UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SOFTWARE PARA GERÊNCIA DE SISTEMAS BASEADO NO PADRÃO WBEM UTILIZANDO O WMI

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

EDSON LUIZ BRAZ DA SILVA JUNIOR

BLUMENAU, JUNHO/2003

2003/1-18

PROTÓTIPO DE UM SOFTWARE PARA GERÊNCIA DE SISTEMAS BASEADO NO PADRÃO WBEM UTILIZANDO O WMI

EDSON LUIZ BRAZ DA SILVA JUNIOR

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Francisco Adell Péricas — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Francisco Adell Péricas

_____

Prof. Sérgio Stringari

_____

Prof. Mauro Marcelo Mattos

AGRADECIMENTOS

Agradeço aos professores do curso de Bacharelado em Ciências da Computação da Universidade Regional de Blumenau, pelo apoio e conhecimento repassado.

A todo o pessoal da Seção de Apoio ao Usuário do Núcleo de Informática da FURB pelo incentivo e compreensão nos momentos de minha ausência.

Aos meus familiares e em especial para meus pais Terezinha e Edson que nos momentos de dificuldade sempre estiveram presentes.

A minha namorada Evelyn por toda sua ajuda e compreensão durante o desenvolvimento deste trabalho.

Agradeço também a todos os amigos que estiveram comigo nesta jornada.

RESUMO

Este trabalho apresenta um estudo sobre o padrão *Web Based Enterprise Management* (WBEM) da *Distributed Management Task Force* (DMTF) através da especificação e implementação de um protótipo de software de gerenciamento de sistemas, utilizando o *Windows Management Instrumentation* (WMI), o *framework* .NET e a linguagem de programação C# com ASP.NET.

ABSTRACT

This work shows a study about the *Web Based Enterprise Management* standard (WBEM) from *Distributed Management Task Force* (DMTF) by a specification and an implementation of a software prototype to manager systems, using *Windows Management Instrumentation* (WMI), .NET framework and programing language C# with ASP.NET.

LISTA DE FIGURAS

Figura 2.1 – Áreas da gerência corporativa 14
Figura 2.2 – Modelo de gerenciamento de redes 15
Figura 2.3 – Arquitetura de gerência via WEB 19
Figura 3.1 – Modelo do fluxo de dados do WBEM 21
Figura 3.2 – Modelo completo do padrão WBEM 22
Figura 3.3 – Estrutura do CIM Meta Schema 24
Figura 3.4 – Camadas do CIM Schema 25
Figura 3.5 – CIM Core Model 27
Figura 3.6 – CIM Schema 28
Figura 3.7 – Arquitetura WMI 33
Figura 3.8 – Espaço para nomes do WMI 35
Figura 4.1 – Diagrama de Casos de Uso 38
Figura 4.2 – Diagrama de Classes 40
Figura 4.3 – Conexão ao serviço WMI 41
Figura 4.4 – Consulta de informações de inventário 42
Figura 4.5 – Consulta de processos 42
Figura 4.6 – Consulta de serviços 43
Figura 4.7 – Recebe notificações 43
Figura 4.8 – Diagrama de *deployment* 46
Figura 4.9 – Arquitetura do Protótipo 46
Figura 4.10 – Página inicial do protótipo 51
Figura 4.11 – Página de visualização de informações de inventário 52
Figura 4.12 – Página de visualização dos processos 53

Figura 4.13 – Página de visualização dos serviços 54

Figura 4.14 – Página de notificações 55

LISTA DE QUADROS

Quadro 3.1 – Exemplo de arquivo MOF 29

Quadro 3.2 – Exemplo de consulta WQL..... 36

Quadro 4.1 – Método para conexão ao serviço WMI..... 47

Quadro 4.2 – Método para consulta de classes 47

Quadro 4.3 – Método para consulta de informações do processador 48

Quadro 4.4 – Classe InfoWMI..... 49

LISTA DE SIGLAS E ABREVIATURAS

API	*Application Program Interface*
CIM	*Common Information Model*
CIMOM	*CIM Object Manager*
CMIP	*Common Management Information Protocol*
DMTF	*Distributed Management Task Force*
DTD	*Document Type Definition*
HTTP	*Hipertext Transfer Protocol*
IIS	*Internet Information Services*
LAN	*Local Area Network*
MIB	*Management Information Base*
MOF	*Managed Object Format*
SNMP	*Simple Management Network Protocol*
UML	*Unifield Model Language*
WAN	*Wide Area Network*
WBEM	*Web Based Enterprise Management*
WQL	*WMI Query Language*
WMI	*Windows Management Instrumentation*
XML	*Extensible Markup Language*
XSL	*Extensible Style Language*

SUMÁRIO

1 INTRODUÇÃO 12

1.1 OBJETIVOS 12

1.2 ESTRUTURA DO TRABALHO 13

2 GERÊNCIA CORPORATIVA 14

2.1 GERÊNCIA DE REDES 15

2.1.1 ARQUITETURA 15

2.1.2 ÁREAS FUNCIONAIS 15

2.1.3 PROTOCOLOS DE GERENCIAMENTO DE REDES 16

2.2 GERÊNCIA DE SISTEMAS 16

2.3 GERÊNCIA DE APLICAÇÕES 17

2.4 GERÊNCIA DE SERVIÇOS 17

2.5 GERÊNCIA VIA WEB 18

3 WBEM 20

3.1 COMPONENTES DA ARQUITETURA WBEM 23

3.1.1 CIM 23

3.1.1.1 CIM META SCHEMA 23

3.1.1.2 CIM SCHEMA 25

3.1.2 CIMOM 29

3.1.3 MOF – MANAGED OBJECT FORMAT 29

3.2 REPRESENTAÇÃO DO CIM EM XML 29

3.3 IMPLEMENTAÇÕES DO WBEM 31

3.4 WINDOWS MANAGEMENT INSTRUMENTATION 32

3.4.1 ARQUITETURA WMI 32

3.4.2 WMI QUERY LANGUAGE 35
4 DESENVOLVIMENTO DO TRABALHO 37
4.1 REQUISITOS PRINCIPAIS DO PROBLEMA 37
4.2 ESPECIFICAÇÃO DO PROTÓTIPO 37
4.2.1 DIAGRAMA DE CASOS DE USO 38
4.2.2 DIAGRAMA DE CLASSES 39
4.2.3 DIAGRAMAS DE SEQUÊNCIA 40
4.3 IMPLEMENTAÇÃO 44
4.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS 44
4.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO 50
4.4 RESULTADOS E DISCUSSÃO 56
5 CONCLUSÕES 57
5.1 EXTENSÕES 58

1 INTRODUÇÃO

Numa publicação de Pereira (2001) é exposto que as redes locais de computadores em corporações têm crescido significativamente nos últimos anos. Este crescimento, somado com a facilidade de acesso à internet de um número cada vez maior de pessoas tem impulsionado o surgimento de diversas tecnologias baseadas na WEB. Dentre estas tecnologias surgem padrões de gerenciamento de redes e sistemas.

A *Distributed Management Task Force* (DMTF) propôs um padrão de gerenciamento corporativo baseado em uma arquitetura WEB chamada *Web-Based Enterprise Management* (WBEM). Este padrão define um modelo de dados totalmente orientado a objetos, onde o repositório destes dados pode ser acessado e gerenciado via WEB, utilizando o protocolo HTTP com a linguagem XML (CASTRO, 2002). A grande inovação deste padrão é a utilização de um *browser* para, de qualquer ponto na rede ou internet, realizar as tarefas de gerenciamento. Esta inovação traz grandes perspectivas futuras, sendo importante o seu conhecimento, principalmente para aqueles usuários ou desenvolvedores de soluções de gerência baseadas em arquitetura WEB.

Este trabalho teve como propósito realizar um estudo teórico do padrão de gerência de sistemas WBEM da DMTF, bem como da sua arquitetura, funcionalidade, componentes, etc. Para ilustrar o funcionamento desta tecnologia, foi especificado e implementado um protótipo utilizando a implementação do WBEM proposta pela Microsoft chamada de *Windows Management Intrumentation* (WMI). Além disso, será feita uma breve descrição das implementações do WBEM existentes atualmente.

1.1 OBJETIVOS

O trabalho desenvolvido teve como objetivo estudar o padrão WBEM da DMTF através da especificação e implementação de um software de gerenciamento de sistemas, utilizando o WMI, o *framework* .NET e a linguagem de programação C# com ASP.NET especificados pela Microsoft.

Os objetivos específicos do trabalho são:

a) coletar e visualizar informações de inventário de um computador conectado à rede;
b) visualizar os processos e serviços disponíveis e em execução no sistema operacional de um computador conectado à rede;
c) receber notificações do sistema operacional de um computador conectado a uma rede em eventos pré-definidos como, por exemplo, se o uso do seu processador ultrapassar 90%;
d) demonstrar o funcionamento do gerenciador de objetos através dos itens acima mencionados.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos, conforme é apresentado a seguir.

O capítulo 1 apresenta a estrutura geral do trabalho: introdução, objetivos, a localização dos assuntos abordados e a organização do trabalho.

O capítulo 2 refere-se ao gerenciamento corporativo e suas respectivas áreas, abordando em detalhes os conceitos relacionados a cada área, bem como caracterizando o nível de gerência onde se encontra este trabalho. Aborda, também, o gerenciamento via WEB: conceitos, arquitetura, componentes e funcionamento.

O capítulo 3 trata do WBEM da DMTF e do WMI da Microsoft. Sobre o WBEM são abordados: conceitos, funcionalidade, arquitetura, componentes, modelo de dados e representação do modelo de dados. Demonstra, também, a representação do CIM em XML e cita as implementações existentes do WBEM, bem como suas características. Já sobre o WMI serão abordados suas características, arquitetura, funcionamento, provedores de acessos e a linguagem de consulta WQL.

O capítulo 4 refere-se ao desenvolvimento do protótipo, onde se apresenta a especificação, implementação e os resultados obtidos sobre o mesmo.

O capítulo 5 apresenta as conclusões e sugestões de continuidade do trabalho.

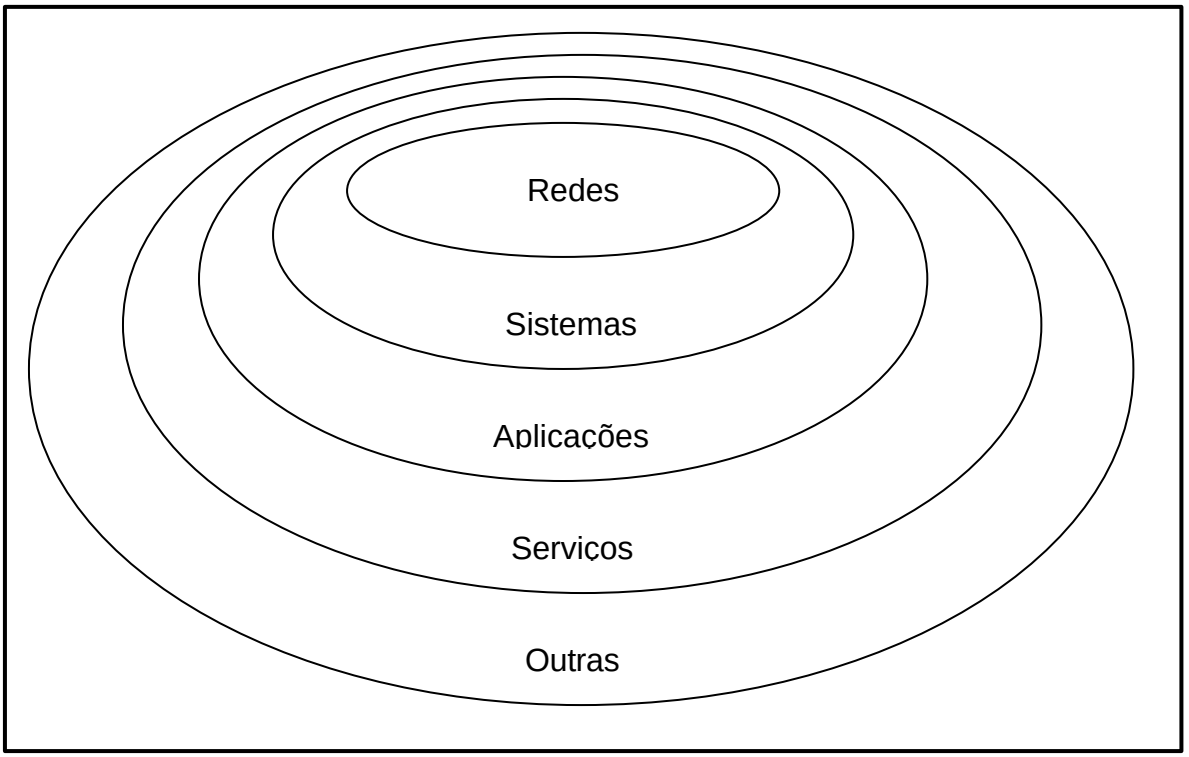
2 GERÊNCIA CORPORATIVA

A gerência corporativa é um conjunto de aplicações associadas a *softwares* e *hardwares* que tem como objetivo monitorar e controlar os recursos de informática de uma empresa (CARVILHE, 2000). Sua principal função é garantir a comunicação das informações da empresa de forma eficiente e segura. Integrando aplicações para se obter o gerenciamento distribuído dos recursos de informática.

A existência de diversas plataformas de *hardware* e *software*, somados com a complexidade de gerenciamento dos mais diversos sistemas, aplicações e serviços de uma rede heterogênea, tornam a tarefa de gerenciamento corporativo desafiadora (CARVILHE, 2000).

A gerência corporativa abrange diversas áreas de gerência definidas em camadas a partir da gerência de redes, seguida da gerência de sistemas, aplicações, serviços e outras disciplinas de gerência que em conjunto definem todos os recursos corporativos que necessitam de gerenciamento (CARVILHE, 2000). A fig. 2.1 demonstra as áreas da gerência corporativa.

Figura 2.1 – Áreas da gerência corporativa



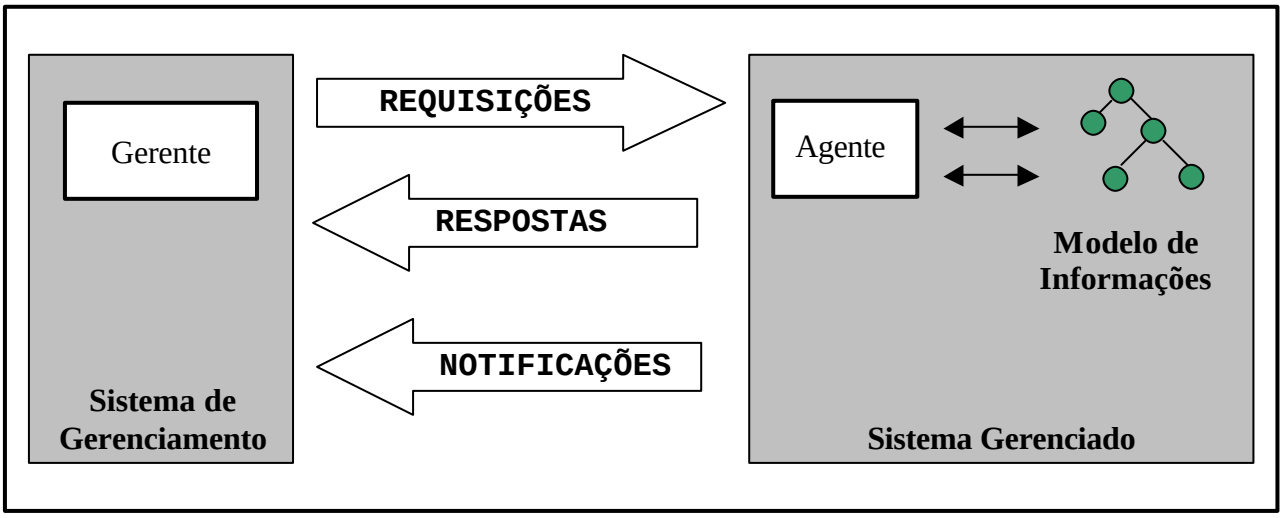
2.1 GERÊNCIA DE REDES

Neste capítulo está descrito a gerência de redes, sua arquitetura, áreas funcionais e protocolos de gerenciamento.

2.1.1 ARQUITETURA

A gerência de redes segundo Stallings (1999) é responsável pela monitoração, controle e configuração de dispositivos de redes. O modelo de gerenciamento baseia-se no paradigma gerente-agente, onde gerente é o próprio sistema de gerenciamento e o agente é instalado nos equipamentos que se deseja monitorar com a tarefa de responder a requisições do gerente. A fig. 2.2 representa este modelo.

Figura 2.2 – Modelo de gerenciamento de redes



2.1.2 ÁREAS FUNCIONAIS

A ISO/IEC dividiu o gerenciamento de redes em áreas funcionais de gerenciamento, permitindo desta forma identificar diferentes atividades características da gerência de redes. Estas áreas dividem-se da seguinte forma (STALLINGS, 1999):

a) **gerência de falhas**: permite a detecção, isolamento e correção de anomalias de uma rede e de seus equipamentos;
b) **gerência de configuração**: permite o controle, a identificação e a coleta de dados de equipamentos e de conexões entre eles, assim como o planejamento, a instalação

e a configuração dos equipamentos de rede de forma a garantir os serviços requeridos pelos clientes de uma rede;

c) **gerência de contabilização**: habilita o uso dos serviços da rede para mediação e determinação de custos, provendo facilidades de definição de parâmetros de bilhetagem e de coleta de registros de cobrança do uso de uma rede;
d) **gerência de desempenho**: permite a geração e a avaliação de relatórios de dados coletados de uma rede, com objetivo de medir, analisar e controlar o seu desempenho, de acordo com requisitos de qualidade de serviço requeridos pelos usuários da rede e de seus equipamentos;
e) **gerência de segurança**: permite prevenir e detectar o uso impróprio ou não autorizado de recursos de uma rede, assim como administrar a sua segurança.

