Arquitetura de Gerenciamento de Redes Baseado em Agentes Móveis Mutáveis

Ralf Luis de Moura

Mestrado em Automação – Departamento de Engenharia Elétrica
Centro Tecnológico – Universidade Federal do Espírito Santo

Arquitetura de Gerenciamento de Redes Baseadas em Agentes Móveis Mutáveis

Ralf Luis de Moura

Dissertação apresentada a Universidade Federal do Espírito Santo como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica

Orientador: Prof. Dr. Anilton Salles Garcia

Vitória
2007

RALF LUIS DE MOURA

ARQUITETURA DE GERENCIAMENTO DE REDES BASEADA EM AGENTES MÓVEIS MUTÁVEIS

Dissertação submetida ao programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisição parcial para a obtenção do Grau de Mestre em Engenharia Elétrica - Automação.

Aprovada em 29 de junho de 2007.

COMISSÃO EXAMINADORA

Prof. Dr. Anilton Salles Garcia
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Luiz Fernando Gomes Soares
Pontifícia Universidade Católica do Rio de Janeiro

Prof. Dr. Rosane Bodart Soares
Universidade Federal do Espírito Santo

Dedicatória

A minha família, principalmente a minha esposa Magna e meus filhos Brenda e Ralf Filho pelo apoio, compreensão e paciência nos meus momentos de ausência por dedicação ao trabalho.

Agradecimentos

A Deus, pelos desafios oferecidos, e, pelas forças para enfrentá-los.
Em especial, ao Prof. Dr. Anilton Salles Garcia, cuja orientação e apoio foram fundamentais na realização deste trabalho.
Aos membros da banca, Prof. Dr. Luiz Fernando Gomes Soares e Prof. Dr. Rosane Bodart Soares, pela dedicação e tempo dedicados à avaliação deste trabalho.
A todos os amigos, pelo apoio e amizade nos momentos que precisei.

Sumário

Lista de Figuras

Lista de Acrônimos

ACL: *Agents Communication Language* (Linguagem de comunicação de agentes);
AMS: *Agent Management System;*
CMIP: *Common Management Information Protocol*;
CORBA: *Common Object Request Broker Architecture*;
CPU: *Central Processing Unit*;
FIPA: *Foundation for Intelligent Physical Agents*;
FM: *Functionality Manager* (Gerenciador de Funcionalidades);
IDL: *Interface Definition Language* (Interface de definição de linguagem);
IDS: *Intrusion Detection System*;
IEC: *International Electrotechnical Commission*;
IETF: *Internet Engineering Task Force*;
IIOP: *Internet Inter-ORB Protocol*;
IP: *Internet Protoco*l;
ISO: *International Organization for Standardization*;
ITU: *International Telecommunication Union*;
J2EE: *Java 2 Enterprise Edition*;
JADE: *Java Agent Development Framework*;
JDK: *Java Development Kit*;
MIB: *Management Information Base*;
OMG: *Object Management Group*;
OSI: *Open Systems Interconnection*;
PDA: *Personal Digital Assistants;*
PDU: Protocol Data Unit – (Unidade de dados do protocolo);
RFC: *Request For Comments;*
RMI: invocação remota de método (*Remote Method Invocation*);
RMON: monitoração remota (*Remote Monitoring*);
SNMP: *Simple Network Management Protocol* (Protocolo de Gerência Simples de Rede);
TCP: *Transmission Control Protocol*;

UC: Unidade de Controle.

UCOM: Unidade de Comunicação.

URL: *Universal Resource Locator*.

Lista de Tabelas

Resumo

Atualmente, grande parte dos sistemas de gerenciamento de rede opera utilizando arquiteturas centralizadas e o protocolo SNMP (*Simple Network Management Protocol*). A interação entre a estação de gerenciamento e seus pontos gerenciados envolve um tráfego intenso de informações, o que em certos casos pode causar sobrecarga no sistema. Estudos indicam que a utilização de agentes móveis como mecanismo de gerenciamento descentralizado pode, em certos casos, reduzir a troca de informações e consequentemente minimizar de forma significativa a sobrecarga do sistema. Porém, a utilização de agentes altamente especializados, pode aumentar a complexidade e a sobrecarga do sistema, uma vez que, a cada necessidade de gerenciamento, um novo agente especialista deverá ser criado e enviado ao ponto de intervenção. Nesta dissertação, uma arquitetura de gerenciamento de redes baseada em agentes móveis mutáveis é proposta. A arquitetura proposta combina as principais arquiteturas de gerenciamento existentes com o uso dos agentes móveis mutáveis desenvolvidos utilizando a arquitetura JADE (*Java Agent Development Framework*) e a plataforma Java™. Essa nova arquitetura agrega características das atuais arquiteturas, propondo métodos mais eficientes e menos complexos que podem ser aplicados em gerenciamento de rede, reduzindo a intervenção humana, com características de auto-gerenciamento.

Abstract

Normally, the network management systems operate using client-server architectures and SNMP protocol (Simple Network Management Protocol). The interaction between the management station and its managed points involves a great volume of traffic information, that can generate overload in the network. Studies indicate that the use of mobile agents as mechanism of decentralized management can reduce the exchange of information and consequently minimize significantly the system overload. However, the use of highly specialized agents can increase the system complexity and also generate overload, once for each management necessity a new specialist agent will have to be created and sent to the intervention point. In this thesis a network management architecture based on changeable mobile agents is proposed. The proposed architecture combines the main existing network management architectures associated with the use of changeable mobile agents, developed using the JADE (Java Agent Development Framework) architecture and the Java™ platform. This new architecture aggregates characteristics of these architectures, proposing less complex and more efficient methods than can be applied in the network management, reducing the human intervention necessity with self-management characteristics.

1. Introdução

Atualmente, a informação é uma das melhores formas de se agregar valor a negócios e a vida das pessoas. A busca por informação é cada vez maior e é cada vez mais desafiador viabilizar o processamento, armazenamento e disponibilização das informações para um crescente número de acessos.
A rede de computadores é um dos principais elementos neste cenário, pois através dela é possível interligar diversos pontos permitindo a troca de informações de maneira rápida e confiável.

Para que esses ambientes possam continuar funcionando de forma rápida e confiável é necessário que os mesmos sejam controlados e monitorados de forma a garantir um nível de serviço adequado. O gerenciamento desses ambientes envolve ações em áreas como segurança, desempenho, falhas, configuração, entre outros.

Existe também uma tendência cada vez maior de descentralização dos ambientes computacionais e a crescente diversificação dos serviços fornecidos. Esse crescimento faz com que a demanda por equipamentos de rede seja cada vez maior, fazendo com que vários novos fabricantes entrem neste mercado promissor. A pulverização de fabricantes gera um problema, pois introduz um alto grau de heterogeneidade de equipamentos que, muitas vezes, não são capazes de comunicar entre si, tornando cada vez mais complexo o controle e o monitoramento desses ambientes. A crescente adição de novos equipamentos (PDA's, *Palms*, *Notebooks's*, etc) como pontos de rede é também uma tendência que pode ser notada. Esses novos equipamentos adicionam necessidades de mobilidade antes não presentes em ambientes de rede. Essa mobilidade permite que se conecte a diferentes sistemas em diferentes locais, mas com a necessidade imperativa de se fazer isto como se tivesse em seu ambiente nativo.

Esses são desafios que as tradicionais formas de gerenciamento de rede não são capazes de lidar, porque utilizam basicamente um paradigma centralizado. Os agentes

móveis inteligentes aparecem como uma possível solução para esse problema. Várias pesquisas estão sendo realizadas usando essas tecnologias com diferentes abordagens. Embora os agentes móveis em sistemas distribuídos sejam uma opção, eles também têm seus problemas e suas limitações.

O principal objetivo desta dissertação é propor uma nova arquitetura que seja mais eficiente, em relação às arquiteturas tradicionais, e adequada às características necessárias para a utilização em ambientes de gerenciamento de redes. O foco é a utilização em gerenciamento de redes, mas não há restrições para a utilização dessa arquitetura em outros tipos de aplicações. A arquitetura proposta combina várias características dos atuais paradigmas existentes, centralizada e descentralizada, incluindo uma nova característica: a capacidade de mutação dos agentes. A proposta apresentada nesta dissertação explora as vantagens e desvantagens de cada paradigma procurando aperfeiçoar o processo de gerenciamento, buscando adicionar melhoria em desempenho e flexibilidade, juntamente com a redução do uso de recursos de rede.

A metodologia utilizada é baseada na comparação entre os paradigmas, na identificação das melhores características e nos problemas que cada um apresenta. Para os problemas identificados, novas formas são consideradas para atenuá-los e juntamente com as melhores características criar uma nova arquitetura.

O principal resultado desta dissertação é uma arquitetura que, comparada com as outras, busque trazer melhoria em desempenho, aumento da flexibilidade, redução da complexidade, otimização do uso dos recursos de rede, aliada à redução da necessidade de intervenção humana no controle, organização, monitoração e gestão das redes (KONSTANTINOU, 2003) (KONSTANTINOU II, 2003).

1.1. Redes Auto-gerenciadas

A AT&T (AT&T, 2004) conceitua o auto-gerenciamento da seguinte forma: O uso de tecnologias de automação de rede para permitir que equipamentos complexos gerenciem-se a si mesmo.

Os Organismos Autônomos de Computação são aqueles que possuem mecanismos auto-suficientes para prover: auto-governo, regulação, correção, organização, escalonamento, planejamento, gerenciamento, administração, otimização, monitoramento, ajuste, sintonia, configuração, diagnóstico de falhas, proteção, cura, recuperação de desastres, aprendizado, conhecimento, representação, evolução e auto-avaliação de eficiência e risco (MONTEIRO, 2005) (CZAP, 2005).

O Auto-gerenciamento é derivado das teorias de controle adaptativo (engenharia de automação) e dos sistemas autônomos de computação. Sua principal idéia é que as entidades virtuais de gerenciamento (ferramentas), que antes apoiavam o administrador da rede em determinadas ações, passem a assumir o controle do gerenciamento da rede, dependendo minimamente do ser humano (MONTEIRO, 2005).

O aumento da demanda por conectividade por parte dos usuários finais de rede resulta em interações cada vez mais complexas, tornando mais difícil prever como várias entidades irão interagir entre si em um sistema. A complexidade de cada componente desse sistema está crescendo tão rápido que mesmo administradores de rede experientes terão dificuldades em lidar com isso. Isso leva à necessidade do aumento da inteligência no sistema, onde os sistemas autônomos (auto-gerenciáveis) aparecem (STRASSNER, 2005).

O principal objetivo de uma rede auto-gerenciada é fazer com que os recursos de rede gerenciem a si próprios de forma a reduzir a necessidade de intervenção humana,

reduzindo assim os custos operacionais do ambiente de rede, além de melhorar a disponibilidade do sistema como um todo.

As redes atuais utilizam de procedimentos operacionais realizados por especialistas em administração de redes para manipular mudanças em recursos de rede de acordo com a necessidade. Os custos dessas operações são altos, uma vez que necessitam da disponibilidade de profissionais especializados, além de serem sujeitos a erros por necessitar de julgamento humano. Redes auto-gerenciadas podem realizar mudanças rápidas em recursos de rede, com o objetivo de corrigir desvios ou resolver falhas, sem a dependência de intervenção humana (KONSTANTINOU, 2002).

Uma rede auto-gerenciada pode cobrir todas as áreas de gerenciamento de redes, porém as principais pesquisas sobre esta área estão ligadas às gerências de configuração, desempenho, segurança e falhas.

Para alcançar o auto-gerenciamento em uma rede, ela deve ser auto-ciente, ou seja, a rede deve ser conhecedora de seu status e ambiente-ciente onde a rede deve ter consciência do ambiente operacional. O auto-gerenciamento é criado através da auto – Configuração, Auto-Diagnóstico, Auto-Ajuste e Auto-Proteção (SHEN, 2005).

Roy (ROY, 2005) lista as funções onde as redes auto-gerenciadas podem atuar agindo com autonomia e reduzindo a necessidade de intervenção humana:

Auto-Configuração: Fornecer mecanismos para que a rede continue operando, mesmo quando dispositivos são adicionados e retirados. Esses mecanismos podem atualizar versões de softwares, atualização de base de informações relacionadas a novos *hardwares* ou *softwares* de forma automática e *on-line*.

Auto-Diagnóstico e correção de falhas: Fornecer mecanismos que permitam a continuação da operação, quando dispositivos da rede apresentarem falhas. Mais

especificamente, permitir que os serviços básicos continuem a serem oferecidos, enquanto a falha está sendo corrigida.

Auto-Ajuste: Fornecer mecanismos para balanceamento de carga, através de operações diretas nos dispositivos da rede.

Auto-Proteção: Fornecer mecanismos que protejam a rede de ataques externos e de tentativas de acesso não permitidas.

Atualmente, várias pesquisas estão sendo desenvolvidas no campo de gerenciamento de redes. As arquiteturas tradicionais ainda são extensivamente utilizadas e mecanismos de auto-gerenciamento não estão ainda sendo empregados em larga escala nas redes existentes.

O auto-gerenciamento abre possibilidades para facilitar o trabalho dos administradores e operadores de rede, executando tarefas de forma autônoma sem depender de intervenção manual.

Mecanismos de auto-gerenciamento que não sejam compatíveis com os protocolos utilizados nas arquiteturas tradicionais como o SNMP, por exemplo, terão dificuldades em serem utilizados, pois o legado atual é muito grande.

Este trabalho busca caminhos para viabilizar o uso de recursos auto-gerenciáveis de forma a explorar suas capacidades, com a preocupação de minimizar a sua interferência na rede e de se adaptar o melhor possível aos ambientes legados e às tecnologias existentes. O foco do trabalho não é definir formas de utilização de recursos de gerência, mas abrir possibilidades para a utilização destes da forma mais transparente possível.

Com o objetivo de proporcionar uma melhor compreensão da dissertação, a mesma encontra-se divida em 6 capítulos e com a seguinte organização:

- Capítulo 1: Introdução

Neste capítulo mostra-se, de maneira resumida, as tendências de crescimento dos ambientes de rede e da computação descentralizada, a necessidade de gerenciamento destas redes cada vez mais complexas e heterogêneas, a necessidade do auto-gerenciamento, além da apresentação da proposta de arquitetura.

- Capítulo 2: Revisão Bibliográfica

Neste capítulo são apresentados vários trabalhos relacionados a esta dissertação com foco na utilização de agentes móveis e no gerenciamento de redes com esta tecnologia.

- Capítulo 3: Agentes móveis e Sistemas multi-agentes

Neste capítulo é apresentada a tecnologia de agentes móveis e descreve-se suas características e potencialidades, além de mostrar a viabilidade da utilização destes em ambientes de gerenciamento de redes.

- Capítulo 4: Proposta de Agentes Móveis Mutáveis

Neste capítulo mostra-se a arquitetura proposta. São descritos detalhes de ambiente, formas de interação e o comportamento do sistema.

- Capítulo 5: Estudo de caso e Simulações

Neste capítulo detalha-se o estudo de caso realizado com o objetivo de comprovar a potencialidade da arquitetura proposta, além do desenvolvimento de um protótipo e a realização de simulações para comparações entre as arquiteturas.

- Capítulo 6: Conclusões

Neste capítulo apresenta-se as principais conclusões do trabalho, além das sugestões de continuidade do mesmo.

Informações complementares ao trabalho, consideradas não essenciais para a compreensão do mesmo, estão apresentadas nos seguintes anexos:

- Anexo A – A Plataforma Jade.
- Anexo B – Lista dos principais elementos do código do protótipo.
- Anexo C – Gerenciamento de redes

2. Revisão Bibliográfica

Pesquisas relacionadas à utilização de agentes como ferramentas de apoio ao gerenciamento de redes são recentes (BRAUN, 2005). A maioria dos trabalhos nesta área propõe a utilização de agentes nas mais diversas arquiteturas com o objetivo de simplificar a sua utilização e maximizar a sua capacidade de interagir com o ambiente de rede de forma autônoma. Neste capítulo são mostrados trabalhos relacionados ao gerenciamento de redes e a utilização de agentes móveis.

2.1. Trabalhos relacionados

Eid (EID, 2005), apresenta a tecnologia de agentes, e as tendências de utilização desta tecnologia. São mostradas as principais áreas de aplicações, como gerenciamento de redes, monitoração, busca e retorno de informação, detecção de intrusos, entre outros. São enumeradas algumas limitações das arquiteturas tradicionais e tendências futuras em vários campos de aplicação.

Em (LANGE, 1999), são descritas as sete principais razões para se utilizar agentes móveis em aplicações distribuídas. São enumeradas as seguintes vantagens da utilização desta tecnologia:

- Os agentes superaram o problema de latência nas redes;
- Os agentes encapsulam protocolos;
- Os agentes interagem de forma assíncrona e autônoma;
- Os agentes adaptam-se dinâmicamente;
- Os agentes são naturalmente heterogêneos;
- Os agentes são robustos e tolerantes a falhas;
- Os agentes reduzem a carga na rede.

Lange também aponta as áreas mais indicadas para a utilização da tecnologia de agentes: comércio pela Internet, assistência pessoal, segurança, telecomunicações e serviços de rede, monitoramento e notificações, entre outros.

Vários estudos (FAHAD, 2003), (RUBINSTEIN, 2001), (RUBINSTEIN II, 2001), (COSTA, 1999), (ADHICANDRA, 2005), fazem comparações entre arquiteturas tradicionais com arquiteturas utilizando agentes móveis. Essas comparações são focadas, principalmente, em parâmetros como desempenho e escalabilidade. Nesses trabalhos, são realizadas simulações com diversos cenários e os resultados indicam vantagens na utilização da arquitetura de agentes móveis em redes grandes com várias sub-redes, e onde a latência é alta. (COSTA, 1999) concluiu, em suas simulações, que o uso de agentes móveis pode se tornar crítico quando o número de recursos visitados pelo agente é alto. Isto se deve a uma característica dos agentes móveis de acumular as informações dos recursos visitados. A Figura 2.1 (RUBINSTEIN II, 2001) mostra, segundo uma simulação, o aumento do tempo de resposta dos agentes móveis e da arquitetura SNMP quando se aumenta o número de elementos gerenciados.

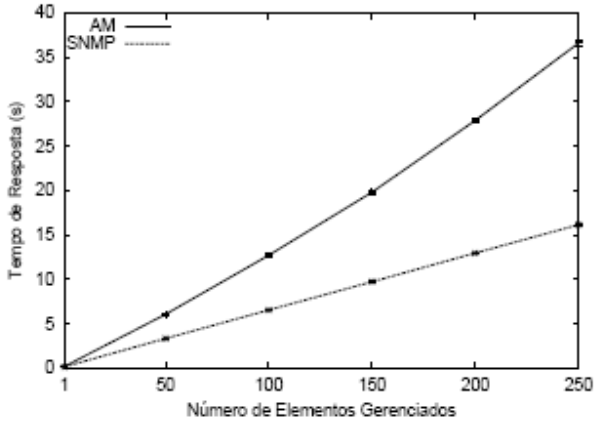

Figura 2.1 – Tempo de resposta: Agente e SNMP

Em (BOHORIS, 2000), é proposta a utilização de agentes móveis no gerenciamento de redes. Este trabalho é parte do projeto MIAMI ACTS (*Mobile Intelligent Agents in the Management of the Information infrastructure*), onde são analisados os impactos e as possibilidades do uso de agentes móveis no gerenciamento de redes e de serviços.

São realizadas também simulações e comparações com as arquiteturas tradicionais de gerenciamento de redes. A utilização de agentes nesta arquitetura objetiva a delegação de tarefas a agentes especializados com foco no monitoramento de desempenho. Os agentes executam as tarefas de monitoração diretamente nos elementos aos quais eles são enviados.

Em (PAGUREK, 2000), é apresentada a necessidade da integração de agentes móveis com o protocolo SNMP. Essa necessidade vem do fato do protocolo SNMP ser o mais importante protocolo em utilização hoje, e que os sistemas legados possuem uma infra-estrutura baseada nesse protocolo. Interações entre esses ambientes legados quase sempre se fazem necessários e este trabalho mostra que a integração entre ambientes de agentes móveis e o protocolo SNMP é possível.

Em (BRANDÃO, 2002), é apresentado o desenvolvimento de agentes para gerenciamento de segurança integrando a tecnologia de agentes com a utilização do protocolo SNMP. Os agentes foram desenvolvidos na plataforma Aglets e com a linguagem Java. A Figura 2.2 ilustra a arquitetura proposta por Brandão, onde os agentes, a partir de uma detecção de intrusos, interagem com os elementos finais através do protocolo SNMP.

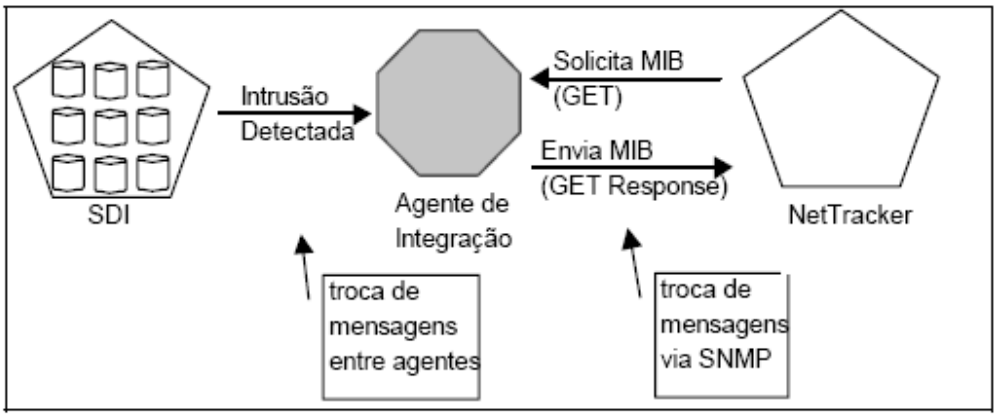

Figura 2.2 – Arquitetura de gerenciamento de segurança

Em (VELLOSO, 2002), é descrita uma ferramenta de gerenciamento de redes baseada em agentes móveis. O objetivo da ferramenta é reduzir o tempo e o número de mensagens trocadas na realização da tarefa de gerenciamento, além de proporcionar uma maior flexibilização. A aplicação permite ao usuário definir tarefas e analisar de forma amigável os dados obtidos. A plataforma de agentes móveis utilizada é o Aglets que é o ambiente de mobilidade que permite a execução, a migração e a troca de mensagens entre os agentes. A Figura 2.3 mostra a arquitetura de gerenciamento proposta por Velloso, onde os agentes móveis são responsáveis por coletar as informações solicitadas pela estação gerente em todos os dispositivos da rede. Essa coleta é realizada a partir da interação entre o agente móvel e um agente tradutor que traduz as necessidades do agente móvel em comandos SNMP.

Figura 2.3 - Modelo de gerenciamento baseado em agentes móveis.

Bianchini (BIANCHINI, 2003) propõe um padrão de construção de sistemas para gerenciamento de redes chamado GeR. Esse padrão permite a integração de diferentes tecnologias como SNMP, Corba e Java, além de cobrir todo o ciclo de vida do desenvolvimento de agentes inteligentes e definir estratégias para a definição e a condução do comportamento de agentes. A Figura 2.4 ilustra a estrutura do padrão, onde vários módulos se interagem:

- ***Administrator***
    - Interage com a ferramenta definindo o gerenciamento dos dispositivos, criando agentes de software e descrevendo seu comportamento na base de conhecimento.
- ***Management Tool***
    - Oferece recursos para a interação entre o administrador e o sistema de gerenciamento e com os agentes de software.
- ***Device***
    - São os dispositivos gerenciáveis, descritos através de suas MIBs (*Management Information Base*), definidos pelo administrador e gerenciados pelos agentes.
- ***Agent***
    - Implementa um conjunto de classes e os seus relacionamentos, para a instanciação de agentes inteligentes.
- **Agent Server**
    - Fornece infra-estrutura e serviços necessários às aplicações distribuídas e multiplataformas em sistemas multi-agentes.
- **Agent Platform**
    - Define um ambiente para desenvolvimento de aplicações de diversos domínios e um conjunto escalável de servidores.

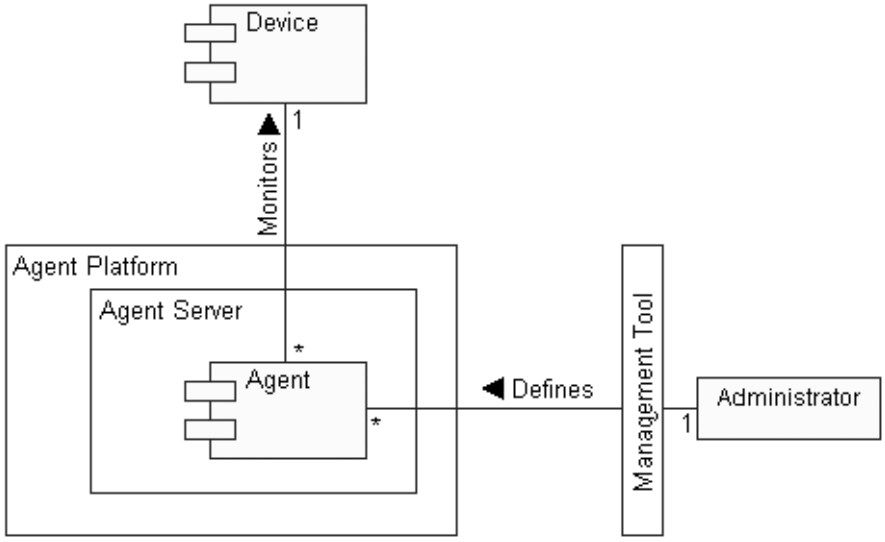

Figura 2.4 – Estrutura do Padrão

Wang (WANG, 2003) propõe uma arquitetura de agentes móveis para dispositivos heterogêneos. A arquitetura é flexível e habilita o acesso de vários dispositivos diferentes a um mesmo sistema de agentes móveis. Isso é possível pela busca da capacidade dos dispositivos antes da instalação dos agentes clientes. Dessa forma, torna-se possível o acesso a alguns dispositivos com recursos limitados como, por exemplo, telefones celulares.

Em (BIESZCZAD II, 1998), é mostrado como os agentes móveis podem ajudar em soluções de gerenciamento de redes. Nesse trabalho é proposto o uso de agentes em áreas funcionais do gerenciamento de redes como gerenciamento de falhas, gerenciamento de configuração, gerenciamento de desempenho e gerenciamento de segurança. Também são relacionadas as vantagens e desvantagens do uso de agentes móveis e é proposta uma arquitetura para a utilização destes.

Em (LOPES, 1999), é dada uma visão geral sobre os paradigmas de agentes de software em *frameWorks* de gerenciamento de redes. É dado foco no uso de programas inteligentes para substituir tarefas desagradáveis e repetitivas, feitas de forma manual, além do uso de agentes na resolução de problemas como congestionamento de redes e respostas em tempo real.

Em (BIGUS, 2001), são apresentadas várias plataformas para desenvolvimento de agentes móveis. Entre elas pode-se destacar: Able, Agent Buider, Aglets, Gossip, Voyager, JetLite, Zeus, FIPA-OS e JADE. Dentre várias ferramentas, JADE foi a plataforma escolhida para o desenvolvimento desse trabalho, devido algumas de suas características como: utiliza a linguagem Java, possui um grande número de bibliotecas prontas, além de seguir todas as recomendações da FIPA (*Foundation for Intelligent Physical Agents*).

Monteiro (MONTEIRO II, 2005), propõe uma arquitetura de um sistema de auto-aprovisionamento e a construção de um protótipo de sistema. Esse protótipo deve possuir funcionalidades para a gerência de Redes, em especial para o processo de aprovisionamento. A descentralização e autonomia dessa arquitetura são proporcionadas pelos agentes móveis. A Figura 2.5 mostra de forma geral a arquitetura proposta por Monteiro, onde uma ontologia geral de redes suporta toda a comunicação entre os módulos do sistema e a interação é fornecida por agentes móveis.

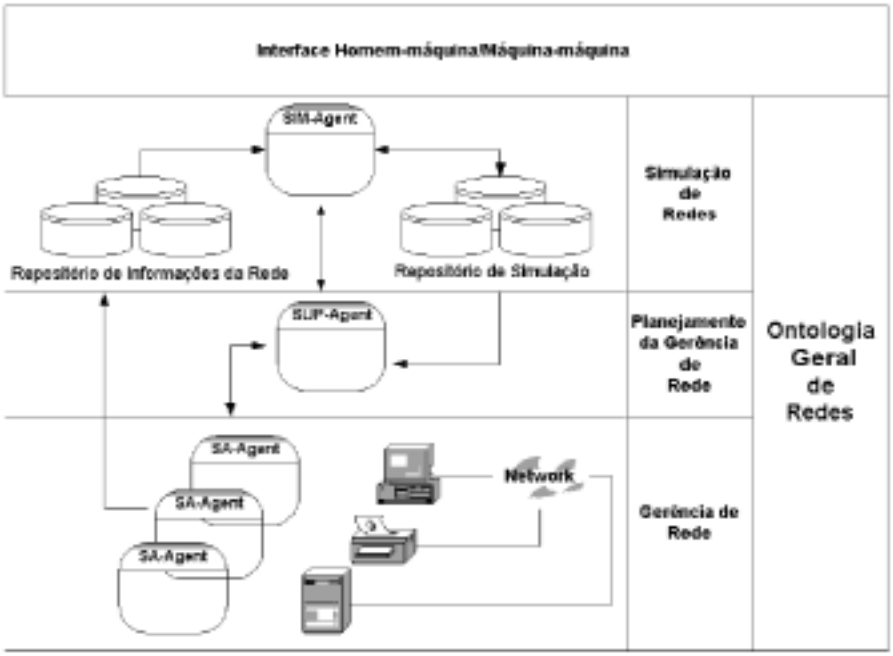

Figura 2.5 – Arquitetura de auto-aprovisionamento

Uma camada de interface Homem-máquina e Máquina-máquina proporcionam algum grau de intervenção humana no sistema, além da integração do sistema com outros que por ventura necessitem dessa facilidade. Um conjunto de ontologias específicas, baseadas na Ontologia Central de Redes, permeia os elementos funcionais da arquitetura, permitindo o entendimento das informações trocadas entre esses elementos. A composição desse sistema exige a integração de um *framework* de agentes, uma ferramenta de simulação e um ambiente de construção e compartilhamento de Ontologias. Ambos são flexíveis, portáveis e integráveis. As ferramentas baseiam-se no uso de plataformas já existentes, de código aberto e livre e escritos na linguagem Java. É proposta uma arquitetura de agentes móveis com o objetivo de viabilizar a flexibilidade, a integração, a descentralização e a autonomia descritas por Monteiro.

2.2. Conclusão do capítulo

A arquitetura proposta nesta dissertação baseou-se em muitos dos requisitos propostos nos trabalhos descritos anteriormente. Os padrões de construção de sistemas de gerenciamento de agentes, a utilização de agentes em soluções de gerenciamento de redes, a comparação do desempenho em sistemas de gerenciamento e o uso de plataforma de desenvolvimento de agentes móveis são algumas das características descritas nos trabalhos anteriores que foram utilizadas neste trabalho. Porém, os trabalhos de pesquisa na área de agentes móveis não propõem soluções para os principais pontos fracos da utilização de agentes. Soluções para a inviabilidade de se utilizar agentes móveis em redes onde existe a necessidade de um grande número de migrações (pontos gerenciados por um mesmo agente) e propostas para reduzir a complexidade no gerenciamento de vários agentes com diferentes especialidades, são pontos ainda não discutidos.

Nesta dissertação, busca-se soluções para os problemas apresentados anteriormente através da fusão das melhores características disponíveis nas arquiteturas centralizada e descentralizada. De forma geral, entende-se como um meio termo entre as duas arquiteturas agregando suas principais vantagens e descartando, ou propondo, alternativas para os pontos onde sua aplicação não é recomendável.

3. Agentes e Sistemas Multi-Agentes

3.1. Introdução

A descentralização dos ambientes computacionais, o crescimento e a diversificação dos serviços e a heterogeneidade dos equipamentos de rede introduzem uma complexidade cada vez maior no que diz respeito ao controle e monitoramento desses ambientes (ELEFTHERIOU, 2000).

A adição de novos equipamentos móveis, como nós de rede, é uma das tendências que podem ser percebidas. Segundo (WANG, 2003), esses novos equipamentos muitas vezes possuem limitações em capacidade de processamento e ainda, normalmente, estão inseridos em um contexto de banda de rede limitada.

Esses são desafios em relação aos quais a forma tradicional de gerenciamento centralizado não é capaz de lidar.

> [...] O gerenciamento tradicional de redes é caracterizado pela fraca flexibilidade (apud GREGOIRE, 1995; MAGEDANZ, 1995) e escalabilidade (GOLDSZMIDT e YEMINI, 1995), e ela não se adapta as necessidades das novas tecnologias [...] em uma abordagem cliente-servidor, por exemplo, tarefas são definidas estaticamente e tendem a utilizar uma parcela significativa da largura de banda e freqüentemente o resultado é uma ineficiente distribuição de carga computacional (apud ISMAIL, 2000; LI ET AL, 2002). (EID, 2005)

Uma das tecnologias emergentes que possuem características que estão muito próximas às necessidades descritas anteriormente é a tecnologia de agentes móveis. O agente é uma entidade de software de tamanho pequeno, mas com autonomia de execução. Possui a capacidade de interagir com outras entidades e com o ambiente

onde está executando e, a partir destas interações, é capaz de tomar decisões que o levarão a executar ou não ações em resposta a esses estímulos externos (MAES, 1995). O agente móvel é um tipo especial de agente que possui a capacidade de se transferir pelos pontos da rede, adquirindo, com esta navegação, uma habilidade de interação muito mais abrangente. Aliada a mobilidade, uma das principais características de um agente é a capacidade de comunicação, onde a troca de mensagens entre entidades de software torna esse tipo de arquitetura extremamente poderosa com várias possibilidades de aplicação.

O objetivo deste capítulo é mostrar formas de utilização de agentes em redes heterogêneas e avaliar sua utilidade, suas limitações, suas vantagens e desvantagens quando aplicados em gerenciamento de rede.

3.2. Conceitos e definições

Segundo (BRAUN, 2005) a palavra agente deriva da palavra ator em Latim, ou seja, uma pessoa que atua em favor de outra.

Na ciência da computação, o termo agente tem sido usado desde meados de 1970, tendo sida introduzida na área de inteligência artificial. Um agente de software é uma entidade de software que continuamente executa tarefas dadas por um usuário em um ambiente particular e restrito. As entidades de software envolvidas podem ser programas de computador, componentes de software, ou, no conceito de orientação a objetos, apenas um simples objeto. Contudo, os verdadeiros agentes de software devem ser vistos como uma extensão de um conceito de objetos ou componentes de software mais geral, onde objetos de software são passivos e agentes são ativos. Embora isso gere alguma polêmica, existe um entendimento comum de que uma entidade de software deve exibir certas características mínimas para se qualificar como um agente. Segundo (BRAUN, 2005) essas características são:

- **Autonomia:** Agentes operam e se comportam de acordo com um plano feito por si mesmo e que geralmente está de acordo com as tarefas dadas pelo usuário. Agentes não precisam ter cada passo de seu plano estipulado pelo seu proprietário com antecedência, e eles não solicitam de seus proprietários uma confirmação a cada passo dado.

- **Comportamento Social:** Agentes têm a habilidade de se comunicar com outro agente, ou com seres humanos, por meio de uma linguagem de comunicação. A comunicação pode ser restrita a uma pura troca de informação ou pode incluir sofisticados protocolos de negociação. Uma linha de pesquisa que aborda o problema de múltiplos agentes cooperando juntos para executar uma única tarefa é chamada de sistemas multi-agentes. Neste caso, um comportamento benevolente é necessário para o sucesso do entendimento.

- **Reatividade:** Agentes percebem seu ambiente por vários sensores e são capazes de reagir a eventos identificados.

- **Pró atividade:** Agentes não somente reagem a estímulos de seu ambiente, mas eles também são capazes de tomar iniciativa.

Agentes móveis são programas de computador que agem como representantes do usuário em uma rede global de sistemas computacionais. O agente conhece seu proprietário, conhece suas preferências, e aprende através da comunicação com ele. O agente móvel é um software agente que pode mover entre pontos de uma rede (BIESZCZAD, 1998) (RUBINSTEIN, 1998). O usuário pode delegar tarefas para o agente, o qual é capaz de executá-las através da rede, de forma eficiente, movendo-se pelos pontos da rede. Agentes móveis suportam usuários migrantes (conectados em partes diferentes da rede em momentos diferentes) uma vez que podem interagir de forma assíncrona. Finalmente, o agente reporta os resultados de

seu trabalho para o usuário através de diferentes canais de comunicação assim como e-mails, Web sites, Pager, ou celulares. A Figura 3.1 mostra um exemplo de uma infra-estrutura que utiliza agentes móveis (BIESZCZAD, 1998), onde os agentes utilizam a própria infra-estrutura fornecida pelo sistema operacional e pela rede para executar e movimentar-se.

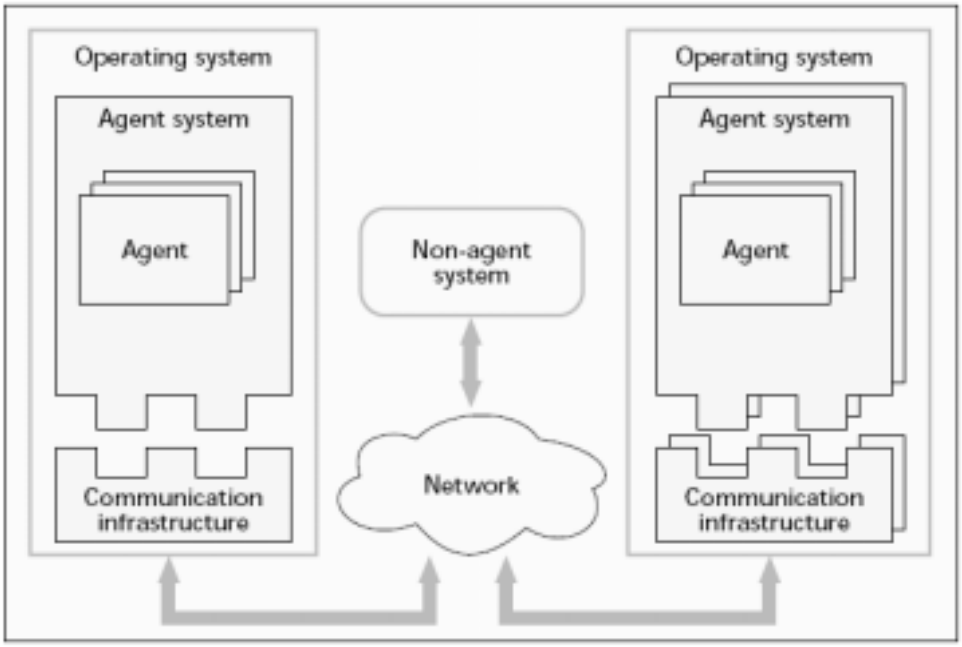

Figura 3.1 – Um exemplo de infra-estrutura de agentes móveis.

3.3. Características de Agentes Móveis

Braun enumera algumas características importantes dos agentes móveis (BRAUN, 2005):

- Agentes móveis são tipicamente usados em redes grandes e heterogêneas onde premissas de confiabilidade, relacionadas à conectividade e segurança, não podem ser feitas.

- A migração de agentes móveis é iniciada pelo agente, em oposição aos sistemas de objetos móveis tradicionais onde esta migração é iniciada pelo sistema operacional ou pelo *middleware*.
- A migração de agentes móveis é feita para acessar recursos disponíveis somente em outros pontos da rede e não para balanceamento de carga como é feito nos sistemas tradicionais.
- Agentes móveis são capazes de migrar mais de uma vez. Esta característica é chamada de *multi-hop*. Depois que um agente visitou um ponto da rede, ele pode migrar para outros pontos continuamente para continuar suas tarefas, em contraste com os códigos móveis tradicionais que migram apenas uma vez.

