UNICEUB – CENTRO UNIVERSITÁRIO DE BRASÍLIA

FAET – FACULDADE DE CIÊNCIAS EXATAS E TECNOLOGIA

CURSO DE ENGENHARIA DA COMPUTAÇÃO

FABRÍCIO RIBEIRO BRIGAGÃO

TRANSCODIFICAÇÃO DISTRIBUÍDA DE VÍDEO

BRASÍLIA/DF

2º. SEMESTRE DE 2007

FABRÍCIO RIBEIRO BRIGAGÃO

TRANSCODIFICAÇÃO DISTRIBUÍDA DE VÍDEO

Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para obtenção do grau de Engenheiro de Computação.

Orientador: Prof. Msc. Fabiano Mariath D`Oliveira

BRASÍLIA/DF

2º. SEMESTRE DE 2007

FABRÍCIO RIBEIRO BRIGAGÃO

TRANSCODIFICAÇÃO DISTRIBUÍDA DE VÍDEO

Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para obtenção do grau de Engenheiro de Computação.

Orientador: Prof. Msc. Fabiano Mariath D`Oliveira

Brasília, 03 de dezembro de 2007.

Banca Examinadora

_____

Prof. Msc. Fabiano Mariath D`Oliveira
Orientador

_____

Prof. Msc. Miguel Archanjo Bacellar Goes Telles Júnior
Examinador

_____

Prof. Msc. Antônio José Gonçalves Pinto
Examinador

Resumo

Neste projeto é apresentada a construção de uma ferramenta de *software* que permite a transcodificação de vídeos utilizando processamento distribuído. A arquitetura cliente-servidor foi utilizada para atingir este objetivo, através da utilização de um servidor *web* que possibilita a conexão de diversos nós de processamento. A aplicação cliente é desenvolvida em Java e se comunica com o servidor através de troca de mensagens utilizando protocolo HTTP. O servidor é responsável por receber o vídeo original, dividindo-o em pedaços menores, os quais são distribuídos aos nós de processamento para conversão. Posteriormente, os arquivos gerados são enviados ao servidor, através de FTP, para unificação, resultando no arquivo final.

Palavras-Chave: Processamento Distribuído, Transcodificação Distribuída de Vídeo, Transcodificação, Processamento de Vídeo, Conversão de Vídeo.

Abstract

This project presents the development of a software tool that enables the transcoding of video files through the use of distributed computing. Client-server architecture was used to achieve this goal, through the use of a web server, which provides an interface for the connection of various computing nodes. The client application was developed using Java and it communicates with the server through message exchange using the HTTP protocol. The server is responsible for receiving the original video, splitting it into smaller files, which are then distributed to the computing nodes for conversion. The generated files are sent back to the server, through the use of FTP, and then joined to create the final file.

Keywords: Distributed Computing, Distributed Video Transcoding, Transcoding, Video Processing, Video Conversion.

Agradecimentos

Primeiramente a Deus, por me conceder esta oportunidade e diversas outras durante a vida.

Aos meus pais, que, embora distantes às vezes, não deixaram de demonstrar o seu apoio e me ajudar em momentos difíceis.

Ao meu tio, que me deu a oportunidade de conhecer o mundo, estudar e permitiu o meu primeiro contato com a área de informática, aqui está o fruto desta oportunidade.

À minha avó que durante muito tempo conviveu com minha irmã e eu, nos ajudando.

À minha irmã que sempre foi a pessoa mais próxima de mim e que me ajudou muito.

Ao meu irmão, agora só falta a sua Jaime!

Aos colegas do Banco do Brasil, não tenho como enumerar todos, mas gostaria de agradecer muito ao Marcelo, por me ajudar com conselhos e idéias nas horas certas, ao meu gerente Jeferson Koch, por permitir a realização deste trabalho e ao Kraucer por me mostrar um projeto no *SourceForge* do qual surgiu a idéia principal deste trabalho.

Aos colegas de faculdade tanto da UFU como do UniCeub, em especial ao Danilo pelos livros de Linux emprestados, ao Rodrigo Benin e a todos os demais!

A todos os professores que passaram pela minha vida e que me ajudaram a chegar até aqui.

Ao Banco do Brasil, por contribuir diretamente para a minha formação profissional e me dar meios de adquirir grande parte do que foi utilizado para este projeto.

Finalmente, ao meu orientador, Fabiano Mariath, pela dedicação e competência com que me orientou.

Sumário

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

AAC	*Advanced Audio Codec*
AJAX	*Asynchronous Javascript and XML*
AVI	*Audio-Video Interleave*
BASH	*Bourne Again Shell*
CD	*Compact Disc*
CRT	*Cathode Ray Tube*
CSS	*Cascading Style Sheets*
DDR	*Double Data Rate*
DVD	*Digital Video Disc*
FTP	*File Transfer Protocol*
GB	*Gigabyte*
GHz	*Gigahertz*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
IPTV	*Internet Protocol TV*
JNI	*Java Native Interface*
JVT	*Joint Video Team*
Kbps	*Kilobits* por segundo
LAMP	*Linux Apache MySQL PHP*
LAN	*Local Area Network*
Mbps	*Megabits* por segundo
MB	*Megabytes*
MD5	*Message-Digest Algorithm 5*
MPEG	*Moving Picture Experts Group*

MSDN	*Microsoft Developer Network*
NTSC	*National Television Systems Committee*
P2P	*Peer to Peer*
PAL	*Phase Alternating Line*
PHP	*Hypertext PreProcessor*
RAM	*Random Access Memory*
RFC	*Request for Comment*
RGB	*Red Green Blue*
RIFF	*Resource Interchange File Format*
RLE	*Run-length Encoding*
SECAM	*Séquentiel Couleur avec Mémoire*
SGBD	Sistema Gerenciador de Banco de Dados
SHA	*Secure Hash Algorithm*
SMPTE	*Society of Motion Pictures and Television Engineers*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
UTP	*Unshielded Twisted Pair*
VHS	*Video Home System*
VSFTPD	*Very Secure FTP Daemon*
W3C	*World Wide Web Consortium*

1 INTRODUÇÃO

1.1 Contextualização do Trabalho

A transcodificação de vídeo consiste na conversão de um sinal de vídeo previamente comprimido em outro sinal com características diferentes, como tamanho da imagem, padrão de compressão, dentre outros, visando, principalmente, adaptar o conteúdo para viabilizar a sua transmissão para uma mídia diferente e a exibição do mesmo em vários dispositivos (SYLVESTER, 2006).

Atualmente, o processo de transcodificação está se tornando o foco de diversos provedores de serviços e fabricantes de eletrônicos a medida que os consumidores percebem a dificuldade do gerenciamento de conteúdo digital e transferência do mesmo entre múltiplos dispositivos eletrônicos. De acordo com a análise de mercado efetuada pelo IDC[1], os usuários tendem a exigir facilidade na transferência de conteúdo digital para qualquer dispositivo, bem como sua disponibilização em qualquer lugar a qualquer hora. O estudo acrescenta que a transcodificação será extremamente importante para as empresas de telecomunicações que visam a disponibilização de vídeo utilizando IPTV, tendo em vista que a utilização da transcodificação resultará em uma economia de largura de banda, permitindo a expansão física e melhoria da qualidade dos serviços oferecidos (SYLVESTER, 2006).

As principais características do processo de transcodificação de vídeo são o alto grau de utilização da capacidade de processamento do equipamento utilizado na conversão e o grande consumo de tempo para a realização do processo (SYLVESTER, 2006).

A computação distribuída é definida como um sistema de computadores onde diversos computadores interconectados compartilham as tarefas de processamento delegadas ao sistema (IEEE, 1990). A utilização do processamento distribuído, através da computação distribuída, aliado a processos divisíveis, como a transcodificação de vídeo, pode resultar em ganhos de desempenho, pois pode-se dividir o mesmo em processos menores, que poderão ser processados em paralelo, diminuindo o tempo total de processamento.

[1] *International Data Corporation*: Empresa líder em inteligência de mercado, consultoria e conferências nos segmentos de Tecnologia da Informação e Telecomunicações. Para mais informações sobre a empresa visite <http://www.idc.com>.

1.2 Objetivo do Projeto

Neste trabalho é apresentado um projeto acadêmico que visa o desenvolvimento de uma ferramenta de *software* que realize a transcodificação de vídeo através da utilização de processamento distribuído.

O desenvolvimento da ferramenta engloba as seguintes atividades:

- Configuração do sistema operacional que hospedará os servidores utilizados;
- Configuração do servidor *web*, servidor FTP e banco de dados;
- Modelagem da base de dados;
- Desenvolvimento dos *scripts* PHP de comunicação entre o servidor e os nós de processamento;
- Desenvolvimento da interface *web* para o cadastramento de solicitações, cadastramento de nós de processamento e acompanhamento de solicitações;
- Desenvolvimento dos *scripts* responsáveis pela divisão e unificação dos arquivos de vídeo em *shell script*;
- Desenvolvimento da aplicação cliente, em Java, com suporte as plataformas *Microsoft Windows* e *OpenSUSE* versão 10.2. A ferramenta será utilizada pelos nós de processamento;
- Integração do servidor e da aplicação cliente com as ferramentas de codificação de vídeo;

Como objetivo secundário é verificado se, através da utilização da ferramenta desenvolvida, ocorre redução do tempo total de processamento de um vídeo, quando comparado à apenas um computador, neste caso o servidor utilizado, efetuando o mesmo processamento.

Como forma de tornar o projeto executável em tempo hábil foram impostas as seguintes restrições de escopo:

- A ferramenta terá suporte de transcodificação para o padrão de compressão de vídeo H.264 (MPEG-4 Parte 10) e para os padrões de compressão de áudio MP3 (MPEG-1 *Layer III*) e AAC (MPEG-4 Parte 3);
- A ferramenta aceitará apenas arquivos de entrada que utilizem o formato encapsulador *Audio-Video Interleave (AVI)*;
- O processo de envio do arquivo original de vídeo para o servidor não será abordado neste projeto. O projeto partirá da premissa que o arquivo original já está disponível em diretório visível ao servidor *web*;

- O projeto não irá implementar mecanismos de segurança do meio de comunicação, pois esta implementação aumentaria a complexidade do projeto e demandaria um esforço maior de desenvolvimento;
- Não serão feitas análises ou discutidos os mecanismos de compressão implementados pelos padrões utilizados neste projeto. Os padrões utilizados constituem um meio auxiliar para a demonstração da possibilidade de realização de transcodificação distribuída de vídeo;
- O funcionamento da aplicação cliente será testado apenas em plataforma *Microsoft Windows* e *Linux*. Para o segundo caso será testada apenas a distribuição *OpenSUSE* versão 10.2;
- A ferramenta será testada utilizando rede local *Ethernet* 10/100 Mbps;
- A ferramenta disponibilizará as seguintes opções para transcodificação

    - Encapsulador de Vídeo: MP4;
    - *Codec* de Vídeo: H.264;
    - *Bitrate* de Vídeo: variável;
    - Tamanho da Imagem: variável;
    - Taxa de Quadros por Segundo (*fps*): variável;
    - Parâmetros Adicionais: variável;
    - *Codec* de Áudio: AAC e MP3;
    - *Bitrate* de Áudio: variável.

1.3 Motivação

A transcodificação de vídeo tende a tornar-se cada vez mais importante nos próximos anos, permitindo a adaptação de conteúdo digital, o qual poderá ser acessado e visualizado por qualquer usuário em grande parte dos dispositivos atualmente existentes (SYLVESTER, 2006).

A utilização da transcodificação também se torna importante para permitir a economia de largura de banda para transmissão de conteúdo digital, contribuindo para um melhor aproveitamento dos recursos e aumentando a quantidade de conteúdo que poderá ser entregue pelas empresas de telecomunicações, resultando em até 50% de economia de largura de banda (SYLVESTER, 2006).

A medida que a quantidade de conteúdo disponível em alta definição aumenta, as empresas de telecomunicações e prestadores de serviços de TV via satélite e a cabo terão que utilizar o processo de transcodificação para permitir a entrega de mais canais de conteúdo

além da disponibilização dos mesmos em mais de um formato, permitindo a sua utilização em diferentes dispositivos (SYLVESTER, 2006).

Outras aplicações do processo de transcodificação no mercado atual e futuro incluem (SYLVESTER, 2006):

- Digitalização de conteúdo por usuários domésticos, permitindo a adaptação do mesmo para a utilização em diferentes dispositivos;
- Melhoria da qualidade de vídeo e áudio de videoconferências em empresas;
- Melhoria dos sistemas de vigilância de ambiente tornando as imagens mais nítidas e realísticas com economia de espaço resultando em uma melhoria na qualidade dos dados que podem ser coletados das mesmas e facilitando a sua análise por parte de máquinas ou pessoas;
- Transmissão de vídeo em alta definição para utilização em novas áreas como a telemedicina e a educação à distância;
- Utilização em aplicações militares possibilitando a disponibilização de imagens de alta qualidade em uma variedade de mídias e formatos com pequena ou nenhuma perda de resolução.

A viabilização das aplicações listadas anteriormente depende diretamente da disponibilização de equipamentos de *hardware* e *software* que consigam suportar a computação intensiva que é necessária para efetuar o processo de transcodificação (SYLVESTER, 2006).

A grande variedade de aplicações do processo de transcodificação e a relativa novidade do tema foram as principais motivações para a realização deste trabalho.

1.4 Estrutura do Projeto

Além deste capítulo introdutório, este trabalho está estruturado em seis capítulos, conforme distribuição abaixo:

No **Capítulo 2** é apresentada uma introdução teórica sobre sistemas distribuídos, onde são abordados os principais conceitos destes sistemas e a sua aplicação no projeto.

No **Capítulo 3** são abordados os conceitos fundamentais de vídeo envolvendo a descrição sucinta dos padrões de codificação utilizados e dos formatos encapsuladores.

No **Capítulo 4** é apresentada a construção da ferramenta. São especificadas as tecnologias utilizadas e os motivos de escolha de cada uma. Este capítulo contém também a descrição das principais funcionalidades da ferramenta e o *hardware* utilizado para a realização do desenvolvimento e dos testes da ferramenta. Ao final do mesmo são apresentadas as considerações finais sobre a etapa de desenvolvimento, contendo as dificuldades encontradas, as limitações da ferramenta e os resultados obtidos.

No **Capítulo 5** são apresentadas a conclusão do trabalho e as sugestões para trabalhos futuros.

Por fim, no **Capítulo 6** são apresentadas as referências bibliográficas que serviram de base para a elaboração deste trabalho.

2 SISTEMAS DISTRIBUÍDOS

Neste capítulo são abordados os conceitos de sistemas distribuídos. Serão citados apenas os conceitos utilizados neste projeto, tendo em vista a grande abrangência do tema abordado.

2.1 Sistemas Distribuídos

Atualmente existem diversas definições de sistemas distribuídos, para este trabalho o sistema distribuído será definido como uma coleção de computadores independentes que são vistos pelos seus usuários como um sistema único e coerente (TANENBAUM, 2002).

Um sistema distribuído possui diversas características, dentre as quais se destacam (TANENBAUM, 2002):

a) As diferenças entre os diversos computadores que compõem o sistema e a forma pela qual os mesmos se comunicam não são vistas pelos usuários;

b) A transparência listada acima também se aplica à organização interna do sistema;

c) Os usuários destes sistemas interagem com os mesmos de forma consistente e uniforme, independente de onde ou quando esta interação ocorre;

d) Os sistemas são relativamente fáceis de expandir;

e) Geralmente estão em contínuo funcionamento, embora certas partes possam estar inativas ou com problemas e;

f) Os usuários não possuem conhecimento que certas partes podem estar sendo desativadas, trocadas ou até se recuperando de um erro durante o funcionamento do sistema.

2.1.1 Objetivos de Um Sistema Distribuído

Embora o desenvolvimento de um sistema distribuído seja interessante, alguns objetivos devem estar presentes para justificar o custo relacionado ao desenvolvimento dos mesmos (TANENBAUM, 2002). Os principais objetivos estão listados nas seções seguintes.

2.1.1.1 Conexão de Usuários e Recursos

O objetivo principal de um sistema distribuído é a disponibilização, de forma eficaz e fácil, de recursos remotos, ou seja, localizados em outros locais ou computadores, a usuários, permitindo o compartilhamento destes recursos entre aqueles de forma controlada (TANENBAUM, 2002).

O conceito da palavra recurso utilizada no parágrafo anterior é abrangente, podendo se referir a quase tudo, mas como exemplo pode-se citar impressoras, computadores, a própria informação (dados), arquivos, redes e unidades de armazenamento, dentre vários outros.

Dois grandes motivos tornam este compartilhamento interessante (TANENBAUM, 2002):

a) Economia: o compartilhamento de recursos resulta em uma redução de gastos com equipamentos relativamente caros. O compartilhamento de impressoras é um bom exemplo pois torna desnecessário a compra de uma impressora para cada setor de uma empresa;

b) Facilidade de Troca de Informações e Colaboração: a utilização de sistemas distribuídos permite a comunicação de usuários em diferentes localidades, os quais podem trocar arquivos e informações. A internet é o maior exemplo desta facilidade pois permite a troca de informações, arquivos de áudio e vídeo entre diversos usuários.

2.1.1.2 Transparência

Outro objetivo importante de um sistema distribuído é esconder o fato que os seus processos e recursos estão fisicamente distribuídos em diversos computadores (TANENBAUM, 2002).

Com base no conceito acima, pode-se dizer que um sistema distribuído é transparente quando o mesmo se apresenta para os seus usuários e aplicações como um único sistema de computadores.

O conceito de transparência pode ser aplicado a diversos aspectos de um sistema distribuído, conforme listado nos próximos itens (TANENBAUM, 2002).

2.1.1.2.1 Transparência de Acesso

A transparência de acesso torna invisível para os usuários as diferenças das formas de representação de dados, bem como a forma pela qual os recursos são acessados (TANENBAUM, 2002).

Em um sistema que implemente o conceito acima, as diferenças de formatos de dados existentes entre os diversos sistemas operacionais ou plataformas, bem como as diferenças dos padrões de nomenclatura de arquivos, no caso de sistemas de arquivos, são invisíveis para os usuários.

2.1.1.2.2 Transparência de Localidade

A transparência de localidade torna impossível para um usuário determinar onde um recurso disponibilizado se encontra fisicamente no sistema. Esta transparência pode ser obtida através da utilização de nomes que representam os recursos físicos (TANENBAUM, 2002).

Como exemplo de transparência de localidade pode-se citar a utilização de um URL[2] para viabilizar o acesso a arquivos hospedados em um determinado *website*. O URL não revela a localização física do arquivo, sendo que o mesmo pode ser movido ou o servidor trocado de lugar sem que o usuário esteja consciente desta mudança.

2.1.1.2.3 Transparência de Migração

A transparência de migração permite a mudança de localização de um determinado recurso sem que o acesso ao mesmo seja afetado (TANENBAUM, 2002).

O exemplo citado no item anterior também se aplica a este, pois embora um *website* possa ser movido de um servidor para outro não é necessário a alteração do URL que permite o acesso ao mesmo.

---

[2] Sigla de *Uniform Resource Locator*. Representa a sequência de caractéres que permite o acesso a um determinado recurso na internet. Ex: http://www.google.com

2.1.1.2.4 Transparência de Relocação

A transparência de relocação é uma evolução da transparência de migração e consiste na possibilidade de mudança de localização de um determinado recurso enquanto o mesmo está em uso, sem que o usuário ou a aplicação percebam esta relocação (TANENBAUM, 2002).

Um exemplo de transparência de relocação ocorre quando um usuário que está utilizando uma conexão de rede sem fio através do seu *notebook* se movimenta de um local para outro sem que o mesmo seja desconectado em nenhum instante.

2.1.1.2.5 Transparência de Replicação

A transparência de replicação permite que, embora um determinado recurso esteja replicado em diversos locais para melhoria de desempenho do sistema, todas as réplicas sejam vistas como um único recurso para o usuário do sistema (TANENBAUM, 2002).

Um exemplo deste tipo de transparência é o espelhamento de servidores. Embora existam vários servidores para o tratamento de requisições provenientes de usuários, para os mesmos o grupo de servidores é visto como apenas um servidor.

2.1.1.2.6 Transparência de Concorrência

O conceito de concorrência envolve a utilização de um mesmo recurso por mais de um usuário em um determinado instante. A transparência de concorrência permite que vários usuários acessem o mesmo recurso ao mesmo tempo, sem que cada um desses usuários saiba que este recurso está sendo acessado pelos demais (TANENBAUM, 2002).

Como exemplo deste tipo de transparência pode-se citar o acesso a registros de uma determinada tabela em banco de dados por vários usuários simultaneamente, pois cada usuário desconhece que o mesmo registro está sendo acessado pelos demais.

2.1.1.2.7 Transparência de Falhas

A transparência de falhas permite que um sistema distribuído desative ou se recupere de uma falha em um determinado recurso sem que o usuário tome conhecimento que a mesma

ocorreu (TANENBAUM, 2002).

2.1.1.2.8 Transparência de Persistência

A transparência de persistência permite que um sistema distribuído mova ou recupere dados da memória física, como um disco rígido, para a memória volátil, sem que o usuário perceba que esta movimentação ocorreu (TANENBAUM, 2002).

2.1.1.2.9 Grau de Transparência

Embora seja desejável que os sistemas distribuídos implementem todos os aspectos de transparência citados anteriormente, deve-se ter em mente que um alto grau de transparência afeta diretamente o desempenho do sistema (TANENBAUM, 2002).

A conclusão é que, ao implementar um sistema distribuído, é importante ter como objetivo a transparência do sistema, em todos os seus aspectos, mas a mesma deve ser considerada em conjunto com outros fatores como o desempenho do mesmo (TANENBAUM, 2002).

2.1.1.3 Abertura

Um objetivo importante de sistemas distribuídos é a abertura dos mesmos. Um sistema é dito aberto quando o mesmo disponibiliza serviços de acordo com regras que descrevem a sintaxe e a semântica destes serviços (TANENBAUM, 2002).

Geralmente, os serviços são especificados através da utilização de interfaces. As interfaces definem de forma clara as operações que estão disponíveis, os parâmetros de entrada e saída, bem como as possíveis exceções que podem ocorrer para um determinado serviço (TANENBAUM, 2002).

A utilização de interfaces também permite que diferentes equipes ou empresas desenvolvam implementações completamente diferentes de uma determinada interface, que operam exatamente da mesma forma, o que permite a troca de um determinado componente por outro sem a necessidade de alterações no restante do sistema (TANENBAUM, 2002).

As especificações de interfaces devem ser completas, possuindo tudo o que é

necessário para que sejam implementadas, e neutras, pois não devem prescrever como uma implementação deve ser desenvolvida (TANENBAUM, 2002).

Interfaces completas e neutras são fatores importantes para a interoperabilidade e portabilidade (BLAIR[3], 1998 *apud* TANENBAUM, 2002). A interoperabilidade é a característica que indica até que ponto duas implementações ou componentes de diferentes fabricantes podem co-existir e trabalhar em conjunto através, apenas, da confiança nos serviços fornecidos por cada um com base em um padrão comum (TANENBAUM, 2002). Já a portabilidade caracteriza até que ponto uma aplicação desenvolvida para um sistema distribuído *A* pode ser utilizada, sem modificações, em um sistema distribuído *B*, que implementa as mesmas interfaces de *A* (TANENBAUM, 2002).

2.1.1.4 Escalabilidade

A escalabilidade pode ser mensurada em pelo menos três dimensões (NEUMAN[4], 1994 *apud* TANENBAUM, 2002):

a) Escalabilidade de Tamanho: o sistema é escalável em tamanho quando pode-se adicionar mais usuários ou recursos ao mesmo;

b) Escalabilidade Geográfica: o sistema é geograficamente escalável quando os usuários e recursos podem estar distantes um dos outros, e;

c) Escalabilidade Administrativa: o sistema é administrativamente escalável quando se mantém facilmente gerenciável mesmo ao se expandir através de diferentes organizações administrativas independentes.

Infelizmente, um sistema distribuído escalável em uma ou mais das dimensões listadas acima, geralmente exibe perda de desempenho à medida que o mesmo se expande (TANENBAUM, 2002).

2.1.1.4.1 Problemas de Escalabilidade

Embora a escalabilidade seja um aspecto importante em sistemas distribuídos, existem vários problemas que devem ser solucionados para que seja possível a implementação da mesma.

---

[3] BLAIR, G.; Stefani, J.B. *Open Distributed Processing and Multimedia*. Reading: Addison-Wesley, 1998.
[4] NEUMAN, B. Scale in Distributed Systems. In: CASAVANT, T.; SINGHAL, M. *Readings in Distributed Operating Systems*. Los Alamitos: IEEE Computer Society Press, 1994. p. 463-469.

