UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA BIOMÉDICA

Ferramenta de Comunicação e Acesso Remoto a Imagens Médicas

CARMEM LÚCIA BORGES

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA – ÁREA DE CONCENTRAÇÃO ENGENHARIA BIOMÉDICA

Dezembro – 2003

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA BIOMÉDICA

Ferramenta de Comunicação e Acesso Remoto a Imagens Médicas

Autora: CARMEM LÚCIA BORGES
Orientador: Prof. Dr. EDUARDO TAVARES COSTA

Banca Examinadora:
Prof. Dr. Eduardo Tavares Costa (FEEC/UNICAMP) - Presidente
Prof. Dr. Eleri Cardozo (FEEC/UNICAMP)
Prof. Dr. Marco Antonio Gutierrez (InCor - SP)
Profa. Dra. Vera Lúcia de Silveira Nantes Button (FEEC/UNICAMP)

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA – ÁREA DE CONCENTRAÇÃO ENGENHARIA BIOMÉDICA

Dezembro – 2003

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DA ÁREA DE ENGENHARIA – BAE – UNICAMP

Borges, Carmem Lúcia
Ferramenta de comunicação e acesso remoto a imagens médicas / Carmem Lúcia Borges. --Campinas, SP: [s.n.], 2003.

Orientador: Eduardo Tavares Costa
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Diagnóstico por imagem. 2. Transmissão de imagem. 3. Java (linguagem de programação de computador). I. Costa, Eduardo Tavares. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: Tool for communication and remote access to medical images
Palavras-chave em Inglês: Diagnostic imaging, Picture transmission e Java (Computer program language)
Área de concentração: Engenharia Biomédica
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Eleri Cardozo, Vera Lúcia de Silveira Nantes Button e Marco Antonio Gutierrez
Data da defesa: 20/12/2003

RESUMO

O trabalho desta dissertação se baseia no desenvolvimento de uma ferramenta computacional (software) que permite a captação, transmissão, leitura, edição, e armazenamento remoto de dados biomédicos, com critérios de segurança, autenticação, autorização e integridade de dados. Como adicional importante, o software permite o acoplamento de um sistema de controle de serviços remoto. A aplicação segue um modelo de requisição de serviços cliente-servidor, sob uma arquitetura em camadas, escrita na linguagem de programação Java com as seguintes funcionalidades: visualização de imagens médicas (formato DICOM), edição remota de laudos com assinatura digital, armazenamento de dados relativos ao paciente (demográficos e imagens) e transmissão de imagens médicas. Seu objetivo principal é mostrar a possibilidade de edificação de softwares de acesso e controle remoto de dados biomédicos baseados em sistemas de segurança e permissão seletiva.

ABSTRACT

This work describes the development of a computational tool (software) that allows acquisition, transmission, reading, edition, and remote storage of biomedical data, following the protocols established for a Virtual Private Network – VPN, implying in security, authentication, authorization and integrity of data. The software allows the coupling of a control system of remote services. The application follows a model of requisition of services customer-server, under an architecture in layers, written in the programming language Java with the following functionalities: visualization of medical images (DICOM format), remote editing of medical findings with digital signature, storage of patient data (demographic and images) and transmission of medical images. Its objective is to show the possibility of construction of software for access and remote control of biomedical data based in security systems and selective permission.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS E LISTAGENS

AGRADECIMENTOS

Os meus agradecimentos pessoais são dedicados aos meus pais, filhos e amigos, sem os quais jamais conseguiria completar meu intento.

Frente a um desafio, as emoções se tornam mais fortes e as capacidades pessoais cedem espaço a elas. Apenas aqueles que insistem em ficar ao nosso redor, coitados deles, são capazes de formar o dique necessário ao suporte das águas do conhecimento, não as deixando escapar.

Neste período de tensão e ansiedade, jogada dentro do grande círculo da vida, muitos conseguiram entender que apenas estar por perto já significava muito e mantiveram presença constante e silenciosa.

Agradeço por todas as palavras amigas e confortantes e, muito, mas muito especialmente, àquele amigo de todas as horas que pôs a mão na massa para me ajudar na programação e me ensinou o que jamais esquecerei: o prazer de se doar sem receber nada em troca.

Às boas risadas, palavrões e brincadeiras, minhas saudades!

AGRADECIMENTOS ESPECIAIS

Os meus agradecimentos especiais são para uma pessoa que sempre acreditou em meu potencial, deu seu aval nas horas duvidosas e manteve a alegria de estar de bem com a vida me incomodando sem tréguas.

Àquele que muito mais que meu orientador, se tornou um amigo, uma pessoa de todas as horas, de quem posso ter o ombro para encostar e o rosto pra encher de beijos de alegria.

Obrigada!

CAPÍTULO 1 - INTRODUÇÃO

A existência de trabalhos de pesquisa inter-institucionais, envolvendo profissionais com qualificações distintas e de diferentes instituições de ensino e pesquisa, tem levado à busca de ferramentas computacionais que permitam a troca de informações, a comunicação remota de dados e, até mesmo, ações remotas para controle de equipamentos de pesquisa e, no caso de instituições hospitalares, equipamentos de diagnóstico (por imagem ou não). Na **Figura 1.1** pode-se ter a visão de um ambiente computacional e de pesquisa envolvendo diferentes instituições.

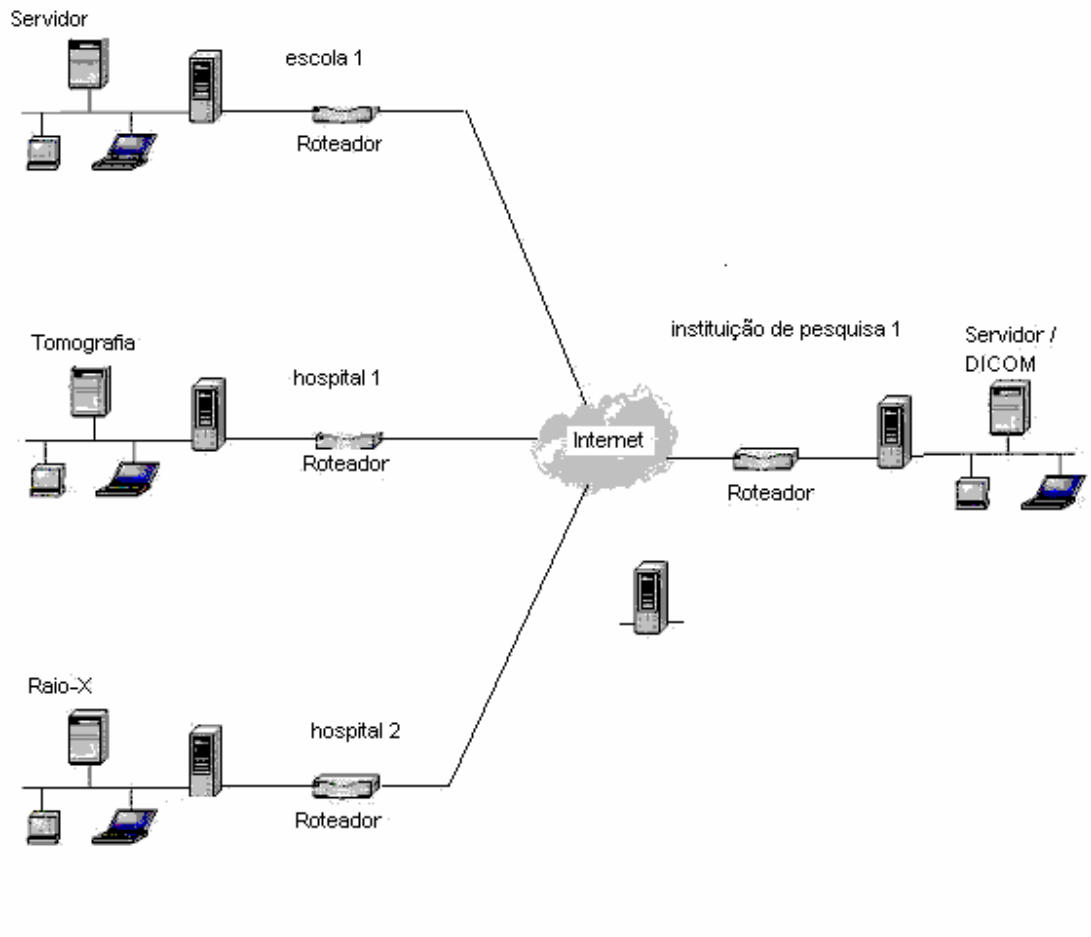

Figura 1.1 - Exemplo de ambiente computacional e de pesquisa inter-institucional.

Frente a uma enorme variedade de possibilidades dentro deste leque de serviços a serem disponibilizados por uma VPN (do inglês *Virtual Private Network* – Rede Privada Virtual), optou-se pelo desenvolvimento de uma ferramenta computacional (*software*) que permitisse a captação, transmissão, leitura, edição, e armazenamento remoto de dados biomédicos, desenvolvido na linguagem de programação Java, orientada a objetos.

O *software* foi projetado, entre outras funcionalidades, para servir como um visualizador de imagens médicas radiológicas (formato DICOM) e editor de laudos remoto.

Em redes hospitalares, as imagens digitalizadas são de grande interesse e conseqüente investimento tecnológico. A transmissão e o armazenamento remoto seguros, levando-se em consideração o fluxo elevado de imagens, são hoje fontes de pesquisa para desenvolvimento de soluções otimizadas.

Cada vez mais hospitais se propõem a realizar serviços onde a distância física pode se tornar um empecilho, como no apoio a decisões baseadas em diagnósticos médicos. A solução digital com comunicação virtual remota vem atendendo a estas necessidades, situando a imagem médica como forma de dado não exclusa deste conceito.

A fim de regularizar e permitir a comunicação entre sistemas digitais diferentes relativos a imagens biomédicas, padrões internacionais foram criados. Atualmente, o protocolo DICOM 3.0 (*Digital Imaging and Communications in Medicine* – Comunicação Digital de Imagens Médicas em Medicina) é considerado o padrão internacionalmente aceito por toda a comunidade médica e industrial-hospitalar, sendo, entretanto, um padrão de adesão, não obrigatório (SIEGEL & KOLODNER, 2000; DREYER & MEHTA, 2001).

O protocolo DICOM não se baseia em desempenho, mas em compatibilidade entre sistemas heterogêneos. Um conjunto de arquivos do tipo '.dcm' (imagens e dados relacionados), pertencentes a um exame de diagnóstico por imagem, pode totalizar um volume grande de bytes a ser armazenado, sendo fundamental a gerência de processos de fluxo e armazenamento (*storage*) (SIEGEL & KOLODNER, 2000).

Ficando o armazenamento dos arquivos originais restrito a um dos lados da conexão virtual e apenas as modificações armazenadas em locais diferenciados, pode-se chegar a soluções mais performáticas em um sistema que envolve dados biomédicos (FELIPE Jr., 2003).

A segurança envolvida na emissão do laudo também aparece como um problema a ser enfrentado, onde várias soluções comerciais existentes no mercado não valorizam significativamente os fatores de autorização e autenticação do laudo (DREYER & MEHTA, 2001).

Os diferentes aspectos acima abordados levaram ao estabelecimento de um software que implementa algumas funcionalidades importantes para comunicação inter-institucional.

CAPÍTULO 2 - OBJETIVOS

Esta dissertação tem como objetivo geral o estudo dos aspectos envolvidos no estabelecimento de serviços remotos relativos a dados biomédicos baseados em sistemas de segurança e permissão seletiva, e, como objetivo específico, o desenvolvimento de um *software* para acesso e controle remoto de dados biomédicos, tais como sinais, textos e imagens.

CAPÍTULO 3 – FUNDAMENTOS TEÓRICOS

3.1 – REDE FÍSICA DE COMUNICAÇÃO REMOTA

3.1.1 – Rede Privada Virtual – VPN

Redes Privadas Virtuais são conexões privadas seguras construídas sobre uma infra-estrutura de acesso público, como a Internet ou uma rede de telefonia. Constituem-se em túneis virtuais estabelecidos entre dois pontos remotos por uma conexão segura (SOUSA, 2002; TANENBAUM, 1989).

As VPNs podem ser segmentadas em três categorias: Acesso Remoto, Intranet, e Extranet. Uma Intranet VPN conecta posições fixas dentro de uma LAN (Rede de Área Local). Uma Extranet estende o acesso limitado de recursos a locais remotos, permitindo o uso de informação compartilhada (Rede de Área Ampla – WAN). Por Acesso Remoto, se conectam *telecommuters* e usuários móveis (**Figura 3.1**).

O Acesso Remoto pode ser mais bem definido como sendo o tipo de acesso no qual o usuário remoto não reside em local fixo, ou seja, o endereço IP (Internet Protocol) do equipamento de comunicação de rede (computador, telefone móvel, etc) do usuário não é fixo, não sendo previamente conhecido antes do estabelecimento da conexão (KOMPELLA & BONICA, 2003).

A máquina de Acesso Remoto aparece com um endereço externo roteável. Este endereço deve ser fornecido pelo provedor ou pelo administrador da rede na qual a máquina está temporariamente inserida. Pode-se notar que a máquina possui dois endereços ativos: um referente à rede física na qual está colocada, e outro virtual, associado à rede lógica estabelecida. Após a conexão na rede lógica, a máquina passa a se comportar como se estivesse fisicamente localizada na parte interna da rede.

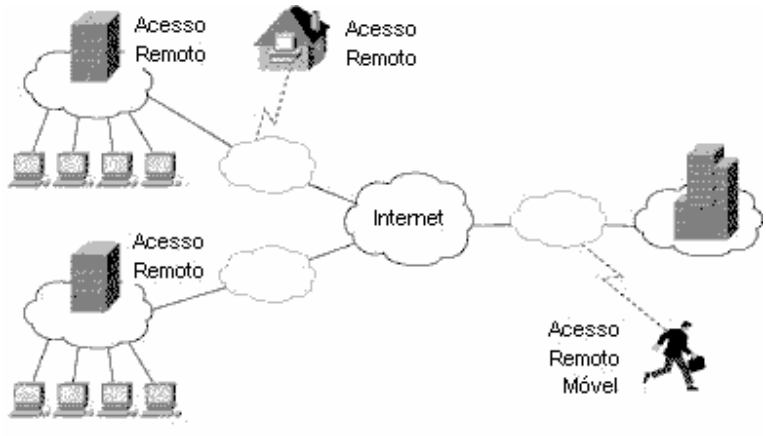

Figura 3.1 - Acesso remoto por uma VPN.

Soluções VPN evocam uma combinação de criptografia, certificação digital, autenticação segura de usuários e controle de acesso para garantir segurança no tráfego de dados. Estas características serão mais bem descritas em outros itens deste capítulo.

3.1.2 – Internet – arquitetura TCP/IP

A arquitetura TCP/IP refere-se basicamente a dois protocolos - Protocolo de Controle de Transmissão (*Transmission Control Protocol* - TCP) e Protocolo de Internet (*Internet Protocol* - IP), seguindo um padrão de comunicação cliente-servidor.

O protocolo IP não é orientado à conexão, isto é, não é elaborado sobre o conceito de conexão e porisso os pacotes de dados podem ser perdidos, atrasados ou serem entregues fora de ordem. A garantia de integridade dos dados e a segurança na transmissão são oferecidas pelo protocolo TCP, orientado à conexão (COMER, 1997).

Numa rede TCP/IP, para se comunicar, cada computador precisa ter o endereço da placa de rede (MAC) e o endereço IP do outro computador. O endereço IP (**Figura 3.2**) é composto de quatro bytes separados por pontos: uma parte representa a rede (NetID) e a outra identifica o equipamento (*host*) na rede (SOUSA, 2002).

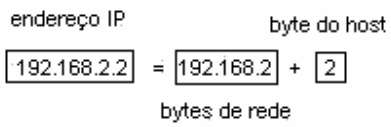

Figura 3.2 - Exemplo de endereço IP com três bytes destinados à rede (NetID).

O protocolo básico utilizado pelas entidades TCP é o protocolo de janela deslizante. Quando envia um segmento, o transmissor dispara um temporizador. Chegando o segmento ao seu destino, a entidade TCP receptora retorna um outro segmento com um número de confirmação igual ao próximo número de seqüência que espera receber. Se o temporizador do transmissor expirar antes de a confirmação ser recebida, o segmento é retransmitido (COMER, 1997).

3.1.3 – Tunelamento

A VPN estabelece túneis virtuais privados que carregam dados usando uma rede pública como meio de transporte. O conceito chave se baseia em uma tecnologia que permite a um protocolo de transporte de rede carregar a informação para outros protocolos dentro de seus próprios pacotes.

Em 1996, dois protocolos de tunelamento se destacavam: Protocolo de Tunelamento Ponto a Ponto (*Point-to-Point Tunneling Protocol* – PPTP) da *Microsoft* ® e Seqüencial Nível 2 (*Layer 2 Forwarding* – L2F) da *Cisco* ®. Estes protocolos evoluíram para uma fusão: o Protocolo de Tunelamento Nível 2 (*Layer 2 Tunneling Protocol* – L2TP), que suporta o estabelecimento de túneis simultâneos múltiplos para um único cliente (importantes na reserva de largura de banda e controle de qualidade de serviços – QoS).

Atulamente, o protocolo aberto Segurança de Protocolo de Internet (*IP Security* - IPSec), caracterizado posteriormente neste mesmo capítulo, fornece serviços de segurança ao nível da camada IP do modelo OSI (*Open Systems Interconnection*) da ISO (*International Organization for Standardization*).

Qualidade de Serviço (*Quality of Service* - QoS) é a capacidade que um elemento da rede (aplicação, *host*, *switch*, ou roteador) tem de oferecer serviços de transporte em nível pré-estabelecido, previsível e consistente (MORAES, 2002). A limitação dos protocolos TCP/IP no controle de latência e capacidade de transmissão se torna um problema para classes de aplicações novas, como as que incluem *streaming* (pacotes dinâmicos) de áudio e vídeo, VoIP e vídeo-conferência. Em redes que exigem alta capacidade de largura de banda e baixa latência, o uso de QoS se justifica.

Atualmente, as empresas fornecedoras de tecnologia VPN usam um misto de protocolos de tunelamento associados a padrões fortes (que oferecem maior segurança) de criptografia, descritos posteriormente neste capítulo.

A VPN, na Internet, acomoda uma variedade de protocolos encapsulados em pacotes IP, com garantia de integridade dos dados, estabelecendo túneis de criptografia e autenticação entre as máquinas pertencentes ao sistema. Isto acontece basicamente através da configuração de gateways, roteadores, firewalls ou similares (níveis 3 ou 4 do modelo OSI/ISO – camadas de rede e transporte – **Tabela 3.1**) (FIGUEIREDO & GEUS, 2001; NAKAMURA, 2000).

Como proteção extra, pode-se implementar criptografia de nível baixo (nível 2 – camada de enlace) ou alto, em nível de aplicativos, o que requer modificações particulares na programação dos mesmos (nível 5 ou superior – camadas de sessão, apresentação e aplicação). O uso de criptografia nas camadas protocolares baixas promove uma conexão segura dentro de uma rede tipicamente insegura como a Internet (CISCO SYSTEMS, 2003).

Tabela 3.1 - Funcionalidades das camadas do modelo OSI/ISO.

Camada (*Layer*)		**Funcionalidade**
7	**Aplicação**	Acesso ao ambiente OSI e sistemas distribuídos.
6	**Apresentação**	Independência nas diferentes representações de dados (sintaxe).
5	**Sessão**	Estabelece, gerencia e termina conexões entre as aplicações.
4	**Transporte**	Transferência de dados entre dois pontos de forma confiável, com funções como controle de fluxo e correção de erro ponto a ponto.
3	**Rede**	Independência das tecnologias de transmissão e comutação. Responsável por estabelecer, manter e terminar conexões.
2	**Enlace de dados**	Transmissão confiável de informação através do enlace físico. Envia blocos de dados sincronizados, com controle (erro e fluxo).
1	**Física**	Transmissão de seqüências de bits não estruturados em um meio físico. Trata as características mecânicas, elétricas e funcionais.

3.1.4 - Criptografia

Criptografia é a técnica de codificação onde os dados a serem criptografados são transformados em uma função parametrizada por uma chave (um enigma). Numa comunicação remota, a criptografia é realizada pelo emissor e a saída do processo transmitida pela rede, sendo posteriormente descriptografada pelo receptor (processo inverso da criptografia) (CARVALHO, 2000).

A privacidade de dados é alcançada pela criptografia, sendo o gerenciamento das chaves seu aspecto mais crítico e de difícil implementação, capaz de negociar e trocar chaves secretas de forma segura (KAGAN, 1998).

A criptografia (**Figura 3.3**) pode ser realizada como Chave Privada (*Secret Key Cryptography* – SKC), utilizando uma única chave para criptografar e descriptografar (a mesma tem que ser conhecida pelo emissor e pelo receptor), Chave Pública (*Public-Key Cryptography* – PKC), baseada em uma chave para criptografar (conhecida pelo emissor) e outra para descriptografar (conhecida pelo receptor), ou Funções *Hash*, onde se usa uma transformação matemática para tornar irreversível (não descriptografável) uma informação (SALOMAA, 1996; KESSLER, 1999).

As Funções *Hash* são utilizadas em aplicações bem específicas, como em assinaturas digitais (descritas posteriormente) sendo basicamente transformações que criam marcas nos dados (como impressões digitais), capazes de detectar alterações nos mesmos (KAGAN, 1998).

Numa VPN, é comum o uso do conceito de Chave Pública associada a uma Chave Privada, garantindo a autenticação do usuário na rede. Através da combinação de Certificação Digital (Chave Pública), emitida por cartórios digitais, e Chave Privada (como em uma senha de acesso), forma-se uma conexão segura – criptografada.

Em criptografia, a largura da chave faz a diferença na defesa contra ataques de força bruta (ataques emitidos simultaneamente por vários computadores) (SALOMAA, 1996). Quanto maior a chave (número de bits), maior a dificuldade de esta ser quebrada, tamanha a quantidade de combinações que podem ser formadas a partir de seus elementos binários. Exemplos de criptografia forte (nível de segurança alto) podem ser dados pelo Padrão de Criptografia de Dados (*Data Encryption Standard* – DES), criptografia de Chave Privada, e o RSA (*Rivest, Shamir, Adleman*), criptografia de Chave Pública com tamanho variável (40/128 bits) (DTOOL, 2003).

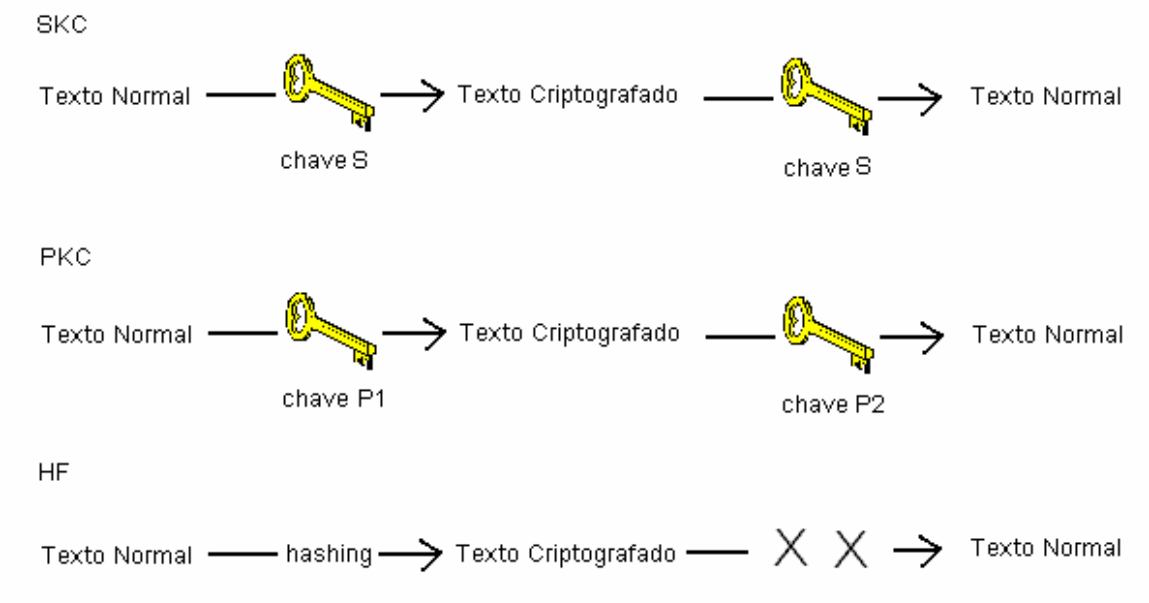

Figura 3.3 – Criptografia: Chave Secreta (SKC), Chave Pública (PKC) e Funções *Hash* (HF).

