UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

TÉCNICAS DE REENGENHARIA DE SOFTWARE APLICADAS EM UMA FERRAMENTA DE REPRESENTAÇÃO 3D DA DISTRIBUIÇÃO DE CORES DE IMAGENS DIGITAIS

TRABALHO DE GRADUAÇÃO

Eduardo Spolaor Mazzanti

Santa Maria, RS, Brasil

2008

TÉCNICAS DE REENGENHARIA DE SOFTWARE APLICADAS EM UMA FERRAMENTA DE REPRESENTAÇÃO 3D DA DISTRIBUIÇÃO DE CORES DE IMAGENS DIGITAIS

por

Eduardo Spolaor Mazzanti

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientador: Profª Drª Marcos Cordeiro d'Ornellas

Trabalho de Graduação Nº 270
Santa Maria, RS, Brasil

2008

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

TÉCNICAS DE REENGENHARIA DE SOFTWARE APLICADAS EM UMA FERRAMENTA DE REPRESENTAÇÃO 3D DA DISTRIBUIÇÃO DE CORES DE IMAGENS DIGITAIS

elaborado por
Eduardo Spolaor Mazzanti

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Profª Drª Marcos Cordeiro d'Ornellas
(Presidente/Orientador)

Prof. Dr. José Antônio Trindade Borges da Costa (UFSM)

Prof. Msc. Oni Reasilvia de O. Sichonany (UFSM)

Santa Maria, 17 de dezembro de 2008.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

TÉCNICAS DE REENGENHARIA DE SOFTWARE APLICADAS EM UMA FERRAMENTA DE REPRESENTAÇÃO 3D DA DISTRIBUIÇÃO DE CORES DE IMAGENS DIGITAIS

Autor: Eduardo Spolaor Mazzanti
Orientador: Profª Drª Marcos Cordeiro d'Ornellas
Local e data da defesa: Santa Maria, 17 de dezembro de 2008.

Sistemas de software evoluem em resposta às exigências de mudanças como, por exemplo, correção de erros, melhorias no desempenho, migrações para novas plataformas ou outras características. A reengenharia de software pode ser uma maneira viável de garantir que sistemas possam continuar em uso, pois ela recupera as informações de projeto de um software existente e as utiliza para modificar o sistema, preservando suas funcionalidades. Neste contexto, este trabalho objetiva o estudo e a aplicação prática de técnicas de reengenharia de software através da reestruturação de uma ferramenta computacional para a representação da distribuição de cores de imagens digitais em diferentes espaços de cores. Ainda, busca preservar as funções da ferramenta, reutilizando os esforços de desenvolvimento passados. A ferramenta reengenheirada será integrada ao sistema de processamento e análise de imagens Anima, que é desenvolvido pela Animati Computação Aplicada, empresa incubada na Incubadora Tecnológica de Santa Maria, a qual é parceira do Laboratório de Computação Aplicada (LaCA) da Universidade Federal de Santa Maria.

Palavras-chave: Reengenharia de software, engenharia reversa, espaço de cores.

ABSTRACT

Graduation Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

SOFTWARE REENGINEERING TECHNIQUES APPLIED IN A TOOL FOR THE 3D REPRESENTATION OF THE COLOR DISTRIBUTION ON DIGITAL IMAGES

Author: Eduardo Spolaor Mazzanti
Advisor: Prof[a] Dr[a] Marcos Cordeiro d'Ornellas

Software systems evolve in response to demands for changes such as errors correction, improvements in performance, migration to new platforms or other characteristics. Software reengineering can be a viable way to ensure that systems can continue in use, as it retrieves the information from an existing software design and uses to modify the system, preserving its functionality. In this context, this work intends the study and the practical application of software reengineering techniques through the restructuration of a computational tool for the representation of the color distribution on digital images on different color spaces. Also, it tries to preserve the functions of this tool by making use of the past development efforts. The reengineered tool will be integrated on the image processing and analysis system Anima, that is developed by the Animati Computação Aplicada, company situated on the Incubadora Tecnológica de Santa Maria, which is a partner of the Laboratório de Computação Aplicada (LaCA) from the Federal University of Santa Maria.

Keywords: Software reengineering, reverse engineering, color space.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AWT	Abstract Windows Toolkit
CIE	Commission International de L'Eclairage
CMYK	Cyan, Magenta, Yellow, Black
GUI	Graphical User Interface
HSI	Hue, Saturation, Intensity
IDE	Integrated Development Environment
ImageIO	Java Image Input/Output API
JAI	Java Advanced Imaging
JOGL	Java OpenGL
LaCA	Laboratório de Computação Aplicada da Universidade Federal de Santa Maria
RGB	Red, Green, Blue
SGDB	Sistema Gerenciador de Banco de Dados
UML	Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

A partir do momento que um sistema de software começa a ser utilizado ele entra em um processo contínuo de mudança. Mesmo que ele tenha sido desenvolvido aplicando boas técnicas de projeto e codificações existentes, os sistemas vão se tornando obsoletos devido a novas tecnologias que são utilizadas.

O tempo de duração de sistemas de software possui grande variação, e muitos sistemas de grande porte permanecem em uso por longos períodos. Algumas organizações ainda dependem desses sistemas antigos, pois estes são responsáveis pela realização de serviços fundamentais e qualquer falha desses serviços teria um sério efeito no seu dia-a-dia. Assim, as organizações precisam mantê-los em operação.

Os sistemas de software sempre evoluem em resposta às exigências de mudanças como, por exemplo, correção de erros, melhorias no desempenho, migrações para novas plataformas ou outras características não funcionais (SOMMERVILLE, 2003). Segundo Warren (WARREN, 1999), existem diversas estratégias para mudanças em software, como: manutenção do software, a transformação da arquitetura e a reengenharia de software, que será a estratégia adotada neste trabalho. A reengenharia de software pode ser uma maneira viável de garantir que sistemas possam continuar em uso. Pode ser muito dispendioso e arriscado seguir outra abordagem para a evolução do sistema (SOMMERVILLE, 2003). A completa substituição ou a reestruturação radical de um sistema pode ser financeiramente impensável pela maioria das organizações. A manutenção de velhos sistemas é cada vez mais custosa e a reengenharia desses sistemas pode prolongar seu tempo de vida útil.

Assim, o objetivo deste trabalho é o estudo e a aplicação de técnicas de reengenharia de software através da reestruturação de uma ferramenta computacional para a representação da distribuição de cores de imagens digitais em diferentes espaços de cores (DA-

RONCO, 2006) , para que este seja estruturado e incorporado ao sistema de processamento e análise de imagens do Laboratório de Computação Aplicada (LaCA) da Universidade Federal de Santa Maria (UFSM), que é desenvolvido em parceria com a Animati Computação Aplicada, empresa incubada na Incubadora Tecnológica de Santa Maria.

A ferramenta que sofrerá o processo de reengenharia foi desenvolvida de acordo com os padrões do antigo sistema de processamento e análise de imagens do LaCA, chamado *Arthemis*. Com o desenvolvimento de um novo sistema, chamado *Anima*, que possui uma nova estrutura, existe a necessidade que esta ferramenta seja reestruturada, de modo que ela seja integrada na arquitetura do novo sistema. A reengenharia apresenta-se como uma solução para esta questão, pois através dela poderá chegar-se, de um modo eficaz e sem altos custos, a uma adequação da estrutura da ferramenta. A recuperação ou renovação do software faz com que se obtenha mais possibilidades de continuidade do serviço e manutenção do sistema.

Considerando o cenário acima, este trabalho de graduação apresenta a seguinte organização: o capítulo 2 apresenta uma revisão bibliográfica a respeito do processo de reengenharia de software, apresentando as atividades presentes nesse processo. Ainda neste capítulo, é apresentada uma visão geral da ferramenta que sofrerá a reengenharia. O capítulo 3 descreve o processo de desenvolvimento da ferramenta, falando dos materiais e métodos aplicados como também a sua implementação. O capítulo 4 mostra os resultados alcançados com o projeto bem como uma avaliação sobre as atividades realizadas. Por fim, no capítulo 5 é apresentada a conclusão deste trabalho.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo apresenta-se uma revisão bibliográfica sobre os tópicos relacionados a este trabalho. Primeiramente serão mostradas questões referentes à reengenharia de software, bem como as atividades que a compõe. Será tratado também sobre manipulação de cores em sistemas de imageamento, bem como será apresentada a ferramenta que é alvo da reengenharia deste trabalho.

2.1 Reengenharia de Software

A reengenharia de software, que pode também ser chamada de recuperação ou renovação, é o processo de recuperar as informações de projeto de um software existente e as utilizar para modificar ou restaurar o sistema, preservando suas funcionalidades ao mesmo tempo em que pode adicionar outras novas, visando a melhoria de sua qualidade global (PRESSMAN, 1995). Para Premerlani e Blaha (PREMERLANI; BLAHA, 1994) o objetivo da reengenharia é reutilizar de forma automática os esforços de desenvolvimentos passados, pretendendo reduzir os custos de manutenção e melhoria na flexibilidade do software.