3.4. Vantagens dos agentes móveis

Os agentes móveis fornecem uma nova e interessante abordagem sobre sistemas distribuídos e existem muitos argumentos em favor dessa técnica. A seguir são descritas as suas quatro principais vantagens (EID, 2005) (BRAUN, 2005):

- **Delegação de tarefas:** Pelo fato dos agentes móveis serem simplesmente um tipo mais especifico de agente de software, um usuário pode utilizar um agente móvel como seu representante e delegar-lhe tarefas. Em vez de utilizar sistemas computacionais com ferramentas interativas que são capazes de trabalhar somente sob um controle direto do usuário, os agentes autônomos de software cuidarão das tarefas e trabalharão sem o controle e contato permanente. Como resultado, o usuário poderá utilizar seu tempo em outras tarefas mais importantes.
- **Interação assíncrona:** Uma vez que o agente móvel tenha sido iniciado e configurado para suas tarefas, ele pode fisicamente sair do computador de

seu proprietário e navegar livremente pela rede. Somente para essa primeira migração uma conexão deve ser estabelecida. Essa característica faz com que os agentes móveis sejam apropriados para computação móvel, onde usuários móveis poderão iniciar seus agentes de dispositivos móveis que oferecem somente uma largura de banda limitada e links de rede voláteis. Pelo fato dos agentes serem menos dependentes da rede, eles podem trabalhar de forma mais estável que aplicações baseadas no paradigma de cliente-servidor.

- **Interfaces de serviços adaptáveis:** As técnicas atuais de sistemas distribuídos oferecem interfaces de serviço, usualmente como uma coleção de funções, constituídas somente de um denominador comum para todos os possíveis clientes. Como conseqüência, muitas das funções das interfaces são mais ou menos primitivas, e clientes provavelmente terão que usar um fluxo de trabalho para realizar tarefas mais complexas. Se a sobrecarga na troca de informações entre servidor e cliente é alta, em comparação com o tempo de execução de cada função, faz mais sentido freqüentemente agregar funções mais avançadas como sendo combinações de várias funções primitivas. Contudo, é difícil enumerar todos os possíveis cenários previamente ou mesmo durante o tempo de execução. Tais funções são raramente disponibilizadas em interfaces multi-propósito. Agentes móveis podem ajudar neste tipo de situação oferecendo uma chance de projetar interfaces dirigidas ao cliente que podem ser otimizadas pelo por ele, mas adaptáveis a várias interfaces do servidor. A chave é a utilização de agentes móveis para traduzir as funções mais complexas solicitadas pelo usuário e enviar as funções primitivas ao servidor. O agente móvel simulará uma constante e altamente especializada interface para o cliente enquanto troca informações com os servidores em sua própria linguagem.
- **Transporte de código versus transporte de dados:** Essa é provavelmente a mais citada vantagem dos agentes móveis, além do estreito relacionamento e das interfaces de serviços adaptáveis. Interfaces de serviços frequentemente oferecem somente funções primitivas para acessar banco de

dados. Uma simples chamada, contudo, pode resultar em um enorme conjunto de dados sendo retornado ao cliente pela falta de precisão da requisição. Em vez de transferir dados ao cliente, onde ele será processado, filtrado, e provavelmente causará uma nova requisição, este código pode ser transferido até a localização dos dados através dos agentes móveis. No pior caso, somente os dados relevantes serão enviados ao cliente, o que reduz o tráfego de rede e economiza tempo, se o código de filtro for menor que os dados a serem processados.

- **Interação com sistema em tempo real:** Os agentes podem se instalar próximos aos sistemas evitando os atrasos que podem ser causados por congestionamento da rede.

3.5. Aplicações dos agentes móveis

Pelo fato da tecnologia que envolve o uso de agentes móveis ser relativamente nova e seus conceitos serem radicais, algum tipo de prova é necessária para mostrar que os agentes móveis, como tecnologia, são indispensáveis. Atualmente, as aplicações utilizando agentes móveis não apresentam novas funcionalidades que as técnicas tradicionais não possam implementar. Contudo, é possível identificar algumas aplicações onde os agentes móveis têm já provado o seu valor. Entre elas pode-se listar:

- **Comércio eletrônico:** Aos agentes móveis podem ser delegadas tarefas que serão executadas de forma assíncrona. Desta forma, eles podem simular, em uma interface única, um conjunto enorme de informações vindas de diferentes fontes. Isto evita grandes trocas de informação e reduz o tempo despedido pelo usuário, além da possibilidade do agente

executar todas as tarefas relacionadas à busca, negociação e conclusão de um negócio de compra em nome do usuário.

- **Retorno de informação:** Outra aplicação para os agentes móveis é a filtragem e retorno de informações. Ele pode receber a tarefa de buscar algum tipo de informação de filtragem complexa, navegar até um ou mais locais onde esta informação pode estar disponível, utilizar de todos os seus recursos para selecionar a informação necessária e retornar a estação solicitante apenas os dados importantes.
- **Tarefas de gerenciamento de redes:** Os agentes podem ter a incumbência de monitorar, configurar e agir sobre possíveis problemas detectados em um ambiente de redes, principalmente no que diz respeito ao desempenho e carga de rede. Esse tipo de tarefa evitaria a monitoração humana e reduziria a necessidade de ações desempenhadas por parte dos administradores de rede.

3.6. Comparação entre agentes móveis e aplicações cliente-servidor

Uma afirmação sempre presente nos estudos de agentes móveis é que uma das principais características e vantagens dos agentes móveis é a habilidade de economizar carga de rede em comparação com as aplicações cliente-servidor tradicionais que não utilizam agentes fixos.

O argumento principal para essa afirmação é o fato dos agentes buscarem as informações mais próximas aos dados, ao contrário dos sistemas cliente-servidor tradicionais. Segundo (JAIN, 2000), existem muitas justificativas para o uso de agentes móveis e elas são mostradas a seguir:

- Beneficio de desempenho, que inclui redução de consumo de banda de rede;
- Redução da latência;
- Redução de utilização de CPU;

- Aumento da tolerância à falhas;
- Benefícios em engenharia de software, que ajudam aos programadores a desenvolverem soluções mais conceituais permitindo uma melhor divisão de módulos e reuso de código.

Segundo Braun (BRAUN, 2005), embora muitos desses argumentos possam ser provados por experimentos, existem também casos onde os agentes móveis produzem uma carga de rede maior que os sistemas tradicionais.

3.6.1. Comparação estática entre Agentes móveis e Cliente-Servidor

Pode ser feita uma comparação estática entre os dois paradigmas, de acordo com uma análise matemática da carga de rede para uma aplicação concreta. Essa abordagem pode ser aplicada para diferentes aplicações e cenários (GRAY, 2001).

Existem estudos que afirmam que nenhum dos paradigmas é superior e que a escolha do paradigma deve ser analisada caso a caso, de acordo com o tipo específico de aplicação e dos tipos de funcionalidades que serão disponibilizadas na aplicação.

(BRAUN, 2005) utiliza um exemplo para realizar esta comparação:

Em um sistema de informação distribuído, cada um dos *N* Servidores armazena *D* documentos. Uma tarefa do cliente é baixar documentos relevantes, através do uso de palavras chaves. O servidor oferece um cabeçalho onde constam todas as palavras chaves dos documentos. Por questões de simplicidade, define-se o seguinte:
(1) A relação entre documentos relevantes e todos os documentos é igual a *i* para todos os servidores (A quantidade de documentos relevantes dividida pelo montante total de documentos).

(2) A informação do cabeçalho tem o tamanho de *h* bits para cada documento e cada documento tem um tamanho de *b* bits.
(3) As requisições enviadas do cliente para o servidor têm um tamanho de *r* bits.

Desta forma, é possível gerar as equações de carga da rede para os dois paradigmas.

Para o paradigma cliente servidor:

$$((D + iD)r + Dh + iDb)N$$

Equação 3.1 – Equação de carga Cliente-Servidor.

Em uma aplicação que utiliza o paradigma cliente-servidor (tendo um elemento centralizador no processamento), um mecanismo de procura irá interagir remotamente com N servidores de documentos. Para cada host, o mecanismo de busca irá lançar D cabeçalhos de requisições de documentos e i X D requisições de conteúdo de documentos.

A equação 3.2 é baseada na abordagem de agentes, onde o *Cma* é o tamanho do código do agente e *s* é o tamanho do estado do agente.

$$(r + Cma + s + N/2 \times iDB)(N + 1)$$

Equação 3.2 – Equação de carga - Agentes.

A cada migração, um agente móvel carrega seu código e seu estado através da rede. Para cada migração j, o tráfego é: r + Cma + S^j, onde r é o tamanho da requisição, Cma é o tamanho do código do agente e S é o tamanho do estado do componente na migração j. S^j é a soma das estruturas de dados internas que representa o estado do agente e a somatória da informação útil coletada pelo agente a cada visita. Decompondo S^j, temos s representando a soma das estruturas internas e iDB

representando os documentos coletados. Como o agente sempre migra para o seu ponto de origem, temos (N +1) migrações.

Baseado na avaliação destes modelos, com valores estimados para todos os parâmetros, pode-se selecionar um único paradigma recomendado para determinada aplicação.

Agentes móveis produzem alto tráfego de rede porque um agente transporta todos os documentos que ele já encontrou, e em todos os outros paradigmas os documentos são enviados ao cliente imediatamente. Logo, os dados do agente móvel aumentam continuamente a cada salto. Assim, ele cresce quadraticamente com o número de servidores.

A análise assume que o custo de transmissão depende apenas do número de bytes transmitidos e não da largura de banda ou dos valores de latência, mas isto é necessário para manter o modelo simples. Em uma rede uniforme, é impossível que uma abordagem de agentes móveis produza uma carga menor que uma abordagem de avaliação remota, porque o código de uma abordagem de avaliação remota é menor e os agentes móveis tem que migrar N+1 vezes, e a outra abordagem, somente N migrações são necessárias.

Existem muitos estudos relacionados à comparação de desempenho entre os agentes móveis e os demais paradigmas existentes. O que estes estudos nos mostram, assim como a abordagem mostrada anteriormente, é que cada caso deve ser analisado com cuidado, pois nem sempre os agentes móveis são superiores aos paradigmas conhecidos. A seguir são listados alguns pontos que mostram as vantagens dos agentes móveis quando algumas características estão presentes.

1. Se o número de requisições durante uma comunicação é alto, muitas transmissões de dados por parte do paradigma cliente-servidor podem ser evitadas pelo uso de agentes móveis.

2. Se o tamanho do resultado do servidor é grande combinado com a alta comprensão e fatores de filtro, então um número muito menor de bytes deve ser retornado ao cliente.

3.7. Processo de migração de agentes

Braun, (BRAUN, 2005) descreve a estrutura geral do processo de migração de agentes móveis.

3.7.1. Terminologia básica

Um agente móvel é um programa que, em muitos sistemas, executam como parte do então chamado servidor de agentes móveis. Estes servidores controlam a execução dos agentes e fornecem funcionalidades básicas para a comunicação, controle, segurança e migração dos agentes. Normalmente, este servidor é chamado de agência. Em cada sistema computacional que quer hospedar agentes móveis, uma agência do mesmo tipo deverá ser instalada (tentativas de padronização estão sendo feitas com a intenção de viabilizar a troca de agentes entre agências diferentes). Todas as agências que estão aptas a trocar agentes móveis formam uma rede lógica que é chamada de sistema de agentes móveis. Cada sistema computacional pode hospedar várias agências em paralelo, e cada agência é alcançável por no mínimo uma URL (Universal Resource Locator) para onde uma migração é direcionada. A URL também serve como o nome da agência.

Quando um agente é criado em uma agência, a agência torna-se a agência casa do agente. O usuário que iniciou o agente é chamado de proprietário do agente, e o proprietário também define o nome do agente. A informação do proprietário é

importante para mostrar para as agências estrangeiras o quão digno de confiança o agente é. O nome do agente é necessário para identificar um agente de forma inequívoca sobre todas as agências do sistema. Todas estas informações sobre a casa do agente, seu proprietário, e o nome, tornam-se atributos do agente. Normalmente, um agente retorna à sua casa depois de ter cumprido as suas tarefas. A outra agência importante é a agência que hospeda o código do agente, elas são chamadas de servidores de código. Normalmente, a agência casa é o servidor de código, mas isto nem sempre é o caso.

Agências são tipicamente sistemas multi-agentes, isto é, uma agência simples pode hospedar muitos agentes em paralelo. Para fornecer uma execução quase paralela, alguns tipos de escalonadores são oferecidos. Em muitos sistemas este processo de escalonamento não é programado dentro do software do servidor, mas é delegada a linguagem de programação e ao sistema operacional. Por exemplo, é comum que cada agente tenha sua própria *thread*. Durante a execução, o agente pode ter a permissão de iniciar uma nova *thread* filha.

3.7.2. A estrutura dos agentes móveis

Agentes móveis consistem de três componentes: código, dados e estado de execução. O código contém a lógica do agente e deve ser separado do código da agência para que ele possa ser transferido sozinho para outra agência. O código também deve ser identificável e legível para uma agência. Normalmente, como em outros programas, um código de um agente consiste de mais de um arquivo.

O segundo componente do agente é o dado. Este termo corresponde a valores de variáveis na instância do agente (assume-se um agente como sendo a instância de uma classe em linguagens orientadas a objeto). O dado é às vezes chamado de estado do objeto. É importante notar que nem todos os itens de dados de um agente são

partes do estado do objeto. Algumas variáveis referenciam objetos que são compartilhados com outros agentes ou com a agência, por exemplo, manipuladores de arquivo, *threads*, interfaces gráficas, ou outros recursos ou dispositivos que não podem se mover para outros servidores. Sendo assim, é importante restringir os dados que pertencem ao agente e que devem ser móveis.

O terceiro componente é o estado de execução. A diferença entre informações do objeto e informações do estado de execução é que os elementos do estado do objeto são diretamente controlados pelo próprio agente, enquanto as informações do estado de execução são controladas pelo processador e pelo sistema operacional. Isto depende muito de como o *toolkit* foi projetado e também da estrutura onde o agente está executando (processador, sistema operacional, etc...). Em alguns *toolkits*, um estado de execução de um agente é compreendido do valor corrente de um ponteiro de instrução e da pilha do processador. Em outros, não é possível detectar o estado de execução de um agente. Em muitos *toolkits* baseados em Java, por exemplo, o agente por si próprio é o responsável por copiar toda a informação sobre seu estado corrente no nível de linguagem de programação e restaurá-lo depois da migração.

3.7.3. A estrutura de migração

Um típico comportamento de um agente móvel é a migração de uma agência para outra. Durante o processo de migração, a agência corrente é chamada de agência transmissora e a outra agência de agência receptora. Durante o processo de migração o transmissor e o receptor devem comunicar sobre a rede e trocar dados sobre o agente que quer migrar. Sendo assim, algum tipo de protocolo é utilizado. Este protocolo é chamado de protocolo de migração. Alguns sistemas simplificam esta tarefa com a comunicação assíncrona, comparável a um envio de e-mail, contudo outros sistemas são desenvolvidos utilizando-se complexos protocolos de rede sobre o TCP/IP.

Um processo de migração completa possui seis passos executados em seqüência. Os passos S3 e R1 são executados em paralelo. A Figura 3.2 ilustra o processo de migração de um agente.

Os primeiros três passos (S1-S3) são executados na agência transmissora.

- S1: Iniciação do processo de migração e suspensão da *thread*. O processo de migração tipicamente inicia com um comando especial, o comando de migração, onde o agente anuncia a sua intenção de migrar para outra agência, cujo nome é passado como parâmetro no comando de migração. A primeira tarefa da agência é suspender a *thread* de execução do agente e garantir que nenhuma outra *thread* filha está ainda executando. Este requerimento é importante para o próximo passo, onde é imperativo que os dados e estados estejam congelados e impossibilitados de serem modificados.
- S2: Capturar os dados do agente e seu estado de execução. O estado corrente de todas as variáveis do agente são serializados, ou seja, seus valores correntes são escritos em uma representação persistente externa, por exemplo, uma memória, um bloco, um arquivo. O estado do agente é também armazenado. O resultado do processo de serialização é um agente serializado, o qual é uma *stream* de bytes que consiste dos dados e o estado do agente.
- S3: Transferência do agente. O agente serializado é transferido para a agência receptora usando o protocolo de migração.

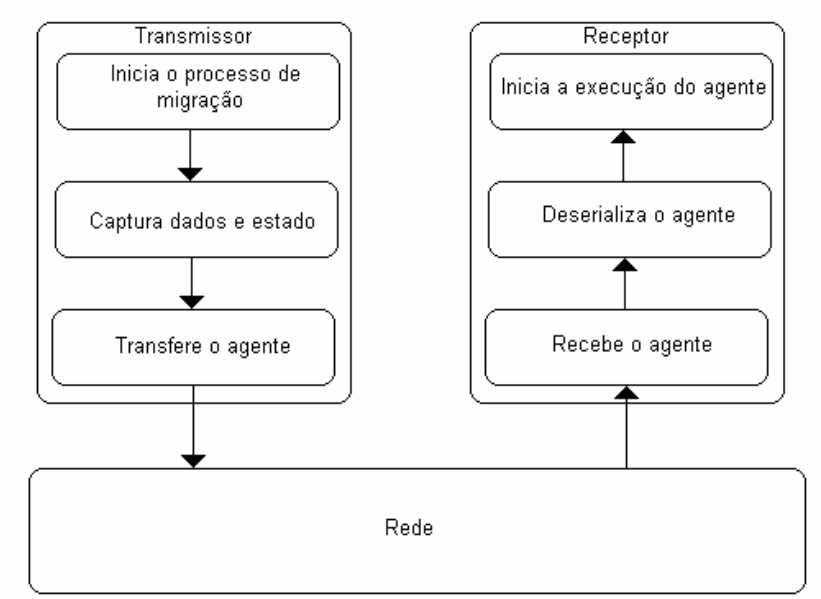

Figura 3.2 – Processo de migração de um agente.

Os três últimos passos são executados na agência receptora.

- R1: Receber o agente. O agente serializado é recebido usando o protocolo de migração. O receptor do agente verifica se o agente pode ser aceito baseado na informação do proprietário do agente e da agência transmissora. A agência receptora pode filtrar agentes de agências não confiáveis ou desconhecidas.
- R2: Deserializar o agente. O agente serializado é deserializado, ou seja, as variáveis e estado de execução são restauradas. O resultado deste passo deve ser uma cópia do agente que existia na agência transmissora exatamente antes de iniciado o processo de migração.
- R3: Iniciar a execução do agente em uma nova *thread*. A agência receptora inicia uma nova *thread* para o agente.

3.8. Comunicação dos Agentes Móveis

Problemas relacionados à comunicação têm sido exaustivamente discutidos nas áreas de inteligência distribuída e de sistemas multi-agentes. A comunicação é um dos mais importantes desafios das aplicações móveis. Aplicações pertencentes a um nível específico de complexidade não devem ser construídas usando-se apenas um único agente que transporta todo o conhecimento e estratégias. É preferível criar um conjunto de diferentes agentes, onde cada um é especializado em resolver um problema específico.

As técnicas descritas a seguir são focadas na tecnologia de agentes móveis e não em comunicação entre agentes e agências. A seguir são descritos alguns tópicos a respeito dos principais problemas de comunicação:

- Onde está o meu parceiro de comunicação? Pelo fato dos agentes móveis poderem se mover através de uma rede de forma autônoma, é geralmente impossível prever onde um agente específico, necessário para comunicação, está em um dado momento. A solução mais amigável para este problema é possuir um serviço de localização transparente. Isto faz com que seja necessário conhecer apenas o nome do agente com o qual se deseja comunicar e não a sua localização.
- Mesmo que se conheça a localização corrente de um agente, entregar uma mensagem a este agente pode causar um erro, porque o agente pode migrar no momento em que a mensagem estiver sendo entregue. O objetivo é assegurar que todas as mensagens sejam encaminhadas a um agente migrante. Este tópico é chamado de disponibilidade da comunicação. Uma forma de disponibilizar sistemas de informação é aumentar o grau de redundância e evitar situações de falha. Isto não acontece somente em sistemas de agentes móveis.

- O quanto eficiente é meu modelo de comunicação? O modelo de comunicação deve fornecer soluções para duas diferentes ações. A primeira é quando um agente migra de uma agência para outra. Isto gera, por exemplo, alguns processos de atualização do diretório de localização. A segunda ação é descobrir a localização do agente alvo e o envio da mensagem para a agência onde este reside. Outra forma de se medir eficiência de um modelo de comunicação é como este se escalona, quando o número de agentes ou agências é aumentado e se ele distingue entre migrações locais, entre *hosts* em uma sub-rede, e migração em grandes redes.

3.8.1. Modelos de comunicação para agentes móveis

Assim como para agentes em geral, as técnicas de comunicação para agentes móveis podem ser classificadas em dois modelos.

3.8.1.1. Passagem de Mensagem

O primeiro tipo de modelo de comunicação é a passagem de mensagens. Este modelo permite que agentes enviem mensagens para outros agentes. Esta é uma forma de comunicação direta onde o transmissor da mensagem deve conhecer o receptor pelo nome e sua localização corrente. Passagem de mensagem é um conceito de comunicação que forma uma base flexível para vários tipos de estratégias de comunicação complexa. Ele não define a estrutura e a semântica do conteúdo da mensagem. Contudo, esta técnica pode ser usada como base para implementar uma troca de mensagens de texto, objetos Java, ou vários tipos de outras estruturas de

mensagens de acordo com alguma linguagem de comunicação de agentes (ACL - *Agents Communication Language*), como por exemplo, o padrão FIPA.

Como parte do modelo de passagem de mensagem, deve existir algum tipo de serviço onde o agente possa encontrar nomes de outros agentes a respeito da descrição dos serviços que outros agentes fornecem. Se nenhum serviço é disponibilizado para fornecer esta informação, então este tipo de comunicação é praticamente somente entre o grupo de agentes que se conhecem com antecedência.

Uma forma simples de passagem de mensagem é a conexão ponto-a-ponto no qual um agente transmissor envia uma mensagem exatamente para um agente receptor. O transmissor solicita a agência para entregar a mensagem à caixa postal do receptor da mensagem. Somente o agente receptor será capaz de ler a mensagem. Mensagens podem ser removidas de uma caixa de correio e entregue para o endereço do agente de duas formas:

- Com a técnica empurrar, a caixa de correio ativa entrega a mensagem ao agente.
- Com a técnica de puxar, o agente busca as mensagens de sua caixa de correio.

Se o agente não está disponível localmente, o componente de mensagem é responsável por localizar o agente receptor e entregar a mensagem a ele.

Outra forma de passagem de mensagem é a comunicação multi-ponto (às vezes chamada de multicast ou broadcast). Pode-se ter, por exemplo, um grupo de agentes que estão trabalhando juntos para resolver um problema. Eles devem comunicar-se para coordenar suas atividades. Uma técnica de comunicação comum é a ponto para multi-ponto, onde o agente transmissor quer entregar uma mensagem para todos os agentes do seu grupo. Outro critério que precisa se estabelecido para a técnica de passagem de mensagem é se as mensagens a serem enviadas serão síncronas ou assíncronas. Em uma comunicação síncrona, quem for enviar uma mensagem, envia

uma mensagem para o endereçado e bloqueia a sua execução até que o endereçado tenha respondido com uma mensagem de resposta. A comunicação síncrona garante que a mensagem é entregue e em caso de erro na entrega, por exemplo, se uma expiração ocorrer, o transmissor será informado e uma exceção pode ser disparada. A comunicação síncrona naturalmente implica que ambos os agentes estarão estáticos durante o processo de comunicação.

Na comunicação assíncrona, quem for enviar uma mensagem, envia a mensagem para o endereçado e continua sua execução. Contudo, o endereçado pode também trocar de estado para aguardar novas mensagens. A comunicação assíncrona permite que o agente receptor decida de forma autônoma como reagir a mensagens que chegam. O inconveniente é que a comunicação é temporariamente atrasada e o agente transmissor não tem a garantia de que a mensagem foi entregue ao agente receptor.

3.8.1.2. Espaço de informação

O segundo modelo de comunicação é chamado de espaço de informação. O modelo espaço de informação fornece, a todos os agentes, um único espaço onde eles podem trocar informações, dados, e conhecimento com o outro. Esta é uma forma indireta de comunicação, porque os agentes não se interagem diretamente; ou seja, eles não têm endereços para postar uma informação. Um agente simplesmente escreve pedaços de dados dentro do espaço de informação e os outros agentes podem lê-los. Em alguns casos até o nome do agente é desconhecido, resultando em uma forma anônima de comunicação.

A maior diferença em relação à passagem de mensagens é que o agente não tem que decidir qual pedaço de informação deve ser enviado para quem. Na passagem de mensagens, a responsabilidade de definir qual informação deve ser enviada é feita pelo agente transmissor. Em adição, cada agente receptor deve decidir qual informação ou

resultado ele tem que receber de outros agentes, que resultado deve ser armazenado em seu repositório, e como reagir a cada mensagem.

Toda abordagem de espaço de informação temporariamente desacopla a comunicação entre os agentes, ou seja, o transmissor e o receptor não precisam estar sincronizados para comunicar-se. A seguir são mostradas duas formas de comunicação utilizando espaço de informação.

Em sistemas *blackboard*, cada pedaço de informação é armazenado sob um identificador que deve ser especificado pelo transmissor e que deve ser conhecido por todos os agentes receptores. *Blackboard* é um bom modelo colaborativo, especialmente quando se procura soluções determinísticas.

A abordagem orientada a tuplas é uma expansão dos sistemas *blackboards* com a adição de mecanismos associativos para compartilhar o espaço de informação. Os itens de dados são organizados em tuplas, ordenados como coleções de informação. As tuplas são melhor identificadas pelo seu conteúdo do que pelo seu nome.

3.8.2. Soluções que fornecem uma comunicação / localização transparente

Sistemas que fornecem localização e comunicação transparentes fornecem mecanismos para que os agentes comuniquem entre si, mesmo que em localizações totalmente diferentes, de forma totalmente transparente para o agente. Algumas ações devem estar presentes para que este tipo de serviço esteja disponível:

- Rastreamento de agentes, o qual envolve a gravação da posição corrente de um agente para ter a possibilidade de encontrá-lo depois.

- Entrega de mensagens, dita que a mensagem deve ser enviada para a localização corrente dos agentes se o agente alvo não residir na mesma agência do agente transmissor.

Existem duas abordagens totalmente opostas: a abordagem da informação completa e a abordagem de nenhuma informação. A abordagem da informação completa assume que todas as agências têm total conhecimento sobre a localização corrente de todos os agentes no sistema. O rastreamento de agentes é uma tarefa bastante pesada porque, a cada migração, todos os agentes devem atualizar sua localização em seu diretório de localização local. Para tornar este processo confiável, um agente deve notificar todas as agências sobre seu processo de migração. Apesar de necessitar de altos volumes para armazenagem da informação, esta abordagem possui um alto nível de redundância uma vez que todas as informações estão duplicadas em todas as agências. A entrega de mensagens é extremamente simples uma vez que basta consultar um único diretório para enviar a mensagem ao destino do agente alvo.

A abordagem de nenhuma informação assume-se que nenhum rastreamento de agente é estabelecido. Portanto, nenhuma agência tem o conhecimento direto da localização corrente de nenhum agente que não esteja residindo localmente. Obviamente, a migração de agentes é mais fácil, porque nenhum diretório deve ser atualizado, porém a localização de agentes e a entrega de mensagens se tornam difíceis. Em redes de grande escala, esta abordagem se tona impraticável.

3.8.2.1. Servidor central e Soluções de agência casa

A abordagem de servidor central utiliza um único servidor de localização para manter o rastro de todos os agentes móveis enquanto eles estiverem migrando pela rede. Existem duas abordagens que podem ser diferenciadas, pelo fato de ser

responsabilidade do servidor somente a localização ou também a entrega de mensagens.

Em ambas as abordagens um agente móvel deve informar ao servidor antes de ele deixar uma agência e depois de chegar a sua nova localização. Se o servidor central é responsável apenas por rastrear a localização dos agentes, o transmissor requisita a localização corrente do agente alvo e envia a mensagem para a agência. Se a localização não puder ser determinada, por exemplo, se o agente está transitando ou não atualizou seu diretório corretamente, nenhuma mensagem poderá ser entregue. A Figura 3.3 ilustra a abordagem de migração utilizando um servidor central

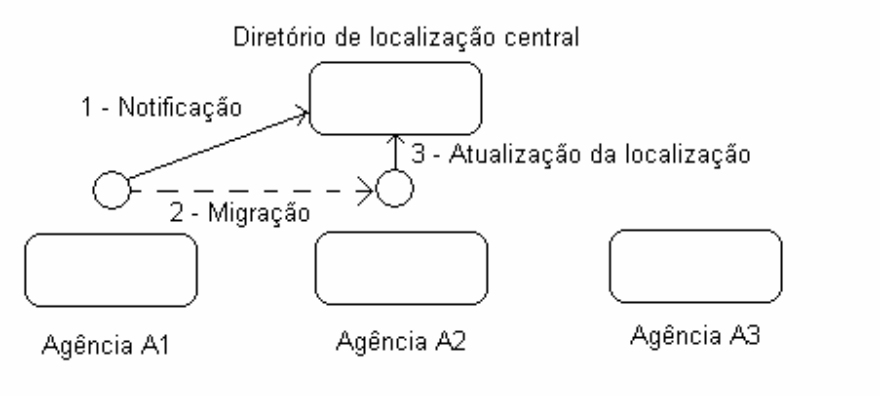

Figura 3.3 – Abordagem de servidor central.

A segunda abordagem não somente armazena a localização, mas também entrega as mensagens. Neste caso o transmissor transmite a mensagem para o servidor central, que então busca a localização corrente do agente e entrega a mensagem.
O rastreamento de agentes é difícil em ambas as abordagens de servidor central descritas, sem considerar que o servidor central se torna um gargalo para o sistema além de ser um ponto único para falha.

Um exemplo de extensão da solução servidor central é a abordagem servidor casa. Este esquema é comparado ao servidor central, porém difere pelo fato de cada agente ter seu próprio servidor central que é localizado na sua agência casa. Lembrando que a agência casa é a agência onde o agente é iniciado. A Figura 3.4 ilustra a abordagem servidor casa.

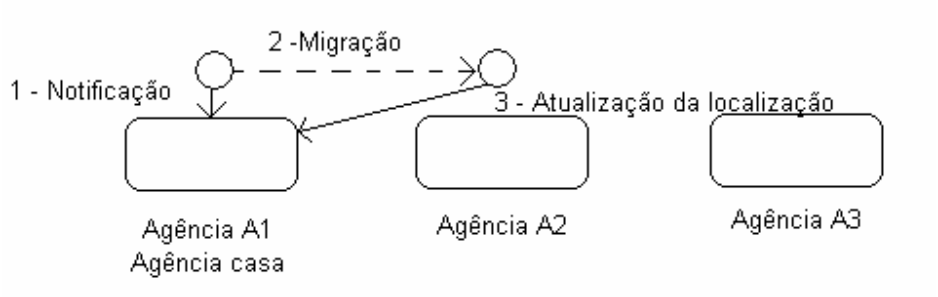

Figura 3.4 – Abordagem de servidor casa.

3.8.2.2. Ponteiros para frente

Esta solução reduz o custo de manter atualizadas as informações de localização. Para cada agente, a agência casa fornece uma âncora que pode ser usada para endereçar mensagens. Se um agente migra, ele deixa seu ponteiro para sua nova localização. Cada agência mantém um diretório local que contém a localização corrente de cada agente que a visitou, além de saber qual agente está localmente residente. Se um agente migra de uma agência A1 para a agência A2, a agência fonte, primeiro, retira o agente da lista dos agentes que estão locais. Se uma agência solicita a entrega de uma mensagem para este agente, a mensagem é armazenada localmente, porque a localização corrente do agente ainda é desconhecida. Após o agente chegar a sua nova agência, ele se registra localmente e envia uma mensagem para a agência de origem. A localização do agente passa ser conhecida e todas as mensagens armazenadas para ele são entregues em sua nova localização. A Figura 3.5 ilustra a abordagem ponteiros para frente.

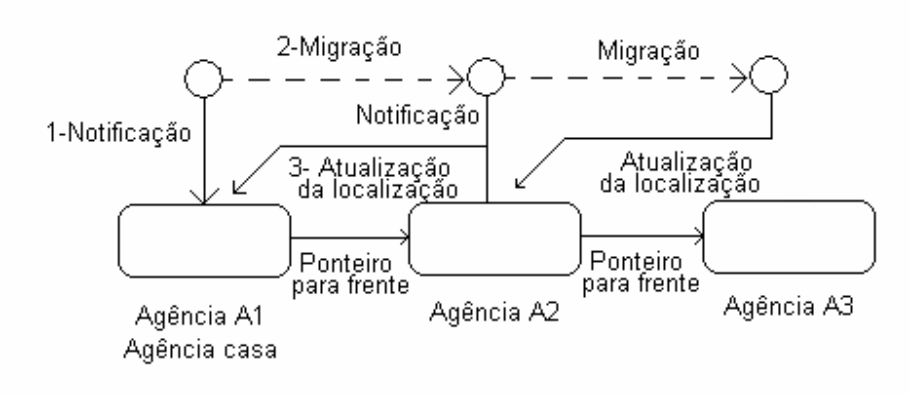

Figura 3.5 – Abordagem ponteiro para frente.

3.8.2.3. Abordagem baseada em *broadcast*

A abordagem baseada em broadcast consiste no envio de mensagens de uma única agência para muitas agências em uma rede.

A vantagem da abordagem baseada em *broadcast* é que ela não requer que uma agência específica sirva como um diretório de localização ou como um ponto para armazenar ponteiros de agentes. Essa abordagem funciona bem em cenários de redes ponto-a-ponto e estações móveis, onde não é possível fazer nenhuma premissa sobre a disponibilidade de um *host* ou infra-estrutura. O rastreamento de agentes é totalmente negligenciado nesta abordagem o que torna a migração de agentes um processo fácil. Contudo, para cada mensagem a ser entregue ao agente, a localização do agente alvo deve ser determinada utilizando o protocolo de *broadcast*, e isso pode ser considerado um método pesado de localização de agentes. A Figura 3.6 ilustra a abordagem *Broadcast*.

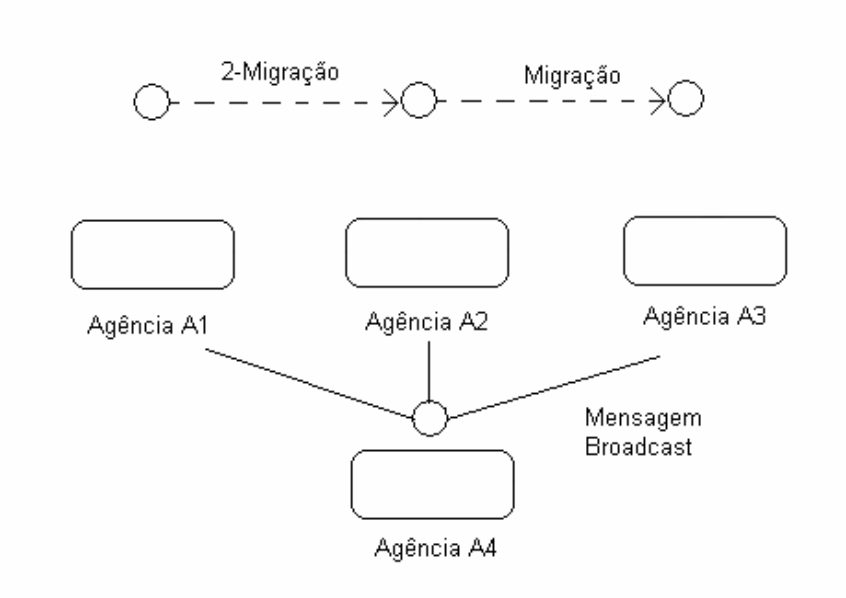

Figura 3.6 – Abordagem Broadcast.

3.8.2.4. Abordagem Hierárquica

A abordagem hierárquica é caracterizada por possuir várias camadas de diretórios de localização. Usualmente, a hierarquia é estruturada em árvore com nós representando agências, e a estrutura de árvore é construída de acordo com a estrutura geográfica da rede. Uma agência em uma folha cobre somente os agentes que estão correntemente nela localizados. Nas camadas superiores, uma agência mantém a localização dos diretórios que contém informações de rastreamento para todas as agências que residem em sua sub-árvore. Todas as agências formam uma arvore de localização. No processo de migração do agente, as informações sobre estes são adicionadas, apagadas e modificadas por toda a estrutura de sua sub-árvore, conforme mostra a Figura 3.7. Esta abordagem é interessante para grandes redes, onde o número de agências é grande.

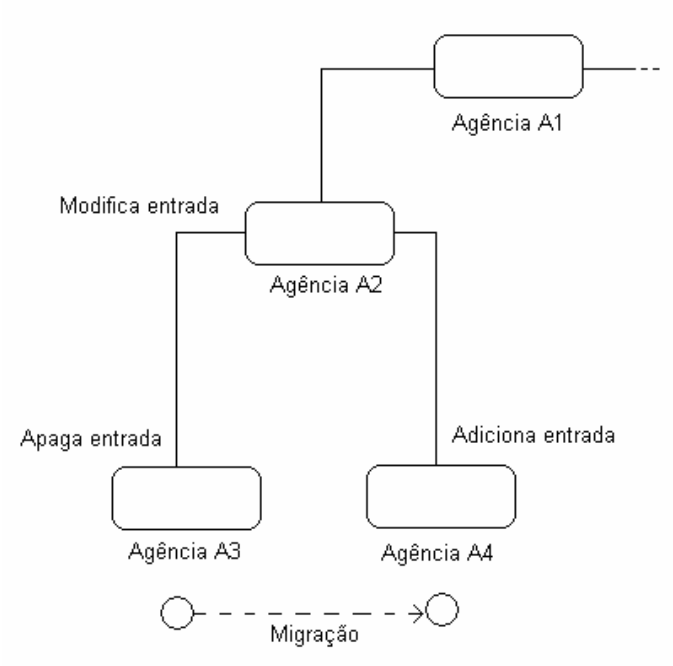

Figura 3.7 – Abordagem Hierárquica.

3.9. Normas e padrões internacionais para agentes

Como os agentes continuam se desenvolvendo e ganhado uso em aplicações industriais, esforços para padronizar suas implementações e assegurar a interoperabilidade continuam. Os dois maiores esforços para padronização são a FIPA (*Foundation for Intelligent Physical Agents*). e a OMG (*Object Management Group*). FIPA é constituída por companhias de telecomunicações e é focada primeiramente nos assuntos no nível de agentes. A OMG é um escritório de padronização que foca o nível de objetos, interoperabilidade e gerenciamento (MANOLA, 1998) (BIGUS, 2001).

3.9.1. FIPA

O primeiro conjunto de especificações FIPA, conduzido para implementações comerciais com iteratividade, foi demonstrado no padrão FIPA 97. As três maiores áreas endereçadas foram gerenciamento de agentes, comunicações entre agentes e integração entre softwares de agentes. As especificações FIPA 97 requeriam CORBA IIOP (*Internet Inter-ORB Protocol*) como protocolo de comunicação, conversão baseada na linguagem de comunicação de agentes e um conjunto de protocolos de interação com diálogos definidos entre agentes. Contudo, existiam vários fraquezas na especificação FIPA 97. O crescimento do Java RMI e http como protocolos alternativos de comunicação foi o primeiro fator no desenvolvimento de uma nova especificação e tecnologicamente mais independente.

