

4.7 Conclusão

Percebemos neste capítulo que o paradigma de agentes móveis possui algumas vantagens interessantes em relação a outros paradigmas, principalmente no contexto da gerência de rede de computadores. As variações no paradigma de agentes móveis (variações de comunidades de agentes móveis) podem ser exploradas quando o objetivo é a redução no tempo da tarefa ou economia de largura de banda. Todas essas características são amplamente utilizadas nos capítulos seguintes.

5. KDEMA – Uma plataforma para implementação de agentes móveis

Observamos no capítulo anterior algumas características interessantes do paradigma de agentes móveis, e que podem ser direcionadas para a gerência de redes de computadores. Contudo, existem situações onde desejamos prototipar, simular ou implementar os agentes móveis, tendo como benefício a coleta de algumas informações sobre sua execução , além da observação prática do que foi posto em teoria.

Para essas aplicações práticas, desenvolvemos o KDEMA: uma plataforma para implementação de agentes móveis. Essa plataforma será utilizada principalmente nas implementações e no modelo de cálculo de consumo de banda.

Apresentamos a seguir algumas informações técnicas da plataforma, necessárias ao entendimento dos capítulos seguintes.

5.1 Introdução

Com os estudos sobre agentes móveis, surgiram necessidades da implementação de algumas comunidades de agentes móveis. São implementações simples, com o objetivo de visualizar aspectos práticos dos agentes móveis, exercendo suas características teóricas, tais como mobilidade ou operação desconectada. Utilizamos a Aglets [AGLETS] e a Zeus [ZEUS], além de consultar vários artigos sobre utilização de outras plataformas. As características técnicas das implementações utilizadas pelas plataformas são sempre diferenciadas e atendiam a prioridades imaginadas em tempo de projeto da própria plataforma. Senão, continuamos: percebemos que umas plataformas dão prioridade à resolução distribuída de problemas; outras à questões de segurança; outras ainda a questões de visualização ou administração.

As plataformas não se preocupam com questões interessantes às implementações voltadas à gerência de redes: consumo de banda, tempo gasto na execução de um problema, liberdade de escolha do momento de entrega de um dado coletado.

Definidos os objetivos deste trabalho, concluímos que : precisamos de alguns aspectos construtivos não existentes nas plataformas avaliadas. Precisamos de uma plataforma que ofereça subsídios para avaliar as comunidades de agentes móveis em

métricas interessantes para a gerência de redes de computadores. Outras características interessantes são execução leve, escalabilidade, portabilidade e simplicidade.

Desta forma desenvolvemos o KDEMA: uma plataforma que contempla todas as características desejadas para a implementação de comunidades de agentes móveis voltada para a gerência de redes de computadores.

5.2 Elementos da plataforma

O KDEMA, sob uma abstração elevada, possui duas figuras:

- O gerente: um computador sob o qual o gerente humano cria, executa e recebe os dados de determinada gerência;
- O elemento gerenciado (*Network Element* ou NE): um computador sob o qual o gerente pode enviar agentes ou tarefas a serem executadas; normalmente é o próprio alvo da gerência a ser trabalhado pelos agentes que ali estão;

Todos os computadores que são “gerente” ou “NE” podem participar diretamente de uma comunidade de agentes móveis, sendo que, computadores que não são gerente e nem NE, podem ainda participar da gerência, como elementos gerenciados, mas através de meios indiretos (como, por exemplo, interfaces SNMP ou proprietárias) e não são considerados participantes da comunidade.

Sob uma abstração mais detalhista, o KDEMA é formado por:

- Gerente (*Manager*): elemento que se executa no computador gerente;
- Ambiente (*Daemon*): elemento que se executa em cada NE;
- Mensagem (*Message*): elemento usado como evelopador de informações que são trocadas entre os componentes da comunidade;
- Seqüência (*Script*): conjunto de mensagens a serem enviadas convenientemente do gerente aos ambientes, de forma seqüencial, para execução de uma gerência;
- Caixa de Entrada e Caixa de Saída (*Inbox* e *Outbox*): elementos responsáveis pela troca de mensagens entre os elementos da comunidade; cada ambiente e o gerente possuem a sua caixa de entrada e sua caixa de saída; todas as mensagens enviadas saem pela caixa de saída; todas as mensagens recebidas entram pela caixa de entrada;

- Agente (*Agent*): o agente móvel em si, conforme definições teóricas; pode se móvel, se procriar, se matar, executar tarefas, etc...;
- Instrução (*Instruction*): uma invocação a um método Java (através de Java.Reflect), representada em um objeto; existem também as possibilidades de implementar laços e condicionais nas instruções;
- Tarefa (*Task*): conjunto de instruções que podem ser enviadas a qualquer agente, para execução; quando uma tarefa é enviada a um agente, passa a fazer parte deste (especialização do agente móvel);
- Conhecimento (*Knowledge*): elemento integrante das tarefas; armazena informações convenientes à tarefa, e que foram coletadas nos NEs ou recebidas do gerente, podem ser enviadas ao gerente ou utilizadas para consumo interno;
- Interface web (HTTP): elemento que oferece, nos ambientes e no gerente uma interface protocolo HTTP, para consulta e administração do elemento;
- Console (Console): elemento que disponibiliza a console da classe Java sendo executada (*Manager.class* ou *Daemon.class*), com suporte a envio de mensagens;
- Controle de nomes (*Named* ou *NamedLocal*): elementos integrantes do gerente (*Named*) ou do ambiente (*NamedLocal*) responsáveis pelos registros e resolução de nomes de todos os ambientes da comunidade; um registro é a atribuição de um nome;
- Coordenador (*Coordinator*): elemento responsável pela coordenação interna do gerente ou do ambiente; tem as funções de comunicação interna entre os elementos, verificação de sintaxe e semântica de mensagens recebidas ou a serem enviadas; instanciação de agentes, tarefas, interface web, etc...

5.3 Mensagens e fluxo de mensagens

As mensagens são elementos envelopadores das informações trocadas entre os elementos da comunidade. As mensagens podem ter vários propósitos e podem ou não,

exigir uma outra mensagem como resposta. Temos uma classificação das mensagens quanto à sua função:

- Controle de nomes: mensagens responsáveis por controlar o registro e a consulta a nomes de ambientes da comunidade;
- Tratamento de agentes: mensagens responsáveis por criar, enviar, mover ou matar agentes nos ambientes;
- Tratamento de tarefas: mensagens responsáveis por enviar tarefas aos agentes, para execução;
- Tratamento de informações: mensagens responsáveis por troca de informações coletadas ou fornecidas, entre ambientes e o gerente;

Cada mensagem enviada pode ou não esperar outra em resposta. Quando uma mensagem induz o ambiente ou o gerente a esperarem uma resposta, este cria um fluxo de mensagem e espera a mensagem de reposta para cancelá-lo. Quando uma resposta esperada não chega, um fluxo de mensagem fica pendente. Quando a mensagem não exige resposta, o fluxo é criado, mas é descartado pela caixa de saída quando a mensagem é enviada. Podemos consultar os fluxos pendentes via HTTP (interface web).

As mensagens disponíveis são:

Tabela 1. Mensagens disponíveis no KDEMA

Token	**Descrição**	**Direções**	**Propósito geral**
Mensagens de controle de nomes			
Dae.reg.req	Daemon.register.request	ambiente – gerente	Enviada pelo ambiente quando este se instancia; informa ao gerente o IP e a porta TCP da caixa de entrada do ambiente.
Dae.reg.res	Daemon.register.response	gerente – ambiente	Enviada pelo gerente ao ambiente, em resposta à dae.reg.req. Contém o nome registrado para o ambiente.
Dae.res.req	Daemon.resolve.request	ambiente – gerente	Solicita ao gerente o IP e Porta da caixa de entrada de determinado ambiente
Dae.res.res	Daemon.resolve.response	gerente – ambiente	Informa ao ambiente, em resposta à dae.res.req, o IP e porta da caixa de entrada de determinado ambiente
Mensagens de manipulação de agentes			
Age.cre.req	Agent.create.request	* – ambiente	Solicita ao ambiente a criação de um agente, com determinado nome
Age.tra.req	Agent.transfer.request	* – ambiente	Solicita a determinado ambiente que providencie a transferência de determinado agente a outro determinado ambiente
Age.dow.req	Agent.download.request	* – *	Solcita a determinado ambiente ou ao gerente que receba o agente e o instancie
Age.kil.req	Agent.kill.request	* – ambiente	Solicita a determinado ambiente que providencia a morte de determinado agente
Mensagens de manipulação de tarefas			
Tas.dow.req	Task.download.request	gerente – ambiente	Solicita que determinado ambiente receba a tarefa e a entregue a determinado agente para execução
Mensagens de manipulação de informações			
Age.kno.inf	Agent.know.information	ambiente – gerente	Envia dado(s) ao gerente
Mensagens de controle da plataforma			
Ping	Ping	* – *	Enviada para verificar se as caixas de entrada e saída de determinado ambiente estão operacionais
Pong	Pong	* – *	Enviada em resposta a uma mensagem ping

Toda PDU das mensagens do KDEMA é formada pelos seguintes elementos:

- Nome do fluxo (*flow*), que controla se a mensagem precisa de uma resposta;
- Origem da mensagem (origem), contém o nome do originador da mensagem;
- Destino da mensagem (destino), contém o nome do destinatário da mensagem;

- Símbolo da mensagem (*token*), contém o significado dessa mensagem;
- Argumento 1 (arg1);
- Argumento 2 (arg2); e
- Argumento 3 (arg3).

Os argumentos 1, 2 e 3 são utilizados de acordo com a sintaxe seguinte:

Tabela 2. Sintaxes das mensagens do KDEMA

Token	**Argumento 1**	**Argumento 2**	**Argumento 3**
Dae.reg.req	IP do ambiente enviador	Porta da caixa de entrada	
Dae.reg.res	Nome dado ao ambiente		
Dae.res.req	Nome do ambiente		
Dae.res.res	Nome do ambiente	IP do ambiente	Porta do ambiente
Age.cre.req	Nome do agente		
Age.tra.req	Nome do agente	Ambiente-destino	
Age.dow.req	Nome do agente	Binário do agente	
Age.kil.req	Nome do agente		
Tas.dow.req	Nome do agente	Nome da tarefa	Binário da tarefa
Age.kno.inf	Nome do agente	Nome da tarefa	Binário dos dados
Ping			
Pong			

5.4 Dinâmica da plataforma

Apresentamos a seguir a vida da plataforma, ou seja, como os elementos são iniciados em relação ao tempo. São informações de vital importância para o bom entendimento da plataforma, e necessárias para qualquer implementação.

5.4.1 Primeiro instante

Ao instanciar um gerente, precisamos informar qual a porta TCP desejada de sua caixa de entrada e qual a porta TCP desejada de sua interface web (HTTP). Nesse ponto o gerente está pronto para agremiar ambientes, constituindo uma comunidade. Uma comunidade só pode ter um gerente.

Quando instanciamos um ambiente, precisamos informar o endereço e a porta da caixa de entrada do gerente. Precisamos informar ainda a porta TCP desejada para a sua caixa de entrada e para sua interface web (HTTP) sob as quais o ambiente vai “escutar”. Em seguida, o ambiente envia suas informações ao gerente, solicitando o seu registro. Como resposta, o gerente envia um nome, sob o qual o ambiente passará a ser

identificado, dentro da comunidade. Nesse ponto o ambiente já faz parte da comunidade e está pronto para utilização.

5.4.2 Instantes seguintes

O gerente e todos os ambientes disponibilizam duas formas de envio direto de mensagens: via console ou via interface web. Quando instanciamos o gerente ou o ambiente, a janela de console passa a oferecer um *prompt* de comandos. Esse *prompt* está disponível para entrada de informações necessárias para o envio de uma mensagem a algum elemento da comunidade. Essas informações são:

- Alvo: elemento destinatário da mensagem;
- Símbolo: qual a mensagem a ser enviada;
- Arg1, Arg2 e Arg3: argumentos da mensagem

A interface web também oferece o mesmo recurso, através de entradas de texto convenientes. Podemos então enviar mensagens aos membros da comunidade e visualizar a reação de cada um.

5.5 Instalação da plataforma e implementação de um exemplo simples

Apresentamos a seguir um exemplo rápido de como implementar uma comunidade de agentes no KDEMA. Tomaremos como exemplo o seguinte cenário: um agente deve se mover para um NE gerenciável, identificar o sistema operacional em execução e retornar a informação para o gerente que o apresentará na interface web.

5.5.1 Questões relativas à versão

A plataforma está em constante desenvolvimento. Algumas características aqui descritas podem sofrer alterações. As versões posteriores podem não ser compatíveis com versões anteriores. Qualquer dúvida ou sugestão, por favor, enviar email para xavier@lrg.ufsc.br. Os arquivos atualizados, documentações relacionadas ou materiais didáticos podem ser encontrados em http://www.lrg.ufsc.br/~kdema.

5.5.2 Requisitos de ambiente

A plataforma (até esta versão, 0.7) foi totalmente desenvolvida em Java, usando a versão 1.4. Acreditamos que ele deva “rodar” também em versões anteriores (1.3, por

exemplo). Não nos prenderemos aqui a detalhes de instalação do Java, de forma que se houver algum problema, consulte outras documentações adequadas.

5.5.3 Instalação

Existem dois pacotes de software em arquivos compactados no formato zip, chamados Manager.zip e Daemon.zip. Descompacte-os em dois diretórios separados, pois existem classes com nomes iguais, mas cada qual para seu pacote.

5.5.4 Exemplo

O cenário é o seguinte: precisamos de um gerente da comunidade e de um ambiente executando em uma máquina (NE), que será gerenciada. Iniciamos instanciando o gerente (Manager.class do pacote Manager), que deve sempre ser o primeiro instanciado em uma comunidade de agentes. Ao ser instanciado, o gerente nos pergunta a porta desejada para sua caixa de entrada e a porta desejada para a sua interface web. Para este exemplo, informamos 1000 e 80, respectivamente. Em seguida são apresentadas na console algumas mensagens de controle.

Instanciamos em seguida, no mesmo computador, o ambiente (Daemon.class do pacote Daemon.zip) na máquina a ser gerenciada . Ao ser instanciado, o ambiente também nos faz algumas perguntas. As primeiras são relativas ao gerente, pois o ambiente deve se registrar para estar disponível para a plataforma. Responda com o IP do gerente, ou ainda com “localhost” (já que nesse exemplo os dois elementos estão instanciados no mesmo computador) e com a porta da caixa de entrada escolhida (no nosso caso, 1000). Em seguida mais duas perguntas, relativas ao ambiente: portas da caixa de entrada e da interface web. Responda 1100 e 1101, respectivamente.

Nesse ponto se tudo estiver correto, observamos algumas trocas de mensagem nas consoles do ambiente e do gerente. Se o registro acontecer corretamente, será apresentado na console do ambiente o seu nome, fornecido pelo gerente. No nosso caso deve ser “Daemon1”.

Para solicitar a troca de mensagens entre o gerente e o ambiente, temos duas alternativas: usamos a console aberta, digitando os parâmetros da mensagem no *prompt* oferecido, ou usamos a interface web disponível através de um apresentador (*browser*) HTTP (Netscape, Internet Explorer ou qualquer outro). As duas alternativas estão disponíveis tanto para o gerente quanto para o ambiente. Para iniciar a interface web do

gerente, entre com a URI que aponte para o computador onde o gerente foi instanciado. Para alcançar a interface web do ambiente, neste exemplo, basta entrar com a mesma URI seguida da porta informada: “http://localhost:1101”. Existe também a possibilidade de alcançar a página do ambiente pela página do gerente, através de *links* convenientes.

Vamos agora recuperar o nome do sistema operacional onde o ambiente está instanciado, e retornar para o gerente. Para isso podemos criar uma tarefa e utilizar um agente para executá-la, coletando a informação desejada e a enviando ao gerente. Para ativar um agente em um ambiente, usamos uma mensagem age.cre.req (agent.create.request). Essa mensagem exige o nome do agente a ser criado como primeiro argumento. No nosso caso, criaremos um agente chamado “007” no ambiente chamado “Daemon1”. Para isso, entre na console ou na interface web do gerente e digite:

Tabela 3. Mensagem criando um agente

Argumento	**Valor**
Alvo	Daemon1
Símbolo	age.cre.req
Arg 1	007
Arg 2	
Arg 3	

Ao enviar essa mensagem, observamos algumas informações nas consoles do gerente e do ambiente e em seguida uma informação na console do ambiente informando que existe um agente ativo. Podemos verificar a existência deste agente através da interface web do ambiente. Estamos agora prontos para enviar a tarefa para execução.

Uma tarefa é composta de instruções, que são invocações a métodos Java. Para montar uma tarefa, precisamos editar o arquivo TaskCompiler.Java (do pacote Manager.zip), entrar com as instruções desejadas e em seguida executá-lo. O resultado de sua execução é a criação de um arquivo com extensão .task que contém as instruções, serializadas em formato próprio. Em Java, o código para executar o que precisamos (coletar o nome do SO e o enviar ao gerente) seria como esse:

1. String wA=System.getProperty(“os.name”);
2. wAgent.setKnowledge(“OSName”,wA);

3. String wKnowledge=wAgent.getSerializedKnowledge();
4. String wAgentName=wAgent.getAgentName();
5. String wTaskName=wTask.getTaskName();
6. String wFlow=wCoordinator.newFlow(“Manager”);
7. Message wMessage=new Message(wFlow,“age.kno.inf”);
8. wMessage.setAllArgs(wAgentName,wTaskName,wKnowledge);
9. wCoordinator.setMessagetoSend(wMessage);

Para esclarecer:

- Na linha 1 recuperamos o nome do sistema operacional em wA;
- Na linha 2 colocamos esse conhecimento em Knowledge do agente;
- Nas linhas 3, 4 e 5 recuperamos todo o conhecimento do agente, o nome do agente e o nome da Task;
- Nas linhas 6 e 7 solicitamos ao coordinator que crie um fluxo de mensagens com o Manager e crie uma mensagem com o token age.kno.inf (agent.Knowledge.information);
- Na linha 8 setamos como argumento da mensagem as informações recuperadas (nome da Task, nome do agente e conhecimento); e
- Na linha 9 solicitamos ao coordinator que envie a mensagem.

