UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

Sistema de Monitoramento Remoto de Recursos Computacionais

Autor: _____
Fernando de Castro Netto

Orientador: _____
Sérgio Barbosa Villas-Boas, Ph.D.

Examinador: _____
Antônio Cláudio Gómez de Sousa, M.Sc.

Examinador: _____
Sergio Palma da Justa Medeiros, D.Sc.

DEL

Agosto de 2005

Agradecimentos

Agradeço, primeiramente, aos meus pais, Marcos e Glória, e ao meu irmão, Daniel, por tudo que fizeram e fazem por mim, e por todo apoio durante à minha vida.

À minha namorada, Marcelle, pelo incentivo e compreensão demonstrados diante de todos os obstáculos encontrados durante o caminho, principalmente nos momentos mais difíceis.

Ao meu amigo Júlio César, que infelizmente faleceu no ínicio do ano, mas sempre esteve ao meu lado nos momentos mais importantes da minha vida, desde o nosso vestibular à minha cerimônia de colação de grau.

Ao meu orientador, Sérgio B. Villas-Boas, por todo apoio e dedicação ao longo deste projeto.

Aos colegas de turma, que contribuíram muito para minha formação pessoal e profissional durante o curso, e a todos que me ajudaram, direta ou indiretamente.

Ao povo brasileiro, por pagar meu curso superior.

Resumo

Neste trabalho foi desenvolvido um sistema capaz de monitorar recursos computacionais (CPU, I/O, Memória, Processos), remotamente através de uma rede de computadores. O sistema constitui-se de uma arquitetura cliente-servidor, onde o software cliente é instalado na máquina monitorada. O servidor é o computador que receberá os dados monitorados, enviados pelo cliente. Foi criada uma base dados onde são armazenados os dados recebidos. A partir dessas informações coletadas, é possível gerar relatórios gerenciais capazes de inventariar com precisão as interfaces e aplicativos instalados nas máquinas, bem como mensurar a capacidade computacional das mesmas. O programa foi desenvolvido na linguagem C++, usando o framework de desenvolvimento Visual Studio da Microsoft. Para armazenamento dos dados, o sistema utiliza o serviço de banco de dados SQLSERVER (versão 2000 para desenvolvimento), e para transmissão dos dados, o mesmo utiliza o protocolo de comunicação. Os testes foram realizados em uma rede de computadores, onde foram coletadas informações de cinco máquinas, durante o período de 8 horas.

Palavras-Chave

Monitoramento de recursos computacionais
C++
TCP\IP
Processos
Consumo de CPU
Consumo de Memória
Mac Address
Software
Endereço IP
Banco de Dados
Classes
Biblioteca
MFC
SQL SERVER
Log do Sistema
Agente
SQL
Linha de comando

Lista de Acrônimos

API: *Application Programming Interface*

LAN: *Local Area Network*

MFC: *Microsoft Foundation Classes*

MAC: *Media Access Control*

IP: *Internet Protocol*

TCP: *Transmission Control Protocol*

I/O: *Input / Output*

CPU: *Central Processing Unit*

OO: *Orientação a Objeto*

DLL: *Dynamic Link Library*

SQL: *Structured query language*

PID: *Process identifier*

STL: *Standard Template Language*

HD: *Hard Disk*

UML: *Unified Modeling Language*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nas últimas décadas, a disputa por novos mercados entre as empresas é bastante acirrada. Sabemos que uma boa infra-estrutura de TI constitui-se em uma ferramenta fundamental para conquistar uma posição competitiva favorável. Esse tipo de arquitetura é capaz de prover aumento na produtividade, redução de custos e informações gerenciais para tomada de decisão.

Por outro lado, uma arquitetura de TI ineficiente provoca um aumento dos custos e redução da produtividade, em função das constantes manutenções corretivas aplicadas, que demandam interrupção das atividades da produção para trocas de equipamentos ou atualizações de aplicativos.

Para montar uma estrutura de computadores adequada, é preciso realizar um estudo de avaliação das necessidades da instituição. Nesse contexto, é preciso determinar os aplicativos (softwares) que serão utilizados pelos diversos grupos de usuários, os sistemas operacionais adequados para os programas instalados e o perfil das máquinas instaladas (processadores, memória, disco rígido).

O objetivo deste projeto consiste em monitorar as arquiteturas computacionais das empresas, prevendo possíveis manutenções ou trocas de equipamentos futuras. O monitoramento consiste em coletar, periodicamente, informações a respeito do funcionamento de cada máquina, através de medições dos recursos computacionais (cpu, memória, lista de processos em execução, I/O). Além disso, o sistema consegue capturar todos os aplicativos instalados nas máquinas, garantindo eficiência no controle de licenças de softwares. A partir das informações fornecidas pelo sistema, a gestão da empresa pode tomar decisões antecipadas impedindo que a produção pare em função de problemas com os computadores.

Na busca por uma solução para o monitoramento da arquitetura de TI, levei em conta que essa estrutura está distribuída em redes de computadores, onde as máquinas podem estar distantes uma das outras. Por causa disso, optei pelo monitoramento remoto, onde consigo consolidar as informações de todos os computadores em um único ponto, armazenando esses dados coletados em um banco de dados.

O sistema foi desenvolvido baseado em uma arquitetura cliente-servidor, onde o software cliente será instalado na máquina monitorada. O servidor será o computador que receberá os dados monitorados, enviados pelo cliente. Há uma base dados instalada no servidor, para armazenar os dados recebidos. O programa

é compatível com o sistema operacional Windows (versões 95, 98, NT 4.0, 2000 e XP).

O processo de desenvolvimento das rotinas para capturar as informações monitoradas, consistiu em um bom tempo de pesquisa na Internet, onde através de sítios destinados a programadores de software, achei diversos trechos de funções escritas em C++. Além disso, encontrei nessa linguagem de programação, facilidades para implementação do projeto, como abstração dos dados através das classes, modularização e reutilização dos códigos. Todos esses fatores contribuíram para o desenvolvimento do sistema nessa linguagem.

Este trabalho está organizado da seguinte forma: o Capítulo 2 traz uma visão geral do sistema, apresentando sua organização e estrutura. No Capítulo 3, será descrita a biblioteca de monitoramento, incluindo os diagramas das classes. O Capítulo 4 descreve a arquitetura cliente-servidor do sistema. No capítulo 5, será apresentada a interface com o banco de dados. O capítulo 6 descreve as telas do sistema. O Capítulo 7 apresenta os resultados dos testes obtidos após a execução do sistema, enquanto que o Capítulo 8 apresenta as conclusões do projeto e algumas propostas para sua continuidade.

Capítulo 2

Estrutura e Organização do Sistema

O sistema desenvolvido pode ser esquematizado como mostra a Figura 2.1. Como podemos notar, o diagrama abaixo ilustra uma arquitetura cliente-servidor, onde temos um agente sendo executado na máquina cliente. Observe que este programa comunica-se com algumas DLLs, também conhecidas como plug-ins. Através dessas bibliotecas dinâmicas, o agente carrega as rotinas de monitoramento e extrai as informações desejadas da máquina cliente. Concluída essa tarefa, os dados são empacotados e enviados através da Internet para um servidor remoto, onde há um serviço aguardando a chegada do pacote. Em seguida, os dados são processados no servidor e armazenados em um banco de dados.

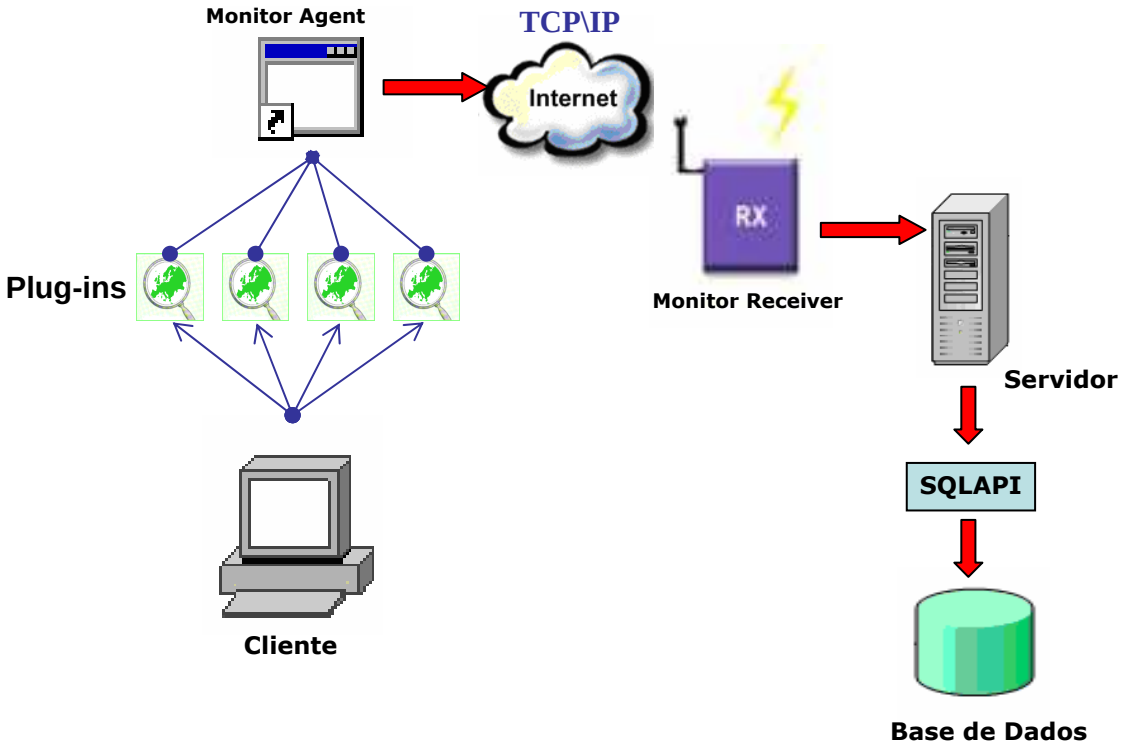

Figura 2.1: Esquematização do Sistema

2.1 Plug-ins

Os plug-ins apresentados no diagrama acima, podem ser descritos como bibliotecas dinâmicas (DLLs), desenvolvidas em C++, com funções que serão carregadas por um outro programa em tempo de execução. Essas bibliotecas possuem duas funções, como podemos ver a seguir:

```
. bool CheckMonitor();
```

Essa rotina é usada para reconhecimento da DLL como plug-in de monitoramento, isto é, o programa externo chama essa função para verificar se a biblioteca possui alguma rotina de monitoramento.

```
. int GetData (bool verbose,
                  const char* LogFileName,
                  std::string IPServer,
                  int port);
```

Essa função é usada para extrair as informações de monitoramento da máquina cliente, e em seguida, enviar os dados para o servidor remoto.

Parâmetros da função:

Verbose: parâmetro de entrada que define se a execução da rotina deverá registrar as etapas em algum arquivo texto. Esse atributo deverá receber um valor booleano (TRUE ou FALSE).

LogFileName: parâmetro de entrada que define o nome do arquivo, onde deverão ser registradas as operações na função. Esse parâmetro será ignorado caso o modo Verbose não esteja ativado.

IPServer: parâmetro de entrada que define o endereço IP do servidor para onde deverá ser encaminhado o pacote de dados.

Port: parâmetro de entrada que define a porta de comunicação usada pelo sistema para enviar e receber os dados.

Retorno da função: em caso de sucesso, a função retorna o número de bytes enviados no pacote. Caso contrário, a função retorna um valor negativo e a descrição do erro será encaminhada para o arquivo texto contendo o registro das operações.

Durante o desenvolvimento do projeto, optou-se pela utilização das bibliotecas dinâmicas buscando, principalmente, a manutenibilidade do sistema. Através da implementação das DLLs, fica evidente que em caso de atualizações ou correções nas rotinas de monitoramento, esse processo fica transparente para o sistema, pois

não há necessidade de recompilar os demais módulos. Além disso, se precisarmos monitorar outros recursos computacionais (além dos que serão monitorados nesse projeto) basta desenvolver novas DLLs, incluindo as funções descritos acima. Com isso, temos uma solução que apresenta uma arquitetura genérica para o monitoramento dos computadores, permitindo o desenvolvimento de novas ferramentas para o sistema, sem necessidade de alterações ou ajustes na versão em produção. No capítulo 3, será descrito detalhadamente o processo de desenvolvimentos do plug-ins do sistema.

2.2 Monitor Agent

Trata-se de um programa, desenvolvido em C++, que pode ser executado a partir de uma linha comando. Conforme foi visto na figura 2.1, este utilitário é executado na máquina cliente, e comunica-se com os plug-ins para carregar as rotinas de monitoramento.

A linha de comando de execução do programa é descrita da seguinte maneira:

MonitorAgent.exe [-v] [-l <arquivo_log>] –m <plug_in> <periodicidade_monitoamento> -p <porta_comunicacao> -s <endereço_ip_servidor>

Tabela 2.1: Parâmetros de execução do programa MonitorAgent.exe

Parâmetro	Descrição	Atributos	Obrigatório
-v	Ativa o modo verbose do programa.	Nenhum	Não
-l	Define o arquivo texto com o registro de operação (log) do sistema.	Nome do arquivo de registros	Não
-m	Adiciona um plug-in de monitoramento	. Nome da DLL . Periodicidade (em minutos) da coleta	Sim
-p	Porta de comunicação	Número da porta de comunicação	Sim
-s	Servidor remoto de destino	Endereço IP do servidor	Sim

Após executar a linha de comando devidamente parametrizada, o programa carrega os parâmetros de entrada, verificando a consistência do comando processado. Caso os parâmetros não tenham sido corretamente informados, o utilitário retorna uma exceção mostrando a linha de comando correta, e em seguida, aborta a execução do programa.

A seguir, temos um diagrama esquematizando todo o fluxo de processos do programa:

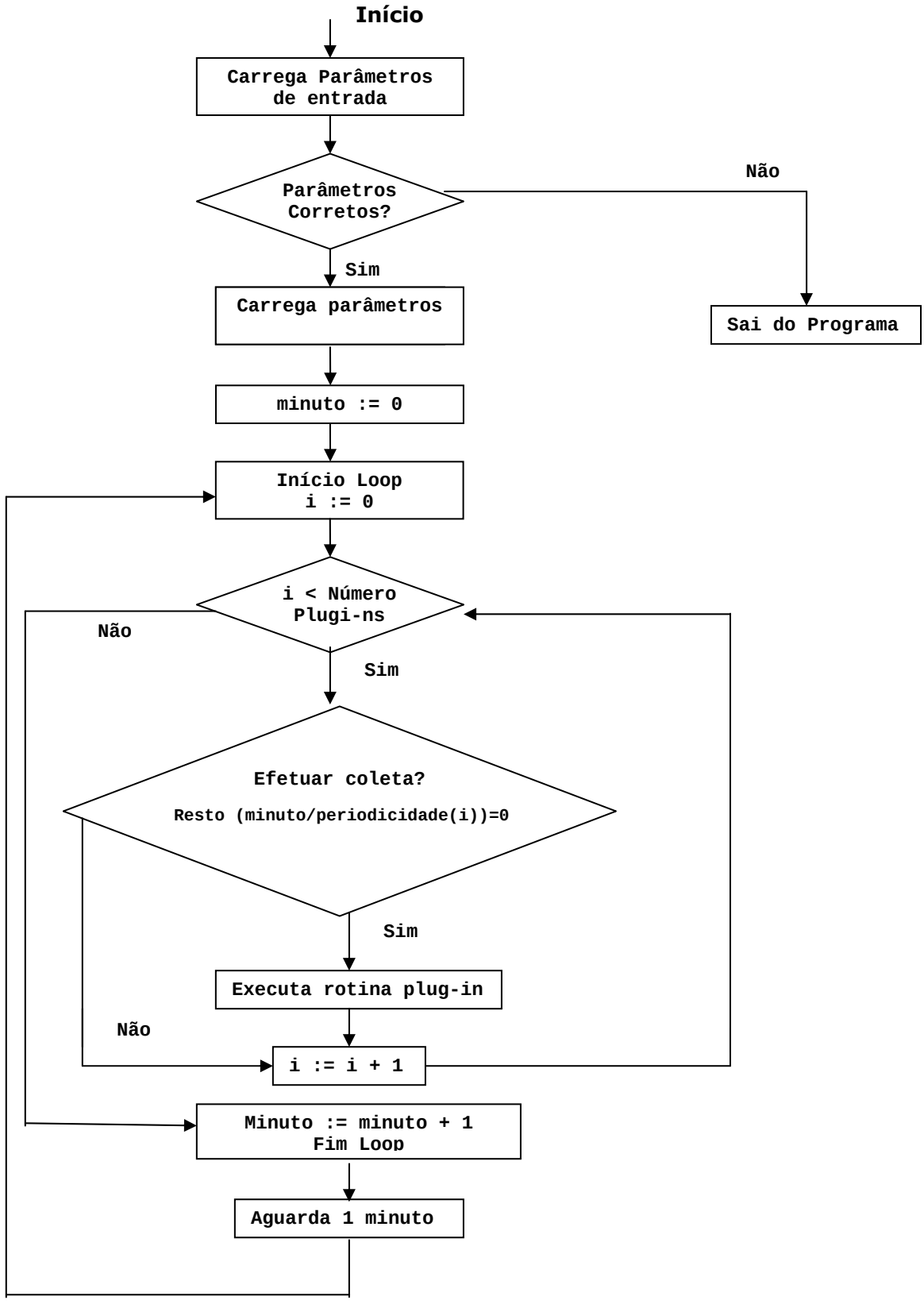

Figura 2.2 Fluxograma dos processos do programa MonitorAgent.exe

Para ilustrar o funcionamento do fluxograma acima, vamos utilizar uma linha de comando como exemplo.

MonitorAgent.exe -v -l C:\temp\monitor.log –m cpu.dll 1 –m processos.dll 10 -p 10000 -s 172.18.0.36

Note que os parâmetros foram corretamente informados no comando, respeitando as opções apresentadas na tabela 2.1. Logo, podemos concluir que a etapa de validação da parametrização retornará sucesso, encaminhando o processo para a etapa de leitura das configurações. A tabela 2.2 ilustra os dados carregados nessa fase.

Tabela 2.2: Configurações da execução do programa

Configuração	Valor
Modo verbose	Ativado
Arquivo de log	C:\temp\monitor.log
Quantidade de plug-ins	2
Plug-in 1	cpu.dll executado a cada 1 minuto
Plug-in 2	processos.dll executado a cada 10 minutos
Porta de comunicação	10000
Servidor remoto	172.18.0.36

A periodicidade de cada plug-in determina a freqüência de execução da função de monitoramento do componente. Foi preciso desenvolver um método para identificar o momento exato em que o programa deve carregar a rotina de monitoramento. Esse algoritmo consiste em usar um atributo, que é incrementado a cada minuto, e em seguida, verificar se o valor dessa variável é múltiplo da periodicidade do plug-in. Caso o valor do resultado seja verdadeiro, a coleta dos dados é realizada nesse instante. Essa verificação é feita para todos os monitores informados pela linha de comando. Depois disso, a aplicação aguarda 1 minuto (freqüência mínima de coleta do plug-in) e retorna para o processo de verificação do momento de coleta de cada componente.

No exemplo acima, as rotinas da DLL cpu.dll serão carregadas a cada minuto, enquanto que as funções da DLL processo.dll serão chamadas pelo programa a cada período de 10 minutos. Note que no primeiro instante, todos os plug-ins serão carregados, e a partir do minuto seguinte, ocorrerá a execução do algoritmo apresentado no parágrafo anterior.

Observe que, caso a linha de comando seja corretamente informada, o programa não apresenta nenhum ponto de finalização. Isso pode ser justificado pela idéia que o programa atuaria como um agente na máquina cliente, podendo ser terminando somente a partir de uma outra aplicação. Esse programa externo seria a interface cliente, que será descrita mais adiante nesse documento.

2.3 Monitor Receiver

Assim como o Monitor Agent, trata-se de um programa, desenvolvido em C++, que pode ser executado a partir de uma linha comando. O objetivo desse utilitário é ficar escutando uma porta de comunicação, aguardando pelo recebimento dos pacotes. Quando os dados são recebidos, os mesmos são processados e armazenados em um banco de dados. Abaixo, temos a linha de comando para execução do programa:

MonitorReceiver.exe [-v] [-l <arquivo_log>] -p <porta_comunicacao> -S <base_dados> -U <usuário_banco> -P <senha_banco>

Tabela 2.3: Parâmetros de execução do programa MonitorReceiverexe

Parâmetro	Descrição	Atributos	Obrigatório
-v	Ativa o modo verbose do programa.	Nenhum	Não
-l	Define o arquivo texto com o registro de operação (log) do sistema.	Nome do arquivo de registros	Não
-p	Porta de comunicação	Número da porta de comunicação	Sim
-S	Base de dados de armazenamento	Servidor@BaseDeDados	Sim
-U	Usuário do banco de dados	Login do usuário do banco	Sim
-P	Senha do banco de dados	Senha do usuário do banco	

Após a execução da linha de comando, o programa carrega os parâmetros de entrada e as interfaces da biblioteca Winsock. A seguir, o programa passa para a etapa onde fica escutando a porta de comunicação, aguardando pela chegada dos dados. Em caso de recebimento dos dados, as informações são processadas e armazenadas no banco de dados. Concluída essa etapa, o programa retorna para a fase onde ficar aguardando pela chegada dos pacotes. A linha de comando, abaixo, ilustra a execução do programa.