2.1.3 PROTOCOLOS DE GERENCIAMENTO DE REDES

Os agentes se comunicam com os gerentes através de um protocolo de gerenciamento de redes do nível de aplicação, que utiliza a arquitetura de comunicação da rede (PEREIRA, 2001). Para que seja possível a comunicação entre um gerente e um agente é necessário que ambos compartilhem o mesmo esquema conceitual de informações.

Segundo Stallings (1996), no gerenciamento de redes baseado no modelo TCP/IP é amplamente utilizado o protocolo de comunicação *Simple Network Management Protocol* (SNMP) e para redes baseados no modelo OSI é utilizado o *Common Management Information Protocol* (CMIP).

Este trabalho não tem a intenção de detalhar estes e outros protocolos utilizados para o gerenciamento de redes, pois não serão diretamente utilizados. Para maiores informações sobre os protocolos de gerenciamento de redes, utilizados na comunicação entre gerentes e agentes, consulte bibliografias específicas desta área, tais como Stallings (1996) e Pereira (2001).

2.2 GERÊNCIA DE SISTEMAS

Segundo Carvilhe (2000), a área de gerência de sistemas trata especificamente dos componentes dos servidores e computadores pessoais. Enquanto a gerência de rede trata da

rede como um todo, à gerência de sistemas trata do que está dentro de cada dispositivo da rede. A gerência de sistemas é responsável pelas seguintes tarefas:

a) inventário de *software* e status dos sistemas operacionais e aplicações;
b) inventário de *hardware* e status da CPU, memória, discos e periféricos;
c) controle de licenças de *softwares*;
d) distribuição e atualização automática de *softwares*;
e) controle remoto de servidores e estações;
f) outras tarefas relativas a sistemas e dispositivos específicos.

2.3 GERÊNCIA DE APLICAÇÕES

A gerência de aplicação, segundo Carvilhe (2000), corresponde ao controle, monitoramento e configuração de aplicações que são executadas num determinado *host* conectado à rede. Esta área de gerência é muito mais complexa que a gerência de sistema, pois existem mais versões de aplicações sendo executadas do que sistemas. E diversas variáveis precisam ser consideradas para a implementação deste tipo de gerência, incluindo a lista de executáveis, arquivos de configuração associados, métodos de comunicação entre processos, versões, compatibilidade entre plataformas entre outras.

A iniciativa mais importante na área de gerência de aplicações é a API *Aplication Response Measurement* (ARM), definida e mantida pela *ARM Working Group* (CARVILHE, 2000).

2.4 GERÊNCIA DE SERVIÇOS

A gerência de serviços, incluindo a gerência de acordo de níveis de serviço, descreve como são definidos, monitorados e apresentados, os serviços contratados externamente pela empresa e os serviços utilizados internamente (CARVILHE, 2000).

Os serviços que podem ser gerenciados incluem a utilização de canais de voz, tempo de resposta diversos, acesso à internet, carga de utilização de LANS e WANS, utilização de e-mails, e outros serviços que possam ser quantificados. Os acordos de nível de serviço são medidos em tempo de resposta, disponibilidade, utilização, número de pacotes entre outros. A

gerência de serviços é a maneira principal de avaliar a qualidade de serviços e os custos envolvidos na empresa.

2.5 GERÊNCIA VIA WEB

A gerência via WEB é definida em Carvilhe (2000) como a utilização de tecnologias WEB para a realização da gerência corporativa. A gerência via WEB é muito nova, existem várias frentes de trabalho definindo padrões, *frameworks*, modelos de objetos, protocolos e API, implementando códigos e testes de produtos. Para que esse tipo de gerência se torne realidade muitas pessoas e organizações aliadas a tecnologias precisam reunir esforços e contribuições para colocar a gerência corporativa sobre o controle desta nova arquitetura. Com a utilização de tecnologias baseadas na arquitetura WEB na gerência corporativa é possível realizar o gerenciamento de qualquer ponto da rede ou da internet através de um *browser* (navegador).

O modelo genérico de *framework* de gerência via WEB é dividido em sete partes, conforme descrito abaixo e apresentado na fig. 2.3 (CARVILHE, 2000):

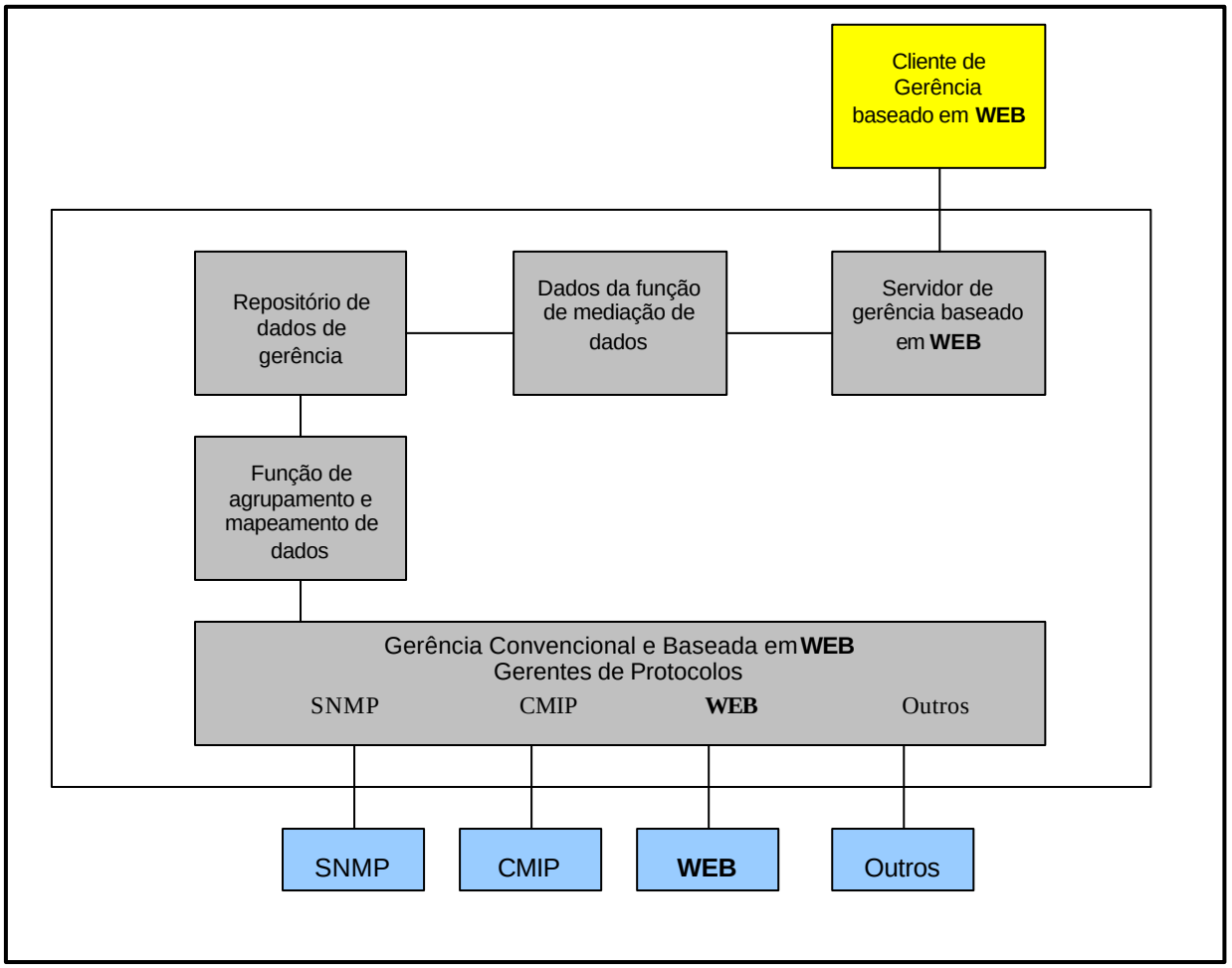
a) **cliente WEB**, é o computador de gerência corporativa que contém o *browser*. O *browser* é a interface padrão para visualização das informações de gerência. O *browser* permite que o usuário digite comandos, solicite informações de gerência e apresente resultados;
b) **servidor WEB**, o servidor WEB corresponde ao programa servidor responsável por atender as requisições realizadas a partir do *browser* e retornar as respostas. O servidor WEB possui uma interface para a função de mediação de dados;
c) **função de mediação de dados**, é a função responsável pela conversão dos dados da gerência do repositório de dados para um formato que seja reconhecido pelo servidor WEB pelas aplicações de gerência;
d) **repositório de dados**, corresponde ao banco de dados onde são armazenadas as informações de gerência;
e) **função de agrupamento e mapeamento**, corresponde à função responsável pela conversão dos dados coletados pelos gerentes em um formato padrão reconhecido pelo repositório de gerência;

f) **gerentes de protocolos**, correspondem ao lado gerente responsável pela gerência convencional e a gerência via WEB. Cada gerente utiliza o seu protocolo para se comunicar com o agente correspondente. Os gerentes de protocolos tratam SNMP, CMIP, WEB entre outros *frameworks* de gerência;

g) **agente**, o agente de gerência corresponde à entidade de processo localizado no dispositivo gerenciado. Muitos agentes são capazes de gerar alarmes indicando que alguma ação predefinida aconteceu no dispositivo gerenciado.

Conforme descrito em Carvilhe (2000), existem vários benefícios relacionados a implementação da gerência baseada em tecnologias WEB. Entre os principais, destacam-se o baixo custo de implementação e a facilidade de uso e acesso através de um *browser* internet.

Figura 2.3 – Arquitetura de gerência via WEB



3 WBEM

Em meados do ano de 1996 um grupo de grandes fornecedores interessados na gerência via WEB, entre eles a BMC Software, CISCO, Compaq, Intel e Microsoft, anunciou a iniciativa de definição de um esquema de gerenciamento corporativo que trabalharia com a WEB e os *frameworks* convencionais de gerência de redes e sistemas. Este *framework* é conhecido como *Web-Based Enterprise Management* (WBEM) (CARVILHE, 2000). Em 1998, a *Distributed Management Task Force* (DMTF) herdou a responsabilidade pelo WBEM (STEINKE, 1999).

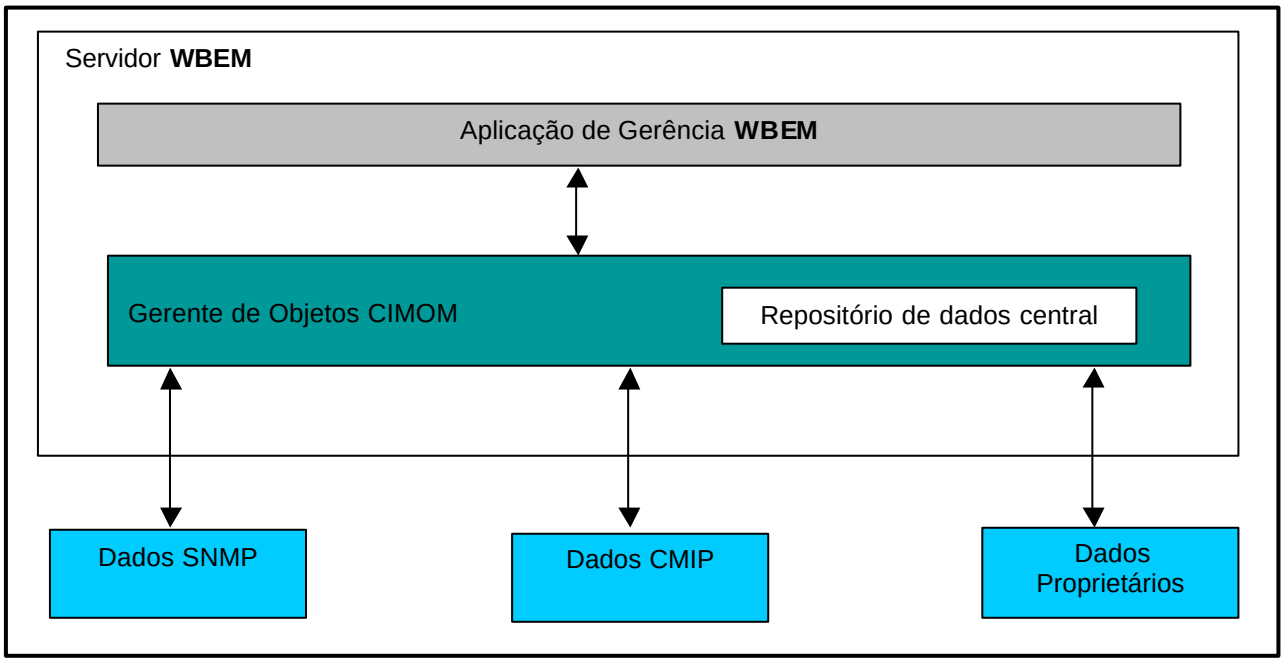
O WBEM foi amadurecendo com a DMTF, de forma que três componentes chaves emergiram: *Common Information Model* (CIM), uma coleção de esquemas orientados a objetos para gerenciamento de informação; HTTP, o protocolo universal para transporte de informações na WEB; e a *Extensible Markup Language* (XML), um poderoso método para criar informação que será transportada via HTTP de uma aplicação para outra, de um *browser* para uma aplicação ou de um *browser* para objetos gerenciados (STEINKE, 1999).

O CIM é um modelo de dados com o propósito de descrever as informações de gerenciamento num formato padrão, seguindo o conceito de orientação a objetos, permitindo que outros modelos, incluindo modelos de informação SNMP e CMIP, sejam mapeados para sua estrutura de dados (STEINKE, 1999).

Para interagir com a aplicação gerente surge o CIM *Object Manager* (CIMOM), responsável pelo controle dos objetos gerenciados, incluindo o armazenamento e o acesso a objetos no repositório central de dados (CASTRO, 2002). O CIMOM possui sua própria linguagem para definição dos objetos gerenciados denominada de *Managed Object Format* (MOF). A fig. 3.1 representa o fluxo de dados básico do modelo WBEM.

Os componentes representados na fig. 3.2 apresentam o modelo completo do padrão WBEM. O servidor WBEM contém o gerenciador de objetos CIMOM, que pode aceitar todos os objetos dos *frameworks* de gerência de sistemas e redes existentes. Os dados de objetos são transformados em um formato padrão e armazenados no repositório de dados. O servidor WBEM contém também o servidor de aplicação, que por sua vez pode ser acessado pelos clientes WBEM.

Figura 3.1 – Modelo do fluxo de dados do WBEM



O *browser* cliente utiliza um conjunto de aplicações residentes no servidor WBEM ou em um sistema distinto, acessando as aplicações de gerência através do protocolo HTTP. Quando executadas, as aplicações de gerência acessam o CIMOM, este interpreta a requisição feita para a informação solicitada sobre o objeto gerenciado e retorna seus valores. A aplicação reconhece os objetos presentes no repositório de dados porque ela entende que a lista de objetos do esquema foi instanciada. As informações dos objetos gerenciados são entregues ao CIMOM para serem colocadas no repositório de dados pelos diversos provedores conectados (CARVILHE, 2000).

Os provedores correspondem a um conjunto de processos que se comunicam com os agentes de gerência de sistemas convencionais nos dispositivos gerenciados, obtendo os valores dos objetos gerenciados. Os provedores são responsáveis pela obtenção e apresentação dos dados de gerência de sistemas e redes, que estão sendo trocados com o agente do dispositivo gerenciado. Existem provedores que utilizam arquiteturas padronizadas como SNMP e CMIP, estes provedores são denominados *Stantard Providers*. Segundo Carvilhe (2000) o *framework* WBEM define os seguintes provedores:

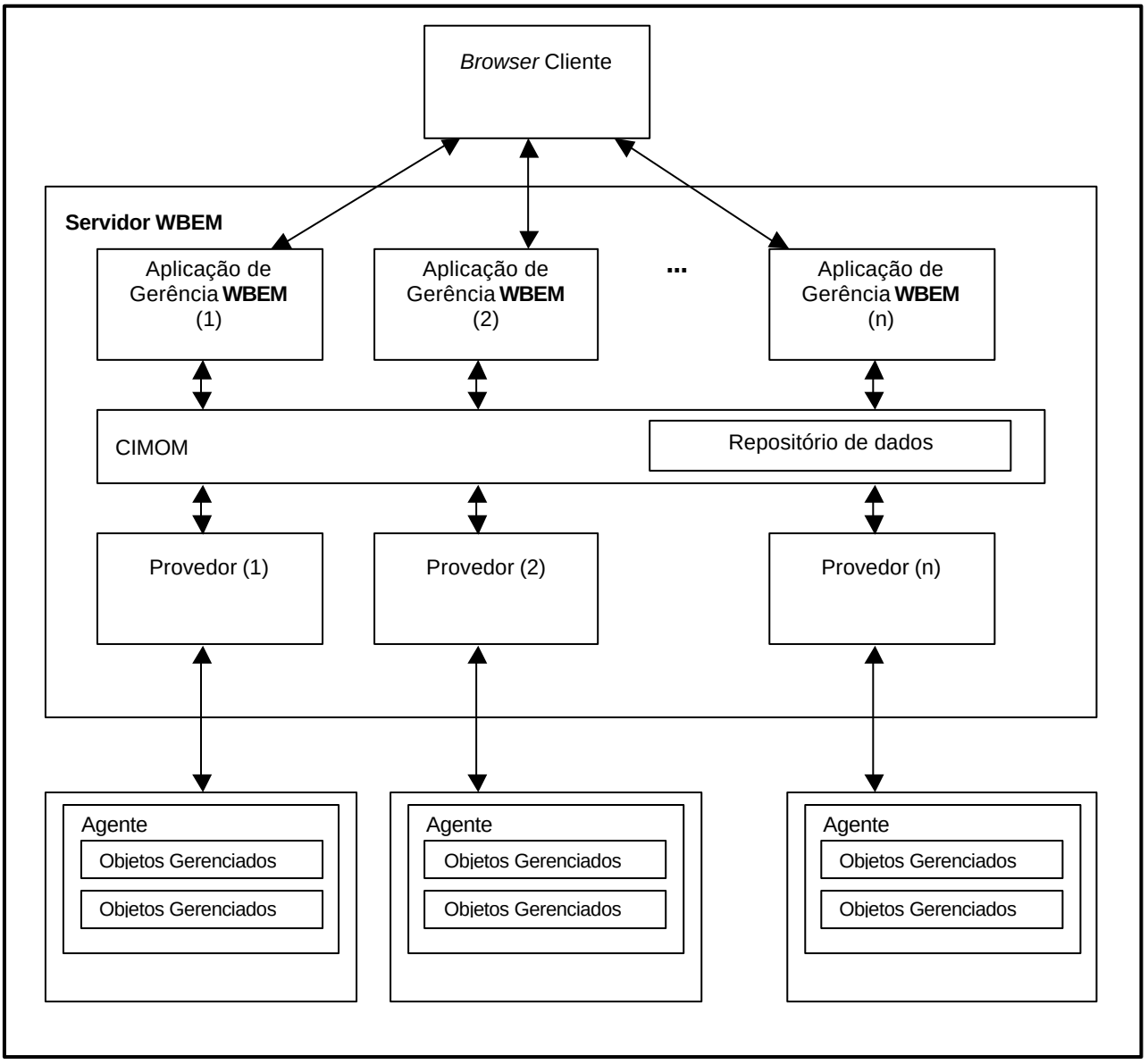
a) **provedores de propriedades**: retornam os valores de propriedade de objetos solicitados através de uma chave de identificação. Retorna informações em uma única instância;

b) **provedores de instância**: retornam valores de instância de objetos. Retornam informações da instância em uma única classe;

c) **provedores de classe**: retornam classes e instâncias. Controlam todas as classes;

d) **provedores de *namespace*:** são capazes de gerenciar um espaço de nome CIMOM pré-definido. Controlam todos os espaços de nomes e instâncias de objetos.

