UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ROBERTO DE SOUSA BOAVA

UTILIZANDO ARQUITETURA MULTICAMADAS NO DESENVOLVIMENTO DE UMA FERRAMENTA DE MONITORAMENTO DE PERIFÉRICOS E LICENÇAS DE SOFTWARES

CRICIÚMA, JULHO DE 2007

ROBERTO DE SOUSA BOAVA

UTILIZANDO ARQUITETURA MULTICAMADAS NO DESENVOLVIMENTO DE UMA FERRAMENTA DE MONITORAMENTO DE PERIFÉRICOS E LICENÇAS DE SOFTWARES

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. M.Sc. Paulo João Martins

CRICIÚMA, JULHO DE 2007

ROBERTO DE SOUSA BOAVA

UTILIZANDO ARQUITETURA MULTICAMADAS NO DESENVOLVIMENTO DE UMA FERRAMENTA DE MONITORAMENTO DE PERIFÉRICOS E LICENÇAS DE SOFTWARES

Submetido ao corpo docente do Departamento de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

_____

Prof. M. Sc. Ana Claudia Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

_____

Prof. M. Sc. Paulo João Martins (UNESC)
Orientador

_____

Prof. M. Sc. Paracelso de Oliveira Caldas (UNESC)

_____

Esp. Carina Búrigo (UNESC)

A Deus e a minha família.

AGRADECIMENTOS

Aos meus queridos pais, João e Zená, pelos ensinamentos que me proporcionaram e por estarem sempre ao meu lado.

A minha noiva, Karine, que sempre esteve ao meu lado, nos momentos difíceis, incentivando-me na conclusão deste trabalho.

À Fátima, minha tia, cuja fé, bondade, coragem e força, me ampara e dignifica, servindo-me como exemplo daquilo que pretendo vir a me tornar como profissional e ser humano.

A minha Avó, Lídia, pelas orações e amor, a que serei eternamente grato.

Ao meu orientador, Prof. Paulo João Martins, pela atenção, orientação e apoio.

Aos mestres e funcionários da UNESC que sempre se mostraram prestativos e atenciosos durante a minha vida acadêmica.

Aos meus amigos, que sempre me deram apoio no que precisei e que sei que posso contar com eles a qualquer hora.

RESUMO

Este projeto consiste no desenvolvimento de uma ferramenta para auxiliar o controle dos periféricos e licenças de softwares instalados nos computadores em uma rede local. A ferramenta foi desenvolvida para funcionar no sistema operacional Windows a partir da versão 95. Para o desenvolvimento desta ferramenta, foi utilizado a tecnologia de multicamadas juntamente com a linguagem de programação Delphi 7.0. No decorrer do trabalho, são apresentadas algumas comparações entre as tecnologias Cliente/Servidor e multicamadas, apresentando as suas vantagens e desvantagens. Dentro do capítulo sobre multicamadas, apresentar-se-ão os protocolos de comunicação para esta tecnologia. Para obter os programas e os periféricos que estão instalados no computador explorou-se os recursos do registro do windows onde são extraidas estas informações.

Palavras chaves: Arquitetura multicamadas; Registro do Windows; Licenças de Softwares;

ABSTRACT

This project consists in a tool development to help the control of peripherals and softwares licenses installed in a network. This tool was developed to work on Windows Operating System, starting in 95 version. To develop this tool, the multilayers technology was used together with the Delphi 7.0 programming language. In this work, some comparisons are presented between multilayers and client/server technologies, detaching its advantages and disadvantages. In the chapter about multilayers, communication protocols will be presented to this technology. To obtain the programs and peripherals installed in the computer, it was explored the windows registry resources where these informations are extracted from.

Key words: Multilayers Architecture; Windows Registry; Softwares Licenses.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API	*Application Programmer Interface*
COM	*Commom Object Model*
COM+	*Component Object Model Plus*
CORBA	*Common Object Request Broker Architecture*
DCE	*Distributed Computing Environment*
DCOM	*Distributed Component Object Model*
DDE	*Dynamic Data Exchange*
DLL	*Dynamic Link Library*
EDI	*Electronic Data Interchange*
EXE	Extensão de Arquivos Executáveis
GUI	*Graphical User Interface*
HTTP	*HyperText Transfer Protocol*
IDL	*Interface Definition Language*
IP	*Internet Protocol*
LPC	*Local Procedure Call*
MSDTC	*Microsoft Distributed Transaction Coordinator*
MTS	*Microsoft Transaction Server*
OLE	*Object Linking and Embedding*
OMG	*Object Management Group*
POO	Programação Orientada a Objetos
RDM	*Remote Data Modules*
RPC	*Remote Procedure Call*
SGBD	Sistema Gerenciador de Banco de Dados

SQL	*Structure Query Language*
TCP	*Transmission Control Protocol*

SUMÁRIO

1 INTRODUÇÃO

Devido à popularização da Internet, do CD-ROM, do Pendrive entre outros equipamentos que possibilitam trocas de arquivos nos computadores das corporações, torna-se muito difícil controlar as instalações de softwares não autorizados por parte dos usuários. Muitas vezes são instalados jogos, programas de demonstração, gratuitos ou até mesmo softwares sem licença de uso. Estas instalações indevidas podem trazer diversos problemas, desde o comprometimento do bom funcionamento do computador até o risco de multas e processos judiciais contra a instituição pelo uso de programas não licenciados (ABES, 2002). Diante deste contexto, o controle dos softwares instalados nos computadores torna-se tão importantes quanto a manutenção dos hardwares.

Com o crescimento do mercado, as corporações vêm investindo em grande escala no uso de computadores e, conseqüentemente, em softwares para melhorar a sua competitividade e desempenho, seja no campo administrativo ou da produção, e assim melhorar o atendimento aos seus clientes.

Desta forma o parque tecnológico tem aumentado, surgindo com isso, a necessidade de analisar e verificar se os softwares utilizados pelos funcionários encontram-se licenciados pela empresa e dentro da legalidade, de acordo com as regras da política da empresa e das leis que regem a proteção aos softwares. Há a necessidade de saber quais softwares os usuários estão utilizando, a fim de analisar o tipo de licença de cada um e permanecer em dia com a legislação pertinente. Sobre a licença de uso dos programas de computadores na área de Tecnologia da Informação, têm-se a necessidade de saber quais softwares necessitam de atualização, e manter um controle sobre quais programas foram atualizados. Por outro lado, precisa ter-se um controle sobre a

manutenção dos computadores, ou seja, necessita-se de uma lista de seus periféricos, pois quando enviados para a manutenção precisa-se de um retorno confiável sobre a originalidade dos equipamentos para um possível controle no ato do retorno. Este controle pode ser realizado de forma manual, podendo ocasionar um gasto substancial de tempo e informações inconsistentes, tal como: anotações errôneas, isto normalmente acontece devido ao pouco conhecimento da pessoa designada a realizar essa atividade.

O objetivo deste projeto foi desenvolver uma ferramenta de inventário de softwares e hardwares para proporcionar o controle sobre a utilização dos aplicativos e periféricos instalados no parque tecnológico das empresas e também de usuários domésticos, de forma a manter um controle mais eficiente, por parte da pessoa responsável pela realização desse diagnóstico.

1.1 OBJETIVO GERAL

Desenvolver o protótipo de uma ferramenta para realizar o controle sobre as licenças dos softwares instalados e controle dos periféricos que se encontram nos computadores do ambiente pesquisado, utilizando-se da arquitetura multicamadas.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos dessa pesquisa consistem em:

a) compreender e aplicar os conceitos sobre a arquitetura multicamadas;
b) relacionar as vantagens e desvantagens no desenvolvimento de softwares, utilizando-se da arquitetura multicamadas, frente a arquitetura cliente/servidor;

c) aplicar os conhecimentos sobre o registro do Windows;

d) montar uma base de dados para realizar um inventário sobre periféricos e os softwares instalados no computador;

e) realizar um inventário dos softwares e hardwares instalados.

1.3 JUSTIFICATIVA

Na elaboração desse projeto pretende-se utilizar os conceitos sobre cliente/servidor no desenvolvimento do sistema, onde um servidor possa se conectar aos computadores de uma rede local, configurados de forma a permitir o acesso aos *hosts*[1], e obter as informações sobre os periféricos e os softwares instalados.

O trabalho proposto tem como característica o desenvolvimento de uma ferramenta, empregando multicamadas para relacionar os periféricos e os softwares instalados nos computadores, em uma rede local dentro da corporação. Para isto, foi utilizada a plataforma Windows, na qual foi gerada uma base de dados com relatórios técnicos, para futuramente realizar uma mineração de dados de forma a obter um histórico referente à data de instalação, versão sobre os softwares e relacionar os periféricos com cada computador da corporação, e assim, poder emitir os relatórios que se fizerem necessários, tais como: relatórios de comparação de quais softwares existiam instalados nos computadores e quais foram removidos, relatório de todos os softwares que foram instalados em um período ou relatórios dos periféricos que estão instalados no computador, bem como relacionar os programas que não estão de acordo com leis de softwares e as políticas de utilização destes pela empresa.

[1] Endereço do computador na rede.

1.4 TRABALHOS CORRELATOS

Conforme Rocha (2002), no trabalho intitulado Análise de desempenho em ambientes cliente/servidor 2-camadas e 3-camadas, foi desenvolvido dois softwares, um utilizando o modelo 2-camadas e o outro utilizando o modelo 3-camadas onde foram feitas simulações utilizando o ambiente ARENA para comparar o desempenho entre os dois modelos.

Foram considerados dois fatores para a simulação no ARENA: tempo de processamento do servidor de aplicação e o tempo de processamento do servidor de banco de dados.

Nos testes realizados, concluiu-se que o modelo 2-camadas é mais adequado, considerando-se o tempo de resposta. Porém, o tempo de resposta pode não ser o único fator determinante para o seu desenvolvimento, pois algumas características e funcionalidades também devem ser levadas em consideração, tais como: manutenção e fácil distribuição e integração com outras plataformas. Considerando-se estes fatores, o modelo 3-camadas encaixa-se de maneira muito mais favorável do que o modelo 2-camadas.

Segundo Cardoso e Esteves (2002), no trabalho intitulado Utilizando componentes em aplicações distribuídas em 3 camadas, o seu objetivo é apresentar a arquitetura de três camadas juntamente com seus conceitos, suas características, vantagens e desvantagens de sua utilização. O trabalho descreve, detalhadamente, cada uma das camadas: a camada de *Interface*, a de negócios e a de dados. Foi desenvolvido um protótipo empregando alguns conceitos de programação distribuída.

Na conclusão, os autores afirmam que é mais viável distribuir o processamento em várias máquinas de médio e pequeno porte do que ter um super

mainframe controlando tudo. O modelo de três camadas facilita e agiliza a manutenção do sistema e não sobrecarrega o servidor de dados, pois as regras e a conexão com os clientes são tratadas no servidor de aplicação.

D'este (2003), com o trabalho intitulado Protótipo de sistema de auxílio ao gerenciamento de software utilizando Agentes Móveis, foi desenvolvido um protótipo utilizando o modelo agentes móveis para o auxílio no gerenciamento de redes. Para demonstrar esse modelo foi construído um protótipo utilizando a linguagem de programação *Delphi* 6.0 para auxiliar no gerenciamento de softwares instalados nos computadores de uma rede.

Conforme Silva Netto (2004), com o trabalho denominado Questões legais sobre desenvolvimento, comercialização, direito autoral e uso de Software, propôs esclarecer os fatos relevantes a respeito de questões legais sobre o desenvolvimento, comercialização, direitos autorais e uso de software. Esse trabalho tem como enfoque a Lei de direitos autorais (Lei nº 9.610/98) e correspondentes, onde poderão ser utilizadas no auxílio do desenvolvimento de projetos de softwares.

1.5 ESTRUTURA DO TRABALHO

Este Trabalho de Conclusão de Curso está organizado em capítulos sendo os mesmos descritos a seguir:

O primeiro capítulo é composto pela introdução do trabalho, definindo seu objetivo geral, objetivos específicos, a justificativa e os trabalhos correlatos.

Logo a seguir é apresentada a arquitetura do registro do Windows, onde são armazenadas as informações dos softwares instalados, de alguns periféricos em cada computador, o seu funcionamento e sua utilidade.

Um estudo sobre licenças de softwares é desenvolvido no terceiro capítulo, sendo feito uma breve descrição sobre os principais tipos.

Cliente/servidor e multicamadas são as arquiteturas apresentadas no quarto capítulo, expondo as suas vantagens e desvantagens, e também a descrição sobre o funcionamento dos protocolos de comunicação em aplicações multicamadas.

No quinto capítulo são descritas algumas rotinas que foram desenvolvidas, a implementação em multicamadas; o funcionamento do sistema desenvolvido bem como a sua *interface* visual e os Resultados obtidos.

A seguir é apresentada a conclusão do trabalho desenvolvido, juntamente com suas considerações finais e os trabalhos futuros.

Por último, apresentam-se as referências bibliográficas e as referências complementares que foram utilizadas para a elaboração desta pesquisa.

2 BASE DE INFORMAÇÃO DO WINDOWS

O Windows possui uma base de dados onde são armazenadas diversas informações sobre os programas instalados: estrutura de diretórios, de drivers, informações de usuários e diversas outras que são necessárias para que o sistema operacional possa executar (MICROSOFT, 2007). Logo abaixo é descrito um pouco sobre esse assunto.

2.1 O REGISTRO DO WINDOWS

É um grande repositório de dados existente desde a versão 3.x. Este repositório é utilizado pelo Windows 9x para o armazenamento de informações referentes à configuração do sistema, para um ou mais usuários, aplicativos e dispositivos de hardware. O Registro contém informações que são utilizadas pelo Sistema Operacional durante a operação, tais como: perfis de cada usuário; aplicativos instalados no computador; tipos de documentos que cada usuário pode criar; os hardwares existentes no sistema e quais portas estão sendo acessadas. O registro já existia no Windows desde a versão 3x, porém os programadores pouco aproveitaram, pois preferiam a utilização de arquivos INI (TORRES, 1998).

Arquivos INI, são arquivos textos comuns, utilizados para armazenar informações sobre as configurações de determinado software. Até mesmo o Windows 9x, por motivos de compatibilidade, ainda utiliza dois arquivos INI para guardar suas configurações, são eles: o Win.ini, que é utilizado para salvar informações sobre os softwares instalados e o System.ini que é utilizado para salvar informações sobre os hardwares instalados. Estes arquivos continuam existindo de forma a manter a

compatibilidade para alguns softwares, pois estes permanecem armazenando suas informações nestes arquivos e não utilizam o Registro (ALECRIM, 2004).