Os principais problemas, podem ser caracterizados de acordo com cada uma das dimensões citadas anteriormente, conforme itens a seguir (TANENBAUM, 2002).

2.1.1.4.1.1 Escalabilidade de Tamanho

Á medida que um sistema distribuído aumenta, cresce o número de usuários e recursos que devem ser suportados e atendidos pelo mesmo. Esta necessidade de crescimento geralmente é confrontada com as limitações de serviços, dados ou algoritmos centralizados (TANENBAUM, 2002).

A tabela 2.1 lista alguns exemplos das limitações de escalabilidade citadas acima.

Tabela 2.1 – Exemplos de Limitações de Escalabilidade (TANENBAUM, 2002)

Conceito	Exemplo
Serviços Centralizados	Apenas um servidor para todos os usuários
Dados Centralizados	Apenas um servidor de dados para a guarda de todos os registros de uma lista telefônica
Algoritmos Centralizados	Efetuar o roteamento de rede tendo como requisito o conhecimento de toda a rede

A utilização de apenas um servidor para o atendimento de todos os usuários pode resultar na criação de um gargalo no sistema à medida que o mesmo cresce (TANENBAUM, 2002). Infelizmente, em algumas situações, como a guarda de dados altamente confidenciais, a descentralização dos serviços é inviável, pois a utilização de mais de um servidor pode resultar em vulnerabilidades de segurança (TANENBAUM, 2002).

A centralização de dados também pode se tornar um gargalo para o sistema distribuído, pois mesmo que o servidor que contenha os dados possua uma grande capacidade de armazenamento e processamento, a quantidade de conexões simultâneas ao mesmo pode resultar na saturação dos meios de comunicação que viabilizam o acesso ao servidor (TANENBAUM, 2002).

Assim como os itens anteriores, a centralização de algoritmos também pode se tornar um gargalo. Como exemplo pode-se citar um roteador que necessite de coletar todas as informações disponíveis em uma rede para, somente após isto, analisar os dados e escolher as

melhores rotas. Como resultado, a transferência desses dados para o roteador consumiria grande parte da capacidade do meio de comunicação e tornaria o processo de roteamento extremamente lento (TANENBAUM, 2002).

De forma geral, a utilização de algoritmos descentralizados é preferível, sendo que os mesmos possuem as seguintes características (TANENBAUM, 2002):

a) Nenhum dispositivo ou máquina possui informações completas sobre a situação atual do sistema;
b) Os dispositivos ou máquinas tomam decisões com base apenas nas informações locais disponíveis aos mesmos;
c) A falha de um dispositivo ou máquina não inutiliza o algoritmo, e;
d) Os algoritmos não assumem implicitamente que todos os dispositivos e máquinas estão sincronizados.

Não é necessário detalhar as três primeiras características pois as descrições das mesmas são claras, mas o mesmo não ocorre com a quarta característica. Como exemplo, pode-se afirmar que um algoritmo que implemente a regra “Precisamente às 12:00:00, todos os computadores deverão reportar a sua situação” irá falhar, pois é impossível garantir a sincronia exata de todos os relógios de todas as máquinas vinculadas a um sistema distribuído (TANENBAUM, 2002).

2.1.1.4.1.2 Escalabilidade Geográfica

A escalabilidade geográfica também possui problemas, pois a grande maioria dos sistemas distribuídos são baseados em comunicação síncrona. Neste tipo de comunicação, um aplicativo ou dispositivo que solicita um serviço aguarda até que receba uma resposta do servidor (TANENBAUM, 2002).

A comunicação síncrona funciona de forma satisfatória em redes locais, onde a comunicação entre dois ou mais computadores é rápida, levando apenas alguns microssegundos (TANENBAUM, 2002).

A mesma solução citada acima já não é aplicável a sistemas distribuídos com usuários e recursos dispersos em diversas localidades, pois nestes casos os meios de comunicação já não são confiáveis, podendo ocorrer perda ou corrupção dos dados transferidos (TANENBAUM, 2002).

2.1.1.4.1.3 Escalabilidade Administrativa

Finalmente, existem limitações inerentes à escalabilidade administrativa. A principal limitação se refere à segurança, ou seja, um sistema distribuído que se expande para mais de uma organização administrativa, terá que se defender de ataques provenientes desta nova esfera. O recíproco também é válido, pois a nova esfera atendida pelo sistema distribuído terá que se defender de ataques provenientes do mesmo (TANENBAUM, 2002).

2.1.1.4.2 Técnicas de Escalabilidade

Com base nas formas de escalabilidade citadas e nos problemas relacionados à implementação das mesmas, os quais geralmente resultam em problemas de desempenho causados pela capacidade limitada dos servidores ou dos meios de comunicação, pode-se dizer que existem basicamente três técnicas de escalabilidade: ocultação das latências de comunicação, distribuição e replicação (TANENBAUM, 2002).

2.1.1.4.2.1 Comunicação Assíncrona

A ocultação das latências de comunicação, ou seja, o tempo que um determinado processo aguarda até que receba uma resposta à uma solicitação, é uma solução aplicável para a obtenção de escalabilidade geográfica. A idéia principal desta solução é evitar que um determinado processo tenha que aguardar o recebimento de uma resposta à uma solicitação enviada a um servidor (TANENBAUM, 2002).

A solução citada acima é obtida através da utilização do conceito de comunicação assíncrona durante a construção da aplicação. Em aplicações que implementam este conceito, enquanto o servidor está processando a requisição, a mesma efetua outros processamentos, geralmente, processos que independem da resposta enviada pelo servidor. Ao receber a resposta do servidor, a aplicação é interrompida e um procedimento é acionado para completar o processamento referente à resposta recebida (TANENBAUM, 2002).

Embora a utilização da comunicação assíncrona viabilize a escalabilidade geográfica, esta solução não é aplicável a todos os casos. Como exemplo pode-se citar aplicações interativas, que recebem dados de um usuário e com base nos mesmos enviam uma requisição. O usuário desta aplicação não pode fazer nada a não ser esperar pela resposta do servidor (TANENBAUM, 2002).

2.1.1.4.2.2 Distribuição

A técnica de distribuição consiste em desmembrar um determinado componente em pedaços menores, espalhando os mesmos por diversas partes do sistema (TANENBAUM, 2002).

Como exemplo de distribuição pode-se citar a internet. Embora a mesma transpareça para a maioria dos usuários como um único sistema de informações baseado em documentos, cada um possuindo o seu próprio endereço URL, sabe-se que a internet está fisicamente distribuída através de diversos servidores, sendo que cada um deles é responsável pela disponibilização de um certo número de documentos (TANENBAUM, 2002).

2.1.1.4.2.3 Replicação

Como os problemas de escalabilidade geralmente se manifestam através da degradação do desempenho do sistema, é, geralmente, uma boa idéia a replicação de alguns componentes em diversas partes do sistema distribuído (TANENBAUM, 2002).

A replicação de recursos resulta em uma melhoria da disponibilidade do sistema, permitindo também o balanceamento de carga entre os componentes, melhorando o desempenho do mesmo (TANENBAUM, 2002). Em sistemas geograficamente distribuídos, a existência de uma cópia de um componente em um local próximo ao da origem da requisição pode resultar em uma redução dos problemas de latência de comunicação (TANENBAUM, 2002).

2.1.1.4.2.4 *Caching*

O *caching* é uma forma especial de replicação, a diferença reside no fato que o procedimento de *caching* é uma decisão efetuada pelo cliente, ao contrário da replicação que é uma decisão efetuada pelo dono do recurso (TANENBAUM, 2002).

Como exemplo de *caching* pode-se citar a guarda de dados efetuada pelos navegadores, também chamados de *web browsers*, para páginas recentemente visitadas pelos usuários. Esta guarda permite que, durante um próximo acesso, parte dos dados, como imagens, não tenham que ser novamente recuperados, contribuindo para a melhoria do desempenho.

O principal problema referente à utilização de replicação e *caching* é a necessidade de manter todas as cópias de um determinado componente exatamente iguais, ou seja, as alterações efetuadas em uma cópia devem ser replicadas em todas as demais. Esta necessidade de atualização pode resultar em problemas de consistência, que podem ou não ser toleráveis em um determinado sistema distribuído (TANENBAUM, 2002).

2.1.2 Modelo Cliente-Servidor

O modelo cliente-servidor é caracterizado por clientes que solicitam serviços de servidores, sendo que existe um consenso entre diversos pesquisadores e desenvolvedores que esta linha de pensamento facilita a compreensão e o gerenciamento da complexidade dos sistemas distribuídos (TANENBAUM, 2002).

Em um modelo cliente-servidor simples, os processos de um sistema distribuído são divididos em dois grupos, que podem se sobrepor: servidores e clientes (TANENBAUM, 2002). O servidor é um processo que implementa um determinado serviço, como um serviço de banco de dados. O cliente é um processo que requisita um serviço de um servidor através do envio de uma requisição e aguarda pela resposta a esta requisição por parte do servidor. Esta interação cliente-servidor é demonstrada na figura 2.1 (TANENBAUM, 2002):

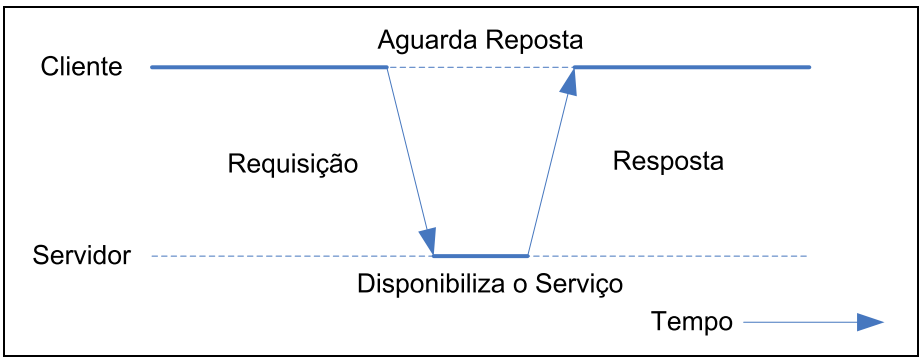

Figura 2.1 – Interação Básica entre um cliente e um servidor

2.1.2.1 Comunicação no Modelo Cliente-Servidor

No modelo cliente-servidor a comunicação entre ambas as partes pode ser implementada através da utilização de um protocolo não orientado a conexões quando o meio de comunicação que integra os mesmos é relativamente confiável, como no caso de redes locais (TANENBAUM, 2002).

Neste tipo de comunicação, o cliente simplesmente empacota uma mensagem, em conjunto com os dados necessários, e envia a requisição para o servidor. O servidor aguarda por requisições e ao receber alguma, o mesmo processa a requisição e empacota os resultados em uma mensagem de resposta, que é enviada ao cliente (TANENBAUM, 2002).

A utilização de um protocolo não orientado a conexão torna a comunicação entre o cliente e o servidor eficiente, mas esta eficiência só é alcançada se não ocorrer perda ou corrupção das mensagens (TANENBAUM, 2002).

Em sistemas distribuídos que utilizam meios de comunicação pouco confiáveis a alternativa é a utilização de protocolos orientados à conexão, sendo que grande parte dos mesmos é baseada em conexões TCP/IP (TANENBAUM, 2002). Neste caso, sempre que um cliente requisita um serviço, o mesmo estabelece uma conexão com o servidor antes de enviar a mensagem. O servidor utiliza as mesmas conexões criadas para responder à solicitação, após isto a conexão é destruída.

2.1.2.2 Divisão da Aplicação em Camadas

O modelo cliente-servidor já foi amplamente debatido, principalmente com relação ao que distingue um cliente de um servidor. Isto ocorre pois em uma aplicação pode-se ter um processo que age como servidor ao receber solicitações de outros processos e que ao mesmo tempo age como cliente ao encaminhar novas requisições, baseadas nas anteriores, para outros servidores, como um banco de dados (TANENBAUM, 2002).

Tendo em vista que grande parte das aplicações que utilizam o modelo cliente-servidor possuem suporte ao acesso a banco de dados, é bastante difundido a seguinte divisão (TANENBAUM, 2002):

1. Camada de Interface com o Usuário
2. Camada de Processamento
3. Camada de Dados

A camada de interface contém tudo o que é necessário para interagir com o usuário, a camada de processamento contém as aplicações e a camada de dados contém os registros sobre os quais operações são realizadas (TANENBAUM, 2002).

2.1.2.2.1 Camada de Interface com o Usuário

A camada de interface com o usuário geralmente é implementada nos clientes e consiste em programas que permitem a interação do usuário com as aplicações (TANENBAUM, 2002).

2.1.2.2.2 Camada de Processamento

A camada de processamento interliga as camadas de interface e dados, e contém as funcionalidades principais de uma aplicação. Esta camada pode ser composta de processos simples ou por várias aplicações que, em conjunto, recuperam dados e efetuam processamento com base nos comandos recebidos da camada de interface (TANENBAUM, 2002).

2.1.2.2.3 Camada de Dados

Em um modelo cliente-servidor, a camada de dados, geralmente implementada no servidor, contém os programas que mantém os dados sobre os quais as aplicações operam (TANENBAUM, 2002). A consistência é uma propriedade importante desta camada pois mesmo que nenhuma aplicação esteja sendo executada, os dados serão sempre armazenados em algum local para futura utilização (TANENBAUM, 2002).

Comumente, são utilizados sistemas gerenciadores de bancos de dados relacionais nesta camada, neste caso, a consistência é mantida através da guarda de dados adicionais como a descrição de tabelas, as restrições para a adição, modificação ou remoção de dados, além de dados específicos de aplicações, como os procedimentos que são acionados somente em determinadas situações, também chamados de *triggers* (TANENBAUM, 2002).

A utilização de bancos de dados relacionais contribui para a separação entre as camadas de processamento e dados, tornando as mesmas independentes, como conseqüência, a alteração na organização dos dados não afeta a organização das aplicações, da mesma forma que a alteração das aplicações não afeta a organização dos dados (TANENBAUM, 2002).

2.1.2.3 Arquitetura Multinível

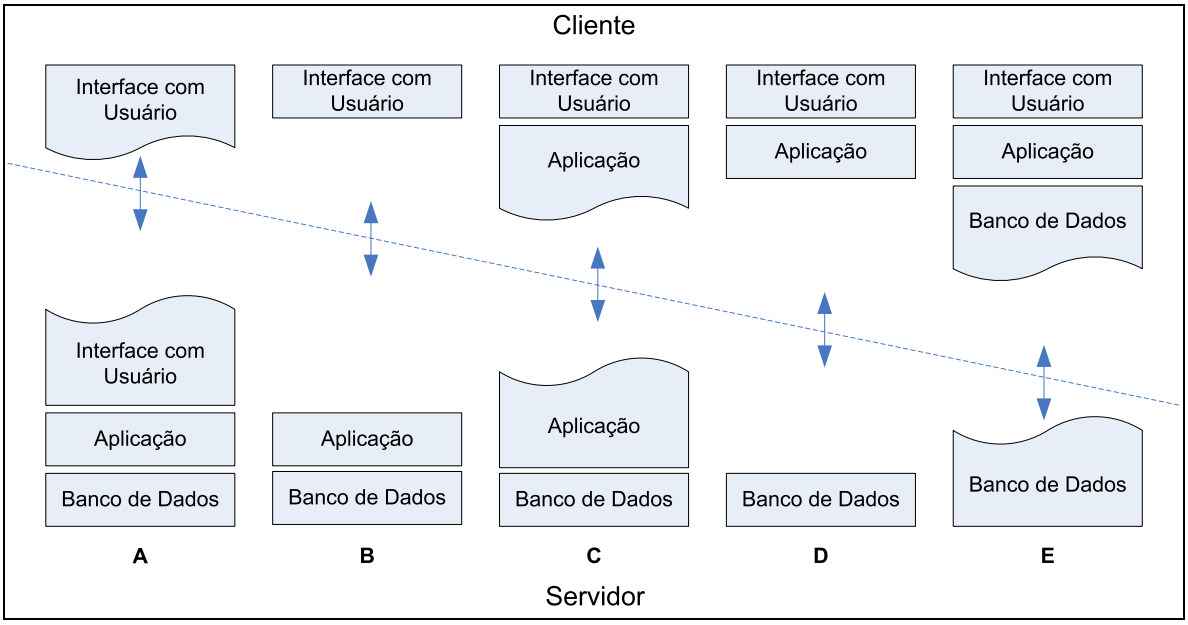

Figura 2.2 – Formas de organização de processos em sistemas cliente-servidor
Fonte: TANENBAUM, 2002

Existem várias formas de organização do modelo cliente-servidor, sendo que uma das abordagens é a divisão dos programas com base nas camadas citadas nos itens anteriores. A figura 2.2 demonstra as diversas formas de divisão das camadas entre o cliente e o servidor conforme descrito abaixo:

a) Nesta forma, somente parte da interface reside na máquina cliente, sendo que o servidor possui controle sobre os dados a serem apresentados;

b) Nesta forma, toda a interface com o usuário reside na máquina cliente, sendo que a mesma se comunica com o servidor através de um protocolo específico de aplicação. A interface efetua o processamento apenas dos dados necessários para a sua exibição;

c) Nesta forma, parte da aplicação também reside na máquina cliente, este modelo é utilizado em aplicações que exigem o preenchimento de formulários por parte do usuário, neste caso, o código que efetua a validação do formulário também é disponibilizado em conjunto com a interface;

d) Nesta forma, a interface e a aplicação residem na máquina cliente. Esta organização é muito utilizada em redes onde a máquina cliente é um computador que se conecta através de uma rede a um banco de dados ou a um sistema de arquivos distribuído. Todo o processamento ocorre no cliente, mas a busca de dados para o processamento e a gravação de arquivos é feita através de operações com o servidor;

e) Nesta forma, a máquina cliente possui, além da interface e da aplicação, parte dos

dados em sua memória. Como exemplo pode-se citar os navegadores *web*, que efetuam a guarda de dados temporários dos *websites* visitados recentemente por um usuário.

2.1.3 Comunicação por Mensagens

A comunicação através da troca de mensagens é caracterizada por dois itens principais: a mensagem utilizada na comunicação e os mecanismos utilizados para receber e enviar as mesmas (JIA, 2005). Nos próximos itens são descritos os dois conceitos.

2.1.3.1 Mensagem

Uma mensagem é uma coleção de objetos de dados e possui duas partes, sendo a primeira o cabeçalho, que geralmente é fixo e a segunda parte, o corpo da mensagem, que pode ou não possuir tamanho variável (JIA, 2005).

As mensagens podem possuir qualquer tamanho e podem conter dados ou ponteiros para dados localizados fora da mesma, sendo que estes dados, geralmente, são determinados pelo processo que está enviando a mensagem, embora alguns dados possam ser provenientes do próprio sistema (JIA, 2005).

Uma característica importante das mensagens consiste no fato de poderem ser estruturadas ou não. Uma mensagem estruturada possui um formato padrão, isto facilita a comunicação em sistemas heterogêneos, pois os mesmos podem interpretar facilmente a mensagem com base no padrão determinado (JIA, 2005). As mensagens também podem não ser estruturadas, neste caso, a principal vantagem é a flexibilidade, pois pode-se modificar a mensagem para que a mesma se adapte ao que será transmitido (JIA, 2005).

2.1.3.2 Mecanismos de Troca de Mensagens

Na comunicação por troca de mensagens as mesmas são enviadas e recebidas através da execução de duas primitivas básicas: *send* (envio) e *receive* (recebimento). O diagrama da figura 2.3 demonstra o uso dessas duas primitivas (JIA, 2005):

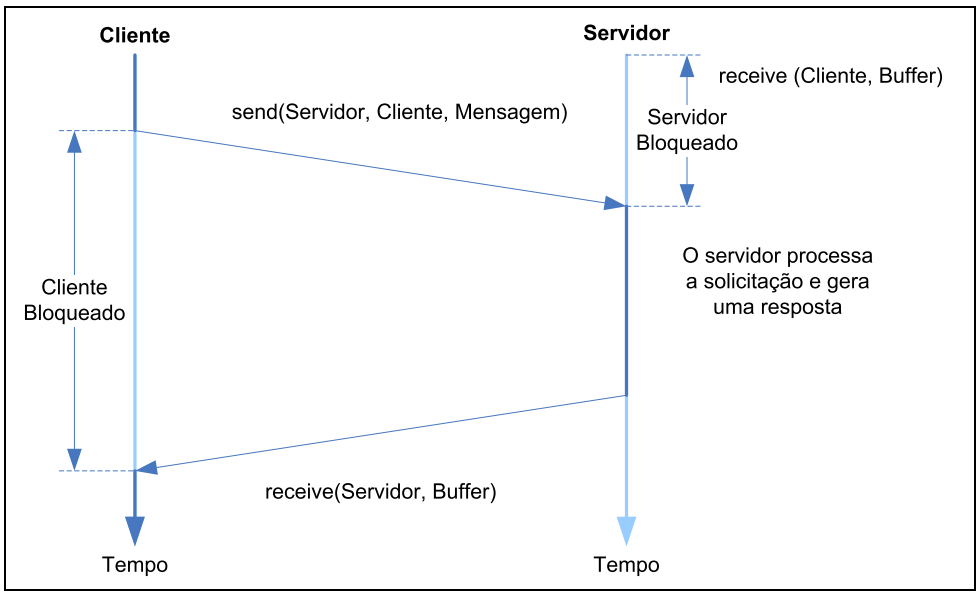

Figura 2.3 – Diagrama de execução das primitivas *send* e *receive*

Com base no diagrama da figura 2.3, pode-se perceber que tanto o servidor como o cliente utilizam a primitiva *send* para enviar mensagens, já a primitiva *receive* é utilizada para receber a resposta gerada pelo servidor.

Além das duas primitivas acima, a comunicação por troca de mensagens possui outras características que a definem, das quais serão explicadas as listadas abaixo (JIA, 2005):

- Comunicação direta e indireta
- Comunicação com bloqueio e sem bloqueio
- Comunicação com *Buffer* e sem *Buffer*
- Comunicação confiável e não-confiável

2.1.3.2.1 Comunicação Direta e Indireta

A comunicação direta é caracterizada pelo envio de mensagens diretamente para o processo receptor, por parte do emissor da mensagem, geralmente, através da utilização de nomes para identificar o processo. A desvantagem deste tipo de comunicação é a falta de transparência, pois caso o processo mude de nome ou local poderá ocorrer falha na comunicação, além disso, qualquer mudança no nome do receptor resultará na necessidade de ajustes de todos os processos que o utilizam (JIA, 2005).

A comunicação indireta é caracterizada pelo envio de mensagens para um local fixo responsável pelo recebimento das mesmas, geralmente, uma porta (JIA, 2005). Uma porta

pode ser descrita, de forma abstrata, como um objeto do sistema operacional onde as mensagens podem ser colocadas e retiradas, ou seja, as mensagens podem ser recebidas e enviadas através de uma porta (JIA, 2005).

Cada porta possui uma fila finita onde as mensagens recebidas ficam até serem retiradas pelo processo receptor para processamento. Geralmente, um processo no servidor possui direitos de uso da porta, o que permite que o mesmo retire as mensagens recebidas. É importante citar que uma porta pode ser utilizada por somente um processo em um determinado instante (JIA, 2005).

Diversos serviços utilizam portas como forma de permitir o recebimento de diversas solicitações de diferentes clientes, como por exemplo, os servidores *web*, que utilizam, comumente, a porta 80 para receberem solicitações.

2.1.3.2.2 Comunicação com Bloqueio e sem Bloqueio

A comunicação por troca de mensagens pode ser classificada como sendo com bloqueio ou sem bloqueio. Na comunicação com bloqueio o emissor fica bloqueado até que a mensagem enviada pelo mesmo seja recebida, processada pelo servidor e uma resposta seja enviada e recebida. Já na comunicação sem bloqueio, o emissor só fica bloqueado até que a mensagem enviada seja copiada para o *buffer* do servidor (JIA, 2005).

A utilização da comunicação sem bloqueio é mais complexa, pois exige a utilização de interruptores para informar ao emissor que uma resposta foi recebida do servidor, além disso, o servidor deve controlar o *buffer*, pois caso o mesmo fique cheio, o servidor terá que bloquear o recebimento de mensagens, o que por sua vez poderá bloquear o emissor (JIA, 2005).

A figura 2.3 listada anteriormente demonstra, claramente, a comunicação com bloqueio pois no diagrama citado o cliente está bloqueado até que receba uma reposta do servidor.

2.1.3.2.3 Comunicação com Buffer e sem Buffer

Na comunicação com a utilização de *buffer*, as mensagens enviadas pelos clientes ao servidor são colocadas em uma área de memória temporária chamada de *buffer* onde aguardam o processamento por parte do processo responsável. Caso o *buffer* fique cheio ou estoure o cliente pode aguardar para tentar enviar, novamente, a mensagem ou o mesmo pode desistir do envio após receber a resposta que o *buffer* do servidor está cheio (JIA, 2005).