MonitorReceiver -v -l C:\temp\receiver.log -p 10000 –S localhost@MonitorDb –U sa –P teste

Tabela 2.4: Configurações da execução do programa

Configuração	Valor
Modo verbose	Ativado
Arquivo de log	C:\temp\monitor.log
Porta de Comunicação	10000
Servidor de banco	Localhost
Base de Dados	MonitorDb
Usuário do banco	As
Senha do usuário	Teste

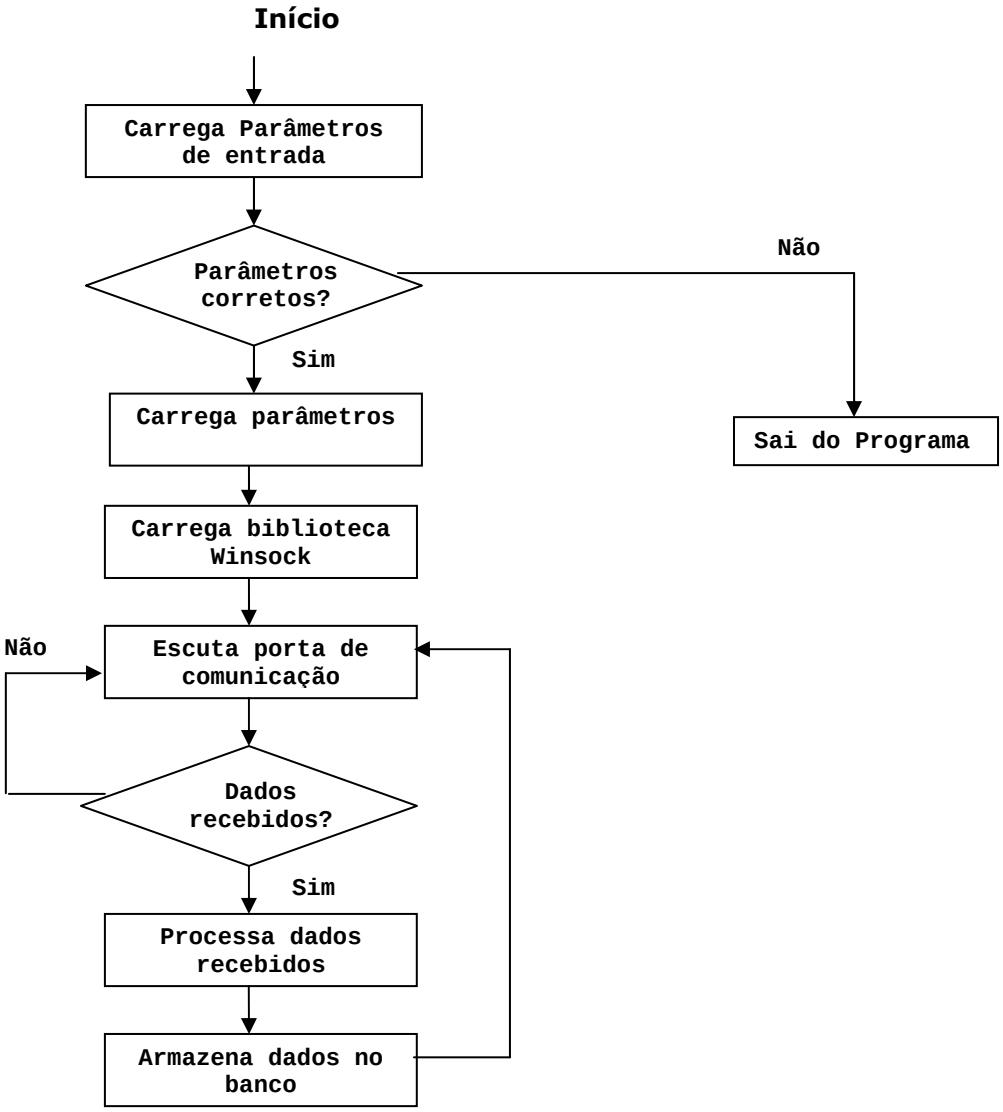

Figura 2.3 Fluxograma dos processos do programa MonitorReceiver.exe

As interfaces de comunicação desse programa serão descritas com maiores detalhes no capítulo 4, onde será apresentada a arquitetura cliente-servidor do sistema.

Como podemos observar no diagrama ilustrado na figura 2.1, há um componente usado para armazenar os dados processados no servidor. Para realização dessa interface, foi usada a biblioteca SQLAPI. Trata-se de um componente, também desenvolvido em C++, usado para acessar e executar comandos SQL na base de dados. O capítulo 5 aborda o tópico de integração do sistema com o banco de dados, e apresentará, detalhadamente, a comunicação dessa aplicação com o banco de dados.

Capítulo 3

Plug-ins de Monitoramento

Conforme foi descrito no capítulo anterior, os plug-ins podem ser descritos como bibliotecas dinâmicas (DLLs) que contém as implementações das rotinas de monitoramento. Essas funções são carregadas periodicamente pelo programa MonitorAgent.exe, durante o processo de coleta dos dados das máquinas monitoradas.

O objetivo desse capítulo consiste em descrever o processo de desenvolvimento dos plug-ins, apresentando as estruturas de dados coletados e enviados para o servidor remoto, bem como o diagrama de classes de cada módulo.

No escopo desse projeto, foram criados 7 plug-ins: os.dll (operating system), cpu.dll, memory.dll, process.dll, network.dll, io.dll e software.dll. Essas DLLs, conforme já foi mencionado anteriormente, foram desenvolvidas através da linguagem de programação C++, com uso do framework de desenvolvimento Visual Studio C++ (versões 6.0 e .NET).

3.1 Classes Globais

Na fase de análise de requisitos do sistema, surgiu a necessidade do aplicativo registrar, através de um arquivo texto de saída, as instruções principais processados pelo sistema. Essas informações deveriam ser classificadas em grupos, de maneira que fosse possível diferenciar um registro de informação, um alerta do sistema ou um erro operacional, durante a execução do sistema.

Outra necessidade abordada, foi o tratamento de erros ocorridos durante a execução do sistema. Qualquer exceção gerada pelo aplicativo, deve ser tratada de forma que a mensagem de erro apresentada objetive uma informação precisa capaz de identificar o problema. Além disso, é preciso evitar que as exceções geradas pelos diferentes módulos não impliquem na interrupção da aplicação como um todo.

A seguir, veremos a implementação de duas classes, CMonitorLog e CMonitorException, que foram desenvolvidas visando a implementação das funcionalidades apresentadas nos parágrafos anteriores.

3.1.1 Classe CMonitorLog

Trata-se de uma classe responsável pelo registro das operações realizadas no sistema. Essa classe é utilizada por outros módulos para registrar as instruções processadas. Abaixo, temos o diagrama da classe.

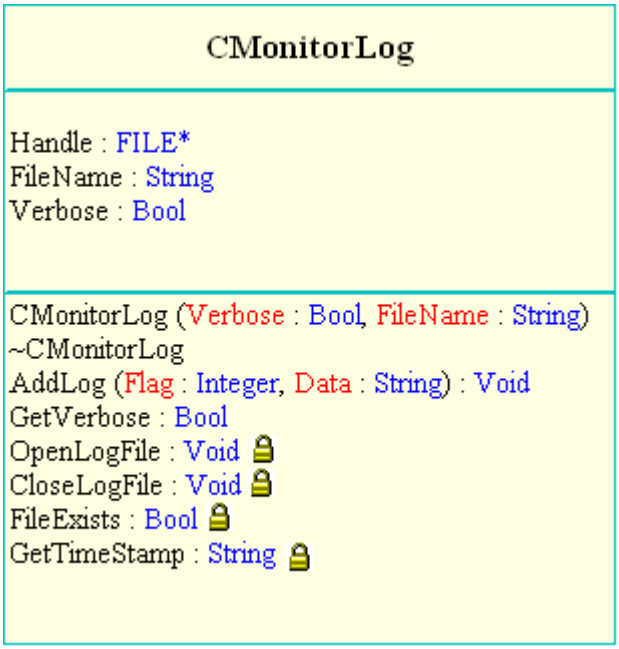

Figura 3.1 Classe CMonitorLog

Tabela 3.1: Descrição dos métodos da classe CMonitorLog

Método	Descrição	Tipo
CMonitorLog	Construtor da classe	Público
~CMonitorLog	Destrutor da classe	Público
AddLog	Registra uma instrução	Público
GetVerbose	Verfica se o mode verbose está ativado	Público
OpenLogFile	Abre\Cria o arquivo de log	Privado
CloseLogFile	Fecha o arquivo de log	Privado
FileExists	Verifica se o arquivo de log existe	Privado
GetTimeStamp	Retorna uma string no formato dd/mm/aaaa HH:MM:SS	Privado

Tabela 3.2: Flags usados como entrada do método AddLog

Flag	Descrição	Valor
Debug	Valor usado para trace do processamento das linhas de código	0
Warning	Valor usado para registrar os alertas	1
Info	Valor usado para identificar uma informação de sucesso no processamento	2
Error	Valor usado para registrar os alertas	3

3.1.2 Classe CMonitorException

Trata-se da classe responsável pelo tratamento dos erros gerados durante a execução do sistema. Essa classe captura as exceções geradas durante a execução do programa, identificando o nome da rotina e módulo onde ocorreu a falha, e o motivo que gerou o erro.

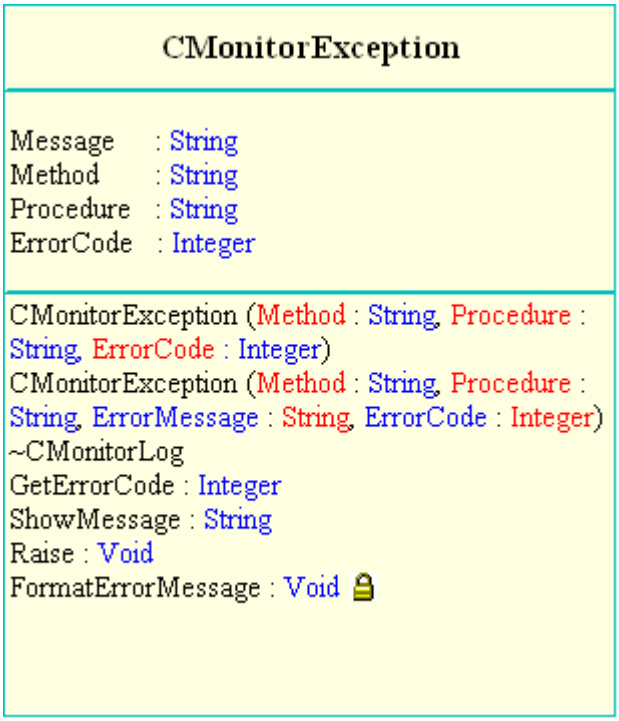

Figura 3.2 Classe CMonitorException

Tabela 3.3: Descrição dos métodos da classe CMonitorException

Método	Descrição	Tipo
CMonitorException	Construtor da classe	Público
~CMonitorException	Destrutor da classe	Público
AddLog	Registra uma instrução	Público
GetVerbose	Verfica se o mode verbose está ativado	Público
OpenLogFile	Abre\Cria o arquivo de log	Privado
CloseLogFile	Fecha o arquivo de log	Privado
FileExists	Verifica se o arquivo de log existe	Privado
GetTimeStamp	Retorna uma string no formato dd/mm/aaaa HH:MM:SS	Privado

Tabela 3.4: Atributos da classe CMonitorException

Flag	Descrição
Message	Armazena a mensagem de erro gerada pela classe
Method	Nome da rotina onde foi gerado o erro
Procedure	Nome da função ou API que gerou o erro
ErrorCode	Código do erro da função ou API

3.2 Sistema Operacional

Um dos principais problemas enfrentados nas arquiteturas de TI é a definição das plataformas que devem ser instaladas nas máquinas da empresa. Essa tomada de decisão deve ser baseada considerando-se o perfil dos aplicativos utilizados pelo usuário.

No contexto do projeto, foi proposta a criação de um plug-in para capturar a versão do sistema operacional instalado na máquina (no caso, versão do Windows instalado). A biblioteca responsável por esse monitoramento é a DLL *os.dll*. Para implementação desse projeto, foi criada a classe COsVersion.

3.2.1 Classe COsVersion

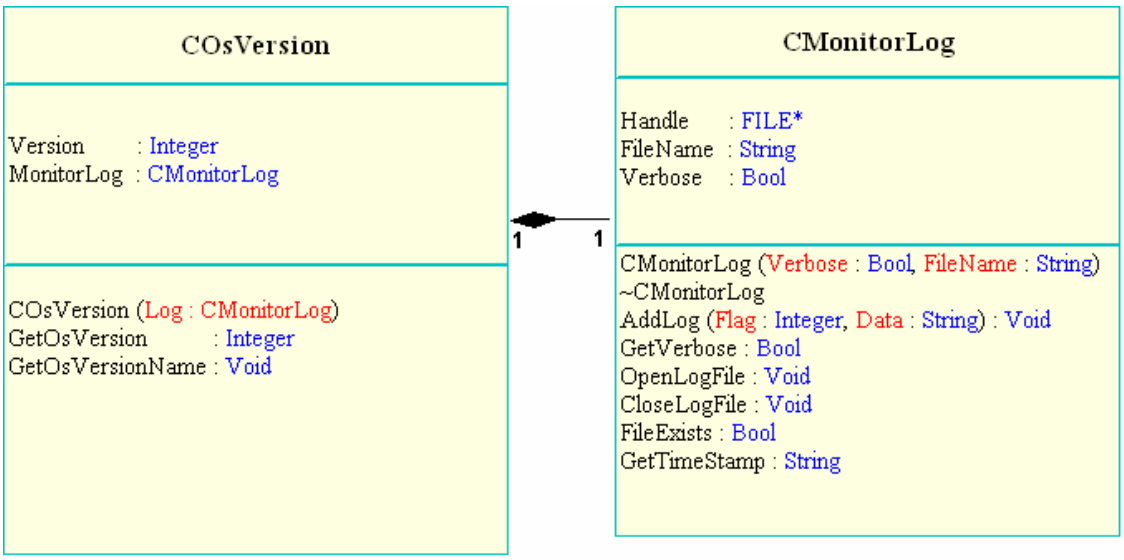

Figura 3.3 Diagrama da classe COsVersion

Através do diagrama acima, podemos notar que há um relacionamento de composição entre as classes COsVersion e CMonitorLog. Isso pode ser explicado pelo da classe COsVersion usar uma instância de CMonitorLog para registro das operações, durante a execução de seus métodos.

Para capturar a plataforma instalada na máquina foi usada a função *GetVersionEx* da API do Windows. Essa rotina consiste em preencher uma estrutura, a *OSVERSIONINFOEX*, com as informações que descrevem a plataforma. A partir desses dados foi possível reconhecer o sistema operacional utilizado. A tabela 3.5 apresenta a lista de sistemas operacionais tratados pelo plug-in.

Tabela 3.5: Lista de sistemas operacionais reconhecidos pelo plug-in os.dll

Constante	Descrição	Valor
Win95	Windows 95	0
Win98	Windows 98	1
WinMe	Windows Me	2
WinNT4	Windows NT 4.0	3
Win2K	Windows 2000	4
WinXP	Windows XP	5
Win2003	Windows Server 2003	6

3.3 CPU

Outro parâmetro que deve ser definido pela gestão de TI é o perfil dos processadores das máquinas. Na escolha desse dispositivo, deve-se levar em consideração às necessidades dos grupos dos usuários, isto é, a capacidade computacional exigida pelas aplicações instaladas.

Para avaliar se o processador utilizado é adequado para o perfil da aplicação, é importante medir o consumo de cpu da máquina ao longo da execução do programa. Essa medida consiste em gerar um valor percentual que determina a parcela da capacidade total de processamento da máquina que está sendo exigida em um determinado momento.

Esse parâmetro de observação é bastante útil, e em muitos casos pode explicar o motivo pelo qual um computador "trava". Caso o consumo de cpu assuma valores próximos a 100%, significa que toda capacidade de processamento oferecida pelo computador foi exigida. Se as aplicações em execução demandarem mais recursos, um processamento acima do valor máximo oferecido pelo computador é solicitado, e em conseqüência disso, a falha ocorre. O monitoramento proposto consiste em ficar, periodicamente, medindo o consumo de cpu. Caso as medidas assumam valores elevados (maiores que 70 % em média), significa que em grande parte do tempo a capacidade total de processamento foi exigida, e um upgrade de processador seria recomendado. Por outro lado, se as medidas assumirem valores baixos (valores menores que 10%, em média), é possível afirmar que os recursos computacionais oferecidos pela máquina estivem ociosos em grande parte do tempo.

No projeto em questão, foi desenvolvido o plug-in cpu.dll. A seguir, veremos em detalhes, a implementação do módulo.

3.3.1 Dados Coletados

Neste plug-in, monitoramos dois parâmetros relacionados ao processador da máquina: consumo de cpu e quantidade de processadores. Para armazenar esses dados, foi criada uma estrutura interna, como podemos ver no diagrama, abaixo:

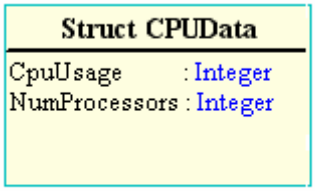

Figura 3.4 Estrutura CPUData

Tabela 3.6: Dados coletados pelo plug-in

Parâmetro	Descrição	Unidade
CpuUsage	Consumo de CPU	Percentual (%)
NumProcessors	Quantidade de processadores	Adimensional

3.3.2 Classe CCpuMethod

Durante o estudo para analisar as rotinas para capturar as informações apresentadas na tabela 3.6, observou-se que os métodos de coleta desses dados diferem-se entre as versões do Windows. Enquanto que nas versões do Windows 9x (Windows 95, Windows 98 e Windows Me) os dados podem ser capturados a partir dos registros, nas versões mais recentes (Windows NT 4.0, Windows 2000, Windows XP e Windows 2003 Server) essas informações podem ser coletadas a partir de contadores de performance disponibilizados na API do sistema operacional.

Como solução desse impasse, foi proposta a criação de uma classe abstrata, com um método de coleta puramente virtual. A implementação das rotinas específicas consistiu no uso dos recursos oferecidos pelas características de herança do paradigma OO, onde foram criadas duas classes filhas.

A figura 3.5 ilustra o diagrama de classes proposto para implementação da solução descrita no parágrafo anterior. A classe CCpuMethod consiste em uma estrutura genérica, com um método virtual, a função *GetData* responsável pela coleta dos dados. Essa estrutura é herdada pelas classes CCPUNT e CCPU9x que contemplam as soluções para os dois tipos de plataforma.

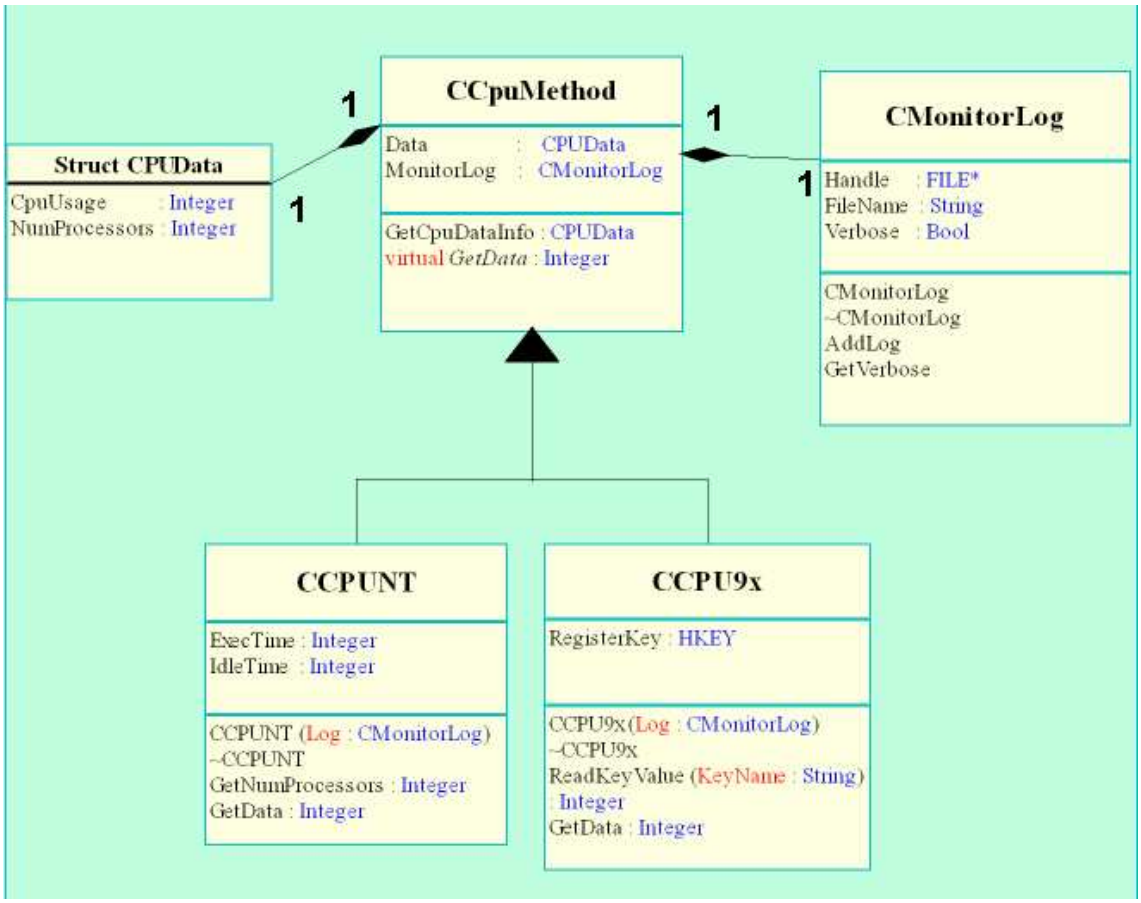

Figura 3.5 Diagrama das classes do plug-in de cpu

3.3.2.1 Classe CCPUNT

A classe CCPUNT, conforme foi visto, implementa a solução específica de coleta de dados de cpu para as versões do Windows NT (Windows NT 4.0, Windows 2000, Windows XP e Windows 2003). A rotina que possui essa implementação é o método virtual GetData.

A coleta dos dados consiste em capturar dados dos contadores de performance do Windows. Esses contadores atualizam essas informações a cada ciclo de 100ns. Para calcular o consumo de cpu, é preciso medir o tempo que o sistema ficou ocioso durante esse ciclo. Analogamente, para saber o parcela de tempo em que o processador foi utilizado, bastaria subtrairmos do intervalo do ciclo (100 ns) o tempo de ociosidade do processador. As equações abaixo ilustram os cálculos de consumo de cpu.

$$ExecTime = 100ns - IdleTime$$

$$CPUUsage = 100 * (ExecTime / 100ns * Num\,Processors)$$

Tabela 3.7: Parâmetros do cálculo de consumo de cpu

Parâmetro	Descrição	Unidade
IdleTime	Parcela de tempo em que o processador ficou ocioso	Ns
ExecTime	Quantidade de tempo de utilização do processador	Ns
NumProcessors	Número de processadores instalados na máquina	Adimensional
CPUUsage	Consumo de CPU	%

A quantidade de processadores instalados no computador pode ser obtida através da função *GetSystemInfo* da API do Windows. O contador que possui as medidas de tempo de utilização e ociosidade dos processadores é o *PERF_100NSEC_TIMER_INV.*

3.3.2.1 Classe CCPU9x

Esta classe implementa a rotina de monitoramento para as versões do Windows 9x (Windows 95, Windows 98 e Windows Me). Assim como na classe CCPUNT, a coleta dos dados consiste em capturar os valores dos contadores de performance. A diferença entre esses dois métodos de busca, está no fato que nas versões mais antigas os contadores encontram-se nos registros do Windows.