O Windows possui um editor de registro onde qualquer usuário pode ter acesso. Porém, recomenda-se muito cuidado, pois qualquer alteração incorreta pode acarretar em sérios problemas que talvez exijam até mesmo a reinstalação do sistema operacional. Para acessar utiliza-se o menu iniciar/executar e digite *regedit*, após isso aparecerá uma janela como mostra a Figura 1 (ALECRIM, 2004).

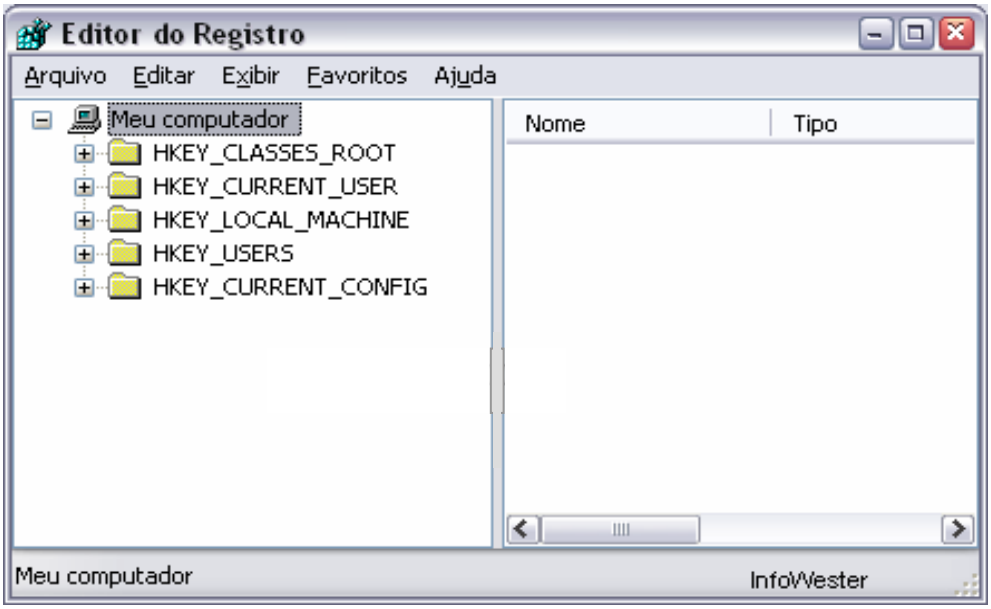

Figura 1. O Editor de Registro
Fonte: MICROSOFT (2001)

O conteúdo do registro está fisicamente armazenado nos arquivos System.dat e User.dat. O System.dat possui as configurações de hardware e de software existente no sistema operacional. No entanto, o User.dat contém as informações exclusivas sobre o usuário e as suas preferências. Para cada usuário existe um arquivo User.dat. É importante ressaltar, que muitas informações do registro são dinâmicas e adquiridas durante a inicialização do computador, como por exemplo, as configurações de dispositivos *Plug and Play*, mas que mesmo não estando armazenadas em nenhum

arquivo estão disponíveis no registro para que possam ser acessadas pelos softwares (TORRES, 98).

2.2 ESTRUTURA DO REGISTRO

O Registro é dividido, basicamente, em seis chaves: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_CONFIG, HKEY_DYN_DATA.

Em cada chave é armazenada um tipo diferente de informação. Abaixo, descreve-se o funcionamento de cada uma delas.

HKEY_CLASSES_ROOT: esta chave pertence a HKEY_LOCAL_MACHINE\Software. Ela existe para manter a compatibilidade com programas de 16 bits, pois no registro do Windows 3.x só havia uma única chave principal, chamada HKEY_CLASSES_ROOT. As informações armazenadas garantem que o programa correto seja aberto quando abrir um arquivo. A abreviação dessa chave é geralmente "HKCR" (TORRES, 1998).

HKEY_CURRENT_USER: esta chave contém a base das informações de configuração para o usuário que estiver utilizando o Windows no momento. As configurações de pastas, de cores de tela, no painel de controle do usuário e até mesmo as unidades mapeadas são armazenadas aqui. Estas informações estão associadas ao perfil de usuário. A abreviação dessa chave é geralmente "HKCU" (TORRES, 1998).

HKEY_LOCAL_MACHINE: é a mais importante do registro, pois ela contém as informações de configuração específicas para o computador. Estas informações se aplicam para todos os usuários. Fisicamente falando, as informações

dessa chave estão armazenadas no arquivo System.dat. A abreviação dessa chave é geralmente "HKLM" e possui as seguintes subchaves (ALECRIM, 2004):

a) **config**: contém as configurações para os Perfis de hardware instalados no computador. Caso sejam criados novos perfis de hardware, novas chaves são criadas. Durante o *boot*[2] o Windows carrega a configuração do perfil de hardware selecionado para a máquina;

b) **hardware**: apresenta informações sobre os hardwares encontrados no computador, tais como: portas seriais e modem;

c) ***network***: guarda informações como *logon*[3], nome do usuário, tipo de servidor, que são criadas quando um usuário se conecta a um micro conectado em rede;

d) ***security***: armazena informações sobre a administração remota e segurança da rede;

e) **software**: informações e configurações de softwares instalados no computador. As informações dessa chave são gravadas no padrão fabricante\programa\versão\configurações. Por exemplo, configurações do programa Delphi são armazenadas na chave Borland\Delphi\7.0\ReportBuilder. Importante ressaltar que as configurações do próprio Windows estão armazenadas nessa chave, em Microsoft\Windows;

f) ***system***: consiste em carregar as configurações e os *drivers* do sistema operacional durante o *boot*.

HKEY_USERS: o Windows 9x permite que mais de um usuário utilize um mesmo micro, cada um com suas configurações particulares, tais como proteção de tela,

[2] Em computação, *boot* é o processo de inicialização que carrega o sistema operacional quando ligamos o computador.
[3] Procedimento utilizado para conectar-se a uma sessão em um computador.

papel de fundo, atalhos presentes na área de trabalho, entre outros. A escolha do usuário é feita no *logon* do Windows, quando o sistema pede o nome do usuário e sua senha. As informações contidas nesta chave são específicas para cada um no sistema. Isto inclui as configurações genéricas, que se aplicam a todos os usuários, e as configurações específicas de um usuário, como por exemplo, a configuração da área de trabalho (TORRES, 1998).

Quando o sistema está configurado para o acesso por apenas um usuário, a chave HKEY_USER contém apenas uma subchave, padrão, contendo todas as configurações pessoais do sistema (proteção de tela, papel de parede, entre outros).

No caso de haver mais de um usuário configurado no sistema, quando ele faz *logon* no sistema, essa chave conterá suas configurações pessoais. Por exemplo, no caso de haver um usuário chamado Roberto, existirá uma chave chamada "Roberto" quando esse usuário entrar no sistema. A chave *default* continuará existindo, contendo as configurações padrão do sistema (ALECRIM, 2004).

Importante ressaltar que nessa chave só encontram-se disponíveis as configurações pessoais do usuário que fez *logon* do sistema. Caso o computador possuir mais de um usuário, a chave só existirá quando o usuário fizer *logon* no sistema. Se no mesmo computador existir o usuário Roberto e o usuário Paulo, a chave Roberto só aparecerá quando o Roberto fizer *logon* no sistema e a chave Paulo só apenas quando o Paulo fizer o *logon* no sistema de forma que um usuário não consiga ver nem alterar as configurações de outro (ALECRIM, 2004).

HKEY_CURRENT_CONFIG: essa chave é um atalho (ou seja, não existe fisicamente, mas apenas aponta para outra área do registro), desta vez para HKEY_LOCAL_MACHINEConfigxxxx, onde xxxx é o perfil de hardware que está atualmente configurado. Como na maioria dos computadores só há um único perfil de

hardware configurado, normalmente essa chave aponta para HKEY_LOCAL_MACHINEConfig001. Pode-se saber qual é o perfil de hardware que está sendo atualmente utilizado no sistema lendo o valor presente em HKEY_LOCAL_MACHINESystemCurrentControlSetcontrol IDConfigDB (MICROSOFT, 2007)

HKEY_DYN_DATA: todas as configurações armazenadas nas chaves anteriores são estáticas, ou seja, são armazenadas em algum lugar do disco rígido (em geral nos arquivos System.dat e User.dat). A chave HKEY_DYN_DATA contém informações dinâmicas e que existem somente na sessão atual. Essas informações são lidas durante o *boot* da máquina e contém informações como a lista de dispositivos *Plug and Play* instalados no micro. Essas informações ficam armazenadas em memória RAM e, portanto, são criadas a cada *boot* da máquina (ALECRIM, 2004).

A seguir será abordado o assunto sobre licenças de softwares, descrevendo-se os seus principais tipos de distribuição.

3 LICENÇAS DE SOFTWARES

Os programas de computador são produtos relativamente novos na vida dos consumidores. Acostumados a comprar o bem e não apenas o direito de uso, várias pessoas acabam cometendo erros de caráter legal ao adquirir programas para uso em seus computadores. Normalmente, quando se adquiri um programa de computador, o que está sendo comercializado é uma licença de uso, e não o programa propriamente dito. Quando se adquire o produto normalmente é feito um contrato, que dá o direito de uso do software pela pessoa que está comprando. Este contrato envolve apenas o desenvolvedor (fabricante) e o usuário (a pessoa que compra a licença de uso) e por isso geralmente é feito quando o usuário vai instalar o software em seu computador e não no ato da compra (BITENCOURT; SANTOS, 2004).

Os contratos de licenças de uso de softwares delimitam os direitos e obrigações do usuário e do seu desenvolvedor. O desenvolvedor deverá especificar a versão do programa licenciado, a validade técnica, a especificação de quais hardwares suporta o software a ser licenciado e as suas especificações, ou seja: tudo que o programa é capaz de fazer (SILVA NETTO, 2004).

Não se pode copiar um programa sem autorização do desenvolvedor, pois o programa é de domínio intelectual da empresa que o desenvolveu. Programas que estão fora dos acordos entre desenvolvedor e usuário são chamados de piratas e o uso dos mesmos é ilegal, sendo assim os usuários que copiarem ou distribuírem programas sem a devida autorização estão sujeitos a multas (SCHNEIDER; UCHÔA, 2000).

A pirataria de software é uma prática ilegal, caracterizada pela cópia sem autorização ou uso indevido de programas de computador legalmente protegidos. Ela ocorre também quando alguém faz mais cópias de um programa do que o permitido ou

quando, por exemplo, uma pessoa empresta a cópia de um programa para outra. Ao adquirir um programa, os consumidores estão comprando apenas o direito de uso daquele software. Ao aceitar o acordo de licenciamento que acompanha quase toda instalação de software, eles não adquirem os direitos de revenda ou reprodução do programa (BITENCOURT; SANTOS, 2004).

Existem várias maneiras de se classificar as licenças e existe uma grande variedade delas. As principais categorias de licenças de uso são: software livre, software comercial, *freeware*, *shareware*, software proprietário e demo (ZIPF, 2003).

3.1 SOFTWARE LIVRE

O software livre é um programa de computador como qualquer outro programa proprietário, possui o mesmo intuito, ou seja, é direcionado para atender um determinado seguimento como, por exemplo: os editores de textos, os editores de imagens, as planilhas de cálculos, entre outros. A diferença dos outros, segundo a Fundação Software Livre é que esta licença deve garantir as quatro liberdades. O primeiro a formalizar o software nesta maneira de pensar sobre a forma de quatro liberdades, foi Richard M. Stallman no início dos anos 80, são elas (SILVA NETTO, 2004):

a) a liberdade de executar o programa, para qualquer propósito;
b) a liberdade de estudar como o programa funciona e adaptá-lo para as suas necessidades. Acesso ao código fonte é um pré-requisito para esta liberdade;
c) a liberdade de redistribuir cópias, permitindo a ajuda ao próximo;

d) a liberdade de aperfeiçoar o programa e tornar as modificações públicas de modo que a comunidade inteira se beneficie da melhoria. O acesso ao código fonte é um pré-requisito para esta liberdade.

3.2 SOFTWARE COMERCIAL

Softwares comerciais são aqueles produzidos e comercializados, cujos desenvolvedores possuem a ambição de conquistar um público-alvo coletivo qualquer, onde normalmente o usuário não tem acesso ao código fonte e para utilizá-lo o usuário deve comprar uma licença de uso. Os softwares comerciais podem ser desenvolvidos a partir de um pedido de um contratante, ou ainda, pode ser desenvolvido a partir da iniciativa própria do programador com o objetivo de retorno financeiro com a sua utilização (SILVA NETTO, 2004).

3.3 FREEWARE

São os programas disponíveis publicamente sem custo de licenciamento para uso, conforme condições estabelecidas pelos desenvolvedores. Os desenvolvedores de programas *freeware* não estabelecem limitações quanto ao uso de seus softwares. Em geral, podem ser utilizado sem custos e normalmente não possuem garantia de manutenção ou atualização. Mesmo sendo um programa gratuito, o *freeware* não autoriza a alteração, e nem obriga o desenvolvedor a expor o código fonte. Muitos usuários pensam que por serem gratuitos, esses softwares possuem uma qualidade inferior aos pagos, mas isso não é verdade, pois muitas vezes são equivalentes ou quando não chegam a serem melhores (BITENCOURT; SANTOS, 2004).

Se alguém desejar alterar o código fonte de um *freeware* necessitará da permissão do programador. O desenvolvedor poderá fornecer o código fonte de forma gratuita ou cobrar para fornecê-lo. O programador tem o direito de barrar e inclusive remover do mercado qualquer programa copiado indevidamente ou sem sua autorização (SILVA NETTO, 2004).

Um fato que merece atenção é que muitas vezes os programas *freeware* são divulgados e disponibilizados para o público antes de estarem totalmente prontos, isto é, ainda faltam algumas funções no programa a serem desenvolvidas, essas funções serão acrescentadas a ele na medida do possível. Programas neste período de desenvolvimento são chamados de versões beta e normalmente são identificados com um número de versão que começa com 0 (zero). Isto significa que ele já pode ser utilizado, mas ainda não faz tudo o que deveria. Geralmente, o contrato de uso de um programa *freeware* afirma que ao usar aquele produto, o usuário isenta o programador de qualquer responsabilidade sobre possíveis estragos originados por defeitos do software. O grande exemplo de um programa *freeware* é o sistema operacional Linux (SCHNEIDER; UCHÔA, 2000).