É possível perceber uma nítida diferenciação entre o desenvolvimento de um novo software e a reengenharia de software. A distinção encontra-se no que se refere ao ponto de partida de cada processo. Enquanto que o desenvolvimento de um software ocorre através de uma especificação escrita do que ainda será construído, a reengenharia tem o seu início em um sistema já existente. Para distinguir estas duas abordagens, Chikofsky e Cross (CHIKOFSKY; CROSS, 1990) chamam o desenvolvimento de um novo software de engenharia progressiva. Esta diferenciação pode ser vista na figura 2.1.

De forma complementar, Sommerville (SOMMERVILLE, 2003) afirma que a reengenharia de software é a tarefa de reimplementar sistemas de software, para que estes

Figura 2.1: Engenharia progressiva e reengenharia (Adaptado de Sommerville, 2003)

tenham uma manutenção mais fácil. Esta tarefa pode envolver a redocumentação, organização e reestruturação do sistema, tradução da linguagem de programação, modificação e atualização da estrutura e dos valores dos dados do sistema. Para Chikofsky e Cross (CHIKOFSKY; CROSS, 1990), a reengenharia, também conhecida como renovação e reconstrução, consiste no exame e alteração do sistema para reconstruí-lo numa nova forma e numa subseqüente implementação dessa nova forma.

Acrescentando a essas definições, Warden (WARDEN, 1992) cita que a reengenharia tem como principal objetivo melhorar o sistema de alguma maneira, através de mudanças que provoquem uma melhoria, contudo, sem modificar suas funções. A figura 2.2 ilustra o processo de reengenharia.

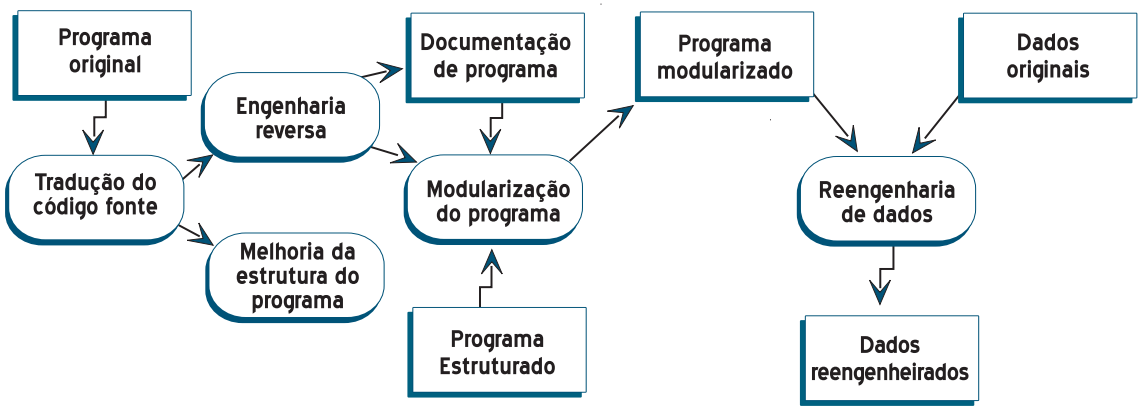

Figura 2.2: Processo de reengenharia (Adaptado de Sommerville, 2003)

Em um processo de reengenharia, a entrada é um software já existente e a saída é uma versão estruturada e modularizada do mesmo software. Simultaneamente com a reengenharia do software, os dados do sistema também podem passar por uma reenge-

nharia. Sommerville (SOMMERVILLE, 2003) cita que as atividades nesse processo de reengenharia são: Tradução do código-fonte, Engenharia reversa, Melhoria da estrutura do programa, Modularização de programa, Reengenharia de dados. A seções que seguem, apresentam uma breve explicação de cada uma dessas atividades.

2.1.1 Tradução do Código-fonte

Quando um programa se encontra escrito em uma linguagem de programação e precisa, por algum motivo como, por exemplo, falta de suporte ao compilador atual, ser migrado para outra linguagem, a tradução do código-fonte se faz necessária. Esta tradução mantém a estrutura e a organização do programa em si inalteradas. A linguagem para a qual o programa será transcrito pode ser uma versão atualizada da linguagem original, como, por exemplo, de PHP 4 para PHP 5, ou pode ser uma linguagem diferente, como, por exemplo, de C++ para Java.

A tradução de código-fonte só é economicamente viável se existir alguma ferramenta para realizar a maior parte da tradução de forma automatizada (SOMMERVILLE, 2003). Muitas vezes, é impossível realizar a tradução de forma completamente automática. As instruções da linguagem original podem não ter nenhuma instrução equivalente na nova linguagem. Assim, é necessário que sejam realizadas mudanças manualmente, para que o sistema possa ser corretamente traduzido.

Como a ferramenta que sofrerá o processo de reengenharia já está na linguagem do sistema que ela será integrada, a linguagem Java, a tradução do código-fonte não será necessária.

2.1.2 Engenharia Reversa

Segundo Pressman (PRESSMAN, 1995), Chikofsky e Cross (CHIKOFSKY; CROSS, 1990), o termo engenharia reversa tem suas origens na análise do hardware, onde a prática de “decifrar” projetos de produtos era mais comum. Uma empresa desmonta um hardware comercializado a fim de entender os “segredos” de projeto e manufatura, num esforço de melhorar seus próprios produtos, como também analisar os produtos dos concorrentes. Rekoff (REKOFF, 1985) define a engenharia reversa como “o processo de desenvolvimento de um conjunto de especificações para um complexo sistema de hardware através de um metódico exame de exemplares deste sistema”. Ele descreve este processo como sendo conduzido por alguém que não seja o desenvolvedor original do sistema: “sem os

benefícios de qualquer documentação original (...) com o propósito de criar um clone do sistema de hardware original...".

A engenharia reversa para software é similar, porém, na maioria dos casos, o programa que passará pelo processo de engenharia reversa não é o do concorrente, e sim o sistema da própria organização. Este processo fornece informações da especificação e do projeto de um sistema de software, a partir de seu código-fonte (PFLEEGER, 2004). Depois, essas informações são armazenadas de forma que nos seja possível manipulá-las. Portanto, a engenharia reversa para o software consiste na análise de um programa, com o objetivo de criar uma representação do programa em um nível de abstração maior que o código-fonte (PRESSMAN, 1995; CHIKOFSKY; CROSS, 1990). O programa em si permanece inalterado no processo de engenharia reversa. O código-fonte geralmente está disponível como entrada deste processo, caso contrário, o processo precisa ser começado com o código-executável (SOMMERVILLE, 2003).

O processo de engenharia reversa, ilustrado na figura 2.3, começa com uma fase de análise utilizando-se ferramentas automatizadas, a fim de descobrir a estrutura do sistema. Esta tarefa, por si só, pode não ser suficiente para recriar o projeto do sistema. Dessa forma, os engenheiros trabalham com o código-fonte do sistema e seu modelo estrutural para adicionar informações obtidas através da compreensão do sistema. Estas informações são mantidas em dicionários de dados, que mantém as informações do sistema, vinculados ao código-fonte do programa. Documentos de vários tipos, como diagramas de programa e de estrutura de dados e matrizes de rastreamento (que mostram onde as entidades são definidas e referenciadas no sistema), podem ser gerados através das informações armazenadas (SOMMERVILLE, 2003; PFLEEGER, 2004).

Depois de gerada a documentação do sistema, outras informações podem ser adicionadas aos dicionários de dados a fim de ajudar a recriar a especificação do sistema. Isto, em geral, envolve mais anotações manuais da estrutura do sistema. A especificação não pode ser inferida automaticamente a partir do modelo do sistema.

2.1.3 Melhoria da estrutura do programa

A reestruturação do programa é feita afim de torná-lo mais fácil de ser entendido e modificado. Em geral, os programas desenvolvem uma complexa estrutura lógica devido a diversas modificações sofridas durante a manutenção. Novos comandos são adicionados,

Figura 2.3: Processo de engenharia reversa (Adaptado de Sommerville, 2003)

sem modificar a estrutura de controle existente. No curto prazo, essa é uma solução com menor risco e mais rápida, uma vez que reduz as chances de um defeito ser introduzido no sistema (SOMMERVILLE, 2003). No longo prazo, entretanto, isso pode resultar em um código incompreensível. Estruturas complexas de código podem também ser utilizadas quando os programadores procuram evitar duplicação de código, como em sistemas que são restringidos por uma memória limitada, por exemplo.

As três principais atividades envolvidas na reestruturação do programa, definido por Pfleeger (PFLEEGER, 2004), podem ser vistas na figura 2.4. Primeiro, é feito uma analise estática a fim de obter informações que servem para representar o código como uma rede semântica ou um grafo direcionado. Esta representação não é necessariamente fácil de ser compreendida pelas pessoas. Geralmente, ela é utilizada apenas por uma ferramenta automatizada.

Logo depois, a representação é refinada por meio de sucessivas simplificações com base em técnicas de transformação. Por fim, uma vez completada as simplificações, um novo programa é gerado.