Na comunicação sem a utilização de *buffer* o cliente envia a mensagem para o servidor e aguarda enquanto a requisição é processada e uma resposta é gerada. Esta forma de comunicação é mais fácil de implementar do que a comunicação com *buffer*.

As principais desvantagens da comunicação com *buffer* são a sua complexidade, pois requer a criação, destruição e gerenciamento de *buffers* e o risco de ocorrer problemas com a consistência dos dados e com os clientes caso o processo dono do *buffer* encerre ou seja terminado pelo sistema operacional (JIA, 2005).

2.1.3.2.4 Comunicação Confiável e Não Confiável

Em sistemas de computadores podem ocorrer diversas falhas, como a falha ou *crash* de um ou mais computadores ou falha nas redes de comunicação. A ocorrência destes eventos pode resultar em perda de mensagens que estão tramitando pelo sistema, as quais podem ser respostas de requisições ou as próprias requisições.

Na comunicação não confiável, o emissor simplesmente envia uma mensagem através da rede de comunicação, sendo que não há nenhuma garantia de que a mesma será entregue ou retransmitida caso seja perdida (JIA, 2005).

A comunicação confiável utiliza mecanismos para garantir que uma determinada mensagem enviada seja recebida. O processo emissor envia a mensagem e aguarda a resposta de aceite, também chamado de *acknowledge* ou *ack*, do servidor. Caso a resposta não seja recebida, o emissor possui um temporizador que expira e indica que não houve resposta dentro do prazo esperado, neste caso, o mesmo pode reenviar a mensagem novamente ou adotar outros procedimentos (JIA, 2005).

O diagrama da figura 2.4 resume as diferenças entre a comunicação confiável e a não-confiável (JIA, 2005):

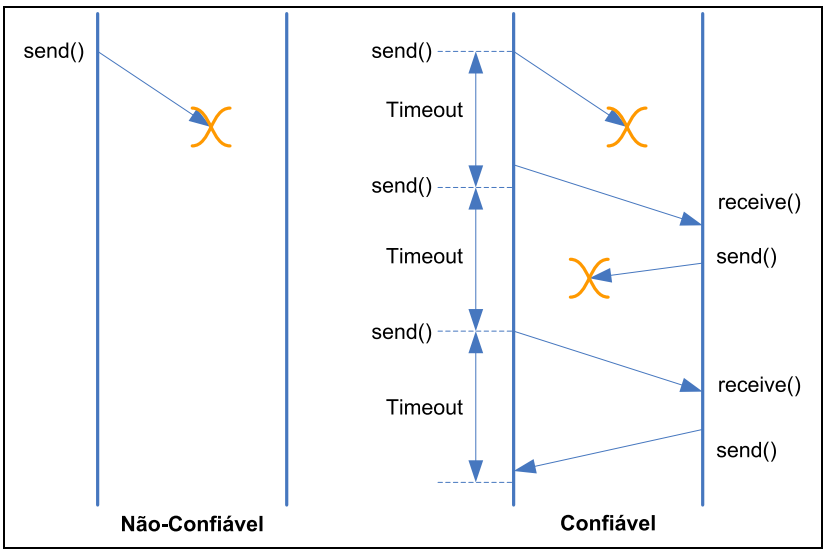

Figura 2.4 – Diagrama de comunicação confiável e não-confiável

No diagrama da figura 2.4, o emissor envia uma mensagem, que é perdida durante a transmissão e não é retransmitida, caracterizando a comunicação não-confiável. A segunda imagem demonstra o mesmo procedimento, mas após a expiração do temporizador o emissor reenvia a mensagem a qual é processada mas a resposta é perdida. Com a utilização do temporizador, novamente, o emissor reenvia a mensagem, a qual é processada e a resposta recebida com sucesso, caracterizando a comunicação confiável.

2.1.3.3 Modelo de Comunicação Utilizado

Neste trabalho é utilizada a comunicação por troca de mensagens através do protocolo HTTP, que é descrito abaixo. Para a transferência de arquivos dos nós de processamento para o servidor foi utilizado o protocolo FTP, também descrito abaixo. A comunicação utilizada, com base nas classificações citadas anteriormente, pode ser classificada como indireta, com bloqueio, sem *buffer* e confiável.

2.1.3.3.1 Protocolo HTTP

O protocolo HTTP, *HyperText Transfer Protocol*, definido pelas RFC 1945 (versão 1.0) e 2616 (versão 1.1), com a coordenação da W3C[5] e IETF[6], é utilizado para a comunicação entre um navegador e um servidor de páginas, também chamado de servidor *web* e possui as seguintes características (COMER, 2006):

[5] *World Wide Web Consortium*. Vide <http://www.w3.org>
[6] *Internet Engineering Task Force*. Vide <http://www.ietf.org>

- Nível de aplicação: o protocolo opera no nível de aplicação e utiliza o protocolo TCP como base para a comunicação, logo, o mesmo utiliza um protocolo de transporte confiável;

- Requisição e resposta: após o estabelecimento da conexão, um dos lados, geralmente o navegador, envia uma requisição HTTP à qual o servidor *web* responde;

- Sem estado: cada requisição é autocontida, ou seja, o servidor não sabe reconhecer se uma nova requisição faz parte de um mesmo navegador que se conectou anteriormente;

- Transferência bidirecional: geralmente, o navegador solicita uma página *web* e o servidor transfere uma cópia ao mesmo. Há também a possibilidade de transferência de dados do navegador para o servidor através da utilização de formulários;

- Suporte ao uso de *cache*: como forma de melhorar o tempo de resposta, os navegadores guardam uma cópia da página requisitada em memória, caso a mesma seja solicitada novamente, o navegador verifica junto ao servidor se houve alguma alteração desde a última cópia armazenada para, após isto, decidir se irá atualizar a página ou apenas exibir a disponível.

O protocolo HTTP permite o envio de requisições GET e POST, por parte dos navegadores, para os servidores *web.* A requisição GET é geralmente utilizada para receber algum arquivo do servidor e possui sintaxe simples, conforme listado abaixo:

```
GET http://www.meusite.com.br/pagina1.html HTTP/1.1
```

O exemplo acima demonstra uma requisição GET para a obtenção da página “pagina1.html” disponível no *website* “www.meusite.com.br”. A última parte da requisição se refere à versão do protocolo HTTP utilizada, neste caso a versão 1.1.

Além das requisições GET, o protocolo HTTP permite o envio de requisições POST, que são utilizadas para a transferência de dados para o servidor. As requisições POST são geradas a partir de formulários HTML (*HyperText Markup Language)* e quando os mesmos são submetidos, os campos destes formulários bem como os seus valores, são enviados ao servidor, que pode recuperar estes dados e efetuar o processamento da requisição (COMER,

2006).

Embora os métodos citados anteriormente sejam os mais conhecidos, o protocolo HTTP suporta diversos outros. A tabela 2.2 exibe um resumo de todos os métodos disponíveis:

Tabela 2.2 – Os métodos internos de solicitações HTTP (TANENBAUM, 2003)

Método	Descrição
GET	Solicita a leitura de uma página *web*
HEAD	Solicita a leitura do cabeçalho da mensagem
PUT	Solicita o armazenamento de uma página *web*
POST	Anexa dados à um recurso (por exemplo, uma página *web*)
DELETE	Remove a página *web,* desde que possua autorização
TRACE	Ecoa a solicitação recebida, utilizado para depuração
CONNECT	Não utilizado atualmente, reservado para uso futuro
OPTIONS	Consulta certas opções e propriedades do servidor

É importante citar que um dos principais problemas da primeira versão do protocolo HTTP era o *overhead* causado pela abertura e fechamento de conexões TCP a cada requisição, o que comprometia o tempo de resposta e tornava a comunicação ineficiente. Este problema foi solucionado na versão 1.1 do protocolo, através da utilização de conexões persistentes, ou seja, a conexão é aberta durante a primeira requisição do cliente ao servidor, após isto, todas as requisições posteriores utilizam a conexão já aberta. Posteriormente, o cliente e o servidor encerram a conexão quando não houver mais necessidade de novas requisições (COMER, 2006).

A utilização de conexões persistentes como forma padrão de comunicação na versão 1.1 do protocolo HTTP solucionou o principal problema de desempenho do mesmo.

Neste trabalho o protocolo HTTP foi escolhido devido à arquitetura cliente-servidor utilizada. A aplicação utiliza um servidor *web* para o recebimento de solicitações dos nós de processamento e envio de respostas, além disso, a facilidade de utilização do protocolo HTTP, em conjunto com o HTML, para o envio e recebimento de mensagens também se tornou decisiva para a escolha do mesmo. A utilização do método POST, para o envio de informações, permite a definição de diversos campos de informação separados, os quais podem ser interpretados por qualquer servidor que utilize o protocolo HTTP. Contribuiu também para esta decisão o fato do protocolo HTTP utilizar o protocolo TCP, que é confiável, para a transferência dos dados, o que ajuda a evitar a ocorrência de falhas.

2.1.3.3.2 Protocolo FTP

O protocolo FTP surgiu da evolução dos primeiros protocolos de transferência existentes inclusive antes do surgimento do TCP/IP (COMER, 2006). Este protocolo possui inúmeras vantagens que o tornam extremamente versátil (COMER, 2006):

- Representação uniforme: o protocolo pode ser utilizado para a transferência de arquivos entre sistemas operacionais e arquiteturas diferentes, por exemplo, pode-se utilizar um computador com o sistema operacional *Microsoft Windows* para enviar e receber arquivos de um servidor FTP localizado em um computador com o sistema operacional *Linux*;

- Acesso interativo: os clientes FTP geralmente são interativos, permitindo que seres humanos acessem e manipulem os dados de um servidor FTP;

- Especificação de formato: o protocolo FTP permite que seja especificado o tipo de um arquivo, como, por exemplo, texto ou binário, e o conjunto de caracteres utilizados;

- Controle de autenticação: o protocolo FTP exige que os clientes enviem o nome de *login* e uma senha para o servidor antes de solicitar qualquer arquivo, caso os dados não sejam válidos, o servidor recusa o acesso do usuário ao mesmo.

Os servidores FTP geralmente utilizam duas portas padrão para permitir a comunicação e a transferência de arquivos com os clientes, as portas 21 e 20. A porta 21 é utilizada para a conexão do usuário ao servidor, nesta porta, após estabelecida a conexão, trafegará todos os dados de controle entre o servidor e o cliente. Como não é possível trafegar dados de controle e transferir arquivos na mesma porta, o servidor FTP utiliza a porta 20 para efetuar a transferência de arquivos para o cliente (COMER, 2006).

O modelo de comunicação acima resultou em inúmeros problemas com *firewalls* de segurança por utilizar portas fixas. Para contornar o problema, o protocolo FTP recebeu uma extensão chamada de FTP passivo, que permite a transferência de arquivos entre o servidor e o cliente em portas aleatórias definidas entre os mesmos, desta forma, a comunicação não fica restrita às portas 20 e 21 (COMER, 2006).

A grande maioria dos servidores e clientes FTP já suportam o modo de transferência passivo, o que os tornam utilizáveis em redes que possuem *firewalls* de segurança (COMER, 2006).

Neste trabalho, o protocolo FTP é utilizado para efetuar a transferência de pedaços de vídeo transcodificados dos nós de processamento para o servidor. A utilização do protocolo permite um maior controle sobre a transferência, além de uma melhoria na segurança do sistema, tendo em vista que é necessário a autenticação prévia do nó de processamento junto ao servidor FTP. O modo de transferência passivo também é utilizado neste projeto, tornando o servidor FTP e a aplicação cliente utilizáveis em redes que possuem *firewalls*.

2.1.3.3.3 Resumo das Características do Modelo

A listagem abaixo exibe um resumo das características do modelo de comunicação utilizado com base nos conceitos explanados nos itens anteriores:

- Comunicação Indireta: a comunicação utilizada é indireta pois utiliza portas no servidor para o recebimento de requisições. A porta 80 é utilizada o servidor *web* e as portas 20, 21 são utilizadas pelo servidor FTP. Para permitir a transferência em modo passivo são utilizadas as portas 1030 a 1045;

- Comunicação sem *buffer*: neste trabalho não há utilização de *buffer* no servidor. A relação entre os nós de processamento e o servidor exige que a resposta seja recebida logo após a requisição para que o nó prossiga com suas atribuições. A utilização de um modelo de comunicação com *buffer* não traria vantagens tendo em vista que o nó de processamento ficaria ocioso enquanto não recebesse uma resposta do servidor, além de tornar o desenvolvimento do projeto mais complexo;

- Comunicação com bloqueio: com base na justificativa do item anterior, pode-se concluir que, neste trabalho, o nó de processamento ficará bloqueado enquanto aguarda a resposta do servidor;

- Comunicação confiável: a comunicação é confiável pois está baseada no protocolo de transporte confiável TCP. Em conjunto com este protocolo, medidas como a utilização de temporizadores garantem que mesmo que uma mensagem seja perdida, o nó de processamento conseguirá se recuperar da falha e adotar os procedimentos estabelecidos.

2.1.4 Tolerância à Falhas

Quando um sistema não consegue atingir os seus objetivos o mesmo é dito falho, ou seja, quando um ou mais serviços disponibilizados por um sistema distribuído ficam indisponíveis, mesmo que parcialmente, o sistema falhou (TANENBAUM, 2002).

O erro faz parte dos possíveis estados de um sistema, sendo que o mesmo pode resultar em uma falha de funcionamento. O erro é causado por uma falha, logo, é importante a detecção da falha como formar de evitar a ocorrência de erros que podem resultar em uma falha de funcionamento do sistema (TANENBAUM, 2002).

As falhas são geralmente classificadas em transientes, intermitentes ou permanentes, conforme descrito abaixo (TANENBAUM, 2002):

a) Transientes: Falhas transientes ocorrem apenas uma vez, ou seja, caso a operação seja repetida, a mesma falha não voltará a ocorrer;

b) Intermitentes: Falhas intermitentes ocorrem às vezes, aparecendo e desaparecendo por algum tempo antes de voltarem a ocorrer. Este tipo de falha é a mais difícil de ser detectada e corrigida;

c) Permanentes: Falhas permanentes continuam a ocorrer até que o motivo da mesma seja encontrado e corrigido. Erros de *software* e componentes queimados são exemplos de falhas permanentes.

A figura 2.5 resume as principais causas das falhas descritas acima (JIA, 2005):

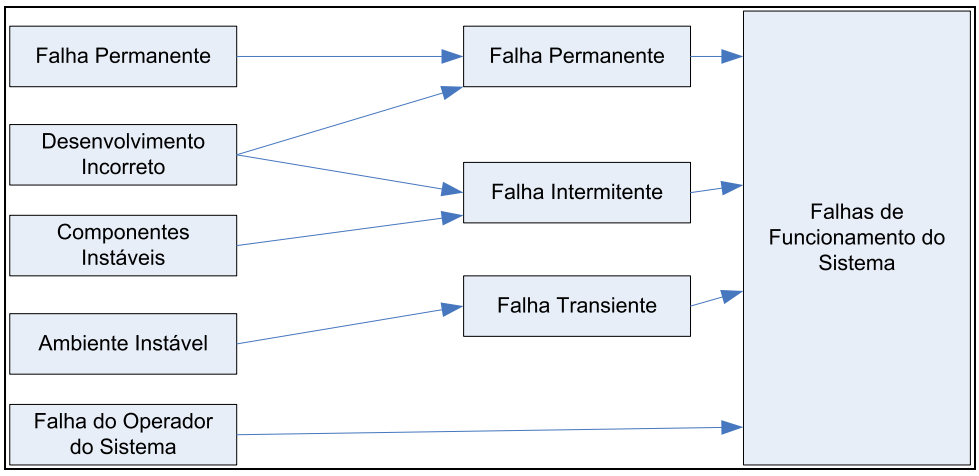

Figura 2.5 – Principais causas de falhas

2.1.4.1 Conceitos Básicos

A tolerância a falhas está intimamente relacionada à confiança no funcionamento de um sistema. Esta confiança no funcionamento está relacionada a diversos requisitos, incluindo os listados abaixo (KOPETZ[7], 1993 *apud* TANENBAUM, 2002):

a) Disponibilidade
b) Confiabilidade
c) Segurança
d) Manutenabilidade

2.1.4.1.1 Disponibilidade

O conceito de disponibilidade é uma propriedade que indica a probabilidade de um determinado sistema estar operando corretamente em um determinado instante e estar disponível para executar as suas funções atendendo a solicitações (TANENBAUM, 2002).

Com base no conceito acima pode-se dizer que para um sistema que possui alta disponibilidade é altamente provável que o mesmo esteja funcionando em um determinado instante.

2.1.4.1.2 Confiabilidade

A confiabilidade é uma propriedade que indica se um sistema consegue funcionar continuamente sem falhas. Um sistema altamente confiável indica que o mesmo irá continuar funcionando durante um longo intervalo de tempo sem falhas (TANENBAUM, 2002).

É importante mencionar que este conceito difere do conceito de disponibilidade, tendo em vista que um sistema pode ser altamente disponível mas não ser confiável, como exemplo, pode-se citar um sistema que fica indisponível por apenas um milissegundo a cada hora, neste caso, o sistema ainda é altamente disponível mas não é confiável (TANENBAUM, 2002).

[7] KOPETZ, H.; VERISSIMO, P. Real Time and Dependability Concepts. In: MULLENDER, S. *Distributed Systems*. Wokingham: Addison-Wesley, 1993. p. 411-446.

2.1.4.1.3 Segurança

A segurança no funcionamento se refere à situação em que um sistema temporariamente deixa de funcionar corretamente sendo que durante este período nada catastrófico acontece (TANENBAUM, 2002).

Um exemplo do conceito acima são os sistemas de controle de uma usina nuclear. Estes sistemas devem ser altamente seguros pois caso os mesmos falhem, sequer por alguns segundos, os resultados podem ser catastróficos (TANENBAUM, 2002).

2.1.4.1.4 Manutenabilidade

O conceito de manutenabilidade indica com que facilidade o sistema que falhou pode ser recuperado. Um sistema altamente manutenível pode ser recuperado rapidamente podendo exibir um alto grau de disponibilidade caso as falhas sejam detectadas e reparadas automaticamente (TANENBAUM, 2002).

2.1.4.2 Tipos de Falhas

Em sistemas distribuídos as falhas podem ocorrer em um ou mais servidores, estas falhas também podem ocorrer nos canais de comunicação entre os mesmos ou até nos dois ao mesmo tempo (TANENBAUM, 2002).

As relações de dependência que geralmente existem entre os servidores de um sistema tornam difícil a detecção do motivo de uma determinada falha, tendo em vista que a origem do problema pode estar em outro servidor ao qual o servidor afetado se conecta (TANENBAUM, 2002).

As falhas podem ser classificadas conforme descrito na tabela 2.3.

Tabela 2.3 – Diferentes tipos de falhas (CRISTIAN[8], 1991 *apud* TANENBAUM, 2002) (HADZILACOS[9], 1993 *apud* TANENBAUM, 2002)

Tipo de Falha	Descrição
1. Falha de *Crash*	O servidor deixa de funcionar, mas estava

[8] CRISTIAN, F. Understanding Fault-Tolerant Distributed Systems. *Commun. ACM*, v. 34, n. 2, p. 55-78, fev. 1991.
[9] HADZILACOS, V.; TOUEG, S. Fault-Tolerant Broadcasts and Related Problems. In: MULLENDER, S. *Distributed Systems*. Wokingham: Addison-Wesley, 1993. p. 97-145.

	funcionando corretamente antes da falha.
2. Falha de Omissão	O servidor não consegue responder às requisições que estão chegando.
2.1 Omissão de Recebimento	O servidor não consegue responder às mensagens que estão chegando.
2.2 Omissão de Envio	O servidor não consegue enviar mensagens.
3. Falha de Temporização	A resposta do servidor está fora do intervalo de tempo especificado.
4. Falha de Resposta	A resposta do servidor está incorreta.
4.1 Falha de Conteúdo	O conteúdo da resposta está errado.
4.2 Falha de Transição de Estado	O servidor desvia do fluxo de controle padrão.
5. Falha Arbitrária	O servidor pode produzir respostas arbitrárias em períodos arbitrários.

Os tipos de falha listados na tabela 2.3 são sucintamente descritos abaixo (TANENBAUM, 2002):

a) Falha de *Crash*: o *crash* de um sistema ocorre quando o mesmo pára de funcionar, deixando de responder a qualquer solicitação após o *crash*. Um exemplo comum é o *crash* de um sistema operacional, quando o mesmo ocorre a única solução é reiniciar o computador;

b) Falha de Omissão: este tipo de falha ocorre quando o servidor não consegue responder à uma solicitação;

i. Omissão de Recebimento: este tipo de falha pode ser ocasionado por uma requisição enviada que nunca foi recebida pelo servidor. Esta falha de recebimento pode ser ocasionada por um erro de conexão ou pelo fato de não haver nenhum processo em modo de escuta para receber novas solicitações;

ii. Omissão de Envio: este tipo de falha ocorre quando o servidor, após processar uma requisição, não consegue enviar a resposta ao solicitante. Geralmente pode ser ocasionada por um estouro do *buffer* de envio do servidor.

c) Falha de Temporização: este tipo de falha ocorre quando uma resposta enviada pelo servidor está fora do intervalo de tempo aceito pelo cliente. Geralmente, esta falha está relacionada à tempos de resposta elevados, o que caracteriza um problema de desempenho do sistema;

d) Falha de Resposta: neste tipo de falha, a resposta fornecida pelo servidor está incorreta, o que a torna um dos tipos de falha mais graves;

   a. Falha de Conteúdo: este tipo de falha ocorre quando o servidor fornece um valor de resposta errado à uma requisição, como, por exemplo, uma ferramenta de busca que sempre retorna páginas que não estão no critério de busca especificado pelo usuário;

   b. Falha de Transição de Estado: este tipo de falha ocorre quando o servidor reage de forma inesperada à uma requisição. Como exemplo, pode-se citar o recebimento pelo servidor de uma mensagem que o mesmo não reconhece, caso não tenham sido tomadas precauções, o servidor poderá incorretamente processar a mensagem conforme o procedimento padrão, caracterizando uma falha.

e) Falha Arbitrária: este é o tipo de falha mais grave, neste caso o servidor produz dados de saída ou mensagens que nunca deveriam ter sido produzidas, mas que não podem ser detectadas como incorretas.

2.1.4.3 Técnicas para Atingir a Confiabilidade

Existem diversas técnicas que podem ser utilizadas para a construção de sistemas distribuídos confiáveis, tais como a redundância e as técnicas para evitar e detectar falhas (JIA, 2005).

Cada uma delas é descrita sucintamente nos próximos itens.

2.1.4.3.1 Redundância

A redundância é um requisito básico em qualquer técnica que visa a obtenção de confiabilidade. Os elementos redundantes são os componentes de um sistema que podem ser removidos sem afetar o desempenho do mesmo, assumindo que o restante do sistema não possua falhas (JIA, 2005).

Existem diversas classificações de redundância, a seguir, cada uma delas é descrita sucintamente (JIA, 2005):

- Classificação de acordo com a utilização de recursos
    - Redundância de Espaço
        - Redundância de *Hardware*: utiliza uma quantidade maior de memória, módulos funcionais e outros equipamentos de *hardware* para fornecer informações de recuperação;
        - Redundância de Software: utiliza versões alternativas ou independentes de algoritmos para fornecer informações de recuperação.
    - Redundância de Tempo: utiliza maior capacidade de processamento através de métodos que podem ser iguais ou diferentes ao original para efetuar uma nova tentativa de execução da operação que falhou.

- Outras classificações
    - Redundância Estática: utiliza componentes redundantes para mascarar falhas de *hardware* ou *software* dentro de um determinado módulo. A saída de cada módulo permanece inalterada desde que a proteção fornecida pela redundância funcione. As falhas em um módulo redundante são tratadas como independentes, ou seja, não possuem interligação com as falhas em outro módulo igual;
    - Redundância Dinâmica: permite que erros sejam exibidos nas saídas dos módulos, sendo que quando um erro é detectado uma ação de recuperação elimina ou corrige o erro resultante. A utilização deste tipo de redundância implica na utilização de processos de recuperação e de detecção de falhas.

