PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA

Proposta de um Framework para Gerência de Clusters

Dissertação submetida à Pontifícia Universidade Católica do Paraná
como requisito parcial à obtenção do grau de

Mestre em Informática Aplicada

por

Daniel Francisco Wandarti

Curitiba, janeiro de 2003

Proposta de framework para gerência de clusters.

Daniel Francisco Wandarti

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Informática Aplicada** na especialidade **Sistemas Distribuídos**, e aprovada em sua forma final pelo curso de Pós-Graduação.

Curitiba, janeiro de 2003.

Prof. Dr. Carlos Alberto Maziero, orientador

Prof. Dr. Carlos Alberto Maziero
coordenador do curso de Pós-Graduação em Informática Aplicada
da Pontifícia Universidade Católica Paranaense.

Banca Examinadora

Prof. Dr. Edgar Jamhour

Prof. Dr. Luiz Nacamura Junior

aos meus pais Henrique e Edite

Agradecimentos

Sou muito grato aos meus pais, pelo apoio e por tudo que fizeram por mim.

Também sou muito grato ao meu orientador, Professor Carlos Maziero, pela sua paciência, dedicação e por suas valiosas informações.

Resumo

O objetivo desta dissertação de mestrado é estudar, propor e implementar uma ferramenta para gerência de clusters abertos. Esta dissertação de mestrado faz parte do projeto ParaPUC que está em desenvolvimento desde outubro de 1998 no programa de Pós-Graduação de Informática Aplicada da PUCPR.

O projeto aqui descrito é inspirado do projeto Beowulf da NASA (e outros projetos similares). Beowulf é um conjunto de PCs de mercado (mass-market comodity off the shelf) interconectados através de uma rede local de baixo custo (LAN - Local Area Network) rodando sobre sistema operacional de mercado e aplicações paralelas que utilizem alguma biblioteca de troca de mensagem no padrão PVM/MPI.

Um cluster de PC beowulf é conhecido pela sua facilidade de construção e baixo custo, onde com apenas alguns PCs e uma rede local é possível obter um cluster com poder de processamento parecidos com o de um supercomputador. Devido à esta facilidade de construção surge um problema com relação ao beowulf, que é a sua gerência. Este problema reside no fato de que é necessário gerenciar vários nós utilizando uma ferramenta com baixo impacto, que seja expansível, escalável e interoperável.

A dissertação é composta de uma revisão bibliográfica sobre arquiteturas paralelas e clusters, uma revisão sobre gerência de sistemas e gerência de clusters e a proposta para uma ferramenta de gerência de clusters baseada em CIM, CORBA e Java. Esta proposta inclui uma proposta para extensão do CIM, a classe Cluster.

Abstract

The goal of this Master thesis is to study, to define and to implement a management tool for open clusters. This activity is being carried under the ParaPUC project of the Postgraduate Program in Applied Computer Science of the Pontifícia Universidade Católica do Paraná.

This project is inspired on the NASA's Beowulf project (and other similar projects). A Beowulf cluster is a set of mass-market off the shelf PCs interconnected through a low cost local area network (LAN), running on operational system of market and parallel applications that use some library of message passing libraries like PVM or MPI.

Beowulf clusters are know by their easiness of construction and low cost. With only standards PCs and a local network is possible to get a cluster with power of processing similar to the one of a supercomputer. Although their construction easiness, the management of such clusters is a problem that remains to be solved in an open way. The management infrastructure should be flexible (to easily support changes in the cluster structure), should use light protocols and have a small memory and processing footprint, should be also escalable and interoperable.

This dissertation is composed of a bibliographical revision about parallel architectures and clusters, a revision about system management and cluster management and the proposal of a tool to manage open clusters based in CIM, CORBA and Java. This proposal includes a CIM extension, the class Cluster.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A procura pelo maior poder de processamento tem sido um dos principais esforços no desenvolvimento de computadores. Antes da era do computador, cientistas e engenheiros não conseguiam imaginar a possibilidade de se executar algumas centenas de instruções por segundo. Entretanto, logo que se depararam com esta possibilidade, clamam por ainda mais poder computacional.

Por séculos, a ciência tem seguido o paradigma de primeiro observar, teorizar e, somente então, testar a teoria através de experimentos. Similarmente, engenheiros têm projetado, desenvolvido e testado um protótipo, para, finalmente, desenvolver um produto final. Porém, é mais barato e mais rápido realizar simulações detalhadas por computador e, então, fazer os experimentos ou construir o protótipo. Há, também, os casos em que a simulação não pode ser acompanhada de experimentação, como a evolução do universo.

Desta maneira, a necessidade de maior poder computacional tem gerado ainda mais expectativas. Mas como construir computadores capazes de efetuar bilhões de operações por segundo? A opção mais simples e direta é estender as tecnologias existentes, que são bem conhecidas, ou seja, tentar construir um computador com arquitetura talvez similar à de Von Neumann, com um processador extremamente rápido.

Seria muito difícil, para não dizer impossível, com a tecnologia atual, ter um processador com capacidade de executar bilhões de cálculos por segundo. Então, a solução é dividir para conquistar. Assim, se um processador não consegue efetuar bilhões de operações por segundo, centenas de processadores conseguem. Com isto, surgiram os supercomputadores e, após, os clusters de computadores.

Dentre os clusters de computadores, o mais difundido, atualmente, é o Beowulf, um cluster aberto, que utiliza PCs de mercado, com sistema operacional de mercado e uma biblioteca padrão de troca de mensagens.

No entanto, com o aparecimento desta nova classe, o Beowulf, surgiu, também, um pro-

blema: como administrar e monitorar de forma efetiva estes computadores, conjuntamente com o sistema operacional e a aplicação? Existem algumas ferramentas propostas, mas nenhuma que auxilie na administração do cluster de forma satisfatória e que tenha, também, as características de ser expansível, escalável, de baixo impacto e interoperável.

A base desta dissertação é uma proposta para uma arquitetura de gerência de clusters Beowulf que satisfaça as condições de ser expansível, escalável, de baixo impacto e interoperável. Para satisfazer estas condições, será utilizado padrão CIM (*Common Information Management*) para a representação da informação gerenciada. A comunicação será feita através de CORBA e a implementação com Java.

A dissertação é composta de 6 capítulos, assim distribuídos:

- no **capítulo dois** é feita uma revisão bibliográfica sobre arquiteturas paralelas e clusters;
- no **capítulo três** é apresentada uma revisão sobre a gerência de sistemas, sendo abordados, também, os padrões - SNMP, OSI, CORBA, CIM, WBEM e JMX, além de uma comparação entre os padrões de representação da informação - SNMP, OSI e CIM;
- o **capítulo quatro** trata de gerência de clusters, apresentando alguns trabalhos existentes - SUMO, ClusterProbe, SCMS e PHOENIX;
- o **capítulos cinco** apresenta a proposta da arquitetura, uma proposta para extensão do CIM e alguns resultados obtidos;
- o **capítulo seis** conclui o trabalho.

Capítulo 2

Conclusão e perspectivas

O desenvolvimento da tecnologia de Clusters de PCs Beowulf foi a solução para muitos problemas existentes na época em que havia apenas arquiteturas proprietárias para processamento paralelo. Entre eles, as padronizações na programação, não havendo mais a necessidade de reescrever programas para as novas tecnologias, e a obtenção de conhecimento em processamento paralelo, considerado, agora, um investimento. Outra vantagem também é o preço, pois agora estes "supercomputadores" são muito mais acessíveis.

A gerência de redes também já teve muitos problemas, até o surgimento de padrões. O primeiro destes padrões foi o SNMP, hoje, o mais difundido. Porém, este ainda possui diversas deficiências, estimulando, assim, o surgimento de novos padrões. Dentre estes, o mais interessante é o CIM, pois objetiva modelar a informação, seja heterogênea ou não, e esta modelagem segue uma orientação a objetos.

Com a tecnologia de clusters abertos ainda há um vazio - a gerência de forma satisfatória.

A criação de uma ferramenta para gerência de clusters que utilizem um padrão para a modelagem da informação e CORBA para comunicação entre gerente e agente foi uma união muito feliz. Com a utilização de CORBA, foi possível o desenvolvimento de uma ferramenta que satisfizesse os requisitos necessários para uma ferramenta de administração de clusters abertos.

A contribuição efetiva deste trabalho está na criação de uma ferramenta para gerência de clusters abertos, visto que grande parte das ferramentas atuais limita-se apenas à monitoração. Outra característica, também ausente em grande parte das ferramentas atuais, é a expansibilidade, sendo possível, ao usuário, acrescentar alguma(s) classe(s) para a gerência de sua própria aplicação.

Alguns tópicos podem ser considerados na continuidade deste objetivo:

- **Módulo de agendamento**: um módulo que saiba a carga de cpu de todos os nós, memória livre e o grau de completude das tarefas e, através de um módulo desenvolvido

para conversar com a aplicação em um determinado protocolo, (exemplo: PVM/MPI [Ge94]) agendar em quais nós devem ser executadas determinadas tarefas [LSW99] [Fer99]. Com tal módulo fica mais fácil detectar um eventual *deadlock* entre processos e também é possível determinar o grau de utilização dos nós [Fer99] [Ge97];

- **Otimização dos OPs**: otimizar os OPs, utilizando linguagem e/ou ORB mais leves para que possam rodar em micros menos potentes, tornando mais viável sua utilização em um *grid* computacional [FdRdR02];
- **Ponte SNMP**: fazer uma ponte SNMP, sendo possível obter dados disponíveis somente através deste protocolo [GXF95] [AMR99];
- **PCIM**: acrescentar à ferramenta as classes relativas ao PCIM [IET02];
- **ORB com multicast**: adaptar a ferramenta para utilizar um ORB que suporte multicast para ser usado no momento que for obter/armazenar valores de propriedades ou invocar métodos de um contexto de nós do cluster [BdSFL02] [UPAM00];
- **Segurança**: desenvolver um módulo de segurança para a ferramenta.

Capítulo 3

Gerência de sistemas

À medida que redes e sistemas crescem, crescem, também, a complexidade, a importância, a heterogeneidade e o custo de gerenciamento. Para minimizar tais fatores surgiu o SNMP (*Simple Network Management Protocol*), padrão criado para ser utilizado em uma variedade de equipamentos e sistemas, que possam ser utilizados de forma independente de fornecedor [Sta93].

Os principais requerimentos para investir em gerência de sistemas são: controlar recursos estratégicos, controle da complexidade, melhora do serviço, balanceamento de necessidades, redução do *downtime* e controle de custos.

Gerência de sistemas é a prática de *(a)* monitorar e controlar uma rede existente de modo que os computadores estejam rodando e que satisfaçam as expectativas dos usuários, *(b)* planejar extensões e modificações da rede para satisfazer a demanda, *(c)* incorporar novos elementos à rede, facilmente, sem interferir nas operações existentes [Lew95].

Uma grande rede não pode ser gerenciada somente por esforços humanos, é necessária a utilização de ferramentas para seu auxílio. À medida que o tempo passa, estas ferramentas têm ficado cada vez mais complexas e também se evidencia o aumento do número de revendedores de equipamentos. Conclui-se, daí, que uma rede que não pode ser gerenciada corretamente possui pouco valor.

3.1 Conceitos básicos

A seguir, alguns conceitos básicos em gerência de sistemas, como o ciclo da gerência, modelo gerente/agente e informações de gerência.

3.1.1 Ciclo da gerência

O ciclo da gerência de sistema é um ciclo composto de:

- **Coleta de dados**: monitoração dos recursos gerenciados;
- **Diagnóstico**: tratamento e análise dos dados coletados para delinear o problema;
- **Ação**: controle sobre os recursos gerenciados para corrigir o problema.

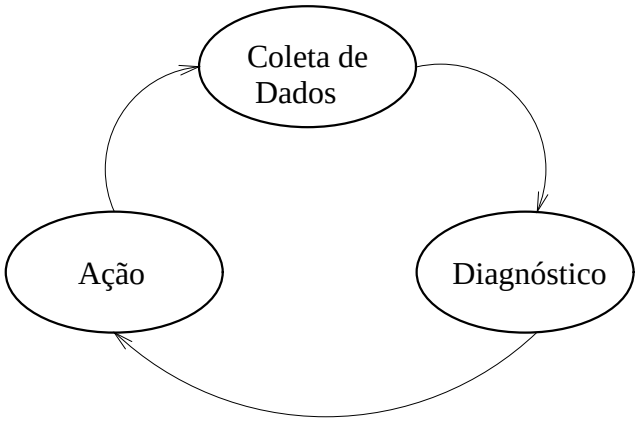

Figura 3.1: Ciclo da gerência

3.1.2 Informações de gerência

As informações para monitoramento podem ser classificadas em:

- **Estática**: caracteriza a configuração corrente e seus elementos. Informações que mudam com pouca freqüência, por exemplo, número de portas de um *router*;
- **Dinâmica**: relacionada aos eventos da rede, por exemplo, número de pacotes enviados;
- **Estatística**: informações que podem derivar das informações dinâmicas, por exemplo, número de pacotes perdidos nos últimos 2 minutos.

Para que a gerência de uma rede ou sistema seja efetiva, é necessário monitorar seu desempenho. Uma das dificuldades mais encontradas na gerência de redes ou sistemas é a seleção e uso apropriados dos indicadores [Sta93].

Algumas dificuldades:

- Muitos indicadores em uso;
- O significado dos indicadores não bem compreendidos;

- Alguns indicadores introduzidos e suportados por somente alguns fabricantes;
- Grande parte dos indicadores não passíveis de comparação entre eles;
- Os indicadores medidos corretamente mas interpretados erroneamente;
- Em muitos casos, o cálculo dos indicadores leva muito tempo, e o resultado final dificilmente pode ser utilizado para controlar o ambiente.

Os indicadores se encaixam nas seguintes categorias: métricas orientadas ao serviço e métricas orientadas à eficiência. Para julgar se uma rede supre as necessidades é preciso verificar se os indicadores de um determinado serviço satisfazem o usuário. Por isto, um indicador orientado a serviço é mais importante que um indicador orientado à eficiência:

- **Disponibilidade**: probabilidade de o sistema estar operacional no instante de tempo *t*;
- **Tempo de resposta**: tempo gasto pelo sistema para reagir a uma determinada entrada de dados;
- **Exatidão**: transmissão exata entre usuário e *host* e vice-versa, logicamente essencial para qualquer rede;
- **Throughput**: é uma métrica orientada à aplicação, como exemplo: pode incluir número de um certo tipo de transações num determinado intervalo de tempo ou número de sessões de usuários para uma aplicação durante certo tempo;
- **Utilização**: métrica mais detalhada que *throughput*, refere-se em determinar o percentual de tempo que um recurso esteve em uso por um determinado tempo.

3.1.3 Modelo básico de gerência

Componentes de um sistema de gerência de rede (figura 3.2):

- **Aplicação de gerência**: gerencia a rede visível para o usuário, como monitoração de desempenho, monitoramento de falhas e monitoramento de contabilização;
- **Função de gerente**: desempenha a função de monitoração básica, retornando informações retornadas por outros elementos da configuração;
- **Função do agente**: recolhe e armazena as informações de gerência para um ou mais elementos e comunica as informações para o monitor;

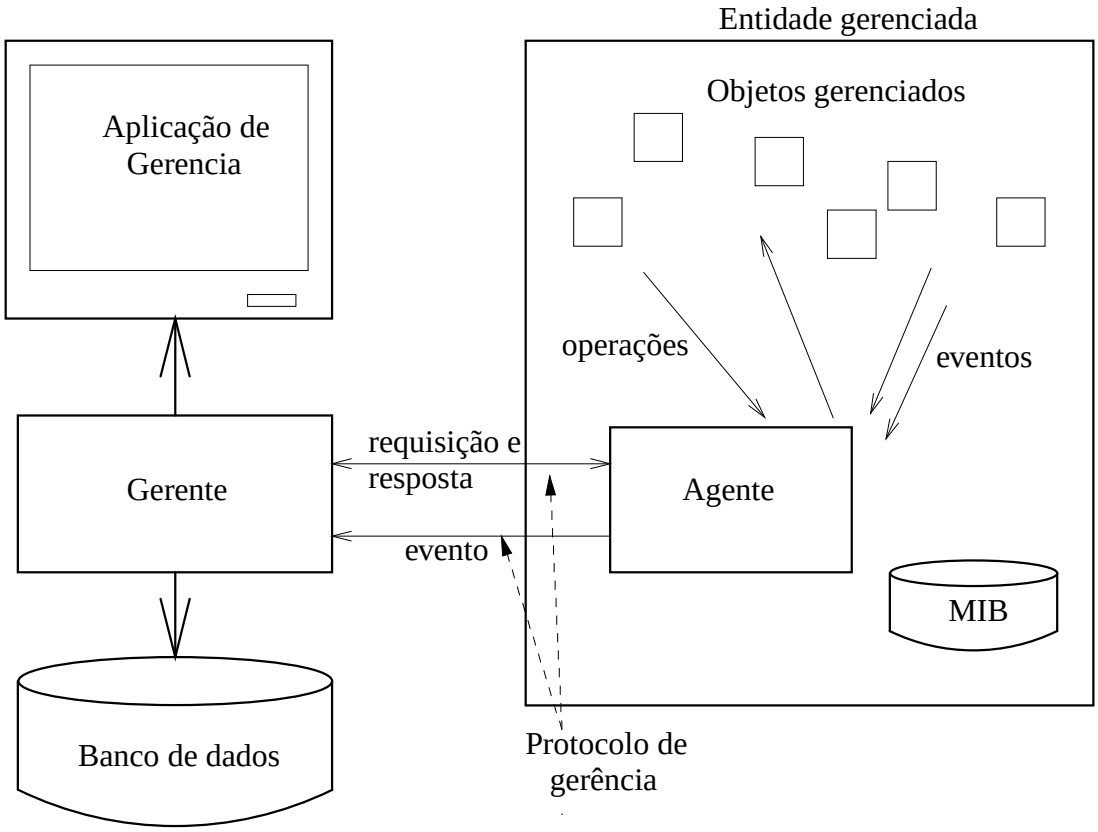

Figura 3.2: Modelo agente/gerente

- **Objetos gerenciados**: informação gerenciada que representa os recursos e suas atividades.

Vale ressaltar um módulo funcional adicional, relacionado com informações estatísticas:

- **Agente de monitoração**: gera análises estatísticas e sumarizadas de informações gerenciais. Se o gerente for remoto, este módulo age como um agente e comunica as informações sumarizadas ao gerente.

Os módulos funcionais podem ser configurados de diversas maneiras. A estação que roda a aplicação de monitoração também é sujeita a ser monitorada. Assim, o monitor geralmente inclui software agente e um conjunto de objetos gerenciados. É vital monitorar o status e comportamento do monitor da rede para garantir a continuidade do desempenho de suas funções e o acesso à própria carga, além da rede.

Aqui não são usados os termos cliente e servidor já que estes podem ter muitos sentidos em um sistema distribuído.

3.1.4 Polling e Event Reporting

A informação utilizada por monitores de redes é coletada e armazenada por agentes e disponibilizada para um ou mais gerentes do sistema. Duas técnicas são usadas para tornar o agente da informação disponível ao gerente:

- **Pooling**: seqüência de interações requisição-resposta entre gerente e agente. O gerente pode indagar qualquer agente (desde que tenha autorização) e requisitar os valores de vários elementos da informação; o agente responde com a informação de sua MIB.

- **event reporting**: a iniciativa é do agente e a função do gerente é ouvir, esperando a chegada de informações. Um agente pode gerar um relatório, periodicamente, informando seu status atual.

Ambos são muito práticos e uma monitoração típica irá utilizar os dois métodos, variando, apenas, a ênfase utilizada.

3.2 Áreas de gerência de sistemas

As cinco grandes áreas de gerência de sistemas são: gerência de falhas, gerência de contabilização, configuração, gerência de desempenho e segurança [Sta93].

3.2.1 Gerência de falhas

Para manter o funcionamento do sistema de forma correta, o gerente precisa cuidar do sistema como um todo e de cada componente essencial, individualmente. Quando uma falha ocorre, o gerente deve: 1. determinar o local da falha; 2. isolar o sistema do ponto de falha; 3. reconfigurar ou modificar o sistema de forma a minimizar o erro; 4. consertar ou trocar os componentes falhos.

