Instituto de Pesquisas Tecnológicas do Estado de São Paulo

MARIO MARQUES

Utilização de Padrões HTTP/XML na Monitoração do Desempenho de um *Cluster* de Servidores.

São Paulo

2005

Mario Marques

Utilização de padrões HTTP/XML na monitoração do desempenho de um *cluster* de servidores

MARIO MARQUES

Utilização de Padrões HTTP/XML na Monitoração do Desempenho de um *Cluster* de Servidores.

Dissertação apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, para obtenção do título de Mestre em Engenharia de Computação.

Área de concentração: Redes de Computadores

Orientador: Dr. Maurício Caldora Costa

São Paulo

Outubro de 2005

M357u Marques, Mario
Utilização de padrões HTTP/XML na monitoração do desempenho de um cluster de servidores. / Mario Marques. São Paulo, 2005.
103p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Maurício Caldora Costa

1. SNMP - Simple Network Management Protocol 2. Monitoração 3. Desempenho 4. Servidor de gerenciamento 5. Gerenciamento de redes de computadores 7. Processamento paralelo 8. Método dos elementos finitos 9. XML - Extended Markup Language 10. HTTP - Hyper Text Transfer Protocol 11. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Centro de Aperfeiçoamento Tecnológico II.Título

05/71 CDU 004.72.057.4(043)

RESUMO

A arquitetura padrão especificada pelo *Internet Engineering Task Force* (IETF) para gerenciamento de redes e de servidores e amplamente utilizada em todo o mundo é a arquitetura baseada no protocolo *Simple Network Management Protocol* (SNMP) e em suas especificações correlacionadas : *Management Information Base* (MIB), *Structure of Management Information* (SMI) e *Basic Encoding Rules* (BER).

Devido a limitações existentes na arquitetura de gerenciamento de redes e de servidores SNMP, vários estudos estão sendo realizados para a definição de uma nova arquitetura de gerenciamento baseada em padrões WEB. Nesse novo contexto, estão sendo propostas arquiteturas que geralmente se baseiam no padrão *Hyper Text Transfer Protocol* (HTTP) como protocolo de transferência de dados e *Extended Markup Language* (XML) como sintaxe de transferência, além da continuidade de utilização da MIB e da SMI para especificação de dados.

Os padrões HTTP e XML foram inicialmente identificados como alternativas adequadas para a definição de uma nova arquitetura de gerenciamento de redes e de servidores devido, principalmente, à sua ampla utilização, facilidade de implementação e também por ser um padrão totalmente aberto.

Apresenta-se neste trabalho, uma exposição sobre a arquitetura SNMP e uma comparação entre as propostas de arquitetura de gerenciamento baseadas em padrões *WEB* que estão sendo estudadas. Além disso, é feito o desenvolvimento, a implementação e a avaliação de uma aplicação de monitoração de desempenho de um cluster de servidores com processamento paralelo, dedicado ao estudo de dispositivos eletromagnéticos avaliados pelo Método dos Elementos Finitos (MEF), por meio da utilização de agentes e clientes que utilizam a tecnologia XML para armazenamento de dados e sintaxe de transferência e o protocolo HTTP para transmissão de dados para o servidor de gerenciamento.

Palavras-chave: Gerenciamento de Redes e Servidores; Gerenciamento de Desempenho; Processamento Paralelo; SNMP-XML; Monitoração.

ABSTRACT

The standard architecture specified by IETF for networks and server management, widely used in the whole world is based on the SNMP protocol and correlated specifications: MIB, SMI and BER.

Due to limitations in the networks and servers management SNMP architecture, several studies are being conducted toward the definition of a new management architecture based on WEB standards. In this new context, architectures based on HTTP standard as a data transfer protocol and XML as a transfer syntax, are being proposed, as well as of the continuity of the MIB and SMI for data specification.

The HTTP and XML standards were initially identified as adequate alternatives for the definition of a new network and server management architecture due, mainly, to their wide utilization, ease of implementation and as open standards.

This work presents a survey about the SNMP architecture and a comparison between Web-based management architectures that are been studied. A development, a deployment and an evaluation of a application performance monitoring cluster of servers with parallel processing, dedicated to the analysis of electromagnetic devices evaluated by the Finite Elements Method, using agents and clients that use the XML technology for data storing and transfer syntax and HTTP protocol for data transmission to the management server.

Keywords: Servers and Networks Management; Performance Management; Parallel Processing; SNMP-XML; Monitoring.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Program Interface*
ASN.1	*Abstract Syntax Notation One*
BER	*Basic Encoding Rules*
CCITT	*Commite Consultatif International de Telegraphique et Telephonique*
CGI	*Common Gateway Interface*
CIM	*Common Information Model*
CMIP	*Common Management Information Protocol*
CMOT	*Common Management Information Protocol over TCP/IP*
CORBA	*Common Object Request Broker Architecture*
DMTF	*Desktop Management Task Force*
DOM	*Document Object Model*
DTD	*Document Type Definition*
EPUSP	Escola Politécnica da USP
FIPS	*Federal Information Processing Standards*
FTP	*File Transfer Protocol*
HTTP	*Hyper Text Transfer Protocol*
HTML	*Hyper Text Markup Language*
IETF	*Internet Engineering Task Force*
IIS	*Internet Information Server*
IIOP	*Internet Inter-ORB Protocol*
IPT	Instituto de Pesquisas Tecnológicas do Estado de São Paulo
ITU-T	*International Telecommunications Union*
JAR	*Java Archive*

JAMAP	*Java Management Plataform*
JDBC	*Java Database Connectivity*
JDOM	*Java Document Object Model*
JRMP	*Java Remote Method Protocol*
JMS	*Java Messaging Server*
JVM	*Java Virtual Machine*
MEF	Método dos Elementos Finitos
MIB	*Management Information Base*
MIME	*Multipurpose Internet Mail Extensions*
MPI	*Message Passing Interface*
NMS	*Network Management System*
OID	*Object Identifier*
OMG	*Object Management Group*
OSI	*Open Systems Interconnect*
PEA	Departamento de Engenharia de Energia e Automação Elétricas
PDU	*Protocol Data Unit*
RAM	*Randomic Access Memory*
RMI	*Remote Method Invocation*
RFC	*Request for Comments*
SGMP	*Simple Gateway Management Protocol*
SMI	*Structure of Management Information*
SMTP	*Simples Mail Transfer Protocol*
SNMP	*Simple Network Management Protocol*

SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
USP	Universidade de São Paulo
W3C	*World Wide Web Consortium*
WBEM	*Web-based Enterprise Management*
WIMA	*Web-based Integrated Management Architecture*
XML	*Extended Markup Language*
XNAMI	*XML-based Architecture for SNMP Management of Network and Application*
XNM	*XML-based Network Management*
XNMP	*XML Network Management Pattern*

Sumário

1 INTRODUÇÃO

1.1 Motivação

A monitoração de redes e de servidores em redes TCP/IP é atualmente especificada e implementada por meio da arquitetura *Simple Network Management Protocol* (SNMP), conforme Case et alii (1990). A arquitetura SNMP é uma arquitetura de monitoração hierárquica e centralizada, na qual geralmente um servidor central de gerenciamento efetua o controle de todos os equipamentos de rede e de servidores vinculados à sua comunidade. Na arquitetura SNMP, a maior parte do processamento é feita pelo servidor central de gerenciamento, também conhecido como *Network Management System* (NMS), que efetua coleta de dados dos elementos gerenciados em intervalos de tempo pré-determinados através do protocolo SNMP com o auxílio de um agente de *software* SNMP que fica instalado em cada elemento gerenciado, conforme Stallings (1999).

No modelo de gerenciamento SNMP, o agente SNMP envia informações de monitoração apenas quando o NMS solicita ou, em casos especiais, quando ocorre algum problema no elemento gerenciado. Dessa forma, na arquitetura SNMP não existe um processo nativo de envio de informações de monitoração, sem que haja uma mensagem prévia de solicitação, conforme Case et alii (1990). Além desse problema, Flatin (2002) identifica outras dificuldades no uso da arquitetura SNMP:

- A escalabilidade da arquitetura SNMP é limitada;
- O mecanismo de codificação de dados do SNMP é ineficiente;
- O mecanismo de recuperação de dados armazenados em tabelas é deficiente;
- Tamanho máximo de mensagem de 484 bytes é considerado pequeno para transmissão de grandes quantidades de dados;
- Os dados não podem ser transmitidos de forma comprimida;
- Aspectos de segurança têm um tratamento muito simples nas versões 1 e 2 do SNMP e contém alguma melhora na versão 3, mas que ainda é insuficiente;
- O protocolo de transporte utilizado não é confiável;
- Dados importantes têm que ser a solicitação de retransmissão tratada pela aplicação;
- A arquitetura SNMP não prevê o gerenciamento hierárquico.

Pelos motivos expostos e devido à necessidade de desenvolver soluções que possam ser adequadas para prover uma solução de monitoração ativa de *cluster* de servidores e baseada em padrões HTTP/XML, foi especificada, desenvolvida e testada, neste trabalho, uma aplicação de monitoração. A aplicação foi desenvolvida de forma genérica, para permitir a sua utilização para monitoração de desempenho de qualquer ambiente em que exista um cluster de servidores ou até mesmo para servidores isolados. Para a efetiva validação da

aplicação de monitoração, utilizou-se um *cluster* de servidores dedicado à análise de dispositivos eletromagnéticos avaliados pelo Método dos Elementos Finitos (MEF), conforme Alves Filho (2000), com a finalidade de verificar a viabilidade da proposta de monitoração e as dificuldades e as limitações que são encontradas com a utilização de monitoração baseada nos padrões HTTP e XML.

Até o momento, alguns estudos têm sido feitos para a definição completa de uma nova arquitetura de gerenciamento baseada principalmente em padrões consagrados na área de transferência de dados na *Internet* e dentre esses trabalhos destaca-se a arquitetura *Web-based Integrated Management Architecture* (WIMA) desenvolvida por Flatin (2002), os trabalhos desenvolvidos na Universidade de Pohang pela equipe do professor Hong, conforme Yoon (2003), que propõe a definição de um *gateway* SNMP-XML para o gerenciamento de redes integradas baseado em XML e os trabalhos de Strauß e Klie (2003) da Universidade de Braunschweig para o gerenciamento Internet baseado em XML.

1.2 Objetivo

O objetivo desse trabalho é efetuar uma avaliação da arquitetura de monitoração HTTP/XML no acompanhamento do desempenho de um *cluster* de servidores. A aplicação desenvolvida foi especificada para atender a monitoração dos principais itens que comprometem o desempenho de um servidor, entre os quais se pode destacar a memória, o processador e a rede. Para avaliação da validade da aplicação desenvolvida, foi utilizado como ambiente de testes um *cluster* de servidores que atualmente executa, em condições de processamento paralelo, a análise de dispositivos eletromagnéticos pelo MEF.

Para efetuar essa avaliação, foi especificada e desenvolvida uma aplicação que implementa as características de monitoração baseadas nos padrões HTTP e XML, permitindo o levantamento das principais métricas obtidas a partir da medição de um sistema com essa conformação. Os dados obtidos a partir dos testes de monitoração são avaliados quanto às características de confiança dos dados coletados, segurança na transmissão das informações, desempenho da solução e facilidade de uso, com o objetivo de propor melhorias e adequações na aplicação de monitoração. Os dados coletados também são utilizados para permitir adaptações na formação do *cluster* de servidores e nos algoritmos utilizados para distribuição de carga entre os servidores, inclusive podendo auxiliar, em tempo real, o processo de balanceamento da distribuição de carga entre os servidores do *cluster*.

1.3 Metodologia de Trabalho

Para o desenvolvimento do presente trabalho, foram realizadas diversas atividades de levantamento de informações, necessárias para conhecimento da arquitetura SNMP, com suas principais vantagens e desvantagens, e dos estudos que já realizados para a proposição de novos modelos de monitoração, notadamente os baseados em padrões HTTP/XML. Essas atividades foram desenvolvidas por meio de pesquisa e leitura de artigos científicos, livros e documentos técnicos.

A partir do estabelecimento de uma base comum de conhecimento das arquiteturas SNMP e dos padrões HTTP/XML, foi desenvolvida a estruturação de uma aplicação de

monitoração de desempenho de servidores. A codificação da aplicação de monitoração foi efetuada com o uso da linguagem Java

A aplicação de monitoração foi validada em um ambiente de testes composto de um *cluster* de servidores localizados nas instalações da Escola Politécnica da USP, no departamento de Engenharia de Energia e Automação Elétricas (PEA). Os resultados obtidos durante os testes serviram de base para a elaboração de uma avaliação da aplicação e da distribuição de carga entre os servidores do *cluster*, elaboração da conclusão geral do trabalho e indicação de perspectivas para futuros trabalhos.

Uma atividade presente durante todas as etapas de realização desse trabalho foi a constante atualização e complementação da presente dissertação.

1.4 Organização do Trabalho

O trabalho está dividido da seguinte forma:

- No capítulo 2 – Arquitetura SNMP - são abordados os principais conceitos da arquitetura de monitoração SNMP. Nesse capítulo é descrito o funcionamento do protocolo SNMP, os seus principais componentes, o formato das mensagens que são trocadas entre o NMS e os elementos gerenciados. O capítulo termina com o estado atual do protocolo SNMP, ressaltando as características envolvidas no desenvolvimento de agentes e NMS, os problemas de segurança existentes nas versões 1, 2 e 3 do protocolo SNMP, os problemas de desempenho e o esquema básico de obtenção de informações, que é baseado no paradigma *pull mode* e no envio de *traps.*

- No capítulo 3 – Principais Conceitos das Propostas de Monitoração com HTTP/XML - são abordados os principais conceitos das propostas de monitoração de redes e servidores que utilizam os padrões HTTP e XML. Nesse capítulo são relacionadas as principais propostas de monitoração que foram desenvolvidas como alternativas ao modelo baseado em SNMP, começando com as alternativas de monitoração baseada em *browser* até as propostas mais recentes de monitoração com HTTP/XML e Java com objetos distribuídos. O capítulo termina com a apresentação do estado atual das tecnologias de monitoração e com a discussão da proposta baseada em conversores em oposição à proposta de comunicação nativa.

- No capítulo 4 – Estrutura da Aplicação de Monitoração - é descrita a estrutura da aplicação de monitoração por meio da utilização da metodologia IDEF0, para análise de processos e, na seqüência, a descrição das classes e de seus métodos e atributos, utilizando-se os principais conceitos de programação orientada a objetos. Também é apresentado neste capítulo o formato do documento XML que é utilizado para armazenamento das informações coletadas durante o processo de monitoração.

- No capítulo 5 – Implementação da Aplicação de Monitoração – são descritos os detalhes de implementação da aplicação de monitoração nos elementos gerenciados e no NMS, abordando características dos principais softwares envolvidos. Descreve-se também o modelo de comunicação adotado, as técnicas para verificação e armazenamento dos dados coletados e a interface gráfica de acompanhamento da monitoração.

- No capítulo 6 – Avaliação dos Resultados Obtidos – descreve-se o ambiente computacional em que os testes foram realizados, a descrição da bateria de testes que foram aplicados na aplicação de monitoração para validação de sua funcionalidade. Apresenta-se também os resultados obtidos durante o processo de monitoração e as principais dificuldades encontradas durante a realização dos testes.

- No capítulo 7 – Conclusão – é apresentada a conclusão do trabalho, são relacionadas as principais contribuições e descreve-se algumas sugestões para novas pesquisas.

2 ARQUITETURA SNMP

Neste capítulo são abordados os principais conceitos da arquitetura SNMP para gerenciamento de redes e servidores, por meio da descrição de seus três elementos principais: o protocolo SNMP, a MIB e a SMI.

2.1 Introdução

A necessidade de definição de uma arquitetura para gerenciamento de redes e servidores surgiu na década de 80, por meio de grupos de estudos pertencentes ao antigo *Commité Consultatif International de Telegraphique et Telephonique* (CCITT) - sendo representado desde 1993 pelo *International Telecommunications Union* (ITU-T) – e de grupos do IETF, conforme Stallings (1999).

O padrão para gerenciamento de redes e servidores proposto pelo ITU-T foi o *Common Management Information Protocol* (CMIP). O padrão CMIP foi elaborado dentro do contexto de desenvolvimento do modelo *Open Systems Interconnections* (OSI). Devido à inexpressiva utilização do modelo OSI e à ampla aceitação do padrão *Transmission Control Protocol/Internet Protocol* (TCP/IP), um novo padrão foi proposto pelo ITU-T, com o objetivo de garantir a continuidade da utilização do padrão CMIP, o qual foi chamado de *Common Management Information Protocol over TCP/IP* (CMOT). O padrão CMOT propôs a utilização do padrão CMIP na camada de aplicação da arquitetura TCP/IP, o que em tese permitiria a utilização conjunta da arquitetura TCP/IP e do padrão CMOT. No entanto, essa iniciativa não teve êxito, e o padrão CMOT foi abandonado na monitoração de redes e servidores, conforme Stallings (1999).

O padrão proposto pelo IETF em 1990 foi o SNMP, conforme descrito no *Request for Comments* (RFC) 1157, e definido em Case et alii (1990). O padrão SNMP é uma evolução do padrão *Simple Gateway Management Protocol* (SGMP), o protocolo pioneiro na Internet para gerenciamento de redes e servidores.

Apesar do padrão CMIP/CMOT e do padrão SNMP terem o mesmo objetivo, isto é, especificar uma proposta de gerenciamento de redes e servidores para a área de Tecnologia da Informação, interesses divergentes fizeram com que a área de telecomunicações adotasse o CMIP como padrão de gerenciamento de Centrais Telefônicas, sendo o SNMP adotado para uso no gerenciamento de redes de dados e de servidores, conforme Mauro e Schmidt (2001).

2.2 Principais Conceitos

2.2.1 Elementos da arquitetura SNMP

A arquitetura SNMP é composta de três elementos principais: o protocolo SNMP, a MIB e a SMI, conforme visto em Case et alii (1990).

O protocolo SNMP é utilizado para efetuar as transferências de dados entre o elemento gerenciado e o NMS, conforme Case et alii (1990) e os objetivos principais que direcionaram o seu desenvolvimento foram:

- Minimizar o número e a complexidade das funções realizadas pelos agentes;

- Permitir que o controle e monitoração obtida sobre os elementos gerenciados, com o uso do protocolo SNMP, seja suficientemente extensível para a inclusão de novas funcionalidades;

- Permitir que a arquitetura de gerenciamento não esteja vinculada a nenhum equipamento específico.

A MIB é a base de informações de gerenciamento, responsável por armazenar os dados coletados do elemento gerenciado e pela representação abstrata dos diversos elementos que compõem o elemento gerenciado, conforme McCloghrie e Rose (1988). Cada item que deve ser monitorado no elemento gerenciado é considerado um objeto da MIB e cada objeto é composto dos seguintes itens de descrição, conforme visto em McCloghrie e Rose (1988):

- Objeto – um nome textual que identifica o tipo do objeto e caracteriza o seu correspondente identificador de objeto;

- Sintaxe – a sintaxe abstrata para o tipo definido de objeto, representada com a utilização de *Abstract Syntax Notation One* (ASN.1) e SMI;

- Definição – uma descrição textual da semântica do tipo do objeto;

- Acesso – somente leitura, somente gravação, leitura e gravação e não acessível;

- Status – obrigatório, opcional ou obsoleto.

A SMI é a especificação da estrutura das informações de gerenciamento, sendo utilizada para a especificação das estruturas de dados que são armazenadas na MIB, conforme visto em Rose e McCloghrie (1990). A SMI define que cada objeto da MIB é formado por três características principais:

- Um nome que é representado como um identificador de objeto;

- O tipo do objeto que define a estrutura de dados do objeto;

- A codificação que indica quantas instâncias daquele objeto estão representadas.

A figura 2.1 apresenta o modelo de comunicação SNMP, em que são apresentadas as interações entre o NMS e o agente SNMP por meio de mensagens de consulta e resposta e de *traps*. O NMS é composto de uma plataforma SNMP, que efetua a formatação das mensagens SNMP, uma API de programação para interação entre a plataforma SNMP e a aplicação do usuário e uma aplicação de usuário que solicita os serviços para a API de programação e apresenta os resultados obtidos para o usuário. Para cada informação que o NMS precise obter, é enviada uma mensagem de consulta para o agente SNMP, que consulta a informação

na MIB e devolve uma mensagem de resposta com a informação solicitada para o NMS. Já os *traps* são mensagens não solicitadas, encaminhadas pelo agente para o NMS, para informar condições de falha no elemento gerenciado.

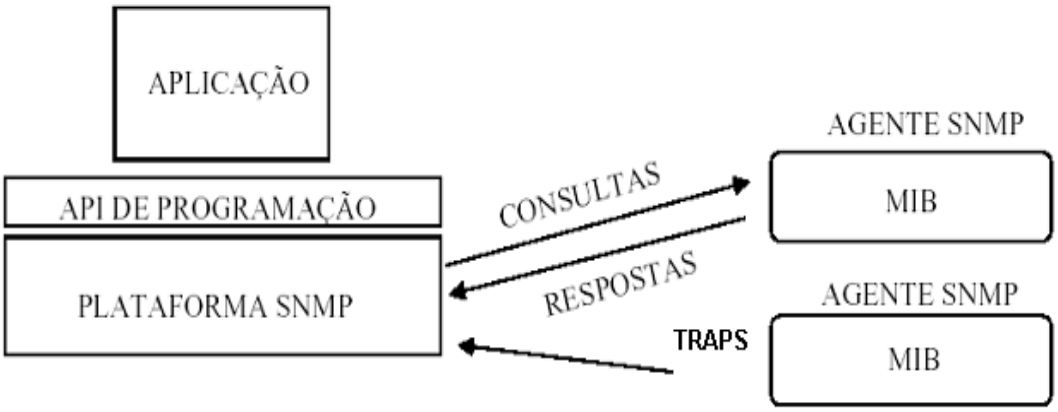

Figura 2.1 – Modelo de comunicação SNMP

2.2.2 Funcionamento do protocolo SNMP

O protocolo SNMP é baseado no paradigma de consulta e resposta e quase todas as informações de gerenciamento são obtidas a partir de solicitações efetuadas pelo NMS ao elemento gerenciado, conforme Stallings (1999). Apenas uma exceção é feita nesse modelo através da implementação da mensagem *trap*, que permite o envio de informações de gerenciamento de modo assíncrono -sem solicitação prévia- por iniciativa do elemento gerenciado para o NMS. O *trap* além de ser assíncrono também não tem mecanismo de confirmação de recebimento, sendo que a perda de um *trap,* por qualquer motivo durante a sua transmissão, acarreta a perda do pacote, pois não ocorre a re-transmissão do mesmo.

O protocolo SNMP versão 1 prevê em sua especificação cinco tipos de mensagem ou *Protocol Data Unit* (PDU). As PDU´s especificadas para a versão 1 são, conforme Case et alii (1990):

- *GetRequest;*
- *GetNextRequest;*
- *GetResponse;*
- *SetRequest;*
- *Trap.*

A PDU SNMP é encapsulada em pacotes TCP/IP, conforme ilustrado na figura 2.2, na qual verifica-se que a PDU SNMP faz parte da porção de dados do quadro SNMP, sendo que a PDU SNMP é encapsulada em um quadro SNMP, quadro esse que por sua vez é a

porção de dados de um segmento UDP, sendo encapsulado em um segmento UDP, que por sua vez é encapsulado em um pacote IP que finalmente é encaminhado pela rede dentro de um quadro *Ethernet*. No elemento gerenciado destinatário desse pacote SNMP o processo inverso é realizado e o quadro SNMP é restaurado a partir do quadro *Ethernet* recebido pela rede.

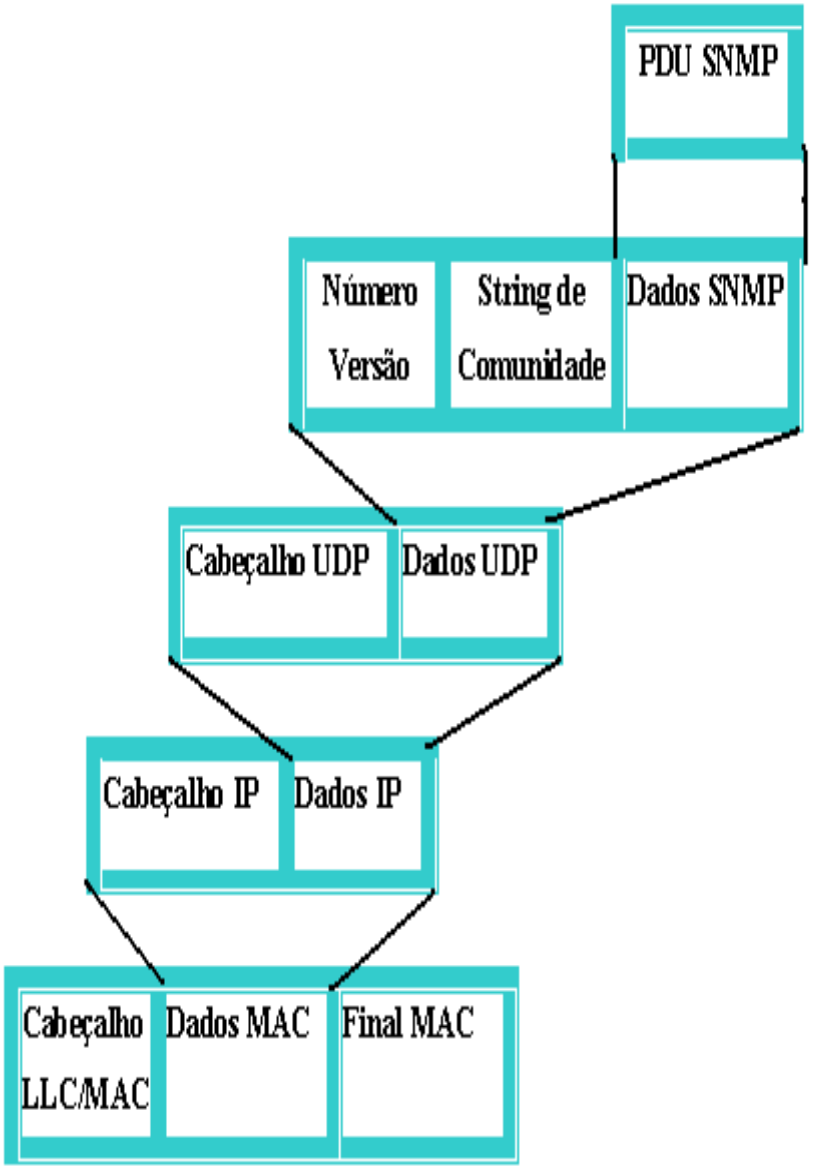

Figura 2.2 – Encapsulamento do quadro SNMP no TCP/IP.

O formato especificado para as PDU´s *GetRequest*, *GetNextRequest*, *GetResponse* e *SetRequest* , conforme Case et alii (1990) está ilustrado na figura 2.3.

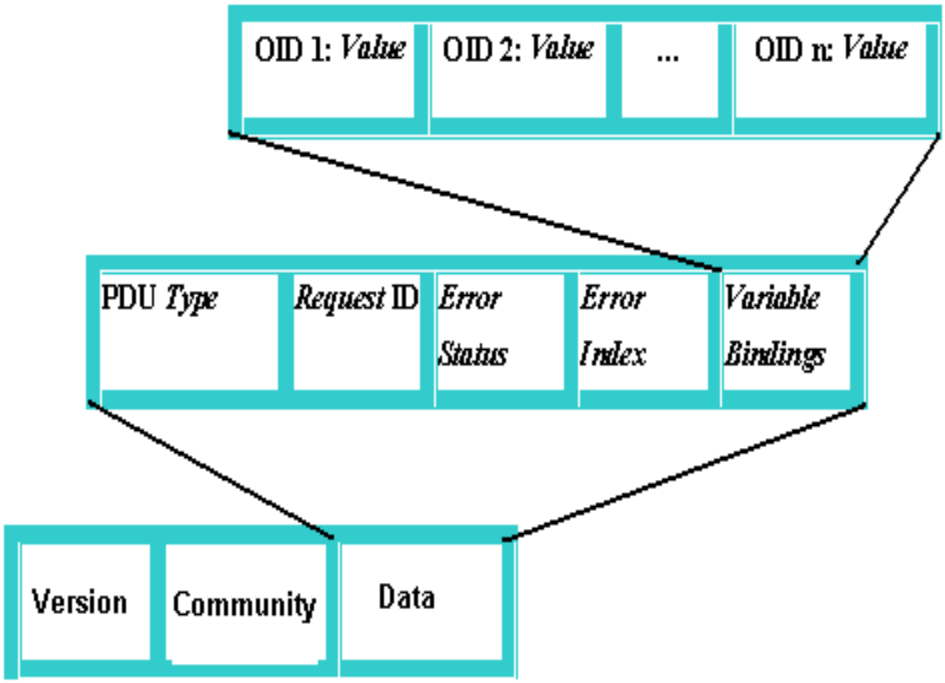

Figura 2.3 – O formato da PDU de requisição e resposta.

O significado dos campos da PDU de requisição e resposta é o seguinte, conforme Case et alii (1990):

PDU type: especifica o tipo da PDU, podendo receber os seguintes valores:

- 0 para *GetRequest*;
- 1 para *GetNextRequest*;
- 2 para *GetResponse*;
- 3 para *SetRequest*.

Request ID: campo utilizado para associar PDU´s de solicitação com PDU´s de resposta.

Error Status: indica um número de erro e tipo de erro. Apenas a PDU de resposta utiliza este campo. As demais PDU´s definem este campo com o valor 0.

Error Index: associa um erro com uma instância de objeto. Apenas a PDU de resposta utiliza este campo. As demais PDU´s definem este campo com o valor 0.

Variable bindings: É o campo de dados da PDU Cada *Variable binding* associa uma instância de objeto com o seu valor atual, com exceção das PDU´s *GetRequest* e *GetNextRequest*, que ignoram este campo.

O formato especificado para a PDU *Trap* está ilustrado na figura 2.4.

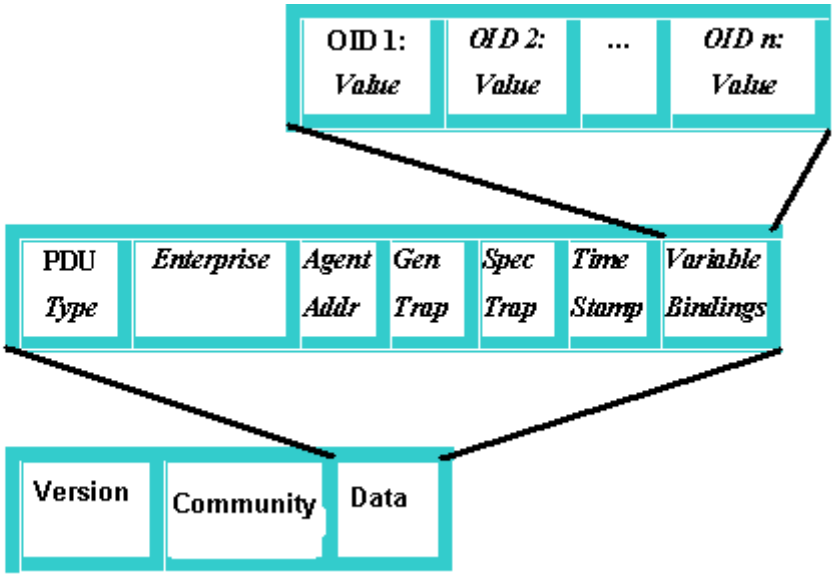

Figura 2.4 – O formato da PDU *Trap*.

