Ana Paula Magalhães Dumont

AGAD - Uma Arquitetura de Gerenciamento Ativo Distribuído

Instituto de Matemática – Mestrado em Informática

Luci Pirmez
D.Sc. – COPPE/UFRJ – Brasil – 1996

Luiz Fernando Rust da Costa Carmo
Docteur, Universite Paul Sabatier/LAAS – Franca – 1995

Rio de Janeiro
2002

AGAD: Uma Arquitetura de Gerenciamento Ativo Distribuido

Ana Paula Magalhães Dumont

Dissertação (Tese) submetida ao corpo docente do Instituto de Matemática da Universidade Federal do Rio de Janeiro – UFRJ, como parte dos requisitos necessários à obtenção do grau de Mestre.

Aprovada por:

_____
Prof. Luci Pirmez, D. Sc.

_____
Prof. Luiz Fernando Rust da Costa Carmo,
Dr. UPS, France

_____
Prof. Vitório Bruno Mazzola, Dr, UPS

_____
Prof. Aloysio de Castro Pinto Pedroza, Dr, UPS

_____
Prof. Ageu Cavalcante Pacheco Junior, D. Sc.

Rio de Janeiro - RJ
Outubro de 2002

FICHA CATALOGRÁFICA

DUMONT, ANA PAULA MAGALHÃES.

Uma Arquitetura de Gerenciamento Ativo Distribuído [Rio de Janeiro] 2002

xi, 97 p., 29,7 cm (IM/NCE/UFRJ, MSc., Informática, 2001)

Dissertação (Mestrado) - Universidade Federal do Rio de Janeiro, IM/NCE

1. Gerência de Redes 2. Gerenciamento Distribuído 3. Tecnologia Ativa

I. IM/NCE/UFRJ II. Título (série)

A toda a minha família,e em especial
aos meus pais Renato e Lucinda

e
aos meus orientadores Luci e Rust

AGRADECIMENTOS

A Deus, pela minha vida.

Aos meus pais, que sempre me amaram, me deram muita forca e se empenharam em me educar, permitindo que eu chegasse até aqui.

A meus avós que, mesmo preocupados, apoiaram minha decisáo e me ajudaram me dando muito amor e carinho.

Aos meus amigos que fiz durante o curso de Mestrado, Cecílio, Reinaldo, Renata, Alexandre, Roberta, Noel, Patrícia, Karina, Cesar, Sidney, dentre muitos, muitos outros, com os quais dividi vários momentos e que sempre estiveram por perto nas horas em que deles precisei.

Aos meus orientadores, Luci e Rust, pela determinacao, cuidado e carinho com que orientaram o desenvolvimento deste trabalho, bem como o meu crescimento como pessoa, e aos professores do Mestrado em Informática do IM/NCE, pelos milhares de conhecimentos passados durante todo o curso.

Ao Núcleo de Computacao Eletrônica da Universidade Federal do Rio de Janeiro, por toda infra-estrutura disponibilizada durante o curso para o desenvolvimento deste trabalho.

Resumo da Tese apresentada ao IM/NCE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AGAD: Uma Arquitetura de Gerenciamento Ativo Distribuído

Ana Paula Magalhães Dumont

Outubro/2002

Orientadores: Profª. Luci Pirmez
Prof. Luiz Fernando Rust da Costa Carmo

Departamento: Informática

Este documento apresenta uma proposta de uma Arquitetura de Gerenciamento Ativo Distribuído (AGAD) que tem por objetivo prover uma infra-estrutura baseada em tecnologia ativa, que permita o gerenciamento distribuído de uma rede dividida em um ou mais domínios administrativos.

Um domínio é composto por um ou mais segmentos de rede local ao qual são interligados estações, roteadores, comutadores, hubs, etc.

As três grandes vantagens que o esquema de gerenciamento da AGAD trouxe em relação ao gerenciamento centralizado do SNMP são: uma redução significativa no tráfego de informações de gerência na rede em relação ao esquema de *polling* do SNMP, redução dos retardos associados às ações de gerenciamento e flexibilidade na atualização ou na introdução, de forma dinâmica, de novas funcionalidades de gerenciamento.

Estas vantagens estão diretamente relacionadas à possibilidade de um componente gerenciador ser executado no próprio elemento gerenciado.

Abstract of Thesis presented to IM/NCE/UFRJ as a partial fulfillment of the requirements for degree of Master of Science (M.Sc.)

AGAD: A Distributed Active Management Architecture

Ana Paula Magalhães Dumont

October/2002

Advisors: Prof[a.] Luci Pirmez
Prof. Luiz Fernando Rust da Costa Carmo

Department: Informatics

This document proposes a Distributed Active Management Architecture (AGAD). The main goal is to enhance the management capabilities by providing an infrastructure based on active technology to deal with escalability issues or problems that comes up with the centralized SNMP scheme. The architecture was structered based on administrative domains wich is composed of network devices like stations, routers, switches, hubs, etc.

The three main advantages of the AGAD management scheme are: considerable reduction in the management information traffic in the network, reduction of management actions delays and flexibility in updating new functionalities on the fly.

These advantages are directly related to the possibility of a manager component being executed in the proper managed element.

SUMÁRIO

Capítulo 1 - Introdução 1

Capítulo 2 – Conceitos Básicos 4

2.1. Gerência de Redes 4

2.2. Gerenciamento centralizado versus Gerenciamento distribuído 5

2.3. Modelos de Gerenciamento 6
2.3.1. Gerenciamento OSI 6
2.3.2 Gerenciamento Internet 8
2.3.3 Diferenças entre SNMP e CMIP 13

2.4. Tecnologia Ativa 14
2.4.1 Redes Ativas 14
2.4.2 Agentes móveis 16
2.4.3 Redes Ativas X Agentes Móveis 17
2.4.4. μCode 18

2.5 Tecnologia ativa no gerenciamento distribuído 19

2.6. Trabalhos Relacionados 20

2.7. Considerações Finais do Capítulo 21

Capítulo 3 - Arquitetura AGAD 23

3.1. Descrição Geral da Arquitetura 24

3.2. Componentes da AGAD 27
3.2.1. Gerente-Mor 28
3.2.2. Gerente de Domínio 32
3.2.3. Inspetores 35
3.2.4. Especialistas 38
3.2.5. Guardião 40

3.3. Comunicação entre os elementos da AGAD 44
3.3.1. Mensagens de controle 44
3.3.2. Mensagens de alarme 44
3.3.3 Mensagens de relatório 44

3.4. Mecanismo de distribuição de código 45

3.5. Considerações Finais do Capítulo 46

Capítulo 4 - Ambiente de Desenvolvimento e Implementação 48

4.1. Ambiente de Desenvolvimento 48
4.1.1. A Linguagem Java 49

4.1.2. Ambiente de Mobilidade utilizado 50
4.1.3. Limitações do μCode 52
4.1.4 API SNMP (AdventNet) 52

4.2. Implementação 53
4.2.1 Classes implementadas no protótipo 53
4.2.2. Diagrama de Seqüência 64
4.2.3 Bases de dados 65
4.2.4. Técnicas utilizadas na implementação do protótipo 66

4.3. Considerações Finais do Capítulo 69

Capítulo 5 - Testes e análise dos resultados obtidos 71

5.1. Latência de carga da arquitetura AGAD 72

5.2 Latência de detecção de um Alarme de Falha 74

5.3. Latência entre a detecção de um alarme de falha e o início da execução do Especialista 76

5.4. Considerações finais do capítulo 79

Capítulo 6 - Conclusões e Trabalhos Futuros 80

6.1. Considerações Finais dos resultados obtidos 80

6.2. Trabalhos Futuros 81

Referências Bibliográficas 85

LISTA DE TABELAS

Tabela 1 - Plataformas utilizadas nos testes 71

Tabela 2 – Medições do 1º teste (em ms) 73

Tabela 3 - Medições do 2º teste referentes ao 1º cenário (em ms) 74

Tabela 4 - Medições do 2º teste referentes ao 2º cenário (em ms) 75

Tabela 5 - Medições do 3º teste (em ms) 78

Capítulo 1 - Introdução

As redes e os sistemas de processamento distribuído estão crescendo em importância e, por conseguinte, tornaram-se críticos no mundo dos negócios. Nos ambientes empresariais, a tendência é por redes maiores e mais complexas, que aceitem um maior número de usuários e aplicações. À medida que essas redes crescem em escala, dois fatos vão ficando mais evidentes: as redes, juntamente com seus recursos e aplicações distribuídas, tornam-se cada vez mais indispensáveis para as organizações e com o crescimento das redes, existe conseqüentemente uma maior possibilidade de ocorrerem problemas, o que pode levá-las a um estado de inoperância ou a níveis inaceitáveis de desempenho.

A complexidade destas redes e dos seus equipamentos faz com que o treinamento de profissionais para o seu gerenciamento seja muito dispendioso. A distância física entre os equipamentos, que pode variar desde alguns centímetros até dezenas de milhares de quilômetros, é um potencializador destes problemas. É natural que o proprietário deseje contar com ferramentas que o auxiliem a exercer sua administração de uma forma mais automática, se possível com um mínimo de intervenção humana.

A primeira iniciativa de criação de sistemas de gerência partiu de um grupo de estudos da OSI [1]. O objetivo era definir uma infra-estrutura de gerenciamento padrão para componentes de rede (roteadores, pontes, etc). O projeto, porém, era muito ambicioso e complexo. Uma força-tarefa foi criada para desenvolver uma outra infra-estrutura mais simplificada para ser usada enquanto o primeiro projeto não ficasse pronto. Uma outra iniciativa de criação de um sistema de gerenciamento surgiu com o modelo de gerenciamento Internet, e a criação do protocolo SNMP (*Simple Network Management Protocol*) [19] que acabou se firmando como padrão de fato para o gerenciamento de equipamentos de rede. Entretanto, peca ele pela falta de escalabilidade. Cada equipamento deve ser gerenciado separadamente, suas mensagens acabam por sobrecarregar a rede ao tentar transferir massas de dados muito extensas. Além disto, o elemento gerenciador – o agente SNMP – é incapaz de tomar decisões para manter o funcionamento do sistema sem a intervenção de um operador humano.

Atualmente, a flexibilidade na inclusão e atualização de novos serviços de forma dinâmica, a busca pela independência de plataforma, redução de tráfego, entre outras vantagens tornaram-se possíveis com o advento da utilização da tecnologia ativa.

A tecnologia ativa engloba dois paradigmas que estão sendo considerados na área de gerenciamento de redes: redes ativas e agentes móveis. Ambos oferecem o suporte a modelos que utilizam recursos computacionais no interior e/ou nas bordas da rede para o carregamento e a execução de programas "sob demanda". Dessa forma, novos tipos de aplicações de controle e/ou gerenciamento, assim como novos serviços, podem ser implementados rapidamente.

O Smart Packets [38] é um sistema de gerenciamento distribuído que utiliza somente o paradigma de redes ativas e que tem a finalidade de distribuir tarefas de gerenciamento no interior da rede.

Outro projeto que utiliza os dois paradigmas simultaneamente é o sistema de gerenciamento distribuído para redes IP chamado ADM (Active Distributed Management for IP Networks) [39] desenvolvido na linguagem Java.

Este trabalho propõe uma arquitetura de gerenciamento distribuído utilizando tecnologia ativa chamada AGAD. Essa arquitetura utiliza apenas o paradigma de agentes móveis. O diferencial desse trabalho em relação aos outros de gerenciamento que utilizam tecnologia ativa, descritos em detalhe na seção 2.6 são: a inclusão de mecanismos de filtragem; a consolidação de informações no nó gerenciado, reduzindo o tráfego de gerenciamento; a facilidade de atualização de funcionalidades com a implementação de componentes de software e a viabilização de independência de plataforma. Com essas vantagens, há uma otimização no uso dos recursos computacionais como memória, CPU e banda passante.

Esta dissertação está dividida em seis capítulos. O capítulo 2 fornece uma introdução à gerência de redes, citando as diferenças entre o gerenciamento centralizado e o distribuído, os modelos de gerenciamento, o conceito de tecnologia ativa e o gerenciamento de redes, utilizando tecnologias ativas. Por fim, há uma descrição mais detalhada dos trabalhos relacionados. O capítulo 3 apresenta a arquitetura AGAD detalhando o seu funcionamento e os seus elementos. O ambiente de desenvolvimento e os detalhes de implementação do sistema são tratados no capítulo 4. O capítulo 5 traz

uma análise dos resultados obtidos e, finalmente, as conclusões e sugestões de trabalhos futuros são relatados no capítulo 6.

Capítulo 2 – Conceitos Básicos

A arquitetura de gerenciamento ativo, AGAD, que está sendo proposta neste trabalho baseia-se em conceitos de gerenciamento e tecnologia ativa.

Este capítulo tem por objetivo apresentar cinco conceitos básicos para a compreensão da AGAD. A seção 2.1 aborda conceitos relacionados a gerência de redes, a 2.2 apresenta o gerenciamento de redes centralizado versus distribuído, a 2.3 descreve os modelos de gerenciamento OSI e Internet, a seção 2.4 apresenta o conceito de tecnologia ativa e a seção 2.5 mostra o uso de tecnologia ativa no gerenciamento de distribuído.

2.1. Gerência de Redes

A gerência de redes pode ser conceituada como a coordenação dos recursos de hardware (modens, roteadores, etc) e de software (ex: protocolos de rede), fisicamente distribuídos pela rede através de ferramentas que permitem detectar problemas que venham a ocorrer nesses recursos. Essas ferramentas devem ser dotadas de mecanismos de monitoramento e de controle dos elementos de rede a fim de permitir seu funcionamento dentro de uma qualidade de serviço (QoS), acordada com os usuários. Os procedimentos de monitoração devem detectar irregularidades na operação da rede antecipadamente, resolvendo-as antes que gerem prejuízos operacionais para a empresa.

O gerenciamento pode ajudar na administração, no planejamento e na expansão da rede, pois permite:

- enviar alertas;
- maximizar a eficiência e a produtividade da rede;
- monitorar os recursos da rede e atender rapidamente às necessidades dos usuários;
- antecipar problemas;
- efetuar análises de desempenho e segurança;
- configurar os componentes da rede (hardware e software) de forma distribuída;

- detectar anomalias e isolar problemas, facilitando o diagnóstico para solução;
- analisar a performance e os limites de capacidade da rede;
- ativar sistemas de segurança e controle de acesso à rede.

A gerência de redes é uma aplicação distribuída devido a natureza distribuída dos recursos a serem gerenciados. Os processos usados no gerenciamento distribuído destes podem ser classificados como processo gerente ou processo agente.

Um processo gerente é o módulo de uma aplicação distribuída que tem responsabilidade de uma ou mais atividades de gerenciamento, ou seja, ele transmite operações de gerenciamento (*actions*) aos agentes e recebe notificações (*events*) destes.

Um processo agente é o módulo de uma aplicação distribuída responsável por executar as diretivas enviadas pelo processo gerente, como também fornecer ao gerente uma visão dos recursos de rede e emitir notificações sobre os mesmos.

2.2. Gerenciamento centralizado versus Gerenciamento distribuído

A abordagem centralizada de gerenciamento (SNMP/ CMIP) [18] é baseada na execução de *pollings* por uma estação de gerenciamento de rede (NMS – *Network Management Station*) a agentes instalados (permanentemente) em componentes gerenciados. A NMS constrói então uma visão do estado da rede e gera alarmes quando problemas são detectados. A NMS pode também enviar comandos aos agentes para a resolução desses problemas. Essa abordagem requer a coleta e a análise de grandes quantidades de dados antes que ações sejam desencadeadas. Essas ações, por sua vez, devem atravessar diversas camadas hierárquicas de protocolos.

À medida que o número de elementos gerenciados cresce, os requisitos computacionais da estação de gerência e o consumo de banda da rede gerenciada também aumentam. Ainda, em redes com grande abrangência geográfica a presença de congestionamento em um ou mais pontos da rede e a distância da NMS aos elementos gerenciados, pode ocasionar retardos na comunicação entre eles. Algumas opções de implementação de uma abordagem distribuída têm sido propostas nos últimos anos, desde gerenciamento por delegação [4] até o uso de agentes móveis [3]. Na maioria das soluções, as tarefas de gerenciamento que, antes, eram somente de responsabilidade da estação central, são distribuídas entre os agentes de software ou objetos remotos.

As opções mais simplificadas de gerenciamento distribuído dividem a rede em domínios hierárquicos de gerenciamento, cada qual com sua NMS, delegando atividades de monitoramento e certas capacitadas de filtragem aos próprios elementos gerenciados, que enviam apenas alarmes ou relatórios resumidos às estações de gerenciamento. Exemplos dessas abordagens são as funções de monitoramento de métricas e sumarização do modelo OSI de gerenciamento [9] e RMON no modelo SNMP [10] .

Opções mais avançadas de gerenciamento distribuído utilizam-se de paradigmas de objetos distribuídos como Corba [52] [60] e Java-RMI [60] ou da chamada remota de procedimentos (RPC) [60]. Entretanto, Corba, DCOM e RMI são muito úteis no projeto e construção de sistemas distribuídos, mas escondem os verdadeiros custos de suas implementações. Como resultado, o uso dos recursos da rede pode se dar de forma ineficiente, principalmente no que diz respeito ao consumo de banda dos enlaces. Outra desvantagem dessas abordagens é que não oferecem o suporte a ambientes verdadeiramente distribuídos, onde agentes podem se comunicar com vizinhos para execução de tarefas de forma distribuída e eficiente. A localização desses agentes também é problemática. Eles devem, normalmente, ser executados em estações, pois são aplicações. Assim, os retardos devido à comunicação entre esses agentes e roteadores ou comutadores da rede podem ser proibitivos. Os melhores locais para esses agentes funcionarem são (também) nos próprios roteadores e comutadores, onde a maior parte das informações de gerenciamento é gerada.

2.3. Modelos de Gerenciamento

Os dois principais modelos de gerenciamento são o OSI e o Internet. O modelo OSI utiliza o protocolo CMIP [59] e o modelo Internet utiliza os protocolo: SNMP [59] e o RMON [25]. A descrição dos modelos OSI e Internet e suas principais funcionalidades são apresentados a seguir.

2.3.1. Gerenciamento OSI

O modelo de gerenciamento de redes definido pela OSI [1] foi a primeira iniciativa para definir uma infra-estrutura de gerenciamento padrão. Neste modelo, a gerência de redes é dividida em cinco áreas funcionais descritas em seguida.

- **gerência de falhas** - tem por objetivo proporcionar um funcionamento contínuo da rede e de seus serviços. É composta por um conjunto de funções destinadas à detecção, diagnóstico e correção ou isolamento de anomalias no funcionamento da rede, quer sejam provocadas por um problema de hardware ou de software.
- **gerência de configuração** - tem por objetivo o gerenciamento dos serviços de interconexão de sistemas abertos. Essa gerência é responsável por um conjunto de funções relativas ao controle, à identificação, à supervisão e à coleta de informações sobre os objetos gerenciados como, por exemplo, a reconfiguração de um roteador devido a um tráfego intenso.
- **gerência de desempenho** - possibilita a avaliação do comportamento dos recursos de rede, tentando garantir a qualidade de serviço acordada com o usuário. Essa gerência fornece um conjunto de funções relativas à coleta de dados estatísticos e a manutenção do histórico destes recursos, com o objetivo de planejar e analisar o sistema.
- **gerência de contabilização** - tem como objetivo a determinação do custo de utilização da rede e dos recursos nela disponíveis. Essa gerência define, para os recursos que podem ser faturados, o custo relativo a sua utilização e a combinação destes custos no caso de vários recursos serem solicitados pelo usuário como, por exemplo, o custo mensal do serviço de comutação de pacotes.
- **gerência de segurança** - corresponde ao conjunto de funções necessárias a criação, supressão e controle dos mecanismos e serviços de segurança e do tráfego de informações sigilosas na rede. Assim, ela suporta a política OSI relacionada ao bom funcionamento da gerência OSI, bem como a proteção dos recursos suportados por esta gerência. Ex: Proteção de senhas com codificação criptografada.

O modelo CMIP (*Common* Management *Information Protocol*) [59] é o padrão OSI para o gerenciamento de redes. O CMIP é um modelo mais poderoso (abrangente) de gerência que o SNMP. Além disso, o fato do CMIP ser um padrão internacional definido pela OSI permite que os fabricantes testem suas implementações através de testes de conformidade e de interoperabilidade.

No modelo CMIP, o propósito de um agente é fornecer informações do objeto gerenciado para o gerente. O gerente envia uma mensagem CMIP para um objeto localizado em um determinado agente. É função do agente decodificar a mensagem e determinar que informações necessitam ser recuperadas ou modificadas.

Um gerente CMIP genérico fornece um mapeamento entre objetos gerenciados CMIP e a representação de dados interna à aplicação. É função do agente fornecer um mapeamento entre a forma interna da informação e a representação externa, face aos objetos gerenciados. A relação gerente/agente não é necessariamente um-a-um, embora um dado objeto esteja associado a um único agente. A um agente podem estar associados vários gerentes, sendo a recíproca falsa.

2.3.2 Gerenciamento Internet

O sistema de gerência Internet possui a seguinte filosofia: o impacto de adicionar-se atividades de gerência sobre o sistema gerenciado deve ser o menor possível. Assim, o gerenciamento Internet é baseado num protocolo simples denominado SNMP (*Simple Network Management Protocol*).

Atualmente, a maioria das plataformas de gerenciamento está baseada no modelo SNMP desenvolvido dentro da arquitetura TCP/IP. O modelo SNMP evoluiu à medida que necessidades adicionais de gerenciamento foram sendo identificadas na Internet como:

- gerenciar de fato uma internet, e não somente monitorá-la;
- gerenciar outros tipos de dispositivos, tais como servidores de banco de dados, impressoras, fontes de energia, *bridges, hubs*, e não somente roteadores;

Para que essas necessidades fossem atendidas, a especificação SNMP define algumas entidades chaves. O gerenciador central é uma aplicação que recebe as informações dos agentes de gerenciamento e as apresenta ao usuário. Os agentes de gerenciamento são módulos de software que permitem a recuperação de informações SNMP.

O modelo SNMP consiste em quatro componentes, descritos em seguida.

- **nós gerenciados** - onde estão implementados os agentes SNMP. O agente SNMP é responsável em atender as requisições de gerenciamento oriundas

do gerente. Cada agente possui uma MIB (*Management Information Base*) [8], onde são mantidas as informações pertinentes ao gerenciamento. A estrutura da MIB é padronizada, de forma que os objetos gerenciáveis possam ser identificados unicamente e tenham uma sintaxe definida. As MIBS concentram os dados coletados do sistema operacional pelos agentes SNMP.

- **protocolo de gerenciamento** – o protocolo utilizado é o SNMP (*Simple Network management Protocol*) que define um conjunto de regras e formatos que tem que ser usado na construção das mensagens para que os agentes e o gerente possam se comunicar.
- **estações de gerenciamento** - o gerenciamento de rede é feito a partir de estações de gerenciamento. Estas estações contêm um ou mais processos que se comunicam com os agentes espalhados pela rede, emitindo comandos e obtendo respostas.
- **informações de gerenciamento** – para permitir que uma estação de gerenciamento se comunique com todos esses componentes tão diversificados, a natureza das informações mantidas por todos os dispositivos deve ser rigidamente especificada. O SNMP descreve (com riqueza de detalhes) as informações exatas que cada tipo de agente deve manter e o formato a ser aplicado a essas informações. A maior parte do modelo SNMP se refere à definição de quem deverá acompanhar o que e ao modo como essa informação será comunicada.

A versão 1 do protocolo de gerenciamento SNMP contém 5 PDUs; as 3 primeiras são iniciadas pelo gerente e as 2 últimas são iniciadas pelo agente.