Figura 3.2 – Modelo completo do padrão WBEM



3.1 COMPONENTES DA ARQUITETURA WBEM

Neste capítulo esta descrita a arquitetura do padrão WBEM.

3.1.1 CIM

O CIM define o modelo utilizado para representar os objetos gerenciados do mundo real através do paradigma de orientação a objetos, onde os objetos gerenciados são modelados usando os conceitos de classes e instâncias (CASTRO, 2002). Segundo Carvilhe (2000) e conforme detalhado em DMTF (1999) o CIM foi projetado para receber informações de agentes SNMP, CMIP, entre outros, permitindo que aplicações corporativas de diferentes desenvolvedores, que utilizam plataformas heterogêneas, descrevam e compartilhem todos os objetos de gerência. A intenção é criar aplicações de gerência corporativa e ferramentas que possam monitorar e controlar todas as redes, sistemas e aplicações existentes numa empresa.

O CIM é composto por um meta esquema (*CIM Meta Schema*) e por esquemas padrões (*CIM Schemas*). Existe ainda um arquivo texto ASCII que utiliza a linguagem *Managed Object Format* (MOF) que contém as definições dos esquemas padrões implementados (CARVILHE, 2000).

3.1.1.1 CIM META SCHEMA

O meta esquema corresponde à definição formal do modelo. Define os termos que serão utilizados para expressar o modelo, incluindo a maneira apropriada de usar estes termos e sua semântica (DMTF, 1999). Os principais elementos são (DMTF, 1999 e CARVILHE, 2000):

a) **esquema**: é definido como um conjunto de classes com um único proprietário, utilizado para a nomeação e administração de classes. Dentro de um esquema os nomes de classes devem ser únicos;
b) **classe**: é um conjunto de instâncias do mesmo tipo, com as mesmas propriedades e métodos. A classe é a unidade básica de definição da estrutura de gerenciamento. Uma classe pode ser vista como um formulário para definição de objetos;

c) **propriedades**: valor utilizado para caracterizar uma instância de uma classe. A propriedade pode ser vista como um casal de funções *get* e *set* que quando aplicadas ao objeto retornam e definem estados deste objeto;

d) **método**: define o tipo de conteúdo que pode ser aplicado sobre uma classe ou instância. A definição de um método é uma assinatura, que contém o nome do método, tipo de retorno e parâmetros;

e) **evento ou *trigger***: é o reconhecimento de algum evento, mudança de estado ou alteração de alguma propriedade de um objeto;

f) **indicação**: são objetos criados como resultado de um evento. As indicações são tipos de classes, possuem propriedades, métodos e podem ser organizadas em hierarquia de tipos;

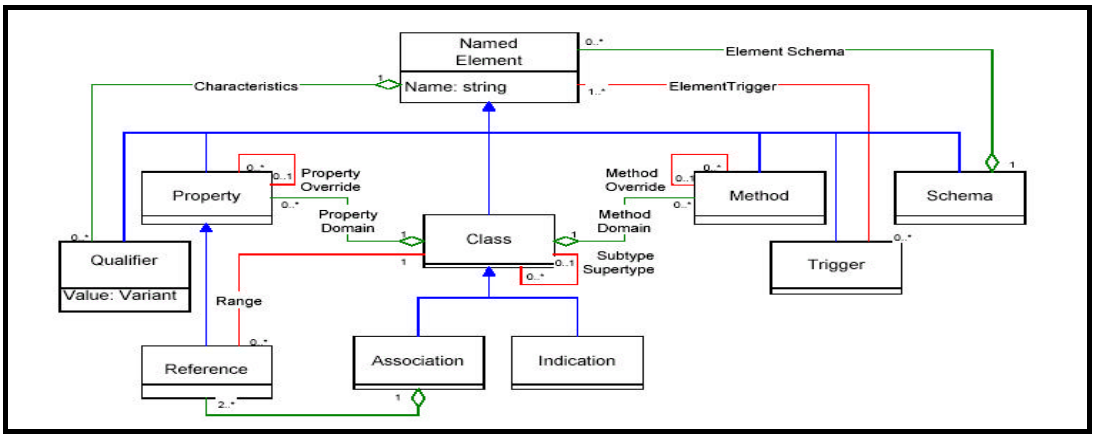
g) **referência**: são propriedades que uma classe ou instância possui. O valor é um ponteiro para um objeto;

h) **associação**: é um tipo de classe que possui uma ou mais referências. São utilizadas para criar relacionamentos entre objetos sem necessidade de alterar as suas definições;

i) **qualificador**: é utilizado para caracterizar classes, instâncias ou métodos. Permite a extensão do esquema de forma limitada e controlada. Por exemplo, existem qualificadores para definir as características de uma propriedade ou de uma chave de uma classe.

A fig. 3.3 demonstra a estrutura do meta esquema. O meta esquema completo é definido em um arquivo MOF e segue diversas regras detalhadas pelo DMTF (1999).

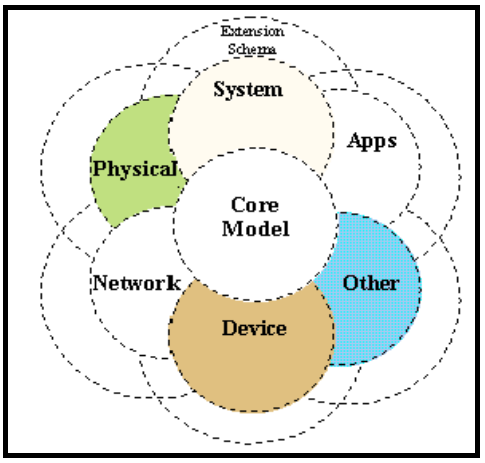
Figura 3.3 – Estrutura do CIM Meta Schema



3.1.1.2 CIM SCHEMA

Segundo Carvilhe (2000), o esquema padrão corresponde a um conjunto padrão de classes com suas propriedades e associações. Estas propriedades e associações fornecem a base necessária para a organização que irá utilizar o padrão WBEM. O esquema padrão foi definido com o objetivo de prover a uniformidade dos procedimentos de requisição sobre a rede de gerenciamento. O esquema padrão (CIM Schema) é composto pelo *Core Model*, *Common Model* e *Extesion Schemas.* A fig. 3.4 representa as diversas camadas do esquema padrão.

Figura 3.4 – Camadas do CIM Schema



O *Core Model* é constituído por um conjunto de objetos que são referenciados por todas as áreas de gerência. Portanto, o *core model* é um conjunto relativamente pequeno de classes, associações e propriedades que provêem o vocabulário básico para a análise e descrição de sistemas gerenciados, sendo o ponto inicial para o estudo de como determinar as extensões para domínios específicos (MICROSOFT, 1999).

O C*ore Model* é constituído das seguintes classes (CARVILHE, 2000):

a) ***Managed System Element***: classe base para os elementos dos sistemas gerenciados. Todos os componentes serão instanciados como uma subclasse desta classe;
b) ***Physical Elemen****t*: classe correspondente aos elementos físicos (exemplos: drivers de disco e adaptadores);
c) ***Logical Elemen****t*: classe referente aos elementos que não são físicos (exemplos: sistemas operacionais e drivers de dispositivos);

d) ***System***: classe correspondente à identificação do sistema, como *PC, hub* ou *router*;
e) ***Network Component***: corresponde a representação lógica do dispositivo de rede física, como um ícone num mapa representando um servidor;
f) ***Service***: classes que representam as capacidades e funcionalidades de um elemento do sistema;
g) ***Physical Package***: classes que representam o conteúdo físico de cada dispositivo (exemplos: um switch possui uma placa, esta placa possui um processador e assim por diante).

Além das classes, o *Core Model* é constituído de várias associações importantes (CASTRO, 2002):

a) ***Component***: relaciona partes de objetos com grupo de objetos;
b) ***System Component***: relacionam um sistema a seus elementos de sistema gerenciados;
c) ***Dependency***: relaciona as dependências existentes e dependências funcionais;
d) ***Contains***: relaciona a posição e localização de componentes físicos para expressar relações de conteúdo.
e) ***Hosted Service***: associação entre serviço e sistema que representam suas funcionalidades;
f) ***HostedAccessPoint***: associação entre *ServiceAccessPoint* e o *System*, permitindo modelar tanto sistemas distribuídos quanto acesso distribuído;
g) ***ServiceSAPDependency***: associação entre *Service* e *ServiceAccessPoint* indicando que o SAP referenciado é requerido no *Service* para o seu funcionamento;
h) ***ServiceAccessBySAP***: esta associação identifica os pontos de acesso para um serviço. Por exemplo, uma impressora pode ser acessada por um sistema *Netware* ou *Apple Macintosh* ou *Windows.*

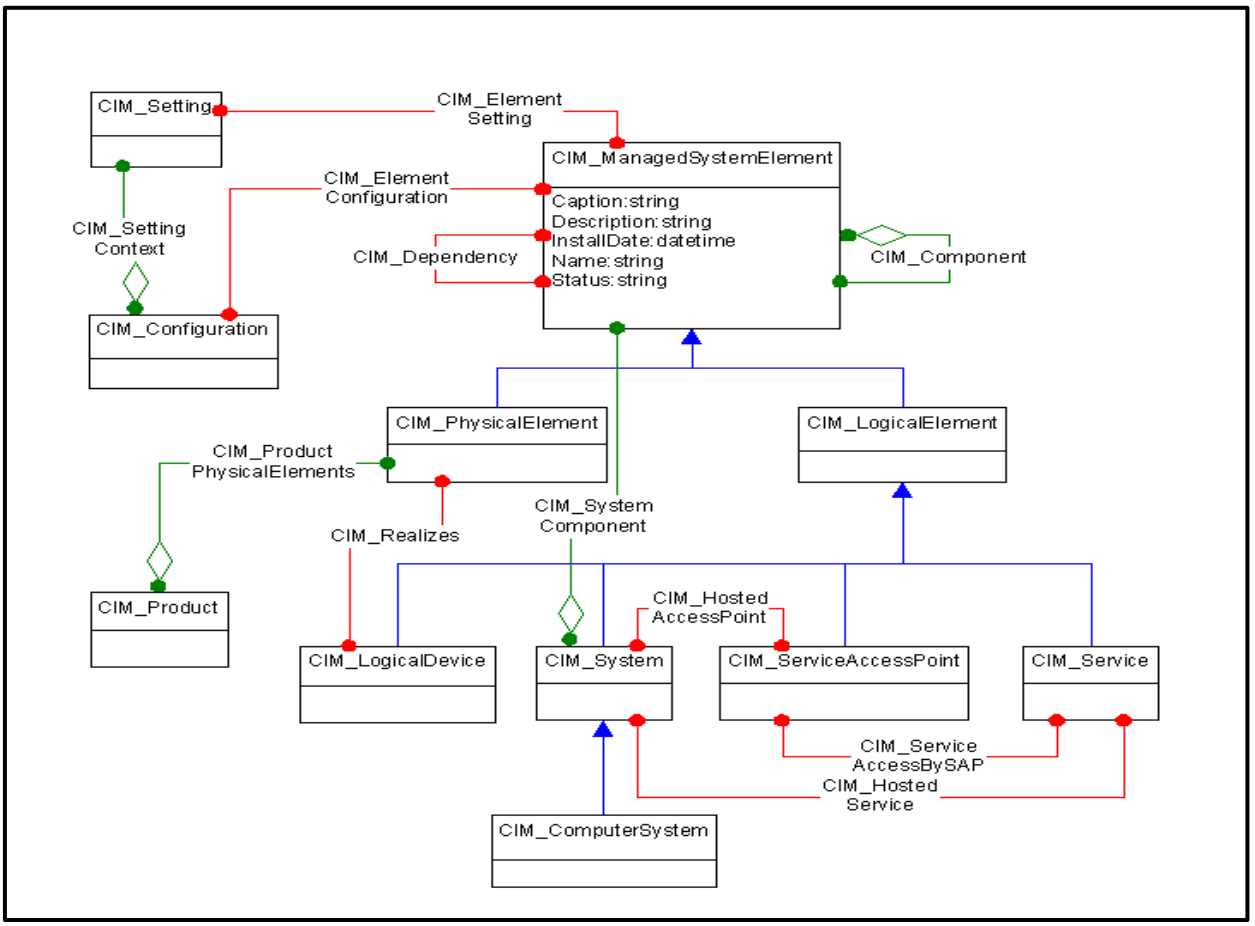
O *Common Model* modela informações de uma área de gerenciamento específica, mas independente de plataforma, tecnologia ou implementação. Este modelo possui uma especificação suficiente para suprir o mínimo necessário no desenvolvimento de aplicações de gerência. Estas áreas de gerenciamento incluem (MICROSOFT, 1999):

a) ***System:*** este modelo descreve os vários níveis dos objetos de sistemas que fazem parte do ambiente de gerenciamento, sendo que estes são representados por vários tipos de sistemas de computadores, vários tipos de aplicações e sistemas de redes;

b) ***Devices***: representação lógica através de unidades discretas que disponibilizam uma compatibilidade básica com o sistema, como armazenamento, processamento, funções de entrada e saída, componentes de interface com o usuário e acesso direto à memória. Este modelo contém as classes que representam os dispositivos que integram os componentes físicos do sistema, ou melhor, as que operam sobre a representação física dos dispositivos do sistema (CASTRO, 2002);

A fig. 3.5 representa o *Core Model.*

Figura 3.5 – CIM Core Model



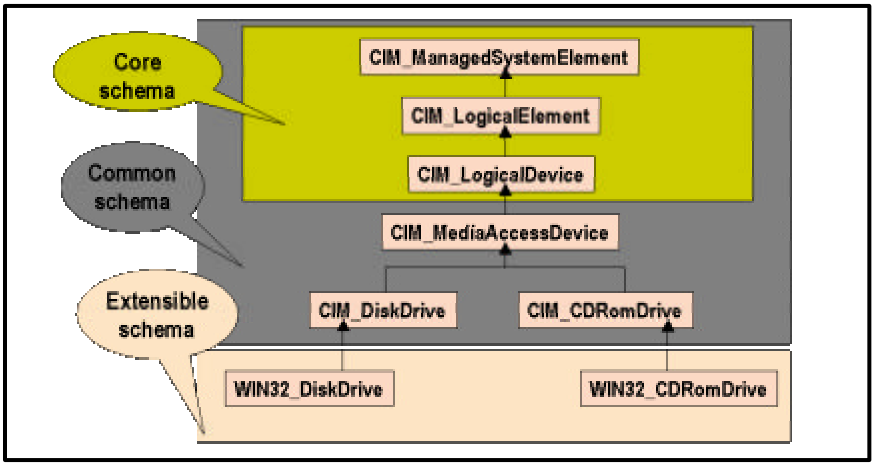
c) ***Networks***: este modelo representa os vários modelos de rede, incluindo topologias, conectividade e acesso fornecido por vários protocolos e serviços necessários para prover acesso de rede (MICROSOFT, 1999);

d) ***Applications***: modela as informações que descrevem e que geralmente são usadas e requeridas por *softwares* de gerência. Podendo ser usados por diversos tipos de aplicações desde *standalone*, distribuídas, internet e muitas outras. A interpretação e as características de vários objetos usados para representar aplicações estão amplamente relacionadas com mecanismos para transformar aplicações de um estado para outro. Uma característica marcante é a idéia do ciclo de vida da aplicação que pode ter os seguintes estados: distribuível, instalável, executável ou executando;

e) ***Physical***: modela o ambiente físico atual, representado por objetos lógicos. A maior parte do gerenciamento de sistemas está relacionado com a manipulação de informações que controlam e representam o estado do sistema. Para o CIM, o modelo *physical* é uma representação do aspecto do ambiente. Vale ressaltar que o modelo físico é uma representação lógica do ambiente envolvido e que é inevitável que se diferencie a dependência do ambiente do sistema e da tecnologia envolvida, ou de qualquer outro fator que possa vir a alterar a sua característica, exigindo uma modelagem mais específica para cada situação (CASTRO, 2002).

As extensões do esquema (*Extensions Schema*) permitem o gerenciamento de ambientes e plataformas específicas. Os exemplos incluem o desenvolvimento de extensões de esquema para Sistemas Operacionais *Unix*, *Microsoft Windows*, acordo de níveis de serviço, entre outros. A fig. 3.6 demonstra com maior clareza os esquemas do CIM.

Figura 3.6 – CIM Schema



3.1.2 CIMOM

O CIMOM, conforme descrito anteriormente, é um gerenciador de objetos que fornece vários mecanismos para que as aplicações de gerência possam trocar dados com os objetos que estão sendo gerenciados. O CIMOM contém um modelo de dados cuja tarefa é consolidar e traduzir os dados de gerência provenientes de diferentes fontes (CARVILHE, 2000).