O DES, algoritmo criptográfico de Chave Privada, tem uma chave única de 64 bits, dos quais 56 são gerados aleatoriamente e usados diretamente pelo algoritmo. Os outros 8 bits são destinados à detecção de erros. O RSA, algoritmo de Chave Pública, resumidamente, multiplica dois números primos altos e, através de operações adicionais, gera dois grupos de números que se constituem chaves, uma pública (para criptografar) e outra privada (para descriptografar).

3.1.5 – Assinaturas digitais

Em princípio, qualquer algoritmo de chave pública pode ser usado em assinaturas digitais. O padrão no setor é o algoritmo RSA, citado anteriormente. O Padrão de Assinatura Digital (*Digital Signature Standard* – DSS), de 512 bits, também é muito utilizado e recomendado pelo NIST (*National Institute of Standards and Technology*) dos EUA (SALOMAA, 1996).

O Algoritmo de Assinatura Digital (*Digital Signature Algorithm* – DAS), pertencente ao DSS, é capaz de gerar e verificar assinaturas. A geração da assinatura utiliza uma chave privada para gerar uma assinatura digital. A verificação da assinatura utiliza uma chave pública correspondente (não a mesma) à chave privada. Cada usuário (emissor e

receptor) possui um par de chaves (privada e pública). Qualquer um pode verificar a assinatura de um emissor empregando a chave pública do mesmo, mas a geração da assinatura só pode ser executada pelo possuidor da chave privada.

O processo de geração da assinatura se inicia com a aplicação de uma função *hash*, especificada pelo Padrão Hash Seguro (*Secure Hash Standard* – SHS), gerando uma versão condensada de dados, o sumário da mensagem (*message digest*). Este sumário oferece a entrada necessária ao DSA para geração da assinatura digital, através de uma chave privada. A assinatura digital é então emitida ao receptor, que a verifica utilizando a chave pública e a função *hash* do remetente (**Figura 3.4**).

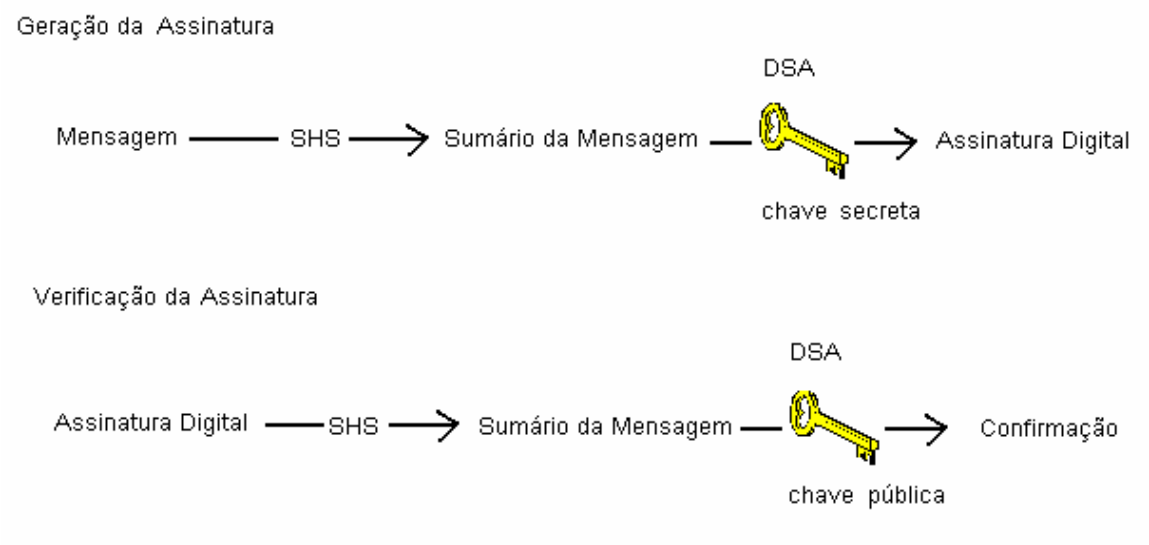

Figura 3.4 – Processos de geração e verificação de assinaturas digitais.

3.1.6 – Segurança da rede

O IPSec usa dois protocolos básicos de segurança – o AH (*Authentication Header*) e o ESP (*Encapsulating Security Payload*), ambos veículos para controle de acesso, baseados na distribuição de chaves criptográficas e gerenciamento de fluxo de tráfego.

Estes dois componentes do IPSec são empregados conjuntamente com algoritmos de Funções *Hash* (tipo de criptografia descrito anteriormente), tais como o MD5 (*Message Digest 5*) e o SHA (*Secure Hash Algorithm*). O protocolo IKE (*Internet Key*

Exchange), incluso no IPSec, fornece política de segurança compartilhada e autenticação de chaves (DORASWAMY & HARKINS, 1999).

Parte crítica de uma solução de segurança, o *firewall* da rede monitora o tráfego nos perímetros da mesma e impõe limitações de acordo com a política de segurança. Em uma solução VPN, *firewalls* protegem a rede de acesso desautorizado a recursos e ataques, como o Negação de Serviço (*Denial of Service*) – incapacidade de realização de serviço por requisição simultânea múltipla em excesso. Para tráfego autorizado, o *firewall* VPN verifica a fonte do tráfego e prescreve os privilégios de acesso que os usuários possuem (FIGUEIREDO & GEUS, 2001; SUMIMOTO & SUZUKY, 2003).

Um elemento adicional a ser utilizado na garantia de segurança é o Detecção de Intrusão (*Intrusion Detection* – ID). Enquanto os *firewalls* permitem ou negam o tráfego, baseados na fonte, destino, porta ou outros critérios, o ID analisa realmente o tráfego através do índice e contexto dos pacotes individuais, determinando se o tráfego é ou não autorizado. *Firewalls* e IDs, atuando conjuntamente, fornecem fortes mecanismos de segurança à rede (NAKAMURA, 2000).

3.1.7 – Qualificação VPN

Uma VPN é qualificada de acordo com as soluções oferecidas pela mesma, como segurança à intrusão, entrega de dados com confiança e integridade em tempo adequado ou facilidade de gerenciamento.

Para esta análise, os elementos essenciais de uma VPN podem ser segmentados em cinco grandes categorias:

- Escalabilidade – a habilidade de adaptação para ir de encontro às necessidades de mudança de largura de banda e conectividade é crucial em uma solução VPN;
- Segurança – tunelamento, criptografia e autenticação de pacotes são fundamentais para a segurança do transporte em redes públicas, assim como a autenticação do usuário e o controle de acesso são essenciais para atribuir privilégios e condições de acesso à rede (GEUS & REZENDE, 2002);
- Serviços VPN – gerência de largura de banda e funções de QoS, tais como enfileiramento, desvio de tráfego em congestionamentos, classificação de

pacotes e distribuição de passagem (opção pelo trajeto mais curto) otimizam o sistema;

- Aplicativos – *firewall*, Ids e auditores de segurança são alguns exemplos de incremento na segurança de uma VPN;
- Gerenciamento – políticas para reforço de segurança e gerenciamento de largura de banda, assim como monitoramento constante da rede são esforços necessários na implementação.

3.1.8 – Benefícios VPN

Alguns benefícios alcançados com a implementação de uma VPN podem ser:

- Redução de custos – a redução média de custos oscila em torno de 20 a 40% em Intranets, e cerca de 60 a 80% em Extranets (as VPNs são geralmente mais baratas do que redes privadas reais usando Linhas Privativas – LPs);
- Agilidade – uso de arquiteturas de rede mais flexíveis e escalonáveis do que as WANs clássicas;
- Gerência de rede facilitada – comparada a uma infra-estrutura de rede privada;
- Simplificação da topologia de rede – formação de uma rede integrada, com diminuição de sua complexidade e custo.

Fator importante, uma arquitetura VPN deve, em sua infra-estrutura de equipamentos, permitir a interoperabilidade entre seus dispositivos, como roteadores, *firewalls* e aplicativos.

3.2 – CONTROLE REMOTO DE DADOS

3.2.1 – Virtual Network Computing – VNC

Desenvolvido pelo Departamento de Engenharia da Universidade de Cambridge, o VNC é um *software* que possibilita a visualização e controle de máquinas remotas, independentemente de plataformas (Macintosh, Silicon Graphics, Unix, Solaris ou Windows).

Através de uma interface que visualiza a área de trabalho do alvo remoto, pode-se executar serviços e obter dados: acesso controlado por autorização seletiva (login por IP).

VNC é um simples protocolo de acesso remoto a interfaces gráficas de usuários, baseado no conceito de atualizações seqüenciais retangulares de imagens (valor do pixel + posição x,y do pixel). Quando a conexão cliente-servidor é estabelecida, o servidor negocia com o cliente algumas configurações, como tamanho da tela, formato dos pixels ou esquemas de codificação, formando a imagem original da tela servidor na tela do cliente. A atualização das modificações encontradas na tela é enviada pelo servidor em resposta a um pedido explícito do cliente. Ações internas no servidor são disparadas pelo cliente através de um dos seus dispositivos de entrada/saída (como mouse ou teclado) (CAMBRIDGE UNIVERSITY, 2003).

3.3 – PROTOCOLO DE COMUNICAÇÃO DE IMAGENS

3.3.1 – DICOM

O padrão internacional DICOM (*Digital Imaging and Communications in Medicine*) é uma especificação detalhada que descreve um meio de formatação e comunicação de imagens médicas com informações associadas. Seus objetivos são alcançar total compatibilidade e melhorar a eficiência do fluxo de trabalho (*workflow*) entre sistemas que envolvam imagens e outras informações de saúde (DICOM HOMEPAGE, 2003).

DICOM é relatado hoje como um padrão internacional na área de saúde (principalmente em radiologia, cardiologia, luz visível e odontologia), sendo um padrão de adesão, não imposição (FELIPE Jr., 2003).

3.3.2 – Histórico

O uso de imagens médicas digitais remonta à década de 70, sendo seu processamento digital pós-captação o responsável pela necessidade de formação de um comitê com o objetivo de criar um método padrão de transmissão.

DICOM é o resultado da aliança de usuários em potencial (*American College of Cardiology* e *American College of Radiology* – ACR) e fabricantes de equipamentos médicos (*National Electrical Manufacturer's Association* – NEMA). O comitê, fundado nos anos 80, reconheceu seu primeiro protocolo padrão, o ACR/NEMA V1, em 1985. Em 1986, o ACR/NEMA V2 surgiu como evolução do anterior (NEMA, 2003).

O DICOM 3.0, aceito e reconhecido como padrão atual, foi apresentado pela primeira vez à sociedade médico-hospitalar em 1992. Não considerado como evolução dos padrões anteriores, rompeu paradigmas com sua estrutura orientada a objetos (imagens, atributos e serviços). Sua versão atualizada data de 1999, mas periodicamente (cerca de dois em dois meses), alguns capítulos de sua documentação são alterados (comitê ativo em sua evolução) (FELIPE Jr., 2003).

Atualmente, o Comitê de Padronização DICOM (*DICOM Standards Committee*) é composto de vários membros, de interesses variados reunidos em grupos de trabalho (*workgroups*), e conta com a colaboração de parceiros, como HL7 (*Health Level 7*), IHE (*Integrating the Healthcare Enterprise*) e ISO (*International Standards Organization*).

Os membros do Comitê de Padronização DICOM são constituídos de vendedores de tecnologia biomédica, usuários e outros interessados em manter universalmente aceito o padrão atual, DICOM 3.0, com o objetivo de facilitação de processos ligados à tecnologia de comunicação de imagens biomédicas. O agrupamento em setores (grupos de trabalho) facilita as discussões e avaliações sobre o assunto.

Durante o desenvolvimento do padrão DICOM, muita atenção foi devotada a estabelecer relacionamentos de funcionamento com outros padrões internacionais.

O protocolo TCP/IP foi adotado em 1993. A cooperação contínua com o CEN (*European Committee for Standardization*) resultou em um número de suplementos conjuntamente desenvolvidos. Paralelamente, a convergência do formato japonês de intercâmbio de mídias (IS&C) com o DICOM exigiu muito trabalho comum com a JIRA (*Japan Industries Association of Radiological Systems*). Nos EUA, o Comitê DICOM participou dos esforços de coordenação para padrões de saúde, começando a definir ligações com o HL7 (padrão de formatação de dados em sistemas clínicos) e resultando na criação, em 1999, de um grupo de funcionamento da junção DICOM-HL7.

O Comitê DICOM está focalizando atualmente sua atenção à evolução dos padrões ligados à Internet. A estratégia do Comitê é integrar o padrão DICOM nas Recomendações de Internet (*Internet Recommendations*) assim que estas se tornarem mais estáveis e disseminadas. Nesta evolução, muito cuidado tem sido tomado para assegurar a consistência do padrão DICOM (DICOM HOMEPAGE, 2003; NEMA, 2003).

3.3.3 – Documentação

Em constante adaptação e evolução, o padrão DICOM engloba uma série de outros protocolos e foi estruturado em um documento original de múltiplas partes (**Tabela 3.2**) (DICOM HOMEPAGE, 2003).

Baseado nestes conceitos descritos nos capítulos, cada produto que trabalha com imagens tem uma documentação própria de capacidades e funcionalidades oferecidas dentro do padrão DICOM, o Estatuto de Compatibilidade (*DICOM Conformance Statement*), considerado documento base para análise de compatibilidade entre dois dispositivos em um sistema de transmissão de imagens.

3.3.4 – Tecnologia

O padrão DICOM refere-se a vários níveis do modelo de rede OSI/ISO e fornece a sustentação para a troca de informação em intercâmbio de mídias. DICOM especifica atualmente um Protocolo de Camada Superior (*Upper Layer Protocol* – ULP) que é usado sobre o TCP/IP (independentemente da rede física).

O ULP define mensagens, serviços, objetos de informação e mecanismos de negociação das associações estabelecidas entre os equipamentos (**Figura 3.5**), assegurando a compatibilidade de comunicação necessária.

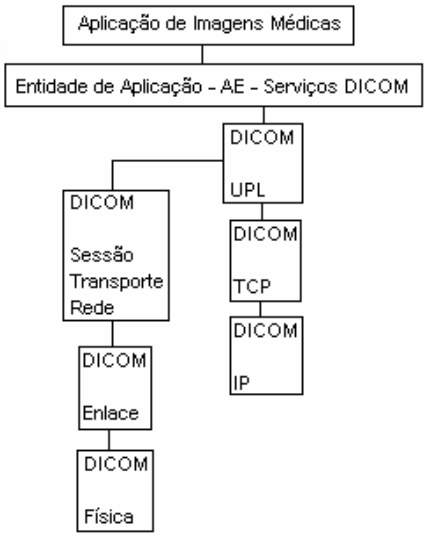

Figura 3.5 – Protocolo DICOM e camadas de rede.

Tabela 3.2 – Partes do documento padrão DICOM.

Partes - documento DICOM	
DICOM Part 1: Introduction and Overview **DICOM Part 2**: Conformance **DICOM Part 3**: Information Object Definitions **DICOM Part 4**: Service Class Specifications **DICOM Part 5**: Data Structure and Semantics **DICOM Part 6**: Data Dictionary **DICOM Part 7**: Message Exchange **DICOM Part 8**: Network Communication Support for Message Exchange	**DICOM Part 10**: Media Storage and File Format for Media Interchange **DICOM Part 11**: Media Storage Application Profiles **DICOM Part 12**: Media Formats and Physical Media for Media Interchange **DICOM Part 14**: Grayscale Standard Display Function **DICOM Part 15**: Security Profiles **DICOM Part 16**: Content Mapping Resource

A independência da rede física permite que o DICOM seja utilizado em inúmeras aplicações, como dentro de uma Intranet, sobre Linhas Privativas (LPs) ou inserido em VPNs.

DICOM não define uma arquitetura nova para um sistema inteiro, nem especifica exigências funcionais além do comportamento definido para serviços específicos. Como exemplo, o armazenamento de objetos-imagem é definido somente nos termos de qual formação deve ser transmitida ou retida, não como as imagens são disponibilizadas ou anotadas. DICOM pode ser considerado como um padrão de comunicação dentro dos limites existentes entre aplicações heterogêneas, dispositivos e sistemas (DICOM HOMEPAGE, 2003).

3.3.5 – Objetos e Serviços no Padrão DICOM

O padrão DICOM utiliza conceitos de programação orientada a objetos, onde classes são definidas, com seus atributos e métodos, e posteriormente instanciadas (criação de um objeto dentro de uma classe).

A arquitetura segue um padrão cliente-servidor, onde são feitas conexões virtuais (*Associations*) entre os dispositivos. Como aspectos relevantes destas conexões estão a negociação (*setup*), a duração da associação e o número de associações possíveis para cada dispositivo (DICOM HOMEPAGE, 2003).

Os Objetos de Informação (*Information Objects*) são associados a Classes de Serviço (*Service Class* ou *Application Entity*) formando um par (objeto, ação) denominado Par Serviço-Objeto (*Service-Object Pair* – SOP). O SOP pode ser referência de uma Classe (*SOP Class*) ou uma instância da classe (*SOP Instance*). Cada SOP é individualizado por um Identificador Único (*Unique Identifier* – UID) (FELIPE Jr., 2003).

O papel de um dispositivo numa funcionalidade, quem solicita ou quem executa a ação – cliente ou servidor, tem de ser definido: Classe de Serviço de Usuário (*Service Class User* – SCU) ou Classe de Serviço de Provedor (*Service Class Provider* – SCP).

Os serviços DICOM são formados por um ou mais comandos, os Elementos de Serviço de Mensagens (*DICOM Message Service Elements* – DIMSE), que podem ser compostos (*Composite* – C) ou normalizados (*Normalized* – N).

Alguns exemplos de Serviços de Classe DICOM são listados abaixo, relacionados a seus comandos específicos dentro do papel exercido pela funcionalidade:

- Verificação (*DICOM Verification*) – verifica a conexão lógica e física – comando C_Echo (SCU e SCP);
- Armazenamento (*DICOM Store*) – envio de cópia de dados – comando C_Store (SCU e SCP);
- Pergunta/Resposta (*DICOM Query/Retrieve*) – pesquisa em banco de dados – comandos C_Find (SCU) e C_Move (SCP);
- Modalidade Lista de Trabalho (*DICOM Modality Worklist*) – disponibilização de dados demográficos (HIS ou RIS) na modalidade (equipamento) – comando C_Find (SCU);
- Impressão (*DICOM Print*) – comandos N_Get, N_Create, N_Set, N_Action e N_Delete (SCU) e N_Eventrep (SCP).

No DICOM Conformance Statement (documento), estão relacionados os Contextos de Apresentação (*Presentation Context*) – codificações de cada funcionalidade: Sintaxe Abstrata (*Abstract Syntax*) – Classe *SOP* e UID – e Sintaxe de Transferência (*Transfer Syntax*) – atributos – tags, especificação VR, tipo de compressão, rede e outros.

A Representação de Valor (*Value Representation* – VR) pode ser do tipo LE (*Little Endian* – padrão Intel) ou BE (*Big Endian* – padrão Motorola), de acordo com a ordem de leitura dos bytes, e ainda Implícita (*Implicit* - I) ou Explícita (*Explicit* - E), dependendo de sua especificação estar ou não explicita no pacote de bytes, compondo uma das quatro possíveis siglas: LEI, LEE, BEI e BEE.

A compressão especificada pela Sintaxe de Transferência pode ser de vários tipos (algoritmos matemáticos diferentes), sendo as mais comumente utilizadas a JPEG (*Joint Photographic Experts Group*) sem perdas (*lossless*) ou com perdas (*lossy*). A *lossless* comprime numa taxa de 3:1, enquanto a *lossy* pode chegar a taxas de 10:1 em sua compressão, produzindo a geração de artefatos na imagem, afetando sua qualidade.

3.3.6 – Informação DICOM

O modelo de informação DICOM segue uma estrutura piramidal (**Figura 3.6**).

Figura 3.6 – Modelo piramidal de informação DICOM (modificado de FELIPE Jr., 2003).

Quando se faz uma requisição de busca de dados, as informações resultantes seguem este modelo de informação na resposta gerada.

O formato de arquivo genérico DICOM contém um pequeno cabeçalho (*Dicom File Meta Information Header*), com 128 bytes de preâmbulo, onde o usuário pode colocar qualquer informação relativa ao paciente, seguido do prefixo de 4 bytes "DICM" e atributos – Elementos de Dados (*Data Elements*), como dados demográficos do paciente e valores dos *pixels* (*pixel data*, ou seja, a imagem do exame propriamente dita) (**Figura 3.7**) (DICOM HOMEPAGE, 2003).

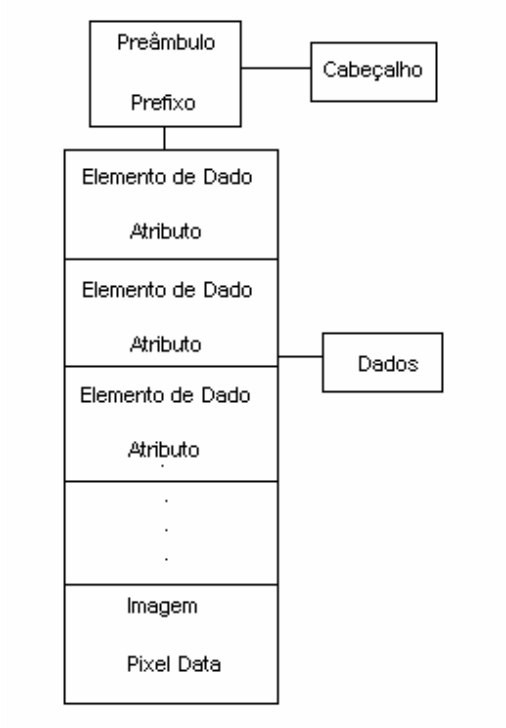

Figura 3.7 – Formato de arquivo genérico DICOM (modificado de DICOM HOMEPAGE, 2003).

3.4 – LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

3.4.1 – Programação JAVA

Com a explosão do uso da Internet nos anos 90, a *Sun Microsystems*, vendo o potencial de utilização Java na criação de páginas Web de conteúdo dinâmico, anunciou formalmente a nova linguagem em 1995.

Linguagem orientada a objetos, Java possibilita o desenvolvimento de programas (aplicativos) portáveis – sistemas multiplataforma (SUN MICROSYSTEMS, 2003).

A Programação Orientada a Objetos (POO) encapsula os dados (atributos) e métodos (comportamentos) em objetos. Os objetos são instanciados (criados pelo método construtor) a partir de uma classe, que pode ser classificada como um tipo definido pelo usuário (KAMIN & MICKUNAS, 1998).

As variáveis de instância e métodos de uma classe pertencem ao escopo da classe, onde o acesso aos mesmos se baseia em critérios de permissão declarados em

cada classe. Como a comunicação entre objetos se faz através de interfaces, isto pode permitir a uma classe ocultar informações, fator benéfico em segurança (HORSTMANN & CORNELL, 2000).

3.4.2 – Tecnologia

As principais tecnologias embutidas na Programação Orientada a Objetos são:

- Herança – forma de reutilização de *software* em que novas classes são criadas a partir de classes existentes, absorvendo seus atributos e comportamentos, e os sofisticando com capacidades que as novas classes exijam;
- Polimorfismo – classes especificadas de forma geral na programação e especificadas de acordo com a necessidade.

O tratamento de exceções Java permite a um programa capturar e tratar erros, não os deixando interferir na execução do programa (uso de *try* e *catch* ou *throws*) (DEITEL & DEITEL, 2001).