- **GetRequest** - utilizada para obter uma instância de um objeto na MIB do agente. É necessário especificar o identificador do objeto (OID) completamente, inclusive o índice, caso contrário é retornado um erro.
- **GetNextRequest** - ao fornecer qualquer OID, o agente retorna o próximo objeto disponível na MIB. Os objetos são organizados na MIB em função do seu OID. Esta PDU permite a navegação através da MIB, mesmo sem saber a sua composição completa.

- **SetRequest -** altera um valor de um atributo de um objeto gerenciado caso o tipo de acesso do objeto permita a escrita. Para tanto, a PDU tem como argumentos: o identificador do elemento de rede a ser gerenciado sobre o qual a alteração será feita, a identificação do objeto gerenciado e o novo valor do atributo afetado. O agente SNMP ao receber uma requisição, consulta a *community,* uma espécie de autenticação, que ele tem armazenado verificando se é a mesma que está presente na requisição. O conceito de *community* foi definida por razões de segurança. Ela é definida nos agentes para autenticar os gerentes que podem fazer alguma operação SNMP nos mesmos. Ao requisitar qualquer informação de um agente, o gerente deve indicar na mensagem SNMP uma *community*.
- **G*etResponse* -** contém as respostas do agente às requisições efetuadas pelo gerente.
- ***Trap*** - PDU assíncrona, emitida pelo agente quando ocorre uma situação anormal que se produz sobre os Objetos Gerenciados. Assim, se um problema acontece em um nó gerenciado, o agente envia imediatamente um comando *trap* ao Gerente sem esperar que este lhe solicite informações. Por exemplo, um agente pode ser programado para enviar ao Gerente uma mensagem de alarme sempre que houver a perda de uma conexão.

A versão 1 do protocolo SNMP, apresenta várias deficiências como as descritas a seguir.

- Baixa segurança, em função da *community* trafegar pela rede sem nenhum tipo de criptografia, limitando assim o uso do SNMP somente para monitoração de variáveis.
- Baixa escalabilidade, pois torna-se problemático quando usado em redes de grande porte, já que opera em cima de requisições freqüentes a variáveis da MIB. Esse mecanismo pode causar congestionamento na rede.
- Ineficiência para aquisição de grandes quantidades de informação. É gerada uma requisição para cada variável consultada. A pesquisa de tabelas muito grandes gera um grande volume de tráfego.
- Padronização limitada para redes IP.
- Inexistência de comunicação entre as estações de gerência.

Para suprir essas deficiências da **primeira** versão do SNMP, foi proposta pelo IETF (*Internet Engineering Task Force*) o SNMPv2 [19]. Esta versão permite uma comunicação segura entre duas estações gerentes e entre os dispositivos gerenciados. O SNMPv2 faz a coleta de grandes volumes de dados gerenciados mais eficiente que a versão anterior e possui melhores mecanismos para os tratamentos de erros. As inovações da versão 2 do SNMP e as novas PDUs que foram criadas são descritas a seguir.

- A Estrutura da Informação Gerenciada (SMI) do SNMPv2 expande a SMI do SNMPv1 para incluir novos tipos de dados e melhorar a documentação associada a objetos. A SMI é agora dividida em quatro partes: definição de objetos, tabelas que proíbem a criação e a remoção de linhas pelo gerente e tabelas que permitem a criação e a remoção de linhas pelo gerente, e definições de notificações e módulos de informação.
- Melhor desempenho no acesso a tabelas, com a criação da PDU *GetBulkRequest*. Esta PDU permite recuperar uma grande quantidade de informações do agente através de uma única mensagem SNMP.
- Comunicação gerente-gerente, através da definição de uma MIB específica para este fim, e de uma nova PDU, a *InformRequest* , que permite notificar eventos pré-definidos entre os gerentes.
- Mecanismos de segurança que permitam a autenticação e a privacidade de uma mensagem SNMP. Esses mecanismos garantem que uma mensagem SNMP não seja alterada, retardada ou repetida no percurso, que seja feita a identificação correta do usuário que enviou a mensagem, e que esta possa ser resguardada contra escuta clandestina.
- Permite usar mecanismos de transportes alternativos ao TCP/IP, tais como o do OSI [7], IPX [18] e do AppleTalk [18] .
- O calcanhar-de-aquiles do SNMPv2 é sua incompatibilidade com os agentes da versão anterior, o que torna a transição de uma versão para outra mais custosa.

Em 1998 foi padronizado a versão 3 do protocolo SNMP que surgiu em função dos problemas de segurança das versões anteriores (principalmente em relação ao comando set) e com o objetivo de viabilizar a padronização de partes da arquitetura cujo

consenso não tenha sido atingido nas versões anteriores. As novas características que o SNMP versão 3 trouxe à versão 2 do SNMP são:

- um novo formato de mensagem SNMP;
- aprimoramento na estrutura do gerente e do agente SNMP;
- novos e flexíveis métodos de segurança para as mensagens (criptografia das mensagens);
- acesso controlado para objetos gerenciados;
- serviço para envio e recebimento de autenticações de mensagens.

Um outro padrão proposto pelo IETF (*Internet Engineering Task Force)* é o padrão RMON (Remote Monitoring) [25]. O RMON surgiu com a necessidade de monitoração da rede como um todo, em vez de componentes individuais, como roteadores, hosts e outros dispositivos. Este padrão foi desenvolvido para ajudar a entender o funcionamento da própria rede, e como os dispositivos individuais afetam a rede como um todo. É possível utilizá-lo para monitorar não somente o tráfego de uma LAN (*Local Area Network*), como também as interfaces de WAN (*Wide Area Network*).

O RMON define uma MIB-II [25] de monitoramento remoto que complementa a MIB-I e permite uma gerência de redes com informações vitais sobre inter-redes. Para redes Ethernet existem dois padrões de RMON: o RMON1 [18] e o RMON2 [18]. O RMON1 efetua a análise dos quadros a nível MAC, enquanto o RMON2 analisa as informações ao nível de rede e de aplicação.

O RMON possui um efetivo e eficiente esquema de monitorar o comportamento de uma sub-rede reduzindo a sobrecarga nos agentes e nas estações de gerenciamento. Ele funciona através de monitores de rede *(probes)* instalados num segmento de rede. Esta funcionalidade pode ser implementada em uma estação dedicada ou não, ou em um elemento de rede como um hub, switch ou roteador. Esses *probes* operam em modo promíscuo capturando todos os quadros que passam no segmento. A partir das informações obtidas destes quadros são geradas diversas estatísticas, as quais são armazenadas na MIB RMON. Uma estação de gerência pode requisitar estas informações do *probe* através do protocolo SNMP.

Pela monitoração do tráfego de pacotes e pela análise dos cabeçalhos, os *probes* fornecem informações sobre enlaces (*links*), conexões entre as estações, modelos de

tráfego e status dos nós da rede. Um monitor pode passivamente (sem *polling*) reconhecer certas condições de erro, tais como o congestionamento no enlace que estiver sendo observado. Quando uma dessas condições de erro ocorrer, o monitor pode registrar o erro e tentar notificar à estação de gerenciamento.

O padrão RMON não implica alteração alguma no protocolo SNMP, que é usado na comunicação entre os *probes* RMON e as estações de gerência.

2.3.3 Diferenças entre SNMP e CMIP

Tanto o SNMP (modelo de gerência Internet) quanto o CMIP (modelo de gerência OSI) têm, naturalmente, o mesmo objetivo: permitir o envio de comandos e receber resultados e notificações de tal modo a coordenar (monitorar e controlar) os recursos lógicos e físicos de uma rede. Mas existem várias diferenças entre os dois que podem ser resumidas em seguida.

- Acesso a dados: SNMP é orientado mais à recuperação individual de itens de uma informação, enquanto CMIP é mais orientado à recuperação de informações agregadas.
- "Polling" x "Reporting": o SNMP trabalha por "polling" (o gerente regularmente pergunta ao agente sobre seu "estado"), enquanto o CMIP usa "reporting" (o agente informa ao gerente quando o "estado" do objeto gerenciado mudou). Esta filosofia de gerenciamento do CMIP é mais vantajosa do que a do SNMP quando se tem um número considerável de agentes a serem consultados.
- Tamanho e desempenho: o SNMP é menor e mais rápido. O CMIP requer maior capacidade de processamento e mais memória.
- Nível de transporte: o SNMP exige um nível de transporte não orientado a conexão, isto é o uso de datagramas (UDP). Já o CMIP exige um nível de transporte orientado a conexão (TCP).
- Padronização: o CMIP, a exemplo, dos demais protocolos OSI, é um padrão internacional, sujeito portanto a testes de conformidade. O SNMP, por sua vez, somente pode dispor de testes de interoperabilidade.
- Um agente OSI é mais "inteligente" que seu equivalente agente SNMP necessitando assim mais espaço em disco e tempo de processamento. Por

outro lado, o volume de tráfego de gerência na rede OSI é bem menor do que a da Internet. Um agente Internet é incapaz de solucionar problema, limitando-se a simplesmente sinalizar ao gerente a ocorrência dos mesmos. Já um agente OSI é um agente ativo, capaz de filtrar informação, só gerando tráfego de gerência, por exemplo, quando o estado de um objeto mudar.

2.4. Tecnologia Ativa

A partir de 1995, após as primeiras publicações oriundas de pesquisa financiada pelo DARPA, a programabilidade dos nós de comutação de uma rede passaram a ser seriamente investigadas. O objetivo dessas investigações foi prover uma maior flexibilidade, na disponibilização rápida de novos serviços em redes de comunicação [16], evitando a espera pelos longos processos de padronização de protocolos e de formato de pacote.

Atualmente, dois paradigmas estão sendo considerados: redes ativas e agentes móveis. Ambos oferecem o suporte a modelos que utilizam recursos computacionais no interior e/ou nas bordas da rede para o carregamento e a execução de programas "sob demanda". Dessa forma, novos tipos de aplicações de controle e/ou de gerenciamento, assim como novos serviços, podem ser implementados rapidamente. Embora os conceitos dessas tecnologias tenham sido originadas em diferentes comunidades de pesquisas, visando resolver diferentes problemas, eles começam a se sobrepor em termos de foco e aplicabilidade, fato que está popularizando o termo tecnologia ativa para as referências a um ou ambos os paradigmas [27][57].

2.4.1 Redes Ativas

O objetivo do desenvolvimento de redes ativas é aumentar a flexibilidade da rede de modo que esta acomode rapidamente os requisitos específicos de QOS das novas aplicações. Aplicações de controle de congestionamento[15], *multicasting*[44] e gerenciamento de redes[45] podem tirar grande proveito no uso do paradigma de redes ativas.

Redes ativas permitem o desenvolvimento de novos serviços e protocolos dinamicamente uma vez que os pacotes chamados ativos contendo programas e dados são transportados entre os nós da rede (ex: roteadores). Além de programas e dados, os

pacotes podem transportar sinalizações que desencadeiem o carregamento desses programas.

Cada nó ativo é composto pelos seguintes elementos: um hardware computacional, um sistema operacional, um ou mais ambientes de execução e aplicações ativas que são executadas nesses ambientes de execução. Uma *workstation* com sistema operacional Linux e máquina virtual Java (ambiente de execução) pode ser considerado um nó ativo e as aplicações ativas seriam *bytecodes* Java executadas neste nó ativo.

Quanto a abordagem de funcionalidade, as redes ativas podem ser classificadas como: discreta, integrada e mista [26][57][51].

Nas arquiteturas que utilizam a abordagem integrada, o código executável é transportado nos próprios pacotes. Esse código é usualmente processado sobre os dados contidos no mesmo pacote. Nesse caso, a aplicação ativa é instalada imediatamente, na hora de sua utilização. Os pacotes ativos referentes a essa abordagem são chamados de cápsulas. Exemplos de projetos que utilizam essa abordagem são IP Option [29], ANEP [30] e Smart Packets [38].

Existem ainda arquiteturas que seguem a abordagem discreta. Nessa abordagem, o código executável é implementado em nós ativos. Os pacotes não transportam o código que será executado e sim referências e parâmetros de funções, que são carregados sob demanda nos nós. O que motivou o desenvolvimento desse tipo de arquitetura foram os problemas de desempenho, devido a sobrecarga de processamento que as arquiteturas com abordagem integrada estão sujeitas. O desempenho nesse caso, também é afetado pelo esforço despendido para tratar dos requisitos de segurança necessário para controlar o acesso aos recursos que os programas podem manipular. Uma alternativa para melhorar o desempenho é fazer a restrição na própria linguagem de programação, só permitindo a execução de operações consideradas seguras, tendo como conseqüência uma menor funcionalidade.

As arquiteturas que utilizam a abordagem mista permitem a utilização de ambas as abordagens citadas anteriormente, e tiram proveito das vantagens de cada uma delas.

2.4.2 Agentes móveis

Um agente que tem a capacidade de transportar o seu estado de execução de uma unidade de processamento para a outra é chamado agente móvel.

Os agentes móveis viabilizam a transformação das redes atuais em plataformas programáveis remotamente. As principais vantagens [33] no uso de agentes móveis em comparação com a abordagem cliente-servidor ou com outros tipos de agentes são: eficiência, economia de espaço, redução do tráfego, interação assíncrona e em tempo real, robustez, operação em ambientes heterogêneos, extensibilidade em tempo real e fácil atualização.

Uma característica importante do sistema baseado em agentes móveis é que neste tipo de sistema o intercâmbio de informações é reduzido pois seu processamento é local, economizando recursos de banda de transmissão. O agente móvel migra para um nó, procura acumular informação e processá-la localmente, para retornar com a informação já processada ao nó de origem.

Um sistema de agentes móveis é composto pelo código do agente, o estado de execução do agente, o contexto de execução, o mecanismo de deslocamento, e as permissões de acesso. O código é o programa que define o comportamento do agente. O estado define o valor dos dados internos do programa do agente, bem como o ponto de execução deste programa. O contexto de execução de um agente é a unidade de processamento que serve de sede para a execução do agente. Dentro de cada unidade de processamento é necessário que haja um serviço de mobilidade, possibilitando o deslocamento do agente de um contexto para o outro. O mecanismos de deslocamento está intrinsicamente associado ao serviço de mobilidade. Como agentes e contextos podem pertencer a domínios administrativos diferentes, é necessário definir quais são as permissões de acesso associadas a um agente dentro de um contexto. Essas permissões servem para disciplinar o acesso dos agentes aos recursos disponíveis no contexto.

Maiores informações sobre agentes móveis podem ser encontradas em [26] .

2.4.3 Redes Ativas X Agentes Móveis

Há uma grande semelhança entre as duas idéias. Na verdade, muitas das arquiteturas de redes ativas usam técnicas de mobilidade do código que são muito próximas da tecnologia de agentes móveis. A tecnologia de redes ativas é mais genérica do que a de agentes móveis em termos de encapsulamento de protocolos, configuração e instalação de serviços e manutenção. A fundamental diferença entre as duas tecnologias é que redes ativas usam o conceito de processamento na camada de rede, ou seja, voltado para o encaminhamento de pacotes, enquanto agentes móveis são executados na camada de aplicação.

Sistemas baseados em agentes móveis são projetados para a construção de um ambiente de computação distribuído e interligado por um sistema de comunicação, enquanto o propósito das redes ativas é disponibilizar e tornar mais eficientes facilidades no sistema de comunicação.

Pode-se considerar, no entanto, que os programas que são transportados juntamente com o fluxo de dados das redes ativas ou que são carregados sob demanda nos nós ativos são agentes móveis de software. Da mesma forma, um elemento de rede, como um nó de comutação, que seja capaz de executar um agente (móvel) pode ser considerado um nó ativo [33].

Quanto as infra-estruturas já implementadas para o suporte a redes ativas e agentes móveis, estas diferenciam-se ainda quanto à disponibilidade de funcionalidades de segurança e proteção do nó ativo. Essas funcionalidades estão presentes apenas nas infra-estruturas de redes ativas.

No gerenciamento de redes, pode-se considerar que as funcionalidades a serem executadas estão, principalmente, no nível das aplicações, uma vez que o gerenciamento não tem como objetivo principal tratar diretamente do roteamento e/ou do encaminhamento de pacotes. Mas também há a necessidade que sejam realizados certos processamentos no nível de rede. Sendo assim, no que refere-se a infra-estruturas para gerenciamento, convém empregar-se o termo tecnologia ativa, que engloba tanto os agentes móveis quanto as redes ativas.

2.4.4. µCode

O µCode[1] [34] é uma API desenvolvida em Java que fornece uma série de primitivas de mobilidade de código, e que foi concebida com o objetivo de superar algumas desvantagens encontradas nos sistemas de códigos móveis atuais, tais como o Java Aglets API [34], Mole [34] e Odyssey [34]. O µCode incorpora os verdadeiros benefícios que o paradigma de mobilidade de código trouxe para o projeto e implementação de aplicações distribuídas.

Em [34] foi definido um modelo analítico de tarefas de gerenciamento objetivando a comparação do modelo cliente-servidor em relação ao paradigma de código móvel na redução de tráfego na rede gerado pelo gerenciamento. Os resultados foram obtidos através da medição do desempenho de uma implementação do SNMPv1 [28] versus várias alternativas de código móvel implementadas com o Java Aglets API. Entretanto, as implementações de várias alternativas de código móvel usando o Java Aglets API evidenciaram limitações que motivaram o desenvolvimento do µCode e que são descritas em seguida.

Nesses sistemas de código móvel, uma das limitações é que as estratégias que regem a realocação de classes não estão completamente sob o controle do programador quando ele determina que um *aglet* deve ser movido de um nó a ouro. Para que essa operação seja bem sucedida, a classe do *aglet* deve satisfazer várias restrições de configuração, as quais não permitem uma atualização dinâmica de forma fácil. Uma outra limitação é a não possibilidade de transferir a um nó destino todas as classes necessárias por um *aglet* de uma só vez, e sim conforme surge a necessidade a transferência ocorre através do mecanismo do carregador de classes do Java (“C*lassLoader*”).

Por fim, as duas últimas limitações estão relacionadas com o tamanho do código e a sobrecarga introduzida pelos protocolos de comunicação. Estes fatores afetam o desempenho da comunicação. O tamanho do código a ser enviado aos nós é um fator chave para determinar o limite no qual o código móvel torna-se útil na redução do tráfego de rede. Tornar o código o mais compacto possível é um dos objetivos dos sistemas de código móvel. A implementação com Java mostra que o tamanho do código a ser enviado aos nós é considerável.

Quanto a escolha do protocolo de comunicação, esta depende do programador. O protocolo de aplicação usado para transferir código móvel pertence a API escolhida pelo programador. O Java *Aglets*, por exemplo, utiliza o protocolo ATP (*Agent Transfer Protocol*) que aumenta significativamente a sobrecarga de transmissão e o processamento, pois este se localiza acima do protocolo TCP ou UDP. Já o SNMP utiliza diretamente o protocolo UDP, não existindo nenhum protocolo intermediário.

2.5 Tecnologia ativa no gerenciamento distribuído

A utilização da tecnologia ativa no gerenciamento distribuído vem sendo muito pesquisada nos últimos anos [2][39]. Os elementos da rede que devem ser gerenciados podem receber apenas funcionalidades básicas referentes à obtenção de informações de gerenciamento. Capacidades genéricas e específicas podem ser providas via agentes enviados a esses elementos. O comportamento desses agentes pode ser modificado a qualquer momento. Com essa flexibilidade, os elementos da rede podem ser configurados para atividades de gerenciamento de acordo com as necessidades do momento. As atualizações de regras para a tomada de decisões, de limites para desencadeamento de ações ou de um protocolo de gerenciamento, por exemplo, poderiam ser feitas mediante simples atualizações dos códigos dos agentes em questão. O mesmo se aplica a mudanças de políticas e inclusão de novos serviços de gerenciamento.

Além da facilidade de atualização de funcionalidades, os agentes podem ser providos de capacidades avançadas em relação ao processamento das informações de gerenciamento. Os tempos de detecção de problemas e de restabelecimento do funcionamento normal do elemento gerenciado podem então ser bastante reduzidos, uma vez que o agente está sendo executado localmente no elemento da rede. Pela mesma razão, o consumo de banda passante da rede para fins de gerenciamento pode ser significativamente reduzido, pois o tráfego de gerenciamento é menor. Os agentes filtram e consolidam as informações destinadas as estações de gerência, enviando apenas um mínimo de informações, como relatórios de ocorrências.

Entretanto, quando o número de elementos a serem gerenciados for pequeno, o consumo de banda e de tempo para a transferência dos agentes móveis certamente serão

maiores do que na utilização de uma abordagem convencional de gerenciamento, por exemplo, via SNMP [10].

2.6. Trabalhos Relacionados

Os trabalhos apresentados a seguir referem-se ao gerenciamento utilizando tecnologia ativa.

Smart Packets [38] é um exemplo de sistema de gerenciamento distribuído que utiliza somente o paradigma de redes ativas. O Active Network protocol (*ANEP*), é um protocolo específico para a transferência de programas que não podem exceder 1 KB. Com o objetivo de ter-se uma proteção no nó ativo, foi implementada uma linguagem de alto nível, Sprocket, que não dispõe de ponteiros, não realiza o acesso a arquivos e nem faz gerenciamento de memória, que são ações potencialmente problemáticas. Um programa em Sprocket é compilado em outra linguagem, Spanner, com o objetivo de minimizar o tamanho do programa a ser transmitido na rede. Um ambiente de execução capaz de executar programas Spanner está presente em cada nó ativo. Um esquema de segurança realiza a autenticação dos programas a serem executados nos nós ativos.

A arquitetura *Active Distributed Management for IP Networks* (ADM) [39] é um exemplo de sistema de gerenciamento distribuído para redes IP que usa tecnologia ativa. A arquitetura ADM é organizada em três camadas: gerenciamento de processos, operação e ferramentas de gerenciamento. A primeira camada é composta pelo ambiente de execução, que utiliza o paradigma de agentes móveis e o de redes ativas, disponibilizando funções para criação, remoção, interrupção, continuação, duplicação, movimentação, comunicação, serviços de diretório e de segurança. A segunda camada, a de operação, disponibiliza padrões de navegação para os fragmentos de código ativo (da terceira camada) que de fato realizam tarefas de gerenciamento. Na terceira camada são implementadas as funções relativas ao gerenciamento propriamente dito, como aplicações ativas. A linguagem utilizada é Java.

O projeto MIAMI [56] implementa o gerenciamento de desempenho com o uso de três agentes, dois estáticos e um móvel. O primeiro, estático, realiza a interface entre o gerenciamento de desempenho e o sistema de gerenciamento. Este agente, quando ordenado, cria um segundo agente que é móvel e o envia a um elemento de rede para que realize funções de monitoramento e de consolidação localmente. Um terceiro

agente, estático, serve como agente *proxy*, realizando a interface entre o agente móvel e o elemento a ser gerenciado. O agente móvel envia informações consolidadas ao primeiro agente estático periodicamente ou assincronamente quando limites configurados para parâmetros de desempenho forem ultrapassados. A linguagem utilizada também é Java.

O diferencial da arquitetura AGAD em relação aos trabalhos descritos é que eles não usam simultaneamente tecnologia ativa, o mecanismo de filtragem e consolidação de informações no nó gerenciado e a implementação de componentes de software. A utilização de componentes de software permite que as diferentes áreas de gerenciamento sejam tratadas independentemente, permitindo que os componentes dos elementos da AGAD sejam configurados dinamicamente e independentemente pelo administrador do sistema.