Caso ocorra uma falha, o sistema que implementa a redundância pode tomar as seguintes decisões (JIA, 2005):

- Isolamento da Falhas: o isolamento da falha limita o efeito da mesma à apenas uma área do sistema, prevenindo a disseminação da mesma;

- Detecção de Falhas: detecção da falha através de verificação de paridade, consistência, violações de protocolo, dentre outros recursos. Vale citar que a detecção não funciona em todos os casos e que existe um período entre o momento em que ocorre a falha e o momento em que a mesma é detectada, o que pode resultar em danos ao sistema;

- Mascaramento de Falhas: mascaramento de falhas através de utilização de técnicas para esconder os efeitos causados pelas mesmas;

- Nova Tentativa: em parte dos casos, a efetuação de uma nova tentativa pode resultar em sucesso, principalmente quando ocorre uma falha transiente que não causa dano físico ao sistema;

- Diagnóstico: o sistema pode coletar informações sobre a localização da falha e/ou suas propriedades;

- Reconfiguração: o sistema pode substituir o componente que está apresentando falhas, isolar o mesmo do resto do sistema ou utilizar parte da capacidade dos recursos do sistema para reprocessar a requisição;

- Reinicialização: este procedimento pode ser utilizado quando a recuperação do sistema é impossível ou quando o mesmo não foi desenvolvido com suporte à recuperação;

- Reparação: consiste na reparação do componente que está apresentando a falha;

- Reintegração: consiste na reintegração do módulo reparado ao sistema.

2.1.4.3.2 Técnicas para Evitar Falhas

As técnicas para evitar falhas são utilizadas para diminuir a probabilidade de falha de um sistema, principalmente de falhas transientes, através da manipulação dos fatores que afetam a quantidade de falhas (JIA, 2005).

As principais técnicas estão listadas abaixo (JIA, 2005):

- Mudanças no Ambiente: através de mudanças no ambiente onde o sistema está localizado pode-se reduzir o número de falhas. Como exemplo pode-se citar a utilização de um sistema de refrigeração para controlar a temperatura do ambiente onde os equipamentos do sistema estão localizados;

- Mudanças na Qualidade: melhoria da qualidade através da realização de testes extensivos, depuração de código e verificação, que contribuem para a eliminação ou redução de erros nos sistemas e falhas de componentes;

- Mudanças na Complexidade: redução do número de componentes ou equipamentos utilizados, diminuição do número de funções, diminuição do número de módulos, diminuição do número de interações entre os módulos. Todos os itens citados contribuem para a redução da complexidade de um sistema, conseqüentemente reduzindo o número de falhas.

2.1.4.3.3 Técnicas para Detecção de Falhas

A premissa básica das técnicas de detecção de falhas consiste em assumir que um sistema irá falhar independente da estrutura ou processo de desenvolvimento do mesmo (JIA, 2005).

As principais técnicas de detecção de falhas estão listadas abaixo (JIA, 2005):

- Duplicação: é aplicável a todas as áreas de desenvolvimento de computadores. Consiste na utilização de duas cópias idênticas, quando uma falha ocorre em uma das cópias, as duas não serão mais idênticas e uma comparação entre as mesmas detecta a falha. Esta técnica está limitada à detecção de falhas que não são comuns a ambas as cópias, ou seja, a mesma não consegue detectar as falhas que ocorrem em ambas as cópias ou falhas que ocorrem no código responsável pela comparação;

- Código de Detecção de Erro: consiste na utilização de verificação de paridade, *checksums* de arquivos e dados, dentre outras formas para a detecção de falhas;

- Auto-verificação: um sistema que implementa a auto-verificação verifica o seu próprio comportamento comparando-o a padrões pré-especificados como formar de confirmar que está funcionando corretamente;

- Temporizadores: os temporizadores são utilizados para verificar se um processo está funcionando corretamente. O seu funcionamento é simples e consiste na ativação de um contador quando o processo é inicializado, este contador é executado como um processo separado e é decrementado sistematicamente, caso o mesmo expire antes do processo reiniciar o mesmo (após a conclusão do processamento), o sistema detecta que houve uma falha no processo;

- Verificação de Consistência: a verificação de consistência verifica os valores intermediários e resultados finais de um processamento para garantir que os mesmos estejam dentro de padrões aceitáveis, que podem ser definidos previamente ou ser uma função dos parâmetros de entrada.

3 CONCEITOS DE CODIFICAÇÃO DE VÍDEO

Neste capítulo são descritos os conceitos básicos de vídeo digital que são inerentes a este trabalho. Em conjunto, também são descritos os encapsuladores de vídeo e padrões de codificação disponibilizados ou utilizados pela ferramenta criada.

3.1 Introdução

No mundo real o ser humano percebe o movimento como um fluxo contínuo de eventos, uma combinação de estímulos visuais, auditivos e temporais. As câmeras de vídeo sempre estiveram presentes para capturar o movimento em meios analógicos, como filmes e fitas VHS.

Conceitualmente, pode-se dizer que o movimento é a captura de uma seqüência de imagens com um intervalo fixo de tempo entre as mesmas. Cada imagem capturada é chamada de *frame* (quadro). A velocidade com que os *frames* são capturados ou exibidos é chamado de *frame rate* (taxa de quadros), a qual é medida em número de *frames* por segundo (*fps)* (WONG, 2007).

3.2 Padrões de Vídeo

Existem três padrões de vídeo que são utilizados pelas televisões analógicas: NTSC, PAL e SECAM. Cada padrão define regras técnicas para a codificação e a transmissão dos sinais, sendo que cada um possui o seu *frame rate* específico e um número específico de linhas por cada *frame* capturado. Estes atributos são importantes, pois são traduzidos para os padrões de vídeo digital, como, por exemplo, o número de linhas de cada padrão, o qual é utilizado para determinar a altura de um *frame* quando o mesmo é convertido para vídeo digital (WONG, 2007).

Abaixo, cada padrão é descrito sucintamente (WONG, 2007):

- NTSC: a sigla significa *National Television Systems Committee*, ou seja, Comitê Nacional de Sistemas de Televisão, órgão dos Estados Unidos que desenvolveu este padrão;
    - Utilização: Este padrão é utilizado na América do Norte, Japão, Taiwan e em partes do Caribe e da América do Sul;

- *Frame Rate*: 29.97 *frames* por segundo (*fps*). Antigamente, o *frame rate* era 30 *frames* por segundo para transmissão de sinal em preto e branco, mas o mesmo foi alterado para poder acomodar informações adicionais de cor, o que diminuiu o *frame rate* para 29.97;

- PAL: a sigla significa *Phase Alternating Line*, ou seja, Linha com Alternação de Fase, referindo-se à como os sinais são codificados neste padrão;
  - Utilização: Este padrão é utilizado em grande parte da Europa Ocidental, Austrália, América do Sul, Nova Zelândia e em grande parte dos países asiáticos;
  - *Frame Rate*: 25 *frames* por segundo (*fps*);

- SECAM: a sigla significa *Séquentiel Couleur avec Mémoire*, ou seja, Cor Seqüencial com Memória.
  - Utilização: Este padrão é utilizado na França, nos países que compunham a União Soviética e na Europa Oriental.
  - *Frame Rate*: 25 *frames* por segundo (*fps*);

3.3 Varredura Progressiva e Entrelaçada

Uma imagem que é exibida em uma televisão ou em um monitor de computador é composta de diversas linhas horizontais, essas linhas são traçadas na tela uma a uma, individualmente, para formar a imagem final (WONG, 2007).

Cada um dos padrões descritos no item anterior possui uma quantidade específica de linhas, conforme listagem abaixo (WONG, 2007):

- NTSC: 525 linhas, das quais 480 são utilizadas para exibir a imagem;
- PAL e SECAM: 625 linhas, das quais 576 são utilizadas para exibir a imagem.

Existe uma diferença importante entre os monitores de computador e as televisões analógicas no que se refere à forma como as imagens são montadas em tela. Os monitores de computador montam a imagem linha a linha em apenas uma etapa, começando pelo canto superior esquerdo até o fim, esta forma de montagem é chamada de varredura progressiva ou *progressive scan* (WONG, 2007).

Já os padrões analógicos utilizados nas televisões montam a imagem em duas etapas: na primeira etapa, as linhas pares são recuperadas e exibidas, na segunda etapa, as linhas ímpares são recuperadas e utilizadas para preencher o espaço existente entre as linhas pares. Esta forma de montar a imagem em tela é chamada de varredura entrelaçada ou *interlaced scan* (WONG, 2007).

Um dos problemas principais da varredura entrelaçada consiste no fato que as linhas de cada etapa são capturadas em intervalos de tempo diferentes, o que resulta em uma descontinuidade na imagem caso um objeto que está se movendo em grande velocidade seja filmado. Estas descontinuidades não são perceptíveis nas televisões analógicas tendo em vista a baixa qualidade de imagem das mesmas, mas o mesmo não ocorre quando essas mesmas imagens são exibidas em um monitor de computador, neste caso, a diferença se torna nítida (WONG, 2007).

As imagens da figura 3.1 mostram uma imagem capturada com varredura entrelaçada, bem como as linhas recuperadas em cada etapa (WONG, 2007).

(a)

(b)

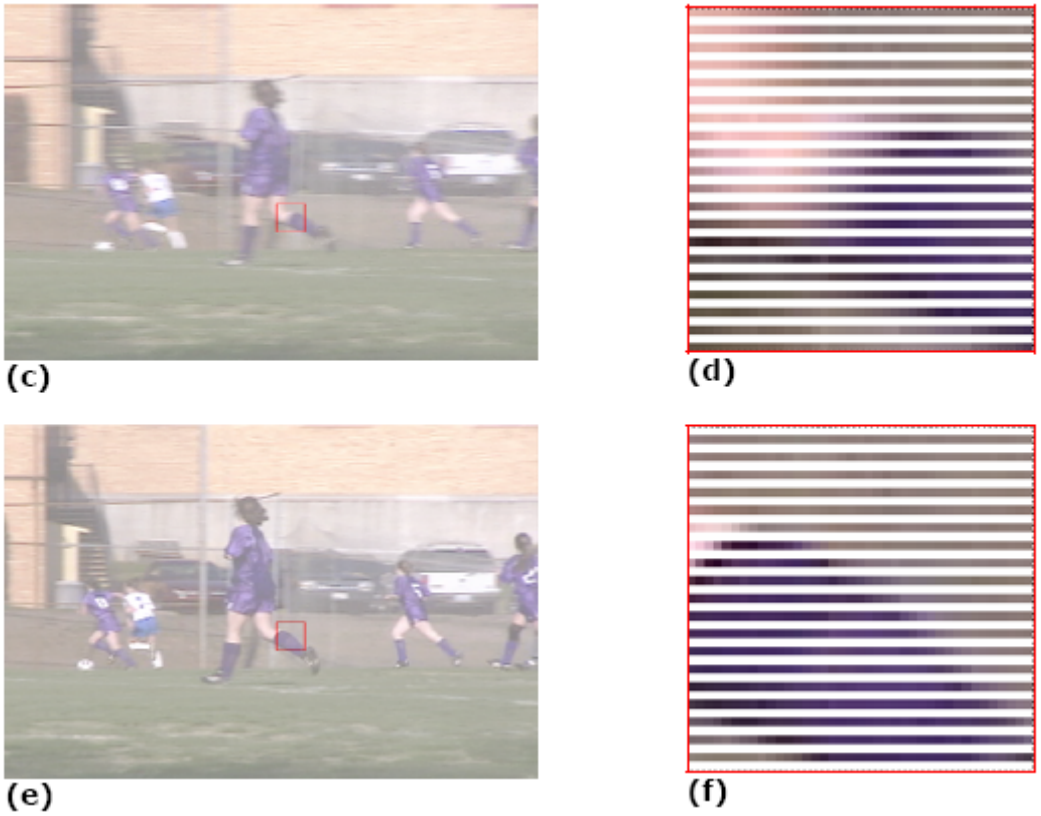

Figura 3.1 – Demonstração de uma imagem entrelaçada

Os itens abaixo descrevem cada imagem da figura 3.1 (WONG, 2007):

(a) Nesta figura é exibida uma imagem capturada com varredura entrelaçada com uma jogadora de futebol correndo, ou seja, uma imagem em movimento;

(b) Nesta figura é exibido o detalhe da perna da jogadora, esta imagem está marcada com um quadrado na imagem *a*. Pode-se perceber, claramente, as linhas entrelaçadas e a perda de definição que ocorre com objetos em movimento quando se utilizada a varredura entrelaçada;

(c) Nesta figura somente as linhas referentes à primeira etapa são exibidas;

(d) Nesta figura é exibido o detalhe da imagem *c*, mostrando as linhas pares referentes à primeira etapa de recuperação;

(e) Nesta figura somente as linhas referentes à segunda etapa são exibidas;

(f) Nesta figura é exibido o detalhe da imagem *e*, mostrando as linhas ímpares referentes à segunda etapa de recuperação.

Uma forma de minimizar os efeitos da varredura entrelaçada em monitores de computador é a utilização da técnica de desentrelaçamento, também chamada de *deinterlace.*

Esta técnica consiste em descartar as linhas de uma das etapas de recuperação e preencher os espaços em branco através da duplicação ou interpolação das linhas que sobrarem (WONG, 2007).

Na figura 3.2 é exibida a mesma imagem anterior após a aplicação do desentrelaçamento (WONG, 2007).

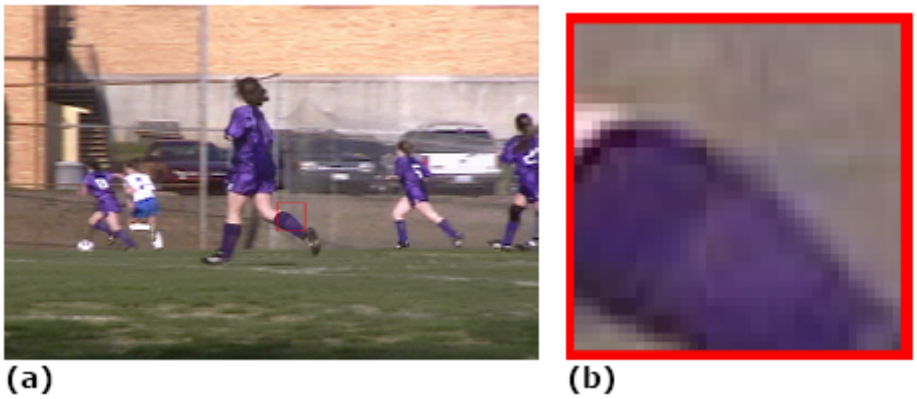

Figura 3.2 – Demonstração de uma imagem desentrelaçada

3.4 Sistemas de Cor

O sistema de cor RGB é muito utilizado para a exibição de imagens digitais, mas para a exibição de vídeo os sistemas de cor YUV e YIQ, que utilizam o conceito de luminância e crominância, são utilizados.

3.4.1 Sistema de Cor RGB

O sistema RGB (*Red Green Blue)* é composto das três cores primárias: vermelho, verde e azul. A combinação dessas três cores resulta em todas as outras cores, sendo que a combinação das mesmas com a máxima intensidade resulta na cor branca (WONG, 2007).

Este sistema de cor é utilizado em monitores CRT, que possuem feixes de luz vermelha, verde e azul. Através da utilização destas três cores o monitor monta a imagem em tela, conforme mostra a figura 3.3 (WONG, 2007).

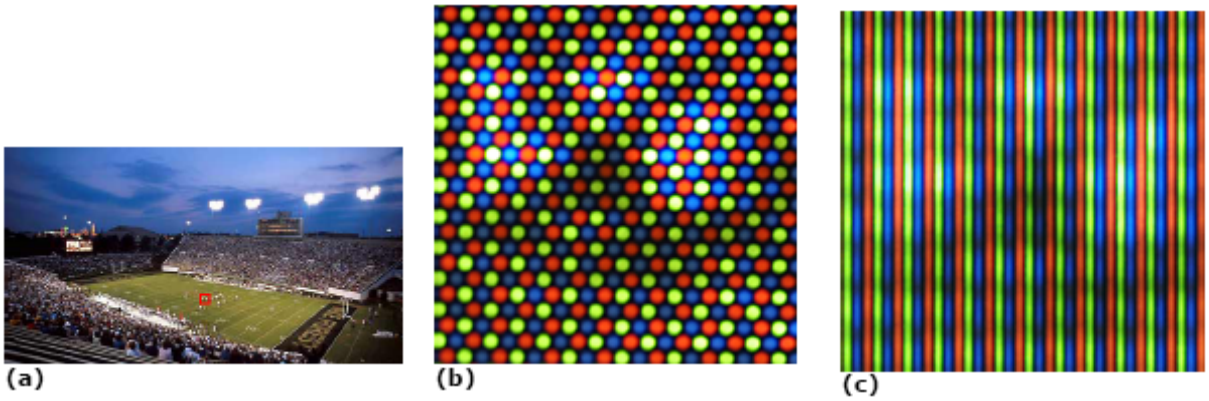

Figura 3.3 – Representação de cores em um monitor CRT

Os itens abaixo descrevem as imagens exibidas na figura 3.3:

(a) Imagem original exibida em um monitor CRT;

(b) Neste detalhe pode-se observar como a imagem é exibida em um monitor CRT. Pode-se ver, claramente, os feixes de luz vermelha, verde e azul, que, em conjunto, tornam possível a exibição da imagem;

(c) Detalhe da exibição da mesma imagem em um monitor *Sony* com a tecnologia *Trinitron*. Os feixes de luz possuem uma organização diferente das existentes em um monitor CRT comum.

O modelo RGB pode ser representado por um cubo em três dimensões sendo que cada um dos três eixos corresponde à uma das cores primárias azul, vermelho e verde. Os valores para cada cor variam de 0 a 255 em cada eixo, sendo que a combinação dos valores de cada um dos eixos resulta em uma cor específica (WONG, 2007).

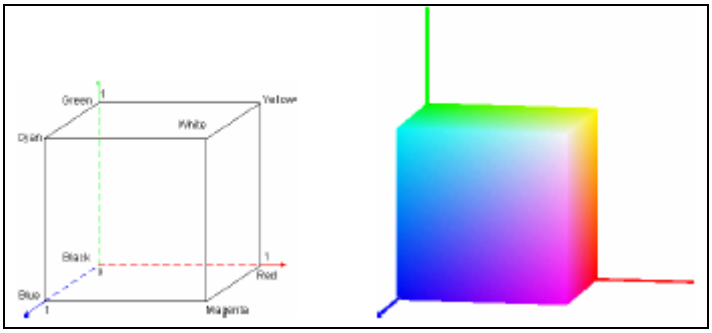

Figura 3.4 – Ilustração do cubo RGB

Com base na figura 3.4 pode-se perceber que a cor branca está localizada na posição (255,255,255).

3.4.2 Luminância e Crominância

O modelo de cor que utiliza luminância e crominância foi criado praticamente na mesma época que a televisão colorida foi inventada, tendo como objetivo a adição de sinais de cor ao sinal de transmissão (WONG, 2007).

A utilização deste modelo permitia que o mesmo sinal fosse utilizado tanto por televisões preto e branco como coloridas. As televisões preto e branco utilizam somente o sinal de luminância (Y), enquanto as coloridas utilizam tanto a luminância como a crominância.

3.4.3 Sistemas de Cor YUV e YIQ

Os sistemas de cor YUV e YIQ dividem a cor em uma componente (Y) de luminância (brilho) e duas componentes (U e V em YUV, ou I e Q em YIQ) de crominância (cor) (WONG, 2007).

O sistema YUV foi criado para utilização em conjunto com o padrão de transmissão PAL, já o sistema YIQ foi o sistema adotado para utilização com o padrão de transmissão NTSC (WONG, 2007).

Ambos os sistemas citados acima podem ter suas componentes convertidas para o sistema RGB através da utilização das fórmulas listadas na tabela 3.1. É importante mencionar que esta conversão é feita automaticamente pelos programas de edição ou visualização de vídeo sempre que necessário (WONG, 2007).

Tabela 3.1 – Conversão dos sistemas YUV e YIQ para RGB (WONG, 2007)

Componente	Fórmula de Conversão RGB
Y	= + 0,299R + 0,587G + 0,114B
U	= - 0,147R – 0,289G + 0,436B
V	= + 0,615R – 0,515G – 0,100B
Y	= + 0,299R + 0,587G + 0,114B
I	= + 0,596R – 0,275G – 0,321B
Q	= + 0,212R – 0,523G + 0,311B

3.4.4 Sistema de Cor YCbCr

O sistema de cor YCbCr consiste em outro modelo que utiliza luminância e crominância e é bastante utilizado para a compressão de vídeo em DVD (WONG, 2007).

Assim como os modelos anteriores, este sistema também pode ter suas componentes convertidas para o sistema RGB. A tabela 3.2 demonstra as fórmulas de conversão para este sistema.

Tabela 3.2 – Conversão do sistema YCbCr para RGB (GUANBARI, 2003)

Componente	Fórmula de Conversão RGB
Y	= + 0,257R + 0,504G + 0,098B + 16
Cb	= - 0,148R – 0,291G + 0,439B + 128
Cr	= + 0,439R – 0,368G – 0,071B + 128

3.5 Amostragem e Quantização

Em um arquivo de vídeo, cada *frame* ou quadro que compõe o vídeo é uma imagem. Cada imagem no vídeo é digitalizada em um processo similar ao utilizado para digitalizar imagens, ou seja, através do processo de amostragem e quantização (WONG, 2007).

3.5.1 Amostragem de Imagem

O processo de amostragem de uma imagem consiste na digitalização dos sinais de cor existentes em uma imagem retirada da natureza. Em uma imagem natural não existe uma divisão exata entre um ponto de cor e outro pois todos se complementam. Para que seja possível a digitalização deste conteúdo é necessário que amostras sejam retiradas desta imagem original, estas amostras serão utilizadas para representar a imagem (WONG, 2007).

A amostragem é feita através da divisão da imagem em diversos pontos igualmente espaçados sendo que de cada um destes pontos uma amostra é retirada. A figura 3.5 demonstra este processo.

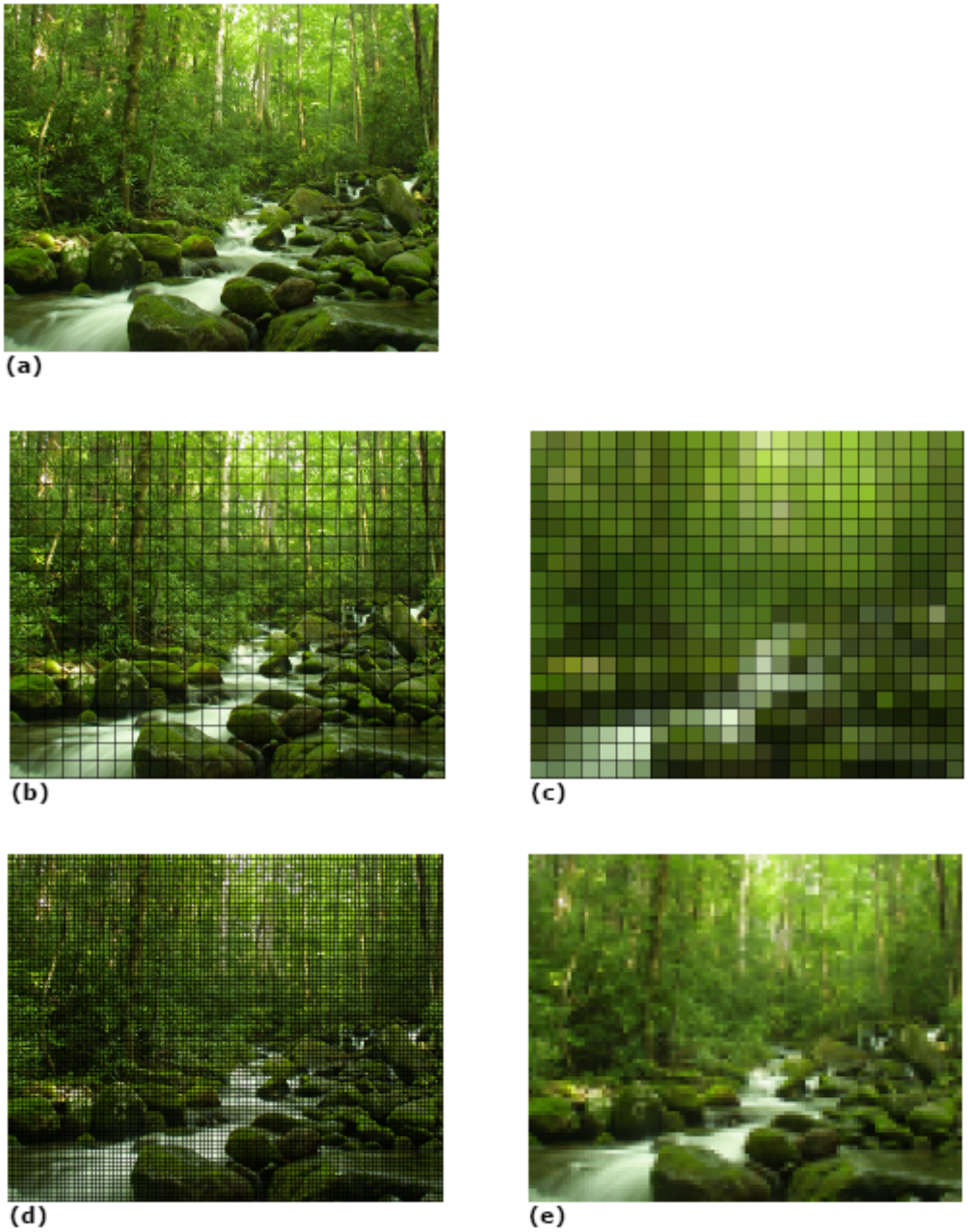

Figura 3.5 – Amostragem de uma imagem

Os itens abaixo descrevem cada imagem da figura 3.5 (WONG, 2007):

(a) Imagem original a ser digitalizada;

(b) Divisão da imagem com base em uma grade de 25 x 20 posições;

(c) Com base em cada posição, uma amostra é retirada e utilizada para gerar uma nova imagem;

(d) Divisão da mesma imagem com base em uma grade de 100 x 80 posições, como forma de aumentar o número de amostras;

(e) Com um número maior de posições, mais amostras podem ser retiradas, reproduzindo de forma mais fiel a imagem original.