Deve-se atentar para a diferença entre falhas e erros. Uma falha é uma condição anormal que requer atenção (ou ação) do gerente para corrigir - talvez até a troca de um equipamento físico - e erro é um simples evento - como uma violação de CRC na transmissão de um pacote.

Problemas no monitoramento de falhas

Em um ambiente complexo, a localização e diagnóstico de uma falha podem ser difíceis. Algumas dificuldades na localização da falha:

- **Falhas não observáveis**: certas falhas não podem ser observadas localmente. Por exemplo, um *deadlock* entre processos distribuídos;

- **Falhas parcialmente observáveis**: um nó falho pode ser observado, mas a observação pode ser insuficiente para indicar o erro. O nó pode não estar respondendo devido à falha de algum dispositivo de baixo nível;

- **Falhas observáveis incertas**: mesmo com observações detalhadas, existem algumas observações que se tornam incertas ou mesmo inconsistentes. O tempo de resposta muito grande de um dispositivo pode ser causado pela falha de *timer* local.

3.2.2 Gerência de contabilização

Monitoramento de contabilização consiste em manter um registro da utilização dos recursos da rede ou sistema por um usuário ou grupo de usuários. O gerente precisa ser capaz de rastrear o uso dos seus recursos, por um usuário final ou classe de usuários finais, por razões, incluindo:

- Um usuário final ou grupo de usuários finais podem abusar dos seus privilégios de acesso e sobrecarregar a rede ao custo de outros usuários finais;

- Usuários finais podem estar tendo uma utilização ineficiente e o gerente pode ajudar ao trocar procedimentos para melhorar o desempenho;

- O gerente da rede está em uma posição melhor para planejar o crescimento da rede caso conheça com detalhes suficientes a atividade dos usuários finais.

3.2.3 Configuração

Gerência de configuração consiste em inicializar a rede ou sistema e desligar parte ou todos seus componentes sem afetar os outros componentes ativos. Também consiste em manter, adicionar ou atualizar a relação entre componentes e o status. Uma reconfiguração é geralmente necessária quando em resposta ao desempenho ou devido a uma atualização, recuperação de falha ou checagem de segurança.

3.2.4 Definindo a informações de configuração

Informações de configuração descrevem a natureza e status dos recursos que são de interesse do gerente do sistema ou rede. A informação de configuração inclui uma especificação

do recurso gerenciado e os atributos destes recursos. Os recursos podem ser físicos (roteadores, facilitadores de comunicação, HDs) ou lógicos (temporizadores, contadores, espaço físico). Atributos podem ser, por exemplo, nome, endereço, número de identificação. Estas informações de configuração podem ser estruturadas de diversas maneiras:

- Uma lista estruturada, simples, de campos de dados, com cada campo contendo um valor único. Esta é a abordagem utilizada pelo padrão SNMP;
- Um banco de dados orientado a objetos. Cada elemento de interesse é representado por um ou mais objetos. Cada objeto contém atributos cujos valores refletem as características do elemento representado. Um objeto pode, também, conter comportamentos, como notificações a serem geradas, caso ocorram certas relações de eventos desse elemento. Esta é a abordagem utilizada pelo padrão OSI;
- Um banco de dados relacional. Campos individuais, no banco, contêm valores que refletem características da rede de elementos. A estrutura do banco reflete a relação entre elementos da rede.

Apesar dessas informações serem acessíveis à estação do gerente, são geralmente armazenadas perto do recurso em questão, seja no agente do nó, se o recurso é parte do nó, seja no proxy do nó, se o nó que contém o recurso não suporta um software agente.

A função de controle deveria habilitar o usuário a especificar os limites e tipos de valores, cujos atributos de um recurso específico de algum agente possam ser configurados. Os limites podem ser uma lista de todos os estados de valores ou os valores máximo e mínimo para parâmetros e atributos. O tipo do valor permitido, também deveria ser especificado.

3.2.5 Gerência de desempenho

Redes de comunicação modernas são compostas de muitos componentes heterogêneos, que necessitam se intercomunicar e compartilhar dados e recursos. Muitas vezes, é crítico para uma aplicação que a comunicação sobre a rede tenha um desempenho mínimo garantido.

Esta gerência é composta de duas categorias funcionais: monitoramento e funcional. Função de monitoramento é a função que rastreia atividades e a função de controle habilita a gerência a fazer ajustes para melhorar o desempenho.

3.2.6 Gerência de segurança

Consiste em gerenciar a proteção da informação de gerência e facilidades de controle de acesso. Isto inclui gerar, distribuir e armazenar chaves de criptografia. Logs são uma

ferramenta muito importante e a gerência de segurança está muito envolvida em coletar, armazenar e examinar os registros de auditoria e logs de segurança.

3.3 Padrões de gerência

Existem algumas propostas e padrões de protocolos para gerência de rede:

- OSI (*Open Systems Interconnection*);
- SNMP (*Simple Network Management Protocol*);
- CORBA (*Common Object Request Broken Architecture*);
- CIM (*Common Information Model*);
- WBEM (*Web-Based Enterprise Management*);
- JMX (*Java Management eXtensions*).

OSI, SNMP e CORBA são patrocinados pela ISSO e IETF (Internet Task Force). CIM e WBEM, pela DMTF (*Distributed Management Task Force*). O JMX é considerado um framework de gerência, patrocinado pela Sun e Java Community Process.

Os protocolos de rede são importantes para se construir uma ferramenta de gerência de rede, mas, mais importante que isto é a maneira como se utilizam estas ferramentas e estes protocolos, já que estas ferramentas servem para auxiliar o gerente da rede.

3.3.1 OSI

O modelo de gerência OSI é baseado em objetos e, seus recursos, supervisionados e gerenciados, são chamados de “objetos gerenciados”. Neste modelo o que um objeto gerenciado é exatamente, não importa muito, já que ele pode ser qualquer coisa (um switch, PBX, multiplexador, programas, algoritmos, etc) [Sta93].

No modelo OSI, os objetos gerenciados são classificados pela sua posição na camada OSI. Se for uma camada específica, são chamados objetos gerenciados da camada *N* (tabela 3.1). Os objetos com mais de uma camada são chamados “sistemas de objetos gerenciados”.

O processo agente situa-se entre os objetos gerenciados e o sistema de controle de rede (*Management Functional Domain - MFD*). Múltiplos MFDs podem existir em uma empresa ou pertencer a um país. Este conceito não existe no modelo de gerência SNMP.

Aspectos de um objeto gerenciado:

- as operações efetuadas em um objeto gerenciado devem fazer parte de sua definição;

Camada	*Descrição*
Camada 7	Protocolo do Nível de Aplicação
Camada 6	Protocolo do Nível de Apresentação
Camada 5	Protocolo do Nível de Sessão
Camada 4	Protocolo do Nível de Transporte
Camada 3	Protocolo do Nível de Rede
Camada 2	Protocolo do Nível de Enlace de dados
Camada 1	Protocolo do Nível de Física

Tabela 3.1: Camadas OSI

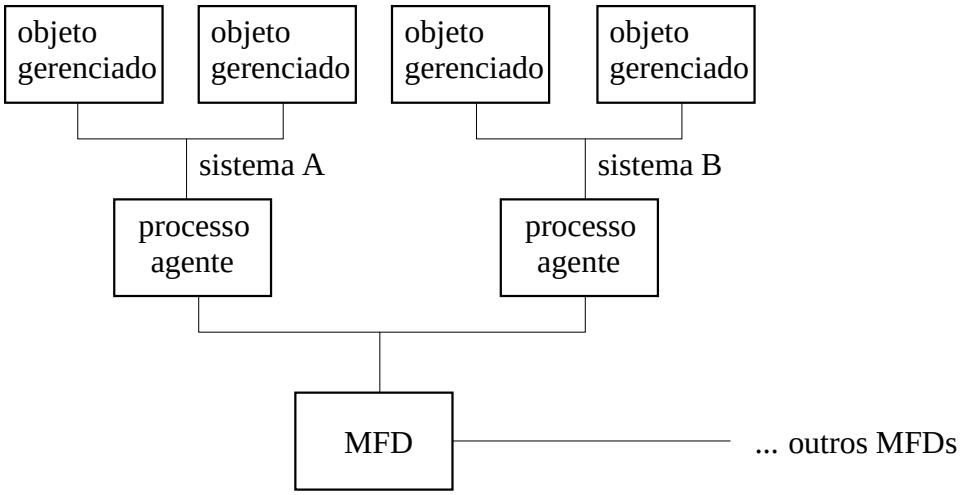

Figura 3.3: MFD

- a definição de um objeto gerenciado pode, também, incluir os efeitos dessa operação no recurso gerenciado;
- o estado/propriedade de um objeto gerenciado pode determinar as suas operações.

Segundo o modelo OSI, um objeto gerenciado é definido por:

- **atributos**: os atributos não podem ser criados, deletados ou renomeados durante a instanciação do objeto. Os atributos podem ser alterados por estímulo interno e externo;
- **operações**: *create* (cria um novo objeto), *delete* (deleta um objeto existente), *action* (executa uma ação/operação no objeto), *get value* (obtém valor de um atributo), *add value* (adiciona um valor para um atributo), *remove value* (remove o valor de um atributo) e *set value* (seta o valor de um atributo);
- **comportamento**: o comportamento significa como o objeto reage a uma operação e certos limites colocados em seu comportamento;

- **notificações**: as notificações dependem do tipo do objeto.

As notificações são enviadas para o objeto e seu comportamento sabe o que fazer com a notificação. Um exemplo pode ser o consumo de CPU onde, a cada intervalo, é gerada uma notificação, mas é o comportamento que define o que fazer com estas notificações. Os atributos que fazem parte de um objeto OSI são:

- **sintaxe**: define o tipo (*data type*);
- **nível de acesso permitido**: leitura, leitura/escrita, escrita e não acessível;
- **status**: obrigatório(1), opcional(2) e obsoleto(3);
- **nome**: não ambíguo.

No modelo OSI existe uma distinção entre objetos e atributos que permite o reuso de nomes entre estes. Os aspectos de um objeto gerenciado são:

- as operações efetuadas em um objeto gerenciado devem fazer parte de sua definição;
- a definição de um objeto gerenciado pode, também, incluir os efeitos dessa operação no recurso gerenciado;
- o estado/propriedade de um objeto gerenciado pode determinar as operações sobre o objeto gerenciado.

Objetos gerenciados

Os objetos gerenciados com características similares são colocados em uma classe de objeto ou *Managed Object Class* (MOC). O processo gerenciador interage com o agente através de:

- dados;
- controle;
- eventos.

“O protocolo OSI é muito completo, mas seu maior problema é sua complexidade, o que o torna difícil de implementar e muito pesado” [Bla95].

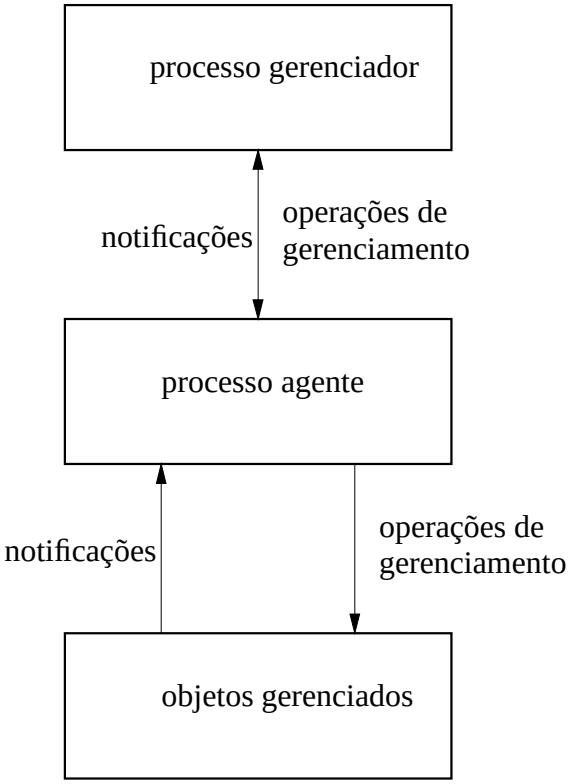

Figura 3.4: Processo gerenciador

3.3.2 SNMP

O protocolo SNMP é o mais difundido na Internet, mas sua maior fraqueza consiste em ser considerado insuficiente para a gerência de redes. A maior vantagem do SNMP é ser um protocolo leve [Bla95].

O SNMP (*Simple Network Management Protocol*) teve sua origem em um protocolo para monitoração de *gateways* IP, o SGMP (*Simple Gateway Management Protocol*), e consiste em uma Base de Informação de Gerenciamento (MIB - *Management Information Base*), onde graças à grande capacidade da MIB de representar os objetos gerenciados, o SNMP tornou-se o protocolo mais utilizado, hoje em dia, para esta tarefa [Sta93]. As principais características de um agente SNMP são:

- sintaxe: define o tipo (*data type*). Ao contrário do OSI, os tipos SNMP são mais limitados, baseados no *Abstract Syntax Notation* (ASN.1) com inteiros, string de octetos, seqüências, seqüências-de e outros tipos;
- nível de acesso permitido: leitura, leitura/escrita, escrita e não acessível;
- status: obrigatório(1), opcional(2) e obsoleto(3);

- nome: não ambíguo

Como não há distinção entre objetos e atributos, não é permitido o reuso de nomes como "*shutting down*" e "*operational*". No SNMP os objetos gerenciados similares são definidos em um grupo. Esta característica difere dos outros padrões - OSI e CIM, que fazem esta distinção.

MIB

A estrutura da informação de gerência (SMI - *Structure of Management Information*) define as regras para descrever a informação. Especificamente, sua função é permitir que a informação de gerência seja descrita da forma mais independente possível dos detalhes de implementação.

Na visão da coleção de objetos gerenciados, como estes residem em um armazenamento virtual, como um banco de dados, o SMI define o esquema para aquele banco de dados. Na realidade, existe um nome mais preciso para este banco de dados, ele é chamado de MIB - *Management Information Base* [Ros96].

As bases do protocolo SNMP são as informações armazenadas sobre os recursos que devem ser gerenciados, onde cada um desses recursos é representado por um objeto e a MIB é uma coleção de estruturas desses objetos. Basicamente, a MIB é um banco de dados com estrutura em forma de árvore. As entidades de gerência da rede podem monitorar seus recursos lendo valores dos objetos na MIB e podem controlar os recursos do sistema modificando estes valores.

Para que a MIB supra as necessidades dos sistemas de gerência de rede, precisa atingir os seguintes objetivos:

- O(s) objeto(s) usado(s) para representar um recurso particular precisa(m) ter a mesma definição independente do sistema;

- Um esquema comum para representação precisa ser utilizado para suportar interoperabilidade.

O esquema para suportar interoperabilidade é o SMI (*Structure of Management Information*). O objetivo do SMI é encorajar simplicidade e extensibilidade. Com este esquema a MIB pode armazenar somente tipos simples, evitando, assim, estruturas complexas.

Na estrutura da MIB todos os objetos gerenciados são organizados em uma hierarquia ou estrutura em árvore. Os objetos folha da árvore são, na realidade, os objetos gerenciados, onde cada qual representa algum recurso, atividade ou informação relacionado ao que ele gerencia.

Partindo da raiz, nesta árvore de representação, há três nós: *iso, ccitt e joint-iso-ccitt*. Sobre o nó *iso*, um dos nós é utilizado para uso de outras organizações, que contém outro nó que pertence ao U.S. Department of Defense (*dod*). Como cada um dos nós tem um representador, o nó *internet* tem a representação 1.3.6.1, identificada por:

```
internet OBJECT IDENTIFIER ::= { iso (1) org (3) dod (6) 1 }
```

A partir do nó *internet* existem quatro nós: *directory* - reservado para uso futuro (X.500), *mgmt* - para objetos definidos no IAB, *experimental* - objetos experimentais e *private* - identifica objetos definidos unilateralmente. A classe UNIVERSAL do ASN.1 consiste em tipos de dados para aplicações independentes. Assim, há os seguintes tipos permitidos para serem definidos em um objeto MIB: *integer, octet string, null, object identifier, sequence e sequence-of*. A classe APPLICATION do ASN.1 consiste em tipos de dados que são relevantes a aplicações particulares, onde cada aplicação define seu próprio tipo de dado. A RFC 1155 lista os tipos de dados de aplicações: *networkaddress, ipaddress, counter, gauge, timeticks e opaque*.

Fraquezas do SNMPv1

Como o próprio nome diz, o SNMP é um protocolo muito simples, a ponto de não satisfazer as atuais necessidades para gerência de redes, sendo muito precário nos itens:

- **Não é seguro**: a forma de autenticação é muito trivial, por isto muitos gerentes de rede desabilitam a possibilidade de utilizar o comando SET, impossibilitando a rede de ser "configurada", permitindo, às pessoas, apenas a visualização das configurações;
- **Não é eficiente**: não é capaz de retornar um grande volume de dados (*bulk data*), para cada pacote é preciso fazer uma requisição;
- **Faltam funções importantes**: o protocolo não pode ser utilizado para criar novas variáveis na MIB, não permite executar comandos de gerência e não permite comunicação gerente-gerente;
- **Falta de confiabilidade**: foi construído sobre UDP o qual não é seguro.

3.3.3 SNMPv2

Para preencher o vazio deixado pelo SNMP foi criado o SNMPv2, onde foram introduzidos itens para recuperação de grandes quantidades e facilidades de segurança, descritos abaixo:

- **GET-BULK**: permite retornar um grande volume de dados;
- **Definições para gerentes de gerentes**: o *Manager-to-manager (M2M)* MIB foi criado para suportar esta topologia;
- **Segurança**: foram feitas duas propostas - USEC - (*User-based Security Model*) e SNMPv2* - mas o modelo USEC era "inseguro" demais e SNMPv2* era complexo demais.

3.3.4 SNMPv3

As principais mudanças feitas no SNMPv3 ([Bac01]) são:

- **Contadores 64 bits**: necessários para poder suportar Gigabit Ethernet;
- **Operador *set* melhorado**: as operações de *set* podem ser testadas para garantir que foi completada com sucesso;
- **Gerenciamento remoto total de dispositivos SNMP**: permite alterar parâmetros de configuração de um agente SNMP utilizando o próprio SNMP, permitindo o gerenciamento remoto total de um dispositivo SNMP;
- **ID único para cada *engine* SNMP**: conjuntamente com a habilidade de endereçar múltiplos contextos dentro de um dispositivo gerenciado, facilita rastrear relações na topologia da rede, auxilia na autenticação e endereça componentes de redes mais complexas que tem múltiplos contextos lógicos com um simples dispositivo gerenciado. Por exemplo, cada porta de um switch pode ser endereçada como uma bridge dentro do objeto switch;
- **Modelo de segurança muito mais poderoso**.

Segurança

O calcanhar de Aquiles do SNMP é a segurança. Este protocolo tem um potencial enorme para liberar uma grande quantidade de informações a qualquer um - inclusive um hacker. Dando acesso físico à rede e tendo a mão um analisador de protocolos, um hacker pode utilizar o SNMP para obter um diagrama da topologia da rede e configurações. Pior ainda, interceptando as operações de set - as quais são baseadas em strings - ele é capaz de alterar configurações de qualquer dispositivo SNMP. Atualmente, os agentes SNMP podem utilizar uma lista de acesso baseada apenas em IP, com o intuito de aumentar a segurança.

O SNMPv3, além de criptografar todas as transmissões, permite ao agente autenticar o usuário que gerou a requisição, garantindo a integridade da mensagem através de assinatura digital e aplicando regras para cada requisição de acesso complexo e granular. Também permite ao administrador especificar o nível de proteção de cada combinação (inseguro, autenticado e autenticado com criptografia).

Este nível de segurança não era possível praticar, em hardware, há dez anos atrás. Hoje, os dispositivos de infra-estrutura possuem RAM e CPU que suportam, não apenas, estes avanços da segurança do SNMP, mas serviços de gerência Web - tudo em apenas um firmware.