A seguir, é descrito o significado dos campos da PDU *Trap*, conforme Case et alii (1990):

PDU type: especifica o tipo da PDU e valor 4 indica *Trap*.

Enterprise: Identifica a empresa que registrou o *Trap*.

Agent address: endereço IP do agente, usado para identificação futura.

Generic trap type: campo que descreve o evento que está sendo informado e pode ter os seguintes valores:

- 0 (*coldStart*) – indica que o agente foi iniciado.
- 1 (*warmStart*) – indica que as configurações do agente foram alteradas.
- 2 (*linkDown*) – indica uma falha na comunicação.
- 3 (*linkUp*) – indica o reinício da comunicação.
- 4 (*AuthenticationFailure*) – indica que o agente recebeu uma requisição de um NMS não autorizado.
- 5 (*egpNeighborLoss*) – indica que a outra ponte de um sistema EGP foi desligada.
- 6 (*enterpriseSpecific*) – outro tipo de *Trap* ocorreu.

Specific trap type: usasdo para identificar um *Trap* não-genérico quando o campo *Generic Trap Type* for igual a 6.

Time stamp: contém o valor do objeto *sysUpTime* do elemento gerenciado no momento em que o *Trap* foi gerado.

2.2.2.1 PDU *GetRequest*

A PDU *GetRequest* é a mensagem especificada no protocolo SNMP para solicitação de dados de gerenciamento. O NMS transmite uma PDU *GetRequest* para um elemento gerenciado a fim de coletar informações que estão armazenadas na MIB do elemento gerenciado. Como resposta à PDU *GetRequest*, o elemento gerenciado transmite para o NMS a PDU *GetResponse*, com os dados solicitados, conforme Case et alii (1990).

2.2.2.2 PDU *GetNextRequest*

A PDU *GetNextRequest* é a mensagem especificada no protocolo SNMP para solicitação de dados de gerenciamento a partir de um ponto determinado na MIB. O NMS transmite uma PDU *GetNextRequest* para um elemento gerenciado informando que ele quer obter dados de gerenciamento do próximo objeto a partir do objeto informado na PDU. Esta PDU é uma forma prática de pesquisa na árvore da MIB, para objetos que possuem quantidade de ocorrências variável, conforme Case et alii (1990).

2.2.2.3 PDU *GetResponse*

A PDU *GetResponse* é a mensagem especificada no protocolo SNMP para resposta às PDU´s *GetRequest*, *GetNextRequest* e *SetRequest* para confirmação da execução de comandos ou retorno com dados solicitados, conforme Case et alii (1990).

2.2.3 Definição de tipos de dados com SMI

O modelo de definição de dados da arquitetura SNMP é baseado na especificação SMI versões 1 e 2. A SMI é um sub-conjunto da especificação ASN.1, sendo que tipos primitivos de variáveis são incorporados à especificação SMI a partir de um processo de herança de especificações do padrão ASN.1, conforme Stallings (1999).

A versão 1 da SMI é especificada através da RFC1155-*Structure and Identification of Management Information for TCP/IP-based Internets* , conforme Rose e McCloghrie (1990). A SMI tem as especificações dos tipos e dos formatos dos dados que podem ser armazenados na MIB e que correspondem a visões abstratas de elementos do mundo real. Dessa forma, a SMI é uma meta-linguagem de especificação de dados, que é utilizada pelos desenvolvedores de softwares de agentes e NMS de monitoração de redes e sistemas para especificação das variáveis que serão utilizadas nos programas de gerenciamento. As variáveis de programas SNMP recebem o nome de objetos.

Os principais tipos definidos na versão 1 da SMI são os seguintes:

- *NetworkAddress* – representa um endereço no padrão da *Internet;*
- *IpAddress* – representa um endereço *Internet* de 32 bits*;*
- *Counter* – representa um número inteiro não negativo que é incrementado de forma monotônica até um limite definido, ocasião em que o contador é novamente definido a partir do zero*;*
- *Gauge* – representa um número inteiro não negativo que é incrementado ou decrementado até um valor limite*;*
- *TimeTicks* – representa um número inteiro não negativo que conta o tempo em centésimos de segundo desde uma época inicial definida*;*
- *Opaque* – utilizado para transmissão de dados de sintaxe arbitrária.

A versão 1 da SMI tem algumas limitações em relação aos tipos de dados que podem ser definidos e a quantidade de bytes que podem ser armazenados em cada uma das variáveis. Por esses motivos, uma nova versão da SMI foi especificada na RFC2578-*Structure of Management Information Version 2 (SMIv2)* , conforme McCloghrie et alii (1999).

A SMI versão 2 é dividida em 3 partes principais :

- Definições de módulos, que serve para definir grupos de informações com características similares;
- Definições de objetos, que são usadas para descrever os objetos gerenciados e suas características;
- Definições de notificações, que são utilizadas para descrever o formato e as características das mensagens de notificação, conhecidas como *traps*.

Os principais tipos definidos na versão 2 da SMI são os seguintes:

- *Integer32* – representa um inteiro entre –2^31 e 2^31-1;
- *OCTECT STRING* – representa binários arbitrários ou dados textuais;
- *Counter32* – representa um número inteiro não negativo que é incrementado de forma monotônica até um limite definido (2^31-1), ocasião em que o contador é novamente definido a partir do zero*;*
- *Gauge32* – representa um número inteiro não negativo que é incrementado ou decrementado até um valor limite (2^31-1)*;*
- *Unsigned32* – representa uma informação de valor inteiro entre 0 e 2^31-1 inclusive;

- *Counter64* – representa um número inteiro não negativo que é incrementado de forma monotônica até um limite definido (2^64-1), ocasião em que o contador é novamente definido a partir do zero.

As principais macros definidos na versão 2 da SMI são as seguintes:

- *Notification-type* – utilizada para descrever transmissões não solicitadas de informações de gerenciamento;

- *Object-Identity* – utilizada para descrever objetos gerenciados;

- *Module-Identity* – utilizada para descrever módulos de informações com características semelhantes.

2.2.4 Armazenamento de dados na MIB

A MIB é uma base de informações de gerenciamento que é instalada internamente a cada elemento gerenciado e que é atualizada pelo agente SNMP local para prover ou armazenar dados concernentes aos itens de monitoração. Os itens mais comumente monitorados são os componentes de *hardware* e *software* disponíveis em um elemento gerenciado.

A primeira versão da MIB é especificada na RFC1158-*Management Information Base for Network Management of TCP/IP-based internets: MIB-II* , conforme Rose (1990), que foi tornada obsoleta através da especificação RFC1213-*Management Information Base for Network Management of TCP/IP-based internets: MIB-II* , conforme McCloghrie e Rose (1991) e atualizada para a versão 2 do protocolo SNMP por meio das RFC2011-*SNMPv2 Management Information Base for the Internet Protocol using SMIv2* , conforme McCloghrie (1996a), RFC2012- *SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2* , conforme McCloghrie (1996b) e RFC2013-*SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2* , conforme visto em McCloghrie (1996c).

A MIB é composta por objetos que efetuam uma representação do comportamento operacional dos componentes reais do elemento gerenciado, conforme visto em Mauro e Schmidt (2001). A estrutura da MIB é em formato de uma árvore invertida e a pesquisa de dados na MIB é efetuada a partir da raiz e continua, de acordo com o argumento de pesquisa, de maneira seqüencial pelos diversos galhos da árvore até alcançar o objeto solicitado. Dessa forma, um objeto que se situe no ramo *system* da MIB terá a denominação alfabética ISO.org.DoD.internet.management.MIB-2.system, que corresponde a identificação numérica 1.3.6.1.1.2.1.1. A figura 2.5 ilustra, de forma resumida, o formato padrão da MIB.

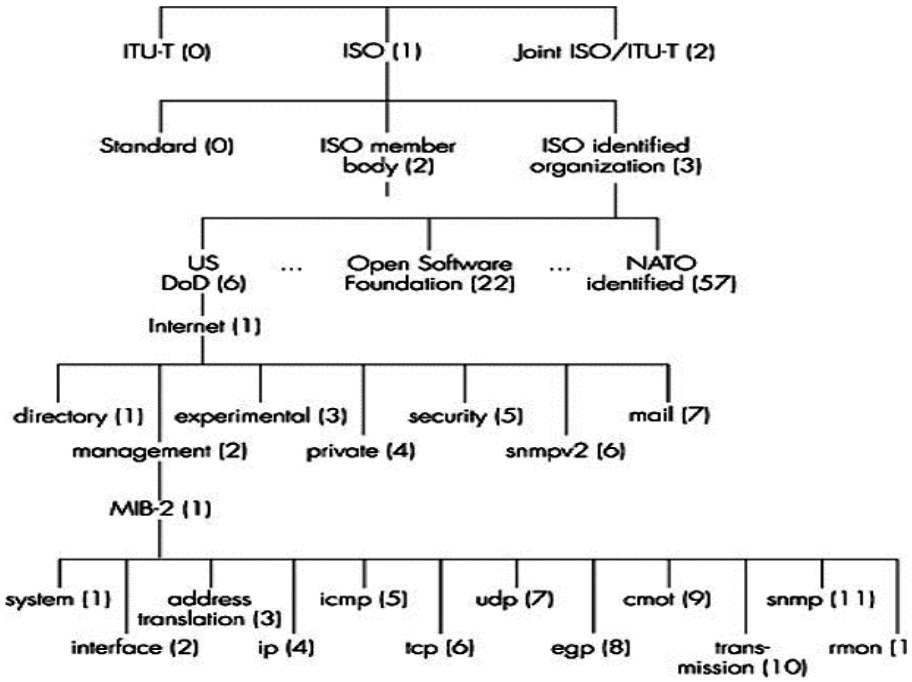

Figura 2.5 – O formato da MIB.

Figura retirada de Stallings (1999).

2.3 Estado Atual da Arquitetura SNMP

Neste subitem serão relacionadas as principais características da arquitetura SNMP que são consideradas como limitações existentes no modelo atual e que indicam a necessidade de elaboração de estudos e propostas para solucionar essas dificuldades.

2.3.1 *Pull Mode* (Modo de puxar)

O modo de comunicação básico utilizado entre o NMS e o agente na arquitetura SNMP é o baseado no paradigma *Pull Mode* , conforme Flatin (1999). Por meio desse método o NMS inicia o processo de comunicação geralmente de maneira programada e com intervalos de tempo regulares, os quais podem variar dependendo de condições identificadas de anormalidade. Por exemplo, se o intervalo de *polling* é de 5 minutos e se não houver resposta ao primeiro *polling*, o NMS pode executar o próximo *polling* em um intervalo de tempo diferente do especificado para o primeiro *polling*.

O *Pull Mode*, conforme Flatin (1999), possui o inconveniente de solicitar a obtenção da informação mesmo que não tenha havido mudança em seu valor, o que ocasiona, geralmente, a transmissão de dados já conhecidos, constituindo-se em *overhead* na rede.

O modo oposto ao *Pull Mode* é o *Push Mode* (modo de empurrar), o qual é utilizado na arquitetura SNMP apenas para o envio de mensagens não solicitadas, que recebem o nome

de *Traps*, e que são utilizadas geralmente para indicar condições de erro no elemento gerenciado.

2.3.2 Segurança

O mecanismo de segurança especificado na versão 1, conforme Case et alii (1990) da arquitetura SNMP é baseado no conceito de comunidades. Por meio dessa implementação é combinada uma palavra chave para acesso às informações do elemento gerenciado, que é utilizada em cada mensagem do NMS para o agente e vice-versa. A comunidade, bem como a mensagem SNMP inteira, são encaminhadas sem criptografia, sendo totalmente vulnerável para leitura. Além disso, a comunicação realizada entre o NMS e o agente não possui nenhum método de autenticação.

As versões 2 e 3 do SNMP especificam melhorias no esquema de autenticação e criptografia, incluindo no processo de autenticação o uso do algoritmo de *hash* MD5 e no processo de criptografia o uso do algoritmo DES, conforme Stallings (1999). Também foi introduzida a utilização de mecanismos de relógio nas mensagens, para evitar fraudes através da utilização de mecanismo de checagem *anti-replay*.

Apesar de as versões 2 e 3 do SNMP, conforme Stallings (1999), apresentarem melhorias significativas em relação aos mecanismos de segurança na troca de mensagens entre o NMS e os elementos gerenciados, essas versões não são amplamente implementadas na Internet e nas redes comerciais. Isto se deve principalmente à necessidade de atualização das versões dos softwares de gerenciamento, o que ocasiona a necessidade de gastos para aquisição e atualização de software. A complexidade da configuração de parâmetros das versões 2 e 3 do protocolo SNMP também é outro fator que tem dificultado a adoção dessas novas versões.

2.3.3 Desempenho

A arquitetura de gerenciamento SNMP, conforme visto em Case et alii (1990) foi concebida em um modelo centralizado e, devido a essa característica, as implementações que foram realizadas pelos desenvolvedores de sistemas de gerenciamento ficaram restritas a números relativamente pequenos de elementos que podem ser monitorados simultaneamente por um NMS, devido ao constante aumento da latência na comunicação entre o NMS e o agente, conforme Yoon et alii (2003). É comum encontrar sistemas de gerenciamento que suportam a monitoração simultânea de apenas 500 endereços IP. Devido a essa limitação existem diferentes modelos de implementação de sistemas de gerenciamento, os quais são geralmente incompatíveis entre si.

2.3.4 Desenvolvimento de agentes e NMS

Devido à especificação SMI ser utilizada apenas para a codificação de programas para sistemas que trabalham com a arquitetura SNMP, existem poucos analistas disponíveis no mercado para desenvolvimento desses sistemas, conforme Stallings (1999). Além disso, a

codificação SMI com sintaxe de transferência BER é complexa e bastante específica, além de ter sua especificação baseada em definição binária de campos, conforme ITU-T (2002).

2.4 Conclusão

Neste capítulo foram apresentados os principais conceitos da arquitetura SNMP, seus componentes e os formatos de transferência de dados entre o NMS e o elemento gerenciado. Foram abordadas também questões relacionadas ao estado atual do protocolo SNMP e também as principais dificuldades que usuários e fabricantes encontram no uso do SNMP para monitoração de redes e servidores, principalmente nas questões relativas a desempenho do protocolo, segurança no envio de dados e autenticação de acessos, dificuldades para treinamento de analistas para o desenvolvimento de aplicativos agentes e NMS (devido ao protocolo SNMP ser uma solução de nicho e a estrutura básica de obtenção de dados ser baseada no paradigma consulta-resposta e no envio de *traps)*.

No próximo capítulo serão apresentados os principais conceitos de novos modelos de monitoração de servidores e redes, baseados em padrões *Web* e que se propõem a solucionar os problemas do protocolo SNMP, conforme apresentados no presente capítulo.

3 PRINCIPAIS CONCEITOS DAS PROPOSTAS DE MONITORAÇÃO COM HTTP/XML

Neste capítulo são abordados os principais conceitos de arquiteturas de gerenciamento de redes e servidores baseados ou relacionados com os padrões *Hyper Text Transfer Protocol* (HTTP) e *Extended Markup Language* (XML). Em algumas propostas, a utilização do protocolo HTTP é realizada apenas em parte do processo de comunicação e complementada com o protocolo SNMP e, em alguns casos, a comunicação é efetuada fim-a-fim pelo protocolo HTTP e a definição de dados é feita no padrão XML.

3.1 Introdução

A constante evolução das redes de computadores tem levado pesquisadores a refletir sobre novas necessidades de monitoração de redes e de servidores, ocasionando a formação de grupos de estudo com o objetivo de avaliar e definir uma nova arquitetura de gerenciamento de redes e servidores que possa corrigir os problemas existentes na arquitetura de gerenciamento SNMP. Dentre os principais estudos que têm sido realizados, aparece, várias vezes na literatura acadêmica, a proposição de modelos de monitoração que utilizam padrões consagrados no ambiente Internet, conforme Flatin (2002).

Os modelos de monitoração propostos pelos pesquisadores possuem algumas variações em suas características, mas, em geral, a utilização de XML como sintaxe de especificação de dados e o protocolo HTTP para transporte de dados são consideradas opções já consagradas, devido à ampla utilização desses padrões em aplicações que são desenvolvidas para o mercado corporativo e doméstico, conforme Ju et alii (2002).

3.2 Principais Conceitos

No livro *Web-Based Management of IP Network and Systems* o autor Jean Philippe Martin Flatin relaciona uma descrição abrangente e didática das principais abordagens utilizadas nos últimos anos para a definição e implementação de soluções de gerenciamento de redes e sistemas com a utilização de soluções baseadas em padrões *Web*. Nas pesquisas realizadas em outras publicações da área não foram encontrados trabalhos que efetuem uma abordagem parecida. Por esses motivos, a descrição efetuada por Flatin (2002) foi utilizada como referência básica nesta parte do trabalho.

3.2.1 Monitoração Baseada em *Browser*

Na categoria de monitoração baseada em *browser*, são incluídos os softwares desenvolvidos geralmente com as tecnologias HTTP, *Hyper Text Markup Language* (HTML) e *Common Gateway Interface* (CGI), e que permitem a publicação de informações sobre a

monitoração de equipamentos de rede e de servidores em um servidor *Web*, como *Apache* ou *Internet Information Server* (IIS). O acesso às informações de monitoração é realizado por meio do uso interativo de um *browser Web* para visualização e acompanhamento de relatórios com informações coletadas dos equipamentos monitorados, conforme Flatin (2002). Um exemplo conhecido de *software* que implementa a monitoração baseada em *browser* é o *Multi-Router Traffic Grapher* (MRTG), desenvolvido por Tobias Oetiker e disponibilizado na Internet no sítio www.mrtg.org, conforme Oetiker (2005).

3.2.2 Meta-gerenciamento Baseado em *Browser*

O Meta-gerenciamento Baseado em *Browser* é uma versão melhorada da Monitoração Baseada em *Browser*, conforme descrito em Flatin (2002). Para melhorar as características de monitoração em tempo real do Meta-gerenciamento Baseado em *Browser*, por meio do acesso aos relatórios de publicação de informações, foram acrescentadas pelas empresas que desenvolvem *softwares* de monitoração, funções complementares que geralmente são executadas em ambientes de monitoração complexos, como por exemplo:

- *Interface* para abertura automática de chamados em centrais de *help-desk*;
- Acompanhamento e controle de chamados abertos;
- Procedimentos de gerenciamento de problemas;
- Documentação técnica.

3.2.3 Gerenciamento Baseado em *Browser*

O gerenciamento baseado em *browser* é uma evolução da Monitoração Baseada em *Browser*, pois nessa modalidade de gerenciamento o operador da área de monitoração executa a verificação de dados de uso de recursos de rede e de servidores, por meio de softwares de coleta, armazenagem e publicação de relatórios em formato HTML e também executa tarefas de verificação de problemas nos equipamentos gerenciados. O acesso aos equipamentos gerenciados é efetuado com o uso de softwares baseados em *browser*, que utilizam o paradigma cliente servidor para acesso aos dados armazenados nos equipamentos gerenciados. O operador, por meio de acesso ao *web browser* de sua estação de trabalho, simula um acesso na condição de cliente ao elemento NMS instalado no equipamento gerenciado. Os dois aplicativos mais utilizados nessa modalidade de acesso são os baseados no protocolo SNMP e o Telnet, conforme Flatin (2002).

3.2.4 Gerenciamento em 3 Camadas

O gerenciamento em 3 camadas pode ser considerado quase como o passo anterior a uma plena comunicação HTTP fim-a-fim. Nessa estrutura de gerenciamento, a comunicação entre o NMS e o agente é efetuada por intermédio de um terceiro elemento, um servidor intermediário, que atua tipicamente como um conversor HTTP para SNMP e vice-versa.

Tipicamente, no gerenciamento em 3 camadas, as solicitações de dados são efetuadas pelo NMS no formato HTTP ao conversor, que transforma a mensagem HTTP em uma mensagem SNMP, que é na seqüência encaminhada para o elemento gerenciado. O elemento gerenciado, ao receber uma mensagem de solicitação de dados em formato SNMP, elabora uma mensagem de resposta no formato SNMP e encaminha a mensagem para o conversor HTTP-SNMP, que efetua o tratamento da mensagem e elabora uma resposta no formato HTTP. Essa mensagem HTTP é então encaminhada para o NMS, conforme Flatin (2002). Na figura 3.1 está representada a comunicação de gerenciamento em 3 camadas.

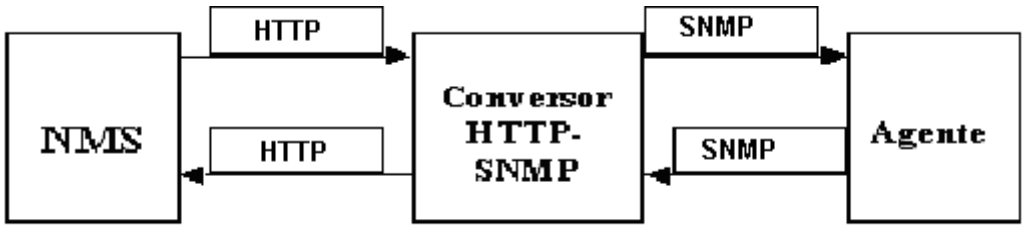

Figura 3.1 – Gerenciamento em 3 camadas.

3.2.5 Gerenciamento Baseado em HTTP

O gerenciamento baseado em HTTP permite a plena comunicação HTTP fim-a-fim entre o NMS e o elemento gerenciado. Tipicamente, nesta modalidade de gerenciamento, as solicitações de dados são efetuadas pelo NMS no formato HTTP e o agente tem embutido em seus componentes internos, as funções de HTTP *server* e HTTP *client* e de SNMP *server* e *client*. O elemento gerenciado, ao receber uma mensagem de solicitação de dados em formato HTTP, converte a solicitação para o formato SNMP e encaminha a mensagem para o seu agente interno SNMP, que efetua o tratamento da mensagem e elabora uma resposta no formato SNMP. Essa mensagem SNMP é convertida para uma mensagem HTTP que é então encaminhada para o NMS, conforme Flatin (2002). Na figura 3.2 está representado o gerenciamento baseado em HTTP.

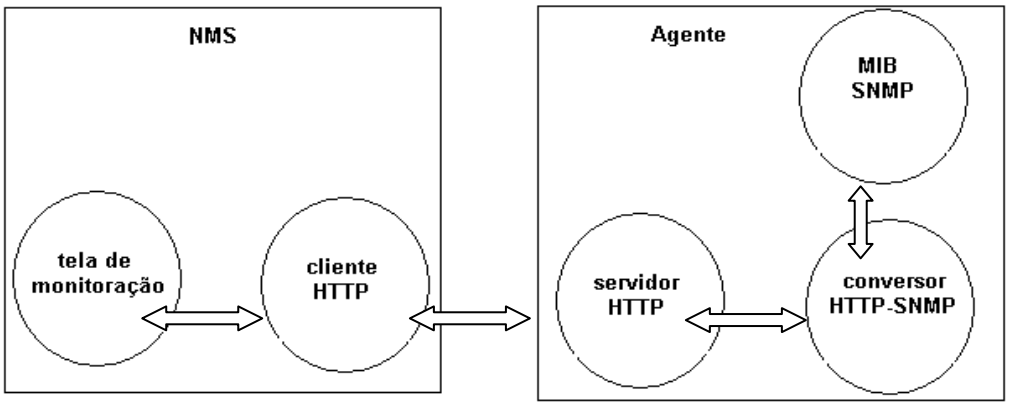

Figura 3.2 – Gerenciamento baseado em HTTP.

3.2.6 Gerenciamento Baseado em XML

A próxima etapa de sofisticação no desenvolvimento de arquiteturas de gerenciamento de redes e servidores compreende a utilização de XML como padrão de representação de informações de gerenciamento e a transferência dessas informações por meio do uso do protocolo de comunicação HTTP. Na literatura acadêmica são identificadas pelo menos quatro arquiteturas de gerenciamento de redes e servidores que fazem uso do conceito de gerenciamento baseado em XML. De acordo com Choi (2004), as quatro arquiteturas são:

- *Web-based Enterprise Management* (WBEM);
- *Web-based Integrated Management Architeture* (WIMA);
- *XML-based Architecture for SNMP Management of Network and Application* (XNAMI);
- *XML-based Network Management* (XNM).

A arquitetura WBEM foi desenvolvida pela equipe do *Desktop Management Task Force* (DMTF) e sua especificação está disponível no sítio http://www.dmtf.org/standards/wbem. A arquitetura WBEM define um modelo de informação chamado *Common Information Model* (CIM), o qual é um padrão para modelagem de objetos gerenciáveis, que são representações de recursos do mundo real. A arquitetura WBEM provê um padrão de representação e acesso às informações de monitoração e especifica um padrão de mapeamento entre a representação de objetos CIM em documentos XML e a transmissão dos documentos com o uso do protocolo HTTP. A arquitetura WBEM ainda não suporta o padrão *Simple Object Access Protocol* (SOAP), mas está sendo atualmente atualizada para incorporar o SOAP, principalmente devido às possibilidades de interoperabilidade que o SOAP proporciona e que podem ser utilizadas na comunicação de aplicativos que implementam a arquitetura WBEM, com aplicativos que implementem outras arquiteturas de gerenciamento com o SOAP, conforme Choi (2004).

A arquitetura WIMA foi desenvolvida por Flatin (2002) e vários de seus conceitos principais são utilizados na implementação do aplicativo de monitoração desenvolvido no presente trabalho. Flatin foi o primeiro pesquisador a propor a utilização de XML para o gerenciamento de redes e sistemas. A arquitetura WIMA propõe a utilização da monitoração baseada no envio de informações do elemento gerenciado para o NMS, contrariamente ao uso de monitoração baseada apenas em consultas realizadas pelo NMS. A arquitetura WIMA prevê que a troca de informações seja realizada por meio do protocolo HTTP, com a utilização de *Multipurpose Internet Mail Extensions* (MIME), permitindo a transferência de diferentes tipos de modelos de dados, como dados SNMP codificados em BER, documentos XML ou objetos CIM, conforme CHOI (2004).

A arquitetura XNAMI foi desenvolvida por John et alii (1999) e permite que um NMS possa estender a MIB do agente SNMP. Na arquitetura XNAMI, o NMS pode transferir código comprimido Java para o elemento gerenciado por meio do uso de uma operação SNMP SET, que provocará no elemento gerenciado a extensão da MIB, com o armazenamento do código Java transferido. O agente XNAMI que está instalado no elemento gerenciado mantém uma cópia dos dados da MIB em formato de documento XML, o qual pode ser recuperado ou alterado por meio das operações tradicionais da arquitetura SNMP. O

documento XML - ou partes do documento - são transferidos entre o agente do elemento gerenciado e o NMS por meio do uso do protocolo HTTP, conforme Choi (2004).

A arquitetura XNM foi desenvolvida pela equipe da Universidade de Pohang, conforme descrito por Choi (2004), e tem dois componentes principais: um agente no elemento gerenciado e um NMS. A arquitetura XNM utiliza XML para transferir informações de gerenciamento com o uso do protocolo HTTP. Adicionalmente, a arquitetura propõe o uso de *Document Object Model* (DOM) para representar e processar os dados e Xpath para viabilizar o acesso a partes específicas de dados de gerenciamento, conforme Choi (2004). Como a arquitetura XNM é uma versão mais elaborada da arquitetura WIMA, pois ela aprofunda e especifica, com maiores detalhes, os mecanismos e tecnologias necessárias para a implementação de uma arquitetura de gerenciamento de redes e sistemas baseada em XML, a arquitetura XNM será utilizada como base para a implementação da aplicação de monitoração do presente trabalho.

3.2.7 Gerenciamento Baseado em Java Distribuído

O mais alto grau de sofisticação no gerenciamento de redes e servidores, conforme descrito por Flatin (2002) é o gerenciamento Baseado em Java Distribuído. Nessa arquitetura são utilizados os mesmos conceitos da arquitetura de gerenciamento Baseado em XML, mas com a utilização de objetos distribuídos, desenvolvidos na linguagem Java e acessados remotamente por meio da especificação *Java Remote Method Invocation* (RMI).

3.3 O Padrão XML

O padrão XML é um padrão aberto para modelagem de informação e que é mantido e disponibilizado gratuitamente pelo *World Wide Web Consortium* , conforme W3C (2004). O XML está sendo amplamente utilizado no desenvolvimento de sistemas *Web* e tem se tornado um padrão de fato para a construção de sistemas que necessitem fazer a transferência de dados entre estações clientes e servidores. A transferência de documentos XML pode ser efetuada através do protocolo de comunicação HTTP, o qual está disponível atualmente em quase todos os servidores e equipamentos de redes, conforme Yoon et alii (2003).

Entre as principais vantagens do padrão XML Flatin (2002) cita as seguintes características:

- a modelagem de informação prevista no padrão XML é versátil e poderosa, sendo possível especificar de forma clara e precisa qualquer tipo de dado.

- a utilização de XML permite que praticamente qualquer programador que conheça XML desenvolva sistemas de armazenamento ou de busca de informações de gerenciamento de redes e de servidores de maneira fácil e a um custo baixo, se comparado ao processo de desenvolvimento de aplicações SNMP.

- existem no mercado vários produtos para desenvolvimento de programas no padrão XML e muitos deles são gratuitos.

- a transmissão de documentos XML é feita sobre o protocolo de comunicação HTTP e considerando-se que qualquer equipamento de redes, bem como servidores e clientes utilizam o HTTP, pode-se acessar informações de elementos gerenciados a partir de qualquer origem ou destino e sem a necessidade de adição ou aquisição de novos softwares.

- a utilização de XML com HTTP permite a utilização dos comandos POST e GET do HTTP, viabilizando a implementação do *Push Mode* a partir do elemento gerenciado para o elemento gerenciador, ocasionando assim economia de banda da rede e recursos computacionais para a execução de atividades de *polling*.

- como o padrão XML trabalha com o conceito de documento, é possível a implementação eficiente do gerenciamento de configuração dos elementos gerenciados, pois uma nova configuração de um equipamento será transferida como um documento único e não como elementos separados e sem conexão (UDP – *User Datagram Protocol*), que é o método de transferência de mensagens no SNMP.

3.3.1 Conversão de SMI para XML

A definição de um padrão para conversão da SMI, que é a estrutura da MIB, para XML é uma necessidade imediata na especificação e desenvolvimento de aplicações de monitoração de redes e sistemas baseada em XML. Por esse motivo, alguns trabalhos têm sido desenvolvidos para a definição de um padrão de conversão da estrutura da MIB para uma estrutura equivalente em documentos XML. Dentre os trabalhos desenvolvidos destacam-se:

- Os conceitos de mapeamento baseado em modelo e em meta-modelo elaborados por Flatin (2002);

- A especificação e elaboração da aplicação libsmi de conversão de SMI para XML elaborada por Strauß e Klie (2003);

- A especificação do algoritmo de conversão elaborado por Yoon et alii (2003);

- A implementação do algoritmo de Yoon et alii (2003) realizada por Choi (2004).

O conceito de mapeamento baseado em modelo, conforme definido em Flatin (2002) é aquele no qual cada MIB corresponde a um documento XML e os elementos e atributos do documento XML têm os mesmos nomes das variáveis e cláusulas da MIB. Um exemplo simples do grupo *interface* da MIB-II, conforme citado no livro de Flatin (2002) e transcrito abaixo:

<interface>

<bandwidth type=”string”>100 Mbit/s</bandwidth>

</interface>

O conceito de mapeamento baseado em meta-modelo, conforme Flatin (2002) é aquele no qual o documento XML é genérico e idêntico para todas as MIB´s, dessa forma todos os elementos e atributos têm nomes genéricos, conforme citado no livro de Flatin (2002) e transcrito abaixo:

<class name=”interface”

<property name=”bandwidth” type=”string”>

<value>100 Mbit/s</value>

</property>

</class>

Como comparação dos dois modelos apresentados por Flatin (2002), verifica-se que o mapeamento baseado em modelo é mais reduzido, legível e intuitivo para o usuário final do que o mapeamento baseado em meta-modelo, mas estas características não são consideradas por Flatin (2002) como decisivas na escolha de um modelo, e que uma avaliação mais criteriosa deve ser realizada.