2.1.4 Modularização de programa

A modularização do programa consiste na reorganização do mesmo, de modo que as partes relacionadas sejam reunidas em um único módulo (SOMMERVILLE, 2003). Dessa forma, fica mais fácil remover redundâncias nesses componentes relacionados, otimizar suas interações e simplificar suas interfaces com o restante do programa.

Diversos tipos de módulos podem ser criados durante o processo de modularização

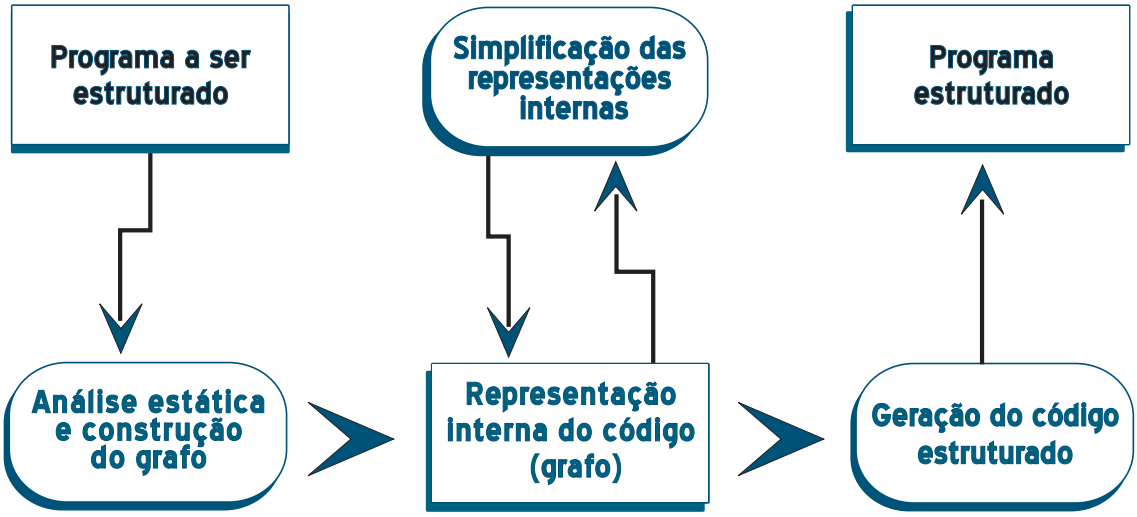

Figura 2.4: Processo de reestruturação de programa

do programa. Dentre eles, Sommerville (SOMMERVILLE, 2003) cita quatro de mais destaque:

- Abstrações de dados: as abstrações de dados ou os tipos de dados abstratos agrupam os dados e o processamento associado, como funções de construção e de acesso, e são flexíveis em relação às mudanças. Desde que a interface seja mantida, as modificações em dados abstratos não devem afetar outras partes do programa.

- Módulos de Hardware: esses módulos agrupam todas as funções relacionadas ao controle de um determinado dispositivo de hardware e estão estreitamente relacionados com as abstrações de dados.

- Módulos funcionais: esses módulos agrupam funções que realizam tarefas semelhantes, estreitamente relacionadas. Todas as funções ocupadas com dados de entrada e a validação destes dados, por exemplo, podem ser agrupadas em um único módulo.

- Módulos de apoio ao processo: todas as funções e todos os itens específicos de dados que apóiam um processo de negócio específico são agrupados nesses módulos. Sommerville (SOMMERVILLE, 2003) cita como exemplo um módulo de apoio ao processo de incluir toda a funcionalidade exigida para apoiar a emissão e o retorno de livros em um sistema de biblioteca.

2.1.5 Reengenharia de dados

A reengenharia de dados é o processo de analisar e reorganizar estruturas de dados e, eventualmente, os valores dos dados de um programa, com o objetivo de torná-lo mais compreensível (SOMMERVILLE, 2003).

A princípio, se a funcionalidade do programa permanecer inalterada a reengenharia de dados não deverá ser necessária. Porém, na prática, existem alguns fatores pelos quais os dados precisam ser modificados, como:

- Degradação dos dados: existência de dados duplicados ou redundantes, armazenados em diferentes partes do programa com diferentes formatos. Ainda, os dados podem não refletir alterações no ambiente externo.
- Limites impostos aos programas: quando o sistema precisa tratar de um volume maior de dados do que foi originalmente previsto pelos projetistas.
- Evolução da arquitetura: na migração de arquitetura centralizada para distribuída, o acesso aos dados pode ser feito remotamente, por vários usuários.

A fim de lidar com os fatores acima citados, existem algumas abordagens citadas por Sommerville (SOMMERVILLE, 2003):

- Reorganização de dados: as redundâncias e duplicações são excluídas e um formato consistente é aplicado aos registros.
- Extensão dos dados: os limites são eliminados, aumentando tamanho de campos, de tabelas e assim por diante.
- Migração de dados: os dados são migrados para o controle de um moderno SGBD. Por exemplo, sistema de arquivos para bancos de dados; SGBD antigo para um SGBD novo.

2.2 Manipulação de Cores em Sistemas de Imageamento

As seções seguintes apresentam uma visão geral sobre a ferramenta que será o alvo de estudo e aplicação das técnicas de reengenharia deste trabalho de graduação. Primeiramente, antes de falar da ferramenta propriamente dita, será feita uma exposição de alguns conceitos e propriedades dos espaços de cores.

2.2.1 Representação de cores em imagens digitais e Modelos de cores

A forma de representação de cores em imagens digitais parte das propriedades experimentais relacionadas ao sistema visual humano, que é formado por milhões de estruturas chamadas cones. Estas estruturas são responsáveis pela identificação das cores, sendo divididas em três grandes grupos, cada um dos quais é responsável por identificar uma faixa espectral que corresponde às três cores consideradas primárias, a saber, o vermelho, o verde e o azul.

O uso de cores no processamento de imagens ocorre, principalmente, em função de dois fatores: primeiro, cores são poderosas para identificar e distinguir objetos; segundo, o sistema visual humano pode distinguir muito mais facilmente entre as muitas cores existentes do que entre os poucos níveis de cinza existentes em imagens monocromáticas (GONZALEZ; WOODS, 2002). Este segundo fator que se relaciona com o objetivo da interface 3D da ferramenta, que utiliza as cores para facilitar a análise humana das imagens.

Existem diversas maneiras de se representar as informações contidas numa imagem. A maneira mais usual, porém não a única, é a representação da imagem em duas dimensões. Em alguns casos, a possibilidade de representações alternativas pode facilitar os processos de análise e processamento de imagens. Por exemplo, um processo de agrupamento pode ser realizado no espaço de cores da imagem através de uma representação em três dimensões deste, podendo resultar na segmentação da imagem original. Logo, a visualização da distribuição de cores em um ambiente em três dimensões pode ser bastante útil para a análise e processamento das imagens. A visualização tridimensional das cores da imagem está diretamente ligada ao espaço de cores que será utilizado, uma vez que este determinará como os objetos estarão distribuídos no gráfico.

De acordo com Gonzalez e Woods (GONZALEZ; WOODS, 2002), um modelo de cor é uma especificação de um sistema de coordenadas tridimensionais e um subespaço dentro deste sistema, onde cada cor é representada por um único ponto contido dentro deste subespaço. Modelos de cores são utilizados para facilitar e tornar possível a representação e a interpretação de cores de acordo com as necessidades dos dispositivos e usuários.

Os modelos de cores podem ser orientados a hardware ou não. Modelos orientados a hardware são assim chamados por terem sido criados devido às necessidades e propriedades dos dispositivos nos quais são utilizados. Neste grupo podem ser citados os dois

modelos mais conhecidos: RGB, utilizado em dispositivos como monitores, e CMYK, utilizado em impressoras. O outro grupo, onde os modelos não são orientados à hardware, é composto por modelos desenvolvidos para facilitar a análise de imagens, tanto análises humanas quanto análises computadorizadas. Alguns desses modelos foram criados com base na forma com que o sistema visual humano diferencia as cores, utilizando conceitos como a existência dos cones no olho humano e os variados comprimentos de onda que são percebidos. Neste segundo grupo podem ser citados modelos como o modelo HSI e o espaço CIE L*a*b*.

2.2.2 Ferramenta de visualização 3D da distribuição de cores

A ferramenta que provê uma interface tridimensional para a representação da distribuição de cores de imagens digitais (DARONCO, 2006) estava incluída no antigo sistema de processamento e análise de imagens do LaCA, chamado Arthemis. Este sistema foi desenvolvido utilizando a linguagem Java e bibliotecas disponíveis para a mesma, como a JAI e a ImageIO. Esta ferramenta, é dividida em dois módulos principais: o módulo das conversões e o módulo da interface.

O módulo de conversões é formado por todos aqueles componentes responsáveis pelo suporte aos espaços de cores, onde estão implementadas as fórmulas de conversão entre esses espaços, que podem ser encontradas em (DARONCO, 2006). Os espaços de cores suportados pela ferramenta, bem como a relação entre eles, podem ser verificados na figura 2.5. É assumido que as imagens carregadas na aplicação estejam no espaço RGB, de onde todos os outros derivam, direta ou indiretamente. Esta variedade de espaços existentes é um ponto importante da ferramenta, pois disponibiliza diversas formas de visualizar a imagem no espaço 3D.