A especificação FIPA 2000 representou o resultado de vários anos de trabalho com contribuição das maiores empresas de tecnologia incluindo SUN, IBM e HP. Talvez a grande mudança no padrão FIPA 2000 foi a definição de uma arquitetura abstrata, permitindo implementações alternativas (FIPA, 2007). A Figura 3.8 (FIPA, 2007) mostra a estrutura básica da arquitetura proposta pela FIPA.

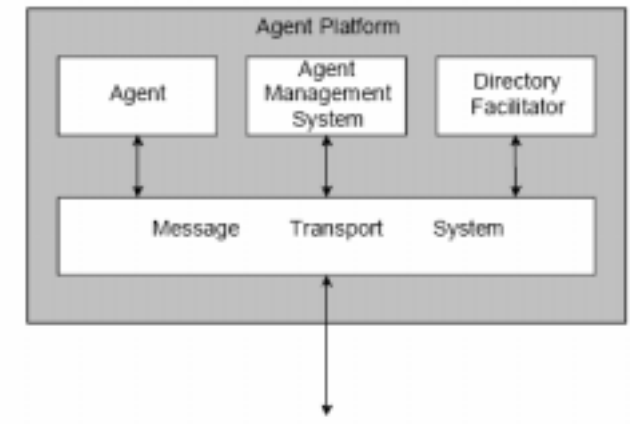

Figura 3.8 – Plataforma FIPA.

A plataforma FIPA fornece um sistema de gerenciamento de agentes (AMS) que controla o ciclo de vida do agente, um diretório facilitador (DF) que é um serviço de páginas amarelas e um sistema de transporte de mensagens (MTS) que fornece uma comunicação interna de agente para agente, assim como mensagens externas com outras plataformas em conformidade com o padrão FIPA.

3.9.2. OMG

O OMG (*Object Management Group)* tem sido o líder na área de sistemas de agentes móveis. A especificação do facilitador de interoperabilidade de sistemas agentes móveis (MASIF) aprovado em 1998 definiu como agentes podem migrar entre plataformas usando a interface de definição de linguagem CORBA (IDL) juntamente com a segurança, serviço de nomes e ciclo de vida CORBA. O MASIF é amplamente utilizado no sistema de agentes IBM Aglets. (OMG, 2007).

3.10. Ambientes e plataformas para construção de agentes baseadas em Java

Várias ações vêm sendo tomadas no sentido de difundir a plataforma Java como a principal ferramenta para desenvolvimento de agentes. Esta plataforma possui um grande potencial e é atualmente a mais utilizada devido a sua capacidade de trabalhar quase que em todas as plataformas de hardware e software disponíveis (BIGUS, 2001).

3.10.1. Agent Building and Learning Enviroment (ABLE)

O IBM ABLE é um framework para desenvolvimento e execução de agentes inteligentes híbridos e aplicações com agentes. ABLE fornece um conjunto de componentes reusáveis *JavaBeans*, chamados de *AbleBeans*, com vários métodos de interconexão para combinar estes componentes e criar softwares de agentes.
AbleBeans implementam acesso a dados, filtros e transformações, aprendizado e capacidade de raciocínio, além de classificação, *clustering*, predição e procura genética (ABLE, 2007).

3.10.2. AgentBuilder

O *AgentBuilder*, da Reticular Systems Inc., é um *toolkit* para desenvolvimento integrado de software para construção de agentes Java. Ele utiliza uma linguagem orientada a agentes de alto nível e fornece um conjunto de ferramentas gráficas de configurar ação de agentes e especificação de *behavious*. *AgentBuilder* tem a intenção de habilitar desenvolvedores que não têm conhecimento de inteligência artificial para construir aplicações de agentes.
O *AgentBuilder* possui ferramentas para desenvolvimento, depuração, gerenciamento e ferramentas de análise (AGENTBUILDER, 2007).

3.10.3. Aglets

Aglets são agentes autônomos desenvolvidos pela IBM. Eles fornecem capacidades básicas que são requeridas para mobilidade. Aglets podem comunicar-se usando *writeboards* que habilitam agentes a colaborar e compartilhar informações de forma

assíncrona. Mensagens síncronas e assíncronas são suportadas pelos aglets (AGLETS, 2007).

3.10.4. FIPA-OS

FIPA *Open Source* (FIPA-OS) originalmente desenvolvido pela Nortel *Networks*, é uma plataforma de código aberto que suporta comunicação usando os padrões de comunicação da FIPA. FIPA-OS fornece um conjunto de serviços na plataforma que são especificados no padrão FIPA, incluindo gerenciamento de agentes, gerenciamento do ciclo de vida, serviços de páginas amarelas e um canal de comunicação e troca de mensagens em conformidade com as especificações da FIPA. O FIPA-OS possui várias classes que podem ser estendidas para a adição de tarefas (*behaviours*), além de ferramentas gráficas de visualização. (FIPA-OS, 2007)

3.10.5. Gossip

Gossip é uma aplicação de demonstração de agentes móveis da *Trylluan* Inc.'s Ele utiliza tecnologia de aprendizagem para traçar o perfil e preferências de seus usuários e executar ações automáticas relacionadas ao comportamento. A implementação é feita em Java em um ambiente seguro de desenvolvimento de muti-agentes. O *agent development kit* é usado para criar agentes móveis *Tryllian*. O corpo do agente pode ser configurado através da definição de conhecimentos e comportamentos. (GOSSIP, 2007).

3.10.6. JADE

O JADE (*Java Development Framework*), desenvolvido pela CSELT em Torino na Itália, é uma ferramenta em conformidade com os padrões FIPA para criação de aplicações de sistemas multi-agentes. O JADE fornece um conjunto de ferramentas para depuração e desenvolvimento de agentes distribuídos.

JADE fornece um conjunto de serviços incluindo serviços de nomes, páginas amarelas, protocolos de transporte e protocolos de interação em conformidade dom os padrões FIPA. Também fornece um conjunto de bibliotecas para customização e criação de novos agentes além de ferramentas gráficas para gerenciamento de agentes (JADE, 2007).

3.10.7. JATLite

Java Agent Template Lite (JatLite) é um conjunto de pacotes Java, desenvolvido pela Universidade de Stanford, que pode ser usado para construir sistemas multi-agentes. JATLite é uma arquitetura em camadas que fornece diferentes protocolos de comunicação em cada camada. O *framework* JATLite foi criado para o desenvolvimento de agentes autônomos com protocolo de comunicação ponto-a-ponto. O *framework* suporta mensagens síncronas e assíncronas. O foco do JatLite é comunicação. (JATLITE, 2007).

3.10.8. Voyager

Voyager, da ObjectSpace, Inc., é um melhoramento do *Object Request Broker* (ORB) que é escrito inteiramente em Java. Um ORB fornece a capacidade de criar objetos em sistema remotos e invocar métodos nestes objetos.
Agentes *Voyager* têm mobilidade e autonomia que é fornecida pela classe base: Agent. Um agente pode mover-se de uma localização para outra deixando o endereço para que futuras mensagens possam ser encaminhadas para ele. As mensagens podem ser síncronas e assíncronas. O *Voyager* não fornece mecanismos de inteligência (VOYAGER, 2007).

3.10.9. ZEUS

O ZEUS é uma ferramenta de construção de agentes que foi desenvolvida pela British Telecom. É um *framework* para desenvolvimento de sistemas de agentes colaborativos que são construídos com grande ênfase em autonomia e cooperação. Os agentes são cooperativos porque possuem recursos e conhecimentos limitados, então combinam suas capacidades para resolver grandes problemas. ZEUS foi desenvolvido em Java devido a sua portabilidade e suporte a *multi-threading*. ZEUS possui um grande grupo de classe para desenvolvimento de agentes e um conjunto de ferramentas para visualização e construção de agentes (ZEUS, 2007).

3.11. Uso de agentes no apoio ao gerenciamento de redes

Em um modelo de rede, uma das partes mais importantes é a que se refere ao seu gerenciamento. Modelos tradicionais de gerenciamento de redes, normalmente, possuem visões estáticas do comportamento e estados limitados dos seus componentes, principalmente quando as visões e estados envolvem elementos correlacionados. As visões estáticas e estados limitados estão relacionados ao fato dos elementos de gerenciamento tradicionais não possuírem a capacidade de adaptação e de interatividade dos agentes móveis. A utilização de agentes, com todas as funcionalidades e potencialidades já descritas anteriormente, é uma abordagem alternativa para criação e manutenção de modelos de gerenciamento redes que confia no uso de agentes móveis e nos princípios de delegação. Em um modelo de gerenciamento de rede inteligente, o comportamento e estado são partes de um modelo e ambos podem ser dinamicamente atualizados. Autonomia, reatividade e pró-atividade são algumas das características que viabilizam o dinamismo e a adaptabilidade necessária.

As redes, que estão em serviço hoje, são usualmente, um conglomerado de ambientes heterogêneos, muitas vezes, incompatíveis por possuírem equipamentos de vários fabricantes. Gerenciar estas redes é um trabalho árduo para o administrador de rede que lida com a proliferação de interfaces homem máquina e problemas de interoperabilidade. Gerenciamento de sistemas de redes legadas é fortemente norteado em modelos cliente-servidor de sistemas distribuídos. Esses modelos aplicam os padrões IETF e ISO. No modelo cliente-servidor, existem agentes fornecendo acesso aos componentes de rede e poucos gerentes que comunicam com os agentes utilizando protocolos especializados como SNMP e CMIP. Os agentes são fornecedores de dados para análise. Como o modelo de gerenciamento de rede é usado para armazenar aspectos dos estados componentes de rede localmente (agentes) e através de tarefas estáticas (pré-definidas), a adaptabilidade fica comprometida. Freqüentemente um gerente tem que acessar vários agentes antes que qualquer

conclusão inteligente possa ser inferida e apresentada aos operadores humanos. O processo quase sempre envolve uma substancial transmissão de dados entre o gerente e os agentes que possam adicionar uma considerável contribuição aos dados da rede. As técnicas de delegação requerem uma infra-estrutura que forneça um ambiente de execução homogênea para as tarefas delegadas.

Uma tecnologia emergente que fornece a base para endereçar estes problemas em sistemas de gerenciamento de redes legadas é a computação de rede baseada na tecnologia Java. Java pode ser considerada uma tecnologia preferencial que outras pelo fato de sua implementação padrão possuir uma rica coleção de hierarquia de classes para redes de comunicação TCP/IP e suporte a infra-estrutura de redes industriais. Java incorpora facilidade de implementação e técnicas de gerenciamento inovadoras entre outras vantagens já comentadas anteriormente (SUN, 2007). Existem várias iniciativas de pesquisa e desenvolvimento de modelos que suportam delegação de tarefas para gerenciamento de redes.

3.11.1. Características importantes em gerenciamento com agentes

Pela introdução de mecanismos de programação móvel pode ser desenhado um sistema mais flexível. Uma parte específica (agente), tendo certos conjuntos de habilidades, pode se mover até próximo ao ponto onde existe a necessidade de gerenciamento. Com esta técnica, largura de banda pode ser economizada em certos casos e agentes especialistas podem ser utilizados.

Quando se projeta agentes para resolver problemas complexos podem-se criar alguns poucos agentes altamente avançados ou tentar dividir e distribuir o problema para ser resolvido por vários agentes menos complexos. Esse tipo de inteligência é chamado de inteligência coletiva. Esse tipo de abordagem é considerado útil em gerenciamento de

redes, principalmente quando a necessidade de intervenção envolve elementos correlacionados.

3.11.2. Pontos de aplicação de agentes móveis

Baseado nas características e potencialidades já comentadas dos agentes móveis é possível aplicar esta tecnologia em vários pontos do gerenciamento de redes. Este tipo de abordagem não retira a necessidade de operadores e administradores na rede, apenas melhora o tempo de resposta e facilita o trabalho desses profissionais.

- **Interatividade mais abrangente em intervenções em elementos correlacionados:** Os agentes móveis podem utilizar da sua mobilidade para acessar dispositivos da rede e buscar informações correlacionadas para resolver situações de crise, ou para buscar informações solicitadas pelo administrador de rede. Esta mobilidade evita que as informações tenham que ser enviadas a um ponto central para análise.

- **Gerenciamento de falhas de rede**: Devido a alto grau de distribuição implícita em sistemas com agentes móveis, a capacidade de iteração de tomada de decisões e de monitoramento e antecipação de problemas, permitem que através de ações pró-ativas as redes melhorem sua qualidade no sentido de evitar situações de falhas. O agente poderá tomar ações como: Identificar falhas, isolar causas, configurar componentes de rede visando à correção da falha ou a redução de seu impacto, armazenar informações históricas para uso posterior.

- **Configuração de redes:** Os agentes podem auxiliar nos trabalhos de configuração de redes nas tarefas de coleta de dados, onde estão próximos aos equipamentos, evitando trafego grande de informações, principalmente em períodos onde o uso é intenso. Configuração de

equipamentos de forma pró-ativa e autônoma e atualização de banco de dados de configuração de rede.

- **Segurança de redes:** Os agentes podem possuir a atribuição de controle de invasão, vírus e outros tipos de acessos não autorizados a informações disponibilizadas na rede. Isto pode ser feito através da monitoração dos pontos de acesso à rede e das informações que devem ser protegidas. Isto é possível através de ações de bloqueio de portas, bloqueio de IP's, configuração de *firewalls*, etc...

- **Desempenho / Qualidade de serviço de redes:** Os agentes podem auxiliar na monitoração da rede através da coleta de dados, na análise dos dados para identificar gargalos e padrões de tráfego, na verificação de limiares com o objetivo de evitar problemas de tempo de resposta. Isto pode ser feito com ações como configuração de equipamentos, bloqueio de tráfegos que estejam "roubando" banda, priorização de tráfegos, etc...

- **Atuação facilitada em dispositivos móveis:** Os agentes móveis podem ser utilizados no monitoramento de dispositivos com conexões voláteis e com baixa capacidade de processamento. A mobilidade permite que, durante um período de conexão relativamente curto, o agente consiga atuar neste tipo de dispositivo, mesmo que ele tenha trocas constantes de endereçamento.

Várias dessas ações são possíveis implementando agentes, utilizando, por exemplo, o suporte do protocolo SNMP. Eles podem, a cada visita a cada ponto da rede, utilizar chamadas SNMP para obter as informações e tomar as ações necessárias baseado no seu conhecimento e na sua especialidade.

A utilização de agentes é vista como um apoio ou como um método de melhorar o desempenho das operações de gerenciamento de redes, pois esta abordagem não poderá funcionar de modo totalmente autônomo visto que existem problemas onde as ações dos agentes não terão efeito ou não poderão ser realizadas. Por exemplo, quando da ocorrência de falha de hardware. Nestes casos espera-se que a detecção da falha seja feita de forma mais rápida.

3.12. Conclusão do capítulo

Neste estudo foi possível perceber a enorme potencialidade da abordagem utilizando agentes móveis inteligentes, aliado ao poder da plataforma Java. O campo de aplicação é grande e muitas vantagens podem ser obtidas a partir do seu uso como, por exemplo: realização de tarefas de forma autônoma, economia, em certos casos, de banda de rede e redução na necessidade de intervenção humana. A inteligência artificial, a capacidade de iteração mais abrangente e a mobilidade são as chaves deste processo.

Os desafios são grandes e há muito que pesquisar e desenvolver para que esta tecnologia seja viável, confiável e segura. Porém, as perspectivas são excelentes. Existem inúmeras iniciativas descentralizadas seguindo por caminhos bastante diferentes. Apesar dos esforços, existe ainda uma necessidade maior de padronização destas tecnologias para viabilizar a interoperabilidade e evitar problemas de comunicação.

Especificamente no gerenciamento de redes, existem vários pontos onde a utilização dos agentes móveis inteligentes é viável e porque não dizer necessária. A navegação por nós heterogêneos onde ferramentas de gerenciamento têm dificuldade de interagir, a redução do *overhead* de gerenciamento, a capacidade de interação entre elementos próximos e correlacionados e a tomada de ações rápidas e pró-ativas são fatores que atestam a sua necessidade.

4. A arquitetura de agentes móveis mutáveis

Este capítulo apresenta a arquitetura proposta nesta dissertação. Na seção 4.1 é dada uma introdução da arquitetura proposta. A seção 4.2 destaca o uso da arquitetura centralizada no gerenciamento de redes, mostrando suas vantagens e desvantagens. A seção 4.3 mostra as vantagens e desvantagens da utilização de agentes móveis como ferramenta de auxílio no gerenciamento de redes. A seção 4.4 descreve a arquitetura de agentes móveis mutáveis e o que ela adiciona a arquitetura tradicional de agentes. A seção 4.5 combina as abordagens centralizadas e de agentes móveis com as novas características dos agentes mutáveis de modo a obter uma nova arquitetura. A seção 4.6 descreve a arquitetura proposta.

4.1. Introdução

A arquitetura proposta neste capítulo é parte da plataforma defendida por Monteiro (MONTEIRO, 2005), onde é apresentado um sistema autônomo de gerência de redes com o uso de agentes móveis inteligentes.

A arquitetura proposta nesta dissertação é baseada na combinação das potencialidades de duas das principais arquiteturas existentes: centralizada e distribuída.

4.2. A arquitetura centralizada no gerenciamento de redes

As principais arquiteturas de gerenciamento de rede utilizam o protocolo SNMP (*Simple Network Management Protocol*) como forma de interação entre um sistema central de gerenciamento e os dispositivos distribuídos pela rede (ADHICANDRA, 2005). Cada nó SNMP é um processo que responde a requisições feitas por um sistema de gerenciamento central.

A arquitetura centralizada é o paradigma mais comum dos atuais sistemas computacionais. Neste paradigma, existe o papel do cliente e do servidor onde a maioria do processamento é executado. O cliente recebe solicitações da máquina servidora e responde a essas solicitações.
A Figura 4.1 mostra o principio de funcionamento de uma arquitetura centralizada.

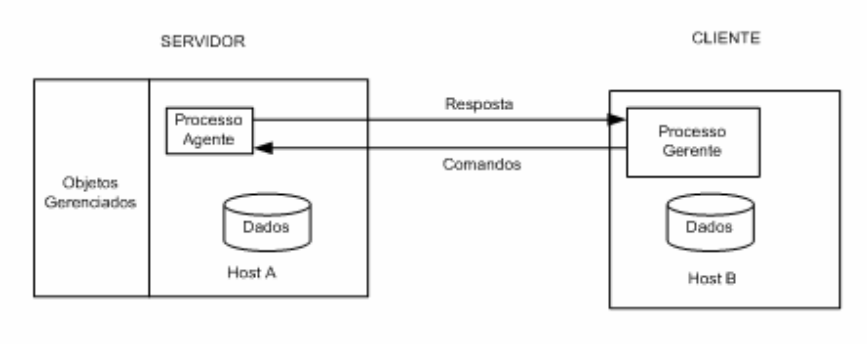

Figura 4.1 – Processo de gerenciamento centralizado.

As principais vantagens de um sistema centralizado são:

- Detecção de problemas correlacionados;
- Acessibilidade e segurança facilitadas;
- Capacidade de armazenar e manipular grandes quantidades de dados;

- Não exige alta capacidade de armazenamento e processamento dos hosts clientes (porque o processamento e a armazenagem são feitos por um elemento central);
- Velocidade na troca de informações.

As principais desvantagens são:

- A interação entre a estação de gerenciamento e seus pontos gerenciados envolve um tráfego intenso de informações, o que em certos casos pode causar sobrecarga no sistema (ADHICANDRA, 2005);
- Difícil expansão (escalabilidade);
- Dificuldade de aplicação em redes de alta heterogeneidade de equipamentos.

4.3. Utilização de agentes móveis no gerenciamento de redes (arquitetura distribuída)

Os agentes móveis realizam suas tarefas movendo-se entre diferentes nós da rede (ADHICANDRA, 2005). O conceito de agentes móveis é baseado no paradigma da programação remota (*Remote Programming* - RP), que consiste basicamente em enviar um procedimento para ser executado em outro hospedeiro. Os agentes movem-se entre nós carregando consigo as informações obtidas de execuções prévias. A Figura 4.2 ilustra o movimento de um agente durante o processo de coleta de dados.

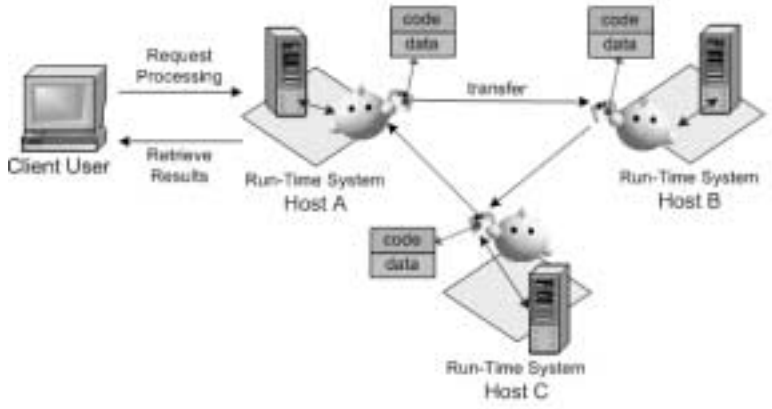

Figura 4.2 – Arquitetura de agentes móveis (WIKIPEDIA, 2007).

As principais vantagens oferecidas pelos agentes móveis são (VELLOSO, 2002) (LANGE, 1999):

- Flexibilidade;
- Interação assíncrona;
- Capacidade de aprendizagem.
- Robustez e Tolerância à falhas;
- Capacidade de realizar processamento paralelo;
- Adaptação dinâmica;
- Facilidade de utilização em ambientes heterogêneos.

As principais desvantagens da utilização de agentes móveis são (GRAY, 2001):

- Segurança das informações;
- Complexidade no gerenciamento;
- Grande tráfego de dados quando o agente precisa realizar muitos saltos entre a origem dos dados e o ponto onde as informações foram solicitadas, uma vez que os agentes levam consigo, além do código, todos os dados coletados durante sua execução.

4.4. Agentes móveis mutáveis

Agentes móveis mutáveis são agentes capazes de adicionar e subtrair, dinâmicamente, funcionalidades em tempo de execução. Essa característica permite que um agente único seja capaz de se transformar em um agente especialista através da adição de um pacote especializado. Os pacotes especializados são implementados com as tecnologias já existentes de interação com os equipamentos de rede como, por exemplo, o protocolo SNMP e inteligência artificial. Quando a ação de um determinado agente é necessária em algum ponto da rede, inicialmente cria-se um agente genérico capaz de realizar as tarefas básicas do sistema e, no momento em que este agente se deparar com uma situação que está além de sua capacidade, ele solicita a uma unidade central um determinado conjunto de funcionalidades. Após receber este novo pacote, ele o adiciona à sua estrutura atual, em tempo de execução, conforme ilustrado na Figura 4.3, se tornando um agente especialista. Entenda-se por ponto de rede, qualquer equipamento com capacidade de memória e de processamento suficiente para execução de programas.

Figura 4.3: Agente genérico se tornando especialista.

O sistema pode também fazer uso da clonagem como forma de distribuir outros agentes genéricos próximos ao local de intervenção sem necessitar enviá-los através da rede. Desta forma, somente os agentes genéricos precisariam ser criados e distribuídos. A principal vantagem desta abordagem está na economia de banda de rede, onde apenas agentes genéricos e pequenos migrariam e, no momento de uma necessidade específica, apenas o pacote necessário trafegaria pela rede. Quando este pacote não for mais necessário ele é simplesmente descartado. Outra vantagem é que não seria

necessário o desenvolvimento de vários agentes especialistas o que facilita o controle, a delegação e distribuição de tarefas.

4.5. Combinação das abordagens

Os dois paradigmas (centralizado e distribuído – utilizando agentes móveis) possuem suas vantagens e desvantagens e a escolha entre eles deve ser feita de acordo com o tipo específico de aplicação e dos tipos de funcionalidades que serão disponibilizadas. Estudos mostram que a utilização de agentes móveis é superior ao paradigma centralizado em situações onde a largura de banda é pequena e há abundância na capacidade do servidor (onde o elemento gerente está executando) (GRAY, 2001). Porém, para aplicações onde a demanda de agentes é alta, se faz necessário um esforço no sentido de controlar a utilização de centenas e até milhares de agentes necessários na execução das tarefas. Controles descentralizados são normalmente complexos e sobrecarregam os agentes com capacidades e funcionalidades que aumentam seu tamanho e complexidade. Nesta situação, um dos pontos que influenciam na utilização da banda da rede é o tamanho do código e o estado do agente (BRAUN, 2005). Como agentes móveis se movimentam pelos equipamentos da rede (JANSEN, 2005), a cada migração, o código do agente e os dados coletados por ele durante o desempenho de suas tarefas são enviados pelos dispositivos de rede. Em redes onde o número de equipamentos é grande, a movimentação entre os equipamentos pode se tornar crítica, o que não acontece em arquiteturas centralizadas onde a movimentação é única e os dados são enviados uma única vez do elemento agente para o elemento gerente (BRAUN, 2005).

As características dos agentes móveis agregadas à capacidade de adaptação dos agentes mutáveis e sua facilidade de controle, ampliam as vantagens da utilização de agentes.

Utilizando recursos destes paradigmas pode-se então maximizar as suas vantagens, ou seja, a criação de uma arquitetura mista onde características da arquitetura centralizada são mantidas, porém reforçadas com a utilização de agentes móveis mutáveis, em tarefas onde a sua utilização se mostra mais eficiente. Busca-se, deste modo, melhoras em desempenho, em flexibilidade, em redução da utilização da banda de rede, aliado a redução na complexidade do controle.

4.6. Arquitetura de gerenciamento baseada em agentes móveis mutáveis

A arquitetura é baseada na utilização de agentes móveis com as mesmas características e vantagens de um sistema multi-agente tradicional. Porém, na arquitetura proposta, diferentemente das arquiteturas descentralizadas tradicionais, o agente não volta ao seu ponto de origem com os dados coletados durante as suas tarefas. Ao término de sua execução, ele envia os resultados diretamente ao ponto de origem. Os agentes da arquitetura proposta encaminham a uma unidade central as informações coletadas a cada fim de tarefa. Uma tarefa pode envolver mais de um dispositivo a ser visitado, Neste caso, o agente migrará carregando os dados coletados até que todos os elementos correlacionados sejam envolvidos. O envio do resultado final diretamente ao elemento origem, foi baseado na arquitetura centralizada. Isto evita os problemas encontrados pelos agentes em redes de grande extensão, onde o número de migrações é alto, e, seriam necessárias várias migrações até atingir o elemento de origem, podendo comprometer, desta forma, o tempo de resposta e, dependendo da relação tamanho do código vs tamanho do dado, a banda de rede.

Os agentes móveis também não possuem um conjunto de funcionalidades fixas e isto viabiliza o uso de um mesmo agente ou de um clone em todas as possíveis tarefas disponíveis no sistema. As vantagens da capacidade de assumir vários papéis (mutação) estão ligadas à redução da complexidade no controle e ao aumento da flexibilidade do sistema. A coordenação de vários agentes com diferentes

especialidades não é necessária. Em principio, apenas um tipo genérico é necessário, bastando apenas o envio de uma funcionalidade específica. A funcionalidade específica é descartada após a conclusão da execução.

4.6.1. As plataformas JADE e JAVA

A plataforma Java se tornou uma espécie de padrão dentre os estudiosos e desenvolvedores de agentes. Inúmeras vantagens podem ser apontadas para justificar a utilização desta plataforma na construção de agentes. Entre elas, pode-se destacar, a portabilidade entre diferentes plataformas (BIGUS, 2001), além do número crescente de programadores que a utilizam.

A viabilidade de se conseguir agregar funcionalidades em tempo de execução a um agente, está no fato dos agentes serem desenvolvidos em Java e utilizarem algumas de suas funcionalidades tais como, as interfaces RMI, *Class.ForName*, e uma importante característica da linguagem conhecida como amarração tardia (*lazy binding*), ou seja, a capacidade do interpretador Java de somente incluir uma determinada parte do código no momento em que ela é necessária (NEWARD,2001).

Existem várias plataformas de suporte a agentes. A maioria delas utiliza Java como linguagem de desenvolvimento. Dentre estas, a JADE foi escolhida como a plataforma a ser utilizada nos experimentos propostos nesta dissertação. A plataforma JADE, além de ser código livre, segue as recomendações da FIPA (*The Foundation for Intelligent Physical Agents*).

4.6.2. Descrição da arquitetura

4.6.2.1. Principio de funcionamento

A arquitetura é organizada através da utilização de agentes capazes de adicionar funcionalidades em tempo de execução e uma unidade central onde a tomada de decisão do sistema acontece. A unidade central fornece as funcionalidades através de classes Java e controla todo o fluxo de agentes pela rede. A unidade central é responsável também pela armazenagem e análise dos dados coletados. A arquitetura é considerada mista, pois utiliza características distribuídas como a utilização de agentes e centralizadas quando utiliza uma unidade central para armazenagem, análise e tomadas de decisão ao sistema. A Figura 4.4 ilustra a utilização de agentes móveis mutáveis e a sua interação na disponibilização de pacotes especializados.

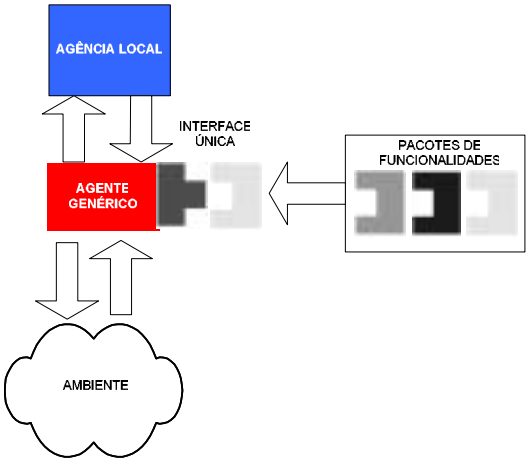

Figura 4.4 – Arquitetura dos agentes mutáveis.

O agente genérico, criado para navegar entre os pontos da rede, utiliza o suporte da arquitetura JADE para efetuar as migrações. A Figura 4.5 mostra o diagrama de classes do agente genérico.

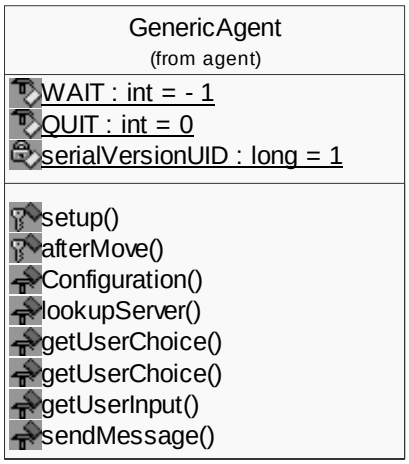

Figura 4.5 – Diagrama de classe do agente genérico.

Estes agentes genéricos possuem uma interface Java única para que diferentes funcionalidades, previamente construídas, possam aderir facilmente. A interface pode ser entendida como um protocolo de comunicação entre o agente genérico e os pacotes de funcionalidades. Estes pacotes trocam informações de forma padronizada, facilitando a compreensão por parte do agente genérico. As interfaces padronizadas são baseadas nas classes abstratas *Behaviour* já disponíveis na plataforma JADE. As classes que estendem a interface padrão são facilmente adicionadas e removidas aos agentes JADE. A Figura 4.6 mostra uma interface Java e uma classe que implementa um pacote de funcionalidade.

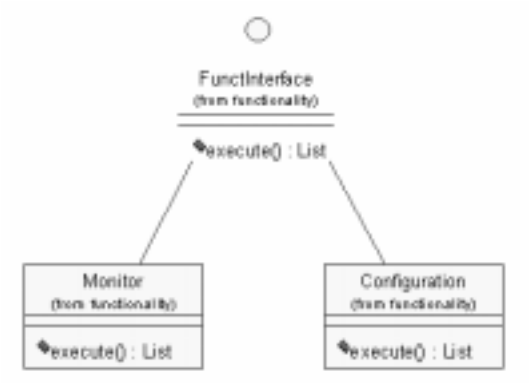

Figura 4.6 – Estrutura de classes dos pacotes de funcionalidades

O processo de solicitação e agregação de novas funcionalidades envolve as agências JADE, também chamadas de container, e um servidor central de processamento abreviado de SCP que mantém o conjunto de funcionalidades. A interação entre as agências JADE, os agentes e o SCP está demonstrada na Figura 4.8. A cada nova necessidade, o agente entra em contato com sua agência local e solicita a comunicação com o SCP. O SCP realiza uma busca em seu banco de dados para localizar as classes que se encaixam de acordo com os critérios fornecidos pelo agente. O SCP possui três módulos operacionais, a unidade de controle, responsável pela criação dos agentes genéricos e por todo o controle central do gerenciamento da rede, a unidade de comunicação que é responsável pela troca de mensagens com o meio e o módulo de gerenciamento de funcionalidades que busca, instância e trata de todas as solicitações de pacotes de funcionalidades. A Figura 4.7 mostra os três módulos e suas relações.

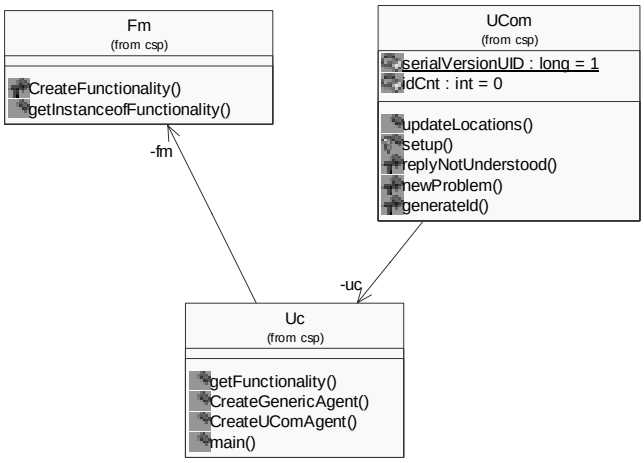

Figura 4.7 – Diagrama de classes SCP

Alguns destes módulos são agentes que estendem a classe *Agent* da biblioteca JADE e estão hospedados em um container JADE. A troca de informação entre os agentes que compõem o sistema é feita através de trocas de mensagens fornecidas pela plataforma JADE. Os agentes que compõem o SCP não fazem uso da migração, uma vez que são construídos para enviar e responder a solicitações de forma centralizada. A Figura 4.8 ilustra a troca de mensagens entre os módulos do sistema.

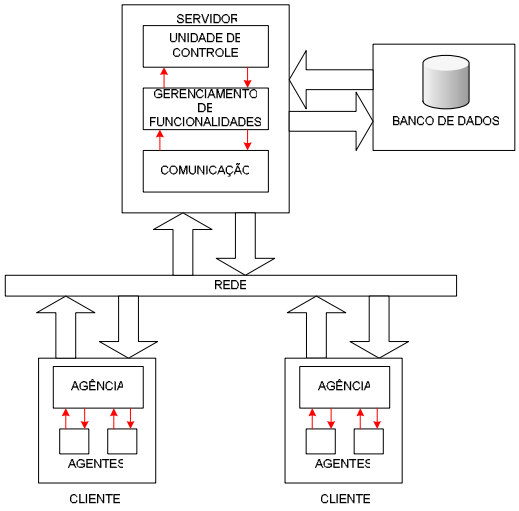

Figura 4.8 – Interação entre os agentes e o SCP.

Na arquitetura proposta, o agente genérico solicita algum tipo específico de funcionalidade através de um envio de mensagem, conforme mostra o exemplo a seguir:

```
Functionality fc = new Functionality();
fc.setName("MonitorAplicationServer" );
fc.setKeyWord("Tomcat");
sendMessage(ACLMessage.REQUEST, fc);
```

A classe: *MonitorAplicationServer,* utilizada no exemplo anterior, é responsável pela monitoração de um servidor de aplicação. Esta classe implementa a interface *FunctInterface,* especialmente para padronizar a forma de interação entre os agentes e os pacotes de funcionalidade. O agente, ao receber este objeto, faz uso da classe *Behaviour* para comunicar-se, fazendo chamadas para início de processamento, troca de informações e parada de processamento. Quando o objeto não for mais necessário, o agente poderá descartá-lo, destruindo-o. Com a utilização de uma interface única, qualquer pacote que a implemente pode ser utilizado pelo agente genérico. Após a execução do pacote de funcionalidade, este retorna uma série de informações ao agente genérico, que imediatamente as encaminha à unidade de controle através de

uma mensagem padrão. Esta mensagem é definida através de uma ontologia especialmente criada e detalhada na sessão 4.6.2.2. Ao receber as informações, a unidade central atualiza o banco de dados central e delega novas atividades aos agentes móveis. A Figura 4.9 mostra a estrutura de disponibilização de pacotes utilizada e descrita a seguir.

O gerenciador de funcionalidades ao receber a mensagem, executa o seguinte processo:

1. Busca no banco de dados a localização da funcionalidade;
2. Localiza no sistema de arquivos a classe de funcionalidade implementada;
3. Instancia a classe criando um objeto em memória, através do seu nome, utilizando o *classforname*: **(FunctInterface) fc = (FunctInterface) Class.forName (nome da classe) .newInstance();**

4. Serializa o objeto e o envia ao agente solicitante, através de uma resposta a solicitação. A transferência de objetos está disponível no pacote de recursos da plataforma JADE (BELLIFEMINE, 2005):

```
ACLMessage reply = request. createReply ();
reply.setPerformative(ACLMessage.INFORM);
Result result = new Result((Action)content, fc); //Objeto com a
funcionalidade
getContentManager().fillContent(reply, result);
send(reply);
```

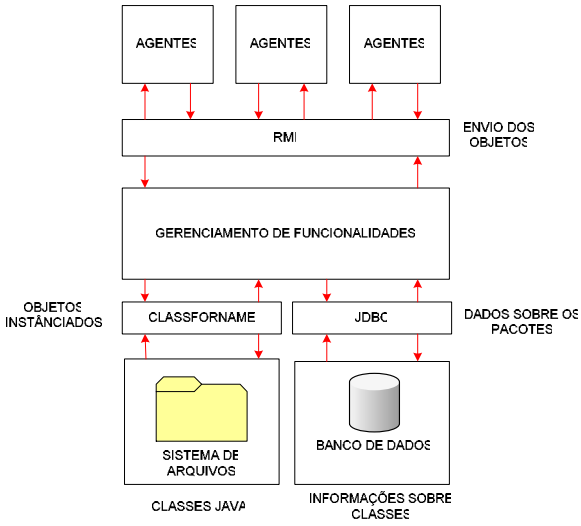

Figura 4.9 – Estrutura de disponibilização de pacotes.

O fluxograma da Figura 4.10 mostra de tratamento dado pela SCP a uma solicitação de funcionalidade enviada por um agente genérico. O processo de comunicação entre agentes é fornecido pela plataforma JADE conforme descrito do anexo A.

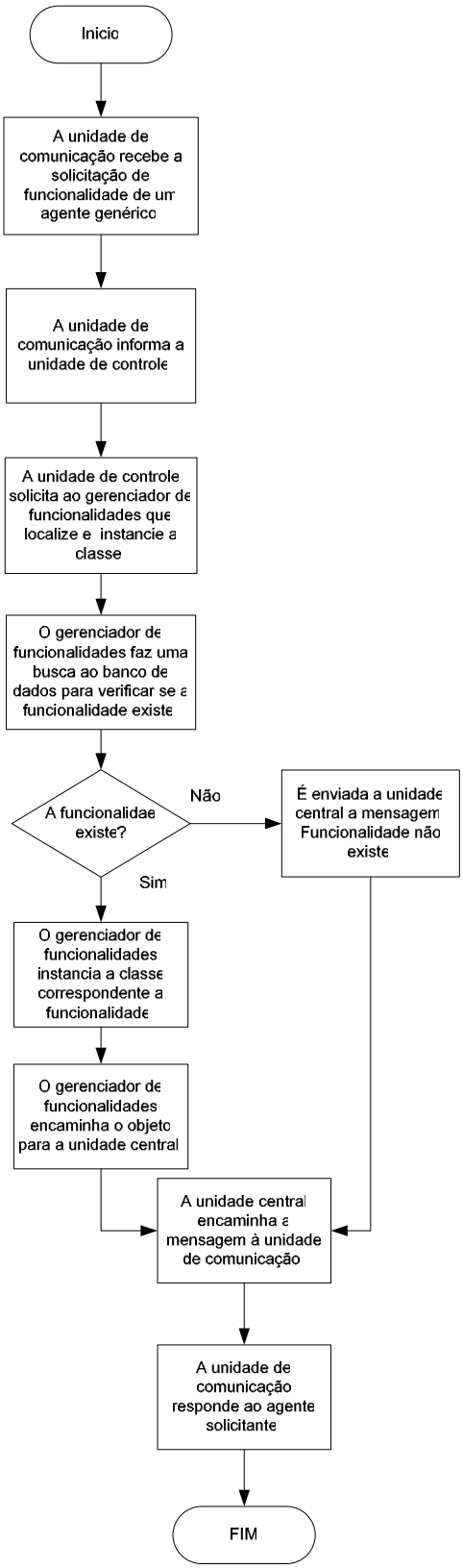

Figura 4.10 – Fluxograma de resposta a uma solicitação de funcionalidade.