O artigo de Strauß e Klie (2003) intitulado *Towards XML oriented Internet Management* apresenta as abordagens que estão sendo utilizadas por alguns fabricantes como Juniper e Avaya, para a definição de um algoritmo de conversão da estrutura da MIB para XML *Schema*, comenta os trabalhos desenvolvidos pela equipe POSTECH e Flatin e apresenta a ferramenta *libsmi*. A ferramenta *libsmi* é uma biblioteca de programas que permite a aplicações de gerenciamento o acesso a informações da MIB a partir de aplicativos que checam, analisam, convertem e comparam definições da MIB com XML *Schema*.

O artigo de Yoon et alii (2003) intitulado *Development of SNMP-XML translator and gateway for XML-based integrated network management*, aborda o problema da necessidade de estudos para a definição de um padrão de conversão e formatação de informações de monitoração em documentos XML. O algoritmo proposto no artigo é baseado no conceito de preservar toda a informação do modelo anterior para o novo modelo. Dessa forma, conforme pode ser verificado na figura 3.3, toda macro *Node* do SMI é convertida em um *Element* do XML, todo *Node name* é convertido em um *Element name* e todas as cláusulas de cada nó são convertidas em atributos do elemento.

SNMP SMI	XML Schema
Node (macro definition)	Element
Node name	Element name
Clauses in node	Attributes of the element

Figura 3.3 – Estrutura de conversão entre SMI e XML.

Figura retirada do artigo de Yoon et alii (2003).

O algoritmo de Yoon et alii (2003) também propõe que seja utilizado o padrão XML *Schema*, devido à capacidade deste padrão em representar vários tipos diferentes de dados, ao contrário do padrão XML *Document Type Definition* (DTD) que é limitado a uma quantidade definida de tipos de dados. A figura 3.4 apresenta o resultado da conversão das expressões da

SMI versão 1 para XML *Schema* com a utilização do algoritmo definido por Yoon et alii (2003).

SMI expression	XML Schema definition
IpAddress : : = OCTET STRING (SIZE (4)) (4))	<xsd:simpleType name = "IpAddress"> <xsd:restriction base = "xsd:string"> < xsd:pattern value = " (([1 - 9] ?[0- 9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([1-9]?[0 - 9] | 1 [0 - 9] [0 - 9]| 2 [0 - 4] [0 - 9] | 25 [0 - 5]) " / > </xsd:restriction> </xsd:simpleType>
Counter : : = INTEGER (0 .. 4294967295)	<xsd:simpleType name = "Counter"> <xsd:restriction base = "xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>
TimeTicks : : = INTEGER (0 .. 4294967295)	<xsd:simpleType name = "TimeTicks"> <xsd:restriction base = "xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>
Opaque : : = OCTET STRING	<xsd:simpleType name = "Opaque"> <xsd:restriction base = "xsd:string"> </xsd:restriction> </xsd:simpleType>

Figura 3.4 – Exemplo de conversão de SMI versão 1 para XML *Schema*.

Figura retirada do artigo de Yoon et alii (2003).

A figura 3.5 apresenta o resultado da conversão das expressões da SMI versão 2 para XML *Schema* com a utilização do algoritmo definido por Yoon et alii (2003).

SMI expression	XML Schema definition
Counter64 ::= INTEGER (0 .. 18446744073709551615)	<xsd:simpleType name = "Counter64"> <xsd:restriction base = "xsd:unsignedLong"> </ xsd:restriction> </xsd:simpleType>
Gauge32 :: = INTEGER (0 .. 4294967295)	<xsd:simpleType name = "Gauge32"> <xsd:restriction base = "xsd:unsignedInt"> </ xsd:restriction> </xsd:simpleType>
Unsigned32 ::= INTEGER (0 .. 4294967295)	<xsd:simpleType name = "Unsigned32"> <xsd:restriction base = "xsd:unsignedInt"> </ xsd:restriction> </xsd:simpleType>

Figura 3.5 – Exemplo de conversão de SMI versão 2 para XML *Schema*.

Figura retirada do artigo de Yoon et alii (2003).

A figura 3.6 apresenta o resultado da conversão de algumas definições da MIB-II para XML *Schema* com a utilização do algoritmo definido por Yoon et alii (2003).

MIB definition	XML Schema definition	Notes
Enumerated INTEGERs (ex) : SYNTAX INTEGER { Up (1) Down (2) Testing (3) }	<xsd:complexType> <xsd:simpleContent> <xsd:restriction base = "xsd:int"> <xsd:enumeration value = "1" /> <xsd:enumeration value = "2" /> <xsd:enumeration value = "3" /> </xsd:restriction> </xsd:simpleContent> </xsd:complexType>	Have one value among listed values.
DisplayString (SIZE (0 .. 255))	<xsd:simpleType name = "DisplayString_0_255"> <xsd:restriction base = "xsd:string"> <xsd:minLength value = "0" /> <xsd:maxLength value = "255" /> </xsd:restriction> </xsd:simpleType>	Assignment of string length.

Figura 3.6 – Exemplo de conversão de definições da MIB-II para XML *Schema*.

Figura retirada do artigo de Yoon et alii (2003).

A tese de Choi (2004) intitulada *An architectural framework for XML-based network management*, propõe a elaboração de uma arquitetura de gerenciamento de redes baseada em XML e apresenta na página 37, uma tabela de conversão de estruturas de dados, indicando como os elementos do padrão SMI são transformados em elementos do XML *Schema*. A tabela 3.1 indica os relacionamentos entre os elementos da SMI e do XML *Schema*, e também apresenta como relacionar os elementos do XML *Schema* com os elementos de acesso de informações do DOM. O algoritmo de conversão utilizado por Choi (2004) é o proposto no artigo de Yoon et alii (2003).

Tabela 3.1 – Conversão entre estruturas de representação de dados.

Tabela retirada do artigo de Choi (2004).

SNMP SMI	XML Schema	DOM Interface
MIB Module	XML document	Document
MIB Module name	Root Element name	Element::tagName
Leaf Node (macro definition)	element with child text node(s)	Element
Node name	element name	Element::tagName
Clauses of MIB node	Attributes of an element	Attr
Object Identifier (OID)	Attribute of type "ID"	Attr

A figura 3.7 apresenta um exemplo de conversão de dados da variável *sysUpTime* da MIB-II. Pode-se verificar na figura 3.7, que o nome da variável, neste caso *sysUpTime*, foi traduzida como nome do elemento de tipo complexo e que as palavras reservadas **SYNTAX**, **ACCESS** e **STATUS**, foram traduzidas como nome de atributos. Além disso, o algoritmo prevê a inclusão de um atributo adicional para identificar o *sysObjectId*, que é um número identificador de cada variável da MIB-II e que no exemplo da figura 3.7 recebe o nome de

oid. A justificativa para inclusão do *oid* se deve à facilidade de busca que este identificador pode propiciar e à grande utilização deste identificador em aplicações de gerenciamento.

MIB II	Translation Result
sysUpTime OBJECT-TYPE **SYNTAX** TimeTicks **ACCESS** read-only **STATUS** mandatory DESCRIPTION "The time..." ::= { system 3 }	<xsd:element name="**sysUpTime**"> <xsd:complexType> <xsd:simpleContent> <xsd:restriction base="TimeTicks"> <xsd:attribute name="**oid**" type="xsd:string" use="fixed" value="1.3.6.1.2.1.1.3"/> <xsd:attribute name="**access**" type="xsd:string" use="fixed" value="read-only"/> <xsd:attribute name="**status**" type="xsd:string" use="fixed" value="mandatory"/> <xsd:attribute name="**description**" type="xsd:string" use="fixed" value= "The time....."/> </xsd:restriction> </xsd:simpleContent> </xsd:complexType> </xsd:element>

Figura 3.7 – Conversão do objeto *sysUpTime* da MIB II para XML.

Figura retirada do artigo de Choi (2004).

De todos os estudos apresentados neste sub-item e que são referentes à conversão de dados SMI para XML, verifica-se que o trabalho desenvolvido por Choi (2004) possui um detalhamento aprofundado no processo de conversão de dados entre as duas estruturas e é utilizado, de forma simplificada – sem o mapeamento preciso entre variáveis da MIB e o documento XML - no presente trabalho.

3.4 Transferência de Dados

Existem várias tecnologias que podem ser utilizadas para efetuar a transferência dos dados entre o elemento gerenciado e o NMS. Nesta parte do trabalho será feita uma breve explanação das principais tecnologias disponíveis.

3.4.1 Datagramas

Os datagramas são mensagens que podem ser enviadas através de uma rede de uma aplicação para outra, com a finalidade de transmitir informações. Tradicionalmente os datagramas utilizam o protocolo de transporte UDP e tem como características principais:

- Sem garantia de entrega;
- Sem controle de fluxo;
- Encaminhamento de mensagens fora de ordem;

- Sem garantia de tempo de entrega;
- Sem estabelecimento de sessão;
- Rapidez na entrega.

A linguagem de programação Java disponibiliza em seu pacote padrão *java.net* a classe *DatagramPacket*. No entanto, devido à necessidade da monitoração do cluster de servidores exigir a coleta de dados de desempenho com garantia de entrega essa opção de comunicação foi descartada.

3.4.2 *Socket*

Um *socket* é uma conexão fim-a-fim que é estabelecida entre os parceiros da comunicação, que no contexto da aplicação de monitoração correspondem respectivamente ao agente e ao NMS. Na utilização de uma comunicação baseada em *socket* o agente faria o papel de cliente e o NMS o papel de servidor. O cliente é a entidade que solicita serviços ao servidor e que utiliza no processo de comunicação o protocolo de transporte TCP. A comunicação por *socket* tem como base a identificação da sessão de comunicação por meio da associação de uma porta a um endereço IP em cada lado da comunicação, como pode ser verificado no diagrama esquemático da figura 3.8.

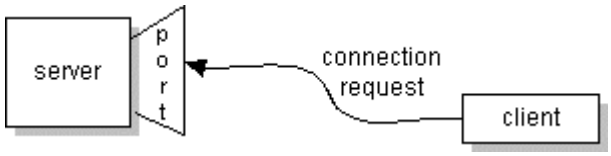

Figura 3.8 – Conexão do cliente ao servidor.

Figura retirada do artigo da SUN Microsystems (2005).

A conexão por *socket* tem como principais características:

- Garantia de entrega;
- Controle de fluxo;
- Encaminhamento de mensagens em ordem;
- Garantia de tempo de entrega;
- Estabelecimento de sessão;
- Entrega um pouco mais lenta do que no datagrama.

A linguagem de programação Java disponibiliza em seu pacote padrão *java.net* as classes *DatagramSocket e MulticastSocket*. Apesar da comunicação por *socket* possuir garantia de entrega e controle de fluxo, ela é considerada uma solução rudimentar, pois não implementa, em suas classes principais, facilidades nativas que permitam o controle da comunicação entre o programa cliente e o programa servidor, ficando a cargo do programador o desenvolvimento de programas complementares que viabilizem o controle da comunicação. Dessa forma, a utilização da comunicação por *socket* para a aplicação de monitoração também foi descartada.

3.4.3 RMI

RMI é a sigla para *Remote Method Invocation* e é uma API desenvolvida pela *Sun Microsystems* para permitir que objetos que estão sendo executados em uma determinada JVM, efetuem chamadas a objetos que estão sendo executados em outra JVM e solicitem a execução de procedimentos no servidor remoto. O servidor remoto executa o procedimento solicitado e retorna os dados para o cliente solicitante, conforme Horstmann e Cornell (2001). A figura 3.9 ilustra a invocação de objetos remotos armazenados em um servidor de objetos com a utilização de RMI e o recebimento do resultado produzido pelo objeto.

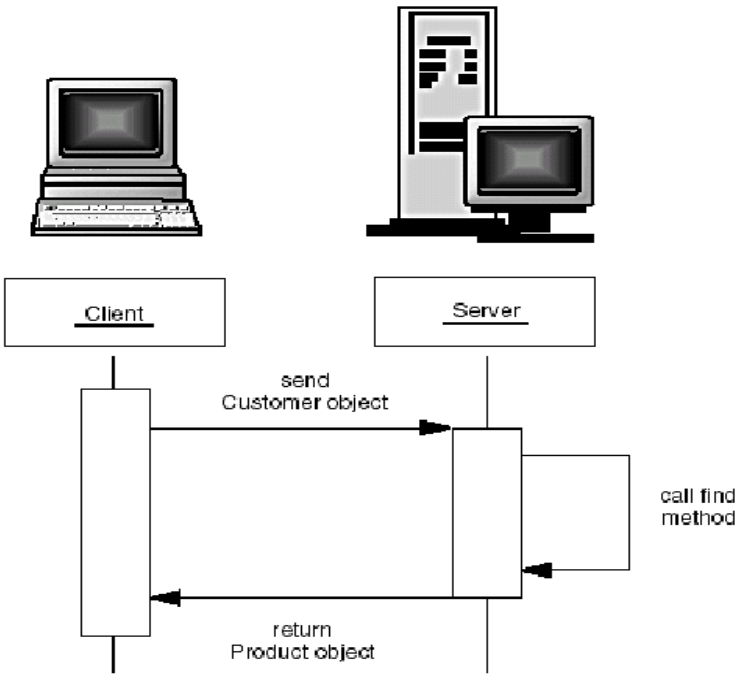

Figura 3.9 – Invocando um método remoto em um servidor de objetos.

Figura retirada de Horstmann e Cornell (2001).

A API RMI permite a comunicação entre aplicações clientes e servidoras escritas na linguagem de programação Java e, por padrão, utiliza o protocolo proprietário de comunicação JRMP (*Java Remote Method Protocol*). Caso seja necessária a comunicação entre um programa cliente escrito em Java e um programa servidor escrito em C++, deve ser

utilizada a API RMI com o protocolo IIOP (*Internet Inter-ORB Protocol*), conforme especificado pela OMG (*Object Management Group*) dentro do padrão CORBA (*Common Object Request Broker Architecture*), conforme Horstmann e Cornell (2001).

A figura 3.10 ilustra a comunicação direta e as possibilidades de interoperabilidade entre clientes e servidores desenvolvidos com o padrão RMI, com o padrão CORBA e um modelo híbrido com RMI-IIOP.

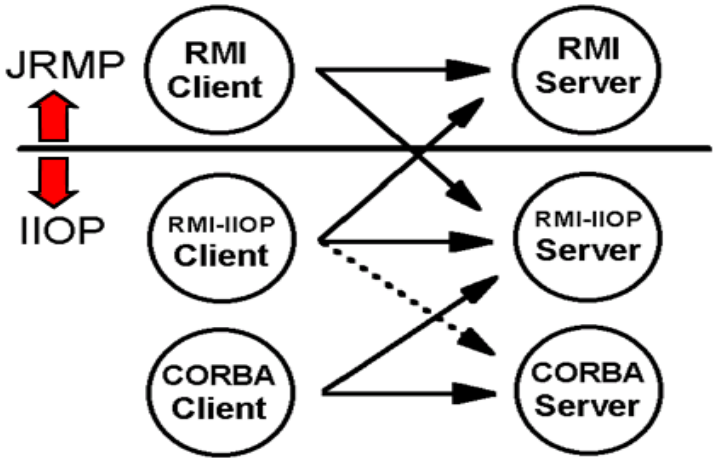

Figura 3.10 – Interoperabilidade entre RMI e CORBA com IIOP.

Figura retirada de Andoh e Nash (1999).

Devido às características da API RMI indicarem a utilização de aplicações escritas apenas em Java ou opcionalmente o uso conjunto com o protocolo IIOP, a utilização da API RMI para comunicação entre os elementos gerenciados e o NMS foi desconsiderada como opção adequada e universal para o desenvolvimento deste trabalho, que propõe a comunicação entre programas escritos em diferentes linguagens de programação.

3.5 O Padrão SOAP

O *Simple Object Access Protocol* (SOAP) é um protocolo simples para troca de mensagens entre aplicações em ambientes distribuídos. O SOAP teve sua especificação iniciada em 1998, por um grupo de representantes de empresas fabricantes de software (Microsoft, DevelopMentor e UserLand *Software*) e em 1999 foi submetido ao W3C a solicitação de padronização da especificação 1.0 do SOAP. A versão atual do SOAP é a versão 1.2 com o *Working Draft* 2, conforme Hendricks (2002).

A especificação atual do SOAP é baseada em XML e tem a sua mensagem dividida em 3 partes, conforme W3C (2003):

- O envelope - serve para descrever o conteúdo da mensagem;

- O *header* - opcional e é utilizado para passar informações adicionais de controle da mensagem;

- O *body* - parte da mensagem SOAP reservada para carregar a carga de dados útil.

O diagrama esquemático da mensagem SOAP pode ser verificado na figura 3.11.

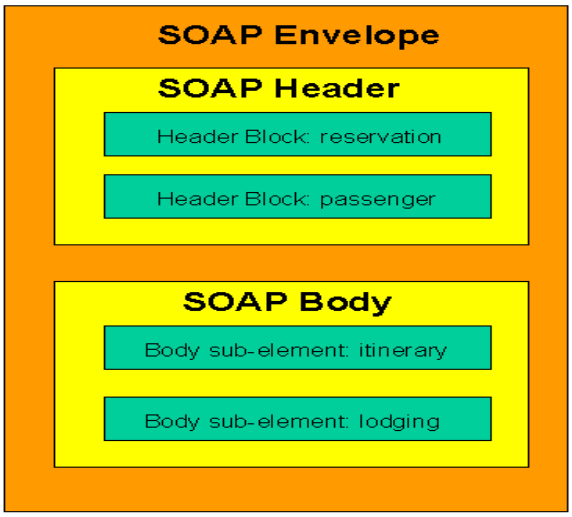

Figura 3.11 – Formato da mensagem SOAP.

Figura retirada do artigo do W3C (2003).

As características principais do protocolo SOAP, conforme definido em Hendricks et alii (2002) são:

- O SOAP é um protocolo superficial e simples, pois tudo o que é necessário é que os dados de XML passem pelo do HTTP. Ele não tenta definir um modelo de programação ou implementar API´s específicas.

- O SOAP não é vinculado a nenhum protocolo de transporte em particular. Essa instrução deve estar clara a partir da pilha de serviços *Web*, pois a camada de troca de mensagens XML é construída no topo da camada da rede de transporte. É possível deduzir que o SOAP equivale a um XML que opera por intermédio do HTTP, o que é perfeitamente razoável.

O SOAP foi definido dentro do paradigma consulta e resposta, sendo que a cada mensagem SOAP de solicitação corresponde uma mensagem SOAP de resposta, conforme podemos verificar no diagrama 3.12.

Figura 3.12 – Troca de mensagens SOAP.

Mack Hendricks et alii (2002) em seu livro Profissional Java *Web Service* relacionam as principais vantagens e desvantagens na utilização do protocolo SOAP, e as principais vantagens são:

- O SOAP pode atravessar *firewalls*;
- Os dados do SOAP são estruturados usando a especificação XML;
- O SOAP pode ser usado, potencialmente, em combinação com vários protocolos de transporte, como HTTP, SMTP e (*Java Messaging Server*) JMS;
- O SOAP mapeia satisfatoriamente para o padrão de solicitação/resposta HTTP e do HTTP *Extension Framework*;
- O SOAP é razoavelmente superficial como um protocolo, pois apenas define mensagens de controle para envio e recepção de documentos XML;
- Existe suporte para SOAP, por parte de vários fornecedores, incluindo a Microsoft, a IBM e a SUN.

As principais desvantagens relacionadas por Mack Hendricks et alii (2002) são:

- Falta de interoperabilidade entre *toolkits* do SOAP;
- Mecanismos de segurança são imaturos;
- Não existe garantia quanto à entrega da mensagem;
- Não existe publicação e nem assinatura.

Apesar das desvantagens descritas no parágrafo anterior, a especificação SOAP faz parte da arquitetura de construção de aplicativos baseados no modelo de *Web Services*, conforme definido pelo grupo OASIS e disponível no sítio http://www.oasis-open.org, conforme Oasis (2005). O diagrama básico de composição das camadas da arquitetura de um *Web Services* está representando na figura 3.13. Nesta figura pode-se verificar a implementação dos protocolos em camadas complementares, sendo que o presente trabalho especifica e implementa um aplicativo de monitoração que compreende as camadas de transporte e tratamento de mensagens XML. As camadas de descrição de serviços e publicação e descobrimento de serviços não são implementadas no presente trabalho.

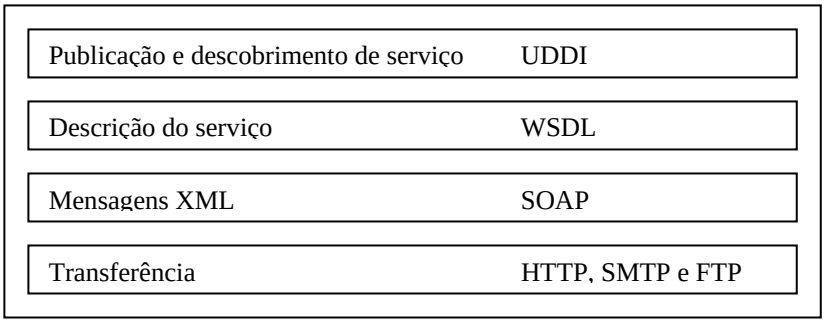

Figura 3.13 – Camadas do *Web Service*.

3.6 Estado Atual

3.6.1 Propostas Baseadas em Conversores

Uma proposta de implementação de uma arquitetura de gerenciamento de redes baseada em XML foi elaborada pela equipe do POSTECH, conforme Yoon et alii (2003) e define a configuração de um NMS XML nativo (não ocorre o processo de conversão de dados de SNMP para XML) que se comunica com agentes XML nativos (não ocorre o processo de conversão de dados de SNMP para XML) de forma direta. No caso em que seja necessária a comunicação com elementos gerenciados que tem instalado agentes SNMP, faz-se necessária a comunicação do NMS XML nativo com um NMS XML-SNMP *gateway* intermediário, que faz a tradução das mensagens XML para mensagens SNMP e as encaminha para o agente SNMP nativo. Quando a mensagem SNMP retorna do agente SNMP, ela é recebida pelo agente XML-SNMP *gateway*, que faz a conversão da mensagem SNMP para XML e a encaminha para o NMS XML, conforme pode ser verificado na figura 3.14.

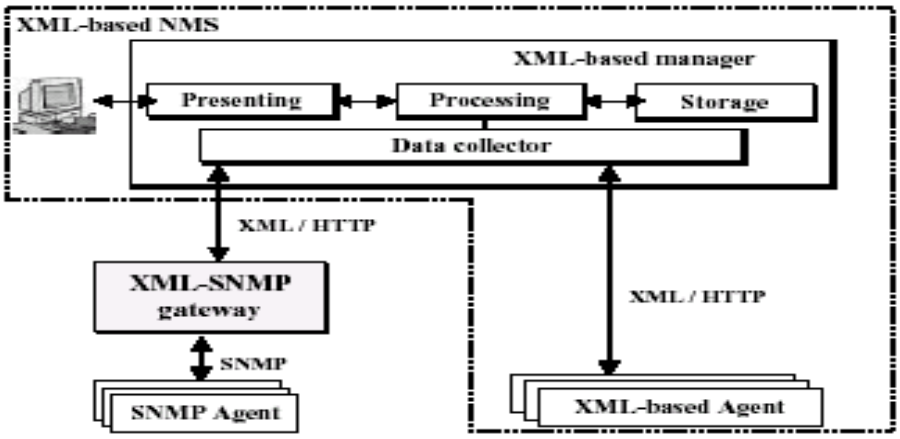

Figura 3.14 – Comunicação baseada no uso de conversores

Figura retirada do artigo de Yoon et alii (2003).

3.6.2 Propostas Baseadas em Comunicação Nativa

Uma proposta de implementação de uma arquitetura de gerenciamento de redes baseada em XML nativo com comunicação direta por HTTP com agentes XML nativo foi elaborada pela equipe do POSTECH, conforme Ju et alii (2002), anteriormente à elaboração da proposta de uma arquitetura híbrida, conforme Yoon et alii (2003). A justificativa levantada pela equipe do POSTECH para a não utilização de soluções XML nativa com agentes XML nativos foi à definição de que isso implica na substituição dos agentes SNMP de todos os elementos gerenciados da Internet por agentes XML nativos, pois a convivência simultânea dos dois agentes nos elementos gerenciados provoca um consumo de recursos computacionais muito grande. O diagrama básico da arquitetura de gerenciamento com XML nativo e agente nativo pode ser verificado na figura 3.15.

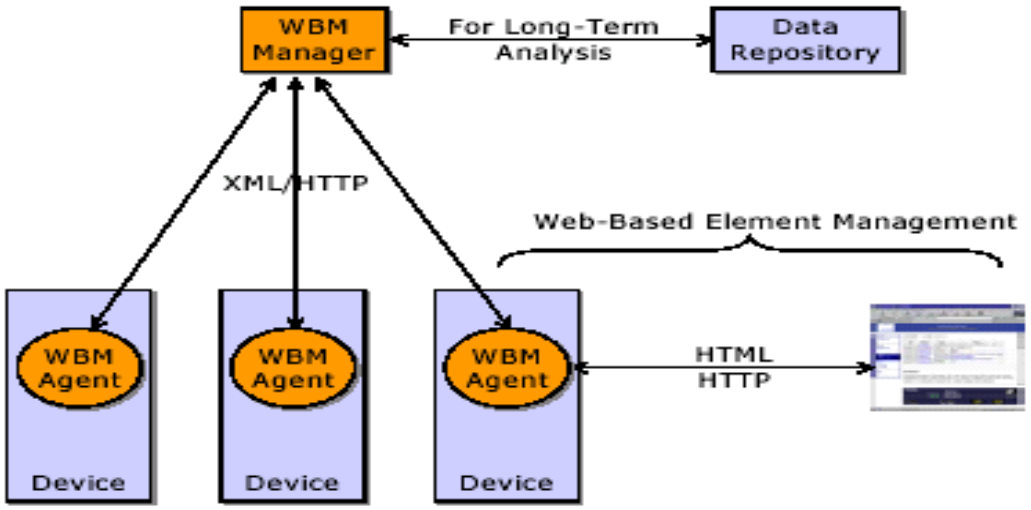

Figura 3.15 – Comunicação nativa

Figura retirada do artigo de Ju et alii (2002).

Além da proposta elaborada pela equipe do POSTECH, conforme Ju et alii (2002), identifica-se uma implementação comercial da arquitetura com NMS XML nativo e agente XML nativo, efetuada pela empresa Juniper, conforme Strauß e Klie (2003). Nessa implementação, a Juniper desenvolveu uma plataforma de gerenciamento para monitoração dos equipamentos de sua fabricação, mas que ao mesmo tempo pode ser utilizada para monitoramento de elementos gerenciados de outros fabricantes que disponibilizem agentes XML nativos em seus equipamentos. A Juniper também disponibiliza gratuitamente em seu sítio na Internet, conforme JUNIPER (2004), alguns programas que podem ajudar outros fornecedores e centros de pesquisa no desenvolvimento de novas versões de softwares que venham a atuar como NMS ou agentes.

3.7 Conclusão

Nesse capítulo foram apresentados os principais conceitos dos modelos de monitoração de redes e servidores que fazem uso de tecnologias baseadas em padrões *Web*, com maior ênfase nos padrões HTTP, XML e SOAP. Também foram apresentadas as tecnologias de transferência de dados e as técnicas para conversão de estruturas da MIB e da SMI para XML *Schema*. É importante notar que a tendência de utilização de tecnologias *Web* para monitoração de redes e servidores se deve a alguns fatores principais, mas dentre esses fatores pode-se ressaltar a ubiqüidade que os padrões HTTP e XML apresentam nos *softwares* e *hardwares* dos mais diferentes fabricantes.

Entre as propostas de monitoração apresentadas, a utilização de conversores apresenta-se como uma opção para permitir a migração e a convivência da monitoração atual baseada na arquitetura SNMP para a monitoração baseada em HTTP/XML. Devido à arquitetura baseada em conversores já ter sido explorada em trabalhos desenvolvidos pela equipe da POSTECH, conforme Yoon et alii (2003), optou-se por elaborar no presente trabalho uma aplicação de monitoração baseada no modelo de comunicação nativa.

O próximo capítulo tratará da estrutura da aplicação de monitoração, apresentando os requisitos que devem ser atendidos, os artefatos que foram gerados durante o processo de estruturação e o documento XML que será utilizado pela aplicação de monitoração para armazenamento e transmissão dos dados coletados.

4 ESTRUTURA DA APLICAÇÃO DE MONITORAÇÃO

Neste capítulo é apresentada e detalhada a aplicação que foi desenvolvida com a utilização de padrões *WEB* (HTTP e XML) para a monitoração do desempenho do cluster de servidores.

4.1 Introdução

A especificação de padrões para a monitoração de servidores e elementos de rede com XML nativo foi elaborada pela equipe do POSTECH, conforme Ju et alii (2002), e descrito no capítulo anterior. Posteriormente, na tese de doutorado intitulada *An Architectural Framework for XML-based Network Management* , conforme Choi (2004), foi aprofundada a especificação de uma arquitetura para gerenciamento de redes baseada em XML. No trabalho de Choi (2004) são inicialmente apresentadas as quatro formas consideradas principais para o gerenciamento HTTP/XML de servidores e elementos de rede, conforme pode ser visto na figura 4.1 e que são:

- a) modelo de gerenciamento SNMP amplamente utilizado atualmente;
- b) NMS SNMP com elemento intermediário atuando como *gateway* para tradução de mensagens XML para SNMP;
- c) NMS baseado em XML com elemento intermediário atuando como *gateway* para tradução de mensagens SNMP para XML;
- d) NMS baseado em XML com comunicação nativa em HTTP/XML com o elemento gerenciado.

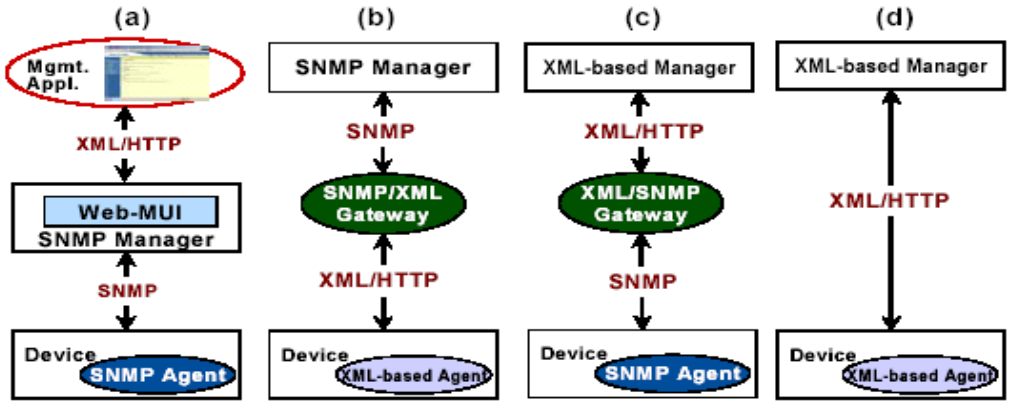

Figura 4.1 – Combinações de NMS e agente

Figura retirada da tese de Choi (2004).