O estilo mais popular de sistema de banco de dados (coleção integrada de dados) é o banco de dados relacional. A SQL (*Structured Query Language* – SQL) é a linguagem universalmente utilizada em sistema de banco de dados relacional para fazer consultas.

Entre os pacotes mais comuns de *software* de banco de dados relacional estão o Microsoft Access, o Sybase, o Oracle, o Informix, o Microsoft SQL Server e o MySQL. Para manipular estes programas, através de programação Java, utiliza-se um pacote de conectividade, o JDBC (*Java Database Connectivity*) (DEITEL & DEITEL, 2001).

O acesso a recursos de rede remotos se baseia na tecnologia RMI (*Remote Method Invocation*), onde métodos são invocados remotamente por Chamadas de Procedimento Remoto (*Remote Procedure Calls* – RPCs). A RPC realiza todas as funções de rede e ordenação (*marshalling*) dos dados (empacotamento de argumentos de função e valores de retorno para transmissão em uma rede) (GROSSO, 2002).

Uma desvantagem da RPC é que ela suporta um número limitado de dados simples, outra, é exigir a implementação de uma Linguagem de Definição de Interface (*Interface Definition Language* – IDL), que descreve as funções a serem invocadas remotamente.

A RMI possibilita a transferência de objetos de tipos de dados complexos via mecanismo de serialização de objeto. A classe *ObjectOutputStream* converte qualquer objeto *Serializable* em um fluxo de bytes que pode ser transmitido através de uma rede. A classe *ObjectInputStream* reconstrói o objeto original para ser utilizado no método receptor (LYON, 1999).

Passos necessários para criar um sistema distribuído com RMI:

- Definir uma Interface Remota (IR) que descreva como o cliente e o servidor podem se comunicar – interface que estende a interface *Remote* (pacote *java.rmi*);
- Definir o aplicativo servidor que implemente a Interface Remota (por convenção, mesmo nome da IR com a extensão *.Impl*);
- Definir o aplicativo cliente que utilize uma Referência de Interface Remota (RIR) para interagir com a implementação de servidor da interface (objeto da classe que implemente a Interface Remota);
- Compilar e executar o servidor e o cliente – uso do compilador remoto *rmic*, posterior carregamento do registro de RMI na porta específica do computador onde o comando será executado (*rmiregistry*) e execução local (*java*).

3.4.3 – Aspectos da linguagem

O programa pode ser um aplicativo, armazenado e executado localmente, ou um *applet*, armazenado num computador remoto, carregado no navegador (*browser*), executado localmente por um interpretador Java embutido no navegador e descartado após a sua execução. Os *applets* também podem ser executados a partir da linha de comando *appletviewer*, que requer um documento HTML para invocar o *applet* (HORSTMANN & CORNELL, 2000).

Os programas Java normalmente passam por cinco fases para serem executados:

- Edição – consiste na edição de um arquivo através de um programa editor;
- Compilação – o comando *javac* traduz o programa Java para *bytecodes* (geração dos arquivos.*class*), a linguagem entendida pelo interpretador Java;
- Carga – para ser executado, o programa deve ser carregado na memória pelo carregador de classe;

- Verificação – antes dos *bytecodes* de um *applet* serem executados, são verificados pelo verificador de *bytecode*, a fim de assegurar a validade dos mesmos e a não violação das regras de segurança Java;
- Execução – o interpretador Java interpreta um bytecode por vez (mais segurança, porém maior lentidão).

O conjunto de ferramentas J2SDK inclue o visualizador (*appletviewer*), o compilador (*javac*), o interpretador (*java*) e outras utilidades a programadores (SUN MICROSYSTEMS, 2003).

Programas Java consistem em pacotes, com classes, compostas de atributos e métodos próprios. As Bibliotecas de Classes – Java APIs *(Applications Programming Interfaces)* são pacotes de classes pré-programadas que podem ser aproveitados em outros programas.

A modularização do programa em pacotes tem o objetivo de facilitar a programação (evolução em etapas), evitar repetição de métodos (podem ser invocados por outras classes ou pacotes) e dificultar a utilização de espaço desnecessário de memória (uso de variáveis locais) (DEITEL & DEITEL, 2001).

CAPÍTULO 4 – DESENVOLVIMENTO DO SOFTWARE

O modelo escolhido para desenvolvimento de um aplicativo que pudesse realizar a captação, transmissão, leitura, edição, e armazenamento remoto de dados biomédicos foi um visualizador de imagens médicas radiológicas (formato DICOM) e editor de laudos remoto.

A transmissão de imagens biomédicas dentro de redes públicas pode ser um problema quanto à segurança, confiabilidade e integridade de dados. Aplicativos integrados a uma VPN (Virtual Private Network) poderiam então ser desenvolvidos como soluções viáveis a esta problemática. Com este propósito, o programa desenvolvido neste trabalho, designado Diagnosis Viewer, foi elaborado com características, descritas posteriormente, baseadas na fundamentação teórica levantada.

4.1 – DESCRIÇÃO GENERALIZADA

O *software*, elaborado sob a linguagem de programação Java, modulado em pacotes, permite a visualização de imagens médicas (formato DICOM), edição remota de laudos com assinatura digital, armazenamento de dados relativos ao paciente (demográficos e imagens) e transmissão de imagens médicas. De forma modular, permite o acoplamento de um sistema para controle remoto de serviços computacionais.

A escolha da linguagem de programação Java decorreu do fato de ser esta uma linguagem orientada a objetos, possibilitando o desenvolvimento de um aplicativo portável (multiplataforma) e modular. As características implícitas na linguagem, descritas no capítulo anterior, pressupõem portabilidade (independência de plataforma operacional), escalabilidade (abertura para capacidade de crescimento modular) e ubiqüidade (condições de operação onipresente), tornando sua evolução facilitada e passível de modificações em módulos (CAMBRIDGE UNIVERSITY, 2003).

A preocupação com portabilidade (capacidade do *software* de ser executado em qualquer plataforma operacional) e interface amigável (facilidade de uso pelo usuário) esteve sempre presente no desenvolvimento do *software*, assim como o respeito às exigências protocolares de segurança de uma VPN.

A segurança embutida na linguagem, codificação de dados e/ou comandos em bytecodes e a verificação posterior dos mesmos antes de sua execução (ver **Listagem A.3** no **Anexo**), garantem a autenticidade e integridade dos dados obtidos (HORSTMANN & CORNELL, 2000). Não foi dado tratamento específico à questão da criptografia dos dados enviados e recebidos.

A aplicação segue um modelo de requisição de serviços cliente-servidor, estando qualquer máquina ligada ao sistema apta a exercer funções específicas de servidor e/ou cliente, dependendo do evento solicitado.

Em sua arquitetura cliente-servidor, onde o cliente requisita algum serviço disponível pelo servidor remoto, o servidor executa o procedimento chamado e transmite o resultado a ser mostrado do lado cliente.

O *software* cliente se conecta ao servidor remoto através de uma interface amigável, dispara a requisição de alguma funcionalidade e traz na tela seu resultado. O cliente invoca métodos das classes do servidor remotamente e, após realizada a função requerida, o servidor transmite a resposta pela rede, conseguindo esta estar disponibilizada de forma segura e íntegra na máquina cliente.

Sendo totalmente modular, o *software* tem em suas classes componentes garantias de acesso restrito ao escopo de classes ou pacotes e programação aberta a futuras implementações.

Para estabelecer e manter a conexão remota, o *software* implementa a tecnologia RMI (Invocação de Métodos Remotos) em sua programação, especificada no capítulo anterior, estabelecendo um canal aberto de transmissão entre cliente e servidor enquanto estiver sendo realizada alguma funcionalidade remota (GROSSO, 2002).

Como o aplicativo trabalha com imagens médicas, o padrão internacional DICOM 3.0 foi respeitado. Entretanto, não se trabalhou com compressão de imagens, apenas com sua transmissão e armazenamento remoto.

4.2 – ESTRUTURA FUNCIONAL

O *software* se baseia em uma arquitetura em camadas estabelecida pelos diferentes níveis de funcionalidade presentes: *Model View Controller* (MVC), conforme

especificação do GOF (*Gang of Four*), grupo de especialistas em programação unidos no ideal de desenvolvimento de padrões de linguagem (SUN MICROSYSTEMS, 2003).

As camadas do sistema foram definidas por funcionalidade e se adequam às necessidades de serviços oferecidos pelo *software*: acesso a banco de dados, interface usuário/cliente, requisição de serviços remotos e controle de regras de negócios (processos de informações).

Funcionalmente, as classes foram agrupadas e implementadas em três grandes pacotes:

- Pacote *cliente* (*unicamp.cliente*): classes responsáveis pela interface cliente, que se preocupa com a apresentação dos dados ao usuário e é responsável pelo acionamento das requisições de serviços, além das classes destinadas ao estabelecimento da conexão com o servidor;
- Pacote *servidor* (*unicamp.servidor*): classes responsáveis pelas regras de negócios (processamento de informações: organização, tratamento e validação) e classes que permitem o acesso ao banco de dados remoto;
- Pacote *objetos* (*unicamp.objs*): classes caracterizadoras dos objetos que trafegam pela rede (objetos serializáveis - *serializables*): *medico*, *exame*, *paciente*, *laudo* e *diretorio*, com seus atributos gerados por tabelas **(**ver **Listagem A.4** no **Anexo).**

Os objetos do *software* se relacionam conforme o diagrama a seguir **(Figura 4.1)**:

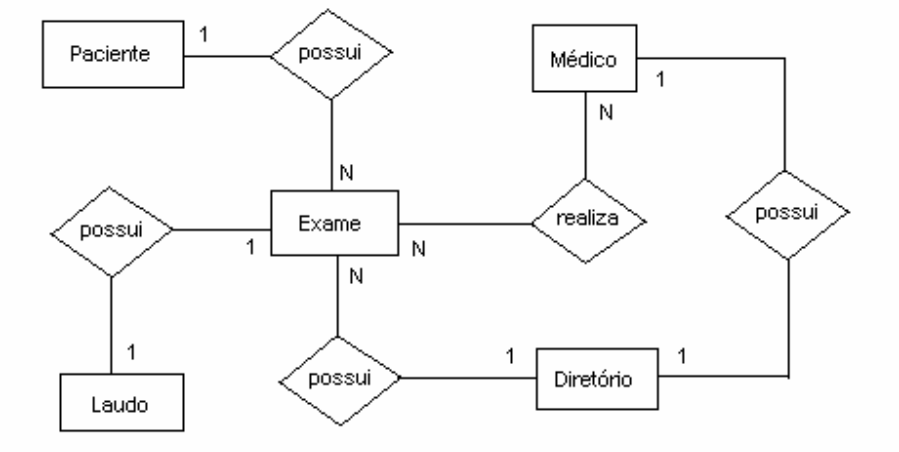

Figura 4.1 – Diagrama Entidade-Relacionamento (DER) do aplicativo desenvolvido.

O *software*, basicamente, trabalha por funcionalidades, estando restrito à seqüência de ações relatadas pelo fluxograma **(Figuras 4.2a, 4.2b e 4.2c)** abaixo:

- Conexão: o cliente tenta estabelecer uma conexão remota com o servidor, este verifica a autenticidade do usuário (login e senha) e, se usuário autêntico, verifica o tipo de permissão do usuário (restrições de acesso relacionadas diretamente ao banco de dados), o caracteriza e a conexão está estabelecida. Caso usuário não autêntico, a conexão não é estabelecida;
- Busca de exames: o cliente requisita ao servidor as informações desejadas armazenadas em um banco de dados remoto. Podem ser pedidos os exames relacionados a algum paciente, determinado período de tempo ou aqueles oriundos de algum tipo de equipamento de diagnóstico por imagem. Através de uma interface cliente-servidor - controle de regras de negócios, o servidor faz a busca e, se o número de exames encontrado for diferente de 0, retorna ao cliente os resultados encontrados na tela;
- Seleção do exame: frente a uma variedade de exames disponibilizados na tela, o usuário seleciona o exame desejado;
- Busca de dados do exame: automaticamente, após o exame selecionado, é feita uma requisição ao servidor dos dados do exame. Se a permissividade do usuário for básica, os dados demográficos do paciente e as informações do exame com o espaço para laudo, podendo este já ter sido digitado ou não, são disponibilizados na tela com as imagens em JPEG. Se o usuário tiver permissividade parcial ou total, os dados são disponibilizados com as imagens em DICOM e um diretório particular do usuário é criado;
- Alterações nas imagens: o usuário pode colocar qualquer imagem em destaque, tornando-a maior em janela independente. Se tiver permissão (usuário com permissividade total), o usuário poderá editá-la e salvá-la localmente ou remotamente, mas somente em seu diretório particular (nome do médico), preservando assim a imagem original (diretório de origem);
- Geração do laudo: a digitação do laudo ou sua edição pode ser realizada, com seu salvamento em diretório particular restrito a certos grupos de usuários (permissividade parcial ou total). Depois de realizado o laudo, este só pode ser salvo (diretório de origem) se lhe for conferida uma assinatura digital válida (usuário com permissividade total).

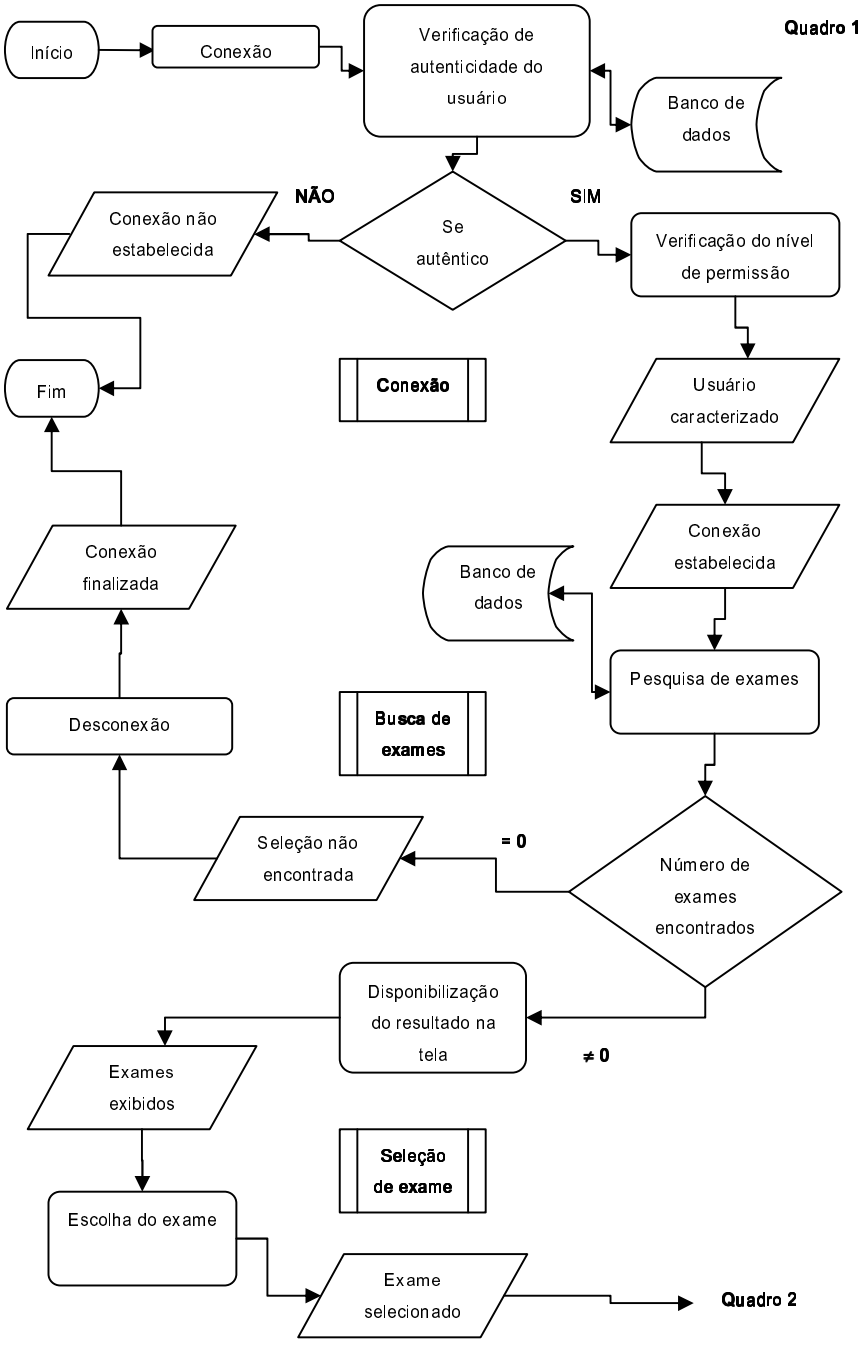

Figura 4.2a – Etapa 1 do fluxograma: conexão, busca e seleção de exames

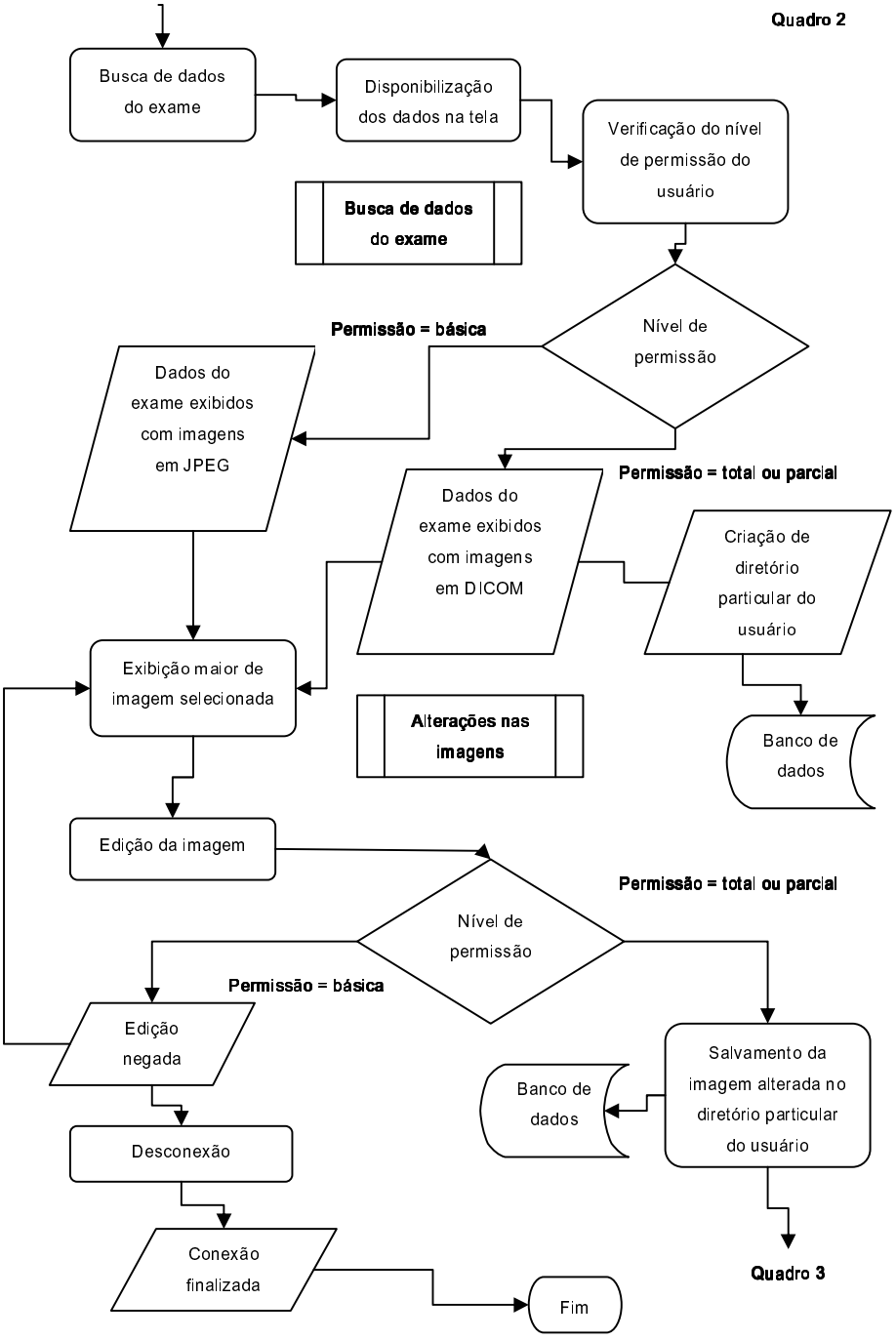

Figura 4.2b – Etapa 2 do fluxograma: busca de dados do exame e alterações nas imagens.

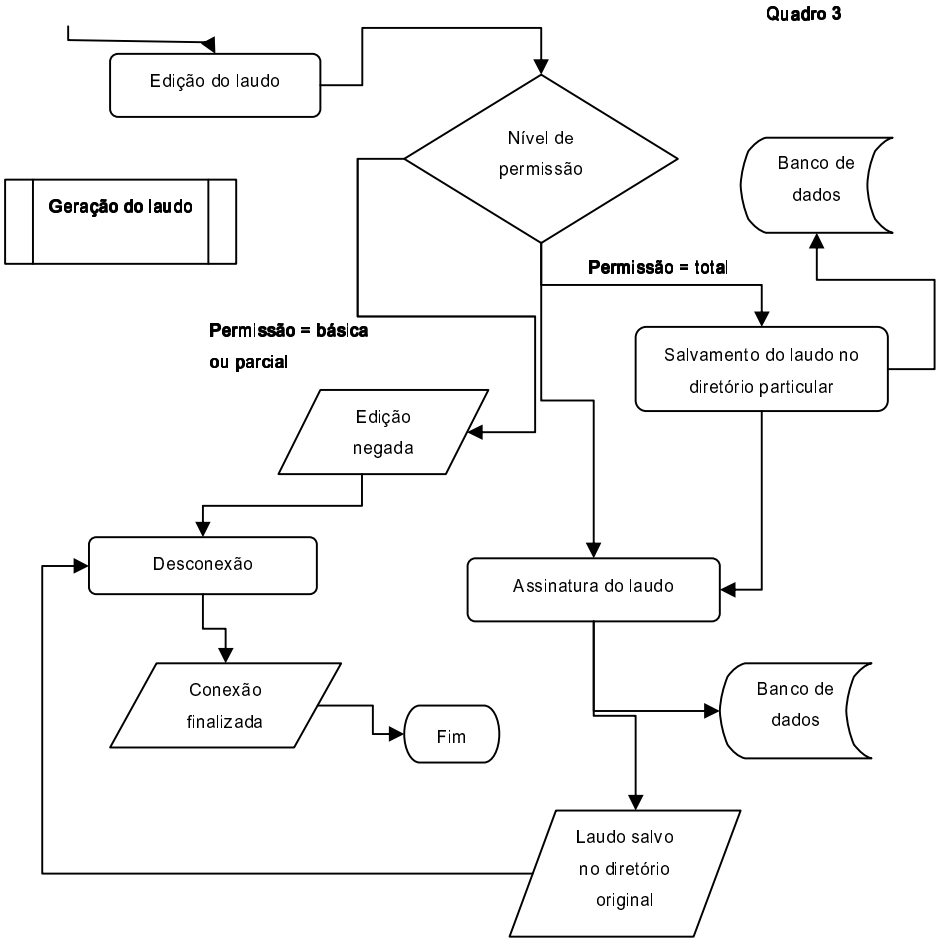

Figura 4.2c – Etapa 3 do fluxograma: geração do laudo.

O código fonte parcial do programa se encontra descrito em um Relatório Departamental (BORGES, 2003).

Algumas telas do *software* são apresentadas a seguir pelas **Figuras 4.3 a 4.6**:

Figura 4.3 - Telas de login do Diagnosys Viewer.