3.5.2 Quantização

Uma imagem natural possui infinitos tons de cores, mas para se digitalizar uma imagem como essa é necessário traduzir esses infinitos tons em um número finito de representações de cores. O processo de codificar esse número infinito de tons de cores utilizando um número finito de cores é chamado de quantização (WONG, 2007).

Pode-se ilustrar este processo com base nas imagens da figura 3.6 (WONG, 2007).

Figura 3.6 – Paletas de Cor

A figura 3.6 demonstra duas paletas de cor, a primeira paleta possui 4 tons de verde e a segunda possui 8 tons de verde. Ambas serão utilizadas para quantizar a imagem da figura 2.10a.

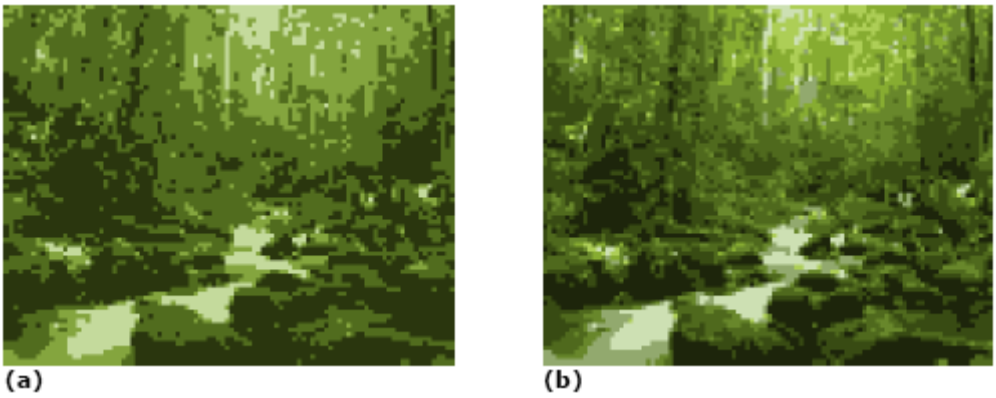

Figura 3.7 – Quantização de uma imagem

A figura 3.7 exibe a mesma imagem quantizada através da utilização da paleta de 4 cores (a) e através da paleta de 8 cores (b). Pode-se perceber que quanto maior o número de cores utilizadas na quantização, melhor será a qualidade da imagem digitalizada quando comparada à original (WONG, 2007).

3.5.3 Amostragem de Movimento

Assim como uma imagem passa pelo processo de amostragem o mesmo ocorre com a dimensão temporal de um vídeo. A taxa de amostragem temporal de um vídeo define o *frame rate* do mesmo, ou seja, quanto maior a taxa de amostragem temporal, maior será o *frame rate*, em *frames* por segundo, do vídeo digitalizado (WONG, 2007).

É importante mencionar que o aumento do *frame rate*, mantendo a duração do filme constante, resulta em um aumento do tamanho final do arquivo de vídeo (WONG, 2007).

3.6 Tamanho de *Frame* e Resolução de Vídeo

Em um vídeo digitalizado o tamanho de cada *frame* ou quadro, ou seja, sua altura e comprimento, é medido em *pixels*, este valor é utilizado para indicar a resolução do vídeo, como por exemplo, 720 x 480 *pixels* (WONG, 2007).

Além da resolução, todo vídeo possui um *aspect ratio* que indica a proporção entre o comprimento do vídeo para a sua altura, este dado é utilizado durante a exibição do vídeo (WONG, 2007). Como exemplo, pode-se citar que o *aspect ratio* para um vídeo NTSC é 4:3 enquanto para um DVD em formato *widescreen* é 16:9.

Embora a grande maioria dos vídeos utilizem *pixels* quadrados, alguns formatos de vídeo podem utilizar *pixels* retangulares. Para permitir a identificação deste *pixels* existe um atributo chamado *pixel aspect ratio*, que, em conjunto com os dados da resolução de um vídeo, permite a obtenção do *aspect ratio* do mesmo (WONG, 2007).

Como exemplo, pode-se ter um vídeo com um *pixel aspect ratio* de 0,9 e resolução de 720 (comprimento) x 480 (altura) *pixels*. Através do cálculo abaixo, é possível obter o *aspect ratio* deste vídeo (WONG, 2007).

Pixel Aspect Ratio : 0,9
comprimento : altura = 720 x 0,9 : 480 = 648 : 480 ≈ 4 : 3 (*aspect ratio*)

Os atributos definidos acima, como o *aspect ratio* e o *pixel aspect ratio* devem ser observados para a correta exibição de um vídeo, caso contrário, o mesmo poderá apresentar deformação da imagem durante a exibição (WONG, 2007).

A figura 3.8 demonstra como a imagem *a* pode ser deformada ao ser exibida sem a utilização do *pixel aspect ratio* correto. Repare nas deformações de altura e comprimento presentes nas imagens *b* e *c.*

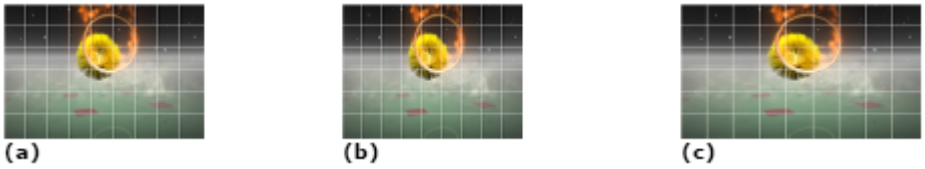

Figura 3.8 – Deformação de um vídeo decorrente *de pixel aspect ratio* incorreto

3.7 Contagem de Tempo

A contagem de tempo é um aspecto importante a ser entendido quando se trabalha com vídeo digital. O tempo em um vídeo digitalizado não é medido apenas em horas, minutos e segundos, mas também em número de quadros ou *frames*, pois durante um segundo podem ser exibidos vários quadros de um vídeo (WONG, 2007).

A SMPTE (*Society of Motion Pictures and Television Engineers)* define como padrão amplamente utilizado a contagem de *frames* com base em horas, minutos, segundos e *frames*, como por exemplo, 00:02:32:07 se refere a um video com duração de 2 minutos, 32 segundos e 7 *frames* (WONG, 2007).

3.8 Conceitos de Compressão

Um arquivo de vídeo digital compactado deve ser descompactado antes que possa ser exibido. A compressão e descompressão são feitas pelos *codecs,* termo composto originário do inglês ***co****mpression/****dec****ompression* (WONG, 2007).

A idéia principal da compressão ou compactação de um arquivo é representar o mesmo conteúdo original utilizando menos dados para atingir este objetivo. As duas principais formas de compressão envolvem o descarte de parte dos dados originais, também chamada de compressão com perdas (*lossy compression*) ou a preservação dos dados originais, alterando somente a forma de codificar os mesmos, também chamada de compressão sem perdas ou *lossless compression* (WONG, 2007).

Geralmente, na maioria dos programas de edição ou conversão de vídeo as opções listadas anteriormente não são exibidas, no seu lugar é exibida uma lista de *codecs* disponíveis, que podem implementar apenas uma ou ambas as técnicas citadas anteriormente (WONG, 2007).

A compressão de vídeo ocorre tanto na dimensão espacial como na temporal (WONG, 2007). Abaixo, cada um destes dois conceitos são explicados.

3.8.1 Compressão Espacial

O objetivo da compressão espacial é a compactação de cada *frame* individualmente, ou seja, o conteúdo de cada *frame* é compactado independente do conteúdo dos demais *frames* (WONG, 2007).

Um exemplo de compressão espacial é o algoritmo RLE, *Run-length encoding*, que consiste na compactação de uma imagem através da substituição de uma seqüência de valores iguais por apenas uma instância do valor seguido pelo número de vezes que o mesmo se repete. Este algoritmo é muito utilizado em compressão de vídeos e imagens em desenhos animados, já que os mesmos possuem grandes áreas nas quais uma mesma cor se repete (WONG, 2007).

3.8.2 Compressão Temporal

Em um vídeo, geralmente as diferenças entre um quadro e outro imediatamente posterior são bem pequenas. Esta característica é explorada pela compressão temporal (WONG, 2007).

Na compressão temporal alguns quadros são escolhidos como chave, neste quadros a informação de todos os *pixels* da imagem é descrita. Já nos quadros intermediários, que estão entre um quadro chave e outro, são descritas apenas as diferenças destas imagens para o quadro chave imediatamente anterior (WONG, 2007).

Esta técnica de compressão funciona bem com vídeos cujo conteúdo contém cenas em movimento contínuo, já com vídeos que possuem mudanças freqüentes de cena a técnica se torna ineficiente (WONG, 2007).

Com base nos conceitos anteriores, pode-se concluir que, caso a diferença entre os quadros posteriores e o quadro chave seja pequena, haverá uma redução no tamanho do arquivo final.

3.8.3 Compressão Com e Sem Perdas

O processo de compactação pode ser feito com ou sem perdas, conforme citado anteriormente.

A compressão sem perdas preserva os dados originais, neste caso, a compressão é feita através da exploração de dados que estão repetidos, os quais são substituídos por representações destas seqüências de valores iguais, reduzindo o tamanho do arquivo (WONG, 2007).

A compressão com perdas descarta ou altera parte dos dados originais, o que pode resultar em perda de qualidade. Embora isso possa ocorrer, os algoritmos de compressão levam em consideração a percepção visual do ser humano ao decidir que dados podem ou não ser descartados de forma que o vídeo, ao ser visualizado, aparente estar com a mesma qualidade do original (WONG, 2007).

Geralmente, a compressão com perdas resulta em um tamanho final de arquivo menor do que o obtido com a compressão sem perdas. É importante mencionar que os dados descartados não podem ser recuperados posteriormente (WONG, 2007).

3.9 Formatos Encapsuladores

O papel do encapsulador, também chamado de *container*, é guardar os dados codificados como a imagem e áudio de um vídeo, em um único arquivo, de forma organizada. Os encapsuladores não definem quais *codecs* serão utilizados, mas apenas guardam os dados codificados com esses *codecs* (WONG, 2007).

É importante mencionar que a utilização de certos *codecs* de vídeo automaticamente resulta na escolha de um determinado formato encapsulador, tendo em vista que alguns formatos encapsuladores possuem suporte a um número limitado de *codecs* de áudio e vídeo.

Neste trabalho serão utilizados dois formatos encapsuladores, AVI e MP4, cada um deles é descritos nos próximos itens.

3.9.1 *Container* AVI

O *container* AVI, que significa *Audio-Video Interleave*¸ ou seja, áudio e vídeo entrelaçado foi criado pela *Microsoft* e é muito utilizado até os dias atuais devido à sua simplicidade e suporte a diversos *codecs*. Os dados a seguir foram retirados da documentação disponível no *website* da MSDN[10] (*Microsoft Developer Network*).

O AVI pode conter tanto vídeo como áudio e o seu formato é baseado no formato RIFF (*Resource Interchange File Format)*. Um arquivo RIFF consiste de um cabeçalho RIFF seguido de nenhuma ou várias listas e blocos, conforme itens abaixo.

- Cabeçalho RIFF: o cabeçalho é descrito da seguinte forma – 'RIFF' fileSize fileType (data)
    - 'RIFF': código FOURCC[11] escrito na forma literal, identificando o formato do arquivo;
    - fileSize : atributo de 4 *bytes* que indica o tamanho dos dados que estão no arquivo. Este atributo se refere ao tamanho do campo fileType e dos dados, não contabilizando os 2 campos iniciais do arquivo;
    - fileType : contém um código FOURCC que identifica o tipo específico do arquivo;
    - (data) : contém os dados, no formato de listas e blocos, em qualquer ordem.
- Bloco: cada bloco em um arquivo RIFF possui o seguinte formato – ckID ckSize ckData
    - ckID : atributo que contém o código FOURCC que identifica os dados que estão contidos no bloco;
    - ckSize : atributo de 4 *bytes* que indica o tamanho dos dados contidos em ckData;
    - ckData : contém zero ou mais *bytes* de dados.
- Lista: cada lista em arquivo RIFF possui o seguinte formato – 'LIST' listSize listType listData

[10] Para mais informações sobre o formato AVI verifique o *website* <http://msdn2.microsoft.com/en-us/library/ms779636.aspx>

[11] Sigla para *four-character code*, é um identificador composto de 4 caracteres que é utilizado em arquivos RIFF para identificar o tipo de arquivo, como o código 'AVI ' e diversos outros dados como blocos de dados, tipos de dados, dentre outros.

- ‘LIST’: código FOURCC escrito na forma literal, identificando que a estrutura é uma lista;
- listSize : atributo de 4 *bytes* que indica o tamanho dos dados contidos em listData incluindo também o campo listType;
- listType : contém um código FOURCC que identifica o tipo da lista;
- listData : contém os dados, que podem ser outras listas ou blocos.

Os arquivos AVI são identificados pelo código FOURCC ‘AVI ‘ no cabeçalho do arquivo RIFF. Todos os arquivos AVI possuem 2 listas que são obrigatórias e que definem o formato dos dados e o conteúdo destes dados, respectivamente. Adicionalmente, o arquivo pode conter uma lista índice, que serve para localizar os blocos de dados dentro do arquivo, conforme demonstrado abaixo.

```
RIFF ('AVI '
      LIST ('hdrl' ... )
      LIST ('movi'... )
      ['idx1' (<AVI Index>) ]
      )
```

A lista *hdrl* define o formato dos dados e é a primeira das listas obrigatórias, seguida pela lista *movi* que contém os dados e é a segunda lista obrigatória. A lista *idx1* é opcional e contém o índice. É importante que esta ordem seja mantida, ou seja, estas listas sempre deverão aparecer nesta seqüência.

As listas *hdr1* e *movi* utilizam blocos para guardar os seus dados, que podem conter mais blocos ou listas, mais informações sobre esta estrutura pode ser obtida no *website* da MSDN citado anteriormente.

Neste trabalho, o *container* AVI é utilizado no arquivo de vídeo original que será transcodificado, sendo essa uma das limitações de escopo deste trabalho. Esta escolha foi feita com base na grande utilização deste *container* tanto em plataformas *Windows* como *Linux* e pelo forte suporte oferecido pela ferramenta de codificação utilizada a este encapsulador.

3.9.2 *Container* MP4

O *container* MP4 é parte integrante da especificação do padrão ISO/IEC MPEG-4, formalmente disponibilizado como a norma ISO/IEC 14496, desenvolvido pelo MPEG, *Moving Pictures Experts Group*, grupo responsável pela elaboração deste padrão, bem como dos padrões MPEG-1 e MPEG-2, bastante premiados e utilizados internacionalmente, a citar como exemplo o padrão MPEG-2 que é utilizado para disponibilização de filmes em DVD pelas produtoras.

O *container* MP4 comumente utiliza a extensão de arquivo .mp4 para identificar os arquivos que o utilizam. Este *container* foi baseado no formato *QuickTime* desenvolvido pela *Apple Computer Inc.* (MPEG-4..., 2002).

A figura 3.9 demonstra a estrutura básica de um arquivo MP4, contendo três fluxos de dados, também chamados de *streams* (MPEG-4..., 2002).

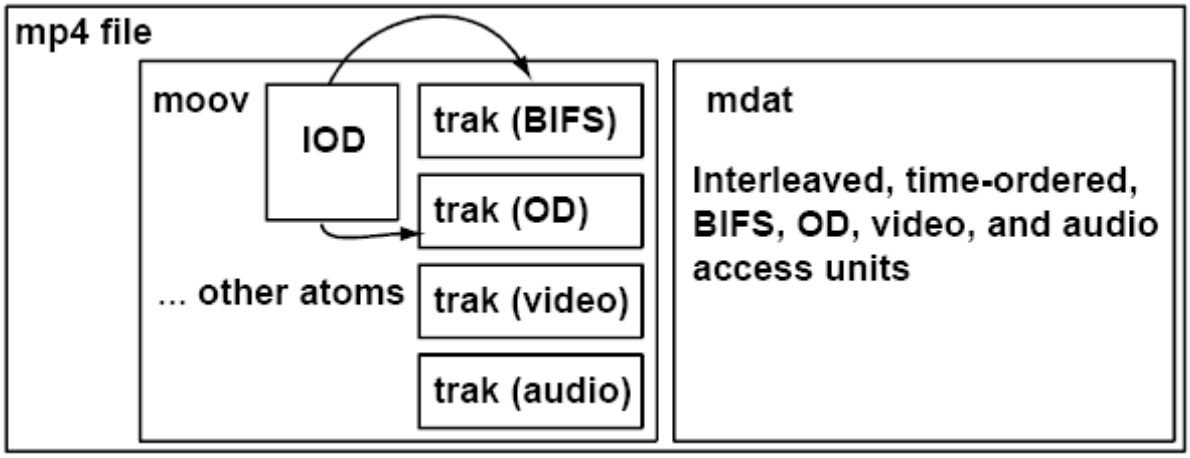

Figura 3.9 – Estrutura básica do *container* MP4

O formato MP4 é composto de estruturas orientadas à objeto chamadas *atoms* ou átomos, que são identificados através de um nome específico e o seu tamanho. A grande maioria dos *atoms* descrevem diversos metadados que são utilizados para fornecer informações como índices, durações de tempo e ponteiros para o local efetivo dos dados multimídia. Todos esses *atoms* estão dentro de um grande átomo chamado *movie atom* (átomo de filme). Os dados multimídia ficam em outros locais, podendo estar tanto dentro do arquivo MP4 em vários *media data atoms* (átomos de dados multimídia) ou até fora do arquivo através de referências utilizando URL (MPEG-4..., 2002).

Este *container* possui diversas vantagens sobre os formatos anteriores, a citar o suporte a uma grande quantidade de tipos de dados como vídeo, áudio e legendas possuindo suporte para os *codecs* de última geração como os baseados no padrão H.264, também chamado de MPEG-4 AVC, e no AAC (*Advanced Audio Codec*), também chamado de MPEG-2 Parte 7 ou MPEG-4 Parte 3. O formato ainda possui suporte a *streaming* de áudio e vídeo via internet com grande eficiência (MPEG-4..., 2002).

Diversos fabricantes já possuem suporte a este formato, como exemplo pode-se citar produtos de peso no mercado mundial como os produtos *iPod* e *iPhone* da *Apple*, os videogames *Sony Playstation 3*, *Sony PSP* e *Microsoft XBOX 360* e inúmeros modelos de celulares.

Neste trabalho o *container* MP4 é utilizado como formato do arquivo final do vídeo convertido. Este *container* foi escolhido pela sua grande aceitação no mercado mundial, além de produzir um arquivo com tamanho final menor quando comparado ao mesmo arquivo encapsulado com o *container* AVI e sua grande integração com o padrão de compressão de vídeo H.264, que será citado posteriormente.

3.10 Padrões de Compressão de Vídeo

Os padrões de compressão de vídeo exploram a redundância espacial e temporal dos vídeos para diminuir o tamanho dos mesmos. A redundância espacial é explorada através da análise de cada *frame* do vídeo para identificar semelhanças entre os *pixels* que compõem o mesmo, desta forma, pode-se economizar espaço representando diversos *pixels* semelhantes como uma seqüência única de *bits* (WONG, 2007).

A redundância temporal é explorada através da comparação entre *frames*, utilizando *frames* chave como base de comparação, com base nesta informação os *codecs* conseguem definir o que mudou de um *frame* para o outro, codificando apenas as mudanças que ocorreram através da utilização de vetores de movimento, reduzindo o tamanho do arquivo final (WONG, 2007).

3.10.1 H.264 (MPEG-4 Parte 10)

O H.264, também conhecido como MPEG-4 Parte 10, é um padrão de compressão de vídeo criado pelas equipes *ITU-T Video Coding Experts Group (VCEG)* e *ISO/IEC Moving*

Picture Experts Group (MPEG), através de uma parceria chamada *Joint Video Team (JVT)*. Os padrões ITU-T H.264[12] e ISO/IEC MPEG-4 Parte 10 (formalmente ISO/IEC 14496-10) são tecnicamente idênticos e foram completados no ano de 2003.

O objetivo do projeto que criou o padrão H.264 era a criação de um padrão capaz de prover um bom nível de qualidade de vídeo utilizando *bitrates* menores do que os padrões anteriores, como o MPEG-4 e MPEG-2, sem que o *design* do padrão se tornasse altamente complexo. Outro objetivo importante era prover flexibilidade ao padrão, de forma que o mesmo pudesse ser utilizado em diversas aplicações em uma gama de redes e sistemas diferentes, incluindo transmissão via satélite, gravação em DVD e vídeo de alta definição (WIEGAND, 2003).

Uma das principais características do H.264 é o fato do mesmo conseguir prover uma ótima qualidade de vídeo utilizando *bitrates* menores do que os padrões anteriores, com reduções variando de um terço à metade da taxa de transferência utilizada pelo padrão MPEG-4 Parte 2. Em termos de economia de largura de banda, isso significa que é possível transmitir mais vídeos em um mesmo canal com qualidade superior utilizando o padrão H.264 quando comparado à qualidade e a taxa de transferência utilizadas pelos padrões anteriores.

A figura 3.10, retirada de um *website*[13], demonstra um teste efetuado utilizando o padrão MPEG-4 Parte 2 e o novo padrão H.264. O vídeo original foi codificado em ambos os padrões utilizando a mesma resolução, neste caso 544 x 368 *pixels*, e *bitrates* de 2 Mbps e 1,2 Mbps. O estudo visava a escolha do padrão que seria utilizado para a transmissão de vídeo em um canal com largura de banda limitada, neste caso, um sistema de satélite.

[12] A especificação técnica está disponível em <http://www.itu.int/rec/T-REC-H.264-200503-I/en/>.
[13] Vide <http://www.balooga.com/mpeg4.php3>. Acesso em 14 de set. 2007.

Figura 3.10 – Diferença de qualidade de imagem entre os padrões MPEG-4 Parte 2 e H.264

O lado direito da imagem foi codificado com o padrão MPEG-4 Parte 2, já o lado esquerdo mostra a continuação da mesma imagem codificada no padrão H.264 ou MPEG-4 Parte 10. Com base na figura 3.10, pode-se perceber, claramente, a superioridade do padrão H.264 quando comparado ao padrão mais utilizado, o MPEG-4 Parte 2, principalmente em cenas em movimento.

As figuras 3.11 e 3.12 também foram retiradas do mesmo *website* e demonstram a qualidade de vídeo obtida quando cada vídeo codificado é comparado ao vídeo original.

Figura 3.11 – Qualidade de imagem obtida com o padrão MPEG-4 Parte 2

Figura 3.12 – Qualidade de imagem obtida com o padrão MPEG-4 Parte 10

Embora as figuras 3.11 e 3.12 possam não deixar nítida a diferença de qualidade quando impressas neste documento, em um monitor de computador a diferença se torna visível, conforme detalhes da figura 3.13.

Figura 3.13 – Detalhe das bordas nos padrões MPEG-4 Parte 2 e H.264

A figura 3.13 foi retirada das imagens anteriores, sendo que a imagem do lado direito corresponde ao vídeo codificado com o padrão MPEG-4 Parte 2 e a do lado esquerdo corresponde ao mesmo vídeo codificado com o padrão H.264. Pode-se verificar que existe uma nitidez e precisão muito maiores nas bordas da perna do jogador no vídeo codificado com o padrão H.264, enquanto no padrão MPEG-4 Parte 2 as bordas não estão nítidas e é clara a grande perda de precisão nos contornos e detalhes.

Os detalhes destes dois padrões, MPEG-4 Parte 2 e H.264 (MPEG-4 Parte 10) estão especificados na norma ISO/IEC 14996, que define o padrão MPEG-4. Neste trabalho não serão abordados os conceitos técnicos específicos dos procedimentos matemáticos de codificação de vídeo nestes padrões.

Para fins de elaboração deste trabalho foram utilizados os dois padrões citados anteriormente. O padrão MPEG-4 Parte 2 é utilizado no vídeo original que será convertido para o padrão H.264, com base nos parâmetros selecionados durante o cadastramento da solicitação de transcodificação.