Em seu trabalho, Choi (2004) propõe a utilização do modelo descrito na opção (c) da figura 4.1, e afirma que o modelo ideal para obter a vantagem máxima de uma arquitetura baseada em XML é conseguido com a utilização de NMS e agente com comunicação nativa

em HTTP/XML e, por esse motivo, a abordagem adotada para o presente trabalho é o desenvolvimento de um modelo que permita a implementação e a realização de testes de uma aplicação de monitoração com comunicação nativa em HTTP/XML.

4.2 Requisitos da Aplicação de Monitoração

A estrutura da aplicação de monitoração nativa HTTP/XML foi elaborada a partir da avaliação dos indicadores que deveriam ser monitorados e também por meio da comparação das principais características geralmente utilizadas em aplicativos de monitoração, sendo considerados os seguintes requisitos:

- Deve ser especificado um agente que possa ser instalado em servidores ou estações de trabalho com o sistema operacional Windows;
- O agente deve ser capaz de monitorar parâmetros de desempenho de utilização de memória, processador e rede;
- O agente deve ser extensível para permitir a configuração de outros parâmetros para coleta;
- O agente deve permitir a configuração do intervalo entre as coletas;
- O agente deve descobrir automaticamente informações referentes ao ambiente em que está instalado;
- O agente deve formatar a mensagem em formato XML e transmiti-la ao gerenciador por meio do uso do protocolo HTTP;
- O agente não permitirá o envio de mensagens que reportem condições de erros nos elementos gerenciados, similares ao mecanismo de envio de *traps* do protocolo SNMP;
- O NMS deve ser capaz de receber mensagens em formato XML de vários agentes;
- O NMS deve armazenar as mensagens recebidas localmente, em formato cronológico e preferencialmente em base de dados;
- O NMS deve permitir a geração de relatórios referentes aos dados armazenados em sua base de dados.

No diagrama esquemático simplificado da figura 4.2, está representada a estrutura da aplicação de monitoração com agente e NMS com comunicação nativa em XML/HTTP e armazenamento em base de dados.

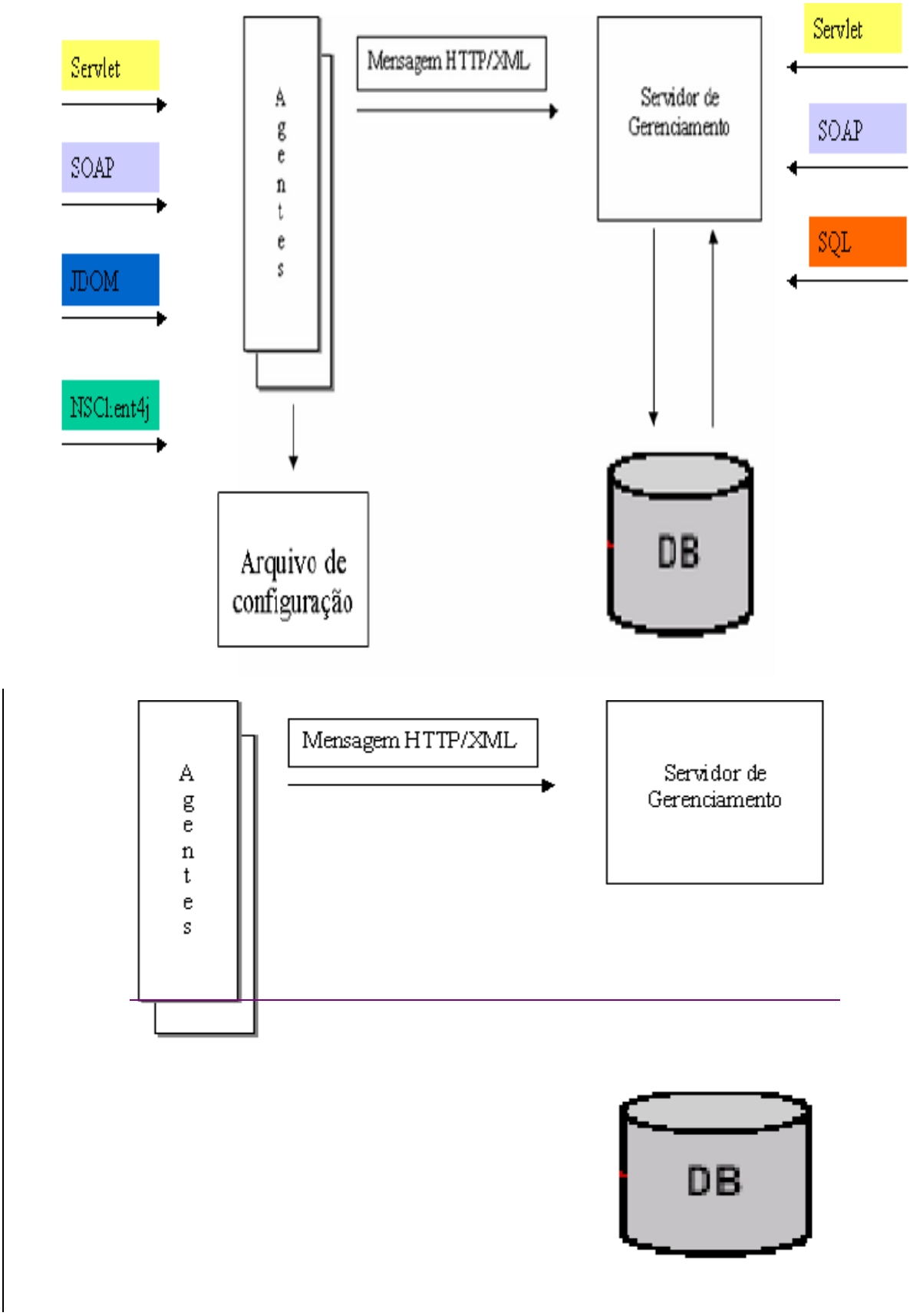

Figura 4.2 – Diagrama esquemático simplificado da arquitetura da aplicação

4.3 Modelagem com IDEF0

O padrão IDEF0 do NIST (1993) foi utilizado para efetuar a estruturação da aplicação. O IDEF0 é um método desenvolvido para criar modelos que representem as decisões, ações e atividades de um sistema e tem, como principal objetivo, permitir a análise de um sistema e promover uma boa forma de comunicação entre todas as pessoas que estão envolvidas com ele, seja o especialista de negócios, o analista ou o programador do sistema. O IDEF0 está publicado como padrão desde 1993 pelo *National Institute of Standards and Technology* por meio da publicação FIPS 183, conforme NIST (1993).

Na figura 4.3 está representada a primeira etapa da estruturação da aplicação com o padrão IDEF0 no nível 0, o qual corresponde a uma visão consolidada de todas as entradas, saídas, mecanismos e regras que compõem a aplicação. Para efeito de documentação, o processo global que representa toda a aplicação modelada, recebeu no nível 0 o nome de *XML Network Management Pattern* (XNMP), conforme pode ser verificado na figura 4.3.

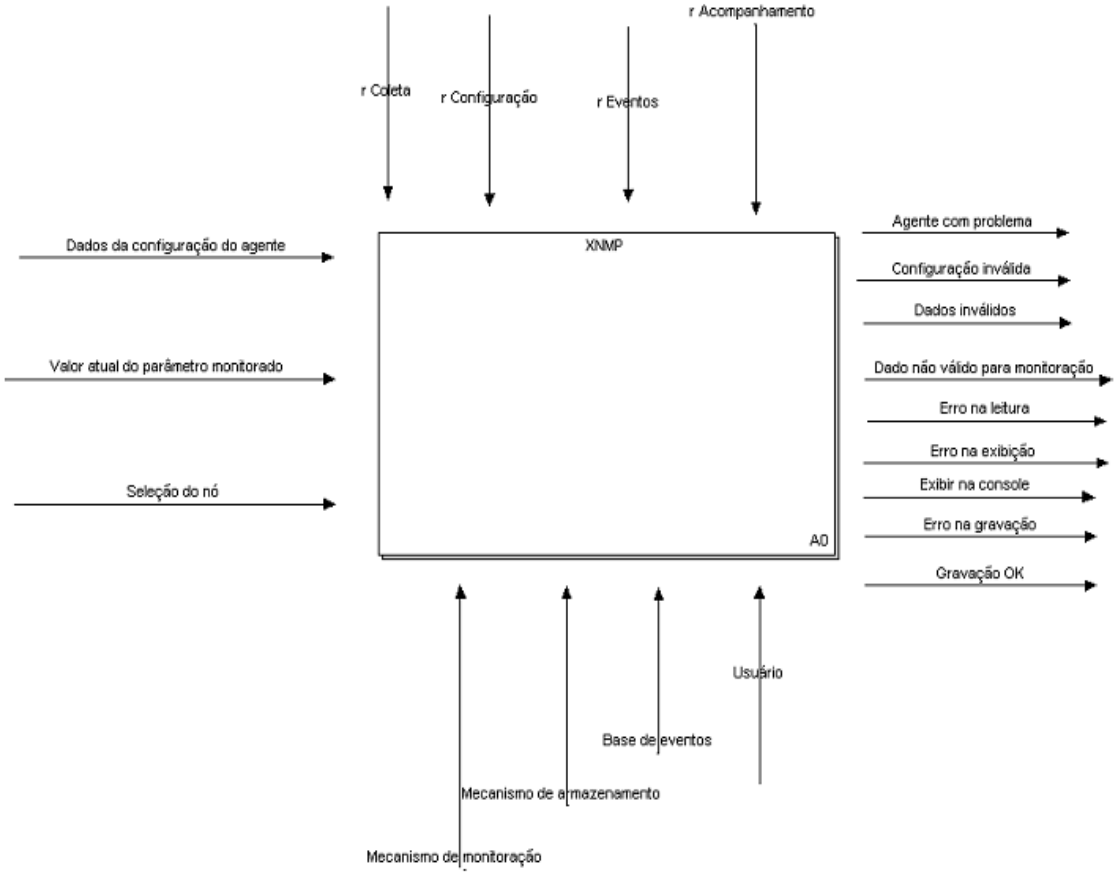

Figura 4.3 –IDEF0 (nível 0) - Aplicação de Monitoração

O próximo passo na estruturação da aplicação, por meio do uso do método IDEF0, é a especificação do nível 1, a partir do modelo definido no nível 0. Na criação do nível 1, são identificados quatro processos principais:

- Configuração do agente;
- Coleta;
- Tratamento de eventos;
- Acompanhamento da monitoração.

O diagrama esquemático do IDEF0 nível 1 está representado na figura 4.4. Neste diagrama são destacados os processos de Configuração do agente, Coleta e Tratamento de eventos, os quais são realizados de maneira contínua e integrada. Já o processo de Acompanhamento da monitoração utiliza as informações armazenadas na base de dados da aplicação de monitoração pelo processo de Tratamento de eventos. Dessa forma, o processo de Acompanhamento da monitoração pode ser iniciado a qualquer momento pelo operador da aplicação de monitoração, que utiliza o processo de Acompanhamento da monitoração para ter acesso aos dados coletados.

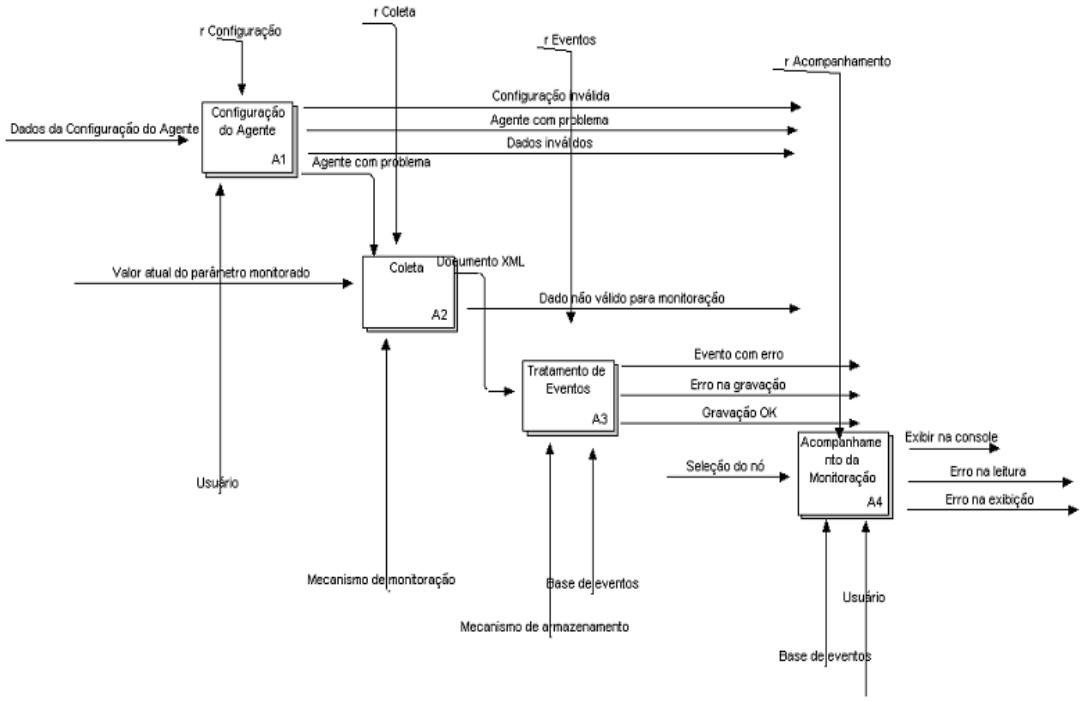

Figura 4.4 – IDEF0 (nível 1) - Aplicação de Monitoração

A partir do IDEF0 nível 1 é realizado o detalhamento de cada processo, com a geração dos diagramas IDEF0 nível2. O primeiro processo a ser analisado é o processo de Configuração do agente, que está representado na figura 4.5. O processo de Configuração do agente é dividido em:

- Levantamento de dados;
- Configuração do agente;
- Validação das alterações.

O processo de levantamento de dados deve ser realizado pelo usuário, a partir da utilização da ferramenta *Windows Performance Monitor*, disponível nos sistemas operacionais Windows XP, Windows 2000 ou Windows 2003. O levantamento de dados é necessário para ajustar as configurações do agente no arquivo de configuração. Identifica-se que em uma nova versão do software do agente, deverá ser prevista a existência de seleção automática de parâmetros de configuração, por meio de detecção automática da versão do sistema operacional.

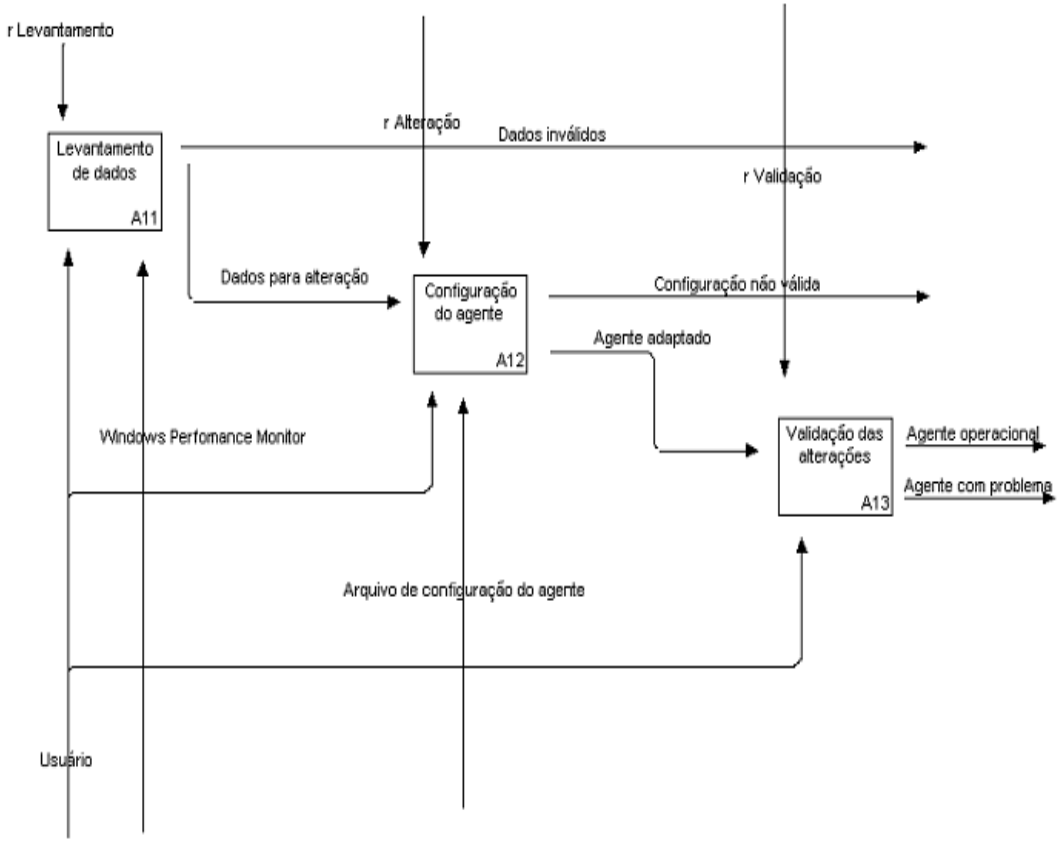

Figura 4.5 – IDEF0 (nível 2) – Configuração do Agente

O segundo processo detalhado no IDEF0 nível 2 é o processo de Coleta, que está representado na figura 4.6. O processo de Coleta no agente é dividido em:

- Coleta de dados;
- Formatação da mensagem.

A coleta de dados é realizada de acordo com o intervalo de tempo definido no arquivo de configuração, podendo ser ajustado para qualquer intervalo de tempo que o usuário especificar. Geralmente para evitar sobrecarga no processamento da aplicação principal, deve ser utilizado um intervalo de tempo de 30 segundos. No arquivo de configuração também estão relacionados os parâmetros que serão monitorados. Não foi especificada a utilização de limitação de envio de mensagens baseada em valores mínimos que os parâmetros de coleta devem possuir, mas a possibilidade de configuração de valores mínimos pode ser disponibilizada em uma nova versão. Após a realização da coleta, é efetuada a formatação da mensagem em padrão XML.

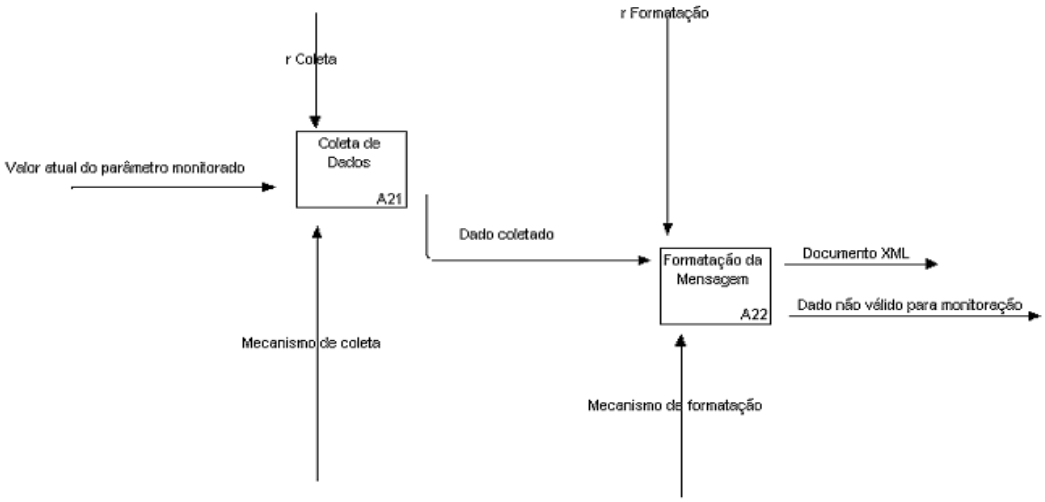

Figura 4.6 – IDEF0 (nível 2) – Coleta

O terceiro processo detalhado no IDEF0 nível 2 é o processo de Tratamento de eventos, que está representado na figura 4.7. O processo de Tratamento de eventos é executado no NMS e é dividido em:

- Recepção do evento;
- Gravação do evento.

O NMS recebe a mensagem enviada pelo agente que corresponde a um documento XML e trata esta mensagem como um evento, efetuando a validação do documento XML. Se o documento é válido, ele é encaminhado para o processo de gravação do evento, o qual faz uma solicitação de gravação dos dados constantes no documento XML na base de dados. Se o documento não é válido é gerado uma mensagem de erro e o documento não é gravado na base de dados.

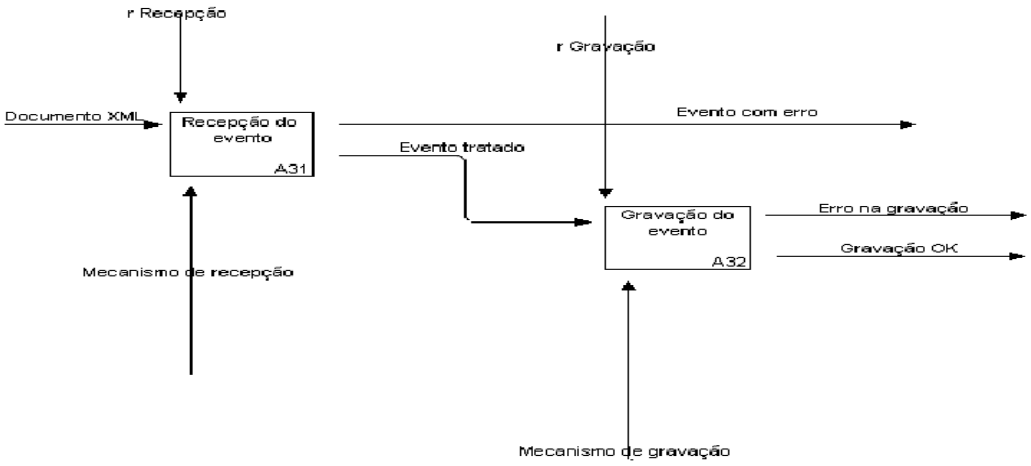

Figura 4.7 – IDEF0 (nível 2) – Tratamento de Eventos

O quarto processo detalhado no IDEF0 nível 2 é o processo de Acompanhamento da monitoração, que está representado na figura 4.8. O processo de Acompanhamento da monitoração de eventos é executado no NMS e é dividido em:

- Leitura de dados;

- Exibição de dados.

O processo de Acompanhamento da monitoração é o único processo que não é executado de forma interligada ao processo de monitoração. O processo de Acompanhamento da monitoração pode ser executado a qualquer momento pelo usuário e este processo efetuará a leitura das informações armazenadas na base de dados, relacionadas a um dos elementos monitorados. Para iniciar a utilização dessa funcionalidade, o usuário deve selecionar o elemento monitorado que deseja consultar os dados e as informações referentes aos valores dos parâmetros monitorados serão mostradas na tela do usuário em formato gráfico. Caso ocorra algum problema no processo de leitura dos dados, uma mensagem de erro será exibida para o usuário.

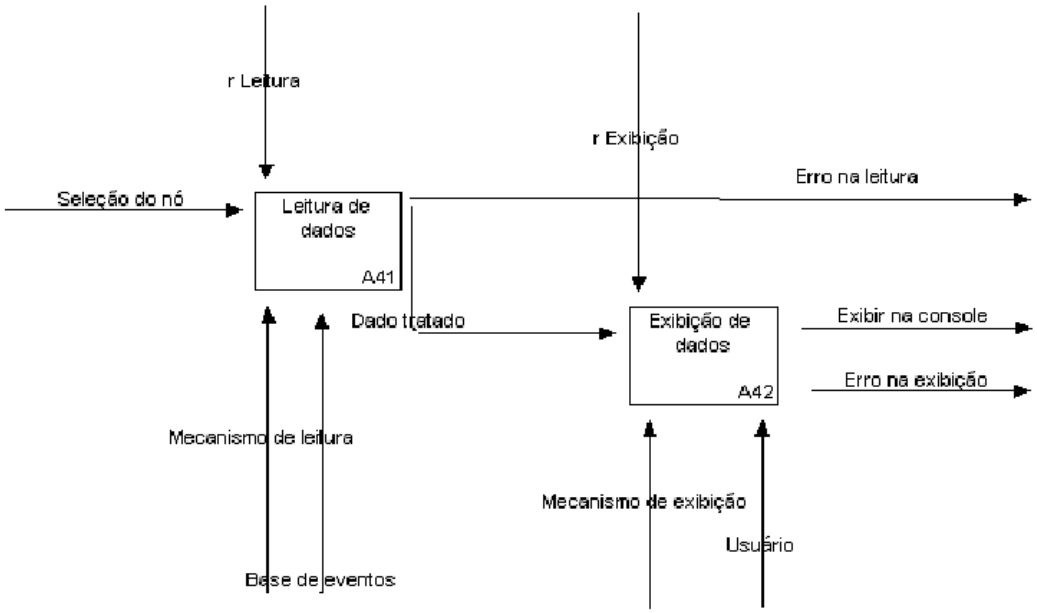

Figura 4.8 – IDEF0 (nível 2) – Acompanhamento da Monitoração

4.4 Principais Classes da Aplicação de Monitoração

A partir da definição do modelo de funcionamento da aplicação de monitoração, com o uso da metodologia IDEF0, efetuou-se a diagramação das classes que comporão a aplicação de monitoração com a utilização da notação Unified Modeling Language (UML). Como a aplicação de monitoração será instalada tipicamente em duas categorias de computadores – aplicação de monitoração atuando como agente de coleta e aplicação de monitoração atuando como NMS da coleta – optou-se por agrupar as classes em dois pacotes:

- pacote *Manager*;
- pacote *Agent*.

O pacote *Manager* é composto das seguintes classes:

- *Manager*;
- *GraphFrame*;
- *EditManagerFrame*;
- *Data.*

A classe *Manager* contém a cláusula *main* e por esse motivo é o ponto de entrada para utilização da aplicação de monitoração pelo usuário e está representada na figura 4.9. A classe *Manager*, ao ser executada, solicita a execução automática das classes *GraphFrame* e *EditManagerFrame*.

Figura 4.9 – Classe *Manager*

As classes *GraphFrame* e *EditManagerFrame* são responsáveis por disponibilizar a interface gráfica para o usuário. A partir da tela monitoração, o usuário pode selecionar o nó do qual deseja verificar as informações coletadas, que serão exibidas em uma tela adicional. As classes *GraphFrame* e *EditManagerFrame* estão representadas na figura 4.10.

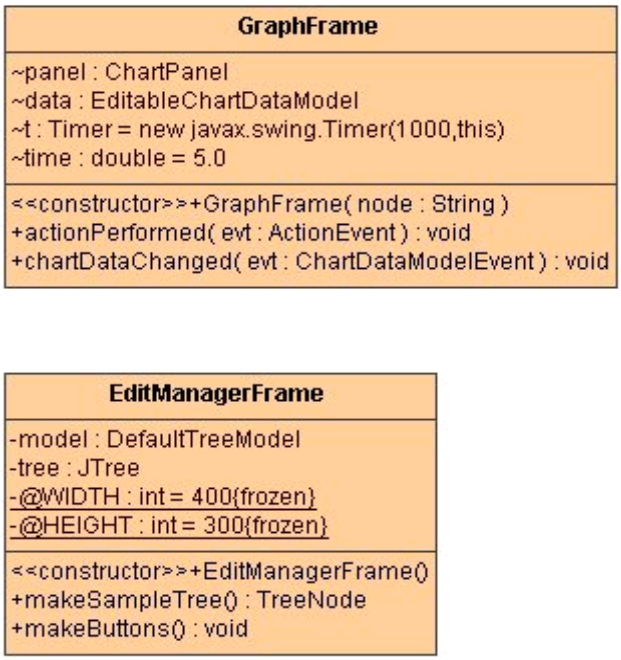

Figura 4.10 – Classes *GraphFrame* e *EditManagerFrame*

Após o usuário ter selecionado um nó do *cluster* para verificar as informações de desempenho, a classe *GraphFrame* aciona a classe *Data* para efetuar a leitura das informações referentes ao nó selecionado, as quais estão armazenadas na base de dados de eventos. A classe *Data* está representada na figura 4.11.

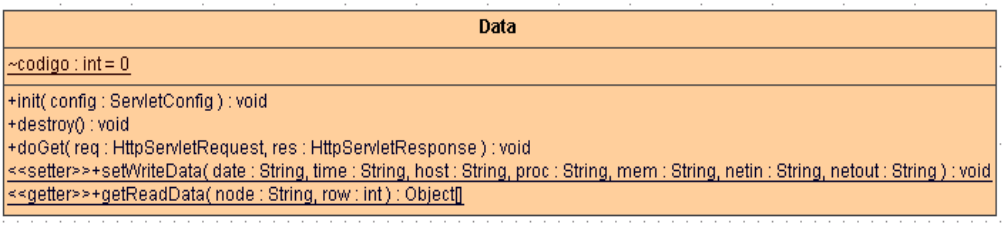

Figura 4.11 – Classe *Data*

O processo de recepção de dados no NMS, a partir de mensagens XML enviadas pelos agentes instalados nos nós monitorados, é realizado de maneira independente ao processo de monitoração da aplicação acionado pelo usuário. Dessa forma, para viabilizar a

recepção e a gravação dos dados, foram definidos os métodos *doGet* e *setWriteData* na classe *Data*. O método *doGet*, representado na figura 4.11, é responsável por receber mensagens XML por meio de uma conexão HTTP com os agentes e solicitar a execução do método *setWriteData*. O método *setWriteData* é responsável por receber os dados repassados pelo método *doGet* e efetuar a gravação no banco de dados.

O pacote *Agent* é composto das seguintes classes:

- *Agent*;
- *Timerizer*;
- *RemindTask*;
- *Mem*;
- *Proc*;
- *Net;*
- *Data.*

A classe *Agent* contém a cláusula *main* e por esse motivo é o ponto de entrada para utilização da aplicação de monitoração nos nós que serão gerenciados e está representada na figura 4.12. A classe *Agent*, ao ser executada, solicita a execução automática das classes *Timerizer* e *RemindTask*.

Figura 4.12 – Classe *Agent*

As classes *Timerizer* e *RemindTask* são responsáveis por controlar a execução temporizada das coletas de dados nas variáveis de monitoração do nó monitorado. As duas classes estão representadas na figura 4.13.

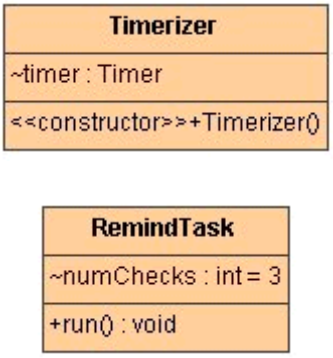

Figura 4.13 – Classes *Timerizer* e *RemindTask*

As classes *Mem*, *Proc* e *Net* são acionadas pela classe *Timerizer* para executarem, respectivamente, a leitura das informações de utilização de memória, utilização de processador e utilização de entrada e saída da placa de rede. As informações coletadas são transformadas para valores percentuais, quando não estão neste formato. As classes *Mem*, *Proc* e *Net* são representadas na figura 4.14.