O equivalente em instruções (objetos Instruction):

- // recupera OSName como String
- wI.setLabel("OSName");
- wI.setInstructionClass("java.lang.System");
- wI.setMethod("getProperty");
- wI.setArgs(new String[] {"os.name"});

- // coloca OSName em Knowledge do agente
- wI.setInstance("_agent");
- wI.setArgs(new String[] {"OSName","*OSName"});
- wI.setMethod("setKnowledge");

- // recupera toda a Knowledge do agente serializada em uma String
- wI.setLabel("Knowledge");
- wI.setInstance("_agent");
- wI.setMethod("getSerializedKnowledge");

- // recupera o nome do agente em uma String
- wI.setLabel("Agent");
- wI.setInstance("_agent");
- wI.setMethod("getAgentName");

- // recupera o nome da Task em uma String
- wI.setLabel("Task");
- wI.setInstance("_Task");
- wI.setMethod("getTaskName");

- // solicita ao coordinator um fluxo com o Manager
- wI.setLabel("wFluxo");
- wI.setInstance("_coordinator");
- wI.setArgs(new String[] {"Manager"});
- wI.setMethod("newFlow");

- // constroi um objeto message com o token age.kno.inf
- wI.setLabel("wMensagem");
- wI.setInstructionClass("Message");
- wI.setMethod("constructor");
- wI.setArgs(new String[] {"*wFluxo","age.kno.inf"});

- // coloca os tres argumentos: agent, Task e Knowledge na mensagem
- wI.setInstance("*wMensagem");
- wI.setMethod("setAllArgs");

- wI.setArgs(new String[] {"*Agent","*Task","*Knowledge"});

- // solicita ao coordinator que envie esta mensagem
- wI.setInstance("_coordinator");
- wI.setMethod("setMessagetoSend");
- wI.setArgs(new String[] {"*wMensagem"});

O arquivo TaskCompiler.java contém essas instruções implementadas. Ao executá-lo, temos como resultado a criação do arquivo “getOSName.task” que também acompanha o pacote Manager.zip e é a serialização dessa seqüência de instruções apresentada.

Podemos então enviar a tarefa criada para execução no agente “007”, que está ativo no ambiente “Daemon1”. Para isso, entramos no gerente com uma mensagem com o símbolo “tas.dow.req” (task.download.request). Esse símbolo pede dois argumentos: o nome do agente a receber a tarefa e o nome da tarefa a ser enviada. Entramos:

Tabela 4. Mensagem enviando uma tarefa

Argumento	**Valor**
Alvo	Daemon1
Símbolo	tas.dow.req
Arg 1	007
Arg 2	getOSName.Task
Arg 3	

Ao submetermos essa mensagem, percebemos várias informações nas consoles. Na console do ambiente verificamos informações apontando que o agente “007” está ativo e possui uma tarefa. Verificamos ainda outra informação dando conta que a tarefa “getOSName” está ativa. Por fim, a tarefa executa as suas instruções.

Na execução dessa tarefa, esta envia uma mensagem “age.kno.inf” (agent.Knowledge.information) para o gerente. Ao receber a mensagem, o gerente recupera o conhecimento enviado e o armazena em seu depósito de conhecimentos (*Knowledge base*). Podemos então verificar o seu conteúdo usando a opção de *Dump* de Base de Conhecimentos na interface web do gerente.

5.6 Conclusão

Como percebemos, a plataforma implementada é simples e de fácil implementação e gerência. Os recursos implementados tentam oferecer ao implementador todas as características desejáveis em uma plataforma voltada para a gerência de redes.

As instruções podem implementar praticamente tudo o que se consegue implementar em Java, com a restrição de não podermos usar tipos básicos (int, char, boolean, byte, etc). Essa restrição vem da tecnologia utilizada para implementar a máquina interpretadora das instruções, baseada no recurso de reflexão do Java. Não apresentamos no exemplo recursos já implementados na plataforma como múltiplos agentes, múltiplas tarefas em um agente, morte ou mobilidade de agentes, laços e condicionais em tarefas. A plataforma ainda está em desenvolvimento, e esperamos que novos recursos a tornem ainda mais funcional e interessante.

6. Comunidades Consideradas

Os agentes móveis possuem características individuais e de comunidade, encontradas em [OHARE 1996] e comentadas no capítulo 4, que nos permitem considerar, para uma mesma gerência, comunidades com características diferentes. Essas diferenças estão em como a comunidade é implementada[2]. Podemos, por exemplo, variar a quantidade de agentes envolvidos ou o momento de entrega do dado coletado. Este capítulo apresenta as comunidades de agentes consideradas para este trabalho, desenvolvendo suas implementações no KDEMA.

6.1 Introdução

As possibilidades de configurações de comunidades de agentes móveis são muitas. Consideramos três configurações onde um ou vários agentes coletam dados que são enviados ao gerente. Elas são apresentadas a seguir:

Configuração a

Nesta configuração, temos a gerência por um agente para toda a sub-rede, onde o agente é criado em um NE desta sub-rede (a.I), colhe o dado e migra para o próximo NE (a.II) sem envio de dados coletados ao gerente; continua migrando por todos os NEs (a.III, a.VI, a.V) até que, ao final da última coleta, envia todos os dados colhidos de uma só vez ao gerente (a.VI). A figura abaixo ilustra a configuração:

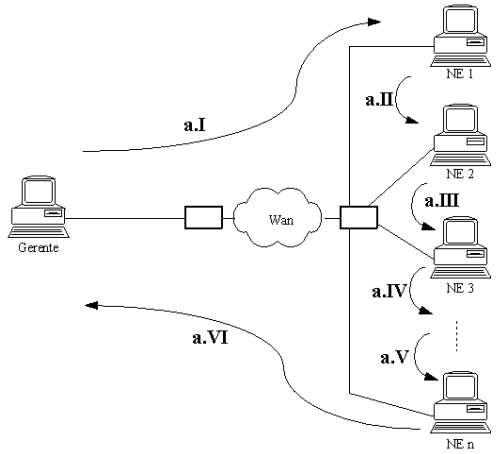

Figura 3. Configuração “a”

[2] Em [OHARE 1996] temos propostas de classificação dos agentes enquanto indivíduos e enquanto comunidade. Essas diferenças, propostas pela classificação, nos permitem conceber comunidades com características diferentes, mas com o mesmo resultado final.

Configuração b

Nesta configuração temos a gerência por um agente para toda a sub-rede, onde o agente é criado em um NE desta sub-rede (b.I), colhe o dado e o envia ao gerente (b.II), se move para o NE seguinte (b.III) fazendo nova coleta e novo envio (b.VI), e continuando até que todos os NEs sejam visitados (b.V, b.VI, b.VII, b.VIII, b.IX). Essa configuração está ilustrada abaixo:

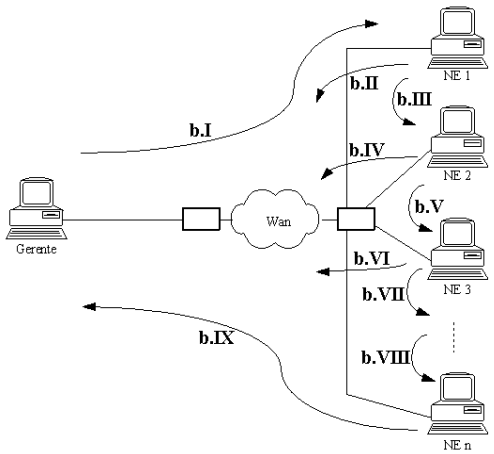

Figura 4. Configuração “b”

Configuração c

Nessa configuração, a gerência por um agente por NE da sub-rede, onde um agente é criado em cada NE gerenciado (c.I, c.III, c.V e c.VII), fazendo em seguida a coleta e por fim o envio de dados coletados ao gerente (c.II, c.IV, c.VI, c.VIII), conforme apresentado abaixo:

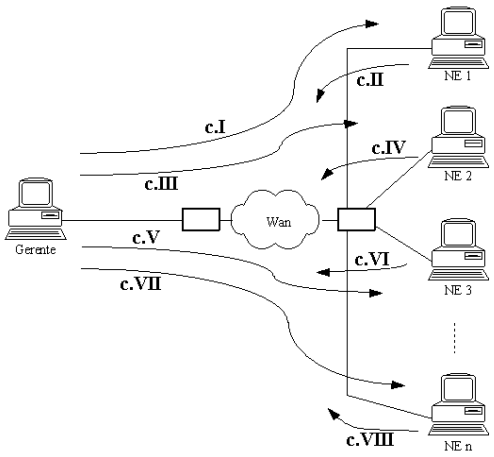

Figura 5. Configuração “c”

Ao observar o comportamento de cada configuração, percebemos que o resultado da gerência será o mesmo (os dados são coletados e entregues ao gerente). Contudo, percebemos que métricas relevantes para a gerência de redes de computadores podem variar entre as configurações e, portanto, serem passíveis de escolha por parte do gerente humano, quanto a algum quesito conveniente. Para este trabalho, nos concentramos na métrica "consumo de banda", que é de extrema importância na gerência de redes.

6.2 Detalhes importantes das implementações

Para implementar e executar as configurações consideradas no KDEMA, precisamos definir a tarefa necessária para cada configuração. A métrica considerada (consumo de banda) poderá ser obtida observando, após cada implementação, a tela de *Log* disponibilizada na interface web de cada elemento (gerente e ambientes). Exemplos da interface web e de telas de *Log* são apresentadas a seguir:

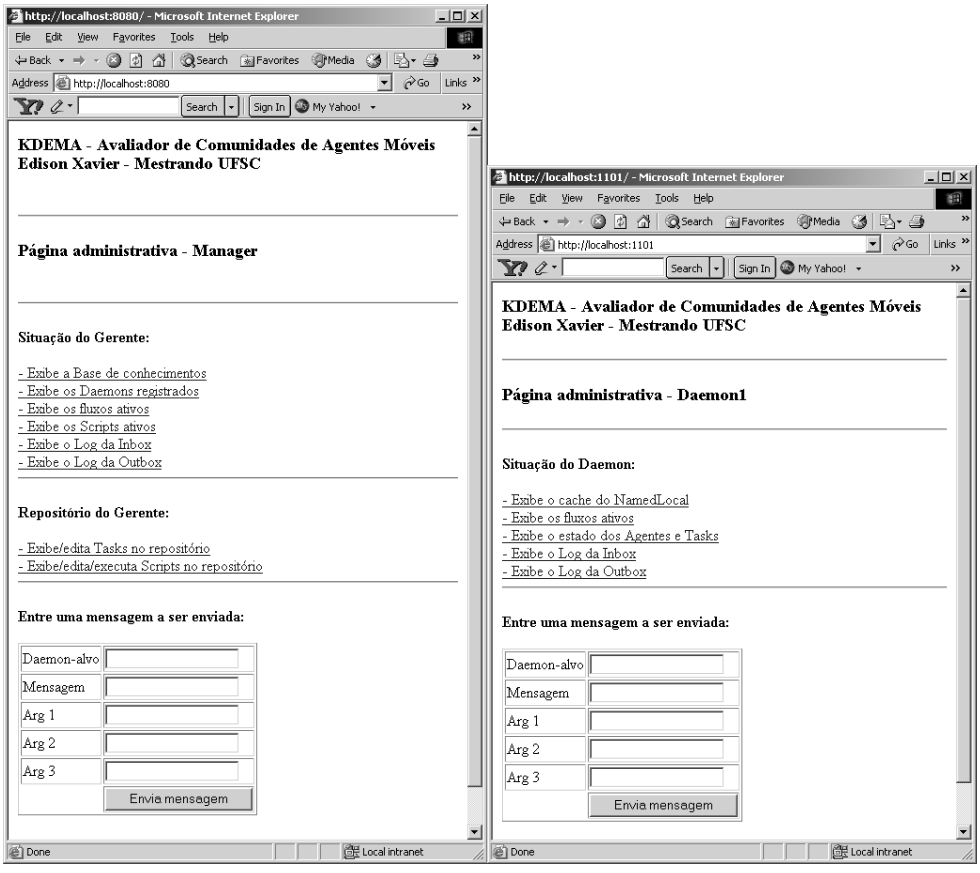

Figura 6. Telas web de administração do gerente e de um ambiente

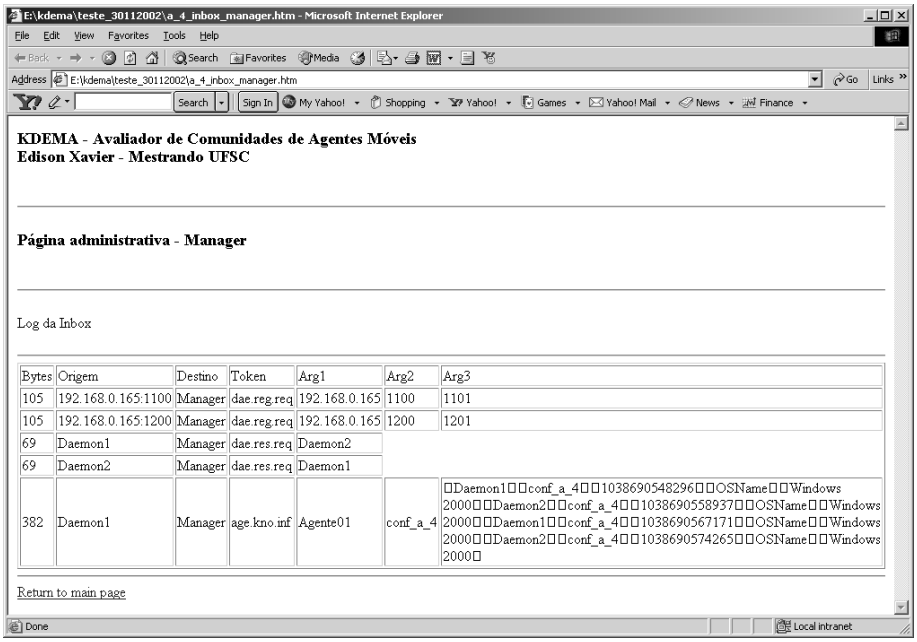

Figura 7. Exemplo de apresentação do *Log* da caixa de entrada do gerente

As coletas determinadas nas tarefas retornam o nome do sistema operacional da máquina que hospeda o agente. O valor coletado corresponde à invocação do método "GetProperty" com o parâmetro "os.name", da classe "Java.Lang.System" do Java.

6.3 Configuração a: Um agente para toda a gerência, enviando os resultados após percorrer todos os NEs.

Para implementar essa configuração, utilizamos as seguintes mensagens:

Tabela 5. Mensagens para a configuração "a"

Seq	Daemon	Token	Arg1	Arg2	Arg3
1	Daemon1	age.cre.req	Agente01		
2	Daemon1	tas.dow.req	Agente01	Tarefa01.Task	

Definimos em seguida uma tarefa com as seguintes instruções:

Tabela 6. Instruções para a configuração "a"

Seq	Label	Classe	Método	Instância	Arg1	Arg2
1	Dado1	Java.lang.System	GetProperty	_null_	Os.name	
2	_null_	_null_	SetData	_null_	Osname	*Dado1
3	_null_	_null_	MoveAgent	_null_	Daemon2	
4	Dado2	Java.lang.System	GetProperty	_null_	Os.name	
5	_null_	_null_	SetData	_null_	Osname	*Dado2
6	_null_	_null_	MoveAgent	_null_	Daemon1	
7	_null_	_null_	Goto	_null_	2	*Dado1
8	_null_	_null_	SendData	_null_		

Quando submetemos as mensagens no gerente, este envia uma mensagem “age.cre.req” ao ambiente “Daemon1”, solicitando que este crie um agente chamado “Agente01”; em seguida o gerente envia ao ambiente “Daemon1” uma tarefa chamada “Tarefa01”, contida no arquivo “Tarefa01.Task”, para que o agente “Agente01” a receba e a execute; ao receber essa tarefa, o agente executa a primeira instrução, recuperando o nome do sistema operacional sob o qual ele está sendo executado, através do método GetProperty(“os.name”) da classe System do Java; observamos que essa instrução 1 possui um rótulo (Label) denominado “Dado1”, que pode ser apontado por outras instruções; em seguida é solicitada uma chamada ao método SetData, interno do KDEMA; esse método armazena na base de conhecimentos do agente um dado contido em Arg2 (rótulo “Dado1”) com o nome fornecido em Arg1 (“osname”); a seguir o agente executa uma chamada ao método interno MoveAgent, que providencia a mobilidade do agente para o ambiente passado como argumento em Arg1 (“Daemon2”, nesse caso); salientamos que, quando acontece a mobilidade de um agente, a sua base de conhecimento e o seu ponteiro de execução da tarefa também são movidos, preservando o estado de execução do agente; esse processo se repete nas instruções 4, 5 e 6, mas desta vez, executados em “Daemon2”; na instrução 7 existe uma chamada ao método interno Goto, que decrementa 1 de seu Arg1 e salta para o rótulo em Arg2 se Arg1 não for zero; nesse caso percebemos que Arg1 foi iniciado com valor 2, pois precisamos que o laço seja feito duas vezes, nos dando 4 NEs visitados; por fim, na instrução 8, o agente executa um método interno, chamado SendData, que envia toda a sua base de conhecimentos para o gerente, através de uma mensagem “age.kno.inf”.