A escolha acima foi feita principalmente pela grande utilização do padrão MPEG-4 Parte 2, muito utilizado em conjunto com o *container* AVI para compressão e guarda de vídeos. Já o padrão H.264 foi escolhido pela sua modernidade permitindo um alto grau de compressão com uma ótima qualidade de vídeo utilizando *bitrates* menores.

No capítulo 4 são citadas as ferramentas que implementam os padrões acima citados e que foram utilizadas para a realização deste trabalho.

3.11 Padrões de Compressão de Áudio

Os padrões de compressão de áudio também exploram a redundância de dados existentes em um arquivo de áudio. Esta compressão pode ser com ou sem perdas, sendo que em compressões com perdas não é possível obter o sinal original novamente.

Neste trabalho são utilizados dois padrões de compressão de áudio, a citar, o MPEG-1 *Layer* III, conhecido como MP3 e o MPEG-4 Parte 3, também chamado de *Advanced Audio Codec* ou AAC. Ambos são explanados sucintamente nos próximos itens.

3.11.1 MP3 (MPEG-1 Layer III)

O padrão MP3, também conhecimento como MPEG-1 *Layer* III (*ISO/IEC 11172-3 e ISO/IEC 13818-3*), utiliza a compressão com perdas para diminuir drasticamente o tamanho do arquivo de áudio gerado. Isto é atingido através da exploração da capacidade de audição do ser humano, retirando as faixas que não são perceptíveis ao ouvido humano, o restante do áudio é codificado de maneira eficiente, resultando em um arquivo de ótima qualidade à 128 kbps (BRANDENBURG, 1999).

Evoluções no padrão permitiram a utilização de *bitrates* variáveis (*Variable Bit Rate – VBR*), ou seja, de acordo com a complexidade da faixa de áudio os *codecs* atuais podem diminuir ou aumentar dinamicamente o *bitrate* necessário para a codificação do mesmo, contribuindo para aumentar a fidelidade do arquivo final produzido.

Neste trabalho o padrão de compressão MP3 é utilizado nos arquivos originais que serão codificados e também está disponível como um dos formatos de áudio suportados para a codificação do arquivo final.

3.11.2 AAC (*Advanced Audio Codec*)

O padrão de compressão AAC (*Advanced Audio Codec*) foi inicialmente definido na especificação MPEG-2 Parte 7 (*ISO/IEC 13817-7*) e foi incluído, com algumas alterações, na especificação MPEG-4 Parte 3 (*ISO/IEC 14496-3*).

O AAC se mostra mais eficiente em relação ao formato MP3 quando são utilizados *bitrates* menores, tipicamente, abaixo de 192 kbps (BRANDENBURG, 1999). O formato utiliza o *container* MP4, também definido na especificação MPEG-4 (Parte 14) para armazenamento dos dados codificados, em contrapartida ao MP3, que utiliza um *container* próprio.

O novo padrão de codificação é utilizado em diversos equipamentos, tipicamente, os mesmos equipamentos que suportam o padrão H.264 também suportam o padrão AAC, como o *Apple iPod*, um dos primeiros produtos a possuir suporte ao formato.

Neste trabalho, o padrão AAC é disponibilizado como uma das opções de compressão de áudio do arquivo final.

4 DESENVOLVIMENTO

4.1 Descrição do *Hardware*

Nesta seção são descritos os equipamentos utilizados para a realização deste trabalho e o papel desempenhado por cada um para a viabilização da ferramenta.

4.1.1 Servidor

Neste trabalho foram utilizados dois computadores que trabalharam como os servidores da ferramenta. Um dos computadores foi utilizado durante o desenvolvimento da ferramenta e nele foram realizadas as primeiras instalações das bibliotecas e configuração das ferramentas.

A configuração do servidor de desenvolvimento está listada na tabela 4.1:

Tabela 4.1 – Configuração do servidor de desenvolvimento

Componente	Configuração
Processador	AMD Athlon X2 3600+ (2 x 1.9GHz)
Disco Rígido	Western Digital Caviar WD2500KS – 250 GB Partições: 1. 100 GB – Windows XP Professional 2. 42 GB – OpenSUSE 10.2 3. 18 GB – Partição FAT32 de compartilhamento
Memória RAM	1 GB DDR2 533MHz
Outros Recursos	Placa de Rede Gigabit 10/100/1000 Mbps 4 Portas USB Gravador de DVD *Dual Layer*

Após estabilizar o ambiente no servidor de desenvolvimento, as configurações foram refeitas no servidor de produção, para o qual foi utilizado um *notebook*, por viabilizar o deslocamento do servidor sem maiores complicações, sendo muito útil durante os testes efetuados em laboratório do UniCeub. A configuração do mesmo é descrita na tabela 4.2.

Tabela 4.2 – Configuração do servidor de produção

Componente	Configuração
Modelo	Acer Aspire 5610 15.4”
Processador	Intel Core Duo T2300 (2 x 1.6 GHz)
Disco Rígido	120 GB IDE 5400 RPM Partições: 1. 80 GB – Windows XP Professional 2. 20 GB – OpenSUSE 10.2 3. 2 GB – Linux *Swap* 4. 18 GB – Partição FAT32 de compartilhamento
Memória RAM	2 GB DDR2 533MHz
Outros Recursos	Placa de Rede Ethernet 10/100 Mbps 4 Portas USB Gravador de DVD *Dual Layer* Placa de Rede Sem Fio 802.11 a/g

4.1.2 Nó de Processamento

Durante o desenvolvimento, o computador utilizado como servidor de desenvolvimento também foi utilizado como ambiente de desenvolvimento e teste da aplicação utilizada pelos nós de processamento.

A configuração do computador utilizado para o desenvolvimento e testes da aplicação cliente está listada na tabela 4.3.

Tabela 4.3 – Configuração do nó de processamento de desenvolvimento

Componente	Configuração
Processador	AMD Athlon X2 3600+ (2 x 1.9 GHz)
Disco Rígido	Western Digital Caviar WD2500KS – 250 GB Partições: 1. 100 GB – Windows XP Professional 2. 42 GB – OpenSUSE 10.2 3. 18 GB – Partição FAT32 de compartilhamento
Memória RAM	1 GB DDR2 533MHz
Outros Recursos	Placa de Rede Gigabit 10/100/1000 Mbps 4 Portas USB Gravador de DVD *Dual Layer*

Após os testes da aplicação no equipamento de desenvolvimento, foram realizados testes em laboratório do UniCeub, sendo que neste caso foram utilizados cinco computadores como nós de processamento, os quais possuíam a configuração listada na tabela 4.4.

Tabela 4.4 – Configuração do nó de processamento de testes

Componente	Configuração
Processador	Intel Pentium 4 2.8 GHz
Disco Rígido	40 GB – Windows XP Professional
Memória RAM	1 GB DDR
Outros Recursos	Placa de Rede Ethernet 10/100 Mbps 4 Portas USB Leitor de DVD com gravador de CD

4.1.3 Comunicação

4.1.3.1 Ambiente de Desenvolvimento

Durante a etapa de desenvolvimento, a comunicação entre os computadores utilizados como servidor e nó de processamento foi efetuada através da utilização de um roteador. As conexões entre os equipamentos e o roteador foram feitas com cabos de rede UTP Categoria 5 com conectores RJ-45, o que possibilita uma velocidade de até 100 Mbps.

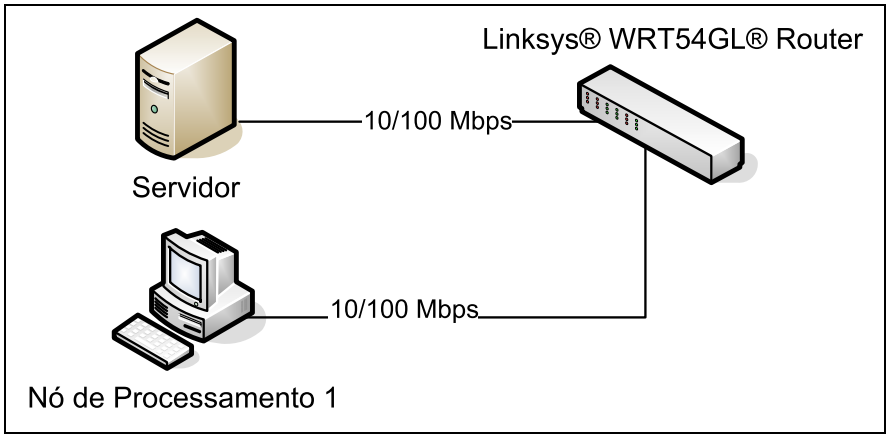

Figura 4.1 – Ambiente de Desenvolvimento - Rede de Comunicação

O roteador utilizado durante a etapa de desenvolvimento foi o modelo WRT54GL do fabricante *Linksys*, exibido na figura 4.2, que possui 4 portas LAN 10/100 Mbps e suporte para conexões de redes sem fio.

Figura 4.2 – Roteador *Linksys* WRT54GL

4.1.3.2 Ambiente de Testes

O laboratório 7002 do UniCeub foi utilizado como ambiente de testes da ferramenta. Neste laboratório a comunicação entre os computadores é feita através da utilização de *hub* modelo SuperStack[14] II PS Hub 40 24 portas, fabricante *3Com,* exibido na figura 4.3. As conexões entre os equipamentos e o roteador foram feitas com cabos de rede UTP Categoria 5 com conectores RJ-45, o que possibilitou uma velocidade de até 10 Mbps utilizando topologia semelhante a exposta na figura 4.4.

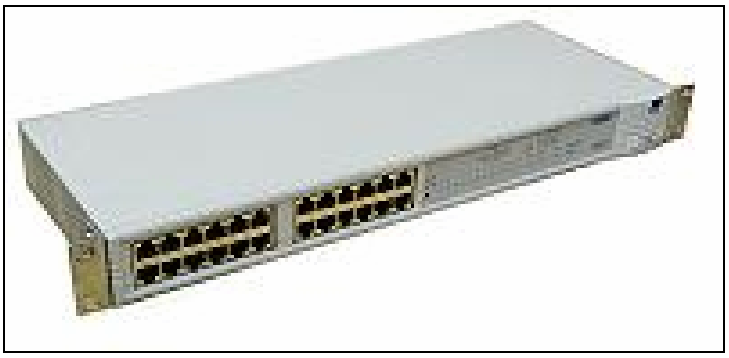

Figura 4.3 – *Hub 3Com* SuperStack II PS 40

[14] Este *hub* teve a sua venda descontinuada em dezembro de 2002. Vide <http://www.3com.com/products/en_US/end_of_life.jsp?sku=3C16406-US>

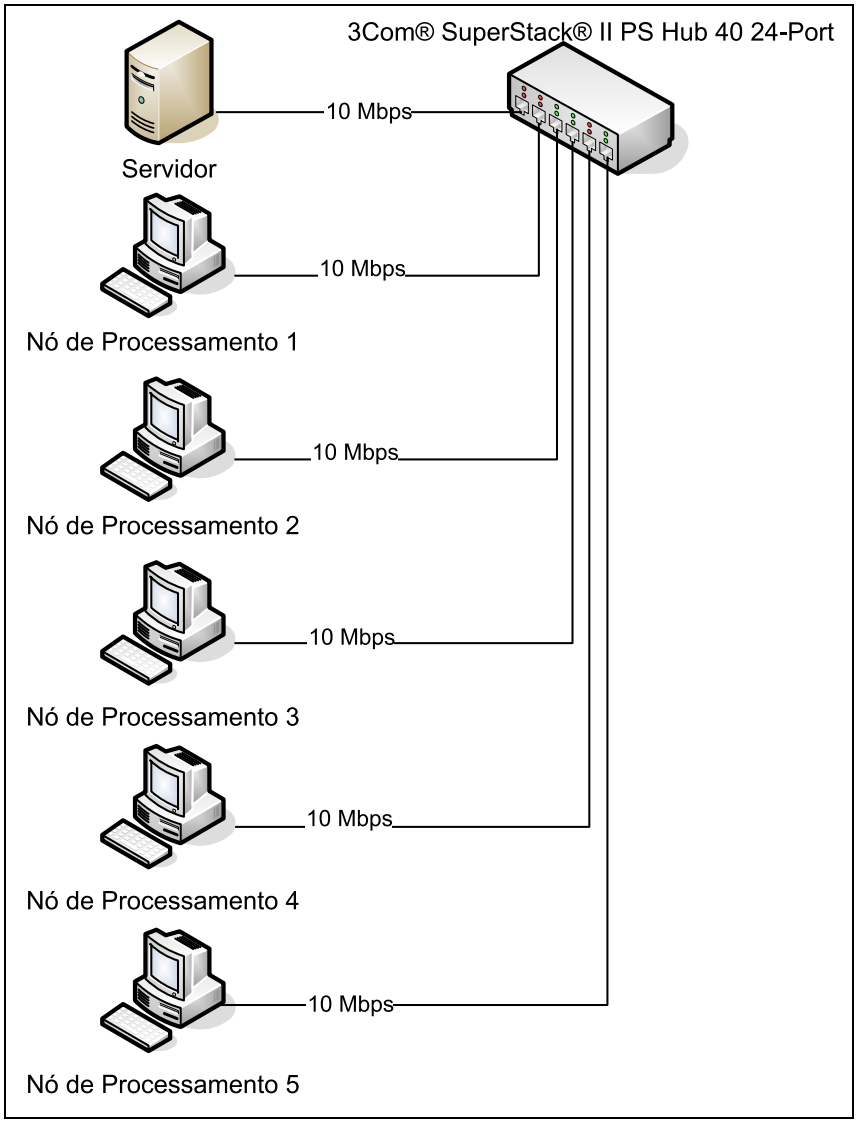

Figura 4.4 – Ambiente de Testes – Rede de Comunicação

4.2 Descrição do *Software*

Esta seção descreve a construção da ferramenta elaborada para viabilizar a transcodificação distribuída de vídeo. São citadas as ferramentas e bibliotecas utilizadas bem como os modelos de dados e comunicação utilizados.

4.2.1 Visão Geral da Ferramenta

A ferramenta construída é constituída de duas partes complementares: o servidor e os nós de processamento. O desenvolvimento contemplou a elaboração das duas partes e a integração das mesmas.

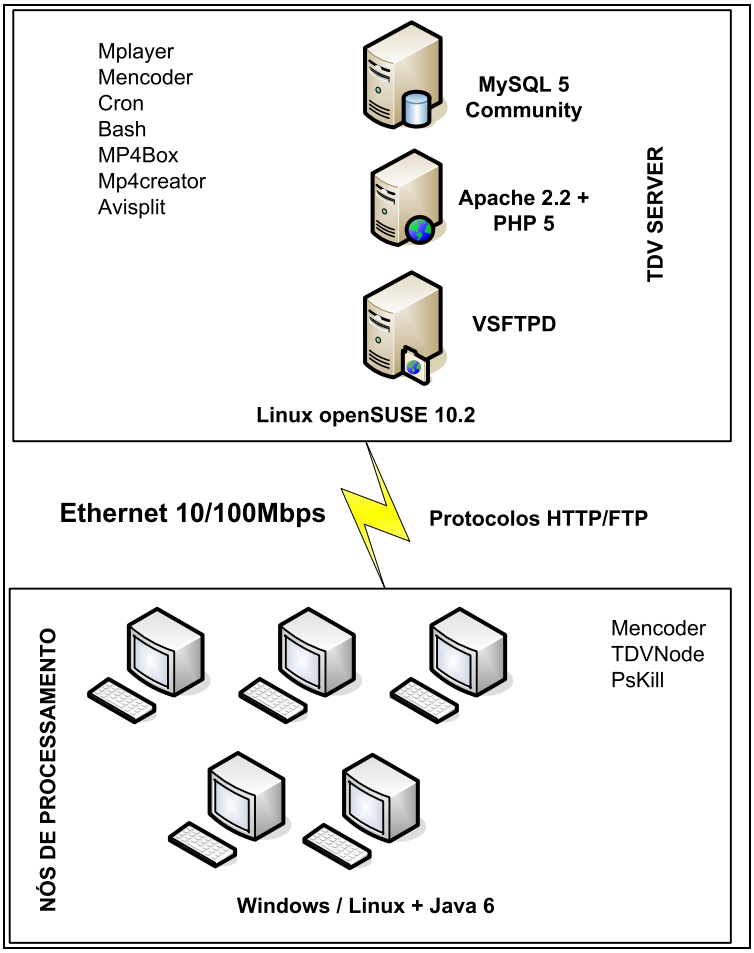

Figura 4.5 – Visão geral da ferramenta

O fluxo geral do processo de transcodificação pode ser descrito resumidamente pela figura 4.6.

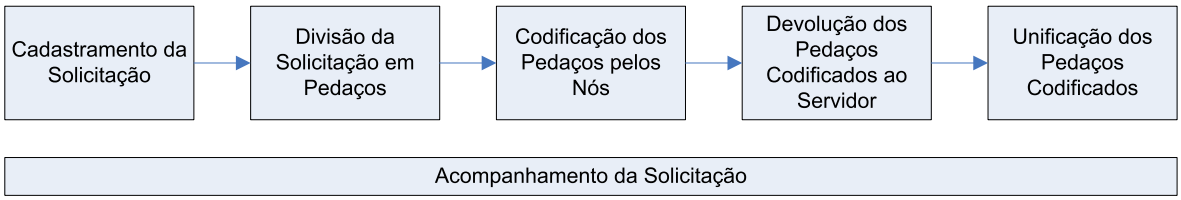

Figura 4.6 – Fluxo do processo de transcodificação distribuída

As diversas tarefas executadas pelos nós de processamento e pelo servidor podem ser descrita pelo diagrama de casos de uso exibido na figura 4.7. Todos os casos de uso listados no diagrama são suportados diretamente pelo servidor. A ferramenta utiliza a figura do administrador como o responsável pelo cadastramento de novas solicitações, cadastramento de nós de processamento e pelo acompanhamento das solicitações.

Os nós de processamento são responsáveis por se autenticar junto ao servidor, solicitar novos pedaços para transcodificação, receber os pedaços alocados do servidor, informar ao mesmo o andamento deste recebimento, efetuar a transcodificação dos pedaços de acordo com os parâmetros repassados pelo servidor, informar ao servidor o andamento da transcodificação, enviar o pedaço transcodificado ao servidor, informando ao mesmo o andamento deste envio e, finalmente, solicitar a sua desconexão do servidor.

As tarefas realizadas pelos nós de processamento necessitam de *scripts* para o recebimento de dados e controle de fluxo no servidor, sendo assim, para cada tarefa executada pelos nós de processamento existe, de forma complementar, uma tarefa no servidor responsável por receber os dados provenientes das primeiras, interpretar e devolver o resultado ao nós de processamento.

Além das funcionalidades listadas anteriormente, o servidor também é responsável pela divisão de novos arquivos, unificação de pedaços transcodificados e a desalocação de pedaços que estão sem atualização há algum tempo. Estas tarefas são agendadas e executadas periodicamente pelo agendador de tarefas do sistema, que varia de nome e composição de acordo com o sistema operacional utilizado.

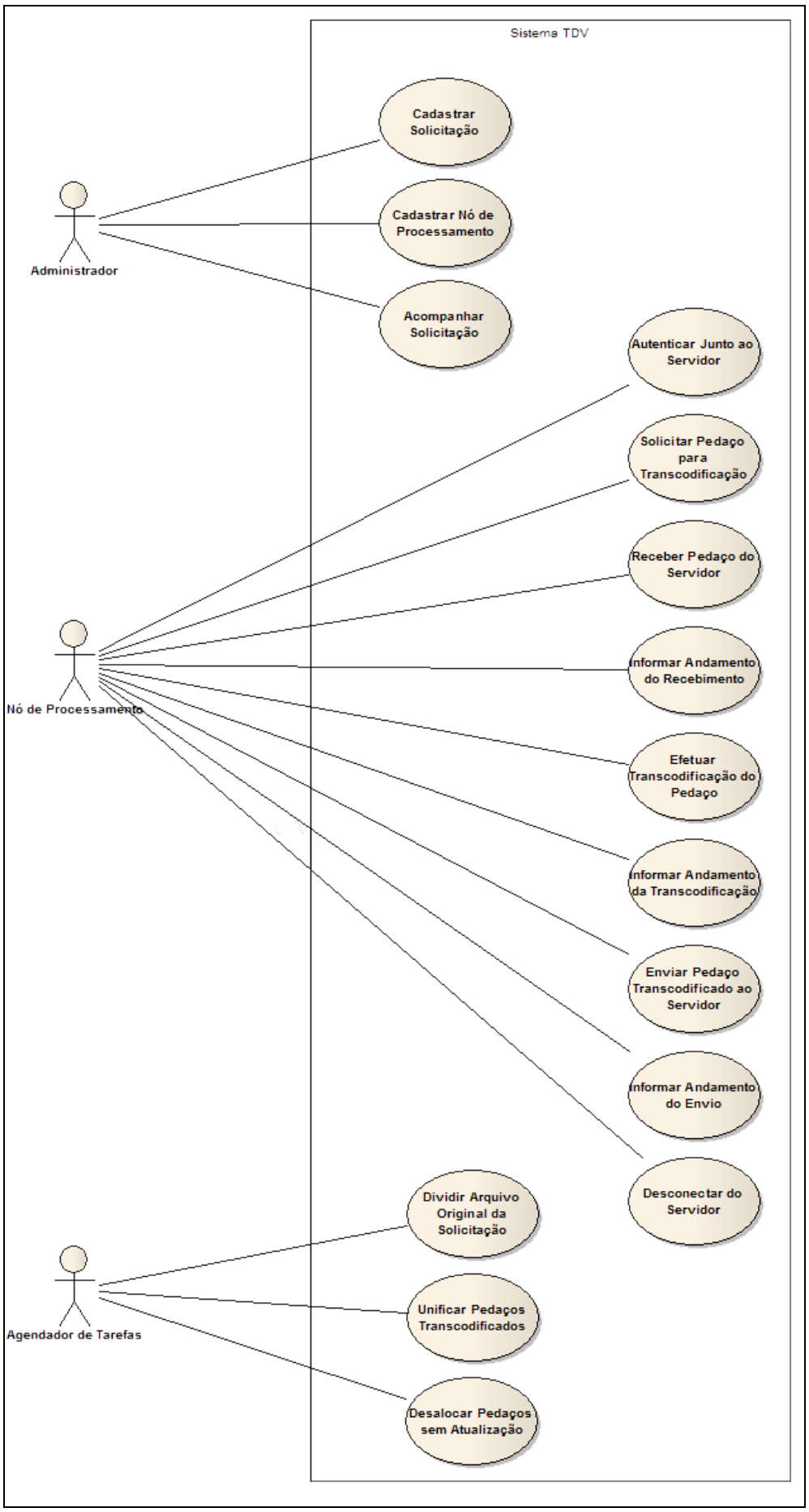

Figura 4.7 – Diagrama de casos de uso

4.2.2 Modelo de Dados

A guarda dos dados utilizados pela ferramenta exigiu a elaboração de um modelo de dados para contemplar todas as necessidades da aplicação. A figura 4.8 exibe o modelo de dados utilizado.