3.4 SHAREWARE

São produtos disponíveis publicamente para avaliação, onde os usuários podem experimentar em um determinado período de tempo sem custo. Quando esse período de tempo esgota-se o usuário torna-se obrigado a registrá-lo ou o mesmo poderá deixar de funcionar. Existem alguns casos que programas *shareware* podem ausentar o usuário do pagamento sob algumas condições, como por exemplo: uso educacional, uso sem fins lucrativos entre outros (SCHNEIDER; UCHÔA, 2000).

Este tipo de licença é uma boa alternativa para se promover a comercialização de um software, pois ele pode ser usado, copiado e distribuído livremente. Porem, seu uso é temporário e estabelecido pelo desenvolvedor. O usuário tem a opção de continuar aproveitando o software até a data estabelecida pelo programador, ou meramente deixar de usá-lo, desinstalando-o de seu computador. Esta é uma forma que o usuário tem de testar na prática as qualidades do software. Caso o usuário desejar adquiri-lo em definitivo, deverá fazer o pagamento da licença de uso ao fabricante (SCHNEIDER; UCHÔA, 2000).

3.5 SOFTWARE PROPRIETÁRIO

Conforme Silva Netto (2004) software proprietário é aquele onde a modificação ou redistribuição é proibida pelo seu desenvolvedor, é necessário pagar por ele ou obter uma autorização do desenvolvedor para usar ou redistribuir. É permitido apenas 1 cópia por parte de quem adquiri o direito de uso para ficar com *backup*, devido ao fato que pode acontecer a perda do CD original. O criador do software proprietário não é obrigado a disponibilizar o código fonte, pois pertence ao seu criador, isto é, este tipo de licenciamento garante ao fabricante plenos poderes sobre a venda, modificação ou distribuição do código fonte.

Geralmente, esse tipo de software é considerado como comercial e por isso, normalmente, os usuários pagam pela sua utilização, o que lhe permite utilizar uma ou mais licenças, dependendo do contrato. Caso queira instalar em mais computadores do que a licença permite, é necessário adquirir do fornecedor o direito de uso de mais licenças. Pode-se citar como exemplo de software proprietário, o sistema operacional Windows (SILVA NETTO, 2004).

3.6 DEMO

Devido a sua distribuição ser gratuita, o software demo ou demonstração é confundido facilmente com o *freeware* e principalmente com *shareware.* Uma das características mais marcantes nos demos é a ausência de algumas funcionalidades que o software completo teria. O demo pode ser distribuído com a mesma *interface* do software completo, sendo desativados ou retirados alguns botões, ou ainda com apenas parte dele, sem acesso a outras janelas (BITENCOURT;SANTOS, 2004).

Para efeito legal, o demo deve representar de forma fiel o software completo a que se refere, de acordo com o Art 1º da Lei de software que determina ser o software capaz de funcionar de modo e para fins determinados (SILVA NETTO, 2004).

> Programa de computador é a expressão de um conjunto organizado de instruções em linguagem natural ou codificada, contida em suporte físico de qualquer natureza, de emprego necessário em máquinas automáticas de tratamento da informação, dispositivos, instrumentos ou equipamentos periféricos, baseados em técnica digital ou análoga, para fazê-los funcionar de modo e para fins determinados (LEI Nº. 9.609, 1998 Art. 1º.).

A seguir será abordado o assunto sobre arquitetura de softwares, sendo descrito sobre a arquitetura Cliente/Servidor com suas vantagens e desvantagens, a arquitetura multicamadas suas vantagens, desvantagens e seus protocolos de comunicação.

4 ARQUITETURA DE SISTEMAS COMPUTACIONAIS

Para o bom entendimento sobre o funcionamento de aplicações desenvolvidas para rede, é importantíssimo entender sobre arquitetura de sistemas computacionais. Assim como a arquitetura, em seu significado tradicional, determina a estrutura de edifícios, casas, construções onde habitamos, no contexto computacional a arquitetura de software determina o que é o sistema em termos de componentes computacionais, define a estrutura, o objeto do projeto e da implementação, ou seja, a maneira em que o sistema será desenvolvido (FERREIRA, 2003).

> Arquiteturas de softwares são as estruturas que incluem componentes, suas propriedades externas e os relacionamentos entre eles, constituindo uma abstração do sistema. Esta abstração suprime detalhes de componentes que não afetam a forma como eles são usados ou como eles usam outros componentes, auxiliando o gerenciamento da complexidade (VAROTO, 2002 p.7).

A arquitetura de software é o conjunto de componentes e seus relacionamentos, portanto ela deve atender os requisitos funcionais e não funcionais do sistema. Pode-se afirmar que a arquitetura é a *interface* entre o problema e a solução técnica. Ela deve atender todas as exceções e regras não importando, se são de negócio ou as que influenciam nos aspectos técnicos, pois ela deve constituir um modelo simples e inteligente de como o sistema deve ser estruturado (VAROTO, 2002).

Tratando-se de tempo e custo do desenvolvimento de sistema, a reutilização é uma característica marcante da arquitetura de software, pois se for bem organizada e estruturada, cada componente computacional pode ser construído de forma a permitir reutilizá-lo (VAROTO, 2002).

4.1 ARQUITETURA CLIENTE/SERVIDOR

Em uma arquitetura cliente/servidor, tradicionalmente encontra-se uma aplicação sendo executada em uma unidade cliente, e esta unidade pode enviar solicitações de dados para um servidor, pedindo para que alguma tarefa seja executada. Em seguida, ele executa essa tarefa e envia a resposta ao cliente, ou seja, o servidor age como provedor de recursos para as entidades clientes. Servidor é alguma máquina da rede responsável por servir as requisições dos clientes conforme os serviços disponíveis nelas, sendo que os servidores mais comuns são os de arquivo, aplicativos de correio, de rede e principalmente de banco de dados. **(**CUNHA, 2003).

Uma das tecnologias mais utilizadas no mundo em ambientes corporativos é a arquitetura cliente/servidor. Isso ocorre, talvez pelo fato deste modelo possuir o melhor custo benefício dentre outros modelos existentes (CARDOSO; ESTEVES, 2002).

O título cliente/servidor define muito bem a essência desta abordagem de processamento computacional. Baseiam-se em duas camadas, uma cliente, computadores pessoais e outra servidora.

> O paradigma cliente/servidor provocou uma mudança radical nos conceitos relativos na forma de manipular e distribuir informações em um ambiente computacional, na medida em que provocou modificações na forma de desenvolvimento de sistemas de aplicação e permitiu que o ambiente corporativo se valesse de decisões estratégicas mais ágeis e rápidas, além de fomentar a busca de novas pesquisas tecnológicas de hardware e software que melhor tirassem benefícios deste novo modelo pelos centros de pesquisas acadêmicas e pela indústria de informática (ROCHA, 2002 p.18).

Normalmente é no servidor que ficam os sistemas mais robustos da rede, tal como o banco de dados, pois geralmente as máquinas clientes não possuem muitos recursos de hardware quanto os servidores, podendo assim não ter a capacidade de executar aplicativos que necessitem de muitos recursos e desempenho.

Apesar de este ambiente ter sido muito utilizado na década de 1990, com a Internet surgiram novas necessidades, como por exemplo, a separação das regras de negócio do programa cliente, divisão de tarefas nos servidores para melhorar o desempenho, desenvolver sistemas independentes da tecnologia do banco de dados, enfim, necessidades não suportadas pela arquitetura cliente/servidor. Com isso, surgiu à tecnologia do desenvolvimento de aplicações multicamadas, acompanhando a evolução de outros modelos da computação, como a Internet, intranet, orientação a objetos, e objetos distribuídos (ROCHA, 2002). A Figura 2 demonstra a arquitetura Cliente/Servidor.

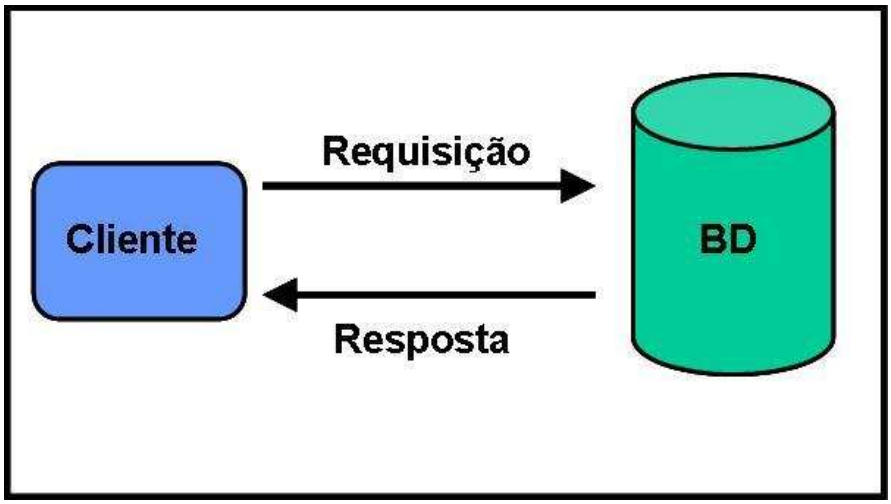

Figura 2. Arquitetura básica de um cliente/servidor
Fonte: DAMÁZIO, E. (2005).

4.1.1 Vantagens do modelo cliente/servidor

São relacionadas logo abaixo algumas vantagens que possuem a arquitetura cliente/servidor (SANTOS JÚNIOR, 2003):

a) permite a diminuição do tráfego na rede em relação ao modelo de processamento centralizado;

b) possibilidade de se utilizar ferramentas de desenvolvimento visual ou linguagens de programação visual, utilizando conceitos de orientação ao objeto. Exemplo. *Delphi*,*Visual Basic*;
c) descentralização de informações e processamento;
d) permiti a integração de tecnologias diferentes, como SGBD que gerenciam banco de dados e linguagem de programação que elaboram os sistemas corporativos;
e) ambiente mais produtivo para o desenvolvedor e para o usuário final;
f) custo baixo de software - hardware.

4.1.2 Desvantagens do modelo cliente/servidor

Como qualquer tipo de tecnologia, também existe algumas desvantagens, tais como (SANTOS JÚNIOR, 2003):

a) dificuldade da gerencia de segurança, usuário e aplicações, pois os dados confidencias que circulam na rede necessitam de criptografia;
b) dificuldade de integração de sistemas;
c) aplicações são executadas nas estações de forma independente, podendo acessar serviços no servidor;
d) problemas de saturação da rede, (desempenho, “gargalo” na rede) *Interface*, regra de negócio e acesso a dados muito acoplados;
e) a lógica do negócio toda inerente ao cliente e sua repetição por todos os clientes da rede, gera uma complexa atualização dos mesmos, em função da necessidade de um grande gerenciamento do software espalhado pela rede;

f) a insegurança dos dados no nível do cliente, o que permite que usuários mal intencionados manipulem as informações do banco de dados;

g) possui baixo encapsulamento[4] de dados, uma vez que o cliente manipula dados do banco de dados;

h) baixo nível de reuso da aplicação devido à lógica de controle do negócio estar centralizada no cliente;

i) o desempenho do sistema pode ser comprometido. Instruções SQL enviadas por meio da rede com dados, sendo baixados no cliente para análise, aumentando o tempo de resposta.

4.2 ARQUITETURA MULTICAMADAS

Conforme Rodrigues (2002) nos últimos anos esta tecnologia vem sendo muito falada, discutida, mas infelizmente, pouco implementada pelos desenvolvedores de softwares. Aplicações multicamadas são tecnologias que podem ser executadas em um conjunto de máquinas distribuídas em uma rede local, onde tais máquinas provêem acesso a objetos distribuídos[5] pela rede. Estas máquinas hora fazem o trabalho do cliente, hora fazem o trabalho do servidor. A distribuição desses objetos dar-se-á pela independência de localização. São sistemas projetados para diminuir e distribuir a carga de processamento entre vários servidores.

A arquitetura em multicamadas envolve a separação das funcionalidades usando camadas, com o objetivo de separar a lógica de apresentação, a lógica de negócio e a conexão com o banco de dados (lógica de acesso a dados). A separação em

[4] São rotinas privadas dentro do objeto. Oculta os detalhes da implementação.
[5] São objetos que armazenam código para executar uma determinada tarefa, independente da localização. Isto é, eles podem estar na mesma máquina (COM) ou em máquinas distintas (DCOM, COM+, CORBA).

camadas torna o sistema mais flexível, de modo que partes podem ser alteradas independentemente. Com o emprego desta arquitetura, qualquer alteração em uma determinada camada não influi nas demais, desde que o mecanismo de comunicação entre elas permaneça inalterado (EDWARDS, 1999 tradução nossa).

A finalidade da arquitetura multicamadas é dividir a *interface*, os dados e as regras de negócio em camadas, de forma que cada camada seja independente uma da outra. Sendo assim, as camadas podem ou não, ficarem em servidores distintos. A finalidade desta arquitetura é abranger a portabilidade e facilitar a manutenção. Assim, é recomendável que estejam em locais diferentes para garantir a portabilidade a cada camada (ROCHA, 2002).

Falar em multicamadas significa dizer que a aplicação será dividida, realmente, em várias camadas, ou geralmente em três camadas bem definidas e distintas como descrito abaixo (DAMÁZIO, 2005).

a) ***interface***: é a parte visual, o software aplicativo, ou seja, o cliente. Onde ele vê os resultados do sistema e da entrada nas informações, essa camada não deve possuir nenhuma complexidade no seu código;
b) **regras de negócio**: ficam no servidor de aplicação[6], que é a segunda camada. Todas as regras de negócio ficam no servidor, deixando assim a aplicação organizada e com regras centralizadas. Toda a complexidade do sistema, desde tomada de decisões, procedimentos deve se concentrar nessa camada;
c) **banco de dados**: representa o servidor de banco de dados, em aplicações multicamadas, servirá apenas como um repositório de dados.

[6] É a máquina/computador físico que ficará responsável pelo processamento das regras de negócio.

Muitos se questionam, se existe a tecnologia multicamadas, algum dia já existiu duas camadas ou uma camada?