6.4 Configuração b: Um agente para toda a gerência, enviando os resultados após cada coleta.

Para implementar essa configuração, definimos as seguintes mensagens:

Tabela 7. Mensagens para a configuração “b”

Seq	Daemon	Token	Arg1	Arg2	Arg3
1	Daemon1	age.cre.req	Agente01		
2	Daemon1	tas.dow.req	Agente01	Tarefa02.Task	

Definimos em seguida uma tarefa com as seguintes instruções:

Tabela 8. Instruções para a configuração “b”

Seq	Label	Classe	Método	Instância	Arg1	Arg2
1	Dado1	Java.lang.System	GetProperty	_null_	os.name	
2	_null_	_null_	SetData	_null_	Osname	*Dado1
3	_null_	_null_	SendData	_null_		
4	_null_	_null_	MoveAgent	_null_	Daemon2	
5	Dado2	Java.lang.System	GetProperty	_null_	os.name	
6	_null_	_null_	SetData	_null_	Osname	*Dado2
7	_null_	_null_	SendData	_null_		
8	_null_	_null_	MoveAgent	_null_	Daemon1	
9	_null_	_null_	Goto	_null_	2	*Dado1

Quando executamos as mensagens no gerente, este envia uma mensagem “age.cre.req” seguida de uma “tas.dow.req”, solicitando ao ambiente “Daemon1” que crie o agente “Agente01” e que este execute uma tarefa contida no arquivo “Tarefa02.Task”; ao executar a tarefa, as instruções 1 e 2 são semelhantes à primeira configuração: o agente recupera o nome do sistema operacional e o armazena em sua base de conhecimentos; mas na instrução 3 ele envia esse dado coletado ao gerente, e em 4 solicita a sua mobilidade para “Daemon2”; em “Daemon2” o processo se repete; na instrução 8 o agente solicita sua mobilidade para “Daemon1” novamente; em 9, por fim, decrementa 1 de seu Arg1 e salta para o rótulo “Dado1” se Arg1 não for 0; com isso temos 4 NEs visitados, com um dado coletado por NE e enviado ao gerente imediatamente após sua coleta, conforme desejávamos.

6.5 Configuração c: Um agente para cada NE.

Para implementar essa configuração, definimos as seguintes mensagens:

Tabela 9. Mensagens para a configuração “c”

Seq	Daemon	Token	Arg1	Arg2	Arg3
1	Daemon1	age.cre.req	Agente01		
2	Daemon1	tas.dow.req	Agente01	Tarefa03.Task	
3	Daemon2	age.cre.req	Agente02		
4	Daemon2	tas.dow.req	Agente02	Tarefa03.Task	
5	Daemon3	age.cre.req	Agente03		
6	Daemon3	tas.dow.req	Agente03	Tarefa03.Task	
7	Daemon4	age.cre.req	Agente04		
8	Daemon4	tas.dow.req	Agente04	Tarefa03.Task	

Definimos em seguida uma tarefa com as seguintes instruções:

Tabela 10. Instruções para a configuração “c”

Seq	Label	Classe	Método	Instância	Arg1	Arg2
1	_null_	Java.lang.System	GetProperty	_null_	os.name	
2	_null_	_null_	setData	_null_	osname	*Dado1
3	_null_	_null_	SendData	_null_		

Quando executamos as mensagens no gerente, uma é enviada ao ambiente “Daemon1” solicitando a criação de um agente; em seguida é enviada uma mensagem contendo uma tarefa que deverá ser recebida e executada pelo agente criado; esse procedimento é repetido para todos os ambientes; cada um dos agentes coleta, então, o nome do sistema operacional da máquina e o envia ao gerente.

6.6 Resultados das implementações das configurações consideradas

Os resultados são coletados das telas de Log das caixas de entrada e saída do gerente e dos ambientes envolvidos na comunidade utilizada. Como visto na fig. 7, temos nessa tela uma coluna chamada “bytes”, que apresenta o tamanho, em bytes, da mensagem enviada pela caixa de saída ou recebida pela caixa de entrada, de cada módulo. Contabilizamos então como consumo, a totalidade de mensagens envolvidas na gerência, excluindo mensagens específicas, que não fazem parte do contexto, como, por exemplo, mensagens “dae.reg.req”, que fazem parte do módulo de resolução de nomes, e que ocorrem somente uma vez durante toda a vida de um ambiente.

Os resultados serão agrupados em três escopos:

- “No gerente”, que se refere à contabilização dos bytes de todas as mensagens que foram recebidas ou enviadas pelo gerente, para algum ambiente qualquer;
- “Entre NEs”, que se refere à contabilização dos bytes de todas as mensagens recebidas ou enviadas pelos ambientes, entre si; exclui-se daqui as mensagens enviadas de um ambiente para o gerente; mensagens enviadas de um ambiente para outro, apesar de constarem tanto na caixa de entrada do emissor quanto na caixa de saída do receptor, são contabilizadas uma única vez;
- “Na gerência”, que se refere à contabilização dos bytes de todas as mensagens enviadas ou recebidas, indiferente de origem e destino, que foram necessárias para a realização da gerência;

Essa divisão foi proposta por nos dar as possibilidades de:

- Quando o gerente e os NEs estiverem distantes, avaliar a quantidade de bytes que estaria sujeito a latências e larguras de banda de um suposto “*link*” diferenciado entre o gerente e os NEs;

- Quando os NEs estiverem em uma mesma rede, avaliar a carga de bytes desta gerência, nesta rede.

Após a experimentação, com uma comunidade de 4 NEs, obtivemos as telas de *Log* de cada elemento. São telas longas e não são apresentadas aqui, mas estão disponíveis em [KDEMA] e no Anexo I deste trabalho. Salientamos que esses dados se referem à camada de aplicação no modelo OSI de camadas: o consumo de banda aqui não está contabilizando cabeçalhos TCP (transporte), IP (rede) ou *Ethernet* (enlace). Os resultados foram refinados, totalizados e estão apresentados na tabela abaixo:

Tabela 11. Dados coletados

Conf	**Consumo de bytes para 4 NEs**		
	No gerente	**Entre NEs**	**Na gerência**
a	1207	3610	4817
b	1505	3298	4803
c	2272	0	2272

6.7 Conclusão

Observamos pelos resultados obtidos que, conforme esperado, existe uma grande variação da quantidade de bytes consumidos pela gerência, considerando cada uma das três configurações e seus escopos. Não entraremos ainda no mérito de discussões ou conclusões sobre cada uma das configurações, tentando apontar vantagens ou desvantagens. O objetivo aqui é justificar a investigação que se segue: o levantamento de um modelo matemático correspondente ao comportamento do KDEMA para a métrica “consumo de banda”. Com o modelo proposto e validado, podemos então desenvolver algoritmos para decisão da melhor configuração, dado um desejo do usuário.

7. Modelos Matemáticos das Configurações para Cálculo de Consumo de Banda no KDEMA

A decisão de melhor configuração poderia ser feita através da análise dos resultados coletados, pelas experimentações de todas as possibilidades imaginadas. Contudo, é improvável que algum gerente humano fique satisfeito com a possibilidade de precisar implementar todas as possibilidades de comunidades, para descobrir qual a mais adequada para a sua necessidade de gerência. Este capítulo apresenta a solução encontrada para a obtenção da métrica "consumo de banda", dadas as configurações consideradas.

7.1 Introdução

Obter a decisão da escolha da melhor configuração no nível teórico pode ser possível através de modelos matemáticos que nos apresente o comportamento esperado de cada uma das configurações consideradas. Para isso, trabalharemos na definição de modelos matemáticos, que possam nos informar, com certa confiabilidade, os comportamentos das configurações consideradas, dada uma gerência.

Consumo de banda é uma métrica diretamente ligada à implementação da plataforma sob a qual estamos avaliando, pois envolve características de comunicação, específicos de cada plataforma. Os modelos matemáticos desse capítulo, dada essa dependência, são específicos para utilização com o KDEMA.

7.2 Características de implementação do KDEMA

Para chegar aos modelos desejados, precisamos levantar algumas características da plataforma utilizada. No KDEMA, a comunicação entre gerente e um agente ou entre agentes é feita através de elementos denominados mensagens. As mensagens são transmitidas através de conexões TCP/IP feitas entre o emissor e o recebedor da mensagem.

Cada mensagem no KDEMA tem uma função específica, conhecida entre os dois lados da comunicação e é construída em um envelope formado por sete fragmentos:

- Nome do fluxo (fluxo), que controla se a mensagem precisa de uma resposta;

- Origem da mensagem (origem), informa ao recebedor quem enviou a mensagem;
- Destino da mensagem (destino), contém o nome do destinatário da mensagem;
- Símbolo da mensagem (*token*), contém o significado dessa mensagem;
- Argumento 1 (arg1);
- Argumento 2 (arg2); e
- Argumento 3 (arg3).

Esses fragmentos (a mensagem) são serializados em uma PDU (*Protocol Data Unit*) do KDEMA, que é posteriormente enviada para o destinatário, onde ela é desserializada em fragmentos e transformada novamente em uma mensagem.

O conjunto de mensagens do KDEMA foi desenvolvido com o objetivo de ser simples e de execução "leve". Não possui nenhum tratamento de tolerância à falhas no que se refere a problemas de sintaxe em algum fragmento ou entrega com confirmação[3]. Quando uma mensagem tiver sido enviada, o emissor considera que ela foi entregue, entendida e atendida, e por isso nunca esperará uma resposta, a não ser em casos em que o fluxo a exigir.

7.2.1 Apresentação dos comportamentos existentes na plataforma

Conhecendo as características das mensagens do KDEMA, e ainda, conhecendo as necessidades das configurações imaginadas, podemos enumerar os comportamentos desejados. São eles:

- O nascimento de um agente em determinado NE, que ocasiona uma mensagem do tipo "age.cre.req" (*agent.create.request*) do gerente para o ambiente respectivo;
- O envio de uma tarefa ao agente, que ocasiona uma mensagem "tas.dow.req" (*task.download.request*) do gerente para o ambiente respectivo;

[3] Apesar das implementações de mensagens no KDEMA utilizarem o TCP, que é um protocolo de transporte orientado a conexão, e portanto confiável, consideramos que a entrega é sem confirmação porque o destinatário não informa ao remetente da mensagem se ela foi recebida e entendida.

- O envio de um dado coletado no NE ao gerente, que ocasiona uma mensagem “age.kno.inf” (*agent.knowledge.information*) do ambiente para o gerente;
- A mobilidade de um agente de um NE para outro, que ocasiona uma mensagem “age.dow.req” (*agent.download.request*) do ambiente-origem para o ambiente-destino.

7.2.2 Definição dos comportamentos necessários às configurações

Considerando uma sub-rede com um número n de NEs, definimos as mensagens envolvidas para as configurações imaginadas:

Conf. a: gerência por um agente com envio de dados ao final da gerência:

- 1 nascimento de agente (age.cre.req) do gerente ao primeiro ambiente;
- 1 envio de tarefa (tas.dow.req) do gerente ao primeiro ambiente;
- n-1 mobilidades (age.dow.req) de um ambiente ao próximo;
- 1 envio de dados (age.kno.inf) com n dados do último ambiente ao gerente.

Conf b: gerência por um agente com envio de dados logo após a coleta:

- 1 nascimento de agente (age.cre.req) do gerente ao primeiro ambiente;
- 1 envio de tarefa (tas.dow.req) do gerente ao primeiro ambiente;
- n envios de dados colhidos (age.kno.inf) de cada ambiente ao gerente;
- n-1 mobilidades (age.dow.req) de um ambiente ao próximo.

Conf c: gerência por n agentes com envio de dados após a coleta:

- n nascimentos de agentes (age.cre.req) do gerente a cada ambiente;
- n envios de tarefas (tas.dow.req) do gerente a cada ambiente;
- n envios de dados colhidos (age.kno.inf) de cada ambiente ao gerente;

7.2.3 Métodos de serialização das mensagens do KDEMA

Existe na biblioteca interna do KDEMA um conjunto de métodos de serialização e desserialização de objetos que utiliza caracteres de *escape* para delimitar os fragmentos

serializados[4]. No processo de serialização, é procurado em cada fragmento a ocorrência indesejável de caracteres similares aos de *escape*. Esses caracteres indesejados são separados por bytes 'separadores', de forma a não serem confundidos com reais delimitadores.

Essa informação é importante, pois na contabilização de tamanhos de fragmentos serializados precisamos considerar os caracteres de inserção, para um melhor resultado do modelo.

7.3 Contabilização dos tamanhos das mensagens

Conhecendo a construção das mensagens do KDEMA, podemos modelar fórmulas matemáticas para cálculo do tamanho em bytes de todas as mensagens necessárias para cada comportamento. Para tanto, consideremos a seguinte tabela:

Tabela 12. Fragmentos das mensagens com seus respectivos tamanhos

Mensagem	**Fluxo**	**Origem**	**Destino**	**Token**	**Arg1**	**Arg2**	**Arg3**	**Escapes**
age.cre.req	no+2	no	Nd	11	na	0	0	28
tas.dow.req	no+2	no	Nd	11	na	nt	Ts	28
age.kno.inf	no+2	no	Nd	11	na	nt	ks	28
age.dow.req	no+2	no	Nd	11	na	as	0	28

Onde:

- no: tamanho do nome do ambiente emissor da mensagem
- nd: tamanho do nome do ambiente destinatário da mensagem
- na: tamanho do nome do agente envolvido
- as: tamanho do agente serializado
- nt: tamanho do nome da tarefa envolvida
- ts: tamanho da tarefa serializada
- ks: tamanho da base de conhecimento serializada – pode ser vazia ou conter n dados coletados

Redução de fórmulas com no (nome do originador) e nd (nome do destinatário)

As variáveis "no" e "nd" significam tamanho de nome da origem ou destino da mensagem, que podem ser o gerente ou os ambientes. As possibilidades para esses campos são "Manager" ou "DaemonX", onde X é um número seqüencial iniciado em 1,

[4] Técnica descrita em [TANENBAUM] como inserção de caracteres (*character stuffing*), utilizada para resolver problemas de enquadramento de dados transmitidos de forma serializada.

e definido automaticamente pelo gerente, por ordem cronológica do registro. Supondo uma comunidade de até 9 ambientes registrados, teremos sempre nomes de tamanho 7. Para comunidades com mais de 9 ambientes, teremos nomes com tamanho 9. Se a mensagem se destina ou origina do gerente, o tamanho é 7.

Temos então, para “no” e “nd” tamanhos que podem variar de 7 a 9, e para efeito de redução consideraremos então como tamanho 8, obtido pela média aritmética dos dois valores. Reduziremos as ocorrências “no” e “nd” para 8 bytes.

Redução de fórmulas com “na” (nome do agente) e “nt” (nome da tarefa)

Nas variáveis “na” (tamanho do nome do agente) e “nt” (tamanho do nome da tarefa) temos valores dependentes do gerente, pois esses nomes são atribuídos em tempo de gerência, a critério do gerente humano. No entanto, para redução de fórmulas, adotaremos nomes de agentes com o formato “AgenteXX” e nomes de tarefas com o formato “TarefaXX”, onde “XX” é um número seqüencial iniciado em 1. Para uma gerência com até 99 agentes e 99 tarefas esse padrão sempre nos retornará tamanhos de nomes de agentes e tarefas com 8 bytes.

Fórmulas com “ts” (tarefa serializada)

Uma tarefa é uma seqüência de instruções que são enviadas para os agentes para execução seqüencial. Uma instruçãoé um objeto que armazena, utilizando *strings*, informações para invocação de um método Java através de Java.Reflect.

Na montagem da matriz para serialização de uma tarefa, temos:

- Nome da tarefa;
- Posição atual (*stack pointer*) de execução; e
- Todos os fragmentos de cada instrução da tarefa.

Uma instrução tem os seguintes fragmentos:

- Rótulo;
- Classe;
- Instância;
- Método;
- Número de parâmetros;

- Parâmetros (pode ocorrer 0 ou n vezes);
- Número de argumentos;
- Argumentos (pode ocorrer 0 ou n vezes)

A quantidade de fragmentos de uma instrução é dada por:

- fi=6+qtde_pars+qtde_args

Os fragmentos de uma instrução nunca precisarão de inserção de *escapes*, por conterem sempre *strings*. Com isso, temos que a quantidade de bytes de *escape* de uma instrução, na montagem de uma tarefa serializada, é dada por:

- ei=4*(6+qtde_pars+qtde_args), ou
- ei=24+4*(qtde_pars+qtde_args)

O tamanho total de uma instrução dentro de uma tarefa serializada é então:

- is=som_tam_frags + ei, ou
- is= 24 + som_tam_frags + 4*(qtde_pars+qtde_args)

A quantidade de fragmentos de uma tarefa serializada é dada por:

- ft=2+som_fi

E por fim o tamanho de uma tarefa serializada é:

- ts=som_tam_frags+qtde_esc_inseridos

Temos o nome da tarefa, com tamanho 8 e o *stack pointer*, com tamanho 2, ambos em formato *string*, donde vem:

- ts=10+(2*4)+som_is, ou
- ts=18+som_is

Fórmulas com “ks” (dados serializados)

Os dados são objetos compostos dos seguintes fragmentos:

- Ambiente;
- Nome da tarefa;

- DataHora (retornada de System.currentTimeMillis() como a quantidade de milisegundos entre agora e 01/01/1970 – atualmente com 13 bytes de tamanho);
- Chave (definido pela instrução que a criou); e
- Valor (definido pela instrução que a criou).