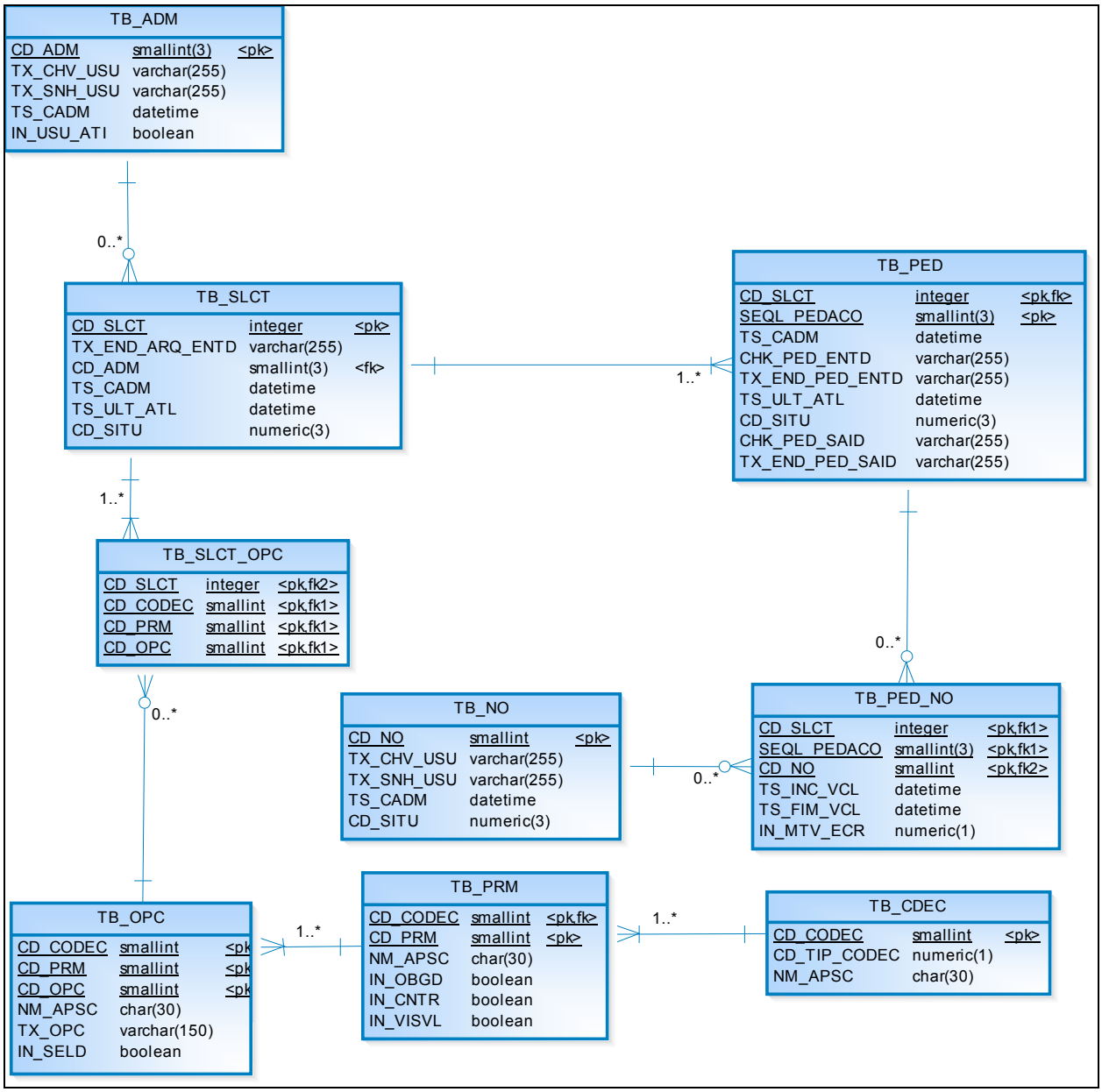

Figura 4.8 – Modelo de dados da ferramenta

Nas próximas páginas, cada tabela é descrita sucintamente, bem como os seus respectivos campos.

Tabela 4.5 – Descrição dos campos da tabela TB_ADM

Tabela	**TB_ADM**			
Descrição	Tabela responsável pela guarda dos dados dos administradores do sistema.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_ADM	Identificador do administrador	X		X
TX_CHV_USU	Chave do usuário administrador			X
TX_SNH_USU	Senha do usuário administrador			X
TS_CADM	Data e hora do cadastramento do usuário administrador			X
IN_USU_ATI	Indicador de atividade do usuário administrador			X

Tabela 4.6 – Descrição dos campos da tabela TB_SLCT

Tabela	**TB_SLCT**			
Descrição	Tabela responsável pela guarda dos dados das solicitações de transcodificação cadastradas.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_SLCT	Identificador da solicitação	X		X
TX_END_ARQ_ENTD	Endereço do arquivo de entrada no servidor			X
CD_ADM	Identificador do administrador responsável pelo cadastramento		X	X
TS_CADM	Data e hora do cadastramento da solicitação			X
TS_ULT_ATL	Data e hora da última atualização da situação da solicitação			
CD_SITU	Código da situação da solicitação			X

Tabela 4.7 – Descrição dos campos da tabela TB_PED

Tabela	**TB_PED**			
Descrição	Tabela responsável pela guarda dos dados dos pedaços vinculados à uma solicitação.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_SLCT	Identificador da solicitação	X	X	X
SEQL_PEDACO	Seqüencial do pedaço	X		X
TS_CADM	Data e hora do cadastramento do pedaço			X
CHK_PED_ENTD	Soma de verificação do arquivo original do pedaço			X
TX_END_PED_ENTD	Endereço do arquivo original do pedaço no servidor			X
TS_ULT_ATL	Data e hora da última atualização da situação do pedaço			
CD_SITU	Código da situação do pedaço			X
CHK_PED_SAID	Soma de verificação do arquivo transcodificado do pedaço			
TX_END_PED_SAID	Endereço do arquivo transcodificado do pedaço no servidor			

Tabela 4.8 – Descrição dos campos da tabela TB_NO

Tabela	**TB_NO**			
Descrição	Tabela responsável pela guarda dos dados dos nós de processamento.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_NO	Identificador do nó de processamento	X		X
TX_CHV_USU	Chave do usuário do nó de processamento			X

TX_SNH_USU	Senha do usuário do nó de processamento			X
TS_CADM	Data e hora do cadastramento do nó de processamento			X
CD_SITU	Código da situação do nó de processamento			X

Tabela 4.9 – Descrição dos campos da tabela TB_CDEC

Tabela	**TB_CDEC**			
Descrição	Tabela responsável pela guarda dos dados dos *codecs* disponíveis para utilização.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_CODEC	Identificador do *codec*	X		X
CD_TIP_CODEC	Código do tipo do c*odec*: 1 – *Codec* de vídeo 2 – *Codec* de áudio 3 – Parâmetros Gerais			X
NM_APSC	Nome de apresentação do *codec*			X

Tabela 4.10 – Descrição dos campos da tabela TB_PRM

Tabela	**TB_PRM**			
Descrição	Tabela responsável pela guarda dos dados dos parâmetros dos *codecs* cadastrados.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_CODEC	Identificador do c*odec*	X	X	X
CD_PRM	Identificador do parâmetro	X		X
NM_APSC	Nome de apresentação do parâmetro			X
IN_OBGD	Indicador de obrigatoriedade do parâmetro			X
IN_CNTR	Indicador se o parâmetro se refere ao *container* utilizado			X

	pelo *codec*			
IN_VISVL	Indicador de visibilidade do parâmetro			X

Tabela 4.11 – Descrição dos campos da tabela TB_OPC

Tabela	**TB_OPC**			
Descrição	Tabela responsável pela guarda dos dados das opções dos parâmetros vinculados a um *codec*.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_CODEC	Identificador do c*odec*	X	X	X
CD_PRM	Identificador do parâmetro	X	X	X
CD_OPC	Identificador da opção	X	X	X
NM_APSC	Nome de apresentação da opção			X
TX_OPC	Texto dos comandos da opção			X
IN_SELD	Indicador se a opção é selecionada automaticamente			X

Tabela 4.12 – Descrição dos campos da tabela TB_PED_NO

Tabela	**TB_PED_NO**			
Descrição	Tabela responsável pela guarda de dados da vinculação de pedaços a nós de processamento.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_SLCT	Identificador da solicitação	X	X	X
SEQL_PEDACO	Identificador do pedaço	X	X	X
CD_NO	Identificador do nó de processamento	X	X	X
TS_INC_VCL	Data e hora do início do vínculo entre o nó de processamento e o pedaço			X
TS_FIM_VCL	Data e hora do fim do vínculo entre o nó de processamento			

	e o pedaço			
IN_MTV_ECR	Indicador do motivo de encerramento do vínculo: 1 – Término normal 2 – Término devido a erro			

Tabela 4.13 – Descrição dos campos da tabela TB_SLCT_OPC

Tabela	**TB_SLCT_OPC**			
Descrição	Tabela responsável pela guarda dos dados da vinculação da solicitação e as suas opções de transcodificação.			
Nome do Campo	**Descrição**	**Chave Primária**	**Chave Estrangeira**	**Obrigatório**
CD_SLCT	Identificador da solicitação	X	X	X
CD_CODEC	Identificador do *Codec*	X	X	X
CD_PRM	Identificador do Parâmetro	X	X	X
CD_OPC	Identificador da Opção	X	X	X

4.2.3 Modelo de Comunicação

A comunicação entre os nós de processamento e o servidor é feita através de troca de mensagens utilizando o protocolo HTTP. O envio de arquivos ao servidor, por parte dos nós de processamento, é feito através do protocolo FTP utilizando o servidor de arquivos presente no equipamento servidor.

O modelo de comunicação foi baseado em código de envio e retorno, cada transação possui um código específico e todos os parâmetros recebidos e devolvidos possuem nomes ou códigos específicos. As tabelas a seguir descrevem todos os códigos utilizados para cada transação.

Tabela 4.14 – Parâmetros da transação “Autentica Administrador”

Transação	Autentica Administrador			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		1	Código da transação
ENT-COD-USUR	X			Chave do usuário
ENT-COD-SENHA	X			Senha do usuário

SAI-COD-RETOR		X	201	Autenticação do administrador efetuada
		X	601	Parâmetros inválidos ou incompletos
		X	901	Erro no servidor
SAI-MSG-INFOR		X		Mensagem de erro ou informativa
SAI-VAR-SESSAO		X		Variável de sessão

Tabela 4.15 – Parâmetros da transação “Autentica Nó de Processamento”

Transação	Autentica Nó de Processamento			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		3	Código da transação
ENT-COD-USUR	X			Chave do usuário
ENT-COD-SENHA	X			Senha do usuário
SAI-COD-RETOR		X	203	Autenticação do nó de processamento efetuada
		X	601	Parâmetros inválidos ou incompletos
		X	901	Erro no servidor
SAI-MSG-INFOR		X		Mensagem de erro ou informativa
SAI-VAR-SESSAO		X		Variável de sessão

Tabela 4.16 – Parâmetros da transação “Solicita Pedaço para Transcodificação”

Transação	Solicita Pedaço para Transcodificação			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		4	Código da transação
PHPSESSID	X			Variável de sessão
SAI-COD-RETOR		X	204	Pedaço alocado
		X	602	Erro de autenticação – variável de sessão inválida ou expirada
		X	603	Erro de alocação – não existem pedaços disponíveis
		X	901	Erro no servidor
SAI-MSG-INFOR		X		Mensagem de erro ou informativa
SAI-NOM-PEDACO		X		Nome do pedaço destinado

SAI-CHECKSUM-PEDACO		X		*Checksum* do arquivo do pedaço
SAI-PRM-CODIFICACAO		X		Parâmetros de transcodificação do pedaço
SAI-PRM-CONTAINER		X		Formato do *container* de destino do arquivo transcodificado

Tabela 4.17 – Parâmetros da transação “Efetua *Download* de Arquivo”

Transação	Efetua *Download* de Arquivo			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		5	Código da transação
PHPSESSID	X			Variável de sessão
SAI-ARQUIVO		X		Arquivo binário
SAI-COD-RETOR		X	205	Arquivo encontrado, proceder com *download*
		X	601	Parâmetros inválidos ou incompletos
		X	602	Erro de autenticação – variável de sessão inválida ou expirada
		X	605	Não existe pedaço alocado para o nó de processamento
		X	901	Erro no servidor

Tabela 4.18 – Parâmetros da transação “Informa Andamento de *Download*”

Transação	Informa Andamento de *Download*			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		6	Código da transação
PHPSESSID	X			Variável de sessão
ENT-COD-INFOR	X		1	*Download* e verificação de *checksum* do arquivo efetuados com sucesso
			901	Erro de *download*
			902	Erro de verificação de *checksum*
SAI-COD-RETOR		X	206	Informação de andamento de *download* recebida

		X	207	Informação recebida – abortar o procedimento
			601	Parâmetros inválidos ou incompletos
		X	602	Erro de autenticação – variável de sessão inválida ou expirada
		X	605	Não existe pedaço alocado para o nó de processamento
		X	901	Erro no servidor

Tabela 4.19 – Parâmetros da transação “Informa Andamento de Transcodificação”

Transação	Informa Andamento de Transcodificação			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		7	Código da transação
PHPSESSID	X			Variável de sessão
ENT-COD-INFOR	X		1	Transcodificação em andamento
			2	Transcodificação concluída
			901	Erro de transcodificação
SAI-COD-RETOR		X	208	Informação de andamento de transcodificação recebida
		X	207	Informação recebida – abortar o procedimento
			601	Parâmetros inválidos ou incompletos
		X	602	Erro de autenticação – variável de sessão inválida ou expirada
		X	605	Não existe pedaço alocado para o nó de processamento
		X	901	Erro no servidor

Tabela 4.20 – Parâmetros da transação “Informa Andamento de *Upload*”

Transação	Informa Andamento de *Upload*			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		8	Código da transação
PHPSESSID	X			Variável de sessão

ENT-COD-INFOR	X		1	*Upload* em andamento
			2	*Upload* concluído
			901	Erro de *upload*
ENT-NOM-PED-CODI	X			Nome do pedaço transcodificado
ENT-CHECKSUM	X			*Checksum* do arquivo transcodificado
SAI-COD-RETOR		X	209	Informação de andamento de *upload* recebida
		X	207	Informação recebida – abortar o procedimento
			601	Parâmetros inválidos ou incompletos
		X	602	Erro de autenticação – variável de sessão inválida ou expirada
		X	604	Erro de verificação de *checksum*
		X	605	Não existe pedaço alocado para o nó de processamento
		X	901	Erro no servidor

Tabela 4.21 – Parâmetros da transação “Desconecta Nó de Processamento”

Transação	Desconecta Nó de Processamento			
Nome do Parâmetro	**Entrada**	**Saída**	**Valor**	**Descrição**
ENT-COD-TRANS	X		9	Código da transação
PHPSESSID	X			Variável de sessão
SAI-COD-RETOR		X	210	Informação recebida, nó de processamento desconectado
			601	Parâmetros inválidos ou incompletos
		X	602	Erro de autenticação – variável de sessão inválida ou expirada
		X	901	Erro no servidor

Em conjunto com as transações descritas anteriormente são utilizados códigos para identificar a situação dos nós de processamento, das solicitações e dos pedaços vinculados às mesmas. As tabelas 4.22, 4.23 e 4.24 descrevem os códigos utilizados.

Tabela 4.22 – Códigos das situações dos nós de processamento

Código da Situação	Descrição da Situação
1	Nó de processamento desconectado
2	Nó de processamento conectado
3	Nó de processamento alocado

Tabela 4.23 – Códigos das situações das solicitações

Código da Situação	Descrição da Situação
1	Solicitação cadastrada, aguardando divisão
2	Solicitação em divisão
3	Solicitação dividida, aguardando codificação
4	Solicitação em codificação
5	Solicitação codificada, aguardando unificação
6	Solicitação em unificação
7	Solicitação concluída
902	Solicitação com erro de divisão
903	Solicitação com erro de unificação

Tabela 4.24 – Códigos das situações dos pedaços

Código da Situação	Descrição da Situação
1	Pedaço cadastrado, aguardando alocação
2	Pedaço alocado
3	Pedaço com *download* solicitado
4	Pedaço com *download* em andamento
5	Pedaço aguardando codificação
6	Pedaço em codificação
7	Pedaço aguardando *upload*
8	Pedaço com *upload* solicitado
9	Pedaço com *upload* em andamento
10	Pedaço aguardando unificação
11	Pedaço com unificação em andamento
12	Pedaço unificado
901	Pedaço com erro de *download*
902	Pedaço com erro de verificação de *checksum* do arquivo original

903	Pedaço com erro de codificação
904	Pedaço com erro de *upload*
905	Pedaço com erro de verificação de *checksum* do arquivo codificado
906	Pedaço com erro de unificação

4.2.4 Descrição do Servidor

Para viabilizar a construção da ferramenta foi necessário a utilização de um servidor, responsável por permitir o cadastramento de solicitações, armazenar os dados da aplicação, dividir os arquivos originais das solicitações, receber e responder às solicitações dos nós de processamento, receber os arquivos transcodificados dos nós de processamento, efetuar a unificação dos arquivos transcodificados e desalocar os pedaços há algum tempo sem atualização.

Com base nos requisitos acima, o servidor englobou a utilização de um servidor *web*, um servidor de arquivos, também chamado servidor FTP, um sistema gerenciador de banco de dados e diversos *scripts* de apoio. Os próximos itens detalham os principais componentes do servidor.

4.2.4.1 Sistema Operacional

O sistema operacional utilizado no servidor foi o *OpenSUSE*[15] 10.2, versão de código aberto da famosa distribuição *SUSE Linux*, desenvolvida pela *Novell*. Este sistema foi escolhido devido à familiaridade do autor deste trabalho com o mesmo, além de possuir uma grande comunidade de usuários e desenvolvedores, possibilitando a obtenção de respostas rápidas para os problemas encontrados.

A escolha do sistema operacional citado proporciona grande flexibilidade e concede aos servidores instalados no mesmo grande estabilidade e velocidade, características típicas de sistemas baseados em *Unix*.

4.2.4.2 Estrutura

Conforme descrito anteriormente, as funcionalidades esperadas do servidor exigiram a utilização de diversas ferramentas adicionais. Cada uma das principais ferramentas utilizadas é

[15] Para maiores informações e obtenção deste sistema operacional visite <http://www.opensuse.org>

descrita nos próximos itens, em conjunto com a estrutura de diretórios da mesma, caso aplicável.

4.2.4.3 Servidor *Web*

Neste trabalho foi utilizado o servidor *web* Apache HTTPD 2.2.6[16], o qual foi escolhido pela sua grande utilização, estabilidade e segurança. A instalação e utilização do mesmo em distribuições *Linux* é fácil e grande parte das mesmas já possuem pacotes específicos que podem ser instalados durante a configuração do sistema operacional.

A utilização deste servidor permitiu a agregação da tecnologia PHP[17], que possibilita a criação de páginas HTML que são geradas dinamicamente a partir de *scripts* PHP. A linguagem PHP permite também a integração com diversos bancos de dados além de vários recursos interessantes como o acesso a uma grande quantidade de funções existentes, leitura e gravação de arquivos, geração de imagens, dentre outros. Agregado a estas funcionalidades tem-se a velocidade de execução da mesma, o que permite o atendimento de diversos nós de processamento e minimiza o atraso.

Ambas as ferramentas citadas acima possuem código fonte aberto, permitindo alterações e novas implementações. A integração da linguagem PHP com o servidor *web* Apache é feita através da edição do arquivo de configuração do servidor o qual está incluso no apêndice B deste trabalho.

O diagrama exibido na figura 4.9 demonstra a organização da estrutura de diretórios do servidor *web* tendo como referência o começo da estrutura de diretórios do sistema operacional. A tabela 4.25 descreve, sucintamente, o objetivo de cada arquivo citado.

Tabela 4.25 – Descrição dos principais arquivos da árvore do servidor *web*

Nome do Arquivo	Objetivo
httpd.conf	Arquivo de configuração do servidor Apache. Vide apêndice B.
php.ini	Arquivo de configuração da linguagem PHP. Vide apêndice B.
tdvDBConf.php	Arquivo que contém os dados de conexão ao banco de dados, localizado fora da árvore *htdocs* para evitar problemas de segurança. Vide apêndice E.

[16] Para maiores informações e obtenção do servidor visite <http://httpd.apache.org>
[17] Para maiores informações sobre a tecnologia PHP visite <http://www.php.net>

index.php	Página de entrada da ferramenta, exibe o *login* do administrador. Vide figura A.1 do apêndice A e apêndice E.
configuracao.php	Arquivo de configuração da ferramenta. Vide apêndice E.
controlador.php	Página acionada pelos nós de processamento para comunicação com o servidor. Vide apêndice E.
/admin/index.php	Página principal do administrador, exibe o menu de opções e aciona diversas outras páginas. Vide figura A.2 do apêndice A e apêndice E.
/admin/menu.php	Página que exibe o menu de opções do administrador. Vide apêndice E.
/admin/logout.php	Página que desconecta o administrador da interface de gerenciamento do servidor. Vide apêndice E.
/admin/acompanhamento/ exibelistagem.php	Página que aciona periodicamente a página pedaco.php para a exibição do relatório de acompanhamento de solicitação por pedaços. Vide figura A.7 do apêndice A e apêndice E.
/admin/acompanhamento/ pedaco.php	Página que monta o relatório de acompanhamento de uma solicitação com detalhamento de cada pedaço vinculado a mesma. Vide figura A.7 do apêndice A e apêndice E.
/admin/acompanhamento/ solicitacao.php	Página que exibe as solicitações cadastradas e a situação das mesmas, possibilitando o detalhamento através da página pedaco.php. Vide figura A.6 do apêndice A e apêndice E.
/admin/cadastro/no.php	Página responsável por coordenar o cadastramento de nós de processamento. Aciona função do arquivo funcoesApresentacao.php, controlando apenas o retorno recebido. Vide figura A.5 do apêndice A e apêndice E.
/admin/cadastro/solicitacao.php	Página responsável pelo início do cadastramento de uma nova solicitação. Exibe a listagem de *codecs* e arquivos de entrada disponíveis. Vide figura A.3 do apêndice A e apêndice E.
/admin/cadastro/solicitacao2.php	Página responsável pela segunda etapa do cadastramento de uma nova solicitação. Exibe os

	parâmetros e opções disponíveis, de acordo com os *codecs* selecionados na etapa anterior. Vide figura A.4 do apêndice A e apêndice E.
/admin/cadastro/solicitacao3.php	Página responsável pela terceira etapa do cadastramento de uma nova solicitação. Aciona funções responsáveis pelo cadastramento em banco de dados e exibe o retorno obtido. Vide apêndice E.
/estilos/padrao.css	Folha de estilos, utilizada para exibição e definição de cores, bordas e estilos dos botões e textos do *website*. Vide apêndice E.
/funcoes/ funcoesApresentacao.php	Arquivo de apoio contendo as funções utilizadas pela interface de gerenciamento *web* como, por exemplo, as funções de cadastramento de solicitações e nós de processamento. Vide apêndice E.
/funcoes/ funcoesControlador.php	Arquivo contendo as funções utilizadas pelo controlador para viabilizar as tarefas realizadas pelo mesmo em conjunto com os nós de processamento. Vide apêndice E.
/funcoes/funcoesApoio.php	Arquivo contendo funções genéricas utilizadas pelos outros arquivos. Vide apêndice E.
/funcoes/sha256.js	*Script*[18] criado com a tecnologia *JavaScript* que implementa o algoritmo de codificação SHA-256, utilizado para a codificação do usuário e senha do administrador durante o *login*. Vide apêndice E.

Para a geração do relatório exibido pela página exibelistagem.php foi utilizada, em conjunto com a tecnologia PHP, a tecnologia AJAX (*Asynchronous Javascript and XML)*, para permitir a atualização do conteúdo da página sem necessitar que o usuário, manualmente, solicite o recarregamento da mesma. O intervalo de atualização do conteúdo pode ser configurado através do arquivo configuracao.php.

O código fonte dos *scripts* e páginas listadas anteriormente está disponível no apêndice E deste projeto.

