RODRIGO ALLAN DOS REIS

R.R. DICOM VIEWER: SOFTWARE VISUALIZADOR DE ARQUIVOS DICOM PARA MAC OS X

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Prof. Dr. Homero Schiabel

São Carlos
2010

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

R375r

Reis, Rodrigo Allan dos
R.R. DICOM VIEWER : Software visualizador de arquivos DICOM para MAC os X / Rodrigo Allan dos Reis ; orientador Homero Schiabel. -- São Carlos, 2011.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

1. Engenharia de Software. 2. DICOM. 3. MAC OS X. 4. Objective-C. 5. Processamento digital de imagens. I. Título.

Dedicatória

Dedico este trabalho aos meus pais, Aécio e Maria, por todos os anos em que me apoiaram, incondicionalmente, e por nunca terem duvidado do meu potêncial.

Ao meu irmão, Fernando, não apenas pelas ótimas idéias, que ajudaram muito na realização deste projeto, mas por sempre estar disposto a me ajudar em tudo.

À minha namorada Bianca, por estar ao meu lado nos momentos mais difíceis, ser tão boa companheira, acreditar em mim, e ter me ajudado a reencontrar o meu caminho.

"Nós somos aquilo que fazemos repetidamente. Excelência, então, não é um modo de agir, mas um hábito." - Aristóteles

Agradecimentos

Agradeço sinceramente ao Prof. Dr. Homero Schiabel, pela ajuda e, principalmente, pela confiança, acreditando na idéia do projeto e na possibilidade da sua realização.

Agradeço também ao Dr. Fernando Ivan dos Reis, meu irmão, que forneceu muitas idéias para o projeto, além de compartilhar seus conhecimentos na área médica.

Resumo

Este projeto tem como objetivo o desenvolvimento de um *software* capaz de mostrar as informações presentes em arquivos de imagens médicas digitais, os arquivos DICOM, dentro da plataforma do sistema operacional Mac OS X. Estes sistemas operacionais, usados pelos computadores da empresa *Apple*, são muito confiáveis e seguros, fazendo crescer a sua utilização em diversas áreas, inclusive a médica. Desta forma, optou-se pela implementação de um *software* que suprisse as necessidades básicas de uma clínica médica - a visualização das imagens e as correspondentes informações contidas nos arquivos DICOM. A linguagem de programação escolhida foi *Objective-C*, que permite uma codificação orientada a objetos, o que, devido às suas qualidades como linguagem, torna o *software* estruturado e simples, facilitando a atualização e inclusão de novas funções. Estas características permitem que o *software* obtido neste projeto possa ser facilmente modificado futuramente para operar em outros dispositivos da *Apple*, os quais possuem sistemas operacionais equivalentes ao Mac OS X, como celulares e computadores de mão.

Palavras-chave: DICOM, Mac OS X, Objective-C, engenharia de software, imagens médicas digitais.

Abstract

This project aims to develop a software capable of showing the information contained in files of digital medical images, DICOM files, inside the platform's operating system Mac OS X. These operating systems, used by the computers of the company Apple, are very reliable and secure, increasing its use in several areas, including medical. Thus, it was chosen to implement a software that met the basic needs of a medical clinic - the viewing of images and related information contained in DICOM files. The chosen programming language was the Objective-C, which allows an object-oriented coding, which, due to its qualities as a language, makes the software simple and structured, making it easier to update and add new functions. These features allow the software obtained from this project to be easily modified to futurely operate in other Apple devices, which have similar operating systems of Mac OS X, such as mobile phones and handheld computers.

Keywords: DICOM, Mac OS X, Objective-C, software engineering, digital medical images.

Sumário

1. Introdução

O indispensável uso de computadores na maioria dos ambientes de trabalho atualmente, a tardia e necessária popularização do padrão de imagens digitais e comunicações em medicina, o padrão DICOM (*Digital Imaging and Communications In Medicine*), a crescente utilização, alto desempenho gráfico e a confiabilidade dos computadores da empresa *Apple*, os *MacIntoshs*, motivou a realização do presente projeto. Ele consiste no desenvolvimento de um *software*, exclusivo para o sistema operacional dos *MacIntoshs*, o Mac OSX, que pudesse ser utilizado em clínicas médicas para a rápida e simples visualização desses arquivos do padrão DICOM.

Este padrão foi criado como uma tentativa de padronizar a comunicação entre diferentes dispositivos de aquisição digital de imagens médicas, possibilitando assim a troca de informações entre clínicas localizadas em diversas posições geográficas. O padrão vem sendo desenvolvido desde a década de 80, e encontra-se atualmente em sua terceira versão. A dificuldade na difusão e a eventual causa de tamanha demora na obtenção de um resultado satisfatório devem-se muito às complicações nas negociações entre as partes interessadas, sobretudo fabricantes de equipamentos e de *softwares*.

As funções designadas para o *software* poderiam ser muitas, desde as organizacionais, como agendamento e controle de exames e horários, até funções mais complexas relativas à manipulação das imagens a serem visualizadas, podendo, desta forma, se tornarem muitas para o projeto em questão. Portanto, optou-se pela implementação de um *software* básico e simples, com funções apenas de visualização do conteúdo dos arquivos médicos, utilizando como base os conceitos de Engenharia de Software, tornando o projeto estruturado e metodológico, podendo assim, posteriormente, ser facilmente atualizado com novas funções. Estas atualizações torná-lo-iam mais útil na rotina das clínicas médicas e possibilitariam sua expansão para dispositivos móveis portáteis da linha *Apple*, que utilizam o mesmo sistema operacional dos *MacIntoshs*, como o *iPhone*, o *iPod* e o recente *iPad*.

2. Objetivos

O objetivo principal do projeto é obter uma ferramenta baseada em *software* livre capaz de visualizar as informações e a imagem digital contidas nos arquivos DICOM, em substituição a estruturas dedicadas e de exclusividade de certos fabricantes de equipamentos particulares de obtenção de imagens médicas digitais. A premissa é o desenvolvimento desse *software* utilizando-se das metodologias provenientes das disciplinas de Engenharia de Software e Linguagens de Programação e Aplicações, além de demais conceitos obtidos no curso relacionados a programação, gerenciamento, e realização de projetos. Portanto, entende-se também como objetivo do projeto a boa qualidade da documentação, que é parte fundamental na qualidade do produto final, sendo necessário para o uso de conceitos como a abstração e a modularidade, que garantem uma boa organização do conteúdo do projeto.

Além disso, tem-se também como objetivo complementar a versatilidade do *software* final, de modo que possa ser facilmente atualizável, permitindo a implementação de versões para outros dispositivos móveis portáteis. Assim, torna-se uma ferramenta bem generalista, possibilitando futuras adições de funções mais específicas, conforme as necessidades do usuário.

3. A linguagem de programação Objective-C

Objective-C é uma linguagem computacional simples projetada para permitir uma sofisticada programação orientada a objeto. Objective-C é definido como um pequeno mas poderoso conjunto de extensões da linguagem padrão ANSI C. Suas adições ao C são em sua maior parte baseadas em *Smalltalk*, uma das primeiras linguagens de programação orientada a objeto. A linguagem Objective-C é projetada para dar ao C capacidades totais de orientação ao objeto, e fazer isso de um modo simples e direto. (Object-Oriented Programming with Objective-C, 2008)

Uma abordagem orientada a objetos no desenvolvimento de um aplicativo torna um programa mais intuitivo de se projetar, mais rápido, mais suscetível a modificações, e mais fácil de entender. A maior parte dos ambientes de desenvolvimento orientados a objeto consiste de, no mínimo, três partes:

- Uma biblioteca de objetos;
- Um conjunto de ferramentas de desenvolvimento;
- Uma linguagem de programação orientada a objetos e uma biblioteca de apoio

Como linguagem, Objective-C tem uma longa história. Ela foi criada pela empresa *Stepstone* no começo da década de 80 por Brad Cox e Tom Love. Foi licenciada por *NeXT Computer Inc*. no fim da década de 80 para o desenvolvimento da *NeXTStep frameworks*, que antecedeu a *Cocoa*. *NeXT* extendeu a linguagem de muitos modos, por exemplo, com a adição de novos protocolos (Object-Oriented Programming with Objective-C, 2008).

3.1. Motivação para usar Objective-C

A linguagem Objective-C foi escolhida pelo *framework Cocoa* (ver seção 4.2) por uma variedade de motivos. Primeiro, e mais importante, ela é uma linguagem orientada a objetos. O tipo de funcionalidade que está presente no *framework Cocoa* só pode ser usado por meio de técnicas de orientação a objetos. Segundo, devido a Objective-C ser uma expansão do padrão ANSI C: programas em C existentes podem ser adaptados para usarem o *framework* sem perder nada do trabalho feito em seus desenvolvimentos originais. Como Objective-C incorpora o C, é possível obter todos os benefícios da linguagem C ao trabalhar com Objective-C. Além disso, Objective-C é uma linguagem simples, de sintaxe pequena e fácil de aprender. Sua terminologia é auto-consciente e tem ênfase no desenvolvimento abstrato, apresenta aos

novos adeptos uma curva íngreme de aprendizagem (Object-Oriented Programming with Objective-C, 2008).

3.2. A Programação orientada a objetos

"*A orientação a objetos é uma abordagem para desenvolvimento de software que organiza os problemas e suas soluções como um conjunto de objetos distintos. A estrutura e o comportamento dos dados estão incluídos na representação*" (PFLEEGER, 2004, p. 210).

Segundo o documento Object-Oriented Programming with Objective-C (Object-Oriented Programming with Objective-C, 2008) a programação orientada a objetos fornece uma abstração dos dados os quais podem ser operados. Além disso, ela fornece um agrupamento concreto entre dados e as operações que podem ser feitas com os mesmos.

As linguagens de programação, tradicionalmente, dividiram o mundo em duas partes - dados e operações nos dados. Dado é estático e imutável, a não ser que operações os mudem. Esta divisão é, obviamente, fundamentada na maneira como os computadores funcionam, por isso não pode ser facilmente ignorada ou deixada de lado. Em algum ponto, todos os programadores - mesmo os programadores orientados a objeto - devem dispor as estruturas de dados as quais seus programas irão usar e definir as funções que irão agir sobre os dados (Object-Oriented Programming with Objective-C, 2008).

A linguagem pode oferecer várias maneiras de suporte para a organização dos dados e das funções, mas ela não dividirá o mundo de maneira diferente. Funções e dados são os elementos básicos do projeto (Object-Oriented Programming with Objective-C, 2008).

A programação orientada a objetos não só contesta esta visão do mundo como a reestrutura em um nível superior. Ela agrupa operações e dados em uma unidade modular chamada objeto e permite a combinação destes em redes estruturadas a fim de formar um programa completo. Neste tipo de linguagem, objetos e interações de objetos são os elementos básicos do projeto (Object-Oriented Programming with Objective-C, 2008).

Todo objeto tem seu estado (dado) e comportamento (operações nos dados). Assim, eles não são tão diferentes de objetos físicos comuns. É fácil ver como um dispositivo mecânico, como um relógio de bolso ou um piano, englobam tanto um estado como um comportamento. Mas quase tudo o que é projetado para fazer um trabalho também englobam. Mesmo coisas simples como uma garrafa combinam estado (o quão cheia está, estar ou não aberta, o quão quente está seu conteúdo) com comportamento (a habilidade de entregar seu conteúdo com diferentes taxas de fluxo, estar aberta ou fechada, suportar temperaturas altas ou baixas). São, portanto, estas semelhanças com coisas reais que dão aos objetos tanto

poder (Object-Oriented Programming with Objective-C, 2008).

Uma representação de um objeto pode ser vista na Figura 1. Segundo Object-Oriented Programming with Objective-C (Object-Oriented Programming with Objective-C, 2008), o objeto combina o estado e o comportamento, e é um grupo de funções relacionadas e uma estrutura de dados que alimenta essas funções. As funções são conhecidas como métodos, e os campos de sua estrutura de dados são as variáveis exemplo.

Figura 1- Representação de um Objeto (Object-Oriented Programming wit Objective-C, 2008)

Um programa consiste em uma rede de objetos interconectados que chamam uns aos outros para resolver uma parte de um quebra cabeças. Cada objeto tem um papel específico para interpretar no projeto total do programa e é capaz de se comunicar com outros objetos. Objetos comunicam-se por mensagens, que são pedidos para realizarem métodos.

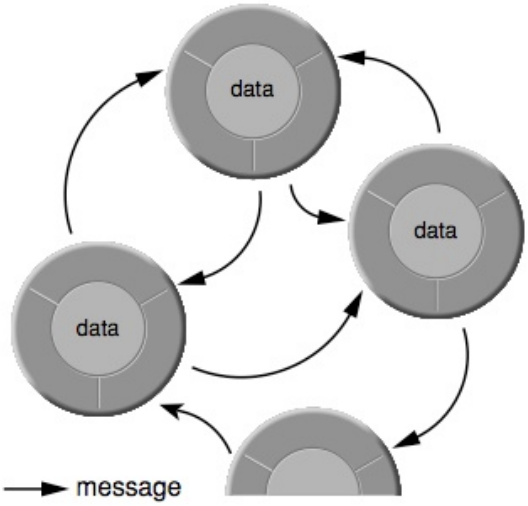

Figura 2 - Rede de Objetos (Object-Oriented Programming wit Objective-C, 2008)

Um programa pode ter mais de um exemplo, ou instância, do mesmo objeto, os quais dizemos serem membros da mesma Classe. Todos os membros de uma classe são capazes de efetuar os mesmos métodos e têm os mesmos conjuntos de variáveis exemplo. Eles também compartilham uma definição comum; cada tipo de objeto é definido somente uma vez.