Um dado a ser serializado sempre tem cinco fragmentos. A quantidade de *escapes* inseridos, considerando que seus fragmentos são sempre *strings*, é:

- ed=4*5, ou
- ed=20

A somatória dos tamanhos dos fragmentos de um dado é:

- td=8+8+13+tcv, onde tcv=tamanho da chave mais o tamanho do valor, ou
- td=29+tcv

Por fim, o tamanho de um dado serializado é dado por:

- ds=td+ed, ou
- ds=29+tcv+20, ou
- ds=49+tcv

Com isso, temos que a quantidade de fragmentos de uma base serializada é dada por:

- fk=qtde_dados*5

E o tamanho de uma base serializada é dado por:

- ks=som_ds, ou
- ks=som(49+tcv), ou
- ks=49*qtde_dados+som_tcv

Redução de fórmulas com “as”

O agente serializado é formado pelos seguintes fragmentos:

- Nome do agente;
- Dados serializados do agente (ks);
- Tarefas serializadas do agente (ts).

Um agente serializado então tem sempre três fragmentos. O nome do agente tem tamanho 8. Os outros dois fragmentos não são *strings* e terão inserção de *escapes*. A quantidade de escapes inseridos em “ks” é:

- eks=2*fk, ou
- eks=2*qtde_dados*5, ou
- eks=10*qtde_dados

A quantidade de *escapes* inseridos em “ts” é:

- ets= 2*ft, ou
- ets=2*(2+som_fi), ou
- ets=4+(2*som_fi)

Com isso temos os novos tamanhos dos fragmentos “ks” e “ts”, chamados “ks2” e “ts2”:

- ks2=49*qtde_dados+som_tcv+eks, ou
- ks2=49*qtde_dados+som_tcv+10*qtde_dados, ou
- ks2=59*qtde_dados+som_tcv, e
- ts2=18+som_is+ets, ou
- ts2=18+som_is+4+(2*som_fi), ou
- ts2=22+som_is+(2*som_fi)

Por fim calculamos os *escapes* para três fragmentos:

- ea=4*3, ou
- ea=12

Com isso, o tamanho de um agente serializado é dado pela soma dos tamanhos dos fragmentos com a soma de seus *escapes*:

- as=na+ks2+ts2+ea, ou

- as=8+(59*qtde_dados)+som_tcv+22+som_is+(2*som_fi)+12, ou
- as=42+(59*qtde_dados)+som_tcv+som_is+(2*som_fi)

Se considerarmos que as chaves coletadas têm o mesmo tamanho de nome e valor, podemos fazer:

- as=42+(59*qtde_dados)+(tcv*qtde_dados)+som_is+(2*som_fi), ou
- as=42+qtde_dados*(59+tcv)+som_is+(2*som_fi)

Fórmulas reduzidas

Adotamos então para efeito de redução:

- no, nd e na = 8;
- nt = 8;
- ts=18+som_is
- ks=(49+tcv)*qtde_dados
- as=42+qtde_dados*(59+tcv)+som_is+(2*som_fi)

Na mensagem “tas.dow.req”, o segundo argumento é o nome do arquivo da tarefa correspondente. Considerando que o nome do arquivo tem o mesmo nome dado à tarefa, teríamos:

- para tarefa sendo “Tarefa01”, o arquivo será “Tarefa01.Task”, ou seja, com 5 bytes a mais.

Temos então:

Tabela 13. Tabela de fórmulas reduzidas

Mensagem	Fórmula original	Fórmula reduzida
age.cre.req	no*2+nd+na+41	73
tas.dow.req	no*2+nd+na+nt+ts+41	104+som_is
age.kno.inf	no*2+nd+na+nt+ks+41	81+(49+tcv)*qtde_dados
age.dow.req	no*2+nd+na+as+41	115+qtde_dados*(59+tcv)+som_is+(2*som_fi)

Não podemos perder de vista que os fragmentos de serialização da mensagem também sofrerão inserção de *escapes* de desmanche. Temos que rever as fórmulas, somando a correção como segue:

- Correção de ks=qtde_dados*5*2, ou
- Correção de ks=10*qtde_dados.

- Correção de ts=(2+som_fi)*2, ou
- Correção de ts=4+(2*som_fi).

- Correção de as=(3+fk+ft)*2, ou
- Correção de as=(3+2+som_fi+qtde_dados*5)*4, ou
- Correção de as=20+4*som_fi+20*qtde_dados.

Tabela 14. Fórmulas reduzidas

Mensagem	Fórmula original	Fórmula reduzida
age.cre.req	no*2+nd+na+41	73
tas.dow.req	no*2+nd+na+nt+ts+41	108+som_is+(2*som_fi)
age.kno.inf	no*2+nd+na+nt+ks+41	81+(59+tcv)*qtde_dados
age.dow.req	no*2+nd+na+as+41	135+qtde_dados*(79+tcv)+som_is+(6*som_fi)

7.4 Definição dos bytes gastos em uma operação

7.4.1 Bytes gastos nos comportamentos

Utilizando a definição anterior das operações e das considerações acima, temos que:

- O nascimento de um agente em determinada máquina ocasiona uma mensagem “age.cre.req”, nos dando 73 bytes transitados entre o gerente e o NE;
- O envio de uma tarefa ao agente ocasiona uma mensagem “tas.dow.req”, resultando em 108+som_is+(2*som_fi) bytes transitados entre o gerente e o NE;
- O envio de uma base de dados colhido do NE ao gerente ocasiona uma mensagem “age.kno.inf”, resultando em 81+(59+tcv)*qtde_dados bytes transitados entre a máquina e o gerente;
- A mobilidade de um agente de um NE para outro ocasiona uma mensagem “age.dow.req”, resultando em 135+qtde_dados*(79+tcv)+som_is+(6*som_fi) bytes entre os NEs envolvidos na mobilidade.

7.4.2 Bytes gastos nas configurações

Em uma sub-rede com n NEs, podemos então somar os bytes envolvidos, considerando os comportamentos de cada cenário.

Gerência por um agente com dados enviados ao final da gerência

Essa gerência ocasionaria os seguintes comportamentos:

- 1 nascimento de agente (73);
- 1 envio de uma tarefa a esse agente (108+som_is+(2*som_fi));
- 1 envio de dados colhidos ao gerente: (81+(59+tcv)*qtde_dados); e
- n-1 mobilidades desse agente para o NE seguinte.

Sendo que:

- Na primeira: 135+1*(79+tcv)+som_is+(6*som_fi)
- Na segunda: 135+2*(79+tcv)+som_is+(6*som_fi)
- Na última (n-1): 135+(n-1)*(79+tcv)+som_is+(6*som_fi)

Temos então, para a somatória de bytes gastos no comportamento de mobilidade:

- bytes=135+(79+tcv)+som_is+(6*som_fi)+135+2*(79+tcv)+som_is+(6*som_fi) + …
- bytes=(n-1)*(135+(n/2)*(79+tcv)+som_is+(6*som_fi))

Isso nos leva às seguintes fórmulas para a gerência:

- Trânsito no gerente: 73+108+som_is+(2*som_fi)+81+(59+tcv)*n
- Trânsito entre NEs: (n-1)*(135+(n/2)*(79+tcv)+som_is+(6*som_fi))
- Trânsito total da gerência: 73+108+som_is+(2*som_fi)+81+(59+tcv)*n+(n-1)*(135+(n/2)*(79+tcv)+som_is+(6*som_fi))

Gerência por um agente para toda a sub-rede com dados enviados após sua coleta

Essa gerência ocasionaria os seguintes comportamentos:

- 1 nascimento de agente: 73;
- 1 envio de uma tarefa a esse agente: (108+som_is+(2*som_fi));
- n envios de um dado colhido ao gerente: n*(81+59+tcv); e

- n-1 mobilidades desse agente para o NE seguinte.

Onde temos:

- na primeira mobilidade: (135+som_is+(6*som_fi))
- na segunda mobilidade: (135+som_is+(6*som_fi))
- na última (n-1) mobilidade: (135+som_is+(6*som_fi))

Ou seja, temos a mobilidade ocorrendo (n-1) vezes, nos dando:

- bytes=(n-1)*(135+som_is+(6*som_fi))

Obtemos então:

- Trânsito no gerente: 73+108+som_is+(2*som_fi)+n*(81+59+tcv)
- Trânsito entre NEs: (n-1)*(135+som_is+(6*som_fi))
- Trânsito total da gerência: 73+108+som_is+(2*som_fi)+n*(81+59+tcv)+(n-1)*(135+som_is+(6*som_fi))

Gerência por um agente para cada host da sub-rede

Essa gerência ocasionaria os seguintes comportamentos:

- n nascimentos de agentes: n*73;
- n envios de tarefas aos agentes: n*(108+som_is+(2*som_fi)); e
- n envios de um dado colhido: n*(81+59+tcv).

Com isso temos:

- Trânsito no gerente: n*(73+108+som_is+(2*som_fi)+81+59+tcv)
- Trânsito entre NEs: 0 (zero)
- Trânsito total da gerência: n*(73+108+som_is+(2*som_fi)+81+59+tcv)

7.4.3 Quadro resumido

Temos então um quadro resumindo os modelos matemáticos encontrados para a produção de bytes utilizando o KDEMA:

Tabela 15. Quadro resumido dos modelos matemáticos

Trânsito	Conf. a	Conf. b	Conf. c
No gerente	73+108+som_is+(2*som_fi)+81+(59+tcv)*n	73+108+som_is+(2*som_fi)+n*(81+59+tcv)	n*(73+108+som_is+(2*som_fi)+81+59+tcv)
Entre Nes	(n-1)*(135+(n/2)*(79+tcv)+som_is+(6*som_fi))	(n-1)*(135+som_is+(6*som_fi))	0
Na gerência	73+108+som_is+(2*som_fi)+81+(59+tcv)*n+(n-1)*(135+(n/2)*(79+tcv)+som_is+(6*som_fi))	73+108+som_is+(2*som_fi)+n*(81+59+tcv)+(n-1)*(135+som_is+(6*som_fi))	n*(73+108+som_is+(2*som_fi)+81+59+tcv)

7.5 Validação dos modelos encontrados

Com as equações definidas, podemos variar a quantidade de NEs envolvidos na gerência e aplicar em cada uma das configurações, obtendo os bytes envolvidos nas situações colocadas: entre gerente e NEs, entre os NEs e na gerência total. Para tanto, precisamos definir as variáveis “tcv”, “som_fi” e “som_is”, pois elas são particulares a cada tipo de gerência.

7.5.1 Ajuste das variáveis de contexto

Variáveis “tcv”, “som_fi” e “som_is”

O tamanho de “tcv” é definido pelo tamanho do nome da chave a ser gravado e pelo tamanho do dado coletado. Nessa validação o dado coletado nos NEs será o nome do sistema operacional, retornado pelo método getProperty(“os.name”) da classe System. Adotaremos gravar esse dado com o nome “osname” nos dando 6 bytes. O retorno coletado (no caso do computador usado para implementações) é “Windows 2000” nos dando 12 bytes. Com isso temos “tcv” como 18 bytes.

As variáveis “som_fi” e “som_is” são dependentes da configuração, ou seja, dependem da implementação da tarefa. Nesse caso precisamos implementar cada uma das tarefas para as respectivas configurações, apurando os valores necessários. Nos nossos estudos, encontramos:

- is= 24 + somatório_tamanhos_fragmentos + 4*(qtde_parâms+qtde_args)
- fi=6+qtde_parâms+qtde_args

A partir dessas equações, e observando as instruções utilizadas na implementação, definimos:

Tabela 16. Instruções para a configuração “a”

Label	Classe	Método	Instância	Pars	Args	Arg1	Arg2	fi	Tam	is
Dado1	Java.lang.System	GetProperty	_null_	0	1	os.name		7	47	75
null	_null_	SetData	_null_	0	2	osname	*Dado1	8	39	71
null	_null_	MoveAgent	_null_	0	1	Daemon2		7	36	64
Dado2	Java.lang.System	GetProperty	_null_	0	1	os.name		7	47	75
null	_null_	SetData	_null_	0	2	osname	*Dado2	8	39	71
null	_null_	MoveAgent	_null_	0	1	Daemon1		7	36	64
null	_null_	Goto	_null_	0	2	n/2	*Dado1	8	32	64
null	_null_	SendData	_null_	0	0			6	28	52
Total								58		536

Tabela 17. Instruções para a configuração “b”

Label	Classe	Método	Instância	Pars	Args	Arg1	Arg2	fi	Tam	is
Dado1	Java.lang.System	GetProperty	_null_	0	1	os.name		7	47	75
null	_null_	setData	_null_	0	2	osname	*Dado1	8	39	71
null	_null_	SendData	_null_	0	0			6	28	52
null	_null_	MoveAgent	_null_	0	1	Daemon2		7	36	64
Dado2	Java.lang.System	GetProperty	_null_	0	1	os.name		7	47	75
null	_null_	setData	_null_	0	2	osname	*Dado2	8	39	71
null	_null_	SendData	_null_	0	0			6	28	52
null	_null_	MoveAgent	_null_	0	1	Daemon1		7	36	64
null	_null_	Goto	_null_	0	2	n/2	*Dado1	8	32	64
Total								64		588

Tabela 18. Instruções para a configuração “c”

Label	Classe	Método	Instância	Pars	Args	Arg1	Arg2	fi	Tam	is
null	Java.lang.System	GetProperty	_null_	0	1	os.name		7	48	76
null	_null_	setData	_null_	0	2	osname	*Dado1	8	39	71
null	_null_	SendData	_null_	0	0			6	28	52
Total								21		199

7.5.2 Equações reduzidas à quantidade de NEs (n)

Considerando as fórmulas encontradas, podemos então substituir “tcv” por 18 e “som_fi” e “som_is” pelos valores encontrados acima. Temos as fórmulas para a validação do modelo, em função da quantidade de NEs (n):

Tabela 19. Fórmulas em função da quantidade de NEs

Trânsito	Conf. a	Conf. b	Conf. c
No gerente	914+77*n	897+n*158	n*580
Entre Nes	(n-1)*(1019+(n/2)*97)	(n-1)*1107	0
Na gerência	914+77*n+(n-1)*(1019+(n/2)*97)	897+n*158+(n-1)*1107	n*580

7.5.3 Aplicação das fórmulas definidas

A aplicação das fórmulas será feita em um intervalo formado pelos números pares de 2 a 20 NEs, para obtermos gráficos que nos transmitam idéias do comportamento das

configurações, em questões de escalabilidade. Após a substituição, nas fórmulas, do número de NEs, obtivemos os seguintes resultados:

Tabela 20. Dados obtidos com a aplicação das fórmulas

	No gerente			**Entre NEs**			**Na gerencia**		
NEs	**Conf a**	**Conf b**	**Conf c**	**Conf a**	**Conf b**	**Conf c**	**Conf a**	**Conf b**	**Conf c**
2	1.068	1.213	1.160	1.116	1.107	-	2.184	2.320	1.160
4	1.222	1.529	2.320	3.639	3.321	-	4.861	4.850	2.320
6	1.376	1.845	3.480	6.550	5.535	-	7.926	7.380	3.480
8	1.530	2.161	4.640	9.849	7.749	-	11.379	9.910	4.640
10	1.684	2.477	5.800	13.536	9.963	-	15.220	12.440	5.800
12	1.838	2.793	6.960	17.611	12.177	-	19.449	14.970	6.960
14	1.992	3.109	8.120	22.074	14.391	-	24.066	17.500	8.120
16	2.146	3.425	9.280	26.925	16.605	-	29.071	20.030	9.280
18	2.300	3.741	10.440	32.164	18.819	-	34.464	22.560	10.440
20	2.454	4.057	11.600	37.791	21.033	-	40.245	25.090	11.600

Esses dados são o resultado da aplicação dos modelos encontrados. Podemos validar esses resultados, comparando-os com os obtidos para 4 NEs, já implementados anteriormente. Contudo, para melhorar a avaliação, implementaremos também para 8 e 20 NEs, utilizando o KDEMA e comparando os resultados.

7.5.4 Implementação das configurações no KDEMA

As tarefas já foram definidas anteriormente. Precisamos implementá-las, executar a gerência, colher os resultados nas telas de Log e comparar com os já encontrados pelo modelo. Faremos a implementação, conforme já ocorrido para 4 NEs, mas desta vez para 8 e 20 NEs.

Após implementação das gerências, e aproveitando os resultados obtidos anteriormente para 4 NEs, obtivemos:

Tabela 21. Validação do modelo com dados práticos

Conf	**NEs**	**Consumo de bytes**								
		No gerente			**Entre NEs**			**Na gerência**		
		Prático	**Modelo**	**Erro (%)**	**Prático**	**Modelo**	**Erro (%)**	**Prático**	**Modelo**	**Erro (%)**
a	4	1207	1222	+1,24	3610	3639	+0,80	4817	4861	+0,91
	8	1511	1530	+1,26	9794	9849	+0,56	11305	11379	+0,65
	20	2448	2454	+0,25	37801	37791	-0,03	40249	40245	-0,01
b	4	1505	1529	+1,59	3298	3321	+0,70	4803	4850	+0,98
	8	2121	2161	+1,89	7722	7749	+0,35	9843	9910	+0,67
	20	4022	4057	+0,87	21053	21033	-0,09	25075	25090	+0,06
c	4	2272	2320	+2,11	0	0	0	2272	2320	+2,11
	8	4544	4640	+2,11	0	0	0	4544	4640	+2,11
	20	11360	11600	+2,11	0	0	0	11360	11600	+2,11

- A coluna “Prático” contém os resultados obtidos através dos Logs do KDEMA, após execução das configurações;
- A coluna “Modelo” contém os resultados obtidos através do modelo matemático proposto;
- A coluna “Erro (%)” contém o erro do modelo matemático, calculado da seguinte forma: erro=modelo/prático*100.