Outra característica de segurança do SNMPv3 é o USM (*User-based Security Model*) e VACM (*Views-based Access Control Model*), que estão apenas com RFCs. Isto permitiria aos fabricantes suportar SNMP seguro e deixar uma porta aberta para novas melhorias de segurança como chave pública ou diretório, sem comprometer as especificações atuais.

3.3.5 CORBA

A especificação *CORBA Facilities* identifica sistema de gerência como um domínio de *facilities* horizontal. Nove facilities de gerência de sistemas são identificadas [OMG98]:

- **Gerência de políticas**: envolve a definição de políticas e sua aplicação na gerência de componentes.
- **Gerência da qualidade de serviço**: permite especificar o nível para disponibilidade, desempenho, confiança e recuperação;
- **Coleção de dados**: definido como processo de recolhimento da informação;
- **Segurança**: provê uma interface comum para a gerência de segurança dos recursos do sistema, independente da implementação da segurança;
- **Gerência de coleções**: define as operações requeridas para haver referências *two-way* entre objetos. É especificado através de *Managed Sets* como parte da especificação XCMF (*common management facilities*);
- **Gerência de agendamento**;
- **Customização**: mecanismo para instâncias de objetos serem estendidas dinamicamente, enquanto mantém segurança de tipos;
- **Gerência de eventos**: incluem geração, registro, filtro, agregação e encaminhamento de notificação de eventos para as aplicações de gerência;

- **Instrumentação**: uma interface comum para coletar, gerenciar e disseminar os dados de um recurso específico no suporte da gerência de sistemas;

A especificação XCMF (*common management facilitie*) é a especificação detalhada de parte das facilities identificadas na arquitetura. XCMF é uma categoria de serviços definidos no XRM (*Systems Management Reference Model*).

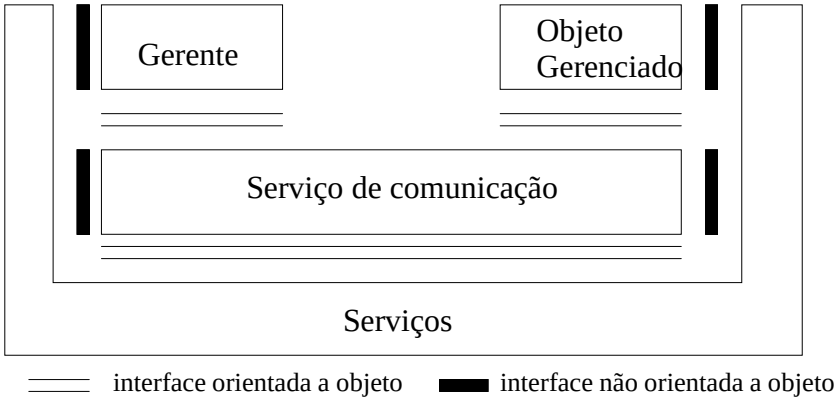

Figura 3.5: Modelo XRM

No XRM não existem agentes, ao invés, os gerentes acessam diretamente os objetos gerenciados através do serviço de comunicação. O XRM possui três componentes:

- **Gerente**: aplicações que implementam funcionalidades do gerente;
- **Objetos gerenciados**: entidades de software que provêem uma interface padrão sobre o comportamento dos recursos do sistema;
- **Serviços**: provê *facilities* comuns, necessárias a todos os componentes.

JIDM

O consórcio X/Open (hoje parte do Open Group) e o NMF (*Network Management Forum*) formaram o grupo JIDM (*Joint Inter-Domin Management*) para especificar o mapeamento entre diferentes domínios de gerência. Os resultados são as especificações ISO e IIMC (*Internet Management Coexistence*) que permitem OSI SM e Internet Management se comunicarem através de XOM (*common Object Model*) e XMP (*common Management Protocol*) [GXF95].

3.3.6 CIM

Idealmente, a informação utilizada para executar tarefas é organizada ou estruturada de modo que permita que grupos ou pessoas diferentes possam usá-la. Isto pode ser completado pelo desenvolvimento de um modelo ou representação de detalhes requeridos pelas pessoas que trabalham com um domínio particular. Tal abordagem pode ser referida como um modelo de informação [DMT99c].

Um modelo de informações requer um conjunto de declarações de tipos ou sintaxe para capturar a representação e uma coleção de expressões atuais necessária para gerenciar aspectos comuns de um domínio.

O CIM (*Common Information Model*) é um modelo de informações orientado a objetos, que se divide em esquemas e formato de objetos gerenciados (MOF). A chave para entender o CIM é entender o modelo de orientação a objetos e como os objetos se relacionam com o esquema CIM. Também serão conceituados:

- Modelo orientado a objetos;
- Classes;
- Superclasses e Subclasses;
- Métodos;
- Propriedades;
- Indicações, associações e referências;
- Qualificadores e sobrecarga;
- Representações estatísticas;
- MOF;
- Esquemas;
- Core model;
- Common model;
- Esquemas de extensão;

Modelo orientado a objetos

Modelo orientado a objetos é um caminho formal para representar algo no mundo real. É feito a partir da teoria dos conjuntos tradicionais e da teoria de classificações. Alguns conceitos básicos [DMT01]:

- Instâncias são coisas;
- Propriedades são atributos;
- Relações são pares de atributos;
- Classes são tipos de coisas;
- Subclasses são subtipos de coisas.

Classes

Classe é uma coleção ou conjunto de objetos com propriedades similares que compartilham dos mesmos objetivos.

Uma classe define a unidade básica de gerência e cada classe é um modelo para um tipo de objeto gerenciado; todas as instâncias do tipo utilizam o modelo. Uma classe contém propriedades e métodos e pode participar de associações onde são o alvo de uma referência feitos pela associação.

Fabricantes independentes podem definir classes customizadas para suportar objetos gerenciados que são específicos para seu ambiente. Por exemplo, uma classe pode ser customizada para representar um banco de dados ou uma fita.

Existe uma hierarquia de classes surgindo, daí, a superclasse e a subclasse. Subclasse é a classe derivada da superclasse. Nesta hierarquia apenas as características são herdadas, sendo necessário implementar as propriedades e métodos na classe herdada.

As classes podem ser estáticas ou dinâmicas. Instâncias de classes estáticas são armazenadas pelo gerente de objetos CIMOM (*CIM Object Manager*) e obtidas através de pedidos. Instâncias de classes dinâmicas são geradas por um provedor.

Superclasses e Subclasses

Classes seguem uma estrutura de hierarquia e podem ter subclasses, também conhecidas como especialização. O pai de uma subclasse é conhecido como superclasse ou generalização. A classe que não tem superclasse é a classe base.

Métodos

Um método é uma operação que descreve o comportamento de uma classe. Nem todas as classes possuem métodos. Um método tem um nome e somente um domínio: a classe a que pertence, e tem uma relação de sobreposição com o método de outra classe. O fato de uma classe ter um método não garante a implementação deste. Em CIM, quando uma classe herda um método, herda apenas a definição, não a implementação.

Propriedades

Propriedade é um valor usado para denotar uma característica de uma classe. Pode ser vista como um par de funções, uma para armazenar o valor e outra para retorná-lo. Caso a propriedade seja apenas de leitura existirá apenas o método de leitura.

A propriedade tem um nome e somente um domínio: a classe a que pertence. Uma propriedade pode ter uma relação de sobreposição com a propriedade de outra classe. O fato de uma classe ter uma propriedade não garante a implementação desta. Quando uma classe herda uma propriedade, herda apenas a definição, não a implementação.

Indicações, associações e referências

Uma indicação é um objeto criado como resultado de um evento. Indicações são um subtipo de classe, podendo ter propriedades e métodos.

Uma associação é uma classe que contém duas ou mais referências, representando um relacionamento entre dois ou mais objetos. É um tipo de classe onde também há um qualificador. Somente associações podem conter referências.

No CIM é muito importante que não seja utilizado o inverso de uma associação.

Referência é uma propriedade especial declarada com a palavra chave REF e representa o nome da regra de uma classe no contexto da associação.

Na figura 3.7 *ContainedElement* e *ContainingElement* são referências, *PhysicalElements* é uma faixa (*range*) e *PhysicalPackage* é um domínio.

Qualificadores e sobrecarga

Qualificadores são utilizados para caracterizar classes, propriedades e outros elementos do esquema. Eles provêem um mecanismo que faz a linguagem de definição de um esquema extensível, de maneira controlada e limitada. Podem ser classes, propriedades, métodos e outros elementos do esquema.

Um qualificador pode ter um nome, um tipo, valor deste tipo, um escopo e um sabor (*flavor*) (o sabor descreve regras que especificam como um qualificador pode ser propagado

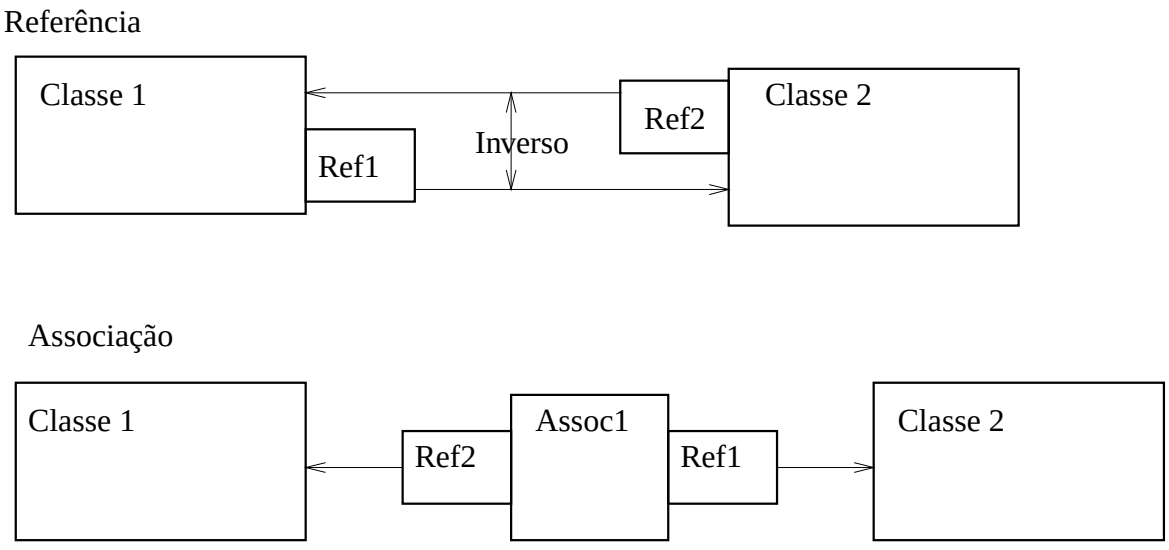

Figura 3.6: Esquema

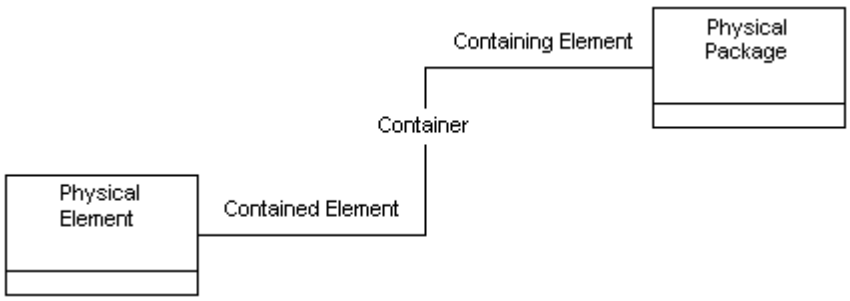

Figura 3.7: Domínio e faixa

para derivar classes e instâncias e quando uma classe derivada ou instância pode, ou não, sobrescrever o valor original do qualificador).

A relação de sobrecarga é utilizada para indicar a substituição entre uma propriedade ou método de uma subclasse e a propriedade ou método sobreposta da superclasse.

Representações de estatísticas

No modelo CIM as estatísticas são feitas separando as propriedades estatísticas em uma classe que é descendente da classe StatisticalInformation. Esta classe provê uma classe base, na qual todas as propriedades de agrupamento estatístico são colocadas.

Algumas informações estatísticas são localizadas diretamente nas classes envolvidas, por exemplo, EstimatedChargeRemaining ou EstimatedRunTime na CIM_Battery - entretanto,

é uma exceção. No curso normal dos eventos, é essencial que as estatísticas sobre objetos gerenciados sejam separadas do objeto em si. A razão é que estatísticas são específicas para uma visão particular do objeto. Ao colocar as informações estatísticas no objeto em si, o esquema as acessará toda vez que for acessar o objeto. Se o objeto contém uma variedade de informações estatísticas, o usuário precisa selecionar, toda vez, as propriedades de interesse para evitar processamento à toa. Outra característica dos valores estatísticos é que muitas vezes vêm em grupos que não se aplicam a um objeto específico.

É importante notar que estatísticas no CIMv2 não são rastreadas por tempo. Como resultado, o caso tradicional é que as classes de estatísticas serão *singleton*, isto é, classes que contém apenas uma instância no sistema, contendo os valores atuais das estatísticas.

MOF

A gerência da informação é descrita em uma linguagem baseada na IDL (*Interface Definition Language*) chamada MOF (*Managed Object Format*). A sintaxe MOF é uma forma de descrever as definições de objetos em uma forma textual. Isto estabelece a sintaxe para escrever definições. Os principais componentes de uma especificação MOF são descrições textuais de classes, associações, propriedades, referências, métodos e declaração de instâncias e seus qualificadores associados. Comentários são permitidos.

Até o momento, nenhuma interface ou protocolo foi definido para a troca de mecanismos. Entretanto, um sistema CIM deve ser capaz de importar e exportar, apropriadamente, construções dadas no formato MOF. Como importar e exportar são detalhes de implementação do sistema.

Exemplo de um arquivo MOF:

```
1  [abstract]
2  class Win32_LogicalDisk
3  {
4  [read]
5  string DriveLetter;
6  [read, Units("KiloBytes")]
7  sint32 RawCapacity = 0;
8  [write]
9  string VolumeLabel;
10
11 [Dangerous]
12 boolean Format([in] boolean FastFormat);
13 };
14
```

Esquemas

Um esquema (*schema*) é uma abstração de algo que existe no mundo real. Em CIM, um esquema é um conjunto de classes com um único dono. Um esquema tem um nome e uma coleção de classes e uma classe pertence a apenas um esquema.

Os esquemas são utilizados para administrar e nomear classes únicas. O nome do esquema é um fator determinante para diferenciar classes e propriedades de outros que podem ter o mesmo nome. O nome de um esquema, classe ou propriedade seguem a sintaxe:

Nomedoesquema_nomedaclasse.nomedapropriedade

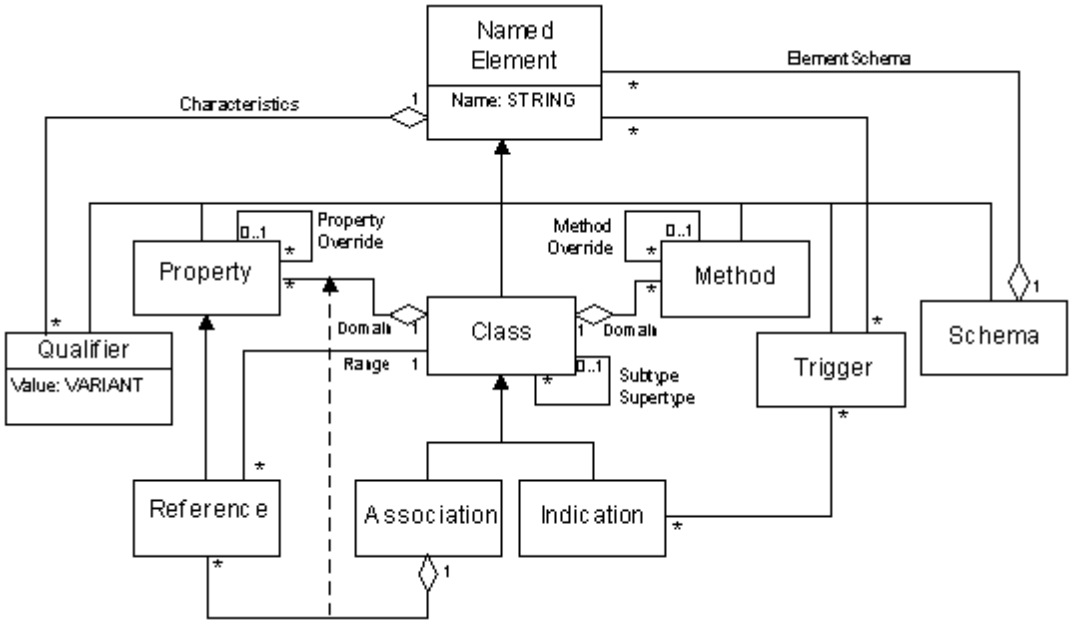

Figura 3.8: Metaschema (obtido de [DMT01])

O esquema CIM é dividido em:

- Core model;
- Common model;
- Esquemas de extensão.

Core model

Core model é um pequeno conjunto de classes, associações e propriedades que provêem

um vocabulário básico para analisar e descrever um sistema de gerência. O *Core model* representa o ponto de início para que o analista determine como estender o esquema comum.

Possíveis extensões incluem a adição de uma classe “Elemento Gerenciado” ou a abstração de uma classe “Elemento do Sistema Gerenciado”. Descendentes desta classe “Elemento Gerenciado”, isto é, classes que representam objetos fora do domínio do “Sistema Gerenciado” podem ser adicionadas ao *Core model*. Por exemplo: desejar que uma aplicação tenha objetos como “Organização” e “Usuários”.

O *Core model* não é parte do meta esquema, mas é parte importante para introduzir uniformidade através dos esquemas que representam aspectos dos ambientes gerenciados.

O *Core model* é um subconjunto do CIM, não específico para nenhuma plataforma. O *Core model* é um conjunto de classes e associações que estabelecem uma estrutura conceitual para o esquema e o resto dos ambientes gerenciados. Sistemas, aplicações, redes e informações relacionadas são modelos estendidos do *Core model*.

Aspectos sistêmicos do *Core model*

Os elementos que compõem um sistema podem ser:

- **Físico**: ocupa espaço e está de acordo com as leis elementares da física;
- **Lógico**: representa uma abstração utilizada para gerenciar e coordenar aspectos do ambiente físico. Elementos lógicos representam, tipicamente, estados do sistema ou *capabilities* do sistema.

Elementos lógicos podem ser:

- **Sistemas**: um grupo de outros elementos lógicos. Desde que sistema é um conjunto de Elementos Lógicos propriamente ditos, um sistema pode ser composto por outros;
- **Componentes de uma rede**: classes utilizadas para prover uma visão topológica da rede;
- **Serviços e pontos de acesso**: provêem um mecanismo para organizar as estruturas que provêem acesso às *capabilities* do sistema;
- **Dispositivos**: uma abstração ou emulação de um dispositivo físico, que pode, ou não, ser realizado fisicamente.

Os aspectos sistêmicos de um *Core Schema* são cobertos pelas seguintes classes:

- `ManagedSystemElement`: esta é a classe base para a hierarquia dos elementos de sistema. Exemplos: componentes de software, como arquivos; e dispositivos, como *disk drives* e controladores, e componentes físicos, como chips e placas;

- `LogicalElement`: esta classe é a base para todos os componentes de um sistema que representam componentes abstratos como perfis, processos ou *system capabilities* na forma de dispositivos lógicos;

- `System`: um elemento lógico que agregue um conjunto numerável de Elementos de Sistema Gerenciados. A agregação opera como um todo. Dentro de alguma subclasse particular do sistema, há uma lista bem definida das classes de ManagedSystemElement, cujos exemplos devem ser agregados.

- `Service`: um elemento lógico que contém as informações necessárias para representar e gerenciar as funções providas por um dispositivo e/ou característica de software. Um Serviço é um objeto de uso geral para configurar e gerenciar a implementação de funcionalidades. Ele não é a funcionalidade propriamente dita.

Common model

O *Common model* é um modelo de informação que captura noções que são comuns às áreas de gerência particulares, mas não é dependente de uma tecnologia particular ou implementação. O modelo de informação é específico, suficiente para prover a base para o desenvolvimento de uma aplicação de gerência. Este modelo provê um conjunto de classes base para extensão dos esquemas específicos de tecnologia. O *Core* e *Common model* são expressos como “esquema CIM” [DMT01]. As áreas comuns são definidas como:

- Sistemas;

- Dispositivos;

- Aplicações;

- Redes;

- Físico.

