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

Cristhian Flamariom Gomes de Carvalho

MODELO DE INTEGRAÇÃO DO WAP NO APOIO À GERÊNCIA DE REDES TCP/IP

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

Prof. Dr. Carlos Becker Westphall

Florianópolis, Agosto de 2002

MODELO DE INTEGRAÇÃO DO WAP NO APOIO À GERÊNCIA DE REDES TCP/IP

Cristhian Flamariom Gomes de Carvalho

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

_____
Prof. Dr. Fernando Álvaro Ostuni Gauthier

Banca Examinadora:

_____
Prof. Dr. Carlos Becker Westphall

_____
Prof. Dra. Carla Merkle Westphall

_____
Prof. Dr. Marcelo Menezes Reis

"O que fazemos
durante as horas de trabalho
determina o que temos;
o que fazemos
nas horas de lazer
determina o que somos".

Charles Schulz

Ofereço aos meus pais Elicio e Cessy, e aos meus irmãos James e Helen que sempre me compreenderam e me apoiaram em todos os sentidos, ao longo deste trabalho e da minha vida. A minha amiga e companheira Maria do Carmo que me fez enxergar a vida com novas nuances.

Agradeço ao meu orientador, pela ajuda, paciência e confiança que teve por mim ao longo deste trabalho.
A toda equipe do LRG, pelas dicas que me foram passadas nas horas mais oportunas.
Aos membros da banca examinadora pela contribuição científica.
Aos professores que eu conheci no PPGCC.
A todos da secretaria do PPGCC que me ajudaram, principalmente, quando fui representante discente.
A todos os meus amigos, pela amizade, pelo apoio e pela convivência harmoniosa.
E, principalmente, a Deus, por permitir que eu chegasse até aqui.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS E ABREVIATURAS

API	*Application Programming Interface*
ARM	*Admin Runtime Module*
ASN.1	*Abstract Syntax Notation 1*
ASP	*Active Server Pages*
ATM	*Assynchronus Transfer Mode.*
ATP	*Agent Transfer Protocol*
CCITT	*Consultative Committee for International Telegraph and Telephone*
CDMA	*Code Division Multiple Access*
CDPD	*Cellular Digital Packet Data*
CGI	*Common Gateway Interface*
CIM	*Common Information Model*
CMIP	*Common Management Information Protocol*
CSD	*Circuit Switched Data*
CTIA	*Cellular Telecommunications Industry Association*
EIA	*Electronic Industry Association*
ETSI	*European Telecommunication Standardisation Institute*
FTP	*File Transfer Protocol.*
GPRS	*General Packet Radio Service*
GSM	*Global System for Mobile Communication*
GUI	*Graphical User Interface*
HDML	*Handheld Markup Language*
HMMP	*HyperMedia Management Protocol*
HMOM	*HyperMedia Object Manager*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
ICMP	*Internet Control Message Protocol.*
IEEE	*Institute of Eletrical and Eletronics Engineers.*

IETF	*Internet Engineering Task Force.*
IMT-2000	*International Mobile Telecommunications 2000*
IP	*Internet Protocol*
ISO	*International Standards Organization*
ITU	*International Telecommunication Union*
LAN	*Local Area Network*
MAN	*Metropolitan Area Network*
MIB	*Management Information Base.*
MMI	*Man-Machine Interface*
NMS	*Network Management Station*
OSI	*Open System Interconnection*
OTA	*Over The Air*
PAP	*Push Access Protocol*
PCS	*Personal Communication Services*
PDA	*Personal Digital Assistant*
PDU	*Protocol Data Unit*
PHP	*PHP: Hypertext Preprocessor*
PIN	*Personal Identification Number*
PPG	*Push Proxy Gateway*
QoS	*Quality of Service*
RFC	*Request For Comments*
RMI	*Remote Method Invocation*
RSA	*RSA (Rivest, Shamir, Adleman) public key algorithm*
SAP	*Service Access Point*
SAR	*Segmentation and Reassembly*
SMIv1	*Structure of Management Information Version 1*
SMS	*Short Message Service*
SNMP	*Simple Network Management Protocol.*
SQL	*Structured Query Language*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol.*
TCP/IP	Conjunto de protocolos da arquitetura de rede da Internet.

TDMA	*Time Division Multiple Access*
Telnet	Serviço de acesso remoto.
TIA	*Telecommunications Industry Association*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol.*
UFSC	Universidade Federal de Santa Catarina.
UMTS	*Universal Mobile Telephony Service*
URL	*Uniform Resource Locator*
USSD	*Unstructured Supplementary Service Data*
W3C	*World Wide Web Consortium*
WAE	*Wireless Application Environment*
WAN	*Wide Area Network*
WAP	*Wireless Application Protocol*
WBEM	*Web-Based Enterprise Management*
WDP	*Wireless Datagram Protocol*
WML	*Wireless Markup Language*
WMLS	*Wireless Markup Language Script*
WSP	*Wireless Session Protocol*
WTA	*Wireless Telephony Application*
WTLS	*Wireless Transport Layer Security*
WTP	*Wireless Transaction Protocol*
WWW	*World-Wide Web*
XML	*Extensible Mark-up Language*

RESUMO

Com o crescimento do uso de redes de computadores e do aumento da complexidade das aplicações envolvidas, a importância do uso de mecanismos e sistemas de gerenciamento de redes vem aumentando ao longo dos últimos anos. Várias arquiteturas e ferramentas de gerência de redes foram desenvolvidas, se destacando o modelo SNMP e, atualmente, o modelo de gerência WEB. O gerenciamento via WEB, trouxe a mobilidade ao administrador, no que se refere à interface entre o usuário que gerencia uma rede e o sistema de gerência da rede. Com o surgimento da tecnologia WAP, que é um conjunto de especificações para padronizar as interações entre equipamentos portáteis, como telefones celulares, e a Internet; há a possibilidade de incrementar a mobilidade promovida pela gerência via WEB: o WAP trouxe a característica da Mobilidade Sem Fio. Agora o administrador ou gerente de uma rede, poderá utilizar um telefone celular, por exemplo, para acompanhar o estado dessa rede. Este trabalho propôs um modelo de integração da tecnologia WAP, com o gerenciamento de redes TCP/IP, utilizando o protocolo SNMP e ferramentas gratuitas ou de domínio público, considerando as características do WAP e as limitações dos dispositivos, como telefones celulares.

ABSTRACT

With the growth of the use of computers networks and of the increase of the complexity of the involved applications, the importance of the use of mechanisms and systems of networks administration have been increasing along the last years. Several architectures and tools of management of networks were developed, standing out the model SNMP and, now, the model of WEB management. The administration through WEB, brought the mobility to the administrator, in the what if refers to the interface among the user that management a network and the system of management of the network. With the appearance of the WAP technology, that is a group of specifications to standardize the interactions among portable equipments, as cellular telephones and Internet; there is the possibility to increase the mobility promoted by the management through WEB: WAP brought the characteristic of the Wireless Mobility. Now the administrator or network manager could use a cellular telephone, for example, for to accompany the state of that network. This work proposed a model of integration of the WAP technology with the administration of TCP/IP networks, using the protocol SNMP and free tools or of public domain, considering the characteristics of WAP and the limitations of the devices, as cellular telephones.

1. INTRODUÇÃO

Com o crescimento das redes de computadores e dos serviços providas pelas mesmas se tornando cada vez mais relevantes, o uso de sistemas de gerenciamento de redes tem se tornado indispensável. Atualmente, as redes baseadas na arquitetura TCP/IP crescem rapidamente, e a interconexão das mesmas com a Internet vem ocorrendo de forma rápida.

A ligação de redes TCP/IP com a Internet trouxe novos desafios para os administradores e gerentes de redes, principalmente questões relacionadas com a segurança e o escalonamento dos serviços prestados. Inúmeras ferramentas e modelos de gerenciamento de redes foram desenvolvidas, utilizando várias tecnologias como o protocolo de gerência SNMP (*Simple Network Management Protocol*), o paradigma dos Agentes Móveis, gerenciamento via Web e a utilização de Java e CORBA (*Common Object Request Broker Architecture*).

O gerenciamento via Web, trouxe a mobilidade ao administrador, no que se refere à interface entre o usuário que gerencia uma rede e o sistema de gerência da rede. Muitos trabalhos forma realizados utilizando o Modelo de Gerência via Web, integrado a outras tecnologias.

Em [BAR98] foi descrita a utilização em conjunto das tecnologias WWW (*World-Wide Web*), linguagem Java, CORBA e o SNMP, para a implementação de um sistema de gerência distribuído.

Em [CER99] foi implementado um ambiente de gerência para redes ATM (*Assynchronus Transfer Mode*) integrado com a tecnologia Web, no qual foi feito também um estudo detalhado das variáveis mais importantes encontradas nas MIBs (*Management Information Bases*) relacionadas ao ATM.

Em [MER97] foi apresentada uma proposta para o gerenciamento de aplicações em rede, utilizando uma abordagem de gerenciamento baseado na Web, juntamente com o protocolo SNMP em conjunto com linguagens de script, analisando também as áreas funcionais de gerenciamento do modelo OSI (*Open System Interconnection*).

Em [AND98] foi realizado um estudo sobre as arquiteturas de gerência baseadas em Web, no qual foram apresentados quatro modelos de integração entre o modelo de gerência SNMP e a Web.

Em [SAU99] foi apresentado o projeto de uma aplicação de gerenciamento de rede baseado na Web, chamada de *WebManager*. A aplicação *WebManager* é uma ferramenta que oferece um serviço de gerência de rede de computadores baseada em Web para monitorar dispositivos como impressoras, estações e roteadores dentro de uma rede TCP/IP.

Analisando estes trabalhos, verificou-se que foi utilizado MIBs do modelo de gerência SNMP, o protocolo SNMP em conjunto com algum mecanismo de leitura dos valores dos objetos gerenciados nas MIBs e a utilização de Web *Browsers* para acompanhamento, visualização ou monitoramento do sistema.

Com o surgimento da tecnologia WAP, que é um conjunto de especificações para padronizar as interações entre equipamentos portáteis, como telefones celulares e PDAs, e a Internet; surgiu a possibilidade de incrementar a mobilidade promovida pela gerência via Web: o WAP trouxe a característica da Mobilidade Sem Fio. Agora o administrador ou gerente de uma rede, poderá utilizar um telefone celular, por exemplo, para acompanhar o estado da rede.

Este trabalho propôs um modelo de integração da tecnologia WAP com as tecnologias existentes, para o gerenciamento de redes TCP/IP, levando-se em consideração as características do WAP e as limitações encontradas nos dispositivos móveis, como os telefones celulares, por exemplo.

1.1. MOTIVAÇÕES

- O surgimento da tecnologia WAP, dando novas possibilidades na geração de novos aplicativos;
- A possibilidade de agregar a mobilidade sem fio para aplicações de gerência de rede, sendo este recurso muito pouco utilizado no gerenciamento de uma rede.

- A dificuldade de encontrar um meio de integração da tecnologia WAP com o modelo de gerência SNMP, utilizando somente ferramentas gratuitas ou de domínio público.

1.2. OBJETIVOS

1.2.1. OBJETIVO GERAL

Propor um modelo de integração do WAP no apoio ao gerenciamento de redes TCP/IP.

1.2.2. OBJETIVOS ESPECÍFICOS

- Identificar as tecnologias atuais para integrar o gerenciamento de redes TCP/IP com a tecnologia WAP;
- Propor uma aplicação do WAP, para interagir com um sistema de gerenciamento de rede TCP/IP;
- Propor meios de otimização da aplicação, levando-se em conta as limitações do WAP;
- Propor um modelo de aplicação que utiliza somente componentes de software gratuitos ou de domínio público.

1.3. ORGANIZAÇÃO DO TRABALHO

No capítulo 2 apresenta-se um estudo geral sobre gerência de redes, a importância do gerenciamento de uma rede, o modelo de gerência e seus componentes: gerentes, agentes, base de informação de gerenciamento e protocolos. Também é realizada uma

descrição das áreas funcionais da gerência de redes de acordo com o modelo OSI (*Open Systems Interconnection*).

No capítulo 3 apresentam-se os conceitos básicos sobre gerência de redes TCP/IP com o uso do modelo de gerenciamento SNMP, seus componentes e suas características. Também são apresentados os conceitos do gerenciamento via Web, as vantagens de sua utilização e modelos de arquitetura representantes da Gerência baseada em Web.

No capítulo 4 é apresentado o modelo da arquitetura WAP (*Wireless Application Protocol*), comparações com o modelo TCP/IP, seus componentes e suas características.

No capítulo 5 apresenta-se a proposta do modelo para o gerenciamento de redes TCP/IP com a utilização do WAP. É descrito também como e quais recursos (ambiente) serão utilizados para validação do modelo proposto.

No capítulo 6 é apresentada a análise de resultados.

No capítulo 7 são apresentadas as conclusões e recomendações futuras para este trabalho.

Por fim, no capítulo 7, é apresentada a listagem das referências bibliográficas utilizadas até aqui para realização deste trabalho.

2. GERÊNCIA DE REDES

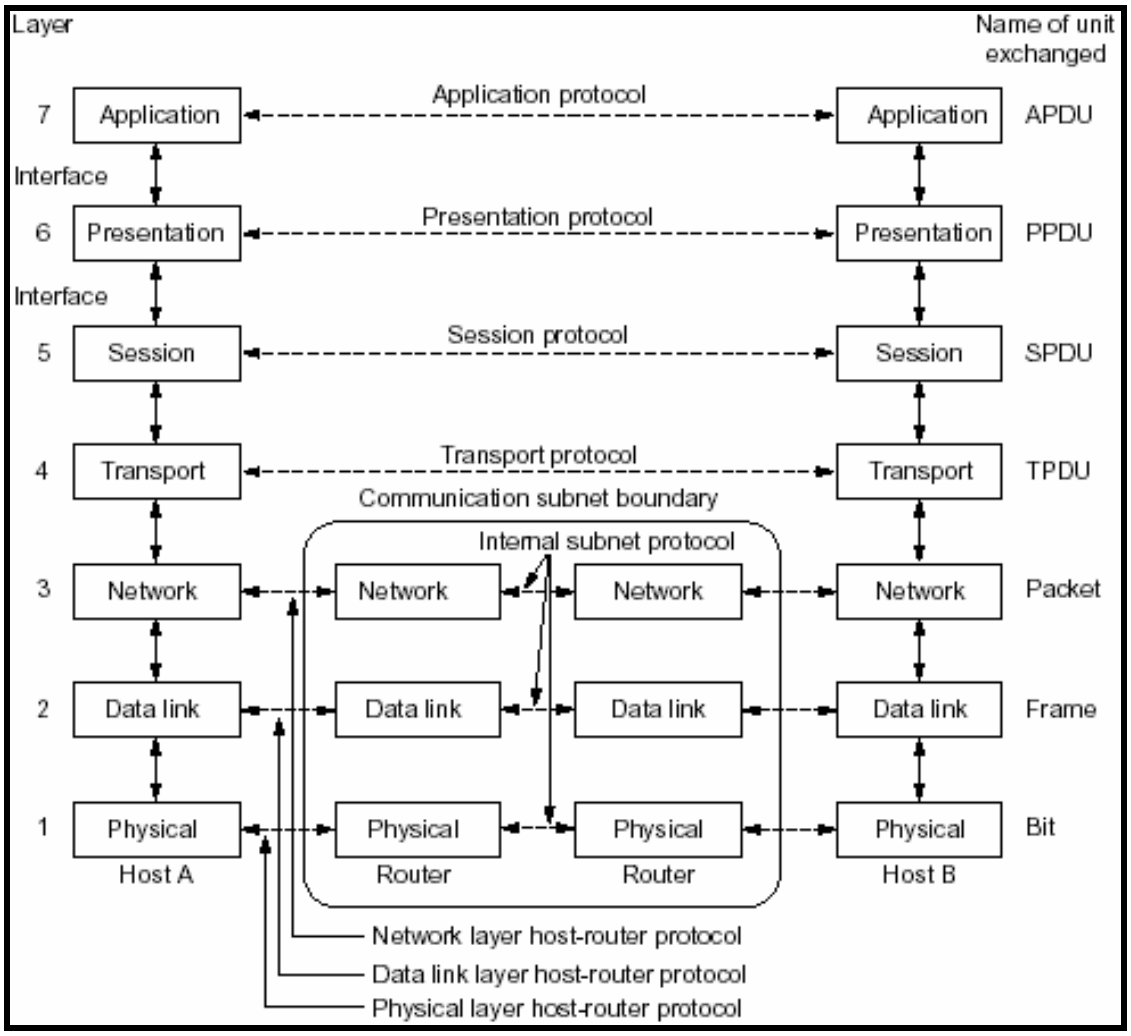
Neste capítulo é realizado um estudo geral sobre gerência de redes, a importância do gerenciamento de uma rede, o modelo de gerência e seus componentes: gerentes, agentes, base de informação de gerenciamento e protocolos. Também é realizado uma descrição das áreas funcionais da gerência de redes de acordo com o modelo OSI (*Open Systems Interconnection*).

2.1. NECESSIDADES DE GERENCIAMENTO

Anos atrás, quando os sistemas computacionais se limitavam a um ou mais *mainframes*, que detinham toda a capacidade de processamento e armazenamento de dados diversos, e os terminais que eram somente usados como uma interface entre os usuários e o computador central, o gerenciamento era facilitado devido à homogeneidade característica desses ambientes. Ao longo dos anos, houve a migração de vários sistemas centralizados para equipamentos distribuídos, heterogêneos e geograficamente dispersos [JOS99], apoiada pela criação e o aperfeiçoamento das redes de comunicação. Essa fusão dos computadores e das redes de comunicação ampliou demasiadamente a complexidade do funcionamento dos sistemas computacionais [TAN97], obrigando o desenvolvimento de sistemas de gerência eficientes.

No projeto das primeiras redes de computadores, o *hardware* foi colocado em primeiro plano em relação ao *software*, estratégia esta que foi deixada para trás ao longo dos anos. Atualmente, o *software* da rede está altamente estruturado numa arquitetura organizada como uma série de camadas ou níveis, cada uma executando um conjunto de funções. A ISO (*International Standards Organization*) propôs um Modelo de Referência chamado OSI, mostrado na Fig. 2.1. Várias arquiteturas de rede possuem uma correlação com o modelo OSI como, por exemplo, o modelo TCP/IP (*Transmission Control Protocol/Internet Protocol*).

A apresentação do modelo de referência OSI é necessária para a futura compreensão da pilha de protocolos do modelo WAP (*Wireless Application Protocol*).



Fonte: [TAN97].

FIGURA 2.1 – Modelo de Referência OSI

Atualmente o número de computadores ligados em rede cresce a todo instante, sejam organizados na forma de uma LAN (*Local Area Network*), MAN (*Metropolitan Area Network*) ou WAN (*Wide Area Network*), conectados através de dispositivos como *bridges*, *Hubs*, *switches*, roteadores, etc. Todos esses equipamentos têm como função proporcionar ou dar suporte a vários tipos de aplicações e serviços. Muitas dessas aplicações precisam funcionar de maneira estável e ininterrupta como, por exemplo, aplicações bancárias, industriais ou médicas. O crescimento do número de

equipamentos ligados em rede aumenta a complexidade do gerenciamento da mesma, forçando a adoção de ferramentas para a devida monitoração e controle.

A gerência de redes propõe soluções de monitoramento e controle de uma rede, com o objetivo de registrar a ocorrência de eventos, detectar e alterar a configuração, contabilizar a utilização de recursos, garantir a segurança, estabelecer limites para o disparo de alarmes e também detectar, diagnosticar e prevenir a ocorrência de falhas.

2.2. ÁREAS FUNCIONAIS DO GERENCIAMENTO OSI

A ISO apresenta uma divisão funcional de necessidades no processo de gerenciamento [BRI93] [JOS99], como parte de sua especificação de Gerenciamento de Sistemas OSI, apesar deste modelo ter servido de guia para diversos desenvolvedores de soluções de gerenciamento.

2.2.1. GERENCIAMENTO DE FALHAS

Uma falha é uma condição anormal cujo retorno à normalidade, é realizado através de uma ação de gerenciamento. A causa pode ser operações incorretas ou um número excessivo de erros, como por exemplo, um bit que teve seu valor alterado por algum problema na linha de comunicação. O objetivo do gerenciamento de falhas é proporcionar um funcionamento contínuo da rede e de seus serviços, através de um conjunto de funções destinadas à detecção, ao isolamento e à correção de anomalias no funcionamento do ambiente, sejam estas provocadas por um problema material ou pelo mau funcionamento de um componente de software.

As seguintes funções estão relacionadas com o gerenciamento de falhas:

- Identificar e isolar falhas;
- Reconfigurar e modificar a rede de maneira a minimizar o impacto na operação sem o componente com falha;

- Consertar ou trocar os componentes defeituosos e restaurar a rede ao seu estado inicial;

2.2.2. GERENCIAMENTO DE CONTABILIZAÇÃO

Os custos e o volume de recursos utilizados pelos usuários devem ser identificados e registrados de forma correta. O gerenciamento de contabilização provê isso através da mensuração do conjunto de facilidades de rede utilizadas. Este gerenciamento é importante, pois conhecendo as atividades dos usuários de uma rede, pode-se planejar o crescimento da mesma e evitar seu uso ineficiente.

As funções, a seguir, estão relacionadas com o gerenciamento de contabilidade:

- Ajudar o administrador no planejamento do crescimento da rede, com base no conhecimento detalhado sobre o uso dos recursos da rede pelos usuários finais;
- Auxiliar no uso mais eficiente da rede.

2.2.3. GERENCIAMENTO DE DESEMPENHO

O gerenciamento de desempenho é importante tanto para garantir a qualidade de serviço acordada com os usuários como para assegurar que esta é atingida com os menores custos possíveis. Ela fornece as funções que permitem coletar dados estatísticos, dados de desempenho como taxa de saída de octetos por segundo, por exemplo; e de manter e examinar o histórico do estado destes objetos, para um planejamento e análise do sistema.

As funções relacionadas com o gerenciamento de desempenho dizem respeito ao monitoramento e controle de:

- Tempo de resposta;
- Carga na rede;

- Vazão ou *Throughput*;
- “Gargalos” na rede.

2.2.4. GERENCIAMENTO DE SEGURANÇA

A segurança de redes é uma das áreas que mais cresce na administração de redes atualmente. O gerenciamento de segurança consiste no uso do gerenciamento de redes para monitorar e controlar mecanismos de segurança como, por exemplo, o controle de acesso aos sistemas e controle de informações sigilosas que trafegam nas redes.

As seguintes funções estão relacionadas com o gerenciamento de segurança:

- Identificar riscos de segurança e suas conseqüências;
- Implantar e manter uma estrutura de rede segura;
- Administrar senhas, controle de acesso e permissões de usuários;
- Monitorar alarmes e *logs*.

2.2.5. GERENCIAMENTO DE CONFIGURAÇÃO

Corresponde ao conjunto de funções relativas ao controle, à identificação, à supervisão e à coleta de informações sobre os objetos gerenciados. O gerenciamento de configuração está principalmente relacionado com a monitoração do *status* pré-definido dos componentes de uma rede durante a operação da mesma, pois cada recurso é configurado para executar certos serviços: servidor de arquivos, servidor WWW (*World Wide Web*), roteamento, etc.

2.3. COMO GERENCIAR

Para uma rápida compreensão da arte de gerenciar são descritos, a seguir, dois aspectos básicos na gerência de redes.

2.3.1. CATEGORIAS DAS FUNÇÕES DE GERENCIAMENTO DE REDES

As funções de gerenciamento de redes podem ser organizadas em duas categorias [JOS99]: o monitoramento da rede e o controle da rede. O monitoramento da rede pode ser definido como uma função de leitura e está relacionado com a observação e análise do estado e do comportamento da configuração da rede e de seus componentes, envolvendo três aspectos de projeto:

Acesso às informações monitoradas: define quais informações são monitoradas e a maneira de obtenção dessas informações;

Projeto dos mecanismos de monitoramento: define a estratégia de obtenção das informações dos recursos gerenciados;

Utilização das informações monitoradas: define como as informações monitoradas são utilizadas para a realização da análise e diagnóstico de problemas nas áreas funcionais de gerenciamento.

O controle da rede pode ser definido como uma função de alteração e está relacionada com a modificação de parâmetros em vários componentes da configuração, o que faz esses componentes executarem ações predefinidas.

2.3.2. MODELO CLÁSSICO DE GERÊNCIA

O sistema de gerenciamento consiste basicamente, na utilização de um computador que estabelece uma comunicação com os diversos componentes ou

dispositivos da rede, coletando informações necessárias para proporcionar o gerenciamento.

Dentro do modelo clássico de gerência, é definida uma arquitetura com os seguintes componentes: gerente, agente, base de informação de gerenciamento e o protocolo de gerência utilizado na comunicação entre o gerente e o agente. A interação ocorre sempre entre o gerente e o agente, sendo este último, responsável por consultar a base de informação de gerenciamento e enviar os dados requisitados, ou não, para o gerente.

Na gerência de redes, os dois protocolos de gerenciamento mais conhecidos são:

o CMIP (*Common Management Information Protocol*), que é utilizado na gerência de redes OSI; e o SNMP (*Simple Network Management Protocol*), que é utilizado na gerência de redes TCP/IP.

Como o escopo deste trabalho é a gerência de redes TCP/IP, é realizada uma descrição geral do funcionamento do protocolo SNMP e das partes que o compõem no Capítulo 3.

2.4. CONCLUSÃO

Atualmente, por menor e mais simples que seja, uma rede de computadores deve possuir algum mecanismo de monitoramento e/ou controle, com o objetivo de garantir a disponibilidade de serviços que são oferecidos aos usuários dessa rede.

É importante também, além do uso de ferramentas para a realização do gerenciamento de redes, é a implantação de uma política de gerenciamento, a qual define as diretrizes da gerência, o que e como gerenciar, as pessoas envolvidas, e também as próprias ferramentas utilizadas.

3. GERÊNCIA DE REDES TCP/IP

Neste capítulo serão apresentados uma breve comparação entre o modelo OSI e o TCP/IP, os conceitos básicos sobre gerência de redes TCP/IP com o uso do modelo de gerenciamento SNMP, seus componentes e suas características. Também são apresentados os conceitos e as características da integração da *Web* com o gerenciamento de rede.

3.1. O MODELO TCP/IP

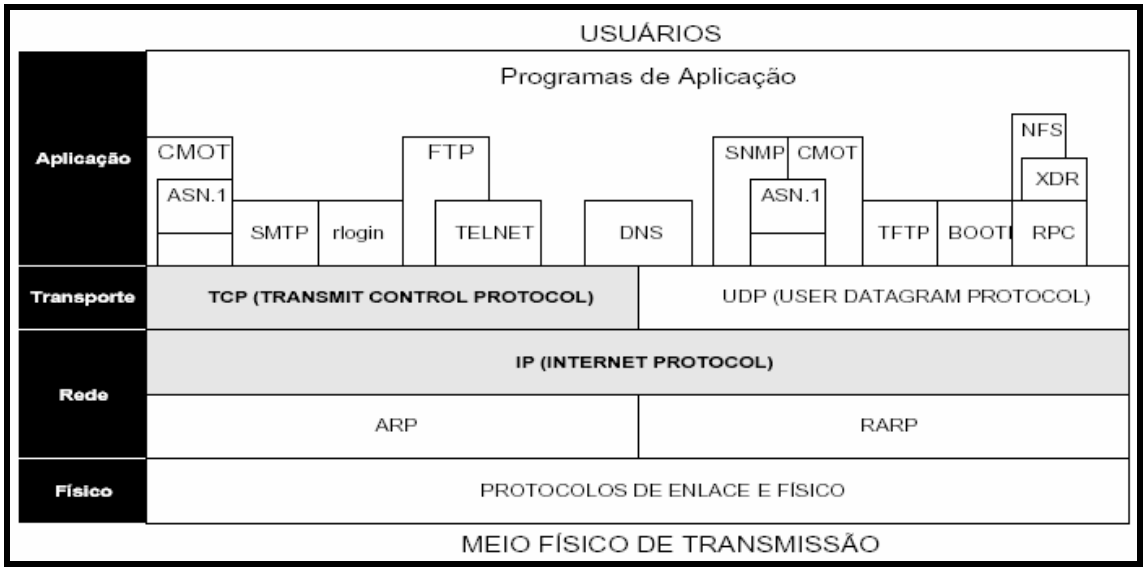
O modelo de referência OSI, juntamente com os protocolos que o compõem, demoraram certo tempo para serem desenvolvidos. Quando a Arquitetura OSI tinha sido definida, a Arquitetura TCP/IP e seus principais protocolos já existiam há dez anos [TAN97].

Atualmente, a pilha de protocolos TCP/IP representa uma das formas mais usadas de comunicação entre computadores remotos, fornecendo um sistema aberto de fato, pois suas especificações e muitas de suas implementações estão disponibilizadas publicamente. A Quadro 3.1 representa um modelo arquitetural do TCP/IP relacionado com o modelo OSI.

OSI	TCP/IP
7 APLICAÇÃO	APLICAÇÃO
6 APRESENTAÇÃO	
5 SESSÃO	
4 TRANSPORTE	TRANSPORTE
3 REDE	REDE
2 ENLACE DE DADOS	INTERFACE DA REDE
1 FÍSICA	

QUADRO 3.1 – TCP/IP versus OSI

Com o objetivo de situar o SNMP dentro da arquitetura TCP/IP, a Fig. 3.1 ilustra os principais protocolos nas camadas da pilha TCP/IP, incluindo o protocolo de gerenciamento.



Fonte: MAZZOLLA, Vitório B. Internet e Intranets, 1999

FIGURA 3.1 – Protocolos da Pilha TCP/IP

3.2. O SNMP

O SNMP é um protocolo da camada de aplicação da arquitetura de rede da Internet, como mostrado na Fig. 3.1, o qual é utilizado para monitorar e controlar tanto os recursos como os serviços proporcionados por esses recursos em uma rede de computadores. Em comparação com o protocolo CMIP do modelo de gerência OSI, o SNMP visa diminuir a complexidade e o número das funções realizadas no gerenciamento [STA99].

No modelo de gerenciamento definido pelo SNMP, há quatro elementos principais: gerenciadores; um ou mais nodos gerenciáveis, cada um contendo uma entidade SNMP chamada de agente; base de informação de gerenciamento; e o protocolo de gerenciamento [MAU01].

3.2.1. GERENCIADORES

O monitoramento e o controle das informações da rede são realizados a partir de um gerenciador ou gerente, que contém um ou mais processos que se comunicam com os agentes localizados nos nodos gerenciados. Os gerentes ou são responsáveis pelo envio de requisições aos agentes, e recebem respostas vinculadas às requisições ou, então, recebem notificações assíncronas dos agentes. Essa notificação é chamada de *trap*. A Fig. 3.2 ilustra, de maneira sucinta, o relacionamento gerente-agente.

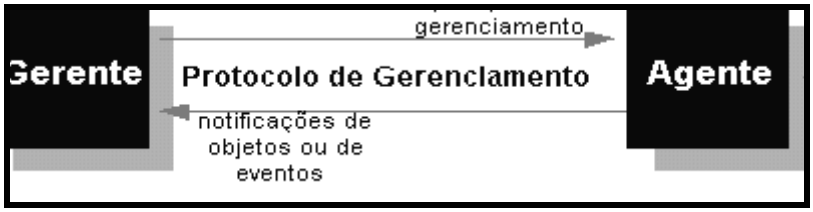


FIGURA 3.2 – Relacionamento Gerente-agente

3.2.2. NODOS GERENCIÁVEIS

Os nodos gerenciáveis podem ser qualquer equipamento conectado na rede, como: computadores, roteadores, comutadores ou *switches*, *Hubs*, *bridges*, impressoras, etc. Todos os nodos gerenciáveis devem possuir um agente SNMP para devida comunicação com um gerente qualquer. Nos nodos gerenciáveis, além dos agentes, temos uma base de informação de gerenciamento.