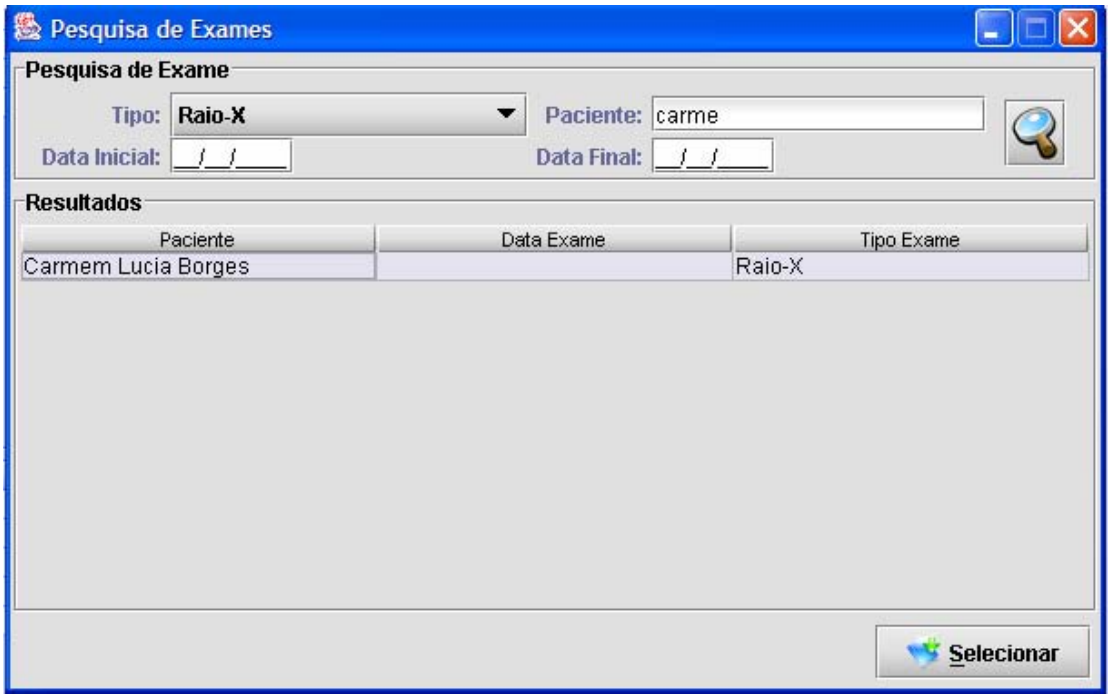

Figura 4.4 - Tela cliente para pesquisa de exames.

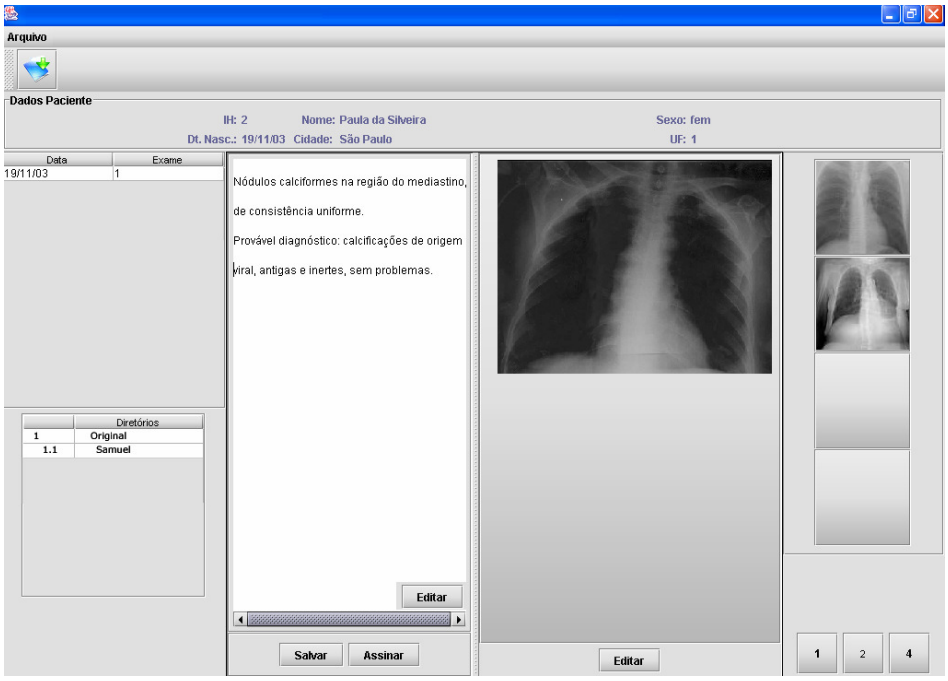

Figura 4.5 - Tela cliente de exame selecionado com imagens no formato JPEG.

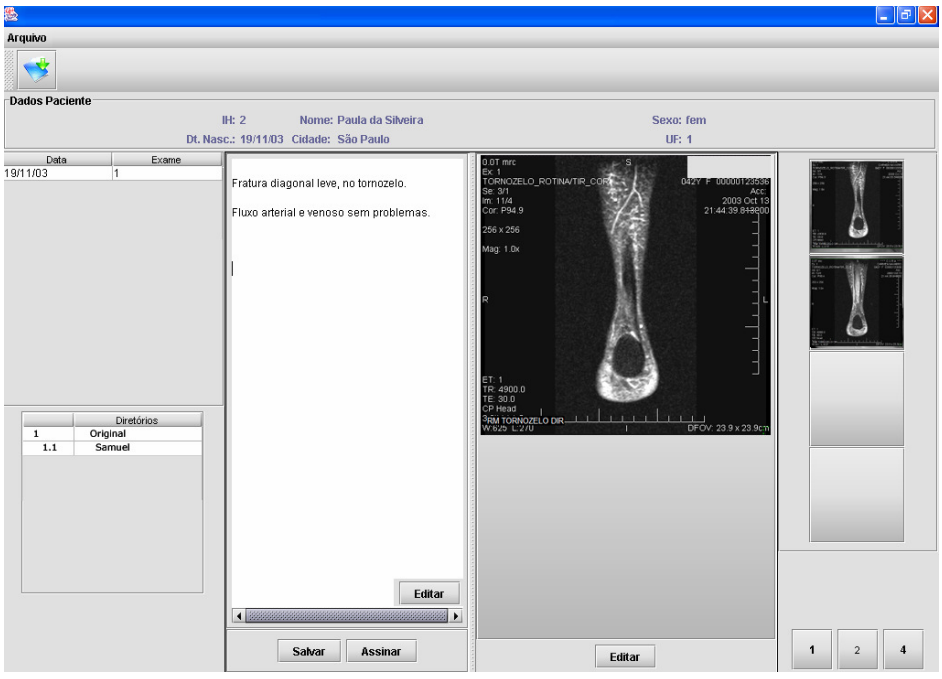

Figura 4.6 - Tela cliente de exame selecionado com imagens no formato DICOM.

CAPÍTULO 5 – TESTES E RESULTADOS

Qualidade de *software* pode ser definida como a combinação de taxas baixas de falhas com satisfação alta dos usuários (HEISER, 1997).

Os testes de qualidade de *software* têm seu enfoque direcionado ao objeto de estudo de controle, como desempenho, carga, entrada e saída de dados (teste de caixa preta), redundância (teste de caixa branca), integridade de dados e funcionalidade.

A aplicação dos testes pode ser por modelos padronizados definidos pelo mercado ou por metodologia própria particular (SARIKAYA, 1996), adaptando o teste aos objetivos desejados.

A escolha do tipo de teste a ser aplicado no *software* depende dos objetivos a que o mesmo se propõe. Por exemplo, em sistemas de comércio eletrônico, testes de carga são necessários para a avaliação da qualidade.

As opções escolhidas para este aplicativo desenvolvido foram de controle de *stress*, transmissão, confiabilidade e análise subjetiva, descritos posteriormente, com metodologia particular (SARIKAYA, 1996) adaptada aos objetos de teste, fazendo uso de *softwares* de testes automáticos existentes no mercado.

O nível de *stress*, por definição, determina o número de acessos simultâneos suportados pelo banco de dados, sistema ou meios de transmissão.

Ferramentas para testes automáticos são programas que avaliam se outro programa funciona como esperado e retornam respostas do tipo "sim" ou "não“, validando os requisitos de um sistema (PIESIGILLI, 2002).

JUnit, desenvolvido por Kent Beck e Erich Gamma, é um ambiente de trabalho (*framework*) que facilita o desenvolvimento e execução de testes automáticos de unidade em código Java. O JUnit pode verificar se cada método de uma classe funciona da forma esperada, mostrando, em caso de falha, a causa em cada teste. Serve de base para extensões (CLARK, 2003).

São direcionados os testes a partir da janela do JUnit **(Figura 5.1)**. Para executar o programa em sua interface gráfica, utiliza-se o comando: *java -cp junit.jar junit.swingui.TestRunner nomedaclasse* (por exemplo: *unicamp.servidor.infra.Servidor*).

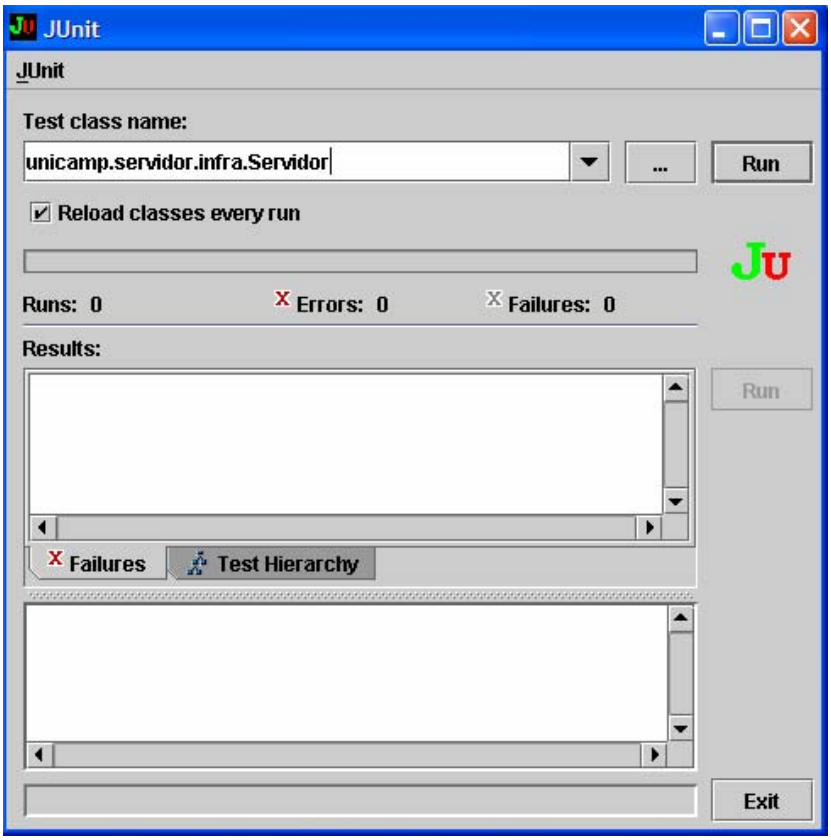

Figura 5.1 – Direcionamento de testes pelo JUnit.

Alguns testes adicionais podem ser acoplados ao Junit, para medir performance e escalabilidade, como o Jmeter, também utilizado neste trabalho, como um 'teste de *stress*' que simula carga pesada em servidor, rede ou objeto. São criados parâmetros **(Figura 5.2)** para averiguação (configuração), e direcionamento de busca e resposta (PIESIGILLI, 2002).

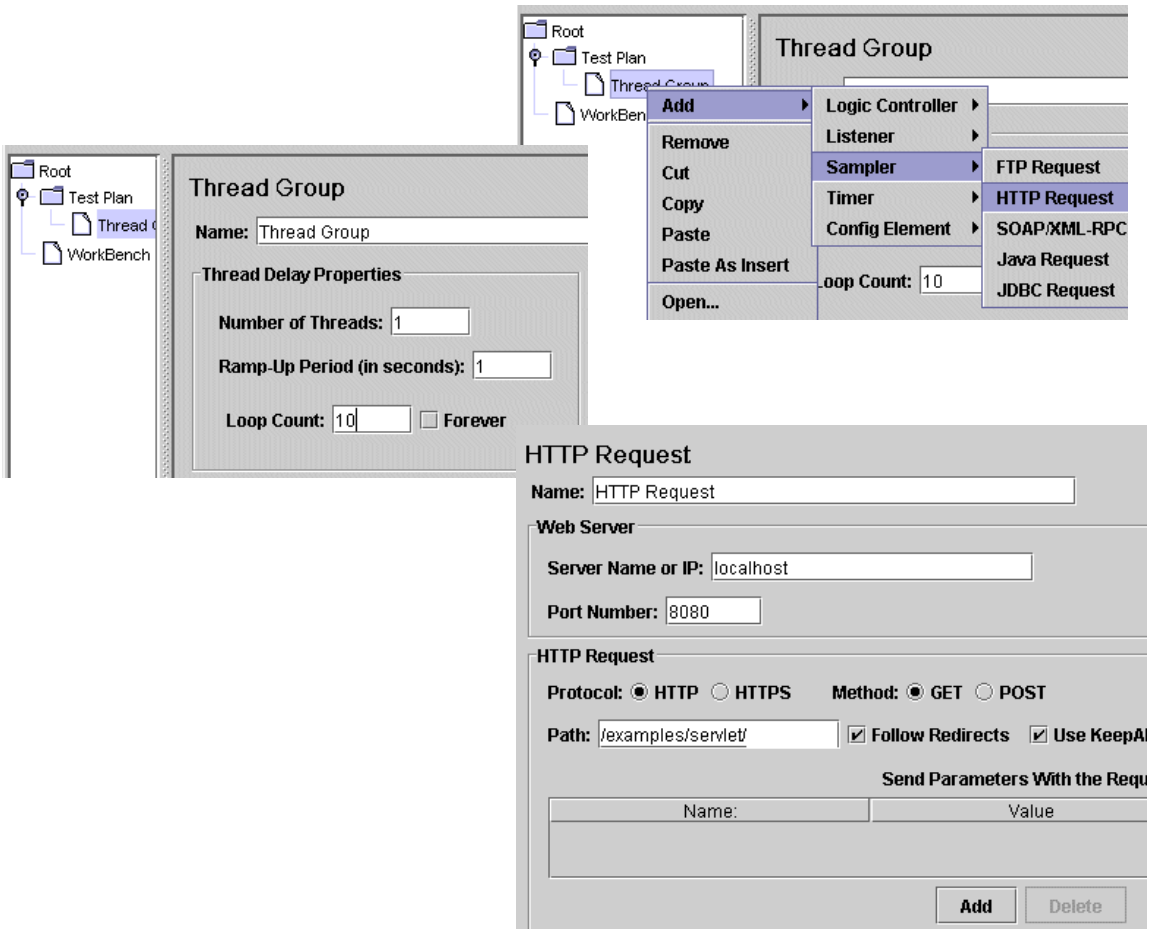

Figura 5.2 – Exemplo de configurações e parâmetros a serem inseridos no Jmeter.

5.1 – EXPERIMENTOS REALIZADOS

Para a realização dos testes foram empregadas as ferramentas JUnit e Jmeter, alterando as configurações necessárias a cada tipo de teste aplicado.

Os testes realizados para o *software* utilizaram como base dois computadores conectados remotamente. Os dados foram inseridos em banco de dados MySQL (*software* livre), estando as imagens DICOM armazenadas em diretórios locais (em cada computador) indexados ao banco de dados por campo específico **(Figura 5.3)**.

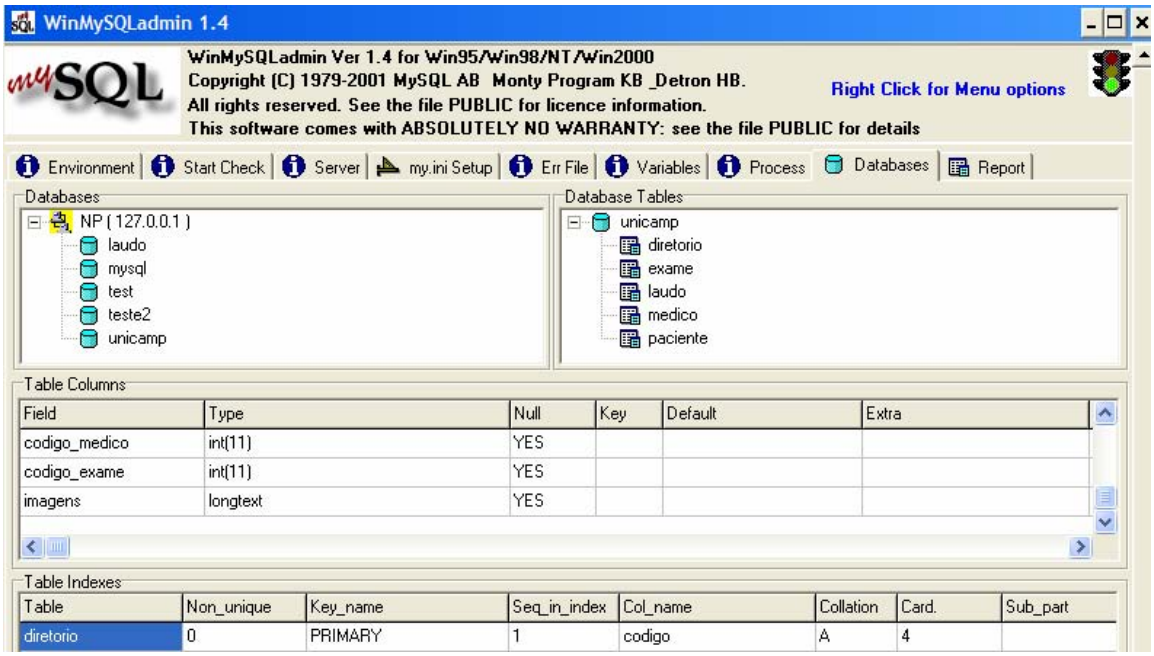

Figura 5.3 – Banco de dados *MySQL* e tabelas criadas.

5.2 – AMBIENTAÇÃO

Os experimentos foram realizados em ambiente doméstico, dentro das dependências da Escola Paulista de Medicina, no setor de Ressonância Magnética, sendo as especificações dos equipamentos empregados nos experimentos descritas a seguir:

Computador A (servidor):

- Processador AMD Duron 1GHz, 256MB RAM;
- Interface de rede 100Mbps;
- Sistema Operacional: Windows 2000.

Computador B (cliente):

- Intel Pentium 200Mhz, 64MB RAM;
- Interface de rede 100Mbps;
- Sistema Operacional: Windows XP.

5.3 – TESTES E RESULTADOS

5.3.1 – Teste de Transmissão

Para o teste de transmissão foram selecionadas aleatoriamente 20 séries de imagens de tamanhos diferentes, pertencentes a tipos de exames diferentes.

As imagens foram captadas por equipamentos pertencentes ao Hospital São Paulo, uma Ressonância Magnética Siemens e uma Tomografia Computadorizada Philips, e enviadas ao computador A através das funcionalidades DICOM (DICOM HOMEPAGE, 2003) C-Storage (SOPClassUIDs 1.2.840.10008.5.1.4.1.1.4 e 1.2.840.10008.5.1.4.1.1.2, respectivamente).

No computador A, as séries de imagens foram colocadas dentro de diretórios específicos criados e indexados ao banco de dados MySQL (2 séries por exame, com imagens selecionadas pelo técnico local), onde os dados textuais dos pacientes, médicos, exames e laudos (fictícios) foram inseridos nas tabelas respectivas.

Procedimento de teste:

O computador B (cliente), através do *software* Diagnosis Viewer, requisita ao computador A (servidor) as imagens relacionadas às séries de imagens, uma série por vez, sendo observado, registrado e comparado o número de imagens que aparecem disponibilizadas na tela do computador B e o existente no computador A, em cada série de imagens.

Resultados obtidos:

Os resultados obtidos foram registrados na **Tabela 5.1** e resumidos a seguir:

- Requisições enviadas: 20 pedidos de série de imagens;
- Requisições respondidas com sucesso total (imagens A = imagens B): 17 (85%);
- Requisições respondidas com sucesso parcial (imagens A ≠ imagens B): 2 (10%);
- Requisições não respondidas: 1 (5%).

Tabela 5.1 - Séries selecionadas e transmitidas do computador A ao B.

Série	Tamanho	Exame	Imagens - A	Imagens - B
1	5,2 MB	RMN 1	25	25
2	4,8 MB	RMN 1	20	20
3	6,6 MB	TC 1	12	12
4	6,2 MB	TC 1	10	10
5	6,2 MB	RMN 2	32	28
6	4,4 MB	RMN 2	15	15
7	4,2 MB	TC 2	6	6
8	4,5 MB	TC 2	8	8
9	6,3 MB	RMN 3	28	25
10	5,2 MB	RMN 3	22	0
11	5, 4 MB	TC 3	11	11
12	4,8 MB	TC 3	8	8
13	3,9 MB	RMN 4	14	14
14	2,5 MB	RMN 4	10	10
15	3,6 MB	TC 4	5	5
16	3, 8 MB	TC 4	6	6
17	2,9 MB	RMN 5	11	11
18	3,2 MB	RMN 5	12	12
19	6,5 MB	TC 5	14	14
20	5,9 MB	TC 5	12	12

5.3.2 – Teste de *Stress*

O teste de *stress* avalia a capacidade do sistema de suportar um número considerado elevado de tarefas similares simultâneas.

Neste caso, foram consideradas como tarefas de teste as requisições de transmissão das séries de imagens selecionadas para o teste de transmissão, descritas anteriormente.

Procedimento de teste:

O computador B (cliente) requisita ao computador A (servidor), através do *software* Diagnosis Viewer, a transmissão de n séries de imagens, de acordo com a **Tabela 5.2**, de forma simultânea, seqüencialmente e sem aguardar a conclusão da requisição anterior, sendo observados e registrados o número de requisições respondidas e as observações inerentes a cada caso de teste.

Resultados obtidos:

Os resultados obtidos foram registrados na **Tabela 5.2** e resumidos a seguir:

- Número de requisições enviadas simultaneamente: 4, 8, 12, 16 e 20;
- Requisições respondidas com sucesso total: se requisições enviadas = 4, 8 e 12;
- Requisições respondidas com sucesso parcial: se requisições enviadas = 16 e 20.

Tabela 5.2 – Relação de requisições enviadas de forma simultânea e resultados obtidos.

Requisições Enviadas	Requisições Respondidas	Requisições Não
4	4	
8	8	
12	12	
16	14	13 e 14 sem resposta
20	15	15, 16 e 19 sem resposta

5.3.3 – Teste de Confiabilidade

Este teste tem como objetivo verificar a confiabilidade do sistema em uma de suas funcionalidades. A funcionalidade escolhida foi a de armazenamento remoto de dados, visto a importância de sua integridade.

Procedimento de teste:

O computador B (cliente), após requisição e abertura de laudo em sua tela, realiza a edição do mesmo e requisita ao computador A (servidor) a gravação das alterações em sua base de dados (banco de dados do computador A).

Foram alterados pelo computador B os 10 laudos pré-existentes no computador A (inserção da palavra 'teste' na primeira linha de cada laudo) e pedidas para serem gravadas suas alterações remotamente, sendo os dados textuais (pré e pós-alterações) comparados e registrados.

Resultados obtidos:

Em todas a alterações de laudos realizadas pelo computador B e gravadas remotamente no computador A, não houve registro de irregularidades.

5.3.4 – Análise subjetiva

Para fins de análise subjetiva do software desenvolvido, quatro profissionais da Escola Paulista de Medicina (médicos e técnicos), profissionais envolvidos em processos de visualização de imagens digitalizadas e emissão de laudos, concordaram em avaliar o software e emitir suas opiniões respondendo a um sucinto questionário.

Procedimento de teste:

Os profissionais tiveram acesso livre ao software, sem treinamento prévio, e puderam realizar quaisquer das funções disponibilizadas pelo mesmo convenientes à sua própria avaliação e às respostas do formulário. Os profissionais tiveram acesso ao computador cliente (B) enquanto as imagens e laudos estavam armazenadas no computador servidor (A).

Aspectos de usabilidade, velocidade, nitidez, eficiência e segurança foram abordados no questionário, nesta ordem, em questões diretas, com respostas selecionadas por grau de intensidade.

A codificação usada para os graus de intensidade sugere as seguintes respostas possíveis: O = ótimo, B = bom, S = suficiente, R = ruim, P = péssimo.
O questionário emitido em folha avulsa e preenchido se encontra anexado a este documento (BORGES, 2003).