Já o módulo da interface é formado pelos componentes do ambiente tridimensional e pelas janelas que exibem as opções de configuração da ferramenta ao usuário e que permitem que ele interaja com o sistema. O ambiente tridimensional possibilita que o usuário verifique a distribuição espacial das cores de uma imagem de uma maneira fácil interativa, podendo configurar parâmetros, modificar cores, realizar rotações, entre outras diversas tarefas existentes com o objetivo de facilitar a visualização do gráfico. A figura 2.6 mostra algumas imagens do ambiente 3D utilizando imagens e espaços de cores diferentes para ilustrar algumas das opções de visualização possíveis.

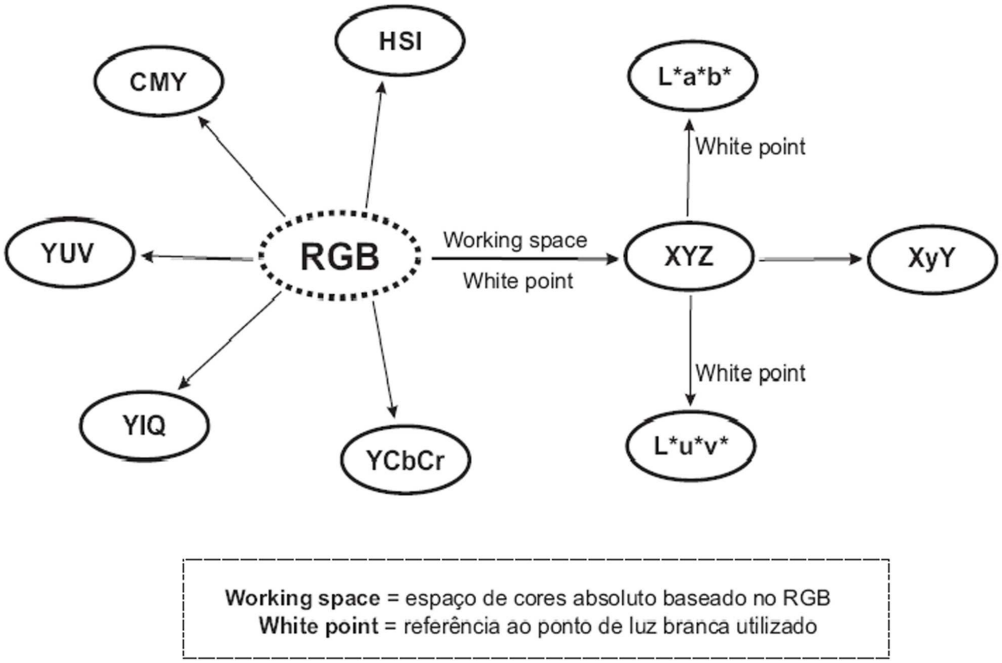

Figura 2.5: Diagrama mostrando a relação entre os espaços de cores implementados (Retirado de Daronco, 2007)

Este é o componente que faz o uso do JOGL (Java OpenGL). O JOGL possui integração com os frameworks AWT e Swing, que são utilizados por esta ferramenta e pelos sistemas Arthemis e Anima para a criação dos componentes da interface gráfica.

Maiores informações acerca do funcionamento da ferramenta podem ser encontradas em (DARONCO, 2006)

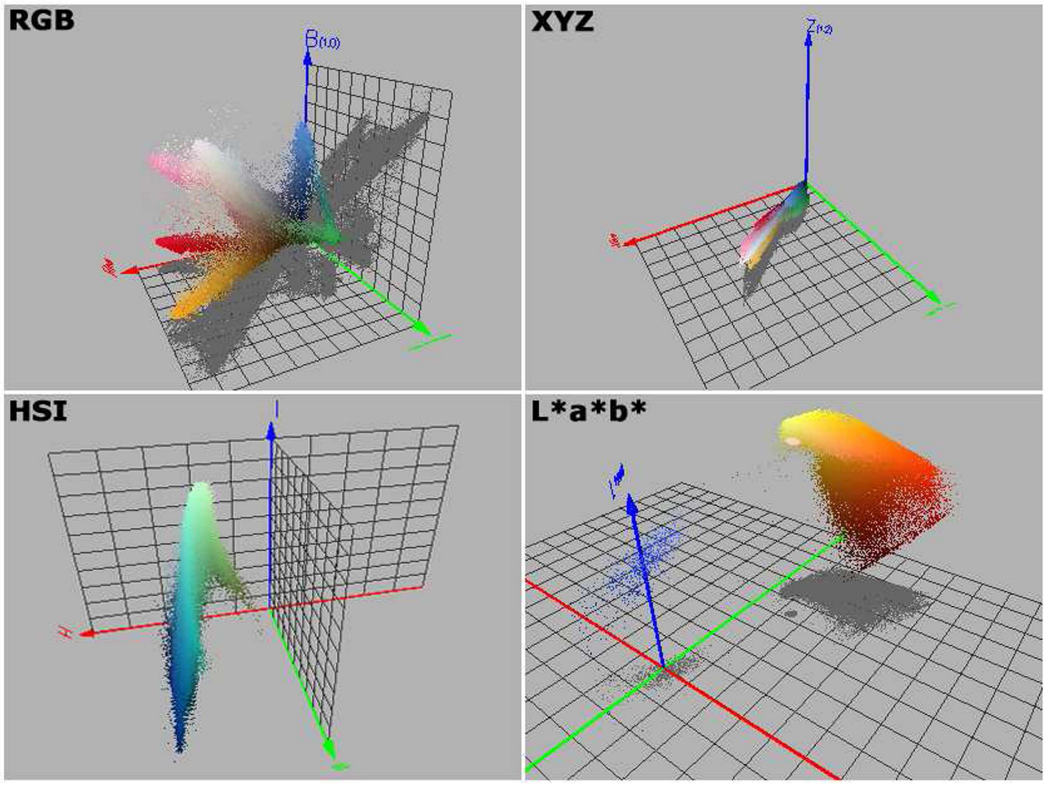

Figura 2.6: Imagens do ambiente 3D desenvolvido (Retirado de Daronco, 2007)

3 DESENVOLVIMENTO

Este capítulo descreve questões relacionas ao desenvolvimento do presente projeto. Inicialmente apresenta-se uma visão geral dos materiais e métodos utilizados para que a ferramenta fosse reengenheirada. A seguir relata-se questões referentes à implementação e inclusão da ferramenta no *Anima*.

3.1 Materiais e Métodos

3.1.1 Sistema de Processamento e Análise de Imagens do LaCA

Antigamente, o LaCA possuía um sistema de processamento de análise de imagens chamado *Arthemis*. Este sistema, desenvolvido pelos alunos de graduação e mestrado, centralizava praticamente toda a produção tecno-científica do grupo e possuía as características básicas para o processamento e análise de imagens como, por exemplo, funções de abertura e salvamento de imagens, zoom, rotação, entre outros (BERNI, 2007).

Por muito tempo, o *Arthemis* supriu as necessidades do grupo. Porém este sistema apresentava alguns problemas que vinham dificultando a sua utilização. Dentre eles, pode-se citar a falta de um controle para habilitar e desabilitar componentes na interface gráfica de acordo com as imagens abertas. Por exemplo, algumas funcionalidades permaneciam habilitadas mesmo com imagens em formatos não suportados, o que gerava uma série de erros caso o usuário utilizasse tais funcionalidades. Além disso, havia uma dificuldade por parte dos acadêmicos de realizar a inclusão de novas funcionalidades na ferramenta. Esta tarefa exigia um bom conhecimento do sistema, o que muitas vezes levava bastante tempo para ser adquirido, já que, devido a grande rotatividade que o LaCA possuía, não havia alunos que conhecessem completamente o funcionamento do sistema. Assim, devido a estes fatores, um novo sistema de processamento e análise de imagens, chamado *Anima*, foi implementado pela Animati Computação Aplicada a qual cede ao

LaCA o direito de desenvolver pesquisas científicas nessa plataforma.

O *Anima* segue basicamente a mesma proposta definida pelo *Arthemis*, de ser um sistema de processamento e análise de imagens que centraliza as diversas ferramentas desenvolvidas pelos integrantes do LaCA e da Animati. Porém, o *Anima* diferencia-se do antigo sistema pela facilidade de adicionar novas funcionalidades e pela estruturação do projeto que permite que cada funcionalidade comporte-se realmente como um *plugin*, podendo ser incluído e retirado do sistema de forma fácil através da configuração de um arquivo XML. Além disso, o *Anima* possui um *middleware* que provê algumas funcionalidades que facilitam a comunicação das ferramentas com o sistema. Entre estas funcionalidades, pode-se citar aquelas que permitem o acesso às imagens abertas no sistema como, por exemplo, obter a imagem que está atualmente selecionada no sistema ou então recuperar todas as imagens multiespectrais que estão abertas. Na figura 3.1, é apresentado um diagrama de pacotes do *Anima*, que mostra como o sistema está estruturado. As novas funcionalidades são inseridas no pacote *features* que faz parte do pacote para desktops do *Anima*.