3.2.3. BASE DE INFORMAÇÃO DE GERENCIAMENTO

Mais conhecida como MIB (*Management Information Base*). As informações de gerenciamento do modelo SNMP relacionam-se com o conceito de “objetos gerenciados”, que corresponde a recursos físicos e lógicos passíveis de serem monitorados e/ou controlados.

O primeiro passo para entender que tipo de informação um dispositivo pode fornecer é conhecer como esses dados são representados no contexto do SNMP. A SMIv1 (*Structure of Management Information Version 1*), definida na RFC 1155, define com exatidão como os objetos gerenciados são nomeados e especifica os respectivos tipos de dados associados. Um objeto gerenciado, no modelo SNMP, é uma variável de dados. Cada objeto tem um valor e é definido por meio de um "tipo de objeto". Ao contrário do modelo OSI de gerência, o modelo SNMP não utiliza os conceitos de classes de objetos e seus respectivos atributos.

Uma característica relevante na organização dos objetos dentro da MIB é a árvore de nomenclatura da MIB, definida na ISO ASN.1 (*Abstract Syntax Notation 1*). As informações dentro da MIB estão estruturadas na forma de uma árvore de nomenclatura, conforme recomendação da ISO e CCITT (*ConsultativeCommittee for International Telegraph and Telephone*), que designa um identificador para qualquer objeto [STA99].

As árvores estão definidas em grupos e cada grupo possui um identificador representado na árvore de nomenclatura, tendo objetos ligados sob este identificador, como ilustra a Fig. 3.3.

Todos os nodos possuem um rótulo, que consiste de um identificador numérico e um texto descritivo. O identificador é uma seqüência de números inteiros que marcam o caminho a partir da raiz da árvore até o objeto. Este identificador é chamado de Identificador de Objeto ou OID (*OBJECT IDENTIFIER*).

Exemplo:

directory(1)

identificador de objetos: 1.3.6.1.1

descrição textual: iso(1).org(3).dod(6).internet(1)

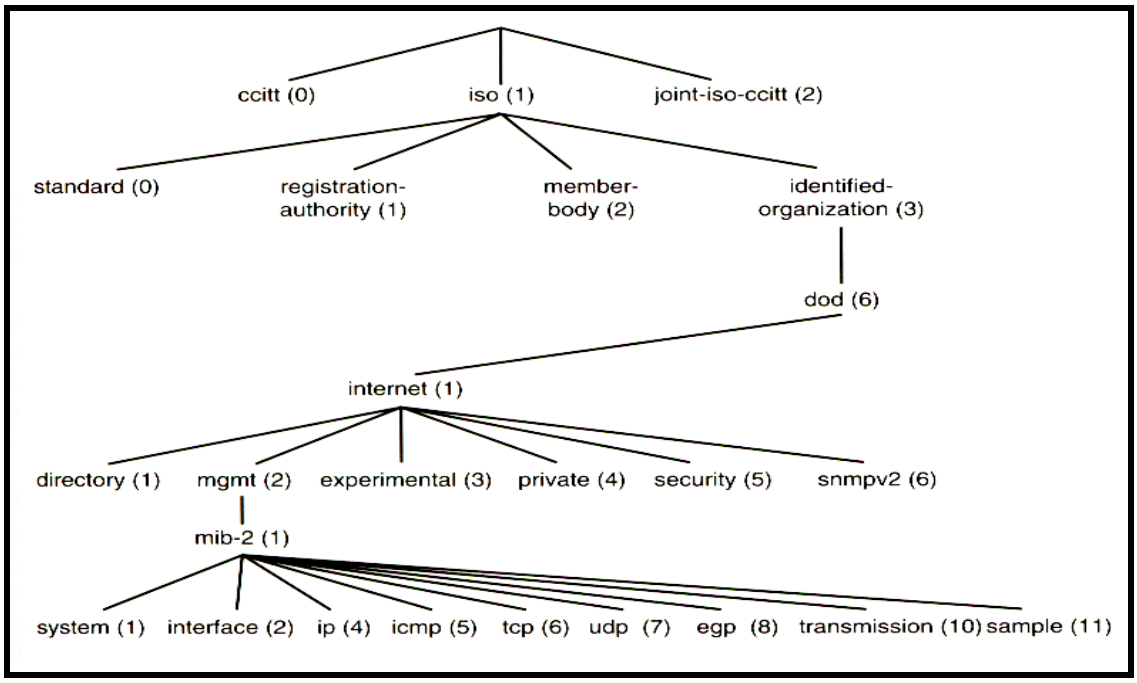


FIGURA 3.3 – Estrutura de Árvore da MIB

A MIB-II, que é especificada na RFC (*Request For Comments*) 1213, é a MIB padrão para o SNMPv1.

O nó raiz da árvore MIB não tem nome ou número, mas tem três grupos:

- ***ccitt(0)***, administrado pelo CCITT
- ***iso(1)***, administrado pela ISO
- ***joint-iso-ccitt(2)***, administrado pela ISO juntamente com o CCITT.

Sob o nó iso(1), estão outros grupos, como é o caso do grupo org(3), definida pela ISO para conter outras organizações. Uma das organizações que está sob grupo org(3) é o Departamento de Defesa dos Estados Unidos, no nó dod(6). A internet(1) está sob o dod(6), e possui quatro grupos:

- ***directory(1):*** contém informações sobre o serviço de diretórios OSI (X.500);
- ***mgmt(2):*** contém informações de gerenciamento, é neste grupo que está o da MIB-II, com o identificador de objeto 1.3.6.1.2.1.
- ***experimental(3):*** contém os objetos que ainda estão sendo pesquisados.

- ***private(4):*** contém objetos definidos por outras organizações.

3.2.4. PROTOCOLO DE GERENCIAMENTO

A comunicação entre os agentes e os gerentes é realizada através do protocolo SNMP, que possui três versões:

SNMP Versão 1: é a versão padrão atual, definida pela RFC 1157 e é um padrão completo da IETF. A segurança do SNMPv1 baseia-se em comunidades, que são como senhas: *strings* de texto puro que permitem que qualquer aplicativo baseado em SNMP (que reconheça a string) tenha acesso a informações de gerenciamento de um dispositivo. Normalmente existem três tipos de comunidades no SNMPv1: *read-only*, *read-write*, e *trap*.

SNMP Versão 2: é freqüentemente citado como SNMPv2 baseado em *strings* de comunidade. Esta versão tem a denominação técnica de SNMPv2c. Ela está definida na RFC 1905, RFC 1906 e RFC 1907, e ainda está com *status* experimental. Apesar disso, alguns fornecedores já começaram aceitá-la na prática.

SNMP Versão 3: no momento é um padrão sugerido, que inclui suporte para autenticação rigorosa e comunicação privativa entre as entidades gerenciadas.

No SNMP, o formato de mensagem que os gerenciadores e agentes utilizam para enviar e receber informações é a PDU (*Protocol Data Unit*). Existe um formato PDU padrão para cada tipo de operação.

O SNMPv1 fornece as seguintes operações:

- ***get:*** requisita o valor de um único objeto da MIB de cada vez.
- ***get-next:*** requisita os valores dos objetos de um mesmo grupo da MIB.
- ***set:*** é utilizada para modificar o valor de um objeto gerenciado da MIB.
- ***get-response:*** é a resposta contendo a informação anteriormente requisitada pela operação *get* ou *get-next*.

- ***trap:*** informação de notificação enviada pelo agente ao gerente.

A Fig. 3.4 e a Fig.3.5 ilustram as operações realizadas entre gerente e agente, exemplificando o modelo de gerência SNMP.

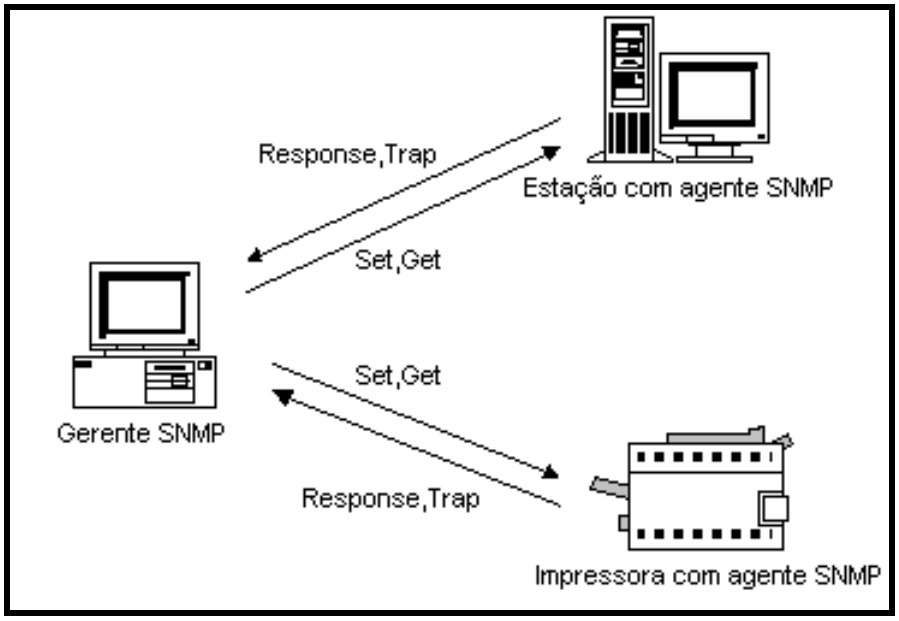


FIGURA 3.4 – Modelo de Gerência SNMP

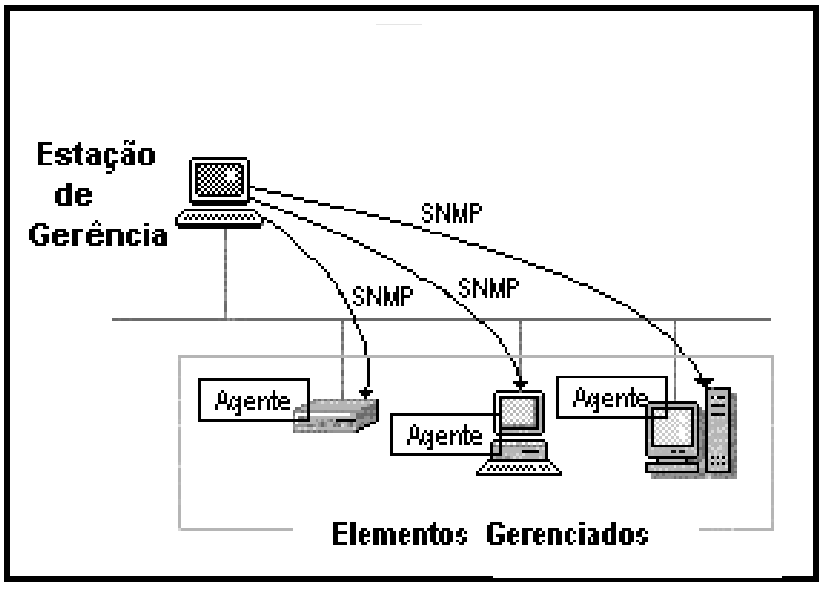


FIGURA 3.5 – Arquitetura Tradicional de Gerência SNMP.

O SNMP usa o UDP (*User Datagram Protocol*) como protocolo de transporte para passagem de dados entre gerentes e agentes, utilizando a porta de comunicação 161 para enviar solicitações e receber as respostas, e a porta de comunicação 162 para receber *traps* de dispositivos gerenciados. A Fig. 3.6 ilustra os protocolos do modelo TCP/IP que o SNMP utiliza.

Com os agentes e gerentes definidos, é preciso definir como realizar o monitoramento, ou melhor, a forma de comunicação. O SNMP apresenta duas formas: *polling* e registro de eventos (*trap*). O *polling* é uma interação freqüente entre o gerente e o agente, em que o gerente verifica se o agente está ativo e pode consultar informações de sua MIB. No registro de eventos, a iniciativa parte do agente. O agente é pré-configurado para enviar informações periódicas sobre o seu estado de funcionamento para o gerente.

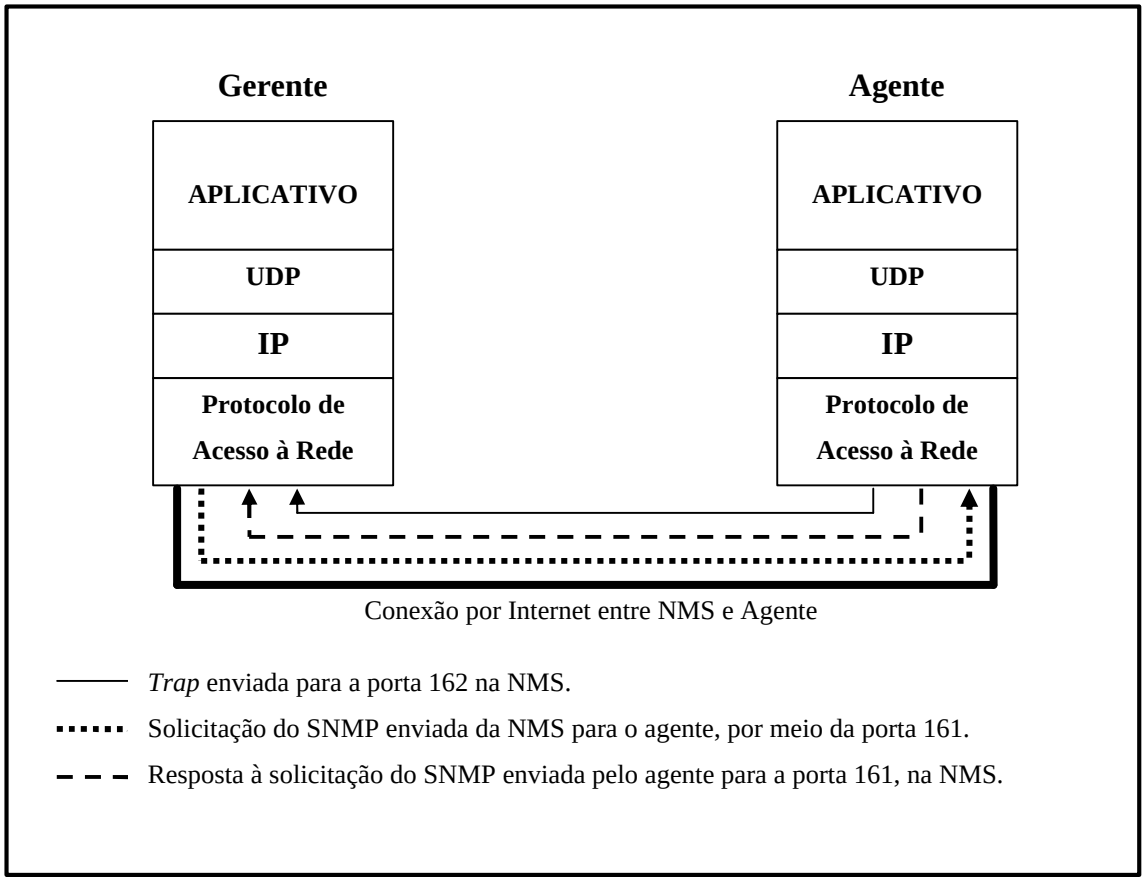


FIGURA 3.6 – Protocolos do Modelo TCP/IP e o SNMP

3.3. GERENCIAMENTO BASEADO EM WEB

A popularidade da WWW oferece uma nova forma de acesso às aplicações de software complexas. Quase toda plataforma suporta *Web Browsers*, e muitas pessoas estão usando a *Web* como uma ferramenta simples para acessar diferentes serviços. Assim, a *Web* está se tornando um recipiente para fins gerais, servindo de interface e ambiente de desenvolvimento para todo tipo de informação distribuída em rede ou armazenada num único banco de dados amplo.

A utilização das tecnologias da WWW, no gerenciamento de rede, tem contribuído para integrar ferramentas poderosas à *Web*, as quais permitem gerenciar inúmeros recursos de rede. As atividades de gerenciamento de rede podem ser apoiadas com a combinação da tecnologia SNMP ao poder da *Web*, evitando, muitas vezes, a necessidade de aquisição de ferramentas de custo elevado.

3.3.1. INTEGRAÇÃO DA WEB COM O GERENCIAMENTO DE REDE

A idéia de integrar o gerenciamento de redes à *Web*, visa diminuir a frustração do usuário com ferramentas convencionais de gerenciamento de rede, substituindo-as, em muitos casos, por ferramentas simples e suportadas por quase todas as plataformas. Atualmente, os usuários demandam interfaces mais simples para tarefas fáceis e complexas.

A escolha da *Web* pode ser justificada em função de:

- Ser uma interface bastante conhecida pelo usuário;
- Estar disponível virtualmente para cada plataforma;
- Ser de propósito geral, capaz de acomodar os serviços de hoje e estar aberta a extensões futuras;
- Ter uma capacidade de hipertexto integrada, ideal para navegar diferentes tipos de informações;

- Ser um programa independente, que pode ser rodado em qualquer plataforma.

3.3.2. ARQUITETURAS USADAS NA GERÊNCIA BASEADA EM *WEB*

Em [AND98] foi realizado um estudo apresentando quatro arquiteturas representantes da Gerência baseada em *Web*, são elas: Arquitetura em Duas Camadas, Arquitetura em Três Camadas, Iniciativa WBEM (*Web-Based Enterprise Management*) e Iniciativa JMAPI (*Java Management Application Programming Interface*).

3.3.2.1. Arquitetura em Duas Camadas

A Arquitetura em Duas Camadas é a mais simples das alternativas para a gerência baseada em *Web*. Nesta arquitetura um servidor HTTP (*HyperText Transfer Protocol*) é embutido nos dispositivos de rede, como roteadores, *switches* ou pontes por exemplo, possibilitando desta maneira que através de páginas HTML, este dispositivo seja monitorado ou controlado, como está ilustrado na Fig. 3.7.

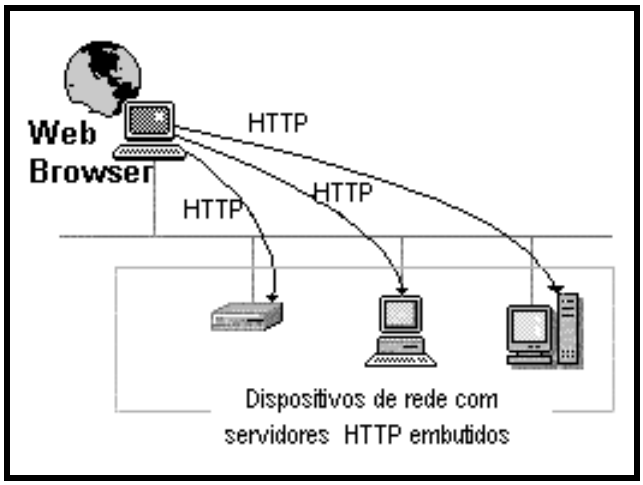


FIGURA 3.7 – Arquitetura em Duas Camadas

Assim cada dispositivo tem sua própria URL (*Uniform Resource Locator*) principal e pode ser gerenciado através de um *Web browser*. As páginas HTML de um determinado dispositivo podem incluir formulários processados por programas CGI (*Common Gateway Interface*) ou outra técnica que possibilite a gerenciamento destes dispositivos via HTTP.

A principal característica desta arquitetura é que ela é uma substituição ou pelo menos uma nova alternativa ao *Telnet* nas atividades de configuração. Ao invés de abrir uma sessão *Telnet* com o dispositivo e utilizar uma interface a caractere para realizar as alterações nos parâmetros de configuração, o usuário acessa via *browser* as páginas HTML do dispositivo e, através do preenchimento de formulários, realiza a sua configuração. Não há dúvidas de que utilizar páginas HTML para configuração de dispositivos é muito mais amigável e simples do que utilizar uma interface a caractere como o *Telnet*. Pode-se observar que a arquitetura em duas camadas não impede que se use outras soluções de gerenciamento tais como o SNMP. O dispositivo é simplesmente equipado com um *dual stack*, isto é, vem com suporte ao SNMP e ao HTTP. Portanto soluções baseadas no SNMP podem ser usadas sem nenhuma mudança.

3.3.2.2. Arquitetura em Três Camadas

Como na arquitetura em duas camadas, a arquitetura em três camadas também utiliza o SNMP e o HTTP simultaneamente. Porém estes dois protocolos são usados de forma diferente como ilustrado na Fig. 3.8.

A estação central de gerenciamento, a NMS (*Network Management Station*), comunica-se com os agentes utilizando o SNMP da mesma maneira como é feito na abordagem tradicional de gerência. No entanto nesta NMS existe um servidor HTTP embutido, o que permite que um *browser* possa ser utilizado para acessar a plataforma ou aplicação que está presente na NMS. Pode-se imaginar a NMS agindo como um servidor *proxy* que se comunica com os agentes via SNMP e com os usuários utilizando o HTTP [SAU99]. Pode-se observar que, enquanto a solução em duas camadas é implementada pelos fabricantes de dispositivos, a arquitetura em três camadas

representa uma evolução alcançada pelos desenvolvedores de plataformas e aplicações de gerência.

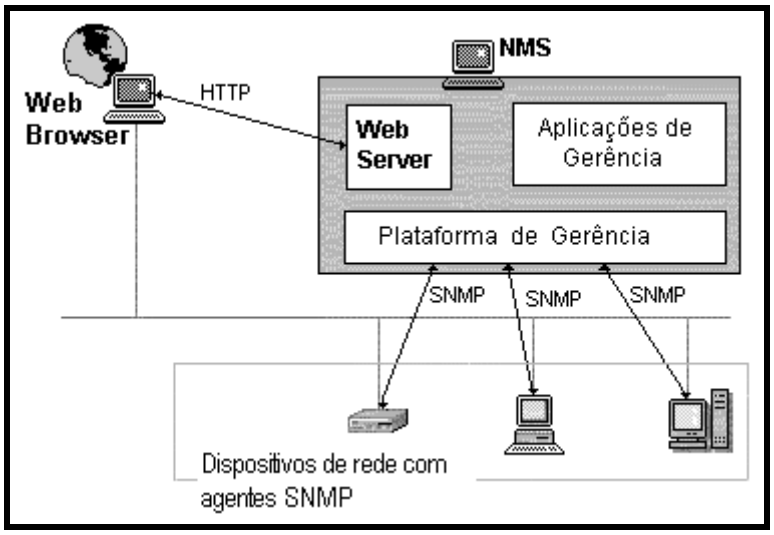


FIGURA 3.8 – Arquitetura em Três Camadas

3.3.2.3. Arquitetura WBEM

WBEM é um acrônimo para *Web-Based Enterprise Management* que, como o nome em inglês ressalta, ambiciona ser uma gerência baseada em *Web* não apenas das redes, mas, de toda a estrutura de informática da empresa. Esta iniciativa está sendo anunciada como a grande solução para os problemas atuais da gerência.

O Gerenciamento da empresa significa muito mais do que somente gerenciar as redes de computadores presentes numa empresa; significa gerenciar toda a estrutura de informática presente, incluindo redes, serviços, *hosts*, aplicações, etc. O principal foco desta iniciativa é permitir isto através da consolidação dos dados vindos de diversas fontes, incluindo estruturas de gerenciamento já existentes como o SNMP ou a OSI. Em outras palavras, para permitir a integração das aplicações de gerência, é uma boa estratégia começar por integrar todos os dados necessários a estas aplicações. Um modelo de dados único deve ser usado e fundamentalmente tal modelo deve ser extensível.

A arquitetura WBEM é mostrada na Fig. 3.9. Os seus principais componentes são o modelo de dados CIM (*Common Information Model*), um novo protocolo de gerenciamento chamado de HMMP (*HyperMedia Management Protocol*) e um módulo gerente chamado de HMOM (*HyperMedia Object Manager*).

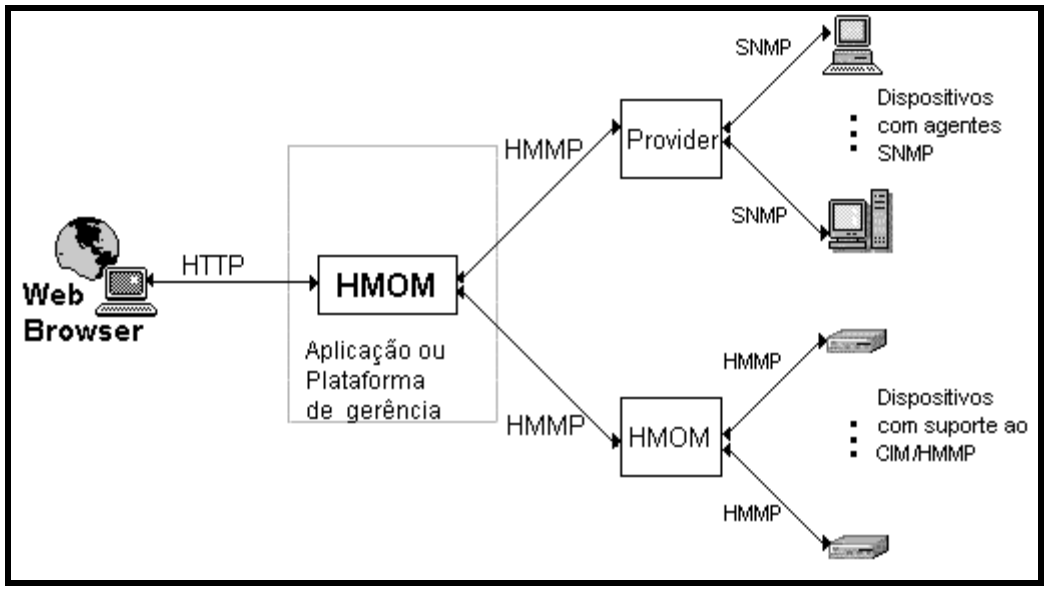


FIGURA 3.9 – Arquitetura WBEM

O CIM é um modelo de dados orientado a objetos usado para descrever o ambiente gerenciado. Ser orientado a objetos já torna este modelo extensível, pela característica da herança presente neste paradigma, e poderoso por poder associar métodos, como código de gerência por exemplo, a estes objetos. A interface do usuário no WBEM poderá ser um *browser* e novas *tags* HTML poderão ser usados para embutir acesso aos objetos CIM através do HTML.

Uma segunda parte desta arquitetura é o protocolo HMMP. Este protocolo é muito mais poderoso que o SNMP por permitir a manipulação remota dos objetos CIM. Este protocolo permite que classes e objetos sejam criados, acessados e removidos; permite executar chamadas a métodos; etc. O HMMP também suporta um esquema de segurança poderoso com suporte a diversos níveis de segurança.

Quando um HMOM é utilizado como um *proxy* consolidando dados vindos de diversas fontes não-HMMP, ele é chamado de um *provider*. Atuando neste papel, ele é capaz de, por exemplo, permitir a integração de dispositivos SNMP a esta nova

arquitetura. Pode-se observar que enquanto um HMOM pode atuar tanto como cliente ou como servidor utilizando o protocolo HMMP, um *provider* pode atuar apenas como um servidor. Um HMOM também é o responsável por tornar a informação de gerência disponível a um *browser* através do HTTP.

3.3.2.4. Arquitetura JMAPI

A sigla JMAPI significa *Java Management Application Programming Interface,* que significa API para gerência escrita em Java. JMAPI é basicamente um conjunto de classes propostas pela *Sun Microsystems* para construir aplicações de gerência.

Como no caso da WBEM, JMAPI também possibita a gerência não apenas de redes de computadores mas também de sistemas e aplicações de informática presentes numa empresa. Ela também prevê a utilização de um modelo de dados extensível orientado a objetos.

No entanto, a solução JMAPI tenta ir um pouco adiante da WBEM no que diz respeito a melhorar a integração entre as aplicações. Esta arquitetura utiliza como interface um *browser* e tenta proporcionar recursos para uma melhor integração entre as aplicações sugerindo uma GUI (*Graphical User Interface*) e oferecendo também APIs padrões para o desenvolvimento de aplicações. A Fig. 3.10 ilustra a arquitetura JMAPI.

A arquitetura JMAPI é bastante similar à arquitetura WBEM. A principal diferença é que, enquanto o WBEM utiliza um novo protocolo, o HMMP, para manipular remotamente os objetos, JMAPI utiliza o mecanismo padrão JAVA para a comunicação: RMI (*Remote Method Invocation*). Comparando as arquiteturas JMAPI e WBEM, percebe-se que o ARM (*Admin Runtime Module*) presente na arquitetura JMAPI corresponde ao HMOM da arquitetura WBEM e que o mesmo acontece com o *Proxy Agent Object Factory (JMAPI)* que corresponde ao *Provider* (WBEM).

Na terminologia JMAPI o que quer que esteja no lado remoto de uma RMI é chamado "*appliance*", seja ele um dispositivo de rede ou um *proxy* rodando numa estação de gerência ou ainda outra máquina qualquer.

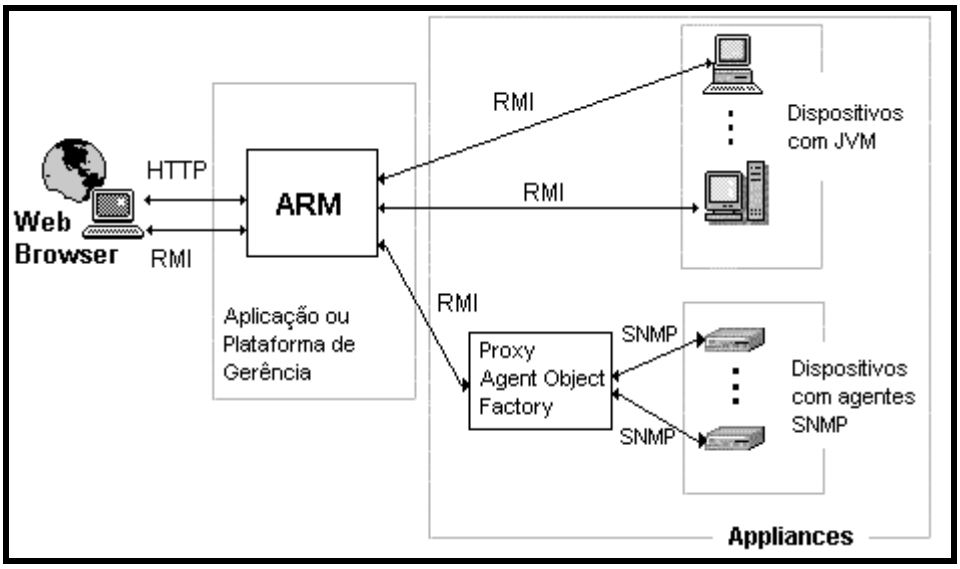


FIGURA 3.10 – Arquitetura JMAPI

Tanto a arquitetura JMAPI quanto a arquitetura WBEM têm a necessidade de oferecer mecanismos, *Proxy Agent Object Factory* na JMAPI e o *Provider* na WBEM, para atender aos dispositivos que só falam SNMP. Este é, talvez, o primeiro requisito de qualquer nova arquitetura de Gerência; pois além da grande base de dispositivos SNMP já presentes na rede, o SNMP continua sendo o protocolo padrão de gerência com maior alcance e importância no mercado e qualquer mudança neste cenário demanda tempo e requer esforços de compatibilização.

3.3.3. UTILIZAÇÃO DA ARQUITETURA EM TRÊS CAMADAS

Para o presente trabalho, foi definida a utilização da arquitetura em três camadas como base para o desenvolvimento do modelo de integração do WAP com a gerência de redes TCP/IP, pois a arquitetura em três camadas é a que oferece maior flexibilidade na escolha das ferramentas de *software* para o desenvolvimento de um modelo otimizado para a integração entre o WAP e o SNMP.

Em [SAU99], foi discutida a implementação de uma aplicação de gerenciamento de rede baseada na *Web*, usando a arquitetura em três camadas. Essa aplicação foi chamada de *WebManager.Version 1*. A Fig. 3.11 ilustra a arquitetura dessa aplicação.