3.3. A Possibilidade de Reutilização da Linguagem Orientada a Objetos

Projetar um programa orientado a objeto não implica, necessariamente, em escrever grandes quantidades de código. A possibilidade de reutilizar as definições de classes dá uma grande oportunidade de criar um programa amplamente de classes planejadas por outros. Pode até ser possível construir programas interessantes inteiramente com o uso de classes que outra pessoa já definiu. À medida que o número de definições de classes cresce, o programador tem mais e mais partes reutilizáveis para escolher.

As classes reutilizáveis vêm de muitas fontes. Os desenvolvimentos de projetos geralmente geram definições de classes reutilizáveis e alguns programadores empreendedores os comercializam. Os ambientes de programação orientada a objetos tipicamente vêm com bibliotecas de classes.

Tipicamente, um grupo de biblioteca de classes trabalha junto para definir parcialmente a estrutura de um programa. Essas classes constituem um *kit*, que pode ser usado para construir uma variedade de aplicativos diferentes. Quando um programador usa um *framework*, ele aceita o modelo de programa que o *framework* fornece e adapta o seu projeto a ele. Um programador pode utilizar um *framework* das

seguintes formas:

- Inicializar os exemplos das classes do *framework*
- Definir subclasses das classes do *framework*
- Definir novas classes para trabalhar com as classes definidas pelo *framework*

4. O Sistema Operacional Unix Mac OSX

Unix é um sistema operacional desenvolvido em 1969 pela AT&T. Hoje os sistemas Unix estão divididos em vários, alguns desenvolvidos pela própria AT&T e outros por organizações lucrativas e não lucrativas. Alguns sistemas operacionais modernos conhecidos atualmente descendem diretamente do sistema Unix, como a nova versão desenvolvida pela própria AT&T, o SVR4 (*System V Release* 4) e o Solaris (da empresa *Sun*, baseada em SVR4). Outros sistemas, embora não tenham relação direta com o desenvolvimento do Unix, ganharam o direito de usar a marca por atenderem a uma série de requisitos impostos pelo consórcio industrial *The Open Group*, que é atualmente o dono da marca Unix. Dentre estes sistemas está o Mac OS X, que é o resultado de algumas modificações no sistema operacional chamado *NeXTStep*, da empresa *NeXT*, que foi comprada pela *Apple* em 1997. O sistema operacional então passou a ser conhecido como Darwin, o qual foi melhorado e só então chegou-se ao sistema Mac OS X.

4.1. Como o Sistema Unix Difere de Outros Sistemas Operacionais

O objetivo de todos os sistemas operacionais é mais ou menos o mesmo: controlar as atividades de um computador. Os sistemas operacionais diferem na maneira como eles fazem seu trabalho e nas características adicionais que oferecem. O Unix é único em seu desenho modular, que permite aos usuários acrescentar ou remover partes para adaptá-lo às suas necessidades específicas. Os programas em Unix são como peças de um quebra-cabeças; os módulos se encaixam com conexões-padrão. Pode-se tirar um módulo e substituí-lo por um outro ou expandir o sistema acrescentando vários módulos. De uma certa maneira, o sistema Unix de cada pessoa é único. Muitos usuários acrescentam ou eliminam módulos sempre que preciso, adaptando suas implementações às suas necessidades. Se um módulo não é necessário, pode-se geralmente removê-lo sem prejudicar a operação do resto do sistema. Essa característica é especialmente útil nas implementações de microcomputadores, onde as unidades de disco têm capacidade limitada; a remoção de programas desnecessários abre espaço para mais arquivos de dados (THOMAS, YATES, 1989).

Além disso, uma das questões principais que tornam os sistemas Unix mais confiáveis que os demais é sua segurança. Por vários motivos – como o bom uso do sistema de usuários, não dando privilégios totais a eles, e o próprio foco da maioria dos invasores nos sistemas operacionais Windows, que são mais frágeis – os computadores que rodam sistemas operacionais Unix acabam sendo muito seguros. A estabilidade do sistema é também muito alta, mantendo assim a integridade total dos arquivos e

trabalhos presentes no computador. Desta forma, a utilização desses sistemas em clínicas médicas, preocupadas com a segurança dos seus exames, vem crescendo nos últimos anos.

4.2. *Cocoa Framework*

O Mac OS X fornece aos seus usuários uma API (*Application Programming Interface*) chamada *Cocoa Framework*. Um *framework*, como citado anteriormente, é uma biblioteca de classes de objetos previamente definidas que podem ser usadas no desenvolvimento de *softwares*. *Cocoa* fornece uma grande coleção de classes definidas para uso na linguagem Objective-C, com duas bibliotecas principais: *Foudation Kit* e *Application Kit*.

Foudation Kit fornece serviços que não estão diretamente ligados à interface gráfica, como manipulação de valores e *strings*, e estrutura de laços. *Application Kit* define as classes usadas para a implementação da interface gráfica.

De uma forma resumida, *Cocoa framework* é um conjunto de bibliotecas, contendo definições de classes de objetos e definidas sob a linguagem Objective-C, que fornece ao usuário a oportunidade de pôr em prática a reutilização da linguagem, configurando-se assim como uma poderosa ferramenta para o desenvolvimento de aplicativos para o sistema operacional Mac OS X.

5. Análise de Sistema e de Requisitos

Antes que o *software* possa ser submetido a engenharia, o "sistema" no qual ele reside deve ser entendido. Para conseguir isso, o objetivo geral do sistema deve ser determinado: o papel do *hardware*, *software*, pessoal, base de dados, procedimentos e outros elementos do sistema devem ser identificados; e requisitos operacionais devem ser conseguidos, analisados, especificados, modelados, validados e generalizados. Objetivos e requisitos operacionais mais detalhados são identificados através da informação do cliente; requisitos são analisados para avaliar sua clareza e consistência; uma especificação, frequentemente incorporando um modelo de sistema, é criada e depois validada tanto pelos profissionais quanto pelos clientes. Finalmente, os requisitos são generalizados para garantir que as modificações sejam controladas adequadamente.

Portanto, esta é a primeira etapa que deve ser realizada no processo de Engenharia de Software, e nela será analisada qual é o objetivo do programa, para que o usuário irá utilizá-lo, que tipo de pessoas irão utilizá-lo, entre outras coisas. Assim, nesta etapa deve-se analisar não apenas o *software*, mas todo o ambiente que o envolve.

Segundo Pfleeger (2004), esta etapa também pode ser chamada de projeto conceitual. O projeto conceitual descreve o sistema em uma linguagem que o cliente possa entender, em vez de utilizar jargão de computação e termos técnicos. Por exemplo, pode-se dizer ao cliente que um menu em uma tela dará aos usuários acesso às funções do sistema. O projeto conceitual pode, até mesmo, enumerar as respostas e as ações aceitáveis que podem resultar dos usuários. Entretanto, não se diz ao cliente como os dados são armazenados ou que tipo de sistema genericamente de banco de dados realizará as manipulações de dados.

5.1 Identificação das necessidades

Este é o primeiro passo no processo de análise de sistemas e o ponto de partida na evolução de um sistema baseado em computador. Consiste em descobrir se realmente existe a necessidade de se desenvolver o *software*.

Para esta verificação será considerado basicamente o que já foi discutido até o momento sobre o ambiente que envolve o sistema operacional dos computadores da *Apple*, o Mac OS X: a confiabilidade e segurança que ele fornece aos seus usuários, a sua recente popularização (preços mais acessíveis), e a relativa escassez de ferramentas que possam atingir os objetivos descritos aqui (o mercado é dominado por apenas um único *software* compatível com Mac OS X capaz de ser realmente utilizado na rotina das

clínicas médicas), além da emergente possibilidade da implementação da ferramenta compatível com dispositivos móveis portáteis. Mas para constatar de fato tal necessidade, mais alguns quesitos precisam ser verificados.

5.1.1 A função do Sistema

O sistema será criado para cumprir a função mais básica exigida por um usuário da área médica em relação a imagens médicas digitais, os arquivos DICOM, que é a sua visualização. Portanto, o *software* deve enxergar os arquivos DICOM presentes em qualquer dispositivo de armazenamento de mídia, como HD (*Hard Disk*), CDs, DVDs, memórias *flash*, e outras, que estejam diretamente ligados ao computador em que o *software* estiver instalado, sendo capaz de exibir todas as informações pertinentes presentes no conteúdo do arquivo, o que não pode ser feito sem a presença de uma ferramenta especializada nesta função.

Desta forma, o programa irá exibir na tela a imagem médica, qualquer que seja o tipo da mesma (gerada por raios X, Ressonância Magnética, ultrassom, etc, já que o que importa é sua característica de se constituir num arquivo digital, geralmente em formado L-JPEG), juntamente com as informações relativas ao exame presentes neste arquivo (Nome do Paciente, Data e Hora do exame, etc).

5.1.2 Desempenho, Qualidade e Confiabilidade

Analisando o ambiente em que o *software* estará inserido, observa-se que o perfil do usuário pode ser muitas vezes decisivo para a avaliação da qualidade e da confiabilidade do *software*. É fácil evidenciar que existem muitos médicos em atuação que possuem pouco conhecimento na área de informática, primeiramente porque os cursos de medicina pouco oferecem em termos de tais conhecimentos aos seus alunos. Além disso, existem muitos médicos que não têm grande facilidade na operação de computadores, procurando até, de certa forma, evitar o uso dos mesmos. Estes profissionais preferem continuar trabalhando com as imagens tradicionais.

Nesta realidade fica explícita a necessidade de um *software* de fácil operação, o que é chamado de *user-friendly*, para que mais profissionais se sintam confortáveis e se capacitem para o uso diário do mesmo. A sua confiabilidade fica restrita à fidelidade das informações exibidas, o que é intrínseco à implementação do programa. Como desempenho para essa proposta em si, espera-se uma alta velocidade no tempo de resposta, já que as funções não são específicas e não exigem nenhum processamento avançado ou complexo de dados.

5.1.3 Tecnologias e Recursos exigidos no desenvolvimento do Sistema

Como a proposta foca a aplicação exclusivamente em sistemas da *Apple,* o desenvolvimento e a utilização do *software* considera computadores com sistema operacional Mac OS X (Figura 3). O desempenho desses computadores não é relevante. Esses requisitos, hoje em dia, são de fácil acesso e de custo regular.

Figura 3 - Computadores Apple

Para o desenvolvimento é necessário uma API – conjunto de ferramentas que possibilitam o desenvolvimento de aplicativos – específica. A *Apple* disponibiliza, gratuitamente, uma API chamada XCode (Figuras 4 e 5), que está sendo utilizada para este projeto. Nela estão presentes os *frameworks* que acompanham a biblioteca *Cocoa framework*.

Figura 4 - Xcode

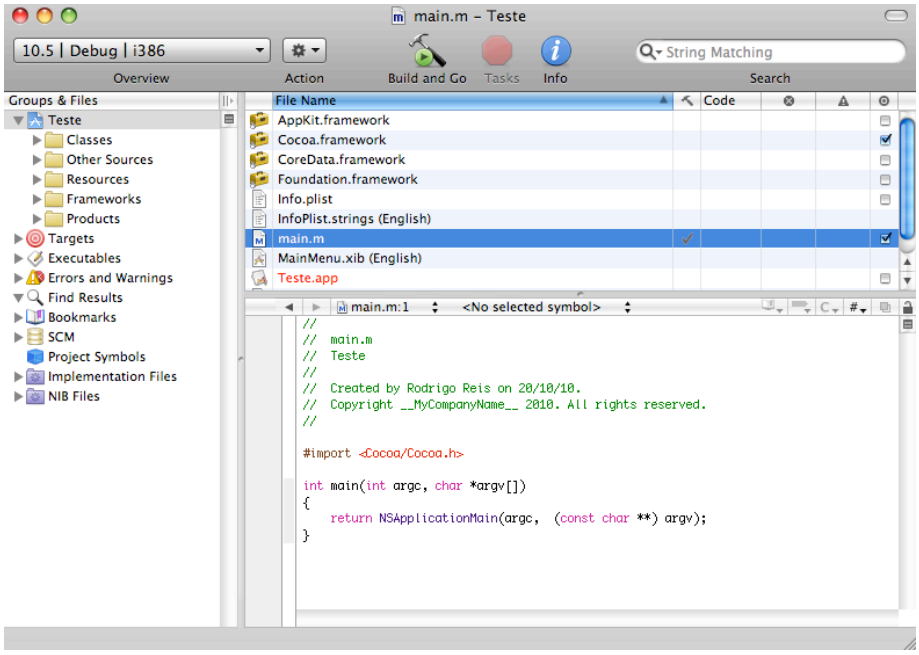

Figura 5 - Área de Trabalho do Xcode

Em relação às tecnologias e recursos adicionais necessários para a instalação e utilização do *software*, nenhuma ferramenta adicional é requerida, sendo necessário apenas o seu carregamento em si.