Percebemos com os resultados obtidos que os modelos definidos calculam com boa aproximação o comportamento de consumo de bytes. Os erros encontrados estão em intervalos máximos de +2,11% e –0,09%, com o erro mínimo detectado de –0,01%. Esses erros aparecem como conseqüência dos processos de redução das fórmulas, e são inexistentes sem essas reduções (100% de acerto entre os resultados teórico e prático).

Observamos também que os menores erros estão nos experimentos com maior quantidade de NEs (20), e que os maiores erros estão nos experimentos com menores quantidades de NEs (4). Isso pode ser explicado pela influência dos tamanhos dos nomes considerados (nome de origem, destino e fluxo) nas reduções das fórmulas. Esses nomes são controlados pela plataforma e em pequenas comunidades (até 9 NEs) os nomes têm um byte a menos que o considerado no modelo.

7.6 Conclusão

Com os resultados obtidos, podemos considerar que o modelo proposto representa o comportamento prático das configurações atendidas pelas equações do modelo, para os propósitos do trabalho. Podemos então utilizá-las em situações onde precisamos saber antecipadamente o comportamento de configurações de agentes móveis, dada uma gerência.

Podemos agora trabalhar em paradigmas que nos auxiliem na decisão da melhor configuração, dadas as características individuais encontradas pelo modelo para cada uma das configurações imaginadas.

8. Modelo de Classificação das Configurações

Com o modelo de cálculo de consumo de banda validado, podemos nos concentrar no estudo de um modelo de classificação das configurações. Com a aplicação direta do modelo de cálculo, variando a quantidade de NEs nas configurações consideradas, obtemos os respectivos consumos, com os quais podemos trabalhar as classificações. Este capítulo estuda alguns aspectos relevantes nessa etapa, e sugere a utilização de inteligência computacional para a resolução de alguns problemas.

8.1 Introdução

Como dito, o modelo de cálculo de consumo de banda encontrado nos apresenta valores absolutos que estão à disposição para qualquer método de classificação. A classificação, neste caso, nos parece uma tarefa simples de ordenação. Contudo, como veremos adiante, essa abordagem simples e direta pode não ser a mais adequada. Não podemos esquecer que essa classificação será suporte para o próximo modelo: o de decisão automatizada.

Iniciando o estudo, aplicaremos o modelo de cálculo de banda entre 2 e 20 NEs, observando os resultados encontrados. Em seguida, inserimos conceitos de inteligência computacional convenientemente utilizados. Por fim, implementamos as soluções propostas através de redes neurais.

8.2 Aplicação do modelo de equações nas configurações definidas

O cálculo dos consumos de banda para as três configurações, considerando os três escopos (“no gerente”, “entre Nes” e “na gerência”), e ainda considerando de 2 a 20 NEs gerenciados, é obtido através da aplicação direta das combinações dos parâmetros às equações adequadas, definidas na tabela 15. Esses resultados nos permitem desenhar gráficos, expostos em seguida:

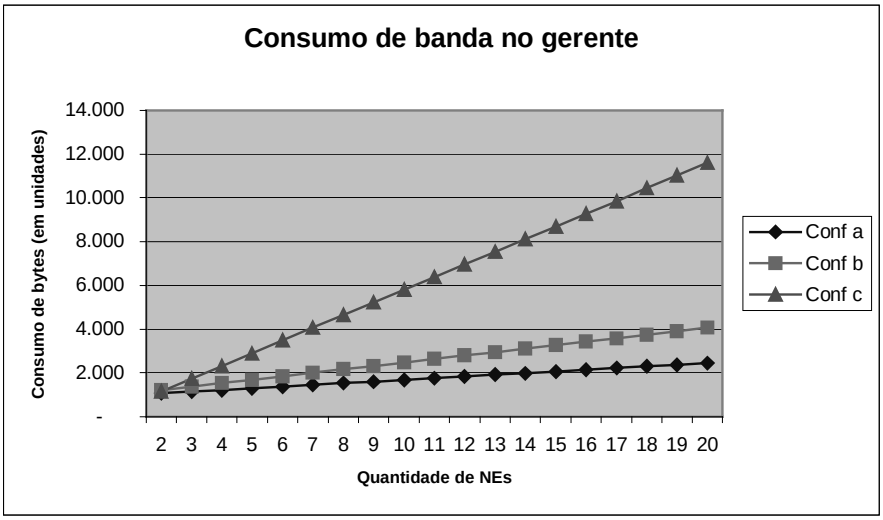

Figura 8. Gráfico do consumo de banda no gerente, obtido pelo modelo definido

Percebemos que a configuração “a” é a de menor consumo de banda, seguida pela configuração “b” (ambas com um agente apenas); a configuração “c” é notadamente a de maior consumo nesta métrica.

O consumo de banda, medido entre os NEs, está apresentado abaixo. Observamos, como previsto nas equações, que as configurações “b” e “c” apresentam retas, ao passo que a configuração “a” apresenta uma parábola:

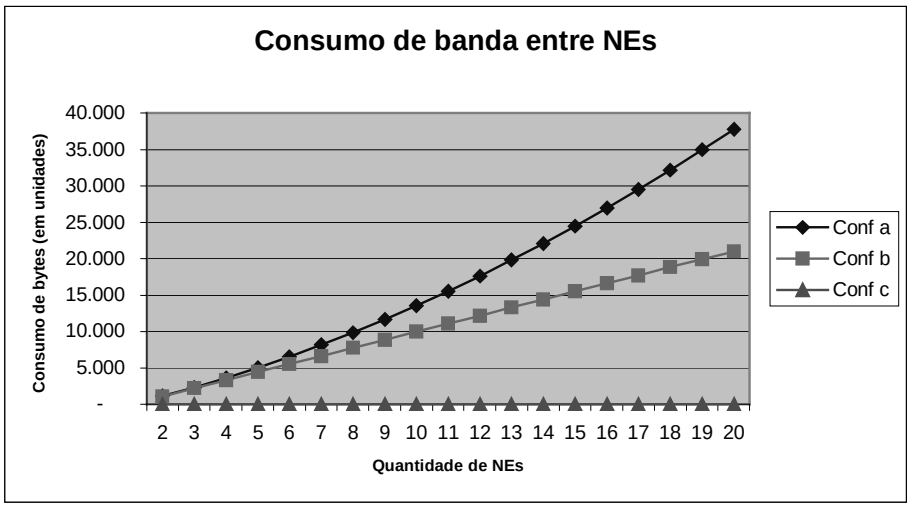

Figura 9. Gráfico do consumo de banda entre NEs, obtido pelo modelo definido

Verificamos que a configuração “a” é notadamente a maior consumidora de banda, seguida pela configuração “b”; a configuração “c” não possui trânsito de bytes entre NEs, pois não existe, nessa configuração, nenhuma comunicação entre os NEs.

Por fim, temos o consumo de banda na gerência, que compreende os consumos de banda no gerente e entre NEs. Percebemos, em conformidade com as equações, retas para as configurações “b” e “c”, e parábola para a configuração “a”:

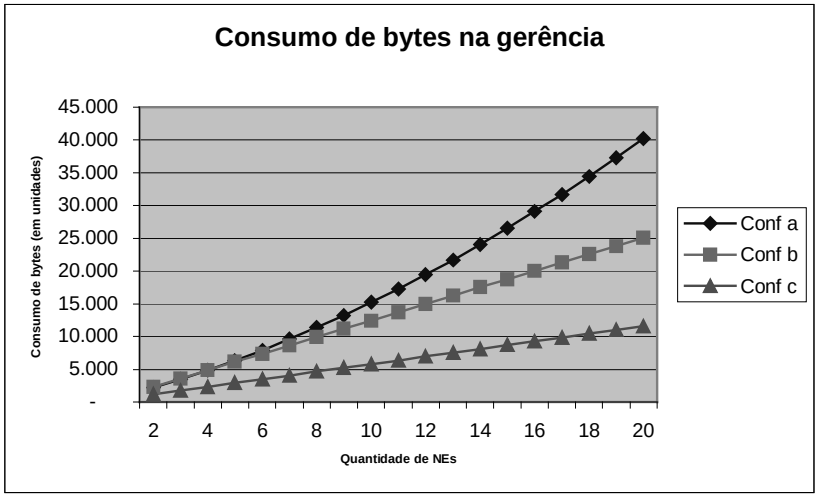

Figura 10. Gráfico do consumo de banda na gerência, obtido pelo modelo definido

Percebemos que a configuração “c” teve o menor consumo, seguida pela configuração “b” e por último, a configuração “a”.

É importante ressaltar que, como observado e de acordo com análise das equações do modelo, os gráficos desenhados correspondem a equações do primeiro grau (reta) ou do segundo grau (parábola com concavidade para baixo). Observando os gráficos e as equações, temos que as tendências, quando considerando NEs maior que 20, sempre serão essas, ou seja, as linhas nunca se cruzarão. Com isso concluímos que as observações feitas para esse intervalo estudado (de 2 a 20 NEs) servirão para qualquer outro, do ponto de vista de classificação das configurações.

8.3 Modelando a classificação das configurações usando inteligência computacional

Para auxilio à decisão de configuração mais adequada, precisamos desenvolver alguma forma de classificação, conforme quesitos utilizados pelo gerente humano. A

classificação imaginada, em um primeiro momento, seria por ordem crescente de consumo de bytes. Temos então, em um exemplo não experimentado, mas conveniente para a explicação, a seguinte situação:

Tabela 22. Exemplo de quadro de consumos de bytes

Configuração	No Gerente	entre NEs	Na Gerência
a	2.200	8.300	10.500
b	2.230	4.100	6.330
c	6.320	0	6.320

Utilizando o método de avaliação citado, teríamos:

- Se o objetivo fosse uma gerência com menor consumo de bytes no gerente, a configuração “a” seria a mais adequada, seguida pela configuração “b”;
- Se o objetivo fosse uma gerência com menor consumo de bytes na gerência, a configuração “c” seria a mais adequada, seguida pela configuração “b”.

8.3.1 O problema do modelo de classificação

Com essa forma de classificação, o modelo escolheria as configurações “a” e “c”, respectivamente para cada uma das duas situações citadas. Mas observando melhor, através de uma análise menos rígida, podemos perceber que:

- Se o objetivo fosse uma gerência com menor consumo de banda no gerente, poderíamos observar que, apesar de a configuração “a” consumir menos bytes, a configuração “b” seria uma forte candidata a ser a melhor opção, isso porque ela tem um consumo maior em apenas 30 bytes (pouco mais de 1%), além de possuir consumo “entre NEs” e “na gerência” consideravelmente menores, indo de encontro a um dos objetivos primordiais da gerência, que é o de sempre consumir menor banda;
- Analogamente ao item acima, teríamos que, se o objetivo fosse uma gerência com menor consumo de banda “na gerência”, poderíamos também eleger a configuração “b” mais adequada que a “c”, pois o consumo “no gerente” é consideravelmente menor, com praticamente o mesmo consumo “na gerência”.

Percebemos com esses exemplos que o modelo de classificação proposto inicialmente pode não corresponder ao utilizado normalmente pelo gerente humano, em seu cotidiano. Para o gerente humano permitir um modelo de decisão automatizada, este modelo precisa trabalhar em cima de parâmetros de decisão mais flexíveis, próximos aos utilizados pelo gerente humano.

8.3.2 Utilizando inteligência computacional através de lógica *fuzzy*

Podemos buscar na inteligência computacional, recursos que nos auxiliem na resolução deste problema. Como visto anteriormente, existem vários paradigmas de IA aplicáveis a problemas de classificação.

Observando o problema detectado, percebemos que a classificação absoluta é inadequada quando a diferença entre os valores é relativamente pequena. Podemos atribuir aos resultados de consumo encontrados, valores de um universo de 0 a 100%, com base no maior resultado, distribuídos proporcionalmente nos outros resultados, obtendo assim resultados relativos. Paralelo a isso podemos montar um gráfico que representa uma função de pertinência *fuzzy*, que classifica o consumo de bytes com as variáveis “muito baixo”, “baixo”, “médio”, “alto” ou “muito alto”.

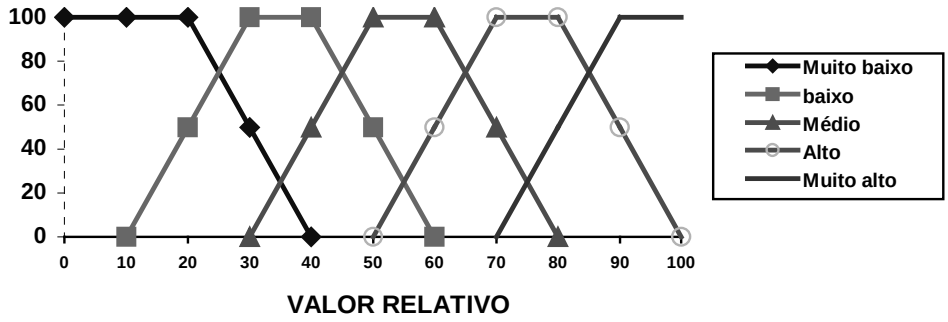

Figura 11. Função de pertinência *fuzzy* para classificação do consumo de bytes

Com o gráfico da função de pertinência adotado, podemos classificar cada um dos grupos de configurações de gerência, através da inferência dos valores relativos nesta função, obtendo as classificações percentuais de cada valor *fuzzy*.

Nesse exemplo hipotético, teríamos agora:

Tabela 23. Classificação das configurações de acordo com o modelo

Escopo	Conf	Valor absoluto	Valor relativo	Muito baixo	Baixo	Médio	Alto	Muito alto
No gerente	a	2200	35	25	100	25	0	0
	b	2230	35	25	100	25	0	0
	c	6320	100	0	0	0	0	100
Entre NEs	a	8300	100	0	0	0	0	100
	b	4100	49	0	55	95	0	0
	c	0	0	100	0	0	0	0
Na gerência	a	10500	100	0	0	0	0	100
	b	6330	60	0	0	100	50	0
	c	6320	60	0	0	100	50	0

Na tabela acima temos:

- A coluna “valor absoluto” representa o resultado da aplicação do modelo matemático encontrado, dada uma configuração e um escopo;
- A coluna “valor relativo”, em percentual, retorna 100% para o maior valor ocorrido dentro do escopo, e valores proporcionais a esse para os outros valores absolutos existentes;
- As colunas “muito baixo”, “baixo”, “médio”, “alto” e “muito alto”, também em percentual, são o resultado da aplicação do valor relativo na função de pertinência.

Com esse novo modelo de classificações, vemos situações tais como ditas no inicio deste capítulo, aparentemente resolvidas. Senão vejamos:

- Se o objetivo fosse uma gerência com o menor consumo de bytes “no gerente”, o novo modelo de classificação faria a opção pela configuração “b”, visto que “b” e “a” são juntas “25% muito baixo”, “100% baixo” e “25% médio”, mas para desempate, “b” possui “médio” e “a” possui “muito alto” para os outros escopos;
- Se o objetivo fosse uma gerência com o menor consumo de bytes “na gerência”, a opção também seria “b”, por critérios análogos aos citados acima;

Voltando à tabela experimentada das gerências para 4, 8 e 20 NEs, temos:

Tabela 24. *Fuzzificação* dos resultados relativos para gerência com 4 NEs

Escopo	Conf	Valor absoluto	Valor relativo	Muito baixo	Baixo	Médio	Alto	Muito alto
No gerente	a	1.222	53	0	35	100	15	0
	b	1.529	66	0	0	70	80	0
	c	2.320	100	0	0	0	0	100
Entre NEs	a	3.639	100	0	0	0	0	100
	b	3.321	91	0	0	0	45	100
	c	0	0	100	0	0	0	0
Na gerência	a	4.861	100	0	0	0	0	100
	b	4.850	100	0	0	0	0	100
	c	2.320	48	0	60	90	0	0

Tabela 25. *Fuzzificação* dos resultados relativos para gerência com 8 NEs

Escopo	Conf	Valor absoluto	Valor relativo	Muito baixo	Baixo	Médio	Alto	Muito alto
No gerente	a	1.530	33	15	35	100	0	0
	b	2.161	47	0	85	65	0	0
	c	4.640	100	0	0	0	0	100
Entre NEs	a	9.849	100	0	0	0	0	100
	b	7.749	79	0	0	5	100	45
	c	0	0	100	0	0	0	0
Na gerência	a	11.379	100	0	0	0	0	100
	b	9.910	87	0	0	0	65	85
	c	4.640	41	0	95	55	0	0

Tabela 26. *Fuzzificação* dos resultados relativos para gerência com 20 NEs

Escopo	Conf	Valor absoluto	Valor relativo	Muito baixo	Baixo	Médio	Alto	Muito alto
No gerente	a	2.454	21	95	55	0	0	0
	b	4.057	35	25	100	25	0	0
	c	11.600	100	0	0	0	0	100
Entre NEs	a	37.791	100	0	0	0	0	100
	b	21.033	56	0	20	100	30	0
	c	0	0	100	0	0	0	0
Na gerência	a	40.245	100	0	0	0	0	100
	b	25.090	63	0	0	85	65	0
	c	11.600	29	55	95	0	0	0

Com esse modelo de classificações proposto, podemos nos dispor de recursos de inteligência computacional para montar sistemas de classificação automatizados, que tenham como entrada a quantidade de NEs e os parâmetros necessários às equações, e que retornem as classificações conforme apresentadas acima.