Pode-se afirmar que duas camadas é o que hoje se conhece como cliente/servidor e uma camada são os famosos mainframes, onde a arquitetura é Uniprocessada. No desenvolvimento de uma aplicação multicamada, nada impede a criação de uma camada intermediária. Deste modo, pode-se ter uma camada de conexão ativa com o banco de dados, escrita dentro dos objetos de regras de negócio. Sendo assim, outros objetos criados dentro da aplicação, como por exemplo, objeto orçamento, objeto pedido, não necessitam criar a sua própria camada de conexão. Eles utilizarão a herança do objeto que conectou no banco de dados. Assim, esses módulos podem chamar a camada intermediária e por meio dela se conectar ao banco de dados. Desta forma, pode-se ter uma pequena noção da quarta camada. A quinta camada pode ser uma camada de funções, métodos que servirão como base para o desenvolvimento da aplicação. Desta forma, pode-se desenvolver aplicações em quantas camadas o responsável pelo projeto do software achar necessário para a sua aplicação, e não apenas em duas ou três camadas (RODRIGUES, 2002).

A Figura 3 mostra uma ilustração do funcionamento da tecnologia multicamadas.

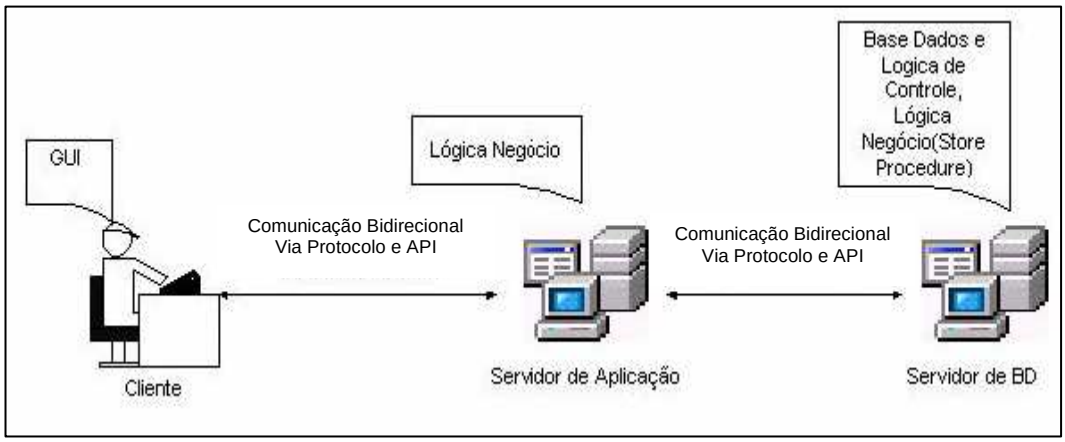

Figura 3. Arquitetura multicamadas
Fonte: ROCHA, C. (2002)

4.2.1 Vantagens da arquitetura multicamadas

No desenvolvimento de novos projetos de softwares com banco de dados, a melhor tecnologia a ser utilizada é a multicamadas. A seguir, é apresentada uma série de vantagens de um sistema desenvolvido em multicamadas (RODRIGUES, 2002):

a) os programas funcionam com vários tipos de banco de dados, sendo que o banco de dados serve apenas como repositório de dados, e não possuem regras de negócio (*Triggers*[7] e *Procedures*[8]);

b) as regras de negócio ficam todas elas, sem exceção, no servidor de aplicação, tornando assim o programa cliente e o servidor em dois programas distintos;

c) o programa cliente torna-se muito mais rápido, pois não se conecta ao banco de dados e trabalha apenas com a *interface* e informações na memória, ou seja, quando os dados estão na memória, os mesmos podem ser filtrados e ordenados rapidamente, não sendo necessária a execução de consultas SQL adicionais ao banco de dados;

d) como os dados do cliente estão apenas em memória, os problemas, de queda de energia, responsáveis por corromper alguns bancos de dados, não mais ocorrerão. Para isto, basta tomar-se as providências necessárias apenas com o servidor, sabendo-se que é somente ele quem acessa o banco de dados. Por exemplo: o uso de *nobreak* apenas no servidor;

e) facilidade na distribuição das aplicações. Como as regras são todas no servidor de aplicação, a atualização dos programas é facilitada. Se a nova versão só altera alguma regra de negócio, a atualização é feita apenas no

[7] Eventos disparados ao inserir, modificar ou apagar registros de um banco de dados.
[8] Procedures são blocos de código que podem ser chamados de diferentes locais em um programa, permitindo sua organização e reutilização.

servidor, não comprometendo a estrutura dos clientes. O mesmo serve para alterações que mudam apenas o *layout* de janelas: esta atualização será apenas no programa cliente, não afetando em nada o servidor de aplicação;

f) a equipe de desenvolvimento pode ser dividida em duas ou mais, dependendo do tamanho do projeto, por exemplo: os programadores mais experientes ficam com o servidor de aplicação, e os menos experientes ficam com a camada do cliente, trabalhando somente na *interface* e nas regras de consistências[9];

g) economia com licenças de bancos de dados. Em aplicações multicamadas, apenas o servidor de aplicação se conecta ao banco de dados, porque os programas cliente estão conectados ao servidor de aplicação. Com isto, existe uma grande redução de custos nas implantações de sistemas com bancos de dados de maior valor. Assim, será necessária a aquisição de apenas uma licença, ao invés de uma licença para cada estação;

h) ótima *performance*, e possibilidades de dividir a carga de informações, em mais de um servidor. Caso o sistema esteja com uma base de dados muito grande e perceba-se queda em sua *performance*, existem várias maneiras de resolver o problema. Pode-se instalar dois servidores de aplicação em duas estações diferentes e o servidor do banco de dados em uma outra estação, e dividindo-se a conexão dos usuários. Por exemplo: os usuários do setor de vendas se conectam ao servidor 1, e os usuários do setor financeiro se conectam ao servidor 2, com essa atitude simples

[9] Regras de consistência são as validações de informações que deverão ser verificadas antes dos dados serem enviados para o servidor de aplicação.

consegue-se registros de melhoria considerável da *performance*;

i) conexões do servidor de aplicação ao banco de dados podem ocorrer em bancos separados, ou seja, um mesmo servidor de aplicação pode se conectar a dois bancos de dados diferentes (por exemplo: Oracle e Sybase), não importando o tipo de banco de dados, apenas devem possuir a mesma estrutura. Isso é muito usado quando se faz uma "limpeza" nas informações e o banco é dividido em um antigo (antes da limpeza), e um novo (depois da limpeza). Caso deseja-se acessar as informações anteriores a manutenção, deve-se conectar no banco de dados antigo, caso contrário, a conexão será no banco de dados novo;

4.2.2 Desvantagens da arquitetura multicamadas

As mudanças de tecnologias geralmente são bastante vantajosas por permitirem à empresa fabricante de software manter-se atualizada e competitiva no mercado. Porém, antes da mudança, é de extrema importância conhecer também as desvantagens que podem acompanhar esta troca. A seguir são apresentadas algumas das desvantagens da tecnologia multicamadas (ROCHA, 2002):

a) é difícil encontrar mão-de-obra qualificada devido a esta tecnologia ser nova e poucas empresas adotarem seu uso;

b) investimento em cursos para os membros da equipe do projeto;

c) leva algum tempo até os programadores adaptarem-se com a nova filosofia de desenvolvimento, atrasando, assim, os projetos em andamento;

d) alguns protocolos de conexão funcionam apenas no Windows, que é o caso dos COM, e COM+ da Microsoft.

4.2.3 Protocolos de comunicação em aplicações multicamadas

Conforme Edwards (1999 tradução nossa) de nada adianta desenvolver um projeto de software em multicamadas, se o software não for estruturado, ou seja, orientado a objetos. Devido a isso, antes de começar a falar de alguns protocolos de comunicação em aplicações multicamadas, é extremante importante saber o que são objetos e alguns princípios da técnica de Programação Orientada a Objetos (POO).

Objeto é uma entidade do mundo real que tem uma identidade. Se cada objeto possui sua própria identidade então se estima que dois objetos sejam diferentes, mesmo tendo as mesmas características.

> Embora objetos tenham existência própria no mundo real, em termos de linguagem de programação um objeto necessita de um mecanismo de identificação. Esta identificação de objeto deve ser única, uniforme e independente do conteúdo do objeto (RICARTE, 1996 p.3).

O conceito sobre orientação a objetos deve ser levado muito a sério, pois somente desta forma pode-se obter sucesso no desenvolvimento de aplicações multicamadas. Assim, o desenvolvimento torna-se mais rápido, devido à reutilização dos códigos. A orientação a objetos, possibilita que a aplicação fique mais segura, pois toda a informação é pesquisada por meio de objetos, que sempre buscam informações de um mesmo local, e não hora de um lugar, hora de outro. Pode-se citar diversos exemplos de melhorias no projeto com a utilização desta técnica, juntamente, com multicamadas. Porem, este não é o foco deste trabalho. Buscou-se aqui apenas proporcionar uma breve introdução para o melhor entendimento do que vem a seguir (EDWARDS, 1999 tradução nossa).

Os protocolos são indispensáveis para o desenvolvimento de um software em multicamadas, por isso é muito importante que estejam bem definidos para evitar surpresas futuras quando o desenvolvimento estiver em andamento. Os principais questionamentos que devem ser feitos antes de se definir qual o protocolo que será usado no desenvolvimento do projeto são (RODRIGUES, 2002):

a) O software funcionará em rede?

b) O software será multi-plataforma?

c) O software terá restrições na criação e manipulação de objetos?

Se estes questionamentos forem respondidos será bem provável que o software terá vida longa e uma história de sucesso.

4.2.3.1 *Commom Object Model*

O COM é uma arquitetura de software orientada a objetos que permite a criação de componentes de softwares por diferentes fabricantes com diferentes linguagens de programação. É o principal protocolo de comunicação entre objetos Microsoft, sendo a base de quase todos os outros protocolos. O COM surgiu para substituir o *Dynamic Data Exchange* (DDE) que possuía os métodos da API[10] do Win32 para transferência de dados entre aplicações por meio de *interfaces*[11]. O DDE foi o primeiro passo no desenvolvimento de diversas tecnologias de integração e comunicação, que surgindo pela primeira vez no Windows 3.0 (HÖLTZ, 2000).

> O COM foi desenvolvido em 1993, tendo como sua função e utilização dar suporte a diferentes módulos/partes de softwares que necessitem se comunicarem. O COM é uma especificação de *Interfaces* binárias para comunicação e pode ser desenvolvido em linguagens distintas, como: Delphi, C++, Visual Basic, JAVA, etc (RODRIGUES, 2002 p.17).

[10] Interface de Programação para Aplicação, é um conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades por programas aplicativos.

[11] Não é a Interface visual. Uma Interface é o que define um tipo que compreende métodos virtuais e abstratos.

Na verdade, o COM não se preocupa com a linguagem de programação em que foi desenvolvido o objeto, mesmo porque isso não importa, uma vez que se disponibilizam seus métodos em *Interfaces*, qualquer chamada a estes métodos deve ser exatamente igual. É por meio da *Interface* que se dá suporte a invocação do objeto escrito em qualquer linguagem de programação. Quando um objeto COM é construído, é extremamente necessário saber a fundo como as *Interfaces* funcionam. Porém, o usuário não necessita conhecer os detalhes do desenvolvimento para utilizá-la. Este isolamento de *Interfaces* e implementação é crucial para objetos COM, pois, isolando-a de sua implementação, é possível construir componentes que podem ser substituídos e reutilizados, isso multiplica a utilidade do objeto. O COM pode ser escrito em forma de DLL[12] ou EXE (executável), sendo que, se for uma DLL, o mesmo é chamado de *In-process*, e se for um EXE e chamado de *Out-of-Process* (RODRIGUES, 2002):

a) ***In-Process*** – possui todas as características de uma DLL, o mesmo é executado na mesma área de memória do processo chamador;
b) ***Out-of-Process*** – é um executável, o qual não faz parte da mesma área de memória do processo chamador.

Na Figura 12 será demonstrado as duas funções (fp_GetSoftwares e fp_GetHardwares) que foram criadas no projeto para serem disponibilizadas por meio de *Interfaces*.

4.2.3.2 *Object Linking and Embedding*

A tecnologia OLE foi desenvolvida pela Microsoft por volta de 1990, para suprir a necessidade de se integrar diferentes aplicações dentro da plataforma Windows,

[12] Uma DLL é uma biblioteca de ligação dinâmica contendo códigos ou dados, que podem ser compartilhados por diferentes programas aplicativos durante sua execução.

de forma a solucionar os problemas de desempenho e confiabilidade do até então utilizado padrão DDE. O OLE é mais flexível, eficiente e mais robusto do que o DDE, mas não possui desempenho necessário para automação (LEANDER, 2000 tradução nossa).

OLE é a incorporação de objetos servidores (*Word, Excel,* entre outros), que aceitam ser manipulados por um controlador (*OLE Automation*). A diferença entre o COM e o OLE é apenas a forma como esses servidores se comunicam. Estes dois protocolos (OLE e COM) muitas vezes se confundem por terem características muito parecidas. A principal diferença entre os dois é (RODRIGUES, 2002):

a) COM é a especificação, é o padrão!

b) OLE é a ligação entre objetos, a tecnologia, a implementação.

Existe uma dificuldade em se trabalhar com OLE, pois seus objetos são muito carregados e utilizá-lo em aplicações proprietárias, torna o processo muito lento. Ao contrário da utilização do COM que a comunicação fica mais rápida (FERRAZ JÚNIOR, 2002).

4.2.3.3 *Distributed Component Model*

O DCOM é uma tecnologia que foi desenvolvida pela Microsoft que possibilita distribuir objetos pela rede. Este protocolo possibilita que componentes de softwares se comuniquem por meio de uma rede de maneira segura, confiável e eficiente. O DCOM tem suas raízes baseadas no COM, OLE e *ActiveX*. Pode-se afirmar que o DCOM é uma extensão do COM para ambientes distribuídos. Deve-se considerar COM e DCOM como sendo uma única tecnologia. Assim, se o objeto necessitar usar somente COM, o mesmo será chamado. Todavia, se o objeto não estiver na máquina

local, o DCOM será acionado, este processo será conhecido apenas em tempo de execução. Possuem como protocolo de comunicação, *Distributed Computing Environment* (DCE), *Distributed Computing Environment-Remote Procedure Call* (DCE-RPC) e *Local Procedure Call* (LPC) (CAMELO, 2002).