5.2 Estudo da Viabilidade

Três áreas principais devem ser analisadas neste estudo: viabilidade econômica, viabilidade técnica e viabilidade legal.

5.2.1 Viabilidade Econômica

Trata-se da avaliação do custo de desenvolvimento confrontada com a renda de retorno derivada do sistema desenvolvido. Como o sistema não exigirá nenhuma ferramenta adicional para o seu funcionamento, e tampouco conhecimentos avançados por parte dos seus usuários, e por se tratar de um projeto acadêmico sem fins lucrativos, onde todas as ferramentas utilizadas foram fornecidas pela universidade ou adquiridas gratuitamente (XCode – www.apple.com), o estudo de viabilidade econômica não pode ser corretamente aplicado ao caso, a não ser se tomarmos como zero o custo de desenvolvimento, o que realmente é o caso, e também como zero a renda de retorno, tornando, em todo caso, o projeto viável.

5.2.2 Viabilidade Técnica

Em termos técnicos, o projeto é totalmente viável. Os conhecimentos adquiridos nas disciplinas de programação e engenharia de software formam, aliados aos conhecimentos adicionais adquiridos em linguagens orientadas ao objeto e ao funcionamento do padrão DICOM, a base da exigência técnica do projeto. O restante é complementado por conhecimentos em gerenciamento de projetos.

5.2.3 Viabilidade Legal

Todas as ferramentas utilizadas no desenvolvimento do projeto são de uso gratuito e de licença aberta, assim como todas as bibliotecas e *frameworks* usados na implementação do código do *software*. Nenhum dado confidencial será manipulado na utilização do *software*, e os arquivos DICOM utilizados

para os testes são de exames disponibilizados para tal fim, tendo os nomes dos pacientes envolvidos nos estudos excluídos do arquivo. Portanto, em termos legais, o projeto é viável.

5.3 Análise Econômica

Após a constatação da viabilidade do projeto, ainda deve-se atentar para mais alguns detalhes econômicos relacionados ao custo-benefício do produto final. Nesta etapa, não apenas os objetivos tangíveis devem ser analisados, mas também os intangíveis, como a satisfação do cliente.

Os custos de aquisição são nulos, já que o projeto não visa fins lucrativos, não há a necessidade de compra de equipamentos adicionais e nem de licenças para a utilização do *software*. Também não existem custos relacionados a instalações, já que o *software* atua de forma autônoma no sistema, e nenhuma modificação no ambiente é necessária.

Quanto aos custos relacionados à iniciação ou treinamento dos usuários, eles também são mínimos, já que o próprio objetivo do projeto é que o *software* seja simples, facilitando a sua utilização.

Para os custos que tenham relação direta com o desenvolvimento do projeto, como já citado anteriormente, o único aplicativo adicional necessário foi o API XCode, o qual pode ser obtido gratuitamente pelo *site* da *Apple*.

Os únicos custos permanentes relacionados ao projeto são unicamente os de manutenção, devido à depreciação do *hardware* no qual ele estará instalado; porém, devido à simplicidade do programa, ele não será responsável por praticamente nada desta depreciação, já que nenhum recurso complexo de *hardware* é utilizado na realização das suas funções.

5.4 Análise Técnica

A partir dos dados coletados no estudo de viabilidade técnica, é possível refinar e detalhar tais necessidades, fazendo com que os procedimentos das etapas técnicas que seguem se tornem mais fáceis.

Assim como já foi ressaltado anteriormente, algumas características importantes da programação orientada a objetos será usada no desenvolvimento deste projeto. Uma delas é a Reutilização.

Para fazer uso desta característica da orientação a objetos, foi necessária uma intensa pesquisa por *frameworks* que pudessem suprir as necessidades, ou parte delas, envolvidas no projeto. Decidiu-se então pelo uso do *framework* ***iiDICOM***, distribuído gratuitamente pela *Imaging Informatics*. Trata-se de uma

série de definições de Classes de objetos relacionados à obtenção das informações presentes nos arquivos DICOM. Sua implementação é bastante simples, e a simplificação causada pelo seu uso é muito grande.

Além do uso correto dos benefícios da programação orientada a objetos, também será utilizado o aplicativo XCode, que permite que o processo de codificação do *software* na linguagem Objective-C aconteça. Desta forma, é imprescindível a correta utilização da ferramenta, utilizando todos os seus recursos com o objetivo de facilitar o desenvolvimento do projeto, como a ferramenta para criação de interfaces gráficas que acompanha o API, o *Interface Builder*, que pode ser visto na Figura 6.

No *Interface Builder* é possível criar instâncias de objetos, como janelas e botões, e as comunicações que eventualmente estas instâncias realizarão, as mensagens. Isto tudo é feito de maneira gráfica, só sendo necessário implementar os métodos que serão ativados pelas mensagens. Este tipo de abordagem só é possível devido à orientação ao objeto.

Assim pode-se fazer um esboço de como o *software* irá comportar-se (Figura 7). Observando o esboço, nota-se que uma classe deverá ser criada, para comportar o objeto de controle, e "chamar" todas as classes definidas nos *frameworks* utilizados. De maneira mais clara, foi feito o que está descrito no capítulo 3.3 como "Definir novas classes para trabalhar com as classes definidas pelo *framework*". Nesta classe também estarão definidos todos os métodos solicitados pelas mensagens vistas no esboço. Os demais itens do esboço são objetos já definidos pelo *framework Cocoa.*

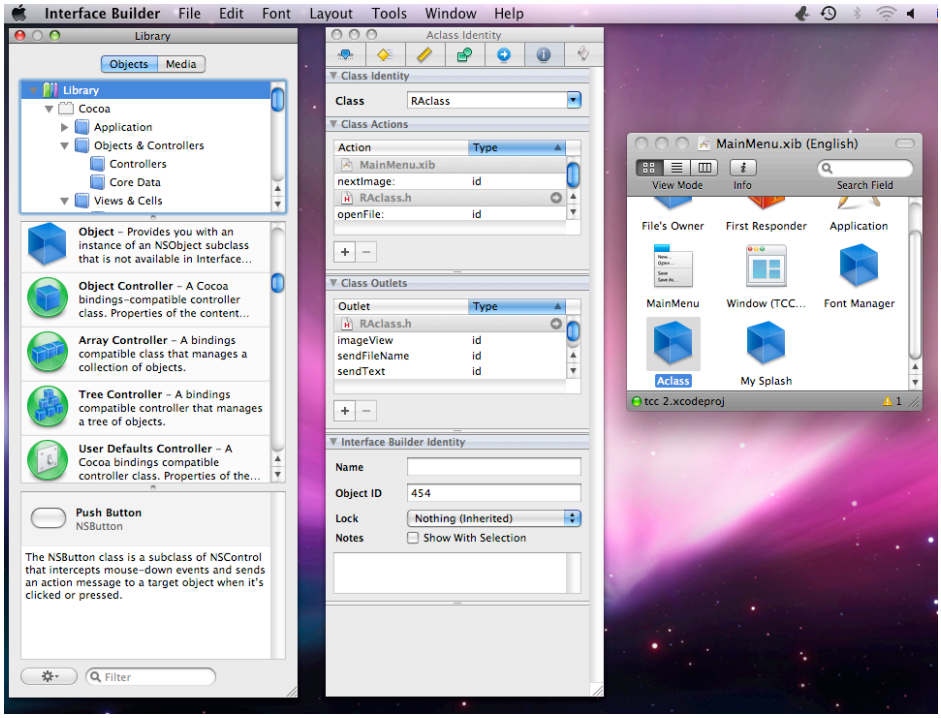

Figura 6 - Interface Builder

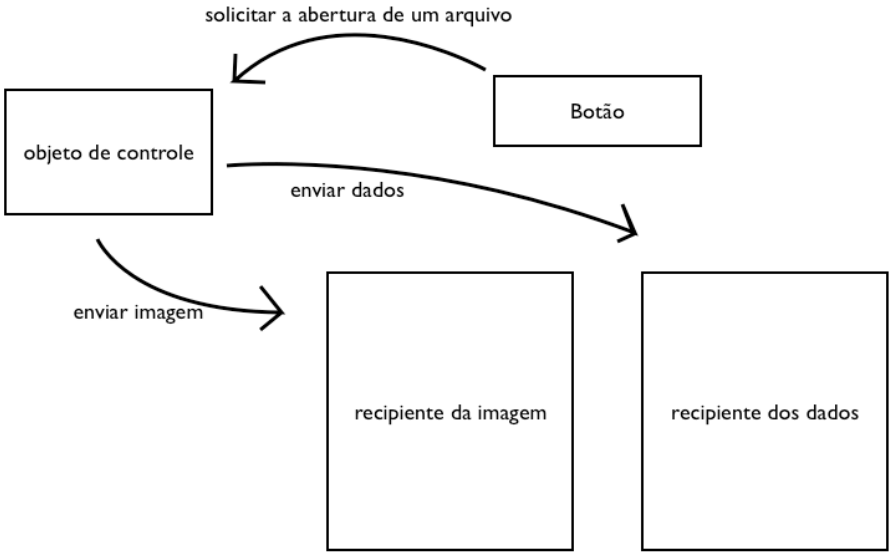

Figura 7 - Esboço do Programa

5.5 Diagrama de Contexto de Arquitetura

Todo sistema baseado em computador pode ser modelado como uma transformação de informação usando um gabarito entrada-processamento-saída Estudos posteriores estenderam essa visão e incluiram duas características adicionais do sistema - processamento e manutenção de interface do usuário, e autoteste. Estas características, embora não estejam sempre presentes, tornam mais robusto qualquer modelo de sistema (PRESSMAN, 2006).

Usando uma representação de entrada, processamento, saída, processamento da interface do usuário e processamento de autoteste, um engenheiro de sistemas pode criar um modelo de componentes do sistema que estabeleça a fundação para os passos posteriores em cada uma das disciplinas da engenharia.

Segundo Pressman (2006), como praticamente todas as técnicas de modelagem usadas na engenharia de sistemas e de software, o gabarito de modelo do sistema permite ao analista criar uma hierarquia de detalhes. Um diagrama de contexto de arquitetura (DCA) fica no nível mais alto da hierarquia. O DCA define todos os produtores externos da informação usada pelo sistema, todos os consumidores externos da informação criada pelo sistema e todas as entidades que se comunicam através da interface ou realizam manutenção e autoteste.

A Figura 8 representa o diagrama de contexto de arquitetura para este projeto.

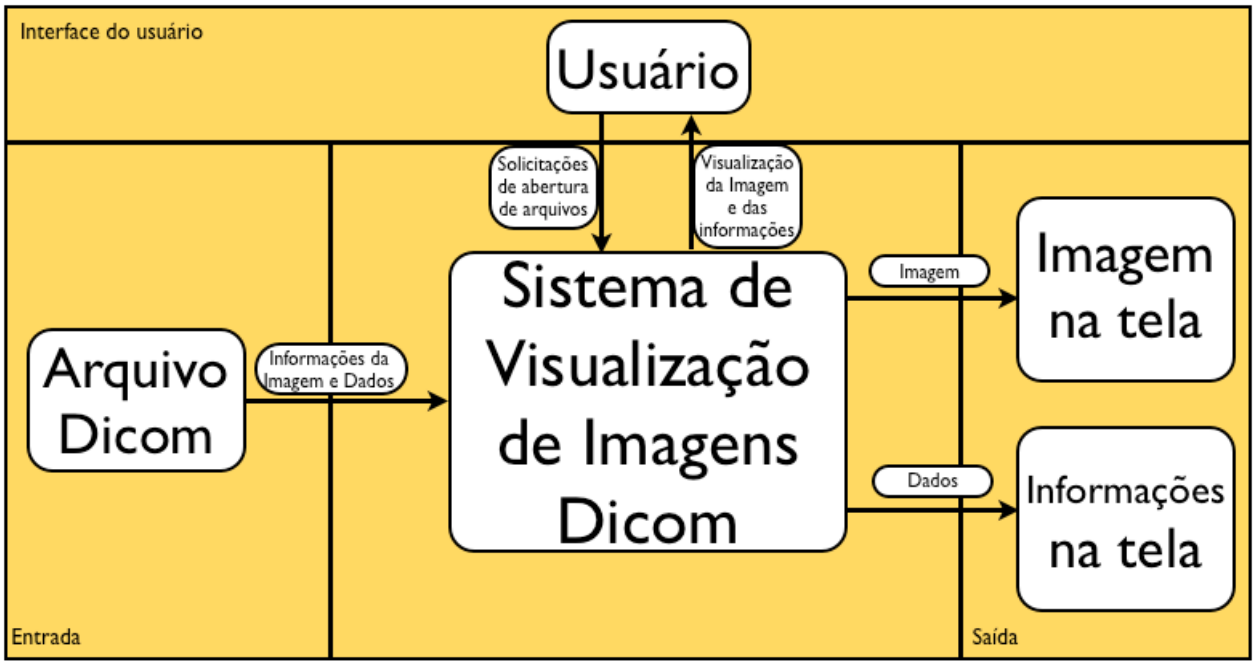

Figura 8 - Diagrama de Contexto de Arquitetura

Como sugere Pressman (2006), nem todos os sistemas apresentam todas as características citadas. É o caso, como pode ser visto na Figura 8, deste projeto, que não possui nenhum mecanismo de autoteste.