4.6.2.2. Ontologia

Toda a comunicação entre agentes é realizada através de uma ontologia criada especificamente para este processo, herdando da classe *Ontology*, disponível na biblioteca JADE. O exemplo a seguir mostra a utilização da ontologia especialmente criada para a comunicação entre os agentes e a unidade central:

add(cs = new ConceptSchema (MONITOR_APPLICATION), MonitorAplication Server.class);

A Figura 4.11 ilustra a ontologia criada para trabalhar com a solicitação e envio das funcionalidades entre a unidade central e o agente genérico.

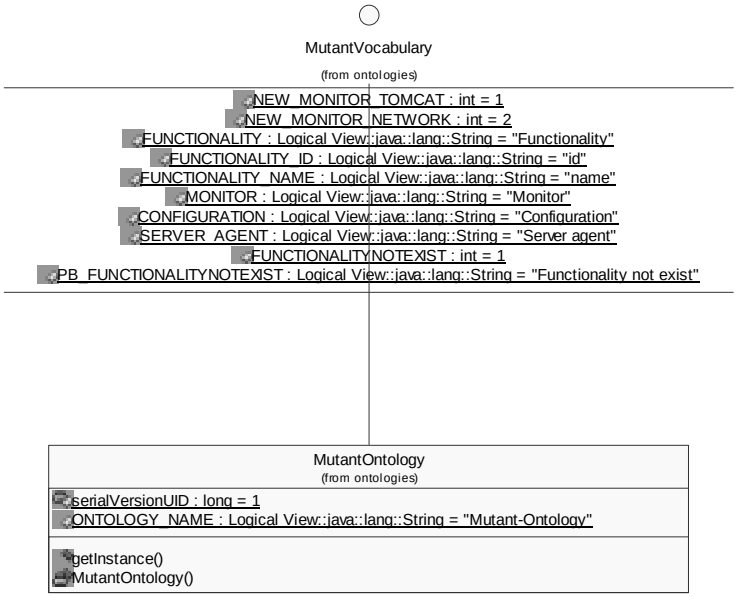

Figura 4.11 – Ontologia.

4.6.3. Banco de dados

A unidade de controle central armazena todas as mensagens enviadas pelos agentes em um banco de dados. A Figura 4.12 descreve as tabelas de funcionalidades e de armazenamento de mensagens através de um diagrama de entidade relacionamento (DER).

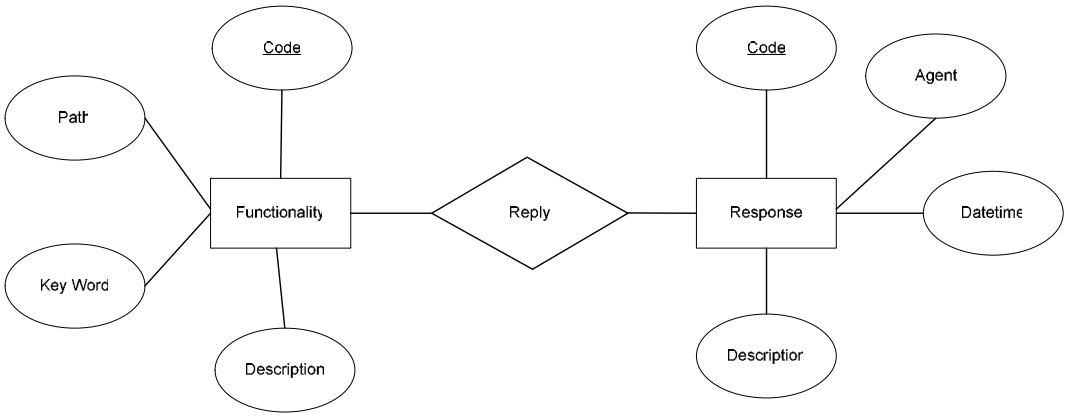

Figura 4.12 – Diagrama de entidade relacionamento.

As mensagens são armazenadas como conjunto de caracteres e ficam disponíveis na base de dados para consultas e análises por parte da unidade central de controle. A seguir é exemplificada uma mensagem retornada por um agente genérico, com a funcionalidade de buscar informações de configuração do *host*.

Code: 6
Agent: GenericAgent_0@MOURAMAX_NEW:1099/JADE
Description: Hardware: x86 Family 15 Model 2 Stepping 9 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free)1.3.6.1.4.1.311.1.1.3.1.12529256MOURAMAX_NEW76
Date Time: 2006-10-10 15:10:13

O banco de dados também é responsável por armazenar as informações referentes aos pacotes disponíveis no sistema. Através de uma palavra chave enviada pela unidade de controle, é possível identificar e localizar no sistema de arquivos a classe que implementa a funcionalidade desejada. Desta forma, é possível instanciá-la, serializá-la e enviá-la ao agente solicitante ou designado pela unidade de controle. A seguir é mostrado um exemplo de uma funcionalidade configurada.

Code: 1
Description: Funcionalidade responsável pela busca de informações de configuração do host.
Key Word: Configuration
Path: C:\funcionalidades\Configuration\Configuration.class

4.6.4. Benefícios da nova arquitetura

A arquitetura baseada em agentes móveis mutáveis adiciona às arquiteturas cliente servidor e de agentes móveis benefícios descritos na Tabela 4.1.

Possíveis benefícios	Justificativa
Auto-gerenciamento	Permite a delegação de tarefas aos agentes móveis.
Uso reduzido de CPU	Não utiliza agentes complexos e especializados e o agente somente utiliza CPU quando estiver no local de intervenção, executando alguma tarefa.
Interação autônoma assíncrona (BIESZCZAD II, 1998).	Continuidade do trabalho de gerenciamento mesmo em condições de interrupção de comunicação, por usar agentes e comunicação assíncrona (BELLIFEMINE, 2005).

Redução na utilização da banda de rede (BIESZCZAD II, 1998).	Utilização de agentes na busca de informações, aumentando a capacidade de filtragem das informações e por estar no local onde estas informações estão disponíveis. Envio das informações assim que possível ao servidor central, evitando o tráfego de agentes e dados entre pontos da rede. Clonagem de agentes genéricos próximos ao ponto de necessidade, evitando tráfego desnecessário. Tráfego de apenas agentes genéricos pequenos. Envio de pacotes de funcionalidades apenas quando estas forem necessárias, além de permitir o descarte destas antes de efetuar uma migração.
Melhoria no tempo de resposta	Permite a clonagem e envio de agentes genéricos próximo aos locais dos eventos.
Redução da complexidade de controle	Não necessita de criar, enviar, controlar e descartar vários agentes especializados. Um único agente é capaz de realizar todas as tarefas disponíveis no sistema.
Segurança	As informações são armazenadas de forma centralizada e não ficam disponíveis nos agentes, reduzindo os possíveis pontos de ataque (pontos a serem protegidos).
Escalabilidade / Mutabilidade	Novas funcionalidades são facilmente adicionadas bastando apenas a criação de novos pacotes especialistas.
Suporte a ambientes heterogêneos	Utilizando a plataforma Java, agrega-se a capacidade de execução em vários ambientes diferentes (BIESZCZAD II, 1998).

Tabela 4.1 – Potenciais vantagens da arquitetura baseada em agentes móveis

4.7. Conclusão do capítulo

As duas principais arquiteturas de gerenciamento de redes utilizadas possuem uma série de vantagens e desvantagens e a sua utilização dependente do ambiente e do tipo de serviço aos quais esses serão empregados.

A arquitetura de agentes móveis mutáveis faz uma mescla das duas arquiteturas de forma a aproveitar os pontos positivos (vantagens) de cada uma das arquiteturas apresentadas e empregando a capacidade de mutação dos agentes para corrigir algumas de suas desvantagens.

Conforme descrito no item 4.6.3 deste capítulo, os benefícios das duas arquiteturas aparecem juntos na arquitetura proposta.

A arquitetura proposta pode, portanto ser utilizada tanto em ambientes propícios para a arquitetura centralizada, quanto para a arquitetura descentralizada, com possibilidade de ganhos em desempenho, uso de recursos da rede, complexidade e flexibilidade.

5. Estudo de caso e simulações

Este capítulo descreve o estudo de caso e a simulação realizada com o objetivo de testar o funcionamento da arquitetura proposta e ao mesmo tempo colocar em prova parte de sua capacidade.

A seção 5.1 detalha a forma que o estudo de caso foi desenvolvido. A seção 5.2 descreve as simulações e seus resultados e faz uma comparação entre as arquiteturas utilizadas.

5.1. Estudo de caso

Para testar a viabilidade técnica da arquitetura proposta, foi implementada uma aplicação com os seguintes requisitos de funcionamento:

- Criação do agente UCom.
- Criação de agentes genéricos
- Solicitação de migração
- Envio de pacotes especializados
- Visualização e armazenamento dos resultados retornados pelos agentes

A Figura 5.1 mostra o conjunto de funcionalidades disponíveis ao usuário da aplicação.

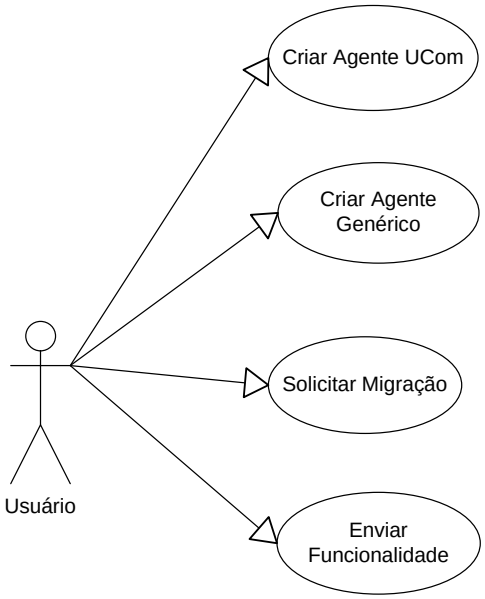

Figura 5.1 - Funcionalidades

A partir desta aplicação, deve ser possível a criação de agentes genéricos, a migração de agentes entre containeres JADE, o envio de pacotes de funcionalidades e o armazenamento do retorno dos dados coletados pelos agentes. Este estudo de caso não tem o objetivo de criar um software com aplicação prática, visa apenas atestar a viabilidade de um mesmo agente assumir várias tarefas diferentes, em locais diferentes, sem necessidade de carregar consigo todo o código e todos os dados coletados, conforme detalhado na descrição da arquitetura no capítulo 4.

A seguir está detalhada a plataforma utilizada na construção da aplicação:

- Java / J2EE: JDK versão: 1.4.2.03 - J2EE versão: 1.3.1
- Banco de dados MySQL versão 5.0
- Sistema Operacional Windows XP
- Servidor de aplicação Tomcat versão 5.28
- Plataforma JADE versão 3.3
- Pacote de integração SNMP - SNMP4J Apache Group
- Driver de conexão mysql connector java versão 5.0.3

A Figura 5.2 ilustra a arquitetura definida para a construção da aplicação. Foi criado um *servlet* que interage com o ambiente JADE e com o servidor de aplicação Tomcat, viabilizando a criação e comunicação com agentes, interação com um ambiente web e com o banco de dados.

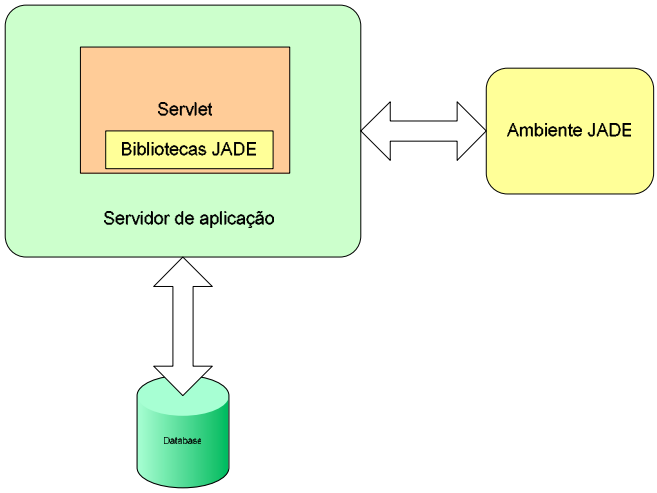

Figura 5.2 – Arquitetura básica da aplicação

5.1.1. Descrição da solução

A plataforma JADE disponibiliza mecanismos para interação entre uma aplicação Java externa e uma instância JADE (Container), sendo então possível criar containers e agentes através do uso de bibliotecas JADE.

A biblioteca que disponibiliza esta funcionalidade é a *jade.core*. O método estático *jade.core.Runtime.instance*() instancia o ambiente de execução JADE do tipo *jade.wrapper* que empacote as funcionalidades de alto nível dos containeres e através de métodos como *createNewAgent*() é possível criar agentes e através do ambiente de execução JADE interagir com eles. A lista de código dos principais módulos deste protótipo está disponível no anexo B.

O servlet gera uma interface web para interação do usuário com o ambiente de execução dos agentes, conforme ilustra a Figura 5.3.

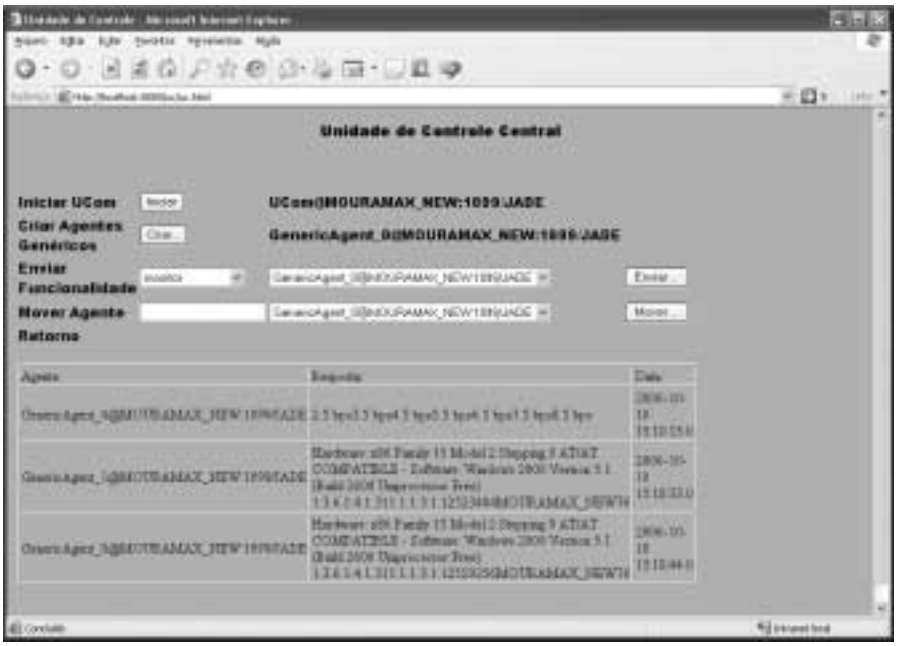

Figura 5.3 – Interface de interação usuário e ambiente JADE

O *servlet* cria, inicialmente, um agente do tipo UCom, (unidade de comunicação) que é responsável pela comunicação entre a unidade de processamento central e os demais módulos envolvidos no sistema. Toda vez que é solicitada alguma funcionalidade do sistema, o agente UCom comunica com o módulo a ser acionado. Por exemplo, caso seja solicitada a criação de um agente, o usuário interage com a interface através do botão “Criar...”, imediatamente o agente UCom comunica-se com o agente Uc (unidade de controle) que cria o novo agente. Quando o envio de uma funcionalidade é solicitado a Uc interage com o gerenciador de funcionalidades (Fm), este responsável pela busca e criação do objeto funcionalidade. No final de cada processo, o agente UCom é o responsável pela interação entre os agentes, conforme descrito no capítulo 4 e ilustrado pelo diagrama da Figura 5.4.

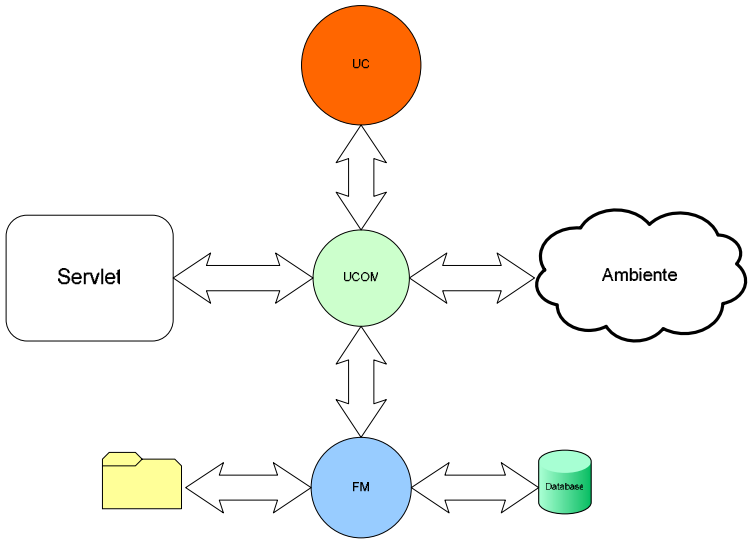

Figura 5.4 - Diagrama de relacionamento

5.1.2. Pacotes de funcionalidades

Para atestar a capacidade dos agentes em assumir papéis diferentes, de acordo com a necessidade, foram criados dois pacotes de funcionalidades. O primeiro nomeado de "*Monitor*", que é um pacote que gera, de forma aleatória, strings com mensagens de utilização de banda. Este pacote foi criado para simular a monitoração de um ponto de rede real. O segundo pacote nomeado de "*Configuration*" se integra com a biblioteca de SNMP4J e busca o objeto *SysDescr* do grupo *system* identificado na MIBII como 1.3.6.1.2.1.1.1 e retorna uma descrição textual da unidade, podendo incluir o nome e a versão do hardware, sistema operacional e o programa de rede.

É importante destacar, que todas as bibliotecas Java necessárias para o funcionamento dos pacotes de funcionalidades devem estar previamente disponíveis em todos os pontos da rede onde este agente irá executar.

5.1.3. Testes de funcionamento

Criação e Migração

A Figura 5.5 mostra a interface disponibilizada pela plataforma JADE para acompanhamento e interação com o ambiente de agentes. Nesta figura estão sendo mostrados dois agentes genéricos criados a partir do *servlet*, em *containers* diferentes.

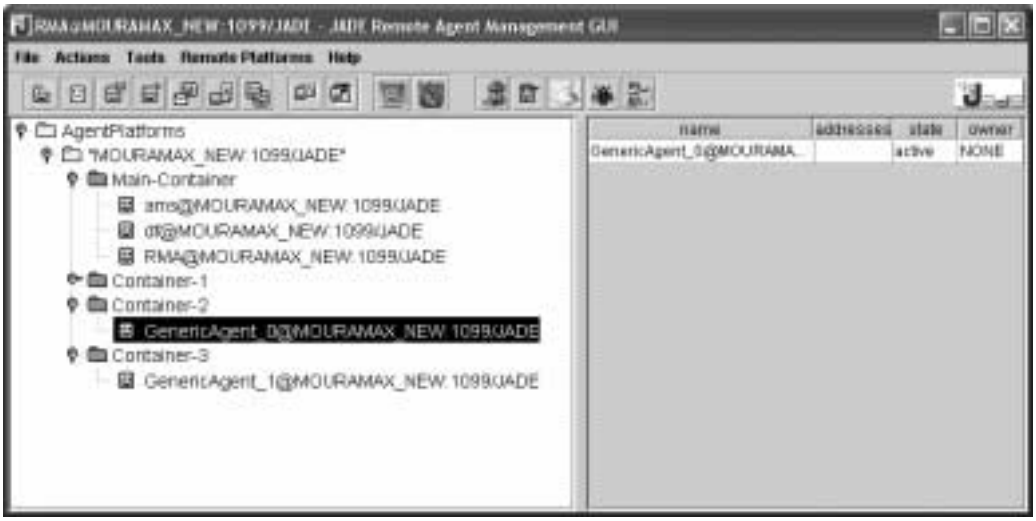

Figura 5.5 – Interface JADE

Estes dois agentes foram criados primeiramente com a finalidade de testar a mobilidade através de um comando via interface web. A Figura 5.6 mostra o agente: GenericAgent_1@MOURAMAX_NEW_1099 migrando para o container superior, após um comando de migração na interface web.

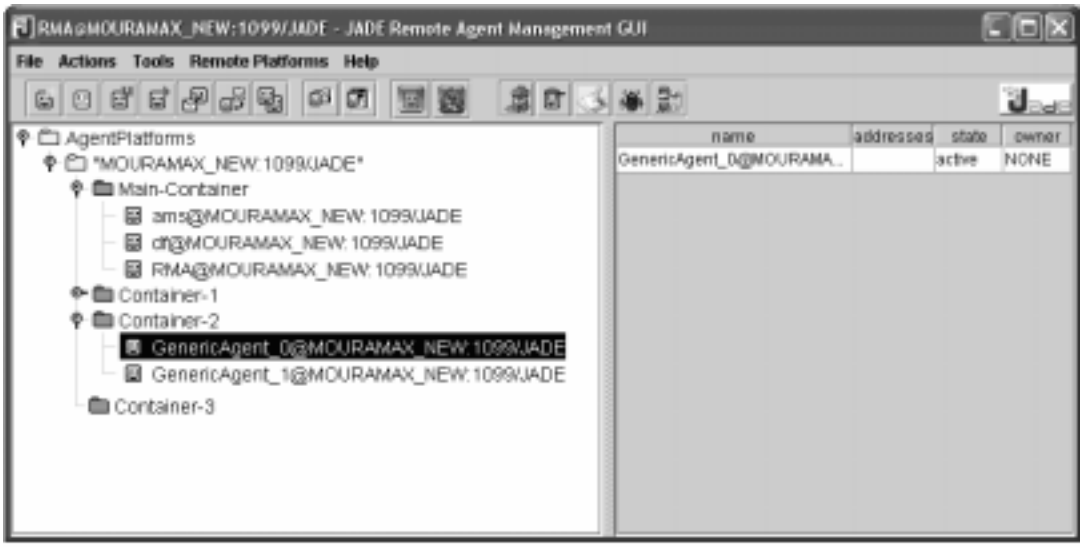

Figura 5.6 – Agente após a migração

A mensagem enviada para o agente genérico (mostrada a seguir) foi capturada do aplicativo Sniffer (Figura 5.7) que acompanha a ferramenta JADE.

```
((action
   (agent-identifier
    :name GenericAgent_1@MOURAMAX_NEW:1099/JADE)
   (Moving
    :id "1"
    :name Container-2)))
```

Figura 5.7 – Comunicação entre o Ucom e o agente genérico

Envio de funcionalidades e resposta

A seguir são mostradas as mensagens de envio de funcionalidade, que são trocadas entre o agente UCom e os agentes genéricos.
Envio da funcionalidade monitor ao agente: GenericAgent_0:

```
(action
   (agent-identifier
    :name GenericAgent_0@MOURAMAX_NEW:1099/JADE)
   (Functionality
    :id "1"
    :name monitor)))
```

Envio da funcionalidade monitor ao agente: GenericAgent_1:

```
((action
   (agent-identifier
    :name GenericAgent_1@MOURAMAX_NEW:1099/JADE)
   (Functionality
```

```
    :id "1"
    :name monitor)))
```

Envio da funcionalidade *configuration* ao agente: GenericAgent_0:

```
((action
   (agent-identifier
    :name GenericAgent_0@MOURAMAX_NEW:1099/JADE)
   (Functionality
    :id "1"
    :name configuration)))
```

A Figura 5.8 mostra as mensagens de retorno na interface gerada pelo servlet.

Agente:	Resposta:	Data:
GenericAgent_0@MOURAMAX_NEW:1099/JADE	2.5 bps3.5 bps4.5 bps5.5 bps6.5 bps7.5 bps8.5 bps	2006-10-10 15:10:13.0
GenericAgent_1@MOURAMAX_NEW:1099/JADE	Hardware: x86 Family 15 Model 2 Stepping 9 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free) 1.3.6.1.4.1.311.1.1.3.1.12523404MOURAMAX_NEW76	2006-10-10 15:18:33.0
GenericAgent_0@MOURAMAX_NEW:1099/JADE	Hardware: x86 Family 15 Model 2 Stepping 9 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free) 1.3.6.1.4.1.311.1.1.3.1.12529256MOURAMAX_NEW76	2006-10-10 15:18:44.0

Figura 5.8 – Respostas dos agentes

Percebe-se que o *Generic_Agent_0* assumiu a funcionalidade de monitoração e logo após recebeu o pacote de busca de configuração e também assumiu esta nova funcionalidade.

Os experimentos deste estudo de caso atestam a viabilidade de se construir agentes capazes de assumir vários papéis em um ambiente de rede, bastando apenas a agregação de novas funcionalidades.

5.2. Simulação

A proposta desta simulação é verificar, em condições similares a uma situação do mundo real, o desempenho que as três principais arquiteturas de gerenciamento teriam em relação à sobrecarga de dados na rede. Esta simulação não tem o objetivo de utilizar a forma mais eficaz de consulta e retorno, e sim criar um ambiente aproximado simulando a forma com que as arquiteturas trabalham. Algumas trocas de mensagem não foram consideradas nesta simulação para simplificar os testes, o tamanho destas mensagens não influenciam de forma significante no resultado final.

O resultado é a comparação entre as arquiteturas centralizadas, totalmente distribuídas (utilizando agentes especialistas) e a arquitetura proposta nesta dissertação. Com esta comparação é possível determinar, a partir de um cenário conhecido, a sobrecarga total na rede quando é necessária a busca de informações em pontos distribuídos. Pode-se, deste modo, simular o tráfego de informações similar ao que ocorre em ambientes de gerenciamento de rede, e, a partir destas informações, selecionar a plataforma que gere um menor impacto.

Foram construídas três situações que simulam a busca de informações em pontos específicos de uma rede, utilizando três cenários diferentes, um para cada uma das arquiteturas descritas anteriormente. Foram interligados quatro servidores através de uma rede Ethernet de 100Mbits/s sendo um deles nomeado de servidor principal, onde os dados são acumulados e armazenados, e três servidores de banco de dados nomeados como servidor de banco de dados 1, 2 e 3, onde as informações estão disponíveis para consulta. Cada servidor de banco de dados possui a seguinte configuração:

- Sistema operacional Windows 2000 Server
- Banco de dados Mysql versão 5.0
- Driver de conexão mysql connector java versão 5.0.3
- Java SDK versão: 1.4.3.03

O servidor principal possui a seguinte configuração:

- Sistema operacional Windows 2000 Server
- Banco de dados Mysql versão 5.0
- Driver de conexão mysql connector java versão 5.0.3
- Plataforma JADE versão: 3.3
- Servidor de aplicação Tomcat versão: 5.28
- Java SDK versão: 1.4.3.03

A Figura 5.9 mostra a estrutura de servidores que foi empregada nas simulações.

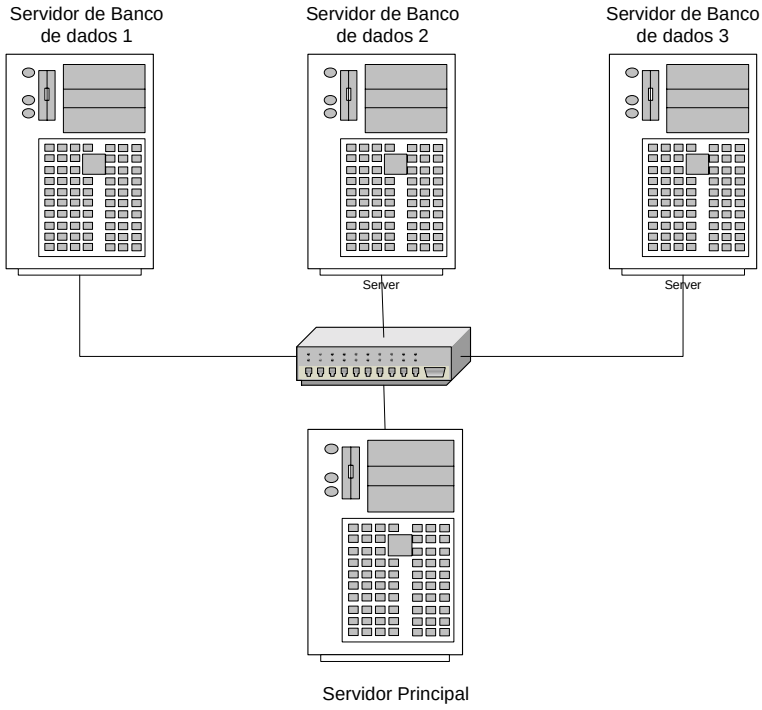

Figura 5.9 – Estrutura dos servidores utilizados na simulação

Em cada servidor de banco de dados foi criado um banco nomeado de Simulação no ambiente mysql com apenas uma tabela nomeada como Dados. A Figura 5.10 mostra a estrutura física desta tabela.

Dados		
PK	id	INTEGER
	Numeros	INTEGER

Figura 5.10 – Estrutura da tabela dados.

Na tabela dados foram inseridos mil registros com números aleatórios na coluna Números que é do tipo *Integer*, o tipo *Integer* no banco de dados mysql possui um tamanho máximo de 4 bytes (MYSQL, 2007). Os dados armazenados na coluna Números simulam uma informação disponibilizada em um *host* remoto, que em um ambiente de gerenciamento, uma determinada arquitetura deverá buscar e retornar para análise e tomada de decisão.

Cada sistema deverá buscar os 1000 números disponíveis em cada uma das tabelas de cada servidor de bancos de dados e retornar a somatória destes números. Portanto, o resultado de cada simulação será:

$$\sum_{1000}(Numeros)$$

Equação 5.1 – Resultado da simulação

Cada sistema deverá converter o resultado da simulação em uma string e enviá-la ao servidor principal para armazenamento.

5.2.1. Sniffer

Os Sniffers são aplicações usadas em redes de computadores, para monitorar, interpretar e validar tipos de tráfego. Basicamente permitem a captura de pacotes nos vários níveis do modelo OSI e a sua observação por parte do administrador (CARDANA, 2006).

Para monitorar a troca de informações entre o servidor principal ou entre cada servidor de banco de dados, foi inserido um aplicativo Sniffer especialmente desenvolvido para capturar a quantidade de bytes transmitidos pelos softwares envolvidos na simulação de forma a permitir a análise do tráfego envolvido.
O resultado da monitoração é a geração de três arquivos no formato texto com o acumulado de bytes gerados pelos acessos das três arquiteturas durante o processo de simulação.

5.2.2. Descrição dos cenários

Descrição do cenário 1

O cenário 1 foi construído para levantamento de dados através da arquitetura centralizada. Foi utilizado o servidor de aplicação *Tomcat* e o *driver* de conexão mysql *connector* java para a criação de um aplicativo simples que acessa direto o banco de dados e possui a capacidade de pesquisar e tratar os dados disponibilizados.
A simulação foi executada da seguinte forma:

- Execução da seguinte *Query* de pesquisa nos três servidores de testes:
  Select numeros from dados;
- Retorno dos dados solicitados.

Descrição do cenário 2

Para a arquitetura distribuída (agentes tradicional), foi criado um agente JADE especializado com a capacidade de pesquisar e tratar os dados disponibilizados no banco de dados.

A simulação foi executada da seguinte forma:

1. Envio de um agente especializado para dois dos servidores de testes;
2. Migração de um dos agentes para o terceiro servidor de testes;
3. Retorno dos dados ao servidor de origem.

Descrição do cenário 3

Para a arquitetura baseada em agentes móveis mutáveis foi criado um agente JADE genérico e um pacote especialista capaz de pesquisar e tratar os dados disponibilizados no banco de dados.

A simulação foi executada da seguinte forma:

1. Envio de um agente genérico a dois dos servidores de teste;
2. Envio dos pacotes especializados aos agentes enviados;
3. Migração de um dos agentes para o terceiro servidor de testes;
4. Retorno dos dados ao servidor de origem.

Detalhamento dos módulos envolvidos na simulação

A Tabela 5.1 mostra o tamanho em bytes de cada módulo utilizado na simulação.

Módulo	Tamanho
Agente genérico	5,32K Bytes
Pacote de funcionalidade	941 Bytes
Agente especialista	6,3K Bytes
Total de dados armazenados na	100K Bytes

tabela	
Quantidade de bytes após somatório e conversão para String	20K Bytes

Tabela 5.1 – Quantidade de bytes de cada módulo na simulação

5.2.3. Resultados da simulação

Os resultados da simulação mostraram que a diferença na quantidade de dados trafegados na rede entre a arquitetura de agentes e a arquitetura cliente servidor cresce quando os dados podem ser processados antes do envio. Isto comprova que, quanto maior a capacidade de tratamento das informações no ponto de origem, menor será a necessidade de tráfego.

A comparação entre as duas arquiteturas de agentes mostrou que o tráfego na arquitetura tradicional aumenta proporcional a quantidade de saltos, ou seja, quanto maior a quantidade de migração entre pontos da rede, maior será a sobrecarga de dados gerada pela arquitetura de agentes especialistas. Outra informação importante está no fato de que, caso haja a necessidade de busca de outro tipo de informação na arquitetura de agentes especialistas, outro agente deve ser criado e enviado ao servidor, o que não ocorre na arquitetura de agentes mutáveis onde apenas é necessário o envio de um novo pacote a ser agregado a um clone do agente genérico.

O descarte de pacotes especialistas e a disponibilização de um mesmo agente ou de um clone para executar uma nova tarefa reduz substancialmente a complexidade no controle do sistema porque é necessário apenas um tipo de agente para executar todas

as tarefas disponíveis. Desta forma, um banco de informações sobre quais agentes estão disponíveis em quais máquinas e a necessidade de algoritmos complexos de rotas de migrações para envio destes a outros pontos de intervenção se torna desnecessário.

A seguir são mostrados os resultados da simulação, A Tabela 5.2 mostra o total de bytes trafegados pela rede na arquitetura distribuída utilizando agentes especialistas. Foi necessário o total de 31580 bytes trafegados pela rede para a conclusão total da simulação.

Migração	Bytes
Servidor central para primeiro servidor	6,3K
Servidor central para segundo servidor	6,3K
Segundo servidor para o terceiro servidor	6,3K + 20 Bytes (Dados coletados no 2º servidor mais o agente)
Primeiro servidor para o servidor central	6,3K + 20 Bytes (Dados coletados no 1º servidor mais o agente)
Terceiro servidor para o servidor central	6,3K + 40 Bytes (Dados coletados no 2º e 3º servidores mais o agente)
Total	**31580 Bytes**

Tabela 5.2 – Bytes trafegados – Arquitetura Distribuída

A Tabela 5.3 mostra o total acumulado de bytes trafegado pela rede na arquitetura de agentes móveis mutáveis. Foi necessário o trafego total de 18882 bytes para a conclusão da simulação.

Migração	Bytes
Servidor central para primeiro servidor	5,32K
Envio do pacote especialista	941 Bytes
Retorno ao servidor central	20 Bytes
Servidor central para segundo servidor	5,32K
Envio do pacote especialista	941 Bytes
Retorno ao servidor central	20 Bytes
Segundo servidor para o terceiro servidor	6,3K
Retorno ao servidor central	20 Bytes
Total	**18882 Bytes**

Tabela 5.3 - Bytes trafegados – Arquitetura Proposta

A Tabela 5.4 mostra o total acumulado de bytes trafegados pela rede na arquitetura centralizada (cliente servidor). Foi necessário trafegar um total de 301500 bytes para a conclusão da simulação.

Migração	Bytes
Envio da query para 1º servidor	500 Bytes
Retorno dos dados	100K Bytes
Envio da query para 2º servidor	500 Bytes
Retorno dos dados	100K Bytes
Envio da query para 3º servidor	500 Bytes
Retorno dos dados	100K Bytes
Total	**301500 Bytes**

Tabela 5.4 - Bytes trafegados – Arquitetura Centralizada

Os dados mostram que a arquitetura de agentes móveis mutáveis apresenta vantagens em relação às demais arquiteturas quando se compara a quantidade de dados trafegados na rede. A Tabela 5.5 e a Figura 5.11 mostram o resultado comparativo entre as três arquiteturas:

Arquitetura	Bytes trafegados
Agentes Mutáveis	18.882 Bytes
Agentes Especialistas	31.580 Bytes
Cliente-Servidor	301.500 Bytes

Tabela 5.5 – Comparação entre as arquiteturas

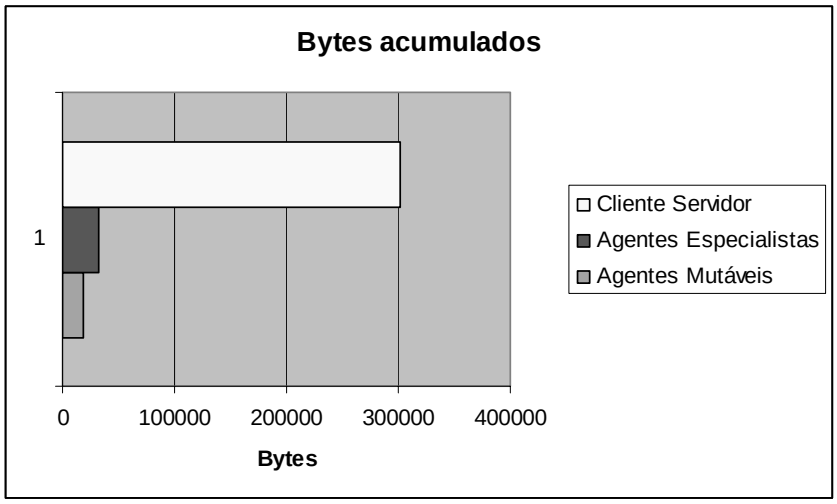

Figura 5.11 - Comparação entre arquiteturas

5.3. Conclusão do capítulo

Através do estudo de caso e das simulações foi possível comprovar alguns dos benefícios (descritos no capítulo 4) da arquitetura proposta. As comparações de volume de dados entre os testes de simulação mostram que o uso de recursos da rede pode ser bastante reduzido quando se utiliza a arquitetura proposta.

A complexidade é um ponto importante de ganho na arquitetura proposta, pois um único agente pode desempenhar todas as tarefas disponíveis no sistema.

O sistema se torna bastante flexível pelo fato de ser possível agregar novas funcionalidades com necessidade mínima de alterações na arquitetura. Em principio, é necessário apenas criar um novo pacote de funcionalidade e cadastrar no sistema.

6. Conclusões

Agentes móveis é um paradigma que vem sendo bastante pesquisado. Entre as principais vantagens dos agentes móveis, podem ser citadas as seguintes: a redução de mensagens veiculadas na rede por serem tratadas localmente; a interação assíncrona que permite um desacoplamento do nó de origem do agente, a distribuição do processamento, a atuação facilitada em problemas correlacionados, a capacidade de interagir com dispositivos móveis com conexões voláteis e a flexibilidade.