O mecanismo de filtragem permite que apenas informações relevantes sejam enviadas pelos Inspetores ao Gerente de Domínio e por este ao Gerente-Mor. Já o mecanismo de consolidação permite que uma única mensagem contenha mais de um tipo de informação (ex: informação de controle e de relato de algum evento). Estes dois mecanismos têm como objetivo uma redução no tráfego de informações de gerência na rede em relação ao esquema de polling e de traps utilizados pelo SNMP.

A utilização de tecnologia ativa permite uma maior eficiência na detecção e solução de problemas, em função da capacidade de processamento local dos Inspetores e Especialistas nos nós gerenciados. Na AGAD só foi implementado a tecnologia ativa agentes móveis e não a de redes ativas.

2.7. Considerações Finais do Capítulo

Neste capítulo foram descritos conceitos básicos de gerência, o conceito de tecnologia ativa e as vantagens da utilização da tecnologia ativa no gerenciamento distribuído.

O uso de tecnologia ativa nos elementos da rede permite que eles sejam configurados para atividades de gerenciamento de acordo com as necessidades do momento, permitindo que novas funcionalidades de gerência possam ser inseridas dinamicamente. Além da facilidade de inserção de funcionalidades, os agentes podem ser providos de capacidades avançadas objetivando o processamento das informações de

gerenciamento, e a atualização de funcionalidades já existentes. Os tempos de detecção de problemas e de restabelecimento do funcionamento normal do elemento gerenciado podem então ser bastante reduzidos, uma vez que o agente está sendo executado localmente no elemento de rede. Consequentemente, o consumo de banda da rede para fins de gerenciamento é reduzido, pois o tráfego de gerenciamento é menor. Apenas um mínimo de informações, como alarmes críticos, e informações consolidadas precisam ser enviadas para as estações de gerência.

Capítulo 3 - Arquitetura AGAD

Este capítulo apresenta uma proposta para uma arquitetura de gerenciamento distribuída denominada AGAD que utiliza tecnologia ativa. A tecnologia ativa viabiliza a presença de agentes de gerenciamento dotados de inteligência e capacidade de processamento para monitorar tendências e erros, bem como viabilizar o envio de agentes especialistas para a realização de ações onde as mesmas forem necessárias.

O esquema de gerenciamento da AGAD traz duas vantagens em relação ao esquema de gerenciamento centralizado do SNMP. A primeira é, uma redução significativa no tráfego de informações de gerência na rede em relação ao esquema de *polling* (leitura de variáveis à distância) utilizado pelo SNMP, devido a descentralização de processamento e controle. A segunda é, uma maior eficiência na detecção e solução de problemas, em função da capacidade de processamento local dos Inspetores e Especialistas nos nós gerenciados (hosts, roteadores, etc), reduzindo os retardos associados às ações de gerenciamento. Estas duas vantagens permitem que haja uma otimização no uso dos recursos computacionais memória, CPU e banda passante, e permite que um gerente de rede possa analisar informações consolidadas sobre as diversas áreas de gerência que foram coletadas e armazenadas em base de dados. Estas análises permitem ao gerente por exemplo, verificar o desempenho da rede, detectar possíveis pontos de gargalo (*bottlenecks*) e, até mesmo, prever uma futura expansão da rede.

A arquitetura AGAD tem também duas outras características importantes. A primeira é o fato de ela ter sido planejada para ser uma arquitetura robusta em relação ao funcionamento dos seus elementos, ou seja, se um elemento falhar, existe um mecanismo responsável pela criação de um novo elemento, com as mesmas funcionalidades daquele que falhou. A segunda característica é a criação de componentes de software para os seguintes elementos da arquitetura: Inspetor, Gerente de Domínio e Gerente-Mor . O uso de componentes de software permite flexibilidade na atualização e na introdução de novas funcionalidades. Cada um dos elementos da arquitetura, possui um ou mais componentes que corresponde a uma das cinco áreas da gerência OSI. Por exemplo, toda monitoração relativa a desempenho nos nós gerenciados é realizada pelo componente de software de gerenciamento de desempenho

instalado no Inspetor. Qualquer componente, independente da área a qual ele pertença, pode ser atualizado de forma dinâmica e transparente, para os demais componentes que o referencia.

Este capítulo está organizado da seguinte forma: a seção 3.1 apresenta uma descrição geral do funcionamento da arquitetura; a seção 3.2 descreve o funcionamento detalhado desta e dos seus componentes gerenciadores; a seção 3.3 descreve a comunicação entre os componentes da AGAD; a seção 3.4 detalha o sistema de distribuição de código e a seção 3.5 apresenta as considerações finais do capítulo.

3.1. Descrição Geral da Arquitetura

A arquitetura AGAD foi projetada com o objetivo de prover um gerenciamento distribuído de nós pertencentes a uma rede organizada logicamente em um ou mais domínios administrativos, utilizando tecnologia ativa. Um domínio administrativo é composto por um ou mais segmentos de rede local, ao qual são interligados estações, roteadores, comutadores, hubs, etc. Cada domínio é controlado por um elemento gerenciador chamado Gerente de Domínio, que é subordinado a um gerente que lhe é hierarquicamente superior chamado Gerente-Mor, conforme apresentado na Figura 1.

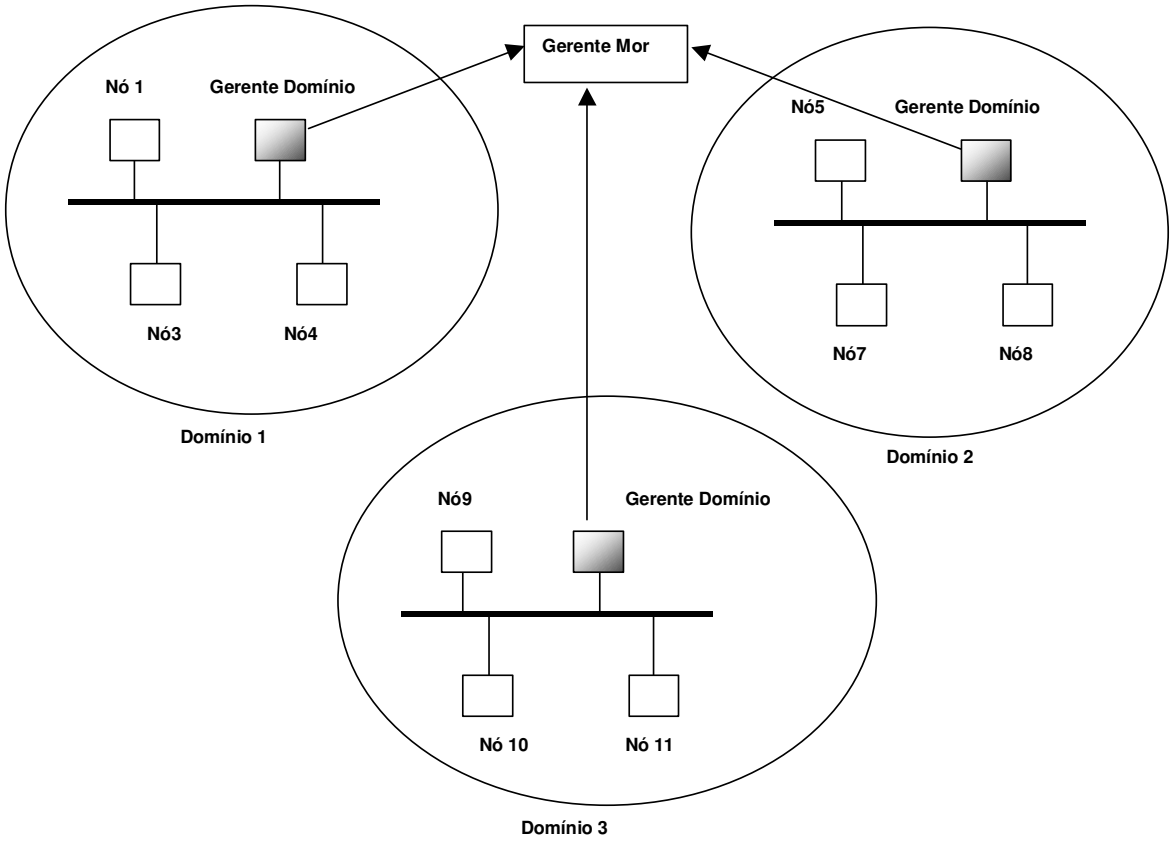


Figura 1 - Arquitetura AGAD

A AGAD é organizada hierarquicamente em, no mínimo, três níveis, conforme a Figura 2. Em cada nível há elementos gerenciadores. No primeiro nível da hierarquia, encontra-se o Gerente-Mor, que monitora um ou mais domínios. O Gerente-Mor é executado em uma única estação de trabalho e é responsável pela coleta de informações estatísticas obtidas através do seu conjunto de Gerentes de Domínio. Para cada conjunto de N domínios há um Gerente-Mor, ou seja, dependendo do tamanho da rede a ser gerenciada, pode haver um ou mais Gerentes-Mores. No segundo nível hierárquico, encontram-se os Gerentes de Domínio que são responsáveis pelo monitoramento dos recursos de rede que se deseje gerenciar dentro de seu domínio. Baseado nas informações recebidas pelos Gerentes de Domínio, um gerente de rede pode traçar curvas sobre diferentes aspectos da rede e detectar a necessidade de aumento de capacidade de alguns elementos, como enlaces e nós de comutação. No terceiro nível

hierárquico, encontram-se os outros elementos gerenciadores, que são os Inspetores e os Especialistas.

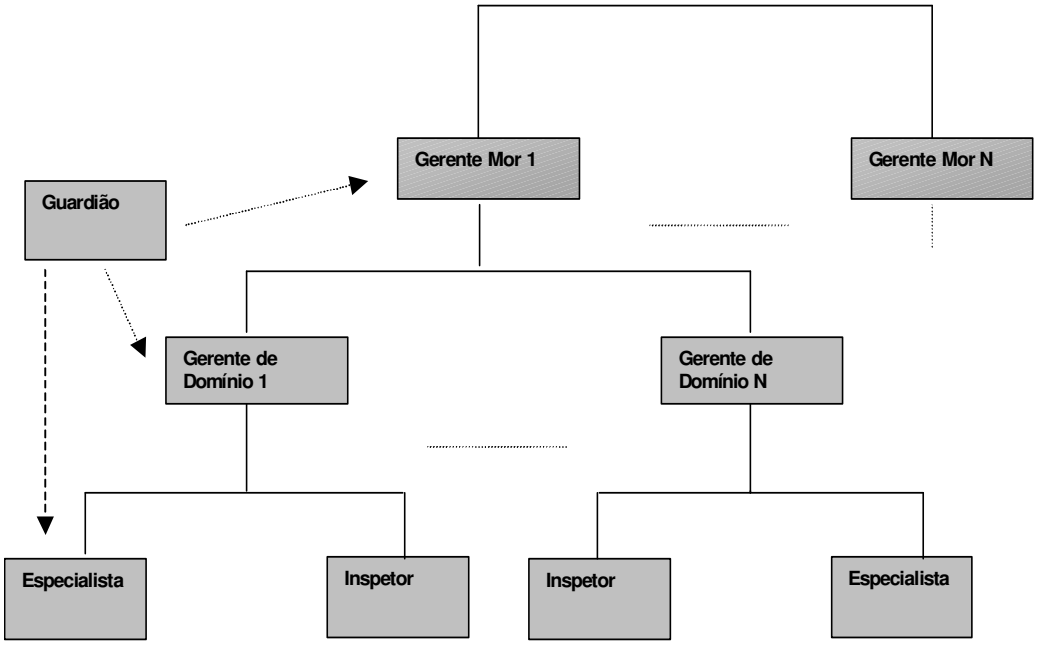


Figura 2 - Hierarquia de gerenciamento AGAD

Os Inspetores são responsáveis pela coleta local de dados relativos a uma ou mais áreas de gerência e pelo processamento local dos mesmos, que é a grande vantagem do gerenciamento distribuído. A obtenção destes dados é feita através de consultas, via SNMP, às variáveis da MIB do elemento de rede que está sendo gerenciado ou através de agentes *proxy*, caso o elemento a ser gerenciado não implemente o protocolo SNMP ou outro protocolo de gerenciamento. Como exemplo, os Inspetores podem consultar o estado das interfaces de um roteador, a percentagem de utilização de CPU e de memória, a taxa de erro das interfaces, etc. Ocorrendo um determinado evento no nó, (por exemplo, a percentagem de utilização de CPU ultrapassar um limite pré-estabelecido ou uma das interfaces do nó falhar), o Inspetor deve enviar alarmes (informações urgentes) sobre o evento ao Gerente de Domínio. Se as informações não forem do tipo alarme, o Inspetor pode consolidá-las e, então, enviá-las ao Gerente de Domínio de forma assíncrona ou não.

O Gerente de Domínio, após a análise das informações recebidas, é capaz de verificar a necessidade ou não do envio de uma versão do Especialista mais adequada

para resolver um determinado problema ou para analisar um ou mais parâmetros. Os Especialistas são *software* autônomos, com funções bem definidas. Os Especialistas apropriados são enviados pelo Gerente de Domínio ao elemento gerenciado em questão com um objetivo bem definido, para o qual foram programados. Um exemplo de ação de um Especialista é aprimorar alguma tarefa executada por um Inspetor. Por exemplo, um Inspetor tem como tarefa verificar o percentual total de utilização de CPU no nó hospedeiro, já um determinado Especialista tem como função determinar qual dos processos em execução possui o maior percentual de utilização de CPU.

O único elemento da arquitetura, presente em todos os níveis hierárquicos é o Guardião. O Guardião tem como objetivo verificar se os demais elementos da AGAD estão funcionando corretamente e se os códigos dos mesmo estão íntegros, ou seja, se não foram alterados inadvertidamente ou intencionalmente. Como existe a possibilidade dos elementos de um sistema de gerência sofrerem algum tipo de ataque, o Guardião deve verificar se algum elemento foi subvertido por algum atacante. Caso o Guardião detecte alguma anomalia, este deve informá-la ao Gerente de Domínio ou ao Gerente-Mor. Para que isto seja possível, todos os elementos da arquitetura possuem, obrigatoriamente, uma interface com o Guardião.

3.2. Componentes da AGAD

Nessa seção, são apresentados de forma detalhada os elementos da arquitetura, incluindo o objetivo de cada um, os seus relacionamentos com os demais elementos da AGAD e o ciclo de vida de cada elemento.

Na arquitetura AGAD, as tarefas de gerência são divididas entre vários elementos, denominados conforme sua especialização: o Gerente-Mor, o Gerente de Domínio, o Inspetor, o Especialista e o Guardião. Estes elementos são capazes de monitorar e tentar solucionar problemas relacionados à falha, desempenho, segurança , configuração e contabilidade. Os elementos Gerente de Domínio e Inspetor são compostos por componentes de software, cada um pertencente a uma das cinco áreas de gerência. Os componentes de software de cada elemento são apresentados conforme a

figura 3.

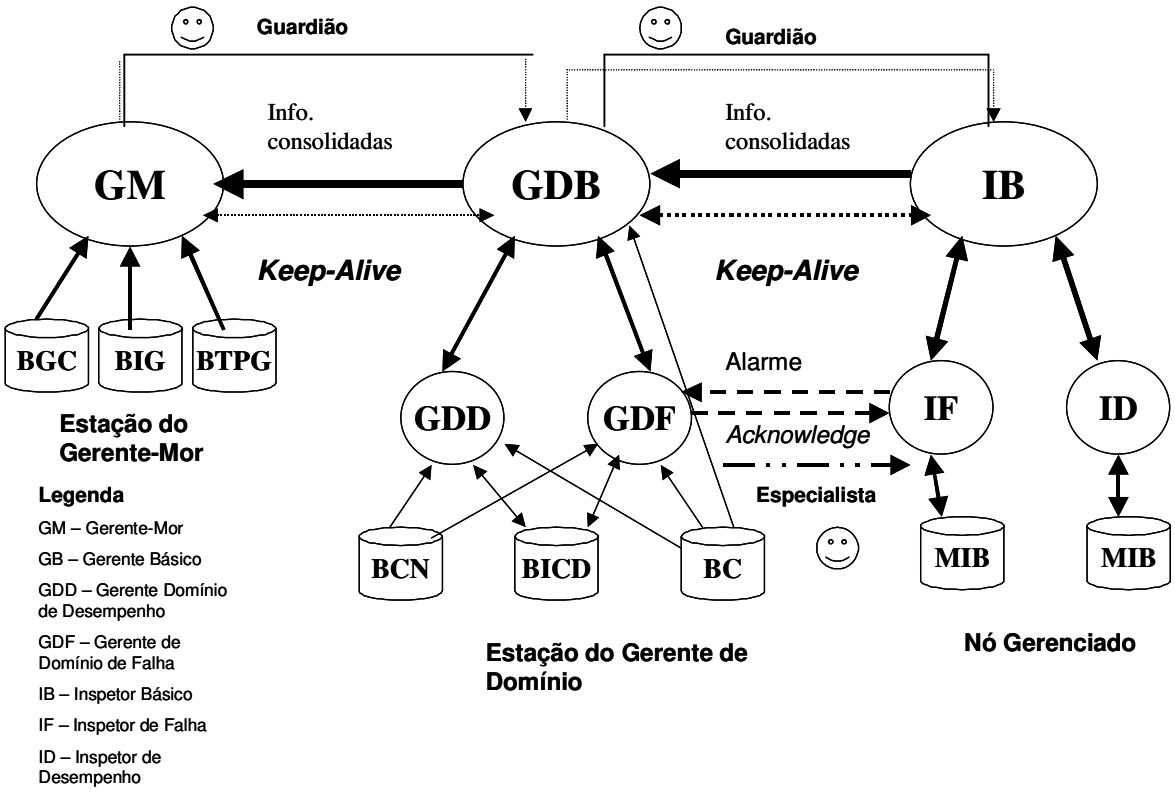


Figura 3 – Componentes gerenciadores de um domínio e as mensagens trocadas entre eles

O funcionamento dos elementos da AGAD pertencentes a um único domínio administrativo é detalhado a seguir.

3.2.1. Gerente-Mor

O Gerente-Mor, criado na inicialização do sistema, é executado em uma estação de trabalho dedicada à gerência. Ele oferece uma interface gráfica (GUI) para o gerente da rede, além de ser o responsável pelo gerenciamento de um conjunto de domínios, através dos Gerentes de Domínio, aos quais delega tarefas de gerenciamento.

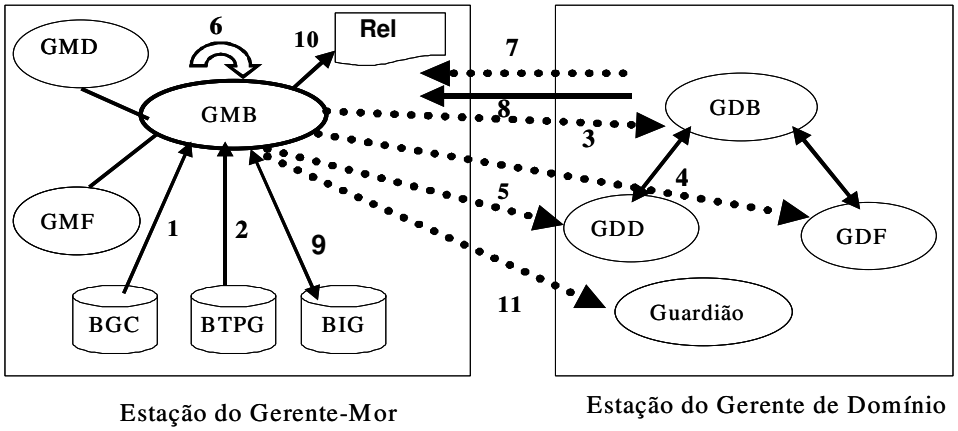


Figura 4 – Ciclo de Vida do Gerente-Mor

O Gerente-Mor possui, em seu código, diferentes componentes de *software* que tratam das diferentes áreas funcionais de gerenciamento (falha, desempenho, segurança, contabilidade e configuração) e possui um componente chamado Gerente-Mor Básico.

O ciclo de vida do Gerente-Mor é descrito em seguida.

1. O Gerente-Mor, ao ser ativado em uma estação de trabalho, cria os Gerentes de Domínio (o Básico e os específicos de cada área de gerência), através da base BGC e os envia para os seus respectivos domínios, baseado em informações topológicas (base de topologia geral). O funcionamento destes Gerentes de Domínio passa a ser monitorado por este Gerente-Mor. Esses procedimentos estão representados na figura 4 através dos fluxos 1,2,3,4 e 5.
2. Após o envio dos Gerentes de Domínio, o Gerente-Mor Básico inicializa o mecanismo para detecção de falhas nestes elementos. Este mecanismo funciona da seguinte forma: o Gerente-Mor ativa um procedimento, chamado detecção de falha, responsável por incrementar um contador, durante um determinado intervalo de tempo ou até esse contador ser resetado pelo Gerente Básico, representando o recebimento da mensagem *keep-Alive* enviada por este. Esta mensagem é enviada,periodicamente, pelo Gerente Básico informando o estado dos gerentes específicos pertencentes a um mesmo nó. Dentro desse intervalo de tempo, o Gerente Básico invoca,

remotamente, o procedimento de detecção de falhas do Gerente-Mor e reseta o valor do contador. Caso esse tempo se esgote (*timeout*) e o contador não tenha sido resetado, o Gerente-Mor detecta que está acontecendo alguma anormalidade. Esta anormalidade pode estar associada a diversos fatores: falha no componente Gerente de Domínio Básico, congestionamento na rede, falha de hardware na estação do Gerente de Domínio, queda de enlace entre o Gerente-Mor e o Gerente de Domínio, problema no sistema operacional da estação, etc. Após essa detecção, o Gerente-Mor envia então, um determinado número de ICMP *echo request* e aguarda os ICMP *echo reply*. Uma falha é confirmada após n tentativas de envio de ICMP *echo request* para todos os enlaces que ligam o Gerente-Mor a um determinado Gerente de Domínio, sem o recebimento de nenhum ICMP *echo reply*. Nesse caso, o Gerente-Mor sinaliza um alarme crítico na interface gráfica, colocando em vermelho o nó e os enlaces que vão até a estação em questão. A decisão do que fazer é tomada pelo administrador da rede. Se o Gerente-Mor receber ICMP *echo reply,* ele envia um Guardião para verificar se a aplicação do Gerente de Domínio falhou. No caso positivo, o Guardião envia uma mensagem de controle avisando sobre a falha e o Gerente-Mor envia um novo componente Gerente de Domínio. Estes procedimentos estão representados na figura 4 através dos fluxos 6 e 7.

3. O Gerente-Mor requisita e recebe informações consolidadas dos Gerentes de Domínio Básico subordinados a ele e as reúne na base de informações de gerenciais chamada BIG, que armazena informações sobre a gerência de desempenho, configuração, contabilização, segurança e falha de todos o(s) domínio(s) gerenciados por ele. Estes procedimentos estão representados na figura 4 através dos fluxos 8 e 9.
4. Baseado nas informações obtidas, o Gerente-Mor Básico gera estatísticas consolidadas e relatórios sobre as informações obtidas, além de disponibilizá-las para o administrador da rede. Estes procedimentos estão apresentados na figura 4 através dos fluxos 9 e 10.

5. Periodicamente, o Gerente-Mor Básico cria Guardiões e os envia para cumprir tarefas relativas ao monitoramento da integridade dos demais componentes gerenciadores. Estes procedimentos são apresentados na figura 4 através do fluxo 11.