Esquemas de extensão

Os esquemas de extensão são compostos de classes que representam objetos controlados que são adições específicas de tecnologia ao *Common model*. Estas classes, tipicamente, aplicam-se a ambientes como os sistemas operacionais UNIX ou Windows.

3.3.7 WBEM

A iniciativa WBEM (*Web-Based Enterprise Management*) é um conjunto de tecnologias padrão de gerência e Internet, desenvolvido para unificar os ambientes de gerência. WBEM provê para a indústria um grupo bem integrado de ferramentas de gerência padrão. Neste, a DMTF desenvolveu um conjunto base de padrões que compõe o WBEM, incluindo CIM, xmlCIM encoding e o mecanismo de transporte que consiste em operações CIM sobre HTTP [DMT99b].

Esta especificação define todas as interações entre produtos CIM e mensagens CIM. Ou seja, o WBEM está mais relacionado em como a comunicação é feita, enquanto o CIM à quais dados são trocados na comunicação.

Uma mensagem CIM é um padrão bem definido de uma requisição ou resposta para troca de informações entre produtos CIM. Atualmente, existem dois tipos de mensagens CIM: mensagens de operações CIM e mensagens de exportação CIM.

Mensagem de operação CIM é usada para invocar uma operação no seu alvo. Mensagem de exportação CIM é apenas informacional e não define uma operação.

Invocação de métodos

Toda requisição de operação CIM no WBEM é definida como invocação de um ou mais métodos. Estas requisições são operações WBEM que agem sobre dados definidos no CIM. Um método pode ser definido como:

- **Intrínsico**: significa que é definido por esta especificação com o objetivo de modelar uma operação CIM;
- **Entrínsico**: significa que o método é definido por uma classe CIM em algum esquema.

Um método intrínseco é caracterizado pelo fato de ser feito contra um *namespace CIM*. Métodos extrínsecos são invocados através do WBEM em uma classe CIM (se estático) ou instância (se não estático).

Uma invocação de um método pode ser feita em uma operação simples ou múltipla. Operação múltipla é um conjunto de operações simples e pode ser feita de forma *batch* em uma simples operação HTTP, diminuindo também o número de *roundtrips* entre o cliente e servidor. Os métodos intrínsecos do WBEM podem ser chamados em um *namespace CIM*, sendo alguns:

- **GetClass**: usada para retornar uma única classe CIM;
- **GetInstance**: retorna uma instância da classe CIM;
- **DeleteClass**: deleta uma classe CIM;
- **DeleteInstance**: deleta uma instância;
- **CreateClass**: cria uma classe nova;
- **CreateInstance**: cria uma nova instância de uma classe;
- **EnumarateClasses**: enumera subclasses de uma classe CIM;
- **EnumerateInstances**: enumera o número de instâncias de uma classe;
- **ExecQuery**: execute a query;
- **GetProperty**: obtém o valor de uma propriedade;
- **SetProperty**: armazena o valor de uma propriedade;

3.3.8 JMX

Baseado em tecnologias comprovadas e na tecnologia desenvolvida através do *Java Community Process*, a especificação do JMX (*Java Management eXtensions*) é um padrão para gerenciar redes, aplicações, dispositivos, etc., através de Java. O JMX é uma extensão aberta e universal da linguagem Java e permite que corporações e provedores de serviços gerenciem ambientes heterogêneos de uma maneira padrão [Mic99].

Devido ao fato do JMX ser aberto, um desenvolvedor precisa, apenas, preocupar-se com que seu produto seja gerenciável através do JMX, assim ele poderá, automaticamente, ser inserido em outros ambientes, como TMN e SNMP, sem nenhuma modificação.

A troca rápida de tecnologias quase obriga a adotar um modelo aberto onde os serviços possam ser acrescentados dinamicamente. Outra vantagem deste modelo é devido à evolução de uma gerência cliente-servidor para uma gerência orientada a serviços.

Arquitetura

A arquitetura do JMX é divida em três níveis:

- **Instrumentação**: provê gerenciamento instantâneo para “qualquer” objeto baseado em Java;
- **Agente**: são contêineres que provêem o coração dos serviços de gerência que podem ser dinamicamente estendidos adicionando recursos JMX;
- **Gerência**: provê componentes que podem operar como um gerente ou agente para distribuição e consolidação dos serviços de gerência.

JMX provê um número de APIs para os protocolos padrões de gerência existentes, independentes do modelo de três níveis e conhecidas como “APIs do protocolo de gerência adicional”. Estas são responsáveis por fazer a integração com as soluções de gerência existentes.

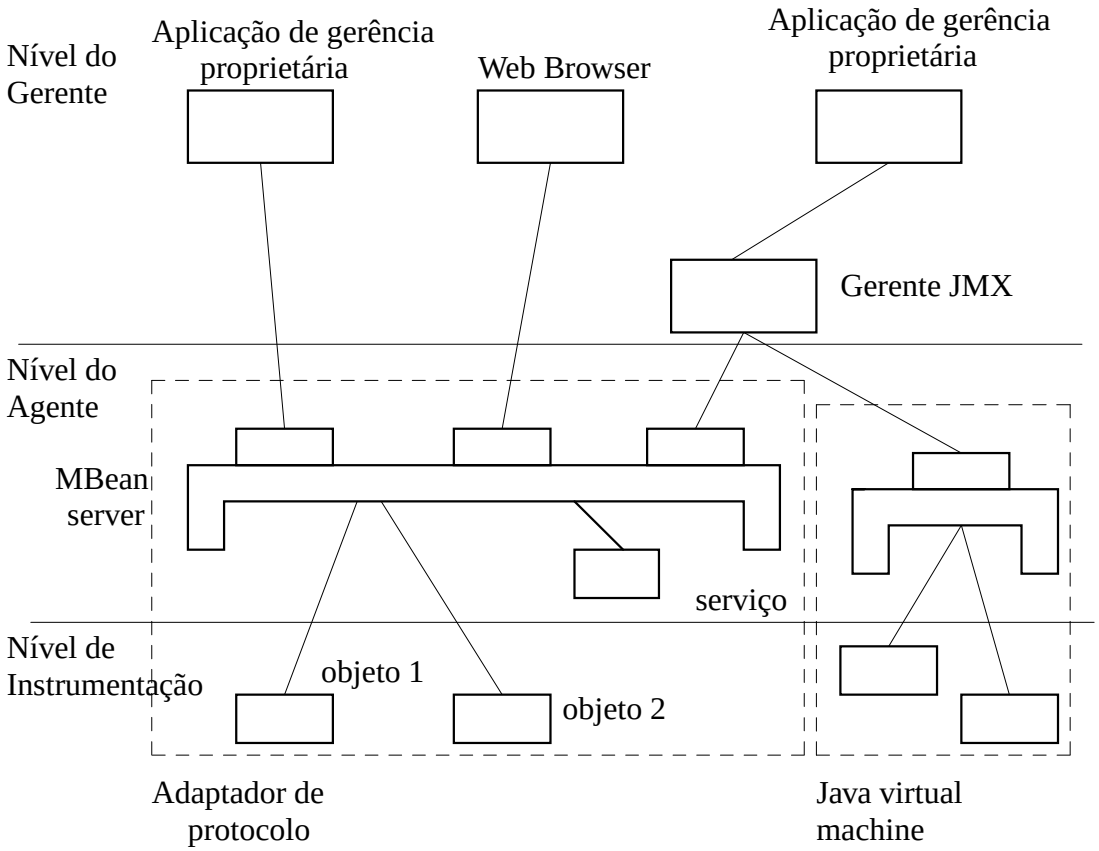

Figura 3.9: Componentes chave do JMX

Recurso gerenciável

Pode ser uma aplicação, um dispositivo, implementação de um serviço ou política. Um bean gerenciável (Mbean - *Managed bean*) é um objeto Java que representa um recurso gerenciável JMX. Um Mbean só é acessível através do agente, preservando o modelo cliente-servidor necessário para algumas *querys* e segurança.

Agente

Um agente é composto de um Mbean servidor ou conjunto de Mbeans representando um recurso gerenciado, um adaptador de protocolo ou um conector. Pode conter serviços de gerência.

Adaptadores de protocolo são MBeans que representam diretamente um protocolo, como HTML ou SNMP. Conectores incluem um componente remoto que provê comunicações, ponto a ponto, com um agente operando sobre uma variedade de protocolos (HTTP, HTTPS, IIOP).

Gerente

O gerente pode controlar um determinado número de agentes, podendo simplificar estruturas de gerência complexas e distribuídas.

Serviços de gerência

Agente e gerente integram serviços que permitem a autonomia e inteligência no agir. Estes serviços tornam possível aos agentes a manipulação de recursos e deixa os gerentes repassarem as informações entre outros agentes ou gerentes. Os agentes são mais autônomos, pois podem incorporar certas tarefas, como *pooling*, e inteligentes, pois podem fazer que o gerente não repasse certos alarmes.

APIs de protocolos de gerência adicionais

O objetivo destas APIs é prover um modo padrão do JMX interagir com tecnologias existentes. Tipicamente, uma aplicação irá usar um ou mais APIs para acessar um sistema legado e expor seus atributos como um recurso gerenciado JMX. Este recurso irá permitir que

qualquer aplicação de gerência compatível com JMX gerencie um sistema legado através do agente JMX. Esta API também cria uma ponte entre as tecnologias atuais e futuras.

JDMK

O JDMK (*Java Dynamic Management Kit*) é uma API Java proprietária da SUN onde um conjunto de ferramentas auxilia a projetar e implementar aplicações de gerência conforme a especificação do JMX.

O JDMK provê um toolkit para integrar gerência SNMP em uma solução JMX, que inclui:

- Desenvolvimento de um agente SNMP com um adaptador de protocolo SNMP;
- Representação de um MIB SNMP como Mbean, gerado através do compilador mibgen;
- Desenvolvimento de um gerente SNMP utilizando a API de gerenciamento SNMP.

3.4 Comparação entre padrões

Abaixo, uma comparação entre SNMP, OSI e CIM, considerando objetivos, interfaces, elementos de um objeto, especificação, taxonomia, referência e seleção de objetos. Não estão incluídos os padrões CORBA, WBEM ou JMX por serem relacionados ao transporte do dado, e não à representação da informação (tabela 3.2).

3.5 Conclusão

Todos os padrões analisados são realmente interessantes, tendo, em suas particularidades, vantagens e desvantagens. Enquanto que o SNMP é o mais difundido é também o que possui mais falhas, o CMIP é um dos mais completos e, também, mais complexos.

A grande promessa é o CIM que, com sua simplicidade, orientação a objetos, grau de completude e adeptos, é o padrão que mais agrada no momento. Dentre os adeptos está a própria IETF, patrocinadora do SNMP, que possui propostas para políticas em CIM - PCIM.

	SNMP	*OSI*	*CIM*
Objetivo	Seu objetivo é gerência da internet e dispositivos de rede baseados em TCP/IP. Seu código não é portável mas tem comunicação interoperável. A vantagem é a gerência de dispositivos *inter-networked*	Seu objetivo é gerência de rede. Seu código não é portável, mas tem comunicação interoperável. A vantagem é a gerência de componentes heterogêneos	Seu objetivo é modelar um conjunto de informações comuns, de modo que possa ser interoperável e expansível. O padrão não discute o meio de transporte, podendo ser usado neste caso o WBEM ou CORBA
Interfaces	Comunicação por objetos que gerenciam interface, operações e notificações. Interface de comunicação entre sistema gerenciado e gerenciador mediados por um agente	Operações *get* e *set* sobre as variáveis. Interface de comunicação entre um agente e gerente onde o agente suporta um ou mais MIBs	O CIM não define as interfaces. O WBEM define operações *get* e *set* sobre as propriedades. Interface de comunicação entre um agente e gerente onde o agente suporta um ou mais MIBs
Características dos objetos	Objetos gerenciados são acessados através de MIBs. Cada tipo tem um nome, sintaxe e um *encoding*. Estes objetos representam variáveis individuais	Um objeto gerenciado é descrito pelo Management Framework como "uma visão de um recurso que pode ser gerenciado através do uso dos protocolos de gerência OSI" [IT89]	Cada objeto gerenciado é representado por uma classe, agrupada em esquemas
Taxonomia	Define tipos de objetos que classificam variáveis individuais	O modelo de gerência de informação classifica objetos gerenciados em classes. As especificações de classes são documentadas por modelos	Diferencia classes, propriedades e métodos, sendo que uma classe pertence a apenas um esquema, e a propriedade/método pertence somente a uma classe
Referência de objetos	Uma referência para um objeto é seu nome. Todos os agentes utilizam nomes idênticos para objetos de um tipo	Não existe um conceito específico de uma referência para um objeto gerenciado a não ser as instâncias de objetos (nomes distintos)	É uma classe definida no padrão
Seleção de objetos	Feita através do nome	Feita através do nome	Feita através do nome

Tabela 3.2: Tabela comparativa dos padrões SNMP, OSI e CIM

Visto a gerência de sistemas, suas áreas e padrões - SNMP, OSI, CORBA, JMX, CIM e WBEM, e uma comparação entre estes, seguem os aspectos da gerência de clusters, onde serão analisados alguns trabalhos existentes relativos à gerência e uma análise crítica destes.

Capítulo 4

Gerência de clusters

Apesar da tecnologia de clusters de PCs encontrar-se bem consolidada, ainda há muitos esforços para se construir ferramentas adequadas à sua gerência. Estas ferramentas devem seguir certos preceitos (interoperável, baixo impacto, expansível e escalável) e precisam se adaptar às características de um cluster.

4.1 Aspectos da gerência de clusters

Para obter um bom resultado ao se projetar um sistema para gerência de clusters, deve-se ter em mente suas características:

1. O cluster pode ser composto de milhares de nós;
2. Estes nós podem ser heterogêneos;
3. O cluster pode aumentar ou diminuir em número de nós dinamicamente;
4. Aplicações de gerência devem interferir o mínimo possível no desempenho do cluster;
5. Alguns clusters podem ter apenas um nó (nó mestre) com acesso externo.

Tendo em mente tais características, o ideal de uma ferramenta que auxilie na gerência de clusters é que seja capaz de:

1. Monitorar CPU, memória, disco, rede e outros recursos locais, de vários nós ao mesmo tempo;
2. Gerar ao menos estatísticas da utilização de CPU e rede dos nós;
3. Manusear configurações do sistema operacional;
4. Enviar um alarme sobre certas circunstâncias, sendo estas configuráveis pelo usuário;
5. Ser escalável;
6. Interferir o mínimo possível (em termos de CPU, memória e rede) na aplicação que estiver rodando no cluster;
7. Ser expansível;
8. Ser capaz de instalar/desinstalar software remotamente, de vários nós ao mesmo tempo;
9. Permitir manusear uma determinada propriedade de vários agentes ao mesmo tempo;
10. Ser possível classificar os nós, através de contextos.

O alarme deve ser configurável, permitindo informar a condição em que deve ser gerado. O alarme deve seguir o mecanismo de ciclo de histerese para não congestionar a rede com alarmes [Sta93].

A ferramenta deve ser expansível, pois se houver necessidade de manusear algum item não previsto será possível incluir itens para gerência da aplicação. É fundamental que a ferramenta seja capaz de manusear algum item, simultaneamente, nos vários nós de um mesmo contexto, possibilitando alterar um parâmetro de configuração dos nós do cluster de desenvolvimento, ou dos nós que rodem Linux.

4.2 Trabalhos existentes

Abaixo, um resumo dos trabalhos existentes relativos à gerência de clusters:

- SUMO - este não é relativo à gerência de clusters, mas foi considerado tema de estudo devido a sua abordagem ser similar à proposta apresentada adiante;
- SMCS;

- ClusterProbe;
- PHOENIX;
- PARMON.

4.2.1 SUMO

Iniciado pelo Centro Nacional de Estudos Espaciais (CNES - *Centre National d'Etude Spatiale*), tem como proposta a abordagem de gerenciar um grande número de sistemas espaciais complexos [BJMR00].

Os requisitos exigidos para esta abordagem são:

- Homogeneização do sistema de informação de gerência;
- Melhorar a competitividade e durabilidade das soluções obtidas através do uso de tecnologias padrões;
- Bom grau de reuso e *factorisation*;
- Alta capacidade de integração com o sistema legado.

Para atender tais requisitos, SUMO combinou os seguintes padrões: arquitetura WBEM, modelo de informações de gerência de sistema CIM e CORBA como infraestrutura de comunicação.

Arquitetura

A arquitetura tem a mesma abordagem que o WBEM. Na estrutura do SUMO, cada objeto OM (*Object Manager*) ou OP (*Object Provider*) é um objeto CORBA.

Os OM oferecem uma interface IDL padrão e extensível, construída a partir das informações CIM geradas pela MIB. Os OP provêem a ponte para SNMP, DMI (configurações do PC) ou outras interfaces através da mesma interface IDL.

CIM

O CIM foi utilizado com o objetivo de resolver a heterogeneidade da informação. Os conceitos de gerência global podem ser sumarizados da seguinte forma: "As potencialidades

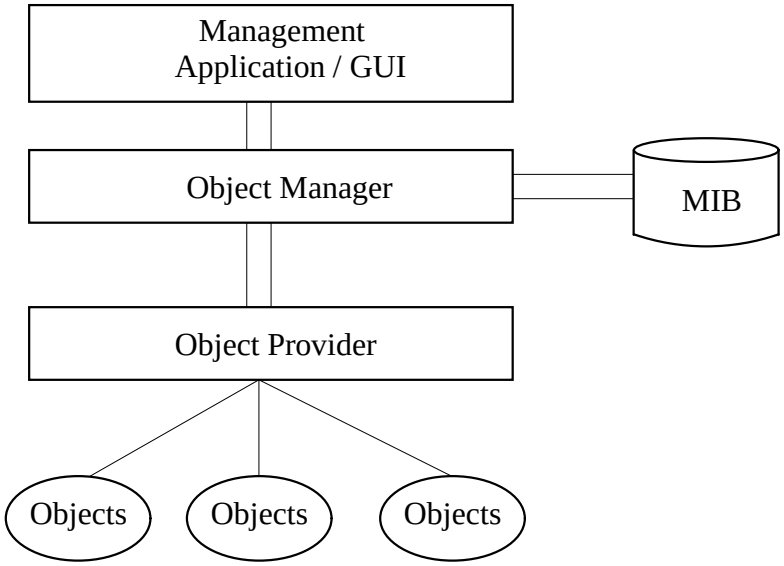

Figura 4.1: Arquitetura WBEM

crescentes da gerência podem ser alcançadas capturando as relações (estática e dinâmicas) entre todos os elementos envolvidos no sistema controlado"[BJMR00].

Devido ao escopo da especificação CIM foram feitas duas extensões:

- SUMO precisou implementar o modelo de gerência CIM e as operações que podem ser feitas na MIB;
- Para que o SUMO fosse operacional e provesse um bom nível de gerência, foram estendidos alguns modelos existentes e criados outros.

Como o CIM foi implementado no SUMO:

- Os dados CIM são armazenados nos OMs do SUMO, utilizando MIB;
- As MIBs dos OMs são originalmente alimentadas com arquivos MOF (*Managed Object Format*);
- As comunicações entre os objetos são feitas utilizando CORBA, mas o tipo de informação trocada na realidade é CIM. Graças a isto, as interfaces são simples e similares às interfaces de gerência tradicionais, contendo os métodos get, set e invoke.

Gerência CORBA com SUMO

Devido ao *CORBA object auto-discovery*, SUMO é capaz de saber quais objetos CORBA estão rodando no seu domínio de gerência, facilitando tarefas simples de gerência.

O *federated naming service* permite oferecer uma maneira simples de instalar, configurar e distribuir um *federated naming service*, que significa criar links entre diversos serviços de nomes e permite vê-los como apenas um serviço. Isto traz diversas vantagens: escalabilidade, desempenho, melhor tolerância a falhas, controle local, etc.