3.1.3 MOF – MANAGED OBJECT FORMAT

A MOF é uma linguagem criada pela DMTF baseada na *Interface Definition Language* (IDL). É utilizada para definir classes e instâncias estáticas ou dinâmicas em formato texto. A sintaxe desta linguagem é definida detalhadamente em DMTF (1999).

O arquivo .MOF é um arquivo texto ASCII que contém as definições dos esquemas padrões (*CIM Schemas*) implementados. O arquivo MOF serve de entrada para o compilador MOF de aplicações de gerência, que tem como objetivo criar as chamadas apropriadas para o CIMOM. O compilador MOF define as classes, propriedades e qualificadores para serem incluídos no banco de dados CIMOM, e também coloca as instâncias dessas classes neste mesmo banco de dados (CARVILHE, 2000). Um exemplo de um arquivo MOF pode ser visualizado no quadro 3.1.

Quadro 3.1 – Exemplo de arquivo MOF

```
//************************************************************
//* Class: Win32_WordTemplate
//* Derived from:
//************************************************************
[dynamic: ToInstance, provider("OffProv")]
class Win32_WordTemplate
{
      [key, read: ToInstance ToSubClass] string Name;
      [read: ToInstance ToSubClass] string Path;
      [read: ToInstance ToSubClass] string Type;
};
```

3.2 REPRESENTAÇÃO DO CIM EM XML

A linguagem MOF fornece uma representação textual de informações de gerenciamento modeladas usando o CIM. No entanto, essa representação sozinha não é o suficiente para transferir informações de gerenciamento em ambientes heterogêneos. Para

transferir informações de gerenciamento em ambientes heterogêneos, um mapeamento da representação para um protocolo de comunicação é necessário. Quando a DMTF ficou com a responsabilidade de desenvolver a arquitetura WBEM, como já mencionado anteriormente, o objetivo era de usar o protocolo HTTP para transporte das informações de objetos gerenciados. Isto levou à especificação de um mapeamento do CIM para XML.

A linguagem XML é um subconjunto da linguagem SGML (*Stantardized Generalized Markup Language*), usada para representar estruturas de dados, como as informações de gerenciamento, em forma textual. Em XML, um documento pode opcionalmente ter uma descrição da sua gramática em anexo. A gramática para um documento XML é descrita usando um mecanismo conhecido como *Document Type Definition* (DTD) que é responsável por descrever os elementos permitidos num documento XML. Um documento que é estruturado de acordo com as regras definidas na especificação XML é dito bem formado (*well formed*). Além de bem formado, um documento XML pode ser válido ou inválido. Um documento válido deve conter uma DTD, e a gramática do documento deve estar de acordo com o que foi especificado no DTD (TEIXEIRA, 1999).

Para que informações de gerenciamento possam utilizar a XML é necessário definir um vocabulário, um DTD, para representar as informações de classes e instâncias do CIM.

Documentos XML não necessariamente contêm informações sobre captura de dados. Isto pode ser alcançado com o uso de planilhas de estilo XSL (*Extesible Style Language*), que podem ser usadas tanto para captura de informações como para transformá-las em outros formatos. Qualquer número de planilha de estilo XSL pode ser associado com um documento XML. Por exemplo, planilhas de estilo XSL podem apresentar graficamente informações ou fornecer uma transformação para o formato MOF (TEIXEIRA, 1999).

Enfim, a definição de uma DTD para o CIM, junto com as capacidades do XSL, fornecem uma maneira de comunicar informações de gerenciamento do CIM em ambientes heterogêneos através do HTTP. A XSL pode ser usada como uma forma padrão de capturar informações de gerenciamento. Não é o intuito deste trabalho entrar em detalhes do funcionamento da XML, portanto, para maiores informações, consulte Moraes (1999).

3.3 IMPLEMENTAÇÕES DO WBEM

Existem diversas implementações do WBEM, entre elas pode-se citar, *WMI, WBEMServices, OpenPegasus, b4wbem* e *OpenWBEM*. A seguir será feito um breve comentário sobre cada uma destas implementações.

WMI: implementação proposta pela Microsoft, inclui um repositório de dados compatível com o CIM e o gerenciador de objetos CIMOM (MICROSOFT, 1999). No entanto, a comunicação entre uma aplicação gerente e o CIMOM é realizada utilizando a arquitetura de objetos distribuídos (DCOM) da Microsoft. Esta implementação será detalhada no próximo capítulo.

WBEMServices: implementação Java do WBEM em código aberto, seja para aplicações comerciais ou não comerciais. O projeto consiste de API, aplicações clientes, aplicações servidoras, além de algumas ferramentas. As API são baseadas na *Java Specification Request* (JSR) 48 e submetidas à aprovação da *Java Comunnity Process* (JCP) 2.0. Possui ferramentas como o CIMOM, repositório de dados com alguns arquivos MOF já adicionados, compilador MOFCOMP, entre outras e suporte aos protocolo HTTP e RMI para transferência de informação. Entre as desvantagens destaca-se a necessidade de implementação dos provedores, que suprem os dados da base de objetos (CASTRO, 2002).

Pegasus: implementação em C++ do WBEM desenvolvido em código aberto pelo *OpenGroup*. Abrange todas os componentes do WBEM. Portável para diversas plataformas, entre elas, *Microsoft Windows, Linux* e a maioria das versões *UNIX*. Distribuído sob licença MIT. Maiores informações podem ser obtidas em Open (2003).

OpenWBEM: implementação em código aberto do WBEM desenvolvida em C++ e com extensões em JAVA, comercial ou não comercial. Realiza operações do CIM sobre HTTP ou HTTPS, possui diversos módulos de autenticação, entre eles: *SIMPLE, PAM e Digest Autentication.* Possui um compilador MOF e uma biblioteca WQL para realizar buscas no repositório de dados. Maiores informações podem ser vistas em Center (2003).

Estes são apenas alguns exemplos de implementações do WBEM. No capítulo seguinte será detalhado o *Windows Management Instrumentation* (WMI).

3.4 WINDOWS MANAGEMENT INSTRUMENTATION

O WMI é a implementação do WBEM desenvolvida pela Microsoft. Esta possui uma infra-estrutura que suporta o modelo de dados CIM, extensões para a plataforma *Windows* e um gerenciador de objetos (CIMOM). Tem como objetivo estabelecer padrões para acessar e compartilhar informações de gerenciamento em uma rede corporativa, podendo controlar e monitorar os componentes do sistema, sejam *softwares* ou *hardwares* (CASTRO, 2002).

3.4.1 ARQUITETURA WMI

A arquitetura WMI, ilustrada na fig. 3.7, consiste dos seguintes elementos (MICROSOFT, 2001):

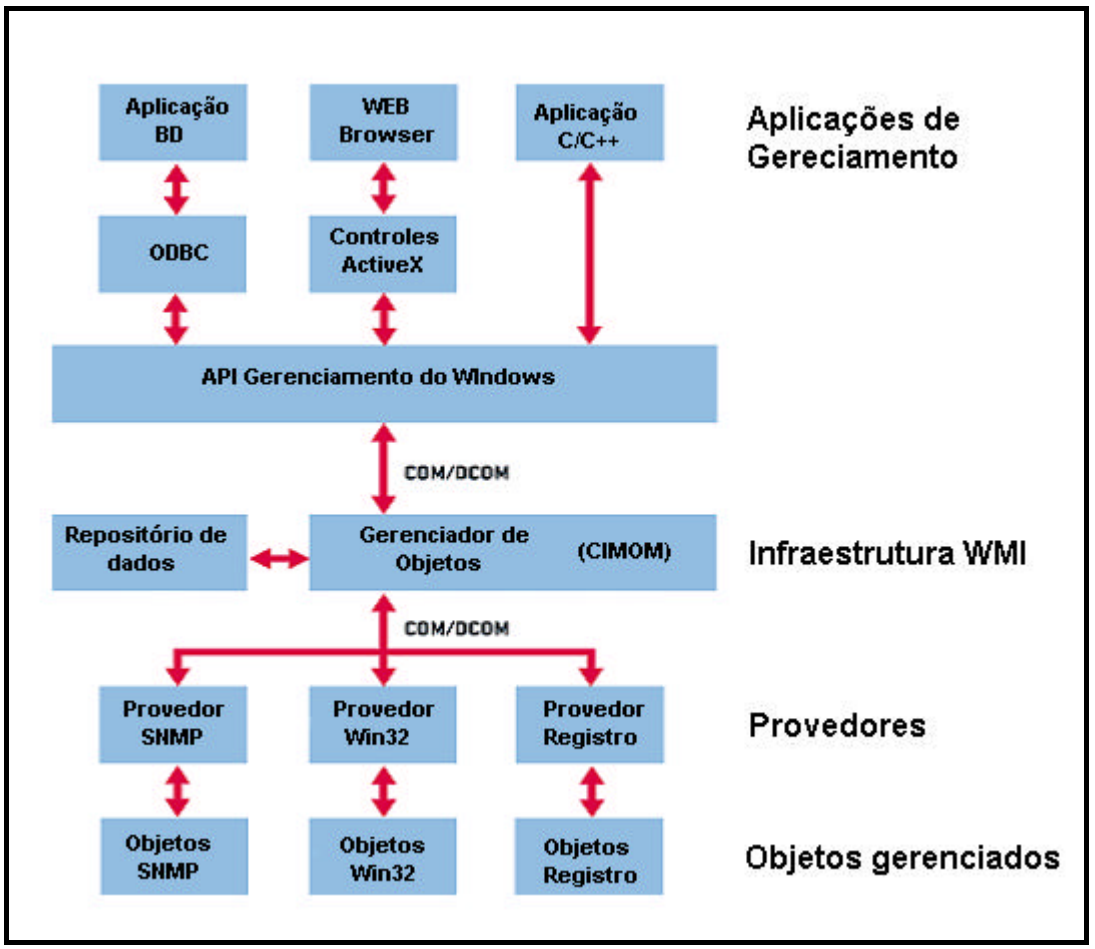
a) ***aplicações de gerenciamento:*** são aplicações que requisitam e processam informações de objetos gerenciados armazenados no repositório de dados. Estes dados são acessados através de uma requisição ao CIMOM utilizando métodos descritos na API do WMI;
b) ***infra-estrutura de gerenciamento***: inclui o CIMOM e o repositório de dados. O CIMOM é responsável pela ligação entre as aplicações de gerenciamento e os provedores de dados;
c) ***provedores***: atuam como mediadores entre o CIMOM e os objetos gerenciados. Quando o CIMOM recebe uma solicitação de uma aplicação de gerenciamento para um dado que não está presente no repositório ou de um evento que não é suportado, este encaminha a requisição para um provedor WMI. Provedores suprem dados e notificações de eventos dos objetos gerenciados específicos de seu domínio;
d) ***objetos gerenciados***: são os objetos físicos ou lógicos modelados através do CIM, que podem ser gerenciados. Por exemplo, um objeto pode ser tanto um *hardware*, como um cabo, quanto um *software*, como uma aplicação de banco de dados.

O processo executável que provê todas as funcionalidades do WMI chama-se *WinMgmt.exe*. O serviço WMI (*WinMgmt.exe*) é responsável por manipular solicitações de clientes, conversar com os provedores e gerenciar o repositório de dados. Este serviço é exposto aos programadores por intermédio de duas API, sendo que, para linguagem C/C++ é utilizada a API COM, que é o acesso de mais baixo nível ao WMI que se pode solicitar. Já

para linguagens de scripts que reconhecem automação, é utilizada a *API Scripting*, que é um empacotador (*wrapper*) em torno da API COM (MARTINSSON, 2002)

A comunicação entre os elementos da arquitetura WMI é realizada através da API *IWbemServices* COM/DCOM (MICROSOFT, 2001). O *framework* .NET provê um *namespace*, para programação em *C#*, com diversas classes implementadas que facilitam as requisições de informações ao serviço WMI. Este *namespace* chama-se *System.Management* e será utilizado para o desenvolvimento do protótipo, sendo exemplificado nos capítulos posteriores.

Figura 3.7 – Arquitetura WMI

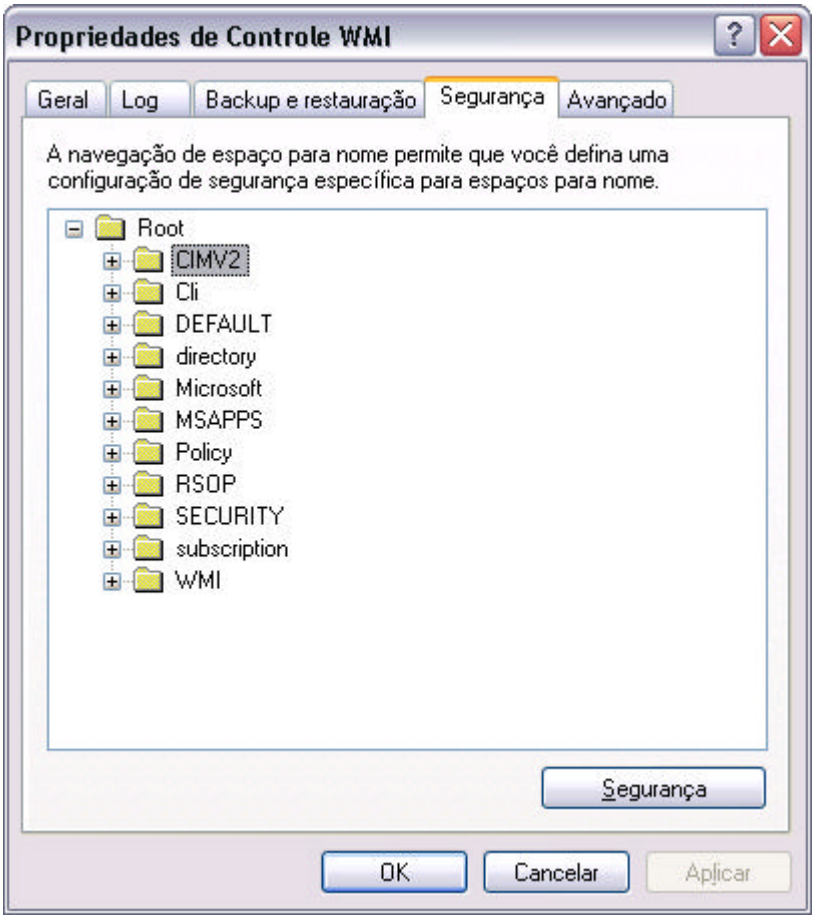


Existem diversos provedores implementados, entre os quais destacam-se (MICROSOFT, 2001):

a) ***Active Directory Provider***: atua como um *gateway* de toda a informação armazenada no serviço do Active Directory;
b) ***Windows Installer Provider*****:** permite controle total do *Windows Installer* e instalações de *softwares* através do WMI. Supre, também, informações sobre qualquer aplicação instalada pelo *Windows Installer*;
c) ***Performance Counter Provider*****:** expõe informações dos contadores de performance. Qualquer contador de performance instalado no sistema torna-se automaticamente visível através deste provedor;
d) ***Registry Provider*****:** permite que chaves de registro sejam criadas, lidas ou alteradas. Eventos podem ser gerados quando uma chave específica for alterada;
e) ***SNMP Provider*****:** atua como um *gateway* entre sistemas e dispositivos que utilizam o SNMP para gerenciamento. Objetos de uma MIB SNMP podem ser lidos ou escritos através deste provedor. *Traps* SNMP podem ser automaticamente mapeadas para eventos WMI;
f) ***Event Log Provider***: provê acesso aos dados de notificações do visualizador de eventos do *Windows 2000*;
g) ***Win32â Provider***: provê informações sobre o sistema operacional, periféricos, sistema de arquivos, entre outras informações relacionadas no esquema Win32.

O WMI foi implementado seguindo o conceito de *namespace*, onde uma aplicação de gerenciamento deve conectar-se a um *namespace* para obter acesso às propriedades dos objetos contidos neste *namespace*. A segurança pode ser implementada com base nestes espaços de nomes onde um administrador pode controlar quais usuários podem ter acesso a um determinado *namespace* (CASTRO, 2002). A fig. 3.8 ilustra os espaços de nomes contidos no WMI, onde é possível determinar sua segurança.

Figura 3.8 – Espaço para nomes do WMI



3.4.2 WMI QUERY LANGUAGE

A WQL (*WMI Query Language*) é uma linguagem de consultas ao WMI. Tem como objetivo enumerar, explorar e gerar relatórios sobre o andamento de um ambiente gerenciado (MARTINSSON, 2002). A WQL é um subconjunto do SQL (*Structured Query Language*, linguagem de consultas estruturada). Diferentemente do SQL tradicional, a WQL é uma linguagem de consultas que não foi concebida para atualizar, eliminar ou inserir dados. A WQL, segundo Martinsson (2002), suporta três tipos de consultas:

a) **consultas de dados**: usada pelos aplicativos de gerenciamento que têm de selecionar associações de dados e instâncias sobre instâncias;
b) **consultas de evento**: usado pelos aplicativos de gerenciamento que implementam a manipulação de eventos. Os eventos são disparados dentro de seu aplicativo de gerenciamento para notificá-lo de alterações como a criação, a eliminação ou a

modificação de um banco de dados. Essas consultas registram o aplicativo de gerenciamento em termos de notificações de eventos;

c) **consultas de esquema**: similar às consultas de dados, porém estas retornam metainformações em vez de instâncias.

Uma consulta dos processos que estão consumindo, por exemplo, mais do que 10MB de memória, utilizando a linguagem WQL pode ser visualizada no quadro 3.2.

Quadro 3.2 – Exemplo de consulta WQL

```
SELECT * FROM Win32_Process WHERE WorkingSetSize >= 10485760
```

Informações detalhadas sobre a construção de consultas utilizando a WQL podem ser encontradas em Martinsson (2002).

4 DESENVOLVIMENTO DO TRABALHO

O objetivo do trabalho foi desenvolver um protótipo para gerência de sistemas baseado no padrão WBEM utilizando o WMI. Neste capítulo serão abordados aspectos relevantes sobre o desenvolvimento deste protótipo tais como os principais requisitos, a especificação e a implementação.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA

Foram levantados alguns requisitos que devem estar presentes no protótipo. Estes requisitos demonstram algumas características que o protótipo precisa ter para que se alcance o resultado final desejado.