Os agentes móveis têm sido utilizados em diversas aplicações distribuídas, tais como: computação móvel, comércio eletrônico, recuperação de informações e gerenciamento de redes.

Este trabalho é focado na aplicação de gerenciamento de redes, onde o modelo por agentes móveis mutáveis é apresentado e comparado a modelos tradicionais de gerenciamento de redes. Mais especificamente, o trabalho analisa o desempenho do gerenciamento por agentes móveis mutáveis, comparando-o aos gerenciamentos centralizado e descentralizado, no que se refere à quantidade de dados trafegados.

Com a elaboração deste trabalho foi possível constatar, e comprovar, alguns pontos relacionados ao uso de agentes móveis no auxilio a gerência de redes e à arquitetura proposta.

O primeiro ponto diz respeito às vantagens de se utilizar agentes móveis mutáveis em ambientes descentralizados. Através dos experimentos apresentados foi possível constatar que o uso destes agentes pode ser uma opção vantajosa por agregar flexibilidade ao sistema e permitir ganho em controle e desempenho.

Outro ponto que foi observado através dos experimentos, é que a arquitetura proposta, comparada com as demais arquiteturas, trouxe uma substancial redução no uso da banda de rede para tarefas de gerenciamento. O montante de dados enviados pela rede foi menor e o controle dos agentes se mostrou mais simples, principalmente pelo fato de utilizar agentes genéricos e agregar funcionalidades ao decorrer do processo.

A seguir são listadas as principais contribuições desta dissertação:

- Proposta de solução para problemas de complexidade no controle encontrados nas arquiteturas descentralizadas.
- Proposta de solução para problemas de tráfego de dados encontrados nas arquiteturas de agentes móveis quando utilizados em grandes redes.
- Proposta de uma arquitetura mais flexível com possibilidade de crescimento sem a necessidade de grandes intervenções no sistema.

Diversas direções podem ser tomadas neste trabalho, a partir dos resultados apresentados:

- Novas simulações considerando outros aspectos da utilização desta arquitetura, como por exemplo: uso em ambientes reais de rede de grande porte, testes e medições de outros aspectos como latência em redes com recursos escassos e utilizando equipamentos móveis tais como celulares, PDA’s e Pages.
- Explorar as potencialidades e limitações desta arquitetura com a utilização em outras áreas como comércio eletrônico e segurança.
- A adição de inteligência aos agentes mutáveis para testes de autonomia e auto-gerenciamento.

7. Referências Bibliográficas

(ABLE, 2007). **ABLE - Agent Building and Learning Environment**. Disponível em: <www.alphaWorks.ibm.com/tech/able>. Acessado em: 14/12/2006.

(ADHICANDRA, 2005). ADHICANDRA, Iwan; PATTINSON, Ebrahim Shaghouei C.. **Using Mobile Agents to Improve Performance**. Proceedings of the 2005 ACM symposium on Applied computing 2005. Santa Fe / New Mexico, 2005.

(AGENTBUILDER, 2007). **AGENT BUILDER**. Disponível em: <www.agentbuilder.com>. Acessado em: 14/12/2006.

(AGLETS, 2007). **AGLETS**. Disponível em: <http://www.trl.ibm.com/aglets/>. Acessado em: 14/12/2006.

(AT&T, 2004). AT&T. **Point of View** - **Self-Managing Network Volume 2**, 2004.

(BELLIFEMINE, 2005). BELLIFEMINE , Fabio; CAIRE, Giovanni; TRUCCO, Tiziana; RIMASSA, Giovanni. **Jade's Programmming Guide**. 2005. Disponível em: <http://jade.tilab.com/doc/programmersguide.pdf>. Acessado em: 09/10/2006.

(BELLIFEMINE, 2003). BELLIFEMINE, F.; Caire, G.; Poggi, A.; Rimassa , G.. **Jade A Write Paper**. 2003. Disponível em: <http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf>. Acessado em: 10/10/2006.

(BILLIFEMINE II, 2005). BILLIFEMINE, Fabio ; CAIRE, Giovanni; TRUCCO, Tiziane; RIMASSA, Giovanni; MUNGENASTI, Roland. **JADE Administrator Guide**. 2005.

Disponível em: <http://jade.tilab.com/doc/administratorsguide.pdf> Acessado em: 15/09/2006.

(BIANCHINI, 2003). BIANCHINI, Calebe de Paula; ALMEIDA, Eduardo Santana de; FONTES, Diogo Sobral; ANDRADE, Rossana Maria de Castro. **Um padrão de Gerenciamento de Redes**. The Third latin American Conference on Pattern Languages of Programming. 2003. Porto de Galinhas – Pernambuco, Brazil.

(BIESZCZAD, 1998). BIESZCZAD, A.; PAGUREK, B.. **Network Management Application Oriented Taxonomy of Mobile Code**, in Proceedings of IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, 1998.

(BIESZCZAD II, 1998). BIESZCZAD, Andrzej; PAGUREK, Bernard, WHITE, Tony. **Mobile Agents for Network Management**. 1998. In Proceedings of IEEE Communications Surveys, 1998 - scs.carleton.ca.

(BIGUS, 2001). BIGUS, Joseph P.; BIGUS, Jennifer. (2nd Ed.). **Constructing Intelligent Agents Using Java.** Danver**:** Wiley**,** 2001.

(BOHORIS, 2000). BOHORIS, C.; PAVLOU, G.; CRUICKSHANK, H.. **Using mobile agents for network performance management.** Network Operations and Management Symposium, 2000. Honolulu, HI, USA**.**

(BRANDÃO , 2002). BRANDÃO, Antonio José dos Santos; MOREIRA, Edson dos Santos. **Agentes Móveis e Sistemas de Gerenciamento SNMP**. WSeg 2002. 2002. Búzios – RJ. Disponível em: <www.ppgia.pucpr.br/~maziero/pesquisa/ceseg/wseg02/07.pdf>

(BRAUN, 2005). BRAUN, Peter; ROSSAK, Wilhelm. (Ed.). **Mobile Agentes - Basic Concepts, Mobility Models, & The Tracy Toolkit.** San Francisco: Morgan Kaufmann, 2005.

(CAIRE, 2004). CAIRE, Giovanni. **Jade Tutorial for Beginners. 2003**. Disponível em: <http://jade.tilab.com/doc/JADEProgramming-Tutorial-for-beginners.pdf>. Acessado em: 12/10/2006.

(CARDANA, 2006). CARDANA , João Manuel Alexandre. **Analisador Comportamental de Rede**. 2006. Dissertação (Mestrado em Informática) – Universidade de Lisboa, Lisboa, 2006.

(CISCO, 2006). **Network Management Basics.** CISCO**:** Internetworking Technologies Handbook. EUA: 2006, Disponível em: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/nmbasics.htm. Acessado em: 25/12/2006.

(COSTA, 1999). COSTA, Tais Freira da Silva. **Avaliação Analítica do Uso de Agentes Móveis na Gerência de Redes.** Dissertação (Mestrado em Ciência da Computação) – Programa de Pós Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 1999.

(CZAP, 2005). CZAP, H.; UNLAND, R.; BRANKI, C.; TIANFIELD, H.. **Self-Organization and Autonomic Informatics (I).** v 135, Frontiers in Artificial Intelligence and Applications. Amsterdam: IOS Press, 2005.

(EID, 2005). EID, Mohamad; ARTAIL, Hassan; KAYSSI, Ayman; CHEHAB, Ali. **Trends in Mobile Agent**. Journal of Research and Practice in Information Technology, vol. 37, no. 4, 2005. p. 331 – 359.

(ELEFTHERIOU, 2000). ELEFTHERIOU, George; GALIS, Alex. **Mobile Intelligent Agents for Network Management Systems**. London Communications Symposium. Londres, 2000.

(FAHAD, 2003). FAHAD ,Tarag; YOUSEF , Sufian; STRANGE, Caroline. **A Study of the Behaviour of the Mobile Agent in the Network Management Systems.** Telecommunications Engineering Research Group – Anglia Ruskin University

(FIPA, 2005). **FIPA specifications**. Disponível em: <http://jade.tilab.com/papers/JADETutorialIEEE/JADETutorial_FIPA.pdf> Acessado em: 15/10/2005.

(FIPA, 2007). **FIPA - The Foundation for Intelligent Physical Agents.** Disponível em: www.fipa.org. Acessado em: 13/12/2006.

(FIPA-OS, 2007). **FIPA-OS**. Disponível em: <http://sourceforge.net/projects/fipa-os/>. Acessado em: 15/12/2006.

(GIMENEZ, 2004). GIMENEZ, Edson Josias Cruz. **Metodologia Pragmática para avaliação de Desempenho e Planejamento de Capacidade em Redes de Computadores.** 2004. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós Graduação de Engenharia Elétrica, Instituto Nacional de Telecomunicações, Santa Rita do Sapucaí, 2004.

(GOSSIP, 2007). **GOSSIP**. Disponível em: <https://gna.org/projects/gossip/>. Acessado em: 15/12/2006.

(GRAY, 2001). GRAY, Robert S.; KOTZ, David; PETERSON, Ronald A.; BARTON, Joyce; CHACÓN, Daria; GERKEN, Peter; HOFMANN, Martin; BRADSHAW, Jeffrey; BREEDY, Maggie; JEFFERS, Renia; SURI, Niranjan. **Mobile-Agent versus Client/Server Performance: Scalability in an information-Retrieval Task**. Proceedings of the Fifth IEEE International Conference on Mobile Agents.Atlanta, Georgia, 2001. p. 229-243.

(ISO/IEC, 1999) ISO / IEC. **7498-4 Information Processing Systems – Open Systems Interconnection – Management Framework.** Ed. Primeira, 1989.

(ITU-T, 1997). TELECOMMUNICATION STANDARDIZATION – ITU-T. **X701**: OSI management – Systems Management framework and architecture. EUA: 1997. z

(JADE, 2007). **JADE**. Disponível em: <http://jade.tilab.com/>. Acessado em: 10/06/2006.

(JANSEN, 2005) JANSEN; Wayne; KARYGIANNIS, Tom. **Mobile Agent Security**. NIST Special Publication 800-19. 1999. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-19/sp800-19.pdf >. Acessado em: 12/12/2006.

(JATLITE, 2007). **JATL - Java Agent Template**. Disponível em: <http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html>. Acessado em: 15/12/2006.

(KONSTATINOU, 2002). KONSTANTINOU, Alexander V.; FLORISSI, Danilo; YEMINI, Yechiam. **Towards Self-Configuring Networks.** DARPA Active Networks Conference and Exposition, p. 143-156, San Francisco: maio 2002.

(KONSTANTINOU, 2003). KONSTANTINOU**,** Alexander V **Towards Autonomic Networks**. Ph.D. Thesis. Columbia University New York, NY, USA. 2003. Disponível em: <http://www1.cs.columbia.edu/dcc/nestor/thesis/ >. Acessado em: 12/01/2007.

(KONSTANTINOU II, 2003). KONSTANTINOU**,** Alexander V.; YEMINI, Yechiam. **Programming Systems for Autonomy**. Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03), 2003.

(KOTSAKIS, 1995). KOTSAKIS ,E. G.,; PARDOE ,B. H.. **Modelling OSI Management Information Base With Object Oriented Analysis.** ISCOM '95 International Symposium on Communications. Taipei, Taiwan: 1995. p. 143-149.

(LANGE, 1999). LANGE , Danny B.; OSHIMA, Mitsuru. **Seven good reasons for mobile agents**. Communications of the ACM. 1999. ACM Press : New York, NY, USA. p. 88-89.

(LESSA, 1999). LESSA**,** Demian. **O Protocolo de Gerenciamento RMON.** RNP – Rede Nacional de Ensino e Pesquisa. Brasília: 1999. Disponível em: <http://www.rnp.br/newsgen/9901/rmon.html> Acesso: 25/10/2006.

(LOPES, 1999) LOPES, Rui Pedro; OLIVEIRA, José Luís. **Software Agents in Network Management.** 1999. 1st International Conference on Enterprise Information Systems – ICEIS99. Setubal, Portugal.

(MAES, 1995). MAES, Pattie. **Modeling Adaptative Autonomous Agent.** Cambridge: MIT Press, 1995

(MANOLA, 1998). MANOLA, Frank. **OMG MASIF**. 1998. Disponível em: http://www.objs.com/agility/tech-reports/9807-agent-standards.html. Acessado em: 16/12/2006.

(MONTEIRO, 2005). MONTEIRO, Maxwell Eduardo. **Gerência de Redes Vs. Inteligência Artificial perspectivas e Frentes de Trabalho.** Artigo desenvolvido em projeto de doutorado do Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, 2005.

(MONTEIRO II, 2005). MONTEIRO, Maxwell Eduardo. **Uma Plataforma Autônoma para Aprovisionamento em Redes de Telecomunicações.** Documento de

Qualificação apresentado ao Programa de Doutorado em Automação do Centro Tecnológico, Universidade Federal do Espírito Santo, 2005.

(MYSQL, 2007). **MYSQL**. Disponível em: <http://www.criarweb.com/artigos/118.php>. Acessado em: 04/04/2007.

(NEWARD,2001). NEWARD, Ted. **Understanding Class.forName().** A JavaGeeks.com White Paper. 2001.

(OMG, 2007). **OMG - The Object Management Group**. Disponível em: <www.omg.org>. Acessado em: 14/12/2006.

(PAGUREK, 2000). PAGUREK, B.; WANG Y.; and WHITE T.. **Integration of Mobile agents with SNMP: Why and How**. IEEE/IFIP Network Operations and Management Symposium, NOMS 2000, Honolulu, 2000.

(PEREIRA, 2001). PEREIRA, Mateus Casanova. **Administração e Gerência de Redes de Computadores**. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2001.

(PRAS, 1995). PRAS, Aiko. **Network Management Architectures**. Dissertação (Mestrado em tecnologia da informação) – Centre for Telematics and Information Tecnology, Netherlands, 1995.

(RAMOS, 2004). RAMOS, Andréa Silva. **Como a Inteligência Artificial está sendo utilizada no auxílio ao Gerenciamento de Redes de Computadores e Telecomunicações.** Artigo desenvolvido em projeto de doutorado do Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, 2004.

(ROY, 2005). ROY , Peter Van; GHODSI, Ali; HARIDI, Seif; STEFANI, Jean-Bernard; COUPAYE, Thierry; HEINEFELD, Alexander R.; WINTER, Ehrhard; YAP, Roland. **Self Management of Large-Scale Distributed Systems by Combining Peer-to-Peer Networks and Components**. European Research Network on Foundations: Rennes, 2005.

(RUBINSTEIN, 1998). RUBINSTEIN , Marcelo Gonçalves; DUARTE , Otto Carlos Muniz Bandeira. **Service Location for Mobile Agent Systems**. IEEE/SBT International Telecommunications Symposium (ITS'98). São Paulo, 1998.

(RUBINSTEIN, 2001). RUBINSTEIN, Marcelo Gonçalves. **Avaliação do Desempenho de Agentes Móveis no Gerenciamento de Redes.** 2001. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação de Engenharia Elétrica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2001.

(RUBINSTEIN II, 2001). RUBINSTEIN, Marcelo G.; DUARTE, Otto. Carlos. M. B.. **Implementação e Simulação de uma Aplicação de Gerenciamento Utilizando Agentes Móveis**. 19º Simpósio Brasileiro de Telecomunicações, Fortaleza, CE, Brazil, 2001.

(SHEN, 2005). SHEN, Chong; PESCH, Dirk; IRVINE, James. **A Framework for Self-Management of Hybrid Wireless Networks Using Autonomic Computing Principles**. Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual Volume. 2005.

(SILVEIRA, 2000). SILVEIRA, Klaubert Herr da. **Desafios para os Sistemas de Detecção de Intrusos (IDS).** RNP – Rede Nacional de Ensino e Pesquisa. Brasília: 2000. Disponível em: <http://www.rnp.br/newsgen/0011/ids.html#ng-2> Acesso: 25/10/2006.

(STALLINGS, 1998). STALLINGS , William. **Snmpv3 A Security Enhancement for Snmp**. IEEE Communications Surveys v. 1. Disponível em: www.comsoc.org/pubs/surveys. 1998.

(STRASSNER, 2005). STRASSNER, Jonh. **Autonomic Architecture to Support Next Generation Services.** Waterford Institute of Technology, Schaumburg / USA: 2005.

(SUN, 2007). **SUN**. Disponível em: <www.sun.com/java>. Acessado em: 01/10/2006.

(VELLOSO, 2002). VELLOSO, Pedro Braconnot; RUBINSTEINY, Marcelo Gonçalves; DUARTE, Otto Carlos M. B. **Uma Ferramenta de Gerenciamento de Redes Baseada em Agentes Móveis**. 2002. Disponível em: <http://www.gta.ufrj.br/ftp/gta/TechReports/VRD02.ps.gz>. Acessado em: 12/11/2006.

(VOYAGER, 2007). **VOYAGER.** Disponível em: <http://www.recursionsw.com/Products/voyager.html?var1=sc2>. Acessado em: 14/12/2006.

(WANG, 2003). WANG, A I; SORENSEN, C-F.. **A mobile agent architecture for heterogeneous devices**. IASTED International Conference on Wireless and Optical Communications; Banff; Canadá, 2003. p. 584-590.

(WIKIPEDIA, 2003). **WIKIPEDIA**. Disponível em: <http://en.wikipedia.org/wiki/Mobile_agents >. Acessado em: 04/04/2007.

(ZEUS, 2007). **ZEUS.** Disponível em: <http://labs.bt.com/projects/agents/zeus/>. Acessado em: 14/12/2006.

Anexo A

A. A plataforma JADE (Java Agent Development Framework)

A.1. Introdução

JADE foi a plataforma escolhida para o desenvolvimento deste trabalho, por ser um projeto de código aberto e utilizar a plataforma Java. Este anexo é baseado nos manuais e documentos da plataforma disponíveis no site oficial: http://jade.tilab.com/. Os documentos utilizados foram: Jade *A Write Paper* (BELLIFEMINE, 2003), *JADE Tutorial for Beginners* (CAIRE, 2004), J*ade's Programmming Guide* (BELLIFEMINE, 2005), e Jade *Administrator Guide* (BELLIFEMINE II, 2005).

JADE é um *middleware* para desenvolvimento e execução de aplicações *peer-to-peer* baseado no paradigma de agentes, que pode facilmente trabalhar e interoperar em ambientes de redes tradicionais ou *wireless*. Os dois maiores aspectos do modelo conceitual são: topologias de sistemas distribuídos com redes *peer-to-peer* e arquitetura de componentes com paradigma de agentes. A topologia de rede afeta o modo pelo qual vários componentes são conectados enquanto especifica o que supostamente um componente espera do outro.

A.2. Modelo Peer-to-peer

Em um modelo *peer-to-peer* não existe distinção entre as funções de cada ponto. Ao contrário de um modelo cliente-servidor, onde os papéis são bem definidos, cada par é capaz de misturar iniciativa e capacidade, ou seja, cada um pode iniciar a comunicação, ser o sujeito ou o objeto de uma requisição, ser pró-ativo. A aplicação lógica não mais

precisa estar concentrada em um servidor, mas distribuída entre todos os pontos da rede. Cada par é capaz de descobrir cada um, podendo entrar, unir-se e liberar-se a qualquer momento. O sistema é totalmente distribuído, assim como o valor do serviço é distribuído através da rede, permitindo que novos modelos de negócio possam ser habilitados.

Uma importante característica deste modelo é que, ao contrário de um modelo cliente servidor, onde o cliente deve saber onde está o servidor mas os servidores não precisam saber onde estão os seus clientes, a localização dos clientes e servidores é totalmente arbitrária e o serviço deve prover formas de entrada, união e liberação de pares a qualquer momento, assim como mecanismos de busca e localização de outros pares. A Figura A1 mostra alguns modelos.

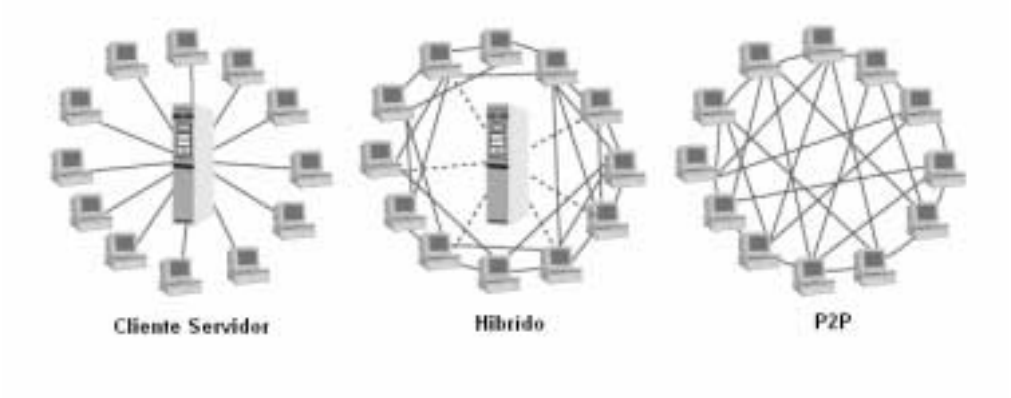

Figura A1 – Modelos

No modelo *peer-to-peer* (P2P) os pontos são totalmente autônomos e descentralizados. A falta de um nó de referência causa dificuldades em manter uma coerência na rede e consequentemente problemas na localização de pares. A complexidade tende a crescer exponencialmente com o número de nós.

A.3. O paradigma de agentes

O paradigma de agentes aplica conceitos de inteligência artificial e teorias de comunicação na tecnologia de objetos distribuídos. O paradigma é baseado em uma abstração de agentes, ou seja, um componente de software que é autônomo, pró-ativo e social, conforme já comentados anteriormente.

O paradigma de agentes é intrinsecamente *peer-to-peer*. Cada agente é um ponto que, potencialmente, necessita iniciar uma comunicação com outro agente, assim como é capaz de fornecer serviços para outros agentes. O papel da comunicação é muito importante em um sistema baseado em agentes, e este sistema engloba três características importantes:

Agentes são entidades ativas que podem dizer "não", e eles são facilmente conectáveis.

Estas propriedades são a base para a escolha de mensagens baseadas em comunicação assíncrona em vez de RPC (*Remote Procedure Call*) chamada de procedimento remoto: um agente que quer se comunicar deve apenas enviar uma mensagem a um certo destinatário. Este tipo de comunicação, na realidade, permite que o receptor selecione quais mensagens ele vai atender e quais irão descartar, ou quais mensagens serão tratadas primeiro e quais terão que esperar para serem atendidas. Isto também permite que o agente emissor não fique bloqueado esperando uma resposta do agente receptor, removendo, portanto, qualquer dependência temporal entre as partes. Desta forma, o agente receptor pode não estar disponível ou até mesmo não existir ainda no momento da comunicação.

As ações e comunicações executadas pelos agentes são consideradas como sendo apenas um tipo de ação.

Fazer com que a comunicação esteja no mesmo nível das ações permite a um agente, por exemplo, 'raciocinar' sobre um plano que inclui ambas: ações físicas (ex: virar para a esquerda) e ações de comunicação (ex: pedir para fechar uma porta).

Para fazer com que a comunicação seja planejável, efeitos e pré-condições de cada possível comunicação devem estar muito bem definidos.

Comunicações transportam um significado semântico.
Quando um agente é um objeto de uma ação comunicativa (ex: quando recebe uma mensagem), ele deve ser capaz de entender o significado da ação e porque esta ação deve ser executada. Isto leva a uma necessidade de uma semântica universal, ou seja, a uma padronização.

A.4. FIPA – The Foundation for Intelligent Physical Agents

Em 1996 a Tilab (oficialmente CSELT) promoveu a criação da FIPA, uma associação sem fins lucrativos de companhias e organizações, compartilhando um mesmo esforço e objetivo de padronizar as especificações para a tecnologia de agentes.

Em 1997 foi lançado o primeiro conjunto de especificações e, mais tarde, em 2002 a FIPA lançou o seu padrão FIPA2000. Desde junho de 2005 a FIPA foi incorporada ao IEEE como sendo o décimo primeiro comitê para promover estudos da tecnologia baseada em agentes e a interoperabilidade de seus padrões com outras tecnologias.

O padrão FIPA é focado em interoperabilidade. Como conseqüência, focado no comportamento externo dos componentes do sistema, deixando em aberto os detalhes internos de implementação e a arquitetura. A Figura A2 mostra os serviços oferecidos pela plataforma.

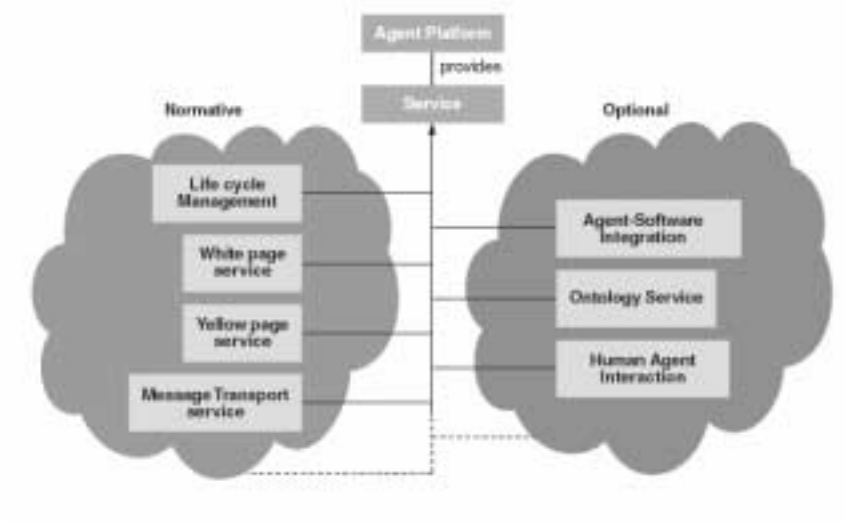

Figura A2 – Padrão FIPA – Serviços fornecidos pela plataforma

De fato, a arquitetura interna do JADE é a única que está completamente em conformidade com os padrões da FIPA. Estes padrões adotam totalmente o paradigma de agentes e, em particular, ele define um modelo de plataforma de agentes além do conjunto de serviços que devem ser disponibilizados. O conjunto destes serviços e destas interfaces padronizadas, permitem que uma sociedade de agentes exista, opere e se gerencie. A Figura A3 mostra o modelo de comunicação padronizado pela FIPA.

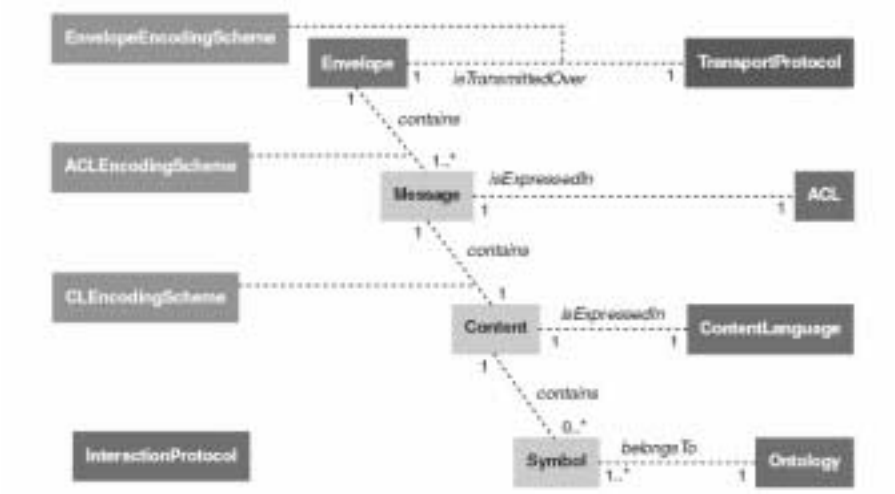

Figura A3 – Componentes do modelo de comunicação padronizado pela FIPA

A.5. Middleware

O termo *middleware* significa “software que permite a interconexão entre duas aplicações diferentes e separadas”. No escopo do JADE seriam todas as bibliotecas de alto nível que permite facilmente, e de forma mais efetiva, o desenvolvimento de aplicações, fornecendo serviços genéricos, úteis, não somente para uma única mas para uma variedade de aplicações como: acesso a dados, controle de recursos, entre outros. Os serviços são fornecidos pelos sistemas operacionais, mas a idéia por detrás do *middleware* é fornecer API’s independentes da plataforma, agregando facilidades nativas dentro de simples blocos reutilizáveis. O *middleware* baseado nesta abordagem permite uma redução considerável no tempo de desenvolvimento. A Figura A4 mostra a função de um *middleware*.

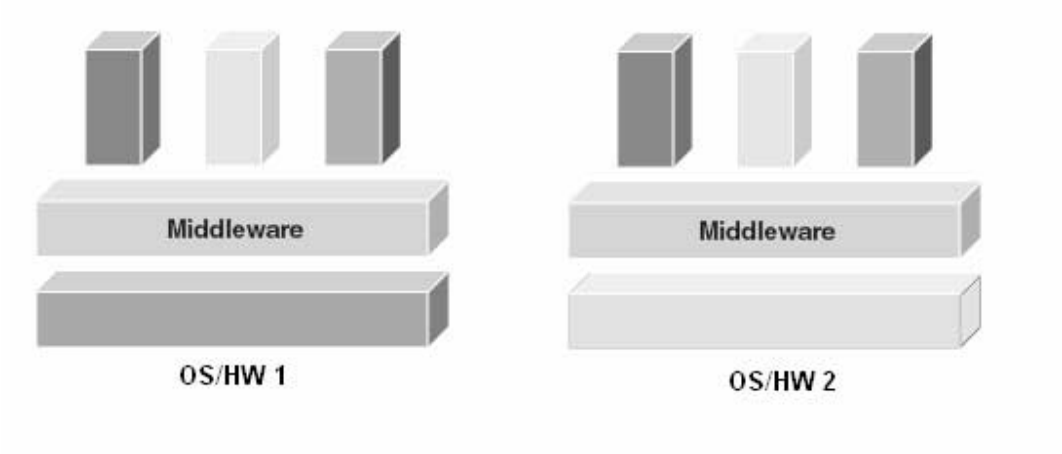

Figura A4 – Função de um *middleware*

A.6. A plataforma

Jade é um *middleware* desenvolvido pela Tilab para o desenvolvimento de aplicações distribuídas multi-agentes baseada na arquitetura de comunicação *peer-to-peer*. Ambos, a inteligência, a iniciativa, a informação, o controle e os recursos podem ser totalmente distribuídos em terminais móveis ou em computadores em redes normais. O

ambiente pode evoluir dinamicamente com os pontos, que em JADE são chamados de agentes, que podem aparecer e desaparecer de acordo com a necessidade e requerimentos do ambiente da aplicação. A Figura A5 mostra alguns dos ambientes onde a plataforma JADE pode funcionar. A comunicação entre os pontos, não importando se a rede é *wireless* ou não, são sempre simétricos, ou seja, cada ponto pode ser quem inicia ou quem responde a uma ação de comunicação. JADE é baseado em Java e é guiado pelos seguintes princípios:

Interoperabilidade: JADE está em conformidade com os padrões da FIPA, e, como conseqüência, os agentes JADE podem interoperar com outros agentes, contando que estes estejam em conformidade com o padrão.

Uniformidade e portabilidade: JADE fornece um conjunto homogêneo de API's que são independentes da rede e da versão de Java. O JADE pode operar sobre o J2EE, J2SE e J2ME.

Fácil de usar: A complexidade do *middleware* é escondida por detrás de um simples e intuitivo conjunto de API's de fácil utilização.

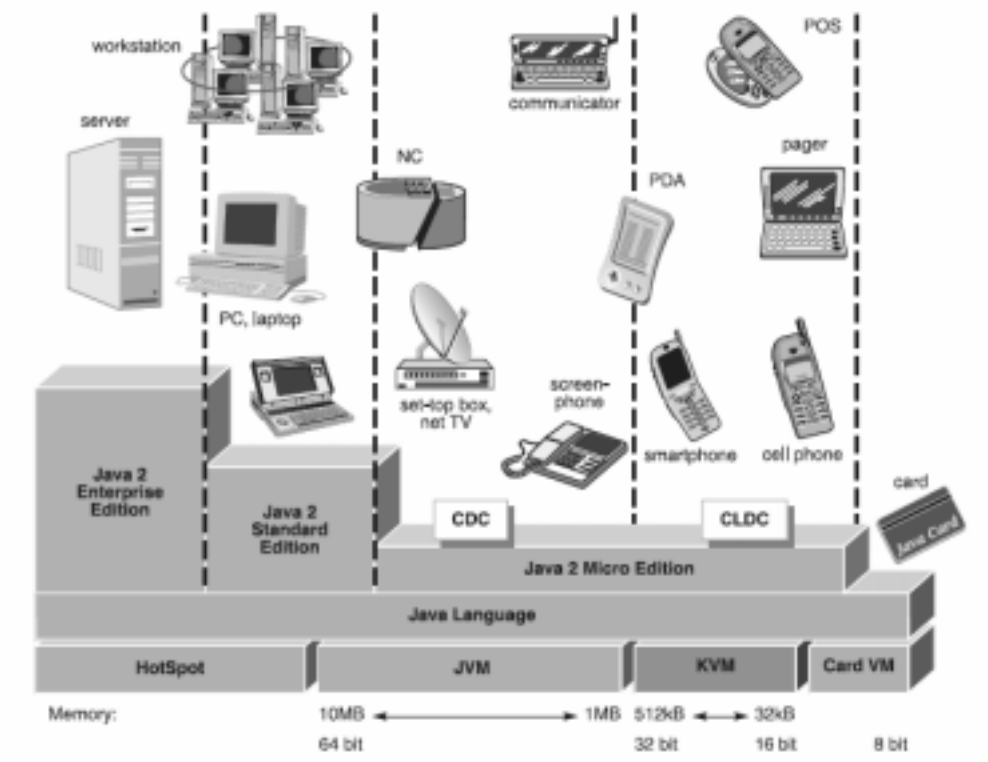

Figura A5 – Plataforma Java

A.7. Modelo da arquitetura

JADE inclui todas as bibliotecas necessárias para o desenvolvimento de agentes e um ambiente de execução que fornece os serviços básicos. O ambiente de execução deve ser iniciado antes da execução dos agentes. Cada instância do ambiente de execução é chamado de *container*, o conjunto de *containers* é chamado de plataforma que fornece uma camada homogênia que esconde os agentes da diversidade e complexidade das camadas inferiores (hardware, sistemas operacionais, etc...).

JADE é bastante versátil, e, é capaz de se integrar a sistemas com recursos limitados, como, por exemplo, telefones móveis e também com sistemas mais complexos como plataformas J2EE e .NET. A Figura A6 mostra de forma superficial a arquitetura JADE.

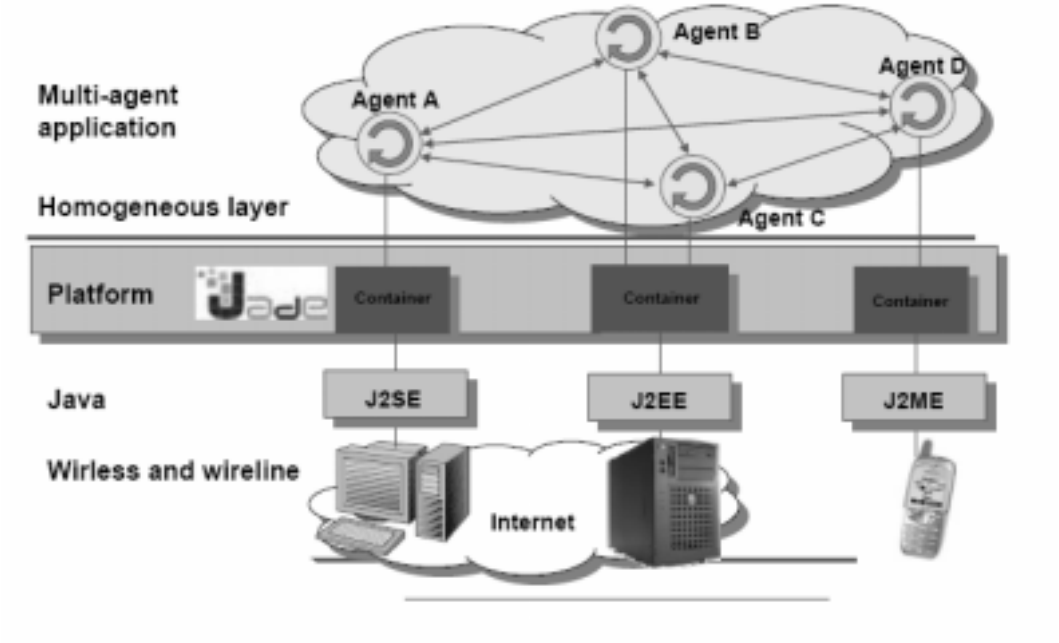

Figura A6 – Arquitetura

O modelo de arquitetura do JADE tem as seguintes características:

- Uma aplicação JADE é composta por uma coleção de componentes ativos chamados de agentes.
- Cada agente tem um nome único.
- Um agente é um ponto que pode comunicar-se com outros agentes de forma bidirecional.
- Um agente vive em um container.

A.8. Modelo de comunicação

A comunicação é baseada na passagem de mensagens assíncronas. Cada agente tem um conjunto de caixas de correio onde as mensagens para estes agentes são inseridas. Quando uma mensagem é colocada na caixa de correio o agente é notificado. Porém, é o agente quem decide quando ele vai ler e responder a determinada mensagem. A Figura A7 mostra o modelo de comunicação JADE.

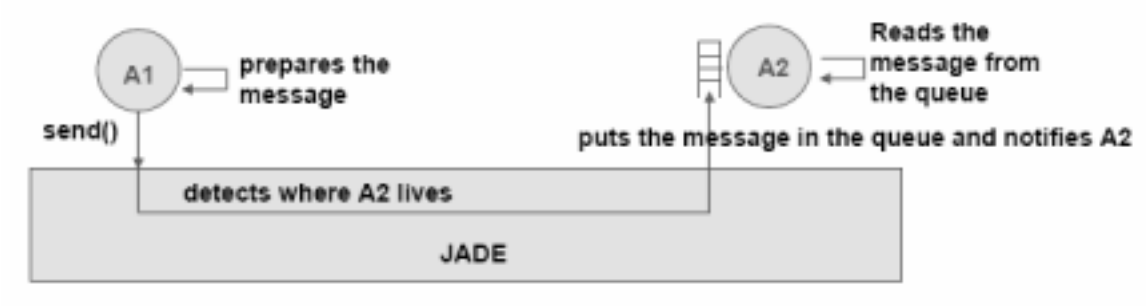

Figura A7- Modelo de comunicação

A.9. Modelo funcional

Do ponto de vista funcional, JADE fornece os serviços básicos necessários para aplicações distribuídas em ambientes fixos ou móveis. JADE permite que cada agente possa, dinamicamente, descobrir outros agentes e comunicar com eles, de acordo com o paradigma *peer-to-peer*. Do ponto de vista da aplicação, cada agente é identificado por um nome e a ele é fornecido uma série de serviços. Ele pode registrar e modificar seus serviços e/ou procurar por agentes para buscar serviços. Ele pode controlar seu ciclo de vida e comunicar com outros pontos.

Os agentes se comunicam trocando mensagens assíncronas, modelo de comunicação mais aceito para comunicação entre entidades heterogênias que não conhecem nada um do outro. Para se comunicar, o agente apenas envia uma mensagem para o destinatário. Os agentes são identificados pelo nome (não existe a necessidade de uma referência ao objeto de destino) e, como conseqüência, não existe dependência temporal entre os agentes em comunicação. O agente transmissor e o receptor não precisam estar disponíveis ao mesmo tempo. O receptor pode mesmo não existir (ainda) ou pode não ser conhecido diretamente pelo transmissor que pode apenas especificar uma propriedade (Ex: agentes interessados em carros) como destino.