4.2.2 SCMS

O SCMS (*SMILE Cluster Management Systems*) [UPAM00] tem como objetivo sanar alguns dos problemas encontrados por administradores de sistemas em grandes clusters como:

- Ausência de mecanismo de ferramenta para navegar e interagir com componentes internos;
- Ausência de um bom mecanismo para endereçar uma parte do coletivo, enviar comandos e coletar os resultados centralmente;
- Muitos serviços essenciais, como ferramentas de sistemas que monitoram atividades de cada nó, em tempo real, e alarme, não estão disponíveis em uma ferramenta integrada.

A estrutura é composta das seguintes camadas:

- **Camada de interconexão de rede**;
- **Camada de SO**;
- **Camada de imagem única do sistema**: provê uma imagem unificada para o usuário de um único recurso;
- **Camada de serviços do sistema**: contém serviços necessários para um controle efetivo do cluster. Entre estes serviços: monitoração em tempo real, log de desempenho e alarme;
- **Camada de ferramentas do sistema**: coleção de ferramentas que podem ser usadas como *building blocks* para a aplicação de gerência (tabela 4.2);
- **Camada de aplicação**: aplicação e GUI.

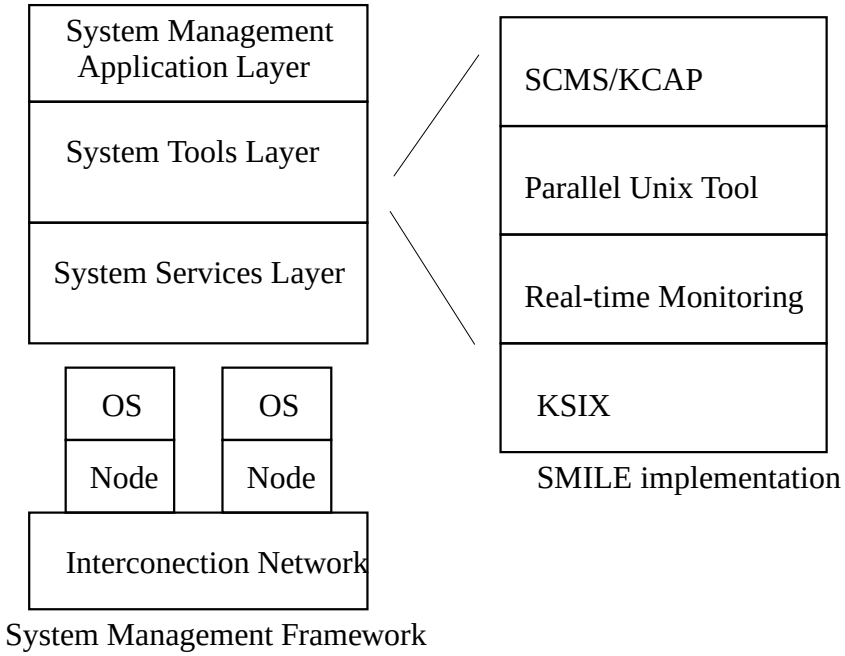

Figura 4.2: Arquitetura SMILE

As principais funcionalidades desta ferramenta são os comandos unix paralelos, o alarme, a monitoração, o serviço de nomes e o serviço de eventos distribuídos (tabela 4.1).

O sistema de monitoração tempo-real portável consiste de um *daemon* chamado CMA (*Control and Monitoring Agent*) que roda em cada nó e coleta estatísticas do sistema, continuamente. Estas estatísticas são reportadas para um servidor de gerência de recursos central chamado SMA (*System Management Agent*). As informações são repassadas, pelo agente, para o gerente através de UDP.

O serviço de alarmes consiste em um processo *daemon* chamado *alarm manager* e cria *daemons* em cada nó chamado *detector*. O relatório pode ser feito através de e-mail e execução de um comando Unix, caso a condição do alarme seja detectada.

O comando unix paralelo provê capacidades mínimas para gerar um comando coletivo e coletar os resultados em um ponto central. A implementação atual roda com desempenho satisfatório em sistemas com 16 nós. Também foi notado que em sistemas que usam NIS para prover uma conta única através do sistema, o comando paralelo é muito mais lento comparado aos sistemas que usam replicação de arquivos. A razão se fundamenta na verificação da conta, que pode ser feita localmente, se todos os arquivos estão replicados, uma vez que o comando rsh cria processos remotos.

Name	*Descrição*
Node Status	Mostra status do nó
Control Panel	Acessa painel de controle de qualquer nó
Disk space	Mostra espaço em disco do nó
FTP	FTP no nó
Node File System	Mostra sistema de arquivos montado
Process Status	Mostra status do processo
Reboot	Reinicia nó
RPM Package	Gerencia de pacotes
Shutdown	Desliga um nó
Telnet	Telnet no nó
User Check	Checa usuário no nó
Start daemon	Inicia daemon de monitoração
System monitor	Mostra monitoração do sistema
Show config	Mostra configuração
Update Config	Atualiza configuração de hardware
Parallel GUI	GUI para todos os comandos paralelos
Alarm GUI	GUI para serviço de sistema de alarme
motherboard	Mostra status *motherboard monitoring*

Tabela 4.1: Funcionalidades do SCMS

Comando	*Descrição*
`pcat`	Concatena arquivos ou mostra conteúdo de arquivos
`pcp`	Distribui arquivos para os nós
`pexec`	Executa um comando em um conjunto de nós
`pfind`	Procura um arquivo em vários nós
`pfps`	Procura um processo no espaço de processos usando nome do processo
`pkill`	Mata um processo no cluster usando nome do processo
`pkillu`	Mata um processo no cluster usando nome do usuário
`pls`	Lista arquivos no cluster
`pload`	Relatório da carga no cluster
`pmv`	Move arquivo para mesmo nó
`ppred`	Executa comando em múltiplos nós condicionalmente usando comand test do Unix
`pps`	Mostra informação do processo em cada nó
`prm`	Apaga um ou vários arquivos em um nó
`ptest`	Testa condição usando comando test do Unix em múltiplos nós

Tabela 4.2: Ferramentas suportadas na implementação atual do SCMS

4.2.3 ClusterProbe

Esta ferramenta teve foco em duas partes: 1) GUI para os administradores visualizarem e gerenciarem o cluster; 2) Módulo de informação sobre utilização de recursos, para ser utilizado no *job scheduling*. As ênfases no projeto foram:

- **Ambiente aberto**: suportar múltiplos protocolos de comunicação, de modo que as aplicações possam se comunicar com a ferramenta para obter informações sobre utilização dos recursos;
- **Flexibilidade**: ser flexível e extensível para se manter atualizada com as trocas dos recursos;
- **Escalabilidade**: monitorar os nós de um grande cluster sem consumir muitos recursos.

A flexibilidade foi alcançada através do projeto do módulo MCI (*Multiprotocol Communication Interface*), capaz de se comunicar através de RMI/Corba, HTTP/HTML, TCP, UDP e SQL.

Para ser escalável foi utilizada uma hierarquia em cascata do domínio de monitoração. Esta hierarquia permite retornar e processar dados em paralelo. É composta de servidor de monitoração, proxy e agente. Deve existir um agente em cada nó, responsável por fazer a interface com os seus recursos.

O servidor de monitoração, que reside em um dos nós com maior poder de processamento e memória, é responsável por manusear as requisições de clientes e encaminhar o resultado de monitoração para o cliente da área de interesse.

As funções do proxy são: 1. registrar ou desregistrar no proxy pai; 2. receber instruções de monitoração do proxy pai e distribuir para seus agentes e proxys filhos; 3. unificar e ordenar dados de monitoração e repassar para o proxy pai; 4. manusear falhas ou repassar para o proxy pai.

4.2.4 PHOENIX

O foco desta ferramenta deteve-se na observação do sistema e na aplicação com granularidade variada [SBF02]. Os requerimentos foram:

- **Livre de intrusão**: a monitoração de uma aplicação necessita preservar seu comportamento;
- **Mínimo overhead**;

- **Iteração mínima com usuário**: a ferramenta não deve alterar o modo como o usuário projeta e implementa;

- **Controle dinâmico**: deve suportar *tracing* seletivo com granularidade variada.

As técnicas de monitoração envolvidas são extremamente importantes para satisfazer a transparência e a portabilidade. Um sistema de observação pode ser implementado com diversas técnicas, com um hardware dedicado, com algum código inserido no kernel, com bibliotecas de monitoração no nível do processo ou com técnicas híbridas. Um dispositivo de hardware tem overhead muito baixo, mas é caro e não portável. Modificar o kernel permite boa transparência, entretanto requer adaptar o código de monitoração toda vez que existir nova versão e não é facilmente portável. Um mecanismo de software é a melhor solução, pela flexibilidade e pelo desempenho razoável.

A observação do sistema é feita através de agentes distribuídos com serviços de monitoração do kernel. Os recursos da aplicação são feitos através de *probes* inseridos automaticamente no fonte da aplicação em tempo de compilação. Os itens monitorados são: comunicação, entrada/saída, memória e processador.

4.2.5 PARMON

Projetada para ser uma ferramenta portável, flexível, interativa, escalável, de locação transparente e ambiente compreensível para monitorar grandes clusters, segue a metodologia cliente/servidor e provê acesso transparente a todos os nós, para serem monitorados de uma máquina de monitoração. Permite ao sistema monitorar atividades de recursos críticos e sua utilização em três níveis: sistema inteiro, nó e componente. Permite monitorar múltiplas instâncias do mesmo componente, como CPU [BMG98].

Os dois principais componentes são *parmon-server* - atividades de recursos do sistema e provedor de informações sobre utilização - e *parmon-client* - responsável pelo GUI, interage com *parmon-server* e usuários para coletar dados em tempo real e apresentar de forma gráfica.

O desenvolvimento do cliente foi feito utilizando Java e o servidor foi desenvolvido em C usando POSIX/Solaris threads. Provê uma imagem única do sistema de qualquer um dos seguintes níveis:

- Hardware;

- Sistema operacional;

- Interfaces de passagem de mensagem;
- Compilador/linguagem;
- Aplicação/ferramentas.

O PARMON permite ao usuário monitorar atividades do sistema e utilização de recursos de vários componentes, dentre eles:

- Utilização de recursos do sistema agregados;
- Atividades de processos;
- Atividades do log do sistema;
- Atividades do kernel;
- Múltiplas instâncias do mesmo recurso;

Arquitetura

O modelo do sistema segue o paradigma cliente/servidor com nós a serem monitorados agindo como servidores e o sistema de monitoração agindo como cliente. Os nós do cluster podem ser monitorados de qualquer estação ou nó do próprio cluster.

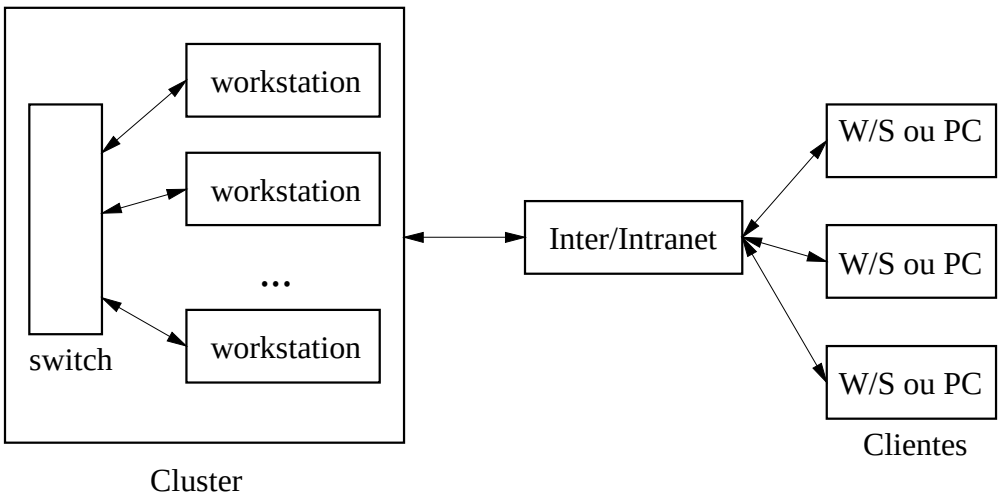

Figura 4.3: Arquitetura PARMON

Características

Algumas características do PARMON para monitoração das atividades e utilização de recursos:

- **Agregação na visualização da utilização de recursos**: o uso de agregação na visualização permite escalar a visualização para o cluster inteiro. A mesma estatística para nós diferentes são combinadas para obter uma estatística única. Esta técnica é chamada utilização grupo/máquina;

- **Atividades dos processos**: a utilização dos recursos da CPU pode ser medida pela monitoração das atividades do processo, a qual ajuda a identificar processos intensivos de CPU/memória.

- **Log do sistema**: permite processar mensagens de sistema arquivos *syslog* por entradas que ocorreram em uma hora específica, ou por entradas que contém uma palavra chave;

- **Atividades do kernel**: suporta instrumentação de software dos recursos do sistema (como CPU, memória, disco e rede) e suas atividades;

- **Visão do componente: física e lógica**: mostra uma figura com sistema físico e alguns poucos componentes, os quais ajudam o usuário a entender rapidamente a visão física da máquina;

- **Controle de dispositivo**: suporta controle de múltiplas instâncias do mesmo recurso;

- **Geração de eventos**: permite ao administrador definir eventos como enviar e-mail quando o usuário ultrapassa limites de utilização de recursos;

- **Diagnóstico**: no Solaris, integra SunVTS para fazer validações, testes e teste de stress;

- **Representação de dados**: usa gráficos de pizza, barra e linha para representar utilização de recursos como disco e memória;

- **Miscelânea**: 1. permite ao usuário especificar um comando para ser executado nos nós selecionados; 2. lista de usuários trabalhando nos nós selecionados; 3. suporta broadcast de mensagens; 4. retorna informações de sistema, configuração e pacotes instalados.

4.3 Análise crítica

Agora, uma análise crítica das ferramentas acima, mostrando suas deficiências e aspectos comuns.

4.3.1 Deficiências

Alguns problemas encontrados nas ferramentas analisadas:

- **SUMO**: não é uma ferramenta desenvolvida com o objetivo de monitorar/gerenciar clusters, não sendo possível adereçar uma alteração para diversos nós, também não mencionando nada a respeito de monitoração ou alarmes;
- **SCMS**: ao contrário do SUMO foi desenvolvida tendo como foco único gerenciar clusters, mas é limitada e pouco expansível. Não usa um modelo de informação, tornando mais difícil sua utilização em ambientes heterogêneos;
- **ClusterProbe**: focando ambiente aberto (múltiplos protocolos), flexibilidade e escalabilidade, não utiliza um modelo de informação, tornando mais difícil sua utilização em ambientes heterogêneos;
- **PHOENIX**: ferramenta para monitoração de clusters, falha na gerência, limitada e de difícil expansão. Também não usa um modelo de informação, tornando mais difícil sua utilização em ambientes heterogêneos;
- **PARMON**: ferramenta para monitoração de clusters, falha na gerência, limitada e de difícil expansão. Além de usar TCP/IP, também não usa um modelo de informação, tornando mais difícil sua utilização em ambientes heterogêneos;

4.3.2 Tecnologias usadas

Tecnologias utilizadas nos trabalhos:

- **SUMO**: desenvolvido em Java, utilizando CORBA e CIM para modelagem da informação;
- **SCMS**: desenvolvido em Java, a monitoração é feita através de UDP e a interface com o usuário é através de HTML, servlets e VRML;

- **ClusterProbe**: desenvolvido em Java, utiliza RMI para comunicação;
- **PHOENIX**: não menciona nada a respeito da tecnologia usada;
- **PARMON**: o cliente foi desenvolvido em Java e o servidor desenvolvido em C ou Java. A comunicação é feita através de sockets (TCP/IP).

4.3.3 Aspectos comuns

Todos os trabalhos apresentados seguem o modelo gerente/agente para monitoração, onde existe um programa, em cada nó, reportando seus dados para um programa gerente.

Dentre as ferramentas de monitoração de clusters (não incluído SUMO) os itens comuns para monitoração são:

- CPU;
- Memória;
- Disco;
- Rede.

As ferramentas ClusterProbe e PHOENIX prevêem a inclusão de novos itens para serem monitorados. Aqui não se incluiu o SUMO por se basear na MIB do CIM, e por não ser uma ferramenta projetada para gerência de clusters.

A linguagem usada para implementação, em quase todos, foi Java, sendo que o PARMON utilizou Java no cliente, Java e C no servidor. PHOENIX não mencionou nada a respeito.

4.4 Conclusão

Infelizmente, os trabalhos direcionados para clusters não são muito abrangentes, ou de difícil expansibilidade por não se utilizarem padrões existentes para a representação da informação.

Assim, percebe-se a ausência de uma ferramenta que auxilie na monitoração e gerência de um cluster, a necessidade de utilização de padrões para facilitar esta gerência e que este padrão para gerência de informações seja expansível e hábil para trabalhar com ambiente heterogêneos.

A seguir, será feita a proposta de uma ferramenta para gerência de clusters abertos baseado em CIM, CORBA e Java. Também será feita uma proposta para extensão do CIM, a classe Cluster.

Capítulo 5

Arquitetura proposta

Será mostrada e detalhada a arquitetura proposta da ferramenta de gerência de clusters, bem como as classes e serviços CORBA utilizados, além da extensão do CIM, a classe Cluster que tem por objetivo fazer com que a gerência do cluster seja feita como um recurso único.

5.1 Ambiente

O desenvolvimento da ferramenta será feito em Java, utilizando CORBA para comunicação entre objetos e a gerência de informações será baseada no padrão CIM.

Como o desenvolvimento será feito utilizando Java, logicamente será necessário ter o JRE em todas as máquinas onde estiver rodando o gerente ou agente, além do CORBA. A linguagem Java foi escolhida devido à transparência ao utilizar CORBA e multi-plataforma [LPR97].

O CORBA utilizado será o JacORB 1.4 que, além de ser freeware, é multi-plataforma e dispõe de vários serviços, dentre eles DII (*Dynamic Invocation Interface*) e DSI (*Dynamic Skeleton Interface*), serviço de eventos, serviço de nomes, serviço de transações, SSL, etc. O CORBA foi escolhido como *framework* para transporte dos dados por ser heterogêneo, de fácil programação e repleto de serviços [OMG95a] [Fel98].

"O modelo usado para implementar um sistema é, geralmente, diferente do modelo usado para gerenciá-lo. Um sistema distribuído, por exemplo, pode ser implementado utilizando

remote procedure call - RPC - mas gerenciado utilizando SNMP ou CMIP. Recentemente, com o advento de sistemas distribuídos orientados a objetos como CORBA, operações (i.e. implementações) e gerência tendem a tornar-se o mesmo, e.g. CORBA tende a ser usado para implementar e gerenciar o sistema. Isto pode ser conseguido ao adicionar objetos CORBA ao sistema cuja tarefa é gerenciar outros objetos, ou adicionando operações às interfaces CORBA para fazer-las gerenciáveis"[Ban97].

"O aumento da capacidade de gerenciamento pode ser atingido ao capturar a relação (estática e dinâmica) entre todos os elementos envolvidos na gerência do sistema. Para tanto, uma linguagem de modelagem com conceitos genéricos é necessária para prover esta "captura". Isto é possível através do *Common Information Model* - CIM, que provê um novo modelo de referência comum e um meta modelo para construir um repositório de informações de gerência"[BJMR00].

A ferramenta irá utilizar CIM devido ao seu repositório de informação homogênea, o que provê a capacidade de gerência de ambientes heterogêneos e sua capacidade de expansão, sendo possível criar um modelo específico para aplicações paralelas ou PCIM (Policy CIM), padrão já proposto pela IETF [IET02]. Não será usado SNMP por ser um padrão mais rígido, não objetivar lidar com ambientes heterogêneos e não ser muito seguro. Apesar de, neste momento, não estar sendo contemplada a segurança, nada impede que seja adicionado um módulo de segurança efetivo - o que não acontece com o SNMP.