Para capturar o consumo de cpu, basta ler os valores da chave *HKEY_DYN_DATA\ PerfStats\StartStat\ KERNEL\CPUUsage.*

3.4 Memória

A quantidade de dados processados pode ser observada através do monitoramento do consumo de memória. Esse parâmetro constitui-se em uma propriedade fundamental para mensurar a capacidade computacional e determinar a performance de uma aplicação.

Sabemos que a memória do computador pode ser classificada como memória física (quantidade de memória RAM disponível) e memória virtual (endereços virtuais usados pelos programas em tempo de execução). A capacidade total de memória de uma máquina pode ser definida como a soma desses dois parâmetros. Essa expressão é conhecida como memória total do computador. O monitoramento deste recurso computacional tem como objetivo capturar o valor da quantidade de memória total disponível, e a parcela desse valor que está sendo utilizada pelo computador.

O monitoramento do consumo de memória de um computador constitui-se em uma grande ferramenta de análise do comportamento dos aplicativos em execução. A demanda por consumo de memória será proporcional à complexidade e

quantidade das instruções processadas pelo programa. Por outro lado, quanto maior a oferta de memória disponível no computador, melhor será a performance (menor tempo de resposta) resultante da execução do programa.

Assim como o consumo de CPU, esse monitoramento é capaz de explicar porque a máquina "trava" durante a execução de alguns programas. Quanto maior o valor do consumo de memória (valores próximos à oferta de recurso disponível) menor será a capacidade de processamento. Caso a razão entre consumo e oferta aproxime-se de 1, temos a situação onde toda capacidade de processamento oferecida foi exigida e a aplicação retorna a seguinte mensagem de erro: "Out of Memory".

Para o projeto, foi desenvolvido o plug-in memory.dll. Os itens a seguir, descrevem em detalhes a implementação do componente.

3.4.1 Dados Coletados

Neste plug-in, monitoramos dois parâmetros relacionados à quantidade de memória processada na máquina: memória total disponível e memória utilizada. Para armazenar esses dados, foi criada uma estrutura interna, como podemos ver no diagrama, abaixo:

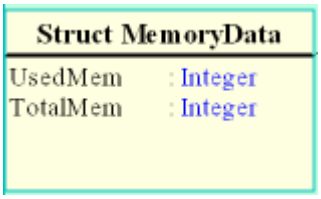

Figura 3.6 Estrutura MemoryData

Tabela 3.8: Dados coletados pelo plug-in

Parâmetro	Descrição	Unidade
UsedMem	Memória consumida	MB
TotalMem	Quantidade de memória disponível	MB

3.4.2 Classe CMemoryMethod

Essa estrutura, assim como a CCpuMethod, trata-se de uma classe abstrata que possui um método virtual responsável pela coleta dos dados. Enquanto que no caso da coleta dos dados CPU, as rotinas eram diferentes para as versões 9x e NT, no caso da memória, uma mesma função pode ser usada nos dois ambientes para capturar as informações. O uso de uma classe abstrata, com a implementação da rotina GetData em uma outra classe que herdaria as características da classe CMemoryMethod, pode ser justificado como uma maneira de manter uma

modelagem única entre os plug-ins, o que facilitaria o entendimento do código. Além disso, caso seja necessário coletar essas mesmas informações de outro ambiente (LINUX, por exemplo), bastaria criar uma nova classe filha herdando as características da classe CMemoryMethod. Com isso, contribuímos para a portabilidade do sistema, sem alterações nos módulos existentes, facilitando também a manutenção do código.

A figura 3.6 ilustra o diagrama de classes usado no plug-in memory.dll. Note que o diagrama abaixo é muito semelhante ao diagrama de classes do plug-in cpu.dll.

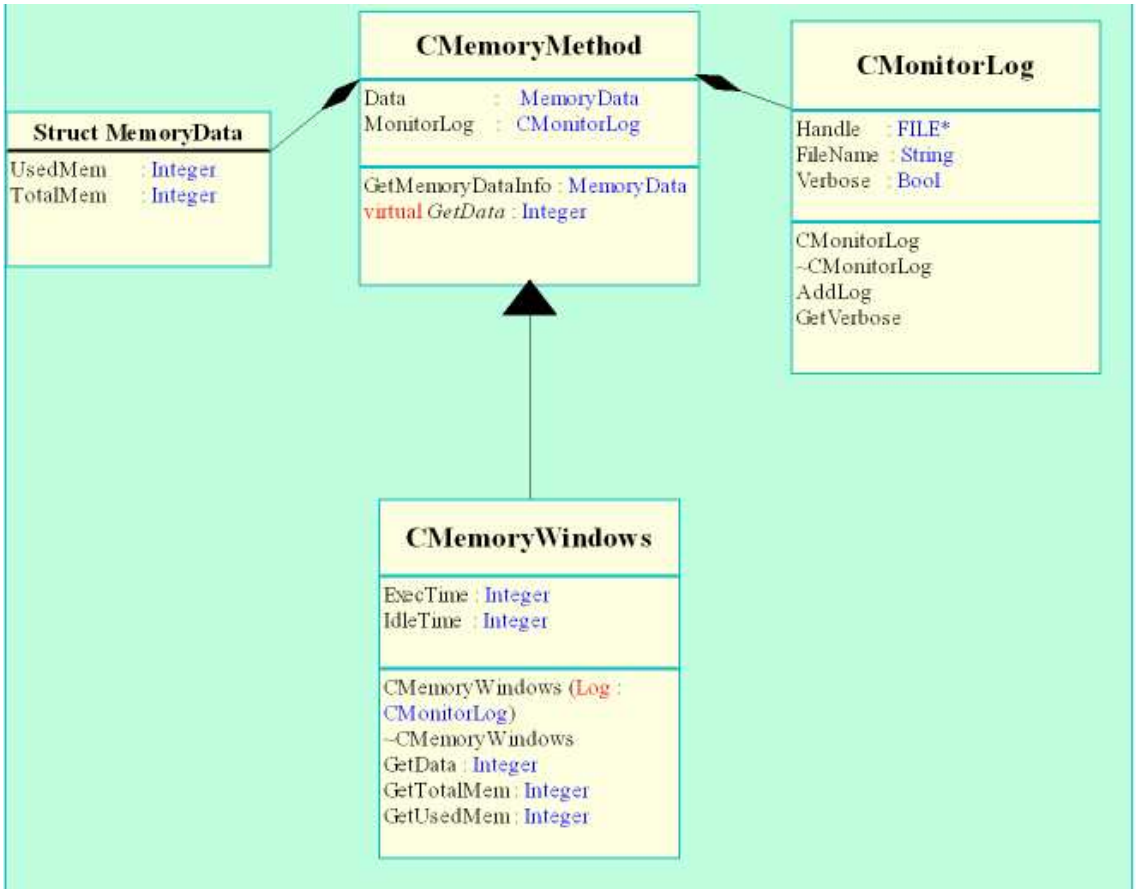

Figura 3.7 Diagrama de classes do plug-in memory.dll

3.4.3 Classe CMemoryWindows

Trata-se da classe que contém a rotina específica de coleta dos dados (no ambiente Windows) relacionados na estrutura CMemoryData. Essa função está implementada no método GetData.

Para capturar os dados de consumo e quantidade de memória do computador, o método usa a função GlobalMemoryStatus da API do Windows. Essa função retorna por referência uma estrutura, MEMORYSTATUS, contendo as informações

sobre a utilização de memória em um determinado instante. A partir dos atributos dessa estrutura, coleto as informações descritas na estrutura CMemoryData.

3.5 Processos

Trata-se de um dos conceitos mais importantes do sistema operacional e pode ser definido com qualquer programa em execução. A metodologia do monitoramento dos processos de um computador consiste em capturar a lista dos programas em execução com seus respectivos consumos de memória e cpu. Esse monitoramento é uma ação complementar às analises de cpu e memória descritas nos itens 3.3 e 3.4. A diferença é que o computador não é mais visto como um todo, mas abordamos uma ótica para cada aplicativo em execução. A idéia é medir a quantidade de recurso computacional (memória e cpu) exigida por uma determinada aplicação. O plug-in desenvolvido para essa análise foi a DLL process.dll.

A partir da lista de processos coletada no monitoramento, é possível definir quais são as aplicações críticas, isto é, quais programas demandam mais recursos computacionais. Em muitos casos, podemos estar julgando a performance de um programa, quando na verdade, quem está demandando a maior parte da capacidade computacional oferecida por um computador pode ser um outro aplicativo (por exemplo, a execução de um programa anti-vírus). Se isso estiver ocorrendo, podemos agendar diferentes horários para execução dos programas, evitando os processos concorrentes.

Outro resultado bastante interessante é a possibilidade de definir o consumo médio de memória exigido por uma determinada aplicação. Após um longo período de monitoramento, teríamos uma quantidade razoável de amostras de consumo de memória. Com base nesses dados, poderíamos usar métodos estatísticos para normalizar os valores, e em seguida, teríamos como resultado uma média e um desvio padrão. A partir dos valores calculados, seria possível mensurar a quantidade de memória mínima e recomendada, com precisão. Com isso, temos uma ferramenta capaz de ajudar equipes de desenvolvimento na especificação dos requisitos mínimos e recomendados para execução do programa.

A seguir, as implementações das classes e estruturas desenvolvidas no plug-in, serão descritas em detalhes.

3.5.1 Dados Coletados

No plug-in de processos, coletamos quatro informações distintas para cada processo em execução: identificador do processo (PID), consumo de CPU, consumo

de memória e nome do programa. Assim como nos outros módulos, foi criada uma estrutura interna, a ProcessData, para armazenar essas informações.

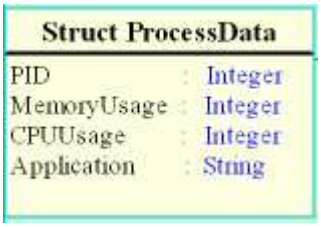

Figura 3.8 Estrutura ProcessData

Tabela 3.9: Dados armazenados na estrutura ProcessData

Parâmetro	Descrição	Unidade
PID	Identificador do processo	Adimensional
MemoryUsage	Consumo de memória	MB
CPUUsage	Consumo de CPU	%
Application	Nome do programa em execução	Adimensional

3.5.2 Classe CProcessMethod

Assim como nos demais plug-ins, também foi implementada uma classe abstrata, procurando manter uma modelagem semelhante com os outros módulos. A classe CProcessMethod é a implementação da classe abstrata. Diferente dos demais módulos, ela possui uma lista de estruturas de armazenamento de dados ao invés de um único atributo. Isso acontece porque nesse caso, deveremos ter mais de um processo em execução, e em função disso surgiu a necessidade de criamos uma lista (representada pela estrutura de dados vector do STL) para armazenar dados de todos os aplicativos que estão sendo executados no momento do monitoramento.

Durante a fase de estudo das rotinas de coleta de dados nos diferentes ambientes, foi observado que para os sistemas operacionais da família Windows 9x, há restrições quanto à captura das informações apresentadas na estrutura da figura 3.7. Nesses ambientes, a API do Windows permite somente que capturemos o identificador do processo (PID) e o nome do processo em execução. Para os sistemas mais recentes (NT 4.0, 2000, XP e 2003) não há essa restrição. Em função disso, para o escopo do projeto, o monitoramento restringiu-se aos ambientes mais recentes, não incluindo os sistemas operacionais da família Windows 9x. Essa implementação pode ser vista através da classe CProcessNT na figura 3.8.

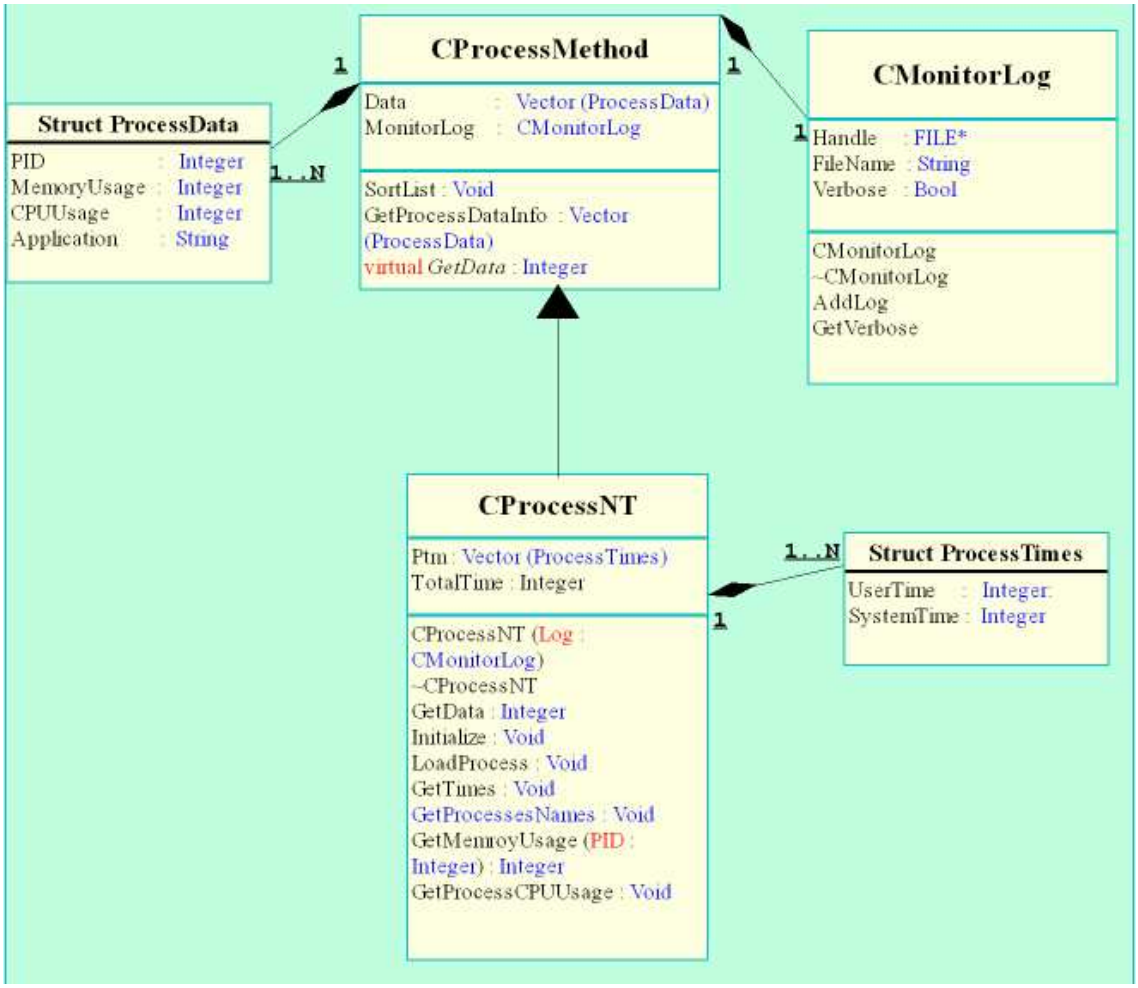

Figura 3.9 Diagrama de classes do plug-in process.dll

3.5.3 Classe CProcessNT

A classe CProcessNT, conforme foi visto, implementa a solução específica de coleta de dados dos processos para as versões do Windows NT (Windows NT 4.0, Windows 2000, Windows XP e Windows 2003). A rotina que possui essa implementação é o método virtual GetData.

A coleta dos dados consiste, inicialmente, em enumerar todos os processos em execução, armazenando-os em uma lista. Nesta etapa, os PIDs, que são os elementos usados para identificar um processo, são carregados. O cálculo do consumo de cpu de cada pode ser feito de maneira análoga ao cálculo de CPU da máquina como um todo. Para um determinado intervalo de tempo, mede-se a parcela desse tempo usada por cada processo. A razão entre a parcela de tempo usada pelo processo e o intervalo de tempo total é o consumo de cpu. Note no diagrama da figura 3.8 que a classe CProcessNT possui dois atributos: Ptm, que é a lista dos tempos de cada processo, e TotalTime, que representa o intervalo de tempo total. As equações abaixo, ilustram esses cálculos:

$$i \in \mathrm{N}$$

$$0 \le i \le Num\,\mathrm{Pr}\,ocess$$

$$CPUUsage(i) = 100 * \left((Ptm(i).UserTime + Ptm(i).SystemTime) \Big/ TotalTime \right)$$

$$\sum_{i=0}^{i=Num\,\mathrm{Pr}\,ocess} CPUUsage(i) = 100$$

Tabela 3.9 Parâmetros do cálculo de consumo de cpu por processo

Parâmetro	Descrição	Unidade
NumProcess	Número de processos em execução	Adimensional
UserTime	Parcela do processo criada pelo usuário do sistema operacional	Ns
SystemTime	Parcela do processo criada pelo usuário system	Ns
TotalTime	Intervalo de tempo decorrido	Ns
CPUUsage	Consumo de CPU	%

A lista dos processos em execução, os tempos de execução de cada aplicativo, o nome de cada programa e o consumo de memória podem ser carregados através das funções *EnumProcesses*, *GetProcessTimes*, *GetModuleBaseName* e *GetProcessMemoryInfo* respectivamente. Essas rotinas estão armazenadas na biblioteca dinâmica PSAPI.DLL.

3.6 Configurações de Rede

Uma rede de computadores corporativa pode ter um número elevado de máquinas conectadas, dependendo da dimensão da estrutura da empresa. As configurações de rede de cada computador constituem-se de um meio de identificarmos uma máquina no universo de computadores da empresa. Conforme será visto mais adiante no capítulo 5 (Integração com o banco de dados), esses parâmetros serão usados para identificar qual foi o computador onde foi realizado o monitoramento. Esse monitoramento permite a geração de inventário preciso, com o número total de máquinas que estão sendo monitoradas.

Na situação ideal, o software cliente instalado em todas as máquinas da empresa, teríamos com precisão o número total de computadores conectados à rede. Essa informação é um grande ferramental na arquitetura de TI, em que ajuda o trabalho dos administradores de rede. Na ótica do gestor, essa ferramenta também apresenta alto valor agregado à medida que apresenta um valor, interpretado como medida de controle, podendo ser encaminhado para diversas áreas como auditoria, contabilidade (contabilização de ativos permanentes), custos, etc.

Como resultado do desenvolvimento desse módulo, temos o plug-in network.dll. Nos próximos itens, temos o detalhamento da implementação dessa DLL.

3.6.1 Dados Coletados

No plug-in, foram coletadas três informações relacionadas à configuração de rede da máquina: o nome do computador (hostname), endereço IP e o endereço da placa de rede (MAC Address). Para armazenar esses dados, foi a criada a estrutura NetworkData, podendo ser vista através da figura 3.9.

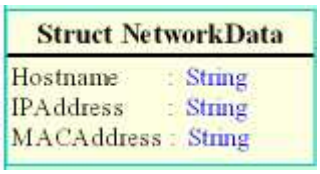

Figura 3.10 Estrutura NetworkData

Tabela 3.10 : Dados armazenados na estrutura NetworkData

Parâmetro	Descrição	Unidade
Hostname	Nome da máquina	Adimensional
IPAddress	Endereço IP	Adimensional
MACAddress	Endereço físicico da placa de rede	Adimensional

3.6.2 Classe CNetworkMethod

Seguindo a modelagem dos demais plug-ins, temos a classe abstrata CNetworkMethod. Assim como nos modelos anteriores, essa classe contém um método puramente virtual, o GetData. Essa função é implementada nas classes filhas, onde serão alocadas as rotinas de captura das informações descritas na tabela 3.10. A figura 3.10 ilustra o diagrama de classes do módulo.

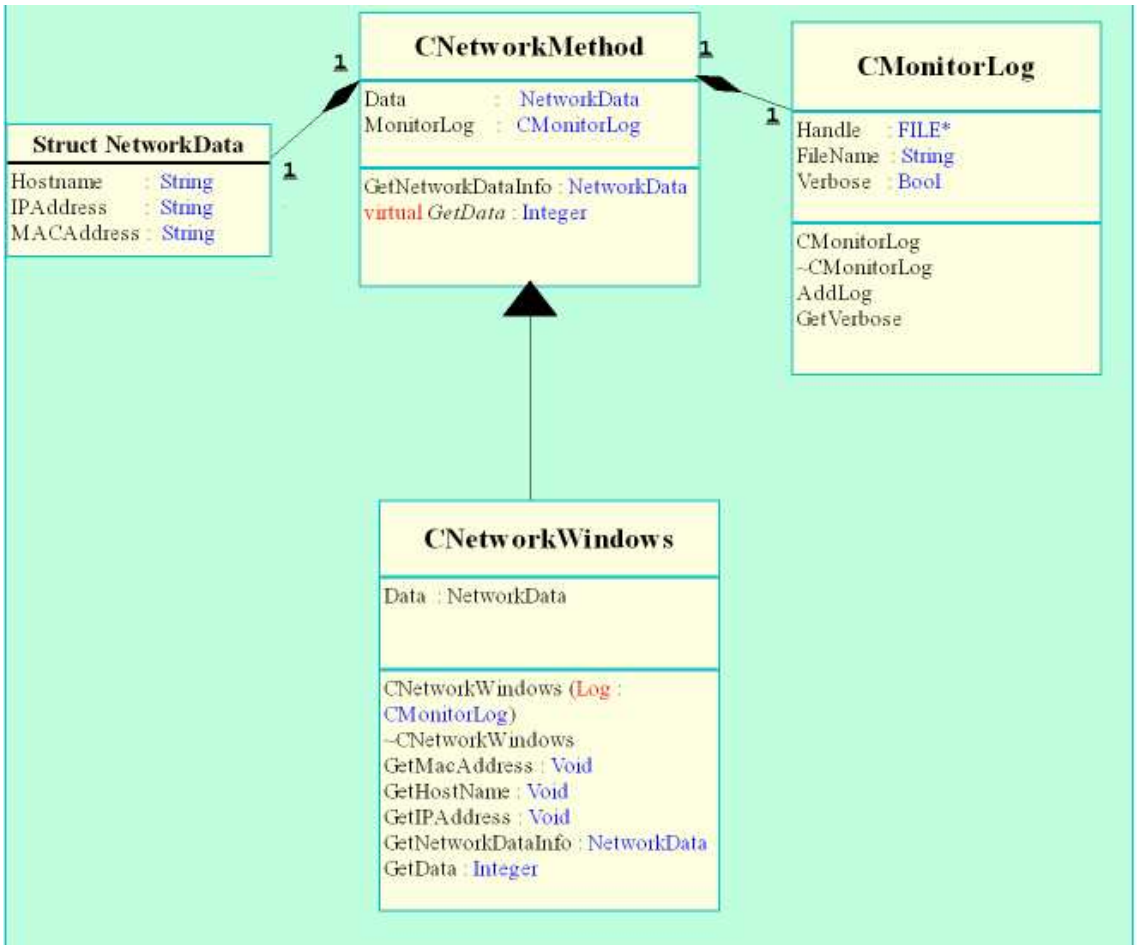

Figura 3.11 Diagrama de classes do plug-in network.dll

3.6.3 Classe CNetworkWindows

A classe CNetworkWindows implementa a solução específica de coleta das configurações de rede para a plataforma Windows. Para coletar o hostname e o endereço IP do computador foram usadas as rotinas *gethostbyname* e *inet_ntoa* da biblioteca Winsock. A identificação do Mac Address foi feita através da rotina *Netbios*, armazenada na biblioteca do NetAPI32.