Apesar deste tipo de comunicação, a segurança é preservada, uma vez que para aplicações que exigem segurança, JADE fornece um mecanismo para autenticar e verificar direitos assinalados aos agentes. Quando necessário, uma aplicação pode verificar a identidade do transmissor e prevenir ações não permitidas.

Todas as mensagens enviadas por um agente devem se transportadas através de um envelope contendo apenas as informações necessárias à camada de transporte. Isso permite, entre outras coisas, criptografar a mensagem separadamente do envelope. A estrutura da mensagem está em conformidade com a linguagem ACL,definida pela FIPA, e inclui campos, como variáveis, indicando o contexto da mensagem a que se refere e um tempo de expiração que pode ser esperado antes da resposta da mensagem, a fim de suportar complexas interações e múltiplas conversações em paralelo.

Para suportar a implementação de conversações complexas, o JADE fornece um conjunto de *skeletons* para padrões de interações típicas na execução de tarefas específicas, assim como negociações, leilões e delegação de tarefas. Por usar estes *skeletons* (implementados como classes Java abstratas) programadores podem livrar-se da carga de ter que lidar com sincronização, tempos de expiração, condições de erro e, em geral, sobre todos os aspectos não relacionados com a lógica da aplicação.
Para facilitar a criação e a manipulação do conteúdo das mensagens, JADE fornece suporte para conversões automáticas entre formatos na troca de conteúdos, incluindo XML e RDF. Este suporte é integrado com algumas ferramentas de criação de ontologias tais como *Protégé*, permitindo aos programadores criarem graficamente suas ontologias.

JADE não é transparente no que diz respeito ao suporte a sistemas com inferência. Se uma inferência é necessária a uma aplicação específica, ele permite que programadores reusem seus sistemas preferidos. Ele já foi testado e integrado com JESS e Prolog.

Para aumentar a escalabilidade e para se adaptar a sistemas com recursos limitados, JADE fornece a possibilidade de execução de múltiplas tarefas paralelas em uma única *thread*. Muitas tarefas elementares, como comunicação, podem ser combinadas para formarem estruturas de tarefas mais complexas como máquinas de estado finito concorrentes.

Em J2SE e ambientes Java, JADE suporta código móvel e execução de estados. Um agente pode parar a execução em um *host*, migrar a um *host* diferente (sem a necessidade do código deste agente já existir nesta máquina) e reiniciar a sua execução do ponto onde foi interrompido. Esta funcionalidade permite, por exemplo, a distribuição de carga computacional em tempo de execução, movendo agentes de máquinas com mais carga para máquinas com menos carga, sem nenhum impacto na aplicação.

A plataforma também inclui serviços de nomes (assegurar que um agente possua apenas um nome) e serviço de páginas amarelas que podem ser distribuídos entre múltiplos *hosts*. Grafos podem ser criados para definir a estrutura de domínios do serviço de agentes.

Outra característica muito importante da plataforma é o conjunto de ferramentas gráficas que permitem depurar, monitorar e gerenciar fases do ciclo de vida da aplicação. Isto significa que existe a possibilidade de controle remoto dos agentes mesmo que estes já tiverem sido instalados e executados. Conversações entre agentes podem ser emuladas, troca de mensagens podem ser capturadas, tarefas podem ser monitoradas e o ciclo de vida do agente pode ser controlado.

A.10. Jade em um ambiente móvel

O ambiente de execução do JADE pode ser executado em uma grande classe de equipamentos desde servidores até telefones celulares. JADE pode ser configurado para se adaptar às características do ambiente no qual irá ser executado, como, por exemplo, em condições com escassez de memória, limitações de processamento de dispositivos móveis, intermitência e variabilidade de endereçamento IP. A arquitetura JADE é modular e ativando certos módulos ao invés de outros, é possível alcançar os objetivos de conectividade, memória e capacidade de processamento.

Um módulo chamado LEAP permite otimizar todos os mecanismos de comunicação quando trabalha com dispositivos com recursos limitados conectados através de redes *wireless*. Ativando este módulo, o JADE divide o *container* em duas partes o *front-end* e o *back-end*. Ao *back-end* é designado parte das funcionalidades do *container*, deixando o *front-end* mais leve em termos de processamento e memória. O *front-end* é capaz de detectar a perda de conexão com o *back-end* e restabelecer assim que possível. Ambos os lados possuem o mecanismo de *store-and-forward*, ou seja, a capacidade de armazenar mensagens que porventura não possam ser enviadas imediatamente para serem enviadas assim que a conexão for restabelecida. A Tabela A1 resume algumas das características da plataforma JADE.

A.11. Características

JADE	
Características técnicas e funcionais	Distribuída, modular com comunicação ponto-a-ponto. Conformidade com os padrões FIPA. Gerenciamento do ciclo de vida do agente. Serviços de páginas brancas e páginas amarelas com a oportunidade de criação. Grafos em tempo de execução. Ferramentas gráficas para debugar, gerenciar e monitorar. Suporte para código de agente, execução de migração. Suporte para protocolos de comunicação complexos. Suporte para mensagens com conteúdo e gerenciamento, incluindo XML e RDF. Suporte para integração com páginas JSP através de taglibs. Suporte camada de segurança. Protocolos de transporte selecionáveis em tempo de execução: RMI JICP (JADE protocolo proprietário), HTTP e IIOP.
Disponibilidade	Código aberto, Licença LGPL
Ambiente de rede	Já testado com Bluetooth, GPRS, W-LAN e a Internet.
Terminais	Todos os terminais que suportam Java MIDP1.0 já testado no Nokia 3650, Motorola Accompli008, Siemens, Compaq iPaq, Psion5MX, HP Joranda 560.
Linguagem	Java: J2EE, J2SE, J2ME CLDC/MIDP1.0

Tabela A1 – Características JADE

A.12. Plataforma

O Jade define três agentes necessários para administração da plataforma:

***Agent Management System* (AMS)**
***Agent Communication Channel* (ACC)**
***Directory Facilitator*(DF)**

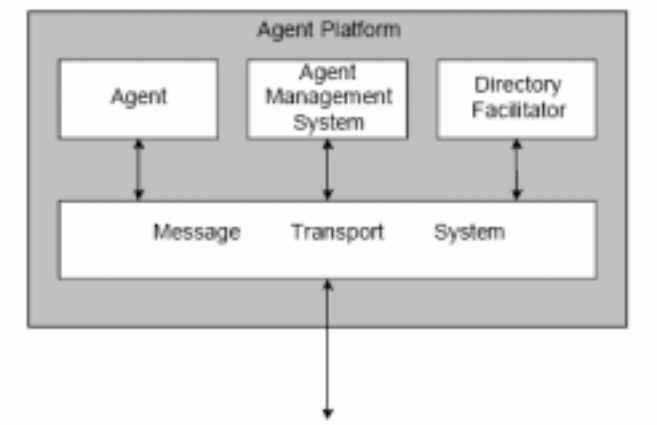

Figura A8 – Referência da arquitetura da plataforma de agentes FIPA

A Figura A8 ilustra a arquitetura dos agentes FIPA. O *Agent Management System* (AMS) é o agente que fornece o controle supervisório sobre o acesso e o uso da plataforma. Somente um AMS pode existir em uma única plataforma. O AMS fornece serviço de *white-page* e de ciclo de vida, mantendo um diretório de identificadores de agentes (AID) e o estado do agente. Todo agente deve se registrar no AMS para ganhar um ID valido.

O *Directory Facilitator* (DF) é o agente que fornece um serviço padrão de *yellow-page* para a plataforma.

O *Message Transport System* (MTS) também conhecido como *Agent Communication Channel* (ACC) é um componente de software que controla todas as trocas de mensagens da plataforma, incluindo mensagens de/para plataformas remotas.

O JADE está em total conformidade com este modelo de arquitetura e quando o JADE é executado imediatamente o AMS e o DF são criados e o módulo ACC é configurado para permitir a comunicação. A plataforma de agentes pode ser dividida entre vários *hosts*, mas somente uma aplicação Java, conseqüentemente uma JVM, é executada em cada *host*. Cada JVM é um *container* básico de agentes que fornece um completo ambiente de execução para a ativação do agente e permite que vários agentes executem concorrentemente em um mesmo *host*. O *container* principal ou f*ront-end* é um container de agentes onde os agentes DF e AMS vivem e onde o RMI *registry*, utilizado internamente pelo JADE, é criado. Os outros containeres de agentes conectam no container principal e fornece um completo ambiente de execução para a execução de qualquer conjunto de agentes JADE. A Figura A9 mostra a arquitetura de agentes distribuídos.

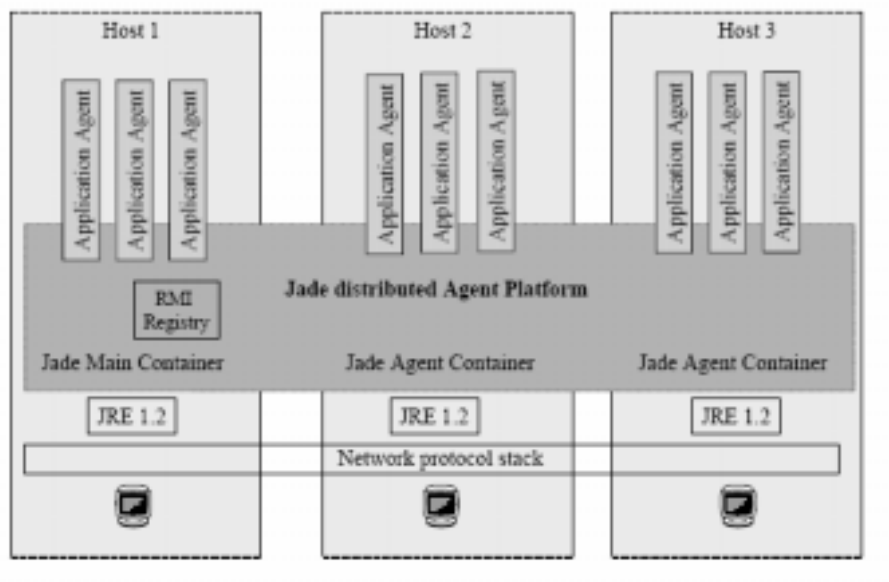

Figura A9 – Plataforma de agentes distribuídas sobre vários containeres

A.13. Programação de agentes utilizando JADE

A.13.1. Ontologia de gerenciamento de agentes FIPA

De acordo com as especificações da FIPA, os agentes DF e AMS comunicam-se utilizando a linguagem de conteúdo FIPA SL0, a ontologia *fipa-agent-management* e o protocolo de interação *fipa-request*.

O JADE está em conformidade com estas especificações através da implementação dos componentes:

O FIPA SL0 é implementado pela classe: *jade.content.lang.sl.SLCodec*. Automaticamente, a capacidade de se utilizar esta linguagem pode ser adicionada a qualquer um agente adicionando o método:

getContentManager().registerLanguage(SLCodec(0));

Os conceitos de ontologia são implementados pelas classes constantes no pacote: **jade.domain.FIPAAgentManagement.**

A classe FIPAManagementOntology define o vocabulário com todos os símbolos da ontologia. A capacidade automática de utilizar esta ontologia pode ser adicionada a um agente usando o seguinte código:

getContentManager().registerOntology(FIPAManagementOntology.getInstance());

Finalmente, o protocolo de interação fipa-request é implementado como um behaviours *ready-to-use* no pacote: **jade.proto**

Toda classe implementando o conceito de ontologia de gerenciamento de agentes é uma simples coleção de atributos, com métodos públicos para ler e escrever nestes atributos. A convenção utilizada para um atributo chamado *attrname* e tipo *attrtype* é:

1 – Se o atributo não é multivalorado então ele pode ser lido e escrito através dos métodos: *getAttrName* e *setAttrName* sobrescrevendo qualquer valor anterior.

2 – Se o atributo é uma seqüência de valores (multivalorado), então existe um método para inserir novos valores: *addAttrName*(attrType e), um para remover todos os valores: *clearAllAttrName* e a leitura é feita por um método que retornar um interador *Iterator getAllAttrName*() permitindo a navegação pela lista de atributos.

O pacote *jade.content.onto.basic* possui uma coleção de classes que são comumente parte de toda ontologia, assim como: *Action*, *TruePreposision*, *Result*, etc... A ontologia básica pode ser unida a qualquer ontologia definida pelo usuário.

- **API de acesso aos serviços DF e AMS**

As características do JADE, descritas até agora permitem uma total interação entre o sistema de agentes FIPA e os agentes definidos pelo usuário, simplesmente enviando e recebendo mensagens como definido no padrão. Contudo, como estas interações foram completamente padronizadas e porque elas são extremamente comuns, a classe *Agent*, permite o acesso a estas funcionalidades através de uma interface simplificada.
Dois métodos foram implementados pela classe Agent para buscar o AID (identificador) do DF e AMS: *getDefaultDF*() e getAMS().

- **Serviço DF**

O serviço DF (*Directory Facilitator*) fornece o serviço de páginas amarelas, ou seja, onde um agente pode procurar outros agentes para prover os serviços que são necessários para que ele possa atingir os seus objetivos.

O pacote *jade.domain.DFService* implementa um conjunto de métodos estáticos para comunicar com o serviço DF padrão (agente de páginas amarelas). Ele inclui métodos para requisitar ações de registro (*register*), desregistro (*deregister*), modificar (*modify*) e procura (*search*) no DF. Cada um destes métodos possui uma versão com todos os

parâmetros necessários. Alguns subconjuntos destes parâmetros podem ser omitidos e, neste caso são configurados com valores default.

Estes métodos bloqueiam todas as atividades do agente até que a opção é executada com sucesso ou até uma exceção *jade.domain.FIPAException* for lançada, ou seja, até que a conversação seja terminada. Em alguns casos, porém, pode ser mais conveniente executar estes métodos de uma forma que não bloqueie a execução. Nestes casos, as classes *jade.proto.AchieveREInitiator* ou *jade.proto.SubscribeInitiator* devem ser usadas em conjunção com os métodos *CreateRequestMessage*(), *CreateSubscriptionMessage*(), *decodeDone*() e *decodeNotification*(), que facilitam a preparação e a decodificação das mensagens a serem enviadas e recebidas do DF. A parte de código a seguir exemplifica um caso de um agente se registrando no DF:

```
DFAgentDescription template = //preenche o template
AID df = getDefaultDF();
ACLMessage subs = DFService.createSubscriptionMessage(this, df, template, null))
Behaviour b = new SubscriptionInitiator(this, subs) {
protected void handleInform(ACLMessage inform) {
try {
DFAgentDescription[] dfds =
DFService.decodeNotification(inform.getContent());
// ...
}
catch (FIPAException fe) {
fe.printStackTrace();
}
}
};
addBehaviour(b);
```

A.13.2. O serviço AMS

O AMS (*Agent Management System*) fornece o serviço de nomes (*naming*) o qual assegura que cada agente tenha um nome único, além de representar a autoridade na plataforma (É possível criar e destruir agentes em *containers* remotos através do AMS).
A classe *AMSService* forma uma dupla com a classe *DFService*, acessando serviços fornecidos pelo agente AMS padrão FIPA, sendo a interface totalmente correspondente a interface do *DFService*.

O JADE chama automaticamente os métodos *register* e *deregister* com o AMS *default* antes da chamada do método *setup*() e logo após o retorno do método takeDown() , ou seja, não é necessário em uma programação normal chamar estes métodos.
Contudo, em certas circunstâncias, o programador poder precisar chamar estes métodos, por exemplo quando o agente deseja se registrar em um AMS de uma plataforma remota, ou quando o agente deseja modificar sua descrição adicionando um endereço privado.

A.13.3. A classe *Agent*

A classe *Agent* é a base para o desenvolvimento de agentes. Porém, sob o ponto de vista dos programadores um agente JADE é uma simples instância de uma classe Java definida pelo programador e herdada da classe *Agent*. Isto implica na herança de características básicas de interação com a plataforma de agentes (registro, configuração, gerenciamento remoto, etc...) e um conjunto básico de métodos que podem ser chamados para implementar um comportamento (*behavior*) customizado.

O modelo computacional de um agente é multitarefa, onde tarefas (ou *behaviors*) são executadas concorrentemente. Cada serviço / funcionalidade fornecido por um agente deve ser implementado como um ou mais *behavior*. Um escalonador implementado na

classe básica *Agents*, escondida do programador, internamente gerencia o escalonamento dos agentes. A Figura A10 ilustra o ciclo de vida de um agente JADE.

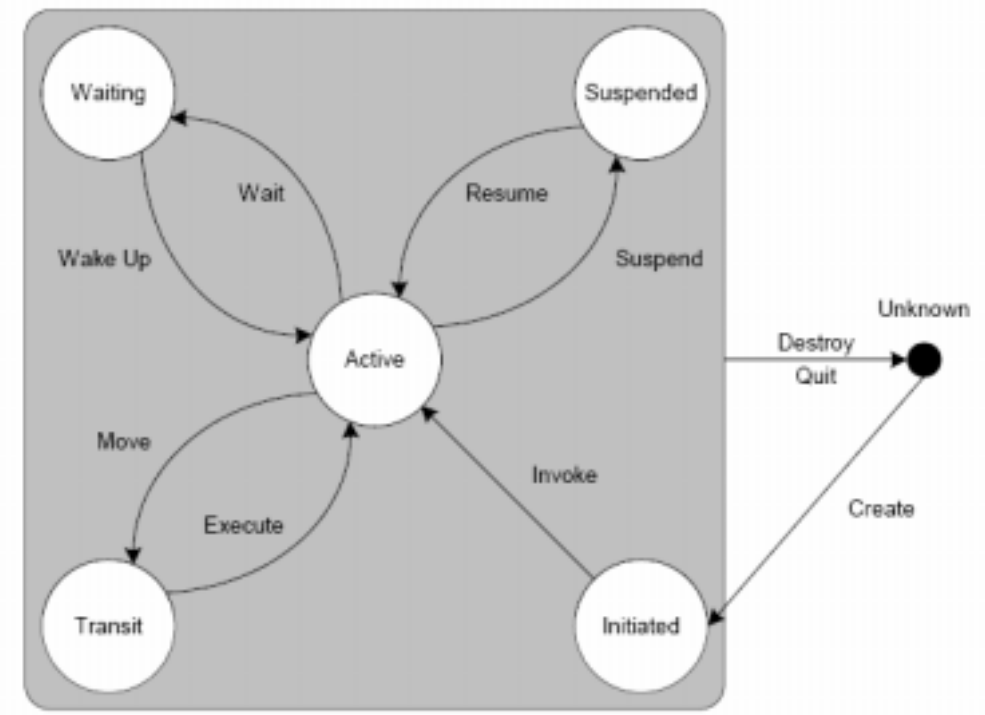

Figura A10 – Ciclo de vida de um agente

Um agente JADE pode estar em um dos vários estados, de acordo com o ciclo de vida definido pela FIPA , conforme mostrados na Figura A10. Os estados podem ser:

INITIATED: O objeto agente é construído, mas ainda não foi registrado no AMS, não tem um nome, endereço e não pode se comunicar com outros agentes.

ACTIVE: O objeto agente está registrado no AMS, tem um nome, um endereço e pode acessar todos os recursos disponíveis.

SUSPENDED: O objeto agente está parado. A *thread* é suspensa e nenhuma tarefa (*behavior*) está sendo executada.

WAITING: O objeto agente está bloqueado, aguardando algo. A *thread* está em estado de espera e pode ser reativada a qualquer momento quando certas condições são atingidas (tipicamente a chegada de mensagens).

DELETED: O agente é definitivamente morto. A *thread* interna é terminada e o agente não está mais registrado no AMS.

TRANSIT: O agente móvel entra neste estado quando está transitando para outra localidade. O sistema continua armazenando as mensagens e as envia para a nova localização.

A classe *Agent* fornece métodos públicos para executar a transição entre estes estados.

- **Iniciando a execução de um agente**

A plataforma JADE controla o nascimento de um agente seguindo os seguintes passos:

1. O construtor do agente é executado;
2. O agente recebe um identificador;
3. O agente é registrado no AMS;
4. O agente é colocado no estado: ACTIVE;
5. O método *setup*() é executado.

De acordo com as especificações da FIPA, um identificador de um agente possui os seguintes atributos:

Nome único global: Por padrão, o JADE compõe o nome através de concatenação do nome local com o símbolo @ mais o identificador do agente na plataforma.

Uma coleção de endereços de agentes: Cada agente herda os endereços de transporte de sua plataforma.

Uma coleção de *resolvers*: Por exemplo, o serviço de páginas brancas onde o agente foi registrado.

O método *setup*() é o ponto onde as tarefas do agente iniciam. O programador deve implementar o método *setup*() para iniciar o agente. Quando o *setup*() é executado, o agente já foi registrado no AMS e seu estado é ACTIVE. O programador deve utilizar o seguinte processo de inicialização:

- (opcional) Se necessário, modificar os dados registrados no AMS.
- (opcional) Configurar a descrição do agente e seus serviços. Se necessário, registrar o agente em outros domínios.
- (necessário) Adicionar tarefas na fila de tarefas prontas usando o método *addBehavior*(), os *Behaviours* são escalonados assim que o método setup() é terminado.

O método *setup*() deve adicionar, no mínimo, uma tarefa para o agente. Depois da execução do *setup*() o JADE executa a primeira tarefa da fila automaticamente e percorre as outras tarefas usando *round-robin* ou escalonador não-preemptivo.
Os métodos *addBehaviour* e *removeBehaviour*() podem ser utilizados para gerenciar a fila de tarefas.

- **Parando a execução de um agente**

Qualquer *behaviour* pode chamar o método *agent.doDelete()* para finalizar a execução do agente. O método *agent.TakeDown()* pode ser sobrecarregado pelo programador para implementar a limpeza necessária. Este método é chamado quando o agente está indo para o estado de DELETED. Quando este método é chamado, o agente ainda está registrado no AMS e pode enviar mensagens para outros agentes. Após a execução do método *takeDown*() o agente é desregistrado e a *thread* é finalizada.

A.14. Comunicação entre agentes

A classe *agents* também fornece uma coleção de métodos para a comunicação entre agentes. De acordo com as especificações da FIPA, os agentes devem se comunicar de forma assíncrona onde objetos da classe *ACLMessage* são trocados. Alguns dos protocolos definidos pela FIPA também estão disponíveis nos *behaviors* “prontos para usar”, no pacote *jade.proto*.

O método *Agent*.*send*() permite enviar uma *ACLMessage*. O atributo *receiver*, armazena a lista dos IDS dos agentes que irão receber a mensagem. O envio remoto é transparente para o programador, uma vez que a plataforma gerencia o mecanismo de endereçamento.

A.14.1.1. Acessando a fila de mensagens privadas

A plataforma coloca todas as mensagens endereçadas a um especifico agente em sua fila privada de mensagens. Por padrão, o tamanho desta fila é ilimitado. No caso de recursos limitados, este comportamento pode ser customizado através do método: *SetQueueSize*(). Vários modos de acesso foram implementados para a busca de mensagens na fila privada:
A fila de mensagem pode ser acessada de forma bloqueada ou não-bloqueada. O modo bloqueado deve ser utilizado com cuidado, porque ele causa a suspensão de todas as tarefas do agente. Ambos os métodos podem passar parâmetros.
O método bloqueado deve ter um parâmetro de tempo de expiração (*timeout*). Este parâmetro especifica o máximo tempo que o agente pode ficar esperando a chegada de uma mensagem.

A.15. Agentes com interfaces gráficas

Uma aplicação utilizando Multi-Agentes pode necessitar de interação com o usuário, a implementação Java de interfaces utilizado no JADE é o *Swing*.

A.16. Linguagem de comunicação de agentes (ACL)

A classe *ACLMessage* representa as mensagens que podem ser trocadas entre os agentes. Ela possui um conjunto de atributos definidos pela FIPA.

Um agente que deseje enviar uma mensagem, deve criar um objeto da classe *ACLMessage*, preencher seus atributos e chamar o método *Agent.send*(). Da mesma forma, um agente que deseje receber uma mensagem deve chamar o método *receive*() ou *blockingReceve*() ambos implementados pela classe *Agent*.

Enviar e receber mensagens também podem ser escalonadas como tarefas independentes pela adição dos behaviours *ReceiverBehaviour* e *SenderBehaviour* na fila de tarefas do agente.

Todos os atributos do objeto *ACLMessage* podem ser acessados através dos métodos de acesso *set*/*get*(Nome do atributo)(), conforme definido nos padrões FIPA (já comentado anteriormente).

Além disso, esta classe também define um conjunto de constantes que devem ser utilizados para referenciar o performativo FIPA, ex: *REQUEST*, *INFORM*, etc. Quando um objeto *ACLMessage* está sendo criado, uma destas constantes devem ser informadas para selecionar o performativo. O método *reset*() limpa os valores de todos estes campos. O método *toString*() retorna toda a mensagem. Este método somente deve ser utilizado para efeito de depuração.

A.16.1. Suporte a mensagem de resposta

De acordo com as especificações FIPA, uma mensagem de resposta deve ser formada através de algumas regras de formação, por exemplo, configurando os valores

apropriados do atributo *in-reply-to*, usando o mesmo conversation-id. O JADE ajuda o programador nesta tarefa através do método *createReply*() da classe *ACLMessage*. Este método retorna um novo objeto *ACLMessage* que é uma resposta válida da mensagem. Então, o programador precisa apenas setar a ação de comunicação e a mensagem.

A.16.2. Suporte a serialização e transmissão de seqüência de bytes

Algumas aplicações podem necessitar enviar uma seqüência de bytes como conteúdo de uma *ACLMessage*. Um uso típico é a passagem de objetos Java entre dois agentes através de uma serialização. A classe *ACLMessage* dá suporte a esta tarefa através dos métodos *setContentObject*() e *getContentObject*(), que automaticamente ativa o uso da codificação BASE64. Esta funcionalidade não está em conformidade com as especificações da FIPA, por isto nenhuma plataforma de agentes pode reconhecer automaticamente o uso da codificação BASE64. Estes métodos devem ser utilizados apropriadamente e deve-se supor que a comunicação é conhecida pelos agentes.

A.16.3. O Codec ACL

Em condições normais, os agentes nunca necessitam de chamar explicitamente um codec de mensagens ACL porque isto é feito automaticamente pela plataforma. Contudo, caso seja necessário chamá-lo em determinadas circunstâncias deve-se utilizar o método *StringACLCodec* para efetuar um *parse* e codificar *ACLMessages* em formato de *string*.

A.16.4. A classe MessageTemplate

O modelo de *Behaviour* do JADE permite que um agente execute várias tarefas ao mesmo tempo. Porém, qualquer agente deve ser configurado com a capacidade de manipular várias conversações simultâneas, uma vez que a fila de mensagens é compartilhada por todos os *behavious* do agente, sendo necessário um padrão no modo de acesso. A classe *MessageTemplate* permite construir estes padrões para cada atributo da *ACLMessage*. Padrões elementares podem ser combinados com operadores OR, AND, para formar padrões mais complexos.

A.17. As tarefas dos agentes. Implementando *behaviours*

Um agente deve ter a habilidade de manipular várias tarefas concorrentes em resposta a vários agentes externos. Para se fazer um gerenciamento eficiente dos agentes, todo agente JADE é composto de uma simples *thread* de execução e todas as suas tarefas são implementadas como objetos *Behaviour*. Agentes *Multi-thread* podem ser implementados, mas não são suportados pela plataforma JADE. Um desenvolvedor que deseje implementar uma tarefa específica, deve definir uma subclasse *Behaviour*, instanciá-la e adicionar na lista de tarefas do agente. A classe *Agent*, o qual deve ser estendida pela classe do desenvolvedor, expõe dois métodos: *addBehaviour*(*Behaviour*) e *removeBehaviour*(*Behaviour*), o qual permite o gerenciamento da fila de tarefas de um agente especifico. A Figura A11 mostra o diagrama de classes da classe Behaviour.

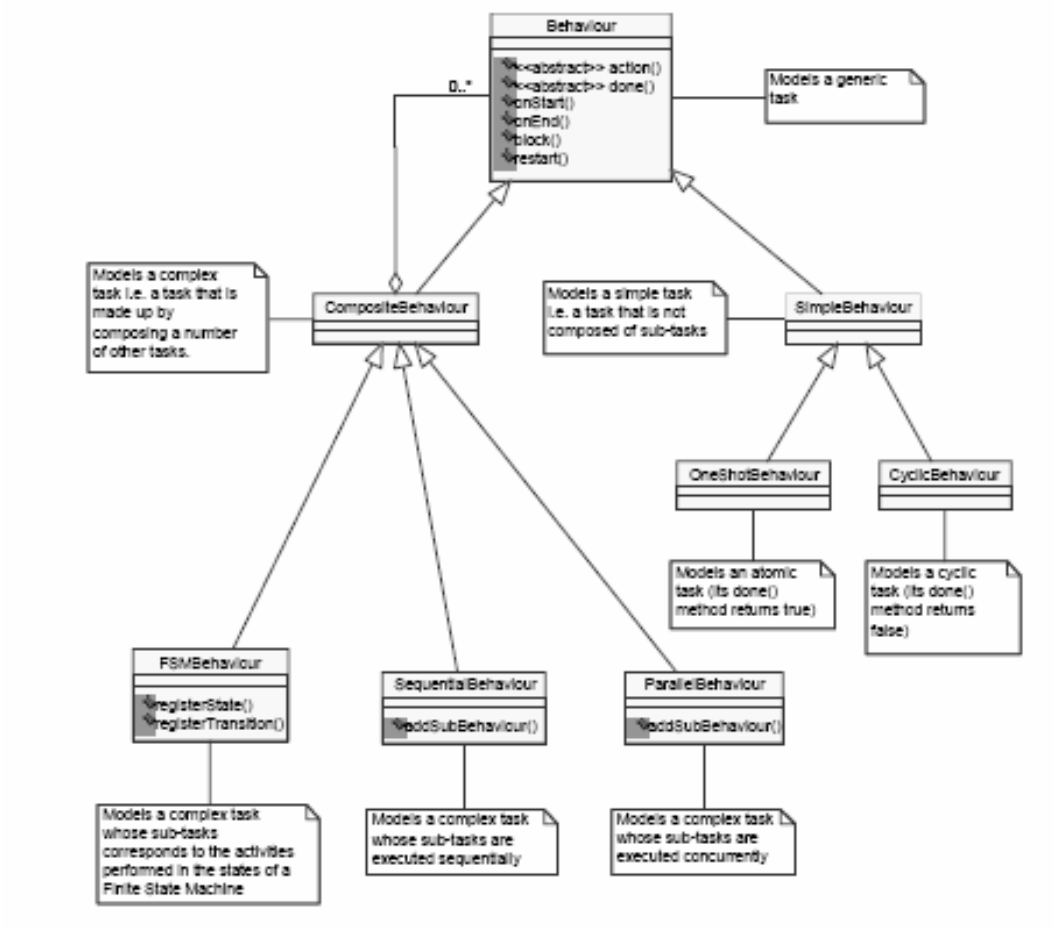

Figura A11 – Modelo UML da hierarquia de classes Behaviour

A.17.1. A classe Behaviour

Esta classe abstrata fornece uma base abstrata para modelar as tarefas dos agentes e configurar a base para o escalonador de tarefas para permitir a transição de estados.
O método *block*() permite bloquear um objeto *Behaviour* até que algum evento ocorra (normalmente a chegada de uma mensagem). Este método não afeta outros *Behaviours* de um agente, permitindo desta forma um controle mais apurado do gerenciamento dos agentes.
Um behaviour bloqueado pode ser reiniciado em três condições:

- Uma *ACLMessage* é recebida por um agente cujo *Behaviour* está esperando;

- É atingido o tempo de *timeout* do comando *block*();
- Um comando *restart*() é explicitamente chamado pelo *Behaviour*;

A classe *Behaviour* possui outros dois métodos chamados *onStart*() e *onEnd*() que podem ser sobrecarregados pelo programador, a fim de executar ações antes e depois da execução do *Behaviour*. Porém, a chamada de um método *reset*() dentro do método *onEnd*() não é suficiente para repetir ciclicamente o *Behaviour*. Neste caso, o *Behaviour* deve ser adicionado explicitamente, como, por exemplo: *myAgent.addBehaviour(this);*

A.17.2. A classe SimpleBehaviour

Esta classe abstrata modela *behaviours* atômicos (simples). Seu método *reset*()não faz nada por *default*, mas pode ser sobrecarregado e customizado pelo programador.

A.17.3. A classe onShotBehaviour

Esta classe abstrata modela *behaviours* atômicos (simples) que podem ser executados apenas uma vez e não podem ser bloqueados. Neste caso o método *done*() sempre retorna *true*.

A.17.4. A classe CyclicBehaviour

Esta classe abstrata modela *behaviours* atômicos (simples) que devem ser executadas sempre. Neste caso o método *done*() sempre retorna *false*.

A.17.5. A classe CompositeBehaviour

Esta classe abstrata modela *behaviours* que são feitos a partir da composição de um número de outros *behaviours* (filhos). Neste caso, as operações executadas pelo *behaviour* não estão definidas no próprio *behaviour*, mas dentro dos seus filhos.

A.17.6. A classe SequencialBehaviour

Esta classe é uma classe do tipo *CompositeBehaviour* que executa seus *sub-behaviour* de forma seqüencial.

A.17.7. A classe ParallelBehaviour

Esta classe é uma classe do tipo *CompositeBehaviour* que executa seus *sub-behaviour* de forma paralela e termina quando uma condição particular de seus *sub-behaviours* é alcançada.

A.17.8. A classe FSMBehaviour

Esta classe é uma classe do tipo *CompositeBehaviour* que executa seus *sub-behaviour* como uma máquina de estados finitos definida pelo usuário. Cada filho representa um estado que deverá ser executado como um estado de uma FSM.

A.17.9. A classe WakerBehaviour

Esta classe é uma classe do tipo *ShotBehaviour* que deve ser executada uma única vez quando um *timeout* ocorre.

A.17.10. A classe TickerBehaviour

Esta classe é uma classe do tipo *CyclicBehaviour* que deve ser executada periodicamente.

A.18. Executando um Bechaviour em uma thread Java dedicada

Como mencionado anteriormente, a escalonador de *behaviours* é executado de forma preemptiva, ou seja, um método *action*() nunca é interrompido para permitir que outro *behaviour* continue a execução. Somente quando o método *action*() de um determinado *behaviour* retorna, o controle é dado a outro behaviour. Esta abordagem tem muitas vantagens em termos de desempenho e escalabilidade. Contudo, quando um *behaviour* necessita executar alguma operação de bloqueio ele bloqueia todo o agente e não somente a si. Uma solução possível para resolver este problema é usar *threads* Java. O JADE fornece uma solução limpa que é a execução em *threads* dedicadas.

Um behaviour JADE pode ser executado como uma *thread* usando a classe *jade.core.behaviours.ThreadedBehaviourFactory*. Esta classe fornece o método *wrap*() que empacota um *behaviour* JADE em um *ThreadedBehaviour*. Adicionando uma *threadBehaviour* em um agente significa que um *behaviour* tradicional está sendo executado em uma *thread* dedicada.

Existem alguns pontos que devem ser observados quando manipulam-se *ThreadedBehaviours*:

- O método *removeBehaviour*() de um agente não interfere em *ThreadedBehaviours*. Ele somente pode ser removido através da captura da *thread* (*gedThread*) e na chamada do método *interrupt*().
- Quando um agente morre, fica suspenso ou é movido, os *ThreadedBehaviours* devem ser mortos explicitamente como comentado no item anterior.
- Quando um *threadedBehaviour* acessa algum recurso do agente que também são acessados por outros *threaded* ou não *threaded behaviours* deve-se tomar cuidado com problemas de sincronismo.

A.19. Protocolos de interação

A FIPA especifica um conjunto de protocolos de interação que podem ser utilizados como modelos para a comunicação de agentes. Para qualquer conversação entre agentes, o JADE distingue o papel do *initiator* e o papel do *responder*. O JADE fornece classes *Behaviours* prontas para ambos os papéis seguindo os protocolos de interação FIPA. Estas classes podem ser encontradas no pacote jade.proto.

Todos os *behaviours initiator* terminam e são removidos da fila de tarefas do agente, assim que alcançam o estado final do protocolo de interação. Para permitir o reuso de objetos Java representando estes *behaviours* sem ter que recriar novos objetos, todos os *behaviours initiators*, incluem um número de métodos *reset* com os argumentos apropriados. Além disso, todos os *behaviours initiators*, com exceção do *FipaRequestInitiatorBehaviour*, são 1:N, podem manipular vários *responder* ao mesmo tempo.

Todos os *behaviours responder*, ao contrário, são cíclicos e são re-escalonados assim que eles alcançam o estado final do protocolo de interação. Esta característica permite

ao programador limitar o número máximo de *behaviours responder* que um agente pode ser executar em paralelo.

A.19.1. AchieveRE

A visão fundamental das mensagens ACL FIPA é que uma mensagem representa um ato de comunicação, ou seja, somente uma das ações que um agente pode executar. O padrão FIPA especifica, para cada ato de comunicação, as pré-condições de viabilidade (condições que devem ser satisfeitas antes do agente poder comunicar) e o efeito racional (a razão pela qual a mensagem é enviada). O padrão especifica também que, tendo executado uma ação, o agente que enviou a mensagem não é autorizado a supor que o efeito racional necessariamente acontece. Por esta razão, ao invés de enviar uma simples mensagem, o protocolo de interação deve ser iniciado por um agente que vai enviar a mensagem, o que permite verificar se o efeito racional foi atingido ou não.

A FIPA já tem especificado alguns destes protocolos, como *FIPA-request*, *FIPA-query*, *Fipa-Request-When*, etc... Pelo fato deles compartilharem a mesma estrutura, o JADE fornece um par de classes: *AchieveREInitiator*/*Responder* que possuem uma implementação homogênea para todos estes protocolos de interação.

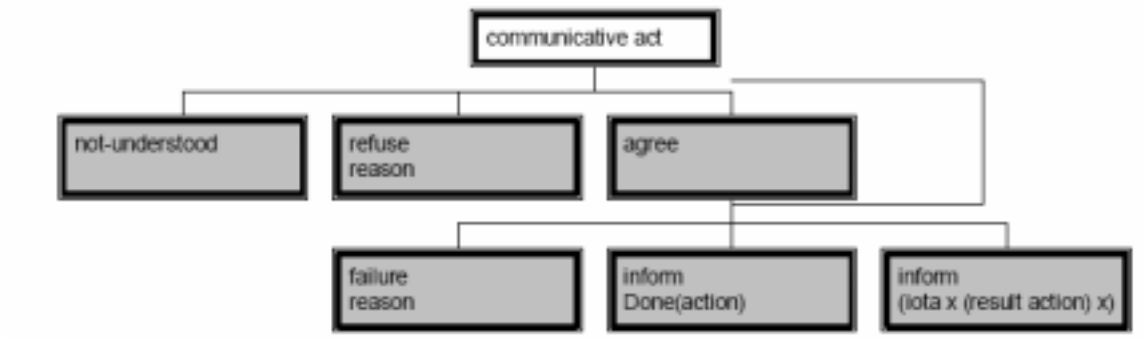

Figura A12 – Estrutura homogênea dos protocolos de interação

A Figura A12 mostra estes protocolos de interação. O *initiator* envia uma mensagem. O *responder* pode responder enviando um *not-understood* (não entendi) ou *refuse* (recusado) para alcançar o efeito racional no ato de comunicação. Além disso ele pode também concordar com a mensagem e responder após executar a ação. Neste caso, ele deve responder com o resultado da ação ou com falha caso algo tenha saído errado.

A.19.1.1. ArchieveREInitiator

Uma instância desta classe pode ser facilmente construída, passando como argumento para o seu construtor a mensagem usada para iniciar o protocolo.