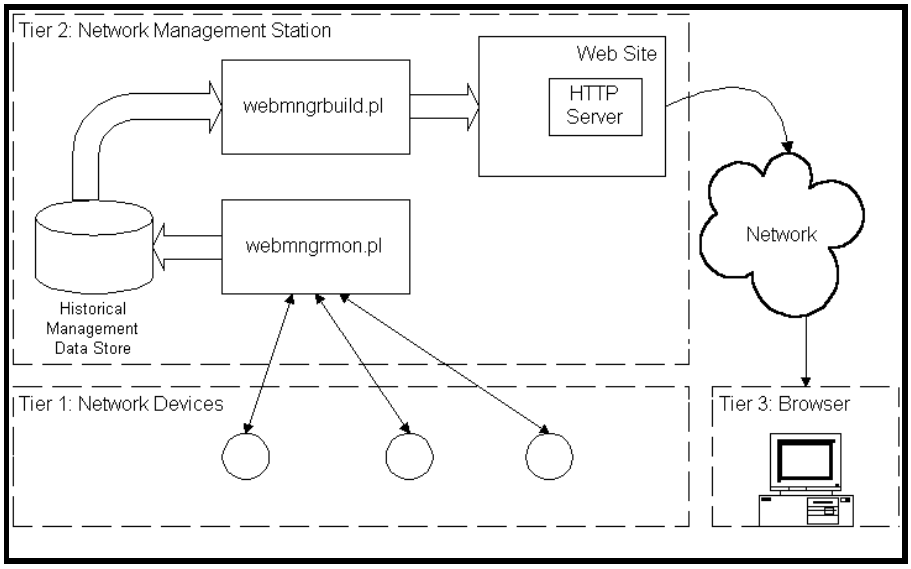


FIGURA 3.11 – Arquitetura *WebManager*

A primeira camada consiste de dispositivos de rede, contendo agentes SNMP, tal como roteadores, *hubs* ou *switches*. A segunda camada é a Estação de Gerenciamento de Rede ou NMS, que obtém dados da primeira camada e os armazena numa base de dados. Em momentos apropriados, estes dados são convertidos para o formato HTML o qual permite o browser na terceira camada acessar as informações de gerenciamento.

O programa chamado webmngrmon.pl (*WebManager Monitor*) executa-se permanentemente em segundo plano e periodicamente comunica-se com os dispositivos de rede usando o SNMP e armazena os valores das variáveis da MIB na base de dados. Na verdade, o programa não se comunica diretamente com os dispositivos, mas utiliza o módulo CMU SNMP, que é um conjunto de ferramentas utilizadas para acessar dispositivos que contém agentes SNMP. A Fig. 3.12 ilustra como é realizado o monitoramento dos objetos gerenciados, no qual o programa webmngrmon.pl recebe informações do módulo CMU SNMP.

O programa chamado webmngrbuild.pl (*WebManager Builder*), por outro lado é executado uma vez por hora (pela facilidade *cron* do UNIX) e converte o banco de dados em páginas de HTML apropriadas.

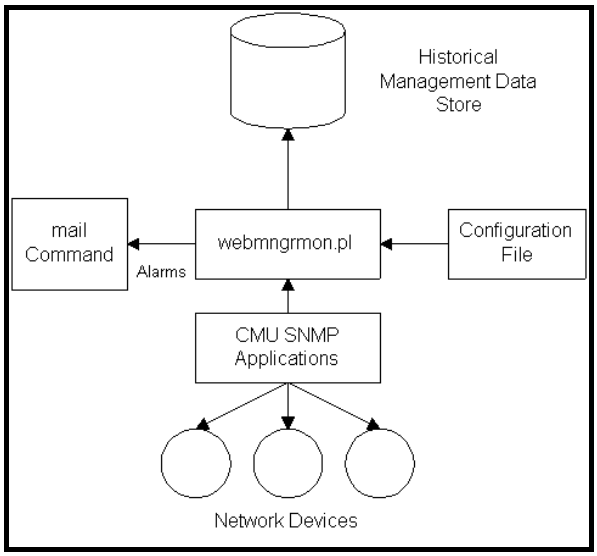


FIGURA 3.12 – Monitoração de Objetos Gerenciados

3.4. CONCLUSÃO

Dentre as quatro arquiteturas usadas na gerência baseada em *Web*, optou-se pela arquitetura em três camadas, por possuir uma maior flexibilidade para adaptar-se à arquitetura WAP. Atualmente, o protocolo SNMP é amplamente utilizado no gerenciamento de redes. Problemas como a segurança e a confiabilidade estão sendo resolvidas com o desenvolvimento do SNMPv2 e SNMPv3. Muitos fabricantes já utilizam a MIB-II e também implementam MIBs proprietárias para seus equipamentos e programas. Devido à grande utilização e ao esforço de órgãos internacionais, como a IETF, na padronização e no desenvolvimento do SNMP, este trabalho utilizou o protocolo como mecanismo de leitura dos objetos gerenciados. Utilizou-se a versão um do protocolo (SNMPv1) por motivos de interoperabilidade entre o gerente e os agentes do ambiente de teste do modelo proposto no Capítulo 5.

4. WAP

Neste capítulo serão apresentados o modelo da arquitetura WAP (*Wireless Application Protocol*), comparações com o modelo TCP/IP, seus componentes e suas características.

4.1. O QUE É WAP?

WAP é a sigla de *Wireless Application Protocol*, que é um conjunto de especificações para padronizar as interações entre equipamentos portáteis e a Internet.

Segundo [RIS01], WAP é um padrão emergente para marcação e apresentação de conteúdo sem fio dirigido explicitamente a dispositivos de mão pequenos e móveis, como telefones celulares, *pagers* e terminais de acesso sem fio.

Conforme [DIAS00], o padrão WAP foi iniciado em 1997, e é mantido pelo Fórum WAP, sendo este um consórcio multinacional de provedores de infra-estrutura sem fio que inclui empresas como Nokia, Phone.com, Ericsson, Motorola, entre outras.

Segundo [DEN00], os objetivos do WAP fórum são:

- Trazer conteúdo da Internet e serviços de dados avançados a telefones móveis e outros terminais sem fio;
- Criar uma especificação de protocolo sem fio global que funcione para todas as tecnologias de rede sem fio;
- Habilitar a criação de conteúdo e aplicações que se ajustam a uma ampla gama de redes sem fio e tipos de dispositivos;
- Abranger e estender padrões existentes de tecnologia, onde quer que seja possível e apropriado.

O Fórum WAP não desenvolve produtos, mas cria padrões livres de licença para a indústria usar no desenvolvimento de produtos. A linha de produtos de cada companhia pode ter suas próprias características, mas deve estar em conformidade com as especificações WAP.

Também é muito importante para os padrões do Fórum WAP serem desenvolvidos do mesmo modo que padrões existentes. Por exemplo, a especificação WAP não deve especificar como devem ser transmitidos os dados sobre a interface aérea.

Quando o Fórum WAP identifica uma área nova da tecnologia onde um padrão não existe, ou existe, mas precisa de alguma adaptação, trabalha para submeter suas especificações para outros grupos de padrões da indústria.

O Fórum WAP estabeleceu uma relação de ligação formal com o W3C (*World Wide Web Consortium*) e a TIA (*Telecommunications Industry Association*). O Fórum WAP está colaborando com estas organizações na área de tecnologias Web dentro do setor sem fio. O W3C, a TIA e o Fórum WAP pretendem continuar trabalhando juntos nas áreas técnicas selecionadas para criarem e promoverem especificações técnicas de interesse para todas as três organizações.

Segundo [DEN00] a especificação WAP é estendida e compatível com as tecnologias existentes, como padrões de redes de dados digitais, tecnologias da Internet, como IP, HTTP, XML, SSL, URL, *scripting* e outros formatos de conteúdo, como PHP, Perl e ASP.

O desenvolvimento de uma especificação própria para a Internet sem fio dá-se pelas limitações de dispositivos de acesso e da própria rede sem fio. Para [DIAS00], os aparelhos celulares tendem a ser cada vez menores. Assim, foram considerados pontos importantes para escrever o padrão WAP:

- Janelas de apresentação das informações de dimensões e resoluções reduzidas. Geralmente possuem de oito a doze caracteres por linha, com uma limitação de número de linhas variando de dispositivo para dispositivo;
- Rede com pouca largura de banda. As conexões ocorrem com velocidades entre 300 bps e 10 Kbps (2G – Segunda Geração da Telefonia Celaular);

- Memória e capacidade de processamento reduzido;
- Dispositivos de entrada de capacidade limitada. Geralmente é o próprio teclado numérico do telefone celular;
- Número pequeno de cores. Geralmente são de cristal líquido de uma única cor;
- Alta latência, ou seja, excessiva demora entre a solicitação e a resposta ou entre o intercâmbio de pacotes individuais;
- Pouca estabilidade de conexão.

4.2. MODELO WEB VERSUS MODELO WAP

Será apresentado um comparativo entre algumas características do modelo WAP e do modelo Web.

4.2.1. O MODELO WEB

Na Internet são usados protocolos de comunicação padrão como HTTP, TCP e IP, como ilustra a Fig. 4.1. O conteúdo das informações pode ser estático ou dinâmico. Conteúdo estático é produzido uma vez e não é mudado ou atualizado muito freqüentemente, por exemplo, uma página de apresentação de uma empresa. Conteúdo dinâmico é aquele que freqüentemente precisa ser alterado, como por exemplo, as páginas de um *site* de notícias, citações acionárias ou informação de conta bancária. Para isso são utilizadas tecnologias como ASP (*Active Server Pages*), CGI (*Common Gateway Interface*), PHP (*PHP: Hypertext Preprocessor*), *servlets* e *aplets* Java.

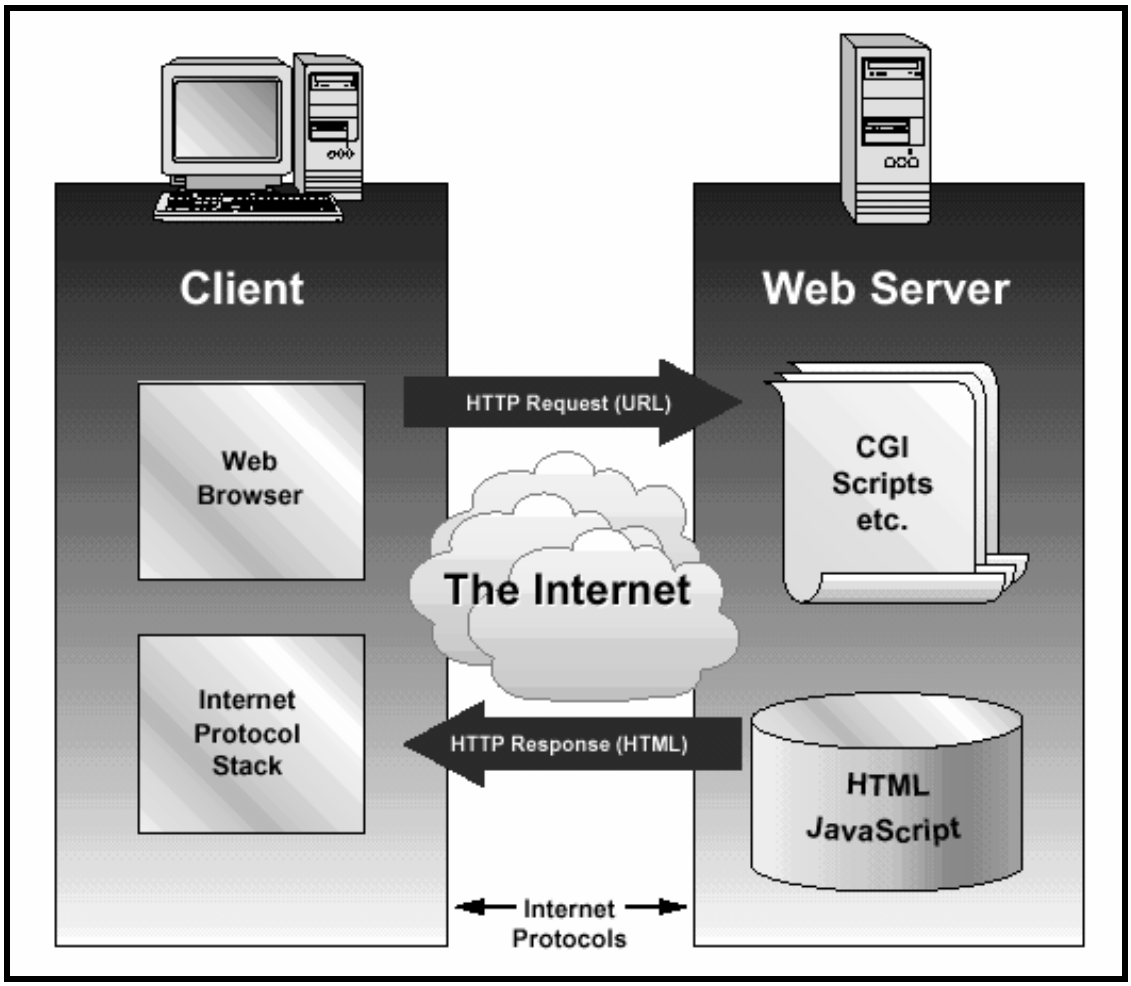


FIGURA 4.1 – Modelo da Internet

4.2.2. O MODELO WAP

WAP também faz uso do paradigma da Internet para proporcionar um serviço independente de plataforma. O WAP está sendo desenvolvido para adaptar o formato das informações providas pela Internet, para um formato compatível com os aparelhos portáteis. Isto é devido às redes de dados sem fio apresentarem um ambiente de comunicação mais limitado, comparado com as redes comuns. Por causa de limitações fundamentais de potência, espectro disponível e mobilidade, redes de dados sem fio tendem a ter:

- Menor largura de banda;
- Maior latência;
- Menor estabilidade de conexão;
- Disponibilidade menos previsível.

Serviços criados usando HTML não ajustariam muito bem em pequenos dispositivos como *handhelds, palm PCs*, PDAs (*Personal Digital Assistants*) e telefones celulares. A largura de banda da comunicação usada em dispositivos móveis sem fio não suportaria a quantidade de dados de uma página em HTML. Então foi desenvolvida uma linguagem adaptada a redes sem fio - o WML (*Wireless Mark up Language*), sendo derivada do XML (*Extensible Markup Language*).

O WML oferece um modelo de navegação projetado para dispositivos com pequenas telas ou visores sem o uso de dispositivos apontadores como o mouse ou o teclado. Para economizar a “valiosa” largura de banda numa rede sem fio, o WML pode ser codificado em um formato binário compacto, pois é importante não construir uma "página" grande, pois em sistemas de telefonia celular, como TDMA e CDMA, as velocidades de transferência são de 9,6 Kbps e 14,4 Kbps, respectivamente.

Codificar o WML (ou arquivo .wml) é um das tarefas executadas pelo *WAP Gateway/Proxy*, que é o sistema intermediário que interliga a rede sem fio com a Internet, como mostra a Fig. 4.2. e Fig. 4.3.

Observa-se que a requisição que é enviada do cliente para o *Gateway* usa o WSP (*Wireless Session Protocol*) ou Protocolo de Sessão Sem Fio. Em sua essência, WSP é uma versão binária de HTTP.

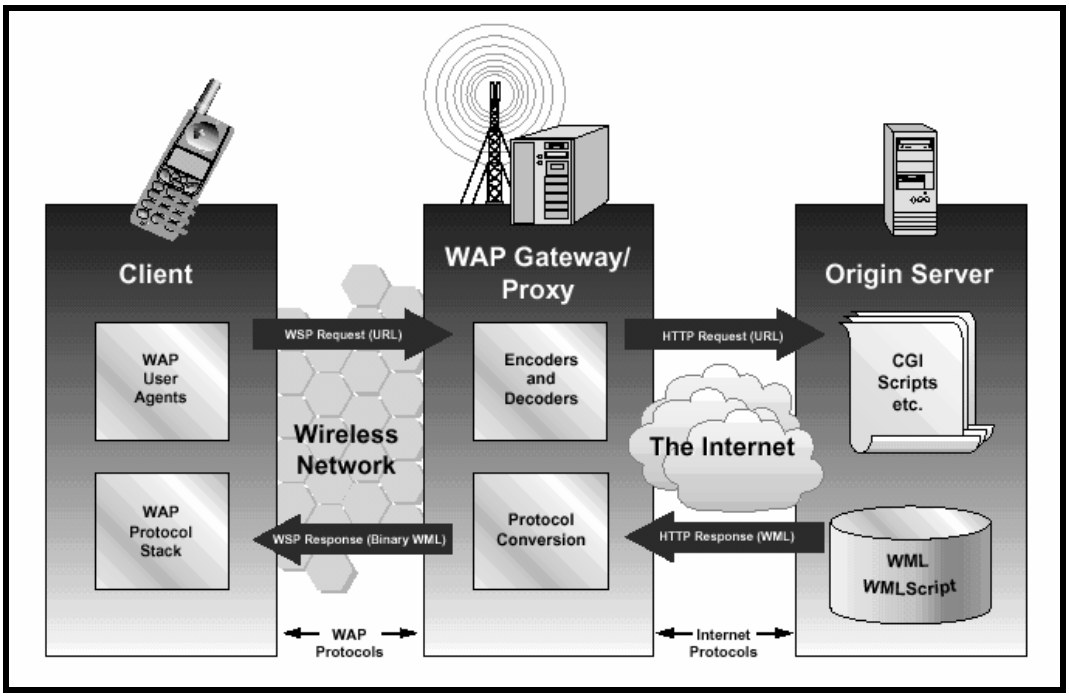


FIGURA 4.2 – O Modelo do WAP.

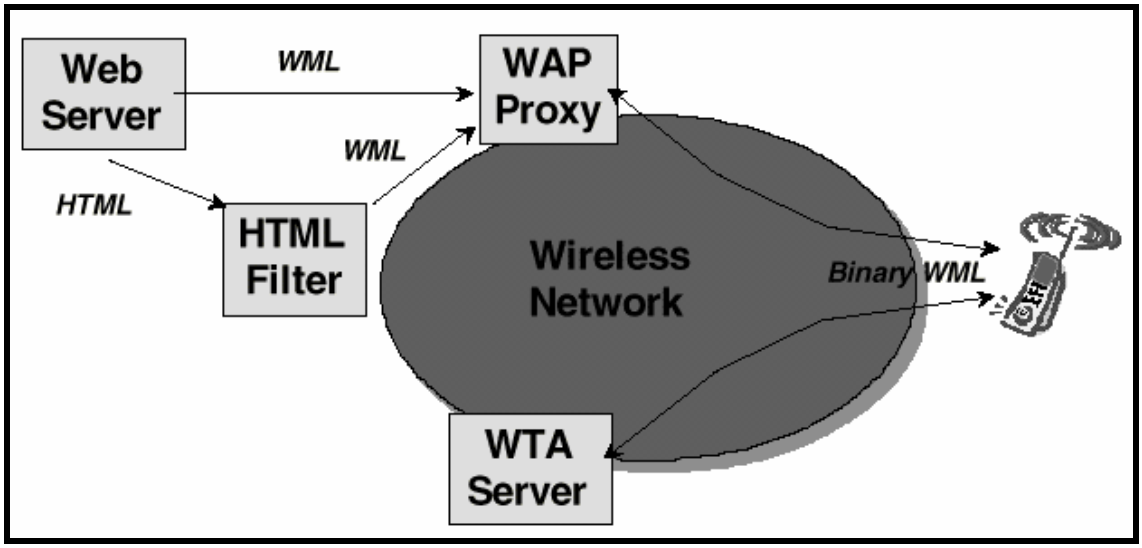


FIGURA 4.3 – Exemplo de uma Rede WAP.

4.3. ARQUITETURA WAP

A arquitetura de WAP provê um ambiente extensível e escalonável para o desenvolvimento de aplicações para dispositivos móveis de comunicação [WAP98]. Isto é alcançado através de um projeto de uma pilha de protocolos, como ilustra a Fig. 4.4.

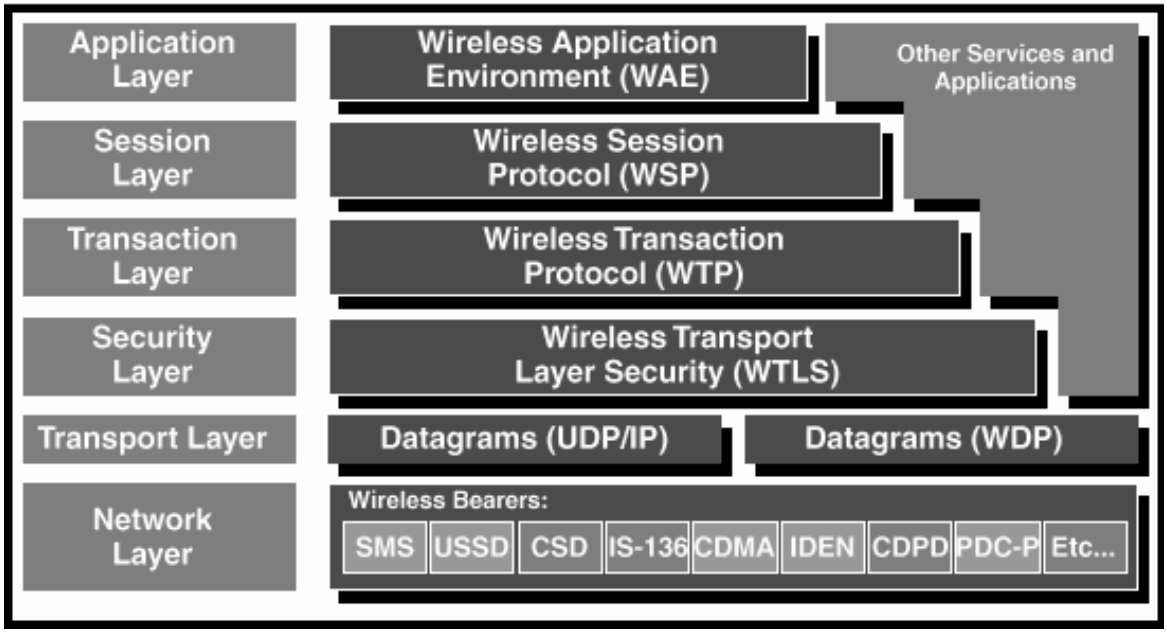


FIGURA 4.4 – Arquitetura Lógica do Modelo WAP

Fazendo uma relação entre a pilha de protocolos e a divisão em camadas, tem-se:

Camada de Aplicação Wireless Application Environment (WAE)

Camada de Sessão Wireless Session Protocol (WSP)

Camada de Transação Wireless Transaction Protocol (WTP)

Camada de Segurança Wireless Transport Layer Security (WTLS)

Camada de Transporte Wireless Datagram Protocol (WDP)

Cada uma das camadas da arquitetura é acessível pelas camadas superiores, como também por outros serviços e aplicações.

A arquitetura WAP estendida em camadas permite que outros serviços e aplicações utilizem as características da pilha WAP através de um conjunto de interfaces bem definidas. Aplicações externas podem acessar diretamente as camadas de sessão (WSP), transação (WTP), segurança (WTLS) e de transporte (WDP). Isso permite que a pilha WAP seja usada por serviços e aplicações que não estejam atualmente especificados pelo WAP, mas valiosos para o mercado de redes sem fio. Por exemplo, aplicações, como correio eletrônico, calendário, catálogo telefônico, bloco de anotações, e comércio eletrônico, ou serviços como páginas amarelas, podem ser desenvolvidas usando a pilha de protocolos WAP.

É esperado que a tecnologia WAP seja útil para aplicações e serviços que não estejam especificados pelo WAP Forum. A Figura 4.5 descreve várias possibilidades de empilhamento dos protocolos da tecnologia WAP. Essas possibilidades são propostas ilustrativas e não constituem uma instrução de adaptação ou interoperabilidade.

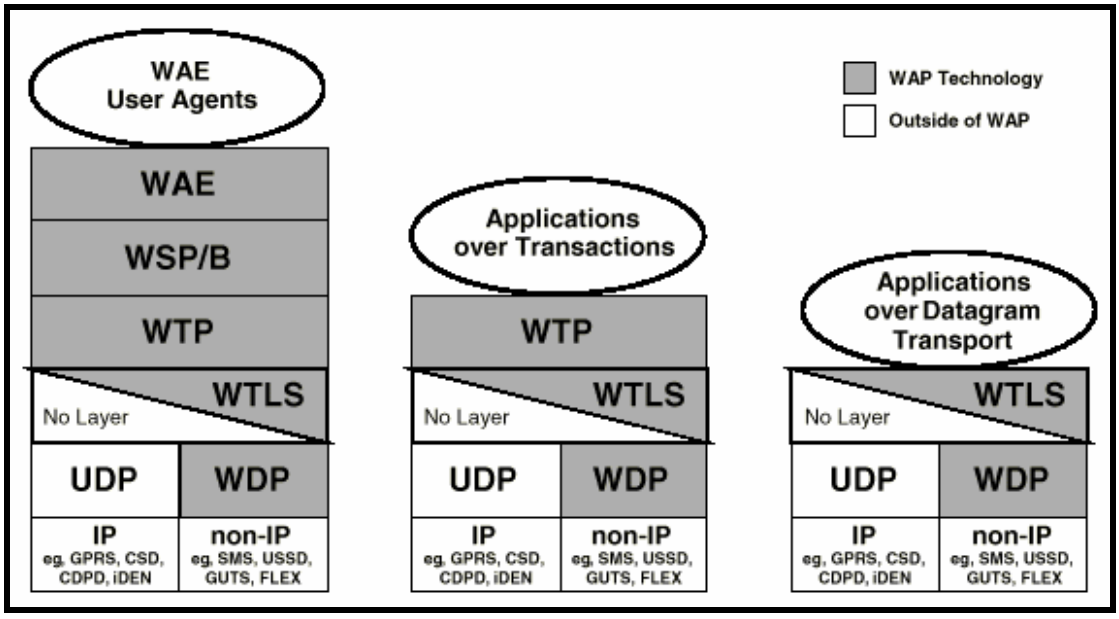


FIGURA 4.5 – Possibilidades de Empilhamento dos Protocolos do WAP.

A pilha que se encontra mais a esquerda representa um exemplo típico de uma aplicação WAP, isto é, representa a configuração normal. A pilha do meio é intencional para aplicações que requerem serviços de transação com ou sem segurança. A pilha que

se encontra mais a direita é intencional para aplicações e serviços que só requerem transporte de pacote de dados (*datagram*) com ou sem segurança.

A Fig. 4.6 ilustra o Modelo de Referência da Arquitetura WAP [WAP98], onde temos

:***Management Entity***: é a entidade de gerenciamento de cada camada, que é responsável pela inicialização e pela configuração do protocolo, e também pelo tratamento de erros.

SAP – Service Access Point: é o ponto de acesso dos serviços oferecidos pela camada, para a camada superior.

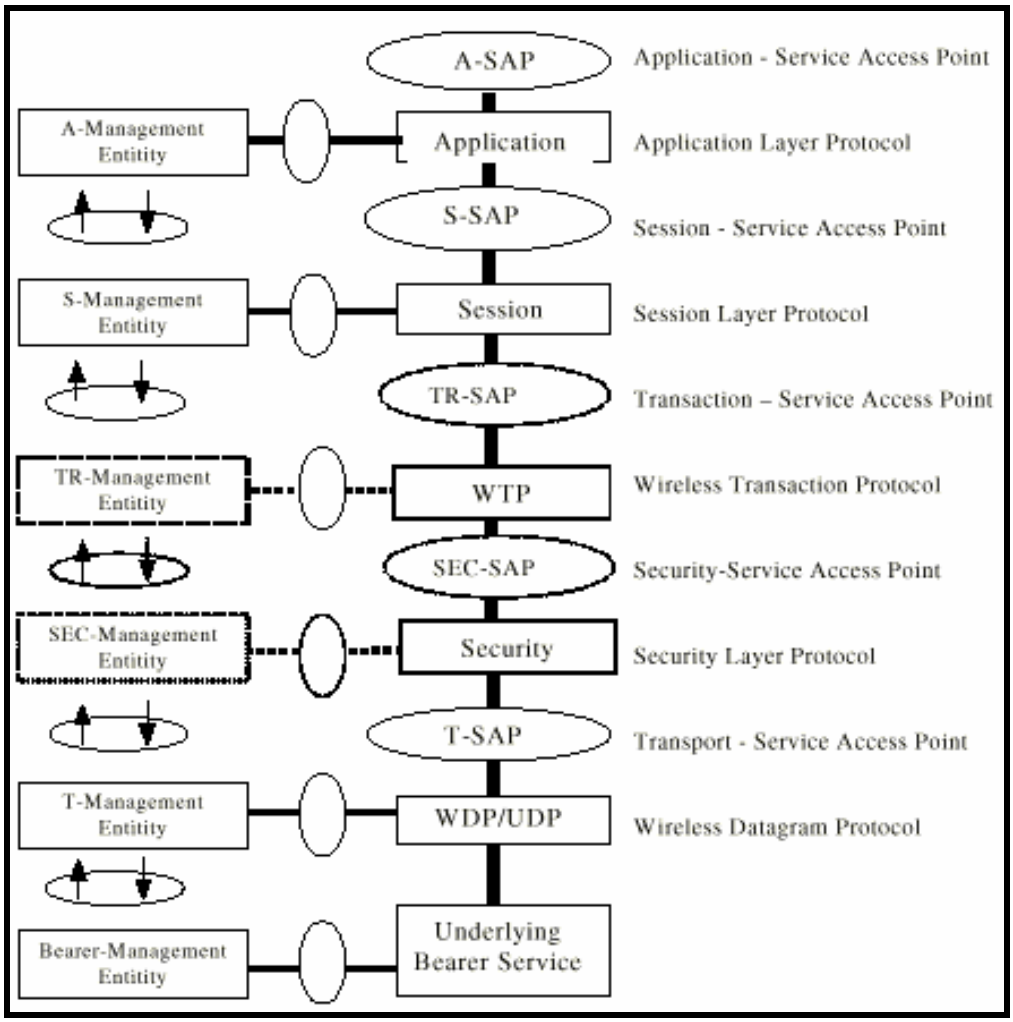


FIGURA 4.6 – Modelo de Referência da Arquitetura WAP

4.3.1. PORTADORES DE SINAIS: *BEARERS*

Os protocolos WAP são projetados para operar sobre uma variedade de diferentes serviços portadoras de sinais:

- Mensagens curtas (SMS – Short Message Service),
- Comutação de Dados por Circuito (CSD - Circuit-switched Data),
- Serviço de Pacotes Via Rádio (GRPS – General Packet Radio Service),
- Troca de Dados em Circuito de Alta Velocidade (HSCSD – High Speed Switch Data), etc.

As portadoras oferecem diferentes níveis de qualidade de serviço (QoS – *Quality of Service*) com respeito à vazão (*throughput*), taxa de erro, e atrasos (*delay*). Os protocolos WAP são projetados para compensar ou tolerar estas variações do nível de serviço.

Considerando que a camada WDP provê a convergência entre o serviço da portadora e o resto da pilha WAP, a especificações do WDP listam as portadoras que são suportadas e as técnicas usadas para permitir que os protocolos WAP “rodem” sobre cada portadora. A lista de portadoras suportadas mudará com o passar do tempo, e portadoras novas serão adicionados à medida que o mercado de redes sem fio evolui [WAP98].

4.3.2. CAMADA DE TRANSPORTE: *WIRELESS DATAGRAM PROTOCOL* (WDP)

O protocolo da camada de transporte na arquitetura WAP é referido como Protocolo de Pacote de Dados Sem Fio (WDP). A camada WDP opera sobre vários tipos de portadoras de sinais. Como um serviço de transporte geral, WDP oferece um serviço consistente aos protocolos de camadas superiores do WAP e comunica-se de forma transparente sobre a portadora disponível [WDP99].

O WDP oferece um serviço consistente através do Ponto de Acesso de Serviço de Transporte T-SAP (*Transport-Service Access Point*) para o protocolo de camada superior do WAP. Esta consistência de serviço permite que aplicações operem de forma transparente sobre diferentes portadoras de sinais. Os variados níveis de cada portadora de sinal, mostrados na Figura 4.7, ilustram a diferença em relação às funções providas pelas portadoras. O WDP pode ser mapeado sobre diferentes portadoras, com diferentes características.