Tabela 3.11: Descrição dos métodos da classe CNetworkWindows

Método	Descrição	Tipo
CNetworkWindows	Construtor da classe	Público
~ CNetworkWindows	Destrutor da classe	Público
GetMacAddress	Coleta o endereço MAC	Privado
GetHostName	Coleta o nome do computador	Privado
GetIPAddress	Coleta o endereço IP	Privado
GetNetworkDataInfo	Retorna a estrutura de dados coletada	Pública
GetData	Rotina principal da classe. Executa as rotinas de coleta de dados.	Pública

3.7 Softwares Instalados

A proposta desse monitoramento consiste em coletar periodicamente a lista dos softwares instalados no computador. Como resultado desse monitoramento, temos mais informações para completarmos o inventário do computador. Esses dados também podem ser usados para determinar com precisão o número de licenças requisitadas por um determinado software (Microsoft Office, por exemplo). Além disso, o monitoramento verifica se os programas instalados estão dentro da política da empresa, isto é, analisa a existência de aplicativos não autorizados no computador.

O plug-in responsável por esse monitoramento é a DLL software.dll, sendo descrita nas próximas seções desse capítulo.

3.7.1 Dados Coletados

No plug-in de software, coletamos um único parâmetro: a lista de softwares instalados. A figura 3.11 ilustra e estrutura criada para armazenar esses dados.

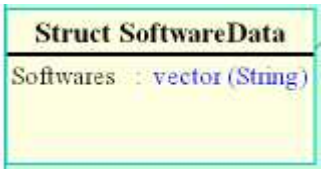

Figura 3.12 Estrutura SoftwareData

3.7.2 Classe CSoftwareMethod

Seguindo a modelagem dos demais plug-ins, temos a classe abstrata CSoftwareMethod. Assim como nos modelos anteriores, essa classe contém um método puramente virtual, o *GetData*. Essa função é implementada nas classes filhas, onde será alocada a rotina para carregar a lista de programas instalados na máquina. O diagrama de classes desse plug-in está ilustrado na figura 3.12.

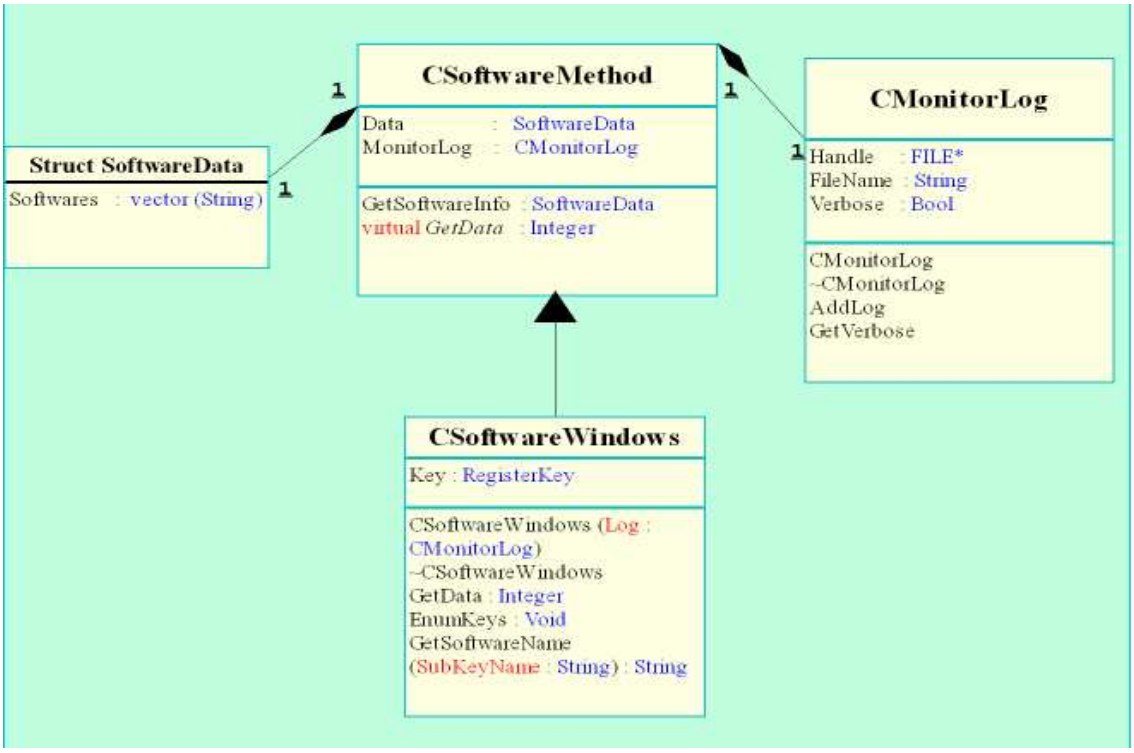

Figura 3.13 Diagrama de classes do plug-in Software.dll

3.6.3 Classe CSoftwareWindows

A classe CSoftwareWindows implementa a solução específica de coleta dos programas instalados na plataforma Windows. A lista dos aplicativos é a mesma vista no menu *"Adicionar ou remover programas"* do Painel de Controle do Windows. A enumeração desses programas consistem em carregar a chave do registro HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall. Em seguida, uma varredura por todas as sub-chaves é feita. Cada elemento percorrido contém o valor de registro *DisplayName*, que armazena o nome do programa instalado. Essa rotina está implementada no método *GetSoftwareName*, enquanto que o atributo *Key* está associado à chave do registro onde a lista dos programas é armazenada.

3.8 I/O

O objetivo desse monitoramento consiste na análise da quantidade de dados que é carregada e armazenada nos discos rígidos. O desempenho do processamento de dados de um computador está diretamente relacionado com a performance do disco rígido. Quanto mais rápido for o acesso ao HD para carregar e salvar dados, menor será o tempo de resposta do processamento de CPU. Isso pode ser justificado pelo fato que enquanto a informação está sendo salva ou carregada do disco, o resto do programa permanece em um estado de espera, até

que essa rotina seja concluída. Como podemos notar, aplicações que demandam escrita e leitura de dados (manipulação de arquivos, por exemplo) tem o tempo de resposta impactado pela performance do disco rígido.

Para o monitoramento da performance do HD, são coletadas as seguintes informações: número de operações de leitura, quantidade de bytes lidos, número de operações de escrita, quantidade de bytes escritos. Para avaliação da performance do HD, podemos fazer as seguintes correlações: caso o número de operações de leitura cresça, enquanto que a quantidade de bytes lidos diminua podemos afirmar que um número maior de operações está resultando em menos informação carregada. Com isso, o tempo de acesso de leitura aumentaria e em função disso, temos uma queda na performance do disco. Analogamente, teríamos a mesma situação para escrita dos dados.

O plug-in io.dll foi desenvolvido para o monitoramento de I/O. As próximas seções detalham o processo de desenvolvimento dessa biblioteca.

3.8.1 Dados Coletados

Conforme pode ser visto na figura 3.13, quatro parâmetros são monitorados no plug-in Io.dll: número de operações de leitura, quantidade de bytes lidos, número de operações de escrita e quantidade de bytes escritos.

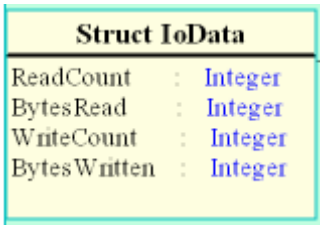

Figura 3.14 Estrutura de dados IoData

Tabela 3.12 : Dados armazenados na estrutura IoData

Parâmetro	Descrição	Unidade
ReadCount	Número de operações de leitura	Adimensional
BytesRead	Número de bytes lidos	Bytes
WriteCount	Número de operações de escrita	Adimensional
BytesWritten	Número de bytes escritos	Bytes

3.8.2 Classe CIoMethod

Assim como no caso do monitoramento de CPU, os métodos de coleta desses dados diferem-se entre as versões do Windows. Enquanto que nas versões do Windows 9x (Windows 95, Windows 98 e Windows Me) os dados podem ser capturados a partir dos registros, nas versões mais recentes (Windows NT 4.0, Windows 2000, Windows XP e Windows 2003 Server) essas informações podem ser coletadas a partir de contadores de performance disponibilizados na API do sistema operacional.

Análogo ao caso do monitoramento de CPU, a solução desse impasse consiste na criação de uma classe abstrata, com um método de coleta puramente virtual. A figura 3.14 ilustra o diagrama de classes implementado no plug-in. A classe CIoMethod é uma estrutura genérica, com um método virtual, a função *GetData* responsável pela coleta dos dados. Essa estrutura é herdada pelas classes CIoNT e CIo9x que contemplam as soluções para os dois tipos de plataforma.

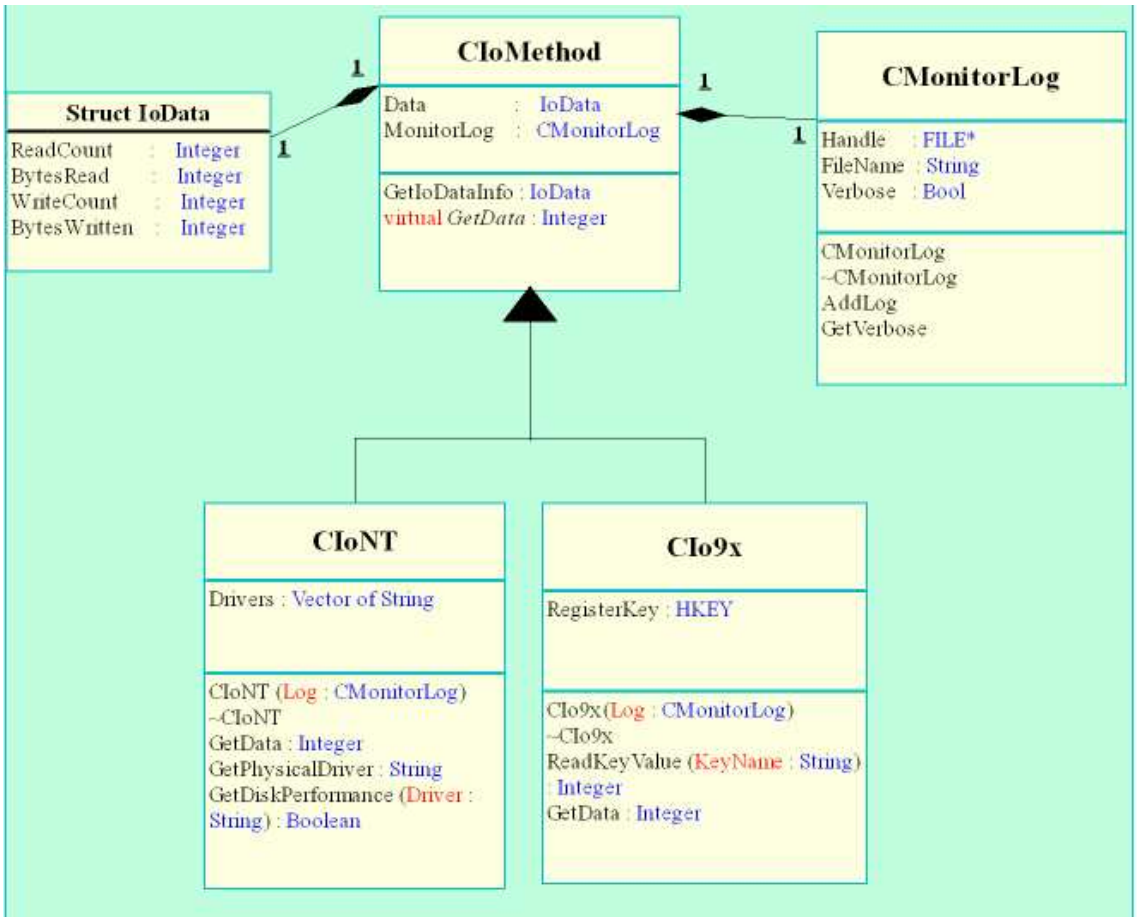

Figura 3.15 Diagrama de classes do plug-in Io.dll

3.8.2.1 Classe CIoNT

A classe CIoNT, conforme foi visto, implementa a solução específica de coleta de dados de cpu para as versões do Windows NT (Windows NT 4.0, Windows 2000, Windows XP e Windows 2003). A rotina que possui essa implementação é o método virtual GetData.

Assim como o monitoramento de CPU, a coleta dos dados consiste em capturar dados dos contadores de performance do Windows. A API do Windows usada para capturar as informações de performance dos discos rígidos é a rotina *DeviceIoControl*. Nessa função, a estrutura `DISK_PERFORMANCE_W2K` é passada por referência. Os que queremos buscar correspondem aos atributos *`BytesRead, BytesWritten, ReadCount e WriteCount`* dessa estrutura.

3.8.2.2 Classe CIo9x

Esta classe implementa a rotina de monitoramento para as versões do Windows 9x (Windows 95, Windows 98 e Windows Me). Assim como na classe CIoNT, a coleta dos dados consiste em capturar os valores dos contadores de performance. A diferença entre esses dois métodos de busca, está no fato que nas versões mais antigas os contadores encontram-se nos registros do Windows.

Para capturar os parâmetros basta ler os valores da chave *HKEY_DYN_DATA\PerfStats\StartStat\VFAT (`BReadsSec, BWritesSec, ReadsSec, WritesSec`)*.

Capítulo 4

Arquitetura Cliente-Servidor

Conforme foi visto no capítulo 2, o sistema foi desenvolvido baseado em uma arquitetura cliente-servidor. Esse paradigma determina a existência de dois componentes básicos: clientes e servidores. Os clientes são os computadores que solicitam informações e serviços, enquanto que os servidores são os computadores que respondem esses pedidos.

No contexto do sistema desenvolvido, os clientes são as máquinas onde estão instalados os plug-ins e o agente (aplicativo MonitorAgent.exe). O papel do software cliente é carregar, periodicamente, as rotinas de monitoramento dos plug-ins e enviar os dados coletados através da rede de computadores. Os servidores são os computadores que recebem esses dados, identificam as informações recebidas e armazenam os resultados em uma base de dados. Essa rotina é feita através do utilitário MonitorReceiver.exe. Os eventos executados pelas interfaces de rede cliente e servidor podem ser ilustrados pelo diagrama de seqüência de eventos apresentado na figura 4.1.

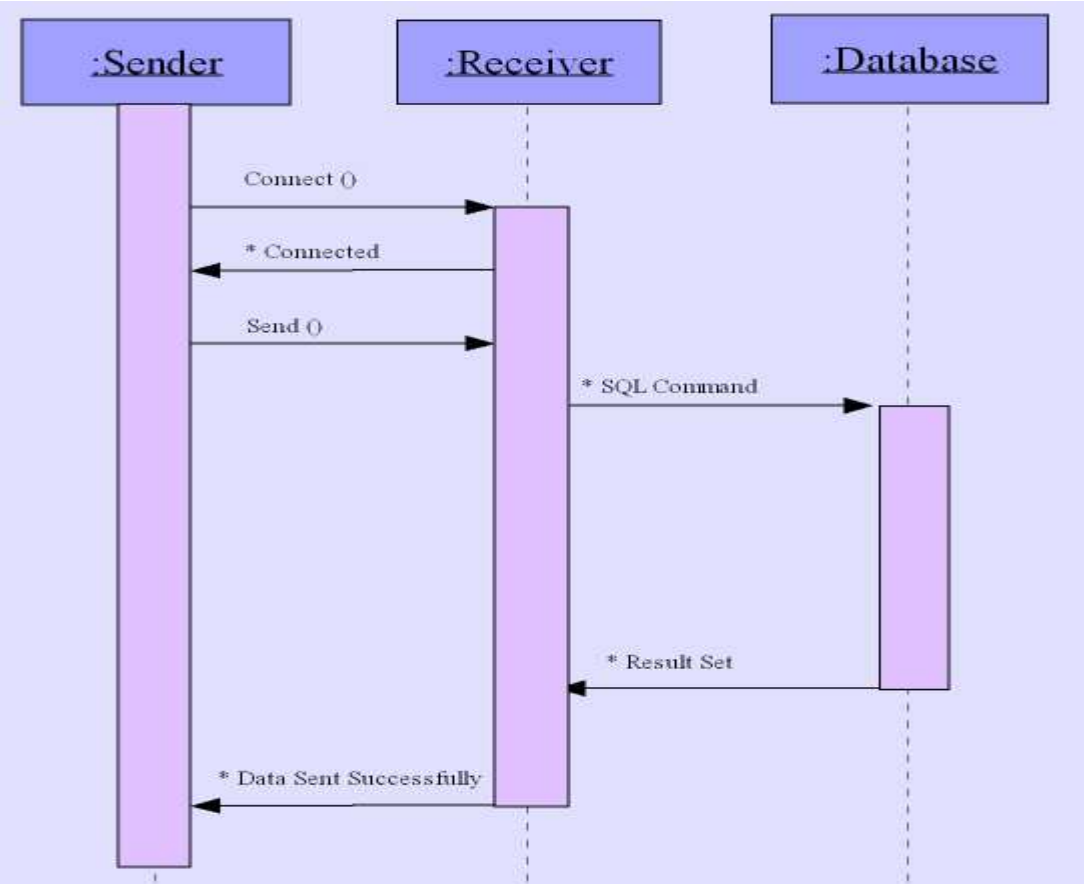

Figura 4.1 Diagrama de seqüência dos eventos da arquitetura Cliente Servidor

A estrutura do sistema também pode ser descrita na forma de interfaces front-ends e back-end. Um programa front-end pode ser definido como uma aplicação que exige interação direta do usuário. O software cliente pode ser associado a uma interface front-end, pois exige que o usuário inicie o serviço e configure as entradas, informando quais plug-ins devem ser ativados com suas respectivas periodicidades de execução. Um programa back-end é definido com um utilitário que realiza uma determinada tarefa sem a intervenção direta do usuário. O módulo do servidor pode ser comparado a um back-end remoto, onde o serviço recebe os dados através da rede e, em seguida, interpreta a informação, armazenando-a em uma base de dados. Como podemos ver, os processos executados no servidor são transparentes na ótica do usuário, sem exigir qualquer intervenção do mesmo.

As seções desse capítulo descrevem a implementação das interfaces cliente e servidor do sistema, mostrando os processos que são executados após a coleta de dados realizada pelos plug-ins. Uma breve apresentação do protocolo de rede utilizado bem como as classes criadas para execução dessas tarefas são mostrados nesse capítulo.

4.1 Protocolo de Rede

O protocolo utilizado para enviar e receber os dados através da arquitetura de redes corporativa foi o TCP\IP. Trata-se de um protocolo da camada de transportes com conexão orientada (uma conexão entre o servidor e o cliente deve ser estabelecida antes da troca de dados) e confiança.

Essa última característica pode ser considerada como justificativa para escolha desse protocolo em relação ao modelo UDP. Para o sistema desenvolvido, é importante garantir que os dados coletados sejam recebidos pelo back-end remoto. O protocolo TCP, em caso de falha durante a transmissão dos dados, retransmite o pacote de informação garantindo que os dados cheguem ao "receiver" de maneira correta.

Apesar de garantir a confiabilidade do sistema, a utilização desse protocolo pode gerar algumas limitações do projeto. Firewalls são ferramentas utilizadas em arquiteturas de rede para bloquear acessos não autorizados (ataques de hackers, por exemplo). Se a porta TCP utilizada para enviar\receber os dados monitorados estiver na lista de portas bloqueadas pelo Firewall, a transmissão das informações não será realizada, inviabilizando o monitoramento remoto.

Para ilustrar essa situação descrita, imagine o seguinte cenário de uso: poderíamos usar o sistema para monitorar os recursos computacionais de uma máquina da rede DEL, como o computador www.del.ufrj.br. O software cliente seria instalado nessa máquina para coletar as informações pertinentes à performance dos recursos computacionais. De maneira análoga, poderíamos instalar a interface servidor em uma máquina externa à rede DEL, como um computador doméstico com IP fixo. Logo, estaríamos usando a Internet como meio de transmissão dos dados monitorados. É provável que exista algum firewall na rede DEL bloqueando a porta TCP usada no sistema, e com isso transmissão dos dados coletados não seria realizada.

Uma maneira de contornar o problema apresentado quanto as limitações do uso do protocolo TCP seria a negociação com o administrador de rede, solicitando que o acesso a uma porta TCP específica seja permitida para um determinado computador externo, necessariamente com IP fixo. Infelizmente sabemos que nem sempre essa solução é possível devido a restrições na política de segurança das empresas e instituições. Uma outra alternativa seria o uso do protocolo HTTP, usado nas aplicações WEB. Essa proposta pode ser implementada nas versões futuras do projeto.