A classe pode ser facilmente estendida sobrecarregando um ou todos dos seus métodos de retorno (*handle* ...*calback*), onde são fornecidos os *links* para os estados do protocolo. Outra forma de estender esta classe é através da sobrecarga de alguns métodos registrando um *behaviour* específico como o manipulador dos estados do protocolo.

A.19.1.2. SimpleArcheiveREInitiator

Esta classe é uma simples implementação de um papel *Initiator*. A principal diferença entre o *AchieveREInitiator* e o *SimpleAchieveREInitiator* é que esta versão do protocolo não permite ao programador registrar um *Behaviour* como um manipulador de estados dos protocolos.

O simples *Initiator* é 1:1, ou seja, se o programador configurar mais de um receptor dentro da *ACLMessage* passando para o seu construtor, a mensagem será enviada

somente para o primeiro receptor. Também esta implementação gerencia a expiração (*timeout*). A classe pode se facilmente estendida através da sobrecarga dos métodos *handle* ... *methods* como comentado para a classe *AchieveREInitiator*.

A.19.1.3. AchieveREResponder

Esta classe é a implementação do papel de *responder*. É muito importante que se passe o *template* de mensagem certo para seu construtor. De fato, ele é utilizado para selecionar qual *ACLMessage* deve ser atendida. O método *createMessageTemplate* pode ser usado para criar um *template* de mensagem para um determinado protocolo de interação. Em alguns casos, *templates* mais seletivos podem ser úteis, como, por exemplo, para ter uma instância desta classe para cada agente possível.
A classe pode ser facilmente estendida através da sobrecarga de um ou mais métodos: *prepare*...*methods* o qual fornece formas de manipular os estados do protocolo e, em particular, para preparar as mensagens de resposta.
Programadores habilidosos podem achar mais útil, ao invés de estender esta classe e sobrecarregar alguns de seus métodos, registrarem um *Behaviour* como gerenciador dos estados do protocolo. Os métodos *registerPrepare*... , permite isto.
Um conjunto de variáveis é disponibilizado para fornecer as chaves para o retorno das informações da área de armazenagem de dados para o *Behaviour*.

A.19.1.4. SimpleAchieveREResponder

Esta classe é uma simples implementação do *AchieveREResponder*. A principal diferença é que esta versão não permite que o programador registre *Behaviours* como gerenciadores dos estados do protocolo. Esta classe pode ser facilmente estendida

pela sobrecarga dos métodos *prepare*...*method* como mencionado para a classe *AchieveREResponder*.

A.19.2. FIPA-Contract-Net

Este protocolo de interação permite ao *Initiator* enviar uma proposta de chamada para um conjunto de *responders* avaliar as propostas e então aceitar uma preferida (ou até rejeitar todas). A Figura A13 mostra o esquema do protocolo de interação.

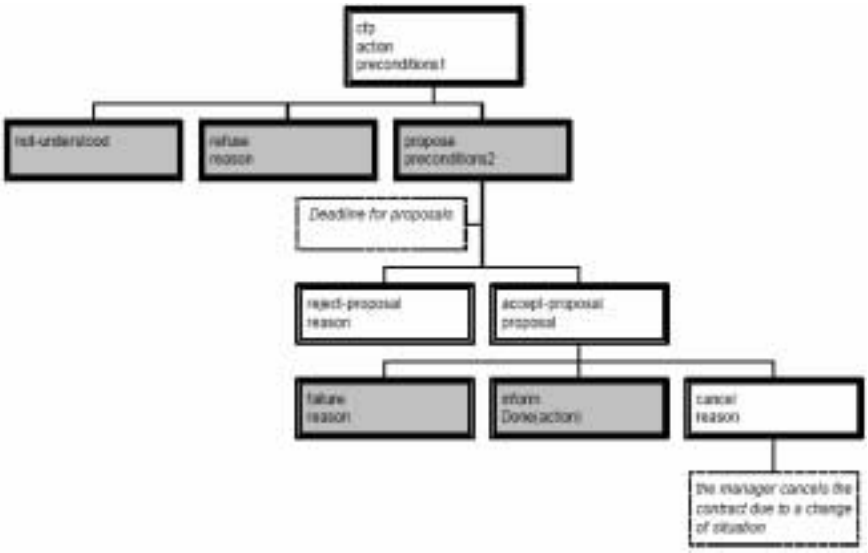

Figura A13 –Protocolo de interação simplificado

O *Initiator* solicita uma proposta de um outro agente, enviando uma mensagem CFP que especifica a ação a ser executada e, se necessário, condições para esta execução. Os *responders* podem, então, responder enviando uma mensagem *PROPOSE* incluindo as pré-condições para a ação. Os *responders* também podem enviar uma mensagem *REFUSE* para recusar a proposta ou, eventualmente, uma mensagem *NOT-UNDERSTOOD* para comunicar problemas de comunicação. O *Initiator* pode então avaliar todas as propostas recebidas e fazer a escolha de qual proposta de qual agente será aceita ou rejeitada. Uma vez os *responders* nos quais as propostas foram aceitas

(que receberam uma mensagem *ACCEPT_PROPOSAL*) terem completado suas tarefas, eles podem, finalmente, responder com um INFORM do resultado da ação ou um *FAILURE* se ocorrer algo errado.

Antes da ação tiver sido executada e a última mensagem tiver sido recebida, o *Initiator* decide cancelar o protocolo enviando uma mensagem *CANCEL*. Esta mensagem não foi implementada no JADE, pelo fato de não estar claramente especificada pela FIPA.

A.19.2.1. ContractNetInitiator

Este *Behaviour* implementa o protocolo de interação *fipa-contract-net*. Do ponto de vista do agente iniciando o protocolo, é o agente que envia a mensagem cfp – *call for proposal*, (Chamada de proposta).

Esta implementação de protocolo fornece um conjunto de métodos de retorno para manipular cada estado do protocolo, e que são chamados quando certo tipo de mensagem (baseado no ato de comunicação) é recebido. Ele também fornece dois tipos de métodos de manipulação coletiva de retornos: *handleAllReponses* e *handleAllResultNotification* que são chamados, respectivamente, depois de todas as mensagens da primeira camada (ex: *not-understood*, *refuse*, *propose*) e depois de todas as chamadas da segunda camada (*failure*, *inform*).

Como uma alternativa para os métodos de retorno, existe a possibilidade da implementação de *Behaviours* como manipuladores.

A.19.2.2. ContractNetResponder

Esta classe *Behaviour* implementa o protocolo de interação *fipa-contract-net* do ponto de vista do *responder*. É muito importante que se passe o *template* de mensagem correto como argumento para o construtor. Ele é usado para selecionar qual *ACLMessage* deve ser atendida. O método *createMessageTemplate* pode ser usado para criar um *template* de mensagem para um dado protocolo de interação. *Templates* mais seletivos podem ser usados em alguns casos como, por exemplo para ter uma instância da classe possível agente.

Esta classe pode ser facilmente estendida pela sobrecarga de um ou mais de seus métodos *prepare*...*methods* o qual fornece meios de manipular os estados do protocolo e, em particular, para preparar as mensagens de retorno.
Programadores podem achar útil, ao invés de estender esta classe e sobrecarregar um ou mais de seus métodos, registrarem um *Behaviour* como o gerenciador dos estados do protocolo.

A.19.3. FIPA-Propose

Este protocolo de interação permite que o *Initiator* envie uma mensagem de proposta para um participante indicando que ele executará alguma ação se o participante concordar. O participante responde aceitando ou rejeitando a proposta, comunicando através de uma proposta de aceite ou de rejeição.

A.19.3.1. ProposeInitiator

Este *Behaviour* implementa o protocolo de interação *fipa-propose*, do ponto de vista do agente iniciando o protocolo, ou seja, o agente que envia a mensagem de proposta. Este *behaviour* também cuida do gerenciamento de expiração (*timeout*) na espera por respostas.

A implementação deste protocolo fornece um conjunto de métodos de retorno que manipulam cada estado do protocolo, e que são chamados quando certos tipos de mensagem são recebidos. Ele também fornece um método de manipulação coletiva de mensagens de retorno *handleAllResponses* que é chamado, respectivamente, depois de todas as camadas de respostas.

Como alternativa aos métodos de retorno, existe a possibilidade de registro de *Behaviours* genéricos, como gerenciadores, ao invés da sobrecarga de métodos.

A.19.3.2. ProposeResponder

Esta classe *Behaviour* implementa o protocolo de interação *fipa-propose*, do ponto de vista do *responder* da mensagem de proposta. É muito importante passar o *template* de mensagem corretamente como argumento para seu construtor, pois ela é utilizada para selecionar qual tipo de *ACLMessage* deve ser atendida. O método *createMessageTemplate* pode ser usado para criar *templates* de mensagens para um dado protocolo de interação. Porém, *templates* de mensagens mais seletivos podem ser úteis em alguns casos, como por exemplo tendo uma instância de classe para cada possível agente.

Esta classe pode ser facilmente estendida sobrecarregando seus métodos, que fornecem o meio de se preparar mensagens de resposta. O método *prepareResponse* é chamado quando uma mensagem de um *Initiator* é recebida e a resposta deve ser retornada.

Programadores podem achar útil, ao invés de estender a classe e sobrecarregar alguns métodos, registrar um específico *Behaviour* como gerenciador de estados do protocolo.

A.19.4. FIPA-Subscribe

Este protocolo de interação permite ao *Initiator* enviar uma mensagem subscrita indicando que deseja uma subscrição. O participante processa a mensagem e responde com uma requisição de pergunta e aceita (*agree*) ou rejeita (*refuse*) a subscrição.

Se o participante concorda com a subscrição, ele comunica todo conteúdo, combinando as condições de subscrição usando um *inform-result*. O participante continua enviando os *inform-result* até que o *Initiator* cancele a comunicação enviando uma mensagem *cancel*, ou comunicando uma falha *failure*. A Figura A14 ilustra o protocolo de iteração *FIPA-Subscribe*.

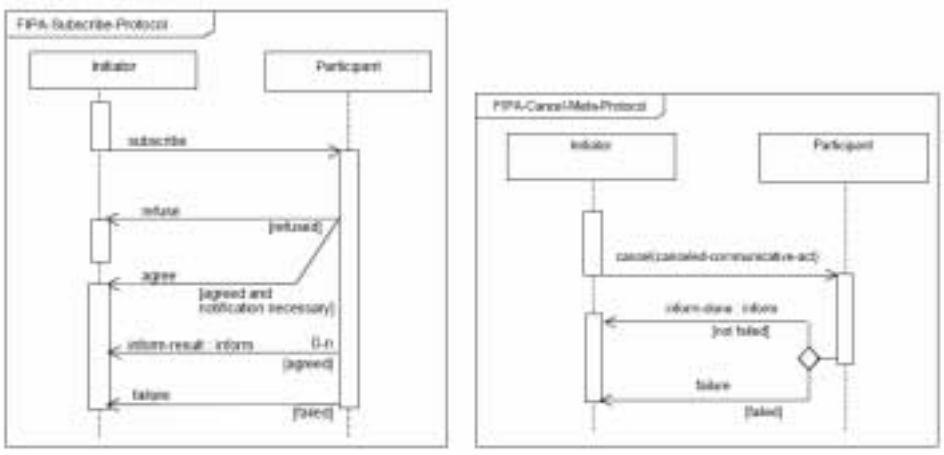

Figura A14 – Protocolo de interação fipa-subscribe

A.19.4.1. Subscribe Initiator

Este *Behaviour* implementa o protocolo de interação *fipa-subscribe* do ponto de vista do agente iniciador do protocolo, que é um agente que envia a mensagem de subscrição e recebe notificações de tempos em tempos quando a condição de subscrição se torna verdadeira (*true*). O *responder* pode então responder enviando uma mensagem de *not-understood* (não entendido), *refuse* (recusa) ou *agree* concordando com a solicitação de subscrição.

A cada vez que a condição de subscrição é verdadeira, o responder envia uma "notificação" ao *Initiator*. O *Behaviour* do *Initiator* termina se uma resposta ou uma notificação for recebida antes da mensagem de subscrição expirar, ou todos os *responders* responderem com mensagens *refuse* ou *not-understood*. Caso contrário, o *Behaviour* irá executar para sempre.

A implementação fornece um conjunto de métodos de retorno para gerenciar cada estado do protocolo; estes são chamados quando certo tipo de mensagem é recebida. A implementação padrão não contém nenhuma operação; a funcionalidade é associada com um estados especifico pela sobrecarga do método.

A.19.4.2. Subscribe Responder

Esta classe *Behaviour* implementa o protocolo de interação *FIPA-Subscribe* do ponto de vista do *responder* da mensagem de subscrição. É muito importante que o *template* de mensagem correto seja passado como argumento para seu construtor, uma vez que ele é utilizado para selecionar a *ACLMessage* que será atendida. O *responder* recebe uma mensagem de subscrição; esta mensagem contém uma condição especifica para esta subscrição. Uma vez que a subscrição requisitada foi examinada, o *responder* deve responder enviando uma mensagem *not-understood*, *refuse* ou *agree* para comunicar o estado da subscrição. Cada vez que as condições de subscrição são verdadeiras, o *responder* deve enviar uma mensagem de notificação (*notification*) ao *Initiator*.

A.19.5. Subscription

Esta classe interna representa uma subscrição. Quando uma notificação é enviada ao agente subscrito, a mensagem de notificação não deve ser diretamente enviada ao agente subscrito, mas deve ser passada ao objeto subscrição.

A.19.5.1. Subscription Manager

Um *responder subscription* somente trata de forçar e controlar a seqüência de mensagens em uma conversação de subscrição, enquanto ele delega o registro / dês-registro da subscrição e a criação de notificações para um objeto gerente de subscrição

(*subscription manager*) que implementa a interface *SubscriptionResponder*.*SubscriptionManager*.
Quando uma nova mensagem de subscrição chega, o *Subscription Responder* invoca o método *register*() de seu *Subscription Manager*, quando ma mensagem de cancelamento é recebida o método *deregister*() é chamado.

A.19.6. Estados genéricos dos protocolos de interação

O pacote *jade.proto.states* contém implementações para alguns dos estados genéricos dos protocolos de interação os quais podem ser úteis para registrar como manipuladores de estados.

A.19.6.1. A classe HandlerSelector

Esta classe abstrata do pacote *jade.proto.states* fornece uma implementação para um agente seletor de *handlers*, onde um *handler* é um *jade.core.behaviours.Behaviour*.
O construtor da classe requer a passagem de três argumentos: Uma referência ao agente, uma referência a base de dados onde a variável de seleção pode ser retornada, e, finalmente, a chave de acesso para retornar a variável de seleção da base de dados.
Esta variável de seleção será posteriormente passada como argumento para o método *getSelectionKey* que deve retornar a chave para a seleção entre os *handlers* registrados. Cada *handler* deve ser registrado através de uma *key* com o método *registerHandlers*.

A.19.6.2. A classe MsgReceiver

Esta é a implementação genérica que espera pela chegada de uma mensagem selecionando um *template*, até que o tempo de expiração é atingido.

A.20. Linguagens de conteúdo e ontologias específicas

Ontologias específicas descrevem os elementos que os agentes utilizam para criar o conteúdo das mensagens. O pacote *jade.content* e seus sub-pacotes permitem a criação de ontologias específicas e o uso independente da linguagem de conteúdo adotada.

A.21. Suporte para mobilidade

Usando JADE, os desenvolvedores de aplicações podem construir agentes móveis, com a habilidade de migrar ou copiar a si mesmo através dos elementos da rede. Os agentes JADES somente suportam a mobilidade dentro da plataforma JADE, não sendo possível a navegação por plataformas diferentes.

Movendo ou clonando é considerado um estado de transição no clico de vida de um agente. Estas ações podem ser iniciadas pelo próprio agente ou pelo AMS.

A.21.1. API JADE para mobilidade

Os dois métodos públicos *doMove*() e *doClone*() da classe agente permitem a um agente JADE migrar para outra parte ou para copiar-se com um nome diferente. A interface abstrata *jade.core.location* representa uma localidade. Os agentes não podem criar sua própria *location*, mas eles podem perguntar ao AMS pela lista de *locations* e escolher uma. O agente pode também perguntar ao AMS onde um determinado agente está.

Mover um agente, envolve em uma transferência de código e estado através de um canal da rede. Sendo assim, o agente deve gerenciar o processo de serialização e de deserialização. O JADE disponibiliza alguns métodos na classe *Agent* para gerenciar estes processos.

Para uma migração de um agente, o método *beforeMove*() é chamado no ínicio da localização quando a operação de mudança for completada com sucesso. A instância do agente é movida para o *container* de destino e é ativada, enquanto a instância original é parada. Conseqüentemente, este método é o método correto para liberar quaisquer recursos utilizados pela instância original do agente. Certamente, se estes recursos forem fechados de antemão e o ato de mover falhar, será requerida a re-abertura dele. Entretanto, como uma conseqüência imediata, qualquer informação a ser transportada pelo agente para a nova localização deve ser configurada antes que o método *doMove*() seja chamado. Por exemplo, configurar um atributo do agente no método *beforeMove*() terá impacto na instância que será movida. O método *afterMove*() é chamado no destino, assim que o agente é movido completamente e sua identificação é configurada.

Para a clonagem, o JADE suporta métodos correspondentes chamados *beforeClone*() e *afterClone*(), executados da mesma forma que os métodos mencionados no parágrafo anterior.

A.21.2. Ontologia da Mobilidade JADE

As ontologias *jade-mobility-ontology* contêm todos os conceitos e ações necessárias para suportar a mobilidade de agentes. JADE fornece a classe *jade*.*domain*.*mobility*.*MobilityOntology*, rodando como um *singleton* e dando acesso para uma única instância compartilhada da ontologia de mobilidade JADE através do método *getInstance*().

A ontologia, que estende de *JADEManagementOntology*, contém cinco conceitos e duas ações, e uma classe apropriada do pacote *jade.domain.mobility* é associada com cada conceito e ação.

Conceitos:

- *mobile-agent-description*: Descreve um agente móvel indo a algum lugar.
- *mobile-agent-profile*: Descreve o ambiente computacional necessário para o agente móvel.
- *mobile-agent-system*: Descreve o ambiente de execução usado pelo agente móvel.
- *mobile-agent-language*: Descreve a linguagem de programação usada pelo agente móvel.
- *mobile-agent-os*: descreve o sistema operacional necessário para o agente móvel.

Ações:

- move-agent: A ação de mover um agente de um local para outro.

- *clone-agent*: A ação de executar uma cópia do agente, possivelmente para executar em outra localização.

A.21.3. Acessando o AMS para a mobilidade de agentes

O JADE AMS tem algumas extensões que suportam a mobilidade de agentes, e ele é capaz de executar as duas ações presentes em *jade-mobility-ontology*. Toda ação relacionada a mobilidade pode ser requisitada para o AMS através do protocolo FIPA-*request*, com *jade-mobility-ontology* como o valor de ontologia e FIPA-SLO como o valor de linguagem.

A ação *move-agent* tem um *mobile-agent-description* como seu parâmetro. Esta ação move o agente identificado pelo nome (*name*) e endereço (*address*) constantes no *mobile-agent-description* para a localização descrita no campo *destination,* também presente no *mobile-agent-description*.

A seguir, um exemplo de um agente que deseja mover de uma localização para outra e envia uma mensagem para o AMS. A Figura A15 mostra um exemplo de uma mensagem AMS para mover um agente.

```
(REQUEST
 :sender (agent-identifier :name RMA@Zadig:1099/JADE)
 :receiver (set (agent-identifier :name ams@Zadig:1099/JADE))
 :content  (
   (action (agent-identifier :name ams@Zadig:1099/JADE)
           (move-agent (mobile-agent-description
            :name (agent-identifier :name Johnny@Zadig:1099/JADE)
            :destination (location
              :name Main-Container
              :protocol JADE-IPMT
              :address Zadig:1099/JADE.Main-Container )
            )
           )
   )
 )
 :reply-with  Req976983289310
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
 :conversation-id  Req976983289310
)
```

Figura A15 – Exemplo de mensagem para o AMS para mover um agente

A.22. Usando o JADE de uma aplicação JAVA externa

Uma instância *singleton* de um ambiente de execução JADE pode ser obtida via o método estático *jade.core.Runtime.instance*(), que fornece dois métodos para criar um *container* principal JADE ou um *container* remoto JADE.

Ambos os métodos de execução retornam um objeto *wrapper*, pertencente ao pacote *jade.wrapper*, que empacota as funcionalidades de alto nível dos *containers* de agentes, assim como instala e desinstala MTPs (*Message Transport Protocol)*, destrói o *container* e, claro, cria novos agentes. O método *createNewAgent*() deste container, retorna um objeto *wrapper* o qual encapsula algumas funcionalidades do agente, mas

reserva a autonomia dos agentes. Em resumo, a aplicação pode controlar o ciclo de vida de uma agente mas ela não pode obter uma referência ao objeto agente, como conseqüência, ela não pode executar métodos neste objeto.

A.23. Segurança JADE

JADE suporta algumas características de segurança como usuário autenticado, autorização para ações de agentes, assinatura de mensagens e criptografia.

A.23.1. Autenticação

A autenticação fornece a garantia que o usuário que está iniciando a plataforma JADE, *containers* e agentes é considerado legitimo dentro de um escopo do sistema operacional onde o *container* principal está sendo executado. Legitimo no processo de autenticação JADE implica que o usuário é conhecido pelo sistema e tem, no mínimo, uma identidade válida com uma senha associada.

Em geral o sistema de autenticação é composto de dois elementos principais: *CallBackhandler* que permite que o usuário forneça seu usuário e sua senha e o *LoginModule* que verifica se o usuário e a senha são válidos.
O mecanismo de autenticação JADE é baseado na API JAAS (J*ava Authentication and Authorization Service*) que permite a utilização de diferentes sistemas de controle de acesso (NT, UNIX, Kerberos).

A.23.2. Permissões

Todas as ações que os agentes executam na plataforma podem ser permitidas e proibidas de acordo com um conjunto de regras. Estas regras são geralmente escritas em um arquivo chamado *policy.txt* que segue a sintaxe do JAAS, mas usa uma política estendida melhor adaptada ao contexto de agentes.

A.23.3. Integridade e confidencialidade das mensagens

Assinatura e criptografia garantem certo nível de segurança quando enviando uma *ACLMessage* na mesma ou em uma plataforma remota. A assinatura garante a integridade da mensagem e a identidade do elemento que enviou a mensagem. Criptografia garante a confidencialidade da mensagem protegendo esta de acessos externos.

Uma *ACLMessage* possui um envelope e um *payload*. A assinatura e a criptografia são aplicadas a todo *payload* protegendo todas as peças importantes da informação. As informações de segurança são armazenadas no envelope (chaves, assinaturas, etc..)

A.24. Limitações

A seguir são listadas algumas limitações na atual versão da plataforma JADE:

- O JADE não possui permissões relacionadas à mobilidade.

- Pedaços de informações trocadas entre *containers* são transferidos sobre canais seguros SSL mas não são assinados.
- Estão sendo esperadas melhorias nos itens relacionados a segurança nas próximas versões do JADE.

Anexo B

B. Lista dos principais elementos do código do protótipo

B.1. Agente genérico

```
package agent;

import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.*;
import jade.content.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.content.onto.*;
import jade.content.onto.basic.*;
import functionality.*;
import ontologies.*;


public class GenericAgent extends Agent implements MutantVocabulary {

  static final int WAIT = -1;
  static final int QUIT = 0;
  private AID server;
  private Codec codec = new SLCodec();
  private Ontology ontology = MutantOntology.getInstance();
  private static final long serialVersionUID=1;
  private static String strMove;

  protected void setup() {

        // Registra a linguagem e a ontologia
         getContentManager().registerLanguage(codec);
         getContentManager().registerOntology(ontology);
         addBehaviour( new MoveCommand(this));
         addBehaviour(new ReceiveMessages(this));

  }

  protected void afterMove() {
        getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL);
        getContentManager().registerOntology(ontology);
        //Adiciona um behavior
         addBehaviour(new WaitUserCommand(this));
  }

  class MoveCommand  extends OneShotBehaviour{
            private static final long serialVersionUID=1;
           MoveCommand(Agent a) {
                        super(a);
                  }
           public void action(){
    Location dest = new jade.core.ContainerID(strMove,null);
           myAgent.doMove(dest);

           }
  }
```

```
class WaitUserCommand extends OneShotBehaviour {
        private static final long serialVersionUID=1;
        WaitUserCommand(Agent a) {
                super(a);
        }

        public void action() {
                GetFunctionality();
                        }
                }

void GetFunctionality() {

  class WaitServerResponse extends ParallelBehaviour {
        private static final long serialVersionUID=1;
        WaitServerResponse(Agent a) {

                super(a, 1);

                addSubBehaviour(new ReceiveResponse(myAgent));

                addSubBehaviour(new WakerBehaviour(myAgent, 5000) {

                        protected void handleElapsedTimeout() {
                          System.out.println("\n\tNo response from server. Please, try later!");
                          addBehaviour(new WaitUserCommand(myAgent));
                        }
                });
        }
}


class ReceiveResponse extends SimpleBehaviour {

        private static final long serialVersionUID=1;
        private boolean finished = false;

        ReceiveResponse(Agent a) {
                super(a);
        }

        public void action() {

                ACLMessage msg = receive(MessageTemplate.MatchSender(server));

                if (msg == null) { block(); return; }

                if (msg.getPerformative() == ACLMessage.NOT_UNDERSTOOD){
                        System.out.println("\n\n\tResponse from server: NOT UNDERSTOOD!");
                }
                else if (msg.getPerformative() != ACLMessage.INFORM){
                        System.out.println("\nUnexpected msg from server!");
                }
                else {
                        try {
                          ContentElement content = getContentManager().extractContent(msg);

                          if (content instanceof Result) {

                                Result result = (Result) content;
```

```
                                        if (result.getValue() instanceof Moving) {


                                        }
                                        else if (result.getValue() instanceof FunctInterface){

                                                FunctInterface fct = (FunctInterface) result.getValue();
                                                java.util.List l = fct.execute();
                                                int i=0;
                                                while(l.size()>i){
                                                        System.out.println("Informações: " + l.get(i));
                                                        i = i+1;
                                                 }
                                        }
                                                 else System.out.println("\n\tUnexpected result from server!");
                                  }
                                  else {
                                        System.out.println("\n\tUnable de decode response from server!");
                                  }
                                }
                                catch (Exception e) { e.printStackTrace(); }
                        }
                        finished = true;
                }

                public boolean done() { return finished; }

                public int onEnd() {

                        return 0;
                }
}

void lookupServer() {


        ServiceDescription sd = new ServiceDescription();
        sd.setType(SERVER_AGENT);
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.addServices(sd);
        try {
                DFAgentDescription[] dfds = DFService.search(this, dfd);
                if (dfds.length > 0 ) {
                        server = dfds[0].getName();
                        System.out.println("Localized server");
                        System.out.println("Servidor: " + server);
                }
                else  System.out.println("\nCouldn't localize server!");
        }
        catch (Exception ex) {
                ex.printStackTrace();
                System.out.println("\nFailed searching int the DF!");
        }
}

void sendMessage(int performative, AgentAction action) {

        if (server == null) lookupServer();
        if (server == null) {
                System.out.println("Unable to localize the server! \nOperation aborted!");
```

```
                return;
            }
            ACLMessage msg = new ACLMessage(performative);
            msg.setLanguage(codec.getName());
            msg.setOntology(ontology.getName());
            try {
                getContentManager().fillContent(msg, new Action(server, action));
                msg.addReceiver(server);
                System.out.println(" Servidor " +  server.getLocalName());
                send(msg);
                System.out.println("Contacting server... Please wait!");
                addBehaviour(new WaitServerResponse(this));
            }
            catch (Exception ex) { ex.printStackTrace(); }
    }
class ReceiveMessages extends CyclicBehaviour{
        Agent ag;
        ReceiveMessages(Agent a){
                super(a);
                ag=a;
        }
         void GetFunctionality(Functionality fc) {

                        sendMessage(ACLMessage.REQUEST, fc);
                 }
          public void action() {
                ACLMessage msg = receive();
                 if (msg == null) { return; }
                        if (msg.getPerformative() == ACLMessage.INFORM){
                                try{
                                 ContentElement content = getContentManager().extractContent(msg);
                                 if (!(content instanceof Result)){
                                        Concept action = ((Action)content).getAction();
                                        if (action instanceof Functionality){
                                                Functionality fc = (Functionality) action;
                                                //Faz uma requisição ao servidor de uma funcionalidade
                                                GetFunctionality(fc);
                                                 }else if(action instanceof Moving){
                                                        Moving mv = (Moving)action;
                                                        strMove = mv.getName();
                                                        MoveCommand mov = new MoveCommand(ag);
                                                        mov.stMove = strMove;
                                                        //Move o agente
                                                        addBehaviour(mov);
                                                 }
                                 }
                                 else {
                                        Result result = (Result) content;
                                        if (result.getValue() instanceof FunctInterface){
                                                FunctInterface fct = (FunctInterface) result.getValue();
                                                //Executa a funcionalidade
                                                 java.util.List l = fct.execute();
                                                 int i=0;
                                                 String response = "";
                                                 while(l.size()>i){
                                                 response = response+  (String)l.get(i);
                                                 i = i+1;
                                        }
                                        //Responde ao servidor
                                        Response rs = new Response();
                                        rs.setResponse(response);
                                        rs.setName(myAgent.getName());
```

```
                                        sendMessage(ACLMessage.INFORM,rs);
                                        }
                                        else System.out.println("\n\tUnexpected result from server!");
                                        }
                                    }catch(Exception e){
                                        e.printStackTrace();
                                    }
                                }
                            }
                        }
                    }
}
```

B.2. FM – Functionality Manager

```
package csp;

import java.sql.ResultSet;
import java.util.*;

import database.DatabaseConn;
import functionality.FunctInterface;
import ontologies.Functionality;

public class Fm {
    FunctInterface CreateFunctionality(String strClass){
        try{
            return (FunctInterface) Class.forName(strClass).newInstance();
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }

    public ArrayList BuscarFuncionalidades(){
            String strName=null;
            ArrayList list = new ArrayList();
            DatabaseConn con = new DatabaseConn();
            con.connect("jdbc:mysql://localhost:3306/specialties");
            ResultSet rs = con.ExecuteQuery("select name from functionality" );
            try{
                while(rs.next()){
                    strName = rs.getString("name");
                    list.add(strName);
                }
                rs.close();
                con.disconnect();
                return list;

            }catch(Exception e){
                e.printStackTrace();
                return null;
            }
    }
    public FunctInterface getInstanceofFunctionality(Functionality fc){
        String strPath=null;
        String strName = fc.getName();
        DatabaseConn con = new DatabaseConn();
        con.connect("jdbc:mysql://localhost:3306/specialties");
        ResultSet rs = con.ExecuteQuery("select path from functionality where key_Word = '" + strName +
"'" );
        try{
```

```
                        if(rs.next()){
                                strPath = rs.getString("path");
                                rs.close();
                                con.disconnect();
                        }
                        if(strPath !=null){
                                con.disconnect();
                                return CreateFunctionality(strPath);
                        }
                        else
                                return null;
                }catch(Exception e){
                        e.printStackTrace();
                        return null;
                }
        }
        //Método utilizado para busca de mensagens no banco de dados
        public ArrayList getServerMessages(){
                        String strName=null,strAgent=null;
                        ArrayList list = new ArrayList();
                        //Connecta ao banco
                        DatabaseConn con = new DatabaseConn();
                        con.connect("jdbc:mysql://localhost:3306/specialties");
                        //Busca as informações na tabela mensagens
                        ResultSet rs = con.ExecuteQuery("select id, message, agent from Messages" );
                        try{
                                while(rs.next()){
                                        //Carrega o array com o agente e a mensagem enviada
                                        strAgent = rs.getString("agent");
                                        strName = rs.getString("message");
                                        list.add(strName);
                                        list.add(strAgent);
                                        //Apaga a mensagem para evitar re-execução
                                        con.Execute("delete from messages where id = " + rs.getInt("id") );
                                }
                                rs.close();
                                con.disconnect();
                                return list;

                        }catch(Exception e){
                                e.printStackTrace();
                                return null;
                        }
        }

        //Método utilizado para busca de mensagens no banco de dados
        public void setServerMessages(String msg, String agentName){
                        try{
                                //Connecta ao banco
                                DatabaseConn con = new DatabaseConn();
                                con.connect("jdbc:mysql://localhost:3306/specialties");
                                //Insere as mensagens
                                String query = "insert into messages(message,agent) values( '" + msg + "','"  +
agentName + "')";
                                con.Execute(query);
                                con.disconnect();
                        }catch(Exception e){
                                e.printStackTrace();
                        }
        }
        //Método utilizado para busca de mensagens no banco de dados
        public void saveAgentsResponse(String msg, String agentName){
```

```
                        try{
                                //Connecta ao banco
                                DatabaseConn con = new DatabaseConn();
                                con.connect("jdbc:mysql://localhost:3306/specialties");
                                //Insere as mensagens
                                String query = "insert into result(message,agent) values( '" + msg + "','"  +
agentName + "')";
                                con.Execute(query);
                                con.disconnect();
                        }catch(Exception e){
                                e.printStackTrace();
                        }
        }
        public Uc_Response getResponse(){
                DatabaseConn con = new DatabaseConn();
                con.connect("jdbc:mysql://localhost:3306/specialties");
                ResultSet rs = con.ExecuteQuery("select * from result");
                Uc_Response resp = new Uc_Response();
                try{
                        while(rs.next()){
                                resp.addName(rs.getString("agent"));
                                resp.addResponse(rs.getString("message"));
                                resp.addData(rs.getString("data"));
                        }
                        rs.close();
                        con.disconnect();
                        return resp;
                }catch(Exception e){
                        e.printStackTrace();
                        return null;
                }
        }
}
```

B.3. UC – Unidade Central

```
package csp;


import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import ontologies.Functionality;
import functionality.FunctInterface;
import java.util.ArrayList;


public class Uc {
        private Fm fm = new Fm();
        public static ArrayList l = new ArrayList();
        //Método responsável pela criação e retorno da funcionalidade
        public FunctInterface getFunctionality(Functionality fc){
                return fm.getInstanceofFunctionality(fc);
        }
        public ArrayList BuscarFuncionalidades(){
                return fm.BuscarFuncionalidades();
```

```
        }
        public void setServerMessages(String msg, String agentName){
                fm.setServerMessages(msg,agentName);
        }
        public ArrayList BuscaAgentesAtivos(){
                        return l;
        }
          //Método responsável pela criação dos agentes genéricos
          public String CreateGenericAgent(int counter){
                try{
                        Runtime rt = Runtime.instance();
                        Profile p = new ProfileImpl();
                        ContainerController cc = rt.createAgentContainer(p);
                        //Criando a lista de argumentos a serem passados para o agente
                        Object reference = new Object();
                        Object args[] = new Object[1];
                        args[0]=reference;
                        AgentController generic = cc.createNewAgent("GenericAgent_" + counter,
                        "agent.GenericAgent", args);
                        generic.start();
                        l.add(generic.getName());
                        return generic.getName();
                }
                catch(Exception e){
                          e.printStackTrace();
                          return null;
                }
              }

          public ArrayList getServerMessages(){
                return fm.getServerMessages();
          }
          public void saveAgentsResponse(String msg, String agentName){
                fm.saveAgentsResponse(msg, agentName);
          }
          public Uc_Response getResponse(){
                return fm.getResponse();
          }
}
```

B.4. UCom – Unidade de Comunicação

```
package csp;
import csp.Interaction;
import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.*;
import jade.content.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.content.onto.*;
import jade.content.onto.basic.*;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import functionality.*;
import ontologies.*;
```

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.Location;
;

/*Classe Ucom responsável pela comunicação entre o CSP (servidor central de processamento)
 * e os agentes distribuídos pela rede
 */
public class UCom extends Agent implements MutantVocabulary {
// -------------------------------------------------------------------
   //Deve ser declarado devido a ser um objeto serializavel
   private static final long serialVersionUID=1;
   private int idCnt = 0;
   private Codec codec = new SLCodec();
   private LocationTableModel availableSiteListModel;
   //Ontologia criado especialmente para os agentes mutantes
   private Ontology ontology = MutantOntology.getInstance();
   private Uc uc = new Uc();

   public void updateLocations(Iterator list) {
                availableSiteListModel.clear();
                for ( ; list.hasNext(); ) {
                        Object obj = list.next();
                  availableSiteListModel.add((Location) obj);
                }
                availableSiteListModel.fireTableDataChanged();
              }

   protected void setup() {
      getContentManager().registerLanguage(codec);
      getContentManager().registerOntology(ontology);
      SequentialBehaviour sb = new SequentialBehaviour();
      sb.addSubBehaviour(new RegisterInDF(this));
      sb.addSubBehaviour(new ReceiveMessages(this));
      addBehaviour(sb);

   }

   class RegisterInDF extends OneShotBehaviour {
              private static final long serialVersionUID=1;
              RegisterInDF(Agent a) {
         super(a);
      }

      public void action() {

         ServiceDescription sd = new ServiceDescription();
         sd.setType(SERVER_AGENT);
         sd.setName(getName());
         sd.setOwnership("rlm1");
         DFAgentDescription dfd = new DFAgentDescription();
         dfd.setName(getAID());
         dfd.addServices(sd);
         try {
            DFAgentDescription[] dfds = DFService.search(myAgent, dfd);
            if (dfds.length > 0 ) {
               DFService.deregister(myAgent, dfd);
            }
            DFService.register(myAgent, dfd);
            System.out.println(getLocalName() + " is ready.");

         }
```

```
    catch (Exception ex) {
      System.out.println("Failed registering with DF! Shutting down...");
      ex.printStackTrace();
      doDelete();
    }
  }
}

class ReceiveMessages extends CyclicBehaviour {
          private static final long serialVersionUID=1;
        public ReceiveMessages(Agent a) {

    super(a);
  }

  public void action() {
    ACLMessage msg = receive();
    if (msg == null) {
        VerifyServerMessages();
        return;
        }
    try {
      ContentElement content = getContentManager().extractContent(msg);
      Concept action = ((Action)content).getAction();
      switch (msg.getPerformative()) {

        case (ACLMessage.REQUEST):
                  //Busca de uma funcionalidade
          if (action instanceof Functionality)
            addBehaviour(new HandleFunctionality(myAgent, msg));
          //Operação de retorno-> guardar em BD
          else if (action instanceof Moving)
            addBehaviour(new HandleMoving(myAgent, msg));
          else replyNotUnderstood(msg);
          break;
        case (ACLMessage.INFORM):
          if (action instanceof Response)
             addBehaviour(new HandleResponse(myAgent, msg));
          break;

        default: replyNotUnderstood(msg);
      }

    }

    catch(Exception ex) { ex.printStackTrace(); }
  }

}

class HandleFunctionality extends OneShotBehaviour {
  private ACLMessage request;
  private static final long serialVersionUID=1;

          HandleFunctionality(Agent a, ACLMessage request) {

    super(a);
    this.request = request;
  }

  public void action() {
```

```
    try {

      ContentElement content = getContentManager().extractContent(request);
      Functionality fc = (Functionality)((Action)content).getAction();
      //Solicita a classe Controladora a criação da funcionalidade
                        FunctInterface  fct = uc.getFunctionality(fc);
                        ACLMessage reply = request.createReply();
                        reply.setPerformative(ACLMessage.INFORM);
                        //Verifica se a funcionalidade foi retornada
      if(fct!=null){
                              //Envia o objeto solicitado
                              Result result = new Result((Action)content, fct);
                              getContentManager().fillContent(reply, result);
      }else
        //Caso a funcionalidade não seja encontrada, retorna um objeto
        //problema
      {
      }
        send(reply);
    }
    catch(Exception ex) { ex.printStackTrace(); }
  }
}
class HandleResponse extends OneShotBehaviour {
        private ACLMessage request;
        private static final long serialVersionUID=1;