O Gerente-Mor, além de possuir a interface gráfica (GUI), pela qual é possível a verificação do estado de toda a rede, possui uma base de dados que contém o código de todos os componentes do sistema, chamada BGC (base geral de código). Esta base BGC é carregada pelo administrador da rede com os códigos de todos os elementos da arquitetura, antes do sistema ser inicializado. Existe, também, uma base chamada BTPG (base de topologia geral), que contém a topologia de todo(s) o(s) domínio(s) subordinado(s) a este Gerente-Mor. A inserção das informações topológicas na base BTPG pode ser manual ou automática. No caso de uma rede local com poucos nós quase sem alteração na topologia física, é mais vantajosa a inserção manual, ou seja, o administrador da rede é responsável por atualizar a topologia da rede na base, através de uma interface gráfica, antes do sistema ser inicializado. Já numa rede com dimensões maiores, com muitas sub-redes e centenas de nós e que sofra alterações topológicas freqüentes, a atualização manual se torna uma tarefa praticamente inviável. Neste caso, é necessária a utilização de um algoritmo específico para a descoberta automática da topologia lógica da rede (camada 3), apresentado em [43]. Assim, a base é povoada, inicialmente, apenas por alguns endereços e, após a inicialização do sistema, a atualização passa a ser feita automaticamente, através deste algoritmo. Em [43], além dos algoritmos utilizados para a descoberta da topologia lógica de uma rede, são apresentados algoritmos criados para a descoberta da topologia física de redes IP heterogêneas. Estes algoritmos, responsáveis pela descoberta da topologia física, baseiam-se nas informações da MIB SNMP padrão, largamente suportada pelos elementos de rede IP modernos. Uma das vantagens da implantação destes algoritmos é o fato que eles não exigem modificações no sistema operacional dos elementos gerenciados. A implementação de um algoritmo eficiente de descoberta da topologia lógica da rede é essencial para uma melhor distribuição dos elementos da AGAD.

O povoamento da base de topologia geral (BTPG) é realizado de forma manual pelo administrador da rede. Já a base de topologia de domínio (BTPD), é povoada apenas com informações dos nós e enlaces de um determinado domínio através de um

mecanismo de réplica da base BTPG. A cada nova informação topológica inserida na base BTPG, pelo administrador, ocorre uma réplica dessa informação para a base BTPD pertencente ao domínio em questão.Todas as alterações realizadas nas bases de topologia geral e de informações dos domínios, efetuadas em uma estação de Gerente-Mor, são replicadas para as respectivas bases dos outros Gerentes-Mores. O objetivo desta réplica é que, no caso de falha em um Gerente-Mor, qualquer outro seja capaz de assumir a gerência dos domínios subordinados ao componente gerenciador que falhou. Esta réplica é gerada na inicialização do sistema e após um intervalo pré-definido. Este intervalo pode ser reconfigurado a qualquer momento, através de intervenção humana, de acordo com o tráfego da rede.

3.2.2. Gerente de Domínio

O Gerente de Domínio é responsável pelo gerenciamento de seu domínio, que inclui a disponibilização de informações de nível domínio e o desencadeamento de ações, quando necessárias, para a reversão de alguma situação desfavorável ou para, pelo menos, a minimização dos efeitos dessa situação.

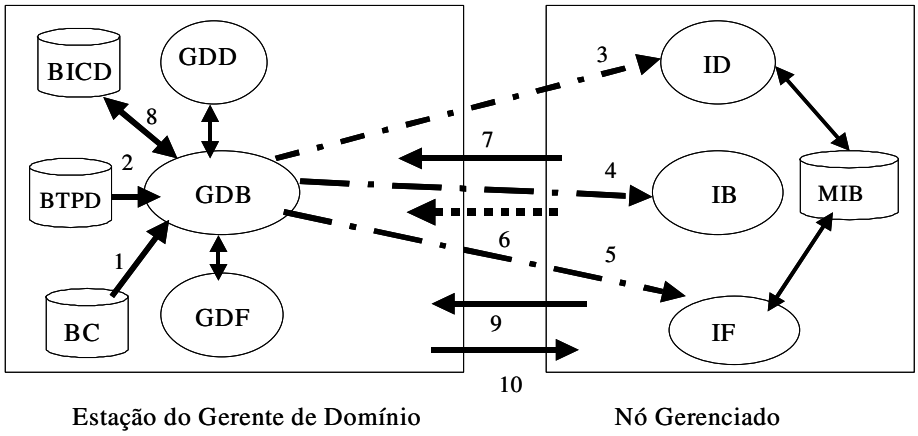
Assim como o Gerente-Mor, o Gerente de Domínio possui, em seu código, diferentes componentes de software que tratam das diferentes áreas funcionais de gerenciamento (falha, desempenho, segurança, contabilidade e configuração) e possui um componente chamado Gerente de Domínio Básico. Eles são responsáveis pela gerência de seu domínio, delegando tarefas para os Inspetores e Especialistas, com o objetivo de distribuir o monitoramento e o controle de variáveis do sistema operacional e da MIB referentes à gerência de falha, desempenho, segurança, configuração e contabilização dos elementos gerenciados. O Gerente de Domínio também possui uma interface gráfica (GUI) e é executado em uma estação de trabalho dedicada à gerência de um domínio.

É ele o responsável pela criação e envio dos Inspetores e Especialistas aos elementos do domínio que devem ser gerenciados mantendo, para isso, uma base de dados chamada **BC** com os códigos executáveis de todos os elementos pertencentes a um domínio. Esta base é carregada, de forma automática , não só na inicialização do sistema, como durante o funcionamento do mesmo, através de um mecanismo de réplica

a partir da base geral de códigos BGC, toda vez que um novo elemento ou componente de domínio for criado ou modificado.

Antes de enviar os Inspetores e Especialistas para os nós do domínio, o Gerente de Domínio consulta a base de dados **BTPD (base de topologia do domínio)**, onde todas as informações sobre a topologia de seu domínio se encontram armazenadas. O processo de inicialização e atualização desta base é o mesmo descrito, na sub-seção 3.2.1, para a base **BTPG**.

A base de dados chamada **BCN** armazena a associação entre uma determinada tarefa de gerência e o Especialista mais apropriado para executá-la. Esta base pode ser atualizada manualmente, antes da inicialização do sistema ou, então, ser inferida com o auxílio de ferramentas de Inteligência Artificial (IA). A classificação do Especialista pelo Gerente de Domínio específico pode se utilizar de técnicas de IA, daí a presença da BCN. Nesta proposta esta base é carregada durante a inicialização do sistema e é consultada pelo Gerente de Domínio específico, a cada necessidade de envio de um Especialista.



Legenda:

7 – Informações Consolidads

6 – Mensagem *Keep-Alive*

9 – Mensagem de Alarme

10 – Acknowledge

Figura 5 – Ciclo de Vida do Gerente de Domínio

O ciclo de vida do Gerente de Domínio é descrito em seguida.

1. O Gerente de Domínio Básico ao ser ativado em uma estação de trabalho, cria os Inspetores e os envia para todos os nós do domínio, baseado nas informações da base BTPD e na base de códigos BC. Estes procedimentos são representados na figura 5 através dos fluxos 1,2,3,4 e 5.
2. Após o recebimento da mensagem *keep-Alive*, remetida por todos os Inspetores, anunciando a sua ativação, o Guardião é enviado, já com um itinerário pré-definido, pelo Gerente de Domínio Básico. O recebimento do *keep-Alive* está representado na figura 5 pelo fluxo 6.
3. Quando o Gerente de Domínio Básico recebe uma mensagem, ele verifica o tipo da mensagem, a área a que ela pertence e o seu remetente, que pode ser um Inspetor, um Gerente de Domínio específico ou o Guardião. Se o remetente da mensagem for um Inspetor específico ou o Guardião, após o recebimento da mensagem, o Gerente de Domínio específico envia uma confirmação para o remetente. Todas as informações recebidas pelo Gerente de Domínio Básico e/ou específico são gravadas na Base de Informações Consolidadas de Domínio (BICD). Esta base só começa a ser povoada quando o Gerente de Domínio Básico ou específico receber a primeira mensagem vinda de um Inspetor ou Guardião. De acordo com o tipo de mensagem recebida e com o conteúdo armazenado na Base de Conhecimento (BCN), o Gerente de Domínio específico é capaz de classificar e verificar se há ou não um Especialista para realizar uma determinada tarefa. Estes procedimentos estão representados na figura 5 pelos fluxos 9 e 10.
4. Com base nas informações recebidas, o Gerente de Domínio Básico emite relatórios e faz a compilação de estatísticas de um determinado domínio, valendo-se de uma versão simplificada da interface (GUI) utilizada também pelo Gerente-Mor .
5. O Gerente de Domínio Básico envia, periodicamente, para o Gerente-Mor ao qual está subordinado, informações consolidadas consideradas importantes sobre seu domínio. Uma destas informações é a mensagem de

controle (*Keep-Alive*) para sinalizar ao Gerente-Mor quais os Gerentes de Domínio específicos não estão ativos. Por medida de segurança, esta sinalização pode ser enviada em diferentes pacotes: em caso de congestionamento, um deles pode ser perdido. O recebimento das informações consolidadas está representada na figura 5 pelo fluxo 7.

Dependendo do tamanho total da rede a ser gerenciada, pode-se imaginar malhas compostas por domínios adjacentes como, por exemplo, um para cada filial no caso de uma empresa, conforme a figura 4. A comunicação entre os Gerentes-Mores e entre os Gerentes de Domínio não será explorada neste trabalho.

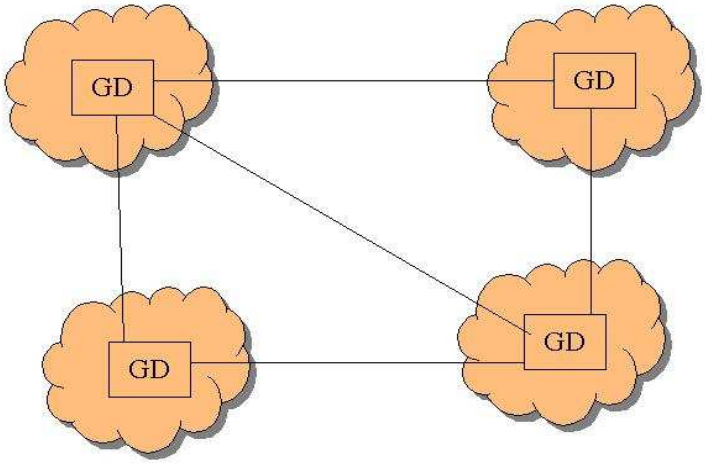


Figura 6 – Malhas compostas por domínios adjacentes

3.2.3. Inspetores

Os Inspetores são responsáveis por realizar, de fato, as tarefas relativas ao gerenciamento dos elementos (estações*, switchs, hubs*, etc) do domínio. Além de coletar informações referentes às funcionalidades que estão sendo gerenciadas no nó em que atuam, eles devem informar ao Gerente de Domínio sobre eventuais anomalias para registro como, por exemplo, a ocorrência de uma falha.

Os inspetores funcionam de forma semelhante aos monitores RMON: coletam, continuamente, informações nas áreas de configuração, segurança, falha, desempenho e contabilização. Esta coleta é feita com o uso das facilidades disponíveis, desde consultas ao sistema operacional à MIB local até o uso de agentes *proxy*. Os dados coletados sofrem um processamento local realizado pelo próprio Inspetor para garantir que apenas um mínimo de informações seja enviado ao Gerente de Domínio. O processamento local tem por objetivo realizar desde a filtragem de informações irrelevantes até a fusão de informações relevantes diferentes em uma mesma mensagem a ser enviada ao Gerente de Domínio Básico.

As consultas as informações dos elementos do domínio podem ser realizadas de forma síncrona, através de solicitação do Gerente de Domínio ou de forma assíncrona, por iniciativa do próprio Inspetor.

O código do Inspetor deve ser otimizado de tal forma que ele não comprometa a carga computacional do nó. O Inspetor é implementado em componentes de software, cada qual pertencente a uma das cinco áreas da gerência, conforme apresentado na Figura 7. Esta implementação é configurável dinamicamente, de forma a permitir que as diferentes áreas funcionais de gerenciamento sejam tratadas independente, de acordo tanto com a natureza do elemento a ser gerenciado quanto com as intenções do administrador do domínio. As funções básicas tratam de tarefas referentes ao funcionamento do Inspetor em si e de tarefas comuns às áreas funcionais de gerenciamento, como por exemplo, a comunicação com o Gerente de Domínio e a interação com o Guardião.

O Inspetor Básico é responsável pela consolidação dos dados (informações de relatório) a serem enviados para o Gerente de Domínio Básico e pelo recebimento de informações vindas do Gerente de Domínio Básico. Já as informações urgentes (alarmes) são enviadas pelos próprios Inspetores específicos ao Gerente de Domínio específico da área.

A figura 7 mostra a modelagem de um Inspetor, onde os dois módulos coloridos representam, respectivamente, um Inspetor de Desempenho e outro o Inspetor Básico já instalados no nó. Os demais módulos mostram que novos componentes pertencentes às outras áreas de gerência podem ser adicionados dinamicamente em um determinado nó. Na inicialização do sistema, apenas os componentes Inspetor Básico, de Falha e

Desempenho, chamados de Inspetores específicos, são enviados aos nós, sendo os outros enviados conforme a necessidade de monitoração e análise, pelo administrador.

O bloco comunicações refere-se às facilidades de comunicação de dados do sistema operacional do componente gerenciado que hospeda o Inspetor. Agentes *proxies* têm por finalidade disponibilizar uma interface padrão com o Inspetor e, ao mesmo tempo, interagir com uma tecnologia proprietária. Agentes legados são aqueles diretamente acessáveis, via interfaces bem definidas, como por exemplo, uma MIB.

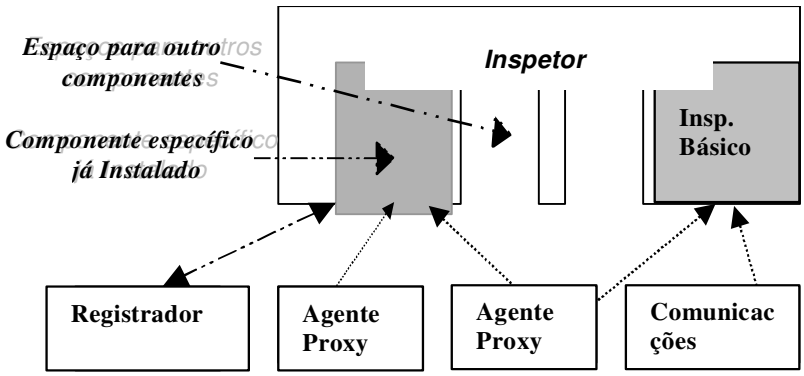
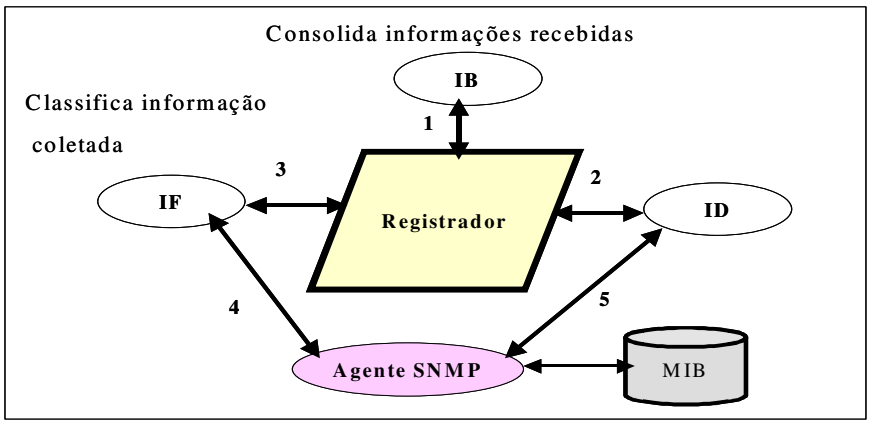


Figura 7 – Modelagem do Inspetor



Legenda:

IB – Inspetor Básico

IF – Inspetor de Falha

ID – Inspetor de Desempenho

Figura 8 – Ciclo de vida do Inspetor dentro do nó gerenciado

O ciclo de vida de um Inspetor Básico e específico, como apresentado na figura 8, obedece as normas descritas em seguida.

1. Os componentes Inspetor Básico e específicos de Falha e Desempenho são criados pelo Gerente de Domínio Básico, na estação da gerência de domínio, e são enviados a todos os nós gerenciados de um domínio;
2. Ao serem ativados no nó destino, estes componentes devem se comunicar com o bloco Registrador para registrarem as suas referências. Isso é feito através da chamada de um determinado método do Registrador. O objetivo do Registrador é oferecer transparência para os demais componentes de um nó quando houver substituição de algum componente Inspetor que esteja sendo referenciado por algum outro Inspetor. Esse procedimento é representado na figura 8 pelos fluxos 1,2 e 3;
3. Após se registrar, o Inspetor específico deve monitorar toda(s) a(s) variável(is) da MIB para o qual foi designado. Esta tarefa está representada na figura 8 pelos fluxos 4 e 5;
4. Se a informação coletada for uma anomalia, deve ser gerado e enviado um alarme diretamente para o Gerente de Domínio específico. Caso a informação coletada não seja uma anomalia, isto é, apenas informativa, ela deve ser enviada para o Gerente de Domínio Básico. Após preencher um *stream* do tamanho do campo *payload* de um quadro Ethernet, por exemplo, o Inspetor Básico envia as informações consolidadas para o Gerente de Domínio Básico.

Dentro de cada área de gerência, existem Inspetores específicos para o elemento a ser monitorado, ou seja, existe, por exemplo, um Inspetor de Desempenho específico para uma estação e um outro para um nó de comutação.

3.2.4. Especialistas

Especialistas são necessários quando os Inspetores detectam situações que exigem ações ou, até mesmo, análises mais específicas. Não possuindo uma configuração dinâmica, semelhante a do Inspetor, os Especialistas apropriados são

enviados pelo Gerente de Domínio específico ao nó gerenciado em questão com um objetivo bem definido como, por exemplo, uma alteração de rota.

Um dos objetivos da AGAD é ter um número grande de especialistas responsáveis pela execução de tarefas específicas. A idéia é a existência de uma granularidade ótima de especialização e um código mais otimizado. O motivo desta especialização ótima é que deve haver um compromisso entre o tempo de processamento nos nós gerenciáveis, a latência de resposta (tempo de resposta) e o consumo de banda.

O código otimizado traz como vantagem um menor consumo de banda e, como desvantagem, o aumento no tempo de classificação do Especialista mais adequado por parte do Gerente de Domínio. O aumento no tempo de classificação pode impactar o tempo de resposta do sistema. Deve existir um compromisso entre o tempo de resposta do sistema e o consumo de banda.

O ciclo de vida de um Especialista independente da área a que ele pertença, é descrito em seguida.

1. O Gerente de Domínio específico cria um Especialista em seu hospedeiro, a partir do recebimento das informações da MIB enviadas por um Inspetor ou através de informações recebidas pelo Gerente de Domínio Básico. De posse destas informações, o Gerente de Domínio específico classifica e verifica qual o Especialista mais adequado para ser enviado ao nó destino;
2. O novo Especialista é enviado para o nó destino. Pode haver a possibilidade de ainda não existir um Especialista para resolver um determinado problema. Neste caso, é sinalizado um alarme crítico na interface gráfica, para que alguma ação seja tomada pelo administrador da rede.
3. Ao ativar-se no nó destino, o Especialista executa as ações para as quais foi enviado.

A seguir são descritos dois exemplos do uso de especialistas de falhas. O primeiro seria uma falha em um enlace (*link*). O Especialista de falha tenta ativar um enlace "*backup*" já que o enlace principal falhou. O segundo exemplo seria uma falha de segurança, percebida através da leitura de arquivos de segurança do sistema operacional (*logs*) por um Inspetor de Segurança. Neste caso, um Especialista de segurança poderia, apenas, informá-la a um sistema de segurança.

3.2.5. Guardião

O Guardião tem como função a verificação do correto funcionamento dos elementos da arquitetura, evitando, por exemplo, que um dos elementos gerenciadores tenha seu código subvertido e prejudique o funcionamento do sistema. A pedido do Gerente Básico ou do Gerente-Mor, o Guardião pode até mesmo, encerrar um elemento que tenha sido subvertido.

O Guardião é criado pelo Gerente-Mor na inicialização do sistema juntamente com os Gerentes de Domínio, e é enviado para todos as estações dos gerentes de domínio, via tecnologia de agentes móveis e consulta a base de topologia geral (BTPG), pelo Gerente-Mor. Ele é o único componente, além dos Gerentes de Domínio e do Gerente-Mor, que tem permissão de solicitar a criação de novos Inspetores, Especialistas e inclusive de um novo Gerente de Domínio Básico ou específico.

Ao chegar na estação do Gerente de Domínio, o Gerente informa ao Guardião todo o itinerário a ser seguido por ele. Existem dois tipos de itinerário. O primeiro, como apresentado na figura 9, cria apenas uma instância do Guardião e este percorre todos os nós indicados no itinerário. O segundo tipo, mostrado na figura 10, cria "clones", isto é, várias instâncias, uma para cada nó. O segundo tipo é mais vantajoso do que o primeiro tipo quando o número de nós de um domínio for grande, já que a latência de percurso do Guardião entre o primeiro e o último nó pode ser muito alta, comprometendo assim o funcionamento do sistema. A desvantagem desse tipo de itinerário é o número de interrupções na estação do gerente a cada chegada de um Guardião. Quando o número de nós de um domínio for pequeno, é mais vantajoso o tipo de itinerário da figura 9, pois só haverá uma instância do Guardião dentro do domínio, e consequentemente haverá apenas uma interrupção na estação do Gerente. Neste tipo de itinerário o tamanho do pacote se mantém constante, pois quando o Guardião detecta alguma anomalia ele a reporta imediatamente ao Gerente e aguarda uma nova instância do elemento gerenciador com problemas. Se ele não detectar nenhuma anomalia não há reporte de informação ao Gerente e nem há armazenamento de informações dentro do Guardião.

Para que o Guardião possa ter um controle mais efetivo sobre os elementos gerenciadores, ele utiliza uma interface, parte obrigatória de todos os elementos gerenciadores. Essa interface deve oferecer funções de autenticação e autorização. Para que o Guardião saiba quais os elementos pertencentes a um determinado nó, ele chama um método do Registrador que possui as referências de todos os elementos gerenciadores presentes no nó, como pode ser visto na figura 11.