4.2 Dados Transmitidos

No capítulos 3, foram apresentados os dados coletados por cada plug-in. Essa seção apresentará a estrutura de dados transmitida através da rede por cada um dos plug-ins descritos anteriormente.

Cada pacote de dados transmitido deve informar, além das informações coletadas, qual computador está sendo monitorado bem como o plug-in que realizou a coleta. Para que isso seja possível, duas informações são acrescentadas no inicio do pacote de dados: endereço IP do computador cliente e identificador do plug-in de monitoramento.

O endereço IP é um modelo de dados que identifica um computador ou outro dispositivo qualquer (impressoras, por exemplo) conectado à rede. Trata-se de um atributo composto de um endereço numérico de 32 bits escritos como 4 números, também conhecidos como octetos, separados por pontos.

O identificador do plug-in pode é um atributo numérico (constante) que identifica a DLL que realizou um determinado monitoramento. A tabela 4.1 enumera os identificadores dos plug-ins desenvolvidos no projeto.

Tabela 4.1: Lista dos identificadores dos plug-ins

Constante	Plug-in	Valor
monitorCPU	Cpu.dll	0
Monitório	Io.dll	1
monitorMEMORY	Memory.dll	2
monitorNETWORK	Network.dll	3
monitorOS	Os.dll	4
monitorPROCESS	Process.dll	5
monitorSOFTWARE	Software.dll	6

As próximas tabelas descrevem os dados que são transmitidos por cada um dos plug-ins.

Tabela 4.2: Dados enviados pelo plug-in Cpu.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
Cpu Usage	Consumo de CPU (%)	Inteiro	4
Num Processors	Número de processadores instalados	Inteiro	4

Tabela 4.3: Dados enviados pelo plug-in Memory.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
Total Memory	Quantidade de memória disponível (MB)	Inteiro	4
Used Memory	Quantidade de memória consumida (MB)	Inteiro	4

Tabela 4.4: Dados enviados pelo plug-in Io.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
Total Bytes Read	Quantidade de bytes lidos	Inteiro	4
Number of Read Operations	Quantidade de operações de leituras	Inteiro	4
Total Bytes Written	Quantidade de bytes escritos	Inteiro	4
Number of Written Operations	Quantidade de operações de escrita	Inteiro	4

Tabela 4.5: Dados enviados pelo plug-in Os.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
Operating System	Versão do sistema operacional	String	Variável

Tabela 4.6: Dados enviados pelo plug-in Process.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
Processes	Número de processos	Inteiro	4
Enviar os dados abaixo para cada processo			
Process Name	Nome do processo	String	Variável
CPU Usage	Consumo de CPU (%)	Inteiro	4
Memory Usage	Memória consumida (MB)	Inteiro	4

Tabela 4.7: Dados enviados pelo plug-in Network.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
HostName	Nome do computador	String	Variável
MAC Address	Identificador da placa de rede	String	15

Tabela 4.8: Dados enviados pelo plug-in Software.dll

Campo	Descrição	Tipo de dados	Tamanho (bytes)
Host	Endereço IP do computador monitorado	String	15
MonitorID	Identificador do plug-in	Inteiro	4
Num. Softwares	Número de softwares instalados	Inteiro	4
Enviar os dados abaixo para cada software			
Software	Nome do programa instalado	String	Variável

4.3 Biblioteca WinSock 2

A biblioteca WinSock 2 foi usada durante o desenvolvimento da arquitetura cliente-servidor. A DLL WSOCK32.dll armazena as funcionalidades dessa biblioteca, sendo responsável pela interface entre a aplicação com as interfaces do protocolo de rede utilizado (no nosso caso, o TCP\IP).

Abaixo são listadas as principais características dessa biblioteca:

- Suporte a múltiplos protocolos de rede
- Independência de protocolo na transmissão de dados
- Garantia na qualidade do serviço (QoS)
- Compartilhamento de sockets através de diferentes processos

Dos itens apresentados acima, os dois últimos podem ser considerados como justificativa na escolha dessa biblioteca. Para o projeto em questão, temos a preocupação de garantir qualidade para os requisitos não funcionais (como

desempenho da aplicação). O fator QoS na interface de rede é uma métrica usada para garantir esse requisito. O último item refere-se ao compartilhamento dos sockets a partir de diferentes processos. Essa característica viabiliza o desenvolvimento do projeto, pois como veremos mais adiante, a nossa arquitetura de rede está subdivida em duas interfaces diferentes (cliente e servidor). O compartilhamento do socket permite que a aplicação seja executada na mesma máquina, onde teríamos diferentes processos (cliente e servidor) acessando o mesmo descritor.

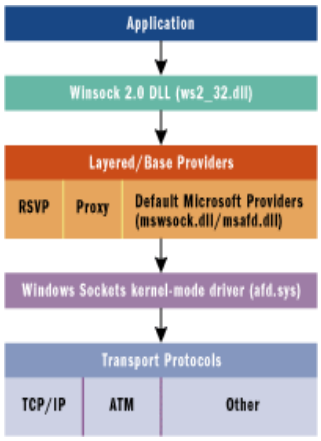

Figura 4.2 Arquitetura da biblioteca winsock 2

4.4 Interface do Cliente

A funcionalidade principal da interface cliente do sistema é transmitir os dados coletados pelos plug-ins. A implementação desse módulo consistiu no desenvolvimento da classe CMonitorSender, como podemos ver na figura 4.3.

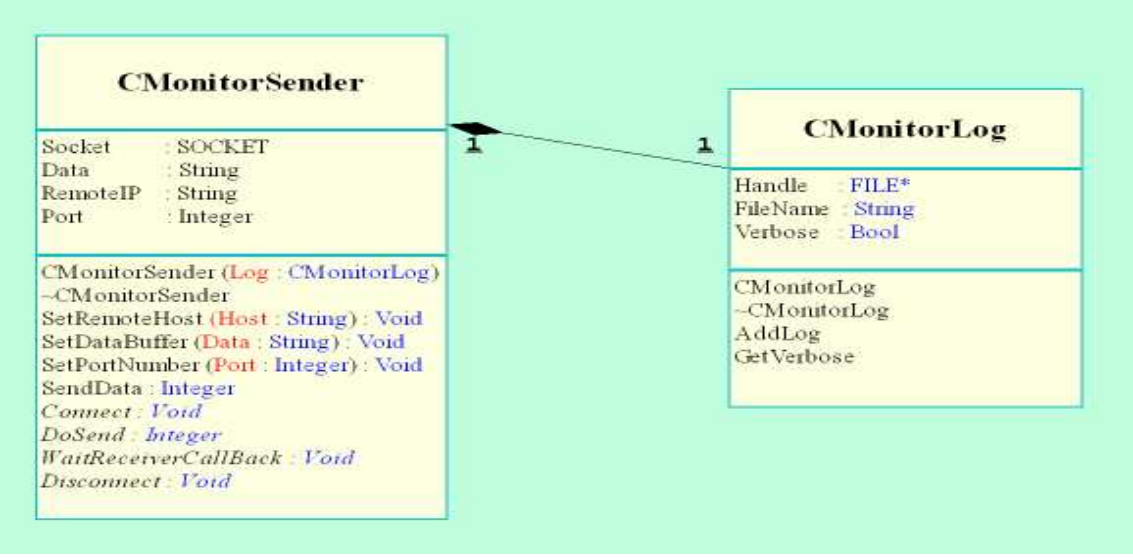

Figura 4.3 Classe CMonitorSender

Tabela 4.9: Atributos da classe CMonitorSender

Atributo	Descrição
Socket	Descritor do socket usado na conexão com o servidor remoto
Data	Armazena o conteúdo dos dados transmitidos
RemoteIP	Endereço IP do servidor de destino
Port	Número da porta de conexão

Tabela 4.10: Descrição dos métodos da classe CMonitorSender

Método	Descrição	Tipo
CMonitorSender	Construtor da classe	Público
~CMonitorSender	Destrutor da classe	Público
SetRemoteHost	Atualiza o endereço IP do servidor de destino	Público
SetDataBuffer	Atualiza o conteúdo dos dados que serão transmitidos	Público
SetPortNumber	Atualiza a porta de conexão usada	Público
SendData	Executa as rotinas de transmissão de dados	Público
Connect	Estabelece uma conexão com o servidor remoto	Privado
DoSend	Transmite os dados para o servidor remoto	Privado
WaitReceiverCallBack	Aguarda a resposta confirmando que os dados foram transmitidos	Privado
Disconnect	Fecha a conexão estabelecida com o servidor>	Privado

O funcionamento desse módulo pode ser descrito a partir de um diagrama de estados UML, ilustrado na figura 4.4. Nesse diagrama, o primeiro estado corresponde à tentativa de estabelecimento da conexão. Em caso de sucesso no estabelecimento da conexão, os dados são enviados (estado "Send Data"), e em seguida a conexão é terminada.

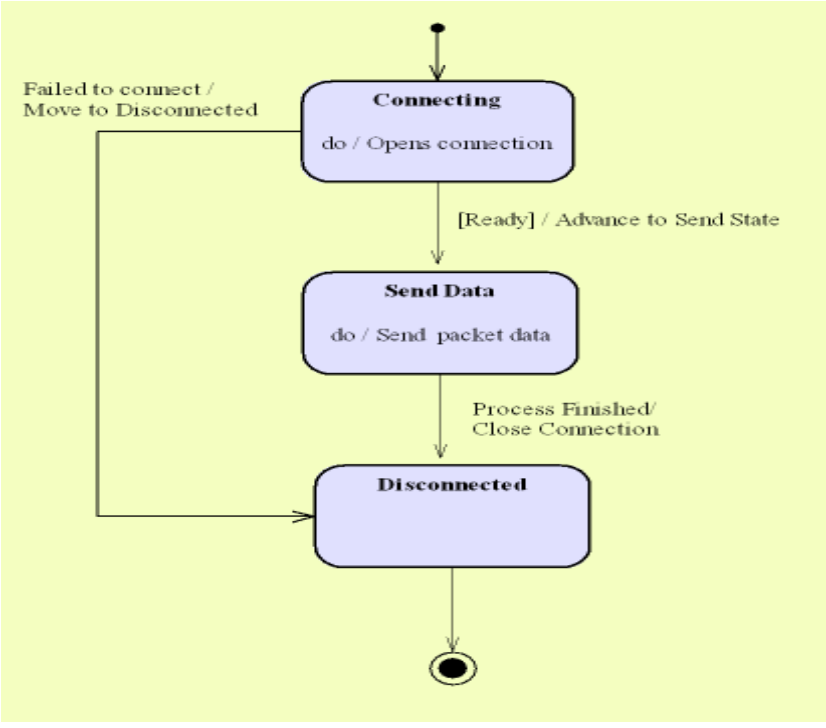

Figura 4.4 Diagrama de estados da interface cliente

4.5 Interface do Servidor

A funcionalidade principal da interface do servidor é receber o pacote de dados enviado pelo cliente e, em seguida, armazenar as informações no banco de dados. A implementação desse módulo consistiu no desenvolvimento da classe CMonitorReceiver, como podemos ver na figura 4.5.

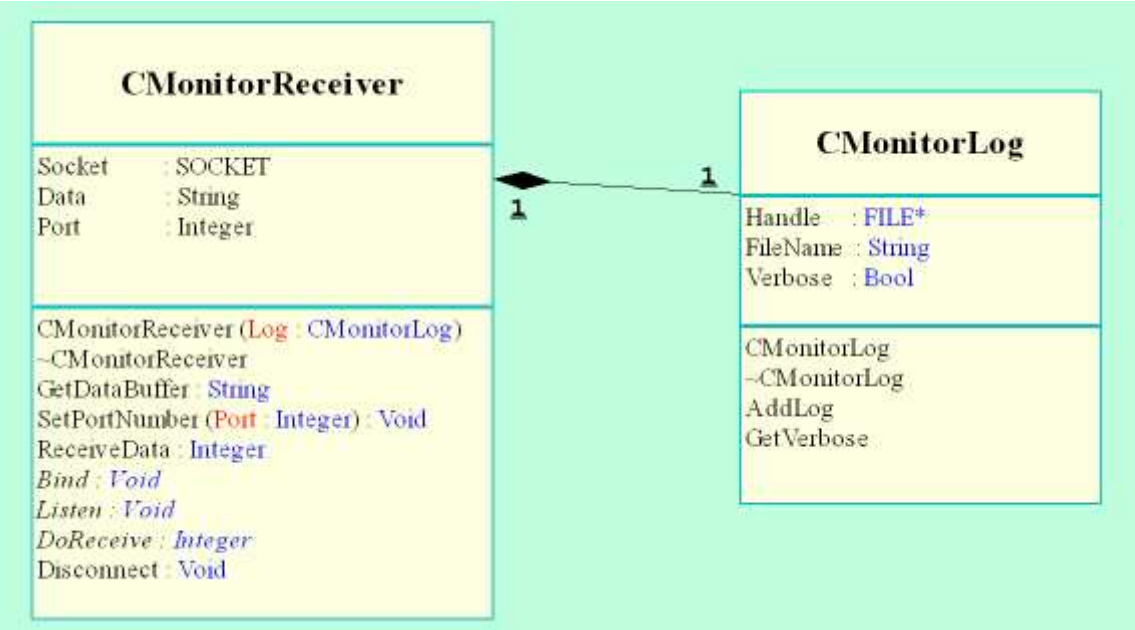

Figura 4.5 Classe CMonitorReceiver

Tabela 4.11 Atributos da classe CMonitorReceiver

Atributo	Descrição
Socket	Descritor do socket usado para receber os dados do cliente
Data	Armazena o conteúdo dos dados recebidos
Port	Número da porta de conexão

Tabela 4.12: Descrição dos métodos da classe CMonitorReceiver

Método	Descrição	Tipo
CMonitorReceiver	Construtor da classe	Público
~CMonitorReceiver	Destrutor da classe	Público
GetDataBuffer	Retorna o buffer com os dados recebidos	Público
SetPortNumber	Atualiza a porta de conexão usada	Público
ReceiveData	Executa a rotina principal da classe	Público
Bind	Prepara descritor de socket para receber os dados	Privado
Listen	Escuta a porta de conexão, aguardando a solicitação do cliente	Privado
DoReceive	Recebe os dados transmitidos	Privado
Disconnect	Destrói o descritor de socket	Privado

O funcionamento da interface do serivor também pode ser descrito a partir de um diagrama de estados UML, ilustrado na figura 4.6. Inicialmente, o descritor do socket é preparado para receber os dados. Essa rotina pode ser vista através do estado "Bind". Em seguida, o módulo aguarda os pedidos de conexão dos clientes (estado "Listen"). Caso alguma conexão seja solicitada, o link é estabelecido e os dados são recebidos no estado "Receive". Concluída essa rotina, o pacote de dados é processado, identificando as informações pertinentes (máquina monitorada, coleta realizada, dados recebidos, etc.) Essas informações são decompostas e armazenadas na base de dados. Em seguida, o módulo retorna para o estado "Liste", onde fica aguardando novas solicitações.

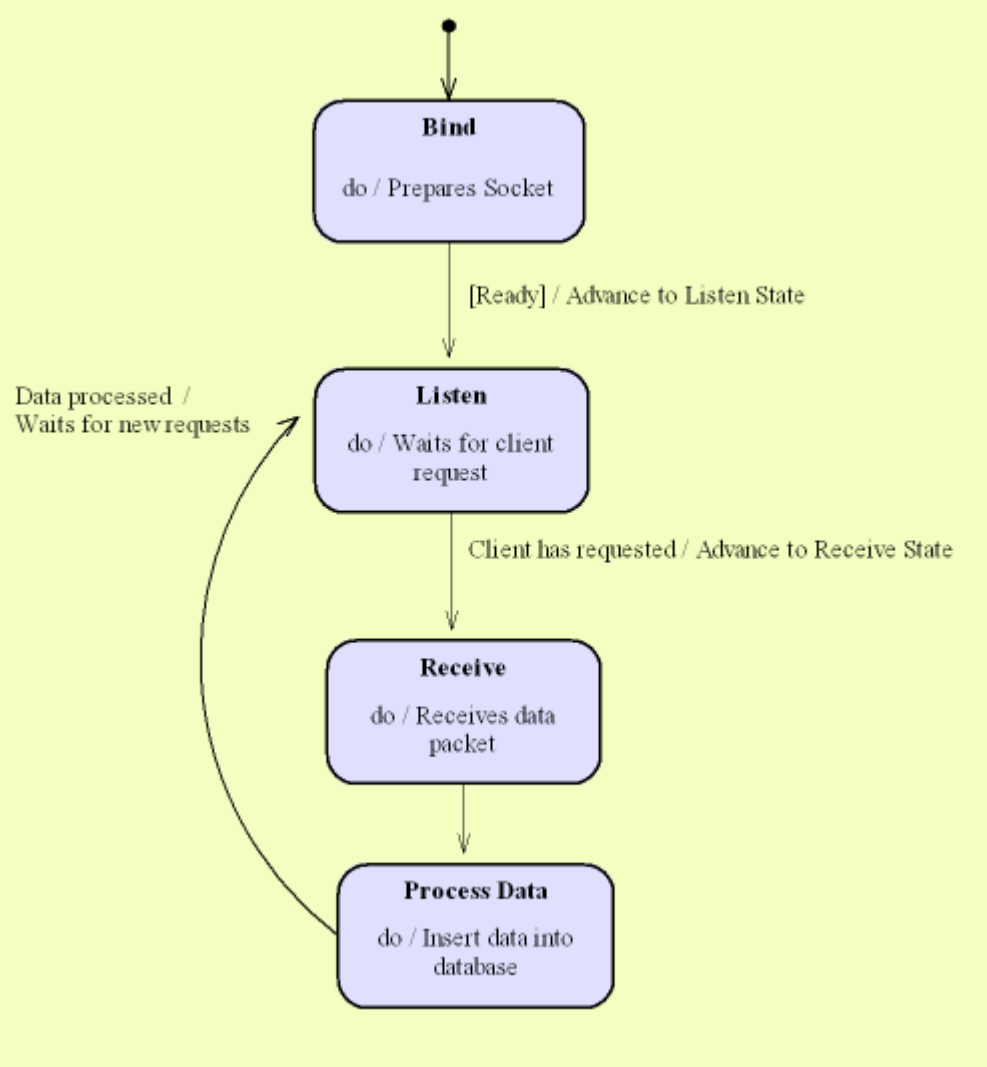

Figura 4.6 Diagrama de estados do módulo servidor

Capítulo 5

Integração com o Banco de Dados

O capítulo 4 apresentou a arquitetura cliente – servidor do sistema, descrevendo o fluxo dos pacotes de dados através da rede. Neste capítulo, a arquitetura de integração do sistema com o banco de dados será apresentada em detalhes.

A interface de comunicação com o banco de dados está implementada no módulo servidor do sistema. Após o recebimento dos dados, as informações são extraídas do pacote, iniciando o processamento dos dados. Essa rotina identifica o tipo de monitoramento realizado, assim como a máquina de onde os dados foram coletados. Em seguida, instruções SQL de inserção são preparadas e executadas na base de dados.

As próximas seções desse capítulo têm por objetivo descrever detalhadamente o processo de modelagem de dados, os critérios de escolha das ferramentas de banco de dados e a interface de comunicação entre o sistema e o servidor de banco de dados.

5.1 Modelo de Dados

A definição do modelo de dados consistiu em uma análise das entidades e relacionamentos existentes no domínio da informação do sistema. Foram definidas algumas sentenças que contemplavam os conceitos introduzidos no sistema. A partir dessas sentenças, uma análise gramatical permitiu a definição das entidades, seus atributos, as chaves candidatas e os relacionamentos.

Após essa análise, o modelo lógico do banco de dados foi construído a partir do mapeamento das definições apresentadas no modelo ER. As tabelas foram construídas a partir das entidades criadas, enquanto que e as chaves candidatas e os relacionamentos, através do processo de normalização, definiram as chaves primárias e os relacionamentos chave primária – chave estrangeira entre das tabelas. A figura 5.1 ilustra o modelo ER esquematizado, enquanto que a figura 5.2 ilustra o modelo de implementação gerado a partir do mapeamento das entidades, relacionamentos e chaves.