Os principais requisitos do protótipo são:

a) operar no sistema operacional *Windows 9x, NT ou 2000*;
b) disponibilizar a interface do protótipo através de ser um *browser* internet;
c) coletar informações de classes do repositório de dados do WMI, com a finalidade de realizar um pequeno inventário de informações de um microcomputador;
d) coletar informações de classes do repositório de dados do WMI que contenham informações dos processos e serviços que um computador está executando. No caso dos serviços, serão mostrados todos os serviços e seu respectivo estado;
e) aguardar e mostrar notificações de eventos, onde serão monitorados todos os novos processos criados numa estação.

4.2 ESPECIFICAÇÃO DO PROTÓTIPO

Para a especificação do protótipo foi utilizada a *Unifield Modeling Language* (UML), utilizando o diagrama de casos de uso, diagrama de classes e diagrama de seqüência. Para auxiliar na construção destes diagramas foi utilizada uma versão de avaliação da ferramenta *case* Enterprise Architect.

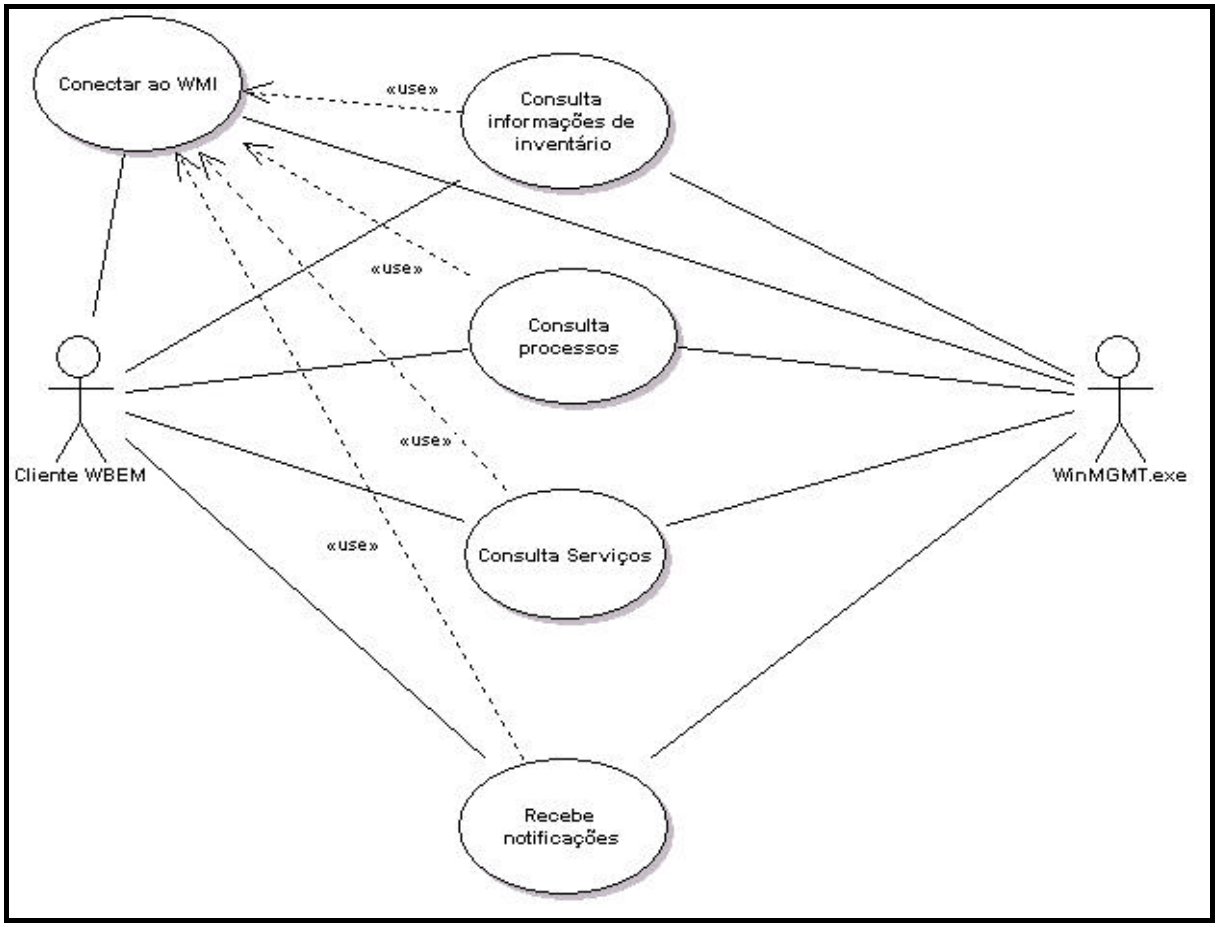
Para obter maiores informações sobre orientação a objetos usando a UML consulte Larman (2000) e Matos (2002). Já para informações sobre a ferramenta *case* Enterprise Architect consulte Sparx (2003).

4.2.1 DIAGRAMA DE CASOS DE USO

Na fig. 4.1 está apresentado o diagrama de casos de uso do protótipo. Estes casos de uso identificam como o usuário interage com o sistema. Os principais casos de uso, descritos detalhadamente no diagrama de sequência desta especificação, são:

a) conexão ao serviço WMI;
b) consulta de informações para realização de um pequeno inventário;
c) consulta dos processos
d) consulta dos serviços;
e) recebimento de notificações.

Figura 4.1 – Diagrama de Casos de Uso

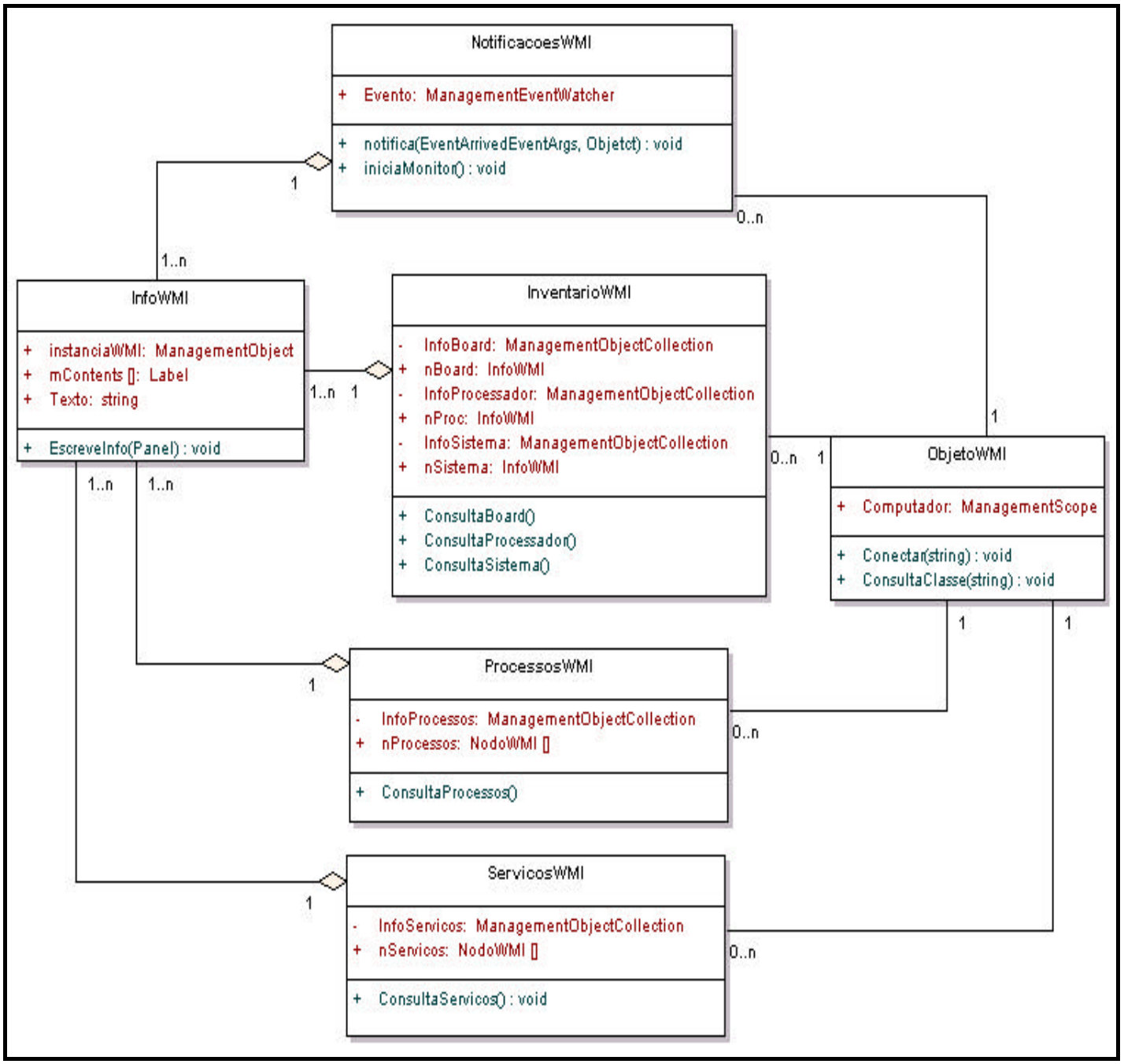


4.2.2 DIAGRAMA DE CLASSES

A fig. 4.2 representa o diagrama de classes. Conforme os requisitos do sistema, foram identificadas as seguintes classes:

a) ***InfoWMI***: classe que armazena informações extraídas do repositório de dados, que são armazenadas em atributos e servem como um *cache* de dados;
b) ***ObjetoWMI***: classe que contém as configurações de acesso ao serviço WMI, bem como um método para conexão e outro para consulta de classes que retorna uma lista de objetos gerenciáveis pertencentes à classe consultada, como por exemplo, quando se consulta a classe *"Win32_Process"*, tem-se como retorno uma lista de instâncias de objetos que representam os processos em execução de um computador;
c) ***InventarioWMI***: contém atributos e métodos para acessar informações do computador a fim de realizar um pequeno inventário. Esta classe recebe como parâmetro para inicialização um *ObjetoWMI*, pois não é necessário re-conectar ao serviço sempre que preciso;
d) ***ProcessosWMI***: classe que possui atributos e métodos para consulta de processos em execução num determinado computador. Recebe como parâmetro um *ObjetoWMI*;
e) ***ServicosWMI***: contém atributos e métodos para consulta de serviços disponíveis, bem como seu estado, tipo e modo de incialização (automático ou manual). Recebe como parâmetro um *ObjetoWMI*;
f) ***NotificacoesWMI***: classe que contém métodos e atributos para iniciar um monitoramento de uma instância presente no repositório de dados do WMI. Para realizar o monitoramento é necessário atribuir ao monitor uma consulta WQL, que pode representar uma criação, remoção ou alteração de instâncias das diversas classes presentes no repositório de dados. Recebe como parâmetro de inicialização um objeto do tipo *ObjetoWMI*.

Figura 4.2 – Diagrama de Classes

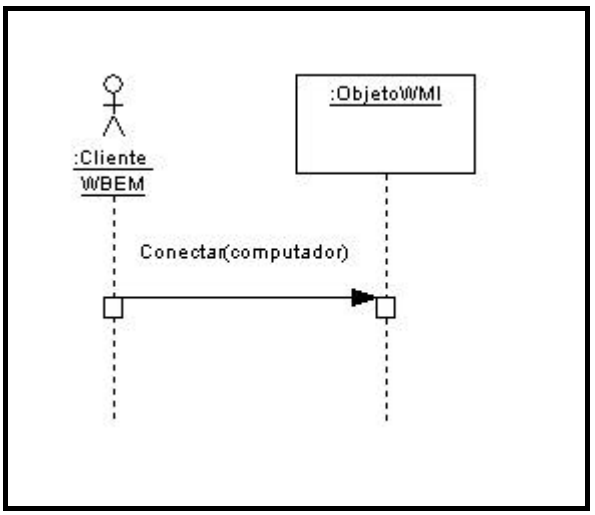


4.2.3 DIAGRAMAS DE SEQUÊNCIA

Nos diagramas de seqüência pode-se observar a execução dos casos de uso e verificar as mensagens trocadas entre os objetos. Os principais diagramas de seqüência encontrados no sistema são:

a) conectar ao serviço WMI: para realizar as consultas ao WMI é necessária uma conexão. O cliente WBEM passa por parâmetro uma *string* contendo o nome do computador (*host*) ou endereço IP ao qual deseja conectar-se. Neste caso não estão sendo considerados aspectos relacionados à segurança, como autenticação, apenas é feita à conexão de acordo com a solicitação do usuário. A fig. 4.3 representa o diagrama de seqüência responsável pela conexão ao serviço WMI;

Figura 4.3 – Conexão ao serviço WMI



b) consulta de informações de inventário: para a execução deste caso de uso, subentende-se que o cliente WBEM já esteja conectado, conforme descrito no diagrama de casos de uso. Para realizar o inventário, o cliente WBEM chama os métodos *ConsultaBoard()* para obter informações relacionadas à placa mãe, *ConsultaProcessador()* para obter informações relacionadas ao processador e *ConsultaSistema()* para obter informações pertinentes do sistema operacional. A fig. 4.4 representa o caso de uso para realização de um pequeno inventário. Após a chamada de qualquer um destes métodos é executado um outro método chamado *ConsultaClasse()*. Este método é responsável pela consulta sobre a classe desejada ao serviço WMI e está implementado na classe *ObjetoWMI*;

c) consulta de processos: neste diagrama pode-se observar a execução da consulta de processos em execução de um computador através do WMI. O cliente WBEM solicita estas informações através do método *ConsultaProcessos()*. A fig. 4.5 representa este caso de uso;

Figura 4.4 – Consulta de informações de inventário

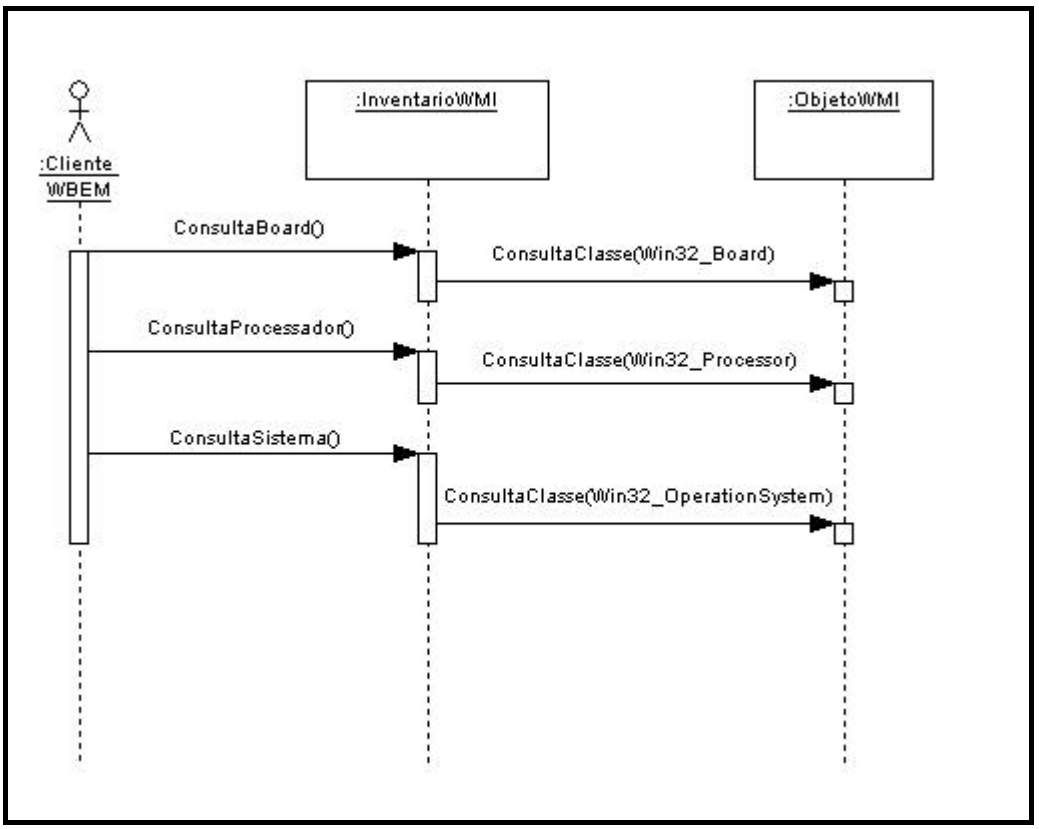
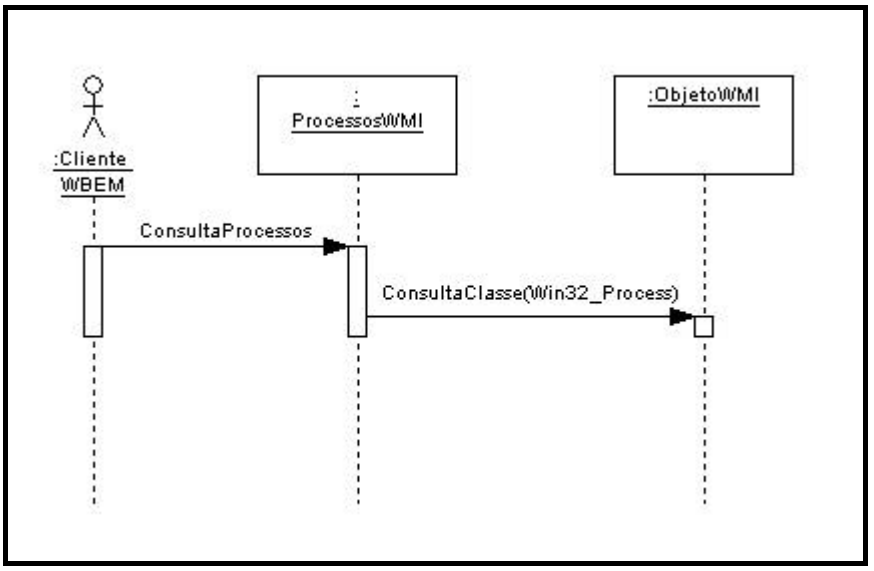