O DCA acima sugere simplesmente que o usuário solicite, através de uma interface, que o arquivo DICOM, localizado no ambiente de entrada, envie suas informações para o sistema de visualização de imagens DICOM. O sistema, então, se comunica com a saída, enviando a imagem e os dados para a tela, o que pode ser visualizado pelo usuário através da interface.

5.6. Análise de Requisitos

Entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software.

Segundo Pressman (2006), a engenharia de requisitos fornece o mecanismo apropriado para entender o que o cliente deseja, analisando as necessidades, avaliando a exequibilidade, negociando uma condição razoável, especificando a solução de modo não ambíguo, validando a especificação e gerindo os requisitos à medida que eles são transformados em um sistema operacional.

5.6.1 Reconhecimento do Problema

As soluções baseadas em computador para a visualização e manipulação de imagens médicas digitais, os arquivos DICOM, são, em sua grande parte, voltadas para o sistema operacional Windows, existindo poucas alternativas para o sistema Mac OS X. Além disso, com a nova tendência de aplicativos voltados aos dispositivos móveis portáteis, como celulares e computadores portáteis, existe uma grande demanda por visualizadores médicos compatíveis com estes equipamentos.

Os requisitos do sistema passaram por algumas alterações no decorrer do projeto, sendo que as funções de manipulação das imagens foram retiradas. Porém é um requisito do projeto permitir, através do desenvolvimento de um bom *software* básico, a inclusão destas funções no futuro. Com isto também pretende-se deixar o *software* pronto para as alterações necessárias para a implementação das versões para iPhone, iPod e iPad.

5.6.2 Revisão

Com relação a revisão do projeto, foi possível contar com a ajuda de um médico para constatar quais alterações eram convenientes. O radiologista Dr. Fernando Ivan dos Reis sugeriu a diminuição do número de informações mostradas pelo *software*. No projeto base, o *software* iria mostrar todas as informações presentes no arquivo, que contém muitos dados inúteis para os médicos. Desta forma, o radiologista sugeriu quais destes dados são realmente relevantes para os médicos na análise da imagem, e

também que essas informações fossem dispostas nos cantos da própria imagem, uma vez que este é o padrão dos melhores *softwares* do mercado neste gênero.

Essas alterações ajudaram a tornar o *software* mais amigável aos médicos, sendo que o torna mais parecido com outros *softwares* já conhecidos. Assim, mais um dos requisitos, a facilidade de uso ou *user-friendly*, pôde ser melhorado.

6. Projeto do Software

Um conjunto de conceitos fundamentais de projeto tem evoluído durante a história da engenharia de software. Apesar de o grau de interesse em cada conceito ter variado ao longo dos anos, cada um resistiu ao teste do tempo. Cada um fornece ao projetista de *software* uma base por meio da qual métodos mais sofisticados de projeto podem ser aplicados.

Em outras palavras, o projeto de *software* é responsável por analisar os requisitos especificados até o momento, e ligá-los às etapas de engenharia e manutenção que se seguirão. Com esta ponte, problemas como a construção de sistemas instáveis e a alta probabilidade de falhas quando pequenas mudanças são feitas, são minimizadas. Um *software* projetado corretamente tem a sua realização muito mais fácil, além de que seu produto final pode ser analisado quanto à sua qualidade.

Para que esse projeto seja feito de maneira a elevar a qualidade do *software*, alguns procedimentos devem sempre ser tomados: abstração, refinamento e modularidade.

Com esses conceitos em mente, é possível descrever o sistema de maneira simples, sendo irrelevante a complexidade do mesmo (abstração). Após a obtenção de um modelo simples e geral, com um alto nível de abstração, são aplicados os conceitos de refinamento e de modularidade, e então uma solução detalhada, estruturada e modularizada é adquirida.

Se estes procedimentos forem tomados juntamente com uma orientação voltada ao fluxo dos dados no sistema, podemos modelar um Diagrama de Fluxo de Dados.

6.1 Diagrama de Fluxo de Dados (DFD)

A abordagem orientada ao fluxo de dados é uma das mais usadas no mundo, e se aplica na confecção de uma ampla variedade de soluções baseadas em computador.

O diagrama de fluxo de dados permite ao engenheiros de software desenvolver modelos do domínio informacional e do domínio funcional ao mesmo tempo. À medida que o DFD é refinado em maior nível de detalhe, o analista realiza uma decomposição funcional implícita do sistema.

Uma DFD deve representar as funções do sistema, as interações entre as funções do sistema, as transformações que o sistema deve realizar, as fontes de informação, o destino dos resultados e os dados mantidos pelo sistema.

Como a análise de requisitos não gerou um número muito grande de funções, e como não existe processamentos de dados complexos relacionados à estas funções, foram necessários apenas dois níveis

(Figuras 9 e 10) para detalhar suficientemente o comportamento do fluxo de dados do sistema.

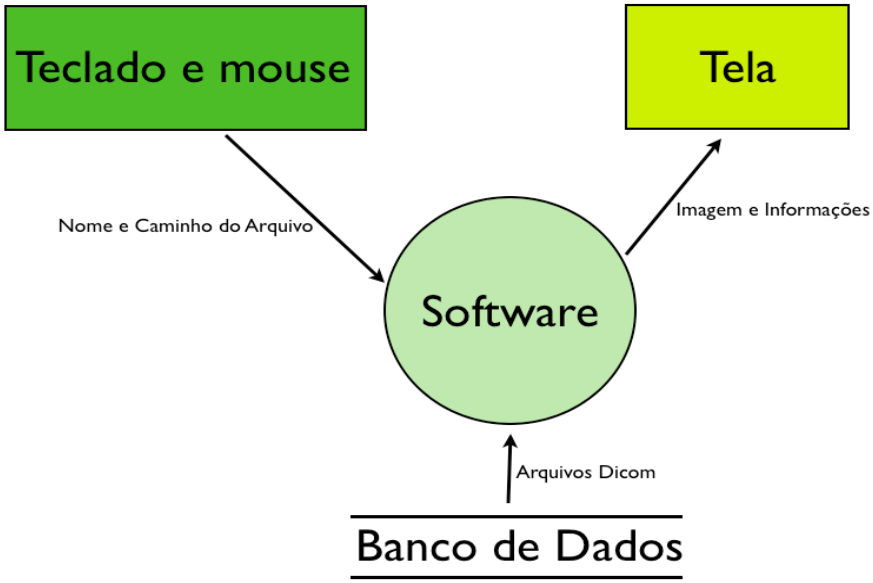

Figura 9 - Diagrama de Fluxo de Dados (Nível 1)

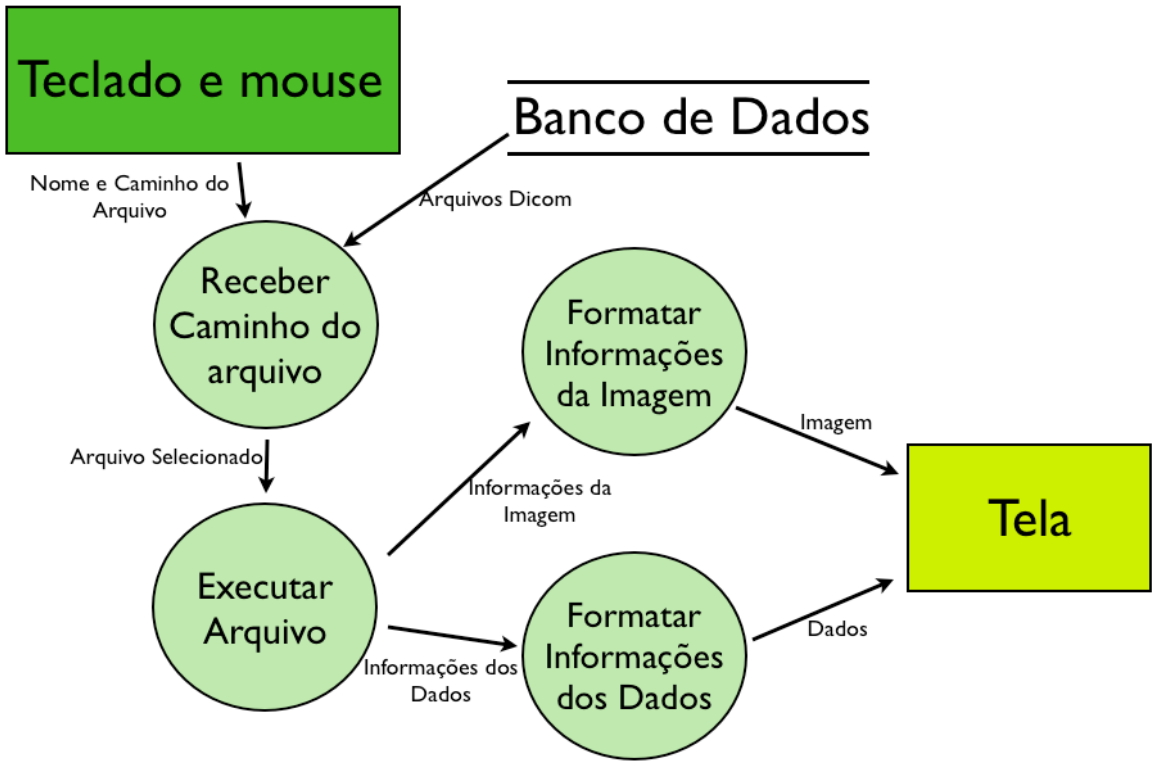

Figura 10 - Diagrama de Fluxo de Dados (Nível 2)

Nos diagramas criados é possível identificar as seguintes formas: retângulos, círculos, barras horizontais paralelas e setas.

Os círculos representam os processos do sistema. São elementos que transformam as informações e correspondem a uma função ou atividade que deve ser executada pelo sistema. As setas são usadas para descrever o movimento de informações de um componente do sistema para outro. Elas traduzem a relação entre os diferentes processos do sistema. As entidades externas são representadas pelos retângulos, que são elementos que estão fora do limite do *software*, mas que interagem com o mesmo, gerando fontes ou recebendo informações. Os fluxos que conectam entidades a processos representam interfaces entre o *software* e o ambiente externo. Por fim, as barras paralelas indicam os repositórios de dados que são armazenados para serem usados pelos processos. Podem então representar fichários, *buffers*, arquivos, banco de dados, etc.

A figura 9 nos dá apenas uma visão geral do sistema, mostrando o fluxo de dados na forma mais simples e abstrata possível. Nela podemos ver que as entidades externas são o teclado, o mouse e a tela, o que configura a interface com o usuário, que fará uso destas entidades para acessar os arquivos presentes no banco de dados.

O nível 2 do DFD, representado na Figura 10, nos dá um detalhamento dos processos contidos no processo geral descrito no nível 1. Portanto, os processos finais que foram obtidos com este refinamento são: receber caminho do arquivo, executar arquivo, formatar informações da imagem e formatar informações dos dados. São estes processos que deverão fazer parte da base principal do *software*, por isso deverão ser melhor definidos para então serem implementados na linguagem de programação escolhida pelo projeto.

6.2 Descrição dos Processos

Os quatro processos obtidos na etapa anterior são descritos da seguinte forma:

Receber Caminho do Arquivo: Este processo recebe do usuário o caminho e o nome do arquivo que será executado pelo programa, e o seleciona da lista de arquivos compatíveis presentes no banco de dados.

Executar Arquivo: Neste processo o arquivo selecionado é analisado e as informações necessárias, da imagem e dos dados, são guardadas para processamento posterior.

Formatar Informações da Imagem: As informações da imagem presentes no arquivo selecionado são processadas para gerar a imagem na tela.

Formatar Informações dos Dados: Formata as informações importantes do arquivo e os prepara para serem exibidos na tela.

6.3 Especificação dos Processos

Esta etapa se encarrega de definir o que deve ser feito dentro de cada um dos processos, de modo a transformar entradas em saídas. Trata-se de um pseudocódigo, criado para facilitar ainda mais o processo de codificação. Assim, elaborou-se o seguinte pseudocódigo:

Processo Principal

```
        Repita
            Se Processo “Receber Caminho do Arquivo”= Verdadeiro
                Processo “Executar Arquivo”
                Processo “Formatar Informações da Imagem”
                Processo “Formatar Informações dos Dados”
            Fim Se
Fim Processo

Processo “Receber Caminho do Arquivo”
        Obtenha(Arquivos_DICOM)
        Obtenha(Nome_e_Caminho_do_Arquivo)
        Se Nome=”*.dcm”
            Ler (Nome_e_Caminho_do_Arquivo)
            Seleciona Arquivo para Execução
            Retornar(Verdadeiro)
        Fim Se
Fim Processo

Processo “Executar Arquivo”
        Obtenha(Arquivo_DICOM)
        Escreva(Informações_da_Imagem)
        Escreva(Informações_dos_Dados)
Fim Processo

Processo “Formatar Informações da Imagem”
        Leia(Informações_da_Imagem)
        Gere(Imagem)
        Escreva(Imagem)
Fim Processo

Processo “Formatar Informações dos Dados”
        Leia(Informações_dos_Dados)
        Gere(Dados)
        Escreva(Dados)
```

Fim Processo

6.4 Diagrama Entidade-Relacionamento (DER)

"*O par objeto/relacionamento é a pedra fundamental do modelo de dados. Esses pares podem ser representados graficamente por meio do diagrama entidade-relacionamento (DER). O DER foi originalmente proposto por Peter Chen [CHE77] para o projeto de sistemas de base de dados relacional e foi estendido por outros. Um conjunto de componentes primordiais é identificado para o DER: objetos de dados, atributos, relacionamentos e indicadores de vários tipos. A finalidade principal do DER é representar objetos de dados e seus relacionamentos*" (PRESSMAN, 2006, p. 151).