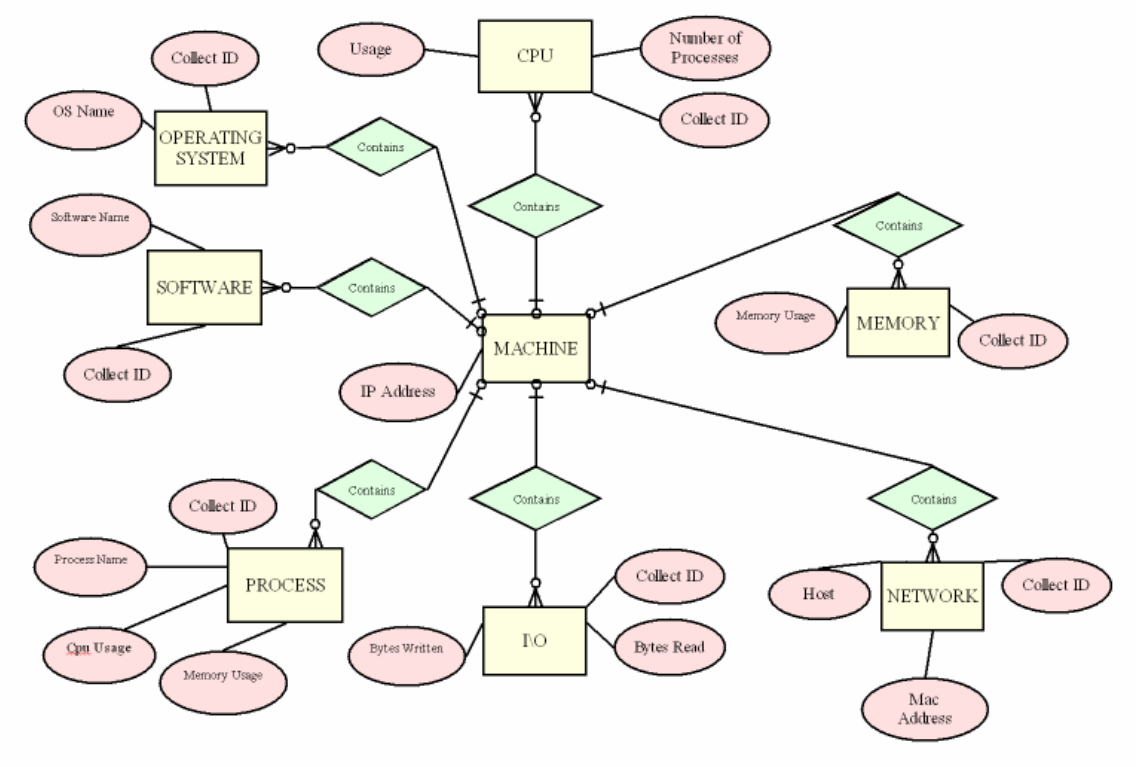

Figura 5.1 Modelo entidade relacionamento

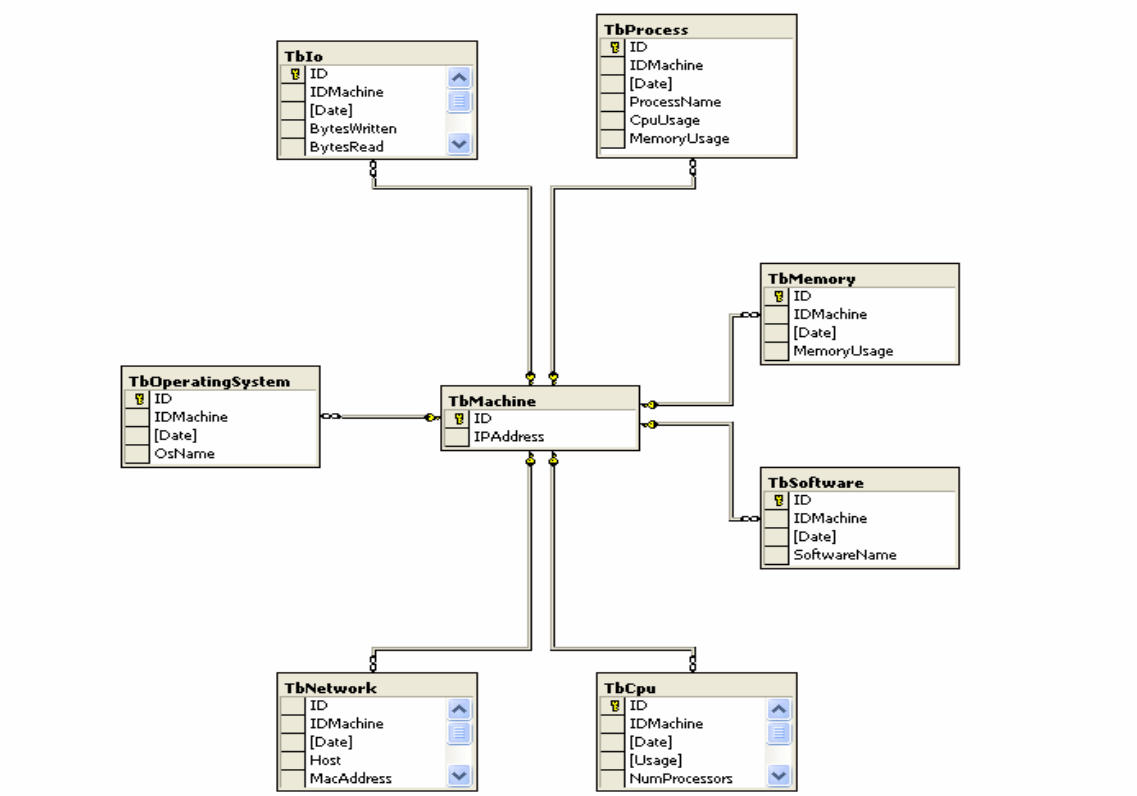

Figura 5.2 Modelo de implementação do banco de dados

5.2 Integração com SQL Server 2000

A escolha do serviço de banco de dados considerou algumas características de um SGDB, como controle de concorrência de transações, suporte a múltiplos usuários e integração com o ambiente Windows. Fatores como a quantidade de ferramentas disponíveis para modelagem e a experiência do programador também foram considerados.

A partir dessa análise, o serviço de banco de dados escolhido foi o SQL Server 2000, versão para desenvolvedor MSDE 2000. Trata-se de uma versão gratuita, com o objetivo de incentivar a criação de novas ferramentas que usem o serviço de banco de dados SQLSERVER 2000.

Existem inúmeros casos de sucesso no mercado de aplicações desenvolvidas no ambiente Windows que usam essa ferramenta para persistência de dados. Confiabilidade, segurança, integridade e usabilidade são as principais características do SQL Server. Por outro lado, alguns fatores podem inviabilizar o desenvolvimento de aplicações com esse banco de dados. O custo, que pode ser traduzido através do preço da licença do software, e a baixa portabilidade (compatível somente com as plataformas Windows NT) devem ser considerados no processo de planejamento do software. Nesses casos, outras soluções (como o PostgreSQL) devem ser analisadas.

5.3 Interface de comunicação com o banco de dados

A interface de comunicação entre o sistema e o servidor de banco de dados é realizada mediante o uso das rotinas da biblioteca SQLAPI. Esse componente contém classes que encapsulam dados e operações responsáveis pelo gerenciamento de conexões e execução de comandos SQL no banco de dados.

Assim como no caso do serviço banco de dados, a definição do componente de comunicação com o banco de dados também resultou em uma análise de possíveis interfaces. Uma possível solução seria a utilização do componente ADO, da Microsoft. Trata-se da solução mais usada nas aplicações atuais e tem como vantagem principal o fato que é compatível com a maioria dos serviços de banco de dados existentes. Como desvantagens podemos citar a questão do desempenho pois utiliza camadas intermediárias (ODBC ou OLEDB) para acessar o banco de dados. Além disso, trata-se de uma solução específica da plataforma Windows, o que inviabilizaria a tentativa de migração do módulo para outra plataforma.

A solução apresenta pela biblioteca SQLAPI é mais interessante do ponto de vista que ela suprime as deficiências apresentadas pelo componente ADO. Trata-se de uma biblioteca multiplataforma (com versões disponíveis para Windows e Linux) escrita em C++ (a mesma linguagem de programação utilizada no desenvolvimento do projeto). Além disso, apresenta uma performance bastante superior, pois suas funções internas comunicam-se diretamente com a API do banco de dados, evitando a passagem através de camadas intermediárias.

Essa solução também permite que algumas características de qualidade de software possam ser atingidas. A utilização desse componente garante um elevado índice de coesão, visto que a comunicação com o banco é feito de forma abstrata e genérica, a partir das operações e dados encapsulados nas classes da biblioteca. Como isso, garantimos a testabilidade do software. Além disso, a troca de dados entre a aplicação e o banco é limitada pelas interfaces da biblioteca, garantindo também um baixo acoplamento. Esse fator garante uma boa manutenibilidade do software, visto que o impacto de atualizações dos diversos módulos está limitado.

Capítulo 6

Telas e Diretórios do Sistema

Nos capítulos anteriores, a arquitetura do sistema e seus módulos foram descritos detalhadamente. O capítulo atual tem por finalidade apresentar as principais telas dos módulos do sistema (cliente e servidor), os componentes de controle e a arquitetura de diretórios.

As interfaces gráficas do sistema foram desenvolvidas em C++, através da biblioteca MFC da Microsoft. As estruturas de pastas foram criadas para melhor compreensão e agrupamento dos diversos arquivos e módulos que integram o sistema.

6.1 Árvore de diretórios do módulo cliente

A figura 6.1 ilustra a arquitetura de pastas armazenada no computador cliente. A tabela 6.1 descreve cada um dos diretórios criados, bem como os arquivos armazenados nas pastas.

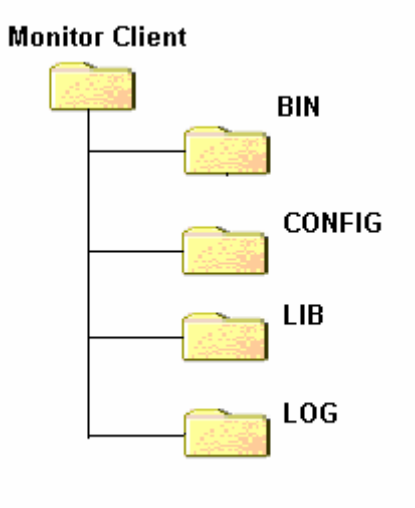

Figura 6.1 Estrutura de diretórios do módulo cliente

Tabela 6.1: Descrição dos diretórios do módulo cliente

Diretório	Descrição	Arquivos
BIN	Armazena os arquivos executáveis do sistema	MonitorAgent.exe MonitorClient.exe
CONFIG	Armazena o arquivo de configuração	MonitorClient.ini
LIB	Armazena as DLLs dos plug-ins	os.dll, process.dll network.dll, io.dll,

		cpu.dll, software.dll memory.dll
LOG	Armazena o arquivo de log do sistema	

6.2 Tela principal do módulo cliente

A figura 6.2 ilustra a tela principal do módulo. Os controles são identificados através de números. Esses elementos são descritos na tabela 6.2.

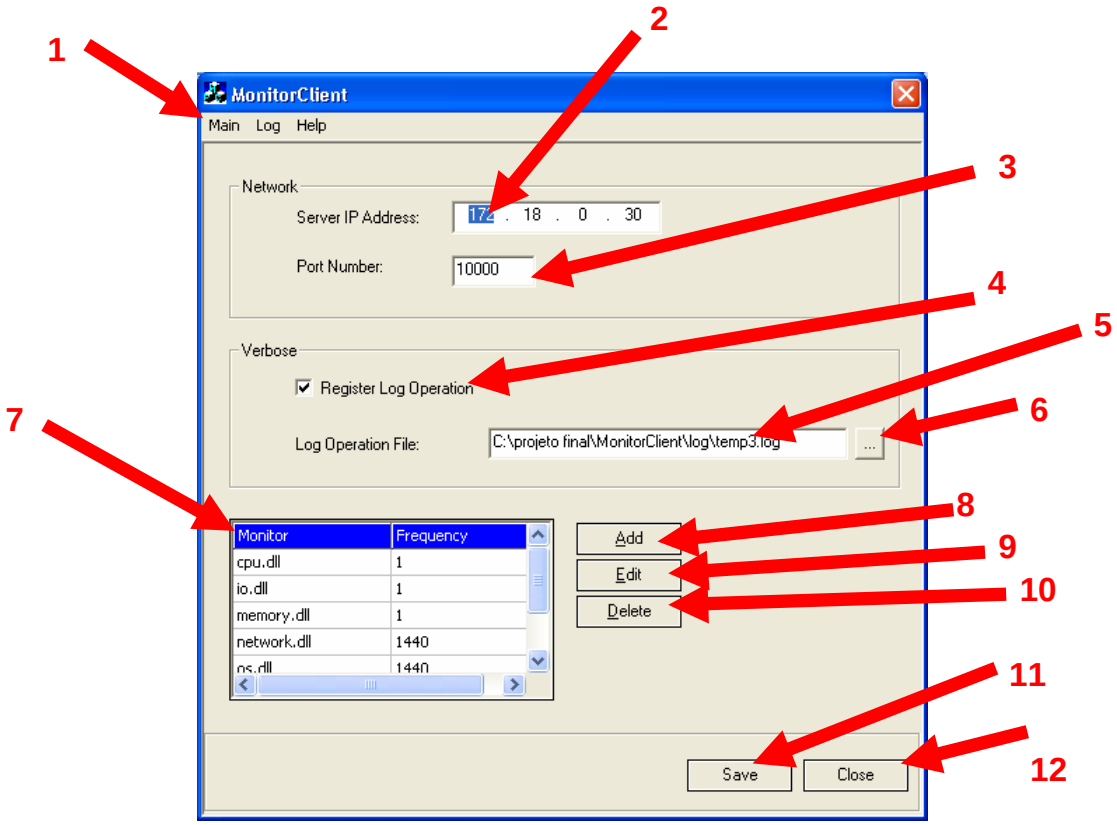

Figura 6.2 Tela principal do módulo cliente

Tabela 6.2 Controles da tela principal do módulo cliente

Identificador	Descrição
1	Menu principal
2	Endereço IP do servidor de destino
3	Porta de comunicação
4	Checkbox do registro de operações
5	Caixa de edição do arquivo de log
6	Botão de seleção do arquivo de log
7	Grid de análises
8	Botão de adição
9	Botão de edição
10	Botão de remoção

11	Botão de salvar
12	Botão de fechar

6.2.1 Menu Principal

O menu do módulo cliente é composto por três sub-menus: *Main*, *Log*, *Help*. As seções 6.2.1.1, 6.2.1.2 e 6.2.1.3 descrevem os componentes de cada um dos sub-menus.

6.2.1.1 Sub Menu Main

Esse menu é composto por 3 itens: *Start Monitor*, *Stop Monitor* e *Quit*. Esse controle hospeda as funções principais do módulo cliente. Ele é responsável por ativar e desativar o agente de monitoramento.

Conforme foi visto no capítulo 2, o programa que executa o papel de cliente é o utilitário MonitorAgent.exe, que pode ser executado a partir de uma linha de comando. A função do item *Start Monitor* é executar esse utilitário, a partir de uma linha de comando, onde os parâmetros correspondem às configurações estabelecidas na tela principal. Sempre que o agente estiver ativo (o programa MonitorAgent em execução) na máquina, essa opção fica desabilitada.

De maneira análoga, o item *Stop Monitor* é responsável por abortar o serviço de monitoramento, isto é, interromper a execução do programa MonitorAgent.exe. Esse item fica habilitado somente se o utilitário estiver em modo de execução no computador.

O item Quit fecha a janela principal do módulo. Essa rotina **não aborta** o agente de monitoramento. Sempre que o usuário executar o programa de configuração (MonitorClient.exe), o aplicativo consulta a lista de processos em execução no computador, verificando se o programa MonitorAgent.exe está em execução. Se isso for verdade, significa que o agente está ativo e, portanto, a opção *Start Monitor* deve ficar desabilitada, enquanto que o item *Stop Monitor* deve ser habilitado. Por outro lado, se o processo não estiver ativo, a opção *Start Monitor* deve ser habilitada e o item *Stop Monitor* deverá estar desabilitado.

6.2.1.2 Sub Menu Log

Esse menu é composto de um único item (See Log File) e tem por finalidade exibir o conteúdo do arquivo de registro de operações do sistema (log). A figura 6.3 ilustra o conteúdo de um arquivo log gerado pelo sistema.

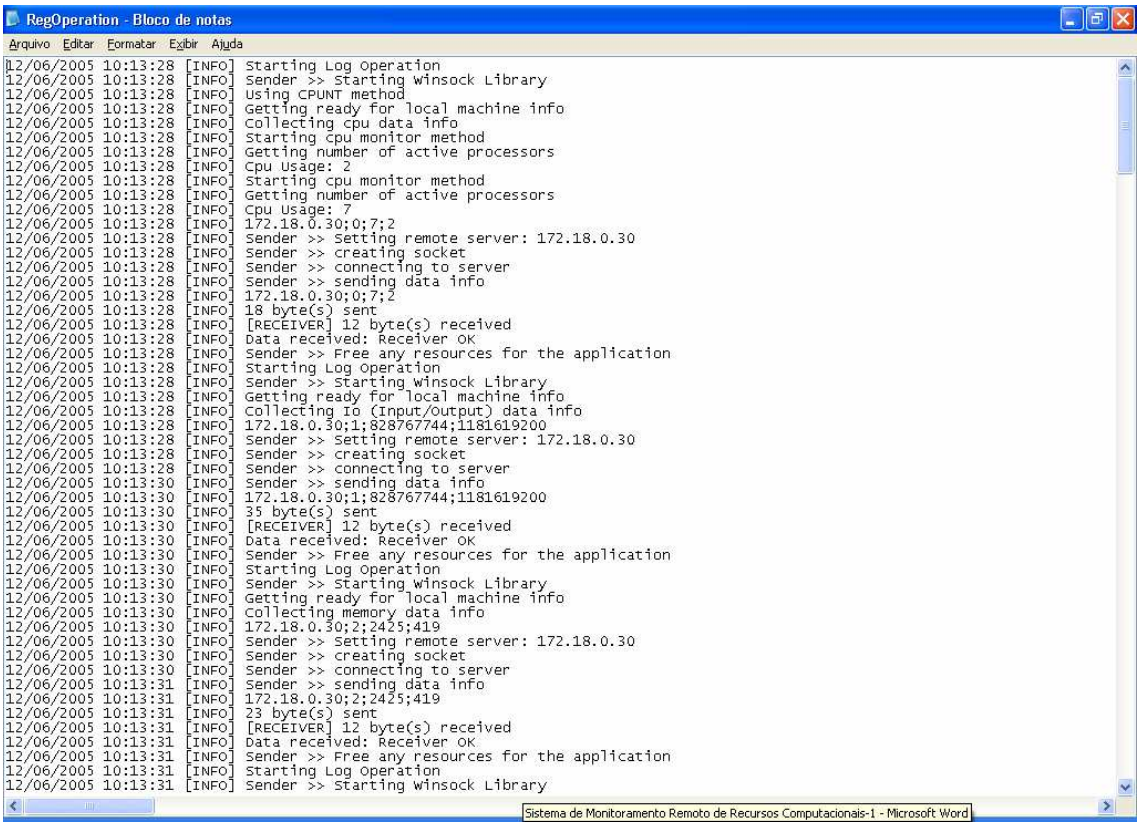

Figura 6.3 Conteúdo do arquivo de log do sistema

Como podemos ver na figura acima, nesse arquivo são registrados os processos executados pelo módulo cliente. Informações como análises de monitoramento, dados coletados, horário da coleta e quantidade de dados enviados são registradas nesse arquivo. De posse dessas informações, é possível verificar se o módulo está funcionando corretamente. Em caso de falhas, os erros são descritos, tornando-se fácil a identificação do problema (erro na conexão com o servidor, problemas na coleta de dados, etc.).

6.2.1.3 Sub Menu Help

O objetivo desse menu é disponibilizar informações a respeito do software. Nesse menu, podemos ter itens como manual do usuário, menu de ajuda e outras informações adicionais do software. Na implementação atual, foi criado um único item informativo, mencionando a versão do produto. Nas versões posteriores do software, novos itens podem ser adicionados.

6.2.2 Endereço IP do Servidor de Destino

Nesse campo, deve ser informado o endereço IP para onde devem ser encaminhados os pacotes de dados. O valor do endereço é composto de 4 bytes cujos valores podem variar de 0 até 255.

6.2.3 Porta de Comunicação

O valor desse campo corresponde à porta de comunicação usada para transmissão dos dados. O módulo cliente deve usar a mesma porta de comunicação usada pelo servidor.

6.2.4 Checkbox do Registro de Operações

Esse controle deve ser ativado caso haja interesse do usuário em registrar as operações processadas (geração do arquivo de log).

6.2.5 Caixa de Edição do Arquivo de Log

Esse controle exibe o nome do arquivo que contém o log das operações do sistema. Essa caixa de edição é habilitada sempre que o checkbox de log é ativado.

6.2.6 Botão de Seleção do Arquivo de Log

Esse botão exibe uma tela de seleção do arquivo que será usado para armazenar o log do sistema. Essa tela está ilustrada na figura 6.4.

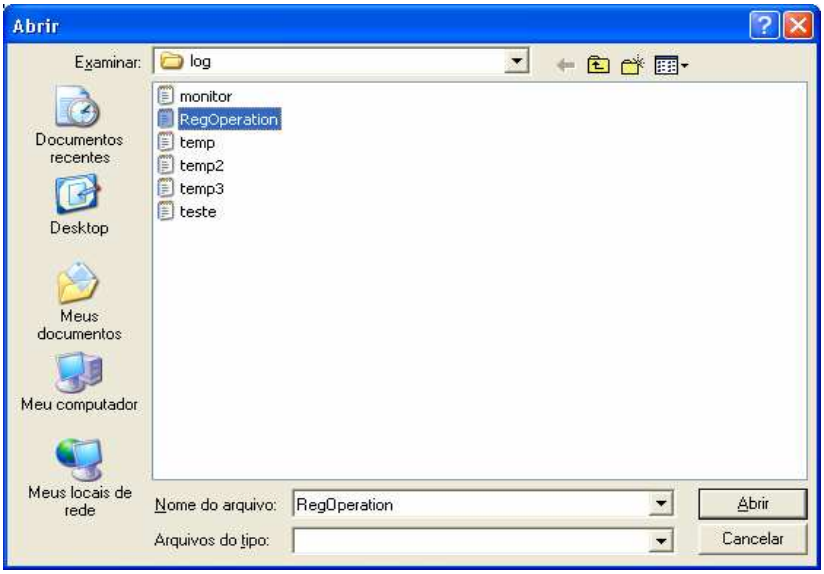

Figura 6.4 Tela de seleção do arquivo de log

6.2.7 Grid de Análises

Nesse controle, são apresentadas as análises (monitoramentos) configuradas. Cada análise possui um plug-in de monitoramento com sua respectiva periodicidade de coleta em minutos. Um exemplo da configuração desse controle pode ser ilustrado através da figura 6.5.

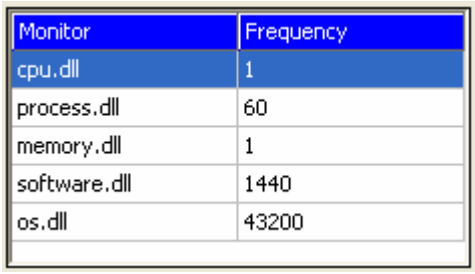

Figura 6.5 Grade de Análises

Como podemos notar na figura acima, temos 5 análises configuradas. A análise dos recursos de CPU é feita através do plug-in cpu.dll. A coleta de dados dessa análise ocorre com freqüência de 1 minuto. O mesmo vale para a análise de memória. No caso dos processos, os dados são coletados a cada intervalo de 1 hora (60 minutos). A lista de softwares e a versão do sistema operacional instalado são coletadas com periodicidades maiores (diária e mensal respectivamente).

6.2.8 Botão de Adição

Controle responsável por chamar a rotina de adição de novas análises na grid de análises. O evento de clique nesse botão exibe uma tela de adição de novas análises. A figura 6.6 apresenta essa tela, onde a combo principal carrega todos os plug-ins armazenados na pasta lib do sistema. Note que essa lista é carregada dinamicamente, isto é, caso um novo plug-in seja criado, ele será incluído na lista de plug-ins.