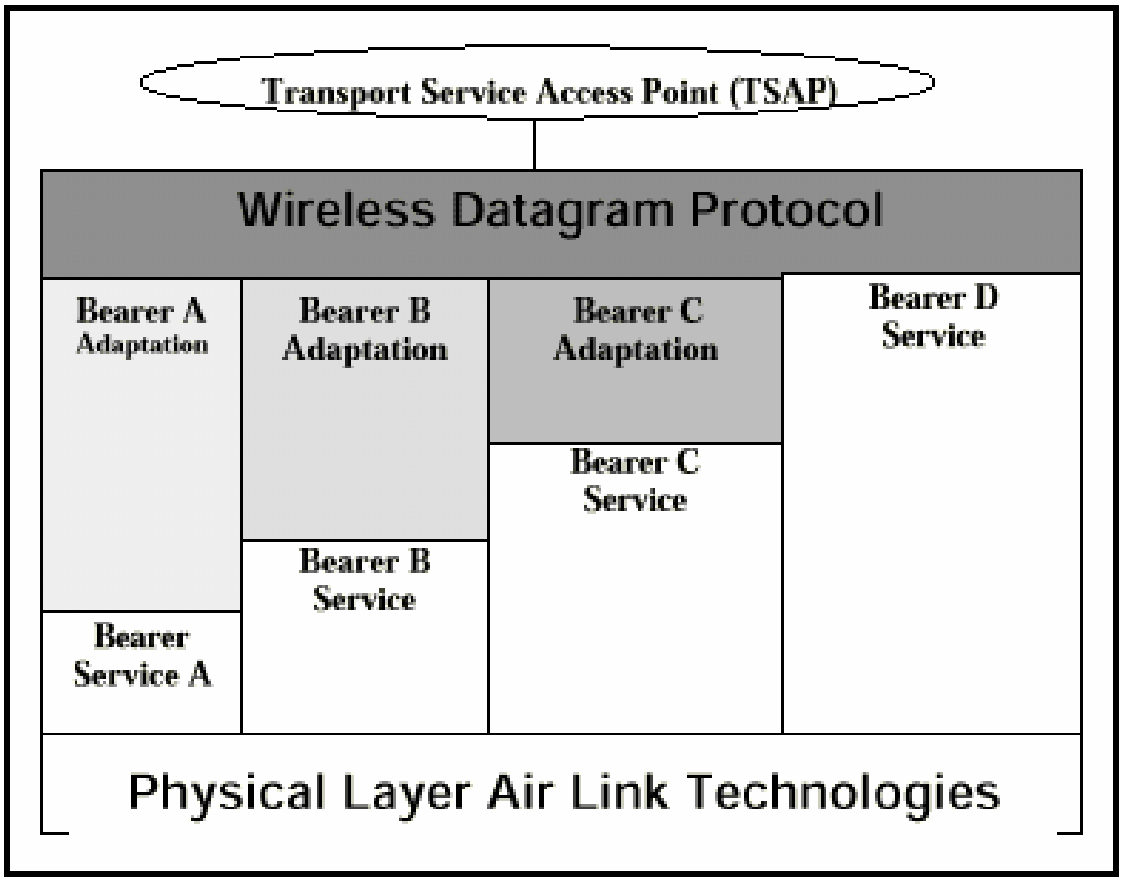


FIGURA 4.7 – Adaptação dos Portadores de Sinais no WDP

Para aperfeiçoar o protocolo com respeito ao uso de memória e eficiência de transmissão de rádio, pode variar muito o desempenho do WDP em cima de cada tipo de portadora. De qualquer forma, o serviço do WDP e as primitivas de serviço permanecerão as mesmas, provendo uma interface consistente para as camadas mais altas.

O WDP suporta várias instâncias de comunicação simultâneas de uma camada mais alta sobre uma única portadora subjacente ao WDP. O número de porta identifica a entidade da camada mais alta sobre o WDP. Esta pode ser outra camada de protocolo assim como o Protocolo de Transação Sem Fio (WTP) ou o Protocolo de Sessão Sem Fio (WSP) ou uma aplicação como correio eletrônico.

Através do reuso dos elementos das portadoras subjacentes, o WDP pode ser implementado para suportar múltiplas portadoras e mesmo assim pode ser aperfeiçoado para uma operação eficiente dentro dos recursos limitados de um dispositivo móvel.

O protocolo WDP, por ser do tipo *datagram transport*, provê um serviço que não garante que os pacotes de dados não serão perdidos, duplicados, ou recebidos numa ordem diferente da do envio. Isso significa que o WDP oferece um serviço do tipo sem conexão.

Uma característica prevista na camada de transporte, é a utilização do protocolo UDP (*User Datagram Protocol*), se a rede suportar o protocolo IP. O Fórum WAP especifica explicitamente que quando a rede suportar o IP, deverá ser usado o UDP na camada de transporte.

Há outros dois serviços importantes que o WDP implementa:

- Segmentação e remontagem de pacotes de dados quando esses são muito grandes para a capacidade da rede subjacente.
- Mecanismo de correção de erros através do WCMP (Wireless Control Message Protocol).

4.3.3. CAMADA DE SEGURANÇA: *WIRELESS TRANSPORT LAYER SECURITY (WTLS)*

Como seu nome implica, o propósito de WTLS é proporcionar segurança para a camada de transporte entre um cliente WAP (dispositivo móvel) e o *WAP Gateway/Proxy*.

O protocolo WTLS é baseado no protocolo TLS (*Transport Layer Security*) 1.0, mas foi aperfeiçoada para canais de comunicação de faixa estreita.

A camada de segurança é modular e pode ser ou não utilizada, dependendo do nível de segurança requerido pelo aplicativo [WTLS99]. Os objetivos principais da camada WTLS são:

- Garantir a integridade dos dados pelo uso de Códigos de Autenticação de Mensagem (MAC);
- Proporcionar a privacidade dos dados através do uso de criptografia.
- Proporcionar a autenticidade dos dados com uso de certificados digitais.

4.3.4. CAMADA DE TRANSAÇÃO: *WIRELESS TRANSACTION PROTOCOL (WTP)*

Um protocolo de transação é definido para proporcionar os serviços necessários para aplicações interativas ou de navegação (requisição/resposta). Durante uma sessão de navegação, o cliente requisita informação de um servidor, que pode ser fixo ou móvel, e o servidor responde com a informação. O par requisição/resposta é chamado de "transação" neste documento. O objetivo do protocolo é proporcionar uma transação com confiabilidade [WTP99].

A transação é a unidade de interação entre o que inicia e o que responde. Uma transação começa com uma mensagem de pedido (*invoke*) gerada pelo emissor. O receptor deverá processar o pedido e responder ao emissor. Dentro do WTP foram definidas várias classes de transação. A mensagem de pedido identifica o tipo de transação requisitada o qual define a ação solicitada para completar a transação. As classes de transação são:

- Requisição não confiável sem mensagem de resultado: As mensagens perdidas não são reenviadas.
- Requisição confiável sem mensagem de resultado: O receptor manda um reconhecimento de recepção, evitando o reenvio da mensagem.
- Requisição confiável com uma mensagem de resultado: Após a recepção da mensagem, o receptor envia um resultado com reconhecimento de recepção implícito e o emissor confirma a recepção desse reconhecimento.

WTP é executado sobre um serviço de pacote de dados (WDP) e opcionalmente sobre um serviço de segurança (WTLS). WTP foi definido como protocolo direcionado para transações leves que é adequado para implementação em "clientes magros" ou *thin clients* (estações móveis) e opera eficientemente sobre redes sem fio. Os benefícios de usar WTP incluem: minimizar o numero de envios e reenvios, por exemplo concatenando as mensagens e retardando o envio de reconhecimento de recepção.

4.3.5. CAMADA DE SESSÃO: *WIRELESS SESSION PROTOCOL (WSP)*

A família de protocolos da Camada de Sessão na arquitetura WAP é chamada de Protocolo de Sessão Sem Fio (WSP). O WSP provê alguns recursos para troca organizada de conteúdo entre aplicações cliente/servidor cooperativas. Especificamente, provê os seguintes recursos para as aplicações [WSP99]:

- Estabelece uma sessão confiável do cliente para o servidor e libera essa sessão de forma organizada;
- Concorda em um nível comum de funcionalidade protocolar usando um mecanismo de negociação;
- Troca de conteúdo entre o cliente e servidor usando codificação compacta;
- Suspende e retoma uma sessão.

WSP provê à camada de aplicação de nível superior do WAP, uma interface consistente contendo dois tipos de serviço de sessão:

- Serviço de Sessão Sem Conexão: opera sobre um serviço de transporte de pacote de dados seguro ou não. É um serviço de sessão não confiável, ou seja, opera sobre o WDP. Neste modo, somente a primitiva de requisição (request) está disponível para os usuários do serviço, e só a primitiva de indicação (indication) está disponível para o provedor do serviço. O serviço sem conexão é muito bom, quando aplicações não precisam de uma entrega de dados confiável e não se

importam com a confirmação. Pode ser usado sem ter que estabelecer uma sessão de fato.

- Serviço de Sessão Com Conexão: opera sobre um protocolo da camada de transação (WTP). É um serviço de sessão confiável. Neste modo, as primitivas de requisição (request) e de confirmação (confirm) estão disponíveis para os usuários do serviço, e as primitivas de indicação (indication) e de resposta (response) estão disponíveis ao provedor de serviço.

4.3.6. CAMADA DE APLICAÇÃO: *WIRELESS APPLICATION ENVIRONMENT (WAE)*

A camada de Aplicação (WAE) é um resultado do empenho do WAP para promover especificações e padrões para toda a indústria para o desenvolvimento de aplicações e serviços que operam sobre redes de comunicação sem fio. WAE especifica um sistema de aplicação para dispositivos sem fio tal como telefones móveis, *pagers* e PDAs (*Personal Digital Assistants*). O sistema se estende e incorpora outras tecnologias WAP, inclusive o WTP e o WSP, como também outras tecnologias de Internet como XML, URLs, *scripting* e vários formatos de conteúdo [WAE99]. O esforço tem como objetivo capacitar os operadores, fabricantes e desenvolvedores a adequar os desafios de implementar serviços e aplicações avançadas de forma diferenciada, de uma maneira rápida e flexível.

A arquitetura WAE inclui todos os elementos da arquitetura WAP relacionados à especificação e execução de aplicações. Neste ponto, a arquitetura WAE é predominantemente concentrada nos aspectos do cliente na arquitetura de sistema do WAP, especialmente itens relacionados aos agentes de usuário, como o *microbrowser* ou dispositivo interpretador de WML (*Wireless Markup Language*).

A Fig. 4.8 ilustra os componentes do WAE

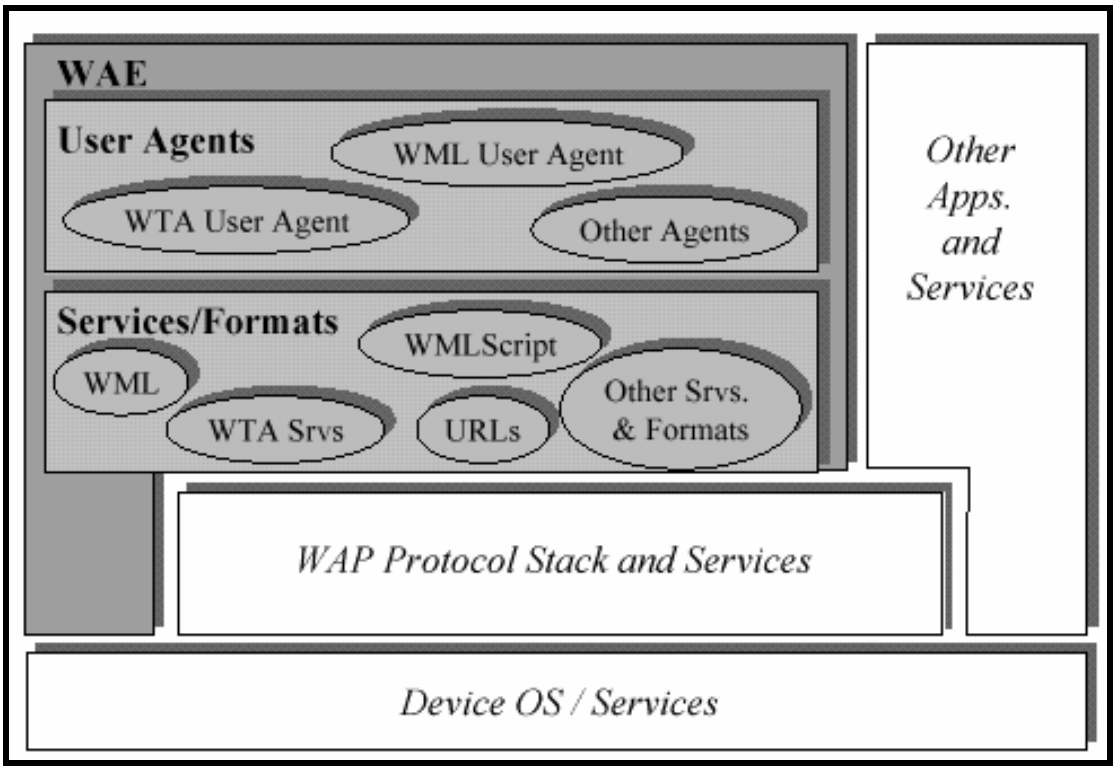


FIGURA 4.8 – Componentes do Modelo WAE.

Os principais componentes são:

- O WML (Wireless Markup Language): uma linguagem similar à linguagem HTML, mas otimizada para o uso em terminais móveis [WML99].
- O WMLScript: uma linguagem similar à JavaScript [WMLS99].
- O WTA (Wireless Telephony Application): WTA é uma coleção de extensões específicas de telefonia para mecanismos de chamada e de características que provêm aos desenvolvedores e usuários Serviços Avançados de Telefonia Móvel. Ver Fig. 4.9.
- Formatos de Conteúdo: Compreende uma série de formatos de conteúdo bem definidos incluindo imagens, entradas de lista telefônica e informações de agenda.

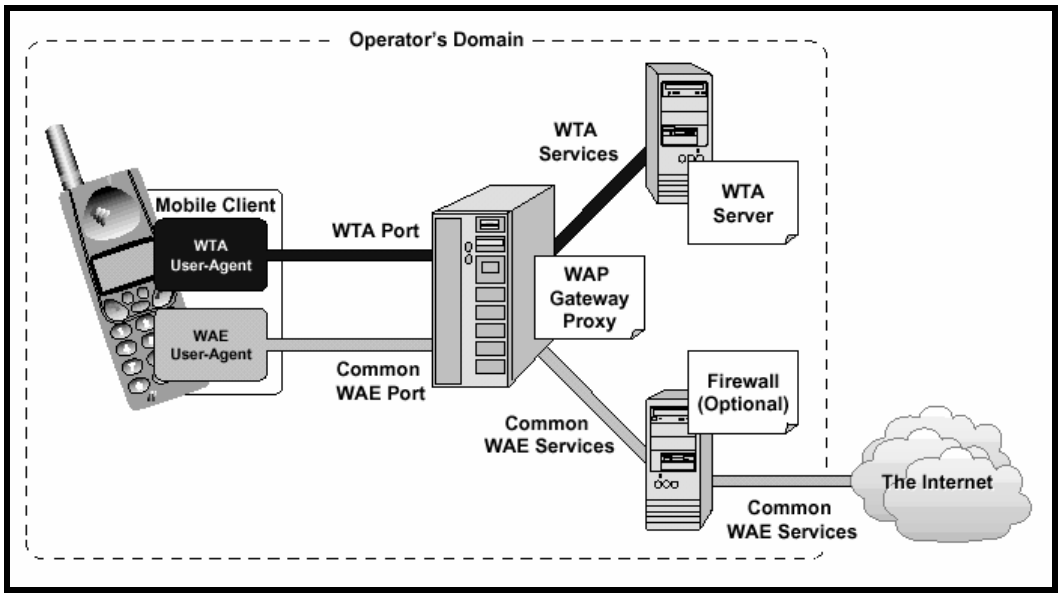


FIGURA 4.9 – WTA no Domínio da Operadora de Telefonia.

Especificamente, a arquitetura WAE é definida principalmente em termos de esquemas de rede, formatos de conteúdo, linguagens de programação e serviços compartilhados. Interfaces não são padronizadas e são específicas a uma implementação particular. Esta aproximação permite ao WAE ser implementado de várias formas sem abrir mão da interoperabilidade e portabilidade.

WAE adota um modelo que, de perto, segue o modelo WWW. Todo o conteúdo é especificado em formatos que são semelhantes aos formatos padrões da Internet. O conteúdo é transportado usando protocolos padrões no domínio WWW, e no domínio *wireless* são usados os protocolos WAP.

A arquitetura WAE permite que todo o conteúdo e os serviços que estão em servidores da Web possam ser incorporados com tecnologias comprovadas (por exemplo, PHP). Todo o conteúdo é localizado usando URLs da Web.

WAE melhora alguns dos padrões da Web levando em conta as características do dispositivo e da rede. Extensões WAE são adicionadas para suportar os Serviços da Rede Móvel como: Controle de Chamada e envio de mensagens. É prestada atenção cuidadosa à memória e à CPU que são encontradas em terminais móveis. Suporte para redes de largura de banda estreita e de alta latência estão incluídos na arquitetura.

WAE assume a existência de um *gateway* responsável em codificar e decodificar dados transferidos de e para o cliente móvel. O propósito da codificação do conteúdo entregue ao cliente é minimizar o tamanho dos dados enviados ao cliente, como também, minimizar a energia computacional requerida pelo cliente, para processar esses dados.

4.4. O WML

4.4.1. ASPECTOS GERAIS

O WML é uma linguagem de documento baseada em etiquetas ou *tags*. Em particular, é uma aplicação de linguagem de marcação generalizada. WML compartilha uma herança com o HTML da Web (HTML-4) e com a Linguagem de Marcação de Dispositivos Sem Fio (HDML-2 - *Handheld Device Markup Language*). WML é especificado como um tipo de documento XML. Está aperfeiçoado para dispositivos de capacidade limitada como telefones e outros terminais móveis sem fio [WML99].

O WML e seu ambiente de suporte foram projetados com certas limitações para pequenos dispositivos de largura estreita considerando pequenos visores (*displays*), aparelhos receptores limitados, conexões de rede de faixa estreita, recursos de memória limitados e recursos computacionais limitados.

O WML está baseado em um subconjunto do HDML versão 2.0. O WML muda alguns elementos adotados pelo HDML e introduz novos elementos, alguns dos quais foram modelados em elementos semelhantes do HTML. O WML resultante implementa um cartão (*card*) e uma metáfora de maço de cartões (*deck*), como ilustra a Fig. 4.10. O *deck* contém construções que permitem a aplicação especificar documentos compostos de vários cartões. Uma interação com o usuário é descrita em um conjunto de cartões que podem ser agrupados em um documento, que é um arquivo “.wml”.

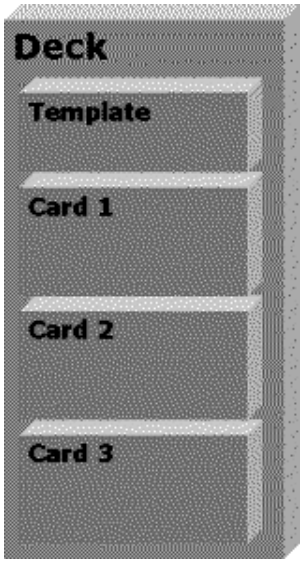


FIGURA 4.10 – Formato de um *Deck*.

A parte nomeada de *template* se comporta como uma função que é compartilhada por todos os *cards* de um *deck*. Logicamente, um usuário navega através de um conjunto de cartões. O usuário navega em um cartão, examina seu conteúdo, pode requisitar informação, pode fazer escolhas, e então passa para outro cartão. Instruções embutidas dentro de cartões podem requisitar serviços em servidores de origem, enquanto necessário pela interação particular. *Decks* são trazidos de servidores de origem, à medida que são requisitados. *Decks* WML podem ser armazenados em arquivos estáticos em um servidor de origem, ou eles podem ser dinamicamente gerados por um gerador de conteúdo que é executado em um servidor de origem. Cada cartão, em um *deck*, contém uma especificação para uma particular interação de usuário.

O WML é especificado de uma certa maneira que permite ser usado em uma grande variedade de dispositivos, mas permitindo que os fornecedores incorporem suas próprias Interfaces Homem-Máquina (*MMI - Man-Machine Interface*). Por exemplo, WML não especifica como implementar requisições introduzidas por um usuário. Ao invés disso, WML especifica a ação de uma maneira abstrata. Isto permite que o WML seja implementado em uma grande variedade de dispositivos e mecanismos receptores. Implementações podem, por exemplo, escolher uma interface baseada em comandos de voz ou que o usuário introduza o endereço da Web. O agente de usuário deve decidir a melhor maneira de apresentar todos os elementos dentro de um cartão, dependendo das

capacidades de dispositivo. Por exemplo, certos agentes de usuário com amplos visores podem escolher apresentar toda informação em um único cartão de uma vez. Por outro lado, outros dispositivos com visores menores podem separar ou quebrar o conteúdo por várias unidades de exibição.

4.4.2. PRINCIPAIS CARACTERÍSTICAS

4.4.2.1. Suporte para Texto e Imagens

WML proporciona, aos desenvolvedores, meios para especificar texto e imagens a serem apresentados ao usuário. Isto pode incluir o *layout* e sugestões de apresentação. WML provê um conjunto de elementos de marcação de texto que incluem vários elementos de ênfase (por exemplo, tipo negrito, itálico, grande, etc.); vários modelos de quebra de linha; e tabelas.

4.4.2.2. Suporte para Entrada de Dados do Usuário

WML suporta vários elementos para solicitar a entrada de dados do usuário. Os elementos podem ser combinados em um ou mais cartões. Todas as requisições para entrada de dados do usuário são feitas em termos abstratos, permitindo ao agente de usuário a liberdade para aperfeiçoar características para o dispositivo em particular. WML inclui um pequeno conjunto de controles de entrada. Por exemplo, WML inclui um controle de entrada de texto e de senha. Campos de entrada de texto podem ser disfarçados impedindo que o usuário final entre com tipos de caracteres incorretos.

4.4.2.3. Pilha de Histórico e Navegação

WML permite vários mecanismos de navegação que usam URLs. Também revela um mecanismo de histórico de primeira classe. Navegação inclui *hyperlinks* no estilo do HTML, elementos de navegação intercartão, como também elementos de histórico de navegação.

4.4.2.4. Suporte Internacional

O conjunto de caracteres para WML é o ISO/IEC-10646. Atualmente, este conjunto de caracteres é idêntico a *Unicode 2.0*. Não há nenhuma exigência que os *decks* WML sejam codificados usando toda a codificação *Unicode*. Qualquer codificação de caracteres que contém um adequado subconjunto formal dos caracteres lógicos em *Unicode*, podem ser usados (por exemplo, EUA-ASCII, ISO-8859-1, UTF-8, Shift_JIS, etc.).

4.4.2.5. Independência de Interface Homem-Máquina (MMI)

A especificação abstrata de *layout* e de apresentação do WML permite a vendedores de dispositivos e terminais controlarem o projeto da MMI de seus produtos.

4.4.2.6. Otimização para Faixa-Estreita

WML inclui uma variedade de tecnologias para aperfeiçoar a comunicação em um dispositivo de faixa-estreita. Isto inclui a habilidade para especificar interações múltiplas (cartões) de usuário em uma transferência de rede (um *deck*). Também inclui uma variedade de meios de administração de estado que minimizam a necessidade de requisições ao servidor de origem. O WML inclui outros mecanismos que ajudam a melhorar o tempo de resposta e minimizar a quantia de dados trocados numa sessão.

4.4.3. UTILIZAÇÃO DO WMLSCRIPT

O WMLScript ou WMLS, é uma linguagem de *script* procedural leve. Melhora o padrão de navegação e de apresentação do WML com capacidades de comportamento, adiciona inteligência para o cliente, provê um mecanismo conveniente para acessar o dispositivo e seus periféricos, e reduz a necessidade de "idas-e-voltas" para o servidor de origem.

O WMLS está baseado em um subconjunto da Linguagem de *Scripting WWW JavaScript*™ e na *ECMAScript*. É um subconjunto estendido do *JavaScript*™ e forma um meio padronizado para adicionar lógica de procedimentos para *decks* WML. WMLS refina o JavaScript™ para o dispositivo de faixa-estreita, integra isto com WML, e provê "ganchos" para integrar futuros serviços e aplicações em dispositivos.

O WMLS proporciona para o programador de aplicação uma variedade de capacidades interessantes:

- A habilidade de conferir a validade dos dados de entrada do usuário, antes que estes sejam enviados ao servidor de conteúdo;
- A habilidade de acessar dispositivos e seus periféricos;
- A habilidade de interagir com o usuário sem introduzir muitos acessos ao servidor de origem (por exemplo, exibir uma mensagem de erro).

4.5. CONCLUSÃO

O WAP é uma tecnologia que foi desenvolvida para proporcionar uma interconexão com a Internet, diversificando ainda mais a criação de novas aplicações e a difusão de informações.

O WAP é uma tecnologia que surgiu recentemente, e que inicialmente foi prejudicada pelo número reduzido de aplicações disponíveis, largura de banda oferecida pelas companhias de telefonia celular insuficiente para evolução das aplicações,

tecnologias primitivas de transferência de dados nas redes de comunicação celular, e rejeição pela maioria dos usuários na utilização do WAP em conseqüência dos custos de sua utilização.

Com o desenvolvimento de novas tecnologias de transferência de dados pelas redes celulares, como o GPRS, e com a evolução e a diversificação das aplicações para a plataforma WAP, o uso de aplicações em telefones celulares, PDAs e dispositivos portáteis através de *Browsers* WAP está em franca evolução.

Este presente trabalho, dentro deste contexto, visa contribuir no desenvolvimento de novos tipos de aplicações para a plataforma WAP.

5. MODELO DE INTEGRAÇÃO DO WAP NO APOIO À GERÊNCIA DE REDES TCP/IP

5.1. INTRODUÇÃO

Como foi descrito nos capítulos anteriores, o gerenciamento de uma rede se faz necessário ou mesmo obrigatório, para manter ativos os serviços disponibilizados. Deve-se escolher um sistema de gerência de rede que possa atender de forma personalizada cada tipo de rede. Para as redes TCP/IP, as ferramentas de gerência estão cada vez mais sofisticadas, provendo inclusive integração com a Web.

Como um dos objetivos deste trabalho é propor um modelo de integração de uma aplicação WAP com o sistema de gerência SNMP utilizando somente ferramentas gratuitas ou de domínio público, levando-se em consideração as limitações do WAP em relação às características limitadas dos dispositivos, como telefones celulares, foi proposto um modelo de integração do WAP no apoio à gerência de redes TCP/IP, que está descrito neste capítulo.

5.2. MODELO PROPOSTO

A Fig. 5.1 ilustra o modelo de integração proposto. Este modelo foi idealizado com os seguintes objetivos:

- Diminuir o processamento no lado do cliente (dispositivo móvel);
- Aperfeiçoar o acesso aos dados de gerenciamento através de um mecanismo mediador de informações;

- Organizar os dados de gerenciamento para proporcionar uma maior capacidade de adaptação ao crescimento das informações manipuladas pelo sistema;
- Tornar o modelo proposto independente de outras aplicações de gerenciamento.

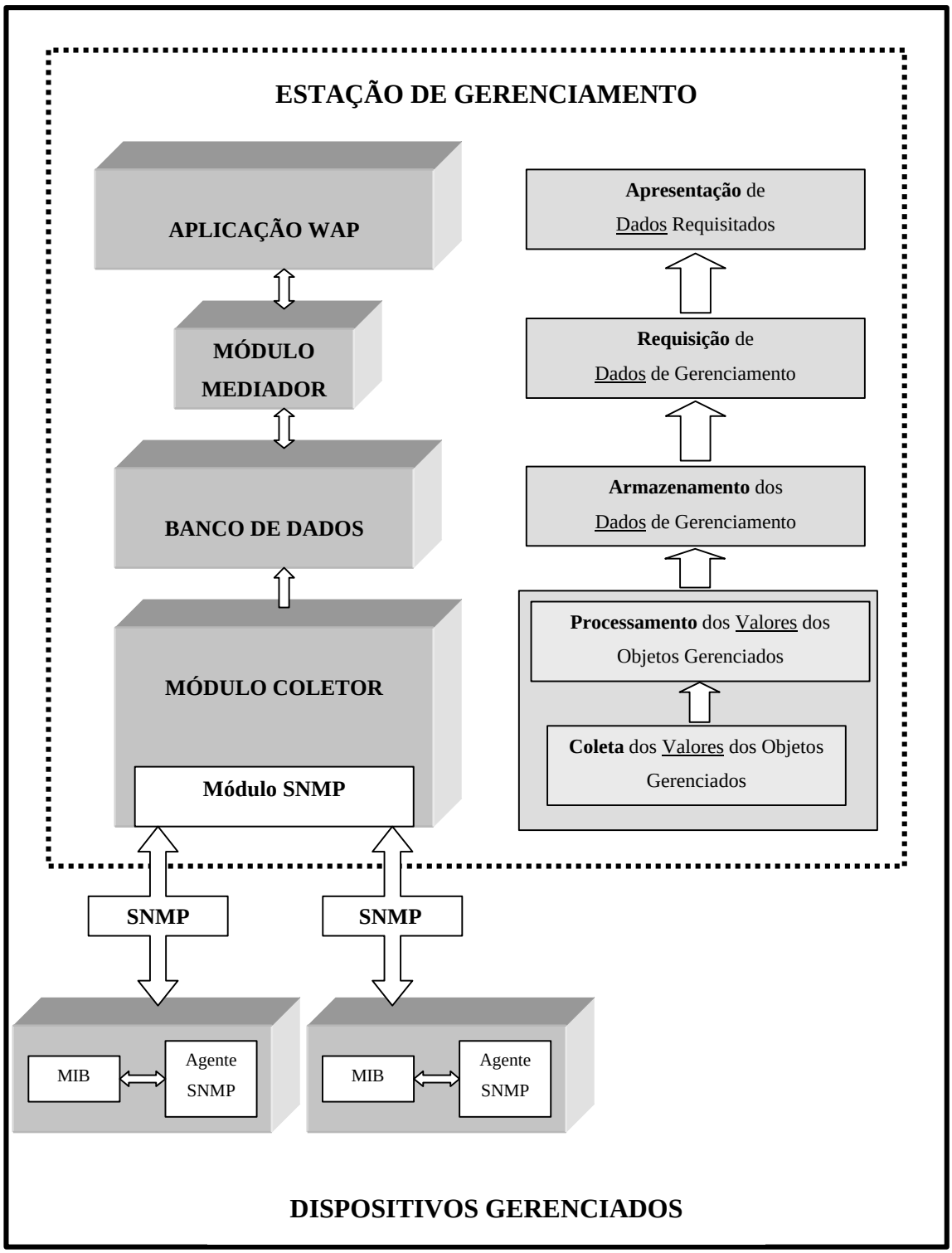


FIGURA 5.1 – Modelo Proposto e o Fluxo de Informações de Gerenciamento

O Módulo Coletor possui as funções de coletar os valores dos objetos gerenciados presentes na MIB dos dispositivos gerenciados e de processar esses valores para armazená-los no Banco de Dados. No Banco de Dados são armazenados somente os dados de gerenciamento, que serão requisitados pelo Módulo Mediador. O Banco de Dados poderá acumular dados de gerenciamento ao longo de um intervalo de meses ou anos, dependendo do nível de gerenciamento realizado. O Módulo Mediador, que funciona como um agente requisitor de dados entre o Banco de Dados e a Aplicação WAP, tem como principal função buscar os dados requisitados pela Aplicação WAP no Banco de Dados, onde todo o processamento dessa busca é realizado na Estação de Gerenciamento, deixando a cargo da Aplicação WAP a formatação dos dados na tela do dispositivo onde o *browser* WAP está sendo executado.