DCOM foi lançado em 1995 e, contrariando o que muita gente pensa, não é exclusiva de Ambientes Microsoft, pois o mesmo pode se executado em ambientes Unix e Macintosh (POTTS; KOPACK, 2003).

4.2.3.4 *Component Object Model Plus*

Como afirma Camelo (2002) para o bom entendimento deste protocolo é importante conhecer o Microsoft *Transaction Server* (MTS), que é o servidor de transações da Microsoft. Na realidade os objetos MTS são objetos COM, mas com suporte a transações para banco de dados. São uma extensão do COM, mas também possuem vários recursos do DCOM.

Como já se sabe o que é o DCOM e MTS, fica mais claro entender o que é o COM+. Assim como MTS é a evolução do DCOM, COM+ é a evolução do MTS, com isso, pode-se afirmar que o COM+ é o protocolo que contém todas as características do DCOM e do MTS juntos. Porém, ainda, fornecem outros recursos bem interessantes como os descritos a seguir (RODRIGUES, 2002):

a) ***pooling***: é a capacidade de um mesmo objeto manter instâncias, atendendo vários clientes ao mesmo tempo. É desta forma que o COM+ é considerado escalonável. É o COM+ que gerencia como a instância usará a memória e é também o responsável pela liberação do objeto. Ele libera a memória a partir do último cliente que esteja fazendo referência

ao mesmo. Quando o cliente faz uma requisição de um método, o COM+ procura por um objeto no *POOL*, se encontrar utiliza o próprio, caso não encontre é instanciado outro objeto imediatamente;

b) **escalabilidade**: pode-se aumentar drasticamente o número de conexões concorrentes ao objeto, pois o COM+ tende a não perder o seu desempenho. Isto se deve ao mecanismo do *Pooling*, explicado anteriormente;

c) **gerenciamento**: é o controle de gerenciamento de objetos do COM+, onde pode-se instalar, exportar e administrar objetos COM+. O *Component Services* mostrado na Figura 4 pode ser encontrado no Windows XP, 2000/2003 e pode ser chamado de “A Casa com COM+”. O COM+ só pode ser instalado em versões do Windows que dão suporte a serviços, e com o MTS já instalado, já os clientes pode ser instalados sem que seja necessários o suporte a serviços;

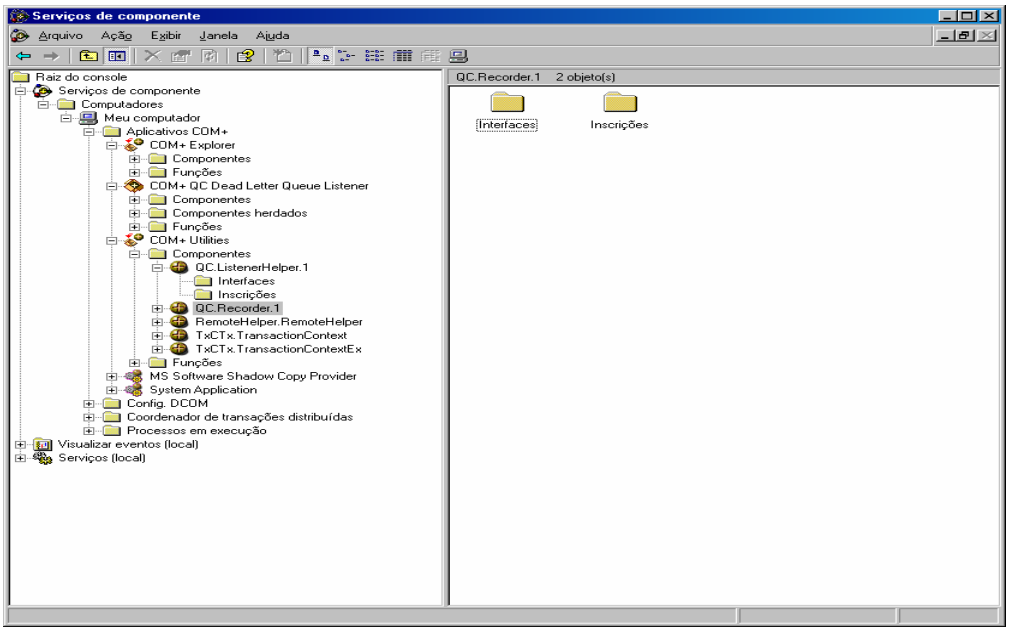

Figura 4. *Components Services* – “A casa do COM+”
Fonte: RODRIGUES, A. (2002)

d) **transações**: é o mecanismo responsável pelas transações no COM+ é o MSDTC, que também se encontra no *Component Services*. Com o COM+ é possível realizar transações em objetos e em banco de dados. As transações de objetos permitem que vários objetos diferentes sejam colocados no contexto de uma única transação, mesmo que esse objeto não faça acesso ao banco de dados.

4.2.3.5 Segurança em DCOM/COM+

Como afirma Rodrigues (2002), a segurança é de fundamental importância nos padrões e protocolos da Microsoft. Pode se afirmar que, nestes protocolos, existe segurança na chamada dos objetos, na criação de um objeto novo e na troca de mensagens. Estes parâmetros de segurança, estão relacionados aos seus usuários ou grupos, isto é, se o usuário tem acesso a determinado objeto ou não.

O COM+ esconde os procedimentos de configuração e segurança, sendo assim, não se faz necessário codificações específicas de segurança. A transparência é conseguida com os seguintes parâmetros (RODRIGUES, 2002):

a) ***anonymous**:* não se permite ao objeto conseguir a identidade do cliente. Todos os objetos que acessam o objeto ficam desconhecidos para o mesmo. Os dados dos clientes são preservados, tornando assim uma opção segura. Por outro lado, é menos seguro para o objeto, pois ele fica obrigado a atender às requisições feitas de qualquer cliente, até mesmo, os desconhecidos;

b) ***identify**:* o objeto descobre apenas o nome do cliente, isto é, quem está requisitando tal informação. É um bom nível de segurança para o cliente,

pois o objeto não pode executar operações com dois clientes iguais ao mesmo tempo, isto pode ser chamado de credenciais para uma melhor compreensão;

c) ***impersonate:*** o objeto sabe e personifica o cliente para execução. Este modo não é seguro para o cliente, pois permite ao objeto usar as credenciais de segurança do cliente para executar operações na máquina onde o objeto está sendo executado.

A Figura 5 mostra as opções de segurança encontradas no objeto COM+.

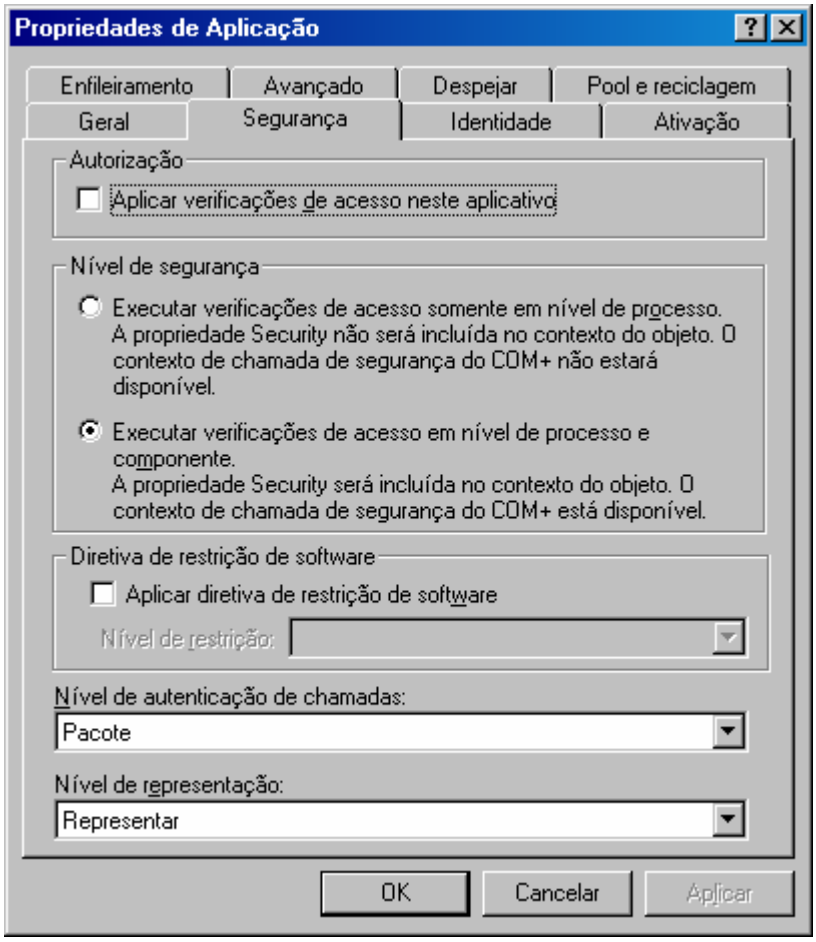

Figura 5. Opções de segurança encontradas no COM+
Fonte: RODRIGUES, A. (2002)

4.2.3.6 *Common Object Request Broker Architecture*

CORBA é um padrão desenvolvido e proposto pela *Object Management Group* (OMG), órgão composto por diversas empresas de informática como a *Sun*, *Cannon*, *IBM*, entre outras. Essa especificação foi projetada no início da década de 1990 com o intuito de fornecer um mecanismo para construir aplicações cliente/servidor em ambientes heterogêneos (POTTS; KOPACK, 2003).

O CORBA propõe a independência de plataforma, de hardware e de linguagem de programação. Com a finalidade de fechar a brecha entre linguagens distintas, uma IDL é usada para detalhar a estrutura de todos os objetos. Os desenvolvedores, então, pegam a IDL e a executam por meio de algum tipo de gerador de código para a linguagem usada em cada lado da transação, a fim de obter a estrutura correspondente específicas da linguagem. Fazendo isso, é possível escrever um cliente Visual Basic que se comunique com um servidor Java, usando a CORBA como a camada de comunicação (RODRIGUES, 2002).

Uma IDL é uma linguagem que possibilita especificar *interfaces* de forma independente da linguagem de programação na qual a especificação é implementada. CORBA determina uma série de mapeamentos padronizados entre IDL e outras linguagens, tais como C, *COBOL*, *Delphi* e *Java* (RICARTE, 1996).

Rodrigues (2002) ainda afirma que o CORBA é concorrente direto do COM/COM+, com todos os recursos do COM+, mas com a vantagem de ser independente de plataforma.

4.3 QUANDO UTILIZAR MULTICAMADAS

A tecnologia multicamadas está em constante crescimento, mas a tecnologia cliente/servidor está muito longe de deixar de ser implementada, pois existem muitas empresas que ainda a utilizam para desenvolvimento de softwares. Surge então a pergunta: “Qual o modelo utilizar?” Esta mesma questão foi apresentada ao *Gartner Group´s*, onde foi desenvolvida uma pesquisa. A Figura 6 mostra que para pequenos projetos, 2-camadas são mais fáceis de desenvolver do que multicamadas. No entanto, à medida que as aplicações tornam-se mais complexas, as aplicações 2-camadas tornam-se exponencialmente mais complexas para desenvolver (EDWARDS, 1999 tradução nossa).

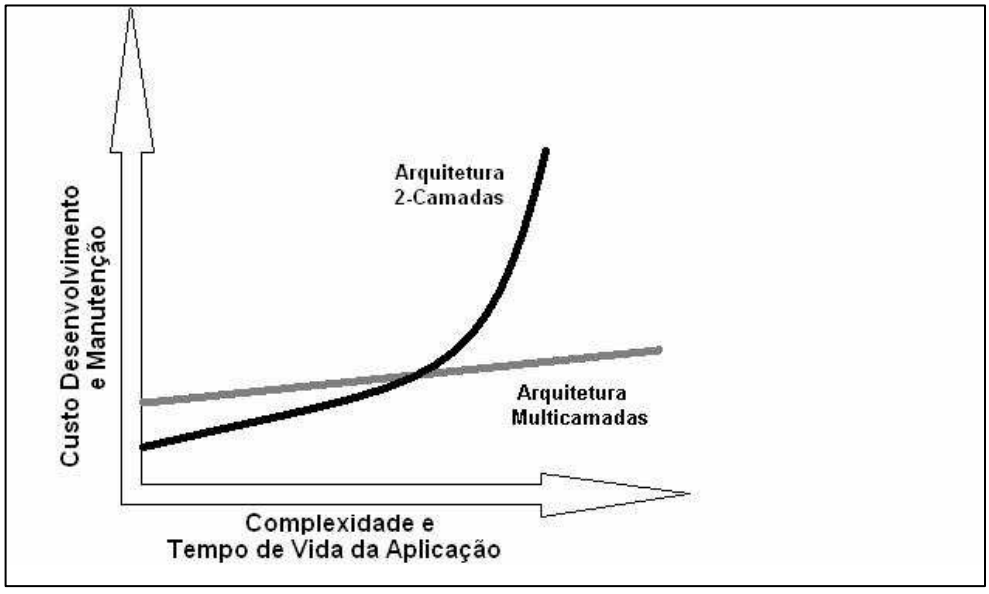

Figura 6. Comparação de 2-camadas e multicamadas
Fonte: EDWARDS, J. (1999).

Edwards (1999 tradução nossa) ainda faz o questionamento. “Onde está o ponto de cruzamento?”. De acordo com o *Gartner Group´s*, deve-se usar multicamadas, se a aplicação possuir algumas das seguintes características:

a) muitos serviços de classes ou aplicação (mais que 50);

b) aplicações programadas em diferentes linguagens ou escritas por diferentes organizações;

c) duas ou mais origens de dados, como dois bancos de dados na mesma aplicação;

d) uma aplicação que tenha vida maior que três anos, especialmente se esperar muitas modificações e implementações;

e) um alto volume de movimentações, mais que 50000 (50 mil) transações por dia ou mais de 300 usuários simultâneos no mesmo sistema, acessando a mesma base de dados;

f) significativa comunicação entre aplicações, incluindo comunicações entre empresas como *Electronic Data Interchange* (EDI).

Edwards (1999 tradução nossa) conclui que utilizar a tecnologia multicamadas é uma aposta segura. No mundo de aplicações Intranet e Internet, a tecnologia multicamadas é a probabilidade favorita. Possibilita iniciar pequeno em escala e função, até a aplicação crescer a proporções maiores. Também, possibilita a criação de um catálogo de componentes personalizados que é rapidamente montado para produzir novas aplicações. Pode-se publicar as *interfaces* para estes componentes, e usar ferramentas GUI[13] para acessá-las. Portanto, programadores departamentais podem incluí-los para suas aplicações. Pode se encapsular e reusar aplicações.