8.4 Implementando a classificação automatizada por redes neurais

Uma das grandes áreas de aplicação das redes neurais é a classificação. As redes neurais, como visto na revisão teórica, podem ser treinadas para situações que envolvem classificação, com resultados interessantes. Podemos levantar as variáveis envolvidas, desde o modelo de equações, necessárias para um sistema de classificação por redes neurais.

8.4.1 Variáveis de entrada e saída da rede neural

Ao analisar o modelo de equações proposto, verificamos que elas estão em função de três variáveis:

- Número de NEs envolvido na gerência; se os gráficos de todas as equações tivessem comportamentos de retas, esse parâmetro poderia ser desconsiderado, pois os valores relativos seriam sempre os mesmos; porém como a configuração “a” possui curvas em dois dos escopos, precisamos considerar essa variável;
- “som_fi”, que é o somatório da quantidade de fragmentos string das instruções das tarefas que o agente vai executar; temos que informar essa variável para as 3 configurações;
- “som_is”, que é o somatório dos tamanhos dos fragmentos string das instruções das tarefas que o agente vai executar; temos que informar essa variável para as 3 configurações;
- “tcv”, que é o tamanho dos valores “chave/valor”; o tamanho de “chave” é definido pelo gerente quando da montagem das instruções, na utilização da instrução “setData”; já “valor” é o valor que será coletado pelo agente e, portanto, o seu tamanho depende de qual informação será coletada; e
- “escopo”, que diz em qual contexto devemos verificar a classificação; os escopos são “no gerente”, “entre NEs” e “na gerência”.

Por outro lado, para essa implementação, desejamos como saída, valores, em percentual, das variáveis “muito baixo”, “baixo”, “médio”, “alto” e “muito alto” para cada uma das configurações (“a”, “b” e “c”), o que nos apresenta 15 variáveis de saída;

A entrada “escopo” foi convenientemente utilizada como entrada para dividir a resposta da rede para cada um dos escopos por vez. Se essa variável não tivesse sido colocada na entrada, teríamos 45 variáveis de saída, sendo 15 saídas por escopo.

Para implementar uma rede neural, precisamos de um conjunto de dados que sejam compostos das entradas e das saídas previamente calculadas. Em seguida separamos a maior parte para a fase de treinamento e o restante para a fase de testes. Para obtermos esses dados, podemos estabelecer intervalos de valores para cada uma

das variáveis de entrada, submetendo esses dados aos modelos levantados (equações e classificação).

8.4.2 Delimitação dos intervalos considerados para as variáveis

Para criar os dados sobre os quais a rede será treinada, precisamos definir os valores mínimo e máximo de cada uma das variáveis da rede (tanto entrada quanto saída). Esses valores (mínimo e máximo) devem cobrir todas as situações para as quais esperamos que a rede faça a classificação. Podemos treinar redes neurais com intervalos enormes, para que ela atenda a praticamente todas as possibilidades de classificação possíveis, considerando valores consistentes. Contudo, para esse estudo, definiremos intervalos limitados à cobertura dos exemplos postos, tendo como objetivo, apenas a demonstração da classificação. Definimos então os seguintes intervalos:

- Número de NEs: universo composto pelos números de 2 a 20, em passos de 2;
- "som_fi": considerado 58 para a configuração "a", 64 para a configuração "b" e 21 para a configuração "c";
- "som_is": considerado 536 para a configuração "a", 588 para a configuração "b" e 199 para a configuração "c";
- "escopo": de 1 a 3, onde 1 representa "no gerente", 2 representa "entre NEs" e 3 representa "na gerência".

Com isso obtemos a tabela seguinte:

Tabela 27. Variáveis de entrada e seus intervalos

Variável	**Intervalo**	**Quantidade**
"NEs"	2,4,6,8,10,12,14,16,18 e 20	10
"som_fi" – conf a	58	1
"som_is" – conf a	536	1
"som_fi" – conf b	64	1
"som_is" – conf b	588	1
"som_fi" – conf c	21	1
"som_is" – conf c	199	1
"escopo"	1,2,3	3
"tcv"	18	1

Observando a tabela, temos 10*3 possibilidades combinatórias, e poderemos gerar sem repetição um conjunto de 30 possibilidades. Podemos ainda constatar que algumas

variáveis estão consideradas como fixas e, portanto, não precisaremos informa-las à entrada da rede.

8.4.3 Normalização das variáveis de entrada e saída

É importante para a rede que as variáveis de entrada e de saída sempre sejam informadas ou esperadas dentro do intervalo [0,1;0,9], por características da função de transferência, sigmoidal, que não possui valores f(x)=0 e f(x)=1. A fórmula utilizada para normalização é:

- N=((0,8*(V-Vmin))/(Vmax-Vmin))+0,1, onde:
- N: valor já normalizado;
- V: valor a ser normalizado;
- Vmin: valor mínimo que poderá ocorrer no universo de valores a ser normalizado;
- Vmax: valor máximo que poderá ocorrer no universo de valores a ser normalizado.

Vamos então estabelecer, de acordo com os intervalos propostos, as fórmulas de normalização de cada variável de entrada:

Tabela 28. Fórmulas de normalização das variáveis de entrada

Variável	Valor inicial	Valor final	Fórmula de normalização
“NEs”	2	20	N=((0,8*(V-2))/18)+0,1
“escopo”	1	3	N=((0,8*(V-1))/2)+0,1

Da mesma forma, temos o quadro de normalização das variáveis de saída. Como as variáveis são todas em percentual, variando de 0 a 100%, temos uma única fórmula que serve para todas as saídas:

Tabela 29. Fórmulas de normalização das variáveis de saída

Variável	Valor inicial	Valor final	Fórmula de normalização
Saída	0	100	N=((0,8*V)/100)+0,1

Com as tabelas de normalização, podemos gerar os dados de entrada com os respectivos valores de saída, para utilização no treinamento da rede.

8.4.4 Produção dos dados para treinamento e teste

Podemos, através das tabelas de intervalos considerados para as entradas, gerar todas as combinações possíveis. Em seguida podemos submeter essas entradas às equações matemáticas de cálculo de consumo de bytes, obtendo o consumo de banda de cada uma das combinações. Em seguida podemos submeter esses resultados ao modelo de classificação, obtendo as respectivas classificações. Com isso teremos todas as combinações de entrada e saída possíveis, dentro dos intervalos considerados. Em seguida submeteremos esses dados ao processo de treinamento.

8.4.5 Fase de treinamento da rede neural

Para essa etapa precisamos normalizar, conforme fórmulas descritas anteriormente, todas as entradas e saídas do conjunto de dados que será submetido ao treinamento. Essa normalização é gerada pela aplicação das fórmulas normalizadoras diretamente nos dados a serem apresentados à rede. Observamos que os dados resultantes são um conjunto de valores, todos compreendidos entre 0,1 e 0,9.

Para os experimentos em rede neural utilizamos o software QwikNet v2.23. Nesse software (em versão demonstração) podemos utilizar um máximo de 500 conjuntos de entrada para treinamento e somente uma camada oculta. O arquivo de treinamento, em formato texto, contém uma cláusula [INPUTS] seguida do número de entradas, uma cláusula [OUTPUTS] seguida do número de saídas e em seguida os dados normalizados, separados por espaço ou quebra de linha.

Em seguida, geramos através de planilha de cálculos, todas as 30 combinações de dados de entrada, obtendo as 15 saídas correspondentes. Ainda em planilha, submetemos todos esses dados (2 entradas e 15 saídas) às fórmulas de normalização, onde obtemos todos os dados normalizados. Em seguida gravamos esses dados em um arquivo tipo texto, no formato descrito anteriormente.

Observamos que esse arquivo possui, conforme previsto, números que estão no intervalo de 0,1 e 0,9. Observamos ainda que esse arquivo possui a extensão trn, padrão do software de treinamento e testes. Em seguida submetemos esse arquivo ao software, conforme apresentado abaixo:

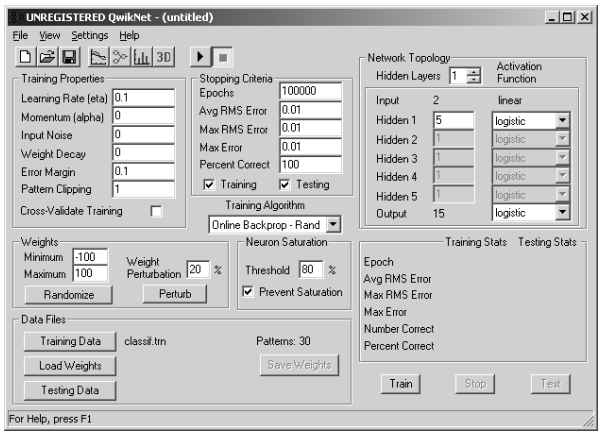

Figura 12. Tela do software usado para treinamento e teste da rede neural

Após a carga do software, carregamos os dados para treinamento através da opção "*Training Data*". Em seguida ajustamos alguns parâmetros importantes tais como função de ativação, critérios de parada e algoritmo de treinamento. Observamos que o software reconheceu as cláusulas [INPUTS] e [OUTPUTS] corretamente, apontando 2 entradas e 15 saídas. Em seguida submetemos a rede ao treinamento:

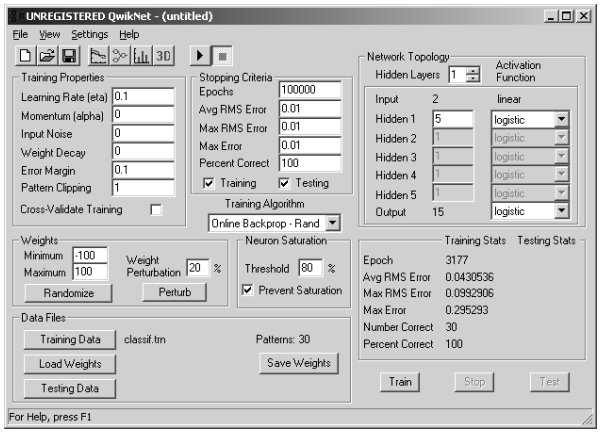

Figura 13. Resultados do treinamento da rede neural

Obtivemos então o treinamento da rede com o resultado de 100% de acerto, conforme visto na figura. Em seguida gravamos esses pesos da rede (parâmetros necessários para futuras utilizações desse treinamento) em um arquivo chamado "classif.wts".

Nesse ponto temos uma rede neural treinada para responder por qualquer entrada, dentro dos limites estabelecidos, nos retornando as classificações das configurações “a”, “b” e “c”.

8.4.6 Fase de testes da rede neural

Podemos submeter quaisquer dados compreendidos entre os intervalos considerados. Vamos submeter, para os três escopos, os valores 4, 8 e 20, que foram exemplificados e experimentados em capítulos anteriores. Vamos também experimentar os valores 5, 10 e 15, compreendidos entre o intervalo, mas não aplicados na fase de testes da rede neural, para testar sua generalidade[5].

Para submeter os dados, passamos pelos mesmos processos já vistos na fase de treinamento, mas desta vez com os dados citados acima. Após todos os processos e submissão à rede, obtivemos os seguintes resultados:

Tabela 30. Configuração a: um agente com dados ao final da gerência

		“Muito Baixo”		**“Baixo”**		**“Médio”**		**“Alto”**		**“Muito Alto”**	
Escopo	**NEs**	**Rede 1**	**Calc 1**	**Rede 2**	**Calc 2**	**Rede 3**	**Calc 3**	**Rede 4**	**Calc 4**	**Rede 5**	**Calc 5**
No gerente	4	1,43	0,00	44,20	36,64	63,09	100,00	13,48	13,36	14,50	0,00
	8	29,40	35,13	89,12	100,00	33,72	14,87	1,02	0,00	0,67	0,00
	20	87,27	94,22	60,30	55,78	0,75	0,00	0,34	0,00	0,03	0,00
	5	3,74	0,00	69,56	76,03	62,49	73,97	5,29	0,00	5,26	0,00
	10	53,71	54,83	86,56	95,17	14,00	0,00	0,65	0,00	0,26	0,00
	15	83,51	81,09	68,27	68,91	1,47	0,00	0,39	0,00	0,05	0,00
Entre NEs	4	0,01	0,00	0,00	0,00	0,63	0,00	2,11	0,00	99,99	100,00
	8	0,02	0,00	0,15	0,00	5,34	0,00	0,05	0,00	99,69	100,00
	20	1,24	0,00	2,23	0,00	0,65	0,00	0,00	0,00	96,07	100,00
	5	0,01	0,00	0,01	0,00	1,38	0,00	0,67	0,00	99,97	100,00
	10	0,08	0,00	0,48	0,00	4,49	0,00	0,02	0,00	99,24	100,00
	15	0,89	0,00	4,21	0,00	2,00	0,00	0,00	0,00	96,06	100,00
Na gerência	4	0,00	0,00	0,00	0,00	0,06	0,00	11,30	0,00	100,00	100,00
	8	0,00	0,00	0,00	0,00	0,40	0,00	0,90	0,00	100,00	100,00
	20	0,11	0,00	0,49	0,00	0,57	0,00	0,01	0,00	99,81	100,00
	5	0,00	0,00	0,00	0,00	0,07	0,00	9,91	0,00	100,00	100,00
	10	0,00	0,00	0,01	0,00	1,12	0,00	0,20	0,00	99,99	100,00
	15	0,01	0,00	0,03	0,00	1,10	0,00	0,07	0,00	99,98	100,00

[5] Em uma fase de treinamento exaustivo, a rede neural pode perder a característica de generalidade, ou seja, só consegue classificar corretamente dados de entrada que estiveram em sua fase de treinamento. No caso deste trabalho, a generalidade é altamente desejada.

Tabela 31. Configuração b: um agente com dados imediatamente após a coleta

		"Muito Baixo"		"Baixo"		"Médio"		"Alto"		"Muito Alto"	
Escopo	**NEs**	**Rede 6**	**Calc 6**	**Rede 7**	**Calc 7**	**Rede 8**	**Calc 8**	**Rede 9**	**Calc 9**	**Rede 10**	**Calc 10**
No gerente	4	0,01	0,00	5,16	0,00	59,48	70,47	59,53	79,53	9,63	0,00
	8	0,22	0,00	68,82	67,13	89,77	82,87	10,27	0,00	0,16	0,00
	20	19,14	25,13	98,81	100,00	23,52	24,87	0,03	0,00	0,03	0,00
	5	0,02	0,00	14,19	9,14	78,74	100,00	53,72	40,86	2,10	0,00
	10	1,15	0,00	87,62	86,47	81,73	63,53	2,03	0,00	0,08	0,00
	15	14,09	12,24	98,01	100,00	36,38	37,76	0,08	0,00	0,03	0,00
Entre NEs	4	0,00	0,00	0,03	0,00	0,68	0,00	37,90	43,69	99,17	100,00
	8	0,00	0,00	0,15	0,00	11,70	6,61	97,18	100,00	38,23	43,39
	20	0,01	0,00	10,62	21,72	81,90	100,00	32,34	28,28	0,30	0,00
	5	0,00	0,00	0,04	0,00	1,60	0,00	73,48	61,24	95,90	88,76
	10	0,00	0,00	0,49	0,00	31,34	31,98	94,46	100,00	11,63	18,02
	15	0,00	0,00	6,67	0,00	85,79	83,08	64,55	66,92	0,55	0,00
Na gerência	4	0,00	0,00	0,00	0,00	0,75	0,00	7,84	1,13	99,69	100,00
	8	0,00	0,00	0,01	0,00	5,30	0,00	77,34	64,55	87,79	85,45
	20	0,00	0,00	1,38	0,00	92,50	88,28	59,97	61,72	0,47	0,00
	5	0,00	0,00	0,00	0,00	0,85	0,00	9,59	18,12	99,61	100,00
	10	0,00	0,00	0,02	0,00	14,43	0,00	96,12	91,33	47,15	58,67
	15	0,00	0,00	0,06	0,00	37,18	46,21	94,26	100,00	14,72	3,79

Tabela 32. Configuração c: um agente para cada NE

		"Muito Baixo"		"Baixo"		"Médio"		"Alto"		"Muito Alto"	
Escopo	**NEs**	**Rede 11**	**Calc 11**	**Rede 12**	**Calc 12**	**Rede 13**	**Calc 13**	**Rede 14**	**Calc 14**	**Rede 15**	**Calc 15**
No gerente	4	3,56	0,00	2,39	0,00	0,48	0,00	2,73	0,00	96,45	100,00
	8	5,22	0,00	1,15	0,00	0,06	0,00	0,14	0,00	97,87	100,00
	20	0,24	0,00	0,06	0,00	0,01	0,00	0,04	0,00	99,81	100,00
	5	5,12	0,00	2,45	0,00	0,23	0,00	0,90	0,00	96,71	100,00
	10	2,60	0,00	0,52	0,00	0,03	0,00	0,08	0,00	98,83	100,00
	15	0,41	0,00	0,10	0,00	0,01	0,00	0,04	0,00	99,71	100,00
Entre NEs	4	94,87	100,00	0,26	0,00	3,16	0,00	0,55	0,00	2,05	0,00
	8	98,64	100,00	1,78	0,00	0,27	0,00	0,01	0,00	2,78	0,00
	20	97,18	100,00	7,02	0,00	0,12	0,00	0,00	0,00	1,03	0,00
	5	96,58	100,00	0,50	0,00	1,44	0,00	0,15	0,00	2,35	0,00
	10	99,12	100,00	1,97	0,00	0,15	0,00	0,00	0,00	2,31	0,00
	15	99,47	100,00	2,87	0,00	0,07	0,00	0,00	0,00	1,29	0,00
Na gerência	4	4,89	0,00	63,69	61,37	87,58	88,63	0,30	0,00	0,08	0,00
	8	11,00	0,00	90,72	96,12	54,11	53,88	0,01	0,00	0,10	0,00
	20	59,03	55,88	94,84	94,12	4,58	0,00	0,00	0,00	0,07	0,00
	5	4,94	0,00	67,29	71,47	86,76	78,53	0,25	0,00	0,07	0,00
	10	17,37	9,46	95,90	100,00	29,88	40,54	0,00	0,00	0,12	0,00
	15	26,45	35,97	96,31	100,00	17,75	14,03	0,00	0,00	0,11	0,00

Nos cabe ressaltar que os dados apresentados pela rede nas colunas "Rede", bem como os dados calculados pelos modelos, apresentados nas colunas "Calc" já estão com medidas reais, ou seja, foram aplicadas nos dados retornadas pela rede as fórmulas inversas às das normalizações. Destacamos ainda que as 15 saídas foram obtidas ao mesmo tempo; isso é importante e até necessário, já que a classificação se baseia em valores relativos, ou seja, o processo de classificação precisa dos valores de todas as configurações (escopos "a", "b" e "c") para oferecer os resultados.