5.2.1. APLICAÇÃO WAP

A aplicação WAP é formada por um conjunto de páginas escritas em WML, com o objetivo de apresentar as informações a respeito do funcionamento da rede. As páginas foram elaboradas levando-se em conta as limitações dos dispositivos móveis. Além das limitações dos dispositivos móveis, para o desenvolvimento da interface de gerência, a aplicação foi desenvolvida para proporcionar o maior conforto possível ao usuário. Para isso, foram determinadas algumas diretrizes:

- Desenvolvimento de *cards* de tamanho limitado, para proporcionar um menor processamento por parte do dispositivo que contém o *browser* WAP. Ou seja, *decks* contendo vários *cards* com menor tamanho possível;
- Número limitado de caracteres em cada *card*. Nesse caso, usa-se como referência um modelo de dispositivo com a menor tela do mercado. Infelizmente não existe uma padronização com relação ao tamanho da tela, ao número de caracteres por linha ou ao número de linhas que uma tela pode ter;
- Utilização de lista de opções ao invés de campos de digitação, pois, a inclusão de dados por parte do usuário geralmente é dificultada pelo tamanho do dispositivo. Essa diretriz também contribui para a otimização do tempo de uso

da interface de gerenciamento, pois o usuário pode realizar uma operação mais rapidamente.

5.2.2. MÓDULO MEDIADOR

É o módulo que requisita as informações ao banco de dados, com o objetivo de proporcionar essas informações de forma dinâmica para a aplicação WAP. Este módulo age como um mediador de informações [FLO98], buscando os dados específicos requisitados pela aplicação WAP, que estão organizadas em forma de tabela na base de dados. O Módulo Mediador é necessário, pois o Banco de Dados pode crescer de acordo com o número de objetos gerenciados pelo Módulo Coletor. O Módulo Mediador foi implementado utilizando o PHP.

PHP é uma linguagem de elaboração de *scripts*, ou seja, é uma linguagem que processa, através de *scripts*, solicitações feitas por um cliente (um telefone celular com *browser* WAP ou um computador com algum navegador) e devolve o resultado para o cliente em arquivos no formato WML ou HTML [PHP02].

Como o PHP é reconhecido ?

Um script PHP é delimitado por tags, como as descritas abaixo:

<?php ?> : padrão XML
<? ?> : padrão SGML
<script language=”PHP" > </script>

O mais usual no WML é usar o formato XML:

<?php ?>

O que distingue o PHP de algo como *Javascript client-side*, é que o código é executado no servidor e não no dispositivo do cliente. Essa característica é determinante para a utilização do PHP com o WML, pois quanto menor for o processamento no lado do cliente (dispositivo móvel), menor será o tempo de navegação por parte do usuário.

5.2.3. BANCO DE DADOS

É uma base de dados contendo as informações que serão requisitadas pelo Módulo Mediador. Estas informações são os dados de gerenciamento, que são os resultados do processamento dos valores dos objetos gerenciados realizado pelo Módulo Coletor. Com o objetivo de uma melhor organização e de proporcionar uma melhor capacidade de adaptação para um futuro aumento dos dados gerenciados, as tabelas foram divididas de acordo com o tipo de gerenciamento, como mostra a Fig. 5.2.

O Banco de Dados usado é o MySQL, o qual é um servidor de banco de dados SQL multi-usuário e *multi-threaded* [MSQ02]. SQL é a linguagem de banco de dados mais popular no mundo. MySQL é uma implementação cliente-servidor que consiste de um servidor e de diferentes programas clientes e bibliotecas.

O servidor MySQL é também rápido e flexível o suficiente para permitir armazenar *logs* e objetos gráficos. As principais vantagens do MySQL são velocidade, robustez e facilidade de uso.

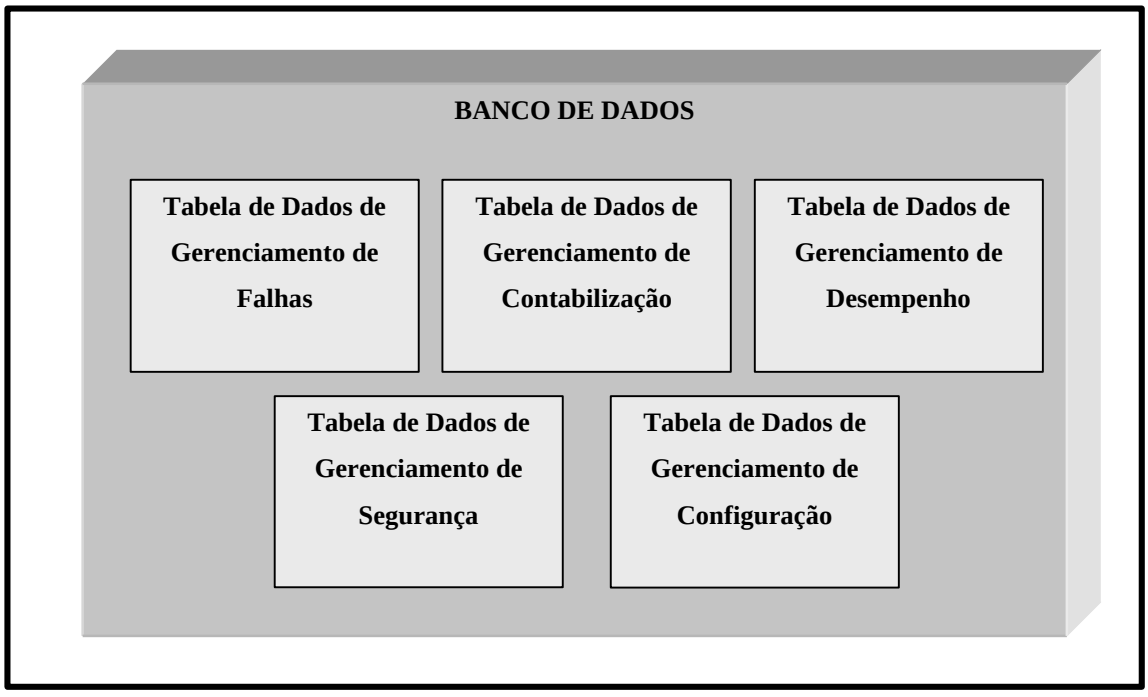


FIGURA 5.2 – Divisão dos Dados de Gerenciamento em Tabelas.

5.2.4. MÓDULO COLETOR

Pode ser considerado como o mecanismo de busca dos valores dos objetos gerenciados nos dispositivos monitorados. As informações são processadas e enviadas para o banco de dados. É neste módulo que é feito um mapeamento das diversas áreas funcionais do modelo OSI sobre os objetos gerenciados. A linguagem utilizada foi o *Perl* [PER02]. Diferentemente do que foi proposto em [SAU99], a aplicação gerente comunica-se diretamente com os agentes SNMP, pois ela utiliza um sub-módulo escrito também em *Perl*, assim não necessitando de nenhuma outra ferramenta e, por conseqüência, torna o modelo proposto independente de outras aplicações de gerenciamento.

Perl significa *Practical Extraction and Report Language*, roda em UNIX, VMS, MS/DOS, *Windows*, *Macintosh*, OS/2, Amiga e outros sistemas operacionais. *Perl* possui funções poderosas para manipulação de textos. A linguagem *Perl* é muito popular para programação de formulários WWW e *gateway* entre sistemas, banco de dados ou de usuários, além de ser muito utilizada em tarefas administrativas de sistemas UNIX, onde a linguagem foi criada e desenvolvida [WAL01].

5.3. IMPLEMENTAÇÃO E AMBIENTE DE TESTES

5.3.1. DESCRIÇÃO DO AMBIENTE DE TESTES

Para realização dos testes, foi utilizado o ambiente de rede do LRG (Laboratório de Redes e Gerência) localizado no Departamento de Informática e Estatística no Centro Tecnológico da Universidade Federal de Santa Catarina. A rede do LRG é composta por sete microcomputadores padrão PC, uma estação *Sparc* e um *hub* gerenciável que possui uma conexão com o Núcleo de Processamento de Dados da UFSC. A Fig. 5.3 ilustra a rede do LRG.

O dispositivo gerenciado foi o *Hub*, por ser responsável pela conexão da Rede do LRG ao NPD. A estação TAURUS é a responsável pelo gerenciamento e é onde se encontra implementado o modelo proposto. É apresentada, no Quadro 5.1, a descrição dos dispositivos envolvidos no teste.

	Hardware	**Software**
Estação TAURUS	- Processador: Athlon 1,13 GHz - Memória: 512 MB	- SO: Windows XP Pro - Servidor Web: Apache v.1.3.26 - PHP: v.4.2.2 - Banco de Dados: MySQL v.3.23.52 - Interprtador Perl: ActivePerl v.5.6.1 - Simulador WAP: OpenWave UP.4.1
Hub	- Hub Ethernet Accton com 16 portas.	- Agente SNMP - MIB-II

QUADRO 5.1 – Descrição dos Dispositivos.

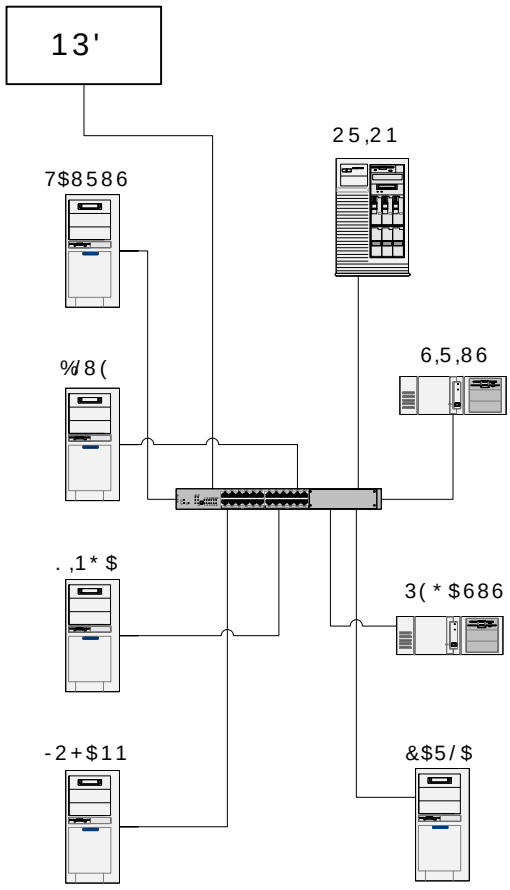


FIGURA 5.3 – Representação da Rede do LRG.

5.3.2. ESCOLHA DA MIB E DOS OBJETOS GERENCIADOS UTILIZADOS

Para os testes foi escolhida a MIB-II por estar presente em praticamente todos os equipamentos gerenciáveis através do protocolo SNMP.

O critério para selecionar quais variáveis foram utilizadas na monitoração de cada dispositivo, é baseado no tipo de área funcional de gerência (Falha, Desempenho, Contabilização, Configuração e Segurança). Para isso foi realizado um mapeamento das áreas funcionais do gerenciamento OSI sobre os objetos gerenciados da MIB. (ANEXO I)

Para a realização dos testes foram escolhidas as seguintes variáveis (objetos gerenciados):

- **ifInOctets** - taxa de bytes recebidos
- **ifInUcastPkts** - taxa de pacotes *unicast* recebidos
- **ifInNUcastPkts** - taxa de pacotes *no-unicast* recebidos
- **ifInErrors** - taxa de erros de entrada
- **ifOutOctets** - taxa de bytes enviados
- **ifOutUcastPkts** - taxa de pacotes *unicast* enviados
- **ifOutNUcastPkts** - taxa de pacotes *no-unicast* enviados
- **ifOutErros** - taxa de erros de saída
- **ifSpeed** - largura de banda da interface

As variáveis acima foram escolhidas para a implementação com o objetivo de demonstrar as funções do Módulo Coletor tanto na leitura dos objetos gerenciados quanto no processamento dos valores das variáveis, resultando em dados de gerenciamento, como será mostrado adiante.

Através desses objetos, foi possível determinar, os seguintes dados de gerenciamento:

- **Taxa em kbps (entrada)**
  = [(ifInOctets x 8) / 1000] / 60
- **Taxa em kbps (saída)**

= [(ifOutOctets x 8) / 1000] / 60

- **Taxa de Utilização (entrada)**
  = [(Taxa em bits/s de entrada) / ifSpeed] x 100
- **Taxa de Utilização (saída)**
  = [(Taxa em bits/s de saída) / ifSpeed] x 100
- **Porcentagem de Erro de Entrada**
  = [ifInErrors / (ifInUcastPkts + ifInNUcastPkts)]
- **Porcentagem de Erro de Saída**
  = [ifOutErrors / (ifOutUcastPkts + ifOutNUcastPkts)]

O Módulo Coletor foi configurado para realizar um *polling* a cada sessenta segundos, requisitando os valores dos objetos gerenciados ao agente SNMP localizado no *hub*. O Módulo Coletor realiza o processamento dos valores e armazena os resultados na Tabela de Dados de Gerenciamento de Desempenho localizada no Banco de Dados. O Módulo PHP irá enviar estes dados para a Aplicação WAP quando esta última requisitar.

5.3.3. FERRAMENTAS UTILIZADAS

Na etapa de desenvolvimento foram utilizadas várias ferramentas, sendo:

- **Aplicação WAP:** para o desenvolvimento dos arquivos em WML, foi utilizado o editor EasyPad WAPtor e o Nokia Mobile Internet Toolkit 3.1. Foi utilizado também um simulador chamado OpenWave UP.4.1. Esta ferramenta simula um telefone celular com um *browser* WAP. A utilização deste simulador em específico se deve ao fato de que a grande maioria dos telefones celulares utilizam *browsers* da OpenWave, bem como, a maioria dos *gateways* utilizados nas companhias de telefonia celular.
- **PHP e MySQL:** foram utilizadas versões pré-compiladas e foram configuradas para funcionarem com o Servidor *Web Apache*.
- **Perl:** foi utilizado o software ActivePerl para execução dos *scripts perl* e o *Perl Express* para edição dos *scripts.*

- **Servidor WEB Apache:** foi utilizado este servidor web pela flexibilidade de configuração e principalmente pelo fato do mesmo possuir versões para vários sistemas operacionais.

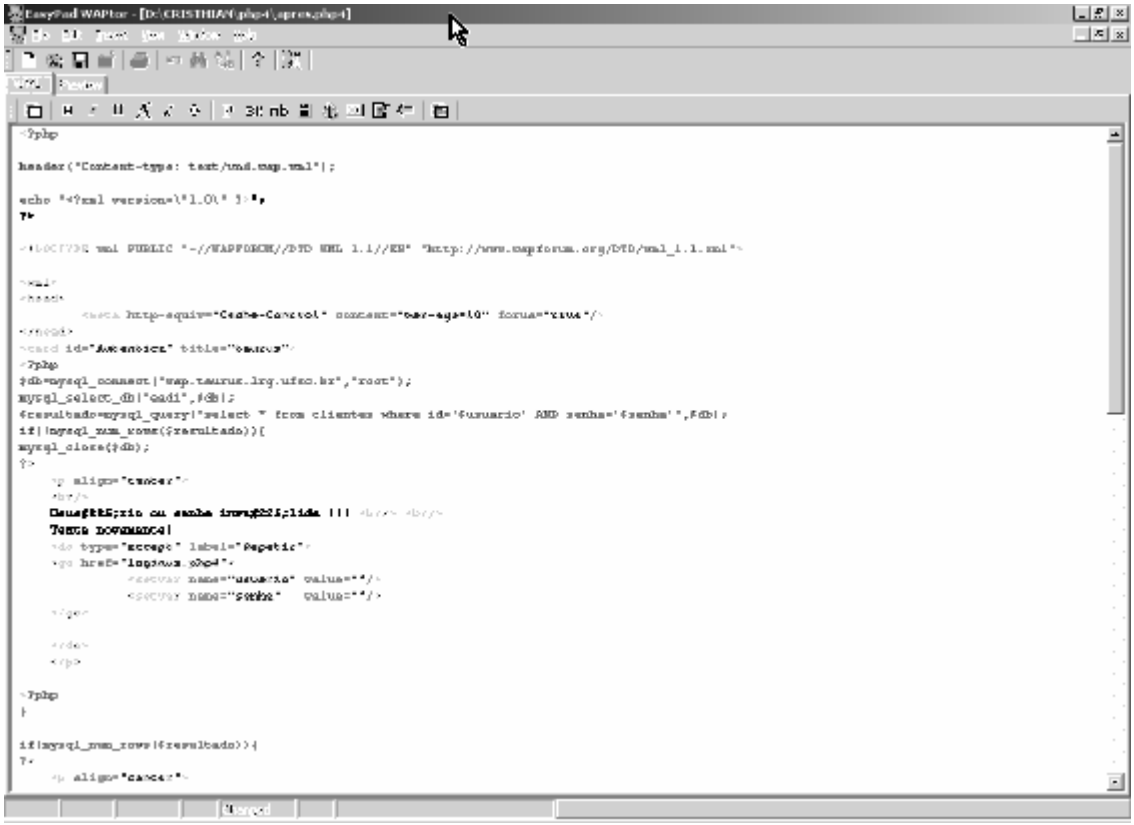


FIGURA 5.4 – Editor de WML/PHP EasyPad WAPTor.

5.3.4. DESCRIÇÃO DA IMPLEMENTAÇÃO

Como informado nos capítulos anteriores o modelo proposto de integração do WAP no apoio à gerência de redes TCP/IP foi implementado utilizando ferramentas de software livre e de domínio público. Além disso, todas as ferramentas utilizadas possuem versões para os sistemas operacionais mais populares como o UNIX, LINUX, FreeBSD e Microsoft Windows®. Pela facilidade encontrada em configurar as versões das ferramentas para o ambiente Windows®, escolheu-se este último como sistema operacional para implementação e testes.

A seguir, será descrito cada parte da implementação.

5.3.4.1. Módulo Coletor

Todo o módulo coletor foi escrito utilizando-se a linguagem Perl. Para edição do código foi utilizado o editor Perl Express por possuir recursos como endentação automática e diferenciação em cores dos argumentos e variáveis. A Fig. 5.5 ilustra a edição de um arquivo perl. O sub-módulo responsável pela busca dos objetos gerenciados foi dividido em dois scripts: *BER.pm* (ANEXO II) e o *SNMP_Session.pm* (ANEXO III). O script *BER.pm* (BER - Basic Encoding Rules) é responsável pela identificação de todos os tipos de variáveis de uma MIB SNMP. Com o *BER.pm* o interpretador perl instalado em qualquer sistema operacional, é capaz de tratar todos os tipos de objetos gerenciados, independente da plataforma. O script *SNMP_Session.pm* é responsável pelos parâmetros de busca dos objetos gerenciados aos agentes SNMP contidos nos dipositivos gerenciados.

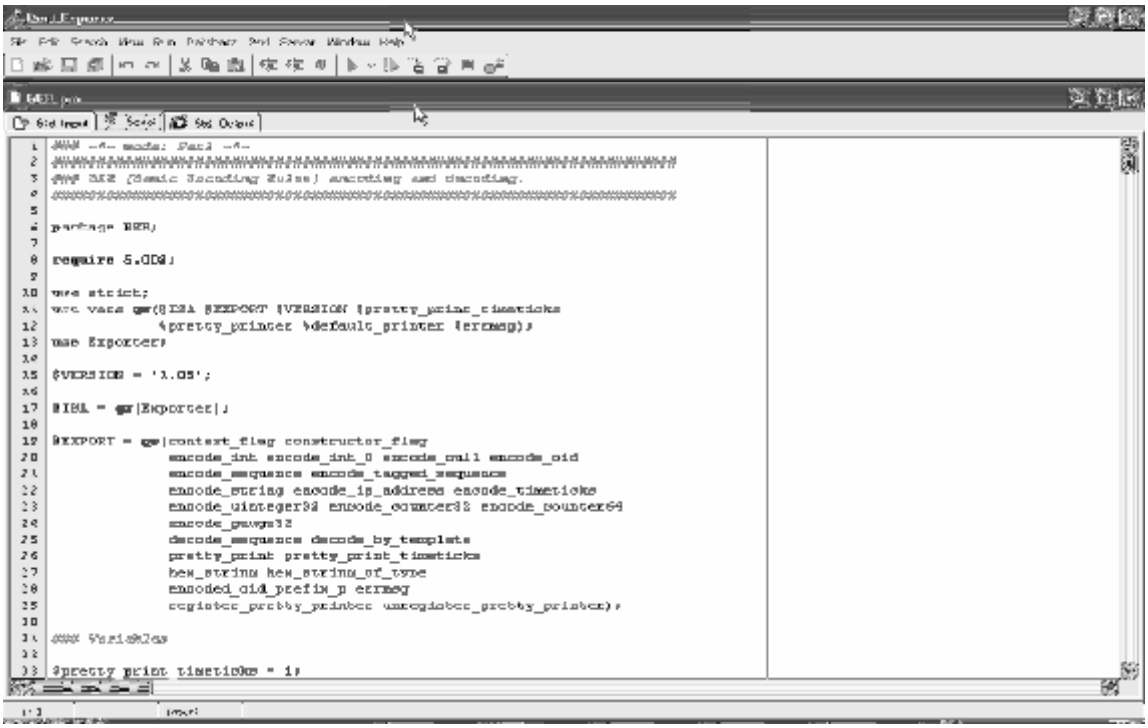


FIGURA 5.5 – Edição do *Script BER.pm* através da ferramenta *Perl Express*.

A partir dos dois scripts acima, *BER.pm* e o *SNMP_Session.pm*, a busca e o processamento dos objetos gerenciados foi realizado utlizando-se o script modelo *req_process.pm* (ANEXO IV). Para cada conjunto de objetos gerenciados foi

configurado um script explicitando as variáveis desejadas, o host e a porta de comunicação. O Módulo Coletor foi configurado para ler informações do HUB do LRG, especificamente a porta 01 do HUB, que faz a interligação do LRG ao NPD.

Para interpretar os scripts *BER.pm, SNMP_Session.pm* e o *req_process.pm* foi instalado no sistema operacional MS Windows® o interpretador *ActivePerl*. A vantagem de se usar a linguagem Perl em modo interpretado é garantir que o *script* possa funcionar em mais de uma plataforma, dependendo somente da instalação do interpretador no sistema. Destaca-se que os *scripts* em *Perl* também podem ser compilados, de acordo com a plataforma de hardware e software, se o número de alterações nos scripts for pequena ou nula, e o administrador de uma rede necessitar desse recurso.

5.3.4.2. Banco de Dados

Como informado nos itens anteriores, foi utilizado o sistema de gerenciamento de banco de dados MySQL por possuir versões para vários sistemas operacionais e por ser uma ferramenta de fácil integração ao PHP.

Devido à escolha dos objetos gerenciados declarados no item 5.3.2, foi implementada somente parte da Tabela de Dados de Gerenciamento de Desempenho, para armazenar os dados relacionados no Quadro 5.2.

Descrição	Dado	Resultado do Processamento do Módulo Coletor
Vazão ou Taxa em kbps de entrada	Vin	[(ifInOctets x 8) / 1000] / 60
Taxa de Utilização % de entrada	TUin	[Vin / ifSpeed] x 100
Porcentagem de Erro de Entrada	PEin	[ifInErrors / (ifInUcastPkts + ifInNUcastPkts)]
Vazão ou Taxa em kbps de saída	Vout	[(ifOutOctets x 8) / 1000] / 60
Taxa de Utilização % de saída	TUout	[(Vout/ ifSpeed] x 100
Porcentagem de Erro de Saída	PEout	[ifOutErrors / (ifOutUcastPkts + ifOutNUcastPkts)]

QUADRO 5.2 – Dados armazenados na Tabela de Gerenciamento de Desempenho.

5.3.4.3. Módulo Mediador

Para a implementação, o Módulo Mediador, que utiliza a linguagem PHP, foi escrito como um script inserido no código fonte da Aplicação WAP, pois, o WML que

é a linguagem usada para escrever páginas WAP é semelhante ao HTML no que diz respeito à apresentação de conteúdo: só apresenta informação estática. Como o HTML, o código WML precisa ser elaborado com algum mecanismo para apresentação de conteúdo dinâmico. O PHP realiza essa função adicionando características de uma linguagem de programação, aqui com a função de requisitar os dados de gerenciamento ao Banco de Dados visto que serão requisitados os valores de *Vin, TUin, PEin, Vout, TUout e PEout*. Essa abordagem não modifica o modelo proposto, pois, o módulo interpretador da linguagem PHP trabalha juntamente com o servidor WEB Apache. A implementação do código do Módulo Mediador como script PHP no código fonte da aplicação WAP, foi determinante para o modelo de integração proposto, pois no servidor WEB, todo script PHP é executado e somente o resultado desse processamento é enviado para o *browser WAP*, que está sendo executado num dipositivo portátil como um aparelho celular.

Para a implementação de validação, foi instalado na Estação de Gerenciamento TAURUS o servidor WEB Apache com os módulos de execução do PHP. Por padrão o Apache já vem habilitado para servir páginas WML, foi necessário apenas definir um *Mime-Type* dizendo que os arquivos com extensão *.wml deverão retornar um cabeçalho do tipo *Content-Type: text/vnd.wap.wml* para a aplicação que fez a requisição. As linhas inseridas no arquivo de configuração de *Mime-Type* estão no Quadro 5.3.

```
application/vnd.wap.sic
application/vnd.wap.slc
application/vnd.wap.wbxml        wbxml
application/vnd.wap.wmlc         wmlc
application/vnd.wap.wmlscriptc   wmlsc
```

QUADRO 5.3 – Configuração de *Mime-Type* no servidor WEB Apache.

Precisou-se, também, configurar o arquivo httpd.conf, incluindo as linhas que estão no Quadro 5.4.

```
AddType text/vnd.wap.wml wml
AddType image/vnd.wap.wbmp wbmp
AddType text/vnd.wap.wmlscript wmls
AddType application/vnd.wap.wmlc wmlc
```

QUADRO 5.4 – Configuração de *Mime-Type* no servidor WEB Apache.

5.3.4.4. Aplicação WAP

Para implementação da aplicação WAP, foi utlizado para editar os arquivos .wml/.php4 a aplicação *Notepad* do MS Windows®.

FIGURA 5.6 – Simulador WAP da OpenWave UP.4.1.

Como não havia um celular com browser WAP para testar a implementação, foi utilizado o Simulador WAP da OpenWave UP.4.1, pois o mesmo simula o comportamento de um Gateway WAP e um telefone celular com browser WAP. A Fig. 5.6 ilustra a ferramenta sendo executada.

Foram implementados os códigos para as telas mostradas na Fig. 5.7 e Fig. 5.8.

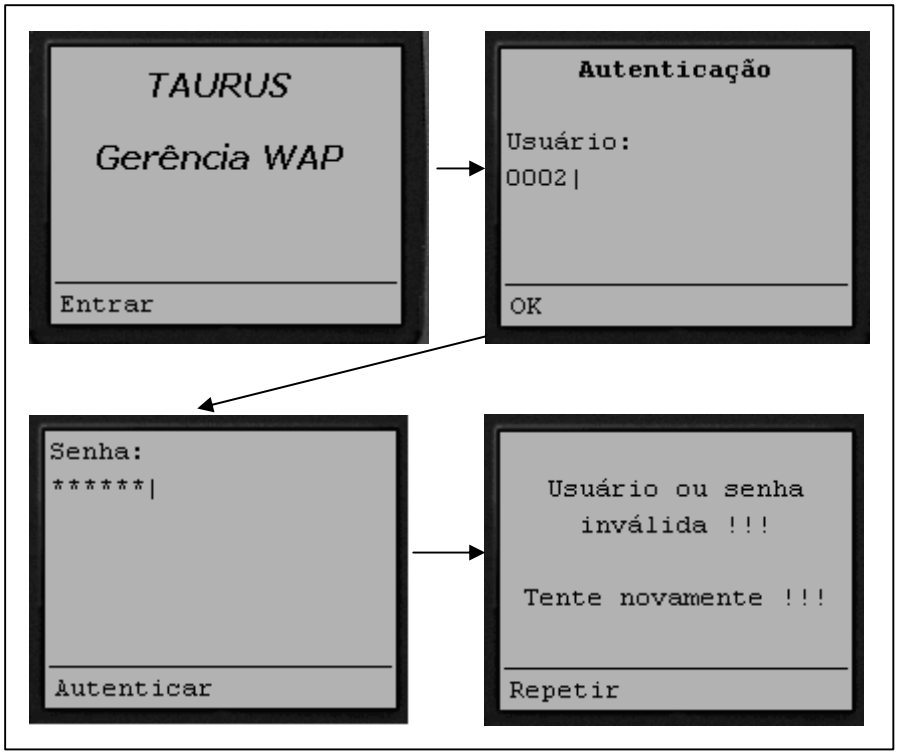


FIGURA 5.7 – Tela de Apresentação e Autenticação.

- Tela de Apresentação no Sistema de Gerência para entrada da identificação e autenticação do usuário cadastrado, de acordo com a Fig. 5.7;
- Tela da apresentação dos valores de *Vin, TUin, PEin, Vout, TUout e PEout,* de acordo com a Fig. 5.8.

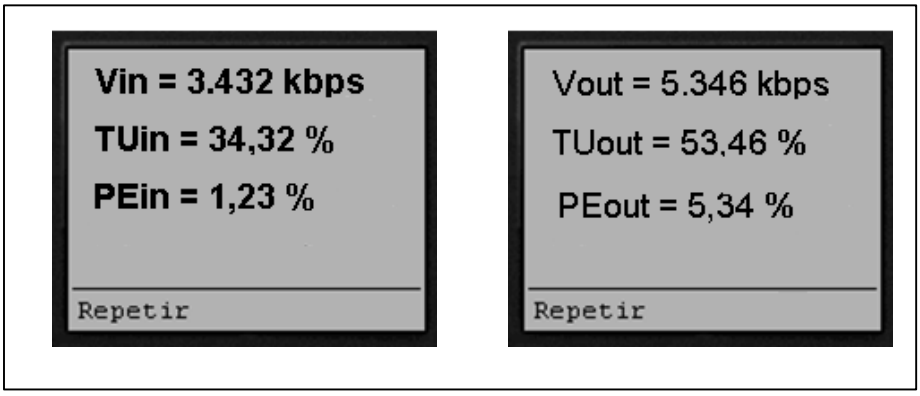


FIGURA 5.8 – Exemplo das Telas de Consulta.

Os códigos da Aplicação WAP sem encontram no ANEXO V.

6. ANÁLISE DOS RESULTADOS

Problemas ocorreram na utilização das rotinas de leitura dos valores dos objetos gerenciados pelo fato do sub-módulo, que faz as requisições aos agentes SNMP, não estar estável o suficiente. Foram feitas algumas modificações no código do sub-módulo para que os testes fossem realizados.

A interface com o usuário se comportou de maneira satisfatória, proporcionando uma navegação confortável. A velocidade na apresentação dos dados foi agradável, variando de acordo com número de linhas mostradas e quantidade de dados requisitados. O Módulo PHP diminuiu bastante a complexidade da implementação no que se refere à apresentação e busca dos dados de gerência, principalmente pelo fato do PHP interagir muito bem com o MySQL.

Mesmo assim, em relação à configuração dos módulos para apresentação das informações ao usuário do browser WAP, constatou-se alguns obstáculos no processo de implementação. Toda modificação ou inclusão de apresentação de um dado de gerenciamento na aplicação WAP, requer modificação nos códigos do Módulo Mediador, para que o mesmo possa requisitar as informações no Banco de Dados. Se as informações requisitadas não estiverem na base de dados, há a necessidade de criar uma entrada no Banco de Dados para que o mesmo possa receber os dados de gerenciamento do Módulo Coletor. O código do Módulo Coletor deve ser alterado para a inclusão das linhas de programação referentes ao processamento dos valores dos objetos gerenciados, bem como das linhas referentes à busca desses valores nos dispositivos gerenciados. Esses problemas de implementação ocorreram por não ter sido implementado um módulo de configuração que fizesse as modificações necessárias para a apresentação de novos dados de gerenciamento, simultaneamente nos módulos Aplicação WAP, Mediador, Banco de Dados e Coletor. Para a construção de uma ferramenta funcional de gerência de redes baseada no modelo proposto de integração do WAP no apoio à gerência de redes TCP/IP, a adição de um módulo de configuração será

extremamente necessária. A Fig. 5.8 ilustra a inclusão de um módulo de configuração ao modelo de integração proposto.