No capítulo a seguir, descreve-se sobre o sistema desenvolvido (Inspector), os componentes utilizados para a sua construção. Também, sobre as suas funcionalidades e os resultados obtidos.

[13] É um mecanismo de interação entre usuário e sistema de computador baseado em símbolos visuais, como ícones, menus e janelas.

5 TRABALHO DESENVOLVIDO - INSPECTOR

Foi elaborado um estudo sobre a arquitetura multicamadas com a finalidade de demonstrar a utilização desta tecnologia, utilizando a linguagem de programação Delphi 7.0. Com esse intuito, foi desenvolvido um sistema para o auxílio no inventário dos softwares e hardwares instalados nos computadores da rede. Este sistema foi desenvolvido para funcionar em qualquer versão da plataforma Windows superior a 95. A Figura 7 mostra a estrutura do sistema.

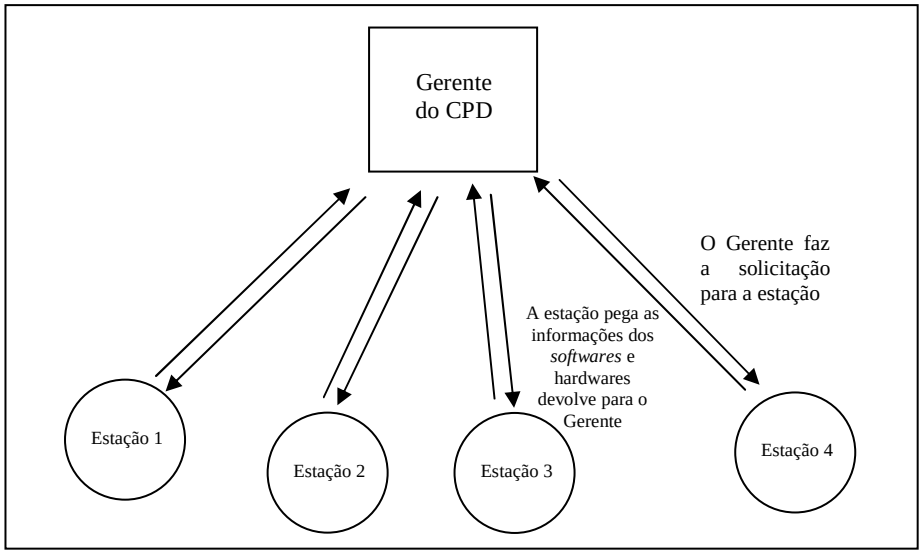

Figura 7. Arquitetura do sistema

As informações de quais softwares estão instalados no computador é adquirida por meio do Registro do Windows, onde é feito uma varredura na chave HKEY_LOCAL_ MACHINE que é a chave onde todos os softwares guardam suas informações básicas. Essas informações são mostradas na tela onde o usuário que está operando o sistema poderá optar por gravar as informações no banco de dados ou simplesmente visualizá-las e depois descartá-las, sendo que, se optar por gravar, o sistema começa a criar um histórico dos softwares instalados, e que, posteriormente, poderão ser realizadas consultas por meio de relatórios. Para descobrir quais os

hardwares que o computador possui, também é a acessado a chave HKEY_LOCAL_ MACHINE, porém não são todos os periféricos que são armazenados nessa chave.

O SGDB utilizado no desenvolvimento do sistema foi o Firebird 2.0.1, por ser gratuito, mas poderia ser qualquer outro, pois que com a utilização da tecnologia multicamada não faz diferença, pois trata o banco apenas como repositório de dados. Para permitir que a aplicação possa utilizar vários tipos de bancos não são utilizados alguns recursos que os bancos disponibilizam, tais como: *Triggers e procedures.*

A modelagem dos dados está demonstrada no modelo E-R[14], da Figura 8. Para realizar a modelagem dos dados foi utilizado o software PowerDesigner 9.5.

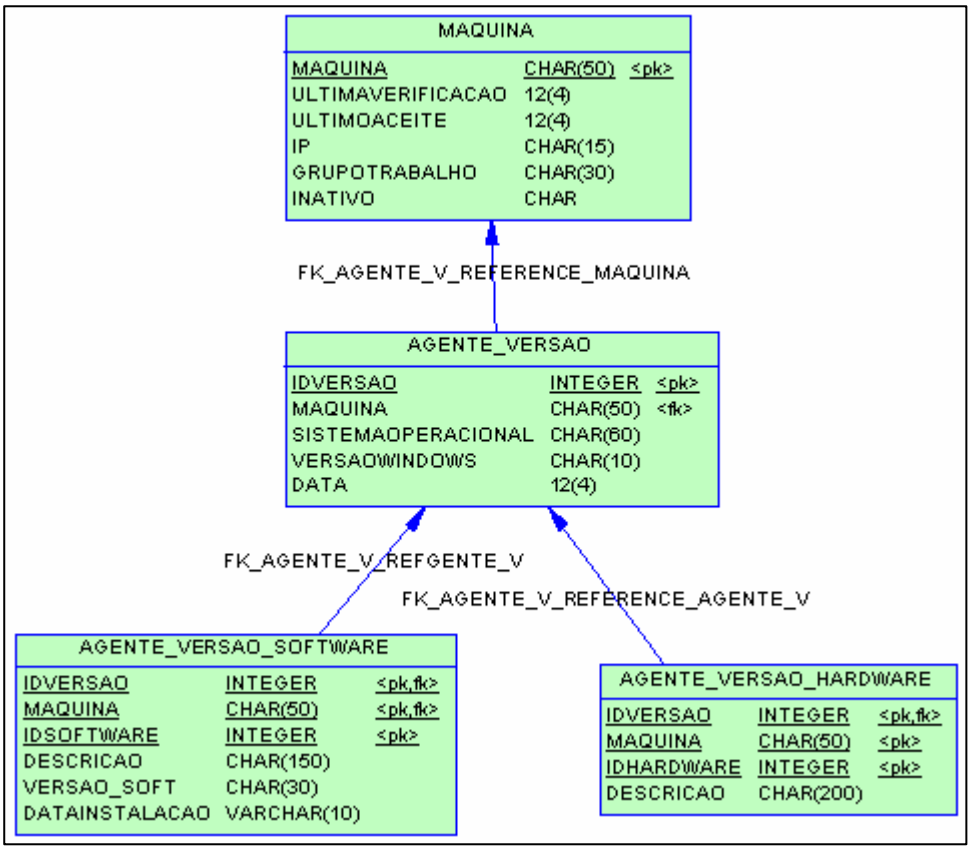

Figura 8. Modelo E-R

[14] Modelo Entidade-Relacionamento, tipo de modelagem de dados usados por banco de dados relacionais.

5.1 COMPONENTES UTILIZADOS

No desenvolvimento do sistema foi utilizada a ferramenta de desenvolvimento *Delphi 7.0*, onde se utilizou a biblioteca *DataSnap.* Segundo Rodrigues (2002), é esta biblioteca do *Borland Delphi 7.0* que oferece todos os recursos de comunicação entre aplicações cliente e servidora. Nela, estão disponíveis os componentes necessários para trabalhar com a tecnologia multicamadas. Estes componentes possuem os mais diversos protocolos de comunicação, onde podem ser citados o TCP/IP, COM, HTTP, entre outros.

É importante salientar que ao se trabalhar com aplicações multicamadas é necessário ter um servidor de *socket* aberto na máquina servidora, para que a comunicação com o cliente seja realizada por meio da porta que é aberta pelo *socket server*. A *Borland* disponibiliza este programa juntamente com a instalação do *Borland Delphi 7.0*. O seu nome é ScktSrv.exe, embora nada implica que seja desenvolvido internamente um socket server próprio. A Figura 9 mostra a biblioteca *DataSnap* na linguagem de programação *Borland Delphi 7.0*.

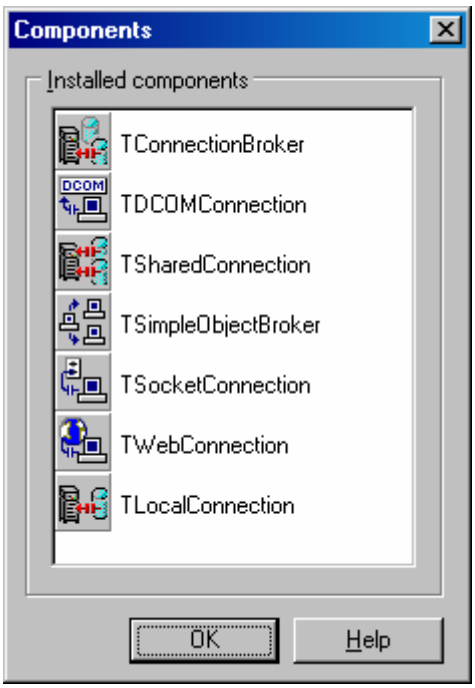

Figura 9. Lista de componentes da biblioteca DataSnap.
Fonte: *BORLAND* (2002)

5.1.1 *TsocketConnection*

É o componente usado para se conectar a um servidor de aplicação, usando o protocolo TCP/IP. Para a conexão ser realizada com sucesso o programa SckSrv.exe deve estar sendo executado na máquina servidora. Suas principais propriedades são (CANTÚ, 2003):

a) ***connected***: indica se a comunicação com o servidor de aplicação está ativa;

b) ***address***: indica o endereço IP da máquina servidora. Deve ser preenchido para a conexão ser realizada com sucesso;

c) ***login prompt***: mostra um diálogo de login remoto para a conexão com o servidor de aplicação;

d) ***port***: especifica a porta de comunicação entre o aplicativo cliente e o aplicativo servidor;

e) ***servername***: especifica o nome do aplicativo servidor com o qual o cliente irá se conectar.

5.1.2 *TSimpleObjectBroker*

Componente responsável por colocar um servidor de aplicação no ar quando algum da sua lista de servidores for desconectado. Ele não faz o *load balance*, o que faz é a substituição de um servidor por outro, caso algum dos servidores fique fora do ar CANTÚ, 2003).

Este componente não deve ser usado com o CORBA, pois este já possui o seu mecanismo de *load balance*.

O componente *TSimpleObjectBroker* também, é utilizado para fazer a ligação estática do seu cliente com o servidor de aplicação. Tem como propriedades principais, as seguintes:

a) ***loadBalanced***: indica se o componente colocará outro servidor no ar assim que um servidor de aplicação cair;

b) ***servers***: é a lista dos servidores de aplicação. Pode ser o IP ou o nome do *host*.

5.1.3 *TConnectionBroker*

Esse componente tem a função de abstrair a camada de conexão dos *ClientDataset* e a chamada dos métodos do servidor de aplicação. Isto é, com o uso do *ConnectionBroker*, pode-se iniciar a implementação do servidor de aplicação em COM+ e, se houver a necessidade de se trocar o tipo de protocolo para *Sockets* ou CORBA,

pode-se fazer de uma forma menos traumática. O que acontece é que os *ClientDataset* não estão ligados diretamente ao *DCOMConnection*, e sim, ao *ConnectionBroker*. As principais propriedades deste componente são (RODRIGUES, 2002):

a) ***connected***: indica se a comunicação com o servidor de aplicação está ativa;
b) ***connection***: indica a qual componente de conexão ele está conectado, ou se irá se conectar, podendo ser um SocketConnection ou um *SharedConnection*. É desta forma, que se faz a abstração da conexão e a chamada dos métodos;
c) ***loginprompt***: especifica se o login de conexão padrão aparecerá antes da conexão ser estabilizada.

5.1.4 *TSharedConnection*

Até a versão 5 do *Borland Delphi*, não existia à opção de criar vários *Remote Data Modules* (RDM) no servidor de aplicação. Todo o código fonte era localizado dentro de apenas um RDM. Com a chegada do *Borland Delphi* 6, e continuado em sua versão 7, surgiu um novo componente onde permite realizar-se a criação de vários RDM no servidor, facilitando assim a visualização do código fonte, e dando maior organização ao projeto do servidor (RODRIGUES, 2002).

O uso deste componente requer um conhecimento um pouco mais avançado, visto que suas ligações não são apenas com propriedades (não são feitas ligações através das propriedades desse componente), pois todas as ligações são feitas via *Interface*, *Class Factory* e com o velho *USES* entre unidades do *Delphi*.

Suas principais propriedades são:

a) ***childname*:** é o RDM ao qual o componente *SharedConnection* pertence (vai comunicar-se). Ao tentar selecionar esta propriedade, o *Delphi* monta uma lista com todos os *CahildRDMs* criados a serem escolhidos;

b) ***parentconnection*:** especifica a conexão do cliente com o servidor de aplicação. Pode ser um *TDCOMConnection* ou um *TSocketConnection*.

5.1.5 *TLocalConnection*

É o componente utilizado para trabalhar na mesma aplicação e conexão, com a administração de vários *Datasets*. Também pode, futuramente, migrar sua aplicação para uma de multicamadas. É como se ele simulasse as camadas (CANTÚ, 2003).

5.2 IMPLEMENTAÇÃO

Primeiramente conheceremos como se cria um servidor de aplicação, porém antes de mostrar como criar, deve-se saber que até a versão 6.0 do *Borland Delphi* era pago um valor por licença de servidor de aplicação, o *DatSnap*, a partir da versão 7.0 passou a ser totalmente gratuito. Em multicamadas é desenvolvido um servidor de aplicação, com o protocolo de comunicação escolhido pelo gerente do projeto. No *Delphi* essa opção está disponível no menu *File/New/other...*, como mostra a Figura 10.