Questões utilizadas: Como classificaria este software segundo:

- 1 - A sua facilidade de utilização, mesmo sem treinamento?
- 2 - Velocidade no aparecimento (visualização) de imagens requisitadas?
- 3 - Nitidez das imagens?
- 4 - Eficiência nas ações disparadas (seleção exames, edição textual, etc)?
- 5 - Segurança das informações?

Resultados obtidos:

Os resultados obtidos, de acordo com os aspectos abordados no formulário, foram registrados na **Tabela 5.3** e resumidos a seguir:

- Usabilidade: conceito B (50%) e conceito S (50%);
- Velocidade: conceito S (50%) e conceito R (50%);
- Nitidez: conceito B (25%) e conceito S (75%);
- Eficiência: conceito B (50%) e conceito S (50%);
- Segurança: conceito B (25%), conceito S (50%) e conceito R (25%).

Tabela 5.3 – Relação de resultados obtidos pelo preenchimento do questionário.

Profissionais	Questão 1	Questão 2	Questão 3	Questão 4	Questão 5
1	S	R	S	S	S
2	S	R	S	S	R
3	B	S	B	B	B
4	B	S	S	B	S

CAPÍTULO 6 – DISCUSSÃO E CONCLUSÃO

6.1 – Discussão

Conforme os fundamentos teóricos estudados, o software satisfez as necessidades de utilização de uma linguagem modular e portável, a linguagem Java, e fez uso de sua codificação em bytecodes como forma de segurança na integridade de dados.

A seleção e permissão seletiva de usuários quanto às funcionalidades oferecidas pelo software foram garantidas pela programação com bloqueios diretos na base de dados (*grants*).

Recursos adicionais de segurança na transmissão ou certificações criptográficas não foram utilizados, apresentando o software, nesta questão, alguma vulnerabilidade.

O uso de chaves criptográficas fortes pode ser um incremental importante a ser desenvolvido, assim como à obediência completa a padrões de uma VPN (Rede Privada Virtual), visto o software ser passível de utilização através de uma rede pública, como a Internet, para comunicação inter-institucional.

Na questão das imagens, o padrão DICOM foi respeitado, podendo haver lentidão na transmissão se o exame tiver um número muito grande de imagens.

Entretanto, o software se mostrou eficiente em seus objetivos pelos testes realizados: *stress*, transmissão, confiabilidade e análise subjetiva.

Algumas funcionalidades foram desenvolvidas baseadas em softwares gratuitos de comunicação de imagens (ImageJ, E-Film e DicomWorks) e aperfeiçoadas de acordo com constatações de necessidades locais.

O software possibilita a gravação de dados (imagens e laudos) em diretórios específicos pessoais, com acesso restrito ao usuário. O usuário que tiver um nível de permissão adequado, após realizar sua primeira busca na base de dados, encontrará, na 'árvore de diretórios', um novo diretório, criado pelo sistema, com nome igual ao do usuário em questão.

O software oferece também a possibilidade de alterações nas imagens com salvamento posterior. Para manter as imagens originais intactas na base de dados, as imagens alteradas só poderão ser salvas nos diretórios pessoais dos usuários.

Por questões legais (o local onde se fez o exame tem o dever de guardá-lo associado ao laudo assinado), o laudo, depois de assinado, automaticamente será salvo remotamente no local de origem das imagens, mesmo que o usuário o salve apenas em seu diretório particular.

Todas as soluções funcionais apresentadas acima inexistem nos softwares gratuitos estudados, apresentando o Diagnosys Viewer as potencialidades futuras de desenvolvimento de customizações amparadas em necessidades reais da rotina radiológica como forma de otimizar os processos.

Desenvolvido totalmente de forma modular, o software pode contemplar outras ações interligadas, como o controle remoto de serviços ou integração com outros sistemas de informações.

A disponibilização de dados na tela (interface usuário), idéia original deste software, se mostra adequada, podendo, no entanto, se tornar mais dinâmica, de acordo com o perfil de cada usuário.

Automatizações de funções e gerações de padrões de rotina, conforme o perfil do usuário, são sugestões de desenvolvimento futuro, baseadas em exemplos de softwares comerciais estudados (Siemens, Philips e GE).

6.2 – Conclusão

A análise de eficiência na realização de tarefas propostas, como transmissão de imagens e gravação remota de dados textuais, mostrou ser o *software* capaz de realizar seus intentos, com segurança e integridade.

O *software* desenvolvido conseguiu ter modularidades bem definidas (conexão, transmissão, abertura de imagens e dados textuais, gravação remota, etc.), retratando possibilidades e facilidades no caminho do desenvolvimento de novos *softwares* de comunicação de imagens biomédicas dentro de uma VPN.

CAPÍTULO 7 – REFERÊNCIAS BIBLIOGRÁFICAS

BORGES, C.L. - **Relatório Departamental do Software Diagnosys Viewer,** DEB/FEEC/UNICAMP, Campinas, 2003.

CAMBRIDGE UNIVERSITY - **Virtual Network Computing**, disponível em <http://www.uk.research.att.com/vnc/index.html>, acesso em 22 de maio de 2003.

CARVALHO, D.B. - **Criptografia: métodos e algoritmos**, Book Express, Rio de Janeiro, 2000.

COMER, D.E. - **Internetworking with TCP/IP**, Prentice Hall, Englewood Cliffs, 1997.

CISCO SYSTEMS - **VPN Model**, disponível em <http://www.cisco.com/warp/public/>, acesso em 22 de abril de 2003.

CLARK, M. – **JUnit Concepts**, disponível em <http://www.junit.org>, acesso em 12 de novembro de 2003.

DEITEL, H.M.; DEITEL, P.J. - **Java - Como Programar**, Bookman, Porto Alegre, 2001.

DICOM HOMEPAGE - **DICOM**, disponível em <http://medical.nema.org/>, acesso em 12 de maio de 2003.

DORASWAMY, N.; HARKINS, D. - **IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks**, Prentice-Hall, New Jersey, 1999.

DREYER, K.J.; MEHTA, A. - **PACS: A Guide to the Digital Revolution**, Springer Verlag, New York, 2001.

DTOOL (Digital Tool) - **VPN Analysis**, disponível em <http://www.dtool.com>, acesso em 20 de março de 2003.

FELIPE Jr., P. - **Introdução ao DICOM. Treinamento PACS**, White Sand Tecnologia e Consultoria, 2003.

FIGUEIREDO, F.J.C.; GEUS, P.L. - **Colocação da VPN na Configuração do Firewall**. III Simpósio sobre Segurança em Informática, 2001, S. José dos Campos, SP. Anais do SSI'01, p. 175-181, 2001.

GEUS, P.L.; REZENDE, E.R.S. - **Análise de Segurança dos Protocolos utilizados para Acesso Remoto VPN em Plataformas Windows**, IV Simpósio sobre Segurança em Informática, Anais do SSI'02, p. 1-8, 2002.

GROSSO, W. - **Java RMI**, O'Reilly, Sebastopol, 2002.

HEISER, J.E. - **An Overview of Software Testing** – IEEE Autotestcon Conference Proceedings, p. 79-82, 1997.

HORSTMANN, C.S.; CORNELL, G. - **Core Java 2**, Sun Microsystems, California, 2000 - Virtual Network Computing, disponível em http://www.uk.research.att.com/vnc/index.html , acesso em 22 de maio de 2003.

KAGAN, R.S. - **Virtual Private Networks - New Strategies for Secure Enterprise Networking** – IEEE WESCON 98 Conference Proceedings, p. 267-272, 1998.

KAMIN, S.N.; MICKUNAS, M.D. - **An Introduction to Computer Science Using Java**, McGraw-Hill, Boston, 1998.

KESSLER, G.C. - An Overview of Cryptography**,** in: **Handbook on Local Area Networks**, Auerbach, 1999.

KOMPELLA, K.; BONICA, R. - **Whither Layer 2 VPNs?**, disponível em <http://www.watersprings.org/links/mlr/id/draft-kb-ppvpn-l2vpn-motiv-00.txt>, acesso em 22 de maio de 2003.

LYON, D.A. - **Image Processing in Java**, Prentice Hall PTR, Upper Saddle River, 1999.

MORAES, A.F. - **Tutorial sobre Qualidade de Serviço**,Tecnologias e Implementação de Redes - SUCESU-SP, 2002.

NAKAMURA, E. T. - **Análise de Segurança do Acesso Remoto VPN**, Anais do XXI Simpósio Brasileiro de Redes de Computadores, p. 35-42, 2000.

NEMA (National Electrical Manufacturer's Association) – **DICOM**, disponível em <http://www.nema.org/index_nema.cfm/694/>, acesso em 12 de maio de 2003.

PIESIGILLI**,** F. – **Ferramentas para Testes Automatizados em Java e C++**. Anais do JavaOne, 2002.

SARIKAYA, B. - **Protocol Test Generation, Trace Analysis and Verification Techniques** - Concordia University, Québec, 1996.

SALOMAA, A. - **Public-key Cryptograph**, Springer-Verlag, Berlin, 1996.

SIEGEL, E.L.; KOLODNER, R. - **Filmless Radiology**, Springer, New York, 2000.

SOUSA, L. - **Conceitos Básicos de TCP/IP**, Tecnologias e Implementação de Redes, SUCESU-SP, 2002.

SUMIMOTO, J.M.; SUZUKY, M. - **Guidelines of Applicability Statements for PPVPNs**. Guidelines VPNs, disponível em <http://www.watersprings.org/links/mlr/id/draft-sumimoto-ppvpn-applicability-guidelines-01.txt>, acesso em 22 de maio de 2003.

SUN MICROSYSTEMS - **JAVA Sun**, disponível em <http://www.java.sun.com>, acesso em 20 de março de 2003.

TANENBAUM, A.S. - **Computer Networks**, Prentice-Hall, Englewood Cliffs, 1989.

ANEXO
ALGUMAS ESTRUTURAS DO PROGRAMA DESENVOLVIDO

Camadas do programa desenvolvido - **Diagnosys Viewer** - e algumas classes básicas funcionais são descritas por meio de figuras e listagens a seguir. Foi utilizado o ambiente de desenvolvimento em Java JBuilder 7. Na **Figura A.1** é mostrada a estrutura em árvore do *software*.

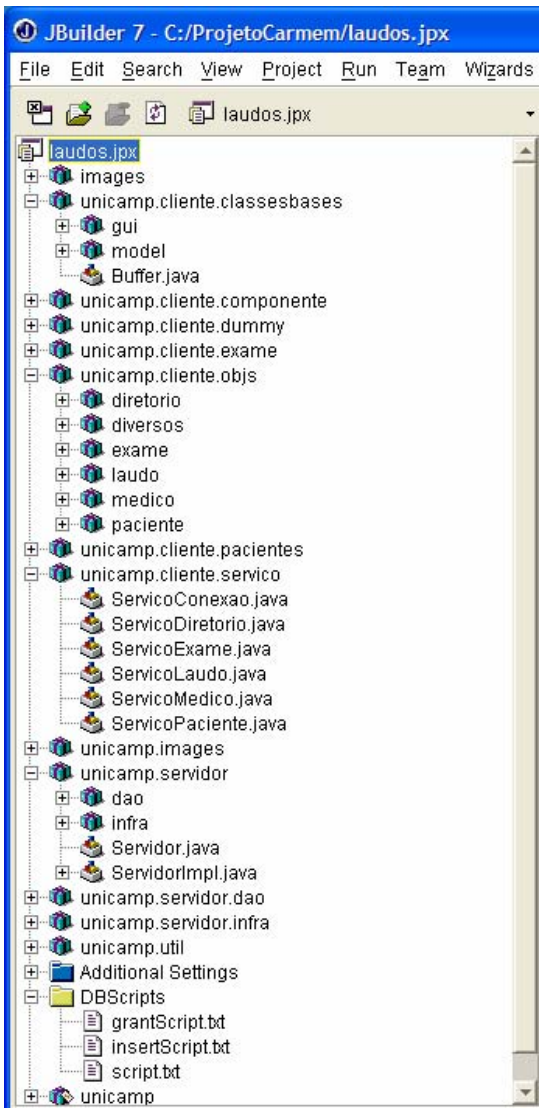

Figura A.1 - Estrutura em árvore do *software*.

As **Figuras A.2 a A.7** são cópias das janelas do desenvolvedor, mostrando as estruturas do código-fonte de algumas das classes implementadas (amostra significativa). Nestas figuras têm-se, nas janelas da esquerda, as estruturas implementadas e o correspondente código-fonte da estrutura sublinhada. De forma similar, as listagens apresentadas são parte de uma amostra. Nem todas as listagens foram mostradas.

Para uma visão completa do código fonte, deve-se consultar o relatório departamental que descreve o *software* desenvolvido (BORGES, 2003).

A escolha da amostra de classes e listagens **(Listagens A.1 e A.2)** apresentadas a seguir se baseou em algumas funcionalidades do *software*:

- O cliente tenta estabelecer uma conexão remota com o servidor (classe *ServicoConexao.java* do pacote *unicamp.cliente.servico*) **(Figura A.2)**;
- O servidor verifica a autenticidade (login e senha – classe *Conexao.java* do pacote *unicamp.servidor.infra)* e a permissibilidade do usuário (restrições de acesso relacionadas diretamente ao banco de dados) **(Figura A.3 e Listagem A.1)**;

Listagem A.1 - Arquivo "Grant.txt" para restrições de acesso a usuários.

```
GRANT SELECT ON *.* TO samuel@epmsp IDENTIFIED BY 'samepmsp' WITH GRANT OPTION;
GRANT SELECT ON *.* TO henrique@epmsp IDENTIFIED BY 'henepmsp' WITH GRANT
OPTION;
GRANT ALL PRIVILEGES ON *.* TO eduardo@unicamp IDENTIFIED BY 'eduuni' WITH GRANT
OPTION;
GRANT ALL PRIVILEGES ON *.* TO sergio@unicamp IDENTIFIED BY 'seruni' WITH GRANT
OPTION;
GRANT ALL PRIVILEGES ON *.* TO vera@unicamp IDENTIFIED BY 'veruni' WITH GRANT
OPTION;
GRANT SELECT ON *.* TO bassani@unicamp IDENTIFIED BY 'basuni' WITH GRANT OPTION.
```

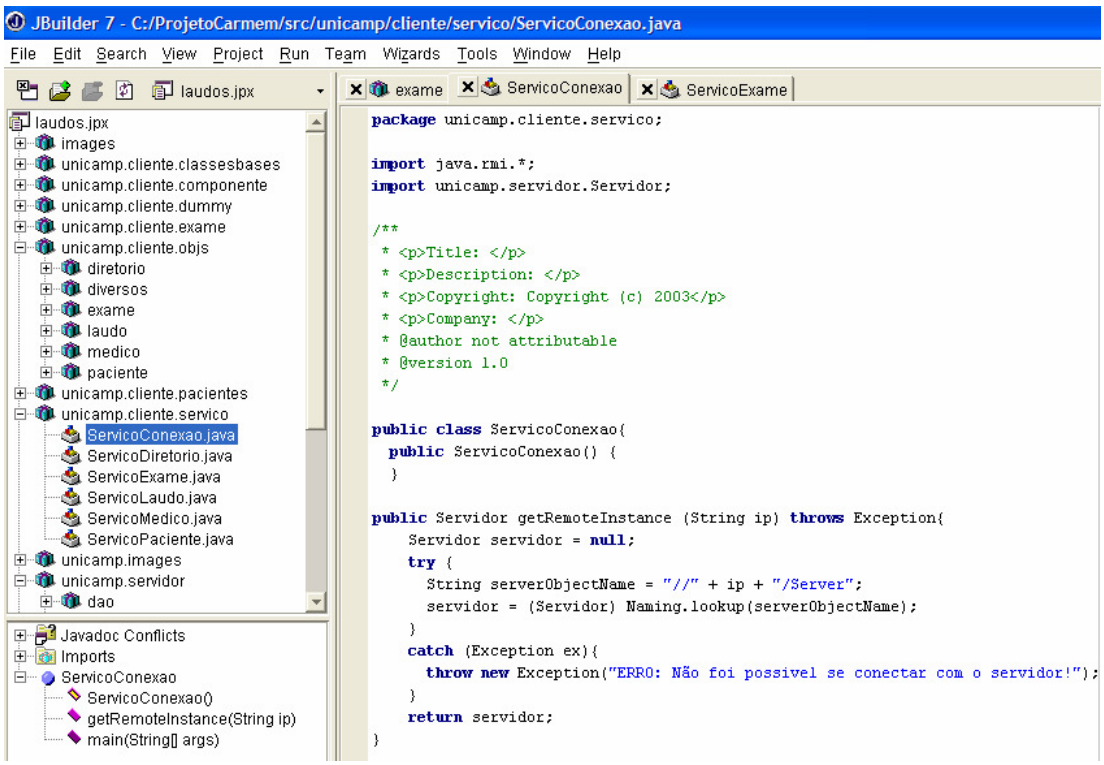

Figura A.2 – Classe *ServicoConexao.java* do pacote *unicamp.cliente.servico.*

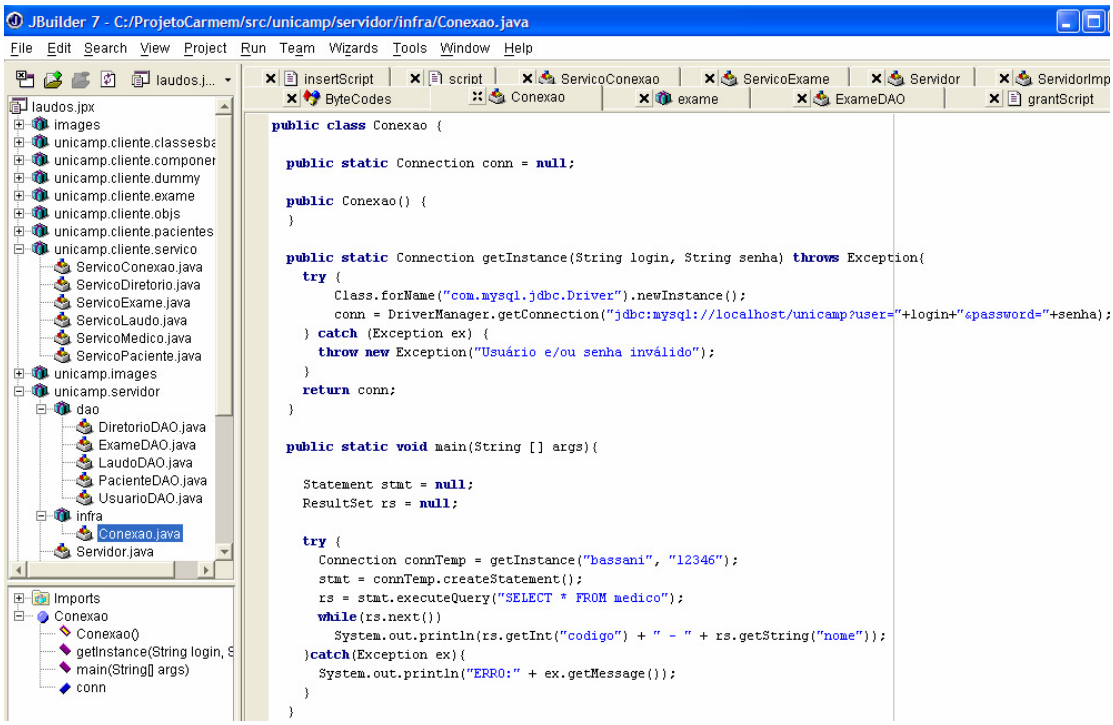

Figura A.3 – Classe *Conexao.java* do pacote *unicamp.servidor.infra.*

- O cliente requisita ao servidor as informações desejadas armazenadas em um banco de dados remoto (classe *ServicoExame.java* do pacote *unicamp.cliente.servico)* **(Figura A.4)**. Podem ser pedidos os exames relacionados a certo paciente, determinado período de tempo ou aqueles oriundos de certo tipo de equipamento de diagnóstico por imagem;

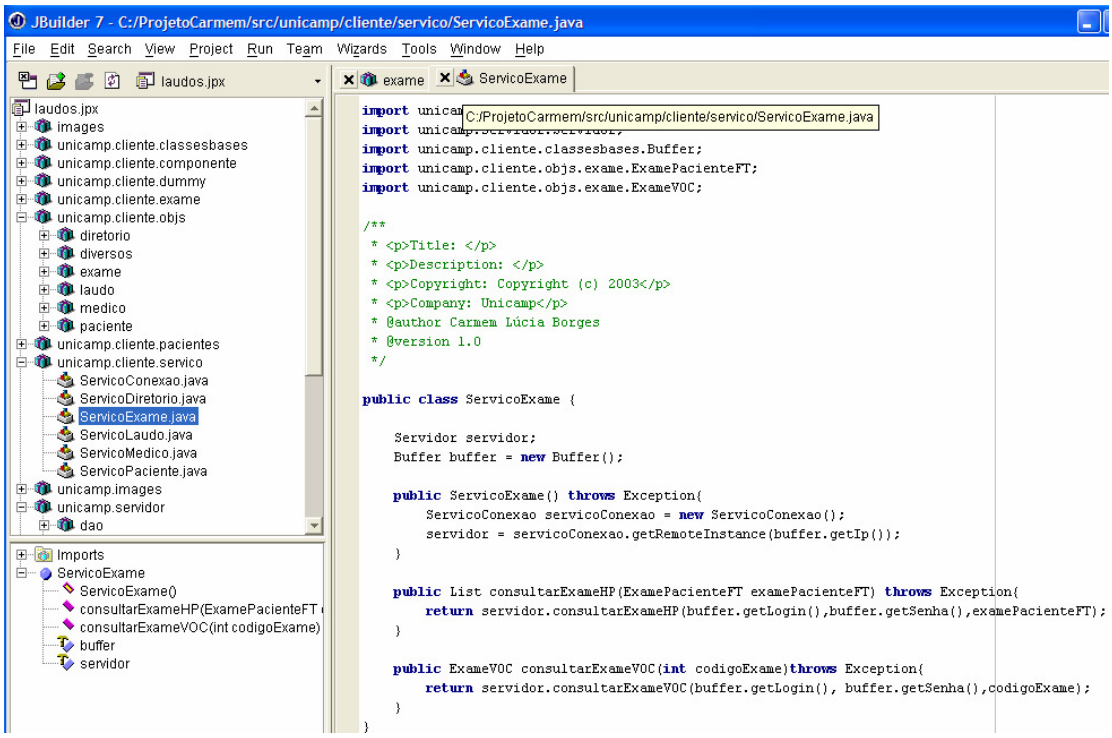

Figura A.4 – Classe *ServicoExame.java* do pacote *unicamp.cliente.servico.*

- Através de uma interface cliente-servidor – controle de regras de negócios (classe *ServidorImpl.java* do pacote *unicamp.servidor)* **(Figura A.5)**, o servidor faz a busca (classe *Servidor.java* do pacote *unicamp.servidor e* classe *ExameDAO.java* do pacote *unicamp.servidor.dao)* **(Figuras A.6 e A.7)** e retorna ao cliente os resultados encontrados;