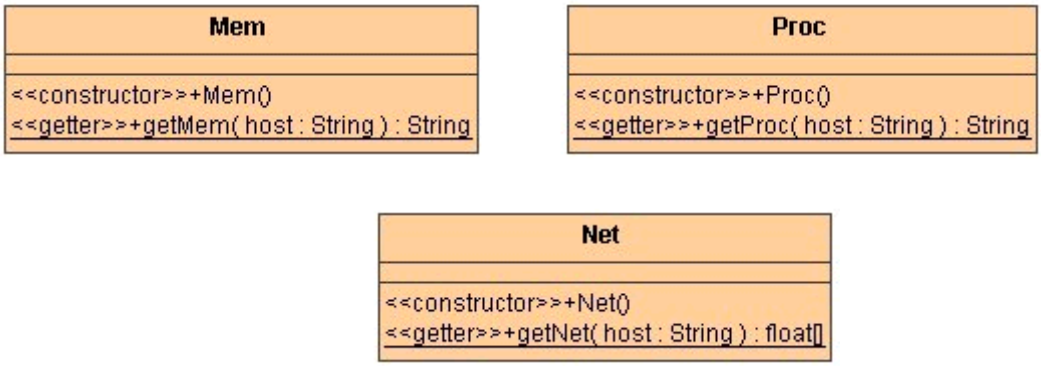

Figura 4.14 – Classes *Mem*, *Proc* e *Net*

As informações que são obtidas pela classe *Net* precisam de conversão de tipos de variável e por esse motivo foi criado o método *getStringToFloat* na classe Data. O método *setXmlToFile* da classe *Data* é responsável por efetuar a criação do documento XML e a gravação do mesmo em arquivo local no nó gerenciado. O arquivo local é sobrescrito toda a vez em que é efetuada uma nova coleta de dados, não sendo utilizado para fins de acompanhamento histórico. Os métodos *setXmlToFile* e *getStringToFloat* estão representadas na figura 4.15.

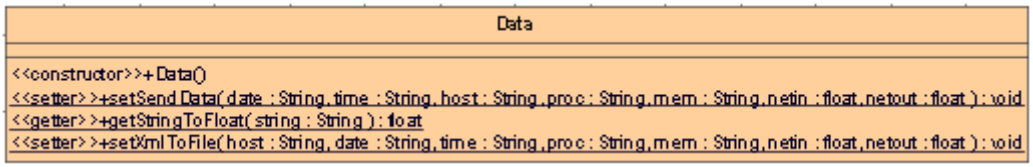

Figura 4.15 – Classe *Data*

Após os dados coletados serem formatados em um documento XML, o método *setSendData* da classe *Data* é acionado para fazer a transmissão do documento XML para o NMS, que receberá o documento e efetuará o armazenamento em banco de dados.

4.5 O Documento XML

As informações de desempenho coletadas nos nós que compõem o *cluster* de servidores monitorados são armazenadas no formato de documentos XML. Devido às informações coletadas não estarem presentes em apenas um grupo da MIB, evitou-se, no presente trabalho, a conversão de vários grupos da MIB para formação do documento XML, pois dessa forma haveria no documento XML mais informações do que necessário, o que poderia comprometer o desempenho da aplicação. Por esse motivo, optou-se por uma abordagem simplificadora e o XML *Schema* utilizado neste trabalho é o seguinte:

```
<xsd:element name=”node”>
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:attribute name=”host” type=”xsd:string”/>
      <xsd:attribute name=”date” type=”xsd:date”/>
      <xsd:attribute name=”time” type=”xsd:time”/>
      <xsd:attribute name=”proc” type=”xsd:string”/>
      <xsd:attribute name=”mem” type=”xsd:string”/>
      <xsd:attribute name=”netin” type=”xsd:decimal”/>
      <xsd:attribute name=”netout” type=”xsd:decimal”/>
    </xsd:simpleContent>
  …</xsd:complexType>
  </xsd:element>
```

Na figura 4.16 está representado o modelo padrão do documento XML que está sendo gerado pela aplicação de monitoração, inclusive com alguns dados coletados que servem como ilustração das informações que são obtidas.

```
<?xml version="1.0" encoding="UTF-8" ?>
<node host="10.1.17.188" date="31/03/2005" time="14:30"
    proc="0.00000" mem="70.62446" netin="0.009982789"
    netout="0.008362636" />
```

Figura 4.16 – O Documento XML

Cada *tag* constante da figura 4.16 tem um valor definido para assinalamento que são:

- *node*: identifica que este documento está associado à representação dos dados coletados em um nó monitorado;

- *host*: armazena o endereço IP do nó monitorado;

- *date*: data em que a coleta foi realizada;

- *time*: hora em que a coleta foi realizada;

- *proc*: armazena o valor percentual da utilização de processador;

- *mem*: armazena o valor percentual da utilização de memória;

- *netin*: armazena o valor percentual da utilização de dados de entrada da placa de rede;

- *netout*: armazena o valor percentual da utilização de dados de saída da placa de rede.

4.6 Conclusão

Nesse capítulo foram apresentados os principais requisitos que o sistema de monitoração deve atender e os artefatos desenvolvidos durante o processo de estruturação da aplicação de monitoração. O processo de estruturação gerou dois artefatos:

- Diagramas IDEF0;

- Classes da aplicação de monitoração.

Além desses artefatos o capítulo contou também com a descrição do formato do documento XML que será transmitido para o NMS e também utilizado para armazenar, de forma temporária, os dados de monitoração no elemento gerenciado.

O próximo capítulo tratará dos aspectos relativos a codificação e implementação da aplicação de monitoração.

5 IMPLEMENTAÇÃO DA APLICAÇÃO DE MONITORAÇÃO

Neste capítulo são apresentadas as características de implementação da aplicação de monitoração de servidores, que foi desenvolvida com a utilização da linguagem de programação Java.

5.1 Introdução

Várias linguagens de programação têm sido utilizadas para o desenvolvimento de *softwares* para utilização como agentes, em elementos gerenciados e para utilização como NMS, em servidores centrais de gerenciamento ou NMS. Dentre as principais linguagens utilizadas para o desenvolvimento desses *softwares* pode-se destacar a utilização predominante das linguagens C, C++ e Java. Para a realização do desenvolvimento da aplicação de monitoração de servidores, conforme proposto neste trabalho, foi adotado o uso da linguagem de programação Java. Neste capítulo serão apresentados os principais conceitos e softwares necessários para a implementação da arquitetura da aplicação de monitoração, conforme detalhada no capítulo 4. A apresentação da aplicação está dividida em 4 seções:

- Coleta de Dados no Agente;
- Tecnologias para Transferência de Dados;
- Armazenamento de Dados no NMS;
- Visualização das Informações Coletadas.

5.2 Coleta de Dados no Agente

Uma grande dificuldade existente atualmente na área de monitoração de redes e sistemas com a utilização de novos padrões que não utilizem a arquitetura SNMP para coleta e transmissão dos dados coletados é justamente a falta de definição de uma arquitetura com os padrões HTTP/XML e que seja aceita pela comunidade acadêmica e adotada pelos fabricantes de *hardware* e *software* da indústria de informática. Por esse motivo, o desenvolvimento de soluções de *softwares* de monitoração, baseadas em padrões HTTP/XML, realizadas por instituições acadêmicas, comerciais ou por organizações de *software* livre, têm evoluído lentamente, conforme Strauß e Klie (2003).

5.2.1 API NSClient4j

Após a realização de várias pesquisas foi verificada a existência de uma solução de *software* livre que permite a monitoração de vários indicadores de desempenho de servidores e estações de trabalho que utilizem o sistema operacional Windows. O *software* que efetua este trabalho chama-se Nagios e está disponível no sítio http://www.nagios.org, conforme Nagios (2005). Na figura 5.1 é apresentado o diagrama esquemático de funcionamento de uma aplicação Nagios. No diagrama citado, o Nagios *Engine* está instalado em um sistema Linux, que atua como NMS e possui um arquivo inicial de configuração. A aplicação Nagios utiliza um processo chamado Checknt para comunicar-se com o serviço NSClient, que reside

no elemento monitorado. O serviço NSClient utiliza o *software Windows Performance Monitor* (WPM), disponível no sistema operacional Windows, para fazer acesso as informações de desempenho, que são coletadas diretamente pelo WPM no sistema operacional. As informações de desempenho coletadas pelo serviço NSClient são então transmitidas para o serviço Checknt, que as repassa para o Nagios *Engine* do NMS para geração de alertas e visualização em interface *Web*.

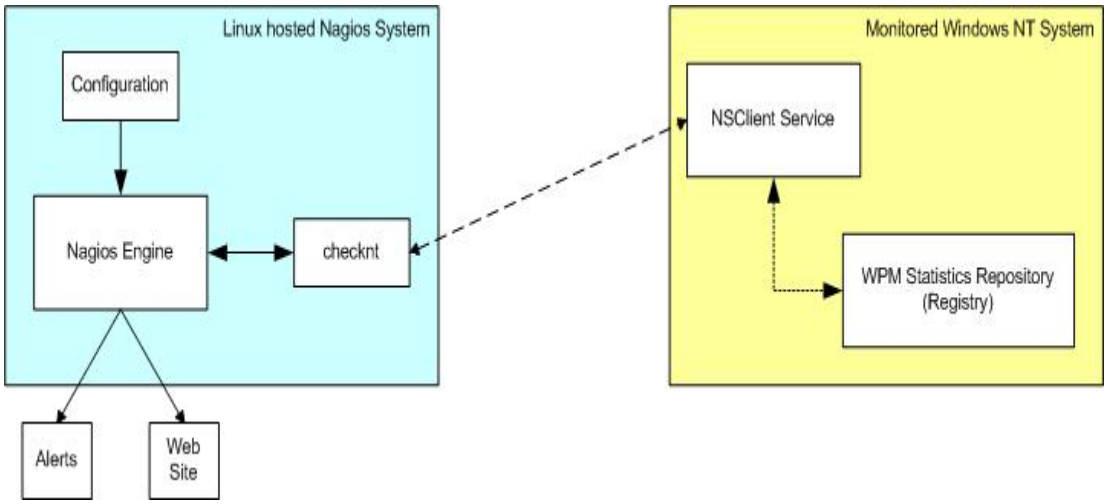

Figura 5.1 – Funcionamento do NSClient

Figura retirada do artigo de Whitehead (2004).

O serviço NSClient deve ser instalado em cada servidor ou estação de trabalho a ser monitorada e este serviço pode ser acessado por meio do serviço Checknt instalado em um NMS, conforme apresentado na figura 5.1, ou por meio de um conjunto de classes Java, que são agrupadas no *Java Archive* (JAR) de nome NSClient4j, que funciona como *Application Program Interface* (API) de comunicação entre um programa Java de um sistema externo e o serviço NSClient, conforme representado em diagrama esquemático na figura 5.2

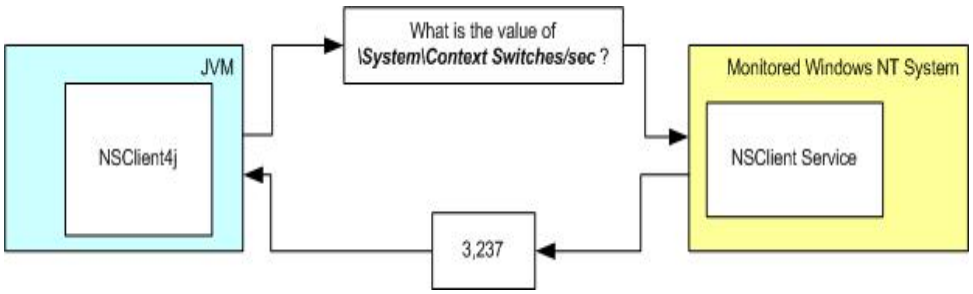

Figura 5.2 – Relacionamento entre o NSClient4j e o NSClient

Figura retirada do artigo de Whitehead (2004).

Na figura 5.2 é apresentada a solicitação de coleta de dados do valor armazenado em um contador de desempenho. A solicitação é iniciada a partir de uma aplicação em Java que é executada dentro da *Java Virtual Machine* (JVM) do elemento monitorado (estação de trabalho ou servidor) e que faz uso de métodos constantes no JAR NSClient4j. A aplicação de monitoração informa o nome do contador que deve ter o dado coletado, que no exemplo da figura 5.2 é o contador *\System\Context Switches/sec*. A solicitação com o nome do contador é encaminhada para o serviço NSClient que repassa o pedido para o WPM, que consulta o valor armazenado no contador especificado, que neste exemplo é 3,237 e retorna o dado coletado para o serviço NSClient que o envia para a aplicação de monitoração.

A solução de coleta de dados disponibilizada pelo JAR NSClient4j tem algumas peculiaridades, que influenciam diretamente no processo de construção do aplicativo de monitoração:

- O nome do contador a ser monitorado deve ser exato;

- O nome do contador é sensível ao idioma em que o sistema operacional Windows foi instalado.

O padrão de formação do nome do contador no WPM segue a seguinte formação:

- Objeto – é uma categoria geral de contadores, por exemplo, processador, memória, disco, processos, interface de rede e outros. Na figura 5.2 o objeto é *System*.

- Contador – é a característica específica do objeto que será monitorada, por exemplo, bytes por segundo, percentual de utilização de memória, percentual de utilização de processador, interrupções por segundo e outros. Na figura 5.2 o contador é *Context Switches/sec*.

- Instância – é a possibilidade de granularidade de um contador. Pode existir mais de uma ocorrência de um determinado objeto em um computador, por esse motivo, cada objeto pode ter seus dados coletados separadamente, indicando o nome do objeto para o Windows ou de forma coletiva pelo cálculo da média, indicando o nome todos. Por exemplo, um computador pode ter mais de um processador ou mais de uma placa de rede. Nesses casos, se for necessário coletar os dados de forma individualizada, deve ser feita uma consulta separada para cada objeto, com a indicação clara do nome que o objeto tem no sistema operacional. É importante ressaltar que, para tornar o processo de implementação da aplicação de monitoração mais versátil, desenvolveu-se facilidade de descoberta automática da interface de rede principal do computador e a inclusão do seu nome na solicitação de coleta de dados, evitando assim a necessidade de ajustes na aplicação de monitoração.

Como o nome do contador é sensível ao idioma em que foi instalado o sistema operacional Windows, a aplicação de monitoração deve ser ajustada de acordo com o idioma utilizado no computador, isto é, português ou inglês.

Na figura 5.3 estão descritas as classes existentes no JAR NSClient4j:

- NSClient4j – contém os principais métodos da API, com destaque principal para o método getPerfMonCounter, que é utilizado pela aplicação de monitoração para encaminhar uma solicitação de coleta de dados;

- CLStat – contém a entrada principal da API com a palavra reservada *main*;

- NSClient4jException – contém os métodos para tratamento de exceções.

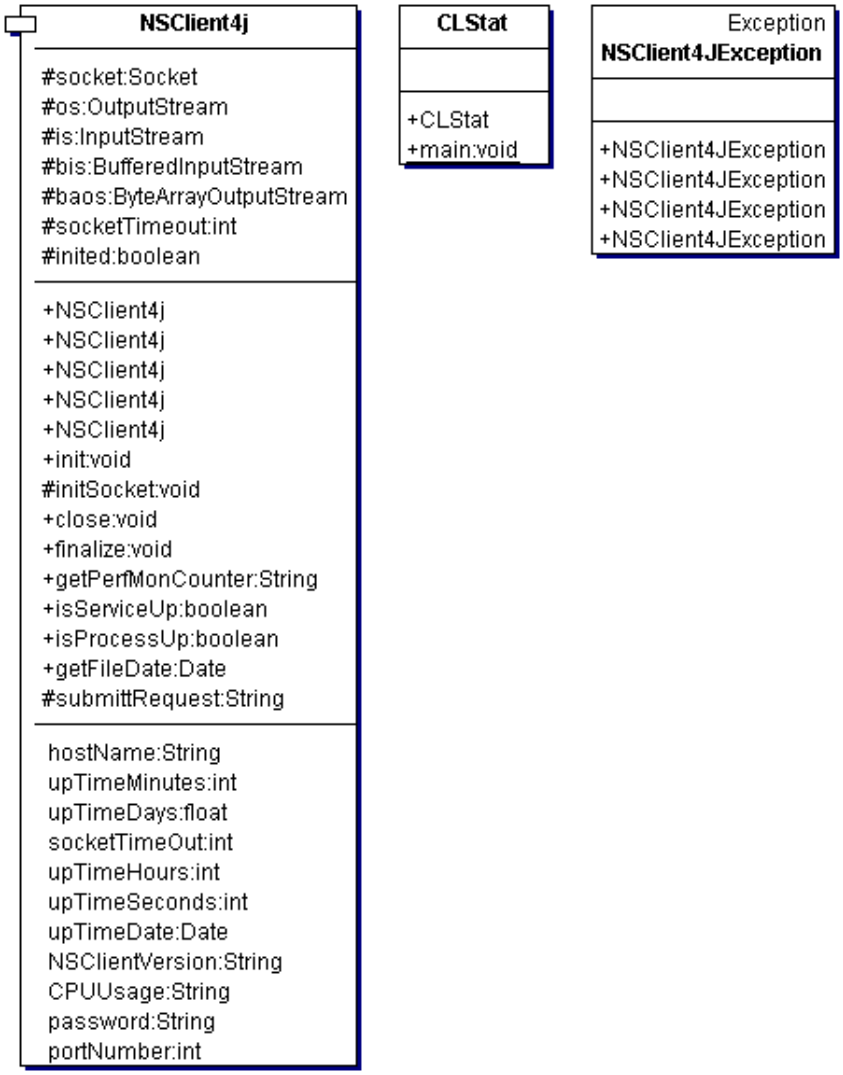

Figura 5.3 – Diagrama de classes do NSClient4j

Figura retirada do artigo de Whitehead (2004).

As classes JAR NSClient4j estão disponíveis para *download* gratuito no sítio http://nsclient.ready2run.nl, conforme NSClient (2005), sendo que a última versão 2.0.1 foi disponibilizada em junho de 2003.

O material disponível no sítio citado acima é complementado com um tutorial disponível no sítio http://www.javaworld.com, que foi elaborado por Nicholas Whitehead em novembro de 2004, conforme Whitehead (2004).

No desenvolvimento da aplicação de monitoração foi utilizada a biblioteca Java padrão disponibilizada pela *Sun Microsystems*, por meio da sua distribuição J2SE 1.4.2, em conjunto com o ambiente integrado de desenvolvimento Eclipse versão 3.0, disponível para *download* gratuito no sítio http://www.eclipse.org, conforme Eclipse (2005).

5.2.2 API JDOM

As informações que são coletadas pelo agente de monitoração devem ser armazenadas em um documento XML antes do envio para o NMS. Para efetuar o processo de criação do documento XML foi escolhida a API *Java Document Object Model* (JDOM), desenvolvida pela *Sun Microsystems* e que é baseada na especificação DOM do *World Wide Web Consortium* (W3C) e que está disponível no sítio http://www.w3c.org. A escolha da especificação DOM para definição do documento XML se deve principalmente aos seguintes fatores:

- A especificação é um padrão internacional;
- A especificação pode ser utilizada em qualquer plataforma de software;
- A especificação pode ser implementada em qualquer linguagem de programação.

A abordagem citada acima pode ser corroborada com a seguinte explicação, retirada do livro *Java and XML* de Brett McLaughlin, conforme McLaughlin (2001):

O padrão DOM, diferentemente do SAX, teve sua origem no World Wide Web Consortium (W3C). O SAX é um software de domínio público, desenvolvido por meio de longas discussões na lista do XMLdev, ao passo que o DOM é um padrão. O DOM não é específico para a linguagem Java e pode representar o conteúdo e o modelo de documentos em qualquer linguagem de programação ou ferramenta. Dessa forma, o padrão DOM pode ser utilizado em qualquer plataforma de software e em qualquer linguagem de programação.

A linguagem de programação Java implementa no pacote *org.jdom* várias classes que permitem a criação em memória e em arquivo de documentos XML, bem como o tratamento de informações armazenadas em documentos XML.

No agente a classe *XmlToFile* implementa a criação do documento XML conforme pode ser verificado no trecho abaixo:

```
Element root = new Element("node");

    Element host1 = new Element("host");

    Element date1 = new Element("date");

    Element time1 = new Element("time");
```

```
Element proc1 = new Element("proc");

Element mem1 = new Element("mem");

Element netin1 = new Element("netin");

Element netout1 = new Element("netout");

root.setAttribute("host", host);

root.setAttribute("date", date);

root.setAttribute("time", time);

root.setAttribute("proc", proc);

root.setAttribute("mem", mem);

root.setAttribute("netin", Float.toString(netin));

root.setAttribute("netout", Float.toString(netout));

Document doc = new Document(root)
```

5.3 Tecnologias para Transferência de Dados

5.3.1 URL

O significado da sigla URL é *Uniform Resource Locator* e uma URL é utilizada para fazer referência a um recurso em uma rede TCP/IP. Em geral, os usuários humanos utilizam a URL como uma informação de entrada em um navegador *Web*, como *Internet Explorer* ou *Mozilla*, para solicitar ao navegador que faça o acesso ao servidor que contém a informação solicitada e apresente a informação resultante na tela do usuário.

A linguagem de programação Java implementa no pacote *java.net* a classe URL, que permite o tratamento de chamadas a URL como referências de recursos na rede ou como objetos, viabilizando não apenas o acesso a recursos, mas também a execução de métodos locais ou remotos, inclusive com a passagem de parâmetros na execução de um método.

Uma URL é composta de duas partes:

- Uma parte que identifica o protocolo;
- Uma parte que identifica o recurso.

A figura 5.4 ilustra a divisão da URL em suas duas partes componentes, sendo que o protocolo utilizado é o HTTP e o nome do recurso aponta para *java.sun.com*, que é uma das referências possíveis ao sítio da empresa *Sun Microsystems* referente à linguagem de programação Java.

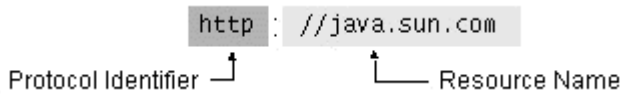

Figura 5.4 – Componentes da URL.

Figura retirada do artigo da SUN Microsystems (2005).

A identificação do protocolo indica o protocolo que deverá ser utilizado para a realização da comunicação. Os protocolos mais utilizados são o HTTP, SMTP, FTP, *Gopher*, *File* e *News*. O nome do recurso depende do protocolo de comunicação que está sendo utilizado, mas na maioria dos protocolos é composto do seguinte formato:

- Nome do *host*;
- Nome ou caminho para acesso a um arquivo;
- Número da porta que será utilizada;
- Referência para um ponto de início da página *Web*.

A classe URL é utilizada na implementação do agente na classe *Data*, conforme pode ser verificado no trecho de código abaixo, extraído da referida classe.

```
// Does the connection with the XNMP server.

    URL u = new URL("http://192.168.0.28:8080/ReceiveData");

    URLConnection uc = u.openConnection();

    uc.setDoOutput(true);

    uc.connect();

    OutputStream out = uc.getOutputStream();

    PrintWriter pw = new PrintWriter(uc.getOutputStream())
```

5.3.2 *Servlet*

O *servlet* é um componente Java que pode ser configurado em um programa Java no elemento gerenciado e que permite a comunicação, por meio do protocolo HTTP, com outro componente *servlet* configurado em um programa Java no NMS. O método de troca de informações entre *servlets* é baseado no modelo de solicitação de resposta, conforme Hall e Brown (2003). A tecnologia *servlet* permite a comunicação programa a programa, sendo muito útil em situações em que não seja necessária a intervenção de operadores humanos no processo de comunicação e transferência de dados.

As principais vantagens, conforme Hall e Brown (2003) na utilização de *servlets* são:

- Eficiência – uma única cópia da classe *servlet* pode tratar várias solicitações HTTP simultaneamente.

- Conveniência – *servlets* já possuem tecnologia embutida para tratamento e análise de dados, principalmente dados transmitidos por meio dos protocolos HTTP, SMTP e FTP.

- Poderoso – totalmente integrado ao servidor *web*, facilitando a interpretação de URL´s e acesso a banco de dados.

- Portável – como os *servlets* são escritos em Java podem ser facilmente utilizados em qualquer modelo de servidor *web*.

- Seguro – os *servlets* são executados dentro da JVM, fazendo uso das características de segurança fornecidas pela execução em um ambiente de execução controlado.

- Barato – os pacotes de implementação de *servlets* são distribuídos gratuitamente pelos fabricantes da tecnologia.

A tecnologia *servlet* é disponibilizada pela Sun por meio das API´s:

- javax.servlet;

- javax.servlet.http.

A API *servlet* é utilizada na implementação do NMS na classe *Data* com o método *doGet*, para recepção dos dados transmitidos pelo agente do elemento gerenciado. Após a recepção dos dados, a classe *Data* efetua a passagem dos dados para o método *setWriteData*, que efetuará a gravação dos dados recebidos na base de dados.

5.3.3 API SOAP

A API SOAP é implementada no agente na classe *Data* com o uso do método *sendData*. A tecnologia SOAP é disponibilizada pela *Apache Software Foundation* por meio das API:

- org.apache.soap.*.

Na implementação da API SOAP foi utilizado o seguinte código de programação:

```
//Construir a mensagem de requisição do SOAP RPC usando o objeto Call

Call call = new Call();

call.setTargetObjectURI(“urn:ReceiveData”);

call.setMethodName(“ReceiveData”);

call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
```

//Criar um objeto URL, que representa a extremidade

URL url = new URL(endPoint);

//Enviar a mensagem de solicitação do SOAP RP usando o método invoke()

Response resp = call.invoke (url, “”);

O documento XML gerado pela aplicação de monitoração será incluído como parte do corpo do envelope SOAP.

5.4 Armazenamento de Dados no NMS

As informações coletadas pelo aplicativo de monitoração no elemento gerenciado são transmitidas para o NMS, conforme definido no sub-item anterior, e então são armazenadas em base de dados. No NMS a aplicação de gerenciamento de base de dados utilizada é o PostgreSQL 8.0, disponível no sítio http://www.postgresql.org, conforme PostgreSQL (2005). O PostgreSQL 8.0 é um *software* livre e de alto desempenho, e por esses motivos foi escolhido para a implementação da aplicação do NMS.

No PostgreSQL foi criada uma tabela para cada servidor do *cluster*. As colunas criadas em cada tabela são:

- codigo;
- *time*;
- *proc*;
- *mem*;
- *netin*;
- *netout*.

A coluna codigo foi incluída para permitir a indexação dos registros da tabela, facilitando a leitura seqüencial de informações. As demais colunas são obtidas a partir dos dados recebidos no documento XML enviado pelo agente de monitoração do elemento gerenciado.

Na aplicação NMS a classe que faz a gravação dos dados recebidos é a classe *Data* com o uso do método *setWriteData*. O método *setWriteData* utiliza comandos da linguagem *Structured Query Language* (SQL) e para isso faz-se necessário a utilização de API´s do pacote *Java Database Connectivity* (JDBC):

- java.sql.*;
- java.io.*.

5.5 Visualização das Informações Coletadas

As informações coletadas pelo elemento gerenciado são armazenadas em base de dados. Quando o usuário precisa efetuar o acesso às informações coletadas, ele deve utilizar a classe *Manager*, a qual providenciará a abertura de uma tela gráfica que exibe a lista de elementos gerenciados, conforme pode ser verificado na figura 5.5. A tela de monitoração contém uma lista ilustrativa dos nós que estão sendo monitorados atualmente. Cada nó é representado individualmente na lista de nós monitorados e, vinculado a cada nó, aparecem três variáveis que estão sendo monitoradas: processador, memória e rede. Para selecionar as informações referentes a um determinado nó, o usuário deve selecionar com o *mouse* o nó desejado e então clicar no botão *show*.

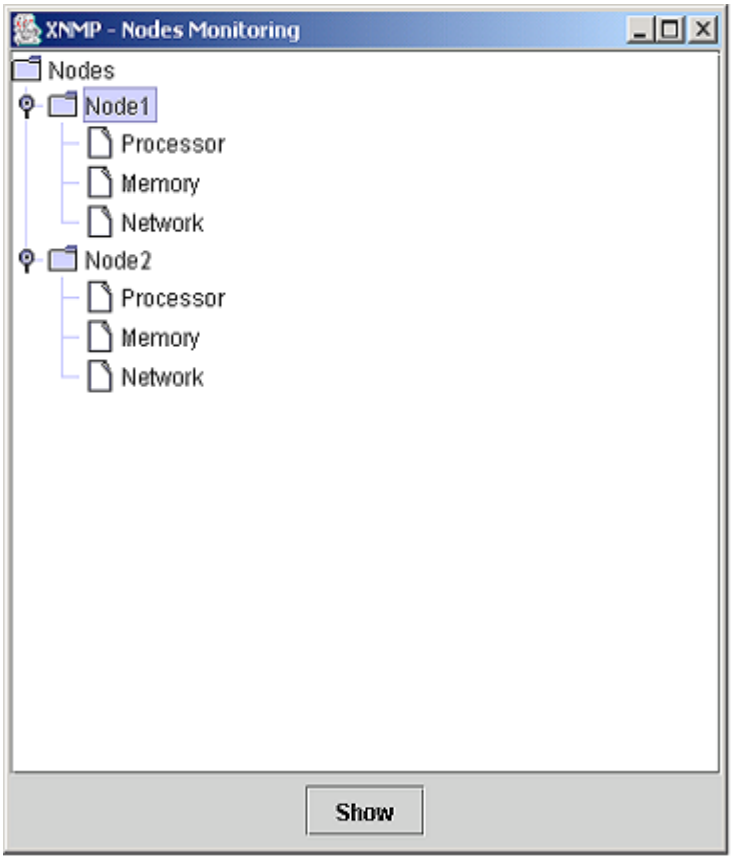

Figura 5.5 – Tela de Monitoração.

Após clicar no botão *show*, aparecerá a tela com as informações monitoradas, conforme pode ser verificado na figura 5.6. No gráfico da figura 5.6, o eixo Y corresponde ao percentual de utilização da variável monitorada e o eixo X corresponde à linha do tempo em segundos.

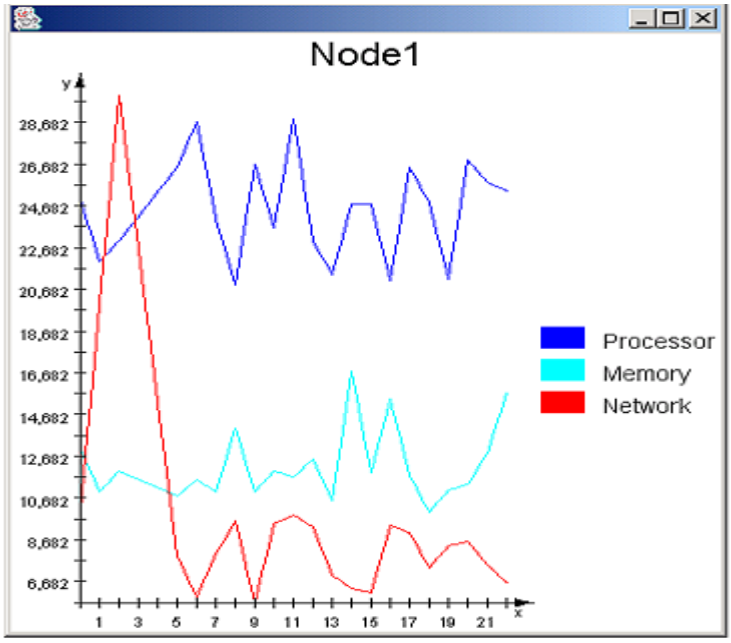

Figura 5.6 – Dados da Monitoração.

A classe *GraphFrame* implementa a apresentação gráfica dos dados coletados com a utilização da API JopenChart versão 0.94 , desenvolvida por Sebastian Mueller e disponibilizada gratuitamente no sítio http//jopenchart.sourceforge.net/index.shtml, conforme Mueller (2005). Os dados de entrada para geração do gráfico podem ser obtidos por meio de leitura em arquivos, valores aleatórios gerados em programas ou dados armazenados em bases de dados. Neste trabalho será utilizada a leitura de dados armazenados em base de dados.