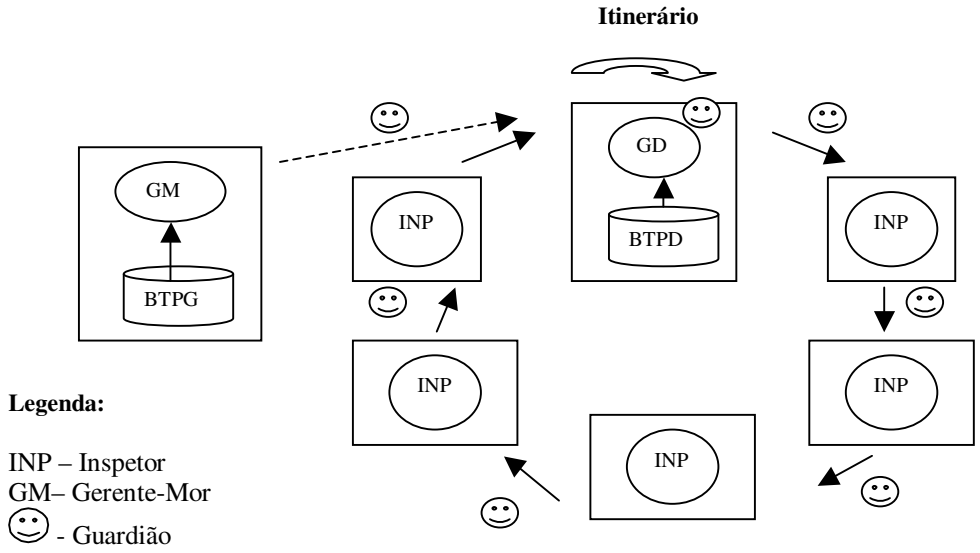


Figura 9 – Primeiro tipo de itinerário do Guardião

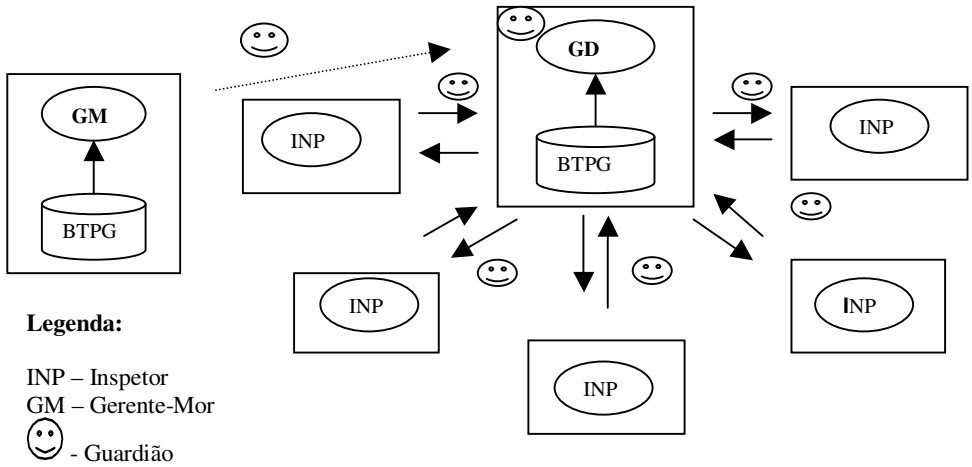


Figura 10 – Segundo tipo de itinerário do Guardião

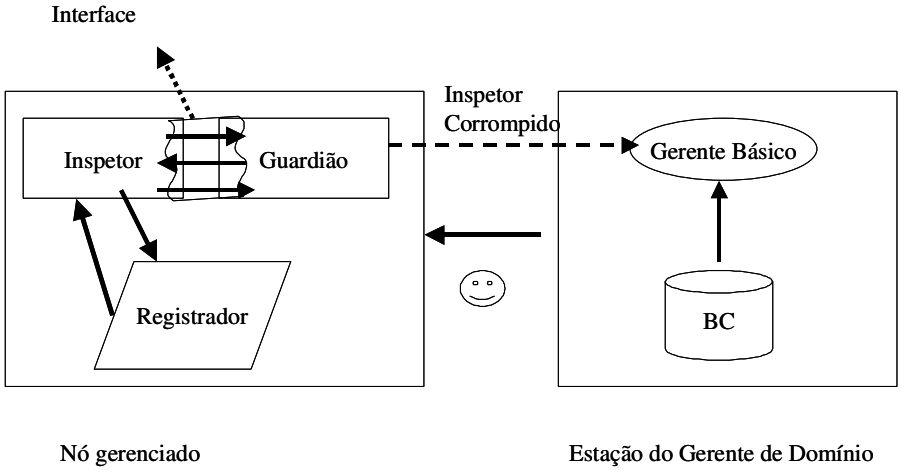


Figura 11 – Ciclo de vida do Guardião

A seguir, é descrito o ciclo de vida do Guardião.

1. Como parte do procedimento de inicialização do sistema é criada uma instância do Guardião na própria estação do Gerente-Mor e são enviadas outras instâncias do Guardião para cada Gerente de Domínio subordinado a este Gerente-Mor. O envio de uma instância do Guardião para cada Gerente de Domínio só ocorre após o envio dos Gerentes de Domínio para as estações de gerência.
2. O Guardião recebe do Gerente de Domínio o itinerário de viagem a ser percorrido dentro do domínio. Para isto, o Gerente de Domínio consulta a base BTPD para obter os endereços de todos os nós pertencentes a seu domínio e os passa como parâmetro para o Guardião.
3. Ao chegar em cada nó gerenciado, o Guardião se informa, através do Registrador, sobre quais elementos gerenciadores estão ativos. Caso o Guardião não encontre o elemento gerenciador que deseja verificar, conclui que o elemento em questão teve a sua operação abortada por algum motivo e solicita ao Gerente de Domínio Básico que este seja recriado. Caso o componente abortado seja um Gerente de Domínio Básico ou específico, o Guardião solicita ao Gerente-Mor a criação e o envio de uma nova instância desse Gerente para o nó em que ele esperava encontrá-lo. A seguir, estão descritos os passos seguidos pelo Guardião na auditoria de um elemento gerenciador.

- Para cada uma das instâncias, o Guardião inicia o processo de verificação obtendo a autorização de acesso. Esta autorização serve para evitar que o acesso aos componentes seja usado para prejudicar o funcionamento dos mesmos. Uma vez autorizado, o Guardião inicia a verificação da integridade do elemento gerenciador;
- A verificação de integridade é feita através de um *checksum no* elemento que está sendo auditado. Caso a verificação de integridade mostre que um determinado elemento gerenciador está corrompido, o Guardião encerra o mesmo, e no caso de falha do componente, o Guardião solicita ao Gerente de Domínio Básico a sua substituição, permanecendo no nó em questão até a chegada do novo elemento.
- Finalizada a verificação, o Guardião passa à próxima instância de elemento daquele nó;
- Uma vez que todos os elementos gerenciadores de um determinado nó tenham sido verificados pelo Guardião, este reporta ao Gerente de Domínio Básico o endereço do nó em que ele se encontra e a informação sobre a consistência dos elementos gerenciadores inspecionados. Após receber um *acknowledge* do Gerente de Domínio Básico, o Guardião migra para o próximo nó do itinerário.
- Caso o Gerente de Domínio Básico perceba que, após um determinado período de tempo, o Guardião não enviou nenhuma informação sobre um determinado nó, ele checa o funcionamento deste por meio do envio de mensagens sucessivas e periódicas e caso o Guardião não responda a estas mensagens, o Gerente de Domínio Básico conclui que o Guardião teve seu funcionamento abortado, cria uma nova instância deste e a envia para o nó em questão.

4. Quando o sistema é encerrado, após o Guardião ter percorrido todos os nós de seu itinerário, o Gerente de Domínio Básico ordena o término do Guardião. Ao receber esta ordem, o Guardião finaliza a verificação de funcionamento dos elementos de um determinado nó e retorna para a estação do Gerente de Domínio Básico ao invés de seguir para o próximo nó do itinerário.

3.3. Comunicação entre os elementos da AGAD

A comunicação entre os elementos da AGAD se dá por intermédio da troca de mensagem. Os três tipos básicos de mensagens trocadas entre os elementos da arquitetura são descritos a seguir.

3.3.1. Mensagens de controle

As mensagens de controle podem ser de três tipos :

- informação de disponibilidade (*Keep-Alive*): como exemplo, a notificação de disponibilidade enviada por um Gerente de Domínio específico para o Gerente de Domínio Básico;
- confirmação (*acknowledge*) enviada pelo Gerente de Domínio específico para um Inspetor específico, após a chegada de uma informação enviada por este;
- informações enviadas pelo Guardião ao Gerente de Domínio Básico.

3.3.2. Mensagens de alarme

Quando um componente Inspetor específico detecta uma anomalia, é gerada imediatamente uma mensagem de alarme que é enviada para o Gerente de Domínio específico da área em questão.

3.3.3 Mensagens de relatório

Dois tipos de relatórios devem ser disponibilizados: Relatórios de Estado e Relatórios para planejamento de Capacidade. O primeiro, consolida e apresenta as últimas informações disponíveis relativas ao desempenho dos elementos gerenciados, individualmente ou em conjunto das cinco áreas de gerência. O segundo tipo, Relatório para Planejamento de Capacidade, consolida informações relativas ao desempenho ao longo de um período de tempo, que pode ser definido pelo operador ou administrador. O detalhamento e a implementação desses relatórios foram deixados fora do escopo deste trabalho.

A partir destas mensagens de controle, alarme e relatório é possível fazer estatísticas, verificar tendências e, até, detectar a necessidade de geração de novos Especialistas ou aperfeiçoamento dos existentes.

3.4. Mecanismo de distribuição de código

O mecanismo de distribuição dos elementos gerenciadores aos nós gerenciados é realizado pelo Gerente-Mor e pelo Gerente de Domínio a partir do conhecimento das informações topológicas. O envio do código de todos os elementos é feito através de consultas às base BGC (Base Geral de Código) e BC (Base de Código). O Gerente-Mor consulta a base BGC e envia primeiro todos os Gerentes de Domínio e uma instância do Guardião para cada domínio, através do uso da tecnologia de agentes móveis. Após a ativação dos Gerentes de Domínio, estes distribuem aos nós pertencentes ao seu domínio, também via tecnologia de agentes móveis, os Inspetores, o Guardião e quando necessário os Especialistas. A comunicação entre os elementos da AGAD, que é via troca de mensagem, ocorrem em cinco situações: (i) entre o Gerente-Mor e o Gerente de Domínio, (ii) entre o Gerente de Domínio e o Inspetor, (iii) entre o Especialista e o Gerente de Domínio, (iv) entre o Guardião e o Gerente de Domínio, e (v) entre o Guardião e o Gerente-Mor.

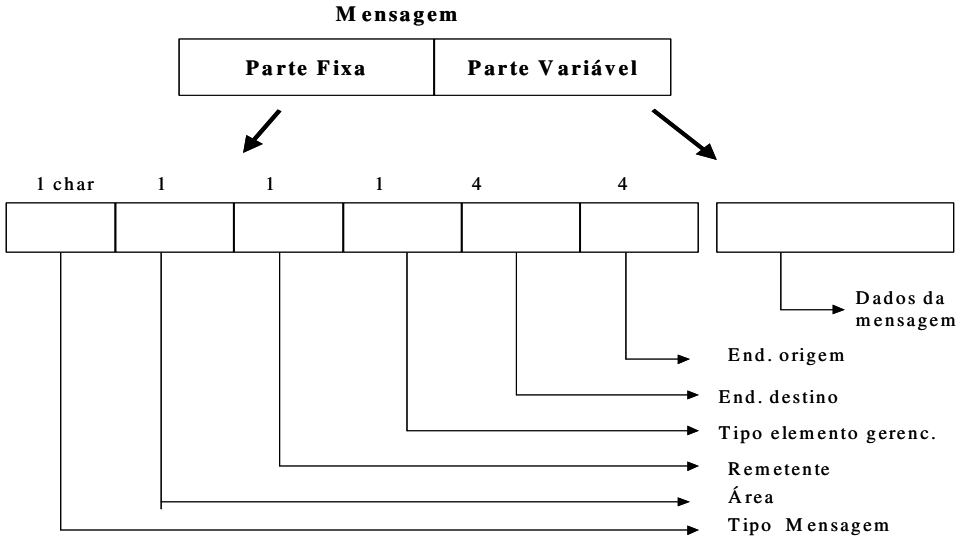


Figura 12 – Formato das mensagens trocadas entre os elementos da AGAD

O formato da mensagem é dividido em duas partes, uma fixa e uma variável. A parte fixa é composta por 6 campos e a parte variável é composta por 1 único campo de tamanho variável.

O formato das mensagens e o tamanho dos campos (em caracteres - *bytes*) são apresentados na Figura 12.

O campo Tipo da Mensagem informa se ela é mensagem do tipo controle, relatório ou alarme. O campo Área da Mensagem informa qual das 5 áreas de gerência (falha, desempenho, segurança, contabilização e configuração) esta mensagem pertence. O campo Tipo do Elemento Gerenciado informa se o elemento gerenciado é um roteador, uma estação, um *switch,* uma aplicação, etc. Os campos Endereço de Origem e Destino contêm, respectivamente, o endereço IP de origem e o de destino na mensagem.

O campo Remetente informa se a mensagem é oriunda de um Gerente de Domínio, Inspetor, Especialista ou Guardião. Já o campo Conteúdo da Mensagem, pertencente a parte variável, contém a informação que deseja-se transmitir.

3.5. Considerações Finais do Capítulo

Neste capítulo foi apresentada a arquitetura de um sistema de gerência distribuído baseado em tecnologia ativa, mais especificamente agentes móveis. O diferencial desta arquitetura em relação a outras similares é que ela reúne várias vantagens de uma só vez: maior eficiência da detecção e solução de problemas, em função da capacidade de processamento local dos Inspetores e Especialistas nos nós gerenciados, filtragem e consolidação de informações realizadas pelos Inspetores e Gerentes de Domínio antes destas serem enviadas para o elemento hierarquicamente superior, e o uso de tecnologia ativa que permite a construção de agentes de gerenciamento dotados de capacidade de processamento para monitorar tendências e erros, bem como enviar agentes para a realizações de tarefas específicas onde as mesmas forem necessárias.

Um item que pode ser considerado como trabalho futuro é o uso de mecanismos de Inteligência Artificial (IA) em ferramentas de análise das massas de dados coletados pelos Inspetores e nos Gerentes de Domínio específicos, com o objetivo de otimizar a classificação do Especialista mais adequado para realizar uma determinada tarefa.

Com o uso de ferramentas de análise das massas de dados coletadas pelos Inspetores, o processo de criação de novas versões de Especialistas específicos para cada anomalia encontrada, se tornará mais simples.

Outra proposta de trabalho para tornar o sistema mais robusto é a implementação da federação dos Gerentes de Domínio e dos Gerentes-Mores. Esta federação permitirá a cooperação entre domínios de gerência, o aproveitamento de códigos, a replicação de algumas bases de dados e a possibilidade de substituição, em caso de falha, por exemplo, de um Gerente de Domínio por outro que esteja mais próximo ou que esteja menos sobrecarregado.

Outra proposta, é a implementação do algoritmo de descoberta automática da topologia da rede descrito neste capítulo.

Capítulo 4 - Ambiente de Desenvolvimento e Implementação

Este capítulo detalha o ambiente de desenvolvimento usado para a implementação do protótipo da arquitetura descrita no capítulo 3, citando as razões que levaram a sua escolha e as vantagens e desvantagens decorrentes do seu uso. A seção 4.1 explica e justifica a escolha da linguagem e do ambiente usados. Os detalhes de implementação são abordados na seção 4.2.

4.1. Ambiente de Desenvolvimento

No desenvolvimento do protótipo utilizou-se o paradigma de Orientação a Objetos, desde a fase de análise até a sua implementação. O ambiente de desenvolvimento que foi escolhido tem como plataforma computadores pessoais, cuja configuração será apresentada no capítulo 5. O sistema operacional escolhido foi o Linux da Red Hat 7.2 com Kernel 2.4.7-10, e o Kit de Desenvolvimento JDK versão 1.3.1. A escolha teve como motivação algumas particularidades que lhe são pertinentes, como por exemplo: robusteza, ser independente de plataforma, a existência de modelos de segurança, que definem políticas de acesso dos programas aos recursos da máquina hospedeira, além de trazer pacotes prontos de métodos e classes como os pacotes java.net e java.rmi. Todas estas características tornam a linguagem Java muito apropriada ao desenvolvimento de sistemas distribuídos de código móvel. A escolha do ambiente de mobilidade, disponível em Java, veio solidificar a escolha desta como linguagem de programação dos elementos do sistema.

O ambiente de mobilidade precisava prover facilidades de comunicação com um nível mínimo de segurança, além de primitivas básicas de movimentação. Dentre os ambientes de mobilidade existentes, teve-se que escolher o melhor ambiente que retratasse o comportamento dos elementos e as possíveis situações de falha que pudessem ocorrer com eles, vindo a comprometer o funcionamento do sistema. O ambiente de mobilidade que satisfez à maior quantidade dos requisitos foi o μCode [34] também baseado na linguagem Java. Concebido para ser usado em conjunto com a linguagem Java, o μCode tem como objetivo superar algumas das desvantagens dos

sistemas de código móveis utilizados atualmente como, por exemplo, o ASDK (*Aglets Software Development Kit*) [35].

A arquitetura AGAD foi concebida para conter Gerentes, Inspetores e Especialistas das cinco áreas de gerência. Entretanto, como o objetivo do protótipo foi testar o funcionamento da comunicação e da substituição de um elemento dinamicamente, não foi necessário englobar elementos de todas as áreas, sendo portanto, desenvolvidos apenas elementos (Inspetores e Gerentes) relativos as áreas de falha e desempenho.

Esta seção está organizada em quatro subsessões. A sub-seção 4.1.1 descreve a linguagem de programação Java, a 4.1.2 apresenta o ambiente de mobilidade adotado, a 4.1.3 mostra as limitações do ambiente de mobilidade escolhido, enquanto que a 4.1.4 apresenta uma breve descrição sobre a API SNMP, desenvolvida em Java, que provê acesso à variáveis da MIB.

4.1.1. A Linguagem Java

A linguagem Java fornece um sofisticado ambiente para o desenvolvimento de sistemas distribuídos, isto é, fornece uma infra-estrutura que é composta por uma máquina virtual (JVM), um mecanismo de chamada remota de métodos (RMI), e uma série de serviços distribuídos. O objetivo de Java RMI é prover o suporte básico para sistemas distribuídos.

As vantagens de se utilizar Java para o gerenciamento distribuído são: provê solução para os problemas de heterogeneidade de nós pertencentes a uma rede a ser gerenciada, através do uso da máquina virtual Java (JVM) que permite, que programas escritos em Java sejam executados, sem alterações, em diferentes plataformas que implementem a JVM; permite o desenvolvimento orientado a componentes de software; e permite programação *multi-threading*. Porém existe uma desvantagem, que é o fato de Java ser uma linguagem interpretada, o que impõe uma limitação severa ao desempenho do sistema. Mesmo com o uso de JITs (*Just in Time Compilers*) ainda há um grande *overhead* associado ao modelo de gerenciamento de memória e de acesso aos dados.

Assim como no Corba [60] , clientes Java RMI chamam métodos de um objeto remoto através de chamadas locais a métodos de um *stub*, que representa localmente o objeto remoto. No RMI, quando o cliente quer chamar um método remoto em um

objeto remoto, na verdade ele chama um método da linguagem Java, que é encapsulado em um objeto substituto chamado stub. O stub reside na máquina cliente e não no servidor. O stub empacota, como um bloco de bytes, os parâmetros usados no método remoto. Esse empacotamento usa uma codificação independente do dispositivo para cada parâmetro.

Ao contrário de Corba, o fato do *bytecode* Java poder ser interpretado em qualquer arquitetura de software e hardware, permite que objetos sejam migrados de uma máquina para outra em ambientes heterogêneos. Através do RMI, objetos podem ser passados como argumentos por valor, já em Corba, eles são passados apenas por referência. Quando um objeto Java é transmitido para uma outra JVM através de RMI, o seu estado é convertido em uma seqüência de bytes (processo de serialização) e enviado para a JVM remota. Ao receber a requisição RMI, a JVM remota carrega o código da classe correspondente e instancia uma cópia do objeto original, deserializando o seu estado. A partir daí, alterações em qualquer uma das duas cópias dos objetos não são sincronizadas.

4.1.2. Ambiente de Mobilidade utilizado

A infra-estrutura utilizada é o μCode [35], que provê a mobilidade fraca, segundo a taxonomia de mobilidade de código de Fuggeta [9]. O ambiente da infra-estrutura é baseado na presença de servidores, denominados μServers, sendo executados sobre máquinas virtuais Java (JVM) em cada plataforma**.** O μCode utiliza μServers como ambientes de execução, sendo executados sobre máquinas virtuais Java (JVM) em cada plataforma. Um μServer é um ambiente computacional para *threads* móveis, que quando migram mantêm seu estado referente aos dados, mas não seus estados de execução, daí a mobilidade implementada ser classificada como fraca. Os μServers executam os agentes, que são *threads* móveis escritas em Java, representando as aplicações ativas. A operação do μCode é baseada na criação, cópia e migração dessas *threads*. O μCode, apesar de ser uma ferramenta nova, em fase de extensão foi utilizada com sucesso no desenvolvimento de uma aplicação de videoconferência como apresentada em [34].

As razões para a escolha da ferramenta μCode como ambiente de mobilidade foram não só o fato dela ser escrita em Java como, também, por ser um ambiente mais

leve e flexível em relação a outras ferramentas de código móvel, como o ASDK, por exemplo. As características que foram empregadas no desenvolvimento do μCode e que o fizeram ser um ambiente de mobilidade de código mais vantajoso do que outros ambientes estão descritas a seguir.

- **Minimização** – a ferramenta objetiva identificar e prover um mínimo conjunto de primitivas e abstrações para expressar diferentes estratégias de realocação de código.
- **Extensibilidade** - o μCode é implementado sobre a JVM na forma de uma camada simples, que não requer processamento adicional significativo em relação à máquina virtual.
- **Flexibilidade** - a ferramenta baseia-se no princípio que o programador deve ser capaz de selecionar diferentes estratégias para o mesmo código em diferentes situações, possuindo diferentes níveis de controle sobre o código móvel.
- **Portabilidade** - o fato do μCode ter sido desenvolvido em Java permite que ele seja executado em qualquer plataforma que suporte a JVM.

A unidade de migração do μCode é um grupo, composto por um conjunto de classes e objetos. Classes individuais e todas as demais classes por elas chamada podem ser adicionadas a um grupo, de forma que não seja mais necessária nenhuma busca à plataforma de origem. Os *threads* gerados a partir de classes recebidas são mantidos em um espaço de classes privativo, de forma a se evitar conflitos de nomes com outras classes locais. É possível, no entanto, publicar-se classes em um espaço de classe compartilhado associado ao μServer, de forma que as mesmas possam ser acessadas por outras classes do próprio μServer ou de μServers remotos.

A programação com o μCode é realizada utilizando-se três níveis de abstração: o núcleo, composto pelas classes *Group* e *ClassSpace* e pelas interfaces *GroupHandler* e *Mobile*; o nível intermediário, formado pela classe MuServer e o nível do usuário, que é formado pelas abstrações que foram implementadas pelo programador. A programação em nível de usuário é feita usando-se as primitivas disponíveis no nível intermediário, que poupa o programador de usar primitivas de mais baixo nível de abstração que estão presentes no núcleo.

Para o desenvolvimento do sistema AGAD utilizou-se a classe *Relocator* do *µCode* para o envio do Gerente-Mor, Gerente Básico, Gerente de Falha, Gerente de Desempenho, Inspetor Básico, Inspetor de Falha e Desempenho, e do Especialista de Falha. Dentro da classe *Relocator* foi utilizado o método *spawnThread* que permite a geração de um *thread* em outro *µServer*, através da utilização de um conjunto de classes especificadas pelo programador.

A classe *MuAgent,* que permite a implementação do paradigma de agente móvel, foi utilizada apenas para a mobilidade do elemento Guardião dentro de um domínio.

4.1.3. Limitações do µCode

Existem duas grandes limitações no µCode. A primeira, é que apenas a mobilidade fraca [34] é disponibilizada, uma vez que os *threads*, ao migrarem, salvam seu estado em relação aos dados, mas não o seu estado de execução. Existe suporte tanto para o envio quanto para a recuperação de códigos, fragmentos ou completos, com invocação síncrona ou assíncrona e execução imediata ou posterior. A segunda limitação, é que o núcleo do µCode não suporta nenhuma forma de comunicação direta entre dois *threads* móveis. Existem pacotes, que estão em fase de desenvolvimento pelo grupo que criou o µCode, que irão prover essa facilidade.