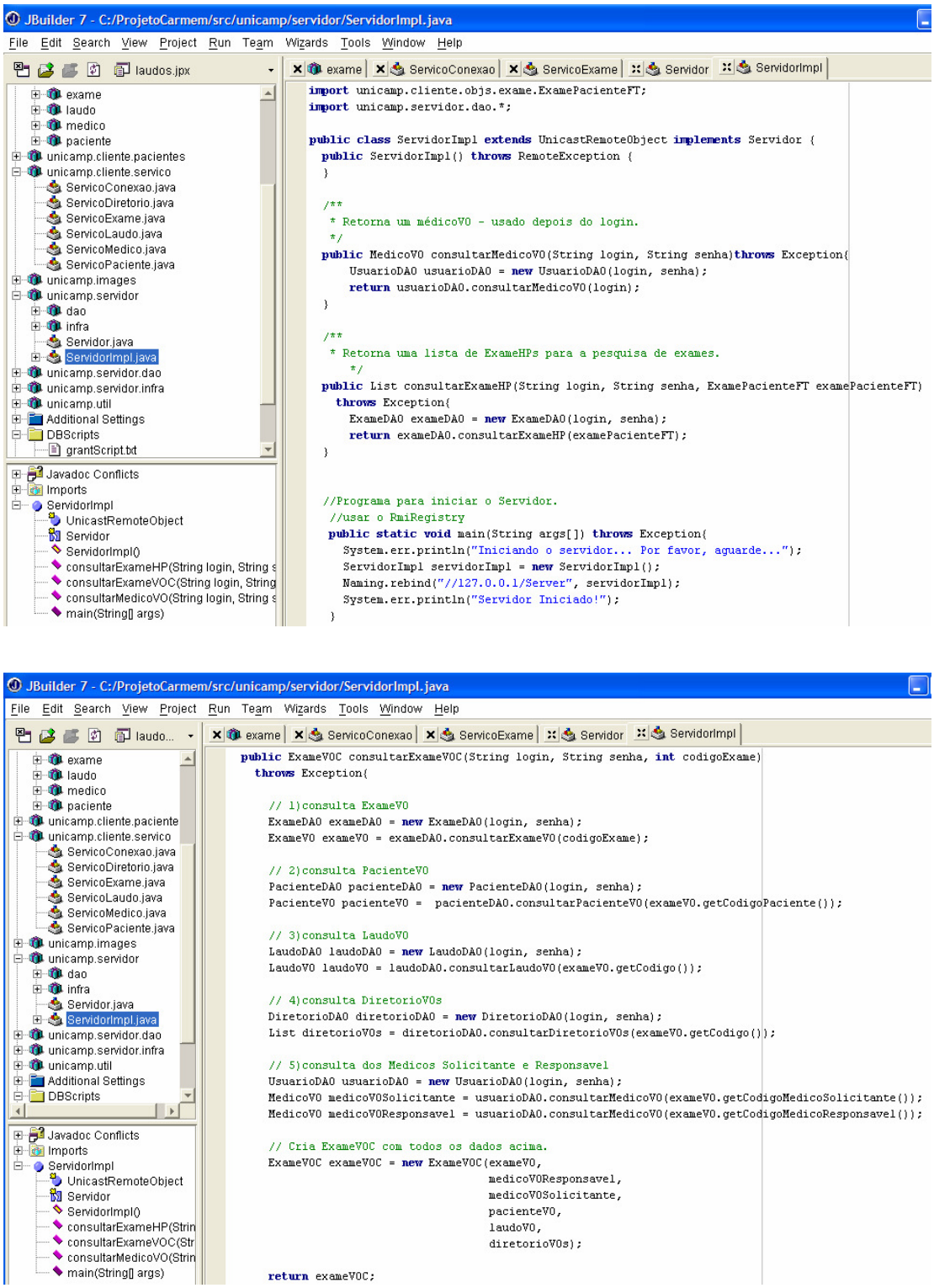

Figura A.5 – Classe *ServidorImpl.java* do pacote *unicamp.servidor.*

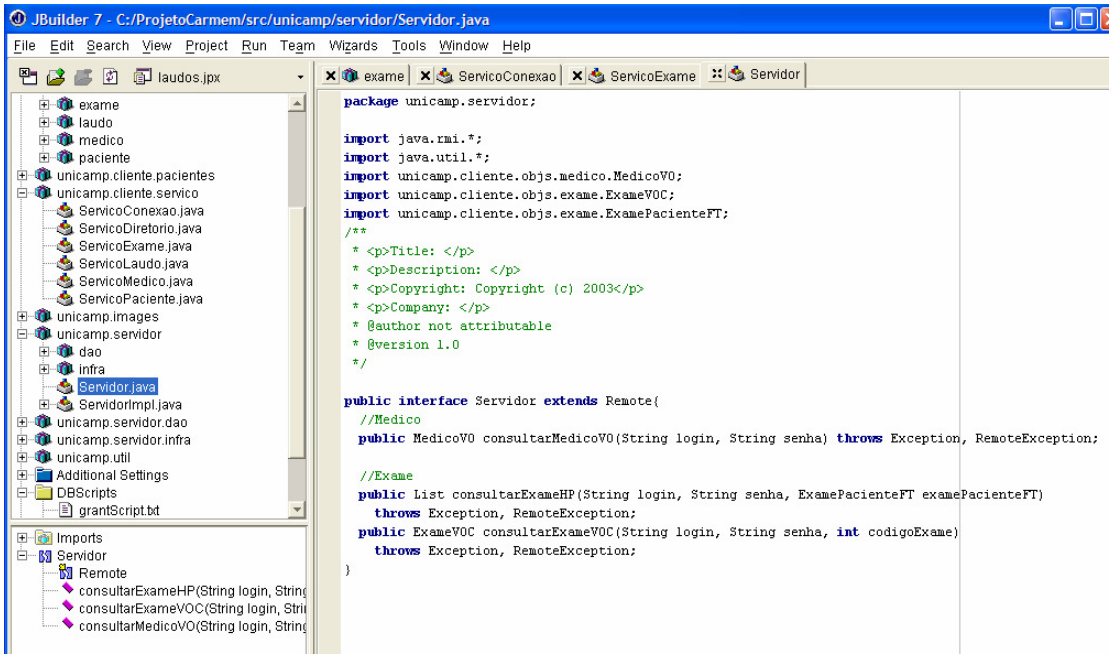

Figura A.6 – Classe *Servidor.java* do pacote *unicamp.servidor*.

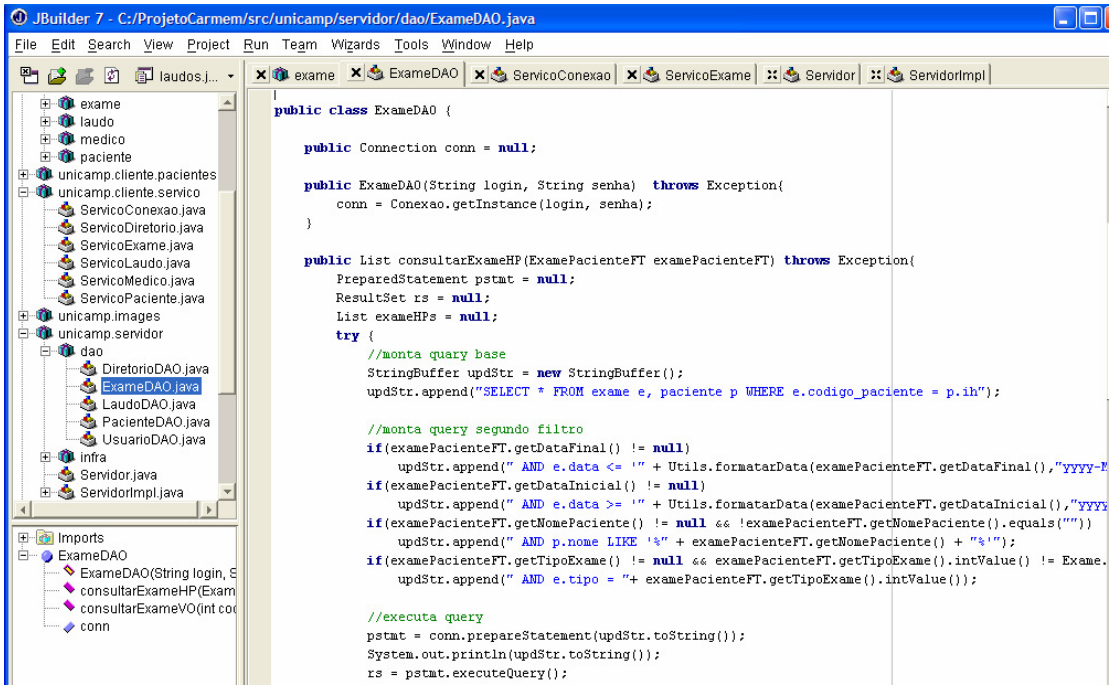

Figura A.7 – Classe *ExameDAO.java* do pacote *unicamp.servidor.dao*.

- O cliente, através de sua interface usuário, monta os resultados em uma tela (interface gráfica do cliente – classe *FramePesquisaExame.java* do pacote *unicamp.cliente.exame.gui*);
- Frente a uma variedade de exames disponibilizados na tela, o usuário escolhe e seleciona o exame desejado, passando este para uma nova tela;
- Na tela, aparecem os dados demográficos do paciente, as informações do exame, as imagens (DICOM ou JPEG) **(Listagem A.2)** relativas ao mesmo (estando a primeira em destaque – tamanho maior) e o espaço para laudo, podendo este já ter sido digitado ou não (classe *PainelResumoPaciente.java* do pacote *unicamp.cliente.classesbases.gui*):
    - A seleção do formato da imagem, original (DICOM) ou comprimida (JPEG com perdas), depende dos privilégios estabelecidos ao usuário que fez a requisição do serviço (ALL PRIVILEGIES ou NOT).

Listagem A.2 - Códigos de leitura DICOM e transformação JPEG aproveitados do *software* ImageJ.

```
public class DICOM extends ImagePlus implements PlugIn {
	public void run(String arg) {
		OpenDialog od = new OpenDialog("Open Dicom...", arg);
		String directory = od.getDirectory();
		String fileName = od.getFileName();
		if (fileName==null)
			return;
		IJ.showStatus("Opening: " + directory + fileName);
		DicomDecoder dd = new DicomDecoder(directory, fileName);
		FileInfo fi = null;
		try {fi = dd.getFileInfo();}
		catch (IOException e) {
			String msg = e.getMessage();
			IJ.showStatus("");
			if (msg.indexOf("EOF")<0) {
				IJ.showMessage("DicomDecoder", msg);
				return;
			} else if (!dd.dicmFound()) {
				msg = "This does not appear to be a valid\n"
				+ "DICOM file. It does not have the\n"
				+ "characters 'DICM' at offset 128.";
				IJ.showMessage("DicomDecoder", msg);
				return; }}
		if (fi!=null && fi.width>0 && fi.height>0 && fi.offset>0) {
			FileOpener fo = new FileOpener(fi);
```

```
                        ImagePlus imp = fo.open(false);
                        if (fi.fileType==FileInfo.GRAY16_SIGNED && imp.getStackSize()==1)
                                convertToUnsigned(imp, fi);
                        if (dd.windowWidth>0.0) {
                                ImageProcessor ip = imp.getProcessor();
                                double min = dd.windowCenter-dd.windowWidth/2;
                                double max = dd.windowCenter+dd.windowWidth/2;
                                if (fi.fileType==FileInfo.GRAY16_SIGNED) {
                                        min += 32768.0;
                                        max += 32768.0; }
                                ip.setMinAndMax(min, max);
                                if (IJ.debugMode) IJ.log("window: "+min+"-"+max); }
                        if (imp.getStackSize()>1)
                                setStack(fileName, imp.getStack());
                        else
                                setProcessor(fileName, imp.getProcessor());
                        setCalibration(imp.getCalibration());
                        setProperty("Info", dd.getDicomInfo());
                setProperty("DicomMap", dd.getMappedDicomInfo());
                        setFileInfo(fi); // needed for revert
                        if (arg.equals("")) show();
                } else
                        IJ.showMessage("DicomDecoder","Unable to decode DICOM header.");
                IJ.showStatus(""); }
        /** Convert 16-bit signed to unsigned if all pixels>=0. */
        void convertToUnsigned(ImagePlus imp, FileInfo fi) {
                ImageProcessor ip = imp.getProcessor();
                short[] pixels = (short[])ip.getPixels();
                int min = Integer.MAX_VALUE;
                int value;
                for (int i=0; i<pixels.length; i++) {
                        value = pixels[i]&0xffff;
                        if (value<min)
                                min = value; }
                if (IJ.debugMode) IJ.log("min: "+(min-32768));
                if (min>=32768) {
                        for (int i=0; i<pixels.length; i++)
                                pixels[i] = (short)(pixels[i]-32768);
                        ip.resetMinAndMax();
                        Calibration cal = imp.getCalibration();
                        cal.setFunction(Calibration.NONE, null, "Gray Value");
                        fi.fileType = FileInfo.GRAY16_UNSIGNED; } } }
class DicomDecoder {
        private static final int PIXEL_REPRESENTATION = 0x00280103;
        private static final int TRANSFER_SYNTAX_UID = 0x00020010;
        private static final int SLICE_SPACING = 0x00180088;
        private static final int SAMPLES_PER_PIXEL = 0x00280002;
        private static final int PHOTOMETRIC_INTERPRETATION = 0x00280004;
        private static final int PLANAR_CONFIGURATION = 0x00280006;
        private static final int NUMBER_OF_FRAMES = 0x00280008;
        private static final int ROWS = 0x00280010;
        private static final int COLUMNS = 0x00280011;
        private static final int PIXEL_SPACING = 0x00280030;
        private static final int BITS_ALLOCATED = 0x00280100;
```

```
        private static final int WINDOW_CENTER = 0x00281050;
        private static final int WINDOW_WIDTH = 0x00281051;
        private static final int RED_PALETTE = 0x00281201;
        private static final int GREEN_PALETTE = 0x00281202;
        private static final int BLUE_PALETTE = 0x00281203;
        private static final int PIXEL_DATA = 0x7FE00010;
        private static final int AE=0x4145, AS=0x4153, AT=0x4154, CS=0x4353, DA=0x4441,
DS=0x4453, DT=0x4454,
                FD=0x4644, FL=0x464C, IS=0x4953, LO=0x4C4F, LT=0x4C54, PN=0x504E,
SH=0x5348, SL=0x534C,
                SS=0x5353, ST=0x5354, TM=0x544D, UI=0x5549, UL=0x554C, US=0x5553,
UT=0x5554,
                OB=0x4F42, OW=0x4F57, SQ=0x5351, UN=0x554E, QQ=0x3F3F;
        private static Properties dictionary;
        private String directory, fileName;
        private static final int ID_OFFSET = 128;  //location of "DICM"
        private static final String DICM = "DICM";
        private BufferedInputStream f;
        private int location = 0;
        private boolean littleEndian = true;
        private int elementLength;
        private int vr;  // Value Representation
        private static final int IMPLICIT_VR = 0x2D2D; // '--'
        private byte[] vrLetters = new byte[2];
        private int previousGroup;
        private StringBuffer dicomInfo = new StringBuffer(1000);
        private boolean dicmFound; // "DICM" found at offset 128
        private boolean oddLocations;  // one or more tags at odd locations
        private boolean bigEndianTransferSyntax = false;
        double windowCenter, windowWidth;
    //mapa que contem a info da imagem de frma arrumada.
    private LinkedHashMap mapDicomInfo = new LinkedHashMap();
    private String valorDicom = "";
        public DicomDecoder(String directory, String fileName) {
                this.directory = directory;
                this.fileName = fileName;
                if (dictionary==null) {
                        DicomDictionary d = new DicomDictionary();
                        dictionary = d.getDictionary(); }
                IJ.register(DICOM.class); }
        String getString(int length) throws IOException {
                byte[] buf = new byte[length];
                int pos = 0;
                while (pos<length) {
                        int count = f.read(buf, pos, length-pos);
                        pos += count; }
                location += length;
                return new String(buf); }
        int getByte() throws IOException {
                int b = f.read();
                if (b ==-1) throw new IOException("unexpected EOF");
                ++location;
                return b; }
        int getShort() throws IOException {
```

```
                int b0 = getByte();
                int b1 = getByte();
                if (littleEndian)
                        return ((b1 << 8) + b0);
                else
                        return ((b0 << 8) + b1); }
        final int getInt() throws IOException {
                int b0 = getByte();
                int b1 = getByte();
                int b2 = getByte();
                int b3 = getByte();
                if (littleEndian)
                        return ((b3<<24) + (b2<<16) + (b1<<8) + b0);
                else
                        return ((b0<<24) + (b1<<16) + (b2<<8) + b3); }
        byte[] getLut(int length) throws IOException {
                if ((length&1)!=0) { // odd
                        String dummy = getString(length);
                        return null; }
                length /= 2;
                byte[] lut = new byte[length];
                for (int i=0; i<length; i++)
                        lut[i] = (byte)(getShort()>>>8);
                return lut; }
        int getLength() throws IOException {
                int b0 = getByte();
                int b1 = getByte();
                int b2 = getByte();
                int b3 = getByte();
                // We cannot know whether the VR is implicit or explicit
                // without the full DICOM Data Dictionary for public and
                // private groups.
                // We will assume the VR is explicit if the two bytes
                // match the known codes. It is possible that these two
                // bytes are part of a 32-bit length for an implicit VR.
                vr = (b0<<8) + b1;
                switch (vr) {
                        case OB: case OW: case SQ: case UN:
                                // Explicit VR with 32-bit length if other two bytes are zero
                                        if ( (b2 == 0) || (b3 == 0) ) return getInt();
                                // Implicit VR with 32-bit length
                                vr = IMPLICIT_VR;
                                if (littleEndian)
                                        return ((b3<<24) + (b2<<16) + (b1<<8) + b0);
                                else
                                        return ((b0<<24) + (b1<<16) + (b2<<8) + b3);
                        case AE: case AS: case AT: case CS: case DA: case DS: case DT:  case
FD:
                        case FL: case IS: case LO: case LT: case PN: case SH: case SL: case SS:
                        case ST: case TM:case UI: case UL: case US: case UT: case QQ:
                                // Explicit vr with 16-bit length
                                if (littleEndian)
                                        return ((b3<<8) + b2);
                                else
```

```
                                        return ((b2<<8) + b3);
                        default:
                                // Implicit VR with 32-bit length...
                                vr = IMPLICIT_VR;
                                if (littleEndian)
                                        return ((b3<<24) + (b2<<16) + (b1<<8) + b0);
                                else
                                        return ((b0<<24) + (b1<<16) + (b2<<8) + b3); } }
int getNextTag() throws IOException {
        int groupWord = getShort();
        if (groupWord==0x0800 && bigEndianTransferSyntax) {
                littleEndian = false;
                groupWord = 0x0008; }
        int elementWord = getShort();
        int tag = groupWord<<16 | elementWord;
        elementLength = getLength();
        // hack needed to read some GE files
        // The element length must be even!
        if (elementLength==13 && !oddLocations) elementLength = 10;
        // "Undefined" element length.
        // This is a sort of bracket that encloses a sequence of elements.
        if (elementLength==-1)
                elementLength = 0;
        return tag; }
FileInfo getFileInfo() throws IOException {
        long skipCount;
boolean isURL = directory.indexOf("://")>0;
        FileInfo fi = new FileInfo();
        int bitsAllocated = 16;
        fi.fileFormat = fi.RAW;
        fi.fileName = fileName;
        if (isURL)
                fi.url = directory;
        else
                fi.directory = directory;
        fi.width = 0;
        fi.height = 0;
        fi.offset = 0;
        fi.intelByteOrder = true;
        fi.fileType = FileInfo.GRAY16_UNSIGNED;
        fi.fileFormat = FileInfo.DICOM;
        int samplesPerPixel = 1;
        int planarConfiguration = 0;
        String photoInterpretation = "";
        if (isURL) {
                URL u = new URL(fi.url+fi.fileName);
                f = new BufferedInputStream(u.openStream());
        } else
                f = new BufferedInputStream(new FileInputStream(directory + fileName));
        if (IJ.debugMode) {
                IJ.log("");
                IJ.log("DicomDecoder: decoding "+fileName); }
        skipCount = (long)ID_OFFSET;
        while (skipCount > 0) skipCount -= f.skip( skipCount );
```

```
		location += ID_OFFSET;
		if (!getString(4).equals(DICM)) {
			f.close();
			if (isURL) {
				URL u = new URL(fi.url+fi.fileName);
				f = new BufferedInputStream(u.openStream());
			} else
				f = new BufferedInputStream(new FileInputStream(directory +
fileName));
			location = 0;
			if (IJ.debugMode) IJ.log(DICM + " not found at offset "+ID_OFFSET+";
reseting to offset 0");
		} else {
			dicmFound = true;
			if (IJ.debugMode) IJ.log(DICM + " found at offset " + ID_OFFSET); }
		boolean inSequence = true;
		boolean decodingTags = true;
		boolean signed = false;
		while (decodingTags) {
			int tag = getNextTag();
			if ((location&1)!=0) // DICOM tags must be at even locations
				oddLocations = true;
			String s;
			switch (tag) {
				case TRANSFER_SYNTAX_UID:
					s = getString(elementLength);
					addInfo(tag, s);
					if (s.indexOf("1.2.4")>-1||s.indexOf("1.2.5")>-1) {
						f.close();
						String msg = "cannot open compressed DICOM
images.\n \n";
						msg += "Transfer Syntax UID = "+s;
						throw new IOException(msg); }
					if (s.indexOf("1.2.840.10008.1.2.2")>=0)
						bigEndianTransferSyntax = true;
					break;
				case NUMBER_OF_FRAMES:
					s = getString(elementLength);
					addInfo(tag, s);
					double frames = s2d(s);
					if (frames>1.0)
						fi.nImages = (int)frames;
					break;
				case SAMPLES_PER_PIXEL:
					samplesPerPixel = getShort();
					addInfo(tag, samplesPerPixel);
					break;
				case PHOTOMETRIC_INTERPRETATION:
					photoInterpretation = getString(elementLength);
					addInfo(tag, photoInterpretation);
					break;
				case PLANAR_CONFIGURATION:
					planarConfiguration = getShort();
					addInfo(tag, planarConfiguration);
```

```
            break;
        case ROWS:
            fi.height = getShort();
            addInfo(tag, fi.height);
            break;
        case COLUMNS:
            fi.width = getShort();
            addInfo(tag, fi.width);
            break;
        case PIXEL_SPACING:
            String scale = getString(elementLength);
            getSpatialScale(fi, scale);
            addInfo(tag, scale);
            break;
        case SLICE_SPACING:
            String spacing = getString(elementLength);
            fi.pixelDepth = s2d(spacing);
            addInfo(tag, spacing);
            break;
        case BITS_ALLOCATED:
            bitsAllocated = getShort();
            if (bitsAllocated==8)
                fi.fileType = FileInfo.GRAY8;
            else if (bitsAllocated==32)
                fi.fileType = FileInfo.GRAY32_UNSIGNED;
            addInfo(tag, bitsAllocated);
            break;
        case PIXEL_REPRESENTATION:
            int pixelRepresentation = getShort();
            if (pixelRepresentation==1) {
                fi.fileType = FileInfo.GRAY16_SIGNED;
                signed = true; }
            addInfo(tag, pixelRepresentation);
            break;
        case WINDOW_CENTER:
            String center = getString(elementLength);
            windowCenter = s2d(center);
            addInfo(tag, center);
            break;
        case WINDOW_WIDTH:
            String width = getString(elementLength);
            windowWidth = s2d(width);
            addInfo(tag, width);
            break;
        case RED_PALETTE:
            fi.reds = getLut(elementLength);
            addInfo(tag, elementLength/2);
            break;
        case GREEN_PALETTE:
            fi.greens = getLut(elementLength);
            addInfo(tag, elementLength/2);
            break;
        case BLUE_PALETTE:
            fi.blues = getLut(elementLength);
```

```
                                addInfo(tag, elementLength/2);
                                break;
                        case PIXEL_DATA:
                                // Start of image data...
                                if (elementLength!=0) {
                                        fi.offset = location;
                                        addInfo(tag, location);
                                        decodingTags = false;
                                } else
                                        addInfo(tag, null);
                                break;
                        case 0x7F880010:
                                // What is this? - RAK
                                if (elementLength!=0) {
                                        fi.offset = location+4;
                                        decodingTags = false; }
                                break;
                        default:
                                // Not used, skip over it...
                                addInfo(tag, null); }
                } // while(decodingTags)
                if (fi.fileType==FileInfo.GRAY8) {
                        if (fi.reds!=null && fi.greens!=null && fi.blues!=null
                        && fi.reds.length==fi.greens.length
                        && fi.reds.length==fi.blues.length) {
                                fi.fileType = FileInfo.COLOR8;
                                fi.lutSize = fi.reds.length; } }
                if (fi.fileType==FileInfo.GRAY32_UNSIGNED && signed)
                        fi.fileType = FileInfo.GRAY32_INT;
                if (samplesPerPixel==3 && photoInterpretation.startsWith("RGB")) {
                        if (planarConfiguration==0)
                                fi.fileType = FileInfo.RGB;
                        else if (planarConfiguration==1)
                                fi.fileType = FileInfo.RGB_PLANAR;
                } else if (photoInterpretation.endsWith("1 "))
                                fi.whiteIsZero = true;
                if (!littleEndian)
                        fi.intelByteOrder = false;
                if (IJ.debugMode) {
                        IJ.log("width: " + fi.width);
                        IJ.log("height: " + fi.height);
                        IJ.log("images: " + fi.nImages);
                        IJ.log("bits allocated: " + bitsAllocated);
                        IJ.log("offset: " + fi.offset); }
                f.close();
                return fi; }
        String getDicomInfo() {
                return new String(dicomInfo); }
    //obtem a informacao exata DICOM atraves de um chave do Dicionario
    public String getMappedDicomInfo(int key) {
           return (String) mapDicomInfo.get(new Integer(key)); }
    public Map getMappedDicomInfo() {
          return mapDicomInfo; }
        void addInfo(int tag, String value) throws IOException {
```

```
                String info = getHeaderInfo(tag, value);
                if (info!=null) {
                        int group = tag>>>16;
                        if (group!=previousGroup) dicomInfo.append("\n");
                        previousGroup = group;
                        dicomInfo.append(tag2hex(tag)+info+"\n");
                //mapeamento das informacoes Dicom.
                mapDicomInfo.put(new Integer(tag), valorDicom);
                //System.out.println(valorDicom + " " + tag + "---" + tag2hex(tag)+info); }
                if (IJ.debugMode) {
                        if (info==null) info = "";
                        vrLetters[0] = (byte)(vr >> 8);
                        vrLetters[1] = (byte)(vr & 0xFF);
                        String VR = new String(vrLetters);
                        IJ.log("(" + tag2hex(tag) + VR
                        + " " + elementLength
                        + " bytes from "
                        + (location-elementLength)+") "
                        + info); } }
        void addInfo(int tag, int value) throws IOException {
                addInfo(tag, Integer.toString(value)); }
        String getHeaderInfo(int tag, String value) throws IOException {
                String key = i2hex(tag);
                //while (key.length()<8)
                //        key = '0' + key;
                String id = (String)dictionary.get(key);
                if (id!=null) {
                        if (vr==IMPLICIT_VR && id!=null)
                                vr = (id.charAt(0)<<8) + id.charAt(1);
                        id = id.substring(2); }
                if (value!=null){
              valorDicom = value;
              return id+": "+value; }
            switch (vr) {
                        case AE: case AS: case AT: case CS: case DA: case DS: case DT:  case
IS: case LO:
                        case LT: case PN: case SH: case ST: case TM: case UI:
                                value = getString(elementLength);
                                break;
                        case US:
                                if (elementLength==2)
                                        value = Integer.toString(getShort());
                                else {
                                        value = "";
                                        int n = elementLength/2;
                                        for (int i=0; i<n; i++)
                                                value += Integer.toString(getShort())+" "; }
                                break;
                        default:
                                long skipCount = (long)elementLength;
                                while (skipCount > 0) skipCount -= f.skip(skipCount);
                                location += elementLength;
                                value = ""; }
                if (id==null){
```

```
            valorDicom = null;
            return null;
          }else{
            valorDicom = value;
            return id+": "+value; } }
        static char[] buf8 = new char[8];
        /** Converts an int to an 8 byte hex string. */
        String i2hex(int i) {
                for (int pos=7; pos>=0; pos--) {
                        buf8[pos] = Tools.hexDigits[i&0xf];
                        i >>>= 4; }
                return new String(buf8); }
        char[] buf10;
        String tag2hex(int tag) {
                if (buf10==null) {
                        buf10 = new char[11];
                        buf10[4] = ',';
                        buf10[9] = ' '; }
                int pos = 8;
                while (pos>=0) {
                        buf10[pos] = Tools.hexDigits[tag&0xf];
                        tag >>>= 4;
                        pos--;
                        if (pos==4) pos--; // skip coma }
                return new String(buf10); }
        double s2d(String s) {
                Double d;
                try {d = new Double(s);}
                catch (NumberFormatException e) {d = null;}
                if (d!=null)
                        return(d.doubleValue());
                else
                        return(0.0); }
        void getSpatialScale(FileInfo fi, String scale) {
                double xscale=0, yscale=0;
                int i = scale.indexOf('\\');
                if (i>0) {
                        xscale = s2d(scale.substring(0, i));
                        yscale = s2d(scale.substring(i+1)); }
                if (xscale!=0.0 && yscale!=0.0) {
                        fi.pixelWidth = xscale;
                        fi.pixelHeight = yscale;
                        fi.unit = "mm"; } }
        boolean dicmFound() {
                return dicmFound; } }
class DicomDictionary {
        Properties getDictionary() {
                Properties p = new Properties();
                for (int i=0; i<dict.length; i++) {
                        p.put(dict[i].substring(0,8), dict[i].substring(9)); }
                return p; }
        String[] dict = {
                //"00020000=ULFile Meta Elements Group Len",
                //"00020001=OBFile Meta Info Version",
```