Uma notação rudimentar do DER já foi introduzida. Objetos de dados são representados por um retângulo rotulado; relacionamentos são indicados por uma linha rotulada conectando objetos. Em algumas variantes do DER, uma linha de conexão contém um losango, que é rotulado com um relacionamento.

A Figura 11 representa a DER deste projeto. Os números entre parênteses indicam que o usuário pode selecionar, ou visualizar, no mínimo um, e no máximo um arquivo DICOM de cada vez. Já os arquivos DICOM podem ser selecionados, ou visualizados, por no mínimo um, e no máximo N usuários, já que estes arquivos podem estar armazenados em um banco de dados comum a uma rede, onde podem estar instalados N *softwares* de visualização.

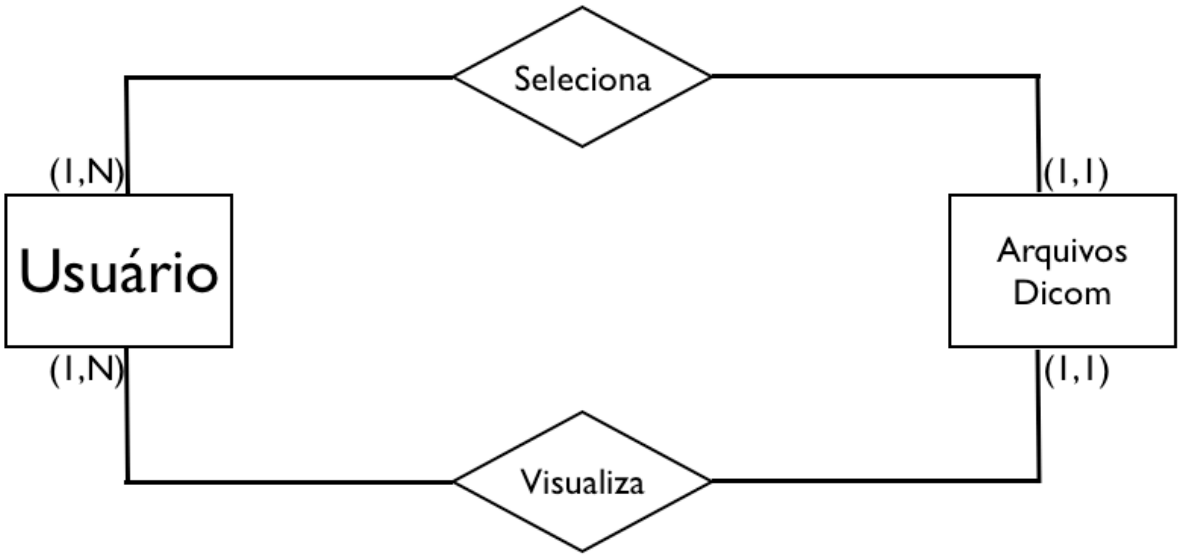

Figura 11 - Diagrama Entidade-Relacionamento

6.5 Codificação

Primeiramente foi necessária a criação de uma classe para alocar os objetos e métodos que serão usados pelo *software*. Desta forma, a classe RAClass foi criada, e nela foram implementados os métodos IBAction "openFile", acionado sempre que o botão "Open DICOM Image" é pressionado, e os IBOutlets "imageView", "sendText" e "sendFileName", que são utilizados para manipular as informações do arquivo selecionado. Estes quatro métodos são os principais responsáveis pelo funcionamento do programa, juntamente com os métodos contidos na biblioteca *iiDICOM.framework.*

Um diagrama de blocos que ilustra a implementação da classe RAClass pode ser visto na Figura 12.

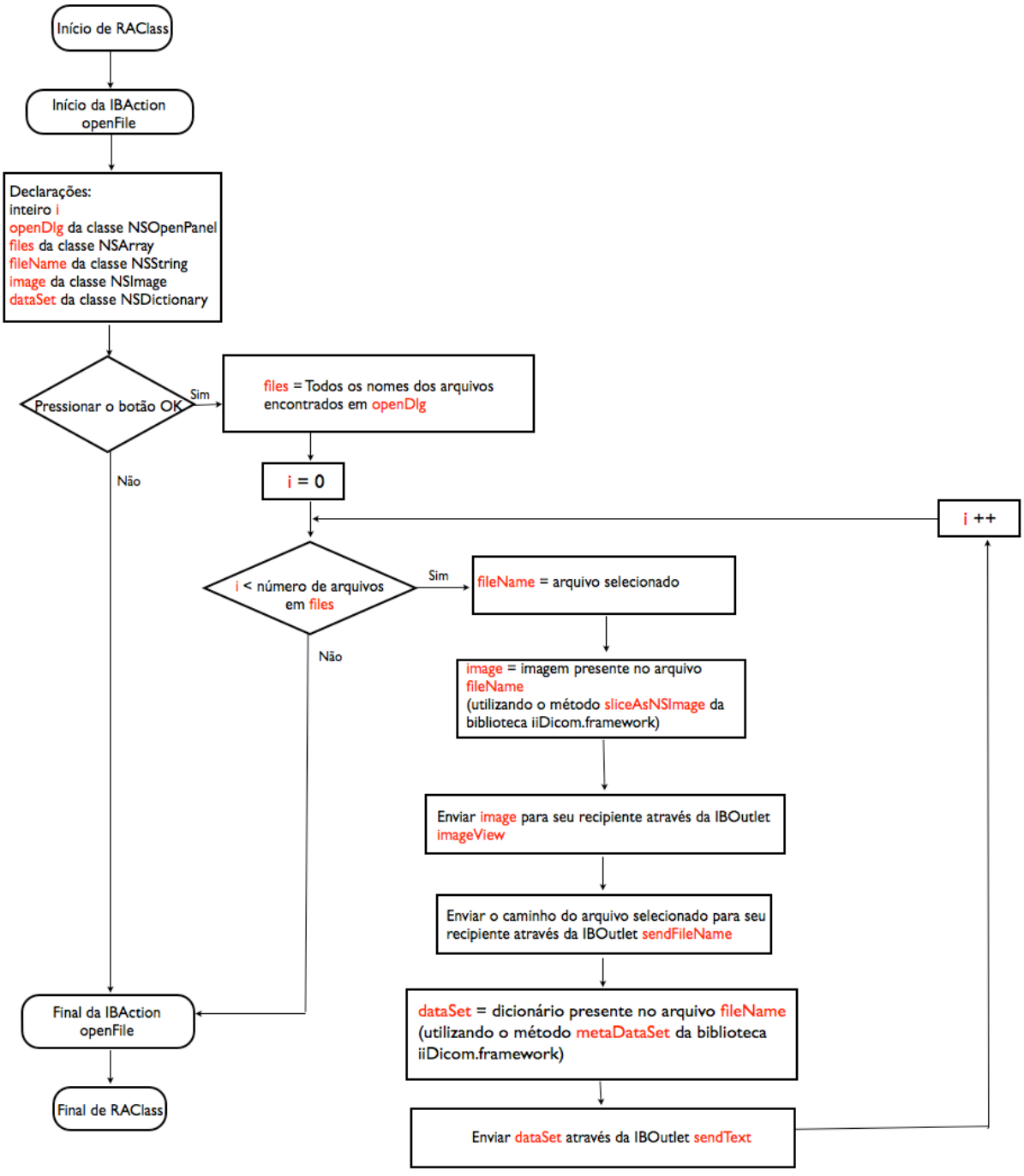

Figura 12 - Diagrama de Blocos

Para um melhor entendimento da estrutura geral do programa foi feito um diagrama de classes, que representa mais claramente as relações que foram formadas entre as classes do software. Esse tipo de modelagem é muito útil, pois define todas as classes que serão necessárias no sistema.

Properties mostram os atributos e *Operations* mostram os métodos que podem ser utilizados pelos objetos instanciados por cada uma das classes.

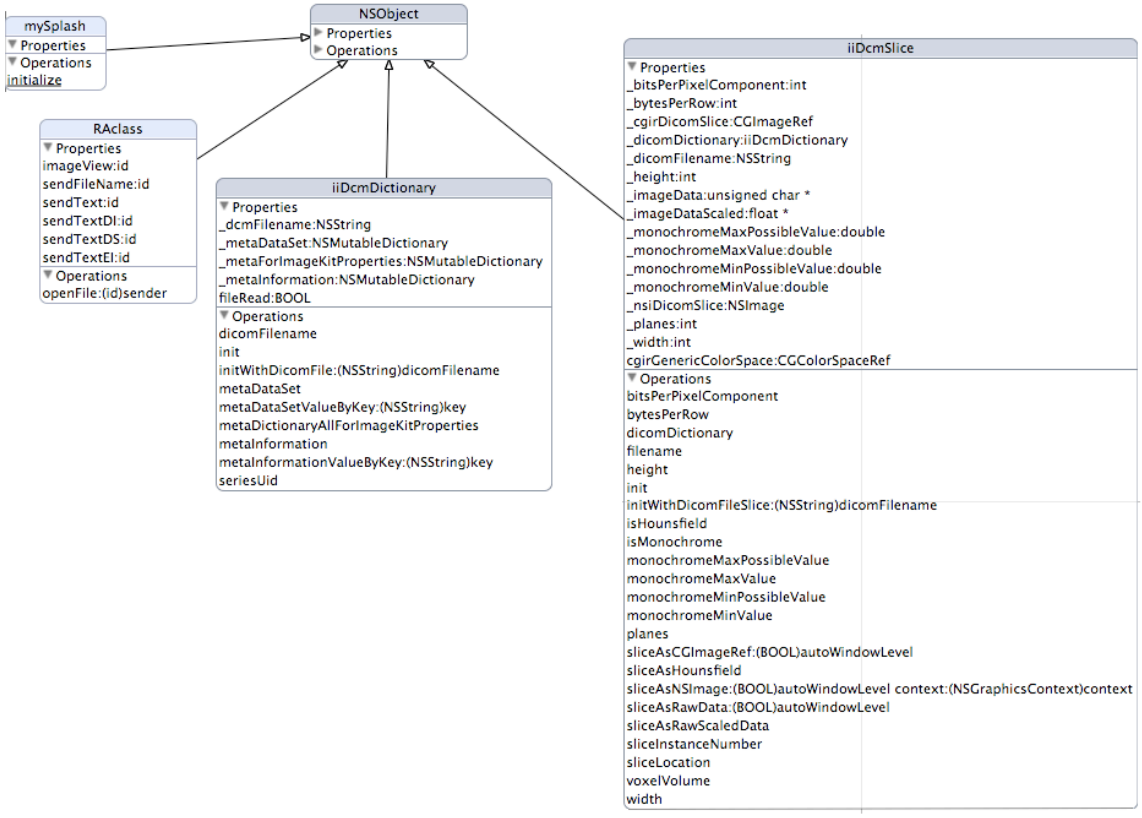

Figura 13- Diagrama de Classes

A classe *NSObject* é a classe raiz da biblioteca *Cocoa*, através dela os objetos herdam todas as funções básicas necessárias para que eles possam se comportar como objetos da linguagem *Objective-C*. A classe *mySplash* foi criada apenas para que uma tela de abertura seja inicializada antes do *software*. As classes *iiDcmDictionary* e *iiDcmSlice* são classes definidas na biblioteca *iiDICOM.framework* e possuem muitos atributos e métodos, porém poucos deles foram utilizados no projeto, como os métodos *metaDataSet* e *sliceAsNSImage*. A classe RAclass é a principal do código

7. Testes

"O teste de software é um elemento de um aspecto mais amplo, que é frequentemente referido como verificação e validação (V&V). Verificação se refere ao conjunto de atividades que garante que o software implementa corretamente uma função específica. Validação se refere a um conjunto de atividades diferentes que garantem que o software construído corresponde aos requisitos do cliente" (PRESSMAN, 2006, p. 289).

Verificação responde à pergunta: estamos construindo o produto corretamente? Enquanto validação responde a: estamos construindo o produto certo?

A etapa de testes pode ser dividida em 4 partes: teste de unidade, teste de integração, teste de validação e teste de sistema, e serão discutidos a seguir.

7.1 Teste de Unidade

"Ao desenvolver um grande sistema, o teste, geralmente, envolve vários estágios. Primeiro, cada componente do programa é testado, isolado dos outros componentes do sistema. Esse teste, conhecido como teste de módulo, teste de componente ou teste de unidade, verifica se o componente funciona de forma adequada aos tipos de entradas esperadas, a partir do estudo do projeto do componente. Sempre que possível, o teste de unidade é feito em um ambiente controlado, de modo que a equipe de teste possa fornecer ao componente a ser testado um conjunto de dados predeterminado, e a observar quais ações e dados são produzidos. Além disso, a equipe de testes verifica as estruturas de dados internas, a lógica e as condições limite para os dados de entrada e saída" (PFLEEGER, 2004, p. 275).