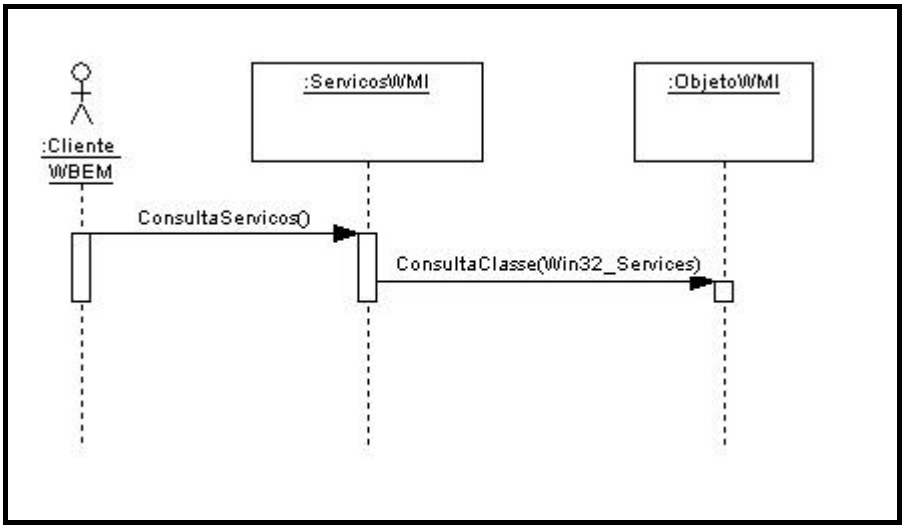
Figura 4.5 – Consulta de processos



d) consulta de serviços: neste caso de uso o cliente WBEM consulta informações dos serviços disponíveis em um computador. Como retorno recebe uma lista com todos os serviços, sendo que estes não necessariamente precisam estar em execução, pois

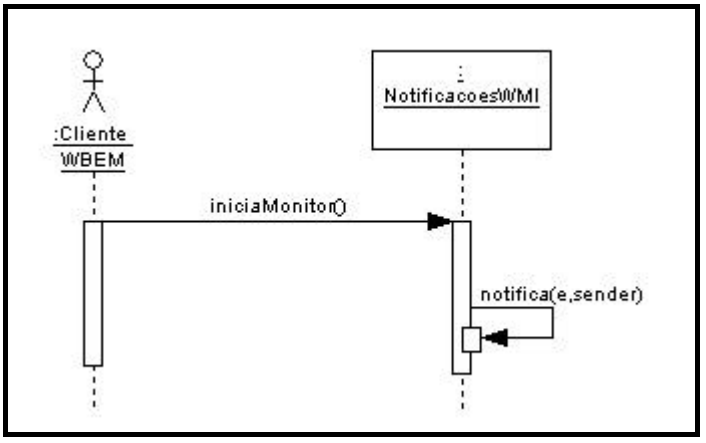
é pertinente que o cliente saiba, também, quais os serviços que estão parados. O método que é executado chama-se *ConsultaServicos().* A fig. 4.6 representa este caso de uso;

Figura 4.6 – Consulta de serviços



e) recebe notificações: este diagrama demonstra como o cliente WBEM irá receber notificações do serviço WMI. Para receber estas notificações é necessário criar um monitor (*watcher*), este monitor é criado e iniciado através do método *IniciaMonitor().* O monitor é adicionado ao serviço WMI e no momento em que forem registrados eventos pertinentes a este é criada uma notificação que retorna ao cliente WBEM. A fig. 4.7 representa este caso de uso.

Figura 4.7 – Recebe notificações



4.3 IMPLEMENTAÇÃO

Nesta etapa é descrita a implementação do protótipo conforme a especificação desenvolvida anteriormente.

4.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

O WMI, conforme mencionado anteriormente, utiliza em sua comunicação objetos distribuídos COM/DCOM, diferentemente da proposta do WBEM que é a utilização do protocolo HTTP. Esta comunicação foge do conceito fundamental que envolve a tecnologia de gerenciamento via WEB, onde as tarefas de gerenciamento são realizadas através de um *browser*.

Para solucionar este problema e ter um protótipo realmente para WEB, no trabalho correlato de Castro (2002) é sugerida a implementação de um *gateway* que atuará como um canal entre o serviço WMI e a aplicação gerente, transmitindo informações de gerência usando o protocolo HTTP.

Neste trabalho foi adotada uma outra solução. Foi utilizado o *namespace System.Management*, que é um conjunto de classes implementadas pela Microsoft para interagir com o serviço WMI, incluso na plataforma .NET, pois a utilização deste abstrai a necessidade de conhecimentos sobre objetos distribuídos (COM/DCOM) facilitando o desenvolvimento de aplicações de gerência sobre o WMI. Maiores informações sobre este *namespace* podem ser obtidas em Pozen (2002).

Um outro conceito incluso na plataforma .NET é a criação de aplicações WEB com ASP.NET (w*eb application*), onde estas aplicações possuem uma interface voltada para a WEB (*browser*).

Utilizando o *System.Management* com intuito de implementar as funções pertinentes ao gerenciamento sobre o WMI em conjunto com uma aplicação WEB é possível desenvolver um protótipo de gerenciamento via WEB, onde a interface com o usuário é um *browser*, ou seja, surge o protocolo HTTP. Portanto, ao invés de implementar um *gateway*, proposto por Castro (2002), utiliza-se os recursos disponíveis na plataforma .NET. Para obter maiores

informações sobre aplicações WEB utilizando a plataforma .NET consulte bibliografias da área, tais como Mitchell (2001).

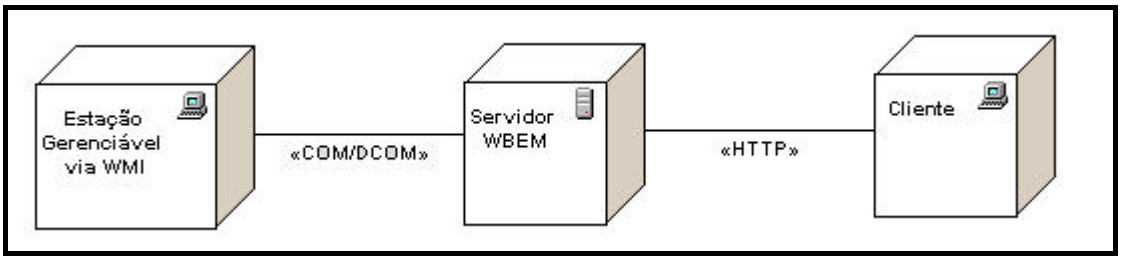
Para o desenvolvimento deste protótipo foram utilizados o ambiente de desenvolvimento Visual Studio .Net Academic, *framework* .NET, linguagem de programação C#, ASP.NET e o sistema operacional Windows 2000 Professional. Sendo que para resolver o problema da comunicação foi utilizado o conceito de aplicação WEB com ASP.NET, conforme justificado anteriormente. Além destas ferramentas também foram utilizados:

a) ***Internet Information Service* (IIS)**: servidor de páginas WEB da Microsoft, incluso no Windows 2000 Professional. Este servidor foi utilizado para hospedar a aplicação WEB;
b) ***IE Web Controls***: conjunto de componentes ASP.NET adicionais para o desenvolvimento de aplicações WEB, que inclui *TreeView* e *TabStrip*, sendo que estes foram utilizados no desenvolvimento do protótipo. Pode ser encontrado em Microsoft (2003);
c) ***MyListView***: componente ASP.NET que cria uma lista de itens no browser do cliente. Pode ser encontrado, com seu código fonte incluso, em Funkelab (2003).

O protótipo foi executado na rede administrativa da FURB, onde as estações gerenciadas estavam executando o serviço WMI. Este serviço, é importante frisar, vem disponível de forma nativa nos sistemas operacionais *Windows 2000* e *Windows XP*. Sendo que para gerenciar estações com sistemas antigos, tais como *Windows 9x, Windows ME e Windows NT 4,* é necessário instalar o *core* do WMI e além disto que o sistema suporte comunicação através de objetos distribuídos sendo que este esteja configurado para permitir o acesso remoto. Para obter informações sobre como proceder para instalar e configurar o serviço WMI consulte Renegative (2002).

O diagrama de *deployment* apresentado na figura 4.8 demonstra a topologia do sistema, possibilitando a visualização da distribuição física de seus componentes. O servidor WBEM é acessado de qualquer ponto da rede ou internet através do protocolo HTTP. Este é responsável por acessar as informações de uma estação gerenciável através do WMI e reportar para o cliente.

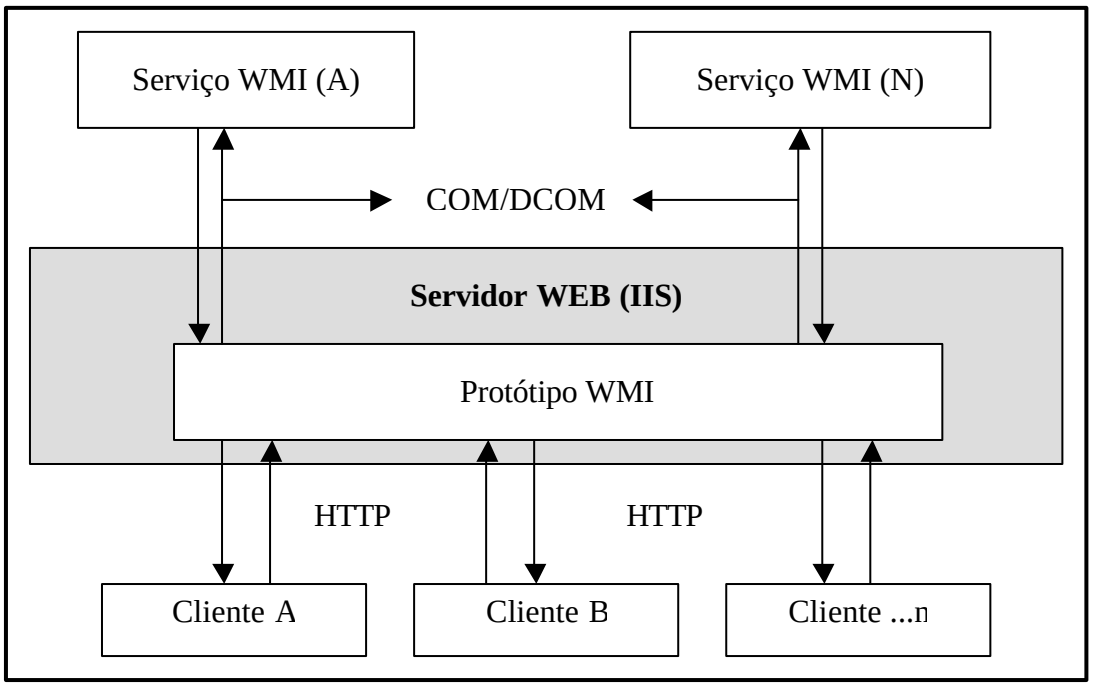
Figura 4.8 – Diagrama de *deployment*



A configuração de acesso remoto, bem como a segurança de uma forma geral, não foram prioridades neste trabalho. O acesso remoto foi configurado para usuários específicos do domínio da rede administrativa da FURB com controle total. Portanto a autenticação é feita através do usuário que efetuou o *logon* na estação cliente, sendo que, qualquer computador ligado a esta rede pode ser um cliente WBEM, desde que enxergue o servidor no qual está hospedada a aplicação do protótipo.

A fig. 4.9 demonstra a arquitetura do protótipo e a interação entre o cliente WBEM que irá executar a aplicação de qualquer ponto da rede ou internet, desde que o servidor WEB esteja configurado para tal, através de um *browser.*

Figura 4.9 – Arquitetura do Protótipo



Na estação hospeda o protótipo são necessários alguns recursos adicionais, além do servidor WEB (*IIS*), para que este suporte o protótipo desenvolvido. Estes recursos são:

framework .NET, apenas o *runtime*, e *IE Web Controls*. Já para a estação cliente, que vai acessar a aplicação do protótipo através de um *browser*, não é necessário nenhuma configuração adicional, apenas é recomendado que seja utilizado o navegador *Internet Explorer 5.0* ou superior.

O quadro 4.1 contém o código para conexão de um computador a um serviço WMI. Este método está implementado na classe *ObjetoWMI* e chama-se *Conectar()*. Neste quadro pode-se perceber que é necessário passar como parâmetro uma *string* que representa o nome (*host*) ou endereço IP do computador que desejamos conectar.

No quadro 4.2 pode-se visualizar o código do método para consulta de classes *ConsultaClasse()* ao serviço WMI. Este é um método muito importante, já que todos os demais métodos de consulta irão executá-lo. Como parâmetro receberá uma *string* que representa a classe que a ser consultada e retorna uma lista de objetos gerenciáveis pertinentes a classe consultada.

Quadro 4.1 – Método para conexão ao serviço WMI

```
public void Conectar(string comp)
{
    try
    {
         ConnectionOptions oConn = new ConnectionOptions();
         this.Computador = new ManagementScope("\\\\" + comp + "\\root\\cimv2", oConn);
         this.Computador.Options.Impersonation = ImpersonationLevel.Impersonate;
         this.Computador.Connect();
    }
    catch (Exception e)
    {
         MessageBox.Show("Servidor não encontrado ou sem permissão de acesso!","Erro");
    }
}
```

Quadro 4.2 – Método para consulta de classes

```
public ManagementObjectSearcher ConsultaClasse(string Classe)
{
 if (Computador.IsConnected)
 {
   try
   {
     ObjectQuery oQuery = new ObjectQuery("select * from " + Classe);
     ManagementObjectSearcher lista = new ManagementObjectSearcher(Computador,oQuery);
     return(lista);
   }
   catch (Exception e) { MessageBox.Show(e.ToString(),"ERRO"); return(null); }
 } else { return(null) }
}
```

No quadro 4.3 pode-se visualizar o método *ConsultaProcessador()* implementado na classe *InventarioWMI*, sendo que é necessário, ao criar-se o objeto do tipo *InventarioWMI*, passar por parâmetro um objeto do tipo *ObjetoWMI*, que contêm informações de conexão e o método *ConsultaClasse()*. Este parâmetro é necessário para não ter a necessidade de conectar ao serviço WMI toda vez que realizar as consultas. Os outros métodos seguem o mesmo funcionamento, apenas mudando o nome da classe a ser consultada.

Quadro 4.3 – Método para consulta de informações do processador

```
public void ConsultaProcessador()
{
   // informações do processador
   ManagementObjectSearcher  oProc = this.wmi.ConsultaClasse("Win32_Processor");
   this.InfoProcessador = oProc.Get();
   try
   {
       foreach (ManagementObject oReturn in this.InfoProcessador)
       {
              nProc = new InfoWMI(oReturn);
              for (int i = 0; i <=11; i++)
              {
                     nProc.mContents[i] = new Label();
              }
                     nProc.mContents[0].Text = "Fabricante:";
                     nProc.mContents[1].Text = oReturn["Manufacturer"].ToString();
                     nProc.mContents[2].Text = "Modelo:";
                     nProc.mContents[3].Text = oReturn["Name"].ToString();
                     nProc.mContents[4].Text = "Clock:";
                     nProc.mContents[5].Text = oReturn["CurrentClockSpeed"].ToString();
                     nProc.mContents[6].Text = "Socket:";
                     nProc.mContents[7].Text = oReturn["SocketDesignation"].ToString();
                     nProc.mContents[8].Text = "Descrição:";
                     nProc.mContents[9].Text = oReturn["Description"].ToString();
       }
   }
   catch (Exception e) {MessageBox.Show(e.ToString(),"ERRO");
   }

}
```

Analisando o código acima pode-se perceber que estão sendo armazenadas as informações em objetos do tipo *Label*. Funciona desta maneira para poder ter um *cache* das informações e não necessitar realizar a consulta novamente. Para cada instância de objeto relacionada ao processador da máquina (“*Win32_Processor*”) está sendo criado um objeto do tipo *InfoWMI* que irá armazenar os dados para um eventual re-uso.

Todos os outros métodos para consulta de classes seguem o mesmo funcionamento: é feita a consulta e são criados objetos do tipo *InfoWMI*. A diferença está na classe que é

realizada a consulta e nos atributos que estão sendo armazenados, pois nem todos os atributos são relevantes.

No quadro 4.4 pode-se visualizar a implementação da classe *InfoWMI*. Nesta classe tem-se os atributos, um construtor e um método *EscreveInfo()*. Este método é responsável pela escrita das informações armazenadas sobre determinado objeto em uma instância do tipo *Panel*, que é passada por parâmetro.

Quadro 4.4 – Classe InfoWMI

```
public class InfoWMI
{
   public System.Web.UI.WebControls.Label [] mContents = new Label[20];
   public ManagementObject instanciaWMI;
   public string Texto = "";
// construtor
   public InfoWMI (ManagementObject o)
   {
      this.instanciaWMI = o;
   }
// adiciona itens ao painel informado
   public void EscreveInfo(Panel painel)
   {
      bool tipo = true;
      painel.Controls.Clear();
      int sPoint = 15, ePoint = 15;
      for (int i = 0; i < mContents.Length; ++i)
      {
        if (mContents[i] != null)
        {
          if (tipo)
          {
             mContents[i].Style.Clear();
             mContents[i].Font.Name = "Courier New";
             mContents[i].Style.Add("LEFT", sPoint.ToString());
             mContents[i].Style.Add("TOP", ePoint.ToString());
             mContents[i].Style.Add("POSITION","absolute");
             mContents[i].Font.Size = FontUnit.Parse("12");
             mContents[i].Font.Bold = true;
             tipo = false;
          }
          else
          {
              mContents[i].Style.Clear();
              mContents[i].Style.Add("LEFT", (sPoint+120).ToString());
              mContents[i].Style.Add("TOP", ePoint.ToString());
              mContents[i].Style.Add("POSITION","absolute");
              mContents[i].Font.Name = "Courier New";
              mContents[i].Font.Size = FontUnit.Parse("12");
              mContents[i].Font.Bold = false;
              ePoint += 30;
              if (mContents[i].Text.Length > 40)
              {
                     ePoint += 10;
              }
              tipo = true;
          }
              painel.Controls.Add(mContents[i]);
          }
        }
      }
   }
}
```

O quadro 4.5 contém o código para iniciar o monitoramento de processos criados na estação gerenciada.