3.1.2 Passos realizados para a reengenharia

A primeira medida tomada antes de realizar qualquer etapa da reengenharia foi transportar o antigo projeto do Arthemis, no qual a ferramenta estava inclusa, que havia sido desenvolvido usando o IDE *Eclipse*, para o IDE *NetBeans*, no qual o *Anima* foi desenvolvido. Esta tarefa foi feita através do *Eclipse Project Importer*, que permite a importação no *NetBeans* de projetos criados no *Eclipse*.

Após esta tarefa inicial, foi buscada uma ferramenta automatizada que realizasse o processo de engenharia reversa, já explicado anteriormente na seção 2.1.2. A engenharia reversa é o processo de transformação de um código-fonte em um modelo pelo mapeamento a partir de uma linguagem de programação específica (BOOCH; RUMBAUGH; JACOBSON, 2005). Podemos entender um modelo como uma simplificação ou abstração da realidade, criado com a finalidade de proporcionar uma melhor compreensão do sistema que esta sendo gerado. Este modelo é formado pelo dicionário de dados, com as classes, atributos e operações, que são de fundamental importância para a criação dos diagramas UML necessários para o presente trabalho. Para a realização desta etapa foi utilizado, então, um *plugin* de UML (*Unified Modeling Language*) para o *NetBeans* que

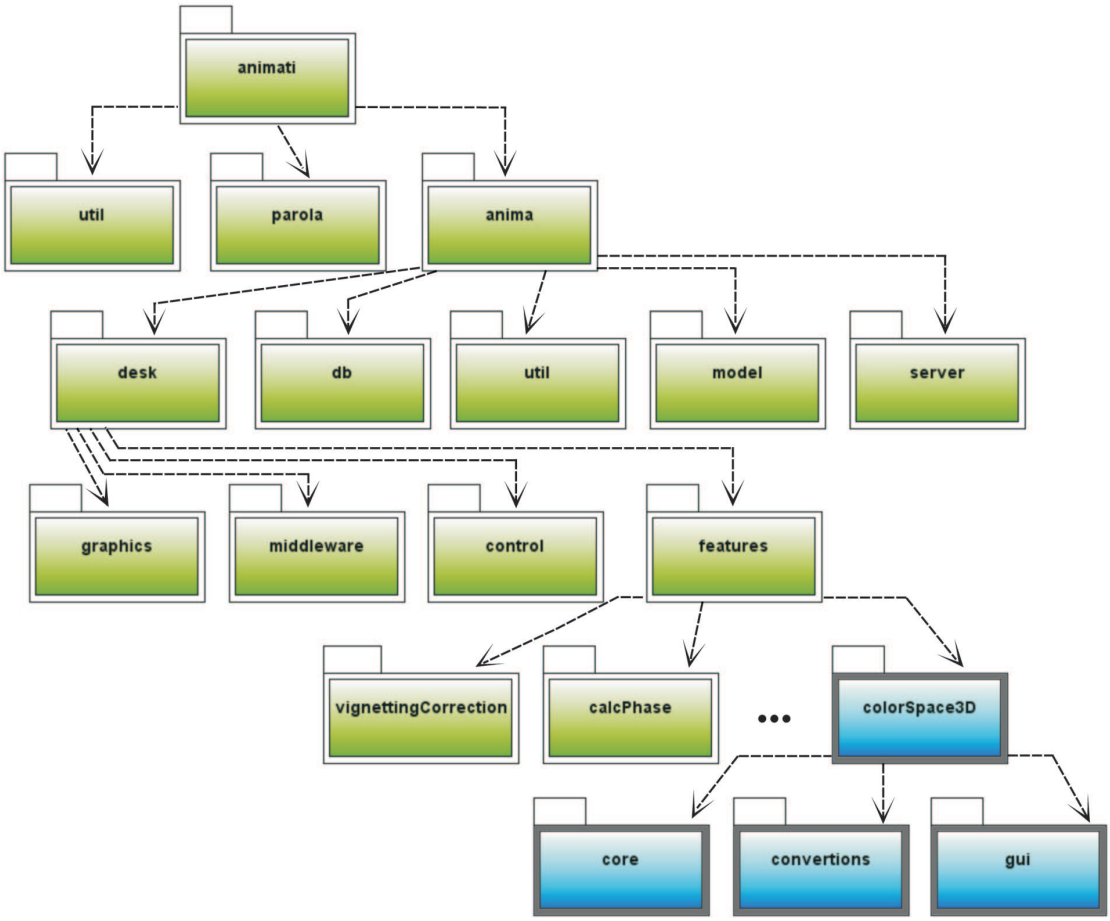

Figura 3.1: Diagrama de pacotes do Anima

permite o desenvolvimento de modelos com o uso de UML (NETBEANS, 2008a). Este *plugin* também suporta todas as principais fases do ciclo de vida do desenvolvimento de um *software*, desde a engenharia reversa e modelos de relatórios HTML para a interação do usuário e modelagem de processos, até a geração de código e implantação de aplicações.

Com esta ferramenta automatizada, o IDE consegue realizar a engenharia reversa de códigos-fonte em Java e transformá-lo em um projeto de modelagem UML, que contém praticamente todas as informações estruturais do código, exceto os comentários e espaços em branco (NETBEANS, 2008a). Na figura 3.2 pode ser visto o dicionário de dados gerado a partir da engenharia reversa.

Quando é feita a engenharia reversa em um código Java com uma modelagem UML, o IDE realiza as seguintes tarefas:

- Gera os elementos de classe no modelo UML para refletir as classes e interfaces do

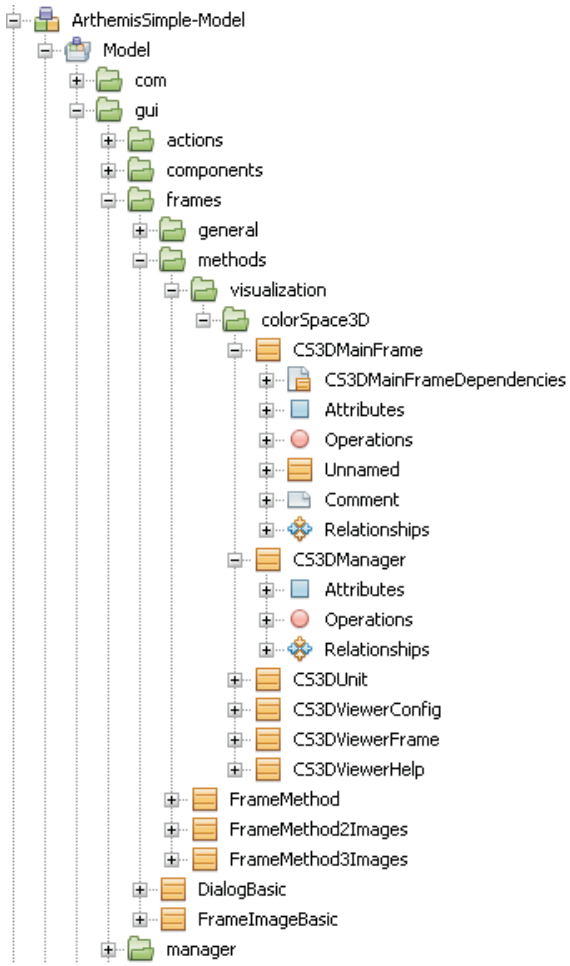

Figura 3.2: Dicionário de dados

código Java;

- Adiciona métodos e membros das classes Java ao modelo UML, como atributos e operações.
- Deriva as dependências de arquivos das declarações `import` do Java.
- Constrói um projeto de modelagem UML e coloca-o na árvore de projetos do IDE, como é visto na figura 3.2

Depois de gerado o modelo UML contendo as informações (o dicionário de dados) a respeito do código-fonte da ferramenta a ser reengenheirada, foi então gerado um diagrama de dependências da classe principal da interface gráfica da ferramenta `CS3DMainFrame`. A partir deste diagrama, que pode ser visto na figura 3.3, foi possível identificar quais classes tinham ligação direta com a interface gráfica da ferramenta.

Além do diagrama de dependências, é possível criar com o *plugin* de UML os seguintes diagramas: diagramas de atividades, diagramas de classes, diagramas de seqüência, diagramas de estado e diagramas de casos de uso. Destes, o digrama de maior utilidade foi o diagrama de classes, pois este é importante não só para a visualização, a especificação e a documentação de modelos estruturais, como também para a construção de sistemas executáveis. Booch (BOOCH; RUMBAUGH; JACOBSON, 2005) define o diagrama de classe como um diagrama que mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos.

A figura 3.4 mostra a modelagem UML de um diagrama de classes que foi gerado a partir do dicionário de dados obtido anteriormente. Este diagrama mostra o relacionamento entre as classes principais da ferramenta, que são responsáveis pela interface gráfica, pelo gerenciamento das janelas abertas pela ferramenta e pela chamada aos métodos de conversões disponíveis. A partir deste diagrama foi possível entender a ligação entre as classes e também como elas interagiam entre si.