Primeiramente realizou-se uma inspeção do código. Uma vez que a descrição de projeto ajuda a codificar e documentar cada componente de programa, o seu programa reflete a sua interpretação do projeto. A documentação explica em palavras e figuras o que o programa deve fazer no código. Portanto, é útil que se revise objetivamente o código e sua documentação, a fim de eliminar interpretações errôneas, inconsistências e outros defeitos (PFLEEGER, 2004).

Os componentes testados nesta etapa foram: a interface gráfica, a estrutura de dados e as condições-limite.

7.1.1 Interface Gráfica

Segundo a análise dos requisitos documentada nos capítulos anteriores, era de grande importância que a interface gráfica apresentasse simplicidade, o que tornaria o programa mais amigável. Este teste tem, portanto, como objetivo verificar que esta simplicidade foi de fato alcançada pela codificação. A tela inicial do programa, que pode ser vista na Figura 12, transmite muito bem tal simplicidade. A interface é formada por apenas um botão (sem contar com os botões de fechar, minimizar e maximizar) que, ao ser pressionado, leva a interface para uma nova janela (Figura 13), que também é de fácil operação. Após a seleção do arquivo desejado, a imagem é mostrada de forma clara, e suas informações dispostas nos cantos da mesma, como foi especificado na análise de requisitos (Figura 14).

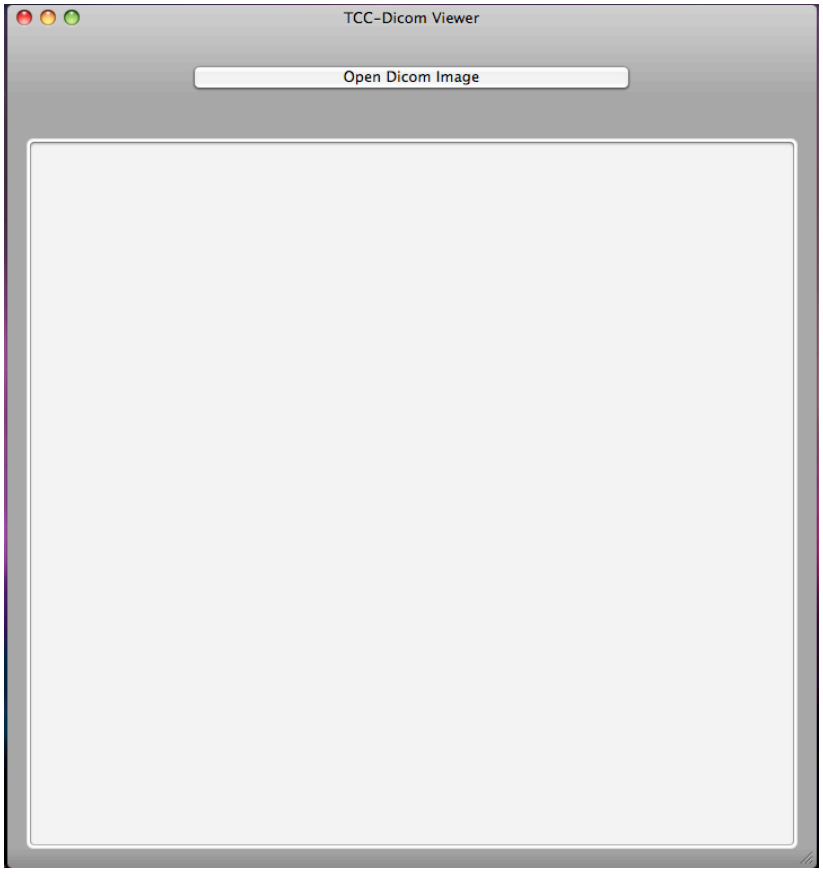

Figura 14 - Tela Inicial

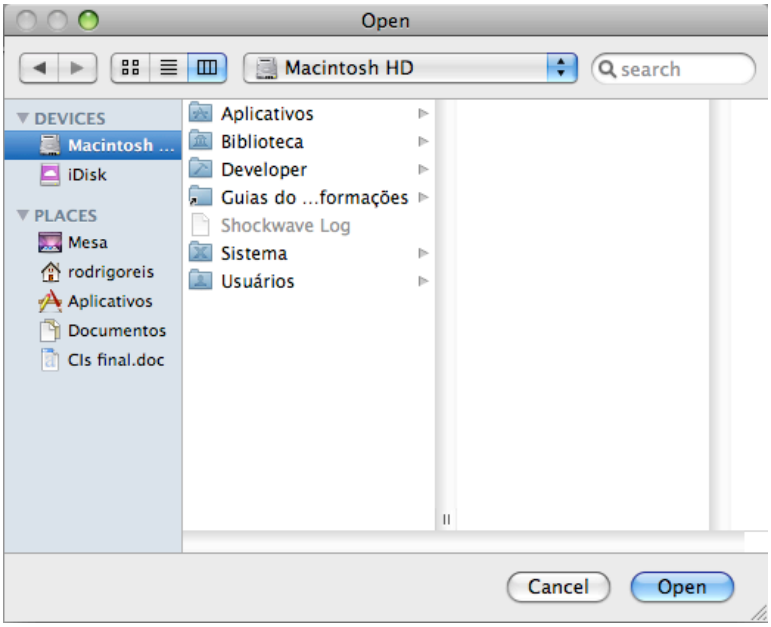

Figura 15 - Janela de Seleção de Arquivos

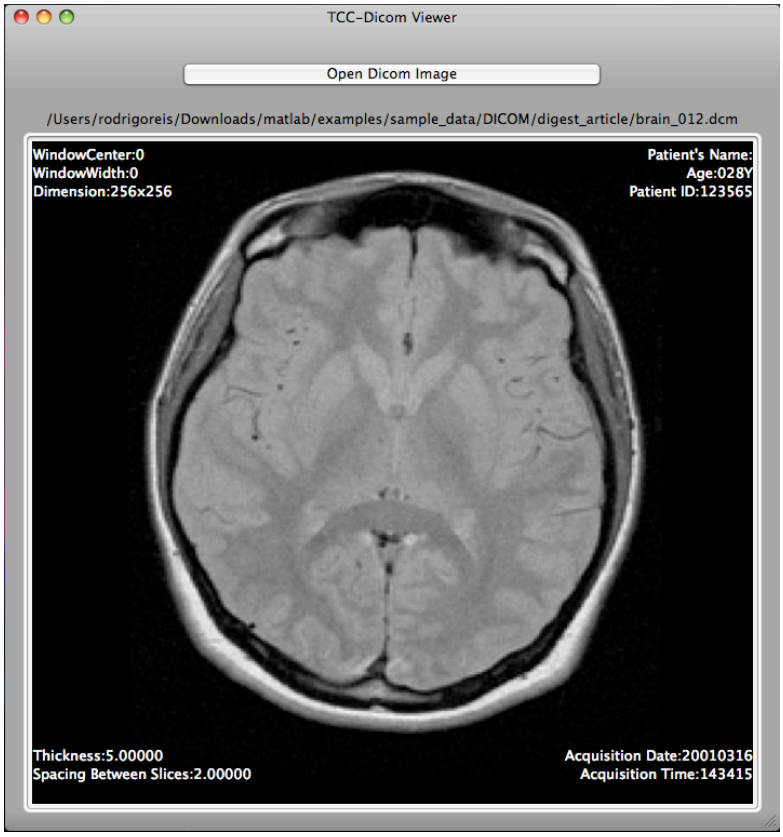

Figura 16 - Exibição da Imagem

7.1.2 Avaliação da Estrutura de Dados e das Condições-Limite

A estrutura de dados local é examinada para garantir que os dados armazenados temporariamente mantenham sua integridade durante todos os passos da execução de um algoritmo. Todos os caminhos independentes (caminhos básicos) ao longo da estrutura de controle são exercitados para garantir que todos os comandos de um módulo tenham sido executados pelo menos uma vez. As condições-limite são testadas para garantir que o módulo opere adequadamente nos limiares estabelecidos para limitar ou restringir o processamento (PRESSMAN, 2006).

A estrutura de dados foi testada utilizando diferentes tipos de arquivos DICOM, abrindo-os seguidamente ou após a reinicialização do *software* (Figuras 16 a 21).

A análise das condições limites também foi testada. Nela tentou-se abrir arquivos que não sejam suportados pelo *software* (Figura 15), o que não foi possível devido à implementação do código, que desabilita a seleção de arquivos que não sejam da extensão **dcm**, própria dos arquivos DICOM.