5.2 Arquitetura

A arquitetura da ferramenta (figura 5.1) foi baseada na arquitetura proposta no WBEM pela DMTF sem necessariamente utilizar algum componente WBEM e será composta dos seguintes itens:

- **GUI**: interface gráfica responsável pela interação com o usuário. A GUI foi projetada para ser executada fora do cluster, sendo que suas requisições serão encaminhadas aos OPs (*Object Provider*) através do OM (*Object Manager* - deverá estar no nó mestre do cluster);

- **Object Manager**: o OM será um objeto CORBA que fará as vezes de um proxy, responsável por receber as requisições da GUI - que estará rodando fora do cluster - e repassá-las para os OPs. O caminho inverso não será válido já que os OPs irão repassar os dados através do serviço de eventos - como é o caso da monitoração e alarme. O OPs irão se registrar no OM - responsável pelo controle dos OPs (nós)

existentes. O OM também é responsável pelo controle de instâncias dos objetos CIM - quando a GUI faz uma requisição para obter um *handle* para uma instância o OM é o responsável por prover este *handle*. As classes CIM estão em um arquivo junto com o OM devendo distribuir o conteúdo deste arquivo para a GUI, quando for iniciada, desde que as versões sejam diferentes. Com isto, as classes CIM que forem adicionadas serão atualizadas apenas no OM;

- **Object Provider**: os OPs também serão objetos CORBA, receberão as requisições da GUI - através do OM. Estas requisições poderão ser dos seguintes tipos: 1. questionando o valor do atributo de algum objeto CIM; 2. ativando a monitoração de alguma propriedade; 3. ativando um alarme, de acordo com o valor de alguma propriedade; 4. invocando algum método.

- **Objetos CIM**: os objetos CIM irão seguir a estrutura de informação determinada pelo padrão CIM. Cada instância de um objeto CIM é representada por um *handle* - controlada pelo OM, e única em todo o cluster. Assim, através do *handle* é possível identificar o nó e a instância do objeto CIM.

5.2.1 Diagrama de eventos

A seguir os diagramas de eventos para:

- A GUI obter o valor de uma propriedade (fig. 5.2);

- Monitorar uma propriedade (fig. 5.3);

- Ativar um alarme de uma propriedade (fig. 5.4).

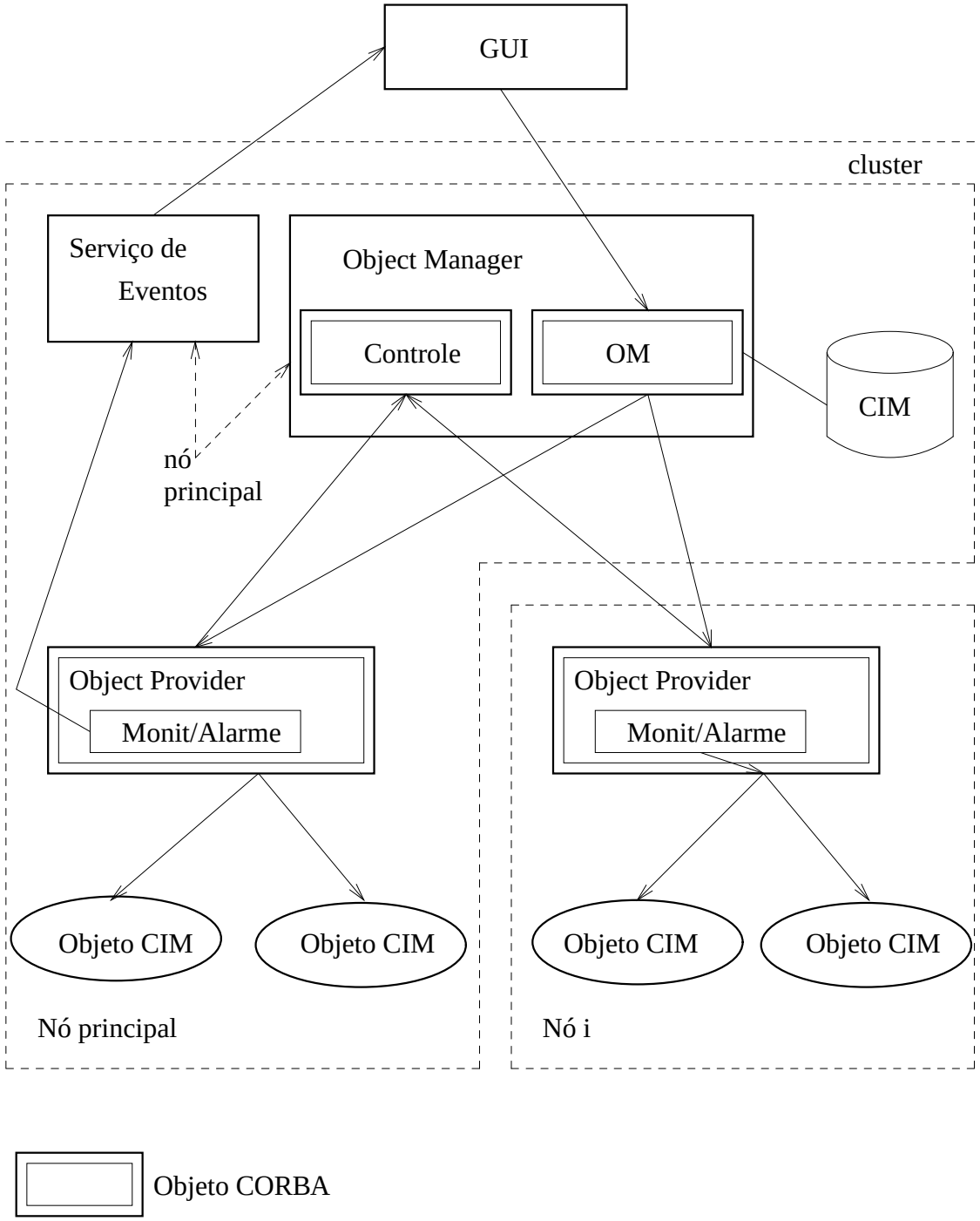

Figura 5.1: Arquitetura proposta

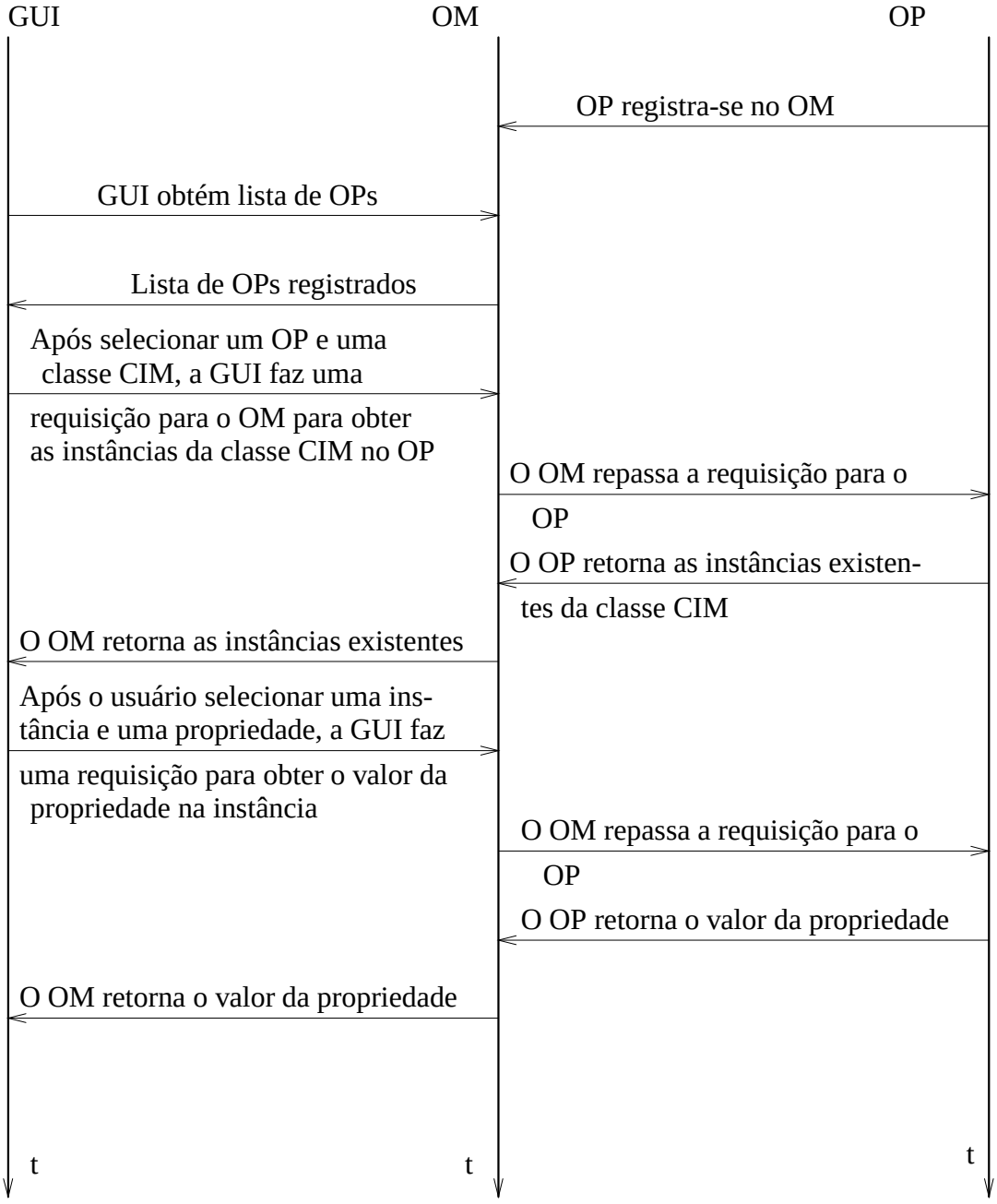

Figura 5.2: Diagrama de eventos para obter o valor de uma propriedade

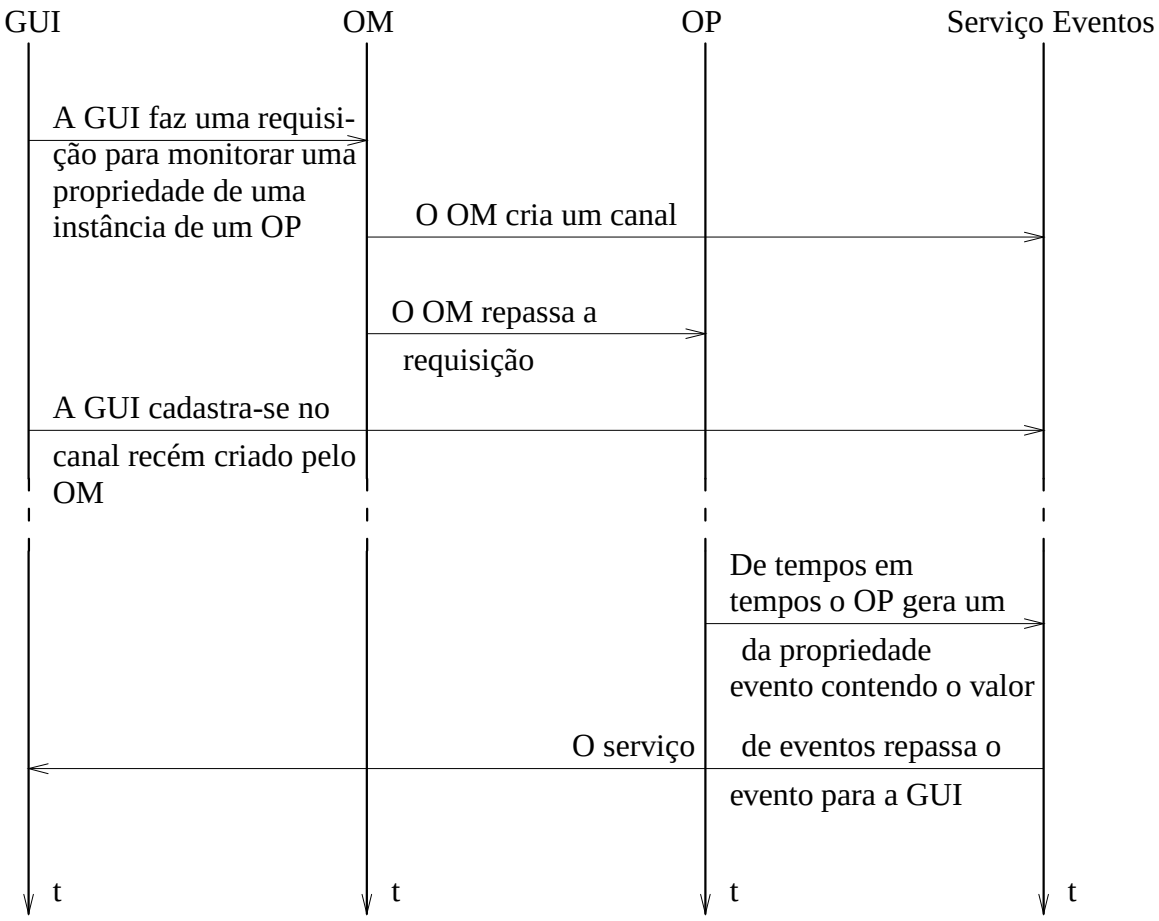

Figura 5.3: Diagrama de eventos para monitorar o valor de uma propriedade

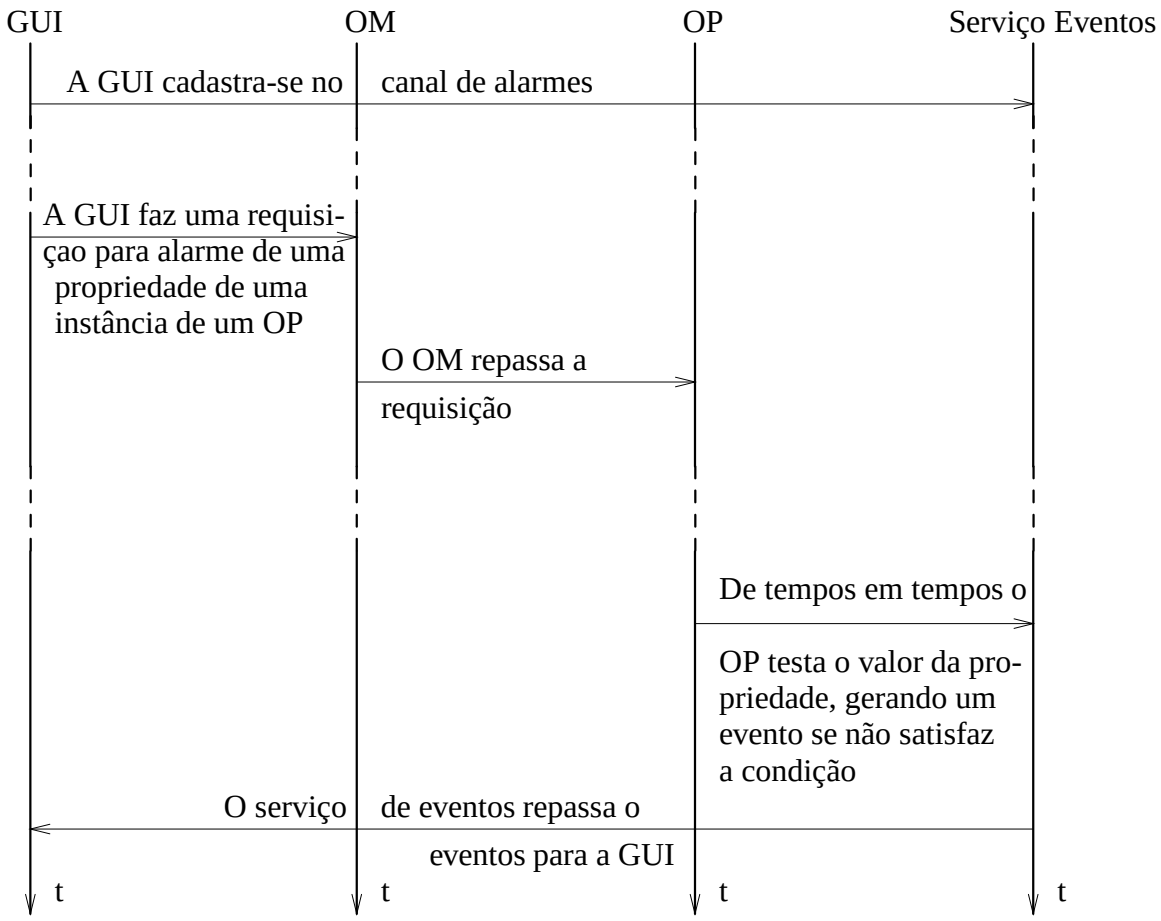

Figura 5.4: Diagrama de eventos para criar um alarme

5.3 Componentes do Sistema

Abaixo, serão mostrados mais detalhadamente os componentes da arquitetura: GUI, Object Manager, Object Provider e Objetos CIM.

5.3.1 GUI

Com o OM servindo como proxy, repassando as requisições para os nós do cluster, é possível executar a GUI de qualquer lugar, desde que tenha acesso ao nó mestre do cluster [FdRdR02]. A GUI também deve se cadastrar no serviço de eventos para poder "ouvir" os alarmes gerados pelos OPs.

5.3.2 Object Manager

O OM terá as seguintes funções:

- fazer o controle dos OPs existentes, permitindo uma visualização dos nós do cluster;
- agir como um proxy, repassando as requisições da GUI para os OPs [LSW99];
- fazer o controle das instâncias dos objetos CIM, provendo um *handle* para cada instância e identificando em qual nó está a instância.

Os métodos serão os mesmos do OP além de:

- **registerObjectProvider (nodeName: String, contexts: String[], agentObjRef: ObjectProviderServer)**: armazena o OP na lista de nós existentes no cluster, armazenando também os contextos a que pertence;
- **unregisterObjectProvider (nodeName: String)**: retira um OP da lista de nós;
- **getObjectProviderLocations (): String[]**: retorna o nome de rede de todos os OPs, onde o nome é separado por vírgula;

5.3.3 Object Provider

Os OPs, ao serem inicializados, irão registrar a sua existência (e a existência do nó no cluster). Então, aguardando as requisições dos OMs, repassa para o objeto CIM e retorna

a resposta do objeto. Recebe a requisição de monitoração de algum atributo, gerando um evento, de tempos em tempos, com o valor deste atributo no seu próprio canal de eventos. Também recebe a requisição de alarme, sendo que o OP fica monitorando o valor do atributo e, caso não satisfaça a condição desejada, gera um evento no respectivo canal de eventos de alarmes.

Os métodos do OP irão consistir basicamente em *get, set e invoke*, além dos métodos para monitoramento e alarme. Fazer com que haja apenas métodos simples, que possam ser utilizados em qualquer classe CIM, torna-os muito poderosos [BJMR00].