4.1.4 API SNMP (AdventNet)

Foi utilizada uma interface baseada em programas Java para a implementação de gerenciamento baseada no padrão SNMPv1, da empresa AdventNet, em sua versão 3.3. [12]. Essa API permite que os Inspetores enviem requisições SNMP para a consulta/atualização à MIB presente nos nós gerenciados. Essas mensagens SNMP são enviadas a um processo (*deamon*) do sistema operacional que é, na verdade, o agente SNMP. O *deamon* SNMP que foi utilizado é o que já vem instalado no Red Hat 7.2, pertencente ao pacote UCD-SNMP v4.2.1-7[12]. O agente SNMP é que, efetivamente, consulta/atualiza as variáveis da MIB que foram especificadas nas requisições SNMP emitidas pelos Inspetores. Todas as informações contidas na MIB são na verdade coletadas pelo sistema operacional e armazenadas em memória.

4.2. Implementação

O objetivo da fase de implementação é a construção de um protótipo que incorpore toda a funcionalidade necessária para analisar a viabilidade do sistema proposto.

O desenvolvimento do protótipo constitui na implementação de todas as classes modeladas durante as fases de análise e projeto, com seus respectivos atributos e métodos.

O protótipo implementado restringiu-se a um único domínio gerenciado por um Gerente de Domínio subordinado a um Gerente-Mor, e uma versão mais simplificada do funcionamento do Gerente-Mor, Gerente de Domínio Básico, Gerente de Domínio de Falha, Gerente de Domínio de Desempenho, Guardião, Inspetores de Falha e Desempenho e de um Especialista descritos no capítulo 3. Além disso, as interfaces gráficas do Gerente-Mor e do Gerente de Domínio estão fora do escopo desta implementação.

Esta seção apresenta todos os detalhes dos elementos da AGAD que foram implementados. A sub-seção 4.2.1 mostra as classes implementadas através do diagrama de classes e uma descrição detalhada das mesmas, a 4.2.2 exibe dois diagramas de seqüência: o primeiro detalhando a seqüência temporal de inicialização dos elementos implementados e o segundo detalhando a seqüência de funcionamento dos elementos da AGAD em um determinado instante do tempo, a 4.2.3 descreve as bases de dados implementadas, a 4.2.4 mostra as técnicas utilizadas na implementação do protótipo: tecnologia Socket x RMI, programação *multi-threading,* o mecanismo de atualização dos componentes de software (ex: Inspetor de Falha), e a comunicação entre os elementos implementados e a 4.3 apresenta as considerações finais do capítulo.

4.2.1 Classes implementadas no protótipo

Esta seção apresenta o diagrama das classes implementadas no protótipo (figura 13), que segue a notação de orientação a objetos UML [61], descrição dos seus objetivos e de seus principais métodos. Algumas dessas classes correspondem aos elementos descritos no capítulo 3.

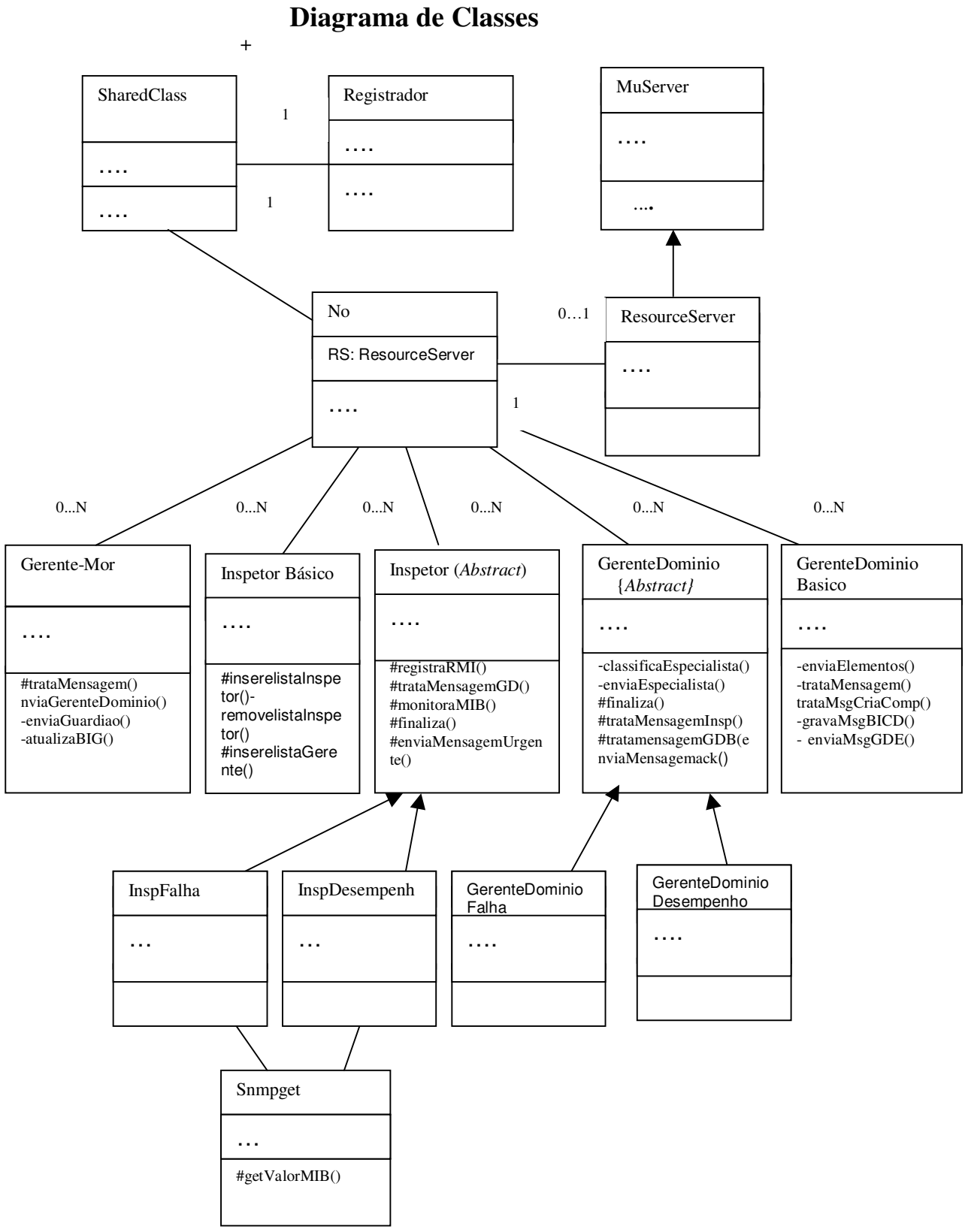


Figura 13 – Diagrama das classes implementadas

A seguir são apresentadas as descrições das classes implementadas e de seus principais métodos.

I. Classe IniciaAgad

Esta classe tem como função iniciar o sistema através da criação do Gerente-Mor.

Seus principais métodos estão descritos em seguida.

- trataMensagem – este método recebe mensagens dos elementos Gerente de Domínio Básico e do Guardião e verifica se a mensagem recebida é uma mensagem do tipo alarme, relatório ou controle. Além do tipo da mensagem, verifica qual é a área de gerência a que a mensagem recebida se refere. Este método aciona o método atualizaBIG.
- enviaGerenteDominio – este método é acionado assim que o Gerente-Mor é ativado. Ele é responsável por criar e enviar os elementos Gerente de Domínio Básico, de Falha e Desempenho para uma determinada estação de gerência. Para isto, o Gerente- Mor consulta a base de topologia chamada BTPG.
- enviaGuardiao– este método é acionado assim que o Gerente Mor recebe um acknowledge do Gerente de Dominio Básico informando que ele já esta ativo. Este método é responsável por criar e enviar o elemento Guardião para a estação do Gerente de Domínio Basico .
- atualizaBIG – este método atualiza a base BIG (base de informações de gerência) com as informações provenientes das mensagens do tipo relatório, alarme e controle enviadas pelo Gerente de Domínio Básico e Guardião. Antes da atualização, verifica-se a área de gerência a que pertence a mensagem recebida, pois cada mensagem é gravada na base tendo como um dos campos da chave a área da gerência.
- finaliza – este método atualiza para *null* a variável que representa o *thread* corrente, com a função de terminá-lo.

II. Classe GerenteDominioBasico

Esta classe é responsável por receber as mensagens enviadas por todos os Inspetores Básicos dentro do domínio gerenciado e distribuí-las, de acordo com a(s) área(s) contida(s) na mensagem, para o Gerente de Domínio de falha ou Desempenho. Além dessa funcionalidade, essa classe recebe informações dos Gerentes de Domínio de Falha e Desempenho e as envia para os Inspetores Básicos. Além disso, ela envia os Inspetores para cada nó gerenciado, e também interage com o Guardião.

Seus principais métodos estão descritos em seguida.

- enviaElementos – este é um dos primeiros métodos chamados depois que o Gerente de Domínio Básico é inicializado. Este método tem a função de enviar para os nós gerenciados o Inspetor Básico, o Inspetor de Falha, o Inspetor de Desempenho e o Guardião consultando a base de topologia do domínio (BTPD).
- trataMsgCriaComp – este método recebe mensagens do Guardião informando sobre algum problema em um determinado elemento. Ao receber esta mensagem, o Gerente de Domínio Básico recupera um novo elemento, idêntico ao que falhou da base de códigos BC e o envia para o nó em questão.
- trataMensagem – este método recebe mensagens consolidadas, do tipo relatório, do Inspetor Básico. Após o recebimento da mensagem, verifica-se a(s) área(s) de gerência contidas na mensagem. O Gerente de Domínio Básico chama o método gravaMensagemBICD que armazena a mensagem na base BICD e a envia para o Gerente de Domínio de Falha ou Desempenho através do método enviaMensagemGDE.
- gravaMsgBICD – grava a mensagem recebida na base de informações consolidadas do domínio (BICD).
- enviaMsgGDE – envia mensagens recebidas por um Inspetor Básico ou Guardião o Gerente de Domínio de Falha ou Desempenho.
- registraRMI – neste método o Gerente de Domínio Básico chama o programa *rmiregistry*, que é utilizado para estabelecer a conexão do Gerente de Domínio Básico com a própria rede e vinculá-lo a uma porta para que

conexões remotas possam ser estabelecidas com outros elementos da arquitetura.

- recebeMensagemDisp – este método recebe mensagens de disponibilidade (*Keep-Alive*) de todos os Inspetores, após a inicialização destes nos nós gerenciados, assim que o sistema é inicializado. Após o recebimento destas mensagens de todos os Inspetores enviados, o método criaGuardiao é acionado.
- criaGuardiao – este método cria e envia uma instância do Guardião que irá percorrer um itinerário pré-definido, configurado pelo administrador da rede.
- finaliza – este método atualiza para *null* a variável que representa o *thread* corrente, com o objetivo de terminá-lo.

III. Classe GerenteDominioFalha

Esta classe recebe informações do Gerente de Domínio Básico e alarmes dos Inspetores de Falha. Ela é responsável por verificar a existência de um Especialista para tentar alterar os estados operacional e/ou administrativo de umas ou mais interfaces de rede.

Seus principais métodos estão descritos em seguida.

- enviaMensagemIE - este método envia uma mensagem para um determinado Inspetor de Falha ordenando a execução de alguma tarefa.
- classificaEspecialista – este método consulta a base BCN (base de conhecimento) para verificar se existe um Especialista específico para solucionar o problema relacionado ao alarme recebido. No caso de existir um Especialista, o método enviaEspecialista é acionado e é passado como parâmetro para este método o IP do nó que enviou a mensagem.
- enviaEspecialista – esse método envia um Especialista da área de falha para um determinado nó.
- enviaMensagemACK – este método envia uma mensagem de confirmação (*acknowledge*) para um Inspetor de Falha confirmando o recebimento de uma mensagem enviada por este.

- finaliza – este método seta para *null* a variável que representa a *thread* corrente.

IV. Classe GerenteDominioDesemp

Esta classe pode receber informações do Gerente de Domínio Básico e pode receber alarmes dos Inspetores de Desempenho. Ele é responsável por verificar a existência de um Especialista capaz de informar qual o processo que está consumindo mais CPU da máquina.

Seus principais métodos estão descritos em seguida.

- classificaEspecialista – este método consulta a base BCN (base de conhecimento) para verificar se existe um Especialista específico para tomar alguma ação em relação à mensagem recebida. Nesse protótipo, o Especialista de Desempenho verifica qual o processo que está ocupando mais CPU em um deteminado momento. No caso de existir um Especialista, o método enviaEspecialista é acionado e é passado como parâmetro para este método o IP do nó que enviou a mensagem.
- enviaMensagemIE – este método envia uma mensagem para um determinado Inspetor de Desempenho ordenando a execução de outra tarefa. Para isso ele aciona, via RMI, o método trataMensagemGD.
- enviaEspecialista – esse método envia um Especialista de desempenho para um determinado nó.
- enviaMensagemACK – este método envia uma mensagem de confirmação (*acknowledge*) para um Inspetor de Desempenho confirmando o recebimento de uma mensagem enviada por este.
- finaliza – este método aualiza para *null* a variável que representa o *thread* corrente, com a função de terminá-lo.

V. Classe ResourceServer

Esta classe estende a classe μServer com um único objetivo: fornecer um objeto do tipo *Hashtable* que pode ser usado por *threads* em execução em um mesmo μServer

com a finalidade de permitir que elas compartilhem uma estrutura de dados, como um *array,* por exemplo.

VI. Classe No

Esta classe é instanciada em todas as máquinas utilizadas nos testes do protótipo, representando um nó da rede. Ao ser invocada, ela recebe como parâmetros o IP e a porta do servidor μServer. Essa classe cria um objeto RS da classe *ResourceServer* que é passado como parâmetro para um *MuServer* que é instanciado. A classe *SharedClass* é movida para o espaço de classe compartilhado da *ResourceServer*, e um objeto *SharedClass* é criado e inserido na *hashtable* de RS. Essa classe fica aguardando a chegada de grupos de códigos. Cada nó está diretamente associado a um μServer que é instanciado através dessa classe. O μServer funciona como um processo *listener* que fica escutando em uma determinada porta à espera da chegada de algum grupo de classes. Ao detectar a chegada de um grupo, formado por uma ou mais classes, na sua área compartilhada (*shared class space*), este grupo é movido para um espaço privativo de classe (*private class space*) que será comum a todas as classes deste grupo quando elas forem instanciadas. Após este procedimento, o μServer instancia a classe que é considerada a classe principal ou raiz desse grupo. A classe Server só tem o método chamado *main* mas, dentro deste método, existem os seguintes procedimentos:

```
// instanciando novo servidor de recursos RS
        ResourceServer RS = new ResourceServer();
// inicia um μServer

new Launcher( RS ).parseArgs( args, 0 );
// copia a classe SharedClass para o class space compartilhado de RS

RS.getPrivateClassSpace().copyClassTo( "SharedClass”);
RS.getSharedClassSpace() );
// mapeia instância de SharedClass para SHARED na hashtable

RS.SharedResources.put( "SHARED", new SharedClass () );
Registrador.AtualizaParteMutavel( "AtualizavelSync" );
// inicia a espera por grupos
RS.boot();
```

VII. Classe Registrador

Esta classe funciona como um *proxy* para todas as classes pertencentes a um nó (μServer). Qualquer objeto que chegue em um μServer deve chamar o método AtualizaParteMutavel dessa classe para atualizar a sua referência. Através dessa classe, qualquer atualização de código de um determinado objeto torna-se transparente para todos os demais, pois quando um objeto quer se comunicar com outro ele chama a classe *SharedClass* seguido do nome do método do objeto alvo.

Seus principais métodos estão descritos em seguida.

- atualizaParteMutavel – este método é responsável por registrar, no μServer em questão, a referência de uma classe que ele recebe como parâmetro. O parâmetro recebido é uma *string* com o nome da classe. A classe Registrador deve declarar uma variável do tipo da interface que a classe que está se registrando implementa.
- informaRef – este método informa ao Guardião a referência de todos os elementos presentes em um determinado nó.

VIII. Classe SharedClass

Esta classe é que realmente chama a classe Registrador.

IX. Classe Guardião

A classe Guardião é responsável por verificar se os elementos/componentes que devem estar ativos estão, e caso não estejam ele deve enviar uma mensagem ao Gerente de Domínio Básico. A comunicação entre os elementos e o Guardião é realizada através da classe Registrador.

Foi implementada uma versão simplificada do Guardião, pois esta implementação não contempla a verificação se um elemento gerenciador está corrompido ou não, e nem o procedimento de autorização de acesso quando o Guardião se comunica com o elemento gerenciador.

Seus principais métodos estão descritos em seguida.

- verificaDisponibilidade – este método aciona o método informaRef da classe Registrador para saber a referência de todos os elementos da AGAD presentes em um determinado nó, independente de seu estado de execução, com o objetivo de verificar se um determinado elemento está ativo através da chamada de um método pertencente a interface IGuardiao que permite a comunicação do Guardião com o elemento da AGAD.
- testaElemento - Como todos os elementos foram implementados através de *threads* e um *thread* pode estar em um estado desconhecido, o Guardião aciona o método *isAlive()* para determinar se um *thread* ainda existe, já que este método retorna verdadeiro para qualquer estado em que um *thread* se encontre.
- enviaMensagemCriacaoComp – este método é responsável por acionar, via RMI, o método trataMensagemCriacaoComp para enviar ao Gerente de Domínio Básico uma mensagem informando que um elemento em um determinado nó falhou. Ele envia também o tipo do elemento que falhou e o IP do nó. Quando o Gerente de Domínio Basico recebe esta informação, ele recupera um novo elemento, igual ao que falhou, da base de códigos BC e o envia para o nó em questão.
- finaliza – este método atualiza para *null* a variável que representa o *thread* corrente, com o objetivo de terminar a sua execução.

XI. Classe InspBasico

O Inspetor Básico tem como função receber dados do tipo relatório dos Inspetores de Falha e Desempenho, filtrá-los e consolidá-los antes de enviá-los através de uma mensagem para o Gerente de Domínio Básico. O Inspetor Básico também recebe dados dos Gerentes Básico, de Falha e Desempenho.

Seus principais métodos estão descritos em seguida.

- consolidaDados – recebe como parâmetro várias informações coletadas pelos Inspetores de Falha e Desempenho e as consolida em uma única mensagem a ser enviada via Socket para o Gerente de Domínio Básico.

- removelistaGerente – este método lê os dados inseridos remotamente pelos Gerentes de Domínio Básico, de Falha e Desempenho em um array e aciona o método enviadadosInspetor.
- inserelistaGerente – este método é chamado remotamente pelos Gerentes de Falha e Desempenho em um *array.*
- enviadadosInspetor – este método verifica qual o Inspetor específico destinatário da mensagem recebida e envia a mensagem para ele.
- inserelistaInspetor – este método insere dados recebidos pelos Inspetores de Falha e Desempenho em um *array*.
- removelistaInspetor – este método lê um array que contém os dados inseridos pelos Inspetores de Falha e Desempenho.
- finaliza – este método atualiza para *null* a variável que representa o *thread* principal do Inspetor de Falha.

XII. Classe InspFalha

Esta classe, dentro de um intervalo de tempo pré-estabelecido pelo administrador da rede, via Gerente de Falha, faz uma leitura na MIB-II (RFC1213-MIB) para verificar os estados de operação (variável *ifOperStatus*), de administração (variável *ifAdminStatus*), a taxa de erro na saída de pacotes (variável *ifOutErrors*), e na entrada (variável *ifInErrors*) de todas as interfaces do nó hospedeiro.

Seus principais métodos estão descritos em seguida.

- enviaMensagemUrgente **-** este método envia alarmes para o Gerente de Domínio de Falha via RMI.
- trataMensagemGD – este método recebe mensagens enviadas pelo Gerente de Domínio de Falha.
- registraRMI – dentro deste método o Inspetor de Falha chama o programa rmiregistry que é utilizado para estabelecer a conexão do aplicativo servidor com a própria rede e vinculá-lo a uma porta para que os elementos possam estabelecer conexão com o Inspetor de Falha.
- monitoraMIB – através deste método o Inspetor de Falha instancia a classe snmpget e passa como parâmetros as variáveis (*ifOperStatus, ifAdminStatus*,

ifInErrors, ifOutErrors) com o objetivo de saber os estados operacional e administrativo de todas as interfaces pertencentes a um determinado nó e se está ocorrendo erro em algumas delas.

- finaliza – este método atualiza para *null* a variável que representa o *thread* principal do Inspetor de Falha, com o objetivo de terminá-lo.

XIII. Classe InspDesempenho

Esta classe, dentro de um intervalo pré-definido pelo administrador da rede, via Gerente de Domínio de Desempenho tem a função de monitorar o percentual de ocupação de CPU do nó hospedeiro, através da MIB *HOST-RESOURCES-MIB* (RFC2790) [6].

Seus principais métodos estão descritos em seguida.

- enviaMensagemUrgente **-** este método envia mensagens urgentes(alarmes) para o Gerente de Domínio de Desempenho via RMI.
- trataMensagemGD – este método recebe mensagens enviadas pelo Gerente de Domínio de Desempenho.
- registraRMI – neste método o Inspetor de Desempenho chama o programa *rmiregistry* que é utilizado para estabelecer a conexão do aplicativo servidor com a própria rede e vinculá-lo a uma porta para que os outros elementos da AGAD possam estabelecer conexão com o Inspetor**.**
- monitoraMIB – através deste método o Inspetor de Falha chama o método getValorMIB de um objeto da classe SnmpGet para saber o percentual total de utilização de CPU do nó.
- finaliza – este método seta para *null* a variável que representa o *thread* principal do Inspetor de Desempenho.

XIV. Classe EspecFalhaInterf

Esta classe tem a função de tentar atualizar as variáveis *ifOperStatus* e *ifAdminStatus* de uma determinada interface do estado “Down” para o estado “Up”.

Seus principais métodos estão descritos em seguida.

- ativaestadoInterface – este método tenta atualizar o valor da(s) variável(eis) *ifOperStatus* e/ou *ifAdminStatus* para "Up".
- finaliza – este método atualiza para null a variável que representa o *thread* corrente.

XV. Classe SnmpGet

Define os métodos de acesso a MIB-II. Essa classe pertence a API AdventNet e recebe como parâmetros as variáveis da MIB que devem ser monitoradas: *ifOperStatus, ifAdminStatus, ifInErrors, ifOutErrors da MIB RFC1213-MIB* relativas as interface(s) de rede de um determinado nó.

Seu principal método é descrito em seguida.

- getValorMIB – este método recebe os parâmetros que devem ser lidos da MIB.

XVI. Classe SnmpSet

Define os métodos de acesso as MIBs. Esta classe pertence a API AdventNet e recebe como parâmetros as variáveis *ifOperStatus* e *ifAdminStatus* relativos as interface(s) de rede de um determinado nó. Esta classe é acionada pela classe EspecFalhaInterf.