Com relação à melhoria da estrutura do programa, discutida na seção 2.1.3, não foi necessária a sua realização. O código-fonte da ferramenta já estava bem estruturado visto que esta ferramenta foi programada somente por uma pessoa e não passou por etapas posteriores de manutenção feitas por terceiros, o que pode levar a um código com uma estrutura lógica complexa.

Devido à nova estrutura do *Anima*, que agrupa cada nova funcionalidade em um pacote chamado `features`, mostrado na figura 3.1, a ferramenta passou por um processo de modularização (seção 2.1.4). Ela foi reagrupada de forma que estivesse de acordo com esta nova estrutura. No *Arthemis*, as funcionalidades eram dividas em duas partes principais, a interface gráfica e os seus métodos. Isto dificultava para um desenvolvedor que tivesse que fazer alguma alteração em uma funcionalidade já existente, por exemplo, pois seu código fonte estava espalhado pelos diretórios do sistema. Hoje, com esta nova estrutura, todo o código-fonte e outros arquivos das funcionalidades do sistema podem ser encontrados dentro de um único pacote referente à funcionalidade.

A ferramenta que foi reengenheirada foi agrupada em um pacote chamado `colorSpace3D` que possui três módulos principais: um com os arquivos referentes à interface gráfica (pacote `gui`), outro com os métodos de conversão entre os espaços de cores bem como outras classes responsáveis pelo gerenciamento dos conversores (pacote `convertions`)

e um terceiro que guarda as classes referentes à geração do gráfico tridimensional (pacote `core`). Os módulos da ferramenta podem ser visto na cor azul no diagrama de pacotes da figura 3.1.

A respeito da reengenharia de dados, não foi necessário fazer grandes alterações na ferramenta. A principal alteração realizada neste sentido foi a eliminação do encapsulamento dos dados antes de ser realizada a conversão do espaço de cores. No sistema *Arthemis* todos os componentes deveriam estender a classe `MethodBasic` e implementar a classe `MethodInterface`, que obrigavam o programador a incluir as fontes (imagens de entrada do componente) e os parâmetros em dois vetores próprios, `sources` e `parameters`, respectivamente, e os resultados (tanto imagens quanto parâmetros) eram colocados em outro vetor, chamado `results`. Na nova estrutura do *Anima*, os componentes não precisam mais estender e implementar nenhuma classe, podendo então os dados ser passados diretamente aos métodos, facilitando assim a sua implementação.

3.2 Implementação

Um dos objetivos deste trabalho era preservar as funções da ferramenta reutilizando os esforços de desenvolvimento passados. Dessa forma, depois da compreensão do relacionamento entre as classes da ferramenta, através dos diagramas gerados anteriormente, foi possível constatar quais as partes da implementação da ferramenta que deveriam ser alteradas.

A janela que possui as opções de configuração dos conversores e que possibilita a execução do componente 3D precisou ser reprogramada pois esta estendia uma classe interna ao sistema *Arthemis*. A nova janela foi programada utilizando o construtor de GUI's (*Graphical User Interface*) do *NetBeans*. Este construtor permite a criação de GUI's Swing arrastando e posicionando os componentes de GUI de uma paleta em uma tela. O construtor de GUI's se encarrega automaticamente do espaçamento e do alinhamento corretos (NETBEANS, 2008b). Dessa forma, a manutenção da interface gráfica da ferramenta ficará de forma mais fácil e intuitiva para programadores menos experientes. Esta é a janela principal da ferramenta, aquela que será exibida quando o usuário executar a ferramenta através dos menus do *Anima*. Ela pode ser vista na figura 3.5

Outra alteração no código-fonte da ferramenta foi a utilização dos métodos fornecidos pelo middleware do Anima. Através de uma chamada a um método desse middleware,

por exemplo, foi possível recuperar a imagem atualmente aberta e em foco no sistema. Este código pode ser visto abaixo:

```
PlanarImage image =
AnimaMiddleware.getCurrentImageFrame().getImage().getPlanarImage();
```

Ainda, uma melhoria adicionada à ferramenta foi a opção de internacionalização de sua interface gráfica. A internacionalização permite que a aplicação seja adaptada para várias linguagens sem a necessidade de recompilação. Programas internacionalizados permitem que elementos textuais, como mensagens de status e rótulos componentes da GUI, sejam armazenados fora do código-fonte e então recuperados dinamicamente, ao invés de serem inseridas diretamente no código-fonte. Os textos internacionalizados são armazenados em arquivos de propriedades na forma de par chave/valor. A chave é o identificador usado pelo programa para encontrar o valor correspondente nos arquivos de linguagem. É criado um arquivo de propriedades para cada linguagem que se deseja traduzir o programa. As chaves são as mesmas em todos os arquivos e somente o seu valor é diferente e corresponde ao texto na língua desejada. Para recuperar os valores das chaves, foi usada a classe Java `ResourceBundle`. No código-fonte de todos os arquivos que representam uma GUI, todos os componentes que apresentam um texto na interface usam a chave para representar o seu significado. Para decidir em que língua a ferramenta deve ser mostrada é feita uma chamada ao *middleware* do sistema que retorna a língua corrente do *Anima*. Até o momento, foram criados arquivos nas linguagens português brasileiro e inglês norte-americano.

A inserção da ferramenta no sistema se dá com a configuração de dois tipos de arquivos XML: o primeiro, `config.xml` , que configura onde a ferramenta ficará na interface gráfica do sistema, e o segundo, `main_en_US.xml` e `main_pt_BR.xml`, que armazena os textos que estão presentes no menu do sistema, garantindo a sua internacionalização. No arquivo de configuração, é possível definir se a ferramenta estará sempre habilitada no menu ou então se deve ser chamada alguma verificação para habilitá-la. No caso da ferramenta em questão, que aceita somente imagens no espaço RGB como entrada para o método de conversões (DARONCO, 2006), foi implementado um método que a habilita somente quando uma imagem RGB esta aberta e em foco no sistema. Desse modo, é impedido que o usuário execute o método com alguma imagem não suportada,

evitando, assim, erros que poderiam ser gerados, como era permitido na antiga versão. Este método é mostrado na figura 3.6