Quadro 4.5 – Código para monitoramento

```
public void IniciaMonitor()
{
      this.evento= new ManagementEventWatcher();
      this.evento.Scope = this.wmi.Computador;
      this.evento.Query = new System.Management.EventQuery("SELECT * FROM
__InstanceCreationEvent WITHIN 1 WHERE TargetInstance ISA \"Win32_Process\"");
      this.evento.EventArrived += new EventArrivedEventHandler(notifica);
      this.evento.Start();
}

public void notifica (object sender, EventArrivedEventArgs e)
{
      MyListViewItem item = new MyListViewItem();
      item.Image = "images/computer.gif";
      item.Text = "Processo criado => " +
((ManagementBaseObject)e.NewEvent["TargetInstance"])["Caption"];
      this.lista.Items.Add(item);

}
```

Neste quadro pode-se visualizar a implementação dos métodos *IniciaMonitor()* e *notifica()*, implementados na classe *NotificacoesWMI*. O método IniciaMonitor é responsável por iniciar um monitor e informá-lo que ao receber uma notificação deve ser executado o método *notifica()*, este irá escrever a notificação numa lista do tipo *MyListView*, que é recebida como parâmetro no construtor desta classe. A seguir será apresentada a operacionalidade do protótipo desenvolvido.

4.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

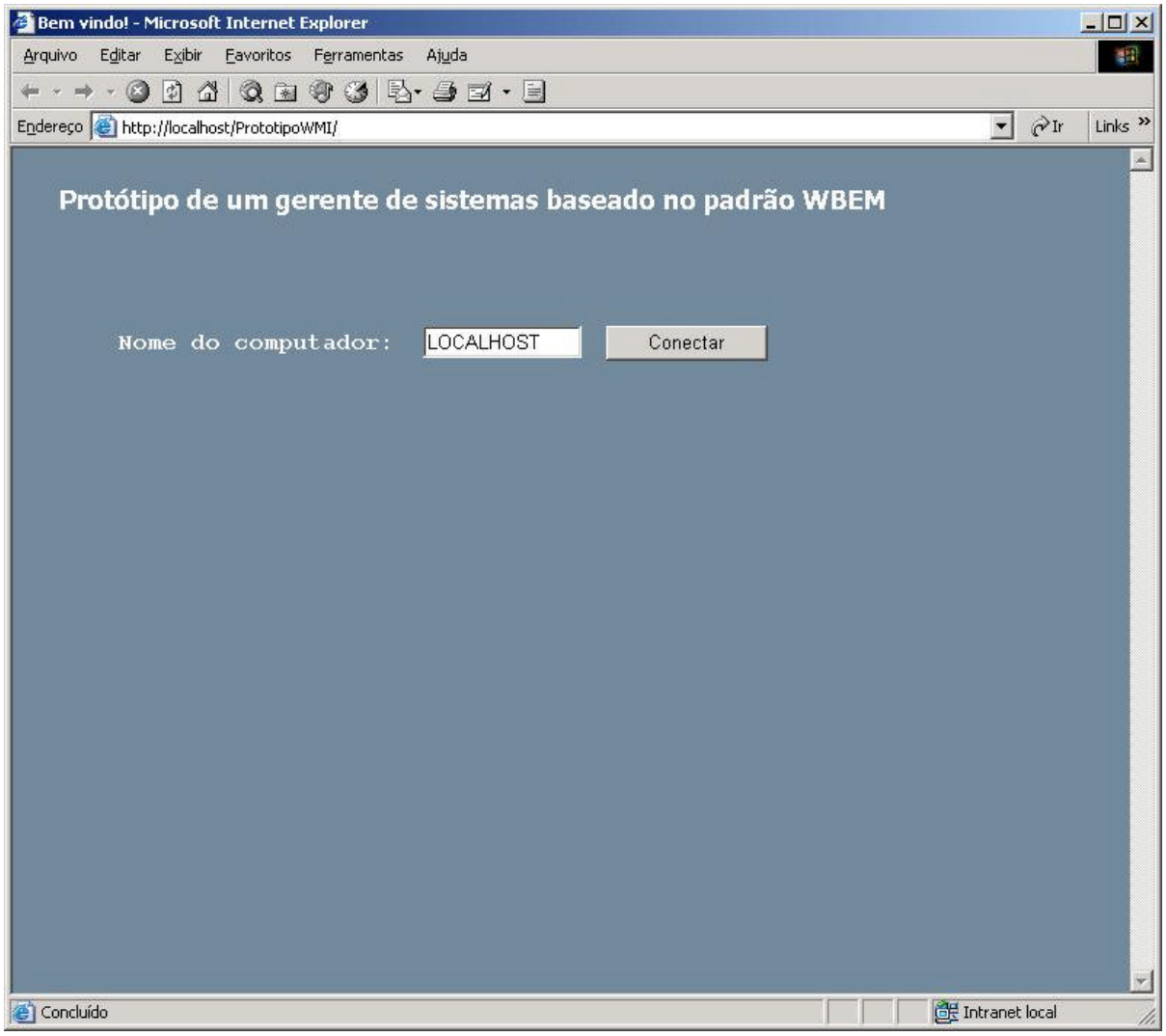
Conforme descrito anteriormente, para execução do protótipo é necessário:

a) na estação servidora, que irá hospedar o protótipo, deve estar instalado o servidor WEB *IIS* , *framework* .NET e *IE WEB Controls*;
b) nas estações que irão acessar a aplicação não é necessário nenhuma configuração adicional, apenas é recomendado a utilização do navegador *Internet Explorer 5.0* ou superior.

A seguir pode-se visualizar as páginas do protótipo, bem como sua descrição em relação à interação com o usuário. Para acessar o protótipo é necessário abrir um *browser* e acessar o local onde este está hospedado, como por exemplo http://localhost/PrototipoWMI.

Na fig. 4.10 pode-se visualizar a página de entrada do protótipo, onde é necessário informar qual estação que se deseja iniciar o gerenciamento. Para isto digita-se o nome da estação (*host*) ou endereço IP e pressiona-se o botão conectar. Se a conexão ocorrer o usuário será redirecionado a outra página que contém as informações de gerenciamento. Caso contrário será exibida uma mensagem de erro ao usuário.

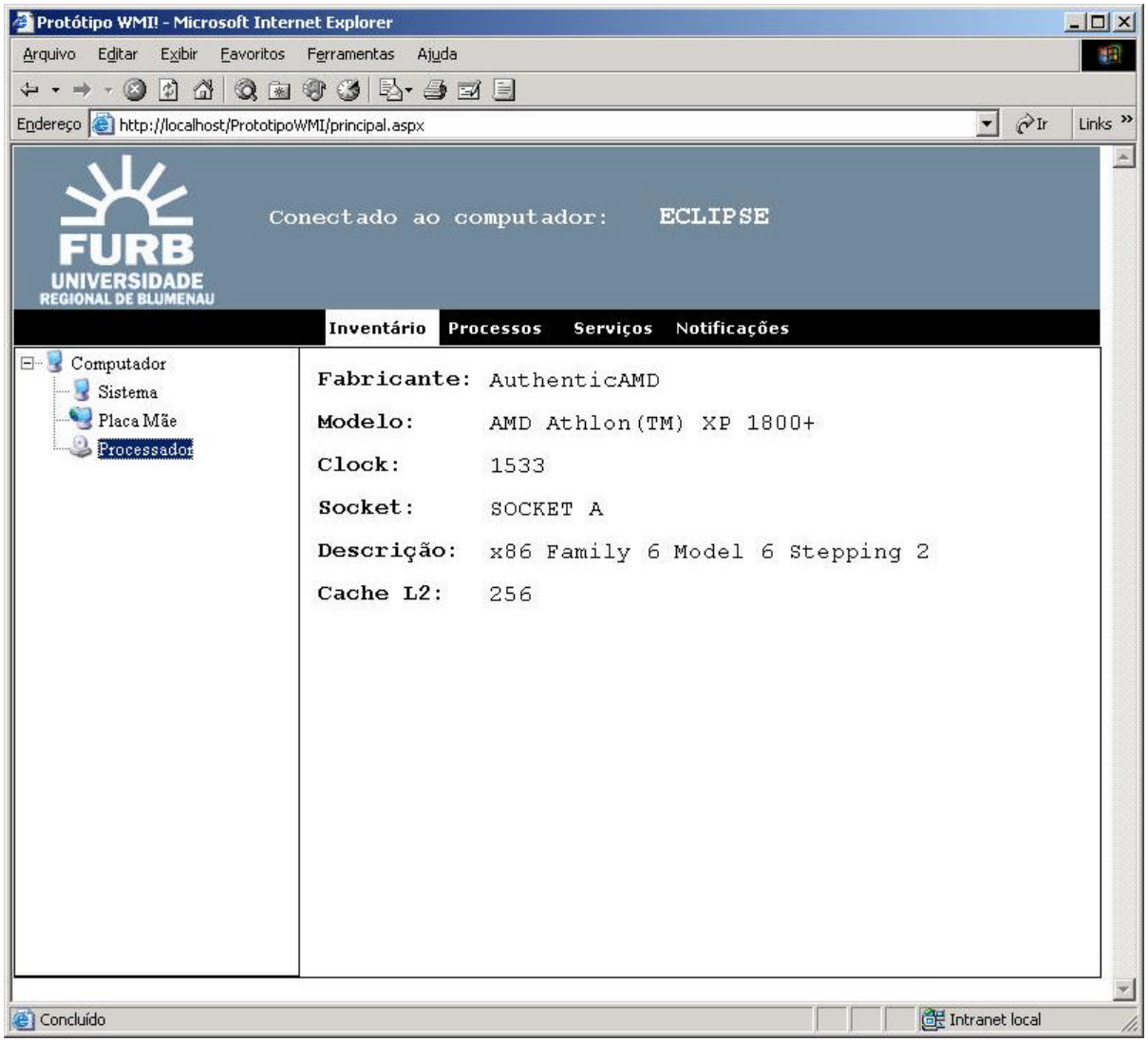
Figura 4.10 – Página inicial do protótipo



Na fig. 4.11 pode ser vista a página de exibição das informações de gerenciamento. Nesta tela existem quatro botões em que o usuário pode interagir: inventário, processos, serviços e notificações. Por padrão, após a conexão, esta página é aberta no item inventário.

No item de inventário o usuário pode interagir com as informações que estão no *frame* à esquerda da página. Neste frame está disposta uma lista de itens na forma de uma árvore, componente *TreeView*. Ao selecionar um item da árvore, no *frame* à direita da página, são exibidas informações detalhadas sobre o item, como pode ser visto na fig. 4.10. Portanto cada item possui informações pertinentes às suas características.

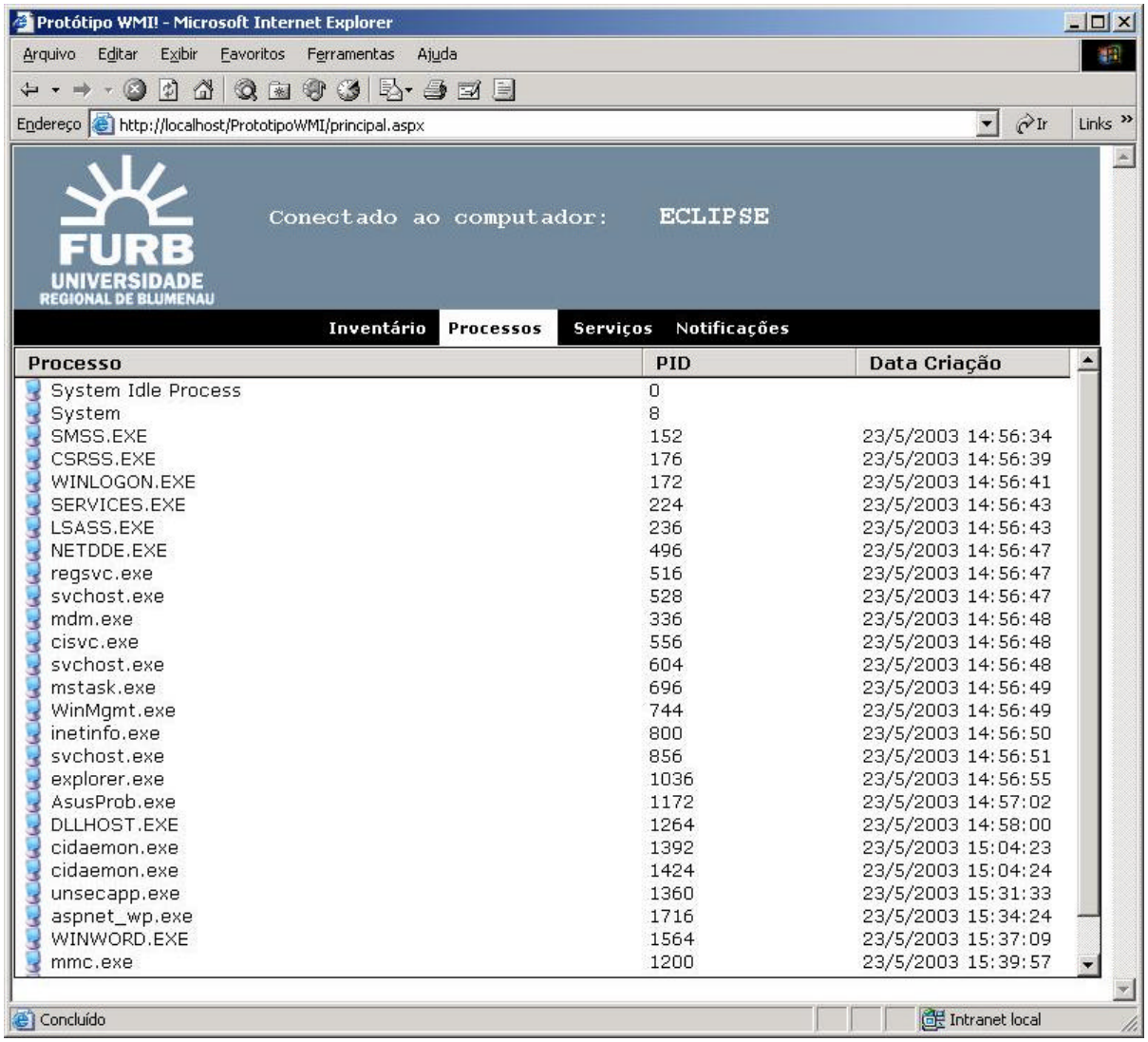
Figura 4.11 – Página de visualização de informações de inventário



Ao pressionar o botão “Processos” o usuário poderá visualizar informações relacionadas aos processos em execução na estação que se solicitou o gerenciamento. A fig. 4.12 mostra como são exibidas estas informações.

O usuário pode visualizar três propriedades dos processos, são elas: nome, PID e data de criação. Estes dados são exibidos na tela usando o componente *MyListView*, semelhante a uma tabela, separando-os em linhas e colunas.

Figura 4.12 – Página de visualização dos processos

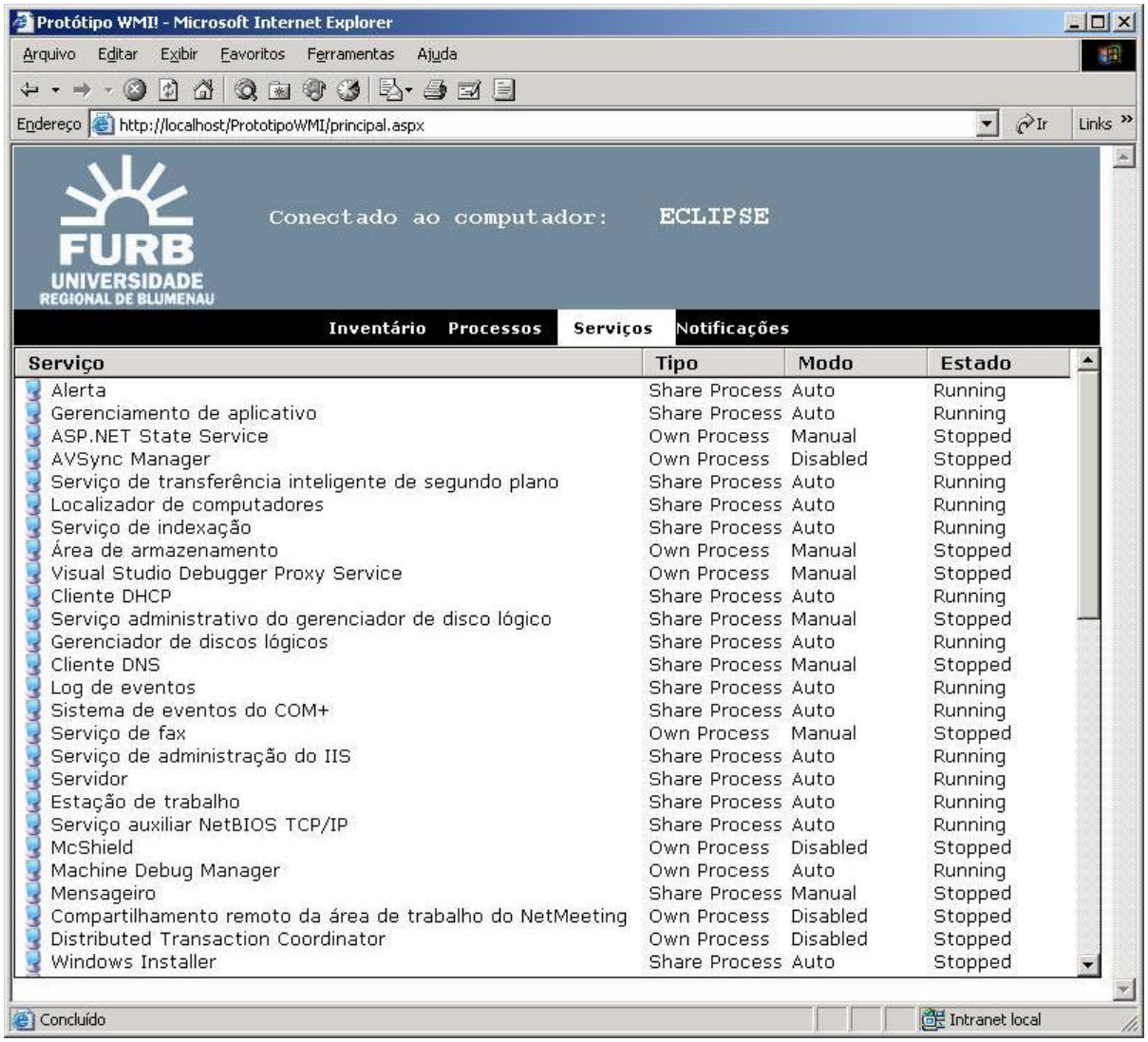


Ao pressionar o botão “Serviços” o usuário irá visualizar informações sobre os serviços disponíveis na estação que está sendo gerenciada. Com relação aos serviços é

possível verificar as seguintes propriedades: nome, tipo, estado e modo do serviço. Na fig. 4.13 pode-se visualizar a página de serviços.