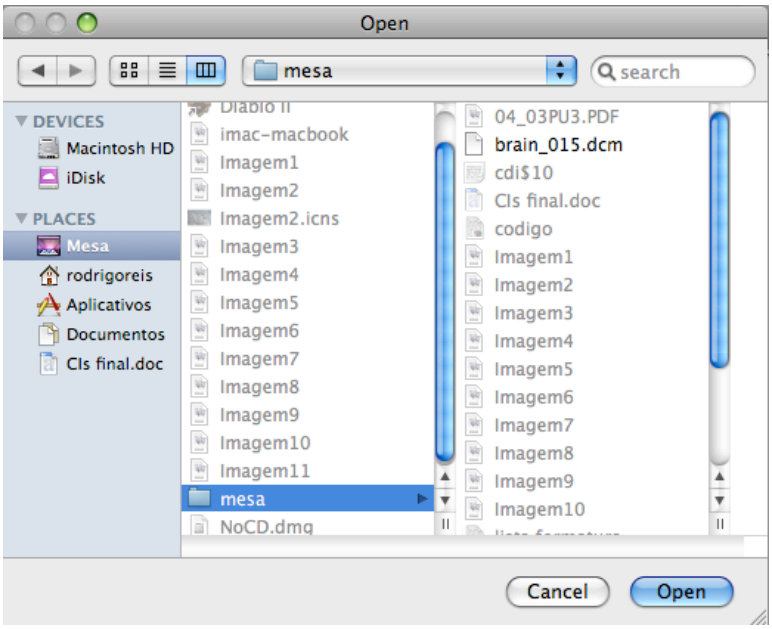

Figura 17 - Seleção Limitada a Arquivos dcm

Figura 18 – lesão do ligamento colateral medial

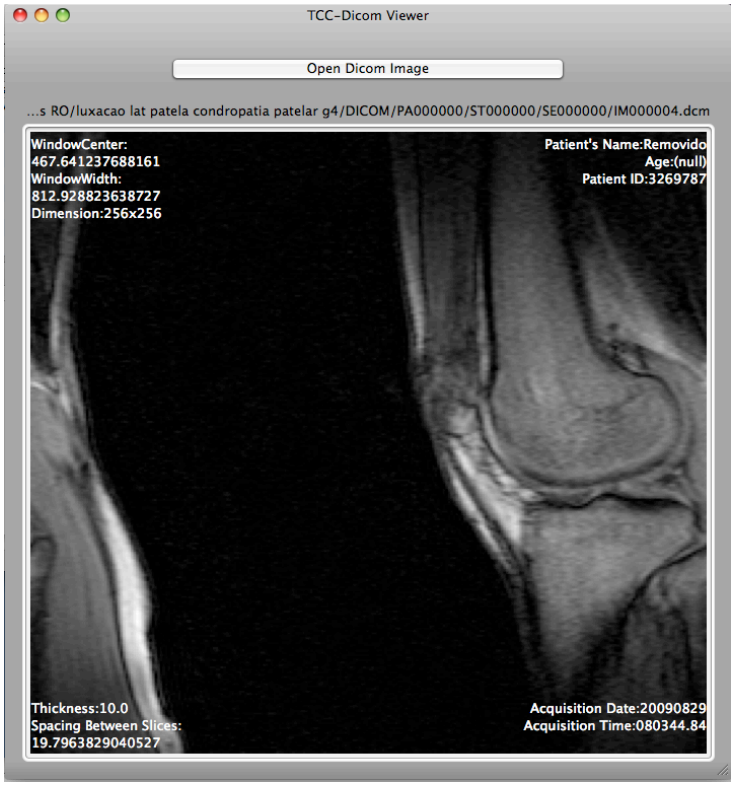

Figura 19 - luxação lateral da patela

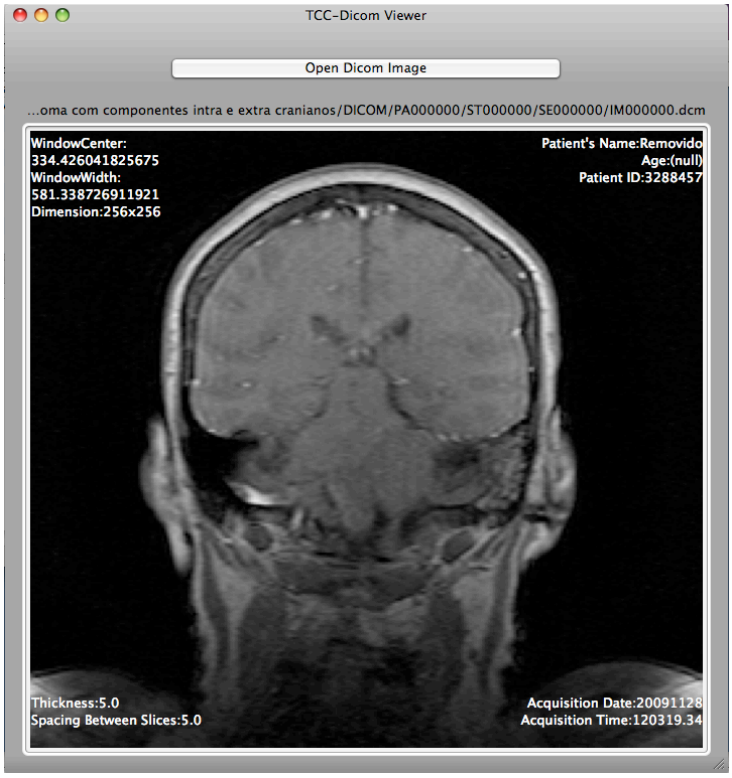

Figura 20 - meningeoma com componentes intra e extra cranianos

Figura 21 - necrose asséptica da cabeça femoral pós trauma

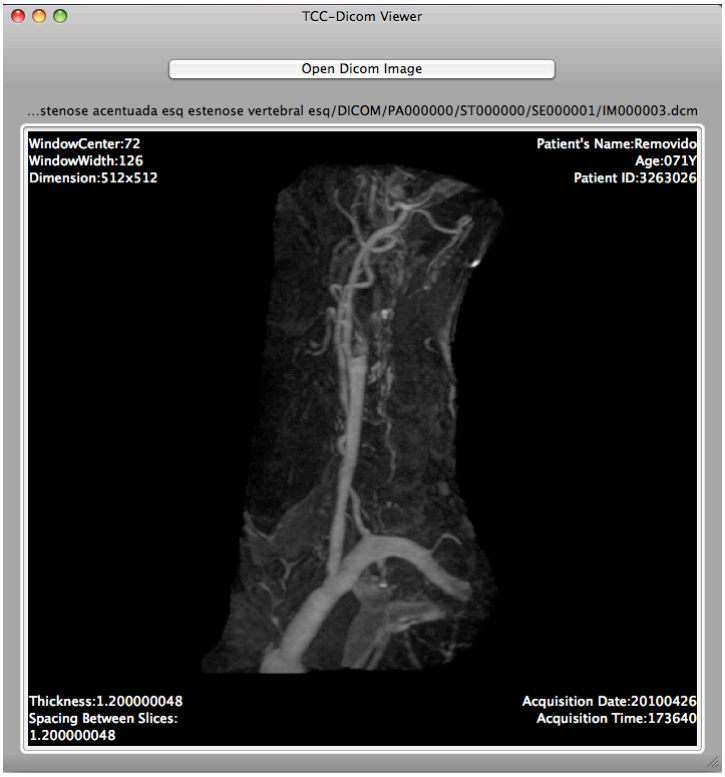

Figura 22 - oclusão da carótida direita

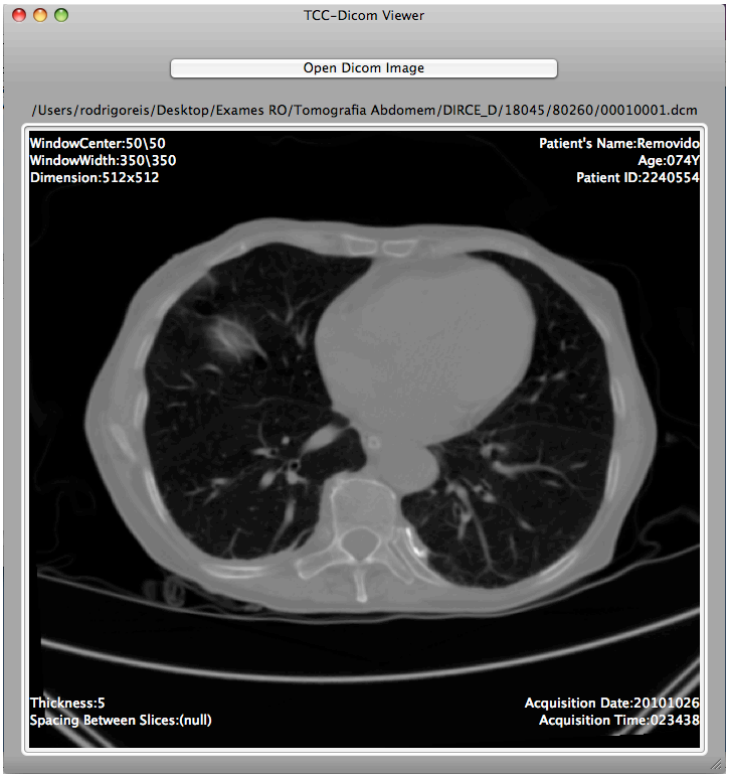

Figura 23 - tomografia do abdomem

7.2 Teste de Integração

Teste de integração é uma técnica sistemática para construir a arquitetura do *software* enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces. O objetivo é, a partir de componentes testados no nível de unidade, construir uma estrutura de programa determinada pelo projeto.

O teste de integração realizado nesta etapa foi do tipo incremental, o qual consiste em dividir em partes a verificação de erros do programa desenvolvido. Isso contribui para uma abordagem mais simplificada na correção de erros. Assim, as interfaces têm a possibilidade de serem inteiramente testadas e uma abordagem sistemática ao teste pode ser aplicada.

"A integração incremental é a antítese da abordagem big-bang. O programa é construído e testado em pequenos incrementos, em que erros são mais fáceis de isolar e corrigir; é mais provável que as interfaces sejam testadas completamente e pode ser aplicada uma abordagem sistemática de teste" (PRESSMAN, 2006, p. 297).

7.3 Teste de Validação

Validação pode ser definida de vários modos, mas uma definição simples (no entanto, rigorosa) é que ela se torna bem sucedida quando o *software* funciona de um modo que pode ser razoavelmente esperado pelo cliente.

Diante da análise de requisitos e sistemas propostos no começo desse documento, fez-se a verificação de congruência entre o *software* pronto e esses requisitos observados. Desde o início, a intenção do *software* era de se constituir numa ferramenta simples para a visualização de arquivos DICOM em sistema operacional Mac OS X. Como projetado, o programa atende a todas essas especificações, apresentando tudo em uma interface amistosa e simples.

7.4 Teste de Sistema

"Teste de sistema é, na verdade, uma série de diferentes testes cuja finalidade principal é exercitar por completo o sistema baseado em computador. Apesar de cada teste ter uma finalidade distinta, todos trabalham para verificar se os elementos do sistema foram adequadamente integrados e executam as funções a eles alocadas" (PRESSMAN, 2006, p. 306).

Assim, fez-se a análise do *software* como um elemento que faz parte de um sistema mais amplo. Neste caso, o programa ainda tem limitações quanto às suas funções; porém, como estas já estavam definidas na análise de requisitos, o *software* desempenha satisfatoriamente o papel para o qual foi projetado.

8. Manutenção e Qualidade

Esta fase de desenvolvimento de *software* envolve toda e qualquer modificação feita após ele estar pronto. Qualquer correção de erro ou nova funcionalidade adicionada é considerada uma atividade de manutenção. De tal modo, esta etapa não tem um fim definido enquanto o *software* está sendo utilizado pelo usuário. Usualmente diz-se que o gasto com manutenção de sistemas existentes é deveras maior do que com o desenvolvimento de novos sistemas.

A manutenção corretiva, que é aquela feita para corrigir pequenos erros no funcionamento do programa, será feita através do contato com o usuário. Estes erros não costumam ser muito complexos e tendem a ser poucos quando o programa passa por uma etapa de testes bem realizada. No caso deste *software*, espera-se uma baixa incidência destes erros enquanto estiver funcionando em seu formato atual.

A classe de manutenções adaptativas tem como objetivo atualizar o *software* para que ele possa continuar operando em sistemas e ambientes novos. A presente ferramenta tem em seu projeto conceitual o requisito de ser futuramente adaptado aos sistemas operacionais dos dispositivos móveis portáteis iPhone, iPod e iPad. Portanto, esta etapa da manutenção passa a ser imprescindível para a completa realização deste projeto.

O sistema operacional destes dispositivos, o IOS4 (também desenvolvido pela *Apple*), tem grande semelhança com o Mac OS X; portanto, esta adaptação pode ser totalmente executada, bastando para isso apenas alguns estudos adicionais sobre o IOS4. A programação em Objective-C é totalmente aceita pelos dispositivos, e é também possível criar aplicativos para eles, utilizando o XCode. Assim, nenhuma alteração no projeto deverá ser feita, sendo necessárias apenas algumas alterações na codificação.

Outra manutenção que está prevista neste projeto é a inclusão de novas funções para o *software*, como a opção de manipulação das imagens (brilho, contraste, zoom, etc). Como o programa foi desenvolvido de maneira clara e simples, a implementação destas funções não será trabalhosa e deverão tomar lugar nas próximas versões do aplicativo.

O principal objetivo da Engenharia de Software é ajudar a produzir *software* de qualidade; empresas que fazem isso são mais competitivas e podem, em geral, oferecer um melhor serviço a um preço mais competitivo também.

A noção de qualidade de *software* pode ser descrita por um grupo de fatores, requisitos ou atributos, tais como: confiabilidade, eficiência, facilidade de uso, modularidade, legibilidade, etc. Podemos classificar estes fatores em dois tipos principais: externos e internos.

Os fatores externos de qualidade são verificados principalmente pelo pessoal que irá utilizar no dia-a-dia o *software* projetado. No caso do projeto, os médicos farão as propostas de mudança, aprimorando-o e trazendo uma melhora no seu desempenho e confiabilidade.

Já os fatores internos de qualidade, foram verificados (legibilidade, portabilidade, modularidade) durante o processo da criação, desde a parte de análise e especificação de requisitos até a efetuação dos testes propriamente ditos.

Os fatores que afetam a qualidade de *software* são divididos em três categorias: revisão do produto, transição do produto e operação do produto.

Revisão do Produto

Os três fatores que foram verificados na revisão foram: a possibilidade de manutenção, que é conseguida na alteração do código-fonte que está organizado de maneira lógica, a capacidade de mudanças e adaptações no *software* conforme dito em relação à manutenção e à possibilidade da realização de testes de cada função, e das variáveis que representam as entradas geradas pela interface com o usuário.

Transição do Produto

O *software*, uma vez que foi criado em linguagem de alto nível Objective-C, pode ser executado nos sistemas operacionais existentes nos computadores *Apple* atuais com o sistema operacional Mac OS X 10.5. Para a execução em sistemas operacionais futuros (como o Mac OS X 10.6), existirá a necessidade de adaptações, as quais não comprometeriam a sua integridade.

Operação do Produto

Em relação à operação, o *software* corresponde aos requisitos esperados, uma vez que o mesmo se propõe a realizar as tarefas que atingem as necessidades básicas de visualização de imagens DICOM, e o *software* também pode ser considerado eficiente, pois é executado com precisão dentro dos limites especificados nas etapas anteriores de projeto.

9. Conclusões

A proposta inicial deste trabalho sofreu, ao longo do seu desenvolvimento, algumas modificações que foram necessárias para definir com clareza o rumo do projeto. Algumas delas se devem a problemas encontrados no seu decorrer que não foram identificados em sua idealização, outras devidas a ajustes essenciais para o seu desenvolvimento. Porém, a ideia central do projeto foi mantida e realizada com sucesso.

Durante o período de trabalho, várias etapas foram construídas e conhecimentos em diversas áreas adquiridos, desde o estudo da formação de imagens digitais, a teoria de projetos de *softwares*, a programação poderosa em Objective-C e o gerenciamento de projetos em si. Algumas destas áreas estavam mais facilmente ao alcance, já outras necessitaram de intensa pesquisa e estudo.

Primeiramente, o padrão DICOM teve de ser estudado e esclarecido, para que então fosse possível enxergar com clareza os próximos objetivos. Passou-se então à elaboração de alguns protótipos do programa, para verificar a real possibilidade da realização do projeto e entender mais profundamente o ambiente onde o *software* seria criado. O próximo passo foi a realização das pesquisas de requisitos, quando o programa sofreu a maior parte de suas modificações, chegando a um patamar que pudesse ser corretamente realizado. Estes requisitos, embora não sejam o ideal para um *software* nesta área, formaram uma base que, através da clareza e modularidade com que o projeto foi estruturado, poderão ser facilmente incrementados para tornar o projeto uma ferramenta bastante útil na área.

Por fim, dentro da idéia proposta e tempo disponível, foram atingidos os objetivos esperados e, o mais importante, obtiveram-se indicativos promissores. Os resultados obtidos criaram expectativas otimistas para a continuidade do projeto e deixaram uma base para tal, não sendo necessários outros recursos senão os já utilizados aqui. Com a vantagem de se constituir, então, um recurso de *software* livre para visualização de arquivos DICOM em qualquer computador que utilize o sistema operacional para o qual foi desenvolvido (Mac OS X), o que é um ganho significativo em função da existência desse tipo de recurso apenas a custos elevados – provenientes de fabricantes estrangeiros.

10. Referências Bibliográficas

Object-Oriented Programming with Objective-C, 2008. Disponível em: <http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC.pdf> Acesso em: 13 mai. 2010

Pfleeger, Shari Lawrence. **Engenharia de software**: teoria e prática / Shari Lawrence Pfleeger [tradução de] Dino Franklin. 2. ed. São Paulo: Prentice Hall/Pearson Education, 2004. 535 p.