Figura 3.3: Diagrama de dependências

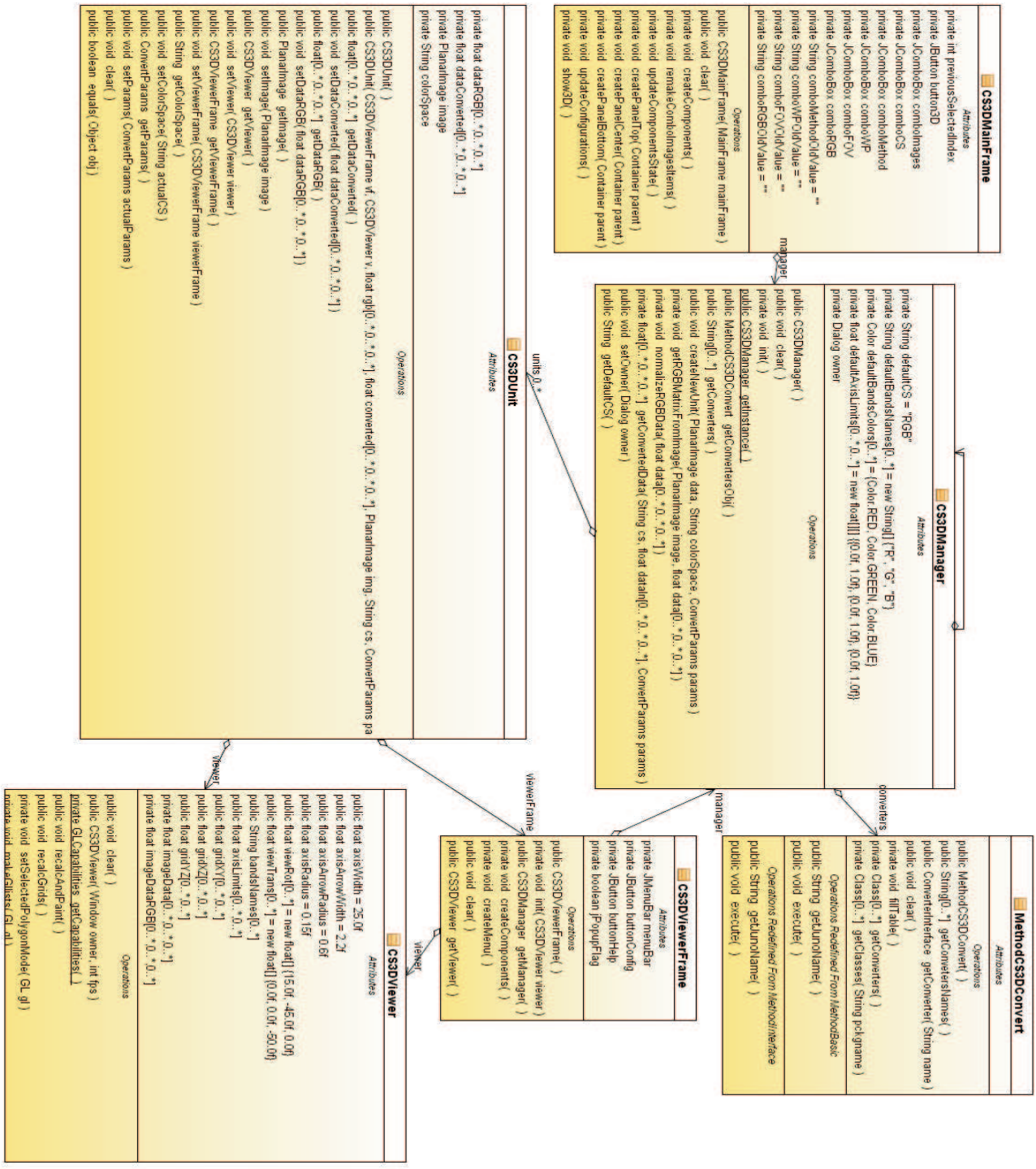

Figura 3.4: Diagrama de classes das principais classes da ferramenta

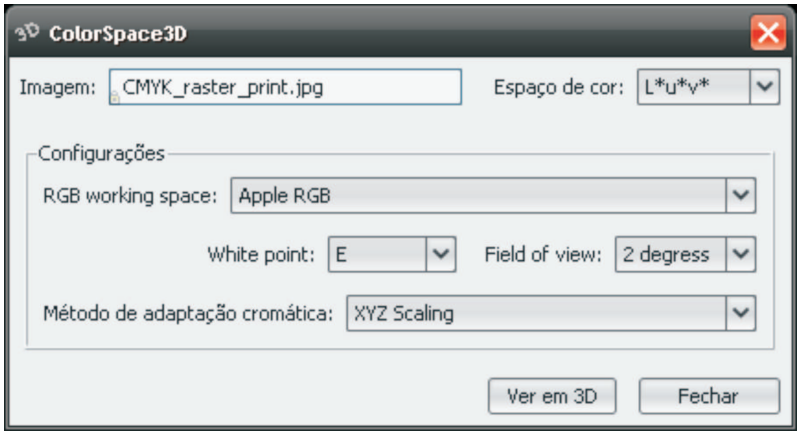

Figura 3.5: Janela de configuração da ferramenta

```
public boolean isEnabled() {
    //pega a imagem em foco no sistema
    PlanarImage image = AnimaMiddleware.getCurrentImageFrame().getImage().getPlanarImage();
    //se existe imagem aberta
    if (image != null) {
        //testa se ela é binária
        if (ImageUtil.isBinary(image.getSampleModel())) {
            return false;
        } else {
            //testa se ela é RGB e não tem canal Alpha
            if (image.getColorModel().getColorSpace().getType() == ColorSpace.TYPE_RGB
                    && !image.getColorModel().hasAlpha()) {
                return true;
            }
        }
    }
    return false;
}
```

Figura 3.6: Código que habilita a ferramenta no menu do sistema

4 RESULTADOS E AVALIAÇÃO

O principal resultado obtido com a realização deste trabalho foi a inclusão da ferramenta para a representação da distribuição de cores de imagens digitais em diferentes espaços de cores no novo sistema de processamento e análise de imagens do LaCA, o *Anima*. Através da aplicação das técnicas de reengenharia citadas na seção anterior, foi possível reestruturar a ferramenta preservando as suas funções e reutilizando praticamente grande parte do esforço de desenvolvimento anteriormente despendido para a construção da ferramenta. As alterações que foram realizadas no código-fonte visavam à independência da ferramenta ao sistema anterior e a utilização do *middleware* que dispõe funcionalidades que facilitam a comunicação da ferramenta com o novo sistema, como o acesso aos diferentes tipos de imagens abertas.

Uma melhoria incluída nesta versão da ferramenta foi a restrição da sua execução com tipos de imagens incompatíveis. Isto se dá através de um método que checa se a imagem selecionada é RGB e então habilita a ferramenta na interface gráfica do sistema. Assim, evita-se que a ferramenta esteja habilitada para tipos incompatíveis de imagens, como imagens binárias. Este problema era possível de acontecer no antigo sistema, pois a ferramenta funcionava com alguns tipos de imagens não suportadas, o que acabava gerando diversos erros que o usuário só tinha conhecimento no momento da utilização da ferramenta. No futuro, se a ferramenta permitir fazer a conversão a partir de imagens em outros espaços de cores basta que o método que habilita a ferramenta na interface principal seja reprogramado a fim de atender estas mudanças. Na figura 4.1 é possível ver a ferramenta em funcionamento dentro do *Anima*.

Das atividades de reengenharia estudadas no trabalho, duas não foram aplicadas nesta ferramenta, mas podem ser úteis dependendo do sistema a ser reengenheirado. A tradução do código-fonte não foi necessária pois a ferramenta foi desenvolvida dentro do

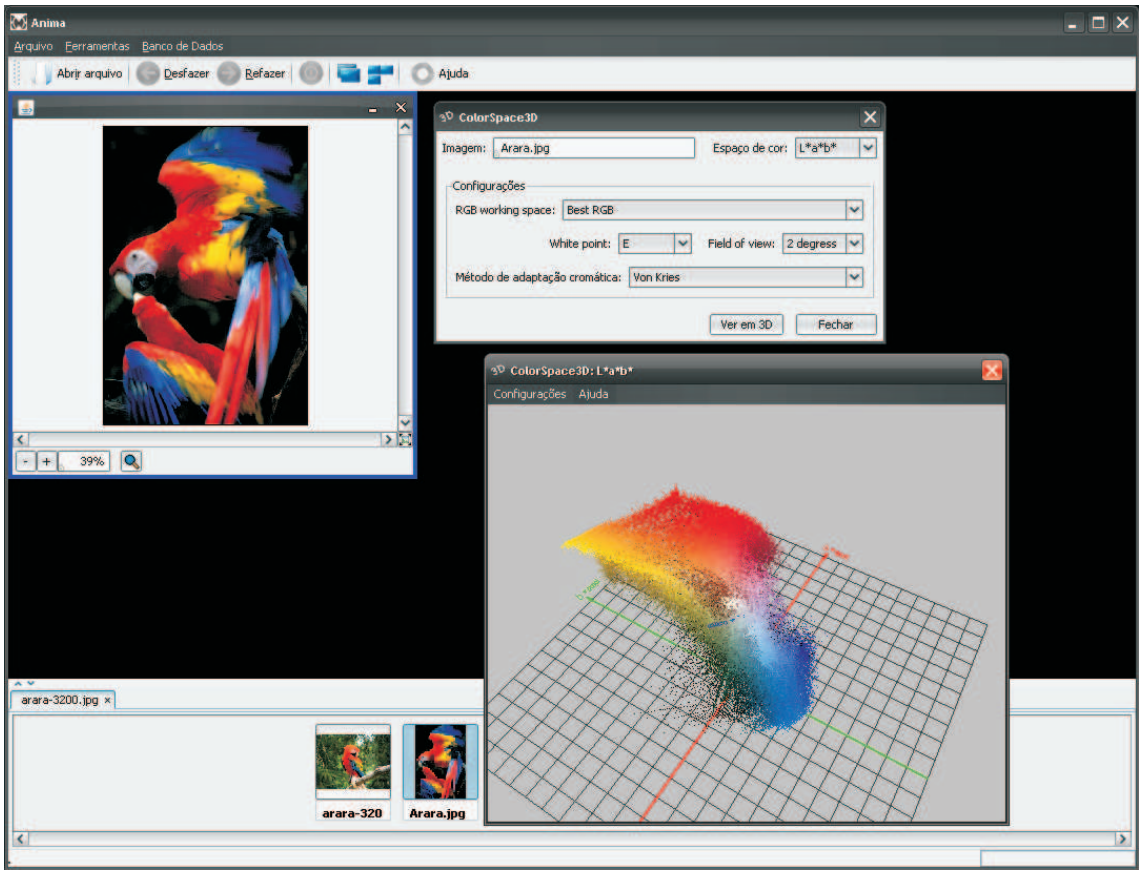

Figura 4.1: Ferramenta em execução no Anima

LaCA e utilizava a mesma linguagem que hoje ainda é usada no laboratório para o desenvolvimento de aplicações de processamento de imagens: a linguagem Java, com o uso da biblioteca JAI. Se no período de tempo em que a ferramenta foi desenvolvida até a data do presente trabalho o laboratório tivesse adotado uma outra linguagem de programação como linguagem padrão, esta atividade deveria ter sido realizada. Este fato pode ocorrer em outras organizações fazendo com que os sistemas que devam ser reaproveitados passem por esta etapa. Outra atividade que não foi realizada é a melhoria da estrutura do programa. Esta etapa é feita em sistemas que apresentam uma estrutura de controle complexa, com muitas ramificações incondicionais, e a lógica de controle não é intuitiva. Geralmente, os programas desenvolvem essa complexa estrutura lógica ao serem modificados durante a manutenção. Como a ferramenta reengenheirada foi desenvolvida somente por uma pessoa e não passou por etapas de manutenção, a sua estrutura lógica estava bem organizada.