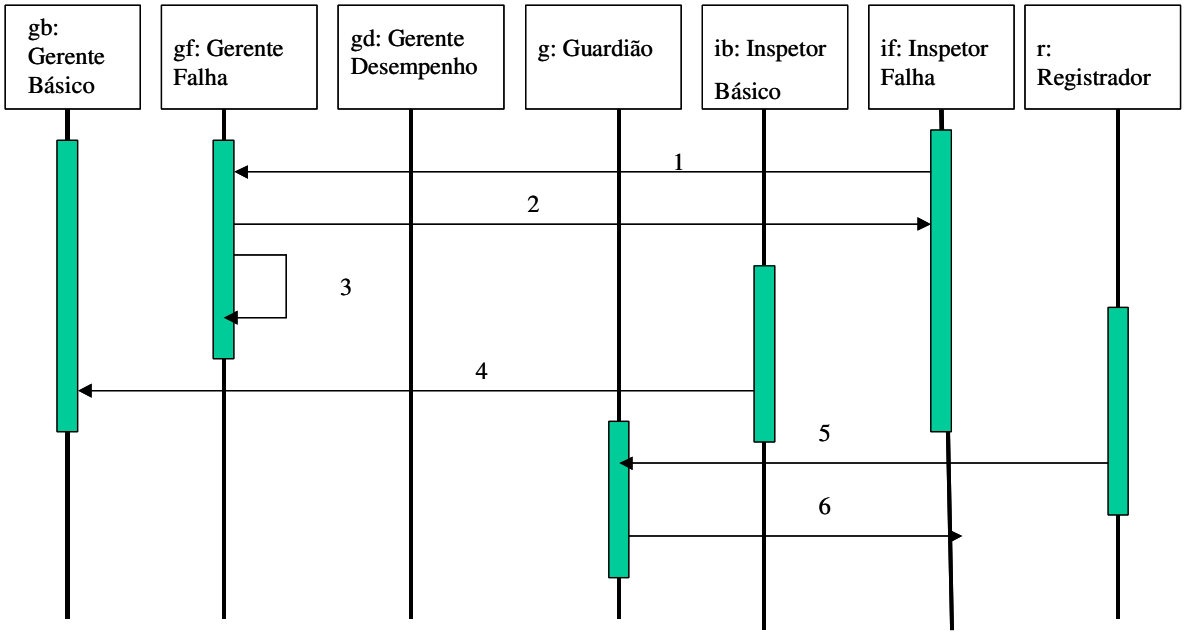
Seu principal método é descrito em seguida.

- setValorMIB – este método recebe os parâmetros e os valores que devem ser atualizados na MIB.

4.2.2. Diagrama de Seqüência

O diagrama de seqüência apresentado nessa sub-seção, seguindo a notação de [61], é do tipo de interação de objeto que enfatiza a seqüência temporal sobre os relacionamentos entre objetos. No diagrama de seqüência, o eixo das ordenadas representa o tempo e o eixo das abcissas representa as mensagens trocadas entre os objetos no decorrer do tempo. O diagrama da figura 14 apresenta um exemplo da

seqüência de mensagens trocadas entre os objetos em um determinado instante do tempo.



Legenda:

1 – Alarme de falha de interface
2 – ACK
3 – Classifica Especialista
4 – Informações consolidadas
5 – Referências dos componentes
6 – Estado de integridade do Inspetor de Falha

Figura 14 - Comportamento dos componentes da AGAD em um determinado instante do tempo

4.2.3 Bases de dados

Todas as bases de dados criadas foram implementadas para esta primeira versão como arquivos-texto. As bases implementadas estão descritas a seguir.

- Base de Conhecimento (BCN) – contém a associação entre os diversos tipos de alarmes e/ou tarefas e os Especialistas.
- Base de Informações Consolidadas do Domínio (BICD) – contém informações de alarme e de relatório sobre um determinado domínio.
- Base de Topologia de Domínio (BTPD) – Essa base contém a topologia do domínio gerenciado.

- Base de Topologia Geral (BTPG) – Esta base contém a topologia geral do domínio. Como o protótipo só contempla um único domínio, o conteúdo desta base é igual ao da base BTPD.
- Base de Código do domínio (BC) – corresponde a um sistema de arquivo (*filesystem*) do sistema operacional onde estão armazenadas todas as classes da AGAD.

4.2.4. Técnicas utilizadas na implementação do protótipo

Esta seção apresenta as técnicas utilizadas na implementação do protótipo. Essas técnicas envolvem a comunicação utilizada entre os componentes, as vantagens da utilização da programação *multi-threading*, as limitações encontradas no ambiente de mobilidade μCode e como essas limitações foram contornadas, e descreve como ocorre o processo de atualização de um componente em execução.

(i) Comunicação Socket e RMI

A comunicação via *Socket* é feita através do estabelecimento de um canal de comunicação bidirecional entre serviços hospedados em diferentes dispositivos da rede, identificados pelos seguintes elementos: endereço-origem + porta origem, endereço destino + porta destino. Cada serviço associa um *socket* a sua outra ponta da conexão. Para se comunicarem, o cliente e o servidor leêm e escrevem no *socket* associado à conexão.

No java existe uma classe chamada *socket*, pertencente ao pacote java.net que é usada para representar uma conexão entre o cliente e o servidor.

Já a comunicação realizada através do RMI (*Remote Method Invocation*) permite que um aplicativo chame métodos e acesse variáveis dentro de outro aplicativo (que pode estar em execução em diferentes ambientes Java ou em sistemas completamente diferentes) e troque objetos através de uma conexão de rede, sem a necessidade de conhecer a pilha de protocolo que está sendo utilizada. O RMI é um mecanismo mais sofisticado para a comunicação entre objetos Java distribuídos do que uma conexão *socket*, pois os mecanismos e protocolos através dos quais os objetos se comunicam são

definidos e padronizados. Em compensação o mecanismo utilizado pelo RMI é mais complexo do que o utilizado pelo Socket.

(ii) Programação *Multi-Threading*

Existem duas formas de se trabalhar em ambientes multi-programados: processo ou *thread*. Um *thread* representa um fluxo de processamento, mas é diretamente associada ao processo que a criou, compartilhando com ele pelo menos as áreas de código e dados. Há dois tipos de *threads*: os mantidos pelo sistema operacional (*Kernel threads*) e os mantidos por bibliotecas de usuário (*user threads* ou *green threads*). Em ambos os casos, há vantagens sobre o uso de processos, tais como um menor *overhead* de chaveamento de contexto, pois leva menos tempo para a criação de um novo *thread* em um processo existente do que abrir um novo processo. Em compensação, o preço pago é alto: como existe acesso simultâneo a recursos da máquina, há a necessidade de controle do acesso e sincronização entre *threads* que disputam os recursos. Portanto, se existe um aplicação ou função que deva ser implementada como um conjunto de unidades de execução relacionadas, é muito mais eficiente desenvolver uma coleção de *threads* do que uma coleção de processos separados.

A JVM faz uso intensivo de *multi-threading*, e por isso há primitivas de sincronização embutidas na linguagem para acesso a recursos compartilhados, como regiões de memória e acesso a disco. A princípio, qualquer objeto Java pode ser usado como elemento de sincronização. A especificação Java define o chamado monitor de acesso, ou seja, uma vez que um *thread* adquire o monitor, outros *threads* que tentem fazer o mesmo ficam com sua execução bloqueada até que o proprietário do monitor o libere total ou parcialmente. Nesse momento, um dos *threads* bloqueados é escolhido e reativado, ganhando acesso ao monitor.

A biblioteca Java fornece uma classe do tipo *thread*. Quando um *thread* é construído, o código e os dados que definem seu contexto são especificados por um objeto passado para seu construtor. O construtor da classe *thread* recebe como argumento uma instância da interface *Runnable*, que é criada por uma classe que implementa essa interface.

Decidiu-se utilizar *threads* para a implementação de todos os elementos descritos no capítulo 3, pois é muito mais eficiente desenvolver uma coleção de *threads* do que uma coleção de processos separados. Dessa forma, todos os elementos descritos foram implementados através de um ou mais threads utilizando a interface *Runnable* do Java.

O *thread* de um elemento que realiza monitoramento de uma ou mais variáveis da MIB, teve seu código desenvolvido especificamente para o comportamento do parâmetro em questão, visando minimizar o seu tempo de execução. Como exemplo, o Inspetor de Falha é composto de um *thread* responsável por monitorar três variáveis referentes a interface de um nó.

Todos os *threads* dos elementos implementados podem ser encerrados por um componente hierarquicamente superior através da chamada do método **finaliza**, comum a todos esses elementos. Este encerramento é usado quando do encerramento das atividades de um elemento em questão devido a dois possíveis motivos: quando o sistema é finalizado ou quando da substituição do código de um componente por uma nova versão.

(iii) Atualização de Componentes de Software

Um componente de *software* em tempo de execução está associado a uma ou mais classes que são acessadas através de interfaces que podem ser descobertas em tempo de execução. No caso desse protótipo, pode ocorrer dois tipos de atualização: a primeira refere-se a substituição de um componente por um correspondente que possui novas funcionalidades. Quando esse novo componente chega em um nó (ex: um novo Inspetor de Falha) ele deve obter a referência do seu componente correspondente anterior, o Inspetor de Falha que está em execução, através de consulta ao método informaRef da classe **Registrador**. Esta classe define um método que funciona como um *proxy,* e que é chamado pelas demais classes quando querem executar um método atualizável.

Ao chegar no nó, esse novo componente chama o método responsável por informar as referências de todos os componentes presentes em um nó, passando como parâmetro o nome do componente desejado, recebendo a referência do mesmo. Após obter esta referência, o novo componente deve acionar o método finaliza, presente em

todos os componentes da AGAD, para encerrar a execução do componente anterior e substitui-lo por uma nova versão de código. Após esta finalização, o novo componente deve se registrar na classe Registrador através do método AtualizaParteMutavel. O método AtualizaParteMutavel é que atualiza o código do método atualizável. A partir desse registro, todas os outros objetos que fazem referência a este componente farão acesso a nova referência, de forma "transparente". Com esta facilidade novos componentes podem ser desenvolvidos e atualizados dinamicamente em um sistema que esteja em funcionamento. Maiores detalhes sobre o mecanismo de atualização utilizado na implementação desse protótipo estão descritos no Apêndice A.

(iv) Comunicação entre os elementos da AGAD

Como o núcleo do μCode , atualmente, não permite que *threads* móveis se comuniquem diretamente, a solução encontrada foi a de criar uma classe *Server* que fornece um objeto de uma classe chamada *SharedInteger* que possui métodos sincronizados de leitura e escrita a uma área compartilhada entre dois *threads* que desejam se comunicar. O exemplo clássico do produtor e consumidor, que vem no pacote μCode, utiliza a solução acima para a comunicação entre os *threads* móveis consumidor e produtor para que eles possam ter acesso de leitura e escrita a uma mesma estrutura de dados. Toda a comunicação entre elementos pertencentes a um mesmo nó (μServer), ou seja, entre o Inspetor Básico e os Inspetores específicos em execução é feita através da classe InspetorBasicoSync que contém os métodos cujo acesso é sincronizado. Já a comunicação entre o Gerente Básico e os Gerentes específicos é feita através da classe GerenteDominioSync.

A comunicação entre elementos de nós diferentes, ou seja, entre Inspetores e os Gerentes é realizada por *Socket*, através de dois pares: endereço origem + porta origem e endereço destino + porta destino. O pacote java.net contém as interfaces, classes e exceções tratadas pelo *Socket*.

4.3. Considerações Finais do Capítulo

Foram apresentados, neste capítulo, os ambientes de desenvolvimento e de implementação do protótipo da arquitetura AGAD, descrita no capítulo 3, além de

explicitadas as razões para a escolha da linguagem Java e do ambiente de mobilidade μCode.

Uma sugestão de trabalho futuro seria o desenvolvimento de um módulo de inicialização automática. Atualmente é necessário disparar o μServer manualmente em cada uma das máquinas a serem gerenciadas, além do preenchimento de um arquivo de configuração com a lista dessas máquinas, simbolizando a base de topologia da rede. No próximo capítulo serão apresentados os resultados dos experimentos com o protótipo.

Capítulo 5 - Testes e análise dos resultados obtidos

Este capítulo tem como objetivo apresentar os testes e a análise de resultados a partir da execução do protótipo descrito no capítulo 4. O protótipo foi implementado apenas para um único domínio, e os testes foram realizados apenas com componentes da área de falha.

Foram realizados três testes diferentes: (i) Latência de carga da arquitetura AGAD, (ii) Latência de detecção de alarme de falha utilizando-se dois cenários: AGAD e Perl (SNMP) e (iii) Latência entre a detecção de um alarme de falha e o início da execução do Especialista.

Os testes foram realizados nas plataformas de hardware que estão apresentadas na Tabela 1.

Plataforma	CPU	Clock (MHz)	Memória Cache (KBytes)	Memória RAM (MBytes)
1	**K6-II**	**500**	**-**	**256**
2	**K6-II**	**500**	**-**	**256**
3	**Pentium II**	**350**	**512**	**256**

Tabela 1 - Plataformas utilizadas nos testes

Em relação as plataformas utilizadas, todas posuem placas de rede de 100 Mbps. A plataforma 1 possui uma placa de rede adicional de 10Mbps fora da rede de teste que serviu para simular falha de queda de enlace desabilitando sua operação diretamente pelo console (*Shell*) do Linux. A rede é Fast Ethernet baseada em um Hub 10/100 Mbps marca 3Com modelo "Dual Speed Hub 8". As versões do Kernel dos sistemas operacionais envolvidos foram 2.4.7 com distribuição Red Hat versão 7.2 (plataforma 1) e 2.4.5 com distribuição Conectiva 6.0 (plataforma 2 e 3).

Em todos os testes foram colocados colocado em execução, em nível usuário, apenas os processos referentes aos programas que estavam sendo testados: a JVM, o ambiente de mobilidade μCode e o *deamon* SNMP (figura 15). Os três testes, exceto o primeiro cenário do 2º. teste, consistiram de uma única rodada com 30 medições, e a média aritmética.

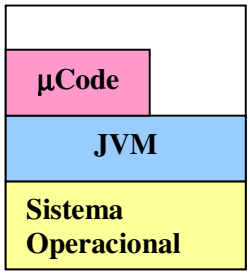


Figura 15 – Ambiente de Software

5.1. Latência de carga da arquitetura AGAD

Este teste tem o objetivo de avaliar o tempo de carregamento dos componentes da arquitetura AGAD. O teste consiste em inicializar o sistema disparando o Gerente-Mor na plataforma 3. Assim que, o Gerente-Mor começa sua execução envia o Gerente Básico, o Gerente de Falha e de Desempenho, o Inspetor Básico, o Inspetor de Falha, o Inspetor de Desempenho, o Especialista de Falha e o Guardião à plataforma 2. O envio é realizado através do método *SpawnThread* do Mucode, conforme a figura 16. Uma vez que os componentes são carregados na plataforma 2, o Gerente Básico cria e envia os Inspetores Básico e de Falha para a plataforma 1. Ao chegar na plataforma 1, o Inspetor de Falha inicia sua execução transmitindo uma mensagem de reconhecimento ao Gerente-Mor instalado na plataforma 3. O tempo medido corresponde ao tempo decorrido entre o início do envio dos componentes pelo Gerente-Mor à plataforma 2 e a chegada da mensagem de reconhecimento ao próprio Gerente-Mor. Este tempo inclui a criação e o posicionamento do Gerente Básico, do Gerente de Falha, do Gerente de Desempenho, do Inspetor Básico, do Inspetor de Falha, do Inspetor de Desempenho e, também do Guardião.

Cada uma das medidas realizadas foram efetuadas à partir da reinicialização do Sistema, ou seja, para cada medida o Gerente-Mor foi executado na linha de comando da console do sistema operacional, com o propósito de eliminar o efeito *Cache* de classe que resultaria em tempos de atraso menores.

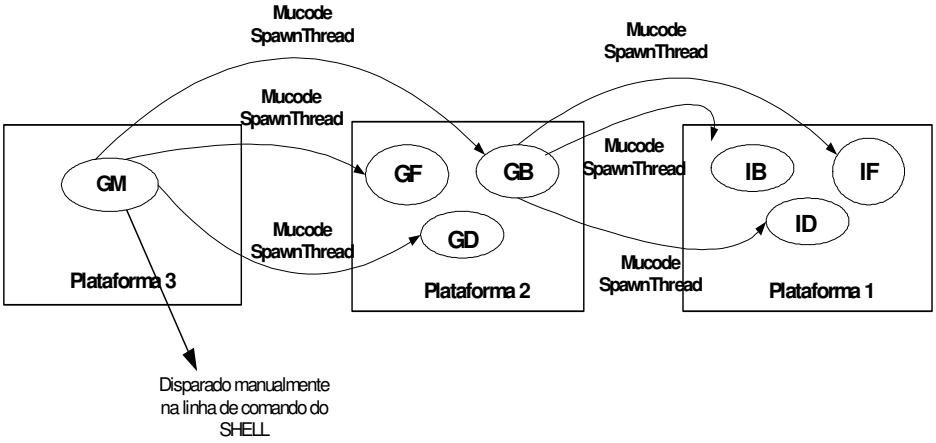


Figura 16– Carga da arquitetura AGAD

Única Rodada					
Medidas	**Tempo**	**Medidas**	**Tempo**	**Medidas**	**Tempo**
1ª.	2207	11ª.	2116	21ª.	2175
2ª.	2093	12ª.	2443	22ª.	2424
3ª.	2121	13ª.	2135	23ª.	2166
4ª.	2081	14ª.	2360	24ª.	2149
5ª.	2204	15ª.	2287	25ª.	2118
6ª.	2575	16ª.	2719	26ª.	2223
7ª.	2135	17ª.	2295	27ª.	2246
8ª.	2486	18ª.	2302	28ª.	2108
9ª.	2118	19ª.	2199	29ª.	2124
10ª.	2244	20ª.	2179	30ª.	2234
Media		**2242**			

Tabela 2 – Medições do 1º teste (em ms)

Os resultados apresentados neste teste mostram que o mecanismo de distribuição automática dos componentes da arquitetura é muito mais rápida e segura se comparado a uma instalação desse sistema realizada pelo administrador da rede. A instalação automática tem a vantagem de ser imune a erros oriundos de falhas humanas, além de eliminar o trabalho árduo da instalação manual para redes de médio e grande porte.

Os tempos também permitem concluir que é possível realizar, automaticamente, configuração de componentes de software em tempo hábil sem causar transtornos aos serviços de gerenciamento prestados.

5.2 Latência de detecção de um Alarme de Falha

O objetivo deste teste é medir a latência na detecção de alarme gerado por uma queda de enlace. Objetivando comparar a latência introduzida pela Arquitetura AGAD e aquela produzida somente pelo sistema operacional, foi montado dois cenários: (i) cenário AGAD e (ii) cenário Perl .

No primeiro cenário, o Inspetor de Falha que fica monitorando o estado operacional da interface de rede (variável *ifOperStatus* da MIB-II da plataforma 1), ao detectar a queda da interface, gera uma mensagem de alarme ao Gerente de Falha. O alarme é enviado, via Socket, ao Gerente de Falha hospedado na plataforma 2. Assim que o Gerente de Falha recebe o alarme, este envia uma mensagem de reconhecimento de volta ao Inspetor de Falha na plataforma 1.

No segundo cenário, o esquema é idêntico ao anterior, excetuando o fato de que é utilizado um script em Perl que aciona o agente SNMP utilizando-se de bibliotecas em C para enviar e receber *traps*. Além disso, as bibliotecas tem acesso a MIB para a leitura do estado operacional da interface para a detecção de falhas.

Cenário AGAD

Neste cenário foi utilizado o Gerente de Falha e o Inspetor de Falha juntamente com a API AdventNet SNMP para acesso ao agente SNMP da plataforma 1 de modo a verificar o estado operacional da interface eth1 que foi monitorada durante os testes.

Rodada	Tempos Médios
1ª.	75
2ª.	77
3ª.	70
4ª.	71
5ª.	69
Média Total	73

Tabela 3 - Medições do 2º teste referentes ao 1º cenário (em ms)

Cenário Perl

Agente SNMP, neste caso, foi o *deamon* SNMP do próprio sistema operacional. O agente SNMP gera uma mensagem de ***Trap*** indicando a falha em uma das interfaces do nó gerenciado. Um *script* na linguagem Perl [5] foi desenvolvido para coletar os tempos apresentados na tabela 4.

Única Rodada					
Medidas	**Tempos**	**Medidas**	**Tempos**	**Medias**	**Tempos**
1ª.	17,096	**11ª.**	27,884	**21º.**	19,117
2ª.	27,338	**12º.**	19,750	**22º.**	26,311
3ª.	28,884	**13º.**	25,192	**23º.**	17,611
4ª.	21,041	**14º.**	23,919	**24º.**	30,960
5ª.	16,482	**15º.**	20,966	**25º.**	24,011
6ª.	31,896	**16º.**	25,471	**26º.**	21,970
7ª.	17,164	**17º.**	19,884	**27º.**	22,864
8ª.	26,778	**18ª.**	24,727	**28º.**	22,649
9ª.	23,506	**19ª.**	22,485	**29º.**	24,355
10ª.	22,074	**20º.**	26,370	**30º.**	18,640
Média			**23,2465**		

Tabela 4 - Medições do 2º teste referentes ao 2º cenário (em ms)

Neste teste, constatou-se que a latência de detecção de um alarme via *trap* SNMP é menor do que aquela observada na arquitetura proposta devido a carga computacional adicional introduzida pela Máquina Virtual Java, pela infraestrutura de mobilidade de código (μCode) e pela aplicação AGAD. Entretanto, este atraso adicional não inviabiliza o emprego da arquitetura AGAD. Essa carga computacional pode ser constatado no esquema da figura 17. O primeiro esquema corresponde às camadas envolvidas no cenário AGAD e o segundo esquema corresponde as camadas envolvidas no cenário Perl.

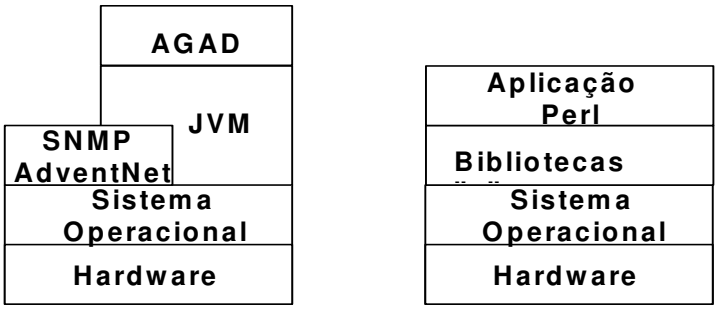


Figura 17 – Camadas envolvidas nos Cenários AGAD e Perl

O uso do µCode e da Máquina Virtual Java como infraestrutura básica para a Arquitetura AGAD impõe um retardo adicional, conforme demonstram os resultados da tabela 3. Entretanto, esses valores adicionais não inviabilizam o emprego da Arquitetura na gerência de redes porque os tempos de reação exigidos em tal ambiente são ainda compatíveis aos tempos de atrasos medidos. Além disso, é esperado que, com o esquema distribuído de gerenciamento juntamente com a Tecnologia Ativa, o tráfego de gerência e a carga computacional para o processamento desse tráfego sejam minimizados em decorrência da filtragem e da consolidação de informações que ocorrem de forma distribuída na rede.

5.3. Latência entre a detecção de um alarme de falha e o início da execução do Especialista

Este teste tem como objetivo medir a latência desde a tomada de decisão, por um Inspetor de Falha de enviar um alarme até o inicio da execução do Especialista de Falha de Interface na plataforma 1. O intervalo de tempo considerado nas medições deste teste é relevante para a implementação do gerenciamento de falha, por exemplo, pois determina o limite de tempo mínimo a partir do qual ações podem, de fato, serem desencadeadas com o objetivo de reparar ou evitar alguma anomalia. Note-se que este tempo é uma estimativa, pois com o crescimento da base de Especialistas o tempo de seleção deve crescer. Neste teste, foi utilizada a tecnologia *Socket* para a comunicação entre o Gerente de Falha e o Inspetor de Falha. Os resultados obtidos estão apresentados na tabela 5.

As operações envolvidas no presente teste são as mesmas do teste 2 no cenário AGAD excetuando o envio do Especialista à plataforma 1 ao invés do reconhecimento. Durante os testes, não havia outro tráfego na rede competindo com o tráfego gerado pelo teste. Além disso, nenhum outro processo senão aqueles concernentes ao teste estava rodando durante as medidas. A figura 18 apresenta o ambiente envolvido nas duas plataformas. Na plataforma 1 encontra-se o Gerente de Falha e na plataforma 2 encontra-se o Inspetor de Falha.