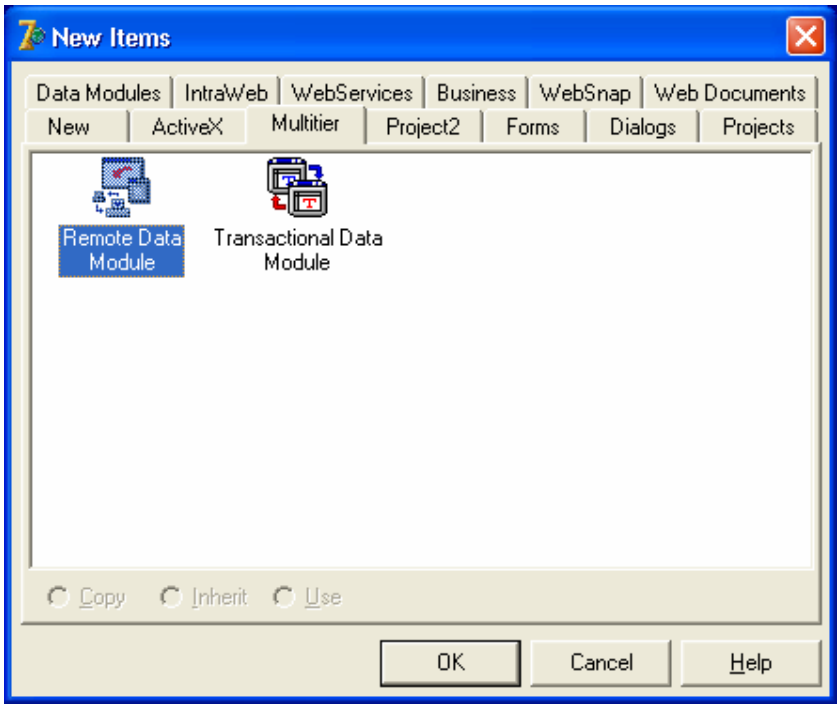

Figura 10. Criação do *Remote Data Module.*
Fonte: *BORLAND* (2002)

Na guia Multitier (multicamadas), conforme mostrado na Figura 10, há duas opções para criar data modules com suporte a multicamadas, no primeiro item, Remote Data Module, é a opção do protocolo de comunicação, utilizando COM, e no segundo item tem-se a opção da utilização de COM+. Selecionando a opção COM e clicando em OK, a partir deste momento o servidor de aplicação começa a ser desenvolvido. Nesse projeto, optou-se por utilizar o COM, visto que já vem instalado em todas as versões do Windows e assim, não necessitando de instalação.

É importante saber que as aplicações multicamadas que utilizam *remotes data modules*, além dos arquivos básicos de projetos do Delphi, têm acrescentado mais um arquivo com a extensão *.tbl, onde ficam armazenados os códigos das *interfaces*, do servidor de aplicação. A Figura 11 mostra um exemplo deste arquivo no Delphi.

```
IInspectRDMDisp = dispinterface
  ['{4D9EFAEA-FB9A-4F06-BA36-5BE6F011A355}']
  function fp_GetSoftwares: WideString; dispid 301;
  function fp_GetHardwares: {??PWideString1}OleVariant; dispid 302;
  function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors: Integer;
               out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; dispid 20000000;
  function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
                         Options: Integer; const CommandText: WideString; var Params: OleVariant;
                         var OwnerData: OleVariant): OleVariant; dispid 20000001;
  function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
                          dispid 20000002;
  function AS_GetProviderNames: OleVariant; dispid 20000003;
  function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
                        dispid 20000004;
  function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
                         var OwnerData: OleVariant): OleVariant; dispid 20000005;
  procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString;
                       var Params: OleVariant; var OwnerData: OleVariant); dispid 20000006;
  end;
CoInspectRDM = class
  class function Create: IInspectRDM;
  class function CreateRemote(const MachineName: string): IInspectRDM;
  end;
implementation
uses ComObj;
class function CoInspectRDM.Create: IInspectRDM;
begin
  Result := CreateComObject(CLASS_InspectRDM) as IInspectRDM;
end;
class function CoInspectRDM.CreateRemote(const MachineName: string): IInspectRDM;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_InspectRDM) as IInspectRDM;
end;
```

Figura 11. Exemplo do arquivo *.TBL

Geralmente o usuário não manipula este arquivo como faz com os códigos fontes. Para manipular este arquivo, ou seja, criar novas *procedures* ou funções, ou até mesmo novas *interfaces*, o *Delphi* disponibiliza um utilitário para o auxílio na manutenção, que é o *Type Library*, acessado através do menu *View*, *opção Type Library*, conforme mostra a Figura 12.

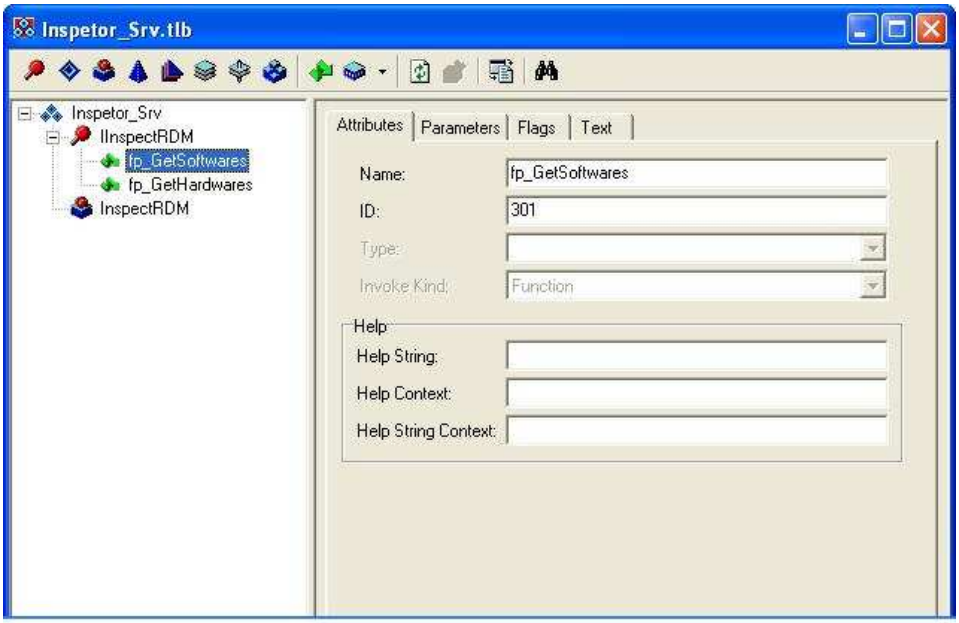

Figura 12. Utilitário *Type Library* para manipulação de *interfaces*
Fonte: *BORLAND* (2002)

No sistema desenvolvido, nota-se que a *interface* Inspetor_Srv, possui dois métodos disponível, conforme mostrado na Figura 12: fp_GetSoftwares e fp_GetHardwares que são rotinas publicadas através da *interface*, e podem ser invocadas pelos clientes que estão conectados.

Para pesquisar os softwares que estão instalados no computador é usada uma função que faz uma busca na chave HKEY_LOCAL_MACHINE do registro do Windows conforme mostra a Figura 13 e a Figura 14.

```
if verInfo.dwPlatformId = VER_PLATFORM_WIN32_NT then
Begin
   SOplataforma := IniFile.Create(GetEnvironmentVariable('windir')+
                                  '\system32\prodspec.ini');
   loLista.Add('SO=' + SOPlataforma.ReadString('Product
               Specification', 'Product', ''));
   loLista.Add('VERSAO=' + IntToStr(verInfo.dwMajorVersion) +
                     '.' + IntToStr(verInfo.dwMinorVersion) + '.' +
                           IntToStr(verInfo.dwBuildNumber));
end
Else
Begin
   Reg.OpenKey('\Software\Microsoft\Windows\CurrentVersion',False);
   loLista.Add('SO=' + Reg.ReadString(Registro98,'Version', ''));
   loLista.Add('VERSAO=' + Reg.ReadString(Registro98,
                           'versionNumber', ''));
end;
```

Figura 13. Descobre qual é o sistema operacional e qual versão

```
Reg.OpenKey('\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall',
False);
Reg.GetKeyNames(ListaChaves);
for i := 0 to ListaChaves.Count -1 do
begin
   if reg.ReadString(ListaChaves[i], 'displayname', '') <> '' then
   begin
     loLista.Add(Reg.ReadString(ListaChaves[i], 'DisplayName', '));
     loLista.Add(Reg.ReadString(ListaChaves[i], 'DisplayVersion',
''));
     loLista.Add(Reg.ReadString(ListaChaves[i], 'InstallDate',
''));
   end;
end;
```

Figura 14. Função para buscar os Softwares instalados

5.3 FUNCIONAMENTO DO SISTEMA

O sistema foi desenvolvido para o funcionamento em ambiente Windows e possui uma *interface* amigável com o usuário. A tela principal possui quatro guias: Máquinas, Versões, Softwares e Hardwares, três menus: Arquivo, Relatórios e Ajuda e a opção Sem Rede.

a) **guia máquinas**: é onde serão listados todos os computadores que estão conectados na rede local da corporação como mostra a Figura 15;

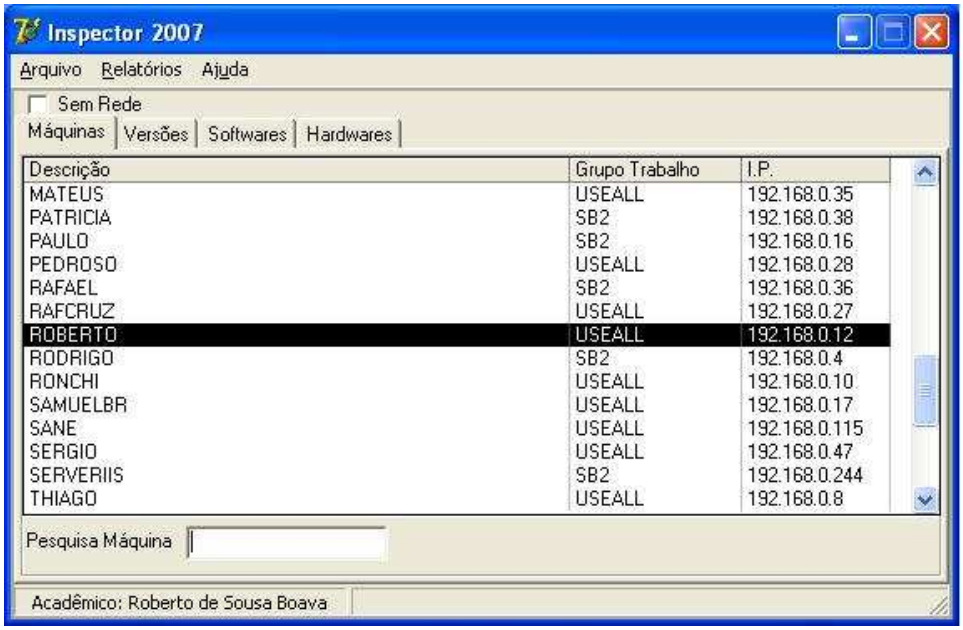

Figura 15. Tela Principal do sistema.

b) **guia versões**: nesta guia são apresentadas as versões que foram gravadas do computador selecionado na guia máquinas, a cada nova verificação o sistema gravará uma nova versão Figura 16;

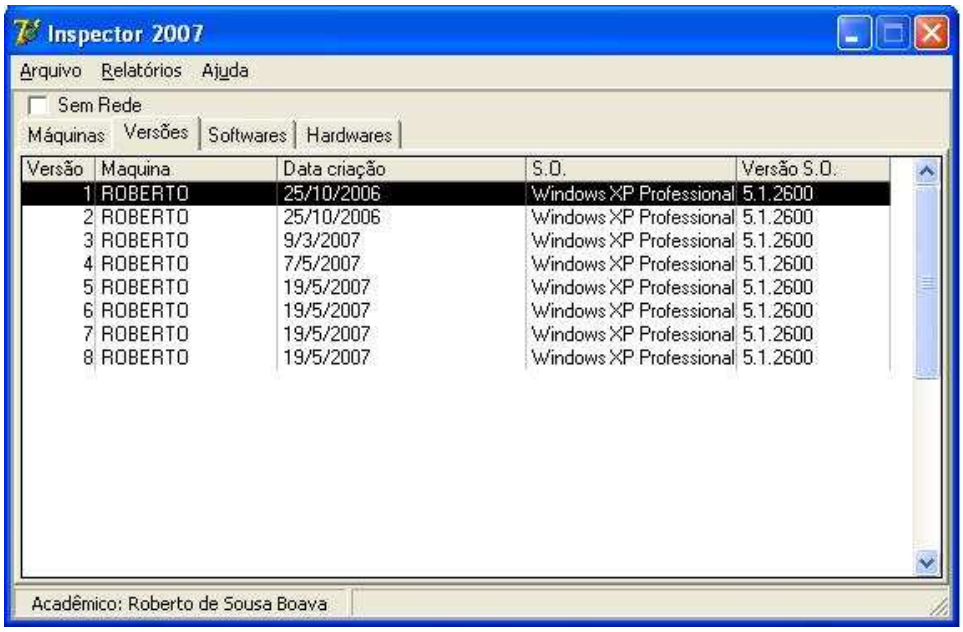

Figura 16. Versões que foram gravadas para o computador Roberto.

c) **guia Software***:* é onde serão listadas as informações dos softwares do computador que foi selecionado na guia Versões, como mostra a Figura 17. Na Figura 15, o computador Roberto está selecionado, na Figura 16 está sendo listado às versões que estão gravadas para este computador e na guia Softwares são listados os programas que estão instalados no computador que está selecionado na guia Máquinas e, conseqüentemente, na guia Versões, no caso o Roberto;

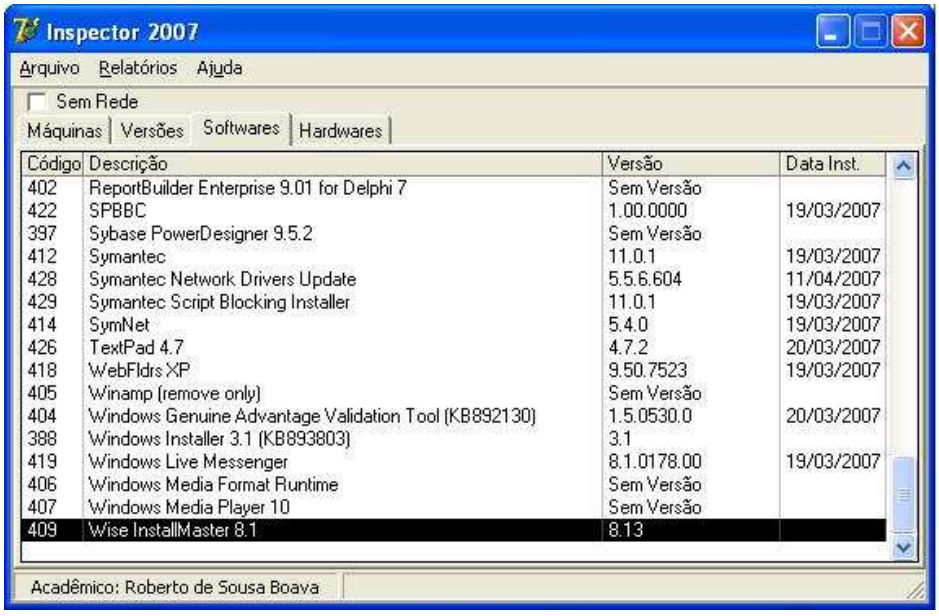

Figura 17. Softwares que estão instalados no computador Roberto na versão 1.

d) **guia hardwares**: onde serão listadas as informações dos hardwares do computador que foi selecionado na guia Versões, como mostra a Figura 18. Na Figura 15, o computador Roberto está selecionado, na Figura 16 estão sendo listado às versões que estão gravadas para este computador e na guia hardwares são listados os componentes que estão instalados no computador que está selecionado na guia Máquinas e, conseqüentemente, na guia Versões, no caso o Roberto.