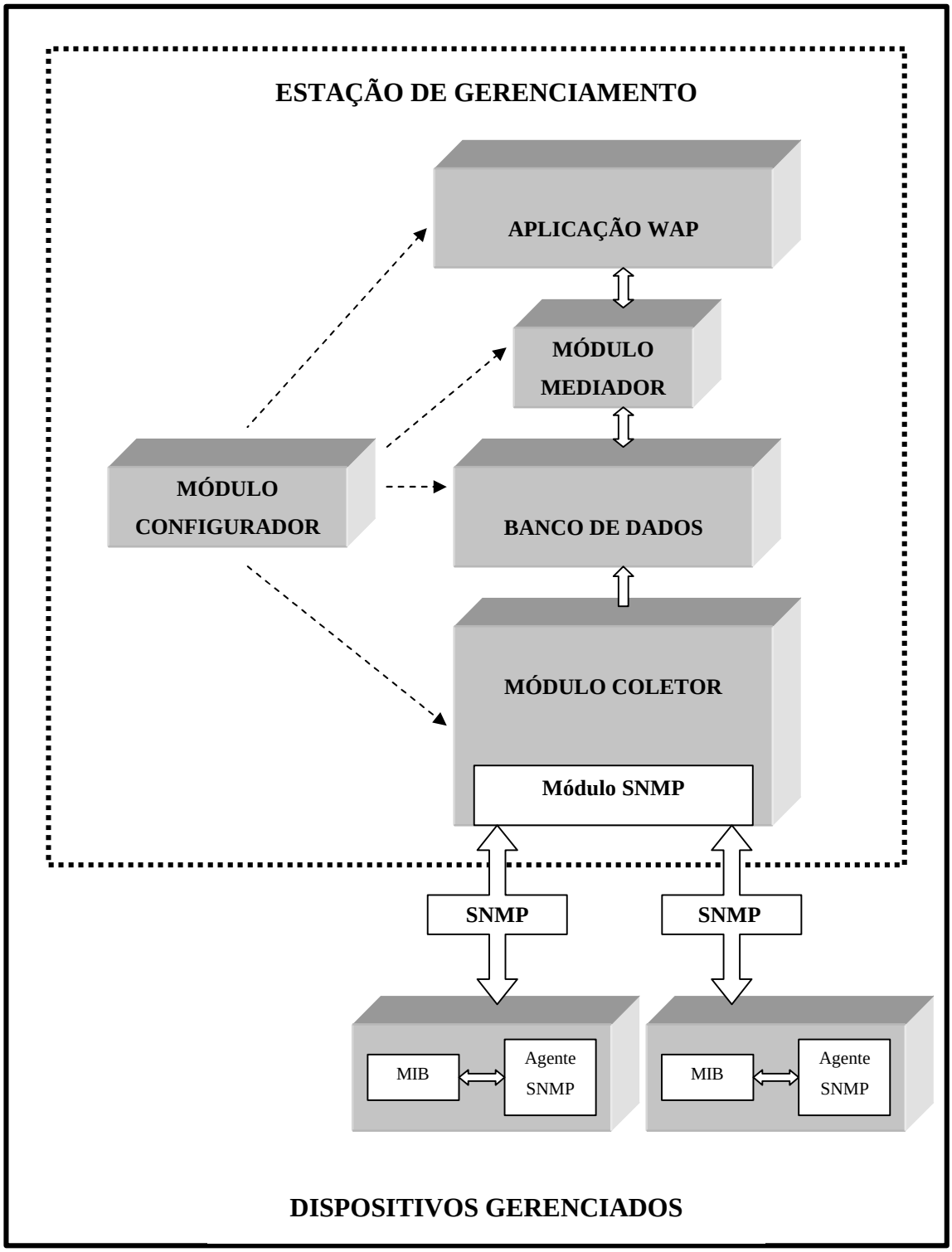


FIGURA 5.9 –Módulo Configurador.

Apesar das dificuldades encontradas na implementação, pôde-se constatar que o modelo de integração do WAP no apoio à gerência de redes TCP/IP é valido, pelo fato de todas as aplicações e ferramentas utilizadas estarem de acordo com os objetivos específicos apresentados no início deste trabalho.

No geral, a implementação do modelo no ambiente do Sistema Operacional Windows, foi facilitada pela interface gráfica deste sistema e pelo fato das versões das ferramentas destinadas a esse sistema operacional serem mais simples de instalar do que num ambiente Unix. Como existem versões das ferramentas utilizadas para várias plataformas, o modelo proposto pode ser implementado em qualquer uma.

7. CONCLUSÃO

Pelos testes realizados verificou-se que o modelo proposto atingiu os objetivos gerais propostos no Capítulo 1. Através da abordagem em elaborar o modelo em vários módulos consegui-se atingir os objetivos específicos que foram:

- A diminuição do processamento no lado do cliente, através da elaboração de uma interface simples e funcional, foi realizada através do desenvolvimento de *decks* contendo um número de *cards* de acordo com o tipo de informação requisitada.
- A otimização do acesso aos dados de gerenciamento através de um mecanismo mediador de informações foi atingida utilizando o PHP para buscar os dados específicos requisitados pela aplicação WAP. Com o crescimento da base de dados, este módulo proporciona um meio rápido de busca, principalmente pela integração do PHP com o MySQL.
- A organização dos dados de gerenciamento para proporcionar uma maior capacidade de adaptação ao crescimento das informações manipuladas pelo sistema, através da armazenagem dos dados em tabelas de acordo com o gerenciamento.
- A utilização de rotinas de acesso direto aos agentes SNMP, através de um sub-módulo escrito também em *Perl*, tornou o modelo proposto independente de outras aplicações de gerenciamento. Essa independência trouxe vantagens como a otimização das requisições aos agentes SNMP e a flexibilidade na alteração do modo como são feitas essas requisições.

Os problemas ocorridos durante os testes, demonstraram que o modelo de integração pode ser aperfeiçoado incluindo em sua arquitetura um Módulo Configurador como foi relatado na análise dos resultados. Como proposta o modelo de integração do

WAP à gerência de redes TCP/IP foi validado pela implementação, pois os objetivos propostos foram alcançados.

O uso de um dispositivo pequeno e portátil como um telefone celular para acompanhar o estado de uma rede, revelou-se como uma ferramenta útil para o gerenciamento de redes. Devido às limitações dos telefones celulares, no que se refere ao tamanho da tela e da tecnologia de transferência de dados usada atualmente, o uso desta solução é limitado quanto ao número de informações que se queira apresentar. Num futuro próximo, com o advento de novos mecanismos de transferência de dados, possibilitando maiores taxas de transferência, juntamente com a unificação de PDA´s com os telefones celulares, o uso de aplicações mais completas de gerenciamento em dispositivos móveis será possível.

7.1. TRABALHOS FUTUROS

Como sugestões para trabalhos futuros a serem realizados a partir do tema deste trabalho, cita-se:

1. A utilização do padrão XML, na base de dados, para uma melhor manipulação e padronização das informações de gerenciamento;
2. Aperfeiçoamento do Módulo Coletor, para ampliar os recursos de captação dos valores dos objetos gerenciados;
3. Inclusão de um recurso para configuração e envio de alarmes ao celular;
4. Utilização do Recurso Push da Arquitetura WAP, para manipulação de alarmes.
5. Implementação e teste de um Módulo de Configuração para a construção de uma ferramenta de gerência de redes TCP/IP, baseada no modelo de integração proposto, para que o mesmo seja aplicado para num ambiente de produção.

REFERÊNCIAS BIBLIOGRÁFICAS

[AND98] ANDRADE, Andréa de F. B. **Gerência de Redes Utilizando Soluções Baseadas em WEB**. Monografia de Especialização do GRC-UFPB. Campina Grande, dezembro, 1998.

[BAR98] BAROTTO, André M. **Realização da Gerência Distribuída de Redes Utilizando SNMP, Java, WWW e CORBA**. Dissertação de Mestrado do CPGCC-UFSC. Florianópolis, abril, 1998.

[BRI93] BrISA. **Gerenciamento de Redes – Uma Abordagem de Sistemas Abertos**. São Paulo, Makron Books, 1993.

[CER99] CERUTTI, Fernando A. **Implementação de um Ambiente de Gerência de Redes ATM Utilizando a Tecnologia Web**. Dissertação de Mestrado do CPGCC-UFSC. Florianópolis, agosto, 1999.

[DEN00] DENEGA, Marcos Antonio. **WAP Tecnologia Sem Fio**. São Paulo: Editora Berkeley, 2000.

[DIAS00] DIAS, Adilson de Souza. **WAP - Wireless Application Protocol: A Internet Sem Fios**. Rio de Janeiro: Ciência Moderna, 2000.

[FLO98] FLORESCU, Daniela; MENDELZON, Alberto; LEVY, Alon. **Database Techniques for the World Wide Web: A Survey**. ACM SIGMOD Record, 27, 3, 59-74, Setembro 1998.

[GER00] **Gerência de Redes**. Disponível em <http://penta2.ufrgs.br>. Acesso em 20 Agosto 2001.

[JOS99] JÚNIOR, José H. T.; SUAVÉ, Jacques P.; MOURA, José A. B. et al. **Redes de Computadores: Serviços, Administração e Segurança**. São Paulo: Makron Books, 1999.

[KED01] KEDZIERSKI, Daniel J. **Abordando o Desenvolvimento de Aplicações para a Difusão de Informações em Redes Sem Fio**. Dissertação de Mestrado do CPGCC-UFSC. Florianópolis, fevereiro, 2001.

[MAU01] MAURO, Douglas R.; SCHMIDT, Kevin J. **SNMP Essencial**. Rio de Janeiro: Ed. Campus, 2001.

[MEI97] MEIRELLES, Luiz F. T. **Uma Proposta para o Gerenciamento de Aplicações em Rede**. Dissertação de Mestrado do CPGCC-UFSC. Florianópolis, 1997.

[MSQ02] **MySQL Open Source Database**. Disponível em <http://www.mysql.com>. Acesso em 15 Março 2002.

[PAO99] **WAP Push Architectural Overview - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[PER02] **The Source For Perl**. Disponível em <http://www.perl.com>. Acesso em 15 Março 2002.

[PHP02] **Professional PHP - Programando**. CASTAGNETTO, Jesus; RAWAT, Harish; SCHUMANN, Sascha; et. Al.

[RIS01] RISCHPATER, Ray. **Desenvolvendo Wireless para Web**. São Paulo: Makron Books, 2001.

[SAU99] SAUVÉ, Jacques P. WebManager: **A Web-Based Network Management Application**. IEEE LANOMS'99 Latin American Network Operations and Management Symposium. Rio de Janeiro. Dezembro, 1999.

[STA99] STALLINGS, W. **SNMP, SNMPv2, SNMPv3 and RMON1 and 2**. Massachusetts, EUA: Addison Wesley, 1999. 3ª Edição.

[TAN97] TANENBAUM, Andrew S. **Computer Networks**. Rio de Janeiro: Ed. Campus, 2001.

[WAE99] **Wireless Application Environment Overview - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WAL01] WALL, Lerry; ORWANT, Jon; CHRISTIANSEN, Tom. **Programação Perl**. Rio de Janeiro: Ed. Campus, 2001.

[WAP98] **Wireless Application Protocol Architecture Specification - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WDP99] **Wireless Datagram Protocol Specification - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WIM99] **WAP Identity Module - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WML99] **Wireless Markup Language - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WMLS99] **Wireless Markup Language Script - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WSP99] **Wireless Session Protocol - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WTLS99] **Wireless Transport Layer Security Protocol - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

[WTP99] **Wireless Transaction Protocol Specification - WAP Forum**. Disponível em <http://www.wapforum.org>. Acesso em 20 Agosto 2001.

ANEXO I

Gerência de Configuração			
Grupo	**Variável**	**Tipo**	**Informação**
System	sysDescr	octet string	descrição do sistema
System	sysLocation	octet string	localização física do sistema
System	sysContact	octet string	pessoa responsável pelo sistema
System	sysName	octet string	nome do sistema
Interfaces	ifDescr	octet string	nome da interface
Interfaces	ifType	integer	tipo de interface
Interfaces	ifMtu	integer	tamanho máximo do datagrama suportado pela interface
Interfaces	ifSpeed	unsigned32	largura de banda da interface
Interfaces	ifAdminStatus	integer	indica se a interface esta administrativamente (up/down/test)
IP	ipFowarding	integer	se o dispositivo é configurado para IP
IP	ipAdEntAddr	ipAddress	o endereço IP desta entrada
IP	ipAdEntIfIndex	integer	numero da interface
IP	ipAdEntNetMask	ipAddress	máscara de sub-rede para endereços IP
IP	ipAdEntBcastAddr	integer	o bit menos significativo do endereco IP de broadcast
IP	ipRouteDest	ipAddress	endereço IP do destino
IP	ipRouteIfIndex	integer	número da interface
IP	ipRouteMetric1	integer	integer métrica de roteamento #1
IP	ipRouteMetric2	integer	integer métrica de roteamento #2
IP	ipRouteMetric3	integer	integer métrica de roteamento #3
IP	ipRouteMetric4	integer	integer métrica de roteamento #4
IP	ipRouteMetric5	integer	integer métrica de roteamento #5
IP	ipRouteNextHop	ipAddress	próxima escala (endereco IP do roteador, usado para roteamento indireto)
IP	ipRouteType	integer	tipo (indireto,direto,valido,invalido)
IP	ipRouteProto	integer	mecanismo usado para determinar a rota
IP	ipRouteAge	integer	idade da rota em segundos
IP	ipRouteMask	ipAddress	máscara da subrede para roteamento
IP	ipRouteInfo	object identifier	ponteiro MIB para um segundo protocolo de roteamento específico
TCP	tcpRtoAlgorithm	integer	algoritmo utilizado para determinar o "time-out"de retransmissão de octetos TCP não confirmados
TCP	tcpRtoMin	integer	valor mínimo permitido para o "time-out"de retransmissão TCP em milisegundos
TCP	tcpRtoMax	integer	valor máximo permitido para o "time-out"de retransmissão TCP, em milisegundos

ANEXO I

Gerência de Configuração			
Grupo	**Variável**	**Tipo**	**Informação**
TCP	tcpMaxConn	integer	limite de conexões que podem ser abertas pela entidade de transporte do dispositivo
TCP	tcpCurrEstab	unsigned32	número de conexões de transporte corretamente abertas
UDP	udpLocalAddress	ipAddress	endereço IP local
UDP	udpLocalPort	integer	porta UDP local
EGP	egpNeighState	integer	estado de cada vizinho EGP
EGP	egpNeighAddr	ipAddress	endereço IP do vizinho EGP
EGP	egpNeighAs	integer	sistema autônomo do vizinho EGP
EGP	egpNeighIntervalHello	integer	intervalo entre retransmissões de comandos Hello
EGP	egpNeighIntervalPoll	integer	intervalo entre retransmissões de comandos Poll
EGP	egpNeighMode	integer	modo de polling desta entidades EGP
EGP	egpNeighEventTrigger	integer	permite iniciar ou finalizar uma comunicação
EGP	egpAs	integer	sistema autônomo local
SNMP	snmpEnableAuthenTraps	integer	indica se o agente SNMP pode enviar traps

QUADRO I.1 – Mapeamento da Gerência de Configuração da MIB-II

Gerência de Falhas			
Grupo	**Variável**	**Tipo**	**Informação**
System	sysObjectID	object identifier	fabricante do sistema
System	sysServices	integer	qual camada de protocolo o sistema serve
System	sysUpTime	time ticks	quanto tempo o sistema está operacional
Interfaces	ifAdminStatus	integer	indica se a interface esta administrativamente (up/down/test)
Interfaces	ifOperStatus	integer	indica o status operacional da interface (up/down/test)
Interfaces	ifLastChange	time ticks	indica quando a interface mudou seu estado operacional
Interfaces	ipRouteDest	ipAddress	endereço IP do destino
Interfaces	ipRouteIfIndex	integer	número da interface
Interfaces	ipRouteMetric1	integer	integer métrica de roteamento #1
Interfaces	ipRouteMetric2	integer	integer métrica de roteamento #2
Interfaces	ipRouteMetric3	integer	integer métrica de roteamento #3
Interfaces	ipRouteMetric4	integer	integer métrica de roteamento #4

ANEXO I

Gerência de Falhas			
Grupo	**Variável**	**Tipo**	**Informação**
Interfaces	ipRouteMetric5	integer	integer métrica de roteamento #5
Interfaces	ipRouteNextHop	ipAddress	próxima escala (endereco IP do roteador, usado para roteamento indireto)
Interfaces	ipRouteType	integer	tipo (indireto,direto,valido,invalido)
Interfaces	ipRouteProto	integer	mecanismo usado para determinar a rota
Interfaces	ipRouteAge	integer	idade da rota em segundos
	ipRouteMask	ipAddress	máscara da subrede para roteamento
Interfaces	ipRouteInfo	object identifier	ponteiro MIB para um segundo protocolo de roteamento específico
Interfaces	ipNetToMediaIfIndex	integer	número da interface
Interfaces	ipNetToMediaPhysAddress	octect string	endereço do meio do mapeamento
Interfaces	ipNetToMediaNetAddress	ipAddress	endereço IP do mapeamento
Interfaces	ipNetToMediaType	integer	como o mapeamento foi determinado (outro,invalido,dinamico,estatico)
EGP	egpNeighState	integer	estado de cada vizinho EGP
EGP	egpNeighStateUps	counter32	número de vezes que um vizinho EGP entrou no estado up
EGP	egpNeighStateDows	counter32	número de vezes que um vizinho EGP entrou no estado down
SNMP	snmpInASNParseErrs	counter32	total de mensagens recebidas com erros ASN
SNMP	snmpInTooBigs	counter32	total de mensagens recebidas com erros "too big"
SNMP	snmpInNoSuchNames	counter32	total de mensagens recebidas com erros "noSuchName"
SNMP	snmpInBadValues	counter32	total de mensagens recebidas com erros "badValue"
SNMP	snmpInReadOnlys	counter32	total de mensagens recebidas com erros "readOnly"
SNMP	snmpInGenErrs	counter32	total de mensagens recebidas com erros "genErr"
SNMP	snmpOutTooBigs	counter32	total de mensagens enviadas com erros "too big"
SNMP	snmpOutNoSuchNames	counter32	total de mensagens enviadas com erros "noSuchName"
SNMP	snmpOutGenErrs	counter32	total de mensagens enviadas com erros "genErr"

QUADRO I.2 – Mapeamento da Gerência de Falhas da MIB-II.

ANEXO I

Gerência de Desempenho			
Grupo	**Variável**	**Tipo**	**Informação**
Interfaces	ifInDiscards	counter32	taxa de descartes de entrada
Interfaces	ifOutDiscards	counter32	taxa de descartes de saída
Interfaces	ifInErrors	counter32	taxa de erros de entrada
Interfaces	ifOutErrors	counter32	taxa de erros de saida
Interfaces	ifInOctets	counter32	taxa de bytes recebidos
Interfaces	ifOutOctets	counter32	taxa de bytes enviados
Interfaces	ifInUcastPkts	counter32	taxa de pacotes unicast recebidos
Interfaces	ifOutUcastPkts	counter32	taxa de pacotes unicast enviados
Interfaces	ifInNUcastPkts	counter32	taxa de pacotes no-unicast recebidos
Interfaces	ifOutNUcastPkts	counter32	Taxa de pacotes no-unicast enviados
Interfaces	ifInUnknownProtos	counter32	taxa de pacotes de protocolos desconhecidos recebidos
Interfaces	ifOutQLen	unsigned32	total de pacotes na fila de saída
IP	ipInReceives	counter32	taxa de datagramas recebidos
IP	ipInHdrErrors	counter32	taxa de erros de cabeçalho de entrada
IP	ipInAddrErrors	counter32	taxa de erros de endereco de entrada
IP	ipForwDatagrams	counter32	taxa de datagramas repassados
IP	ipInUnknownProtos	counter32	taxa de datagramas de entrada para um protocolo desconhecido
IP	ipInDiscards	counter32	taxa de datagramas de entrada descartados
IP	ipInDelivers	counter32	taxa de datagramas de entrada entregues com sucesso
IP	ipOutRequests	counter32	taxa de datagramas de saída (não inclui os datagramas repassados)
IP	ipOutDiscards	counter32	taxa de datagramas de saida descartados
IP	ipOutNoRoutes	counter32	taxa de descartes ocorridos por falta de informação de roteamento
IP	ipRoutingDiscards	counter32	taxa de entradas de roteamento descartadas
IP	ipReasmReqds	counter32	taxa de datagramas recebidos necessitando de remontagem
IP	ipReasmOKs	counter32	taxa de datagramas com sucesso na remontagem
IP	ipReasmFails	counter32	taxa de datagramas com falhas na remontagem
IP	ipFragOKs	counter32	taxa de datagramas com sucesso na fragmentação
IP	ipFragCreates	counter32	taxa de fragmentos gerados
ICMP	icmpInMsgs	counter32	taxa de recebimento de mensagens
ICMP	icmpInErrors	counter32	taxa de erros de entrada
ICMP	icmpInDestUnreachs	counter32	taxa de mensagens de Destino não Alcançado recebidas
ICMP	icmpInTimeExcds	counter32	taxa de mensagens de Tempo Excedido recebidas

ANEXO I

Gerência de Desempenho			
Grupo	**Variável**	**Tipo**	**Informação**
ICMP	icmpInParmProbs	counter32	taxa de mensagens de Problemas com Parâmetros recebidas
ICMP	icmpInSrcQuenchs	counter32	taxa de mensagens de Fonte Apagada recebidas
ICMP	icmpRedirects	counter32	taxa de mensagens de redirecionamento recebidas
ICMP	icmpInEchos	counter32	taxa de mensagens de Ecos (requisição) recebidas
ICMP	icmpInEchoReps	counter32	taxa de mensagens de Respostas a Eco recebidas
ICMP	icmpInTimestamps	counter32	taxa de mensagens de requisição de timestamp recebidas
ICMP	icmpInTimestampReps	counter32	taxa de mensagens de respostas ao pedido de timestamps recebidas
ICMP	icmpInAddrMasks	counter32	taxa de mensagens de Requisição de Máscaras de Endereços recebidas
ICMP	icmpInAddrMaskReps	counter32	taxa de mensagens de Resposta a Máscaras de Endereços recebidas
ICMP	icmpOutMsgs	counter32	taxa de saída de mensagens
ICMP	icmpOutErrors	counter32	taxa de erros de saida
ICMP	icmpOutDestUnreachs	counter32	taxa de mensagens de Destino não Alcançado enviadas
ICMP	icmpOutTimeExcds	counter32	taxa de mensagens de Tempo Excedido enviadas
ICMP	icmpOutParmProbs	counter32	taxa de mensagens de Problema com Parâmetros enviadas
ICMP	icmpOutSrcQuenchs	counter32	taxa de mensagens de Origem Apagada enviadas
ICMP	icmpOutRedirects	counter32	taxa de mensagens de re-direcionamento enviadas
ICMP	icmpOutEchos	counter32	taxa de mensagens de Eco (requisição) enviadas
ICMP	icmpOutEchoReps	counter32	taxa de mensagens Respostas de Eco enviadas
ICMP	icmpOutTimestamps	counter32	taxa de mensagens de requisição de Timestamp requisição enviadas
ICMP	icmpOutTimestampsReps	counter32	taxa de mensagens de respostas ao pedido de Timestamp enviadas
ICMP	icmpOutAddrMasks	counter32	taxa de mensagens de Requisição de Máscara de Endereços enviadas
ICMP	icmpOutAddrMaskReps	counter32	taxa de mensagens de Resposta de Máscara de endereços enviadas
TCP	tcpAttemptFails	counter32	número de tentativas de conexões falhadas
TCP	tcpEstabResets	counter32	número de reinicializações de conexões estabelecidas
TCP	tcpRetransSegs	counter32	número de segmentos retransmitidos
TCP	tcpInErrs	counter32	número de pacotes recebidos
TCP	tcpOutRsts	counter32	número de vezes que a entidade tentou reinicializar uma conexão

ANEXO I

Gerência de Desempenho			
Grupo	**Variável**	**Tipo**	**Informação**
TCP	tcpInSegs	counter32	taxa de segmentos TCP recebidos
TCP	tcpOutSegs	counter32	taxa de segmentos TCP enviados
UDP	udpInDatagrams	counter32	taxa de datagramas recebidos
UDP	udpOutDatagrams	counter32	taxa de datagramas enviados
UDP	udpNoPorts	counter32	taxa de datagramas enviados
UDP	udpInErrors	counter32	taxa de datagramas UDP recebidos com erro
EGP	egpInMsgs	counter32	taxa de mensagens recebidas
EGP	egpInErrors	counter32	taxa de mensagens recebidas com erro
EGP	egpOutMsgs	counter32	taxa de mensagens enviadas
EGP	egpOutErrors	counter32	taxa de mensagens não enviadas devido a ocorrência de erros
EGP	egpNeighInMsgs	counter32	taxa de mensagens recebidas deste vizinho EGP
EGP	egpNeighInErrs	counter32	taxa de mensagens recebidas com erro deste vizinho EGP
EGP	egpNeighOutMsgs	counter32	taxa de mensagens enviadas a este vizinho EGP
EGP	egpNeighOutErrs	counter32	taxa de mensagens não enviadas e este vizinho EGP
EGP	egpNeighInErrMsgs	counter32	taxa de mensagens de erro recebidas deste vizinho EGP
EGP	egpNeighOutErrMsgs	counter32	taxa de mensagens de erro enviadas a este vizinho EGP
SNMP	snmpInPkts	counter32	taxa de pacotes SNMP recebidos
SNMP	snmpOutPkts	counter32	taxa de pacotes SNMP enviados
SNMP	snmpInTotalReqVars	counter32	taxa de Get/Get-Next-Requests enviados
SNMP	snmpInTotalSetVars	counter32	taxa de Set-Requests recebidas
SNMP	snmpInGetResquests	counter32	taxa de Get-Requests recebidas
SNMP	snmpInGetNexts	counter32	taxa de Get-Next-Requests recebidas
SNMP	snmpInSetRequests	counter32	taxa de Set-Requests recebidas
SNMP	snmpInGetResponses	counter32	taxa de Get-Responses recebidas
SNMP	snmpInTraps	counter32	taxa de Traps recebidas
SNMP	snmpOutGetRequests	counter32	taxa de Get-Requests enviadas
SNMP	snmpOutGetNexts	counter32	taxa de Get-Next-Reqests enviadas
SNMP	snmpOutSetRequests	counter32	taxa de Set-Requests enviadas
SNMP	snmpOutResponses	counter32	taxa de Get-Responses enviadas
SNMP	snmpOutTraps	counter32	taxa de Traps enviadas

QUADRO I.3 – Mapeamento da Gerência de Desempenho da MIB-II.

ANEXO I

Gerência de Contabilização			
Grupo	**Variável**	**Tipo**	**Informação**
Interfaces	ifInOctets	counter32	taxa de bytes recebidos
Interfaces	ifOutOctets	counter32	taxa de bytes enviados
Interfaces	ifInUcastPkts	counter32	taxa de pacotes unicast recebidos
Interfaces	ifOutUcastPkts	counter32	taxa de pacotes unicast enviados
Interfaces	ifInNUcastPkts	counter32	taxa de pacotes no-unicast recebidos
Interfaces	ifOutNUcastPkts	counter32	Taxa de pacotes no-unicast enviados
IP	ipOutRequests	counter32	taxa de datagramas de saida (não inclui os datagramas repassados)
IP	ipInReceives	counter32	taxa de datagramas de recebidos
IP	ipInDelivers	counter32	taxa de datagramas de entrada entregues com sucesso
TCP	tcpActiveOpens	counter32	número de vezes que o sistema abriu uma conexão
TCP	tcpPassiveOpens	counter32	número de vezes que o sistema recebeu um pedio de abertura de conexão
TCP	tcpInSegs	counter32	número total de segmentos TCP recebidos
TCP	tcpOutSegs	counter32	número total de segmentos TCP emitidos
TCP	tcpConnState	Integer	estado da conexão
TCP	tcpConnLocalAddress	ipAddress	endereço TCP local
TCP	tcpConnLocalPort	Integer	endereço IP local
TCP	tcpConnRemAddress	ipAddress	endereço TCP remoto
TCP	tcpConnRemPort	Integer	endereço IP remoto
UDP	udpDatagrams	counter32	número total de datagramas UDP recebidos
UDP	udpOutDatagrams	counter32	número total de datagramas UDP enviados
UDP	udpLocalAddress	ipAddress	endereço IP local
UDP	udpLocalPort	Integer	porta UDP local
SNMP	snmpInPkts	counter32	taxa de pacotes SNMP recebidos
SNMP	snmpOutPkts	counter32	taxa de pacotes SNMP enviados
SNMP	snmpInTraps	counter32	taxa de traps recebidas
SNMP	snmpOutTraps	counter32	taxa de traps enviadas

QUADRO I.4 – Mapeamento da Gerência de Contabilização da MIB-II.