5.6 Conclusão

Nesse capítulo foram apresentados os componentes de software que fazem parte da aplicação de monitoração do cluster de servidores e também os métodos utilizados para a implementação da aplicação. A aplicação de monitoração é composta por duas partes, que atuam em sincronismo e que são, respectivamente, o agente, que é instalado em cada elemento gerenciado, e o NMS.

No desenvolvimento da aplicação agente foram utilizadas as seguintes tecnologias:

- API NSClient4j;
- API JDOM;
- API SOAP;
- Servlet.

No desenvolvimento da aplicação NMS foram utilizadas as seguintes tecnologias:

- API SOAP;

- API JDBC;
- Servlet
- API JOpenChart.

O próximo capítulo descreve o ambiente em que são realizados os testes com a aplicação de monitoração, a bateria de testes utilizada e são apresentados e comentados os resultados obtidos durante os testes.

6 AVALIAÇÃO DOS RESULTADOS OBTIDOS

Neste capítulo é apresentado o ambiente em que os testes foram realizados, com a descrição dos computadores que fizeram parte do *cluster* de servidores. A bateria de testes que foi aplicada também é descrita, com as diferentes opções de carga e de tipo de dados (distribuição homogênea ou heterogênea de dados) em que os testes foram realizados. Avalia-se os resultados obtidos com a realização dos testes e aponta-se sugestões para melhoria da distribuição de carga no *cluster* de servidores. Por fim, descreve-se as dificuldades encontradas durante a realização dos testes.

6.1 Introdução

Os testes foram realizados em computadores instalados no Laboratório de Eletromagnetismo Aplicado do PEA da Escola Politécnica da USP (EPUSP). Os computadores que fizeram parte do *cluster* de teste são considerados equipamentos para usuário final, tendo, portanto, capacidade de processamento e de armazenamento compatível com a utilizada usualmente por usuários finais no desenvolvimento de suas atividades de rotina. Durante a realização dos testes os usuários não utilizaram os computadores para que não houvesse interferência no processamento do teste.

6.2 Ambiente de Realização dos Testes

O ambiente de realização dos testes foi composto por cinco computadores. O computador central atua como o controlador do teste e ele é responsável pelo gerenciamento das atividades desenvolvidas pelos demais computadores do *cluster*.

O gerenciamento destas atividades pelo computador central é feito através de mensagens por meio do uso da API MPICH, que é uma implementação do padrão *Message Passing Interface* (MPI) e que está disponível no sítio http://www-unix.mcs.anl.gov/mpi/mpich/index.htm do *Argonne National Laboratory* do Departamento de Energia dos Estados Unidos da América, conforme Argonne (2005).

Os testes foram realizados considerando-se a análise numérica de um dispositivo eletromagnético através do Método dos Elementos Finitos. O dispositivo analisado representa um eletroímã, cuja geometria varia em função de valores atribuídos a parâmetros dos quais as coordenadas de alguns de seus pontos são dependentes. O eletroímã pode ser visto na figura 6.1, juntamente com a malha de elementos finitos utilizada em sua análise.

O objetivo final desta aplicação é encontrar o conjunto de valores ótimos dos parâmetros de geometria para os quais a força resultante no eletroímã seja máxima, o que exige que sejam realizadas várias simulações por elementos finitos até que seja obtido este conjunto de valores. Isso implica que, para cada conjunto de parâmetros simulado, é necessária a realização das seguintes etapas que compõem o processo padrão de resolução por elementos finitos:

- alterar os dados da geometria do eletroímã, obtidos a partir da leitura inicial do arquivo de projeto referente ao software de elementos finitos;

- gerar a malha de elementos finitos para a geometria atualizada;

- montar e resolver um sistema de equações matriciais, referente aos dados da malha de elementos finitos gerada;

- salvar os dados da solução do sistema de equações em disco (opcional);

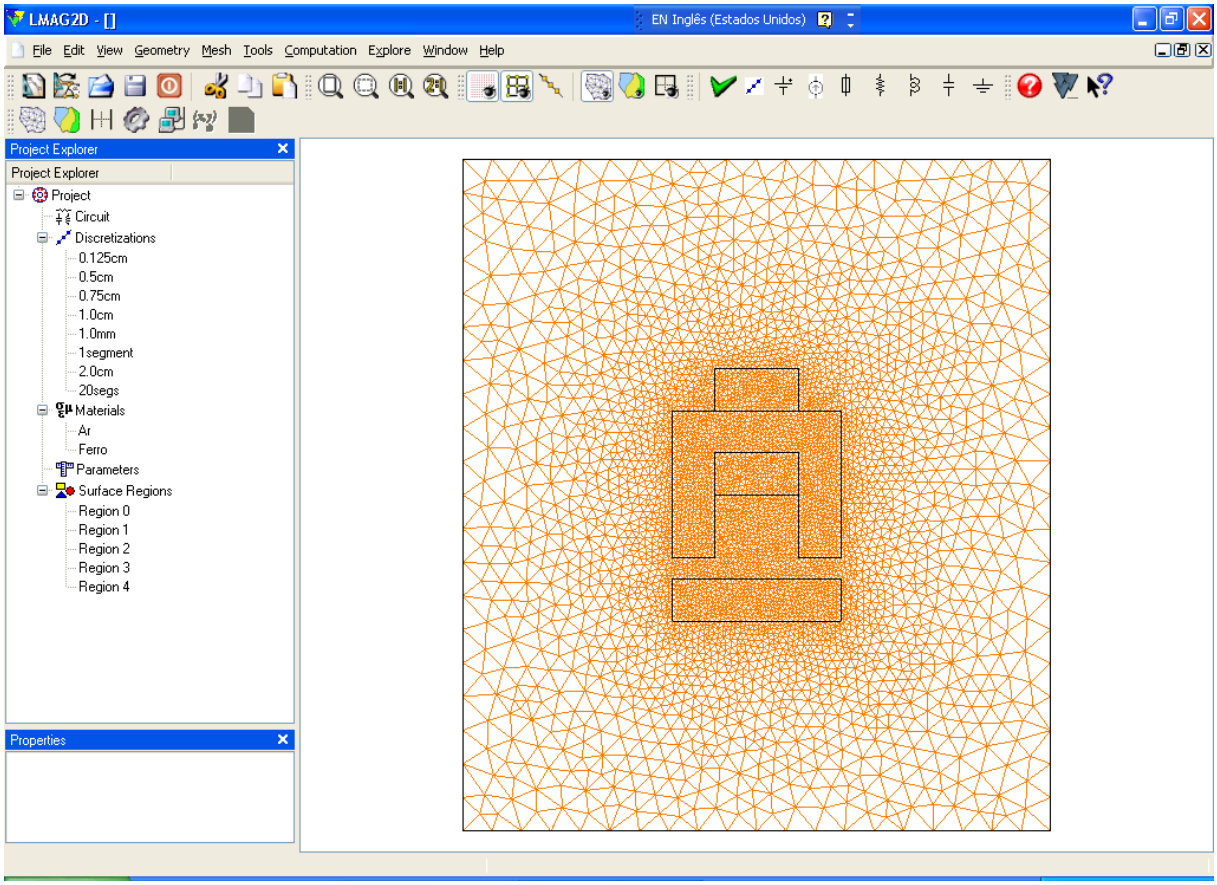

Figura 6.1 – Eletroímã e a malha de elementos finitos.

No intuito de avaliar mais precisamente o tempo de processamento dos diferentes computadores no processo de otimização do eletroímã, optou-se por não fazer a alteração efetiva dos valores dos parâmetros de geometria, de modo a se trabalhar sempre com o mesmo sistema matricial que, neste caso, é composto por 4915 equações (quando são utilizados elementos de primeira ordem, em que cada equação corresponde ao vértice de um triângulo) ou 19417 equações (no caso em que os elementos utilizados são de segunda ordem, onde as equações correspondem aos vértices dos triângulos e aos pontos médios de suas arestas).

Os testes foram divididos nos seguintes casos:

- Caso 1 – cálculo com elementos de segunda ordem e sem gravação de resultados parciais no disco rígido de um computador remoto;

- Caso 2 – cálculo com elementos de primeira ordem e sem gravação de resultados parciais no disco rígido de um computador remoto;

- Caso 3 – cálculo com elementos de segunda ordem e com gravação de resultados parciais no disco rígido de um computador remoto;

- Caso 4 – cálculo com elementos de primeira ordem e com gravação de resultados parciais no disco rígido de um computador remoto;

- Caso 5 – cálculo com elementos de segunda ordem, com gravação de resultados parciais no disco rígido de um computador remoto e com distribuição homogênea de dados;

- Caso 6 – cálculo com elementos de segunda ordem, sem gravação de resultados parciais no disco rígido de um computador remoto e com distribuição homogênea de dados.

A distribuição homogênea dos dados significa que todos os computadores do cluster realizaram o mesmo número de resoluções do sistema matricial, independente do seu poder de processamento. Na distribuição heterogênea, o número de resoluções efetuadas por cada processador depende do seu poder de processamento.

A aplicação de monitoração foi instalada em quatro computadores que compunham o *cluster* e que possuem processadores AMD Athlon, com sistema operacional Microsoft Windows XP versão 2002 com *service pack* 2. Os computadores utilizados foram:

- Amstel;

- Budweiser;

- Erdinger;

- Superbock.

O computador Amstel possui dois processadores AMD Athlon MP 1800+ com 1.53 GHz, memória *Randomic Access Memory* (RAM) de 512 *megabytes* e placa de rede de 100 *megabits* por segundo.

O computador Budweiser possui processador AMD Athlon com 1.20 GHz, memória RAM de 512 *megabytes* e placa de rede de 100 *megabits* por segundo.

O computador Erdinger possui processador AMD Athlon XP 1800+ com 1.53 GHz, memória RAM de 512 *megabytes* e placa de rede de 100 *megabits* por segundo.

O computador Superbock possui processador AMD Athlon XP 2400+ com 1.99 GHz, memória RAM de 512 *megabytes* e placa de rede de 100 *megabits* por segundo.

A tabela 6.1 consolida as informações referentes aos itens tipo de processador, quantidade de memória RAM e velocidade da placa de rede dos computadores que fazem parte do *cluster*.

Tabela 6.1 – Consolidação dos dados dos computadores do *cluster*.

	Processador	Memória	Entrada de rede
Amstel	AMD Athlon MP 1800+ com 1.53 GHz	512 *megabytes*	100 *megabits* por segundo
Budweiser	AMD Athlon com 1.20 GHz	512 *megabytes*	100 *megabits* por segundo
Erdinger	AMD Athlon XP 1800+ com 1.53 GHz	512 *megabytes*	100 *megabits* por segundo
Superbock	AMD Athlon XP 2400+ com 1.99 GHz	512 *megabytes*	100 *megabits* por segundo

Para a realização dos testes não foi efetuada a criação de rede isolada, sendo que o tráfego de rede gerado pelos testes concorreu com o tráfego rotineiro gerado na rede do laboratório, devido à indisponibilidade de um laboratório em rede isolada e a expectativa de que o tráfego gerado pela rede é muito baixo, não interferindo nos resultados obtidos durante a realização dos testes.

6.3 Descrição dos Resultados Obtidos e Dificuldades Encontradas

Os resultados obtidos durante a realização dos testes são apresentados nesta subseção, conforme gerados pela API JOpenChart, com um gráfico separado para cada um dos quatro itens monitorados e separados também pelos quatro computadores. Os quatro itens considerados de maior importância para o funcionamento adequado da aplicação de cálculo de motores elétricos pelo MEF e que foram monitorados durante a realização dos testes são:

- Utilização de processador;
- Utilização de memória;
- Utilização de tráfego de entrada na placa de rede;
- Utilização de tráfego de saída na placa de rede.

A figura 6.2 contém as informações de utilização do computador Amstel durante a execução do caso 1. O eixo y informa o percentual de utilização e o eixo x informa o intervalo de amostragem no decorrer do tempo. As amostras são realizadas a cada 8 segundos e para facilitar a visualização das informações, cada ponto do eixo x representa um intervalo de 8 segundos, totalizando 25 amostras. Pode-se verificar na figura 6.2 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 12% e chegou a 20%. A utilização da placa de rede ficou quase em 0%.

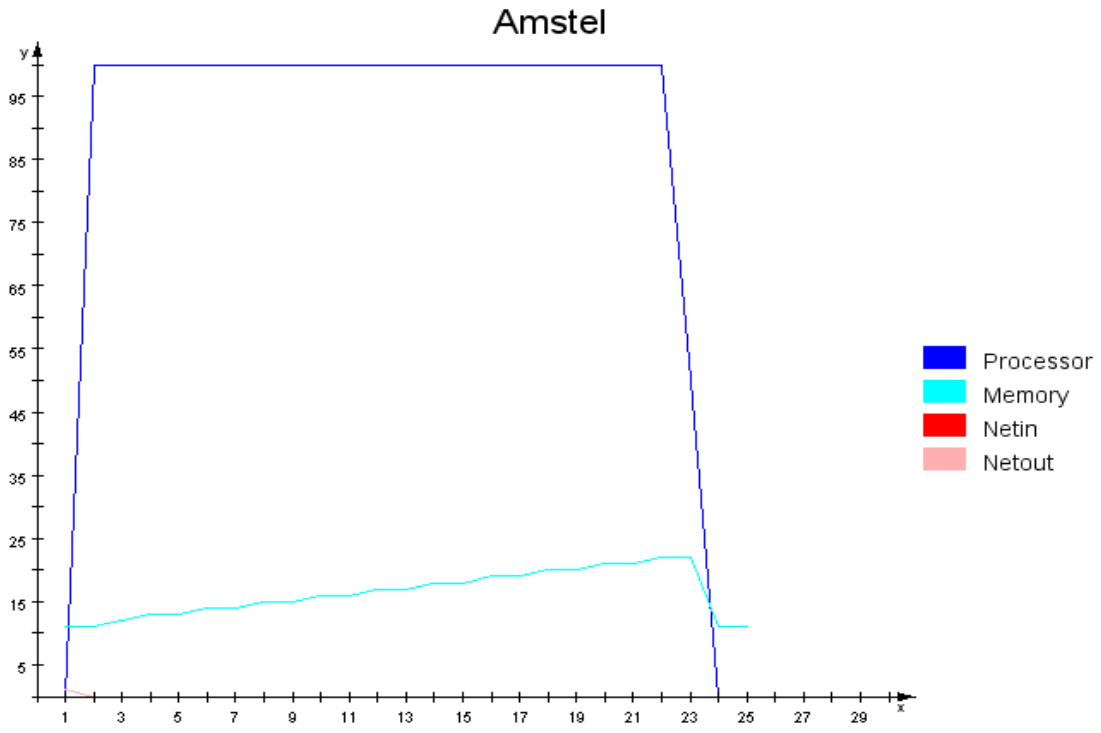

Figura 6.2 – Computador Amstel caso 1.

A figura 6.3 contém os dados de utilização durante a execução do caso 2 no computador Amstel. Pode-se verificar na figura 6.3 que a utilização do processador teve um crescimento exponencial e manteve-se em 100%. A utilização de memória começou com 12% e manteve-se estável. A utilização da placa de rede ficou quase em 0%. Um problema verificado na geração do gráfico da figura 6.3 é a representação dos dados do eixo y em uma escala fracionada, que dificulta a leitura direta dos dados. Verifica-se que o fracionamento é feito automaticamente pela API JOpenChart de geração dos gráficos, quando os dados do eixo y são de pequeno valor.

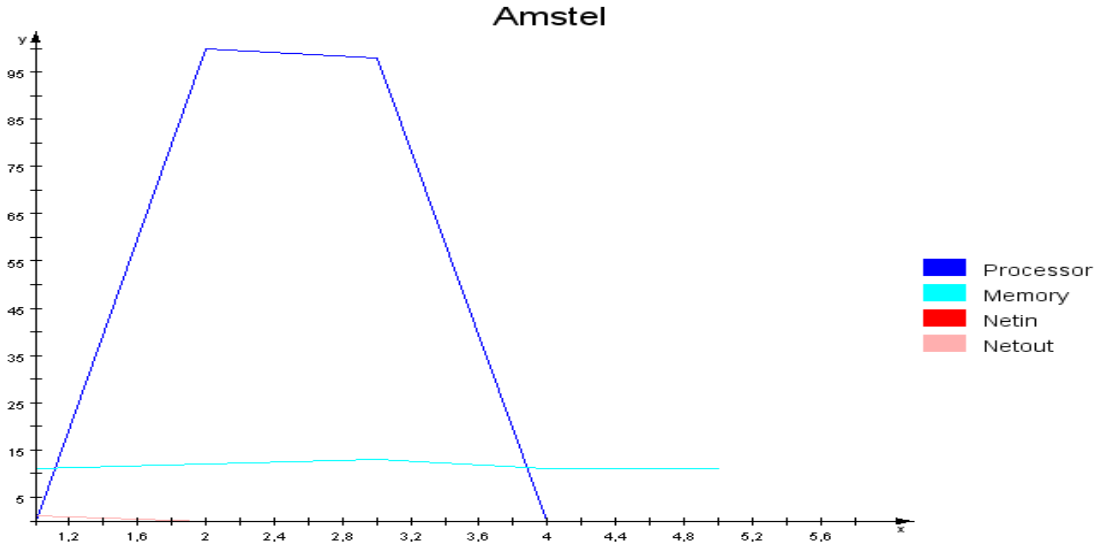

Figura 6.3 – Computador Amstel caso 2.

A figura 6.4 contém os dados de utilização durante a execução do caso 3 no computador Amstel. Pode-se verificar na figura 6.4 que a utilização do processador teve um oscilação constante entre 35% e 100%. A utilização de memória começou com 12% e manteve crescimento contínuo até alcançar 20%. A utilização da placa de rede oscilou entre 0% e 5% nos sentidos de recepção e envio de dados.

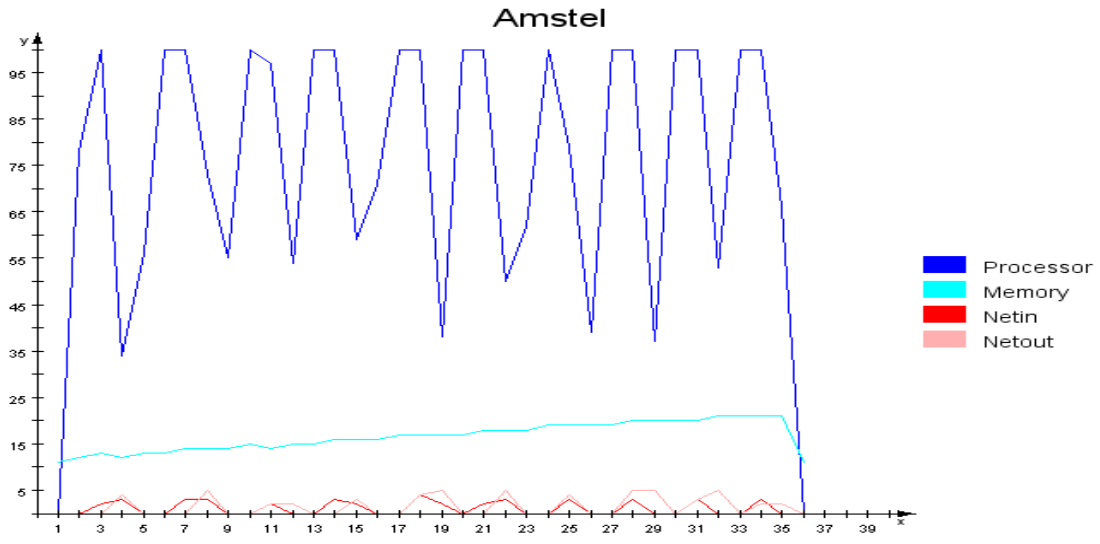

Figura 6.4 – Computador Amstel caso 3.

A figura 6.5 contém os dados de utilização durante a execução do caso 4 no computador Amstel. Pode-se verificar na figura 6.5 que a utilização do processador teve oscilação entre 46% e 84%. A utilização de memória começou com 12% e manteve-se praticamente estável durante a realização dos testes. A utilização da placa de rede oscilou entre 0% e 4% nos sentidos de recepção e envio de dados.

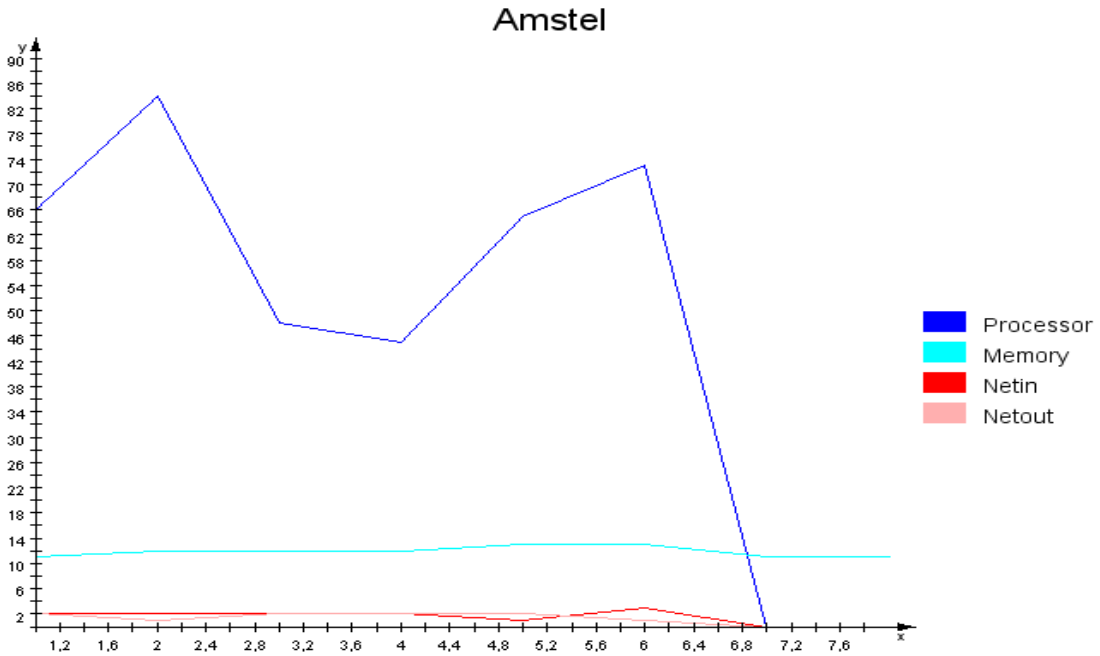

Figura 6.5 – Computador Amstel caso 4.

A figura 6.6 contém os dados de utilização durante a execução do caso 5 no computador Amstel. Pode-se verificar na figura 6.6 que a utilização do processador teve oscilação quase constante entre 35% e 100%. A utilização de memória começou com 13% e manteve crescimento contínuo até alcançar 21%. A utilização da placa de rede oscilou entre 0% e 9%, com maior carga no sentido de recepção de dados.

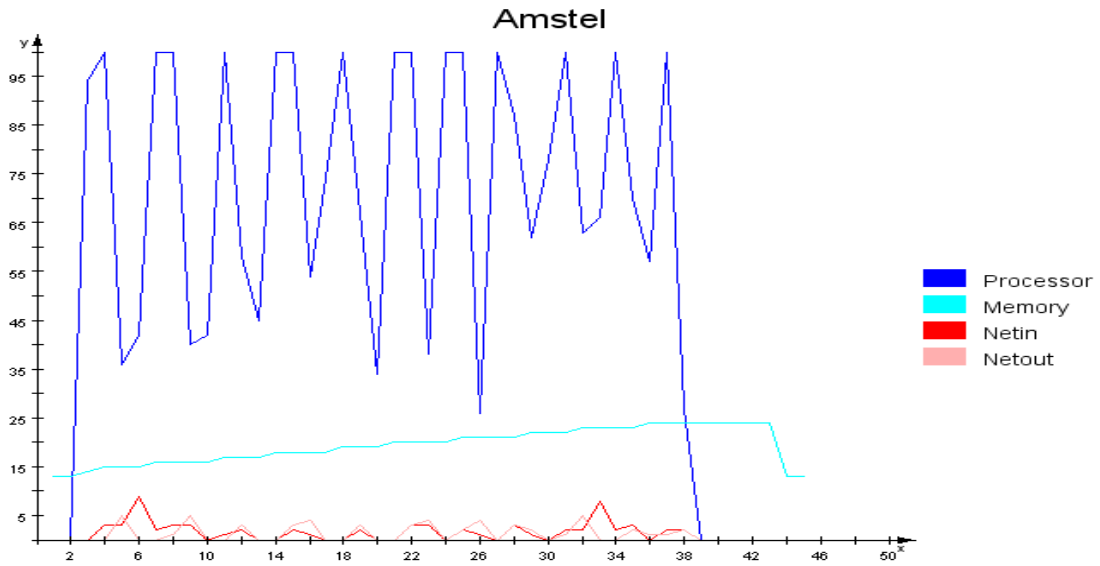

Figura 6.6 – Computador Amstel caso 5.

A figura 6.7 contém as informações de utilização do computador Amstel durante a execução do caso 6. Pode-se verificar na figura 6.7 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 17% e chegou a 21%. A utilização da placa de rede ficou quase em 0%.

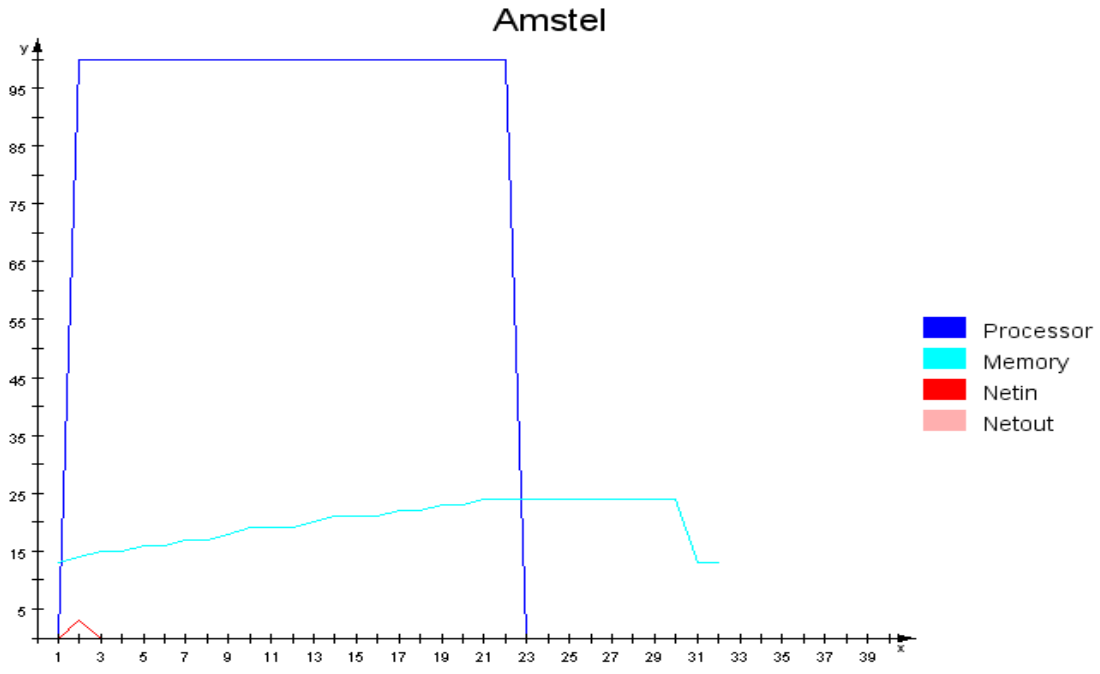

Figura 6.7 – Computador Amstel caso 6.

A figura 6.8 contém as informações de utilização do computador Budweiser durante a execução do caso 1. Pode-se verificar na figura 6.8 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 12% e chegou a 15%. A utilização da placa de rede ficou próximo de 0%.

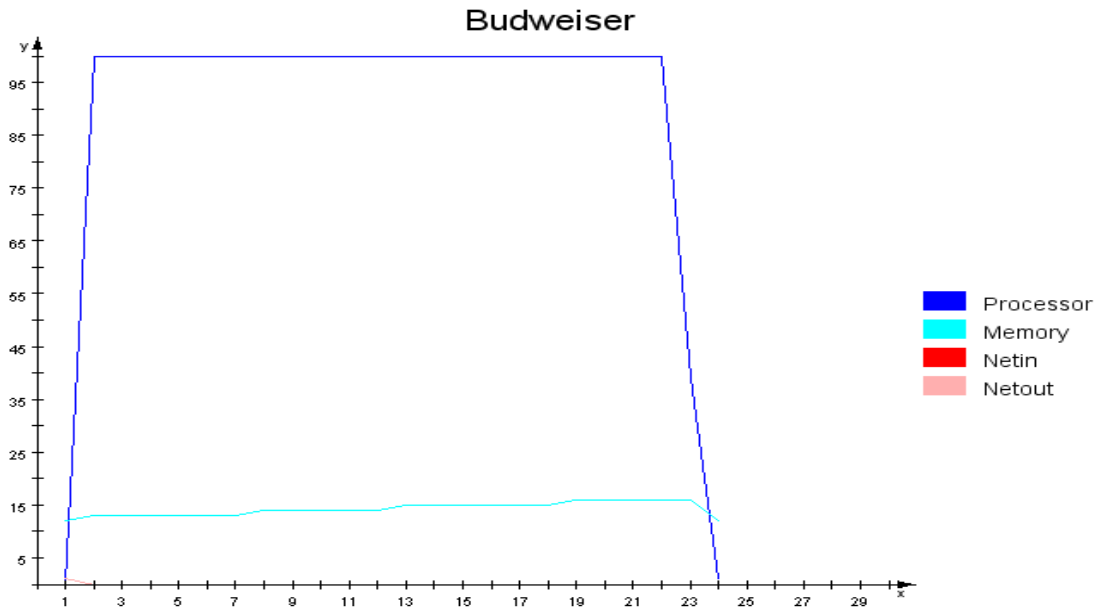

Figura 6.8 – Computador Budweiser caso 1.

A figura 6.9 contém os dados de utilização durante a execução do caso 2 no computador Budweiser. Pode-se verificar na figura 6.9 que a utilização do processador teve um crescimento exponencial e manteve-se em 100%. A utilização de memória começou com 12% e manteve-se estável. A utilização da placa de rede começou em 1%, variou para 0% e no final do teste estava com 7% no sentido de recepção de dados.

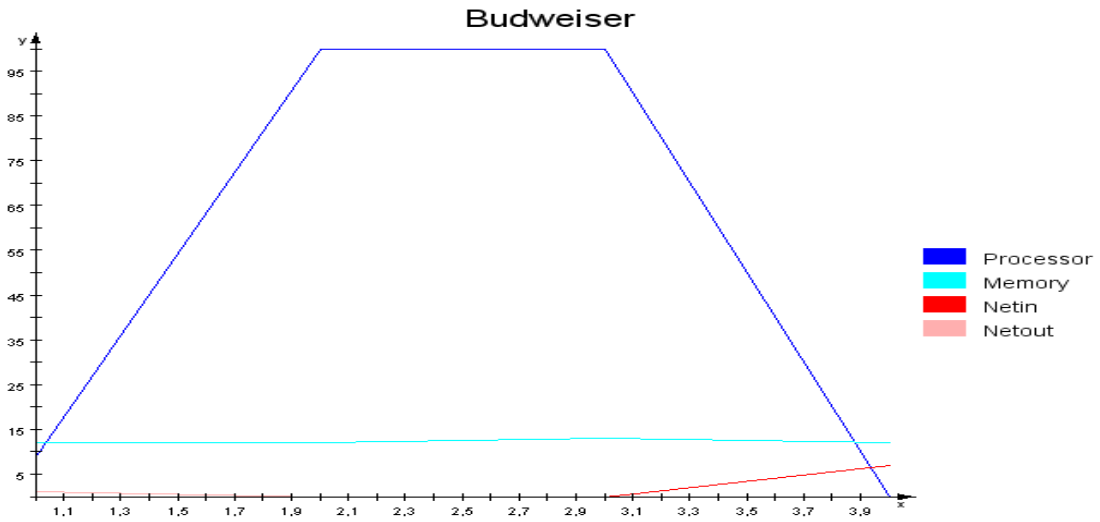

Figura 6.9 – Computador Budweiser caso 2.