"00020002=UIMedia Storage SOP Class UID",
"00020003=UIMedia Storage SOP Inst UID",
"00020010=UITransfer Syntax UID",
"00080005=CSSpecific Character Set",
"00080008=CSImage Type",
"00080012=DAInstance Creation Date",
"00080013=TMInstance Creation Time",
"00080014=UIInstance Creator UID",
"00080016=UISOP Class UID",
"00080018=UISOP Instance UID",
"00080020=DAStudy Date",
"00080021=DASeries Date",
"00080022=DAAcquisition Date",
"00080023=DAImage Date",
"00080024=DAOverlay Date",
"00080025=DACurve Date",
"00080030=TMStudy Time",
"00080031=TMSeries Time",
"00080032=TMAcquisition Time",
"00080033=TMImage Time",
"00080034=TMOverlay Time",
"00080035=TMCurve Time",
"00080042=CSNuclear Medicine Series Type",
"00080050=SHAccession Number",
"00080052=CSQuery/Retrieve Level",
"00080054=AERetrieve AE Title",
"00080058=AEFailed SOP Instance UID List",
"00080060=CSModality",
"00080064=CSConversion Type",
"00080070=LOManufacturer",
"00080080=LOInstitution Name",
"00080081=STInstitution Address",
"00080082=SQInstitution Code Sequence",
"00080090=PNReferring Physician's Name",
"00080092=STReferring Physician's Address",
"00080094=SHReferring Physician's Telephone Numbers",
"00080100=SHCode Value",
"00080102=SHCoding Scheme Designator",
"00080104=LOCode Meaning",
"00081010=SHStation Name",
"00081030=LOStudy Description",
"00081032=SQProcedure Code Sequence",
"0008103E=LOSeries Description",
"00081040=LOInstitutional Department Name",
"00081050=PNAttending Physician's Name",
"00081060=PNName of Physician(s) Reading Study",
"00081070=PNOperator's Name",
"00081080=LOAdmitting Diagnoses Description",
"00081084=SQAdmitting Diagnosis Code Sequence",
"00081090=LOManufacturer's Model Name",
"00081100=SQReferenced Results Sequence",
"00081110=SQReferenced Study Sequence",
"00081111=SQReferenced Study Component Sequence",
"00081115=SQReferenced Series Sequence",

"00081120=SQReferenced Patient Sequence",
"00081125=SQReferenced Visit Sequence",
"00081130=SQReferenced Overlay Sequence",
"00081140=SQReferenced Image Sequence",
"00081145=SQReferenced Curve Sequence",
"00081150=UIReferenced SOP Class UID",
"00081155=UIReferenced SOP Instance UID",
"00082111=STDerivation Description",
"00082112=SQSource Image Sequence",
"00082120=SHStage Name",
"00082122=ISStage Number",
"00082124=ISNumber of Stages",
"00082129=ISNumber of Event Timers",
"00082128=ISView Number",
"0008212A=ISNumber of Views in Stage",
"00082130=DSEvent Elapsed Time(s)",
"00082132=LOEvent Timer Name(s)",
"00082142=ISStart Trim",
"00082143=ISStop Trim",
"00082144=ISRecommended Display Frame Rate",
"00082200=CSTransducer Position",
"00082204=CSTransducer Orientation",
"00082208=CSAnatomic Structure",
"00100010=PNPatient's Name",
"00100020=LOPatient ID",
"00100021=LOIssuer of Patient ID",
"00100030=DAPatient's Birth Date",
"00100032=TMPatient's Birth Time",
"00100040=CSPatient's Sex",
"00101000=LOOther Patient IDs",
"00101001=PNOther Patient Names",
"00101005=PNPatient's Maiden Name",
"00101010=ASPatient's Age",
"00101020=DSPatient's Size",
"00101030=DSPatient's Weight",
"00101040=LOPatient's Address",
"00102150=LOCountry of Residence",
"00102152=LORegion of Residence",
"00102180=SHOccupation",
"001021A0=CSSmoking Status",
"001021B0=LTAdditional Patient History",
"00104000=LTPatient Comments",
"00180010=LOContrast/Bolus Agent",
"00180015=CSBody Part Examined",
"00180020=CSScanning Sequence",
"00180021=CSSequence Variant",
"00180022=CSScan Options",
"00180023=CSMR Acquisition Type",
"00180024=SHSequence Name",
"00180025=CSAngio Flag",
"00180030=LORadionuclide",
"00180031=LORadiopharmaceutical",
"00180032=DSEnergy Window Centerline",
"00180033=DSEnergy Window Total Width",

"00180034=LOIntervention Drug Name",
"00180035=TMIntervention Drug Start Time",
"00180040=ISCine Rate",
"00180050=DSSlice Thickness",
"00180060=DSkVp",
"00180070=ISCounts Accumulated",
"00180071=CSAcquisition Termination Condition",
"00180072=DSEffective Series Duration",
"00180080=DSRepetition Time",
"00180081=DSEcho Time",
"00180082=DSInversion Time",
"00180083=DSNumber of Averages",
"00180084=DSImaging Frequency",
"00180085=SHImaged Nucleus",
"00180086=ISEcho Numbers(s)",
"00180087=DSMagnetic Field Strength",
"00180088=DSSpacing Between Slices",
"00180089=ISNumber of Phase Encoding Steps",
"00180090=DSData Collection Diameter",
"00180091=ISEcho Train Length",
"00180093=DSPercent Sampling",
"00180094=DSPercent Phase Field of View",
"00180095=DSPixel Bandwidth",
"00181000=LODevice Serial Number",
"00181004=LOPlate ID",
"00181010=LOSecondary Capture Device ID",
"00181012=DADate of Secondary Capture",
"00181014=TMTime of Secondary Capture",
"00181016=LOSecondary Capture Device Manufacturer",
"00181018=LOSecondary Capture Device Manufacturer's Model Name",
"00181019=LOSecondary Capture Device Software Version(s)",
"00181020=LOSoftware Versions(s)",
"00181022=SHVideo Image Format Acquired",
"00181023=LODigital Image Format Acquired",
"00181030=LOProtocol Name",
"00181040=LOContrast/Bolus Route",
"00181041=DSContrast/Bolus Volume",
"00181042=TMContrast/Bolus Start Time",
"00181043=TMContrast/Bolus Stop Time",
"00181044=DSContrast/Bolus Total Dose",
"00181045=ISSyringe Counts",
"00181050=DSSpatial Resolution",
"00181060=DSTrigger Time",
"00181061=LOTrigger Source or Type",
"00181062=ISNominal Interval",
"00181063=DSFrame Time",
"00181064=LOFraming Type",
"00181065=DSFrame Time Vector",
"00181066=DSFrame Delay",
"00181070=LORadionuclide Route",
"00181071=DSRadionuclide Volume",
"00181072=TMRadionuclide Start Time",
"00181073=TMRadionuclide Stop Time",
"00181074=DSRadionuclide Total Dose",

"00181080=CSBeat Rejection Flag",
"00181081=ISLow R-R Value",
"00181082=ISHigh R-R Value",
"00181083=ISIntervals Acquired",
"00181084=ISIntervals Rejected",
"00181085=LOPVC Rejection",
"00181086=ISSkip Beats",
"00181088=ISHeart Rate",
"00181090=ISCardiac Number of Images",
"00181094=ISTrigger Window",
"00181100=DSReconstruction Diameter",
"00181110=DSDistance Source to Detector",
"00181111=DSDistance Source to Patient",
"00181120=DSGantry/Detector Tilt",
"00181130=DSTable Height",
"00181131=DSTable Traverse",
"00181140=CSRotation Direction",
"00181141=DSAngular Position",
"00181142=DSRadial Position",
"00181143=DSScan Arc",
"00181144=DSAngular Step",
"00181145=DSCenter of Rotation Offset",
"00181146=DSRotation Offset",
"00181147=CSField of View Shape",
"00181149=ISField of View Dimensions(s)",
"00181150=ISExposure Time",
"00181151=ISX-ray Tube Current",
"00181152=ISExposure",
"00181153=ISExposure in uAs",
"00181154=DSAverage Pulse Width",
"00181155=CSRadiation Setting",
"00181156=CSRectification Type",
"0018115A=CSRadiation Mode",
"0018115E=DSImage Area Dose Product",
"00181160=SHFilter Type",
"00181161=LOType of Filters",
"00181162=DSIntensifier Size",
"00181164=DSImager Pixel Spacing",
"00181166=CSGrid",
"00181170=ISGenerator Power",
"00181180=SHCollimator/grid Name",
"00181181=CSCollimator Type",
"00181182=ISFocal Distance",
"00181183=DSX Focus Center",
"00181184=DSY Focus Center",
"00181190=DSFocal Spot(s)",
"00181191=CSAnode Target Material",
"001811A0=DSBody Part Thickness",
"001811A2=DSCompression Force",
"00181200=DADate of Last Calibration",
"00181201=TMTime of Last Calibration",
"00181210=SHConvolution Kernel",
"00181242=ISActual Frame Duration",
"00181243=ISCount Rate",

"00181250=SHReceiving Coil",
"00181251=SHTransmitting Coil",
"00181260=SHPlate Type",
"00181261=LOPhosphor Type",
"00181300=ISScan Velocity",
"00181301=CSWhole Body Technique",
"00181302=ISScan Length",
"00181310=USAcquisition Matrix",
"00181312=CSPhase Encoding Direction",
"00181314=DSFlip Angle",
"00181315=CSVariable Flip Angle Flag",
"00181316=DSSAR",
"00181318=DSdB/dt",
"00181400=LOAcquisition Device Processing Description",
"00181401=LOAcquisition Device Processing Code",
"00181402=CSCassette Orientation",
"00181403=CSCassette Size",
"00181404=USExposures on Plate",
"00181405=ISRelative X-ray Exposure",
"00181450=CSColumn Angulation",
"00181500=CSPositioner Motion",
"00181508=CSPositioner Type",
"00181510=DSPositioner Primary Angle",
"00181511=DSPositioner Secondary Angle",
"00181520=DSPositioner Primary Angle Increment",
"00181521=DSPositioner Secondary Angle Increment",
"00181530=DSDetector Primary Angle",
"00181531=DSDetector Secondary Angle",
"00181600=CSShutter Shape",
"00181602=ISShutter Left Vertical Edge",
"00181604=ISShutter Right Vertical Edge",
"00181606=ISShutter Upper Horizontal Edge",
"00181608=ISShutter Lower Horizontal Edge",
"00181610=ISCenter of Circular Shutter",
"00181612=ISRadius of Circular Shutter",
"00181620=ISVertices of the Polygonal Shutter",
"00181700=ISCollimator Shape",
"00181702=ISCollimator Left Vertical Edge",
"00181704=ISCollimator Right Vertical Edge",
"00181706=ISCollimator Upper Horizontal Edge",
"00181708=ISCollimator Lower Horizontal Edge",
"00181710=ISCenter of Circular Collimator",
"00181712=ISRadius of Circular Collimator",
"00181720=ISVertices of the Polygonal Collimator",
"00185000=SHOutput Power",
"00185010=LOTransducer Data",
"00185012=DSFocus Depth",
"00185020=LOPreprocessing Function",
"00185021=LOPostprocessing Function",
"00185022=DSMechanical Index",
"00185024=DSThermal Index",
"00185026=DSCranial Thermal Index",
"00185027=DSSoft Tissue Thermal Index",
"00185028=DSSoft Tissue-focus Thermal Index",

"00185029=DSSoft Tissue-surface Thermal Index",
"00185050=ISDepth of Scan Field",
"00185100=CSPatient Position",
"00185101=CSView Position",
"00185104=SQProjection Eponymous Name Code Sequence",
"00185210=DSImage Transformation Matrix",
"00185212=DSImage Translation Vector",
"00186000=DSSensitivity",
"00186011=SQSequence of Ultrasound Regions",
"00186012=USRegion Spatial Format",
"00186014=USRegion Data Type",
"00186016=ULRegion Flags",
"00186018=ULRegion Location Min X0",
"0018601A=ULRegion Location Min Y0",
"0018601C=ULRegion Location Max X1",
"0018601E=ULRegion Location Max Y1",
"00186020=SLReference Pixel X0",
"00186022=SLReference Pixel Y0",
"00186024=USPhysical Units X Direction",
"00186026=USPhysical Units Y Direction",
"00181628=FDReference Pixel Physical Value X",
"0018602A=FDReference Pixel Physical Value Y",
"0018602C=FDPhysical Delta X",
"0018602E=FDPhysical Delta Y",
"00186030=ULTransducer Frequency",
"00186031=CSTransducer Type",
"00186032=ULPulse Repetition Frequency",
"00186034=FDDoppler Correction Angle",
"00186036=FDSterring Angle",
"00186038=ULDoppler Sample Volume X Position",
"0018603A=ULDoppler Sample Volume Y Position",
"0018603C=ULTM-Line Position X0",
"0018603E=ULTM-Line Position Y0",
"00186040=ULTM-Line Position X1",
"00186042=ULTM-Line Position Y1",
"00186044=USPixel Component Organization",
"00186046=ULPixel Component Mask",
"00186048=ULPixel Component Range Start",
"0018604A=ULPixel Component Range Stop",
"0018604C=USPixel Component Physical Units",
"0018604E=USPixel Component Data Type",
"00186050=ULNumber of Table Break Points",
"00186052=ULTable of X Break Points",
"00186054=FDTable of Y Break Points",
"00186056=ULNumber of Table Entries",
"00186058=ULTable of Pixel Values",
"0018605A=ULTable of Parameter Values",
"00187000=CSDetector Conditions Nominal Flag",
"00187001=DSDetector Temperature",
"00187004=CSDetector Type",
"00187005=CSDetector Configuration",
"00187006=LTDetector Description",
"00187008=LTDetector Mode",
"0018700A=SHDetector ID",

"0018700C=DADate of Last Detector Calibration",
"0018700E=TMTime of Last Detector Calibration",
"00187010=ISExposures on Detector Since Last Calibration",
"00187011=ISExposures on Detector Since Manufactured",
"00187012=DSDetector Time Since Last Exposure",
"00187014=DSDetector Active Time",
"00187016=DSDetector Activation Offset From Exposure",
"0018701A=DSDetector Binning",
"00187020=DSDetector Element Physical Size",
"00187022=DSDetector Element Spacing",
"00187024=CSDetector Active Shape",
"00187026=DSDetector Active Dimension(s)",
"00187028=DSDetector Active Origin",
"00187030=DSField of View Origin",
"00187032=DSField of View Rotation",
"00187034=CSField of View Horizontal Flip",
"00187040=LTGrid Absorbing Material",
"00187041=LTGrid Spacing Material",
"00187042=DSGrid Thickness",
"00187044=DSGrid Pitch",
"00187046=ISGrid Aspect Ratio",
"00187048=DSGrid Period",
"0018704C=DSGrid Focal Distance",
"00187050=LTFilter Material LT",
"00187052=DSFilter Thickness Minimum",
"00187054=DSFilter Thickness Maximum",
"00187060=CSExposure Control Mode",
"00187062=LTExposure Control Mode Description",
"00187064=CSExposure Status",
"00187065=DSPhototimer Setting",
"0020000D=UIStudy Instance UID",
"0020000E=UISeries Instance UID",
"00200010=SHStudy ID",
"00200011=ISSeries Number",
"00200012=ISAcquisition Number",
"00200013=ISImage Number",
"00200014=ISIsotope Number",
"00200015=ISPhase Number",
"00200016=ISInterval Number",
"00200017=ISTime Slot Number",
"00200018=ISAngle Number",
"00200020=CSPatient Orientation",
"00200022=USOverlay Number",
"00200024=USCurve Number",
"00200032=DSImage Position (Patient)",
"00200037=DSImage Orientation (Patient)",
"00200052=UIFrame of Reference UID",
"00200060=CSLaterality",
"00200080=UIMasking Image UID",
"00200100=ISTemporal Position Identifier",
"00200105=ISNumber of Temporal Positions",
"00200110=DSTemporal Resolution",
"00201000=ISSeries in Study",
"00201002=ISImages in Acquisition",

```
                "00201004=ISAcquisition in Study",
                "00201040=LOPosition Reference Indicator",
                "00201041=DSSlice Location",
                "00201070=ISOther Study Numbers",
                "00201200=ISNumber of Patient Related Studies",
                "00201202=ISNumber of Patient Related Series",
                "00201204=ISNumber of Patient Related Images",
                "00201206=ISNumber of Study Related Series",
                "00201208=ISNumber of Study Related Images",
                "00204000=LTImage Comments",
                "00280002=USSamples per Pixel",
                "00280004=CSPhotometric Interpretation",
                "00280006=USPlanar Configuration",
                "00280008=ISNumber of Frames",
                "00280009=ATFrame Increment Pointer",
                "00280010=USRows",
                "00280011=USColumns",
                "00280030=DSPixel Spacing",
                "00280031=DSZoom Factor",
                "00280032=DSZoom Center",
                "00280034=ISPixel Aspect Ratio",
                "00280051=CSCorrected Image",
                "00280100=USBits Allocated",
                "00280101=USBits Stored",
                "00280102=USHigh Bit",
                "00280103=USPixel Representation",
                "00280106=USSmallest Image Pixel Value",
                "00280107=USLargest Image Pixel Value",
                "00280108=USSmallest Pixel Value in Series",
                "00280109=USLargest Pixel Value in Series",
                "00280120=USPixel Padding Value",
                "00281050=DSWindow Center",
                "00281051=DSWindow Width",
                "00281052=DSRescale Intercept",
                "00281053=DSRescale Slope",
                "00281054=LORescale Type",
                "00281055=LOWindow Center & Width Explanation",
                "00281101=USRed Palette Color Lookup Table Descriptor",
                "00281102=USGreen Palette Color Lookup Table Descriptor",
                "00281103=USBlue Palette Color Lookup Table Descriptor",
                "00281201=USRed Palette Color Lookup Table Data",
                "00281202=USGreen Palette Color Lookup Table Data",
                "00281203=USBlue Palette Color Lookup Table Data",
                "00283000=SQModality LUT Sequence",
                "00283002=USLUT Descriptor",
                "00283003=LOLUT Explanation",
                "00283004=LOMadality LUT Type",
                "00283006=USLUT Data",
                "00283010=SQVOI LUT Sequence",
                "7FE00010=OXPixel Data",
                "FFFEE000=DLItem",
                "FFFEE00D=DLItem Delimitation Item",
                "FFFEE0DD=DLSequence Delimitation Item" };
}
```

```
import ij.*;
import ij.process.*;
import ij.gui.*;
import ij.io.*;
import com.sun.image.codec.jpeg.*;
import java.awt.image.*;
import java.awt.*;
import java.io.*;
public class JpegWriter implements PlugIn {
        public static final int DEFAULT_QUALITY = 75;
        private static int quality;
   static {setQuality(ij.Prefs.getInt(ij.Prefs.JPEG, DEFAULT_QUALITY));}
   public void run(String arg) {
      ImagePlus imp = WindowManager.getCurrentImage();
      if (imp==null)
          return;
      imp.startTiming();
      saveAsJpeg(imp,arg);
      IJ.showTime(imp, imp.getStartTime(), "JpegWriter: "); }
   public void saveAsJpeg(ImagePlus imp, String path) {
      //IJ.log("saveAsJpeg: "+path);
      int width = imp.getWidth();
      int height = imp.getHeight();
      BufferedImage   bi = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
      try {
         FileOutputStream  f  = new FileOutputStream(path);
         Graphics g = bi.createGraphics();
         g.drawImage(imp.getImage(), 0, 0, null);
         g.dispose();
         JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(f);
         JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(bi);
         param.setQuality((float)(quality/100.0), true);
         encoder.encode(bi, param);
         f.close(); }
      catch (Exception e) {
        IJ.showMessage("Jpeg Writer", ""+e); }  }
         /** Specifies the image quality (0-100). 0 is poorest image quality,
                highest compression, and 100 is best image quality, lowest compression. */
   public static void setQuality(int jpegQuality) {
      quality = jpegQuality;
         if (quality<0) quality = 0;
         if (quality>100) quality = 100; }
   public static int getQuality() {
      return quality;  } }
```