ANEXO I

Gerência de Segurança			
Grupo	**Variável**	**Tipo**	**Informação**
UDP	udpLocalAddress	ipAddress	endereço IP local
UDP	udpLocalPort	Integer	porta UDP local
SNMP	snmpInBadCommunityNames	counter32	total de pacotes com uma community string incorreta
SNMP	snmpInBadCommunityUses	counter32	total de pacotes com community string que não permite a operação requisitada

QUADRO I.5 – Mapeamento da Gerência de Segurança da MIB-II

ANEXO II

```
### -*- mode: Perl -*-
########################################################################
### BER (Basic Encoding Rules) encoding and decoding.
########################################################################

### Encoding

package BER;

require 5.002;

use strict;
use vars qw(@ISA @EXPORT $VERSION $pretty_print_timeticks
            %pretty_printer %default_printer $errmsg);
use Exporter;

$VERSION = '1.05';

@ISA = qw(Exporter);

@EXPORT = qw(context_flag constructor_flag
             encode_int encode_int_0 encode_null encode_oid
             encode_sequence encode_tagged_sequence
             encode_string encode_ip_address encode_timeticks
             encode_uinteger32 encode_counter32 encode_counter64
             encode_gauge32
             decode_sequence decode_by_template
             pretty_print pretty_print_timeticks
             hex_string hex_string_of_type
             encoded_oid_prefix_p errmsg
             register_pretty_printer unregister_pretty_printer);


$pretty_print_timeticks = 1;

sub encode_header ($$);
sub encode_int_0 ();
sub encode_int ($);
sub encode_oid (@);
sub encode_null ();
sub encode_sequence (@);
sub encode_tagged_sequence ($@);
sub encode_string ($);
sub encode_ip_address ($);
sub encode_timeticks ($);
sub pretty_print ($);
sub pretty_using_decoder ($$);
sub pretty_string ($);
sub pretty_intlike ($);
sub pretty_unsignedlike ($);
sub pretty_oid ($);
sub pretty_uptime ($);
sub pretty_uptime_value ($);
sub pretty_ip_address ($);
sub pretty_generic_sequence ($);
sub register_pretty_printer ($);
sub unregister_pretty_printer ($);
sub hex_string ($);
sub hex_string_of_type ($$);
sub decode_oid ($);
sub decode_by_template;
sub decode_by_template_2;
sub decode_sequence ($);
sub decode_int ($);
```

ANEXO II

```
sub decode_intlike ($);
sub decode_unsignedlike ($);
sub decode_intlike_s ($$);
sub decode_string ($);
sub decode_length ($@);
sub encoded_oid_prefix_p ($$);
sub decode_subid ($$$);
sub decode_generic_tlv ($);
sub error (@);
sub template_error ($$$);

sub version () { $VERSION; }


sub universal_flag      { 0x00 }
sub application_flag    { 0x40 }
sub context_flag        { 0x80 }
sub private_flag        { 0xc0 }

sub primitive_flag      { 0x00 }
sub constructor_flag    { 0x20 }


sub boolean_tag                 { 0x01 }
sub int_tag             { 0x02 }
sub bit_string_tag      { 0x03 }
sub octet_string_tag    { 0x04 }
sub null_tag                    { 0x05 }
sub object_id_tag       { 0x06 }
sub sequence_tag        { 0x10 }
sub set_tag                     { 0x11 }
sub uptime_tag                  { 0x43 }


sub long_length                 { 0x80 }


sub snmp_ip_address_tag                 { 0x00 | application_flag () }
sub snmp_counter32_tag                  { 0x01 | application_flag () }
sub snmp_gauge32_tag                    { 0x02 | application_flag () }
sub snmp_timeticks_tag                  { 0x03 | application_flag () }
sub snmp_opaque_tag                     { 0x04 | application_flag () }
sub snmp_nsap_address_tag               { 0x05 | application_flag () }
sub snmp_counter64_tag                  { 0x06 | application_flag () }
sub snmp_uinteger32_tag                 { 0x07 | application_flag () }

sub snmp_nosuchobject                   { context_flag () | 0x00 }
sub snmp_nosuchinstance                 { context_flag () | 0x01 }
sub snmp_endofmibview                   { context_flag () | 0x02 }


BEGIN {
   $default_printer{int_tag()}            = \&pretty_intlike;
   $default_printer{snmp_counter32_tag()}  = \&pretty_unsignedlike;
   $default_printer{snmp_gauge32_tag()}    = \&pretty_unsignedlike;
   $default_printer{snmp_counter64_tag()}  = \&pretty_unsignedlike;
   $default_printer{snmp_uinteger32_tag()} = \&pretty_unsignedlike;
   $default_printer{octet_string_tag()}    = \&pretty_string;
   $default_printer{object_id_tag()}       = \&pretty_oid;
   $default_printer{snmp_ip_address_tag()} = \&pretty_ip_address;

   %pretty_printer = %default_printer;
}


sub encode_header ($$) {
   my ($type,$length) = @_;
```

ANEXO II

```
    return pack ("C C", $type, $length) if $length < 128;
    return pack ("C C C", $type, long_length | 1, $length) if $length < 256;
    return pack ("C C n", $type, long_length | 2, $length) if $length < 65536;
    return error ("Cannot encode length $length yet");
}

sub encode_int_0 () {
    return pack ("C C C", 2, 1, 0);
}

sub encode_int ($) {
    return encode_intlike ($_[0], int_tag);
}

sub encode_uinteger32 ($) {
    return encode_intlike ($_[0], snmp_uinteger32_tag);
}

sub encode_counter32 ($) {
    return encode_intlike ($_[0], snmp_counter32_tag);
}

sub encode_counter64 ($) {
    return encode_intlike ($_[0], snmp_counter64_tag);
}

sub encode_gauge32 ($) {
    return encode_intlike ($_[0], snmp_gauge32_tag);
}

sub encode_intlike ($$) {
    my ($int, $tag)=@_;
    my ($sign, $val, @vals);
    $sign = ($int >= 0) ? 0 : 0xff;
    if (ref $int && $int->isa ("Math::BigInt")) {
            for(;;) {
                $val = $int->copy()->bmod (256);
                unshift(@vals, $val);
                return encode_header ($tag, $#vals + 1).pack ("C*", @vals)
                        if ($int >= -128 && $int < 128);
                $int->bsub ($sign)->bdiv (256);
            }
    } else {
            for(;;) {
                $val = $int & 0xff;
                unshift(@vals, $val);
                return encode_header ($tag, $#vals + 1).pack ("C*", @vals)
                        if ($int >= -128 && $int < 128);
                $int -= $sign, $int = int($int / 256);
            }
    }
}

sub encode_oid (@) {
    my @oid = @_;
    my ($result,$subid);

    $result = '';
    shift @oid if $oid[0] eq '';

    return error ("Object ID too short: ", join('.',@oid))
            if $#oid < 1;
    return error ("first subid too big in Object ID ", join('.',@oid))
            if $oid[0] > 2;
    $result = shift (@oid) * 40;
    $result += shift @oid;
    return error ("second subid too big in Object ID ", join('.',@oid))
```

ANEXO II

```
            if $result > 255;
    $result = pack ("C", $result);
    foreach $subid (@oid) {
            if ( ($subid>=0) && ($subid<128) ){ #7 bits long subid
               $result .= pack ("C", $subid);
            } elsif ( ($subid>=128) && ($subid<16384) ){ #14 bits long subid
               $result .= pack ("CC", 0x80 | $subid >> 7, $subid & 0x7f);
            }
            elsif ( ($subid>=16384) && ($subid<2097152) ) {#21 bits long subid
               $result .= pack ("CCC",
                                    0x80 | (($subid>>14) & 0x7f),
                                    0x80 | (($subid>>7) & 0x7f),
                                    $subid & 0x7f);
            } elsif ( ($subid>=2097152) && ($subid<268435456) ){ #28 bits long subid
               $result .= pack ("CCCC",
                                    0x80 | (($subid>>21) & 0x7f),
                                    0x80 | (($subid>>14) & 0x7f),
                                    0x80 | (($subid>>7) & 0x7f),
                                    $subid & 0x7f);
            } elsif ( ($subid>=268435456) && ($subid<4294967296) ){ #32 bits long subid
               $result .= pack ("CCCCC",
                                    0x80 | (($subid>>28) & 0x0f), #mask the bits beyond 32
                                    0x80 | (($subid>>21) & 0x7f),
                                    0x80 | (($subid>>14) & 0x7f),
                                    0x80 | (($subid>>7) & 0x7f),
                                    $subid & 0x7f);
            } else {
               return error ("Cannot encode subid $subid");
            }
    }
    encode_header (object_id_tag, length $result).$result;
}

sub encode_null () { encode_header (null_tag, 0); }
sub encode_sequence (@) { encode_tagged_sequence (sequence_tag, @_); }

sub encode_tagged_sequence ($@) {
    my ($tag,$result);

    $tag = shift @_;
    $result = join '',@_;
    return encode_header ($tag | constructor_flag, length $result).$result;
}

sub encode_string ($) {
    my ($string)=@_;
    return encode_header (octet_string_tag, length $string).$string;
}

sub encode_ip_address ($) {
    my ($addr)=@_;
    my @octets;

    if (length $addr == 4) {
      return encode_header (snmp_ip_address_tag, length $addr).$addr;
    } elsif (@octets = ($addr =~ /^([0-9]+)\.([0-9]+)\.([0-9]+)\.([0-9]+)$/)) {
      return encode_ip_address (pack ("CCCC", @octets));
    } else {
      return error ("IP address must be four bytes long or a dotted-quad");
    }
}

sub encode_timeticks ($) {
  my ($tt) = @_;
  return encode_intlike ($tt, snmp_timeticks_tag);
}
```

ANEXO II

```
#### Decoding

sub pretty_print ($) {
   my ($packet) = @_;
   return undef unless defined $packet;
   my $result = ord (substr ($packet, 0, 1));
   if (exists ($pretty_printer{$result})) {
            my $c_ref = $pretty_printer{$result};
            return &$c_ref ($packet);
   }
   return ($pretty_print_timeticks
               ? pretty_uptime ($packet)
               : pretty_unsignedlike ($packet))
            if $result == uptime_tag;
   return "(null)" if $result == null_tag;
   return error ("Exception code: noSuchObject") if $result == snmp_nosuchobject;
   return error ("Exception code: noSuchInstance") if $result == snmp_nosuchinstance;
   return error ("Exception code: endOfMibView") if $result == snmp_endofmibview;

   # IlvJa
   # pretty print sequences and their contents.

   my $ctx_cons_flags = context_flag | constructor_flag;

   if($result == (&constructor_flag | &sequence_tag) # sequence
                        || $result == (0 | $ctx_cons_flags) #get_request
                        || $result == (1 | $ctx_cons_flags) #getnext_request
                        || $result == (2 | $ctx_cons_flags) #get_response
                        || $result == (3 | $ctx_cons_flags) #set_request
                        || $result == (4 | $ctx_cons_flags) #trap_request
                        || $result == (5 | $ctx_cons_flags) #getbulk_request
                        || $result == (6 | $ctx_cons_flags) #inform_request
                        || $result == (7 | $ctx_cons_flags) #trap2_request
                        )
   {
            my $pretty_result = pretty_generic_sequence($packet);
            $pretty_result =~ s/^/    /gm; #Indent.

            my $seq_type_desc =
            {
               (constructor_flag | sequence_tag) => "Sequence",
               (0 | $ctx_cons_flags)          => "GetRequest",
               (1 | $ctx_cons_flags)          => "GetNextRequest",
               (2 | $ctx_cons_flags)          => "GetResponse",
               (3 | $ctx_cons_flags)          => "SetRequest",
               (4 | $ctx_cons_flags)          => "TrapRequest",
               (5 | $ctx_cons_flags)          => "GetbulkRequest",
               (6 | $ctx_cons_flags)          => "InformRequest",
               (7 | $ctx_cons_flags)          => "Trap2Request",
            }->{($result)};

            return $seq_type_desc . "{\n" . $pretty_result . "\n}";
   }

   return sprintf ("#<unprintable BER type 0x%x>", $result);
}

sub pretty_using_decoder ($$) {
   my ($decoder, $packet) = @_;
   my ($decoded,$rest);
   ($decoded,$rest) = &$decoder ($packet);
   return error ("Junk after object") unless $rest eq '';
   return $decoded;
}

sub pretty_string ($) {
   pretty_using_decoder (\&decode_string, $_[0]);
```

ANEXO II

```
}

sub pretty_intlike ($) {
   my $decoded = pretty_using_decoder (\&decode_intlike, $_[0]);
   $decoded;
}

sub pretty_unsignedlike ($) {
   return pretty_using_decoder (\&decode_unsignedlike, $_[0]);
}

sub pretty_oid ($) {
   my ($oid) = shift;
   my ($result,$subid,$next);
   my (@oid);
   $result = ord (substr ($oid, 0, 1));
   return error ("Object ID expected") unless $result == object_id_tag;
   ($result, $oid) = decode_length ($oid, 1);
   return error ("inconsistent length in OID") unless $result == length $oid;
   @oid = ();
   $subid = ord (substr ($oid, 0, 1));
   push @oid, int ($subid / 40);
   push @oid, $subid % 40;
   $oid = substr ($oid, 1);
   while ($oid ne '') {
            $subid = ord (substr ($oid, 0, 1));
            if ($subid < 128) {
               $oid = substr ($oid, 1);
               push @oid, $subid;
            } else {
               $next = $subid;
               $subid = 0;
               while ($next >= 128) {
                        $subid = ($subid << 7) + ($next & 0x7f);
                        $oid = substr ($oid, 1);
                        $next = ord (substr ($oid, 0, 1));
               }
               $subid = ($subid << 7) + $next;
               $oid = substr ($oid, 1);
               push @oid, $subid;
            }
   }
   join ('.', @oid);
}

sub pretty_uptime ($) {
   my ($packet,$uptime);

   ($uptime,$packet) = &decode_unsignedlike (@_);
   pretty_uptime_value ($uptime);
}

sub pretty_uptime_value ($) {
   my ($uptime) = @_;
   my ($seconds,$minutes,$hours,$days,$result);
   $uptime = int ($uptime / 100);

   $days = int ($uptime / (60 * 60 * 24));
   $uptime %= (60 * 60 * 24);

   $hours = int ($uptime / (60 * 60));
   $uptime %= (60 * 60);

   $minutes = int ($uptime / 60);
   $seconds = $uptime % 60;

   if ($days == 0){
```

ANEXO II

```
            $result = sprintf ("%d:%02d:%02d", $hours, $minutes, $seconds);
    } elsif ($days == 1) {
            $result = sprintf ("%d day, %d:%02d:%02d",
                                    $days, $hours, $minutes, $seconds);
    } else {
            $result = sprintf ("%d days, %d:%02d:%02d",
                                    $days, $hours, $minutes, $seconds);
    }
    return $result;
}


sub pretty_ip_address ($) {
    my $pdu = shift;
    my ($length, $rest);
    return error ("IP Address tag (".snmp_ip_address_tag.") expected")
            unless ord (substr ($pdu, 0, 1)) == snmp_ip_address_tag;
    ($length,$pdu) = decode_length ($pdu, 1);
    return error ("Length of IP address should be four")
            unless $length == 4;
    sprintf "%d.%d.%d.%d", unpack ("CCCC", $pdu);
}

# IlvJa
# Returns a string with the pretty prints of all
# the elements in the sequence.
sub pretty_generic_sequence ($) {
    my ($pdu) = shift;

    my $rest;

    my $type = ord substr ($pdu, 0 ,1);
    my $flags = context_flag | constructor_flag;

    return error (sprintf ("Tag 0x%x is not a valid sequence tag",$type))
            unless ($type == (&constructor_flag | &sequence_tag) # sequence
                    || $type == (0 | $flags) #get_request
                    || $type == (1 | $flags) #getnext_request
                    || $type == (2 | $flags) #get_response
                    || $type == (3 | $flags) #set_request
                    || $type == (4 | $flags) #trap_request
                    || $type == (5 | $flags) #getbulk_request
                    || $type == (6 | $flags) #inform_request
                    || $type == (7 | $flags) #trap2_request
                    );

    my $curelem;
    my $pretty_result; # Holds the pretty printed sequence.
    my $pretty_elem;   # Holds the pretty printed current elem.
    my $first_elem = 'true';

    # Cut away the first Tag and Length from $packet and then
    # init $rest with that.
    (undef, $rest) = decode_length ($pdu, 1);
    while($rest)
    {
            ($curelem,$rest) = decode_generic_tlv($rest);
            $pretty_elem = pretty_print($curelem);

            $pretty_result .= "\n" if not $first_elem;
            $pretty_result .= $pretty_elem;

            # The rest of the iterations are not related to the
            # first element of the sequence so..
            $first_elem = '' if $first_elem;
    }
    return $pretty_result;
```

ANEXO II

```
}

sub hex_string ($) {
   &hex_string_of_type ($_[0], octet_string_tag);
}

sub hex_string_of_type ($$) {
   my ($pdu, $wanted_type) = @_;
   my ($length);
   return error ("BER tag ".$wanted_type." expected")
            unless ord (substr ($pdu, 0, 1)) == $wanted_type;
   ($length,$pdu) = decode_length ($pdu, 1);
   hex_string_aux ($pdu);
}

sub hex_string_aux ($) {
   my ($binary_string) = @_;
   my ($c, $result);
   $result = '';
   for $c (unpack "C*", $binary_string) {
            $result .= sprintf "%02x", $c;
   }
   $result;
}

sub decode_oid ($) {
   my ($pdu) = @_;
   my ($result,$pdu_rest);
   my (@result);
   $result = ord (substr ($pdu, 0, 1));
   return error ("Object ID expected") unless $result == object_id_tag;
   ($result, $pdu_rest) = decode_length ($pdu, 1);
   return error ("Short PDU")
            if $result > length $pdu_rest;
   @result = (substr ($pdu, 0, $result + (length ($pdu) - length ($pdu_rest))),
               substr ($pdu_rest, $result));
   @result;
}

# IlvJa
# This takes a PDU and returns a two element list consisting of
# the first element found in the PDU (whatever it is) and the
# rest of the PDU
sub decode_generic_tlv ($) {
   my ($pdu) = @_;
   my (@result);
   my ($elemlength,$pdu_rest) = decode_length ($pdu, 1);
   @result = (# Extract the first element.
               substr ($pdu, 0, $elemlength + (length ($pdu)
                                                - length ($pdu_rest)
                                                )
                        ),
               #Extract the rest of the PDU.
               substr ($pdu_rest, $elemlength)
               );
   @result;
}

sub decode_by_template {
   my ($pdu) = shift;
   local ($_) = shift;
   return decode_by_template_2 ($pdu, $_, 0, 0, @_);
}

my $template_debug = 0;

sub decode_by_template_2 {
```

ANEXO II

```
my ($pdu, $template, $pdu_index, $template_index);
local ($_);
$pdu = shift;
$template = $_ = shift;
$pdu_index = shift;
$template_index = shift;
my (@results);
my ($length,$expected,$read,$rest);
return undef unless defined $pdu;
while (0 < length ($_)) {
        if (substr ($_, 0, 1) eq '%') {
            print STDERR "template $_ ", length $pdu," bytes remaining\n"
                    if $template_debug;
            $_ = substr ($_,1);
            ++$template_index;
            if (($expected) = /^(\d*|\*)\{(.*)/) {
                    ## %{
                    $template_index += length ($expected) + 1;
                    print STDERR "%{\n" if $template_debug;
                    $_ = $2;
                    $expected = shift | constructor_flag if ($expected eq '*');
                    $expected = sequence_tag | constructor_flag
                       if $expected eq '';
                    return template_error ("Unexpected end of PDU",
                                           $template, $template_index)
                       if !defined $pdu or $pdu eq '';
                    return template_error ("Expected sequence tag $expected, got ".
                                            ord (substr ($pdu, 0, 1)),
                                           $template,
                                           $template_index)
                       unless (ord (substr ($pdu, 0, 1)) == $expected);
                    (($length,$pdu) = decode_length ($pdu, 1))
                       || return template_error ("cannot read length",
                                                 $template, $template_index);
                    return template_error ("Expected length $length, got ".length $pdu ,
                                           $template, $template_index)
                      unless length $pdu == $length;
            } elsif (($expected,$rest) = /^(\*|)s(.*)/) {
                    ## %s
                    $template_index += length ($expected) + 1;
                    ($expected = shift) if $expected eq '*';
                    (($read,$pdu) = decode_string ($pdu))
                       || return template_error ("cannot read string",
                                                 $template, $template_index);
                    print STDERR "%s => $read\n" if $template_debug;
                    if ($expected eq '') {
                       push @results, $read;
                    } else {
                       return template_error ("Expected $expected, read $read",
                                              $template, $template_index)
                              unless $expected eq $read;
                    }
                    $_ = $rest;
            } elsif (($rest) = /^A(.*)/) {
                    ## %A
                    $template_index += 1;
                    {
                       my ($tag, $length, $value);
                       $tag = ord (substr ($pdu, 0, 1));
                       return error ("Expected IP address, got tag ".$tag)
                              unless $tag == snmp_ip_address_tag;
                       ($length, $pdu) = decode_length ($pdu, 1);
                       return error ("Inconsistent length of InetAddress encoding")
                              if $length > length $pdu;
                       return template_error ("IP address must be four bytes long",
                                              $template, $template_index)
                              unless $length == 4;
```

ANEXO II

```
                $read = substr ($pdu, 0, $length);
                $pdu = substr ($pdu, $length);
              }
              print STDERR "%A => $read\n" if $template_debug;
              push @results, $read;
              $_ = $rest;
        } elsif (/^O(.*)/) {
              ## %O
              $template_index += 1;
              $_ = $1;
              (($read,$pdu) = decode_oid ($pdu))
                || return template_error ("cannot read OID",
                                          $template, $template_index);
              print STDERR "%O => ".pretty_oid ($read)."\n"
                if $template_debug;
              push @results, $read;
        } elsif (($expected,$rest) = /^(\d*|\*|)i(.*)/) {
              ## %i
              $template_index += length ($expected) + 1;
              print STDERR "%i\n" if $template_debug;
              $_ = $rest;
              (($read,$pdu) = decode_int ($pdu))
                || return template_error ("cannot read int",
                                          $template, $template_index);
              if ($expected eq '') {
                push @results, $read;
              } else {
                $expected = int (shift) if $expected eq '*';
                return template_error (sprintf ("Expected %d (0x%x), got %d (0x%x)",
                                                $expected, $expected, $read, $read),
                                       $template, $template_index)
                        unless ($expected == $read)
              }
        } elsif (($rest) = /^u(.*)/) {
              ## %u
              $template_index += 1;
              print STDERR "%u\n" if $template_debug;
              $_ = $rest;
              (($read,$pdu) = decode_unsignedlike ($pdu))
                || return template_error ("cannot read uptime",
                                          $template, $template_index);
              push @results, $read;
        } elsif (/^\@(.*)/) {
              ## %@
              $template_index += 1;
              print STDERR "%@\n" if $template_debug;
              $_ = $1;
              push @results, $pdu;
              $pdu = '';
        } else {
              return template_error ("Unknown decoding directive in template: $_",
                                     $template, $template_index);
        }
      } else {
        if (substr ($_, 0, 1) ne substr ($pdu, 0, 1)) {
              return template_error ("Expected ".substr ($_, 0, 1).", got ".substr ($pdu, 0, 1),
                                     $template, $template_index);
        }
        $_ = substr ($_,1);
        $pdu = substr ($pdu,1);
      }
}
return template_error ("PDU too long", $template, $template_index)
  if length ($pdu) > 0;
return template_error ("PDU too short", $template, $template_index)
  if length ($_) > 0;
@results;
```

ANEXO II

```
}

sub decode_sequence ($) {
   my ($pdu) = @_;
   my ($result);
   my (@result);
   $result = ord (substr ($pdu, 0, 1));
   return error ("Sequence expected")
            unless $result == (sequence_tag | constructor_flag);
   ($result, $pdu) = decode_length ($pdu, 1);
   return error ("Short PDU")
            if $result > length $pdu;
   @result = (substr ($pdu, 0, $result), substr ($pdu, $result));
   @result;
}

sub decode_int ($) {
   my ($pdu) = @_;
   my $tag = ord (substr ($pdu, 0, 1));
   return error ("Integer expected, found tag ".$tag)
            unless $tag == int_tag;
   decode_intlike ($pdu);
}

sub decode_intlike ($) {
   decode_intlike_s ($_[0], 1);
}

sub decode_unsignedlike ($) {
   decode_intlike_s ($_[0], 0);
}

my $have_math_bigint_p = 0;

sub decode_intlike_s ($$) {
   my ($pdu, $signedp) = @_;
   my ($length,$result);
   ($length,$pdu) = decode_length ($pdu, 1);
   my $ptr = 0;
   $result = unpack ($signedp ? "c" : "C", substr ($pdu, $ptr++, 1));
   if ($length > 5 || ($length == 5 && $result > 0)) {
            require 'Math/BigInt.pm' unless $have_math_bigint_p++;
            $result = new Math::BigInt ($result);
   }
   while (--$length > 0) {
            $result *= 256;
            $result += unpack ("C", substr ($pdu, $ptr++, 1));
   }
   ($result, substr ($pdu, $ptr));
}

sub decode_string ($) {
   my ($pdu) = shift;
   my ($result);
   $result = ord (substr ($pdu, 0, 1));
   return error ("Expected octet string, got tag ".$result)
            unless $result == octet_string_tag;
   ($result, $pdu) = decode_length ($pdu, 1);
   return error ("Short PDU")
            if $result > length $pdu;
   return (substr ($pdu, 0, $result), substr ($pdu, $result));
}

sub decode_length ($@) {
   my ($pdu) = shift;
   my $index = shift || 0;
   my ($result);
```

ANEXO II

```
    my (@result);
    $result = ord (substr ($pdu, $index, 1));
    if ($result & long_length) {
            if ($result == (long_length | 1)) {
               @result = (ord (substr ($pdu, $index+1, 1)), substr ($pdu, $index+2));
            } elsif ($result == (long_length | 2)) {
               @result = ((ord (substr ($pdu, $index+1, 1)) << 8)
                          + ord (substr ($pdu, $index+2, 1)), substr ($pdu, $index+3));
            } else {
               return error ("Unsupported length");
            }
    } else {
            @result = ($result, substr ($pdu, $index+1));
    }
    @result;
}

sub register_pretty_printer($)
{
    my ($h_ref) = shift;
    my ($type, $val, $cnt);

    $cnt = 0;
    while(($type, $val) = each %$h_ref) {
            if (ref $val eq "CODE") {
               $pretty_printer{$type} = $val;
               $cnt++;
            }
    }
    return($cnt);
}

sub unregister_pretty_printer($)
{
    my ($h_ref) = shift;
    my ($type, $val, $cnt);

    $cnt = 0;
    while(($type, $val) = each %$h_ref) {
            if ((exists ($pretty_printer{$type}))
               && ($pretty_printer{$type} == $val)) {
               if (exists($default_printer{$type})) {
                        $pretty_printer{$type} = $default_printer{$type};
               } else {
                        delete $pretty_printer{$type};
               }
               $cnt++;
            }
    }
    return($cnt);
}

#### OID prefix check

sub encoded_oid_prefix_p ($$) {
    my ($oid1, $oid2) = @_;
    my ($i1, $i2);
    my ($l1, $l2);
    my ($subid1, $subid2);
    return error ("OID tag expected") unless ord (substr ($oid1, 0, 1)) == object_id_tag;
    return error ("OID tag expected") unless ord (substr ($oid2, 0, 1)) == object_id_tag;
    ($l1,$oid1) = decode_length ($oid1, 1);
    ($l2,$oid2) = decode_length ($oid2, 1);
    for ($i1 = 0, $i2 = 0;
              $i1 < $l1 && $i2 < $l2;
              ++$i1, ++$i2) {
             ($subid1,$i1) = &decode_subid ($oid1, $i1, $l1);
```

ANEXO II

```
            ($subid2,$i2) = &decode_subid ($oid2, $i2, $l2);
            return 0 unless $subid1 == $subid2;
    }
    return $i2 if $i1 == $l1;
    return 0;
}

### decode_subid OID INDEX

sub decode_subid ($$$) {
    my ($oid, $i, $l) = @_;
    my $subid = 0;
    my $next;

    while (($next = ord (substr ($oid, $i, 1))) >= 128) {
            $subid = ($subid << 7) + ($next & 0x7f);
            ++$i;
            return error ("decoding object ID: short field")
               unless $i < $l;
    }
    return (($subid << 7) + $next, $i);
}

sub error (@) {
  $errmsg = join ("",@_);
  return undef;
}

sub template_error ($$$) {
  my ($errmsg, $template, $index) = @_;
  return error ($errmsg."\n  ".$template."\n  ".(' ' x $index)."^");
}

1;
```

ANEXO III

```
### -*- mode: Perl -*-
######################################################################
### SNMP Request/Response Handling
######################################################################

package SNMP_Session;

require 5.002;

use strict;
use Exporter;
use vars qw(@ISA $VERSION @EXPORT $errmsg
                $suppress_warnings
                $default_avoid_negative_request_ids
                $default_use_16bit_request_ids);
use Socket;
use BER '1.05';
use Carp;

sub map_table ($$$ );
sub map_table_4 ($$$$);
sub map_table_start_end ($$$$$$);
sub index_compare ($$);
sub oid_diff ($$);

$VERSION = '1.08';

@ISA = qw(Exporter);

@EXPORT = qw(errmsg suppress_warnings index_compare oid_diff recycle_socket ipv6available);

my $default_debug = 0;

my $default_timeout = 2.0;

my $default_retries = 5;

my $default_backoff = 1.0;

my $default_max_repetitions = 12;

$SNMP_Session::default_avoid_negative_request_ids = 0;

$SNMP_Session::default_use_16bit_request_ids = 0;

$SNMP_Session::recycle_socket = 0;

my $ipv6_addr_len;

BEGIN {
   $ipv6_addr_len = undef;
   $SNMP_Session::ipv6available = 0;

   if (eval {require Socket6;} &&
            eval {require IO::Socket::INET6; IO::Socket::INET6->VERSION("1.26");}) {
            import Socket6;
            $ipv6_addr_len = length(pack_sockaddr_in6(161, inet_pton(AF_INET6(), "::1")));
            $SNMP_Session::ipv6available = 1;
   }
}

my $the_socket;

$SNMP_Session::errmsg = '';
$SNMP_Session::suppress_warnings = 0;

sub get_request      { 0 | context_flag () };
```

ANEXO III

```
sub getnext_request  { 1 | context_flag () };
sub get_response     { 2 | context_flag () };
sub set_request      { 3 | context_flag () };
sub trap_request     { 4 | context_flag () };
sub getbulk_request  { 5 | context_flag () };
sub inform_request   { 6 | context_flag () };
sub trap2_request    { 7 | context_flag () };

sub standard_udp_port { 161 };

sub open
{
   return SNMPv1_Session::open (@_);
}

sub timeout { $_[0]->{timeout} }
sub retries { $_[0]->{retries} }
sub backoff { $_[0]->{backoff} }
sub set_timeout {
   my ($session, $timeout) = @_;
   croak ("timeout ($timeout) must be a positive number") unless $timeout > 0.0;
   $session->{'timeout'} = $timeout;
}
sub set_retries {
   my ($session, $retries) = @_;
   croak ("retries ($retries) must be a non-negative integer")
            unless $retries == int ($retries) && $retries >= 0;
   $session->{'retries'} = $retries;
}
sub set_backoff {
   my ($session, $backoff) = @_;
   croak ("backoff ($backoff) must be a number >= 1.0")
            unless $backoff == int ($backoff) && $backoff >= 1.0;
   $session->{'backoff'} = $backoff;
}

sub encode_request_3 ($$$@) {
   my($this, $reqtype, $encoded_oids_or_pairs, $i1, $i2) = @_;
   my($request);
   local($_);

   $this->{request_id} = ($this->{request_id} == 0x7fffffff)
            ? -0x80000000 : $this->{request_id}+1;
   $this->{request_id} += 0x80000000
            if ($this->{avoid_negative_request_ids} && $this->{request_id} < 0);
   $this->{request_id} &= 0x0000ffff
            if ($this->{use_16bit_request_ids});
   foreach $_ (@{$encoded_oids_or_pairs}) {
     if (ref ($_) eq 'ARRAY') {
            $_ = &encode_sequence ($_->[0], $_->[1])
              || return $this->ber_error ("encoding pair");
     } else {
            $_ = &encode_sequence ($_, encode_null())
              || return $this->ber_error ("encoding value/null pair");
     }
   }
   $request = encode_tagged_sequence
            ($reqtype,
             encode_int ($this->{request_id}),
             defined $i1 ? encode_int ($i1) : encode_int_0 (),
             defined $i2 ? encode_int ($i2) : encode_int_0 (),
             encode_sequence (@{$encoded_oids_or_pairs}))
              || return $this->ber_error ("encoding request PDU");
   return $this->wrap_request ($request);
}

sub encode_get_request {
```

ANEXO III

```
   my($this, @oids) = @_;
   return encode_request_3 ($this, get_request, \@oids);
}

sub encode_getnext_request {
   my($this, @oids) = @_;
   return encode_request_3 ($this, getnext_request, \@oids);
}

sub encode_getbulk_request {
   my($this, $non_repeaters, $max_repetitions, @oids) = @_;
   return encode_request_3 ($this, getbulk_request, \@oids,
                                          $non_repeaters, $max_repetitions);
}

sub encode_set_request {
   my($this, @encoded_pairs) = @_;
   return encode_request_3 ($this, set_request, \@encoded_pairs);
}

sub encode_trap_request ($$$$$@) {
   my($this, $ent, $agent, $gen, $spec, $dt, @pairs) = @_;
   my($request);
   local($_);

   foreach $_ (@pairs) {
     if (ref ($_) eq 'ARRAY') {
               $_ = &encode_sequence ($_->[0], $_->[1])
                 || return $this->ber_error ("encoding pair");
     } else {
               $_ = &encode_sequence ($_, encode_null())
                 || return $this->ber_error ("encoding value/null pair");
     }
   }
   $request = encode_tagged_sequence
               (trap_request, $ent, $agent, $gen, $spec, $dt, encode_sequence (@pairs))
                 || return $this->ber_error ("encoding trap PDU");
   return $this->wrap_request ($request);
}

sub encode_v2_trap_request ($@) {
   my($this, @pairs) = @_;

   return encode_request_3($this, trap2_request, \@pairs);
}

sub decode_get_response {
   my($this, $response) = @_;
   my @rest;
   @{$this->{'unwrapped'}};
}

sub decode_trap_request ($$) {
   my ($this, $trap) = @_;
   my ($snmp_version, $community, $ent, $agent, $gen, $spec, $dt,
               $request_id, $error_status, $error_index,
               $bindings);
   ($snmp_version, $community,
    $ent, $agent,
    $gen, $spec, $dt,
    $bindings)
               = decode_by_template ($trap, "%{%i%s%*{%O%A%i%i%u%{%@",
                                          trap_request);
   if (!defined $snmp_version) {
               ($snmp_version, $community,
                $request_id, $error_status, $error_index,
                $bindings)
```