A figura 6.10 contém os dados de utilização durante a execução do caso 3 no computador Budweiser. Pode-se verificar na figura 6.10 que a utilização do processador teve comportamento variável caracterizados por picos e vales intercalados que oscilaram entre 35% e 100%. A utilização de memória começou com 12% e manteve crescimento contínuo até alcançar 15%. A utilização da placa de rede oscilou entre 0% e 3% nos sentidos de recepção e envio de dados.

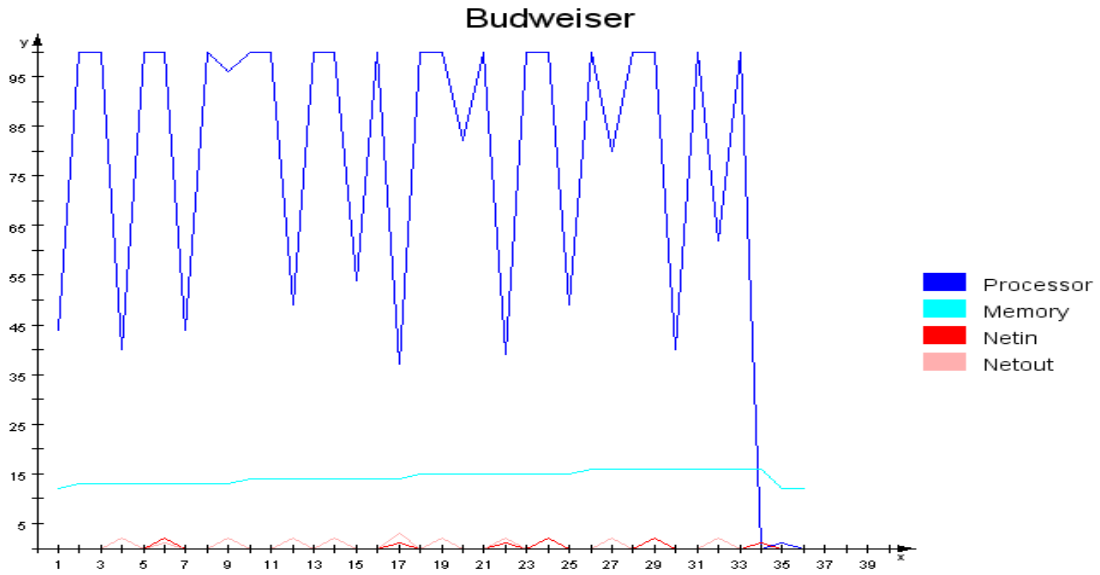

Figura 6.10 – Computador Budweiser caso 3.

A figura 6.11 contém os dados de utilização durante a execução do caso 4 no computador Budweiser. Pode-se verificar na figura 6.11 que a utilização do processador teve oscilação entre 62% e 95%. A utilização de memória começou com 13% e manteve-se praticamente estável durante a realização dos testes. A utilização da placa de rede oscilou entre 0% e 2% nos sentidos de recepção e envio de dados.

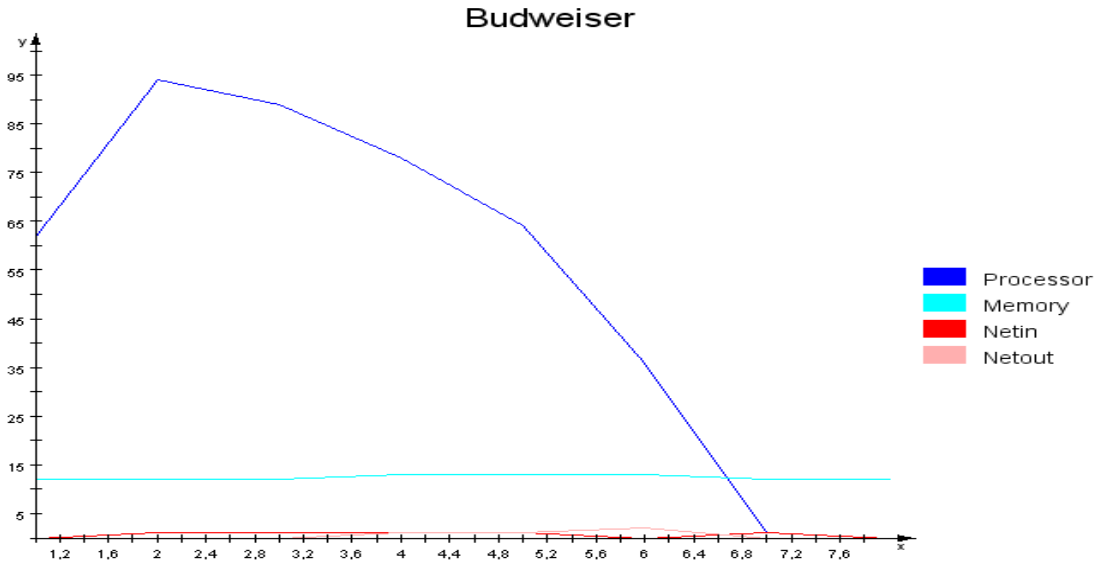

Figura 6.11 – Computador Budweiser caso 4.

A figura 6.12 contém os dados de utilização durante a execução do caso 5 no computador Budweiser. Pode-se verificar na figura 6.12 que a utilização do processador teve um crescimento exponencial e ficou com oscilação entre 35% e 100%. A utilização de memória começou com 13% e manteve crescimento contínuo até alcançar 16%. A utilização da placa de rede oscilou entre 0% e 9%, com maior carga no sentido de recepção de dados.

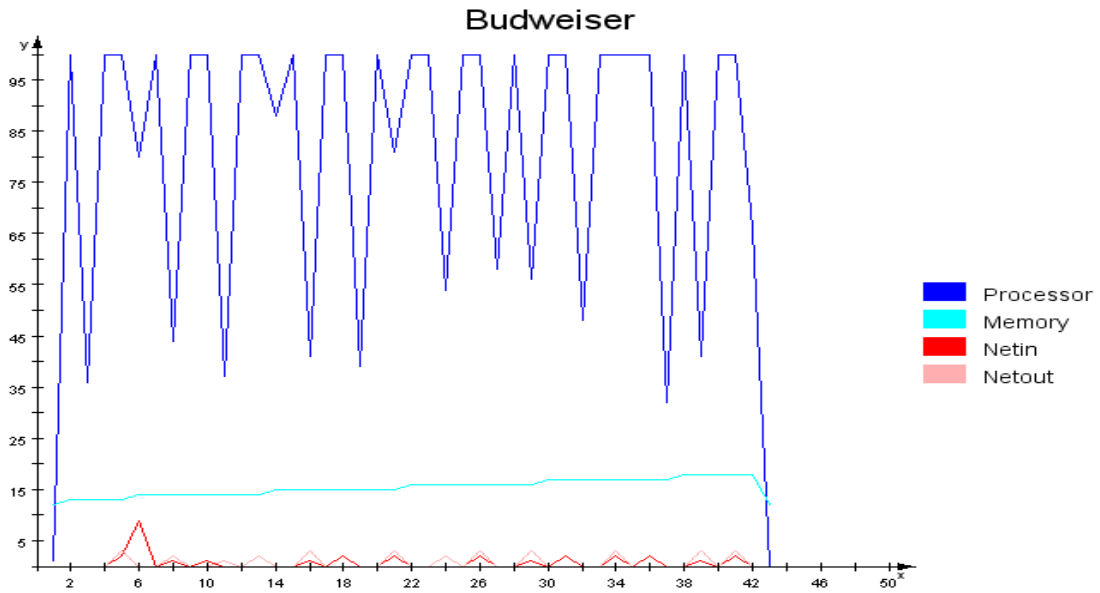

Figura 6.12 – Computador Budweiser caso 5.

A figura 6.13 contém as informações de utilização do computador Budweiser durante a execução do caso 6. Pode-se verificar na figura 6.13 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 14% e chegou a 16%. A utilização da placa de rede ficou com oscilações entre 0% e 7% no envio e na recepção de dados.

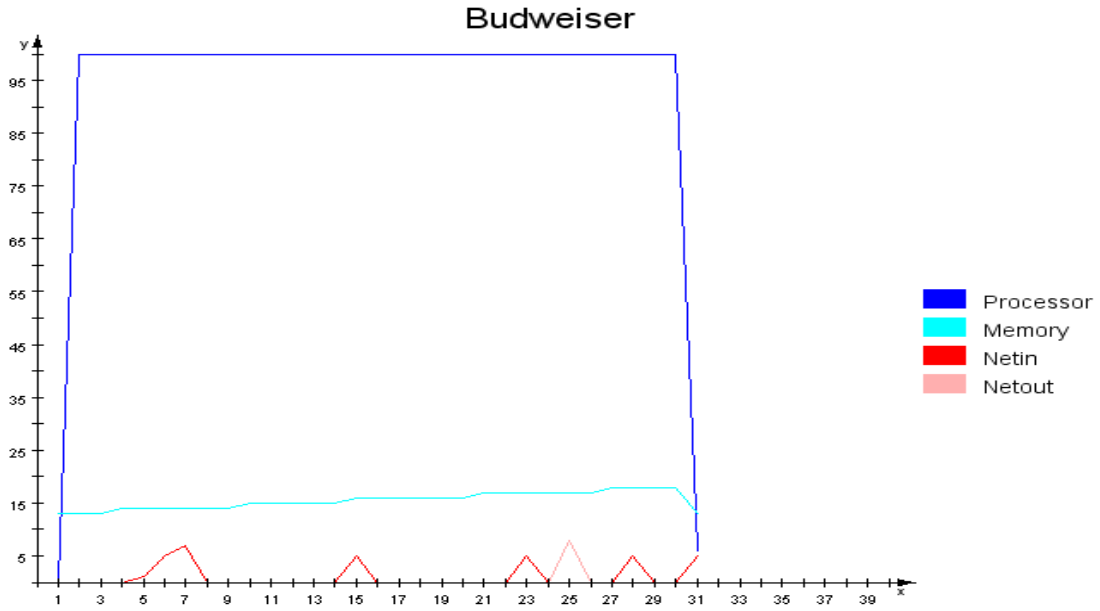

Figura 6.13 – Computador Budweiser caso 6.

A figura 6.14 contém as informações de utilização do computador Erdinger durante a execução do caso 1. Pode-se verificar na figura 6.14 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 12% e chegou a 20%. A utilização da placa de rede ficou quase em 0%.

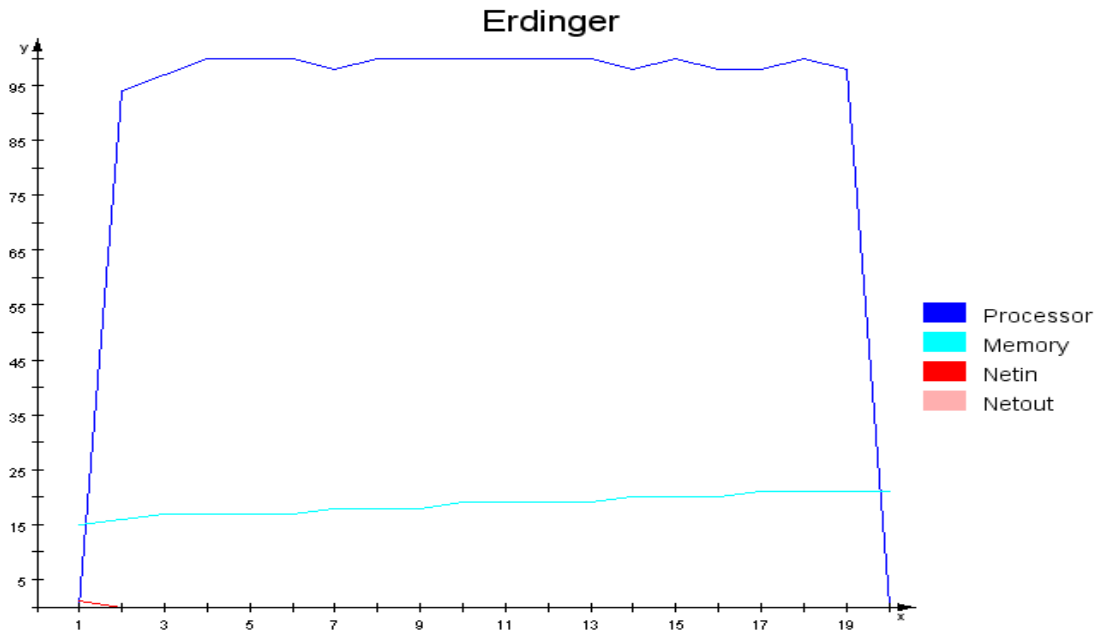

Figura 6.14 – Computador Erdinger caso 1.

A figura 6.15 contém os dados de utilização durante a execução do caso 2 no computador Erdinger. Pode-se verificar na figura 6.15 que a utilização do processador teve um crescimento contínuo de 84% até 96%, com queda acentuada até 0%. A utilização de memória começou com 16% e manteve-se estável. A utilização da placa de rede ficou em quase 0%.

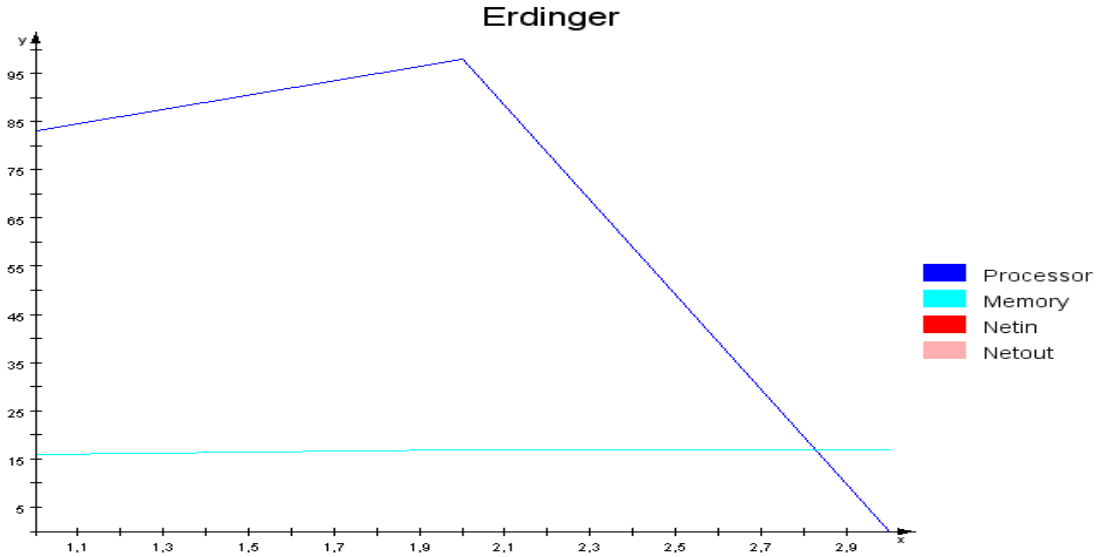

Figura 6.15 – Computador Erdinger caso 2.

A figura 6.16 contém os dados de utilização durante a execução do caso 3 no computador Erdinger. Pode-se verificar na figura 6.16 que a utilização do processador teve um crescimento exponencial e ficou com oscilação entre 0% e 100%. A utilização de memória começou com 16% e manteve crescimento contínuo até alcançar 19%. A utilização da placa de rede oscilou entre 0% e 3% nos sentidos de recepção e envio de dados.

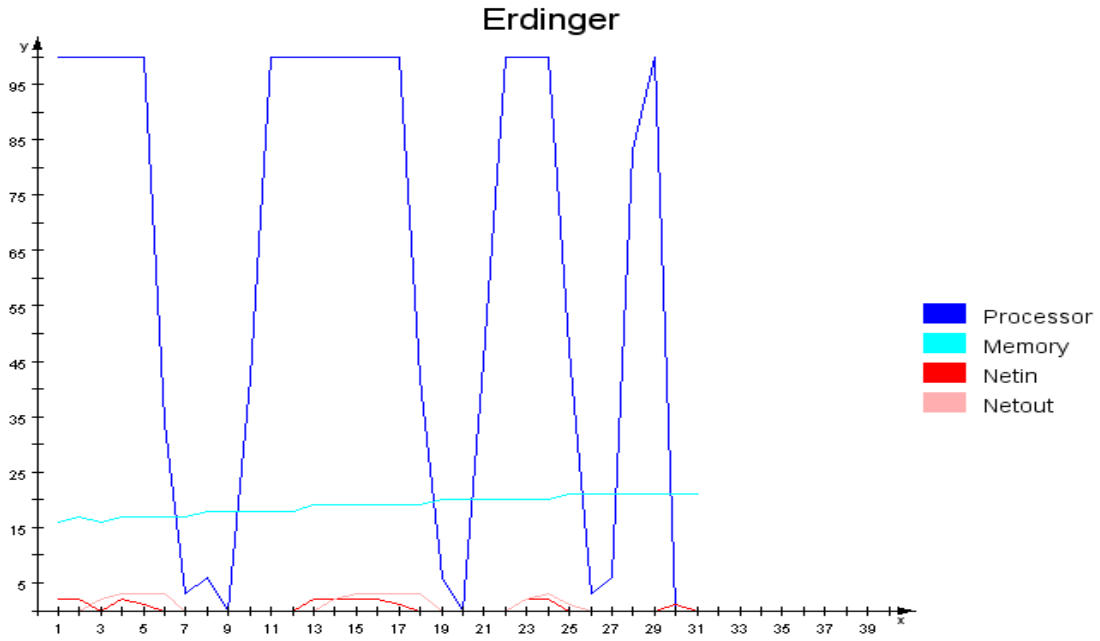

Figura 6.16 – Computador Erdinger caso 3.

A figura 6.17 contém os dados de utilização durante a execução do caso 4 no computador Erdinger. Pode-se verificar na figura 6.17 que a utilização do processador teve oscilação entre 0% e 72%. A utilização de memória começou com 16% e manteve-se praticamente estável durante a realização dos testes. A utilização da placa de rede oscilou entre 0% e 2% nos sentidos de recepção e envio de dados.

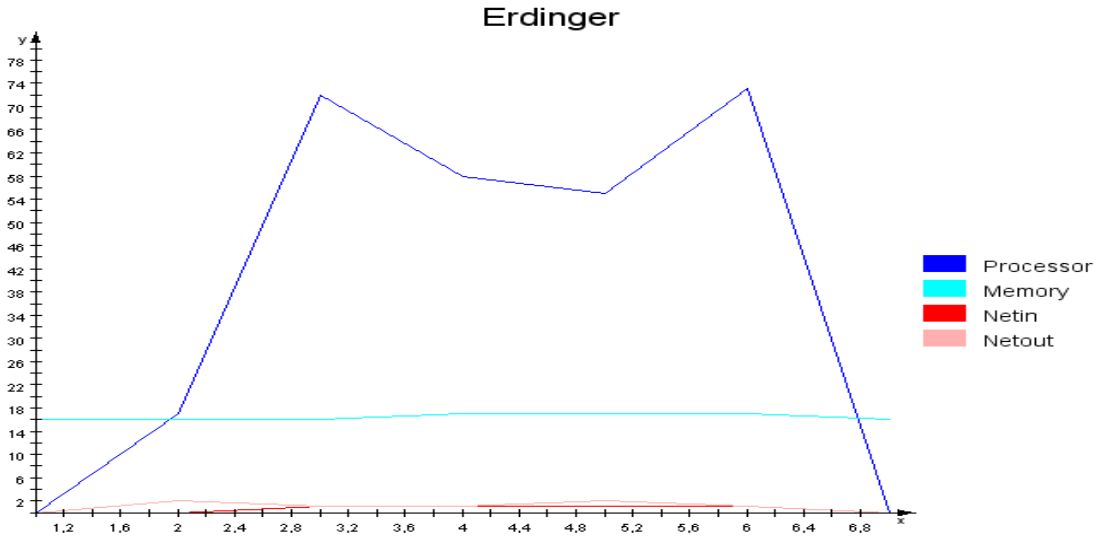

Figura 6.17 – Computador Erdinger caso 4.

A figura 6.18 contém os dados de utilização durante a execução do caso 5 no computador Erdinger. Pode-se verificar na figura 6.18 que a utilização do processador teve um crescimento exponencial e ficou com oscilação entre 0% e 100%. A utilização de memória começou com 16% e manteve crescimento contínuo até alcançar 20%. A utilização da placa de rede oscilou entre 0% e 3% na recepção e no envio de dados.

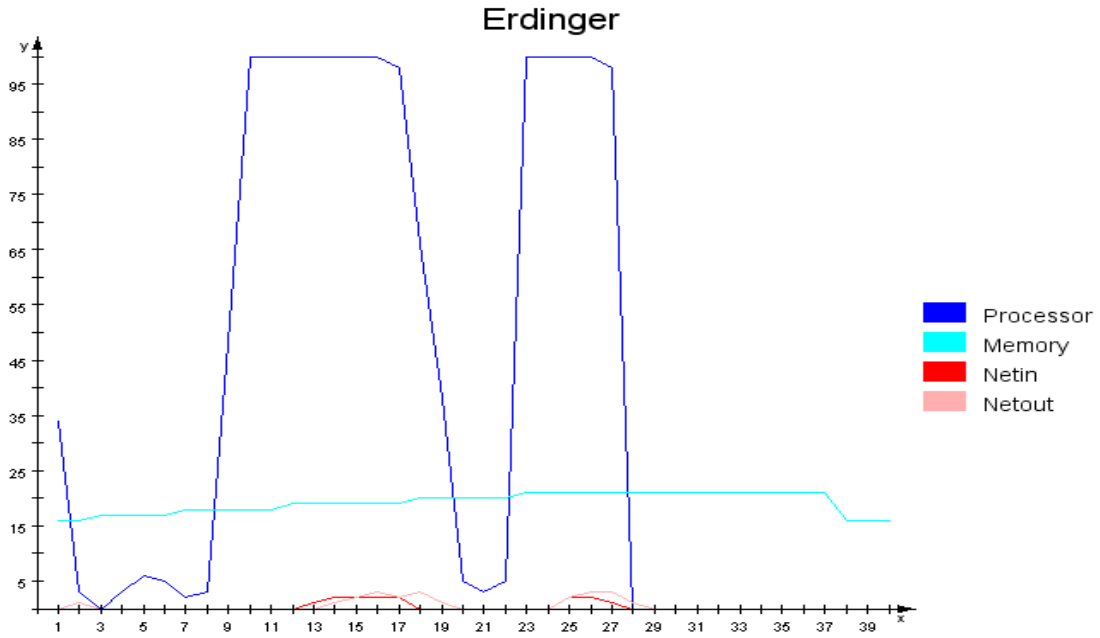

Figura 6.18 – Computador Erdinger caso 5.

A figura 6.19 contém as informações de utilização do computador Erdinger durante a execução do caso 6. Pode-se verificar na figura 6.19 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 16% e chegou a 19%. A utilização da placa de rede ficou quase em 0%.

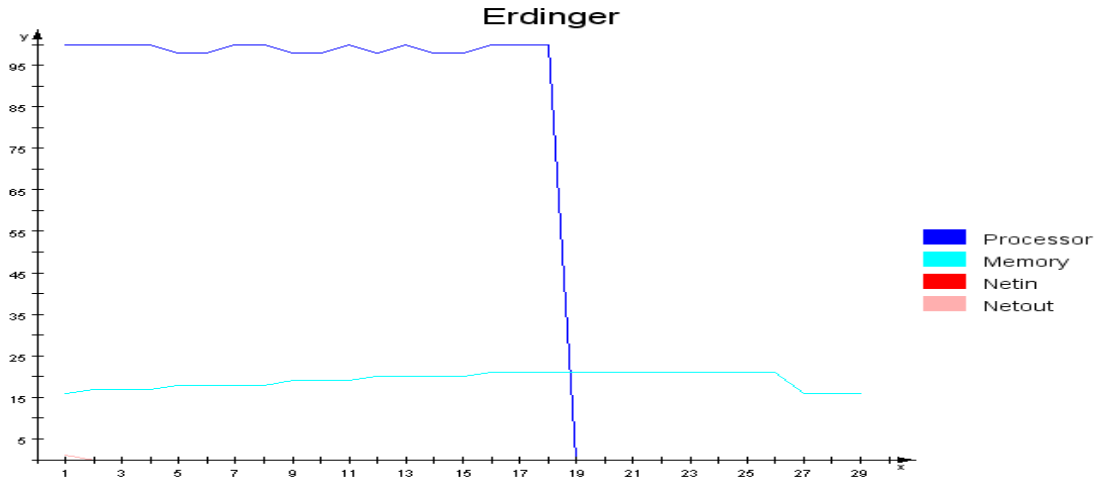

Figura 6.19 – Computador Erdinger caso 6.

A figura 6.20 contém as informações de utilização do computador Superbock durante a execução do caso 1. Pode-se verificar na figura 6.20 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 15% e chegou a 19%. A utilização da placa de rede ficou quase em 0%.

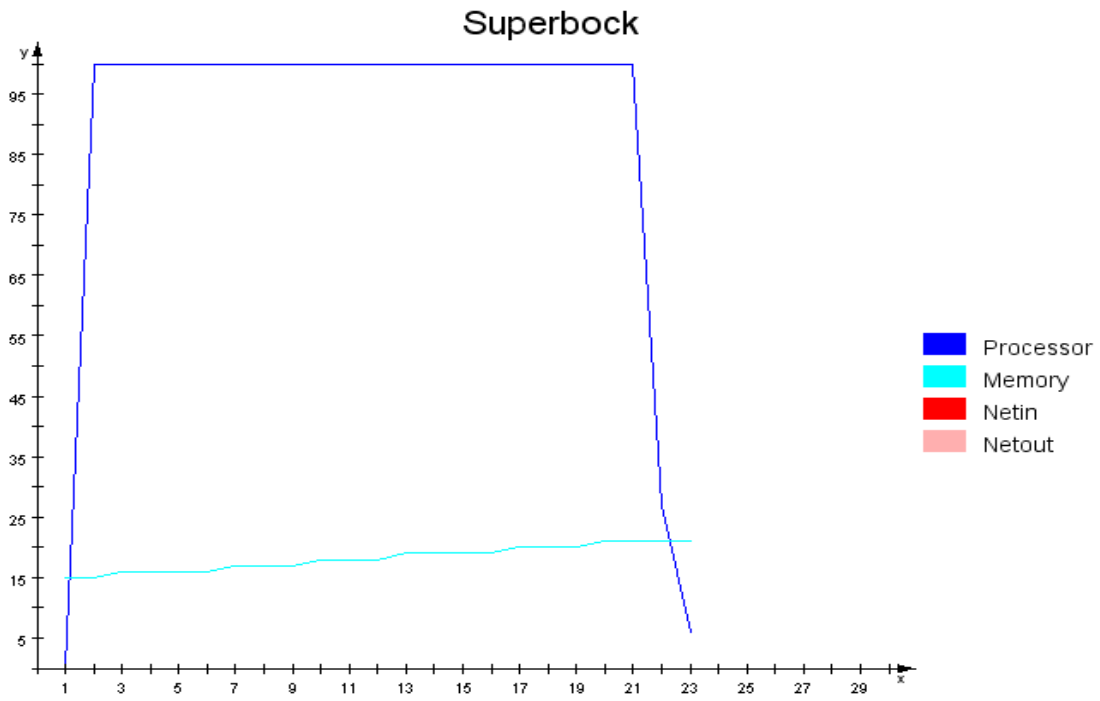

Figura 6.20 – Computador Superbock caso 1.

A figura 6.21 contém os dados de utilização durante a execução do caso 2 no computador Superbock. Pode-se verificar na figura 6.21 que a utilização do processador manteve-se em 100%. A utilização de memória começou com 15% e manteve-se estável. A utilização da placa de rede ficou quase em 0%.

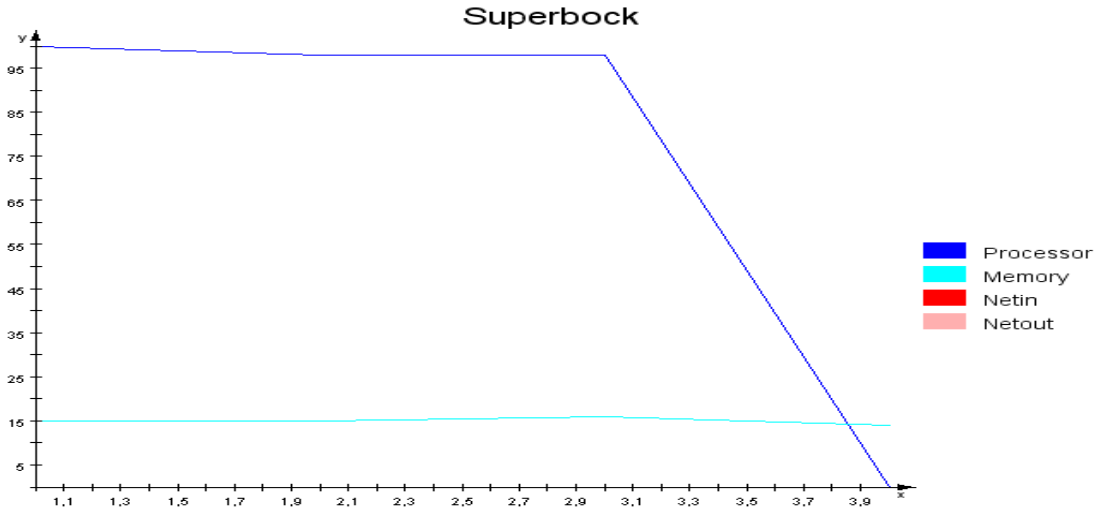

Figura 6.21 – Computador Superbock caso 2.

A figura 6.22 contém os dados de utilização durante a execução do caso 3 no computador Superbock. Pode-se verificar na figura 6.22 que a utilização do processador teve um crescimento exponencial e ficou com oscilação entre 12% e 100%. A utilização de memória começou com 15% e manteve crescimento contínuo até alcançar 19%. A utilização da placa de rede oscilou entre 0% e 4% nos sentidos de recepção e envio de dados.

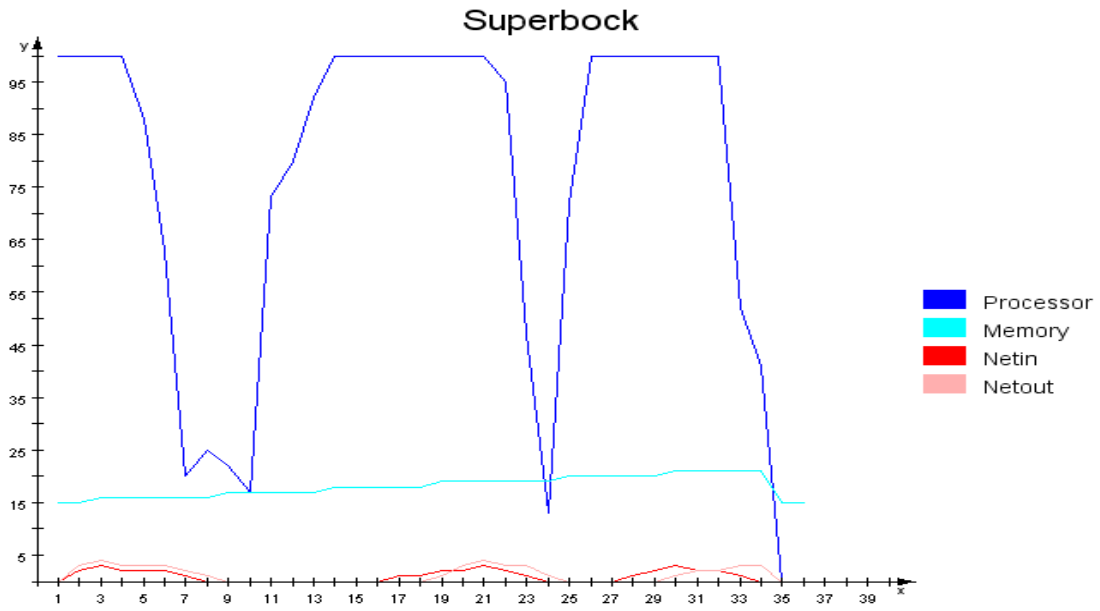

Figura 6.22 – Computador Superbock caso 3.