Figura 18. Hardwares que estão instalados no computador Roberto na versão 1.

e) **sem rede**: o sistema foi projetado para funcionar em rede, essa opção foi desenvolvida apenas para quando precisar testá-lo em um computador que não estiver conectado a nenhuma rede.

O menu Arquivo é constituído pelas funções: checar rede agora, atualizar lista, verificar agora, inativar computador e fechar, como demonstrado na Figura 19.

a) **checar rede agora**: quando acionada, essa função fará uma verificação na rede local em busca de novos computadores conectados a rede, se detectar alguma máquina nova o sistema mostrará na guia máquinas e gravará este novo computador, automaticamente, no banco de dados,

caso contrário, ou seja, não encontrar nenhum computador novo, emitirá uma mensagem informando que não foram encontradas máquinas novas;

b) **atualizar lista**: quando o sistema é aberto a guia máquinas estará vazia. Quando essa função for acionada o sistema fará uma consulta no banco de dados e preencherá esta guia com todas as máquinas que já estão gravadas no banco de dados e que ainda estão ativas;

c) **verificar agora**: quando acionada o sistema encontrará na rede o computador selecionado na guia máquinas e acionará o servidor de aplicação. O servidor de aplicação fará uma busca no registro do Windows para adquirir as informações sobre os softwares e hardwares instalados no computador. O sistema somente permitirá acionar essa função se algum computador da guia máquinas estiver selecionado, quando não existir nenhum selecionado ou a lista estiver vazia, será emitida uma mensagem informando que nenhum computador foi selecionado;

d) **inativar computador**: essa função servirá para inativar um computador que não pertence mais a rede da corporação. Por exemplo, um funcionário é demitido ou decide sair da empresa, então a outra pessoa que assumirá o computador provavelmente trocará o nome da máquina na rede, então se deve inativar o nome antigo e a partir, desse momento, inicia-se um novo ciclo para essa máquina;

e) **fechar**: apenas fecha o aplicativo.

Figura 19. Menu Arquivo

O menu Relatórios é composto pelos seguintes relatórios: Histórico de computador e Comparativo de versões, como mostra a Figura 20.

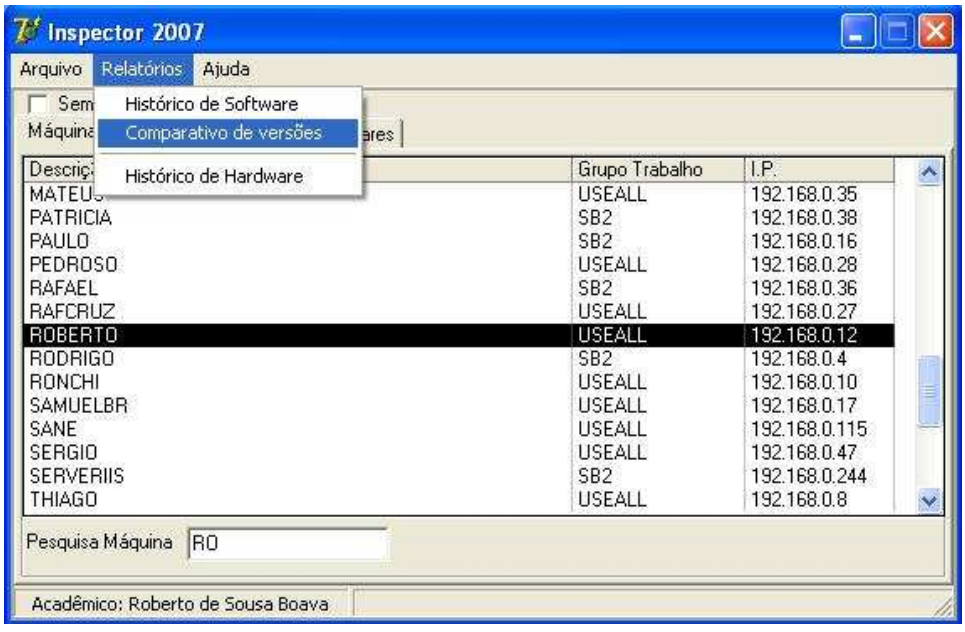

Figura 20. Menu Relatórios

a) **histórico de computador**: é apresentado na tela do computador para ser visualizado, um relatório com todos os softwares instalados agrupados por versão. Este relatório possui como filtros: período, onde se pode listar os softwares instalados no computador em um determinado espaço de

tempo. Versão onde se pode listar os softwares instalados em um intervalo, como por exemplo: da versão 5 até a 8. Por último, o filtro por máquina onde pode ser escolhido para listar os dados de apenas 1 computador, caso seja digitado o seu nome no filtro ou optar por listar os dados de todos os computadores se o filtro ficar em branco;

b) **comparativo de versões**: nesse relatório pode-se comparar os softwares instalados de uma versão para outra, ou seja, o que tinha instalado no computador ROBERTO na versão 3, que não tem mais na verão 4;

c) **Histórico de Hardwares**: é apresentado na tela do computador para ser visualizado, um relatório com os hardwares instalados onde estão agrupados por versão. Este relatório possui como filtros: período, onde se pode listar os periféricos instalados no computador em um determinado espaço de tempo. Versão onde se pode listar os hardwares instalados em um intervalo, como por exemplo: da versão 5 até a 8. Por último, o filtro por máquina onde pode ser escolhido para listar os dados de apenas 1 computador, caso seja digitado o seu nome no filtro ou optar por listar os dados de todos os computadores se o filtro ficar em branco.

5.4 RESULTADOS OBTIDOS

Os testes foram realizados em uma INTRANET na UNESC e na Empresa USEALL Software Ltda., com os resultados obtidos sendo muito satisfatórios, pois se conseguiu obter o nome de todos os aplicativos instalados e alguns periféricos instalados nos computadores em uma rede local.

Devido ao sistema permitir que o usuário grave todas as informações que ele captura em um banco de dados, fica muito prático e rápido consultar informações de softwares e hardwares das máquinas conectadas na rede da corporação, como por exemplo, comparar quais softwares existia instalado na versão 3 e que não mais estavam instalados na versão 5, emitir uma listagem de quais softwares existiam instalados em comutados no mês passado.

No decorrer do desenvolvimento deste trabalho encontraram-se algumas dificuldades, pois não existem muitos livros sobre multicamadas com a linguagem de programação Delphi.

CONCLUSÃO

O desenvolvimento de aplicações, utilizando a arquitetura multicamadas é uma realidade que vem caminhando para se tornar um padrão de desenvolvimento. Isto acontece devido ao fato desta arquitetura ser considerada uma tecnologia bem planejada, pois possui diversas vantagens comparadas a outros modelos. O desenvolvimento utilizando essa arquitetura possibilita dividir a equipe de desenvolvimento em várias equipes menores. Por exemplo, quem gosta de trabalhar com a parte visual do sistema pode ficar em uma equipe. Sendo assim, não precisará se preocupar com as validações que serão necessárias, deixando essa parte com a equipe que irá trabalhar com as regras de negócio. Pode-se criar também uma equipe para trabalhar apenas com o banco de dados. Desta forma pode-se dividir a equipe em várias menores sem que uma entre no desenvolvimento da outra.

Devido ao estudo feito no decorrer deste trabalho, pode-se concluir que para o desenvolvimento de aplicações, utilizando banco de dados que precisam de comunicação entre objetos, alto desempenho e segurança nas informações a arquitetura multicamadas é a solução que contempla da melhor forma essas necessidades.

Com relação a trabalhos futuros, pode-se listar algumas opções no que diz respeito a continuidade do desenvolvimento do Inspector, como são citados abaixo.

a) desenvolver esse sistema para o funcionamento em outros sistemas operacionais, tais como: Linux, Unix;
b) integrar o inspector a um *Web Services;*
c) desenvolver o inspector para o funcionamento com outros protocolos.

REFERÊNCIAS

ABES. **Manual ABES de Gerenciamento de Software**. 2002. Disponível em: http://www.abes.org.br/templ1.aspx?id=223&sub=223. Acessado em: 21/10/2006.

ALECRIM, **Emerson. Manipulação básica do Registro do Windows**. 2004. Disponível em: < http://www.infowester.com/tutregistrowin.php >. Acessado em: 04/09/06.

BITENCOURT, Jossiane Boyen; SANTOS, Marlise Bock. **Software livre:** Perspectivas teóricas e práticas de utilização na informática educacional. Porto Alegre, RS: Universidade Federal do Rio Grande do Sul, 2004.

BORLAND Delphi Enterprise. 2002.

CAMELO, Dioclécio Moreira. **Web service.** Curso de Especialização de sistemas de informação e aplicações Web, Manaus, 2002.

CANTÚ, Marco. **Dominando o Delphi 7 "A bíblia".** São Paulo: Mackron Books, 2003.

CARDOSO, Nuno Oliveira; ESTEVES, Giovanni Pedro Paz. **Utilizando componentes em aplicações distribuídas em 3 camadas.** Goiânia, GO: Trabalho de Conclusão de Curso em Ciência da Computação, Universidade Federal de Goiás, 2002.

CUNHA, Daniel Pezzi. **Um estudo das estratégias de replicação e reconciliação de banco de dados móveis em um ambiente wireless.** Florianópolis, SC: Dissertação submetida à Universidade Federal de Santa Catarina, 2003.

DAMÁZIO, Eduardo Luiz Rodrigues. **Integração de base de dados utilizando web services: Arquitetura e estudo de caso em programação distribuída.** Criciúma, SC: Bacharel em Ciências da Computação, Universidade do Extremo Sul Catarinense, 2005.

D'este, Gary V. **Protótipo de sistema de auxílio ao gerenciamento de software utilizando Agentes Móveis.** Criciúma, SC: Bacharel em Ciências da Computação, Universidade do Extremo Sul Catarinense, 2003.

EDWARDS, Jeri. **3-Tier cliente/servidor at work.** Revised edition.Wiley Computer Publishing, 1999.

FERRAZ JÚNIOR, Fábio. **Desenvolvimento de um sistema de monitoramento e supervisão para o processo de torneamento.** São Carlos, SP: Dissertação de mestrado, Universidade de São Paulo, 2002.

FERREIRA, Cláudio Luiz. **Framework para gerenciar dados de integração do usuário em ambientes hipermídia de aprendizagem**. Florianópolis, SC: Dissertação de mestrado, Universidade Federal de Santa Catarina, 2003.

HÖLTZ, Rudy Hamilton. **Estudo sobre Transações Distribuídas nas Modernas Tecnologias de Objetos.** Porto Alegre, RS: Mestrado em Engenharia da Computação, Universidade Federal do Rio Grande do Sul, 2000.

LEANDER, Rick. **Building Application Servers**. New York. Cambridge University Press, 2000.

LEI DE DIREITO AUTORAL, LEI Nº 9.610, 1998.

LEI DE PROGRAMA DE COMPUTADOR, LEI Nº. 9.609, 1998.

MICROSOFT. **Descrição do Registro do Microsoft Windows.** 2001. Disponível em <http://support.microsoft.com/kb/256986/PT-BR/>. Acesso em: 22/04/2007.

POTTS, Stephen; KOPACK, Mike. **Aprenda *Web Services* em 24 Horas**. Rio de Janeiro, Campus, 2003.

RICARTE, Ivan Luiz Marques. **Programação orientada a objetos com C ++.** 1996. Disponível em: <http://www.dca.fee.unicamp.br/courses/POOCPP/POOCPP.html>. Acesso em: 05/06/2006.

ROCHA, Carlos André de Sousa. **Análise de desempenho em ambientes cliente/servidor 2-camadas e 3-camadas.** Florianópolis, SC: Mestrado em Ciências da Computação, Universidade Federal de Santa Catarina, 2002.

RODRIGUES, Anderson Haertel. **Sistemas Multicamadas com Delphi DataSnap e DbExpress:** Conceitos, Implementações e Macetes. Florianópolis: Visual Books, 2002.

SANTOS JÚNIOR, José Maria Rodrigues. **Arquitetura de Aplicações.** Aracaju, SE: Universidade Tiradentes, 2003.

SILVA NETTO, Vicente Cassiano. **Questões legais sobre desenvolvimento, Comercialização, direito autoral e uso de Software**. Natal, RN: Curso de ciências da computação, Universidade Federal do Rio Grande do Norte, 2004.

SCHNEIDER, Bruno de Oliveira; UCHÔA, Joaquim Quinteiro. **Legalidade de Software.** Lavras, MG: Universidade Federal de Lavras, 2000.

TORRES, Gabriel. **O Registro do Windows 9x.** 1998. Clube do Hardware. Disponível em: < http://www.clubedohardware.com.br/artigos/491/1>. Acessado em: 26/07/06.

VAROTO, Ane Cristina. **Visões em arquitetura de software**. São Paulo, SP: Dissertação de mestrado, Universidade de São Paulo, 2002.

ZIPF, José Gil Fausto. **Um Estudo baseado na análise da competitividade sistêmica.** Florianópolis, SC: Dissertação de mestrado, Universidade Federal de Santa Catarina, 2003.

BIBLIOGRAFIA RECOMENDADA

BATTISTI, Júlio. **Criando aplicações em 2,3 ou n-camadas.** 2003. Disponível em: < http://www.juliobattisti.com.br/artigos/ti/ncamadas.asp >. Acesso em: 08/05/2006

MECENAS, Ivan. **Firebird & Delphi 6**: Guia do desenvolvedor, 2002.

NOBRE, Marcos Aurélio. Interbase; **O poderoso sistema gerenciador de banco de dados relacional**, 2000.