ANEXO III

```
                = decode_by_template ($trap, "%{%i%s%*{%i%i%i%{%@",
                                                trap2_request);
            if (!defined $snmp_version) {
                ($snmp_version, $community,$request_id, $error_status, $error_index, $bindings)
                        = decode_by_template ($trap, "%{%i%s%*{%i%i%i%{%@", inform_request);
            }
            return $this->error_return ("v2 trap/inform request contained errorStatus/errorIndex "
                                                .$error_status."/".$error_index)
                if defined $error_status && defined $error_index
                && ($error_status != 0 || $error_index != 0);
    }
    if (!defined $snmp_version) {
            return $this->error_return ("BER error decoding trap:\n  ".$BER::errmsg);
    }
    return ($community, $ent, $agent, $gen, $spec, $dt, $bindings);
}

sub wait_for_response {
    my($this) = shift;
    my($timeout) = shift || 10.0;
    my($rin,$win,$ein) = ("","");
    my($rout,$wout,$eout);
    vec($rin,$this->sockfileno,1) = 1;
    select($rout=$rin,$wout=$win,$eout=$ein,$timeout);
}

sub get_request_response ($@) {
    my($this, @oids) = @_;
    return $this->request_response_5 ($this->encode_get_request (@oids),
                                                get_response, \@oids, 1);
}

sub set_request_response ($@) {
    my($this, @pairs) = @_;
    return $this->request_response_5 ($this->encode_set_request (@pairs),
                                                get_response, \@pairs, 1);
}

sub getnext_request_response ($@) {
    my($this,@oids) = @_;
    return $this->request_response_5 ($this->encode_getnext_request (@oids),
                                                get_response, \@oids, 1);
}

sub getbulk_request_response ($$$@) {
    my($this,$non_repeaters,$max_repetitions,@oids) = @_;
    return $this->request_response_5
            ($this->encode_getbulk_request ($non_repeaters,$max_repetitions,@oids),
             get_response, \@oids, 1);
}

sub trap_request_send ($$$$$$@) {
    my($this, $ent, $agent, $gen, $spec, $dt, @pairs) = @_;
    my($req);

    $req = $this->encode_trap_request ($ent, $agent, $gen, $spec, $dt, @pairs);
    ## Encoding may have returned an error.
    return undef unless defined $req;
    $this->send_query($req)
            || return $this->error ("send_trap: $!");
    return 1;
}

sub v2_trap_request_send ($$$@) {
    my($this, $trap_oid, $dt, @pairs) = @_;
    my @sysUptime_OID = ( 1,3,6,1,2,1,1,3 );
    my @snmpTrapOID_OID = ( 1,3,6,1,6,3,1,1,4,1 );
```

ANEXO III

```
    my($req);

    unshift @pairs, [encode_oid (@snmpTrapOID_OID,0),
                             encode_oid (@{$trap_oid})];
    unshift @pairs, [encode_oid (@sysUptime_OID,0),
                             encode_timeticks ($dt)];
    $req = $this->encode_v2_trap_request (@pairs);
    ## Encoding may have returned an error.
    return undef unless defined $req;
    $this->send_query($req)
            || return $this->error ("send_trap: $!");
    return 1;
}

sub request_response_5 ($$$$$) {
    my ($this, $req, $response_tag, $oids, $errorp) = @_;
    my $retries = $this->retries;
    my $timeout = $this->timeout;
    my ($nfound, $timeleft);

    ## Encoding may have returned an error.
    return undef unless defined $req;

    $timeleft = $timeout;
    while ($retries > 0) {
            $this->send_query ($req)
               || return $this->error ("send_query: $!");
            # IlvJa
            # Add request pdu to capture_buffer
            push @{$this->{'capture_buffer'}}, $req
               if (defined $this->{'capture_buffer'}
                        and ref $this->{'capture_buffer'} eq 'ARRAY');
            #
     wait_for_response:
            ($nfound, $timeleft) = $this->wait_for_response($timeleft);
            if ($nfound > 0) {
               my($response_length);

               $response_length
                        = $this->receive_response_3 ($response_tag, $oids, $errorp);
               if ($response_length) {
                        # IlvJa
                        # Add response pdu to capture_buffer
                        push (@{$this->{'capture_buffer'}},
                             substr($this->{'pdu_buffer'}, 0, $response_length)
                             )
                             if (defined $this->{'capture_buffer'}
                                      and ref $this->{'capture_buffer'} eq 'ARRAY');
                        #
                        return $response_length;
               } elsif (defined ($response_length)) {
                        goto wait_for_response;
                        # A response has been received, but for a different
                        # request ID or from a different IP address.
               } else {
                        return undef;
               }
            } else {
               ## No response received - retry
               --$retries;
               $timeout *= $this->backoff;
               $timeleft = $timeout;
            }
    }
    # IlvJa
    # Add empty packet to capture_buffer
    push @{$this->{'capture_buffer'}}, ""
```

ANEXO III

```
            if (defined $this->{'capture_buffer'}
               and ref $this->{'capture_buffer'} eq 'ARRAY');
   #
   $this->error ("no response received");
}

sub map_table ($$$) {
   my ($session, $columns, $mapfn) = @_;
   return $session->map_table_4 ($columns, $mapfn,
                                 $session->default_max_repetitions ());
}

sub map_table_4 ($$$$) {
   my ($session, $columns, $mapfn, $max_repetitions) = @_;
   return $session->map_table_start_end ($columns, $mapfn,
                                         "", undef,
                                         $max_repetitions);
}

sub map_table_start_end ($$$$$$) {
   my ($session, $columns, $mapfn, $start, $end, $max_repetitions) = @_;

   my @encoded_oids;
   my $call_counter = 0;
   my $base_index = $start;

   do {
            foreach (@encoded_oids = @{$columns}) {
               $_=encode_oid (@{$_},split '\.',$base_index)
                     || return $session->ber_error ("encoding OID $base_index");
            }
            if ($session->getnext_request_response (@encoded_oids)) {
               my $response = $session->pdu_buffer;
               my ($bindings) = $session->decode_get_response ($response);
               my $smallest_index = undef;
               my @collected_values = ();

               my @bases = @{$columns};
               while ($bindings ne '') {
                     my ($binding, $oid, $value);
                     my $base = shift @bases;
                     ($binding, $bindings) = decode_sequence ($bindings);
                     ($oid, $value) = decode_by_template ($binding, "%O%@");

                     my $out_index;

                     $out_index = &oid_diff ($base, $oid);
                     my $cmp;
                     if (!defined $smallest_index
                        || ($cmp = index_compare ($out_index,$smallest_index)) == -1) {
                        $smallest_index = $out_index;
                        grep ($_=undef, @collected_values);
                        push @collected_values, $value;
                     } elsif ($cmp == 1) {
                        push @collected_values, undef;
                     } else {
                        push @collected_values, $value;
                     }
               }
               (++$call_counter,
                &$mapfn ($smallest_index, @collected_values))
                     if defined $smallest_index;
               $base_index = $smallest_index;
            } else {
               return undef;
            }
   }
```

ANEXO III

```
    while (defined $base_index
                && (!defined $end || index_compare ($base_index, $end) < 0));
    $call_counter;
}

sub index_compare ($$) {
  my ($i1, $i2) = @_;
  $i1 = '' unless defined $i1;
  $i2 = '' unless defined $i2;
  if ($i1 eq '') {
     return $i2 eq '' ? 0 : 1;
  } elsif ($i2 eq '') {
     return 1;
  } elsif (!$i1) {
     return $i2 eq '' ? 1 : !$i2 ? 0 : 1;
  } elsif (!$i2) {
     return -1;
  } else {
    my ($f1,$r1) = split('\.',$i1,2);
    my ($f2,$r2) = split('\.',$i2,2);

    if ($f1 < $f2) {
      return -1;
    } elsif ($f1 > $f2) {
      return 1;
    } else {
      return index_compare ($r1,$r2);
    }
  }
}

sub oid_diff ($$) {
  my($base, $full) = @_;
  my $base_dotnot = join ('.',@{$base});
  my $full_dotnot = BER::pretty_oid ($full);

  return undef unless substr ($full_dotnot, 0, length $base_dotnot)
    eq $base_dotnot
      && substr ($full_dotnot, length $base_dotnot, 1) eq '.';
  substr ($full_dotnot, length ($base_dotnot)+1);
}

# Pretty_address returns a human-readable representation of an IPv4 or IPv6 address.
sub pretty_address {
    my($addr) = shift;
    my($port, $addrunpack, $addrstr);

    # Disable strict subs to stop old versions of perl from
    # complaining about AF_INET6 when Socket6 is not available

    if( (defined $ipv6_addr_len) && (length $addr == $ipv6_addr_len)) {
                ($port,$addrunpack) = unpack_sockaddr_in6 ($addr);
                $addrstr = inet_ntop (AF_INET6(), $addrunpack);
    } else {
                ($port,$addrunpack) = unpack_sockaddr_in ($addr);
                $addrstr = inet_ntoa ($addrunpack);
    }

    return sprintf ("[%s].%d", $addrstr, $port);
}

sub version { $VERSION; }


sub error_return ($$) {
    my ($this,$message) = @_;
    $SNMP_Session::errmsg = $message;
```

ANEXO III

```
    unless ($SNMP_Session::suppress_warnings) {
            $message =~ s/^/  /mg;
            carp ("Error:\n".$message."\n");
    }
    return undef;
}

sub error ($$) {
    my ($this,$message) = @_;
    my $session = $this->to_string;
    $SNMP_Session::errmsg = $message."\n".$session;
    unless ($SNMP_Session::suppress_warnings) {
            $session =~ s/^/  /mg;
            $message =~ s/^/  /mg;
            carp ("SNMP Error:\n".$SNMP_Session::errmsg."\n");
    }
    return undef;
}

sub ber_error ($$) {
  my ($this,$type) = @_;
  my ($errmsg) = $BER::errmsg;

  $errmsg =~ s/^/  /mg;
  return $this->error ("$type:\n$errmsg");
}

package SNMPv1_Session;

use strict qw(vars subs);          # see above
use vars qw(@ISA);
use SNMP_Session;
use Socket;
use BER;
use IO::Socket;
use Carp;

BEGIN {
    if($SNMP_Session::ipv6available) {
            import IO::Socket::INET6;
            import Socket6;
    }
}

@ISA = qw(SNMP_Session);

sub snmp_version { 0 }

# Supports both IPv4 and IPv6.
# Numeric IPv6 addresses must be passed between square brackets []
sub open {
    my($this,
       $remote_hostname,$community,$port,
       $max_pdu_len,$local_port,$max_repetitions,
       $local_hostname,$ipv4only) = @_;
    my($remote_addr,$socket,$sockfamily);

    $ipv4only = 1 unless defined $ipv4only;
    $sockfamily = AF_INET;

    $community = 'public' unless defined $community;
    $port = SNMP_Session::standard_udp_port unless defined $port;
    $max_pdu_len = 8000 unless defined $max_pdu_len;
    $max_repetitions = $default_max_repetitions
            unless defined $max_repetitions;

    if ($ipv4only || ! $SNMP_Session::ipv6available) {
```

ANEXO III

```
        # IPv4-only code, uses only Socket and INET calls
if (defined $remote_hostname) {
        $remote_addr = inet_aton ($remote_hostname)
          or return $this->error_return ("can't resolve \"$remote_hostname\" to IP address");
}
if ($SNMP_Session::recycle_socket && defined $the_socket) {
        $socket = $the_socket;
} else {
        $socket = IO::Socket::INET->new(Proto => 17,
                                        Type => SOCK_DGRAM,
                                        LocalAddr => $local_hostname,
                                        LocalPort => $local_port)
          || return $this->error_return ("creating socket: $!");
        $the_socket = $socket
          if $SNMP_Session::recycle_socket;
}
$remote_addr = pack_sockaddr_in ($port, $remote_addr)
        if defined $remote_addr;
} else {
        # IPv6-capable code. Will use IPv6 or IPv4 depending on the address.
        # Uses Socket6 and INET6 calls.

        # If it's a numeric IPv6 addresses, remove square brackets
        if ($remote_hostname =~ /^\[(.*)\]$/) {
          $remote_hostname = $1;
        }

        my (@res, $socktype_tmp, $proto_tmp, $canonname_tmp);
        @res = getaddrinfo($remote_hostname, $port, AF_UNSPEC, SOCK_DGRAM);
        ($sockfamily, $socktype_tmp, $proto_tmp, $remote_addr, $canonname_tmp) = @res;
        if (scalar(@res) < 5) {
          return $this->error_return ("can't resolve \"$remote_hostname\" to IPv6 address");
        }

        if ($SNMP_Session::recycle_socket && defined $the_socket) {
          $socket = $the_socket;
        } elsif ($sockfamily == AF_INET) {
          $socket = IO::Socket::INET->new(Proto => 17,
                                          Type => SOCK_DGRAM,
                                          LocalAddr => $local_hostname,
                                          LocalPort => $local_port)
            || return $this->error_return ("creating socket: $!");
        } else {
          $socket = IO::Socket::INET6->new(Proto => 17,
                                           Type => SOCK_DGRAM,
                                           LocalAddr => $local_hostname,
                                           LocalPort => $local_port)
            || return $this->error_return ("creating socket: $!");
          $the_socket = $socket
            if $SNMP_Session::recycle_socket;
        }
}
bless {
          'sock' => $socket,
          'sockfileno' => fileno ($socket),
          'community' => $community,
          'remote_hostname' => $remote_hostname,
          'remote_addr' => $remote_addr,
          'sockfamily' => $sockfamily,
          'max_pdu_len' => $max_pdu_len,
          'pdu_buffer' => '\0' x $max_pdu_len,
          'request_id' => (int (rand 0x10000) << 16)
            + int (rand 0x10000) - 0x80000000,
          'timeout' => $default_timeout,
          'retries' => $default_retries,
          'backoff' => $default_backoff,
          'debug' => $default_debug,
```

ANEXO III

```
                'error_status' => 0,
                'error_index' => 0,
                'default_max_repetitions' => $max_repetitions,
                'use_getbulk' => 1,
                'lenient_source_address_matching' => 1,
                'lenient_source_port_matching' => 1,
                'avoid_negative_request_ids' => $SNMP_Session::default_avoid_negative_request_ids,
                'use_16bit_request_ids' => $SNMP_Session::default_use_16bit_request_ids,
                'capture_buffer' => undef,
               };
}

sub open_trap_session (@) {
   my ($this, $port) = @_;
   $port = 162 unless defined $port;
   return $this->open (undef, "", 161, undef, $port);
}

sub sock { $_[0]->{sock} }
sub sockfileno { $_[0]->{sockfileno} }
sub remote_addr { $_[0]->{remote_addr} }
sub pdu_buffer { $_[0]->{pdu_buffer} }
sub max_pdu_len { $_[0]->{max_pdu_len} }
sub default_max_repetitions {
   defined $_[1]
            ? $_[0]->{default_max_repetitions} = $_[1]
              : $_[0]->{default_max_repetitions} }
sub debug { defined $_[1] ? $_[0]->{debug} = $_[1] : $_[0]->{debug} }

sub close {
   my($this) = shift;
   ## Avoid closing the socket if it may be shared with other session
   ## objects.
   if (! defined $the_socket || $this->sock ne $the_socket) {
            close ($this->sock) || $this->error ("close: $!");
   }
}

sub wrap_request {
   my($this) = shift;
   my($request) = shift;

   encode_sequence (encode_int ($this->snmp_version),
                    encode_string ($this->{community}),
                    $request)
     || return $this->ber_error ("wrapping up request PDU");
}

my @error_status_code = qw(noError tooBig noSuchName badValue readOnly
                           genErr noAccess wrongType wrongLength
                           wrongEncoding wrongValue noCreation
                           inconsistentValue resourceUnavailable
                           commitFailed undoFailed authorizationError
                           notWritable inconsistentName);

sub unwrap_response_5b {
   my ($this,$response,$tag,$oids,$errorp) = @_;
   my ($community,$request_id,@rest,$snmpver);

   ($snmpver,$community,$request_id,
    $this->{error_status},
    $this->{error_index},
    @rest)
            = decode_by_template ($response, "%{%i%s%*{%i%i%i%{%@",
                                  $tag);
   return $this->ber_error ("Error decoding response PDU")
     unless defined $snmpver;
```

ANEXO III

```
    return $this->error ("Received SNMP response with unknown snmp-version field $snmpver")
                unless $snmpver == $this->snmp_version;
    if ($this->{error_status} != 0) {
      if ($errorp) {
                my ($oid, $errmsg);
                $errmsg = $error_status_code[$this->{error_status}] || $this->{error_status};
                $oid = $oids->[$this->{error_index}-1]
                  if $this->{error_index} > 0 && $this->{error_index}-1 <= $#{$oids};
                $oid = $oid->[0]
                  if ref($oid) eq 'ARRAY';
                return ($community, $request_id,
                        $this->error ("Received SNMP response with error code\n"
                                      ."  error status: $errmsg\n"
                                      ."  index ".$this->{error_index}
                                      .(defined $oid
                                              ? " (OID: ".&BER::pretty_oid($oid).")"
                                              : "")));
      } else {
                if ($this->{error_index} == 1) {
                  @rest[$this->{error_index}-1..$this->{error_index}] = ();
                }
      }
    }
    ($community, $request_id, @rest);
}

sub send_query ($$) {
    my ($this,$query) = @_;
    send ($this->sock,$query,0,$this->remote_addr);
}

sub sa_equal_p ($$$) {
    my ($this, $sa1, $sa2) = @_;
    my ($p1,$a1,$p2,$a2);

    # Disable strict subs to stop old versions of perl from
    # complaining about AF_INET6 when Socket6 is not available
    if($this->{'sockfamily'} == AF_INET) {
                # IPv4 addresses
                ($p1,$a1) = unpack_sockaddr_in ($sa1);
                ($p2,$a2) = unpack_sockaddr_in ($sa2);
    } elsif($this->{'sockfamily'} == AF_INET6()) {
                # IPv6 addresses
                ($p1,$a1) = unpack_sockaddr_in6 ($sa1);
                ($p2,$a2) = unpack_sockaddr_in6 ($sa2);
    } else {
                return 0;
    }
    use strict "subs";

    if (! $this->{'lenient_source_address_matching'}) {
                return 0 if $a1 ne $a2;
    }
    if (! $this->{'lenient_source_port_matching'}) {
                return 0 if $p1 != $p2;
    }
    return 1;
}

sub receive_response_3 {
    my ($this, $response_tag, $oids, $errorp) = @_;
    my ($remote_addr);
    $remote_addr = recv ($this->sock,$this->{'pdu_buffer'},$this->max_pdu_len,0);
    return $this->error ("receiving response PDU: $!")
                unless defined $remote_addr;
    return $this->error ("short (".length $this->{'pdu_buffer'}
                                  ." bytes) response PDU")
```

ANEXO III

```
            unless length $this->{'pdu_buffer'} > 2;
    my $response = $this->{'pdu_buffer'};
    if (defined $this->{'remote_addr'}) {
            if (! $this->sa_equal_p ($remote_addr, $this->{'remote_addr'})) {
               if ($this->{'debug'} && !$SNMP_Session::recycle_socket) {
                        carp ("Response came from ".&SNMP_Session::pretty_address($remote_addr)
                            ."; not ".&SNMP_Session::pretty_address($this->{'remote_addr'}))
                                unless $SNMP_Session::suppress_warnings;
               }
               return 0;
            }
    }
    $this->{'last_sender_addr'} = $remote_addr;
    my ($response_community, $response_id, @unwrapped)
            = $this->unwrap_response_5b ($response, $response_tag,
                                                    $oids, $errorp);
    if ($response_community ne $this->{community}
       || $response_id ne $this->{request_id}) {
            if ($this->{'debug'}) {
               carp ("$response_community != $this->{community}")
                        unless $SNMP_Session::suppress_warnings
                           || $response_community eq $this->{community};
               carp ("$response_id != $this->{request_id}")
                        unless $SNMP_Session::suppress_warnings
                           || $response_id == $this->{request_id};
            }
            return 0;
    }
    if (!defined $unwrapped[0]) {
            $this->{'unwrapped'} = undef;
            return undef;
    }
    $this->{'unwrapped'} = \@unwrapped;
    return length $this->pdu_buffer;
}

sub receive_trap {
    my ($this) = @_;
    my ($remote_addr, $iaddr, $port, $trap);
    $remote_addr = recv ($this->sock,$this->{'pdu_buffer'},$this->max_pdu_len,0);
    return undef unless $remote_addr;

    if( (defined $ipv6_addr_len) && (length $remote_addr == $ipv6_addr_len)) {
            ($port,$iaddr) = unpack_sockaddr_in6($remote_addr);
    } else {
            ($port,$iaddr) = unpack_sockaddr_in($remote_addr);
    }

    $trap = $this->{'pdu_buffer'};
    return ($trap, $iaddr, $port);
}

sub describe {
    my($this) = shift;
    print $this->to_string (),"\n";
}

sub to_string {
    my($this) = shift;
    my ($class,$prefix);

    $class = ref($this);
    $prefix = ' ' x (length ($class) + 2);
    ($class
     .(defined $this->{remote_hostname}
       ? " (remote host: \"".$this->{remote_hostname}."\""
       ." ".&SNMP_Session::pretty_address ($this->remote_addr).")"
```

ANEXO III

```
      : " (no remote host specified)")
    ."\n"
    .$prefix."  community: \"".$this->{'community'}."\"\n"
    .$prefix." request ID: ".$this->{'request_id'}."\n"
    .$prefix."PDU bufsize: ".$this->{'max_pdu_len'}." bytes\n"
    .$prefix."    timeout: ".$this->{timeout}."s\n"
    .$prefix."    retries: ".$this->{retries}."\n"
    .$prefix."    backoff: ".$this->{backoff}.")");
##    sprintf ("SNMP_Session: %s (size %d timeout %g)",
##    &SNMP_Session::pretty_address ($this->remote_addr),$this->max_pdu_len,
##                  $this->timeout);
}

sub receive_request {
   my ($this) = @_;
   my ($remote_addr, $iaddr, $port, $request);

   $remote_addr = recv($this->sock, $this->{'pdu_buffer'},
                                        $this->{'max_pdu_len'}, 0);
   return undef unless $remote_addr;

   if( (defined $ipv6_addr_len) && (length $remote_addr == $ipv6_addr_len)) {
            ($port,$iaddr) = unpack_sockaddr_in6($remote_addr);
   } else {
            ($port,$iaddr) = unpack_sockaddr_in($remote_addr);
   }

   $request = $this->{'pdu_buffer'};
   return ($request, $iaddr, $port);
}

sub decode_request {
   my ($this, $request) = @_;
   my ($snmp_version, $community, $requestid, $errorstatus, $errorindex, $bindings);

   ($snmp_version, $community, $requestid, $errorstatus, $errorindex, $bindings)
            = decode_by_template ($request, "%{%i%s%*{%i%i%i%@", SNMP_Session::get_request);
   if (defined $snmp_version)
   {
            # Its a valid get_request
            return(SNMP_Session::get_request, $requestid, $bindings, $community);
   }

   ($snmp_version, $community, $requestid, $errorstatus, $errorindex, $bindings)
            = decode_by_template ($request, "%{%i%s%*{%i%i%i%@", SNMP_Session::getnext_request);
   if (defined $snmp_version)
   {
            # Its a valid getnext_request
            return(SNMP_Session::getnext_request, $requestid, $bindings, $community);
   }

   ($snmp_version, $community, $requestid, $errorstatus, $errorindex, $bindings)
            = decode_by_template ($request, "%{%i%s%*{%i%i%i%@", SNMP_Session::set_request);
   if (defined $snmp_version)
   {
            # Its a valid set_request
            return(SNMP_Session::set_request, $requestid, $bindings, $community);
   }

   # Something wrong with this packet
   # Decode failed
   return undef;
}

package SNMPv2c_Session;
use strict qw(vars subs);            # see above
use vars qw(@ISA);
```

ANEXO III

```
use SNMP_Session;
use BER;
use Carp;

@ISA = qw(SNMPv1_Session);

sub snmp_version { 1 }

sub open {
   my $session = SNMPv1_Session::open (@_);
   return undef unless defined $session;
   return bless $session;
}

## map_table_start_end using get-bulk
##
sub map_table_start_end ($$$$$) {
   my ($session, $columns, $mapfn, $start, $end, $max_repetitions) = @_;

   my @encoded_oids;
   my $call_counter = 0;
   my $base_index = $start;
   my $ncols = @{$columns};
   my @collected_values = ();

   if (! $session->{'use_getbulk'}) {
            return SNMP_Session::map_table_start_end
               ($session, $columns, $mapfn, $start, $end, $max_repetitions);
   }
   $max_repetitions = $session->default_max_repetitions
            unless defined $max_repetitions;

   for (;;) {
            foreach (@encoded_oids = @{$columns}) {
               $_=encode_oid (@{$_},split '\.',$base_index)
                        || return $session->ber_error ("encoding OID $base_index");
            }
            if ($session->getbulk_request_response (0, $max_repetitions,
                                                                        @encoded_oids)) {
               my $response = $session->pdu_buffer;
               my ($bindings) = $session->decode_get_response ($response);
               my @colstack = ();
               my $k = 0;
               my $j;

               my $min_index = undef;

               my @bases = @{$columns};
               my $n_bindings = 0;
               my $binding;

               while ($bindings ne '') {
                        ($binding, $bindings) = decode_sequence ($bindings);
                        my ($oid, $value) = decode_by_template ($binding, "%O%@");

                        push @{$colstack[$k]}, [$oid, $value];
                        ++$k; $k = 0 if $k >= $ncols;
               }

               $base_index = undef;
              walk_rows_from_pdu:
               for (;;) {
                        my $min_index = undef;

                        for ($k = 0; $k < $ncols; ++$k) {
                           $collected_values[$k] = undef;
                           my $pair = $colstack[$k]->[0];
```

ANEXO III

```
                    unless (defined $pair) {
                        $min_index = undef;
                        last walk_rows_from_pdu;
                    }
                    my $this_index
                        = SNMP_Session::oid_diff ($columns->[$k], $pair->[0]);
                    if (defined $this_index) {
                        my $cmp
                            = !defined $min_index
                                ? -1
                                : SNMP_Session::index_compare
                                    ($this_index, $min_index);
                        if ($cmp == -1) {
                            for ($j = 0; $j < $k; ++$j) {
                                unshift (@{$colstack[$j]},
                                         [$min_index,
                                          $collected_values[$j]]);
                                $collected_values[$j] = undef;
                            }
                            $min_index = $this_index;
                        }
                        if ($cmp <= 0) {
                            $collected_values[$k] = $pair->[1];
                            shift @{$colstack[$k]};
                        }
                    }
                }
                ($base_index = undef), last
                    if !defined $min_index;
                last
                    if defined $end
                    and SNMP_Session::index_compare ($min_index, $end) >= 0;
                &$mapfn ($min_index, @collected_values);
                ++$call_counter;
                $base_index = $min_index;
            }
        } else {
            return undef;
        }
        last if !defined $base_index;
        last
            if defined $end
            and SNMP_Session::index_compare ($base_index, $end) >= 0;
    }
    $call_counter;
}

1;
```

ANEXO IV

```
use BER;
require 'SNMP_Session.pm';

$session = SNMP_Session->open ($host, $community, $port)
   or die "couldn't open SNMP session to $host";

# Set $oid1, $oid2... to the BER-encoded OIDs of the MIB
# variables you want to get.

if ($session->get_request_response ($oid1, $oid2, ...)) {
   ($bindings) = $session->decode_get_response ($session->{pdu_buffer});

   while ($bindings ne '') {
      ($binding,$bindings) = &decode_sequence ($bindings);
      ($oid,$value) = &decode_by_template ($binding, "%O%@");
      print &pretty_print ($oid)," => ", &pretty_print ($value), "\n";
   }
} else {
   die "No response from agent on $host";
}

%ugly_oids = qw(sysDescr.0      1.3.6.1.2.1.1.1.0
              sysContact.0    1.3.6.1.2.1.1.4.0);
foreach (keys %ugly_oids) {
   $ugly_oids{$_} = encode_oid (split (/\./, $ugly_oids{$_}));
   $pretty_oids{$ugly_oids{$_}} = $_;
}
...
if ($session->get_request_response ($ugly_oids{'sysDescr.0'},
                                    $ugly_oids{'sysContact.0'})) {
   ($bindings) = $session->decode_get_response
                ($session->{pdu_buffer});
   while ($bindings ne '') {
      ($binding,$bindings) = &decode_sequence ($bindings);
      ($oid,$value) = &decode_by_template ($binding, "%O%@");
      print $pretty_oids{$oid}," => ",
            &pretty_print ($value), "\n";
   }
} ...

sub walk_function ($$$) {
  my ($index, $val1, $val3) = @_;
  ...
}

...
$columns = [$base_oid1, $base_oid3];
$n_rows = $session->map_table ($columns, \&walk_function);

grep (defined $_ && ($_=pretty_print $_), ($val1, $val3));
```

ANEXO V

Código da Tela de Entrada

```
<?php

header("Content-type: text/vnd.wap.wml");

echo "<?xml version=\"1.0\" ?>";
?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
<head>
      <meta http-equiv="Cache-Control" content="max-age=10" forua="true"/>
</head>
<card id="Autentica" title="taurus">
<?php
$db=mysql_connect("wap.taurus.lrg.ufsc.br","root");
mysql_select_db("eadi",$db);
$resultado=mysql_query("select * from clientes where id='$usuario' AND senha='$senha'",$db);
if(!mysql_num_rows($resultado)){
mysql_close($db);
?>
   <p align="center">
   <br/>
   Usu&#225;rio ou senha inv&#225;lida !!! <br/> <br/>
   Tente novamente!
   <do type="accept" label="Repetir">
   <go href="loginus.php4">
          <setvar name="usuario" value=""/>
          <setvar name="senha"   value=""/>
   </go>

   </do>
   </p>

<?php
}

if(mysql_num_rows($resultado)){
?>
   <p align="center">
   <i>
   <?php
   $nome=mysql_result($resultado,0,"nome_abrev");
   echo "$nome,";
   ?>
   <br/>
   Seja bem vindo!</i>
   <br/>
   <br/>
   <?php
   if (mysql_result($resultado,0,"alt")=='A')
         {
         echo "Ocorreram altera&#231;&#245;es desde o seu &#250;ltimo acesso !!!!";
         $sql_result=mysql_query("update clientes set alt='S' where id='$usuario'");
         }
   if (mysql_result($resultado,0,"alt")=='S')
         {
         echo "N&#227;o ocorreram altera&#231;&#245;es desde o seu &#250;ltimo acesso !!!!";
         }
   mysql_close($db);
   ?>
   <do type="accept" label="Avan&#231;ar">
   <?php
   $temp=time();
```

ANEXO V

```
   if (($temp-$vtime)<300){
        $nome=mysql_result($resultado,0,"nome_abrev");
        echo "<go href='principal.php4' method='post'>";
        $temp=time();
        echo "<postfield name='vnome' value='$nome' />";
        echo "<postfield name='vtime' value='$temp' />";
        mysql_close($db);
    }
    else
    {
        echo "<go href='index.wml' method='post'>";
    }
    ?>
        <setvar name="usuario" value=""/>
        <setvar name="senha"   value=""/>

   </go>
   </do>
   </p>

<?php
}
?>
</card>
</wml>
```

ANEXO V