Métodos

Os métodos do OP são:

- **getNumberOfInstances (className : String) : int**: retorna o número de instâncias de uma classe CIM. Ex: caso o nó tenha dois HDs, o número de instâncias da classe CIM_HD é dois;
- **getInstance (className: String, instanceNumber : int) : Handle**: obtém o handle que está relacionado a um ponteiro de uma instância. Caso a classe java não tenha sido instanciada, ela é instanciada e relacionada a um handle, neste momento;
- **createInstance (className: String): Handle**: cria uma instância CIM e instancia um objeto java que representa esta instância;
- **deleteInstance (handle : Handle)**: deleta uma instância CIM. Ex: caso tenha sido retirado um HD, deve-se deletar a instância CIM associada ao HD;
- **getAttribute (handle : Handle, attrName : String) : CIMAttribute**: obtém o valor do atributo de uma instância CIM;
- **getAttribute (contextName : String, instanceNumber : int, attrName : String) : CIMAttribute[]**: obtém o valor do atributo de uma instância CIM de todos os nós da contexto;
- **setAttribute (handle : Handle, cimattribute : CIMAttribute)**: seta o valor de um atributo de uma instância CIM;
- **setAttribute (contextName : String, instanceNumber : int, cimattribute : CIMAttribute)**: seta o valor do atributo de uma instância CIM de todos os nós do contexto;

- **getBulk (handle : Handle) : String**: obtém o valor de todos os atributos de uma instância CIM, retornando em um formato XML;

- **invoke (handle : Handle, cimmethod : CIMMethod) : any**: invoca um método de uma instância CIM e retorna o mesmo valor retornado pelo método;

- **invoke (contextName : String, instanceNumber : int, cimmethod : CIMMethod) : any[]**: invoca um método de uma instância CIM e retorna o mesmo valor retornado pelo método, em todos os nós do contexto;

- **setMonitor (handle: Handle, property: String)**: informa o nome da propriedade de uma determinada instância CIM que deseja monitorar. A partir deste momento, o OP irá enviar um evento (através do serviço de eventos), de tempos em tempos, com o valor desta propriedade;

- **setMonitor (contextName : String, instanceNumber : int, property: String)**: informa o nome da propriedade de uma determinada instância CIM que deseja monitorar em todos os nós do contexto. A partir deste momento, o OP irá enviar um evento (através do serviço de eventos) de tempos em tempos com o valor desta propriedade;

- **unsetMonitor (handle: Handle, property: String)**: informa o nome da propriedade de uma determinada instância CIM que não deseja monitorar mais;

- **unsetMonitor (contextName : String, instanceNumber : int, property: String)**: informa o nome da propriedade de uma determinada instância CIM que não deseja monitorar mais, em todos os nós do contexto;

- **setAlarm (handle: Handle, cimeventAlarm: CIMEventAlarm)**: informa o nome da propriedade, a condição, e o tempo em milisegundos para testar a condição. Caso a condição não seja satisfeita, é gerado um evento no serviço de eventos através do canal de alarme;

- **setAlarm (contextName : String, instanceNumber : int, cimeventAlarm: CIMEventAlarm)**: informa o nome da propriedade, a condição, e o tempo em milisegundos para testar a condição em todos os nós do contexto. Caso a condição não seja satisfeita, é gerado um evento no serviço de eventos através do canal de alarme;

- **unsetAlarm (handle: Handle, property: String)**: informa o nome da propriedade de uma determinada instância CIM que não deseja monitorar mais;

- **unsetAlarm (contextName : String, instanceNumber : int, property: String)**: informa o nome da propriedade de uma determinada instância CIM que não deseja monitorar mais, em todos os nós do contexto;

5.3.4 Objetos CIM

Os objetos CIM serão objetos Java comuns, gerados a partir de um compilador MOF (descrição textual dos esquemas CIM) implementados no decorrer deste trabalho. Estes objetos terão as seguintes características:

- **Propriedades estáticas**: são propriedades que não mudam com o tempo, e serão armazenadas em um arquivo XML pelo objeto CIM. Exemplo: InstallDate, Caption, Description;
- **Propriedades dinâmicas**: são propriedades que precisam ser calculadas/processadas a todo o momento. Exemplo: DiskSpace, LoadPercentage;
- **Métodos**: métodos providos pela classe. Exemplo: shutdown, reset.

Métodos

Os métodos do objeto CIM são:

- **getNumberOfInstances (): int**: retorna o número de instâncias desta classe CIM;
- **getInstance (instanceNumber: int): ClassCIM**: retorna uma instância do objeto java que representa a instância CIM;
- **createInstance ()**: cria uma nova instância;
- **deleteInstance ()**: destroi a atual instância CIM;
- **getAttribute (cimattribute: CIMAttribute)**: obtém o valor da propriedade da classe representada pelo handle;
- **setAttribute (handle: Handle, cimattribute: CIMAttribute)**: seta o valor da propriedade;
- **invoke (handle: Handle, cimmethod : CIMMethod): any**: invoca o método e retorna o resultado do método;

5.4 Contexto

Um contexto serve para identificar um agrupamento de OPs que tenham alguma característica comum, onde um OP pode pertencer a diversos contextos. Neste caso, um contexto pode ser usado para identificar os nós com um determinado sistema operacional, processador, para dividir o cluster em nós utilizados para produção ou desenvolvimento.

Existem duas formas para adicionar um OP em um determinado contexto: alterar diretamente um arquivo XML que informa os contextos que o OP pertence, invocar o método addToContext, que adiciona o novo contexto no arquivo XML.

As operações do OM para obter/armazenar uma propriedade, invocar um método, criar/destruir um monitor e criar/destruir um alarme podem ser feitas num contexto, através de multicast. Esta parte não será implementada neste trabalho, sendo deixada como parte dos trabalhos futuros, mencionados no último capítulo.

Quando o OM recebe uma requisição para atuar em um determinado contexto, irá fazer um multicast no grupo relacionado ao contexto. Este multicast deve ser confiável, garantindo que todos os OPs do contexto receberão a requisição.

5.5 Extensibilidade

A utilização da MIB CIM torna possível estender os modelos previamente definidos pela DMTF, sendo possível utilizar um modelo criado especialmente para aplicações paralelas ou o PCIM (*Policy CIM*), que é uma extensão da IETF para políticas [IET02].

Para tanto, a ferramenta terá um compilador MOF, uma ferramenta implementada no desenvolvimento deste trabalho, que terá duas funções: criar uma representação XML das classes CIM e gerar as classes Java que representem as classes CIM.

A representação XML irá seguir as especificações da própria DMTF [DMT99a]. As classes Java geradas terão como padrão as propriedades estáticas, ou seja, todas as propriedades são consideradas estáticas e serão armazenadas em XML. Caso alguma das propriedades não seja estática, o código gerado deve ser alterado para o processamento desta propriedade. Isto também serve para os métodos.

5.6 Referência

Entre alguns dos tipos definidos pelo CIM está a referência de classes, de modo que uma propriedade ou parâmetro de um método possa ser uma referência a uma instância de uma classe em algum nó da rede.

Internamente, uma referência será feita através da classe CIMReference, e sua estrutura será vista mais adiante, na descrição das estruturas IDL.

Para resolver este problema na GUI, a referência a estas classes será feita através de forma textual da seguinte maneira: `nomederede.classeCIM.caption` ou `nomederede.classeCIM[numerodainstancia]`. Vale ressaltar que o `caption` de todas as instâncias em um mesmo nó devem ser diferentes uns dos outros, sendo parte da implementação de uma classe CIM, que ao receber uma requisição para armazenar o valor de uma propriedade `caption` deve testar para ver se não é conflitante com o `caption` de outras instâncias.

Então, no momento em que o usuário fizer alguma referência ao invocar um método ou armazenar o valor de uma propriedade, será utilizada a sintaxe acima descrita.

Internamente, a referência por ser um *handle* - que identifica o nó, a classe CIM e a instância, como será feita pela GUI.

5.7 Estruturas

As estruturas existentes são:

```
1
2  module pucpr {
3          module macorb {
4                  typedef sequence <wstring> strings;
5
6                  typedef sequence <wstring> UnboundedStrings;
7
8                  exception CIMException
9                  {
10                     string message;
11                 };
12
13                 exception CIMAgentNotAlive
14                 {
15                     string message;
16                 };
17
18                 struct CIMAttribute {
19                         wstring className;
```

```
20                      wstring attrName;
21                      long instanceNumber;
22                      any value;
23              };
24
25              struct CIMEventAlarm {
26                      CIMAttribute cimattr;
27                      wstring operator;
28                      any operand;
29                      boolean operandAbsolute;
30                      boolean shouldDesactivate;
31                      any limitToReativate;
32                      long interval;
33                      wstring message;
34              };
35
36              struct CIMParameter {
37                      wstring name;
38                      wstring value;
39              };
40
41              struct CIMMethod {
42                      wstring className;
43                      wstring methodName;
44                      any ret;
45                      sequence <CIMParameter> cimpar;
46              };
47
48              typedef sequence <CIMAttribute> UnboundedCIMAttribute;
49      };
50  };
```

5.8 Monitoração

Quando a GUI faz uma requisição de monitoramento de alguma propriedade, o OP lança uma thread, que, periodicamente, obtém o valor atual desta propriedade no objeto CIM e gera um evento utilizando o canal `nomederede_propriedadecaption_nomedapropriedade`. Assim, caso a GUI peça para monitorar a propriedade `LoadPercentage` do *processador 0* no nó `beowulf3`, será criado um canal no serviço de eventos com o nome `beowulf3_processor0_LoadPercentage`. O dado repassado ao gerar o evento é a estrutura `CIMEventAlarm` [OMG95b].

5.9 Alarme

"Filtros são usados para determinar em quais eventos em uma rede deve-se agir ou relatar. Filtros são vitais em esquemas de gerência de redes. Eles previnem a rede de se tornar sobrecarregada com relatórios não essenciais sobre eventos triviais, e, ainda de mais importância, permitem o relatório de informações críticas"[Bla95].

Quando a GUI faz uma requisição de alarme de alguma propriedade, o OP lança uma thread que, de tempos em tempos, obtém o valor atual desta propriedade no objeto CIM, testa se o valor satisfaz a condição informada e, caso não satisfaça, gera um evento utilizando o canal `alarmchannel_macorb`. Assim, a GUI deve se cadastrar neste canal no serviço de eventos para poder receber os alarmes gerados pelo OP. O dado repassado ao gerar o evento é a estrutura `CIMEventAlarm` [OMG95b].

As condições possíveis variam de acordo com o tipo da propriedade. Para propriedades do tipo string, boolean, char, referência ou valuemap as condições são “igual e diferente”. Para propriedades de tipo numérico ou datetime, são “menor, menor igual, igual, diferente, maior igual e maior”. Vale ressaltar que *valuemap* não é um tipo definido na especificação CIM, mas quando a propriedade tem um qualificador ValueMap - o qual mapeia todas as opções possíveis daquela propriedade [UPAM00].

As condições podem ser testadas sobre um valor absoluto ou variação (delta). A variação é calculada da seguinte forma: o valor atual da propriedade - o valor da última leitura. A variação só pode ser calculada sobre propriedades de tipo numérico.

Quando a propriedade for de tipo numérico ou datetime, também é possível informar o valor de reativação do alarme. Ou seja, quando um evento do alarme for gerado, o alarme será desativado até que o valor da propriedade tenha atingido um valor intermediário.

Supondo que a condição de um alarme seja que o número de pacotes enviados em um minuto seja menor que 100: quando o número de pacotes for maior ou igual a 100, um evento do alarme será gerado e o alarme não será gerado até que este número de pacotes fique abaixo de 80 (valor também informado pelo usuário).

Caso a propriedade não seja do tipo numérico ou datetime, o alarme só será reativado quando o valor satisfizer novamente a condição informada.

Ao receber a requisição para criar um novo alarme, ou requisição para destruir um existente, o OP deve salvar as informações de todos os alarmes existentes em um arquivo XML, assim, no momento que for inicializado, o OP deve ler este arquivo para poder recriar os alarmes criados antes.

5.10 Classes CIM

Neste momento, apenas alguns objetos CIM são implementados, valendo ressaltar que a ferramenta é extensível e a qualquer momento é possível adicionar novas classes CIM, através da compilação de arquivos MOF [FdRdR02] [UPAM00]. A implementação atual, quando estiver rodando em Linux, irá obter o valor das propriedades através de alguns arquivos localizados diretório `/proc` ou através de alguns aplicativos (top e ifconfig) que também se utilizam destes arquivos. O diretório `/proc` é um *file system* virtual, onde é possível obter informações do sistema operacional sobre processos, memória disponível, e assim por diante. As classes escolhidas para implementação são:

- **CIM_System**:
  - **CIM_OperatingSystem**: através desta classe será possível visualizar os processos existentes (NumberOfProcess), memória total (TotalVirtualMemorySize), memória livre(FreePhysicalMemory, FreeVirtualMemory), reinicializar (Reboot()) e shutdown (Shutdown()). Para obter a memória total e virtual será analisado o arquivo /proc/meminfo, para visualizar os processos existentes será analisada a saída da aplicação top;
  - **CIM_Process**: cada classe de CIM_Process representa uma única instância de um programa rodando. Através desta classe será possível visualizar os processos rodando na máquina e seus detalhes. A obtenção dos dados dos processos será feita através da saída da aplicação top;
- **CIM_Device**:
  - **CIM_Processor**: através desta classe será possível monitorar a carga dos processadores (LoadPercentage). Para obter os dados do processador será analisado o arquivo /proc/cpuinfo, e para obter a carga será analisado a saída da aplicação top;
  - **CIM_EthernetAdapter**: através desta classe será possível monitorar/gerenciar os IPs (NetworkAddress), número de pacotes transmitidos (TotalPacketTransmitted) e recebidos (TotalPacketReceived). Para obter os dados de rede será analisada a saída da aplicação ifconfig.
- **CIM_Application**: através das classes deste modelo será possível gerenciar os softwares instalados em cada nó:
  - **CIM_ExecuteProgram**;

- **CIM_ModifySettingAction**;
- **CIM_CreateDirectoryAction**;
- **CIM_RemoveDirectoryAction**;
- **CIM_CopyFileAction**;
- **CIM_RemoveFileAction**;
- **CIM_Action**: o objetivo desta classe é executar comandos do Shell, permitindo instalar/desinstalar software nos nós. Esta instalação só será possível através de instaladores "silenciosos", que não tenham interação com o usuário.

5.10.1 Classe Cluster

Aqui, será proposta uma nova classe CIM, chamada Cluster, onde haverá algumas informações estatísticas, fazendo com que seja possível "controlar" o cluster como um recurso único. Os métodos estatísticos desta classe devem ser consultados sempre no nó mestre do cluster, pois se forem consultados em outro nó, que não seja o mestre, não irão retornar valores estatísticos, mas apenas valores do próprio nó. A figura 5.5 mostra como funcionam os métodos estatísticos desta classe.

Este processo é similar à monitoração, mas com uma diferença, a classe Cluster é responsável por obter os dados de todos os outros nós e sumarizar, para então repassar para a GUI. Os nós que não fizerem parte do mesmo contexto não irão gerar eventos, ignorando a invocação do método. Por isto, a classe Cluster não deve esperar a chegada de um evento de todos os nós, utilizando, para efeito de cálculo, os valores disponibilizados pelos outros agentes.

Para iniciar a monitoração, devem ser invocados os métodos extrínsecos da classe, ao invés de invocar métodos intrínsecos (como `setMonitor`), como usado na monitoração. A principal razão é devido à monitoração de processos. A monitoração de um processo será feita pelo seu nome, pois cada nó pode ter vários processos similares rodando, cada um, com seu próprio identificador. Assim, caso o usuário deseje monitorar a quantidade de memória usada por uma aplicação no cluster, deverá monitorar todos os processos similares, ou seja, processos com o mesmo nome.

As propriedades da classe Cluster são:

- **ClusterName**: string com o nome do cluster, podendo ser Beowulf, ParaPUC, etc;
- **Context**: é um vetor de strings com os contextos a que o nó pertence, podendo ser Cluster para todos os nós, produção para os nós do subcluster de produção, Linux para

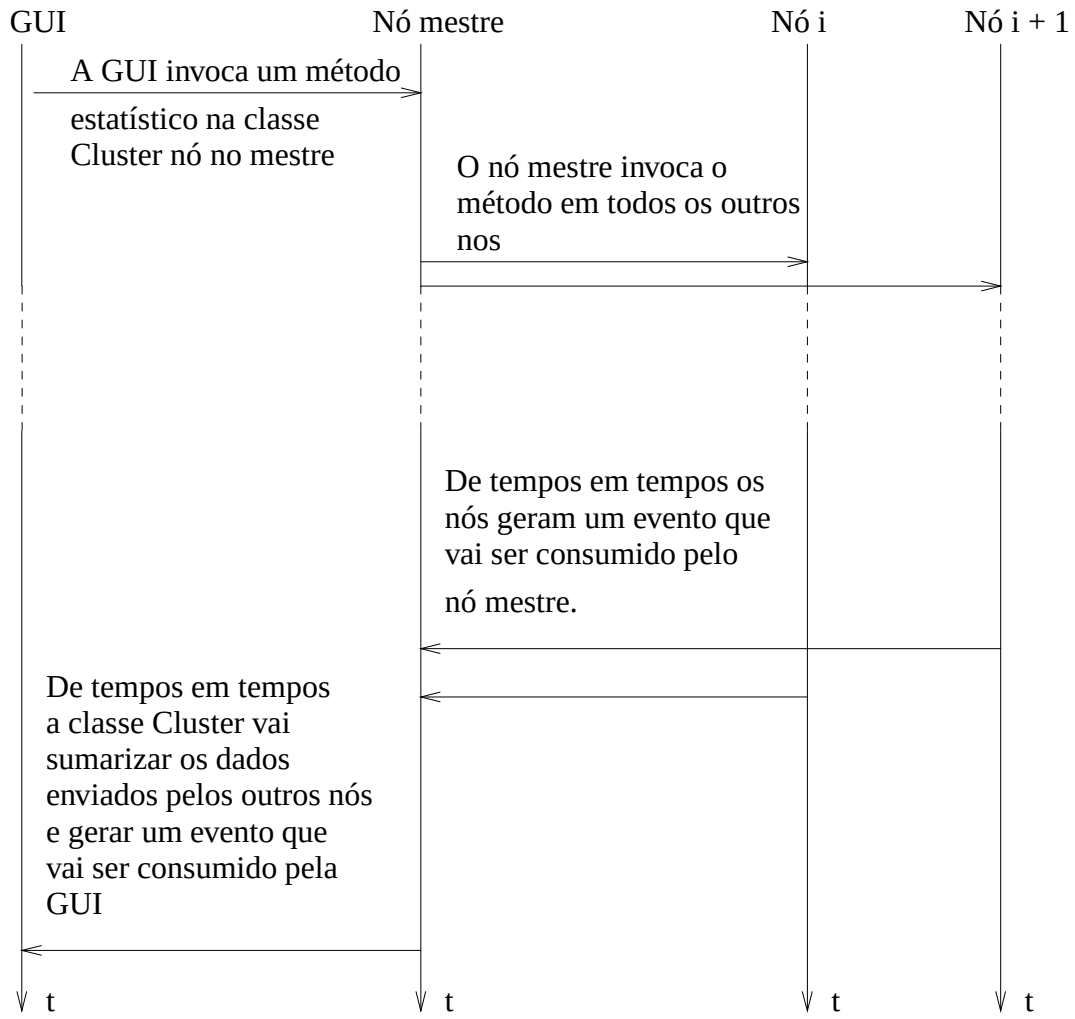

Figura 5.5: Diagrama para classe Cluster

os nós que utilizem Linux e assim por diante;

- **IsMaster**: é um boleano que indica se o nó é mestre ou não. Caso a arquitetura do cluster não tenha apenas um nó mestre, deve ser considerado nó mestre aquele no qual o OM está rodando;

- **LoadPercentage**: é um método que inicia a monitoração da média do percentual da utilização de determinado processador de todos os nós do contexto;

- **ProcessMemory**: inicia a monitoração da somatória da memória utilizada por processos similares em todos os nós do contexto;

- **ProcessLoadPercentage**: inicia a monitoração da média de CPU utilizada por processos similares em todos os nós do contexto;

- **NumberOfProcess**: é um método que inicia a monitoração do número de processos similares rodando em todos os nós do contexto.

O MOF da classe Cluster:

```
// ==================================================================
// Version:     1.0.0
// Date:        14/02/2003
// ==================================================================

// ==================================================================
// Pragmas
// ==================================================================
#pragma locale ("en_US")

// ==================================================================
//    Cluster
// ==================================================================
   [Version ("1.0.0"), Description (
     "Cluster é uma classe que reúne informações do cluster, "
     "sendo alguma destas informações estatísticas, fazendo com "
     "que o cluster possa ser gerenciado como um único recurso.") ]
class Cluster : CIM_ManagedElement {
      [MaxLen (64), Description (
        "Nome do cluster que o nó pertence.") ]
   string ClusterName;
       [MaxLen (64), Description (
        "Contexto a que o nó pertence. Contexto pode ser Cluster para "
        "todos os nós, Produção para os nós do subcluster de produção, "
        "Linux para os nós que utilizem Linux e assim por diante.") ]
   string Context[];
      [Description (
        "Indica que esta classe está presente no nó mestre do cluster.") ]
   boolean IsMaster;
       [Description (
        "Obtém a média do percentual da utilização de uma determinada "
        "CPU de todos os nós do contexto.") ]
   uint32 LoadPercentage([IN] string context, [IN] uint32 instanceNumber);
      [Description (
        "Obtém a somatória da memória utilizada por processos similares "
        "em todos os nós do contexto.") ]
   uint32 ProcessMemory([IN] string context, [IN] string processName);
      [Description (
        "Obtém a média do percentual da utilização de CPU de processos "
        "similares em todos os nós do contexto.") ]
   uint32 ProcessLoadPercentage([IN] string context,
                [IN] string processName);
      [Description (
        "Obtém o número de processos similares rodando em todos os "
        "nós do contexto.") ]
   uint32 NumberOfProcess([IN] string context, [IN] string processName);
};
```

5.10.2 Adicionando classes CIM

Para adicionar novas classes CIM, é necessário seguir os passos:

1. O arquivo MOF com a definição das classes que serão adicionadas deve ser concatenado ao arquivo MOF das classes existentes ou adiciona-se uma diretiva de compilação para incluir este arquivo;
2. Execute o compilador MOF para processar o novo arquivo MOF;
3. Renomeie o arquivo out.xml para CIM_Macorb.xml;
4. Copie ou mova este arquivo para o diretório raiz, onde se encontra a ferramenta;

Para acrescentar alguma propriedade dinâmica ou algum método abra o fonte da classe `pucpr.macorb.agent.Trap`, alterando os métodos `Trap` (método responsável pela inicialização da nova classe) e `isAttributeTraped` (método que indica que o atributo da classe é dinâmico).