Observa-se no gráfico da figura 19 todos os passos que ocorreram entre os componentes das plataformas 1 e 2. Os processamentos incluídos nesse intervalo de tempo podem ser organizados nas seguintes etapas: (i) Inspetor: preparo e envio do Alarme de queda de uma interface (variável *ifOperStatus =Down*), (ii) µServer do Inspetor: envio da mensagem, via Socket, para o Gerente de Falha, (iii) segmento de rede: transporte de mensagem, (iv) µServer do Gerente de Falha : recepção da mensagem via Socket e entrega da mesma ao componente do Gerente de Falha, (v) Gerente de Falha: recepção, interpretação da mensagem e escolha, mediante consulta à BCN, do Especialista mais apropriado, (vii) Gerente de Falha: recuperação da BC, preparo e envio do código do Especialista ao elemento gerenciado adequado, (viii) µServer do Gerente de Falha: envio do código ao elemento gerenciado, (ix) segmento de rede: transporte do Especialista, (x) µServer do elemento gerenciado: recepção e carregamento (instanciação) do código do Especialista que foi recebido, (xi) Inicio da execução do Especialista de Falha de Interface.

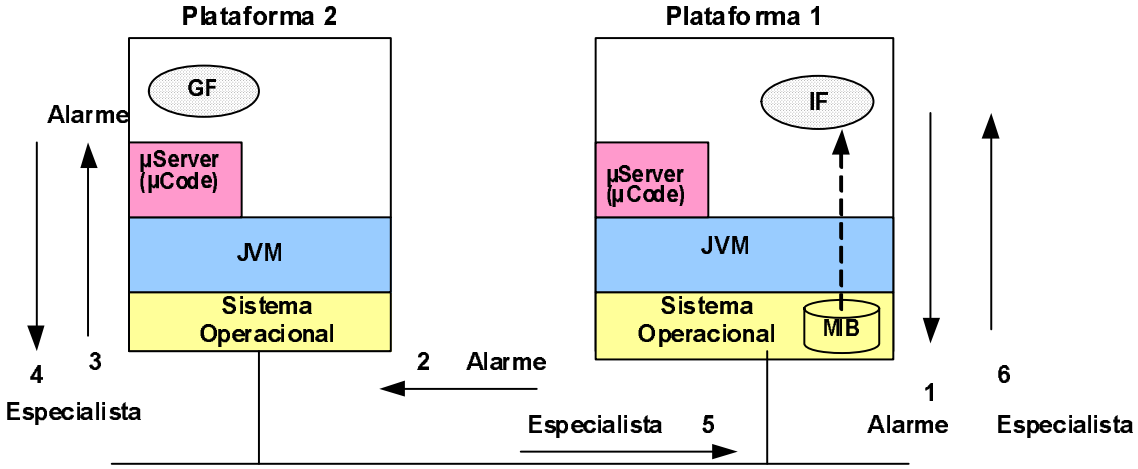


Figura 18 – Ambientes envolvidos nas plataformas 1 e 2.

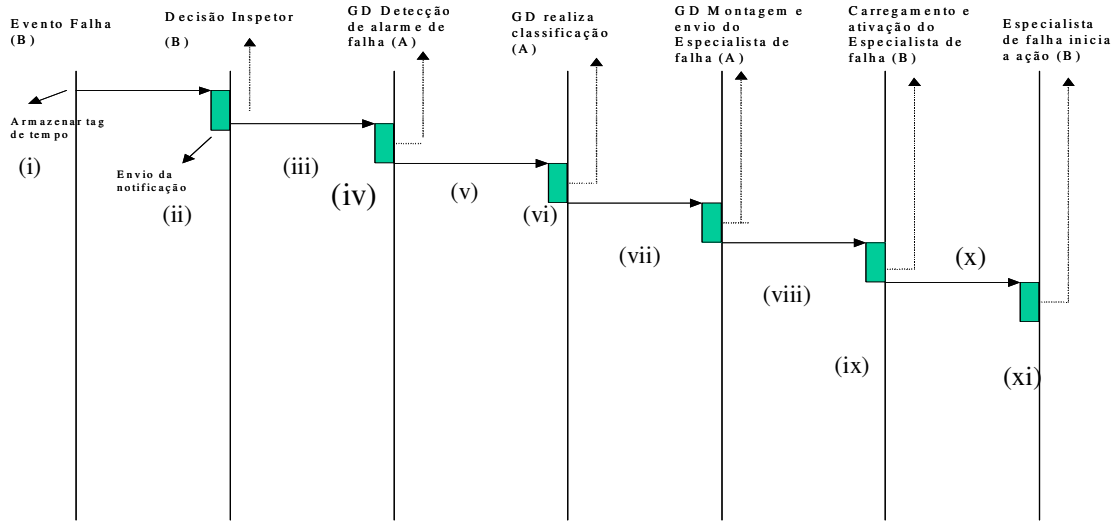


Figura 19 - Passos que ocorreram entre os componentes das plataformas 1 e 2.

Única Rodada					
Medidas	**Tempos**	**Medidas**	**Tempos**	**Medias**	**Tempos**
1ª.	34	**11ª.**	34	**21º.**	44
2ª.	33	**12º.**	35	**22º.**	43
3ª.	34	**13º.**	44	**23º.**	44
4ª.	34	**14º.**	34	**24º.**	45
5ª.	45	**15º.**	44	**25º.**	44
6ª.	44	**16º.**	49	**26º.**	44
7ª.	43	**17º.**	45	**27º.**	34
8ª.	44	**18ª.**	35	**28º.**	43
9ª.	45	**19ª.**	43	**29º.**	44
10ª.	44	**20º.**	44	**30º.**	45
Média			**44**		

Tabela 5 - Medições do 3º teste (em ms)

Os resultados obtidos neste teste indicam que o tempo referente ao envio de uma mensagem de alerta ao Gerente de Domínio de Falha, do processamento desse Gerente para escolha, preparo e envio de um Especialista, bem como o seu carregamento via *Socket* e µCode não requer nem um segundo. Este tempo se comparado ao tempo do operador visualizar um alarme na console e tentar resolver o problema é desprezível e,

pode futuramente, viabilizar a implementação de um gerenciamento pró-ativo tendo como base a arquitetura AGAD.

5.4. Considerações finais do capítulo

Nesse capítulo foram descritos os resultados dos testes realizados com o protótipo desenvolvido conforme a arquitetura descrita nos capítulos anteriores.

Foram aplicados três testes com o objetivo de medir os tempos de atraso nas diversas fases de operação do Sistema AGAD e determinar o seu impacto assim como a sua viabilidade em aplicações de gerenciamento de rede.

Os resultados obtidos no primeiro teste mostram que o mecanismo de distribuição automática dos componentes da arquitetura é muito mais rápida, segura e escalável se comparado a uma instalação desse sistema realizada pelo administrador da rede. Os tempos também permitem concluir que é possível realizar, automaticamente, configuração de componentes de software em tempo hábil sem causar transtornos aos serviços de gerenciamento prestados.

Através dos resultados obtidos no segundo teste constatou-se que a latência de detecção de um alarme via *trap* SNMP é menor do que a observada na arquitetura proposta devido a carga computacional adicional introduzida pela Máquina Virtual Java, pela infraestrutura de mobilidade de código (μCode) e pela AGAD. Entretanto, este atraso adicional não inviabiliza o emprego da arquitetura AGAD.

Os resultados obtidos no terceiro teste permitem concluir que, mesmo com a presença do ambiente μCode e da Máquina Virtual Java, é viável a utilização de tecnologia ativa para a implementação de gerenciamento de falha e desempenho, por exemplo, pois o tempo necessário para o Especialista estar em condições de desencadear uma ação é menor do que um segundo,.

De maneira geral, os testes foram satisfatórios ao determinar a viabilidade do sistema proposto.

Capítulo 6 - Conclusões e Trabalhos Futuros

A demanda por níveis diferenciados de QoS, bem como a constante necessidade de evolução nas redes de comunicação torna necessária a existência de um sistema de gerenciamento flexível e ativo. Visando atender a esses requisitos, este trabalho apresenta a arquitetura de um sistema de gerenciamento distribuído, baseado em tecnologia ativa, para o gerenciamento de uma rede dividida em vários domínios, sendo que a nível da implementação o enfoque foi dado apenas para um único domínio, e em relação as área de gerenciamento foi dado ênfase as áreas de falha e desempenho. Graças ao uso de mobilidade, é possível realizar a configuração, atualização e alteração das várias áreas de gerenciamento tanto nas estações quanto nos nós intermediários (ex: roteadores), além da distribuição dos elementos, dinamicamente.

O esquema de gerenciamento da AGAD trouxe as seguintes vantagens em relação ao gerenciamento centralizado do SNMP: uma redução significativa no tráfego de informações de gerência na rede em relação ao esquema de *polling* do SNMP, redução dos retardos associados às ações de gerenciamento e flexibilidade na atualização ou na introdução, de forma dinâmica, de novas funcionalidades de gerenciamento, através da utilização de componentes de software.

Neste capítulo são feitas algumas considerações sobre os resultados obtidos e são sugeridos trabalhos futuros que venham a complementar e melhorar o sistema proposto. A seção 6.1 descreve algumas considerações finais sobre os resultados obtidos e a seção 6.2 descreve sugestões de trabalhos futuros.

6.1. Considerações Finais dos resultados obtidos

Os resultados obtidos no primeiro teste mostram que o mecanismo de distribuição automática dos componentes da arquitetura é muito mais rápida e segura se comparado a uma instalação desse sistema realizada pelo administrador da rede. Este mecanismo de distribuição automática permite uma maior escalabilidade para a arquitetura. Além disso, os tempos também permitem concluir que é possível realizar, automaticamente, configuração de componentes de software em tempo hábil sem causar transtornos aos serviços de gerenciamento prestados.

O segundo teste apresentou a diferença de tempos entre o cenário AGAD e o cenário SNMP, mostrando que os valores adicionais apresentados no cenário AGAD não inviabilizam o emprego da Arquitetura na gerência de redes porque os tempos de reação exigidos em tal ambiente são ainda compatíveis aos tempos de atrasos medidos. Além disso, é esperado que, com o esquema distribuído de gerenciamento juntamente com o uso da tecnologia ativa, o tráfego de gerenciamento e a carga computacional exigida para o processamento desse tráfego sejam minimizados em decorrência da filtragem e da consolidação de informações que ocorrem de forma distribuída na rede.

Os resultados obtidos no terceiro teste indicam que o tempo referente ao envio de uma mensagem de alerta ao Gerente de Domínio de Falha, do processamento desse Gerente para escolha, preparo e envio de um Especialista, bem como o seu carregamento via μCode não requer nem um segundo, tempo que pode viabilizar, por exemplo, a implementação de um gerenciamento de desempenho pró-ativo tendo como base uma arquitetura de gerenciamento distribuída e que usa tecnologia ativa, a AGAD. Pode-se concluir também que se este tempo for comparado ao tempo total desde o operador de rede visualizar um alarme na console até ele tentar solucionar o problema é desprezível.

6.2. Trabalhos Futuros

Um das principais sugestões de trabalho futuro é a comparação da arquitetura AGAD com o gerenciamento tradicional SNMP, semelhante aos experimentos realizados por Rubinstein [2] utilizando o NS (*Network Simulator*), além do desenvolvimento de um sistema de gerenciamento pró-ativo, que utilizasse os componentes da AGAD como infra-estrutura.

Uma segunda sugestão de trabalho futuro é a Três outras sugestões seriam: o desenvolvimento de novos elementos Inspetores e Especialistas; a implementação do algoritmo de descoberta de topologia lógica, citada no capítulo 3; inserção de mecanismos de segurança e de proteção aos nós ativos e demais elementos que possam vir a ser gerenciados, e o uso de inteligência artificial (IA) em ferramentas automatizadas de análise dos dados coletados e nos elementos Gerente de Domínio

específico, para que esses possam otimizar a classificação do Especialista mais apropriado para realizar uma determinada tarefa, também citada no capítulo 3.

Por fim, como última sugestão, objetivando tornar o sistema mais robusto é a implementação da federação dos Gerentes de Domínio e dos Gerentes-Mores. Esta federação permitirá a cooperação entre domínios de gerência, e a possibilidade de substituição, por exemplo, de um Gerente de Domínio (código e bases de dados) responsável por uma região da rede que, por acaso, fique isolado do Gerente de Domínio original.

Apêndice A

A seguir encontra-se um exemplo genérico que mostra a sequência de passos que tiveram que ser seguidos no protótipo AGAD para a implementação da atualização dinâmica de componentes de software dos elementos Inspetor e Gerente de Domínio.

1) Uma interface com o método a ser atualizado teve que ser definida:

```
public interface MinhaInterface {
        // método a ter seu código atualizado
        public void  Tarefa();
}
```

2) A classe que trás o método a ser atualizado tem que implementar a interface e definir o método:

```
class MinhaClasse implements MinhaInterface {
  // método a ser atualizado
        public void Tarefa () {
  }
```

3) Uma classe mestre tem que definir um método proxy que é chamado pelas demais classes quando querem executar o método atualizável. Essa classe deve ter o método que atualiza o código do método atualizável.

```
class Mestre {
  // Declaração da classe que receberá a classe que atualiza o código
  public static Class NovaClasse;
  // declaração da interface para a o código atualizável
  static MinhaInterface ParteMutavel;
  // método proxy, que é de fato chamado pelo restante do código
        public static void Tarefa () {
        // Chamada ao código atualizável
     ParteMutavel.Tarefa();
  }
```

```
public static void AtualizaParteMutavel( String S ) {
    NovaClasse = null;
    // obtém a classe a partir do seu nome
    try { NovaClasse = Class.forName( S ); }
    catch ( java.lang.ClassNotFoundException e ) {...}
    // instancia a nova classe
    try {
        ParteMutavel = (MinhaInterface)NovaClasse.newInstance();
    }
    catch ( java.lang.InstantiationException e ) {...}
    catch ( java.lang.IllegalAccessException e ) {...}
    catch ( java.lang.ExceptionInInitializerError e ) {...}
    catch ( java.lang.NullPointerException e ) {...}
  }
}
```

4) Quem utiliza o método atualizável tem que executá-lo a partir da classe mestre:

```
Mestre.Tarefa();
```

5) A classe que vai atualizar um método tem que executar o método de atualização passando o nome da classe que tem o novo método:

Mestre.AtualizaParteMutavel("MinhaClasse");

Referências Bibliográficas

[1] Y. Yemini, "The OSI Network Management Model", IEEE Communications magazine,vol.31, pp.20-29, May 1993.

[2] Rubinstein, M. G., Duarte, O. C. M. e Pujolle, G., *Evaluating the Network Performance Management Based on Mobile Agents.*

[3] M. Kahani and H. W. P. Beadle, "Descentralized approaches for Network Management", Computer Communications Review,vol.27,no.3, pp. 36-47, July 1997.

[4] G.Goldszmidt and Y. Yemini, "Delegated Agents for Network Management", IEEE Communications Magazine, vol.36, no.3, pp.66-70, Mar.1998.

[5] Siever E., Spainbour Stephen, "Perl (Guia Completo)" – O'Reilly.

[6] IETF, "Host Resource MIB", ftp://ftp.isi.edu/in-notes/rfc2790.txt, RFC 2790, Mar. 2000.

[7] Goldszmidt, G. e Yemini, Y., *Distributed Management by Delegation*, 15a conferência internacional sobre sistemas de computação distribuídos, 1995.

[8] UCD-SNMP, http://net-snmp.sourceforge.net/tutorial

[9] A.Fuggeta, G.P. Picco, G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering", Vol. 24, 1998.

[10] C-W.Tsai and R.-S.Chan. SNMP through WWW. International Journal of Network Management, 8(2);104-119, march-April 1998.

[11] HorstmannS. Cay, "Core Java 2 - Volume II", vol.2, pp.223-273, 2000.

[12] "AdventNet SNMP - Java API"- www.Adventnet.com

[13] Meunier Philippe, "Phosphorous: Shared Variables on Top of PVM" http://www.-inf.enst.fr/~demeure/phosphorus/phosphorus.html

[14] D. Raz and Y. Shavitt, "An Active Network Approach to Efficient Network Management," Tech. Rep. 99-25, DIMACS,May 1999, http:/dimarcs.Rutgers.edu/TechnicalReports/abstracts/1999/99-25.html.

[15] K. Psounis, "Active Networks: Applications, Security, Savety, and Architectures," IEEE Commum. Surveys, vol.2, no.1, 1st qtr. 1999, http:/www.comsoc.org/pubs/surveys/1q99issue/psounis.html

[16] M. Zapf *et al*., "Descentralized SNMP Management with Mobile Agents," 6th *IFIP/IEEE Int'l. Symp. Integrated Network Mgmt*., Boston, MA, May 1999.

[17] S. Bhattacharjee, K. Calvert, and E. W. Zegura, "An Architecture for Active Networking," HPN '97, Apr. 1997.

[18] W. Stallings, "SNMP and SNMPv2: The Infrastructure for Network Management", IEEE Communications Magazine, vol.36, no.3, pp. 37-43, Mar.1998

[19] William Stallings, "SNMP, SNMPv2,SNMPv3, and RMON 1 and 2", Addison-Wesley Longman,Inc.,1999 , Third Edition.

[20] Wetherall, D., Legedza, U., and Guttag, J. Introducing new internet services: why and how. IEEE Network 12,3 (Junho de 1998),12-19.

[21] Wetherall, D.J.,Guttag, J. V., and Tennenhouse, D.L. ANTS: Network services without the red tape. IEEE Computer 32,4 (Abril de 1999), 42-48

[22] Smith, J., ET AL. Activating Networks: A progress report. IEEE Computer 32,4 (Abril de 1999), 32-41

[23] Hartman, J.J., et al. Joust: A Plataform for Liquid Software. IEEE. Computer 92,4 (Abril de 199), 50-56.

[24] Vijay P. Kumar. Router Architecture for Differentiated Services of Tomorrow's Internet. IEEE Communication Magazine, 152-164.

[25] IETF, Remote Network Monitoring Management Information Base, http://www.ietf.org/rfc/rfc1757.txt?number=1757, RFC 1757, Feb. 1995.

[26] Schwartz Beverly, Jackson W., Smart Packets: Applying Active Networks to Network Management, 2000.

[27] Bieszczad, A., Pagurek, B. E White, T., *Mobile Agents for Network Management*, IEEE Communication Surveys, Fourth Quarter, 1998.

[28] OSHIMA, M.; KARJOTH, G. Aglets Specification. Disponível na INTERNET via www.url: www.trl.ibm.co/aglets/spec_alpha.html

[29] David J. Wetherall and David L. Tennenhouse, The ACTIVE-IP option, In the 7th ACM SIGOPS European Workshop. Sept. 1996.

[30] D. Scott Alexander et al .Active Network Encapsulation Protocol (ANEP), http:/www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt, RFC Draft, July 1997.

[31] Keshav, S. e Sharma, R., *Achieving Quality of Service through Network Performance Management*, .

[32] V.A. Phan, A.Karmouch: "Mobile Software Agents:an Overview", IEEE Communication, Magazine, July 1998.

[33] Hu, C., Chen, W., *A Mobile Agent-based Active network Architecture*, International Conference on Parallel and Distributed Systemas, ICPADS´00, IEEE, 2000.

[34] Picco, G.P., µCode: A Lightweith and Flexible Mobile Code Toolkit , Proceedings of the 2nd International Workshop on Mobile Agents 98 (MA'98),

[35] Bradshaw, J. et al, *Software Agents*, AAAI Press / The MIT Press, Menlo Park, California, 1997.

[36] Raz, D. e Shavitt, Y., *Active Networks for Efficient Distributed Network Management*, IEEE Communications Magazine, Março de 2000.

[37] Bush, S. F., Active *Virtual Network Management Protocol*,

[38] Schwartz, B., Jackson, A. W., Strayer, T., Zhou, W., Rockwell, D. e Partridge, C., Smart Packets: *Applying Active Networks to Network Management*, ACM Transactions on Computer Systems, Fevereiro de 2000, páginas 67-68.

[39] Kawamura, R. e Stadler, R., *Active Distributed Management for IP Networks*, IEEE Communications Magazine, Abril de 2000.

[40] Bohoris, C., Pavlou, G., Cruickshank, H., *Using Mobile Agentes fos Network Performance Management*, IEEE/IFIP Network Operations and Management Symposium (NOMS´00), Honolulu, Hawaii, 2000.

[41] Sabata, B, Chatterjee, S., Davis, M. Sydir, J. J. e Lawrence, T. F., *Taxonomy for QoS Specifications*, WORDS´97, Newport Beach, California, 1997.

[42] Ferreira, A. B. H., *Dicionário Básico da Língua Portuguesa*, Folha de São Paulo, 1995.

[43] Yuri Breitbart, Garofalakis Minos, Topology Discovery in Heterogeneous IP Networks, Information Sciences Research center – Bell Labs, Lucent Technologies.

[44] Goldszmidt, G. e Yemini, Y., *Distributed Management by Delegation*, 15a conferência internacional sobre sistemas de computação distribuídos, 1995.

[45] Wetherall, D.J.,Guttag, J. V., and Tennenhouse, D.L. ANTS: A toolkit for building and dinamically deploying network protocols. In IEEE OPENARCH'98 (São Francisco, CA, Abril de 1998).

[46] H. M. Deitel, "Java 2 Como Programar".

[47] Object Management Group (OMG), *Unified Modeling Language Specification (draft), v 1.4*, fevereiro de 2001.

[48] Wheterall, D. J., *Active Network Vision and Reality: Lessons from a Capsule-based system*, 17º Simpósio em Princípios de Sistemas Operacionais da ACM, SOSP´99, dezembro de 1999.

[49] Dan Decasper and Bernhard Plattner, DAN: Distributed Code Cashing for Active Networks, Proc. IEEE INFOCOM '98, San Francisco, CA, 29 March-2 April 1998.

[50] Rose, M., K., IETF RFC1213 "Management Information Base for Network Management of TCP/IP-base Insternets", 1991.

[51] Tennenhouse, D. L. et al, *A Survey of Active Network Research*, IEEE Communications Magazine, janeiro de 1997.

[52] Sammoud khaled, "A Management Request Broker based on Transactional Mobile Agents for in-tina servives", Global Telecommunications Conference – Globecom '99.

[53] A.Sahai and C.Morin, "Towards distributed and dynamic network management", in Proceeding of the IEEE/IFIP Network Operations and Management Symposium (NOMS98), New Orleans, USA, Feb. 1998.

[56] Oliveira, M. Introducão à Gerência de Redes ATM, 16º Simpósio Brasileiro de Redes de Computadores, maio de 1998.

[57] Site de pesquisa – http://ceiteseer.nj.nec.com/cs

[58] IETF, "Management Information Base for Network Management of TCP/IP based internet: MIB II", http://www.ietf.org/rfc/rfc1213.txt?number=1213, RFC 1213.

[59] IETF, “A Simple Network Management Protocol (SNMP), http://www.ietf.org/rfc/rfc1157.txt?number=1157, RFC 1157.

[60] Horstmann, Cay, “Core Java – Volume II – Recursos Avançados”, Dezembro de 1999.

[61] Rumbauch, James e Jacobson, Ivar: “UML Guia do Usuário”, 2000.

[62] **Dumont, Ana Paula, Edmundo, L., Pirmez, L. Carmo, L.F.R.C**. A Viabilidade do Gerenciamento de Desempenho Pró-Ativo baseado em uma Arquitetura de Gerenciamento que usa Tecnologia Ativa, Maio de 2002.