Os serviços diferem dos processos pois, não necessariamente, precisam estar em execução. Portanto, é pertinente que o usuário possa verificar todos os serviços disponíveis e saber seu respectivo estado, modo e tipo.

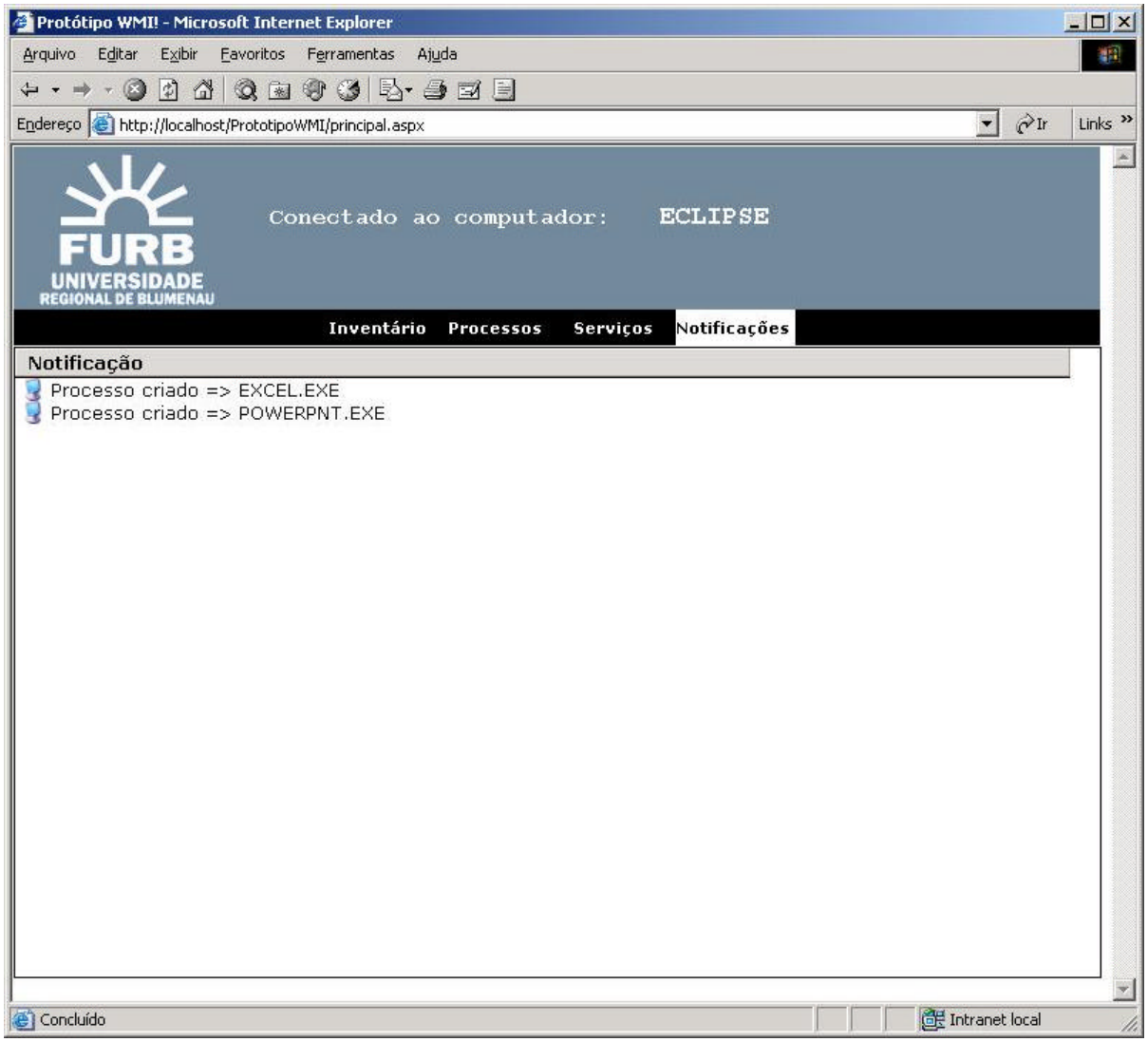
Figura 4.13 – Página de visualização dos serviços



Serviço	Tipo	Modo	Estado
Alerta	Share Process	Auto	Running
Gerenciamento de aplicativo	Share Process	Auto	Running
ASP.NET State Service	Own Process	Manual	Stopped
AVSync Manager	Own Process	Disabled	Stopped
Serviço de transferência inteligente de segundo plano	Share Process	Auto	Running
Localizador de computadores	Share Process	Auto	Running
Serviço de indexação	Share Process	Auto	Running
Área de armazenamento	Own Process	Manual	Stopped
Visual Studio Debugger Proxy Service	Own Process	Manual	Stopped
Cliente DHCP	Share Process	Auto	Running
Serviço administrativo do gerenciador de disco lógico	Share Process	Manual	Stopped
Gerenciador de discos lógicos	Share Process	Auto	Running
Cliente DNS	Share Process	Manual	Stopped
Log de eventos	Share Process	Auto	Running
Sistema de eventos do COM+	Share Process	Auto	Running
Serviço de fax	Own Process	Manual	Stopped
Serviço de administração do IIS	Share Process	Auto	Running
Servidor	Share Process	Auto	Running
Estação de trabalho	Share Process	Auto	Running
Serviço auxiliar NetBIOS TCP/IP	Share Process	Auto	Running
McShield	Own Process	Disabled	Stopped
Machine Debug Manager	Own Process	Auto	Running
Mensageiro	Share Process	Manual	Stopped
Compartilhamento remoto da área de trabalho do NetMeeting	Own Process	Disabled	Stopped
Distributed Transaction Coordinator	Own Process	Disabled	Stopped
Windows Installer	Share Process	Auto	Stopped

Na fig. 4.14 pode-se visualizar a página de notificações. Esta página funciona da seguinte maneira: o usuário pressiona o botão notificações, a página fica em espera num período de aproximadamente vinte segundos por eventos de criação de processos e ao término deste período são exibidos os eventos.

Figura 4.14 – Página de notificações



Esta implementação utilizando o conceito de aplicação WEB é inadequada para o monitoramento de eventos. Uma sugestão para melhor implementar este módulo é o desenvolvimento de uma aplicação, utilizando o *framewok* .NET, onde as notificações seriam enviadas via mensagem de correio eletrônico. No apêndice A pode-se visualizar o código de uma aplicação desenvolvida para ilustrar o monitoramento de eventos do serviço WMI em que as notificações são enviadas através de mensagens eletrônicas.

4.4 RESULTADOS E DISCUSSÃO

Os módulos de inventário, processos e serviços atenderam aos requisitos do protótipo. Como resultado tem-se um gerenciamento de sistemas através da WEB, sendo possível de qualquer ponto da rede ou internet (desde que o servidor WEB esteja devidamente configurado para tal) acessar as estações e visualizar as informações de gerenciamento.

Já para o módulo de notificações ficou claro que recebê-las através de um *browser* não é uma forma adequada. Sendo que como sugestão foi colocado o desenvolvimento deste módulo em separado e que as notificações sejam recebidas através de mensagens de endereço eletrônico.

Até o presente momento não foi possível aperfeiçoar este módulo. Foram realizadas diversas pesquisas na WEB onde ficou constatado que não é possível fazer de outra maneira devido a limitações da arquitetura das aplicações WEB, descritas em Mitchell (2001).

5 CONCLUSÕES

Com o desenvolvimento deste trabalho, através da leitura de diversos artigos, pode-se concluir que a utilização de tecnologias baseadas na arquitetura WEB é uma tendência. Cada vez mais se observa o uso de técnicas e ferramentas voltadas para a WEB no desenvolvimento de aplicações. Isto se tornou uma realidade na área de gerência devido ao surgimento do padrão WBEM.

Este padrão não representa apenas um modelo em que as informações são exibidas num *browser*, mas também uma forma de reduzir custos e facilitar o gerenciamento. Pôde-se perceber que existem diversas implementações deste padrão, sendo que o WMI, implementação proposta pela Microsoft, mostrou-se muito útil para o gerenciamento de estações que utilizam sistemas operacionais da Microsoft.

Com a utilização do WMI é possível realizar diversas operações de gerenciamento que vão desde a criação de um inventário até instalações de *software* remotamente. Conforme apresentado no capítulo 3 seção 4, o único inconveniente de sua arquitetura deve-se ao fato da utilização de objetos distribuídos na sua comunicação ao invés do protocolo HTTP. Porém, com a utilização do *framework* .NET em conjunto com uma aplicação WEB em ASP.NET, pôde-se perceber que é possível desenvolver aplicações de gerência sobre o WMI de forma rápida e com uma interface voltada para a WEB. Portanto as ferramentas utilizadas para a implementação do protótipo se demonstraram adequadas. A utilização da UML para especificação também se demonstrou adequada.

Uma das principais características do gerenciamento é a recepção de notificações de eventos. Neste módulo pode-se concluir que a utilização de uma aplicação WEB como interface de notificação de evento não é uma forma adequada de monitoramento. Como solução para este caso foi sugerida a implementação de uma aplicação, utilizando o *framework* .NET, sem a interface para WEB, onde as notificações são enviadas via mensagens de endereço eletrônico. Um exemplo de aplicação implementada seguindo este conceito pode ser encontrado no apêndice A.

O protótipo desenvolvido limita-se a mostrar os processos e serviços e não possui rotinas para interação destes com o usuário, como por exemplo, finalizar um processo ou alterar o modo de inicialização de um serviço.

Uma restrição a esta proposta deve-se ao fato de que é possível gerenciar apenas estações que estão rodando o serviço WMI. E este serviço é disponível apenas para ambientes da Microsoft. O protótipo também não pode estar hospedado em um servidor WEB que não seja o IIS, pois é necessário que esteja instalado o *framework* .NET e este é disponível apenas para sistemas operacionais da Microsoft.

Como contribuições do trabalho pode-se citar:

a) um estudo sobre os conceitos de gerência corporativa;
b) um estudo sobre a gerência corporativa baseada na arquitetura WEB;
c) um estudo sobre o padrão WBEM da DMTF;
d) um estudo sobre o WMI;
e) o desenvolvimento de um protótipo que demonstra na prática o funcionamento da gerência de sistemas através da WEB;
f) a utilização do *namespace System.Management* e da plataforma .NET no desenvolvimento de aplicações de gerenciamento de sistemas utilizando o WMI;

5.1 EXTENSÕES

Como sugestões para trabalhos futuros pode-se citar:

a) implementar rotinas de interação do usuário com a lista de processos e a lista de serviços;
b) aumentar a lista de informações de inventário, adicionando informações sobre discos rígidos, memória, etc;
c) aprimorar a autenticação de usuários para acesso ao serviço WMI;
d) desenvolver uma aplicação de notificações, aprimorando o código fonte incluído no apêndice A;
e) implementar um módulo para instalação de *softwares* remotamente, utilizando o WMI.

REFERÊNCIAS BIBLIOGRÁFICAS

CARVILHE, José Luís Vieira. **A utilização de tecnologias WEB em sistemas de gerência corporativa**. 2000. 102 f. Monografia (Especialização em Sistemas Distribuídos) - Pontifícia Universidade Católica do Paraná, Curitiba.

CASTRO, João Carlos. **Gerenciamento de sistemas baseado no padrão WBEM**. 2002. 71 f. Monografia (Tecnólogo em Processamento de Dados) – Universidade Tiradentes, Aracaju.

CENTER, 7 Inc. **OpenWBEM project.** [S.l], 2003. Disponível em: <http://www.openwbem.org/>. Acesso em: 27 fev. 2003.

DMTF, Distributed Management Task Force. **Common Information model (CIM) specification version 2.2**, [S.l], 1999. Disponível em: <http://www.dmft.org/standards/cim_spec_v22/>. Acesso em: 27 fev. 2003.

FUNKELAB, Funkelab Corporation. **Funkelab website**. [S.l], 2003. Disponível em: < http://www.funkelab.com>. Acesso em: 27 abr. 2003.

LARMAN, Craig. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2000. xi, 492 p.

MARTINSSON, Tobias. **Desenvolvendo scripts XML e WMI para o Microsoft SQL Server 2000**. São Paulo: Makron Books, 2002. xviii, 412 p.

MATOS, Alexandre Veloso de. **UML: prático e descomplicado**. São Paulo: Érica, 2002. 187p.

MICROSOFT, Microsoft Corporation. **The official Microsoft ASP.NET site**. [S.l], 2003. Disponível em: < http://www.asp.net>. Acesso em: 27 abr. 2003.

MICROSOFT, Microsoft Corporation. **Learn-WMI,** [S.l], 1999. Disponível em: <http://www.microsoft.com/downloads/release.asp?releaseid=12570>. Acesso em: 27 fev. 2003.

MICROSFT, Microsoft Corporation. **WMI and CIM concepts and terminology**, [S.l], 2001. Disponível em: <http://www.microsoft.com/hwdev/driver/WMI/WMI-CIM.asp>. Acesso em: 27 fev. 2003.

MITCHELL, Scott. **ASP.NET: tips, tutorials and code**. Indianopolis: Sams, 2001. xiii, 878p.

MORAES, Neilande. **Iniciando em XML.** São Paulo: Makron, 1999. 404p.

OPEN Group. **Pegasus project**. [S.l], 2003. Disponível em: <http://www.openpegasus.org/>. Acesso em: 27 fev. 2003.

PEREIRA, Mateus Casanova. **Administração e gerência de computadores.** 2001**.** 106 f. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Santa Catarina, Florianópolis.

POZEN, Zina. **System.Management lets you take advantage of WMI APIs within managed code.** [S.l], 2002. Disponível em: <http://msdn.microsoft.com/msdnmag/issues/02/05/WMIMan/default.aspx>. Acesso em: 27 fev. 2003.

RENEGATIVE, Renegative Inc. **Instructions on installing WMI**. [S.l], 2002. Disponível em: <http://www.regenative.com/PDFs/Win%2098_ME_WMI.pdf>. Acesso em: 15 abr. 2003.

SPARX System. **Enterprise architect user guide**, [S.l], 2003. Disponível em: <http://www.sparxsystems.com.au/EAUserGuide/index.html>. Acesso em: 15 abr. 2003.

STALLINGS, William. **SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.** Boston: Addison-Wesley, 1999. xv, 619 p.

STEINKE, Steve. **Network and systems management with XML,** [S.l], 1999. Disponível em: <http://www.networkmagazine.com/article/NMG20000509S0016/1>. Acesso em: 27 fev. 2003.

TEIXEIRA, S.R; MORAES, L.F; TEIXEIRA, J.H. **Gerência baseada na web**. [S.l], 1999. Disponível em: <http://www.ravel.ufrj.br/publicacoes/suzana_rel_webmngt.pdf>. Acesso em 27 fev. 2003.

APÊNDICE A

Código fonte desenvolvido em C# para ilustrar o funcionamento de uma aplicação de monitoramento de eventos utilizando o WMI onde as notificações são enviadas através de mensagens eletrônicas.

```
// Desenvolvido por Edson Luiz Braz da Silva Junior
// data: 22/05/2003

using System;
using System.Web.Mail;
using System.Management;
using System.Threading;

namespace MonitorWMI
{
 class Monitor
 {
   public void criaMonitor (string computer)
   {

     ConnectionOptions oConn = new ConnectionOptions();
     ManagementScope comp = new ManagementScope("\\\\"+computer+"\\root\\cimv2", oConn);
     try
     {
       comp.Connect();
       try
       {
              if (comp.IsConnected)
              {
                 ManagementEventWatcher novo = new ManagementEventWatcher();
                 novo.Scope = comp;
                 novo.Query = new EventQuery("SELECT * FROM __InstanceCreationEvent
WITHIN 1 WHERE TargetInstance ISA \"Win32_Process\"");
                 novo.EventArrived += new EventArrivedEventHandler(enviaEmail);
                 novo.Start();
              }
       }
       catch
       {
              Console.WriteLine("Não foi possível conectar ao host: " + this.computer);
       }
     }
     catch
     {
         Console.WriteLine("Erro ao criar objetos");}
     }
   }
```

```
   public void enviaEmail (object sender, EventArrivedEventArgs e)
   {
      try
      {
               Console.WriteLine("Processo criado!!!" +
((ManagementBaseObject)e.NewEvent["TargetInstance"])["Caption"]);
               MailMessage Message = new MailMessage();
               Message.To = "mail@server.br";
               Message.From = "mail@server.br";
               Message.Subject = "Processo criado";
               Message.Body = ("Foi criado um processo no computador: " + this.computer);
               SmtpMail.SmtpServer = "smtp.server.br";
               SmtpMail.Send(Message);
      }
      catch
      {
               Console.WriteLine("Ocorreu um erro ao enviar email!!!");
      }
   }


   class Start
   {
      // The main entry point for the application.
      [STAThread]
      static void Main(string[] args)
      {
               Monitor novoMonitor = new Monitor();
               novoMonitor.criaMonitor("localhost");
               while (true)
               {
                     System.Threading.Thread.Sleep(1000);
               }
      }
   }
}
```