Percebemos com esses dados que a rede soube reconhecer e classificar as configurações, mesmo para dados que não foram apresentados a ela (5, 10 e 15 NEs), confirmando a sua capacidade de generalizar a classificação.

8.5 Conclusão

Com essa implementação, verificamos a aplicabilidade conveniente e interessante das redes neurais para a classificação das configurações de agentes móveis, conforme o modelo proposto no capítulo anterior.

Com o modelo de classificação proposto, seguido por uma implementação por redes neurais, encerramos o processo de classificação das configurações de agentes móveis. Vamos, a seguir, estudar a avaliação das classificações e modelos de escolha da mais adequada.

9. Modelo de Decisão da Configuração mais Adequada

A partir do modelo de classificação de configurações proposto, podemos obter decisões sobre qual a configuração mais adequada. Essa decisão é o objetivo maior do trabalho e utiliza todos os resultados dos modelos definidos nos capítulos anteriores.

9.1 Introdução

Para tomar a decisão de qual a configuração mais adequada, o gerente humano ainda precisa de um parâmetro: o seu desejo considerando o conjunto de classificação das configurações.

Se a decisão depende de um contexto ou alguma particularidade, conhecida ou avaliada por critérios não modeláveis ou não dedutíveis, podemos oferecer ao gerente humano as classificações apresentadas nos capítulos anteriores, deixando para ele a decisão final sobre qual será a gerência mais adequada.

Por outro lado, se o desejo obedece a alguma lógica previsível e implementável, podemos construir sistemas de inferência, apoiados em paradigmas de inteligência computacional (regras de inferência, árvores de decisão, redes neurais ou algoritmos genéticos), para nos ajudar nessa decisão.

9.2 Modelando a decisão por regras de inferência

O modelo de melhor decisão poderia ser proposto e implementado em uma seqüência de regras de inferência ou árvore de decisão do tipo: “se expressão então reação”. Para montar essas regras de inferência baseadas no modelo de classificações, iniciamos definindo os seguintes conceitos:

- Escopo: define o escopo da avaliação da regra; pode ser “no gerente”, “entre NEs” ou “na gerência”;
- Valor: pode ser um resultado esperado, como “muito baixo”, “baixo”, “médio”, “alto” ou “muito alto”;
- Operador relacional: pode ser “igual a”, “diferente de”, “maior que” ou “menor que”.

O objetivo desses conceitos é permitir, através de combinações que obedeçam a determinadas semânticas, a construção de regras de inferência que expressem algum

critério de avaliação das classificações das configurações de comunidades de agentes móveis. Essas condicionais dão origem às expressões, formadas por um escopo, um operador relacional e um valor. São exemplos de expressões:

- No gerente igual a baixo; ou
- Entre NEs diferente de alto.

Implementamos ainda expressões mais complexas, separadas por operadores *booleanos* (e, ou), nos permitindo expressões tais como:

- (no gerente igual a muito baixo) ou (no gerente igual a médio); ou ainda
- (entre NEs diferente de alto) e (no gerente diferente de alto).

Por fim temos as reações, que apontam a conclusão da inferência, no caso de a expressão ser avaliada como "a mais" verdadeira. Temos então as regras de inferência do tipo:

- Se (no gerente igual a baixo) então (configuração adequada); ou
- Se (no gerente igual a médio) e (entre NEs igual a médio) então (configuração inadequada).

Com os conceitos apresentados, a construção das regras já nos permite representar alguns critérios interessantes utilizados na decisão da melhor configuração. Mas quando duas ou mais regras retornarem "configuração adequada", precisamos de algum critério de desempate. Introduzimos o conceito de valores das regras[6], onde utilizamos os percentuais, obtidos no modelo de classificação, das variáveis "muito baixo", "baixo", "médio", "alto" e "muito alto". Na avaliação das regras, cada expressão retorna o seu percentual de verdade. Na decisão final, prevalece aquela regra em que o percentual de verdade é o mais alto.

Temos como exemplo conveniente as seguintes regras de inferência:

a) se (no gerente igual a muito baixo) então (configuração adequada);

b) se (na gerência igual a muito baixo) então (configuração adequada).

[6] O conceito de cálculos de valores das expressões é utilizado na Lógica Fuzzy, no processo defuzzyficação, como apresentado na revisão teórica.

Essas regras, na tabela para 20 NEs, nos retornam os seguintes resultados:

a) 95% para a config “a”, 25% para a config “b” e 0% para a config “c”;

b) 0% para a config “a”, 0% para a config “b” e 55% para a config “c”.

Percebemos que a regra “a” nos retornou 95% para a configuração “a” e a regra “b” nos retornou 55% para a configuração “c”. Teríamos então a regra “a”, aplicada sob a configuração “a”, como a mais verdadeira, de forma que esta configuração seria a escolhida.

Com esses recursos temos condições de montar regras de inferência interessantes, e talvez suficientes para a maioria das necessidades de decisão automatizadas. Mas um problema ainda existente com esses recursos está no fato de que as condicionais só podem ser montadas em cima de operadores relacionais de igualdade ou não igualdade. Quando precisarmos criar uma regra que diga que em determinado escopo desejamos um consumo classificado como “igual ou menor que médio”, teremos que criar três regras: uma com a comparação “igual a médio”, outra com “igual a baixo” e por fim uma com “igual a muito baixo”.

Por isso vamos também utilizar, nas expressões, operadores relacionais complexos tais como “maior que” ou “menor que”, que nos retorna o maior percentual atendido pelo operador. Quando “menor que alto” nos retornar 20%, 75% e 50%, respectivamente para “médio”, “baixo” ou “muito baixo”, prevalecerá o percentual de 75% (o maior deles). Senão, vejamos o exemplo:

a) se (no gerente menor que médio) então (conf adequada);

b) se (na gerência igual a muito baixo) então (conf adequada).

Considerando experimentações anteriores, temos as seguintes avaliações:

a) 95% para a config “a”, 100% para a config “b”, 0% para a config “c”;

b) 0% para a config “a”, 0% para a config “b” e 55% para a config “c”.

Nesse novo conjunto de regras, teríamos a configuração “b”, na regra “a”, como a mais verdadeira e, portanto, a selecionada.

Por fim, introduzimos o conceito de expressões complexas, onde temos expressões simples separadas por operadores lógicos “ou” ou “e”. No caso do operador

“ou”, temos o maior percentual, entre os operandos, selecionado; já no caso do operador “e”, temos o menor. Por exemplo:

a) (no gerente igual a muito baixo) “ou” (entre NEs igual a baixo) então (config adequada);

b) (no gerente menor que baixo) “e” (entre NEs menor que médio) então (config adequada).

Teríamos as seguintes avaliações:

a) 95% “ou” 20% para a config “a”, 25% “ou” 20% para a config “b” e 0% “ou” 0% para a config “c”, donde selecionamos a config “a”, com 95%;

b) 95% “e” 0% para a config “a”, 100% “e” 100% para a config “b” e 0% “e” 100% para a config “c”, nos selecionando a config “b”, com 100%.

Com essas regras e seus percentuais levantados, teríamos a configuração “b” selecionada, com 100% de verdade na regra “b”.

Finalizando, percebemos que a questão de escalabilidade é muito importante, pois as classificações relativas das configurações, conforme modelo de classificações, variam conforme a quantidade de NEs. Temos, por exemplo conveniente, a avaliação das últimas regras citadas acima, mas agora para 4 NEs. Teremos:

a) 0% “ou” 0% para a config “a”, 0% “ou” 0% para a config “b” e 0% “ou” 0% para a config “c”, não tendo nenhuma configuração selecionada;

b) 35% “e” 0% para a config “a”, 0% “e” 0% para a config “b” e 0% “e” 0% para a config “c”, também não selecionando nenhuma configuração verdadeira.

Neste caso o gerente humano seria convidado a rever a construção das regras de inferência, ou optar por uma das configurações por outros critérios, ao contrário da avaliação para 20 NEs, onde uma regra foi seguramente eleita.

9.2.1 Validando o modelo de decisão por cálculos das regras

Para a utilização do modelo, precisamos definir as regras sob as quais a decisão será tomada. Essas regras devem tentar espelhar o gerente humano, simulando seu raciocínio na tomada de decisão.

Definiremos regras convenientes, obtidas através do seguinte exemplo de contexto:

- A máquina gerente está localizada em uma rede “x”;
- Os NEs, em uma quantidade de 20, estão em outra rede “y”, que é de grande disponibilidade de banda e baixa latência (por exemplo, uma rede Ethernet 100Mbits);
- O caminho de “x” para “y” possui uma alta latência e uma pequena disponibilidade de banda, além de ser de alto custo[7];

Esse contexto nos permite montar as seguintes regras:

- se (no gerente menor que baixo) então (configuração adequada);
- se (no gerente menor que médio) e (na gerência menor que alto) então (configuração adequada);

Ressaltamos que o contexto e as regras foram imaginados. O modelo foi concebido para ser aplicado a qualquer contexto, com quaisquer regras.

Aplicando o modelo de decisão para essas regras, e considerando os intervalos apresentados em 8.3.2 com os conseqüentes resultados de classificação apresentados em 8.3.6, temos:

[7] Podemos supor aqui enlaces tarifados por bytes transitados ou por pacotes, onde a economia de banda consumida é relevante.

Tabela 33. Modelo de decisão

NEs	Config	Regra1	Regra2			Melhor valor da configuração	Melhor valor encontrado	Melhor configuração apontada
		Expr 1	Expr 1	Expr 2	Operador "e"			
		Valor	Valor	Valor	Valor			
2	a	0,00	0,00	0,00	0,00	0,00	0,00	
2	b	0,00	0,00	0,00	0,00	0,00		
2	c	0,00	0,00	100,00	0,00	0,00		
4	a	0,00	36,64	0,00	0,00	0,00	0,00	
4	b	0,00	0,00	0,00	0,00	0,00		
4	c	0,00	0,00	88,63	0,00	0,00		
6	a	2,30	100,00	0,00	0,00	2,30	2,30	a
6	b	0,00	34,91	0,00	0,00	0,00		
6	c	0,00	0,00	80,47	0,00	0,00		
8	a	35,13	100,00	0,00	0,00	35,13	35,13	a
8	b	0,00	67,13	0,00	0,00	0,00		
8	c	0,00	0,00	96,12	0,00	0,00		
10	a	54,83	95,17	0,00	0,00	54,83	54,83	a
10	b	0,00	86,47	0,00	0,00	0,00		
10	c	0,00	0,00	100,00	0,00	0,00		
12	a	67,96	82,04	0,00	0,00	67,96	67,96	a
12	b	0,00	99,35	15,15	15,15	15,15		
12	c	0,00	0,00	100,00	0,00	0,00		
14	a	77,34	77,34	0,00	0,00	77,34	77,34	a
14	b	8,56	100,00	36,42	36,42	36,42		
14	c	0,00	0,00	100,00	0,00	0,00		
16	a	84,38	84,38	0,00	0,00	84,38	84,38	a
16	b	15,46	100,00	55,50	55,50	55,50		
16	c	0,00	0,00	100,00	0,00	0,00		
18	a	89,85	89,85	0,00	0,00	89,85	89,85	a
18	b	20,83	100,00	72,70	72,70	72,70		
18	c	0,00	0,00	100,00	0,00	0,00		
20	a	94,22	94,22	0,00	0,00	94,22	94,22	a
20	b	25,13	100,00	88,28	88,28	88,28		
20	c	0,00	0,00	94,12	0,00	0,00		

Onde:

- NEs: indica a quantidade de NEs considerada para o cálculo;
- Config: indica a configuração considerada;
- Expr1, Expr2: indicam os melhores valores (em percentual) da classificação obtida pelo modelo de classificações (e não pela rede neural); percebemos que Regra1 possui somente uma expressão, ao passo que Regra2 possui duas expressões;

- Operador “e”: como Regra2 é formada por duas expressões relacionadas por um operador “e”, esta coluna trará o menor valor entre as duas expressões;
- Melhor valor da config: verifica o melhor valor alcançado pela configuração, dentre as regras disponíveis;
- Melhor configuração apontada: verifica, dentre as três configurações, a que teve o melhor valor, e, portanto, será a escolhida.

Com a observação das respostas do modelo, apresentadas na tabela acima, podemos perceber várias situações interessantes:

- Para 2 e 4 NEs, o modelo não apresentou sugestões; analisando os dados, percebemos que para pequenos números de NEs, como a classificação é relativa e nivelada pelo maior valor como 100%, e além disso todas as configurações obtiveram consumos próximos, suas classificações relativas ficaram sempre acima de 50%, não havendo nenhuma configuração com “muito baixo” diferente de 0%;
- Para todas as outras quantidades de NEs, a configuração “a” foi a escolhida; percebemos que a configuração “a” venceu sempre pela regra 1, concluindo que a configuração “a” não é adequada quando avaliamos “consumo na gerência”;
- A configuração “b” sempre venceu a regra 2, indicando que ela é muito interessante para gerencia combinada “(no gerente menor que médio) e (na gerência menor que alto)”; percebemos ainda que para maior escala poderemos ter a configuração “b” na regra 2 ganhando da configuração “a” na regra 1;
- A configuração “c” foi bem no quesito “na gerência maior que alto”, mas na combinação de regras sempre foi a menos indicada; resgatando a teoria, percebemos que o ambiente proposto dá enfoque à utilização do paradigma de agentes móveis, onde um agente migra por todos os NEs, retornando somente os dados ao gerente; a configuração “c” dá enfoque ao paradigma cliente-servidor, apesar de ser implementado em agentes móveis e, por isso, não obteve sucesso nesse exemplo.

Podemos ainda fazer algumas outras dezenas de observações acerca da tabela apresentada. Isso mostra a riqueza de subsídios que o gerente humano pode tirar do modelo de decisão proposto. Não podemos perder de vista que esses cálculos foram feitos em cima de um conjunto de apenas duas regras, de certa forma, simples, em um ambiente igualmente simples. Uma simples alteração em uma regra ou inclusão de outra pode alterar completamente os resultados obtidos.

Por um lado, percebemos que uma implementação por redes neurais acabaria escondendo todas essas informações, mas por outro, teríamos uma solução mais direta e objetiva.

9.3 Implementando a decisão por redes neurais

As redes neurais, como no modelo de classificação das configurações, podem ser utilizadas para implementar a decisão modelada pelas regras de inferência.

Existe, contudo, uma inconveniência: as redes neurais precisam ser treinadas e testadas, para, em um segundo momento, serem utilizadas. Essas etapas (treinamento, teste e experimento), que precisariam ser refeitas a cada nova mudança em qualquer regra de decisão, não são imediatas e nem tampouco de processamento leve. Contudo, se o conjunto de regras é fixo, essa solução se torna novamente atraente e viável.

9.3.1 O problema dos passos intermediários

As redes neurais são implementações que não permitem avaliação dos passos intermediários. Por essa característica, precisamos definir o que teremos de entrada e o que desejamos como saída das redes, cientes de que nada intermediário, no processo de decisão, estará disponível. Se precisarmos avaliar os critérios utilizados pela rede, ou conhecer algum detalhe no processo de decisão, as redes neurais não são uma solução adequada. Contudo, se o necessidade se resume em submeter as configurações ao processo de decisão, observando a decisão tomada, então podemos utilizar as redes neurais.