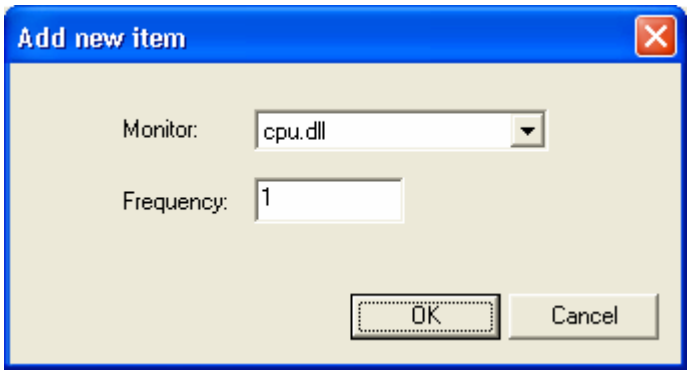

Figura 6.6 Tela de adição de novas análises

6.2.9 Botão de Edição

Esse botão é responsável por chamar a rotina de edição da periodicidade de monitoramento (campo *frequency*). Assim como o botão de adição, esse controle também ativa um evento que exibe uma tela para informar o valor da freqüência de coleta dos dados.

6.2.10 Botão de Remoção

Remove uma análise configurada.

6.2.11 Botão de Salvar

Captura todas as configurações da tela e armazena essas informações no arquivo monitorclient.ini. Esse arquivo mantém a persistência das configurações que são restauradas quando o usuário abre a janela.

6.2.12 Botão de Fechar

A função desse controle é a mesma executada pelo item *Quit* do sub menu *Main*. O evento termina o programa de configuração. Essa rotina não interrompe o funcionamento do programa agente, isto é, se o utilitário for ativado pelo programa de configuração, a chamada ao evento de fechar a janela não implica no fim do processo criado pelo agente.

6.3 Árvore de diretórios do módulo servidor

A figura 6.7 ilustra a arquitetura de pastas armazenada no computador servidor. A tabela 6.3 descreve cada um dos diretórios criados, bem como os arquivos armazenados nas pastas.

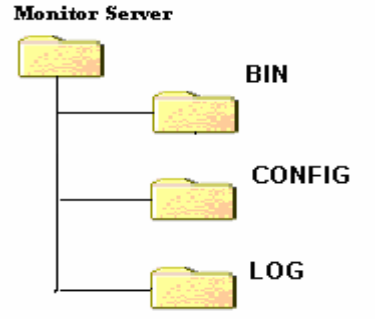

Figura 6.7 Estrutura de pastas do módulo servidor

Tabela 6.3: Descrição dos diretórios do módulo servidor

Diretório	Descrição	Arquivos
BIN	Armazena os arquivos executáveis do sistema	MonitorReceiver.exe MonitorServer.exe
CONFIG	Armazena o arquivo de configuração	MonitorServer.ini
LOG	Armazena o arquivo de log do sistema	

6.4 Tela principal do módulo servidor

A figura 6.8 ilustra a tela principal do módulo. Os controles são identificados através de números. Esses elementos são descritos na tabela 6.4.

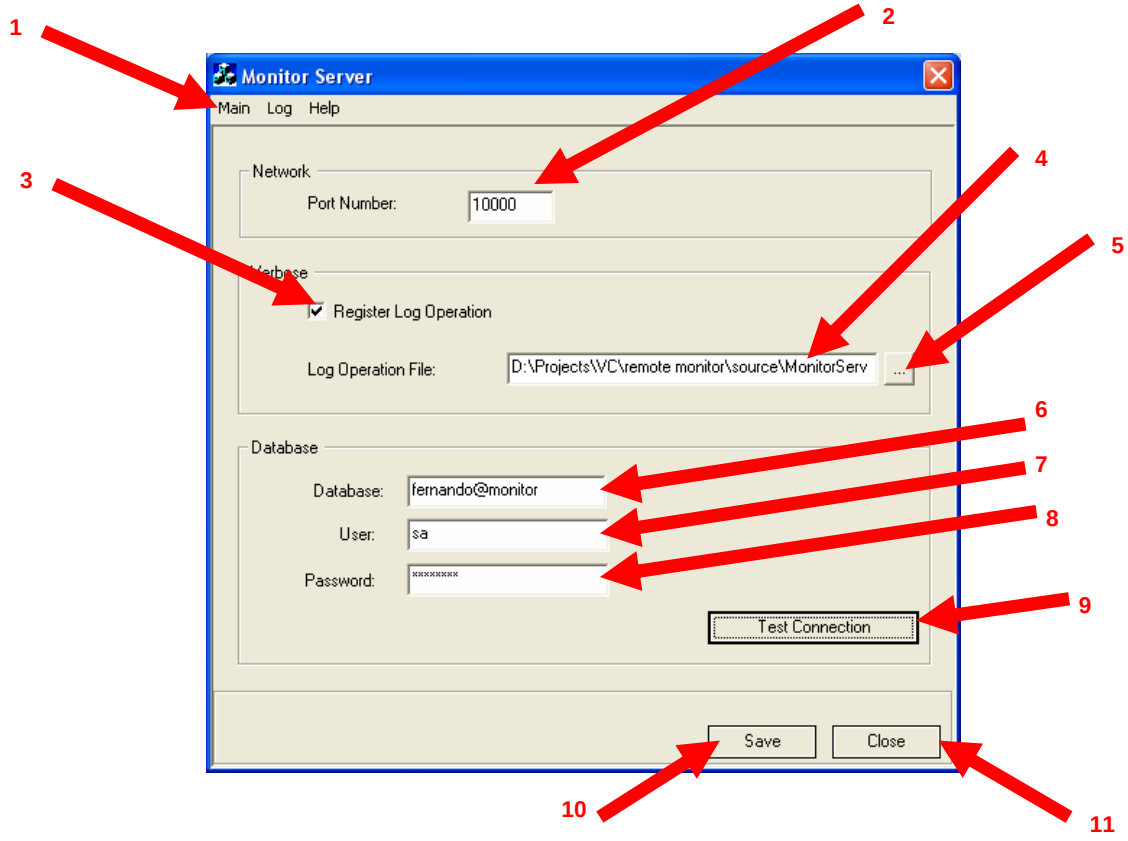

Figura 6.8 Tela principal do módulo servidor

Tabela 6.4 Controles da tela principal do módulo servidor

Identificador	Descrição
1	Menu principal
2	Porta de comunicação
3	Checkbox do registro de operações
4	Caixa de edição do arquivo de log
5	Botão de seleção do arquivo de log
6	Caixa de edição do servidor do banco de dados

7	Caixa de edição do usuário do banco de dados
8	Caixa de edição da senha do banco de dados
9	Botão de teste de conexão
11	Botão de salvar
12	Botão de fechar

6.4.1 Menu Principal

Assim como o módulo cliente, o menu do aplicativo servidor é composto por três sub-menus: *Main*, *Log*, *Help*. As seções 6.4.1.1, 6.4.1.2 e 6.4.1.3 descrevem os componentes de cada um dos sub-menus.

6.4.1.1 Sub Menu Main

Esse menu é composto por 3 itens: *Start Receiver*, *Stop Receiver* e *Quit*. Esse controle hospeda as funções principais do módulo. Ele é responsável por iniciar e interromper o processo que aguarda o envio das informações através da rede.

Conforme foi visto no capítulo 2, o programa que executa o papel do processo servidor é o utilitário MonitorReceiver.exe, que pode ser executado a partir de uma linha de comando. A função do item *Start Receiver* é executar esse utilitário, a partir de uma linha de comando, onde os parâmetros correspondem às configurações estabelecidas na tela principal. Sempre que o receiver estiver ativo (o programa MonitorReceiver em execução) na máquina, essa opção fica desabilitada.

De maneira análoga, o item *Stop Receiver* é responsável por abortar o processo executado, isto é, interromper a execução do programa MonitorReceiver.exe. Esse item fica habilitado somente se o utilitário estiver em modo de execução no computador.

O item Quit fecha a janela principal do módulo. Essa rotina **não aborta** o receiver. Sempre que o usuário executar o programa de configuração (MonitorServer.exe), o aplicativo consulta a lista de processos em execução no computador, verificando se o programa MonitorReceiver.exe está em execução. Se isso for verdade, significa que o back-end está ativo e, portanto, a opção *Start Receiver* deve ficar desabilitada, enquanto que o item *Stop Receiver* deve ser habilitado. Por outro lado, se o processo não estiver ativo, a opção *Start Receiver* deve ser habilitada e o item *Stop Receiver* deverá estar desabilitado.

6.4.1.2 Sub Menu Log

Esse menu é composto de um único item (See Log File) e tem por finalidade exibir o conteúdo do arquivo de registro de operações do sistema (log). A figura 6.9 ilustra o conteúdo de um arquivo log gerado pelo sistema.

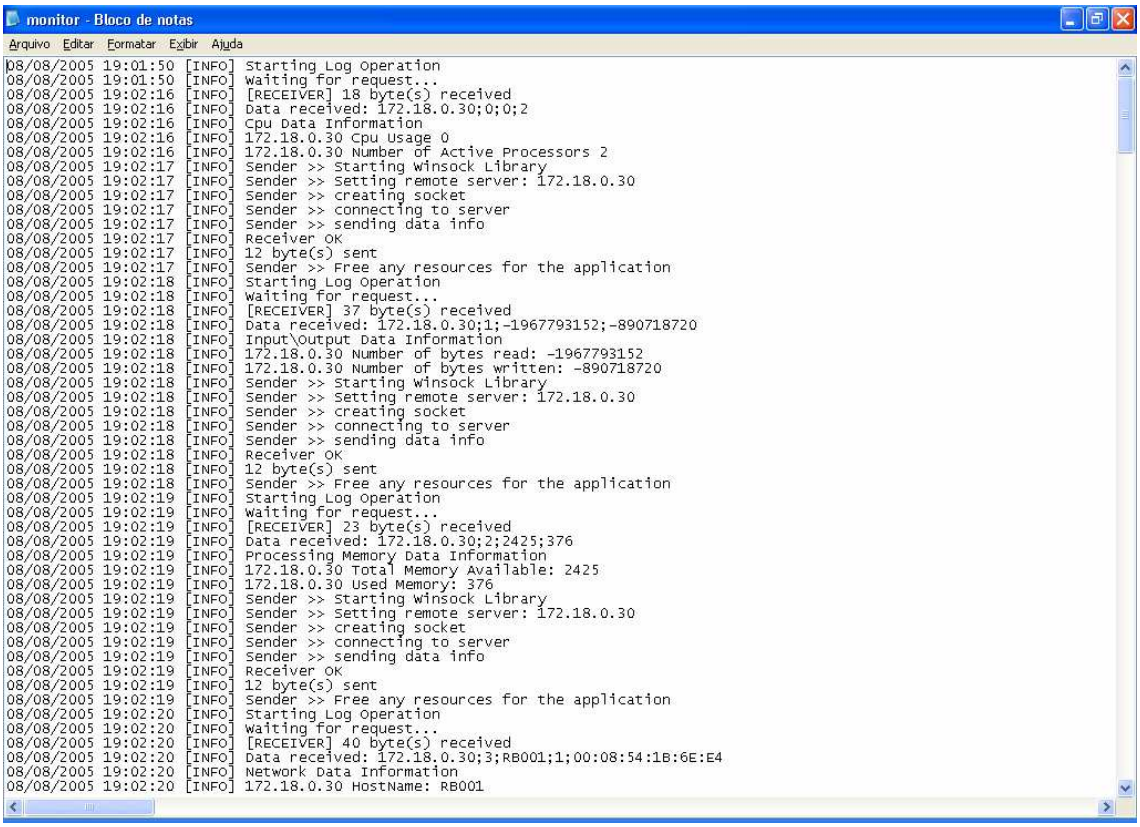

Figura 6.9 Tela do registro de operações do módulo servidor

6.4.1.3 Sub Menu Help

O objetivo desse menu é disponibilizar informações a respeito do software. Nesse menu, podemos ter itens como manual do usuário, menu de ajuda e outras informações adicionais do software. Na implementação atual, foi criado um único item informativo, mencionando a versão do produto. Nas versões posteriores do software, novos itens podem ser adicionados.

6.4.2 Porta de Comunicação

O valor desse campo corresponde à porta de comunicação usada para transmissão dos dados. Lembre-se que a porta de comunicação informada nesse campo deve ser a mesma configurada no módulo cliente.

6.4.3 Checkbox do Registro de Operações

Esse controle deve ser ativado caso haja interesse do usuário em registrar as operações processadas (geração do arquivo de log).

6.4.4 Caixa de Edição do Arquivo de Log

Esse controle exibe o nome do arquivo que contém o log das operações do sistema. Essa caixa de edição é habilitada sempre que o checkbox de log é ativado.

6.4.5 Botão de Seleção do Arquivo de Log

Esse botão exibe uma tela de seleção do arquivo que será usado para armazenar o log do sistema. Essa tela está ilustrada na figura 6.4.

6.4.6 Caixa de edição do servidor do banco de dados

Esse controle contém o nome do servidor de banco de dados usado para armazenar os dados recebidos. O dicionário de dados define o conteúdo desse campo da seguinte maneira:

Campo = Nome do serviço de banco de dados + '@' + Nome da base de dados

Exemplo: **localhost@monitorDB**

6.4.7 Caixa de edição do usuário do banco de dados

Informa o login do usuário do banco de dados.

6.4.8 Caixa de edição da senha do banco de dados

Informa a senha do usuário de banco de dados.

6.4.9 Botão de teste de conexão

Realiza um teste de validação dos parâmetros informados nos campos 6.4.7, 6.4.8 e 6.4.9. O teste consiste em abrir uma conexão com o banco de dados. Em caso de sucesso, uma mensagem confirmando que os parâmetros foram informados corretamente será exibida. Caso contrário, a interface retorna uma mensagem descrevendo o erro (problemas de autenticação, base de dados inválida, etc.).

6.4.10 Botão de Salvar

Captura todas as configurações da tela e armazena essas informações no arquivo monitorserver.ini. Esse arquivo mantém a persistência das configurações que são restauradas quando o usuário abre a janela.

6.4.11 Botão de Fechar

A função desse controle é a mesma executada pelo item *Quit* do sub menu *Main*. O evento termina o programa de configuração. Essa rotina não interrompe o funcionamento do programa receiver, isto é, se o utilitário for ativado pelo programa de configuração, a chamada ao evento de fechar a janela não implica no fim do processo criado pelo back-end.

Capítulo 7

Testes e Resultados

Nesse capítulo serão apresentados os testes do sistema realizados, descritos através das metodologias empregadas. Nessa etapa foram aplicados diversos processos, com o objetivo de verificar se os requisitos funcionais do sistema estavam sendo implementados corretamente. Como veremos nas próximas seções, essas rotinas consistem em testes de unidade, integração dos módulos, validação e do sistema.

Os testes foram realizados em uma rede de computadores corporativa, onde os programas clientes foram instalados em diversas máquinas. Os agentes foram ativados, e as análises foram configuradas com uma determinada periodicidade. Em uma dessas máquinas, o módulo servidor também foi instalado. Esse computador recebia periodicamente as informações, armazenando-as em uma base de dados. Os resultados obtidos através do funcionamento do sistema servem como dados de validação dos requisitos funcionais do sistema, isto é, verificam se a arquitetura desenvolvida é capaz de realizar o monitoramento dos recursos computacionais.

7.1 Testes de Unidade

Os testes de unidade concentram-se seus esforços na verificação do funcionamento das menores unidades do projeto do sistema. Os casos de testes de unidade elaborados ficaram focados nos algoritmos implementados nos plug-ins. Nesse processo, foram usadas duas metodologias de testes: testes estruturais e testes funcionais.

Os testes estruturais, também chamados de testes caixa-branca, são processos que requerem a verificação das rotinas através do acompanhamento das instruções de programação no código-fonte. Essa rotina tem por finalidade principal testar a lógica implementada em baixo nível, verificando o funcionamento do programa. Esse teste é realizado percorrendo todos os caminhos possíveis (ao menos uma vez), testando todas as condições lógicas e acessando as estruturas de dados. Essa tarefa é de extrema importância, pois através da varredura dos possíveis caminhos percorridos durante o processamento do sistema, verifica-se a existência de erros catastróficos, isto é, defeitos que afetam diretamente a confiabilidade do sistema.

Os testes funcionais também são conhecidos como testes caixa-preta. O objetivo desse processo consiste em verificar o grau de conformidade do programa,

isto é, se os resultados apresentados estão de acordo com os requisitos funcionais especificados. Para ilustrar esse teste, podemos usar como caso de teste o plug-in de monitoramento da CPU. Sabemos que o resultado esperado por esse plug-in é um valor percentual entre 0 e 100 %. Outra característica que podemos avaliar é que quanto maior o processamento exigido no processador, maior deverá ser o consumo de CPU. A tabela 7.1 mostra o momento de uma coleta e o valor capturado no instante.

Tabela 7.1 Coleta dos dados do plug-in cpu.dll

DateTime	Usage (%)
13/06/2005 15:35:54	26
13/06/2005 15:36:58	47
13/06/2005 15:38:05	68
13/06/2005 15:39:09	17
13/06/2005 15:40:15	31
13/06/2005 15:41:19	39
13/06/2005 15:42:25	9
13/06/2005 15:43:28	63
13/06/2005 15:44:34	9
13/06/2005 15:45:38	9
13/06/2005 15:46:45	13
13/06/2005 15:47:49	29
13/06/2005 15:48:55	32
13/06/2005 15:49:59	13
13/06/2005 15:51:05	9
13/06/2005 15:52:09	30
13/06/2005 15:53:14	9
13/06/2005 15:54:18	31
13/06/2005 15:55:24	13
13/06/2005 15:56:27	13
13/06/2005 15:57:33	30
13/06/2005 15:58:36	18
13/06/2005 15:59:43	44
13/06/2005 16:00:46	58

A partir dos dados apresentados na tabela acima, nota-se que o plug-in ficou coletando dados com uma periodicidade de aproximadamente 1 minuto. Podemos observar que o maior consumo registrado foi de 63%, enquanto que o menor consumo registrado foi de 9%. Durante os testes, foi observado que os instantes que apresentaram maior consumo de CPU coincidiram com os momentos de maior demanda de processamento.

7.2 Testes de Integração

Os testes de integração consistem em verificar o funcionamento das diferentes unidades em conjunto. Nessa etapa os acoplamentos entre as interfaces dos módulos do sistema são testados. No contexto do projeto, as seguintes interfaces devem ser testadas: comunicação entre agente e plug-in, arquitetura de cliente-servidor e a interface de integração com o banco de dados.

Conforme foi apresentado no capítulo 2, a interface de comunicação entre o agente e o plug-in é definida através da rotina GetData em cada DLL. Os testes estruturais consistem em acompanhar os caminhos possíveis apresentados no fluxograma da figura 2.2. O teste funcional consiste em avaliar se a comunicação entre o programa MonitorAgent.exe e as DLLs dos plug-ins foi estabelecida com sucesso.

Para os casos de testes da integração da arquitetura cliente-servidor, os testes caixa-branca devem verificar os dados que estão sendo enviados e recebidos nos módulos cliente e servidor, respectivamente. Nos testes caixa-preta, um programa Sniffer foi instalado em uma máquina cliente e na máquina servidora. Através desse utilitário, é possível monitorar os fluxos de pacotes de dados na rede, identificando o fluxo de informação enviado pela interface cliente e recebido no computador servidor. A figura 7.1 mostra a tela do sniffer, onde podemos ver que a máquina cliente (IP 172.18.0.5) envia o pacote de dados, contendo a lista de processos coletada, para o servidor (IP 172.18.0.30).

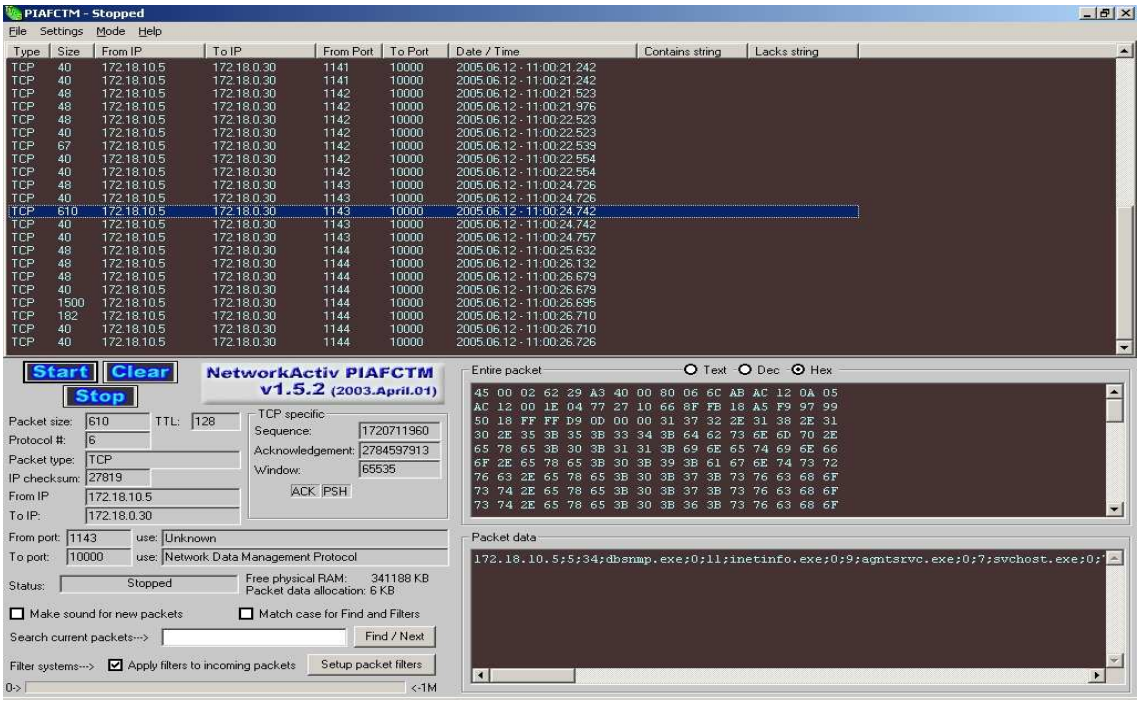

Figura 7.1 Tela do Sniffer

Os casos de testes da integração do módulo servidor com o banco de dados consistem em executar consultas na base de dados, verificando o resultado. No momento que uma coleta é recebida, a mesma deve ser armazenada na base de dados. A figura 7.2 mostra a tela do aplicativo SQL Query Analyzer do Microsoft SQL Server 2000. Usamos esse programa para executar consultas na base de dados, verificando se os dados estavam sendo armazenados na base. Cada máquina monitorada deve ser representada através de um registro na tabela TbMachine, apresentada no modelo de implementação das tabelas na figura 5.2.