Pressman, Roger S.. **Engenharia de software / Roger S. Pressman** ; tradução [de] Rosângela Delloso Penteado ; revisão técnica [de] Fernão Stella R. Germano, José Carlos Maldonado, Paulo Cesar Masiero. 6. ed. São Paulo : McGraw-Hill, c2006. 720 p.

Thomas, Rebecca; Yates, Jean**. Unix** : guia do usuario / Rebecca Thomas, Jean Yates ; [Trad] Maria Claudia de Oliveira Santos. 2. ed. São Paulo : McGraw-Hill, 1989. 744 p.

Digital Imaging and Communications in Medicine (DICOM). Rosslyn: National Electrical Manufacturers Association, 2004.

Stallings, William. **Operating systems** : internals and design principles / William Stallings. 3. ed. Upper Saddle River, N.J. : Prentice Hall, 1998. 781 p.

11. Anexos

11.1 Anexo 1: O padrão DICOM

Com a introdução da Tomografia Computadorizada (CT) seguido de outras modalidades de diagnósticos digitais em 1970, e com o crescimento do uso de computadores em aplicações clínicas, a ACR (*American College of Radiology*) e a NEMA *(National Eletrical Manufacturers Association*) reconheceu a emergente necessidade de um método padronizado de transferência de imagens e informações associadas entre os dispositivos fabricados por vários fornecedores. Esses dispositivos produziam uma grande variedade de formatos de imagens digitais.

A ACR e a NEMA formaram um comitê conjunto em 1983 para desenvolver um padrão para:

Promover a comunicação de informações de imagens digitais, independentemente do fabricante do dispositivo

Facilitar o desenvolvimento e expansão do arquivamento de imagens e sistemas de comunicação (PACS), que também possam interagir com outros sistemas de informações hospitalares

Permitir a criação de bases de dados de informações de diagnósticos que possam ser acessados por uma grande variedade de dispositivos distribuídos geograficamente.

Este padrão, que é atualmente designado *Digital Imaging and Communications In Medicine* (DICOM), engloba várias melhorias importantes para as versões anteriores do padrão ACR-NEMA:

É aplicável para um ambiente de rede. O padrão ACR-NEMA era aplicável para um ambiente ponto a ponto apenas; para a operação em ambiente de rede era necessário uma *Network Interface Unit* (NIU). DICOM suporta a operação em um ambiente de rede usando o protocolo padrão de redes TCP/IP

É aplicável em um ambiente de mídia *off-line*. As versões anteriores do padrão não especificavam um formato de arquivo ou escolhiam uma mídia física ou arquivo de sistema lógico. DICOM usa para isso mídias industriais padrões como CD-R e MOD e arquivos de sistema lógico como ISO 9660 e sistema de arquivo PC FAT 16.

Especifica como os dispositivos que acessam o padrão reagem aos comandos e dados que são transferidos ou movimentados. Os padrões anteriores eram confinados a transferência de dados, mas o DICOM especifica, através do conceito de Classes de Serviço, as semânticas dos comandos e dados associados.

Especifica níveis de conformidade (aderência ao padrão). Os padrões anteriores apenas especificavam um

nível mínimo de conformidade. DICOM descreve explicitamente como um implementador deve estruturar uma Declaração de Conformidade (Aderência) para selecionar opções específicas.
É estruturado como um documento de múltiplas partes. Isso facilita a evolução do padrão em um ambiente de mudanças rápidas, simplesmente adicionando novos recursos. As instruções definidas pela ISO de como estruturar um documento de múltiplas partes foram seguidas na construção do padrão.
Introduz Objetos de Informações explícitas, nao só para imagens e gráficos, mas também para formas de onda, relatórios, impressão, etc.
Especifica uma técnica estabelecida para indentificar unicamente qualquer Objeto de informação.

Definições:

Atributo: Uma propriedade de um Objeto de Informação. Um atributo tem um nome e um valor, que são independentes de qualquer esquema de codificação.

Comando: Um pedido para operar em uma informação através da rede

Elemento de Comando: Uma codificação de um parâmetro de um comando que transmite o valor deste parâmetro.

Canal de Comando: O resultado da codificação de um conjunto de Elementos de comando DICOM usando o esquema de codificação DICOM.

Declaração de Conformidade: Uma declaração formal que descreve a implementação de um produto específico que usa o padrão DICOM. Especifica a Classe de Serviços, Objetos de Informação, e Protocolos de Comunicação suportados pela implementação.

Dicionário de Dados: Um registro de Elementos de Dados DICOM, o qual atribui uma única *Tag*, um nome, características de valor, e semânticas de cada Elemento de Dado.

Elemento de Dado: Uma unidade de informação definida por uma entrada única do dicionário de dados.

Conjunto de Dados: Informações trocadas que consistem em um conjunto estruturado de Atributos. O valor de cada Atributo do conjunto de Dados é expresso como um Elemento de Dado.

Canal de Dados: O resultado da codificação de um Conjunto de Dados usando o esquema de codificação DICOM.

Objeto de Informação: Uma abstração de alguma informação real.

Classe de Objeto de Informação: Uma descrição formal de um Objeto de Informação, que inclui uma descrição do seu propósito e os respectivos Atributos que este possui. Não inclui os valores desses Atributos.

Instância de Objeto de Informação: Inclui os valores dos Atributos da Classe de Objeto de Informação.

Mensagem: Uma unidade de dado do Protocolo de Trocas de Mensagens, que é trocada entre dois aplicativos DICOM. A mensagem é composta de um Canal de Comando seguido por um Canal de Dados opcional.

Classe de Serviços: Uma descrição estruturada de um serviço que é suportado por aplicativos DICOM cooperantes usando comandos DICOM específicos e agindo em uma específica classe de objeto de informação.

São definidos dois tipos de Classe de Objeto de Informação:

Classe de Objeto de Informação Normalizada inclui somente os atributos inerentes a entidade do mundo real representada, por exemplo, data do estudo ou hora do estudo; o nome do paciente é inerente ao paciente e não ao estudo, por isso não é incluída nesta classe.

Classe de Objeto de Informação Composta pode incluir adicionalmente atributos que estão relacionados mas não são inerentes à entidade do mundo real.

Para representar a ocorrência de uma entidade do mundo real, uma instância de Objeto de Informação é criada, a qual inclui valores para os atributos da Classe de Objeto de Informação. Este valor do atributo pode mudar durante o tempo para representar precisamente as mudanças de estado da entidade a qual ela representa. Isto é conseguido fazendo-se diferentes operações básicas sobre a instância para apresentar um conjunto de serviços definidos como uma Classe de Serviços.

Uma Classe de Serviços associa um ou mais Objetos de Informação com um ou mais Comandos a serem realizados sobre esses objetos.

Objetivos do Padrão DICOM:

O Padrão DICOM facilita a interoperabilidade de dispositivos solicitando conformidade. Em particular, ele:

Endereça a semântica de comandos e dados associados. Para que aconteça a interação entre os dispositivos, devem existir padrões de como é esperado que estes dispositivos reajam a comandos e dados associados, e não somente sobre a informação que será trocada pelos dispositivos.
Endereça a semântica de serviços de arquivos, formatos de arquivos e diretórios necessários para a comunicação *off-line*.
É explícito em definir os requerimentos para conformidade das implementações do padrão. Em particular, uma declaração de conformidade deve especificar informações suficientes para determinar as funções que são esperadas na interoperabilidade com outro dispositivo.
Facilita operação em ambientes de rede.
É estruturado de maneira a acomodar a introdução de novos serviços, dando suporte à futuras aplicações em imagens médicas.

Este padrão foi desenvolvido com ênfase em imagens médicas diagnósticas como são usadas em radiologia, cardiologia e disciplinas relacionadas; no entanto, é também aplicável a várias outras comunicações relacionadas à clínicas e ambientes médicos.

A figura 23 apresenta um modelo básico e geral do padrão, cobrindo tanto a *comunicação on-line* (ambiente de rede) como a *off-line* (armazenamento em arquivos).

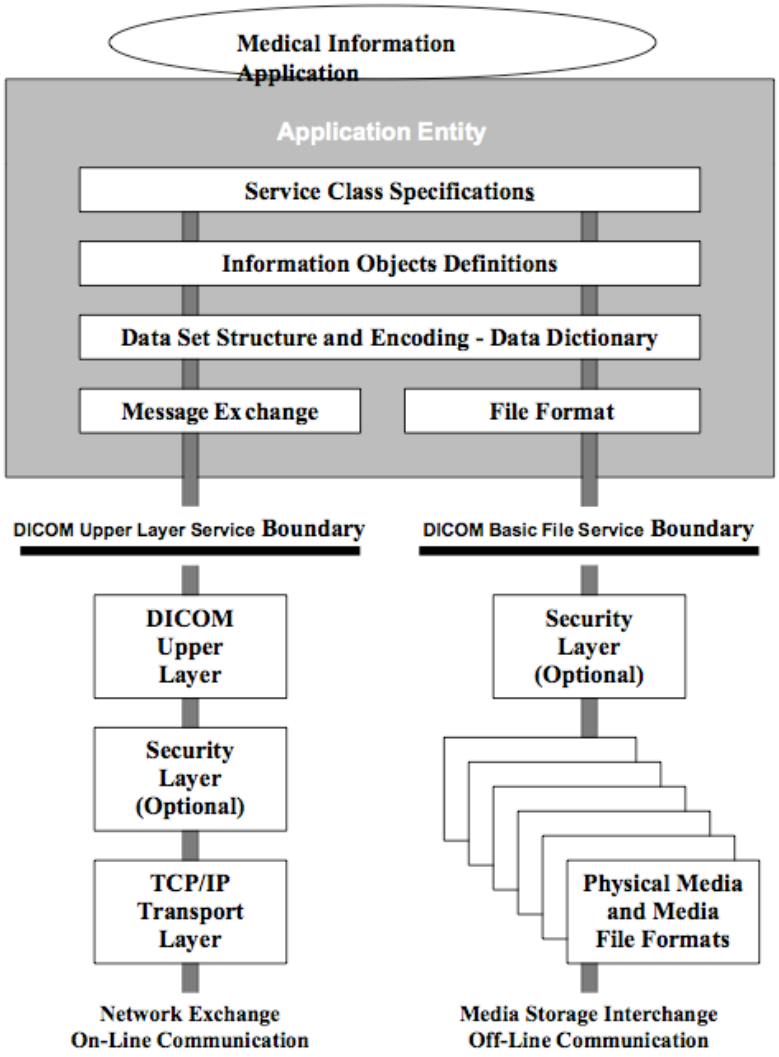

Figura 24 - Modelo geral do padrão (Digital Imaging and Communications in Medicine, 2004)

Estrutura de Dados e Semântica:

O Padrão DICOM especifica como os aplicativos DICOM constroem e codificam as informações de um conjunto de dados que resulta do uso dos Objetos de Informação e Classes de Serviços. É também especificado o suporte a diversas técnicas padrões de compressão de imagem, como JPEG com perdas e sem perdas. São definidas também a semântica de inúmeras funções que são comuns a vários Objetos de Informação.

Dicionário de Dados:

Todos os elementos de dados DICOM disponíveis para representar alguma informação são definidos no Dicionário de Dados, junto com os elementos utilizados na codificação de mídia transferível e uma lista de ítens de identificação única associados pelo DICOM.

Para cada elemento é especificado:

Sua *tag*, que consiste em um grupo e um número de elemento
Seu nome
Seu valor representativo (*string*, inteiro, etc)
A multiplicidade do seu valor (quantos valores por atributo)
Se já foi retirado do padrão

Para cada ítem de indentificação única é especificado:

Seu valor único, que é numérico com componentes múltiplos separados por pontos decimais, limitado a 64 caracteres
Seu nome
Seu tipo
Em que parte do Padrão DICOM ele está definido

Uma parte da especificação deste dicionário está mostrado na Figura 24.

Tag	Name	VR	VM	
(0008,0001)	*Length to End*			*RET*
(0008,0005)	Specific Character Set	CS	1-n	
(0008,0008)	Image Type	CS	1-n	
(0008,0010)	*Recognition Code*			*RET*
(0008,0012)	Instance Creation Date	DA	1	
(0008,0013)	Instance Creation Time	TM	1	
(0008,0014)	Instance Creator UID	UI	1	
(0008,0016)	SOP Class UID	UI	1	
(0008,0018)	SOP Instance UID	UI	1	
(0008,001A)	Related General SOP Class UID	UI	1-n	
(0008,001B)	Original Specialized SOP Class UID	UI	1	
(0008,0020)	Study Date	DA	1	
(0008,0021)	Series Date	DA	1	
(0008,0022)	Acquisition Date	DA	1	
(0008,0023)	Content Date	DA	1	
(0008,0024)	Overlay Date	DA	1	
(0008,0025)	Curve Date	DA	1	
(0008,002A)	Acquisition Datetime	DT	1	
(0008,0030)	Study Time	TM	1	

Figura 25 - Dicionário de Dados (Digital Imaging and Communications in Medicine, 2004)

Função de Visualização de Imagens em Tons de Cinza:

Esta função fornece métodos de calibração de sistemas de display particulares com o propósito de apresentar as imagens consistentemente em diferentes *displays* (monitores e impressoras).

A função escolhida é baseada na percepção visual humana, que possui sensibilidade não linear dentro da faixa de luminância dos dispositivos de visualização, portanto foi utilizado um modelo de sistema visual humano.