Das demais atividades realizadas, a que levou mais tempo para ser executada foi a engenharia reversa. Para executar esta atividade foi necessário primeiramente encontrar

uma ferramenta automatizada que gerasse o dicionário de dados do componente a ser reengenheirado. Após isto, foram criados alguns diagramas estruturais, como diagrama de classes e diagrama de dependências, com o objetivo de compreender de forma clara como a ferramenta estava estruturada.

Os códigos-fonte que estavam espalhados por alguns pacotes do antigo sistema foram agrupados dentro de um mesmo pacote e então foram separados em módulos (GUI's, conversores, núcleo), de forma que suas partes relacionadas fossem reunidas. Assim, fica mais fácil a remoção de redundâncias nesses componentes relacionados, a otimização das suas interações e a simplificação das suas interfaces com o restante do programa. Ainda, principalmente, a ferramenta torna-se plugável, isto é, pode ser facilmente inserida e retirada do sistema com a configuração de dois ou três arquivos XML.

Houve, também, uma simplificação na forma de como os dados eram tratados para executar o método de conversão entre os espaços de cores. Como no *Anima* os métodos podem ser declarados livremente, sem nenhuma dependência, os dados, que antes precisavam ser encapsulados e desencapsulados toda vez que o método fosse chamado, puderam ser passados diretamente ao método de conversão. Isto facilitou a compreensão do código e, de certa forma, até houve uma otimização, visto que menos estruturas precisam ser usadas. Essa diferença pode ser vista na figura 4.2

Figura a

```
/**
 * Realiza a conversão dos espaços de cores.
 * @param cs Espaço destino.
 * @param dataIn Dados da imagem de entrada.
 * @param params Parâmetros de conversão.
 */
private float[][][] getConvertedData(String cs, float[][][] dataIn, ConvertParams params) {
    try {
        Vector<Object> sources = new Vector<Object>(1);
        sources.add(dataIn);
        Vector<Object> parameters = new Vector<Object>(3);
        parameters.add(defaultCS);
        parameters.add(cs);
        parameters.add(params);
        Vector<Object> results;
        results = Methods.create(MethodCS3DConvert.class, sources, parameters);
        return (float[][][])results.get(0);
    } catch (MethodException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura b

```
/**
 * Realiza a conversão dos espaços de cores.
 * @param cs Espaço destino.
 * @param dataIn Dados da imagem de entrada.
 * @param params Parâmetros de conversão.
 */
private float[][][] getConvertedData(String cs, float[][][] dataIn, ConvertParams params) {
    try {
        // aqui nao preciso encapsular todos dos dados
        return converters.execute(dataIn, defaultCS, cs, params);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura 4.2: Simplificação da chamada ao método de conversão: (a) Arthemis (b) Anima

5 CONCLUSÃO

Este trabalho apresentou o estudo de técnicas de reengenharia de software e a sua aplicação prática através da reestruturação de uma ferramenta para a representação da distribuição de cores de imagens digitais em diferentes espaços de cores. A reengenharia de software foi aplicada para a reestruturação pois ela representa um modo eficiente, com riscos reduzidos, e sem altos custos de realizar a adequação da estrutura da ferramenta.

Primeiramente, foi realizada uma pesquisa acerca das atividades que compõe o processo de reengenharia de software, a fim de compreender de que forma elas poderiam ser aplicadas no contexto deste trabalho. Após, buscou-se entender o funcionamento da ferramenta a ser reengenheirada através da sua execução no antigo sistema *Arthemis*. Feito isto, a ferramenta então passou pelas atividades de engenharia reversa, modularização do programa e por uma pequena reengenharia de dados. Não foi necessário que a ferramenta passasse por uma tradução do código-fonte, pois a mesma já se encontrava na linguagem de programação utilizada no novo sistema, e ,devido ao fato de não ter passado por etapas de manutenção, o seu código-fonte estava bem estruturado, sendo que não foi necessária a melhoria da sua estrutura.

A etapa da engenharia reversa mostrou-se muito importante para a realização deste trabalho. Através dela, foi possível desenvolver uma maior compreensão da estrutura da ferramenta, entendendo a ligação entre as classes e como elas interagiam entre si. Esta etapa foi realizada com a ajuda de uma ferramenta automatizada que criou o dicionário de dados referente à ferramenta, a partir do qual foi possível gerar alguns diagramas UML. A modularização do programa, entre outras coisas citadas na seção anterior, proporcionou que todos os arquivos da ferramenta fossem agrupados em um mesmo pacote. Isto, aliado à estrutura do sistema *Anima* que é configurado através de arquivos XML, fez com que a ferramenta comportasse-se como um *plugin*, ou seja, agora ela pode ser inserida e retirada

do sistema de forma fácil e rápida. Com a reengenharia de dados, os dados foram alterados para refletir as mudanças da ferramenta, o que acabou levando a uma simplificação em determinadas partes do código, tornando-o, assim, mais compreensível.

Uma dificuldade enfrentada durante a realização deste trabalho foi encontrar bibliografia sobre a aplicação prática de técnicas de reengenharia de software. A maioria dos livros estudados abordava somente conceitos teóricos sobre as atividades presentes no processo de reengenharia. Algumas delas ainda tratavam o assunto somente de forma superficial, sem a amplitude necessária para compreender profundamente o processo de reengenharia de software. Assim, muito dessa compreensão foi obtida durante o desenvolvimento do trabalho.

Com a reengenharia desta ferramenta e a sua inserção no sistema de processamento e análise de imagens Anima, é possível agora que sejam adicionadas novas funcionalidades tendo ela como base, pois a mesma já se encontra em funcionamento. Dentre elas, a possibilidade da seleção de *pixels* no gráfico 3D seria de grande utilidade, pois a partir daí operações de segmentação da imagem poderiam ser aplicadas. Para que alterações no ambiente 3D sejam feitas é preciso modificar a classe `CS3DViewer`, que é responsável pela renderização do ambiente 3D para exibição da distribuição de cores da imagem. Ainda, outra melhoria que poderia ser incluída seria possibilitar o carregamento de imagens em espaços de cores distintos ao RGB. Para isso é necessário implementar as equações de conversão dos modelos suportados para o RGB e também disponibilizar novos comandos na interface para que o usuário selecione o espaço no qual a imagem será carregada. Mais melhorias referentes à ferramenta podem ser encontradas em (DARONCO, 2006).

Para concluir, este trabalho atingiu os objetivos pretendidos. A ferramenta foi reengenheirada e inserida no sistema *Anima*, preservando suas funções e reutilizando os esforços de desenvolvimento passados. Os conhecimentos adquiridos neste trabalho podem ser aplicados em outras ferramentas que necessitam ser inseridas no *Anima*.

REFERÊNCIAS

BERNI, J. C. A. **Desenvolvimento e Implementação de Métodos de Correção de Iluminação para Imagens Digitais**. Santa Maria: Curso de Ciência da Computação. Universidade Federal de Santa Maria., 2007.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML Guia do Usuário**. 2.ed. Rio de Janeiro: Editora Campus, 2005.

CHIKOFSKY, E. J.; CROSS, J. H. Reverse Engineering and Design Recovery: a taxonomy. **IEEE Softw.**, Los Alamitos, CA, USA, v.7, n.1, p.13–17, 1990.

DARONCO, L. C. **Interface 3D para representação da distribuição de cores de imagens digitais em diferentes espaços de cores**. Santa Maria: Curso de Ciência da Computação. Universidade Federal de Santa Maria., 2006.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2.ed. Upper Saddle River, New Jersey: Prentice Hall, 2002.

NETBEANS. **UML Developer Wiki**. Disponível em: <http://wiki.netbeans.org/UML>. Acesso em: novembro de 2008.

NETBEANS. **Swing GUI Builder**. Disponível em: <http://www.netbeans.org/features/java/swing.html>. Acesso em: novembro de 2008.

PFLEEGER, S. L. **Engenharia de Software**: teoria e prática. 2.ed. São Paulo: Prentice Hall, 2004.

PREMERLANI, W. J.; BLAHA, M. R. An approach for reverse engineering of relational databases. **Commun. ACM**, New York, NY, USA, v.37, n.5, p.42–ff., 1994.

PRESSMAN, R. S. **Engenharia de Software**. 3.ed. São Paulo: Pearson Makron Books, 1995.

REKOFF, M. On reverse engineering. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.3/4, p.244–252, 1985.

SOMMERVILLE, I. **Engenharia de Software**. 6.ed. São Paulo: Addison Wesley, 2003.

WARDEN, R. Reengineering - A Practical Methodology With Commercial Applications. **Software Reuse and Reverse Engineering in Practice**, [S.l.], 1992.

WARREN, I. **The Renaissance of Legacy Systems**: method support for software-system evolution. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.