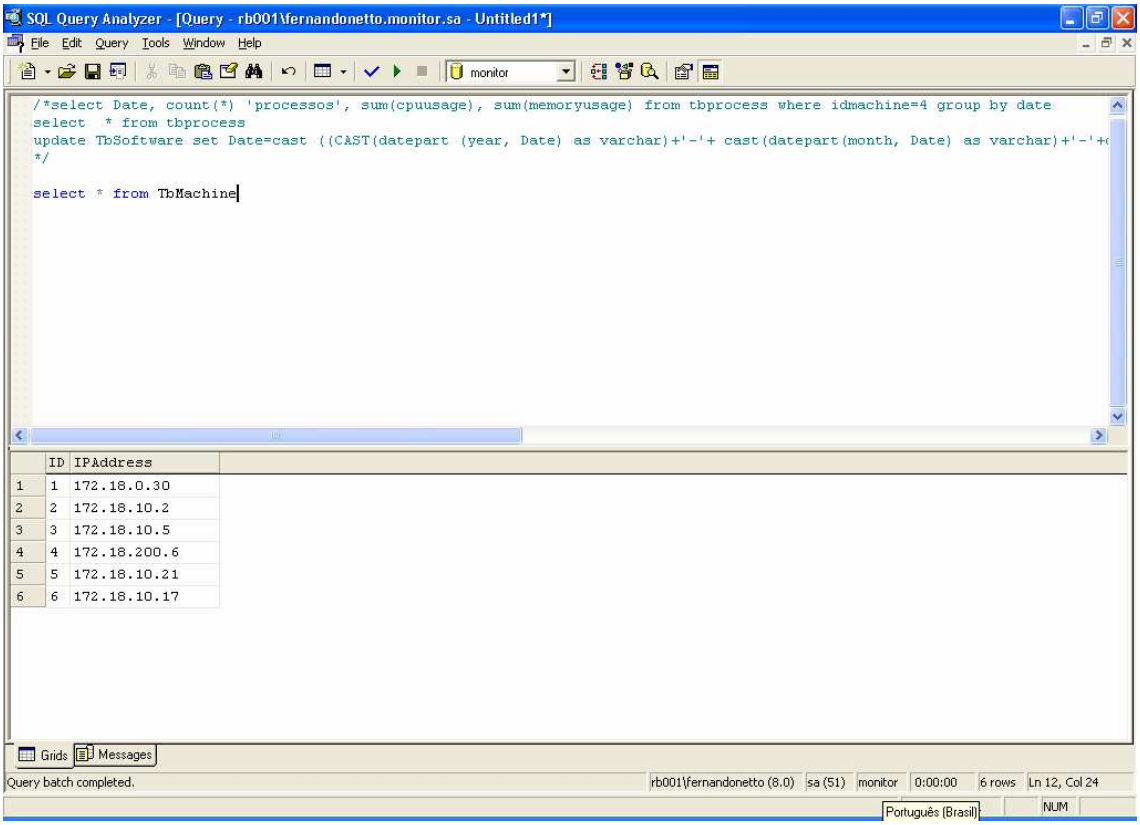

Figura 7.2 Tela do SQL Server Query Analyzer

7.3 Testes de Validação

Concluída a fase de testes de integração, podemos iniciar a etapa de testes de validação do sistema. Essa tarefa consiste em verificar o grau de conformidade do funcionamento do software com os requisitos funcionais do sistema. Esse processo consiste em testar o software através da ótica do usuário, isto é, verificar se as expectativas do funcionamento do produto estão atendendo as necessidades do próprio.

O sistema foi testado a partir da verificação das funcionalidades das janelas de configuração do software nos módulos cliente e servidor. Diversos cenários de uso foram criados para abordar as possíveis interações entro o usuário e o sistema. As funcionalidades foram decompostas, e para cada função foi criado um diagrama de caso de uso UML. A figura 7.3 apresenta um diagrama de caso de uso abordado durante os testes.

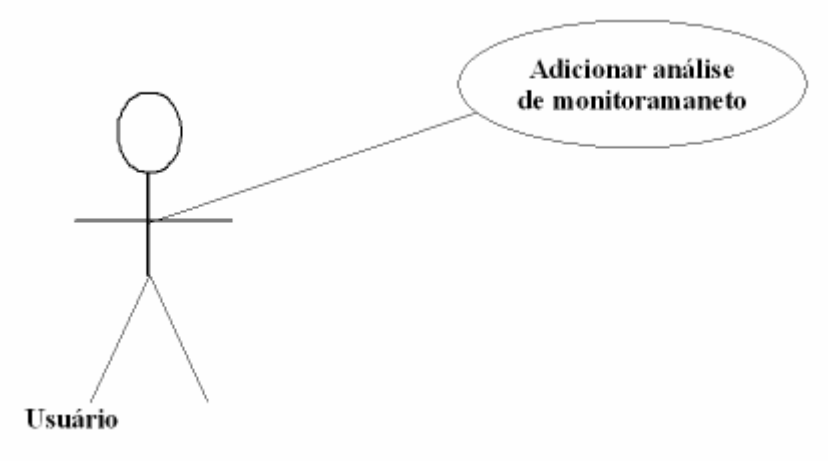

Figura 7.3 Diagrama de caso de uso do sistema

Caso de Uso: Adicionar análise de monitoramento

Fluxo Básico:

1. O usuário solicita a inclusão de uma nova análise clicando no botão "Add" na janela de configuração do módulo cliente.

2. O sistema apresenta uma nova janela, contendo os campos dois campos: uma lista dos plug-ins disponíveis e o valor da periodicidade do processo de monitoramento.

3. O usuário deve selecionar o plug-in de monitoramento desejado e informar a freqüência de coleta dos dados.

4. O usuário deve clicar no botão "OK" para confirmar a inclusão dos dados.

5. O sistema adiciona o elemento e cria um novo registro na grid de análises, exibindo as configurações incluídas.

7.4 Testes de Sistema

As tarefas dessa etapa consistem em testar o sistema por completo, isto é, verificar o funcionamento de todos os módulos do sistema integrados, e em seguida, analisar se os resultados obtidos com o processo de monitoramento remoto satisfazem as expectativas do usuário.

Os seguintes processos de testes foram executados:

1. Ativar o receiver no módulo servidor.
2. Escolher uma máquina cliente.
3. Cadastrar as análises de acordo com as informações apresentadas na tabela 7.2.

Tabela 7.2 Análises de monitoramento para o teste do sistema

Análise	Periodicidade de coleta
CPU	1 minuto
Memória	1 minuto
Software	1 dia (1440 minutos)
Interfaces de Rede	1 dia (1440 minutos)
Sistema Operacional	1 dia (1440 minutos)
Processos	2 minutos

4. Ativar o monitoramento dos dados no módulo cliente.
5. Acompanhar o registro das operações nos módulos cliente e servidor.

Os testes do sistema duraram aproximadamente 2.5 horas. O ambiente de testes escolhido foi a rede de computadores da empresa onde trabalho, em função da grande diversidade de equipamentos.

7.5 Resultados do Teste de Sistema

A seção 7.4 apresentou o ambiente de testes do sistema, descrevendo os processos de monitoramento realizados em uma máquina dentro de uma rede de computadores. Os resultados do sistema podem ser obtidos através da extração das informações coletadas na base de dados. Esse processo consiste em realizar consultas nas tabelas onde os dados encontram-se armazenados.

A partir dos dados coletados, é possível inventariar algumas características do computador monitorado. A tabela 7.3 apresenta algumas características obtidas através da análise dos dados. Essas informações foram obtidas através de

consultas às tabelas TbNetwork (colunas Host e MacAddress), TbCpu (coluna NumProcessors) e TbOperatingSystem (coluna OsName).

Tabela 7.3 Inventário do computador

Propriedade	Valor
Host	leandroquemel.riskcontrol.com.br
Número de Processadores	1
Mac Address	00:01:03:8A:18:45
Sistema Operacional	Windows 2000

Outra característica do computador que podemos obter é a lista de softwares instalados. Para isso, basta realizar uma consulta na tabela TbSoftware (coluna SoftwareName). A lista de softwares instalados é apresentada na tabela 7.4.

Tabela 7.4 Lista de aplicativos instalados

3Com 56K V.90 Mini PCI Modem
Adobe Acrobat 5.0
Adobe Reader 7.0
Aqua Data Studio
ATI Control Panel
ATI Display Driver Utilities
CVSNT
Dell ResourceCD
GetRight
Google Gmail Notifier
HTML Help Workshop
J2SE Runtime Environment 5.0 Update 2
Java 2 Runtime Environment, SE v1.4.2_06
LiveUpdate 1.7 (Symantec Corporation)
Lucent Win Modem
Microsoft .NET Framework 1.1
Microsoft Data Access Components KB870669
Microsoft Office 97, Standard Edition
Microsoft Outlook 98
Microsoft SQL Server 2000
Microsoft SQL Server Desktop Engine (SERVER)
Microsoft VGX Q833989
Microsoft Visual C++ 6.0 Standard Edition
MSN Messenger 7.0
MySQL Connector/ODBC 3.51
MySQL Servers and Clients 4.0.14b
ORiNOCO AP Manager
Outlook Express Q823353
PCTEL 2304WT V.92 MDC Modem
PowerArchiver

PuTTY version 0.55
Symantec AntiVirus Client
TortoiseCVS 1.8.7
TSW WebCoder 5
WebFldrs
WinCvs 1.3
Windows 2000 Hotfix - KB840315
Windows 2000 Hotfix - KB873339
Windows 2000 Hotfix - KB885835
Windows 2000 Hotfix - KB885836
Windows 2000 Hotfix - KB889293
Windows 2000 Service Pack 4
Windows Media Player 7.1
WinSQL Lite with ODBC Drivers
WinVNC 3.3.3

Conforme foi apresentado no capítulo 5, as informações dos processos em execução são armazenadas na tabela TbProcess. A análise dos dados armazenados nessa estrutura permite uma análise bastante interessante. As colunas da tabela informam o consumo de CPU e memória em um dado instante. Um relatório pode ser criado como ferramenta analítica da informação, extraindo os dados de consumo máximo, médio e mínimo dos recursos computacionais por processo. Com base nessa análise, é possível mensurar os requisitos mínimos de processador e memória que um computador deveria ter para executar um determinado aplicativo. Essa análise torna-se mais precisa a medida que a quantidade dados coletados de um determinado processo aumenta. Poderíamos desenvolver métodos estatísticos para normalizar a curva de consumo de recursos computacionais exigidos pelo aplicativo em execução em diversos cenários (diferentes plataformas). Com isso, o valor médio obtido seria o parâmetro recomendado pelo software, enquanto que o valor mínimo pode ser obtido analisando a cauda inferior da curva normal.

A tabela 7.5 apresenta a lista dos processos com seus respectivos valores de consumo máximo e médio de recursos computacionais.

Tabela 7.5 Lista dos processos executados

Processo	Consumo Max Cpu (%)	Consumo Médio Cpu (%)	Consumo Max Memória (MB)	Consumo Médio Memória (MB)
vptray.exe	0	0	3	3
lsass.exe	0	0	2	1
hidserv.exe	0	0	1	1
spoolsv.exe	0	0	4	4
Rtvscan.exe	2	0	8	5
qttask.exe	0	0	1	1
mspmspsv.exe	0	0	1	1
EXCEL.EXE	22	2	44	18

Processo	Consumo Max Cpu (%)	Consumo Médio Cpu (%)	Consumo Max Memória (MB)	Consumo Médio Memória (MB)
DefWatch.exe	2	0	1	1
services.exe	2	0	8	5
MAPISP32.EXE	0	0	1	1
gnotify.exe	2	0	4	3
winlogon.exe	2	0	2	2
regsvc.exe	0	0	0	0
taskmgr.exe	2	0	1	1
iexplore.exe	34	1	19	10
atiptaxx.exe	0	0	2	2
ORACLE.EXE	0	0	2	2
sqlservr.exe	0	0	4	3
pageant.exe	0	0	1	1
Ati2evxx.exe	0	0	1	1
svchost.exe	2	0	7	2
Riskcontrol.exe	**100**	**17**	**53**	**27**
MSTask.exe	0	0	2	2
jusched.exe	0	0	1	1
MonitorAgent.exe	0	0	3	3
smss.exe	0	0	0	0
Unknown	4	0	0	0
sqlmangr.exe	0	0	3	3
WinMgmt.exe	2	0	0	0
OSA.EXE	0	0	2	2
mysqld-nt.exe	0	0	5	3
MsnMsgr.Exe	30	1	27	7
internat.exe	0	0	1	1
outlook.exe	0	0	13	12
Explorer.EXE	28	1	13	9

Através dos dados acima, é possível notar que o aplicativo RiskControl.exe em determinados momentos consome 100% do tempo de CPU. O plano de execução desse programa deve ser agendado para horários não conflitantes com outros programas que também demandam quantidades de recursos computacionais relevantes, tais como os programas anti-vírus. Uma outra alternativa seria fazer um upgrade de processador.

As análises de consumo de CPU e memória também podem ser consolidadas, isto é, uma visão do conjunto de processos como um todo. Essas informações podem ser extraídas das tabelas TbCPU e TbMemory. As figuras 7.4 e 7.5 ilustram os gráficos criados no Ms-Excel a partir das consultas realizadas.

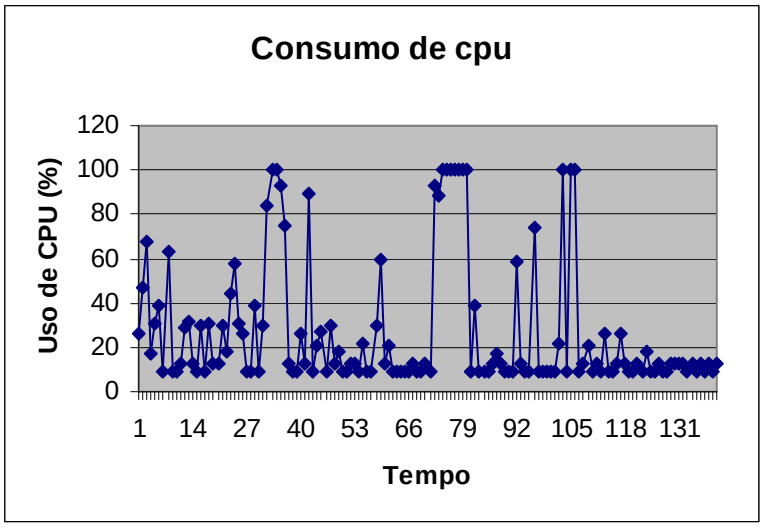

Figura 7.4 Gráfico do consumo de CPU

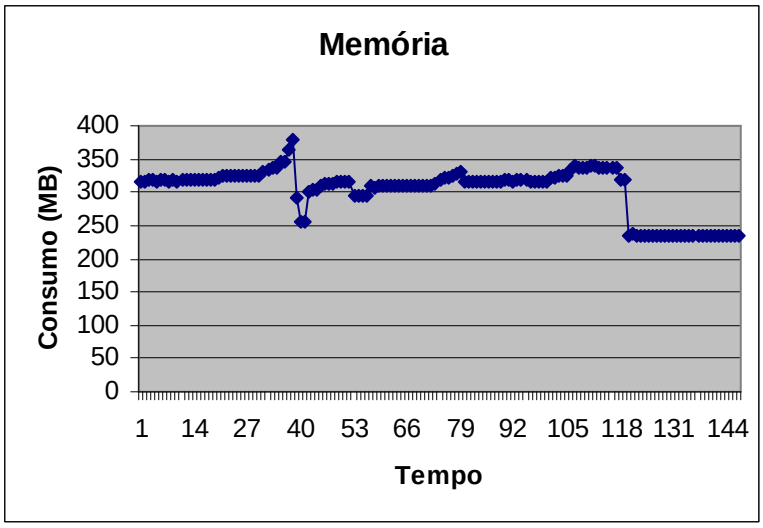

Figura 7.5 Gráfico do consumo de Memória

Capítulo 8

Conclusões

8.1 Conclusões do Projeto

O projeto proposto teve como objetivo realizar o monitoramento remoto de recursos computacionais de computadores dentro de uma arquitetura de redes. Como podemos ver através dos resultados apresentados no capítulo 7, o objetivo foi alcançado com êxito. Informações como o consumo de CPU e Memória foram capturados pelo agente, e enviados através da rede usando o protocolo TCP\IP para um back-end remoto. Em seguida, as informações recebidas foram armazenadas. O sistema desenvolvido funcionou perfeitamente nas máquinas com sistema operacional Windows (versões 98, 2000 e XP), garantindo a confiabilidade do sistema.

A utilização da linguagem C++ no projeto foi bastante eficiente. Através do paradigma OO foi possível desenvolver módulos independentes, compostos por classes que encapsulavam as operações e dados pertinentes. Com isso, garantimos elevada coesão nos módulos e baixo acoplamento nas interfaces de comunicação. Essa estrutura garantiu a manutenibilidade do projeto, visto que o impacto nas alterações em um módulo é reduzido devido ao baixo acoplamento. Outro fator importante adquirido foi a alta reutilização de código, reduzindo o custo de desenvolvimento do projeto.

Os testes de unidade comprovaram que as medidas coletadas pelos plug-ins estavam de acordo o funcionamento das máquinas. Os testes de integração verificaram as interfaces de comunicação entre os diversos módulos (agente-plug-in, arquitetura cliente-servidor, receiver - base de dados). Os testes de validação fizeram uma varredura no sistema através da ótica do usuário. Os testes de sistema realizaram as rotinas de verificação do sistema integrado como um todo.

Os resultados finais comprovaram que o projeto é capaz de monitorar um computador remotamente. Os consumos de CPU e memória podem ser medidos com uma freqüência de até 1 minuto. A lista de processos pode ser usada como ferramenta complementar, detalhando o consumo dos recursos por cada aplicativo. Listas dos softwares instalados e outras características das máquinas como interfaces de rede e sistema operacional instalado também podem ser monitoradas.

8.2 Propostas para continuidade do projeto

Embora os requisitos funcionais estabelecidos tenham sido cumpridos com êxito, o sistema apresenta algumas limitações. A primeira restrição envolve o fato que o sistema funciona somente nas plataformas Windows. Como o número de servidores e desktops da família UNIX está crescendo, seria interessante estender o monitoramento para esses sistemas operacionais. Conforme foi visto nos diagramas do capítulo 3, as classes dos plug-ins foram desenvolvidas através da metodologia Generalização\Especialização onde foram criadas superclasses de monitoramento para cada recurso, e subclasses específicas de monitoramento nas plataformas Windows. A extensão para incluir os sistemas operacionais da família UNIX pode ser feita através da criação de subclasses específicas que contém a implementação da coleta dos dados nesses ambientes.

Outra questão que deve ser abordada é o protocolo de comunicação utilizado para transmissão dos dados. O protocolo TCP é bastante eficiente e apresenta características bastante interessantes como segurança, confiança e desempenho. No entanto, o monitoramento pode ficar restrito às redes locais. Isso pode acontecer, pois é bastante comum o uso dos dispositivos firewalls, que são responsáveis por bloquear algumas portas de transmissão de dados, garantindo maior segurança nas arquiteturas de rede. O uso do protocolo http torna-se interessante à medida que possibilita o monitoramento remoto usando a Internet como meio de comunicação. Por exemplo, através desse protocolo, poderíamos monitorar uma máquina do DEL enviando dados para um back-end remoto, localizado em qualquer parte do globo.

Como foi visto no capítulo 5, o serviço de banco de dado usado foi o SQL SERVER da Microsoft. Embora esse serviço apresente características bastante atraentes tais como confiabilidade, segurança e desempenho, a utilização dessa ferramenta implica em um alto custo para o projeto, em função do preço da licença do produto. Uma alternativa seria a integração do sistema com um serviço de banco de dados gratuito, como o PostgreSQL.

Poderíamos imaginar versões futuras do produto multiplataformas, isto é, o software operando em diversos sistemas operacionais tais como o Wndows e os sistemas da família UNIX. Essa questão pode ser justificada pelo fato que há uma tendência cada vez mais forte em adotar os sistemas operacionais UNIX, tanto para servidores quantos para estações de trabalho. Para que isso se torne possível, poderíamos modificar as interfaces gráficas do sistema (que estão implementadas em C++, através da biblioteca MFC), usando os recursos da biblioteca multiplataforma WxWindows.

Por fim, a última questão que gostaria de ressaltar é a modelagem de dados desenvolvida. As tabelas criadas atendem às expectativas exigidas em um modelo conceitual de banco: desempenho, garantia de integridade dos dados, redução de anomalias e redundância de dados através da normalização. Porém, a adição de um novo plug-in ou a modificação na estrutura de dados enviados por um componente implicaria na alteração das estruturas do banco, através de comandos DDL. Uma alternativa seria a elaboração de modelo de dados genérico, reduzindo os impactos causados nas alterações do projeto.

Referências Bibliográficas

[1] **Pressman, Roger S. – Engenharia de Software 5ª Edição**

[2] **TANENBAUM, A. S., Computer Networks. 3 ed. Prentice Hall, 1996.**

[3] **VILLAS-BOAS, S. B., *C/C++ e Orientação a Objetos em Ambiente Multiplataforma*. Rio de Janeiro, RJ, Brasil, 2001.**

[4] **RAMEZ E. ELMASRI, SHAMKANT NAVATHE, Sistema Gerenciador de Banco de dados**

[5] **TANENBAUM, A. S., Sistemas Operacionais Modernos 2 ed. Prentice Hall, 1996.**

[6] **LARMAN, CRAIG, Applying UML and Patterns 2 ed. Prentice Hall, 2001.**