        HandleResponse(Agent a, ACLMessage request) {

          super(a);
          this.request = request;
        }

        public void action() {

          try {

            ContentElement content = getContentManager().extractContent(request);
            Response rs = (Response)((Action)content).getAction();
            uc.saveAgentsResponse(rs.getResponse(), rs.getName());
          }
          catch(Exception ex) { ex.printStackTrace(); }
        }
      }

class HandleMoving extends OneShotBehaviour {
  private static final long serialVersionUID=1;
  private ACLMessage request;

  HandleMoving(Agent a, ACLMessage request) {

    super(a);
    this.request = request;
  }

  public void action() {

    try {
      MakeOperation mo = (MakeOperation)((Action)content).getAction();
      ACLMessage reply = request.createReply();
      Object obj = processOperation(mo);
      if (obj == null) replyNotUnderstood(request);
```

```
          else {
            reply.setPerformative(ACLMessage.INFORM);
            Result result = new Result((Action)content, obj);
            getContentManager().fillContent(reply, result);
            send(reply);
            System.out.println("Operation processed.");
          }
        }
        catch(Exception ex) { ex.printStackTrace(); }
      }
    }

    void replyNotUnderstood(ACLMessage msg) {
      try {
        ContentElement content = getContentManager().extractContent(msg);
        ACLMessage reply = msg.createReply();
        reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
        getContentManager().fillContent(reply, content);
        send(reply);
        System.out.println("Not understood!");
      }
      catch(Exception ex) { ex.printStackTrace(); }
    }

    String generateId() {
      return hashCode() + "" + (idCnt++);
    }

private void VerifyServerMessages(){
            ArrayList l = uc.getServerMessages();
            int i=0;
            while(i<l.size()){
                        try{
                                    if(!(TrataMensagem(l))){
                                                System.out.println("Erro ao processar mensagem!");
                                    }
                           i++;
                           i++;
                        }catch(Exception e){
                                    e.printStackTrace();
                        }
            }

}
private void sendMessage(String AgentName, AgentAction ac){
            try{
                        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                        msg.setLanguage(codec.getName());
                        msg.setOntology(ontology.getName());
                        AID agAID = new AID(AgentName, AID.ISLOCALNAME);
                        msg.addReceiver(new AID(AgentName, AID.ISLOCALNAME));
                        getContentManager().fillContent(msg, new Action(agAID,ac));
                        send(msg);
            }catch(Exception e){
                        e.printStackTrace();
            }

}
//Le a mensagem do banco de dados e decodifica
private boolean TrataMensagem(ArrayList l){
            for(int i=0;i<l.size();i=i+2){
                        String AgentName =(String)l.get(i+1);
```

```
                String msg =(String)l.get(i);
                int index = msg.indexOf(constants.Constants.MSGFUNCT);
                if(index!= -1){
                        //Decodofica a Funcionalidade e envia ao agente
                        msg = msg.substring(constants.Constants.MSGFUNCT.length(),msg.length());
                        Functionality fc = new Functionality();
                        fc.setName(msg);
                        fc.setId("1");
                        AgentAction ac = (AgentAction) fc;
                        sendMessage(AgentName,ac);
                }else
                {
                        index = msg.indexOf(constants.Constants.MSGMOV);
                        if(index!= -1){
                                //Decodifica o movimento e envia ao agente
                                msg = msg.substring(constants.Constants.MSGMOV.length(),msg.length());
                                Moving fc = new Moving();
                                fc.setName(msg);
                                fc.setId("1");
                                AgentAction ac = (AgentAction) fc;
                                sendMessage(AgentName,ac);
                        }else
                        {
                                System.out.println("Mensagem não identificada!");
                                return false;
                        }
                }
        }


        return true;
}
public AID lookupAgent(String agent) {
        AID GncAgent = new AID();
        ServiceDescription sd = new ServiceDescription();
                sd.setType(GENERIC_AGENT);
                DFAgentDescription dfd = new DFAgentDescription();
                dfd.addServices(sd);
                try {
                        DFAgentDescription[] dfds = DFService.search(this, dfd);
                        if (dfds.length > 0 ) {
                                GncAgent = dfds[0].getName();
                                System.out.println("Localized Agent");
                                System.out.println("Generico: " + GncAgent);
                        }
                        else  System.out.println("\nCouldn't localize Agent!");
                         return GncAgent;
                }
                catch (Exception ex) {
                        ex.printStackTrace();
                        System.out.println("\nFailed searching int the DF!");
                        return null;
                }
          }


}
```

B.5. Interface de funcionalidade

```
package functionality;

import java.util.*;

public interface FunctInterface  {
        public List execute();

}
```

B.6. Pacote de funcionalidade

```
package functionality;

import snmp.*;

import java.io.Serializable;
import java.net.InetAddress;
import java.util.ArrayList;
import java.util.List;

public class Configuration  implements FunctInterface,Serializable{
        private static final long serialVersionUID=1;
        private List l;

        public List  execute() {
                try
                 {
                        //Busca as informações disponíveis no host local
                         InetAddress hostAddress = InetAddress.getByName("localhost");
                        String community = "public";
                        int version = 0;    // SNMPv1
                        SNMPv1CommunicationInterface comInterface = new
SNMPv1CommunicationInterface(version, hostAddress, community);
                        String baseID ="1.3.6.1.2.1.1";
                        SNMPSequence pair;
                        SNMPObjectIdentifier snmpOID;
                        SNMPObject snmpValue;
                        SNMPVarBindList tableVars = comInterface.retrieveMIBTable(baseID);
                        l = new ArrayList();
                        for (int i = 0; i < tableVars.size(); i++)
                         {
                                air = (SNMPSequence)(tableVars.getSNMPObjectAt(i));
                                snmpOID = (SNMPObjectIdentifier)pair.getSNMPObjectAt(0);
                                snmpValue = pair.getSNMPObjectAt(1);
                                //Adiciona os elementos recuperados na lista de retorno
                                l.add(snmpValue.toString());

                        }
                    return l;
                    }
                catch(Exception e)
                {
                   System.out.println("Exception during SNMP operation:  " + e + "\n");
                   return l;
                }
        }
```

```
}
```

B.7. Ontologia

```
package ontologies;

import functionality.*;
import jade.content.onto.*;
import jade.content.schema.*;


public class MutantOntology extends Ontology implements MutantVocabulary {
  private static final long serialVersionUID=1;
  public static final String ONTOLOGY_NAME = "Mutant-Ontology";
  private static Ontology instance = new MutantOntology();

  public static Ontology getInstance() { return instance; }

  private MutantOntology() {

    super(ONTOLOGY_NAME, BasicOntology.getInstance());

    try {

      // Functionality
      ConceptSchema cs = new ConceptSchema(FUNCTIONALITY);
      add(cs, Functionality.class);
      cs.add(FUNCTIONALITY_ID, (PrimitiveSchema) getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
      cs.add(FUNCTIONALITY_NAME, (PrimitiveSchema) getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
      // Moving
            cs = new ConceptSchema(MOVING);
      add(cs, Moving.class);
      cs.add(MOVING_ID, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
      cs.add(MOVING_NAME, (PrimitiveSchema) getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
      cs = new ConceptSchema(RESPONSE);
      add(cs, Response.class);
      cs.add(RESPONSE_ID, (PrimitiveSchema) getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
      cs.add(RESPONSE_LIST, (PrimitiveSchema) getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
      cs.add(RESPONSE_NAME, (PrimitiveSchema) getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
          add(cs = new ConceptSchema(MONITOR), monitor.class);
          add(cs = new ConceptSchema(CONFIGURATION), Configuration.class);

    }
    catch (OntologyException oe) {
      oe.printStackTrace();
    }
  }
}
```

Anexo C

C. Gerenciamento de Redes

C.1. Introdução

Os ambientes de comunicação tiveram um enorme crescimento em complexidade. Esse desenvolvimento propicia a criação de serviços cada vez mais exigentes em relação ao ambiente de rede. Seguindo esta tendência, os equipamentos de redes ficaram mais complexos e, muitas das vezes, heterogêneos e incompatíveis.

O gerenciamento desses ambientes tornou-se um desafio para os atuais operadores e administradores de rede, que têm que lidar com a proliferação de interfaces variadas homem-máquina e com problemas de interoperabilidade.

Os sistemas atuais de gerenciamento de rede são fortemente ligados ao modelo cliente-servidor de sistemas distribuídos, que se mostra ineficiente em relação a abordagens autônomas, quando a necessidade de comunicação e retorno de informações é alta, necessidade esta quase sempre presente em gerenciamento de redes (RAMOS, 2004) (BIESZCZAD, 1998).

Estão disponíveis no mercado diversas ferramentas poderosas para gerência de redes, mas a grande maioria é limitada em algumas funcionalidades importantes como gerenciamento de configuração e de serviços. Além disso, são muitas vezes de plataforma fechada e exigem uma grande demanda de trabalho por parte dos administradores de rede, uma vez que, toda a tomada de decisão fica sob sua responsabilidade.

O administrador de redes possui, atualmente, uma grande quantidade de dados, mas nem sempre é fácil transformá-los em informações úteis para se tomar ações em tempo hábil. Diante desse cenário, percebe-se a necessidade da criação de instrumentos mais autônomos, pró-ativos e inteligentes, que possam auxiliar o administrador de redes não só na coleta de dados e extração de informações, mas na tomada de decisões e execução de ações que efetivem o controle e planejamento das redes (MONTEIRO, 2005).

C.2. Conceitos

Existem na literatura vários conceitos sobre gerência de redes. Esses conceitos são, na maioria das vezes, específicos para cada ramo de aplicação e normalmente são gerados por organizações de padronização, como ISO (*International Organization for Standardization*) e ITU (PRAS, 1995) (ISO/IEC, 1999) (ITU-T, 1997).

Neste trabalho, é utilizada a seguinte definição: Gerenciamento de redes é o ato de iniciar, monitorar e modificar as operações das funções primárias de uma rede (PRAS, 1995), funções primárias estas que estão diretamente ligadas às necessidades do usuário da rede.

C.3. A importância da gerência de redes

A importância da gerência de redes é melhor entendida a partir do entendimento dos seus objetivos. A função principal da gerência de rede é estabelecer técnicas e práticas que garantam a melhor relação entre a demanda dos usuários, a qualidade do tipo de serviço oferecido e o funcionamento viável, técnica e economicamente, das redes (MONTEIRO, 2005). Para garantir esta demanda, são necessários os seguintes objetivos funcionais (GIMENEZ, 2004):

Assegurar o funcionamento:

A garantia do funcionamento da rede implica, além de evitar paralisações dos serviços, a garantia de qualidade de serviço para todo tipo de serviço oferecido pela rede. O uso de ferramentas adequadas na monitoração e levantamento de informações é importante para garantir o bom funcionamento da rede.

Assegurar Bom Desempenho:

Assegurar o funcionamento da rede não é suficiente se o desempenho desta não estiver adequado. A gerência de redes, utilizando-se de ferramentas adequadas, deve monitorar todas as variáveis relevantes ao sistema e tomar as medidas necessárias, em tempo hábil, para garantir um nível mínimo de desempenho.

Reduzir os custos de manutenção:

A fatia maior dos custos não se concentra na compra e instalação de equipamentos, mas sim na manutenção e gerenciamento da mesma (GIMENEZ, 2004). Os custos vinculados à gerência de rede crescem com o aumento dos dispositivos gerenciáveis e com a prática da gerência reativa. A gerência reativa, que é a forma de gerenciamento mais utilizada pelos administradores de redes, é, na maioria dos casos, ineficiente, pois quase sempre se resolve problemas após a percepção destes por parte dos usuários. Estudos mostram que a utilização de mecanismos pró-ativos na gerência de rede podem, efetivamente, reduzir sensivelmente os custos de gerência (GIMENEZ, 2004).

C.4. Áreas funcionais de gerenciamento de redes

A arquitetura de gerenciamento OSI define cinco áreas funcionais para prover as necessidades do usuário no gerenciamento de suas redes (ISO/IEC, 1999) (PEREIRA, 2001):

- Gerenciamento de configuração
- Gerenciamento de falhas
- Gerenciamento de desempenho
- Gerenciamento de segurança
- Gerenciamento de contabilização

Apesar de apresentarem objetivos distintos (ISO/IEC, 1999), as áreas funcionais relacionam-se no sentido de que informações geradas em uma área podem ser utilizadas como suporte para decisões em outras áreas.

C.4.1. Gerência de Configuração

O objetivo da gerência de configuração é fornecer os controles necessários para a coleta de informações, monitoramento e fornecimento de dados para a preparação, iniciação, partida, operação contínua e, posteriormente, a suspensão da interconexão entre sistemas abertos. A gerência de configuração viabiliza a identificação de objetos gerenciados, a coleta de informações de condições e de demanda sobre estes objetos e a disponibilização desses dados para usos específicos, tais como (ISO/IEC, 1999), (PEREIRA, 2001):

- Atribuição de valores iniciais aos parâmetros de um sistema aberto (manutenção das configurações do sistema).
- O início e o encerramento de operações sobre objetos gerenciados.
- Alteração na configuração de sistemas abertos.
- Associação de nomes a conjuntos de objetos gerenciados.
- Manutenção da versão dos softwares do sistema de rede (sistemas operacionais, drivers, etc...).
- Atualizações de software e eventuais atualizações de hardwares.
- Escalonamento das alterações.

Em um primeiro momento, pode parecer que os itens de controle da gerência de configuração listados anteriormente, são simples de serem executados. Porém, em entrevista com profissionais da área, percebe-se que em grandes redes, onde o existe um número elevado de equipamentos heterogêneos, a administração dessas informações se torna bastante complexa. A seguir são listadas algumas das dificuldades apontadas por profissionais da área:

Dificuldade em manter um registro fiel das informações relativas ao hardware e software de cada equipamento participante da rede.

A dificuldade aparece devido a problemas de comunicação entre áreas que trabalham simultaneamente nos equipamentos. Por exemplo: Equipes de manutenção de hardware e software, equipes de instalação e configuração de software, equipes de manutenção de rede, ações diretas realizadas por usuários, entre outras. Todas estas equipes podem, em um determinado momento, atualizar, alterar, incluir excluir itens de software, hardware e configurações, porém sem executar uma atualização na base de informações de configuração.

Falta de uma base organizada de configuração.

Muitas vezes, uma base única organizada com as informações de configuração não existe, o que dificulta a manutenção dos dados.

Dificuldade em encontrar ferramentas de apoio aderentes a vários equipamentos diferentes.

Existem várias ferramentas disponíveis no mercado, porém, são de custo relativamente alto e muitas vezes não conseguem suportar todos os equipamentos disponíveis na rede.

C.4.2. Gerência de Falhas

A gerência de falhas é responsável pela detecção e localização de falhas, isolamento da falha do resto da rede, correção da falha e, em certos casos, a re-configuração de equipamentos de forma a minimizar o impacto do problema no restante da rede. Falhas podem fazer com que a rede fique impossibilitada de cumprir seus objetivos operacionais. As falhas podem apresentar-se de forma transitória ou permanente e são normalmente detectadas a partir da coleta e mapeamento de informações na rede. Os objetivos da gerência de falhas são (ISO/IEC, 1999):

- Manter e examinar logs de erros.
- Agir em notificações de detecção de erros.
- Procurar e identificar falhas.
- Desenvolver e executar seqüências de diagnósticos.
- Corrigir falhas.

O ideal é que as falhas sejam identificadas e corrigidas antes que seus efeitos sejam percebidos. Algumas ações que podem ser tomadas neste sentido são:

- Monitoramento das taxas de erro do sistema.
- Análise da severidade dos relatórios de alarme.

Porém, na maioria das vezes, para que as correções de falhas ocorram, a intervenção humana se faz necessária. Por esse motivo, estas ações nem sempre são tomadas em um tempo hábil, porque muitas das vezes é humanamente impossível que os administradores da rede possam estar 100% do tempo monitorando 100% dos eventos relevantes. Para minimizar esse problema, várias ferramentas de apoio ao gerenciamento de falhas foram desenvolvidas. Porém, a autonomia destas é muitas vezes limitada, além de quase sempre gerarem uma sobrecarga de dados na rede.

C.4.3. Gerência de Desempenho

A gerência de desempenho tem como objetivo avaliar o comportamento dos recursos em um ambiente de rede e verificar sua eficiência. A avaliação da eficiência se dá através de parâmetros estatísticos tais como: atrasos, vazão, disponibilidade e número de retransmissões (PEREIRA, 2001). A avaliação pode encontrar problemas como: taxa de utilização de recursos, volume de tráfego excessivo, tempo de resposta alto e estrangulamentos.

O gerenciamento de desempenho é composto por um conjunto de funções responsáveis por garantirem que não ocorra insuficiência de recursos quando a taxa de utilização do sistema chegar próximo ao seu limite máximo (PEREIRA, 2001). Para isto, um valor limite e um valor de alerta são definidos para cada recurso monitorado do sistema. Ao atingir estes valores, notificações são enviadas aos administradores da rede para que ações sejam tomadas com o objetivo de reduzir o problema. A seguir, são listados alguns pontos que devem ser realizados para se buscar garantir o desempenho do sistema (ISO/IEC, 1999):

- Buscar informações estatísticas.
- Manter e examinar dados históricos.
- Determinar o desempenho do sistema diante de condições naturais e artificiais.
- Alterar modos de configuração do sistema com o propósito de conduzir as atividades de gerenciamento de desempenho.

Assim como a gerência de falhas, a gerência de desempenho também necessita, na maioria das vezes, de intervenção humana para serem corrigidas. As ferramentas de apoio a esta área de gerência, são também pouco autônomas e também geram sobrecarga de dados na rede.

C.4.4. Gerência de Segurança

O objetivo do gerenciamento de segurança é dar suporte a aplicação de políticas de segurança, protegendo os recursos da rede de acessos indevidos. As funções são:

- Criação, destruição e controle de serviços e mecanismos de segurança.
- Distribuição de informações de segurança.
- Reporte de eventos de segurança.
- Proteção da informação.
- Controle do acesso a recursos.
- Registro dos eventos de segurança.

Mecanismos como, por exemplo, (IDS - *Intrusion Detection System*). Sistemas de detecção de intrusos fazem suas monitorações nos cabeçalhos dos pacotes e também em seu campo de dados, possibilitando a verificação de ataques no nível de aplicação (para pacotes TCP e UDP) (SILVEIRA, 2000).

C.4.5. Gerência de contabilização

A gerência da contabilização permite estabelecer taxas de utilização a recursos no ambiente de rede e os custos a serem identificados na utilização destes recursos (PEREIRA, 2001). As funções do gerenciamento da contabilização são:

- Contabilização do tráfego nas fronteiras da rede (pacotes de entrada e saída, etc...).
- Detecção de gastos excessivos de um utilizador de recursos que limitem a utilização da rede.
- Utilização ineficiente dos recursos da rede.
- Previsão de recursos necessários para a adição e reajustamento de recursos na rede.
- Fonte de informação para informações de taxação.

C.5. Arquitetura típica de Gerenciamento de redes

A recomendação X.701 da ITU-T descreve a arquitetura empregada na maioria dos sistemas de gerenciamento de redes (MONTEIRO, 2005) (CISCO, 2006). A Figura C.1 mostra a arquitetura típica de gerenciamento de redes.

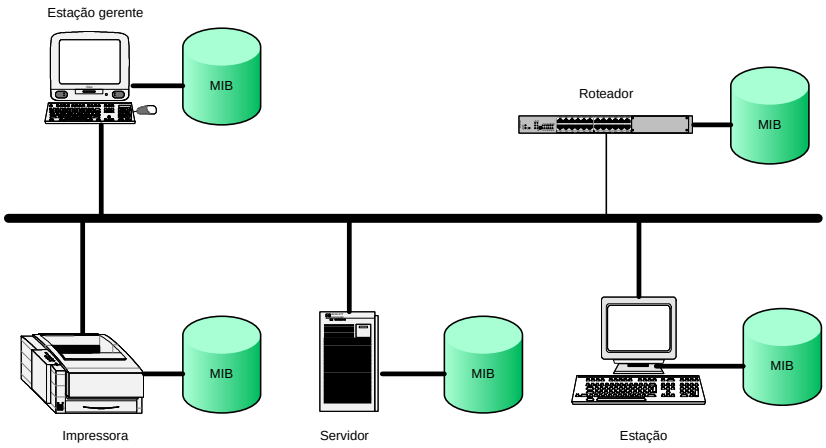

Figura C.1 – Arquitetura típica de gerência de redes

As atividades de gerenciamento são feitas através da manipulação de dispositivos gerenciáveis (ITU-T, 1997). A seguir são listadas as seis principais entidades desta arquitetura (MONTEIRO, 2005):

- Dispositivos gerenciáveis
- Processo gerente
- Processo agente
- Base de informações de gerência
- Funções de gerência
- Protocolo de gerência

Dispositivos Gerenciáveis

Qualquer dispositivo que possua a capacidade de processar, armazenar e disponibilizar informações relevantes à gerência de rede é um potencial dispositivo gerenciável. A opção de torná-lo gerenciável ou não, é feita a partir de análises realizadas pelo administrador da rede. As informações possíveis de serem disponibilizadas pelo dispositivo são relacionadas pela base de dados denominada MIB (*Management Infomation Base*).

Processo Gerente

O processo gerente é o ponto central de armazenamento das informações enviadas pelos dispositivos gerenciáveis. Um processo gerente é um mecanismo de software que obtém as informações através de chamadas de requisição, ou através de chamadas efetuadas pelo próprio dispositivo gerenciado. Um processo gerente pode controlar vários dispositivos gerenciáveis.

Processo agente

As informações de gerência em um dispositivo gerenciado são chamadas de objetos gerenciáveis (MONTEIRO, 2005). O processo agente é responsável pelo armazenamento destas informações e pela disponibilização destas quando requisitado pelo processo gerente.

Base de informações de gerência

A base de informações de gerência, definida pela ISO como MIB (*Management Information Base*), é um esquema conceitual que engloba relacionamentos entre

objetos e determinadas operações que podem ser executados por ele. A MIB possui um modelo de implementação das informações de gerenciamento. O modelo abstrato entre outras coisas define (KOTSAKIS, 1995):

- Princípios de nomeação de objetos
- A estrutura lógica das informações de gerenciamento
- Conceitos relacionados com classes de objetos gerenciáveis e o relacionamento entre eles.

Como a MIB é um conjunto de dados padronizado, organizada de forma hierárquica e conhecida pelos gerentes e agentes, ela viabiliza a troca de informações entre eles. A Figura C.2 (GIMENEZ, 2004) mostra de forma resumida a organização da MIB.

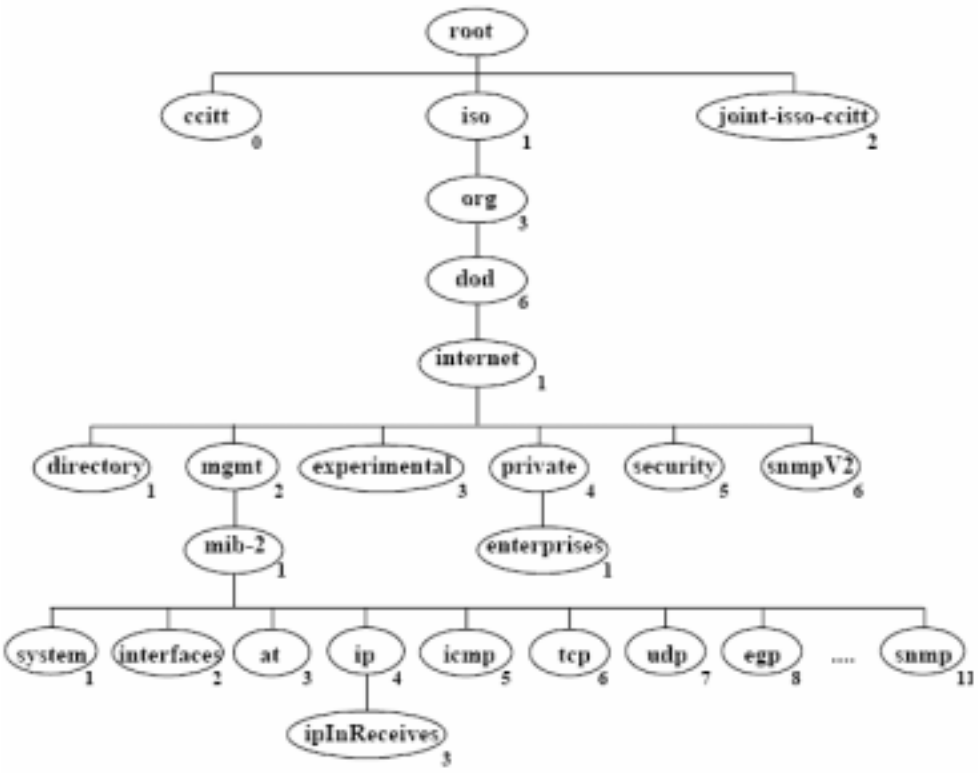

Figura C.2 - Estrutura Hierárquica da MIB

Funções de gerência

As funções de gerência são formas padronizadas, utilizadas pelos gerentes, para troca de informações com seus agentes. As funções a seguir são típicas de qualquer padrão ou sistema de gerência (GIMENEZ, 2004):

- GET: usado pelo gerente para requisitar uma informação de gerência ao agente.
- SET: usado pelo gerente para requisitar ao agente a alteração de uma configuração no objeto gerenciável.
- RESPONSE: usada pelo agente para responder a uma requisição efetuada pelo gerente.
- RESPONSE_EVENT: usado pelo agente para reportar ao gerente a ocorrência de um evento pré-determinado.

Protocolo de gerência

É o responsável por encapsular as primitivas de gerência e seus respectivos parâmetros, transformando-os em PDUs (*Protocol Data Units*) padronizados, garantindo a perfeita comunicação entre agente e gerente. Tudo que se refere à codificação, interpretação e apresentação dos dados de gerência, é proporcionado pelo protocolo (MONTEIRO, 2005).

C.6. O protocolo SNMP

O protocolo SNMP foi publicado em 1988 (STALLLINS, 1998), como protocolo auxiliar, durante a especificação e padronização do protocolo CMOT (*Common Management Information Service and Protocol over TCP/IP – RFC 1095*), cujo objetivo era adaptar o protocolo CMISE (Modelo OSI) ao padrão TCP/IP (GIMENEZ, 2004).

O SNMP é um protocolo projetado para dar remotamente a um usuário a capacidade de administrar uma rede de computadores buscando e atribuindo valores em dispositivos e monitorando eventos da rede. Ele se baseia na arquitetura clássica centralizada, onde o programa cliente (gerente) faz conexões ao programa servidor (agente), presente nos nós da rede, trazendo informações ou modificando o status da base de dados (MIB) do agente. É função do agente (STALLINS, 1998):

- Coletar e manter informações sobre o ambiente local.
- Fornecer informações ao gerente, em resposta a uma solicitação ou quando algum evento pré-determinado acontece (*Trap*).
- Responder ao gerente comandos para alterar a configuração local ou parâmetros de operação.

As estações gerente geralmente fornecem uma interface aos usuários onde o administrador da rede pode observar os eventos da rede. Esta interface permite aos usuários enviar comandos (por exemplo: desativar *links*, coletar estatísticas ou informações de desempenho) e fornece a lógica para sumarização e formatação da informação coletada pelo sistema.

Por se basear em TCP/IP, o SNMP é totalmente compatível com a Internet. O seu funcionamento é baseado na troca informações da rede através de mensagens, tecnicamente conhecidas como Unidades de Dados de Protocolo (PDU). A mensagem (PDU) pode ser encarada como um objeto que contém variáveis que têm nomes e valores. A PDU também contém informações de autenticação do gerente (*manager*), para que o agente saiba que foi realmente o gerente que solicitou ou modificou dados de sua MIB (SNMPv3).

Há quatro tipos de PDU que o protocolo SNMP emprega para monitorar uma rede: duas lidam com a leitura de dados dos terminais (*get*), uma lida com a atribuição de valores aos dados dos terminais (*set*), e a última, o *trap*, é usada para monitorar eventos de rede como o ligamento ou o desligamento de um terminal da rede.

Se um usuário precisa saber se um terminal está ligado à rede, ele usaria SNMP para enviar uma PDU de leitura (*get*) a este terminal. Se o terminal estiver conectado à rede, o usuário receberia de volta a PDU (*get-response*), com o valor "sim, o terminal está conectado". Se o terminal estivesse desligado, o usuário receberia um pacote enviado pelo terminal, que está sendo desligado, informando sobre o desligamento. Esta última seria uma PDU de *trap*. A Figura C.3 (GIMENEZ, 2004) mostra o esquema de formação das mensagens SNMP.

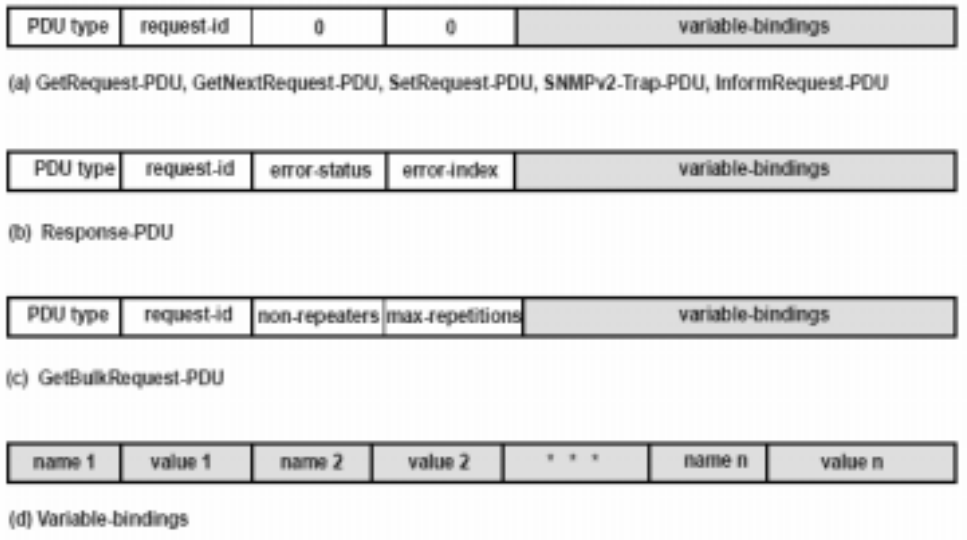

Figura C.3 – Estrutura das mensagens SNMPv2

Entre as vantagens do protocolo SNMP, pode-se citar:

- Popularidade: Agentes SNMP estão disponíveis para dispositivos de rede que variam de computadores, pontes, *modems* e impressoras.
- Simplicidade de utilização.
- Fácil implementação.
- Facilidade de expansibilidade.

A principal desvantagem apontada ao protocolo SNMP, é relacionado a aspectos de segurança. Como tentativa de solucionar este problema, foram publicados um conjunto de documentos definindo uma estrutura para incorporar características de segurança,

incluindo segurança de rede e controle de acesso (SNMPv3). O SNMPv3 não substitui as versões anteriores (SNMPv1 e SNMPv2), ela somente incorpora mecanismos de segurança, mantendo totalmente a compatibilidade. A figura C.4 (STALLINS, 1998) mostra o esquema de formação de mensagens utilizando o SNMPv3.

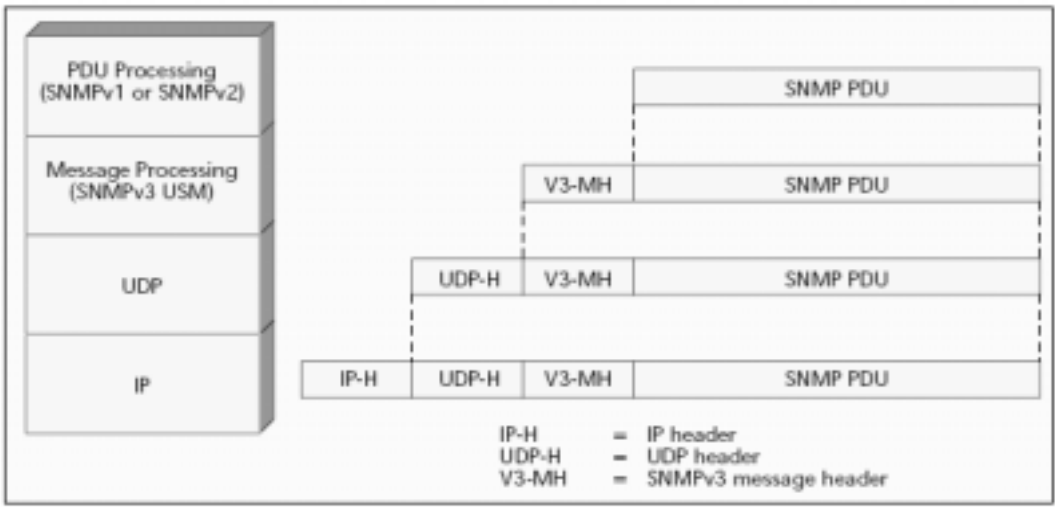

Figura C.4 – Arquitetura SNMPv3

O SNMP não é adequado para gerenciar redes interligadas em enlaces de longa distância, porque estas, normalmente trabalham com taxas menores que as das redes locais e o tráfego intenso de informações de gerenciamento pode provocar congestionamento entre os enlaces que conectam estas redes (LESSA, 1999).

C.7. RMON

O padrão RMON, para monitoramento remoto, oferece uma arquitetura de gerenciamento distribuída para análise de tráfego, resolução de problemas, demonstração de tendências e gerenciamento pró-ativo de redes de modo geral.
Criado pelos mesmos grupos que desenvolveram o TCP/IP e o SNMP, o RMON é um padrão IETF de gerenciamento de redes.

As principais características do RMON são: interoperabilidade independentemente de fabricante, capacidade de fornecer informações a respeito das causas de falha no funcionamento normal da rede, assim como da severidade dessa falha, além de oferecer ferramentas para diagnóstico da rede. Além destas características, o RMON oferece um mecanismo pró-ativo para alertar o administrador dos eventuais problemas da rede, além de métodos automáticos capazes de coletar dados a respeito desses problemas.

O RMON tornou-se um padrão por volta de 1990. O RMON II foi publicado recentemente para estender as capacidades do RMON, cujo padrão está disponível nas RFCs (*Request for comments*) 1757 e 1531, e apresentam padrões para redes Ethernet e Token Ring. A RFC 1757 define o padrão RMON de gerenciamento. Segundo a RFC, o RMON não é uma pilha de protocolos, nem um protocolo por si só. Na realidade, trata-se de uma extensão de MIB, para ser utilizada com protocolos de gerenciamento de rede em redes baseadas em TCP/IP (LESSA, 1999).

O RMON permite a implementação de um sistema de gerenciamento distribuído, atribuindo aos diferentes elementos da rede a função de monitor remoto (GIMENEZ, 2004). Cada elemento RMON tem a função de coletar, filtrar, analisar as informações de gerenciamento de rede e notificar as estações gerentes eventos significantes e situações de erro. A Figura C.5 (GIMENEZ, 2004): mostra uma estrutura de redes com gerenciamento RMON.

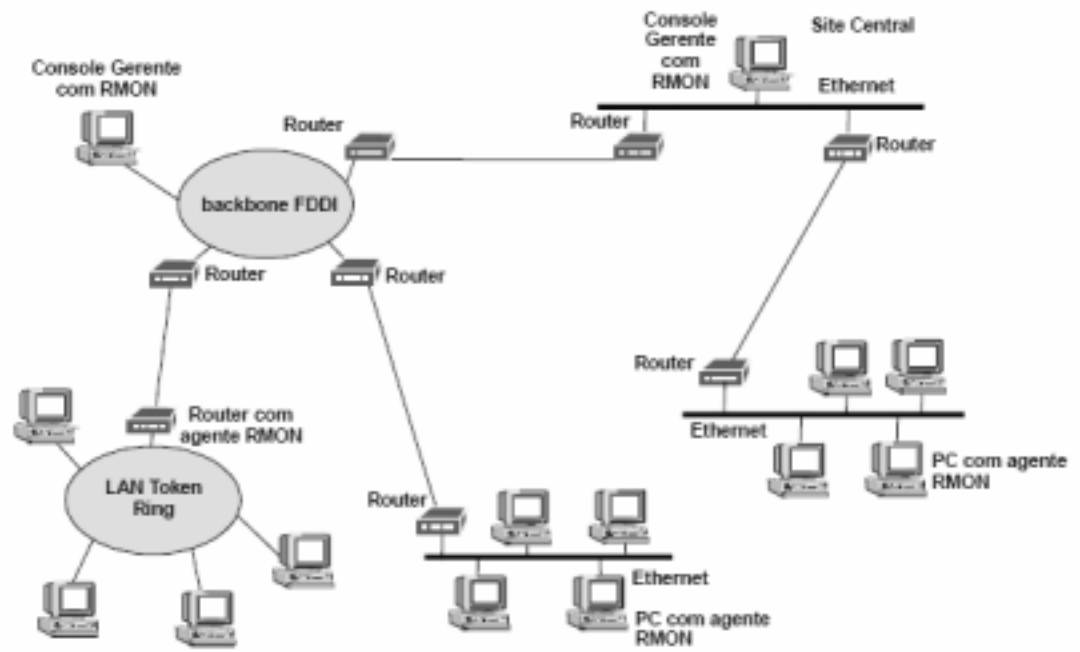

Figura C.5 – Estrutura de redes com gerenciamento RMON

Os objetivos do RMON podem ser resumidos em (LESSA, 1999) (GIMENEZ, 2004):

Operação Off-line

Existem situações em que uma estação de gerenciamento não estará em contato contínuo com seus dispositivos de gerenciamento remoto. Esta situação pode ocorrer como conseqüência de projeto, a fim de que se reduzam os custos de comunicação, ou por falha da rede, quando a comunicação entre a estação de gerenciamento e o monitor fica comprometida em sua qualidade. Por esta razão, a MIB RMON permite que um monitor seja configurado para realizar suas atividades de diagnóstico e coleta de dados estatísticos continuamente, mesmo quando a comunicação com a estação de gerenciamento seja impossível ou ineficiente. O monitor poderá então comunicar-se como a estação de gerenciamento quando uma condição excepcional ocorrer. Assim, mesmo em circunstâncias em que a comunicação entre monitor e estação de gerenciamento não é contínua, as informações de falha, desempenho e configuração podem ser acumuladas de forma continua, e transmitidas à estação de gerenciamento conveniente e eficientemente quando necessário.

Monitoramento Pró-ativo

Dados os recursos disponíveis no monitor, é normalmente desejável e potencialmente útil que ele execute rotinas de diagnóstico de forma contínua e que acumule os dados de desempenho da rede. O monitor estará sempre disponível no início de uma falha. Deste modo, ele poderá notificar a estação de gerenciamento da falha, assim como armazenar informações estatísticas a seu respeito. Esta informação estatística poderá ser analisada pela estação de gerenciamento numa tentativa de diagnosticar as causas do problema.

Detecção e Notificação de Problemas

O monitor pode ser configurado para reconhecer condições, que, normalmente, são de erro e verificar pelas mesmas continuamente. No advento de uma destas condições, o evento pode ser registrado e as estações de gerenciamento notificadas de várias formas.

Valor Agregado aos Dados

Considerando o fato de que os dispositivos de gerenciamento remoto representam recursos dedicados exclusivamente a funções de gerenciamento, e considerando também que os mesmos localizam-se diretamente nas porções monitoradas da rede, pode-se dizer que estes dispositivos permitem a agregação de valor aos dados coletados. Por exemplo, indicando quais os *hosts* que geram a maior quantidade de tráfego ou erros, um dispositivo pode oferecer (à estação de gerenciamento) informações preciosas para a resolução de toda uma classe de problemas.

Gerenciamento Múltiplo

Uma organização pode ter mais de uma estação de gerenciamento para as várias unidades da empresa, para funções distintas, ou como tentativas de proporcionar recuperação em caso de falha (*crash recovery*). Como tais ambientes são comuns na prática, um dispositivo de gerenciamento de rede remoto deverá ser capaz de lidar com múltiplas estações de gerenciamento concorrendo para a utilização de seus recursos.