As **Listagens A.3 e A.4** (codificação e tabelas) foram referenciadas em capítulos anteriores e são apresentadas a seguir:

Listagem A.3 - Codificação em bytecodes

```
public static String criptografar(String texto) {
```

```
//Criptografar textos
return Integer.toString(texto.hashCode());
MessageDigest digest;
try {
   digest = MessageDigest.getInstance("SHA-1");
}catch (NoSuchAlgorithmException e) {
   throw new RuntimeException("Falha na criptografia!"); }
//digest texto usuario...
digest.update(texto.getBytes());
byte[] textoDigest = digest.digest();
return new String(textoDigest); }
```

Listagem A.4 – Tabelas dos objetos do *software*.

```
CREATE TABLE Paciente
(
ih INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR (50),
sexo INT,
logradouro VARCHAR (100),
numero VARCHAR (5),
complemento VARCHAR (5),
cidade VARCHAR (20),
uf INT,
cep VARCHAR (10),
pais VARCHAR (20),
telefone1 VARCHAR (50),
telefone2 VARCHAR (50),
data_nascimento DATE
);
CREATE TABLE Medico
(
codigo INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR (50),
especialidade VARCHAR (30),
instituicao VARCHAR (30),
crm VARCHAR (6),
uf_crm INT,
assinatura VARCHAR (200),
login VARCHAR (10)
);
CREATE TABLE Laudo
(
codigo INT AUTO_INCREMENT PRIMARY KEY,
texto LONGTEXT,
codigo_exame INT,
codigo_medico_laudante INT,
codigo_medico_assinante INT,
data DATE,
FOREIGN KEY (codigo_medico_laudante) REFERENCES Medico (codigo),
FOREIGN KEY (codigo_medico_assinante) REFERENCES Medico (codigo),
FOREIGN KEY (codigo_exame) REFERENCES Exame (codigo)
```

```
);
CREATE TABLE Exame
(
codigo INT AUTO_INCREMENT PRIMARY KEY,
tipo INT,
codigo_medico_solicitante INT,
codigo_medico_responsavel INT,
codigo_paciente INT,
data DATE,
FOREIGN KEY (codigo_medico_solicitante) REFERENCES Medico (codigo),
FOREIGN KEY (codigo_medico_responsavel) REFERENCES Medico (codigo),
FOREIGN KEY (codigo_paciente) REFERENCES Paciente (codigo)
);
CREATE TABLE Diretorio
(
codigo INT AUTO_INCREMENT PRIMARY KEY,
codigo_medico INT,
codigo_exame INT,
imagens LONGTEXT,
FOREIGN KEY (codigo_medico) REFERENCES Medico (codigo),
FOREIGN KEY (codigo_exame) REFERENCES Exame (codigo)).
```

Seguem nas próximas páginas as **Figuras A.8 a A.13** com os diagramas UML de algumas das classes implementadas:

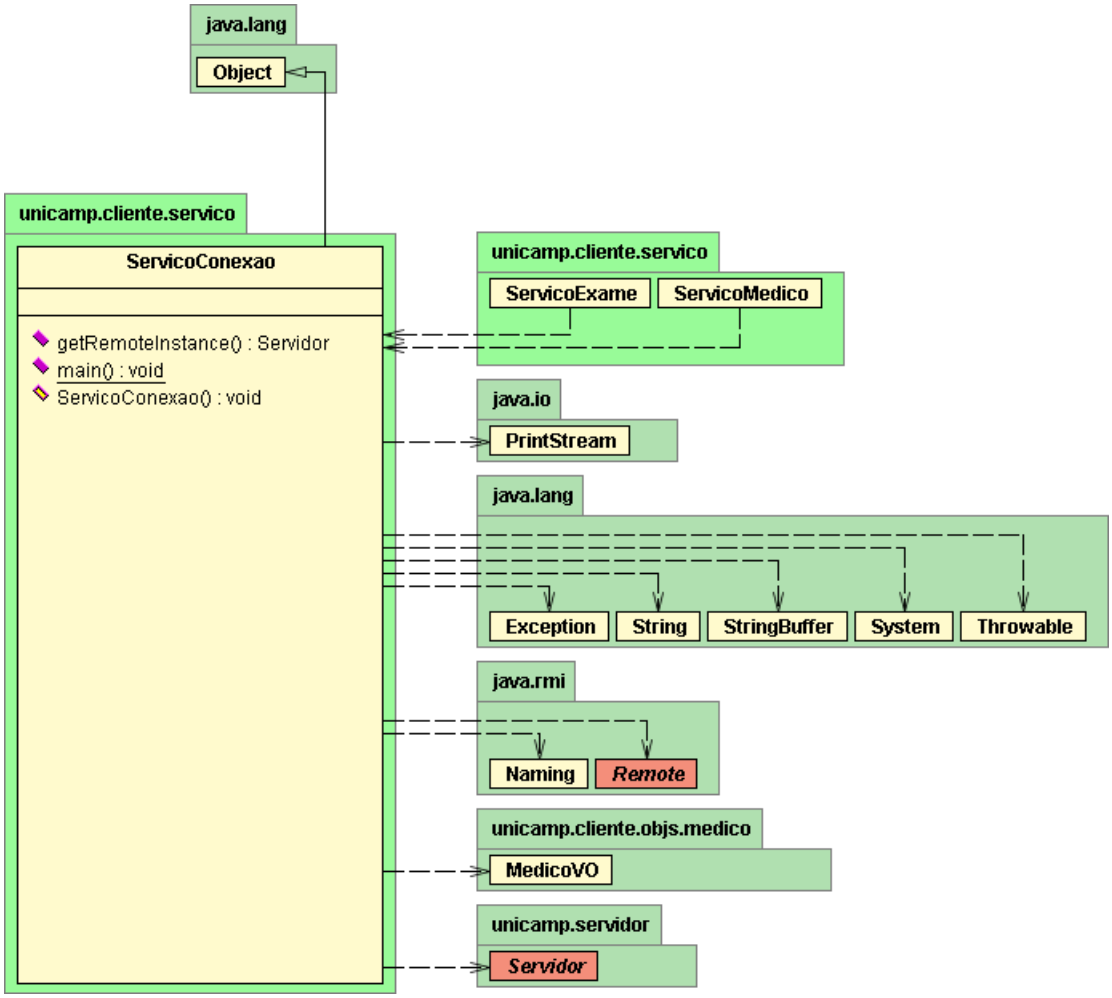

Figura A.8 – Diagrama UML da classe *ServicoConexao*, mostrando suas ligações com outras classes, invocando métodos (→) ou tendo seus métodos requisitados (←).

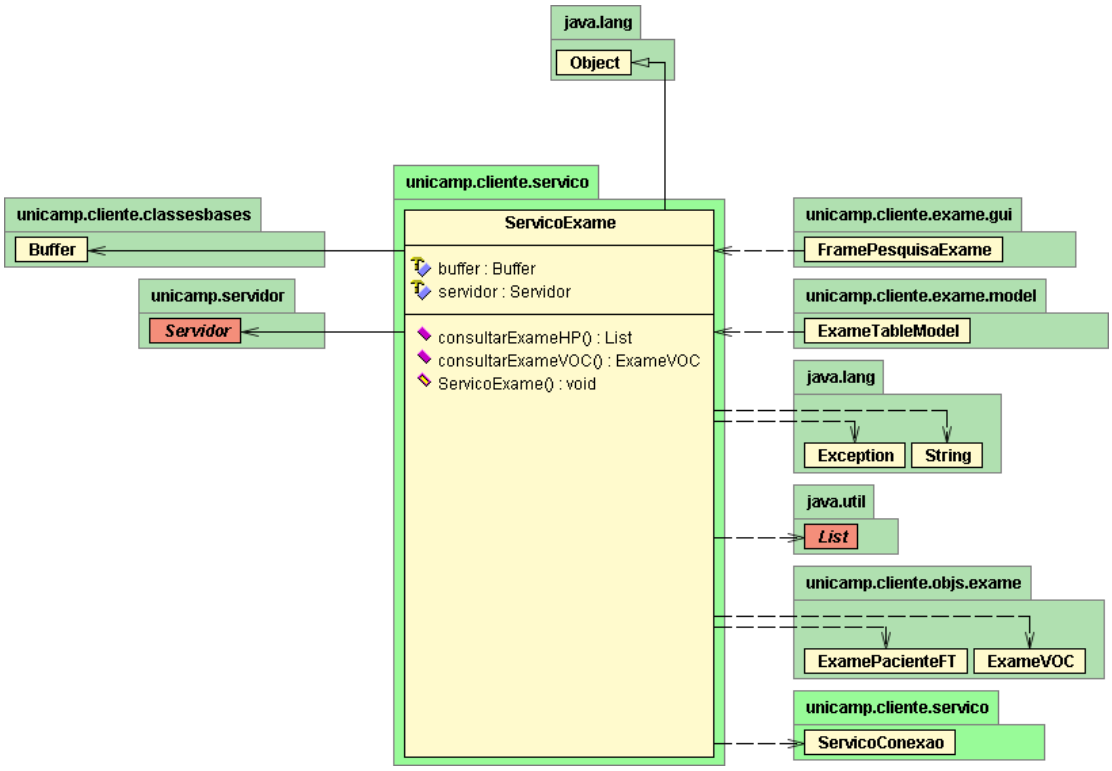

Figura A.9 – Diagrama UML da classe *ServicoExame*, mostrando suas ligações com outras classes, invocando métodos (→) ou tendo seus métodos requisitados (←).

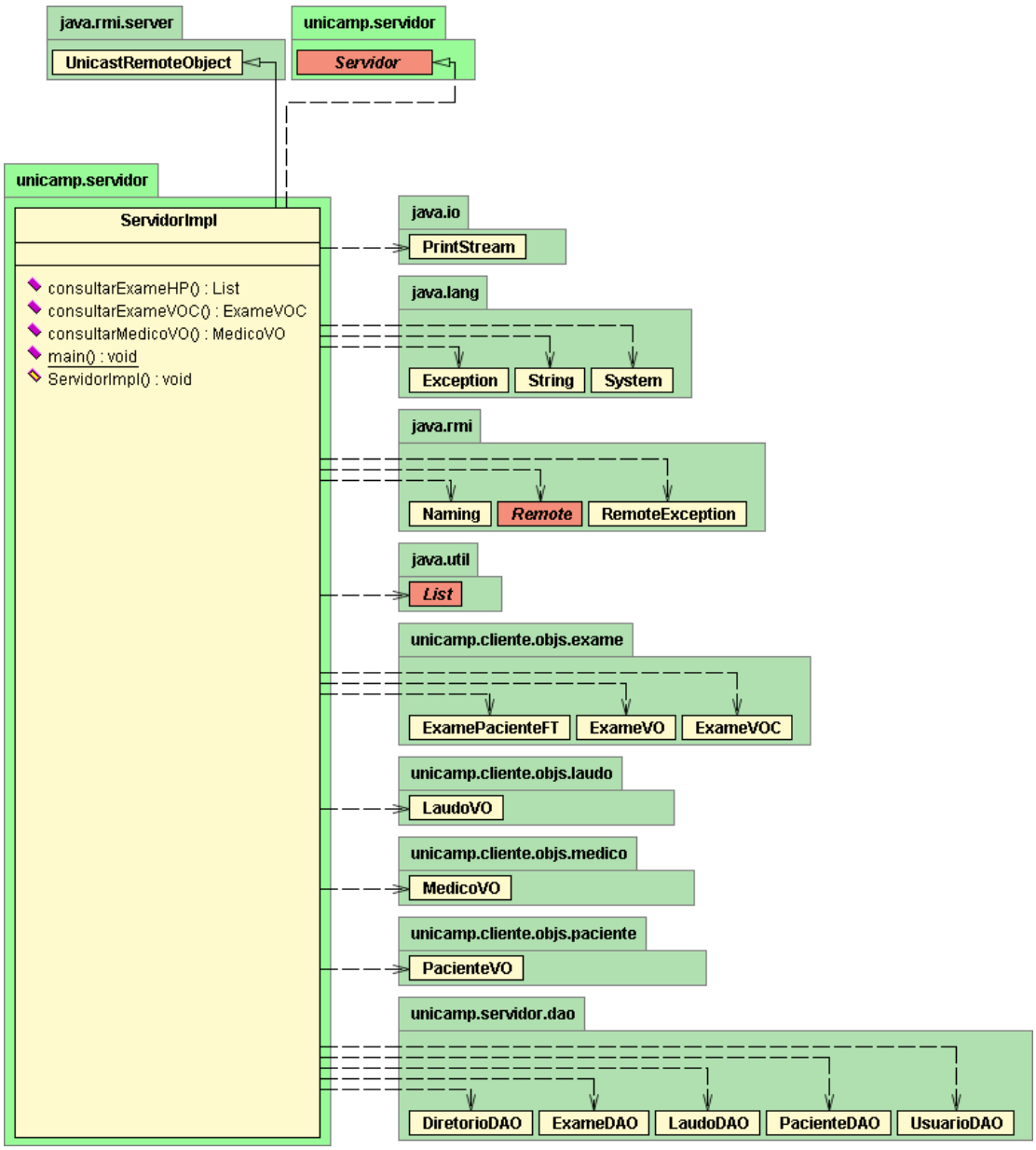

Figura A.10 – Diagrama UML da classe *ServidorImpl*, mostrando suas ligações com outras classes, invocando métodos (→) ou tendo seus métodos requisitados (←).

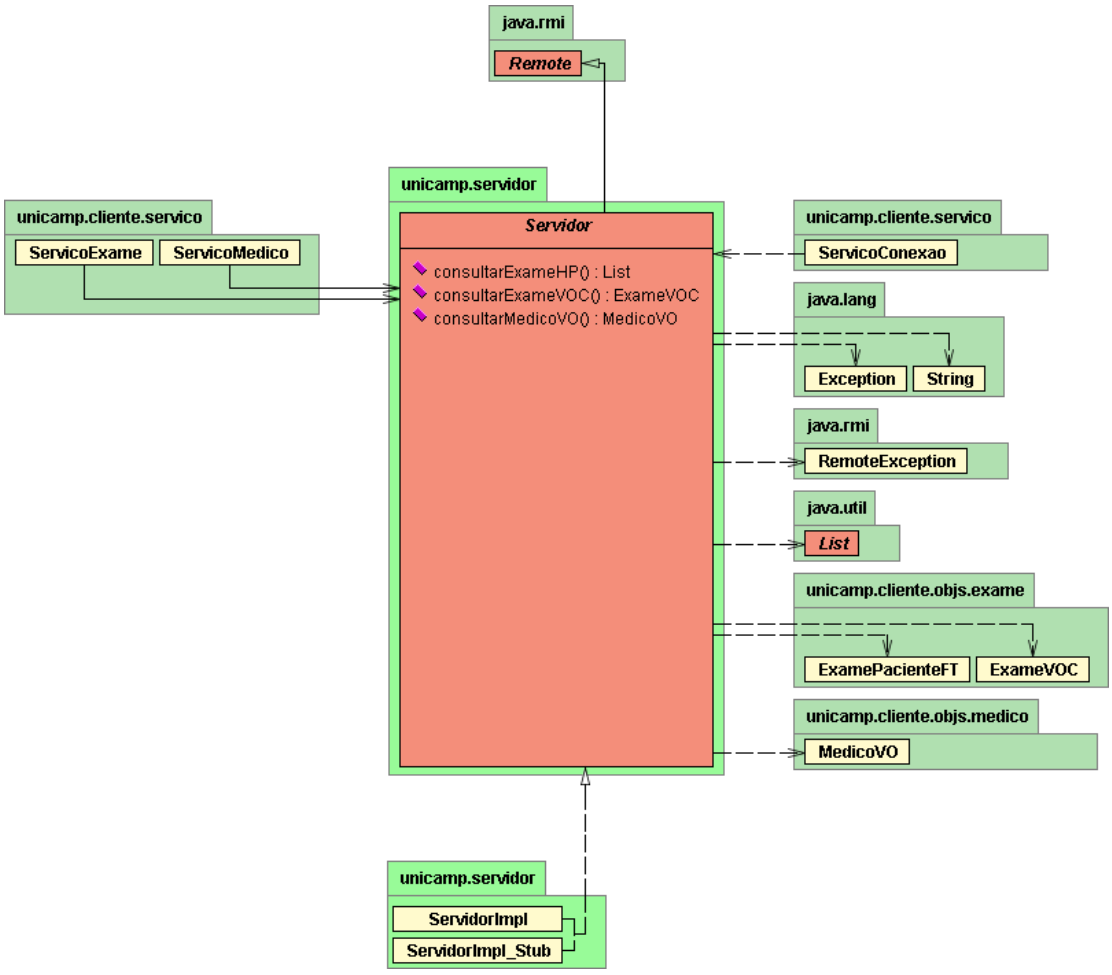

Figura A.11 – Diagrama UML da classe *Servidor*, mostrando suas ligações com outras classes, invocando métodos (→) ou tendo seus métodos requisitados (←).

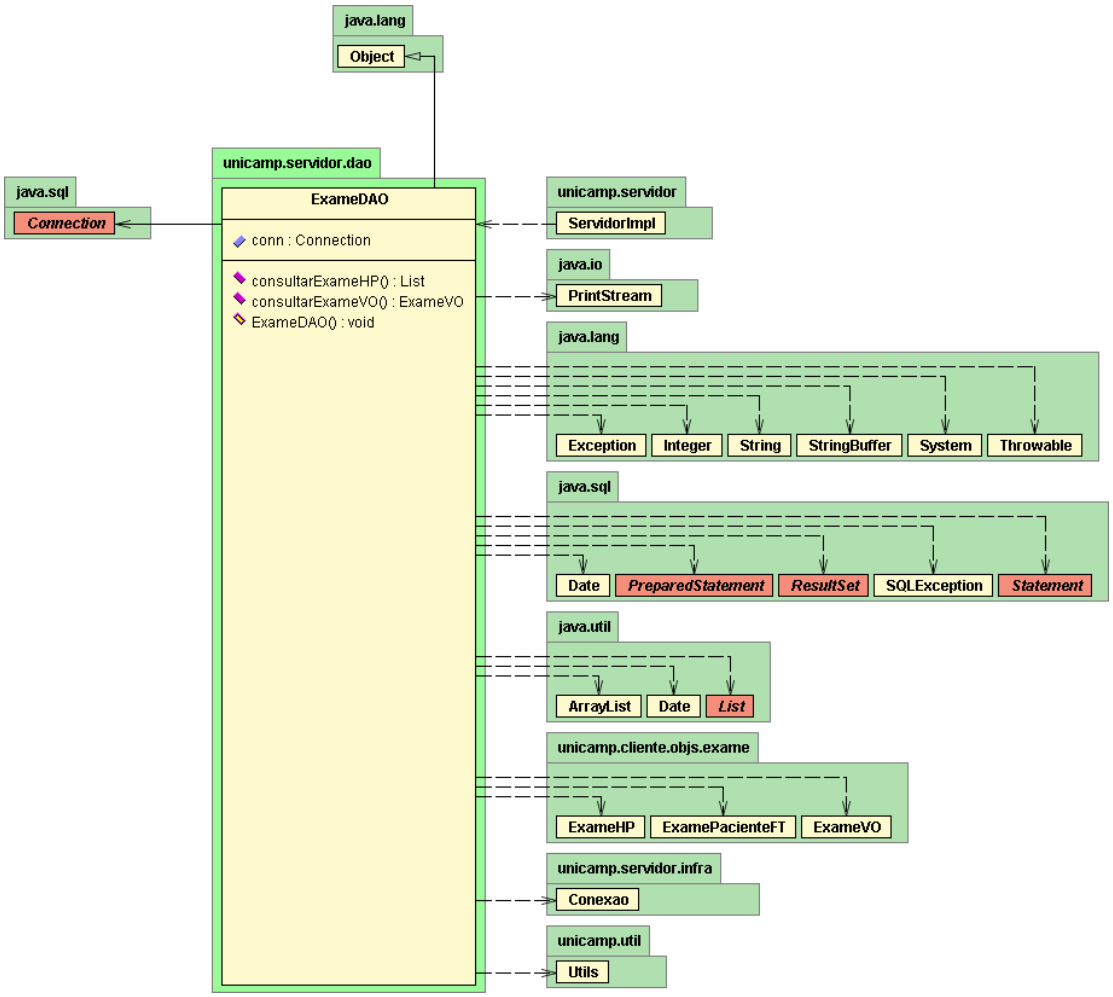

Figura A.12 – Diagrama UML da classe *ExameDAO*, mostrando suas ligações com outras classes, invocando métodos (→) ou tendo seus métodos requisitados (←).

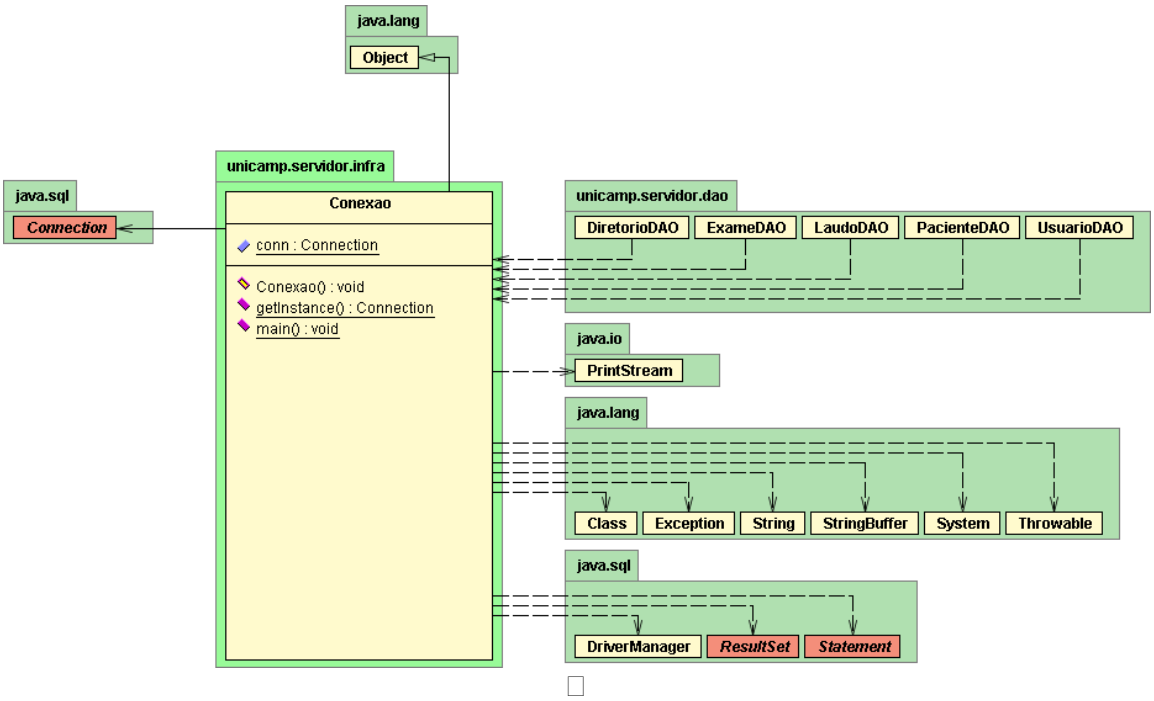

Figura A.13 – Diagrama UML da classe Conexao, mostrando suas ligações com outras classes, invocando métodos (→) ou tendo seus métodos requisitados (←).