A figura 6.23 contém os dados de utilização durante a execução do caso 4 no computador Superbock. Pode-se verificar na figura 6.23 que a utilização do processador teve oscilação entre 38% e 72%. A utilização de memória começou com 14% e manteve-se praticamente estável durante a realização dos testes. A utilização da placa de rede oscilou entre 0% e 2% nos sentidos de recepção e envio de dados.

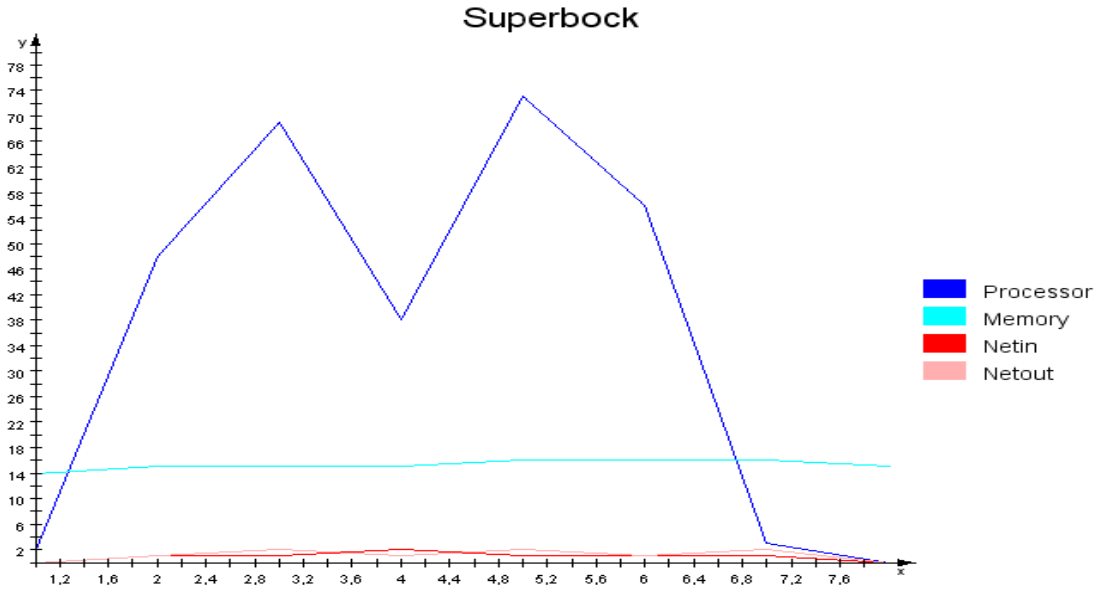

Figura 6.23 – Computador Superbock caso 4.

A figura 6.24 contém os dados de utilização durante a execução do caso 5 no computador Superbock. Pode-se verificar na figura 6.24 que a utilização do processador oscilou entre 14% e 100%. A utilização de memória começou com 14% e manteve crescimento contínuo até alcançar 18%. A utilização da placa de rede oscilou entre 0% e 5% na recepção e no envio de dados.

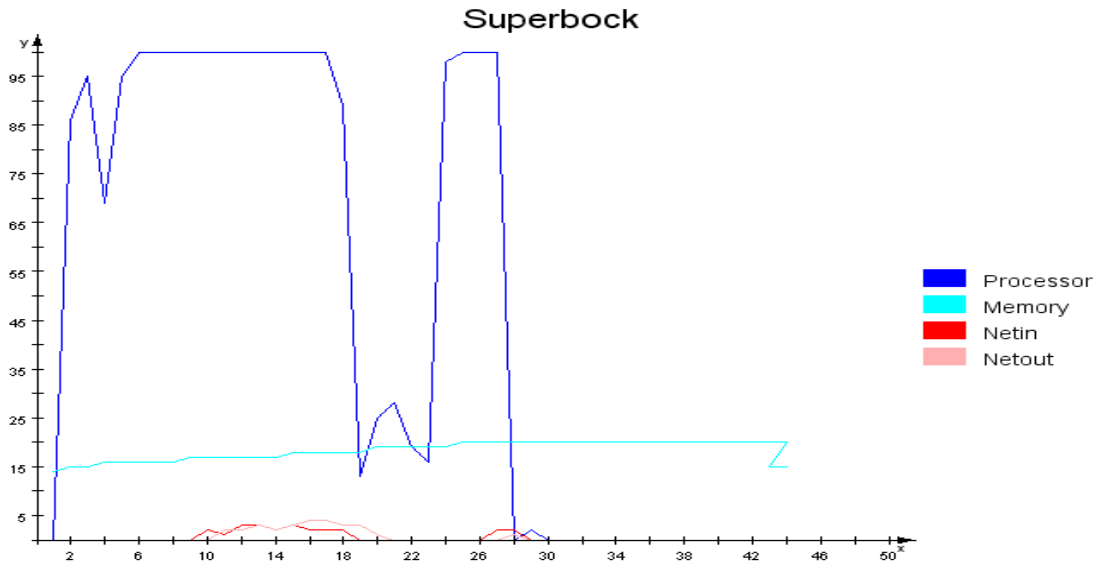

Figura 6.24 – Computador Superbock caso 5.

A figura 6.25 contém as informações de utilização do computador Superbock durante a execução do caso 6. Pode-se verificar na figura 6.25 que a utilização do processador foi contínua e manteve-se em 100%. A utilização de memória começou com 14% e chegou a 18%. A utilização da placa de rede teve picos de recepção de dados de 7% e ficou quase em 0% no envio de dados.

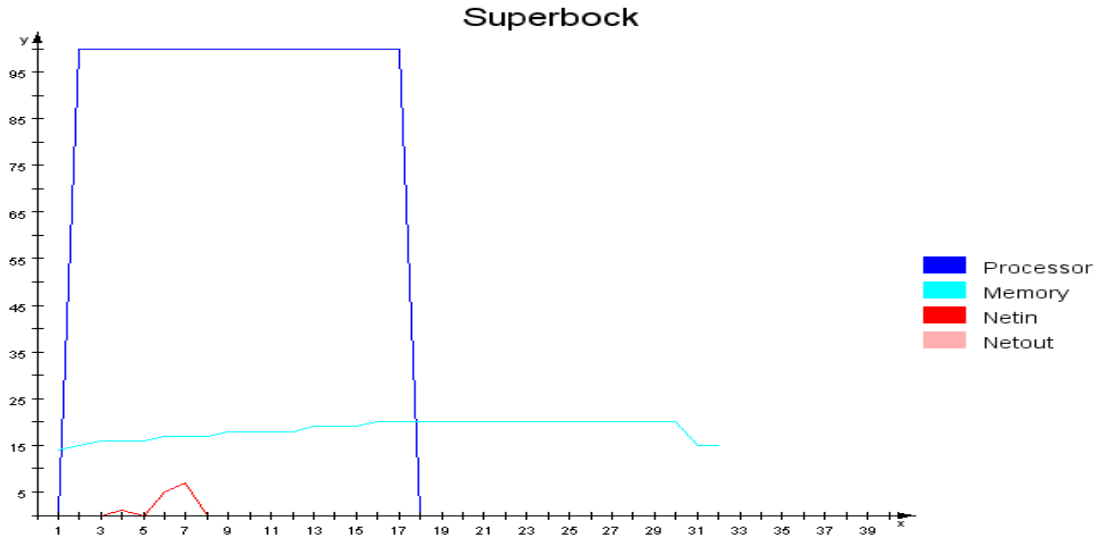

Figura 6.25 – Computador Superbock caso 6.

A tabela 6.2 consolida as informações constantes nos gráficos de utilização de processador, memória, entrada e saída de rede dos computadores Amstel, Budweiser,

Erdinger e Superbock para o caso 1. Pode-se verificar na tabela que a utilização de entrada e saída de rede foi constante em todos os computadores e igual a 0 %, pois não ocorre gravação de dados no disco rígido do computador remoto. A utilização de memória ficou entre 12% a 20% em todos os computadores, sendo que o computador Budweiser teve a menor utilização final (15%) e o computador Superbock a maior utilização inicial com 15%. A utilização de processador foi constante de 100%. É importante ressaltar que como o processamento executado nos quatro computadores tem característica de processamento matemático intensivo e sem gravação parcial de informações em disco, verificou-se através das informações de monitoração coletadas que a ocupação dos processadores é feita de forma contínua.

Tabela 6.2 – Consolidação dos dados de utilização caso 1.

	Processador	Memória	Entrada de rede	Saída de rede
Amstel	**100%**	**12% - 20%**	**0%**	**0%**
Budweiser	**100%**	**12% - 15%**	**0%**	**0%**
Erdinger	**100%**	**12% - 20%**	**0%**	**0%**
Superbock	**100%**	**15% - 19%**	**0%**	**0%**

A tabela 6.3 consolida as informações constantes nos gráficos de utilização de processador, memória, entrada e saída de rede dos computadores Amstel, Budweiser, Erdinger e Superbock para o caso 2. Pode-se verificar na tabela que a utilização de entrada e saída de rede foi quase 0% em todos os computadores, com exceção da utilização da saída da placa de rede do computador Budweiser que chegou a 1% e na utilização de entrada chegou a 7%, provavelmente pelo fato dos computadores não estarem em rede isolada. A utilização de memória ficou em torno de 12% a 16%, sendo que os computadores Amstel e Budweiser tiveram a menor utilização (12%), o computador Superbock (15%) e a maior utilização do computador Erdinger com 16%. A utilização de processador variou de 0% a 100%, devido ao processamento ter sido realizado sem gravação de dados intermediários em disco, sendo que apenas o computador Erdinger não chegou a utilizar toda a capacidade de processamento (ficou entre 84% e 96%).

Tabela 6.3 – Consolidação dos dados de utilização caso 2.

	Processador	Memória	Entrada de rede	Saída de rede
Amstel	**0% - 100%**	**12%**	**0%**	**0%**
Budweiser	**0% - 100%**	**12%**	**0% - 7%**	**0% - 1%**
Erdinger	**84% - 96%**	**16%**	**0%**	**0%**
Superbock	**100%**	**15%**	**0%**	**0%**

A tabela 6.4 consolida as informações constantes nos gráficos de utilização de processador, memória, entrada e saída de rede dos computadores Amstel, Budweiser, Erdinger e Superbock para o caso 3. Pode-se verificar na tabela que a utilização de entrada e saída de rede variou de 0% a 5%, sendo que o computador Amstel teve a maior utilização (5%), os computadores Budweiser e Erdinger a menor utilização (3%) e o computador Superbock com uma utilização média (4%). A utilização de memória ficou em torno de 12% a 20%, sendo que os computadores Amstel e Budweiser tiveram a menor utilização inicial (12%) e o computador Amstel a maior utilização final com 20%. A utilização de processador variou de 0% a 100%, sendo que os computadores Amstel e Budweiser iniciaram o processamento com carga de 35%, o computador Superbock com 12% e o computador Erdinger iniciou com 0%, devido ao processamento ter sido realizado com gravação de dados intermediários em disco, o que ocasiona a interrupção do processamento para gravação de dados intermediários.

Tabela 6.4 – Consolidação dos dados de utilização caso 3.

	Processador	Memória	Entrada de rede	Saída de rede
Amstel	**35% - 100%**	**12% - 20%**	**0% - 5%**	**0% - 5%**
Budweiser	**35% - 100%**	**12% - 15%**	**0% - 3%**	**0% - 3%**
Erdinger	**0% - 100%**	**16% - 19%**	**0% - 3%**	**0% - 3%**
Superbock	**12% - 100%**	**15% - 19%**	**0% - 4%**	**0% - 4%**

A tabela 6.5 consolida as informações constantes nos gráficos de utilização de processador, memória, entrada e saída de rede dos computadores Amstel, Budweiser, Erdinger e Superbock para o caso 4. Pode-se verificar na tabela que a utilização de entrada e saída de rede foi quase constante em todos os computadores (0% a 2%), com exceção do computador Amstel que apresentou utilização de 0% a 4%. A utilização de memória ficou em torno de 12% a 16%, sendo que o computador Amstel apresentou a menor utilização (12%) e o computador Erdinger a maior utilização (16%). A utilização de processador variou de 0% a 95%, sendo que nenhum computador utilizou a capacidade total de processamento. Este fato se deve a necessidade da interrupção do processamento para efetuar a gravação dos dados parciais no disco remoto.

Tabela 6.5 – Consolidação dos dados de utilização caso 4.

	Processador	Memória	Entrada de rede	Saída de rede
Amstel	**46% - 84%**	**12%**	**0% - 4%**	**0% - 4%**
Budweiser	**62% - 95%**	**13%**	**0% - 2%**	**0% - 2%**
Erdinger	**0% - 72%**	**16%**	**0% - 2%**	**0% - 2%**
Superbock	**38 – 72%**	**14%**	**0% - 2%**	**0% - 2%**

A tabela 6.6 consolida as informações constantes nos gráficos de utilização de processador, memória, entrada e saída de rede dos computadores Amstel, Budweiser, Erdinger e Superbock para o caso 5. Pode-se verificar na tabela que a utilização de entrada e saída de rede foi variada (de 0% a 9%), com a menor utilização tendo ocorrido no computador Erdinger (3%). A utilização de memória ficou em torno de 13% a 21%, sendo que os computadores Amstel e Budweiser tiveram a menor utilização (13%) e o computador Amstel a maior utilização com 21%. A utilização de processador variou de 0% a 100%, devido ao processamento ter sido realizado com gravação de dados intermediários em disco, o que ocasiona a interrupção do processamento para gravação de dados intermediários.

Tabela 6.6 – Consolidação dos dados de utilização caso 5.

	Processador	Memória	Entrada de rede	Saída de rede
Amstel	**35% - 100%**	**13% - 21%**	**0% - 9%**	**0% - 9%**
Budweiser	**35% - 100%**	**13% - 16%**	**0% - 9%**	**0% - 4%**
Erdinger	**0% - 100%**	**16% - 20%**	**0% - 3%**	**0% - 3%**
Superbock	**14% - 100%**	**14% - 18%**	**0% - 5%**	**0% - 5%**

A tabela 6.7 consolida as informações constantes nos gráficos de utilização de processador, memória, entrada e saída de rede dos computadores Amstel, Budweiser, Erdinger e Superbock para o caso 6. Pode-se verificar na tabela que a utilização de entrada e saída de rede variou de 0% a 7%, com exceção da utilização da placa de rede dos computadores Amstel e Erdinger que permaneceram em 0%. A utilização de memória ficou em torno de 14% a 21%. A utilização de processador foi constante em 100%, devido ao processamento ter sido realizado sem gravação de dados intermediários em disco.

Tabela 6.7 – Consolidação dos dados de utilização caso 6.

	Processador	Memória	Entrada de rede	Saída de rede
Amstel	**100%**	**17% - 21%**	**0%**	**0%**
Budweiser	**100%**	**14% - 16%**	**0% - 7%**	**0% - 7%**
Erdinger	**100%**	**16% - 19%**	**0%**	**0%**
Superbock	**100%**	**14% - 18%**	**0% - 7%**	**0%**

A tabela 6.8 consolida as informações referentes à quantidade de resoluções que foram executadas em cada computador para cada caso (informação fornecida pelo aplicativo de gerenciamento da distribuição de resoluções entre os computadores), bem como o tempo

total de duração de cada resolução em cada computador. A quantidade de resoluções varia em todos os computadores para os casos 1 a 4, pois a distribuição de resoluções para estes casos é heterogênea e somente para os casos 5 e 6 a quantidade de resoluções é a mesma. Verifica-se que apesar da quantidade de resoluções variar em cada computador, durante a execução do caso 2 (resolução de primeira ordem e sem gravação de dados em disco) o tempo de resolução (quatro unidades de tempo) é praticamente o mesmo em todos os computadores e durante a execução do caso 4 (resolução de primeira ordem e com gravação de dados em disco) o tempo de resolução (sete) é praticamente o mesmo em todos os computadores, mas é o dobro do tempo necessário no caso 2. Dessa forma, pode-se concluir que, apesar da distribuição ser heterogênea, a duração da execução é homogênea e deve-se avaliar a necessidade de gravação de dados em disco durante o processo de execução das resoluções, pois aumenta em 100% o tempo de execução do caso 2 em relação ao caso 4.

Uma avaliação semelhante é feita na comparação da execução do caso 1 (resolução de segunda ordem e sem gravação em disco) em relação ao caso 3 (resolução de segunda ordem e com gravação em disco), pois no caso 3 a execução das resoluções demorou de 20 a 24 unidades de tempo (cada unidade corresponde a 8 segundos) e durante a execução do caso 3 a execução das resoluções demorou de 30 a 36 unidades de tempo, o que corresponde a um aumento de 50%. Este aumento se deve a gravação de dados em disco durante a execução do caso 3, o qual interrompe o processamento contínuo das resoluções. Também para o caso 3 deve-se avaliar a necessidade de gravação dos dados em disco, pois o aumento de tempo é considerável.

Na avaliação da execução dos casos 5 (resolução de segunda ordem, homogênea e com gravação em disco) e do caso 6 (resolução de segunda ordem, homogênea e sem gravação em disco), verifica-se uma distribuição de tempo variada entre os computadores no caso 5 (de 28 a 43 unidades de tempo) e no caso 6 (de 18 a 31 unidades de tempo). A variação de tempo em cada caso deve-se à capacidade diferente de processamento em cada computador. O aumento de tempo de 50% no caso 5 em relação ao caso 6, se deve à gravação de dados em disco, devendo ser reavaliado ou modificado o processo de gravação dos dados.

Tabela 6.8 – Consolidação dos dados de resolução/duração.

	Caso 1	Caso 2	Caso 3	Caso 4	Caso 5	Caso 6
Amstel	**25/24**	**25/4**	**24/36**	**22/7**	**25/39**	**25/23**
Budweiser	**26/24**	**25/4**	**24/34**	**21/7**	**25/43**	**25/31**
Erdinger	**31/20**	**31/3**	**31/30**	**31/7**	**25/28**	**25/19**
Superbock	**24/23**	**25/4**	**26/35**	**29/7**	**25/28**	**25/18**

6.4 Conclusão

Nesse capítulo foram apresentadas as características principais dos computadores que fizeram parte do ambiente de testes da aplicação de monitoração, bem como os resultados

obtidos com a realização dos testes e as dificuldades de apresentação das informações coletadas com a utilização da API JOpenChart.

A partir da análise dos dados coletados, verifica-se que a utilização de placa de rede e de memória é muito baixa e que a utilização de processador é alta, ficando na maior parte do tempo em 100%. Constata-se também que o processamento não ocupa 100% do processador apenas durante execução do caso 4, devido à pequena quantidade de equações que devem se executadas e à necessidade de gravação de dados em disco remoto ser predominante. Também se verifica a ocorrência de intervalos de queda de ocupação do processador que duram em média 8 segundos, seguidos de outros 8 segundos de processamento, nos casos em que ocorre gravação de dados em arquivos em disco remoto (casos 3, 4 e 5). Dessa forma, verificações devem ser feitas no algoritmo de distribuição de carga para permitir a utilização mais intensiva de processador, o que pode ocasionar a redução do tempo total de processamento no *cluster* de servidores.

Finalmente, a comparação entre os casos 1 e 2 ou 3 e 4 permite verificar que as resoluções com elementos de segunda ordem resultam em um aumento gradativo do uso de memória do computador ao longo do processo. Este fato é um forte indício de que está ocorrendo vazamento de memória (memória alocada e não devolvida ao sistema) no algoritmo que utiliza elementos de segunda ordem.

O próximo capítulo apresenta as considerações finais do presente trabalho, as contribuições obtidas com a implementação da aplicação de monitoração e propõe alguns itens para a realização de futuras pesquisas.

7 CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as conclusões obtidas com a realização do presente trabalho de dissertação. Também estão incluídas as contribuições que foram obtidas por meio dos estudos e testes realizados com o desenvolvimento da aplicação de monitoração de *cluster* de servidores e por derradeiro as sugestões para o desenvolvimento de pesquisas futuras para expansão das características desenvolvidas neste trabalho, bem como a exploração de áreas não cobertas por este trabalho.

7.1 Introdução

O objetivo desse trabalho foi o de efetuar a avaliação da arquitetura de monitoração HTTP/XML no acompanhamento do desempenho de um *cluster* de servidores. Para atingir este objetivo optou-se pela especificação e o desenvolvimento de uma aplicação de monitoração HTTP/XML, a partir dos conceitos definidos em diversos trabalhos e especificações da comunidade acadêmica, conforme Flatin (2002), Choi (2004), Ju et alii (2002) e Strauß e Klie (2003).

Para avaliar o funcionamento da aplicação de monitoração foi utilizado, como ambiente de testes, um *cluster* de servidores que efetuam a resolução de sistemas de equações matemáticas obtidas através da modelagem numérica da análise de dispositivos eletromagnéticos.

O desenvolvimento da aplicação de monitoração e os testes de acompanhamento de carga no cluster de servidores foram realizados com sucesso e também se indica a sugestão de revisão do processo de divisão do sistema de equações, com a finalidade de tornar contínua a utilização da capacidade do processador dos computadores componentes do *cluster*.

7.2 Contribuições

Como o presente trabalho comprova-se a viabilidade da especificação, do desenvolvimento e da utilização de uma aplicação de monitoração que faz uso de padrões HTTP/XML para monitoração de servidores, em substituição ao padrão SNMP atualmente utilizado na maioria dos ambientes computacionais de empresas e universidades.

Com a aplicação de monitoração construída, foram realizados testes para avaliação da adequação da aplicação de monitoração no acompanhamento do desempenho de um *cluster* de servidores e com foco principal no acompanhamento de quatro indicadores:

- Processador;
- Memória;
- Entrada de rede;
- Saída de rede.

Os dados de utilização obtidos a partir da monitoração do cluster de servidores poderão ser utilizados para a realização de novos testes para a realização de ajustes nos algoritmos que efetuam a divisão e a carga das matrizes que são processadas no *cluster* de servidores, pois já foi identificado que, em algumas situações, a utilização de processador não ocorre de maneira contínua, durante o processamento das matrizes.

Os resultados obtidos na especificação da aplicação de monitoração também podem servir de subsídio para a continuação de novos trabalhos na área de monitoração de redes e sistemas com o uso de padrões HTTP/XML.

7.3 Sugestões para Pesquisas Futuras

Durante o desenvolvimento do presente trabalho foi verificado que algumas alterações e extensões podem ser pesquisadas, com o objetivo de tornar a aplicação de monitoração mais eficiente e também para dar continuidade a aspectos que ainda não estão bem definidos nas propostas desenvolvidas pela comunidade acadêmica e entre as quais cita-se:

- Descoberta e seleção automática da língua do idioma utilizado no sistema, com a utilização correta da nomenclatura de contadores, evitando-se a configuração do arquivo de parâmetros adicionais;

- Melhoria da interface gráfica de monitoração, permitindo a melhor visualização das informações coletadas;

- Desenvolver um agente de monitoração para o sistema operacional Linux;

- Implementação das camadas de descrição de serviços e publicação e descobrimento de serviços, conforme definido na arquitetura de *Web Services*, conforme Flatin (2002);

- Substituição do sistema de gerenciamento de base de dados relacional para o modelo de base de dados de documentos, com a implementação do *software* livre Xindice, disponível no sítio http://xml.apache.org/xindice/, conforme Apache (2005).

REFERÊNCIAS

ALVES FILHO, A. **Elementos finitos: a base da tecnologia CAE**. Editora Érica, São Paulo, 2000.

ANDOH, A. ; NASH, S. **RMI over IIOP**. 1999. Disponível em http://www.javaworld.com. Acessado em 08/06/2005.

APACHE SOFTWARE FOUNDATION. **Apache XINDICE.** Disponível em http://xml.apache.org/xindice/. Acessado em 08/08/2005.

ARGONNE NATIONAL LABORATORY. **The MPICH Home Page.** Disponível em http://www-unix.mcs.anl.gov/mpi/mpich/. Acessado em 03/08/2005.

CASE, J.; FEDOR, M. ; SCHOFFSTALL, M. ; DAVIN, J. **RFC1157 – A Simple Network Management Protocol**. May 1990, 35 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

CHOI, M. J. **An Architectural Framework for XML-based Network Management.** PHD Thesis**,** Pohang, Korea, 2004, 133 p.

COSTA, M.; PALIN, M.; CARDOSO, J.; VERARDI, S. Processamento paralelo aplicado no cálculo parametrizado por elementos finitos. **In: MOMAG 2004 – 11º SBMO – Simpósio Brasileiro de Microondas e Optoeletrônica / 6º CBMAG – Congresso Brasileiro de Eletromagnestismo**, **2004.** São Paulo. 2004. CD-ROM.

DMTF. **WBEM Profile Template.** March 2004. Disponível em : http://www.dmtf.org/wbem. Acessado em 06/07/2005.

ECLIPSE. **ECLIPSE**. Disponível em : http://www.eclipse.org/. Acessado em 06/07/2005.

FLATIN, J.P.M. Push vs. Pull in Web-Based Network. **Proc. 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99).** Boston, MA, USA, May 1999, pp. 3–18. IEEE Press, 1999.

FLATIN, J. P.M. **Web-Based Management of IP Networks and Systems.** 1º ed., Wiley, September 2002, 272 p.

HALL, M. ; BROWN, L. **Core Servlets and Java Server Pages - Volume I-Core Technologies.** Prentice Hall, 2003, 736 p.

HENDRICKS, M.; GALBRAITH, B.; IRANI, R.; MILBERNY, J.; MODI, T.; TOST, A; TOUSSAINT, A.; BASHA, S.; CABLE, S. **Profissional Java Web Service.** Editora Alta Books, Rio de Janeiro, 2002, 536 p.

HORSTMANN, C. S. ; CORNELL, G. **Core Java 2: Volume II-Advanced Features.** Prentice Hall, New Jersey, 2001, 1232 p.

IBM. **Java Development User Guide for Eclipse.** June 2002. Disponível em : http://www.eclipse.org. Acessado em 06/07/2005.

ITU-T. International Telecommunication Union. **Recommendation X.690 – Infomation technology – ASN.1 encoding rules: Specification of Basis Encoding Rules (BER), Canonical Enconding Rules (CER) and Distinguished Enconding Rules (DER).** 07/2002.

JOHN, A.; VANDERVEEN, K.; SUGLA, B. XNAMI-An extensible XML-based paradigm for network and application management instrumentation. **Proc. of IEEE International Conference of Network.** 1999, pp. 115-124**.**

JU, H. T.; CHOI, M. J.; HAN, S.; OH, Y.; YOON, J. H.; LEE, H. and HONG, J. W. An embedded Web Server Architecture for XML-Based network management, **IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)**; April,2002, Florence, Italy 5-18.

JUNIPER NETWORKS. **JUNOScript API Software**. Disponível em http://www.juniper.net/support/junoscript. Acessado em 13/10/2004.

MCLAUGHLIN, B. **Java and XML**. 1º ed., O'Reilly & Associates Inc., Sebastopol, CA, 2001, 362 p.

MAURO, D. ; SCHMIDT, K. **Essential SNMP.** 1º ed., O´Reilley, July 2001, 291 p.

McCLOGHRIE, K.; ROSE, M. **RFC1066 - Management Information Base for Network Management of TCP/IP-based internets**. March 1988, 89 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

McCLOGHRIE, K. **RFC2011 – Management Information Base for the Internet Protocol using SMIv2**. November 1996, 18 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

McCLOGHRIE, K. **RFC2012 – Management Information Base for the Transmission Control Protocol using SMIv2**. November 1996, 10 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

McCLOGHRIE, K. **RFC2013 – Management Information Base for the User Datagram Protocol using SMIv2**. November 1996, 6 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

McCLOGHRIE, K. ; ROSE, M. **RFC1213 – Management Information Base for Network Management of TCP/IP based internets: MIB-II**. March 1991, 43 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

McCLOGHRIE, K. ; PERKINS, D ; SCHOENWAELDER, J. **RFC2578 – Structure of Management Information Version 2 (SMIv2)**. April 1999, 42 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

MUELLER, S. **JOpenChart Developer Guide**. April 2002, 7 p. Disponível em http://sourceforge.net. Acessado em 20/06/2005.

NAGIOS. **Nagios.** Disponível em http://www.nagios.org. Acessado em 06/07/2005.

NIST. **Integration Definition for Function Modeling.** December 1993. Disponível em http://www.itl.nist.gov/fipspubs/idef02.doc. Acessado em 06/07/2005.

NSCLIENT. **NSClient.** Disponível em http://nsclient.ready2run.nl. Acessado em 06/07/2005.

OASIS. **Web Services/SOA**. Disponível em http://www.oasis-open.org. Acessado em 06/07/2005.

POSTGRESQL. **PostgreSQL 8.0.3 Documentation**. Disponível em http://www.postgresql.org. Acessado em 06/07/2005.

OETIKER, T. **Multi Router Traffic Grapher**. June 2005. Disponível em http://www.mrtg.org. Acessado em 06/07/2005.

ROSE, M.; McCLOGHRIE, K. **RFC1155 – Structure and Identification of Management Information for TCP/IP-based Internets**. May 1990, 22 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

ROSE, M. **RFC1158 – Management Information Base for Network Management of TCP/IP-based internets: MIB-II**. May 1990, 133 p. Disponível em http://www.ietf.org. Acessado em 01/10/2004.

STALLINGS, W. **SNMP, SNMPV2, SNMPV3, and RMON 1 and 2.** 3º ed., Addison Wesley Longman, 1999, 640 p.

STRAUß, F.; KLIE, T. Towards XML oriented Internet Management. **Proc. 8th IFIP/IEEE International Symposium on Integrated Network Management**, Colorado Springs, pages 505-518, March 2003.

SUN MICROSYSTEMS. **The Java Tutorial – a practical guide for programmers.** Disponível em http://java.sun.com/docs/books/tutorial/index.html. Acessado em 19/05/2005.

WHITEHEAD, N. **Access Windows Performance Monitor counters from Java.** November 2004. Disponível em http://www.javaworld.com. Acessado em 03/05/2005.

W3C. Extensible Markup Language (XML) 1.0, W3C Recommendation, 04 February 2004. Disponível em http://www.w3.org/TR/2004/REC-xml-20040204/. Acessado em 13/10/2004.

W3C. SOAP version 1.2 part 0:Primer, W3C Recommendation, 24 June 2003. Disponível em http://www.w3.org/TR/2003/REC-soap12-part0-20030624/. Acessado em 27/06/2005.

YOON, J.H.; JU, H. T. and HONG, J. W. Development of SNMP-XML translator and gateway for XML-based integrated network management. **International Journal of Network Management 2003**;13: 259–276 (DOI: 10.1002/nem.478).