Depois de alterada a classe `Trap`, crie uma nova classe com base no modelo `CIM_Processor`, onde o nome desta deve ser idêntico ao nome da classe definida no arquivo MOF. Recompile o código fonte e distribua as classes geradas entre os nós do cluster.

5.11 Áreas de gerência

A seguir, uma análise da ferramenta quanto às cinco áreas de gerência de sistemas:

- **Falhas**: a gerência de falhas é feita através da configuração de alarmes;
- **Contabilização**: as classes previstas no padrão CIM relacionadas à gerência de contabilização não foram implementadas neste momento;
- **Configuração**: a gerência de configuração é feita através da obtenção/armazenamento de propriedades, invocação de métodos e inventários dos nós do cluster existentes;
- **Desempenho**: a monitoração do desempenho é feita através da monitoração de algumas propriedades nos nós;
- **Segurança**: a segurança não está contemplada nesta proposta;

5.12 Exemplos

São mostradas, abaixo, algumas telas da implementação deste sistema, iniciando pela tela principal da GUI (figura 5.6). Também são mostrados os tipos de condição (<, <=, =, !=, >= e >) para o tipo de propriedade (inteiro) que são permitidos para criar um alarme de acordo com o ciclo de histerese. Caso o usuário tenha criado um alarme com a condição <= 10, a tela mostrada para o usuário, quando a carga não for menor igual que 10, será igual à da figura 5.7.

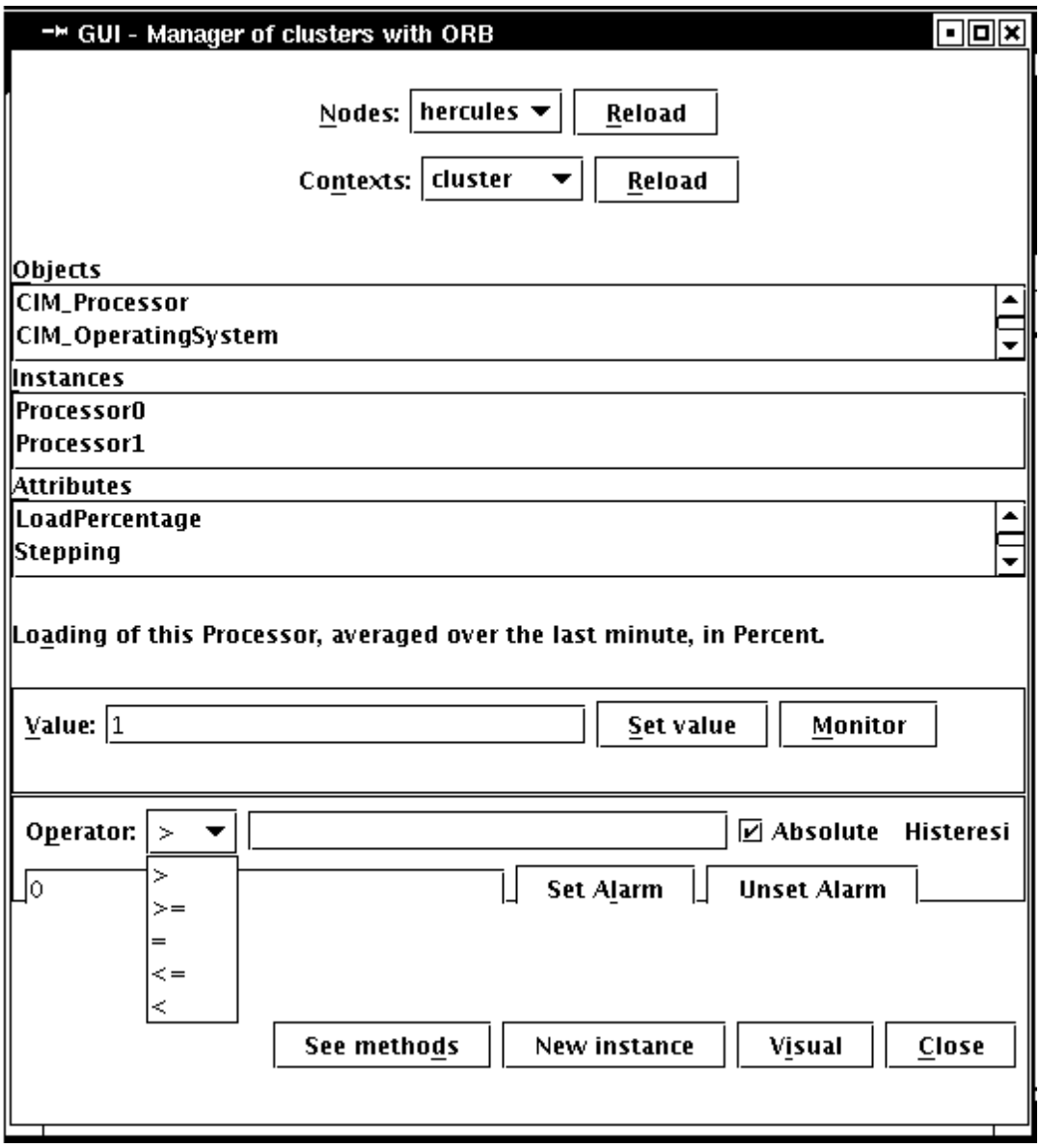

Figura 5.6: Tela principal da GUI

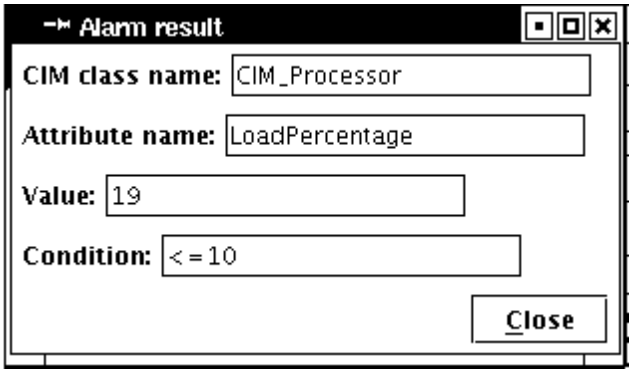

Figura 5.7: Tela quando um alarme é gerado na GUI

As telas da figura 5.8 mostram o usuário monitorando a carga da CPU. Na primeira tela consta a carga do segundo processador - somente `hercules` contém dois processadores, portanto o gráfico para `maq03` está vazio. Na segunda tela consta a carga do primeiro processador.

5.13 Resultados obtidos

Devido a alguns problemas não foi possível testar a aplicação em um cluster com uma aplicação paralela rodando, de modo a avaliar se a ferramenta atingiu, de fato, os objetivos propostos: ser escalável e de baixo impacto.

Apesar deste problema, pode-se afirmar que a ferramenta atingiu o objetivo de ser interoperável, considerando que os testes foram feitos com um agente rodando em Linux e o outro agente rodando em um Windows 98 [1]. A ferramenta também atingiu o objetivo de ser expansível, tendo em vista que, a partir de um arquivo MOF - representando em forma textual a estrutura de uma classe CIM, conjuntamente com o compilador de arquivos MOF, é possível acrescentar uma classe CIM com propriedades estáticas e sem métodos. Logicamente, caso o usuário queira acrescentar uma classe CIM, com propriedades dinâmicas ou métodos, haverá um certo esforço de codificação.

[1] com o valor de algumas propriedades não significativo

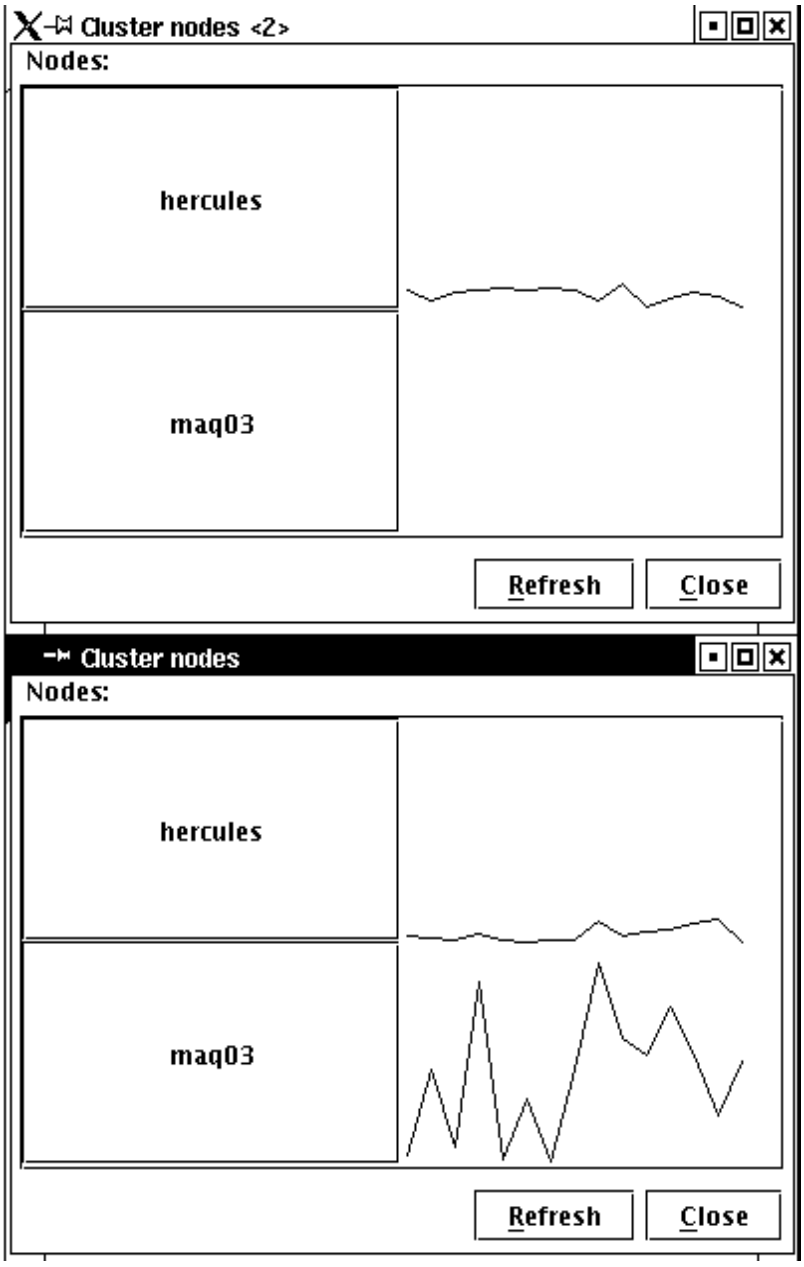

Figura 5.8: Tela de monitoração das CPUs na GUI

5.14 Conclusão

A proposta de uma ferramenta para gerência de clusters abertos, baseada em CIM, CORBA e Java, a princípio, possui tudo para auxiliar na gerência de um cluster de forma satisfatória e em todos os níveis desejados, independente do seu tamanho.

Capítulo 6

Conclusão e perspectivas

O desenvolvimento da tecnologia de Clusters de PCs Beowulf foi a solução para muitos problemas existentes na época que havia apenas arquiteturas proprietárias para processamento paralelo. Entre eles estão as padronizações na programação, não havendo mais a necessidade de ficar reescrevendo programas para as novas tecnologias, e a obtenção de conhecimento em processamento paralelo agora é considerado um investimento. Outra vantagem também é o preço, pois agora estes "supercomputadores" são muito mais acessíveis.

A gerência de redes também já teve muitos problemas, até o surgimento de padrões para a gerência destas redes. O primeiro destes padrões foi o SNMP, que hoje é o mais difundido. Mas este padrão ainda tem muitas deficiências, estimulando assim o surgimento de novos padrões. Dentre estes padrões o mais interessante é o CIM, pois tem como objetivo modelar a informação, seja heterogênea ou não, e esta modelagem segue uma orientação a objetos.

Como a tecnologia de clusters abertos ainda tem um vazio - sua gerência de forma satisfatória, a criação de uma ferramenta para gerência de clusters que utilizem um padrão para a modelagem da informação e CORBA para comunicação entre gerente e agente foi uma união muito feliz. Com a utilização de CORBA, foi possível o desenvolvimento de uma ferramenta que satisfizesse os requisitos necessários para uma ferramenta de administração de clusters abertos.

A contribuição efetiva deste trabalho está na criação de uma ferramenta para gerência de clusters abertos, onde grande parte das ferramentas atuais limita-se a apenas monitoração. Outra característica, também não presente em grande parte das ferramentas atuais, é a presença de uma ferramenta que seja expansível, sendo possível ao usuário acrescentar

alguma(s) classe(s) para a gerência de sua própria aplicação.

Alguns tópicos podem ser considerados para a continuidade deste trabalho:

- **Módulo de agendamento**: um módulo que saiba a carga de cpu de todos os nós, memória livre e o grau de completude das tarefas nos respectivos nós, e através de um módulo desenvolvido para conversar com a aplicação em um determinado protocolo (exemplo: PVM/MPI [Ge94]) agendar em quais nós devem ser executadas determinadas tarefas [LSW99] [Fer99]. Com um módulo como este fica mais fácil detectar um eventual *deadlock* entre processos. Com isto também é possível determinar o grau de utilização dos nós [Fer99] [Ge97];

- **Otimização dos OPs**: otimizar os OPs, utilizando talvez linguagem e/ou ORB mais leves para que possam rodar em micros menos potentes, tornando assim mais viável sua utilização em um *grid* computacional [FdRdR02];

- **Ponte SNMP**: fazer uma ponte SNMP, sendo assim possível obter dados disponíveis somente através deste protocolo [GXF95] [AMR99];

- **PCIM**: acrescentar à ferramenta as classes relativas ao PCIM [IET02];

- **ORB com multicast**: adaptar a ferramenta para utilizar um ORB que suporte multicast para ser usado no momento que for obter/armazenar valores de propriedades ou invocar métodos de um contexto de nós do cluster [BdSFL02] [UPAM00];

- **Segurança**: desenvolver um módulo de segurança para a ferramenta.

Referências Bibliográficas

[AMR99] Gerd Aschemann, Thomas Mohr, and Mechthild Ruppert. Integration of SNMP into a CORBA and Web-based management enviroment. In *KiVS'99*, pages 339–346, Março 1999.

[Bac01] Dan Backman. Basking in Glory-SNMPv3. *NetComputing*, June 2001.

[Ban97] Bela Ban. A generic management model for CORBA, CMIP and SNMP. Master's thesis, Universität Zürich, Zurique, Dezembro 1997.

[BdSFL02] Alysson Neves Bessani, Joni da Silva Fraga, and Lau Cheuk Lung. Mjaco - integração do multicast IP na arquitetura CORBA. In *XX Simpósio Brasileiro de Redes de Computadores, SBRC'2002. SBC*, pages 65–80, Maio 2002.

[BJMR00] D. Bénech, F. Jocteur-Monrozier, and A. Rivière. Supervision of the CORBA enviroment with SUMO: a WBEM/CIM based management framework. In *International Symposium on Distributed Objects and Applications, 2000*, pages 241–250, November 2000.

[Bla95] Uyless Black. *Network management standards : SNMP, CMIP, TMN, MIBs, and object libraries*. McGraw-Hill, 1995.

[BMG98] Rajkumar Buyya, Krishna Mohan, and Bindu Gopal. Parmon: A comprehensive cluster monitoring system. In *Proceedings of the International Conference on High Performance Computing on Hewlett-Packard Systems (HiPer'98)*, Junho 1998.

[DMT99a] DMTF. Representation of *CIM* in *XML*. `http://www.dmtf.org/standards/documents/WBEM/DSP201.html`, Julho 1999.

[DMT99b] DMTF. Web-based enterprise management (*WBEM*) initiative. `http://www.dmtf.org/standards/standard\_wbem.php`, Julho 1999.

[DMT99c] DMTF - Desktop Management Task Force. *Common Information Model (CIM) specification*, June 1999.

[DMT01] DMTF - Desktop Management Task Force. *CIM Tutorial*, June 2001.

[FdRdR02] Tiago C. Ferreto, César A. F. de Rose, and Luiz de Rose. Rvision: An open and high configurable tool for cluster monitoring. In *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium*, pages 75–82, November 2002.

[Fel98] Pascal Felber. *Inter-Domain Management Specifications: Preliminary CORBA/CMISE Interaction Translation Architecture*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland, June 1998.

[Fer99] Fritz Ferstl. Global resource director (GRD). In *1st IEEE Computer Society International Workshop on Cluster Computing*, pages 339–346, November 1999.

[Ge94] Al Geist and et.al. *PVM, Parallel Virtual Machine - a users's guide and tutorial for network*. MIT Press, 1994.

[Ge97] James E. Gursha and et.al. *High Performance Cluster Configuration System Management*. Digital Press, 1997.

[GXF95] Joint Inter-Domain Working Group, X/Open, and Network Management Forum. *Inter-Domain Management Specifications: Preliminary CORBA/CMISE Interaction Translation Architecture*. Open Group, 1995.

[IET02] IETF. Policy core information model extensions. `http://www.ietf.org/internet-drafts/draft-ietf-policy-pcim-ext-08.txt`, Maio 2002.

[IT89] ITU-T. *ITU-T Recommendation X.700 | ISO/IEC 7498-4: 1989, Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework*, page Part 4: Management Framework. ITU-T, 1989.

[Lew95] Lundy Lewis. *Managing computer networks: a case-based reasoning aproach*. Artech House, 1995.

[LPR97] Mika Leppinen, Pekka Pulkkinen, and Aapo Rautiainen. Java and CORBA-based network management. *IEEE Computer*, 30(1):83–87, January 1997.

[LSW99] Zhengyu Liang, Yundong Sun, and Cho-Li Wang. Clusterprobe: An open, flexible and scalable cluster monitoring tool. In *1st IEEE Computer Society International Workshop on Cluster Computing*, pages 261–268, November 1999.

[Mic99] Sun Microsystems. Java management extensions white paper. Technical report, Palo Alto - CA, Junho 1999.

[OMG95a] OMG. *The Common Object Request Broker: Architecture and Specification*, June 1995.

[OMG95b] OMG. *CORBAservices: Common Object Services Specification*, June 1995.

[OMG98] OMG. *Object Management Architecture Guide, Second Edition*, Setembro 1998. OMG TC Document 92-11-1.

[Ros96] Marshall T. Rose. *The simple book: an introduction to networking management*. Prentice Hall, 1996.

[SBF02] Céline Boutros Saab, Xavier Bonnaire, and Bertil Folliot. A flexible monitoring platform to build cluster management services. In *IEEE Cluster2000 Conference: Technology and Applications*, pages 75–85, Janeiro 2002.

[Sta93] William Stallings. *SNMP, SNMPv2 and CMIP*. Addison Wesley, 1993.

[UPAM00] Puchong Uthayopas, Surachai Paisitbenchapol, Thara Angskun, and Jullawadee Maneesilp. System management framework and tools for beowulf cluster. In *The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, pages 935–940 vol. 2, November 2000.