9.3.2 As duas soluções propostas

Nesse modelo, precisamos entrar com as informações da gerência a ser avaliada, tendo, como saída, a configuração mais adequada, de acordo com as regras que serviram de base para o treinamento da rede. Podemos ainda oferecer ao gerente humano alguns

detalhes do processo de decisão, através de outra solução que seja composta por duas redes, onde a saída da primeira é entrada para a segunda. Essas soluções estão detalhadas abaixo:

- Decisão com uma rede neural: teremos como saída a configuração selecionada; nesse caso a rede precisa ser treinada com valores inferidos em todo o processo estudado anteriormente (desde o encontro de valores, através das equações do modelo de consumo encontrado, passando pelo modelo de classificação e, finalmente, inferindo no modelo de decisão, em um grupo de regras de decisão pré-estabelecidas pelo gerente humano;
- Decisão com duas redes neurais: teremos duas redes neurais em série, de forma que as saídas da primeira são as entradas para a segunda; nesse caso precisamos, dentro de todo o processo descrito na solução anterior, definir os domínios das soluções que cada uma das redes se responsabilizará; essa solução vem de encontro ao problema da falta de conhecimento de passos intermediários, pois ao separarmos a resolução do problema em duas redes neurais, podemos separar os domínios de cada uma exatamente no ponto que desejamos conhecer ou observar.

A primeira situação é prática, pois vai direto ao objetivo procurado. A segunda solução é detalhista, pois nos da oportunidade de conhecer algumas informações do processo de decisão, antes de eleger a mais adequada.

Não vamos nos preocupar com a implementação desse modelo em algum paradigma de IA: a sua validação já se deu na tabela 29 e discutido no item 8.2.1. e sua implementação, por exemplo, em redes neurais, seguiria exatamente os mesmos passos descritos para a implementação do modelo de classificação.

As conclusões sobre este capítulo, dada sua importância, foram colocadas no capítulo seguinte.

10. Conclusão

O nosso objetivo com esse trabalho é obter um modelo de decisão automatizada, responsável por apontar ao gerente humano a melhor configuração de comunidades de agentes móveis, na gerência de redes de computadores, sob a métrica “consumo de banda”. E esse objetivo foi alcançado. Contudo, além das observações acerca do modelo de decisão, identificamos em todas as etapas do projeto, situações imprevisíveis, e não menos importantes, a que o gerente humano está sujeito.

Percebemos ainda, em trabalhos correlatos, que existe uma grande dificuldade de se avaliar configurações de agentes móveis na prática, dada a falta de padronização entre as implementações das plataformas. Esses trabalhos adotam como solução, simulações para obtenção de alguns parâmetros, para finalmente avaliar a configuração de agentes. Contudo, através de modelos matemáticos retirados de uma plataforma, obtivemos resultados teóricos bem próximos dos encontrados em suas implementações.

A gerência de redes de computadores automatizada sempre é desejável, dada a sua possibilidade de pró-atividade e de correção de problemas imediatamente após seu surgimento. O modelo de decisão apresentado tenta oferecer ao gerente humano, dadas as suas possíveis atitudes convertidas em regras de inferência, a decisão de melhor configuração para uma gerência.

10.1 Aplicações dos modelos

As equações foram levantadas com características que as tornam genéricas:

- Estão divididas em três configurações de comunidades de agentes móveis, dentre as quais uma será eleita a mais adequada;
- Estão em função da quantidade de NEs , de forma que podemos aplicar em qualquer quantidade de elementos gerenciados (escalabilidade);
- Dada a necessidade de gerência genérica – como, por exemplo, a coleta de n dados ou execução remota de determinada rotina, as equações estão também em função das instruções e em função da quantidade de dados coletados;
- Definidas em configurações primárias de agentes móveis, ou seja, podemos extrapolar as equações em ambientes complexos, compostos por

várias sub-redes, sendo que cada sub-rede teria o seu conjunto de equações, e o resultado para a rede como um todo seria o somatório dos cálculos das partes.

Para uma aplicação dos modelos, podemos então:

- Definir a gerência a ser aplicada: coleta de dados, execução remota de rotina, etc;
- Implementar as instruções necessárias para tal gerência, através do KDEMA;
- Submeter esses parâmetros ao modelo, obtendo as classificações das configurações.

Nesse ponto já temos ótimos recursos para auxílio à tomada de decisões por parte do gerente humano. Não obstante, podemos ainda nos beneficiar do modelo de tomada automatizada de decisão, da seguinte forma:

- Implementação de regras de inferência, de acordo com o contexto e desejos particulares do ambiente-alvo da gerência; vale ressaltar que essas regras serão definidas uma só vez, ou seja, a partir da gerência seguinte, a implementação das regras não seria mais necessária;
- Construção de situações de decisão, com entradas e saídas para treinamento de redes neurais; novamente vale ressaltar que essa tarefa seria executada uma só vez.

10.2 Principais contribuições

Este trabalho, considerando sua linha de pesquisa e seus resultados, obteve as seguintes contribuições:

- Afirma a utilização dos agentes móveis na gerência de redes, através de teoria seguida de sua validação prática;
- Observa um dos pontos de decisão envolvidos no processo de gerência como um todo: o ponto de “como fazer” a coleta de dados; verifica a aplicabilidade dos agentes móveis nesse ponto, dadas as suas características de configurações;

- Sugere a implementação deste ponto de decisão através de paradigmas de inteligência computacional – um passo importante para a obtenção da gerência automatizada;

10.3 Trabalhos futuros

Esse trabalho nos direciona para as seguintes frentes de pesquisa:

- Implementar todos os modelos apresentados no KDEMA (cálculo de consumo de banda, classificação das configurações e decisão da melhor configuração);
- Estudar outros modelos para classificação e escolha de melhor resultado, contemplando gerências condicionais, onde teríamos as equações em função de probabilidades de acontecimentos de gerência; esse recurso nos permitiria utilizar inteligência nos agentes, que podem ou não enviar dados coletados ou se mover, baseado no contexto local (do NE);
- Avaliar o comportamento do modelo para outras funções de inferência *fuzzy*, aplicadas no modelo de classificação;
- Aprimorar a construção de regras de decisão, oferecendo ao gerente maiores possibilidades de representação dos critérios de decisão.

Anexo 1 – Telas de Log do KDEMA

Para coletarmos os dados da métrica considerada (consumo de banda), utilizamos os recursos de *Log* do KDEMA. Eles estão disponíveis tanto no gerente quanto em cada ambiente, via interface web, podendo ser acessado por qualquer navegador HTML.

Foram feitas coletas das experimentações para 4, 8 e 20 NEs. Contudo, estão apresentados somente os *Logs* para 4 NEs, já que os procedimentos de contabilização do consumo de banda são similares para os experimentos com 4, 8 ou 20 NEs. As colunas Arg1, Arg2 e Arg3 não foram apresentadas para minimizar o tamanho das tabelas.

O KDEMA possui um conjunto de mensagens utilizadas pelos seus elementos de comunicação para resolução de nomes. Existem mensagens de registro de nomes, utilizadas quando um ambiente é iniciado, e mensagens de consulta a nomes, onde um nome de elemento é convertido em um IP/Porta. Essas mensagens não foram consideradas na contabilização do consumo de banda.

Daemon1 – 4 NEs – Configuração "a"

Tabela 34. Log da Inbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.165:1100	dae.reg.res
70	Manager	Daemon1	age.cre.req
755	Manager	Daemon1	tas.dow.req
86	Manager	Daemon1	dae.res.res
1200	Daemon2	Daemon1	age.dow.req
1402	Daemon2	Daemon1	age.dow.req

Tabela 35. Log da Outbox

Bytes	Origem	Destino	Token
105	192.168.0.165:1100	Manager	dae.reg.req
69	Daemon1	Manager	dae.res.req
1104	Daemon1	Daemon2	age.dow.req
1306	Daemon1	Daemon2	age.dow.req
382	Daemon1	Manager	age.kno.inf

Daemon2 – 4 NEs – Configuração “a”

Tabela 36. Log da Inbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.165:1200	dae.reg.res
1104	Daemon1	Daemon2	age.dow.req
86	Manager	Daemon2	dae.res.res
1306	Daemon1	Daemon2	age.dow.req

Tabela 37. Log da Outbox

Bytes	Origem	Destino	Token
105	192.168.0.165:1200	Manager	dae.reg.req
69	Daemon2	Manager	dae.res.req
1200	Daemon2	Daemon1	age.dow.req
1402	Daemon2	Daemon1	age.dow.req

Manager – 4 NEs – Configuração “a”

Tabela 38. Log da Inbox

Bytes	Origem	Destino	Token
105	192.168.0.165:1100	Manager	dae.reg.req
105	192.168.0.165:1200	Manager	dae.reg.req
69	Daemon1	Manager	dae.res.req
69	Daemon2	Manager	dae.res.req
382	Daemon1	Manager	age.kno.inf

Tabela 39. Log da Outbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.165:1100	dae.reg.res
91	Manager	192.168.0.165:1200	dae.reg.res
70	Manager	Daemon1	age.cre.req
755	Manager	Daemon1	tas.dow.req
86	Manager	Daemon1	dae.res.res
86	Manager	Daemon2	dae.res.res

Daemon1 – 4 NEs – Configuração “b”

Tabela 40. Log da Inbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.189:1100	dae.reg.res
70	Manager	Daemon1	age.cre.req
819	Manager	Daemon1	tas.dow.req
86	Manager	Daemon1	dae.res.res
1096	Daemon2	Daemon1	age.dow.req
1106	Daemon2	Daemon1	age.dow.req

Tabela 41. Log da Outbox

Bytes	Origem	Destino	Token
105	192.168.0.189:1100	Manager	dae.reg.req
154	Daemon1	Manager	age.kno.inf
69	Daemon1	Manager	dae.res.req
1096	Daemon1	Daemon2	age.dow.req
154	Daemon1	Manager	age.kno.inf
1106	Daemon1	Daemon2	age.dow.req

Daemon2 – 4 NEs – Configuração “b”

Tabela 42. Log da Inbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.155:1200	dae.reg.res
1096	Daemon1	Daemon2	age.dow.req
86	Manager	Daemon2	dae.res.res
1106	Daemon1	Daemon2	age.dow.req

Tabela 43. Log da Outbox

Bytes	Origem	Destino	Token
105	192.168.0.155:1200	Manager	dae.reg.req
154	Daemon2	Manager	age.kno.inf
69	Daemon2	Manager	dae.res.req
1096	Daemon2	Daemon1	age.dow.req
154	Daemon2	Manager	age.kno.inf
1106	Daemon2	Daemon1	age.dow.req

Manager – 4 NEs – Configuração "b"

Tabela 44. Log da Inbox

Bytes	Origem	Destino	Token
105	192.168.0.189:1100	Manager	dae.reg.req
105	192.168.0.155:1200	Manager	dae.reg.req
154	Daemon1	Manager	age.kno.inf
69	Daemon1	Manager	dae.res.req
154	Daemon2	Manager	age.kno.inf
69	Daemon2	Manager	dae.res.req
154	Daemon1	Manager	age.kno.inf
154	Daemon2	Manager	age.kno.inf

Tabela 45. Log da Outbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.189:1100	dae.reg.res
91	Manager	192.168.0.155:1200	dae.reg.res
70	Manager	Daemon1	age.cre.req
819	Manager	Daemon1	tas.dow.req
86	Manager	Daemon1	dae.res.res
86	Manager	Daemon2	dae.res.res

Daemon1 – 4 NEs – Configuração “c”

Tabela 46. Log da Inbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.165:1100	dae.reg.res
70	Manager	Daemon1	age.cre.req
344	Manager	Daemon1	tas.dow.req

Tabela 47. Log da Outbox

Bytes	Origem	Destino	Token
105	192.168.0.165:1100	Manager	dae.reg.req
154	Daemon1	Manager	age.kno.inf

Manager – 4 NEs – Configuração “c”

Tabela 48. Log da Inbox

Bytes	Origem	Destino	Token
105	192.168.0.165:1100	Manager	dae.reg.req
154	Daemon1	Manager	age.kno.inf

Tabela 49. Log da Outbox

Bytes	Origem	Destino	Token
91	Manager	192.168.0.165:1100	dae.reg.res
70	Manager	Daemon1	age.cre.req
344	Manager	Daemon1	tas.dow.req

Referências Bibliográficas

[AGLET] Sourceforge: Open Source Software Development WebSite. Aglet Software Development Kit. Disponível em http://sourceforge.net/projects/aglets. Acesso em 15 jan. 2003.

[ARANTES 2002] ARANTES, Juliana; WESTPHALL, Carlos; CUSTÓDIO, Ricardo. Modelo Analítico para Avaliar Plataformas Cliente/Servidor e Agentes Móveis Aplicado à Gerência de Redes. Anais do 20º Simpósio Brasileiro de Redes de Computadores (Vol. I). Búzios (RJ), p. 424-439.

[BARRETO 1999] BARRETO, Jorge M. Inteligência Artificial no Limiar do Século XXI, 2ª Edição, Florianópolis, 1999.

[BALDI 1997] BALDI, Mario; GAI, Silvano; PICCO, Gian. Exploiting Code Mobility in Decentralized and Flexible Network Management, In: Proceedings of the First International Workshop on Mobile Agents, 1997.

[BIESZCZAD 1998] BIESZCZAD, Andrzej; PAGUREK, B.; WHITE, T. Mobile Agents for Network Management. Disponível em http://citeseer.nj.nec.com/ bieszczad98mobile.html. Acesso em 15 jan. 2003. IEEE Communications Surveys, 1998.

[BOHORIS 2000] BOHORIS, C.; PAVLOU, G.; CRUICKSHANK, H. Using Mobile Agents for Network Performance Management – In: Network Operation and Management Symposium, Vol 07, pp 637-652. Sep 2000.

[DAS 2002] DAS, S.; SHUSTER, K.; WU, C.,. ACQUIRE: agent-based complex query and information retrieval engine. In: Proceedings of the first international joint conferente on Autonomous agents and multiagents systems.

[EBERHART 1996] EBERHART, Puss, SIMPSON, Pat; DOBBINS, Roy, Computational Intelligence PC Tools, Academic Press Limited, 1996.

[FIPA] The Foundation for Intelligent Physical Agents, FIPA. Fipa Specifications. Disponível em http://www.fipa.org/specifications/index.html. Acesso em 15 jan. 2003.

[HURST 1997] HURST, Leon; CUNNINGHAM, Padraig; SOMMERS, Fergal. Mobile Agents Smart Messages. In: Proceedings of the 1st International Workshop on Mobile Agents, 1997.

[ISO-OSI 9595] International Organization for Standarlization. Open Systems Interconnection – Common Management Information Protocol Specification. Disponível em http://www.iso.org. Acesso em 15 jan. 2003.

[ISO-OSI 10165] International Organization for Standarlization. Open Systems Interconnection – Management Information Services – Structure of Management Information – Management Information. Disponível em http://www.iso.org. Acesso em 15.jan. 2003.

[JAVA] Sun MicroSystems. Java 2 Platform. Disponível em http://java.sun.com. Acesso em 15 jan. 2003.

[KARNOUSKOS 2002] KARNOUSKOS, S.; VASILAKOS, A, Neuro-fuzzy applications: Active electronic mail. In: Proceedings of the 17th symposium on Proceedings of the 2002 ACM Symposium on applied computing.

[KAZI 1999] KAZI, Rumeel; MORREALE, Patrícia. Mobile agents for active network management. IEEE Military Communications Conference, Vol 18, pp. 149. 153. Oct 1999.

[KDEMA] Plataforma KDEMA. Disponível em http://www.lrg.ufsc.br/~kdema. Acesso em 15 jan. 2003.

[MAGEDANZ 1996] MAGEDANZ, T.; ROTHERMEL, K.; KRAUSE, S. Intelligent Agents: An Emerging Technology for Next Generation Telecommunications? In: INFOCOM, 1996.

[OHARE 1996] OHARE, G.; JENNINGS, N. Foundations of Distribuited Artificial Intelligence, A Wiley-Interscience Publication, Nova York, 1996.

[QWIKNET] Kagi Software. QwikNet 2.23. Disponível em http://www.kaji.com/cjensen. Acesso em 15 jan. 2003.

[RFC1157] SNMP – A Simple Network Management Protocol. Disponível em http://www.ietf.org/rfc/rfc1157.txt?number=1157. Acesso em 15 jan. 2003.

[RUBINSTEIN 2001] RUBINSTEIN, Marcelo, Evaluation of the performance of mobile agents in the management of networks. Rio de Janeiro. 2001.

[STALLINGS 1998] STALLINGS, William. - SNMP and SNMPv2: The infrastructure for network management, IEEE Communications Magazine, 1998.

[TANENBAUM] TANENBAUM, Andrew; Redes de Computadores, Editora Campus, 6ª Tiragem, 1997.

[THOTTAN 1998] THOTTAN, Marina; JI, Chuanyi. Proactive Anomaly Detection Using Distributed Intelligent Agents, IEEE Network, Vol 12, pp. 21-27, Sep 1998.

[WHITE 1998] WHITE, Tony; BIESZCZAD, Andrzej; PAGUREK, Bernard. Distribuited Fault Location in Networks Using Mobile Agents. Intelligent Agents for Telecommunication Applications. In: Proceedings of the Second International Workshop on Intelligent Agents for Telecommunication (IATA), 1998.

[XAVIER 2002] XAVIER, Edison; KOCH, Fernando; WESPHALL, Carlos, Avaliação de Variações da Configuração de Agentes Móveis na Gerência de Redes, In: I2TS'2002 – Florianópolis - SC.

[YEMINI 1993] YEMINI, Yechiam. The OSI Network Management Model. Disponível em http://www.comsoc.org/livepubs/surveys/public/1q00issue/Yemini.pdf. Acesso em 15 jan. 2003. IEEEE Communications Magazine, 1993.

[ZEUS] British Telecom. ZEUS. Disponível em http://more.btexact.com/projects/agents/zeus. Acesso em 15 jan. 2003.