[18] *Script* criado por Christoph Bichlmeier, para mais informações visite <http://www.bichlmeier.info/sha256.html>

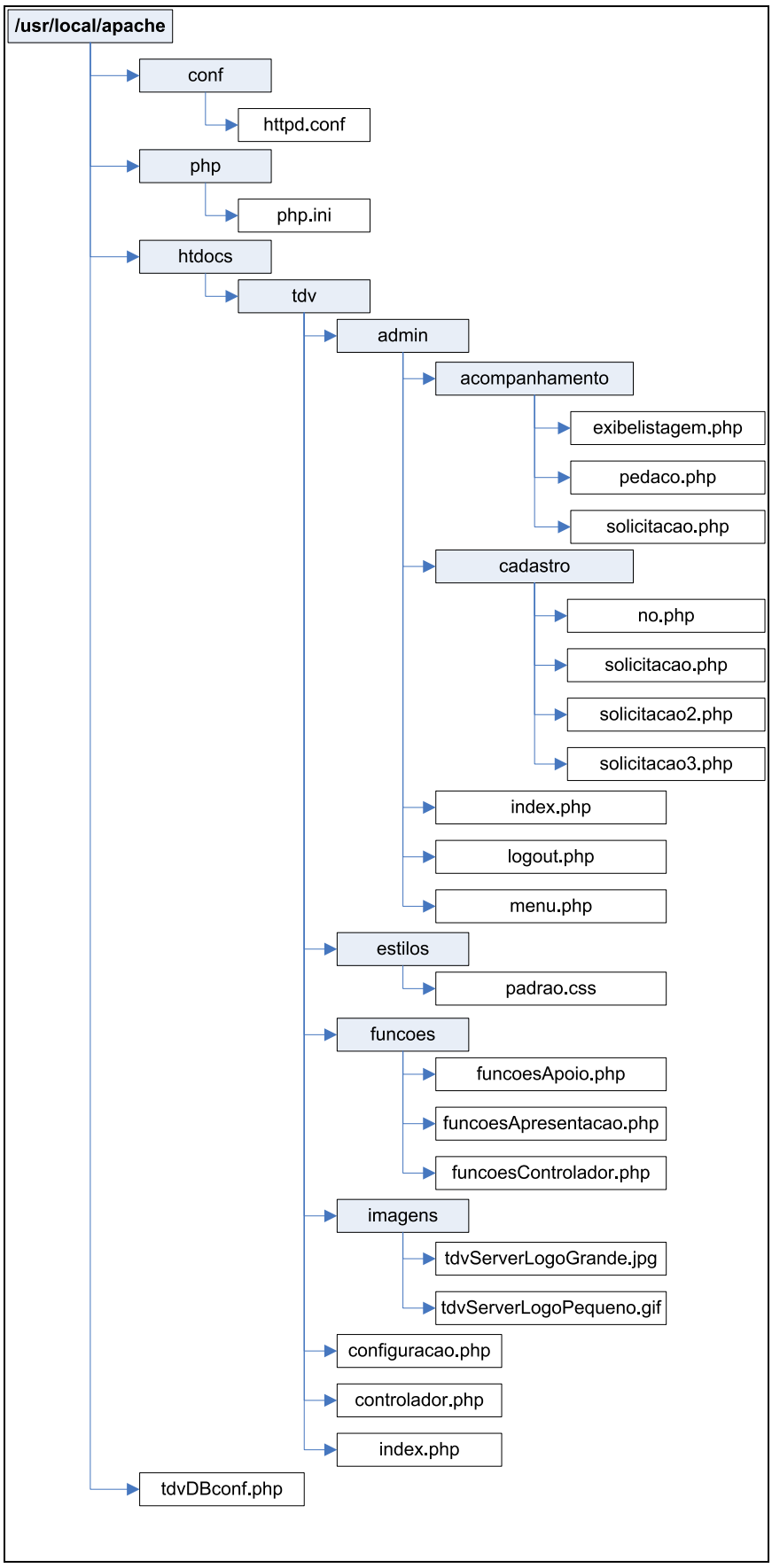

Figura 4.9 – Estrutura do servidor web

4.2.4.4 Servidor de Arquivos

Neste trabalho foi necessário a utilização de um servidor de arquivos no servidor para receber os arquivos transcodificados pelos nós de processamento. A utilização de um servidor de arquivos permite maior controle sobre o andamento do envio ao servidor e está menos sujeito a falhas quando se trabalha com arquivos que podem ultrapassar vários *megabytes* em tamanho. A utilização do protocolo HTTP para este tipo de transferência poderia resultar em estouro do tempo de execução do *script* no servidor *web*, resultando na perda da transferência.

Para solucionar o problema acima foi necessário a utilização de um servidor de arquivos que utilizasse o protocolo FTP, descrito anteriormente. A ferramenta escolhida foi o VSFTPD[19] (*Very Secure FTP Daemon)* versão 2.0.5-24. De acordo com o próprio autor da ferramenta, o mesmo é mais seguro e rápido quando comparado aos outros servidores FTP existentes para *Linux*.

Esta escolha foi feita pela grande aceitação desta ferramenta, sendo que a mesma é utilizada para servir arquivos em algumas grandes empresas e grupos que trabalham com desenvolvimento de distribuições e ferramentas *Linux* como a *Red Hat*[20], o *Novell SUSE*[21], a distribuição *Debian*[22] e o *FreeBSD*[23], dentre diversos outros.

Para a guarda dos arquivos enviados foi criado um diretório especial no servidor, localizado em */srv/tdv/pedacos/fina*l, conforme figura 4.10. Este diretório é visível tanto para o servidor VSFTPD como para o servidor *web*, tendo em vista que o segundo acessa os arquivos recebidos para efetuar a verificação de *checksum* dos mesmos com base nos dados repassados pelo nó de processamento responsável pelo envio. As pastas que estão com tonalidade mais escura no diagrama não são relevantes a este contexto e serão explicadas posteriormente.

A tabela 4.26 exibe a listagem dos principais arquivos vinculados ao servidor VSFTPD.

Tabela 4.26 – Descrição dos principais arquivos da árvore do servidor VSFTPD

Nome do Arquivo	Objetivo
/etc/vsftpd.conf	Arquivo de configuração do servidor VSFTPD. Vide apêndice B.

[19] Vide <http://vsftpd.beasts.org>
[20] Vide <http://www.redhat.com>
[21] Vide <http://www.suse.com>
[22] Vide <http://www.debian.org>
[23] Vide <http://www.freebsd.org>

/etc/vsftpd.user_list	Arquivo contendo a lista de usuários com permissão para acesso ao servidor. Vide apêndice B.
/var/log/vsftpd.log	Arquivo que guarda o registro das atividades executadas pelo servidor.

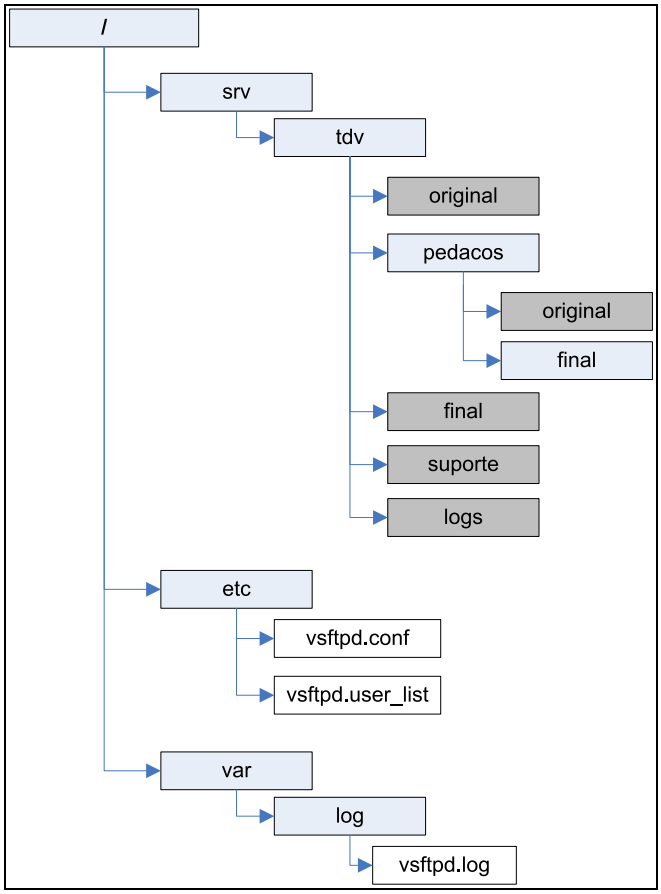

Figura 4.10 – Estrutura e arquivos principais do servidor VSFTPD

4.2.4.5 Sistema Gerenciador de Banco de Dados

Os requisitos do projeto resultaram diretamente na necessidade de utilização de um banco de dados para a guarda das informações referentes aos nós de processamento, solicitações e pedaços. O banco de dados escolhido para este fim foi o *MySQL*[24] *5.0.45 Community Edition*. A escolha deu-se devido a familiaridade com o mesmo, além da ótima integração existente com a linguagem PHP, possibilitando grande velocidade na recuperação e tratamento dos dados. Em conjunto com o servidor *web* Apache e o sistema operacional *Linux*, a solução assume a característica da arquitetura denominada *LAMP* (*Linux Apache MySQL PHP*), altamente confiável e de grande utilização nos servidores de hospedagem de páginas.

[24] Vide <http://dev.mysql.com>

A ferramenta de modelagem utilizada, o *Sybase PowerDesigner*[25] 12.5, versão de avaliação, gera automaticamente o *script* SQL para a criação das tabelas citadas no item 4.2.2. Para a administração do banco de dados e execução de pesquisas foram utilizadas as ferramentas *MySQL Administrator* e *MySQL Query Browser*, respectivamente, ambas disponíveis gratuitamente no *website* da ferramenta *MySQL*.

O banco de dados *MySQL* possui suporte a diversos tipos de tabelas de dados, entre os quais se destacam o *MyISAM*[26] que possui grande velocidade e é o mais utilizado atualmente, sendo recomendado para aplicações onde a maioria dos comandos são de leitura de dados, e o *InnoDB*, que possui suporte a transações e é recomendado em aplicações que possuem grande quantidade de atualizações ou gravações de registros.

Neste trabalho foi utilizado o tipo de tabela *InnoDB* justamente pelo suporte a transações, permitindo desfazer um conjunto de alterações, denominado transação, na base de dados caso uma ou mais das alterações apresente erro durante a sua execução. A utilização deste tipo de armazenamento contribui para manter a consistência da base de dados.

A configuração do banco de dados é armazenada no arquivo */etc/my.cnf*, criado automaticamente durante a instalação da ferramenta, disponível no apêndice B deste trabalho. Em conjunto com a criação das tabelas foram impostados os dados padrões das mesmas. O *script* de criação das tabelas, gerado pela ferramenta *PowerDesigner* e o *script* contendo os dados padrões para inserção nas tabelas estão disponíveis no apêndice C deste projeto.

[25] Para mais informações e obtenção de versão gratuita de avaliação da ferramenta visite <http://www.sybase.com.br/products/modelingmetadata/powerdesigner.shtml>

[26] Para mais informações sobre os tipos *MyISAM* e *InnoDB* visite <http://dev.mysql.com/doc/refman/5.0/en/storage-engines.html>

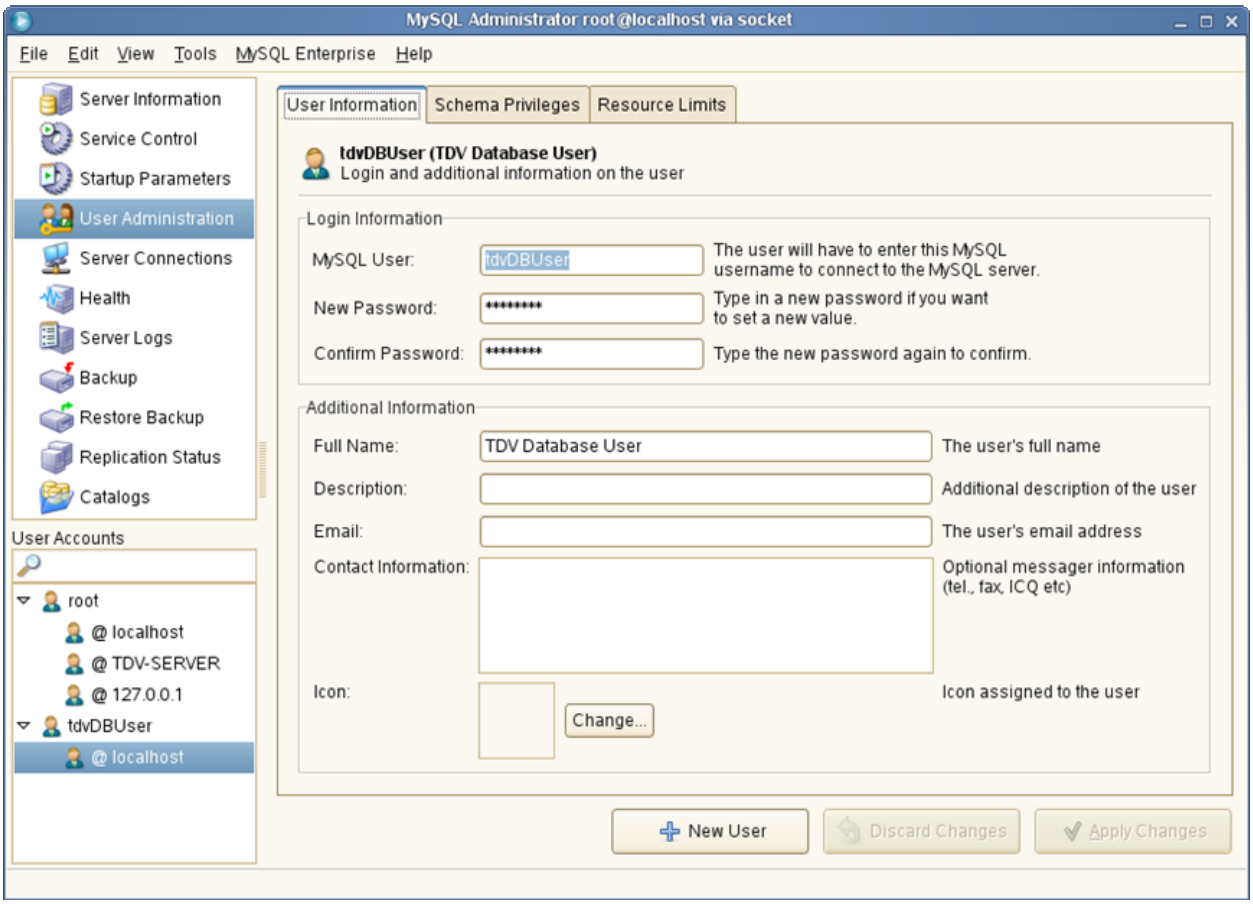

Figura 4.11 – Ferramenta de administração *MySQL Administrator*

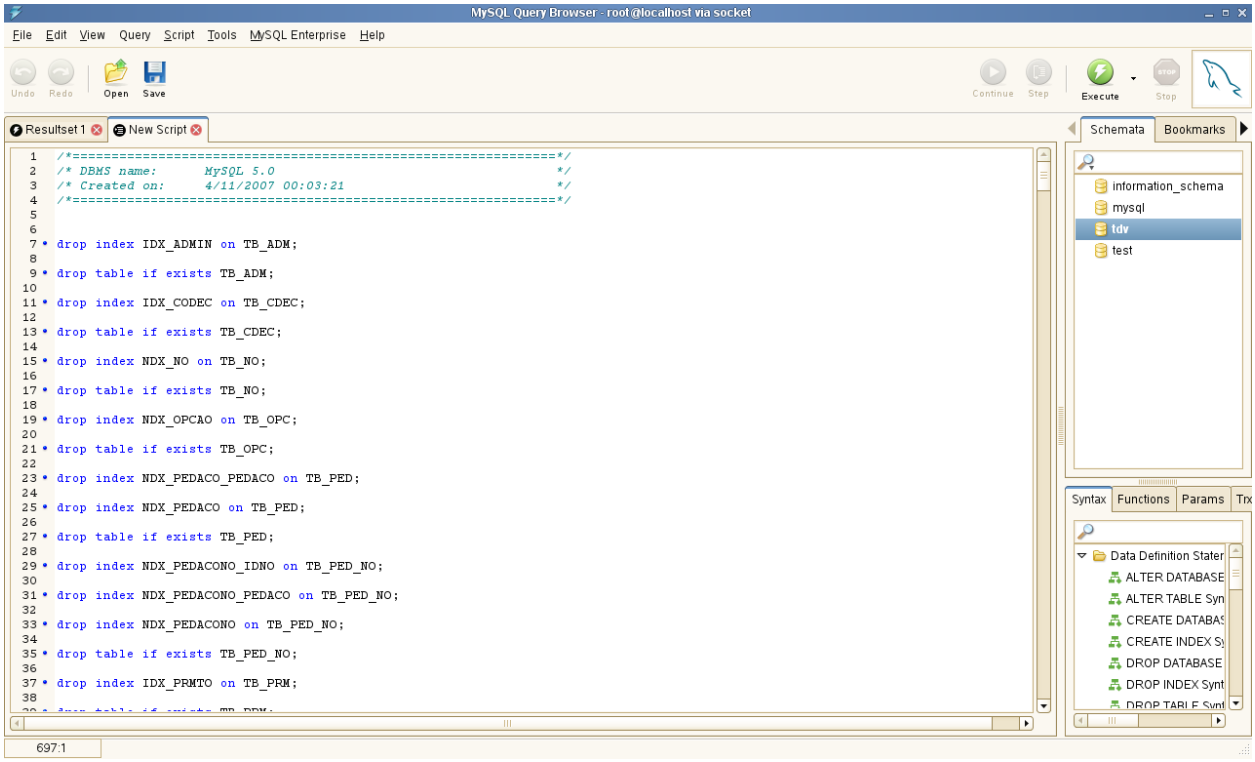

Figura 4.12 – Ferramenta *MySQL Query Browser*

4.2.4.6 Ferramentas de Visualização e Codificação de Vídeo

A ferramenta *MPlayer*[27], composta do visualizador *MPlayer* e do codificador *Mencoder*, foi utilizada neste projeto. A ferramenta possui código fonte aberto e é uma das melhores ferramentas de visualização de vídeo para o ambiente *Linux*, podendo ser utilizada também nas plataformas *Windows*, *FreeBSD* e *Mac OS X*.

A ferramenta *Mencoder* possui diversas dependências de bibliotecas, essas bibliotecas permitem a mesma codificar vídeos em diversos formatos, tornando-a extremamente versátil. As principais bibliotecas, que são importantes para este trabalho estão listadas abaixo:

- *x264*[28] : biblioteca de código fonte aberto que implementa o padrão de codificação MPEG-4 Parte 10, ITU-T H.264;
- *FAAC*[29] : biblioteca de código fonte aberto que implementa o padrão de codificação de áudio MPEG-4 Parte 3, também chamado de AAC (*Advanced Audio Coding*);
- *LAME*[30] : biblioteca de código fonte aberto que permite a codificação de áudio para o formato MPEG-1 *Layer* 3 (MP3).

Como este projeto visa criar uma ferramenta capaz de ser utilizada tanto em ambiente *Linux* como *Windows* foi necessário obter uma versão da ferramenta que já possuísse as bibliotecas necessárias compiladas. Felizmente existem usuários da comunidade que efetuam essas compilações sendo que neste projeto foi utilizado o *Mencoder Sherpya Build*[31], que já engloba as bibliotecas *x264*, *FAAC* e *LAME*.

Uma característica importante da ferramenta *Mencoder* consiste em sua capacidade de efetuar codificação utilizando mais de uma *thread* ao mesmo tempo, ou seja, em processadores que possuem mais de um núcleo, a ferramenta inicia automaticamente outras *threads*, de acordo com o número de núcleos, tornando a codificação mais rápida. Este recurso está disponível em todas as plataformas.

Para encapsular as faixas de áudio e vídeo codificadas em um *container* MP4 foram utilizadas duas ferramentas:

---

[27] Para mais informações visite <http://www.mplayerhq.hu>
[28] Para mais informações sobre o projeto visite <http://www.videolan.org/developers/x264.html>
[29] Para mais informações sobre o projeto visite <http://www.audiocoding.com>
[30] Para mais informações sobre o projeto viste <http://lame.sourceforge.net/index.php>
[31] Para mais informações e obtenção da ferramenta visite <http://oss.netfarm.it/mplayer-win32.php>

- *mp4creator* : ferramenta integrante do pacote MPEG4IP[32], projeto de código fonte aberto que possui diversas ferramentas de apoio visando, principalmente, a transmissão de áudio e vídeo via internet (*streaming*). A ferramenta em questão permite a criação de arquivos MP4 a partir de trilhas de áudio e vídeo;

- *MP4Box* : ferramenta do projeto GPAC[33] que possui diversas funcionalidades como o encapsulamento de áudio e vídeo em MP4 e 3GPP[34] além de possuir suporte a diversas outras tecnologias. A ferramenta em questão foi utilizada para permitir a unificação de diversos arquivos MP4 em apenas um arquivo final;

- *AviSplit* : ferramenta integrante do pacote Transcode[35], projeto que disponibiliza diversas ferramentas de transcodificação de áudio e vídeo, além de ferramentas de conversão entre *containers* e obtenção de informação sobre arquivos de áudio e vídeo. A ferramenta em questão foi utilizada para efetuar o processo de divisão do arquivo original em pedaços.

Os parâmetros de codificação utilizado neste trabalho foram obtidos através de consultas a diversas fontes, com destaque para a documentação da ferramenta *MPlayer* e dos parâmetros utilizados pelas ferramentas listadas abaixo:

- *JMencode*[36] : ferramenta de código fonte aberto com o propósito de criar uma interface visual em Java para a codificação de vídeos utilizando a ferramenta *Mencoder*;

- *MediaCoder*[37] : ferramenta livre de transcodificação de vídeo que utiliza diversas ferramentas e bibliotecas tanto proprietárias como de código aberto.

4.2.4.7 *Scripts* de Apoio

Para efetuar os procedimentos de divisão do arquivo original da solicitação e unificação dos pedaços transcodificados pelos nós de processamento foi necessário a criação de *scripts* que executam estes procedimentos periodicamente no servidor.

Estes *scripts* foram desenvolvidos através da utilização de linguagem *shell script*, que permite a realização dos procedimentos com grande velocidade. Existem diversos

[32] Para mais informações sobre o projeto visite <http://mpeg4ip.sourceforge.net/index.php>
[33] Para mais informações sobre o projeto visite <http://gpac.sourceforge.net/index.php>
[34] Para mais informações sobre este novo padrão para celulares visite <http://www.3gpp.org>
[35] Para mais informações sobre o projeto visite <http://www.transcoding.org>
[36] Para mais informações sobre o projeto visite <http://jmencode.sourceforge.net>
[37] Para mais informações sobre o projeto visite <http://mediacoder.sourceforge.net>

interpretadores de *shell script*, neste projeto foi escolhido o *BASH*[38] *(Bourne Again Shell)* pois engloba diversas funcionalidades de outros interpretadores de *shell script*, a citar, *Korn*[39] *Shell* (ksh) e *C Shell (csh)*.

Para efetuar o acesso a base de dados foram utilizados *scripts* desenvolvidos em PHP, que recebem parâmetros dos *shell scripts* e devolvem os resultados de forma estruturada. A utilização de *scripts* PHP para o acesso a base facilitou a manipulação de dados, o que seria extremamente trabalhoso através dos *shell scripts*, além de permitir a reutilização de algumas funções da interface *web* e do arquivo de configuração *configuracao.php*. O projeto explorou um recurso interessante da linguagem PHP que é justamente a possibilidade de passagem de parâmetros via linha de comando para o interpretador PHP, ou seja, é possível acionar páginas desenvolvidas em PHP sem a necessidade de utilização de um servidor *web*.

Os *shell scripts* desenvolvidos são acionados através da utilização do *daemon* CRON, um agendador de tarefas que utiliza o tempo como fator determinante para a execução de comandos agendados. O *daemon* CRON está presente em todas as distribuições *Linux* e permite a execução de tarefas repetitivas como o *backup* de arquivos e, neste caso, a verificação de arquivos pendentes de divisão e pedaços pendentes de unificação. A inclusão de tarefas é feita pelo arquivo */etc/crontab*, que é lido pelo *daemon CRON*.

Neste projeto, os *shell scripts* são acionados de minuto a minuto pelo *daemon CRON*, e verificam a base de dados por solicitações pendentes de divisão, unificação ou pedaços alocados que estão há algum tempo sem atualização. Como os processos de divisão e unificação podem se estender por mais de um minuto, os *shell scripts* desenvolvidos possuem controle de trava, ou seja, ao iniciar, o próprio *shell script* verifica se já existe uma instância do mesmo em execução, caso positivo o mesmo aborta o procedimento. Esta verificação utiliza o identificador do processo no sistema operacional, tornando possível a averiguação da situação do mesmo.

Em conjunto com o controle de trava os *shell scripts* desenvolvidos também geram arquivos de *log*, registrando as atividades realizadas e o horário de início e término das mesmas. Os arquivos são nomeados de acordo com o dia da execução do *shell script*, tornando fácil a localização dos registros.

A figura 4.13 demonstra a localização dos arquivos relacionados aos processos executados pelos *shell scripts* dentro da estrutura geral do servidor. A tabela 4.27 descreve a função de cada pasta e arquivo.

[38] Para mais informações sobre o projeto visite <http://www.gnu.org/software/bash>
[39] Vide <http://www.kornshell.com>

Tabela 4.27 – Descrição dos principais arquivos e pastas dos *scripts* de apoio

Nome do Arquivo ou Pasta	Objetivo
/etc/crontab	Arquivo onde são registradas as tarefas a serem executadas pelo *daemon CRON*. Vide apêndice D.
/srv/tdv/original	Pasta onde são guardados os arquivos originais que podem ser utilizados em solicitações de transcodificação.
/srv/tdv/pedacos/original	Pasta onde são armazenados os pedaços criados pelo *shell script* de divisão.
/srv/tdv/pedacos/final	Pasta onde são armazenados os pedaços transcodificados pelos nós de processamento.
/srv/tdv/final	Pasta onde são armazenados os arquivos transcodificados, após o processo de unificação.
/srv/tdv/suporte/ TDVDivisao.sh	*Shell script* que efetua a divisão do arquivo original da solicitação em pedaços, gravando-os na pasta /srv/tdv/pedacos/original. Vide apêndice D.
/srv/tdv/suporte/ TDVDivisao.php	*Script* PHP utilizado pelo *shell script* de divisão. Este *script* verifica a base de dados, inclui registros e atualiza a situação de solicitações e pedaços. Vide apêndice D.
/srv/tdv/suporte/ TDVUnificacao.sh	*Shell script* que efetua a unificação dos pedaços transcodificados para a geração do arquivo final, gravando-o na pasta /srv/tdv/final. Vide apêndice D.
/srv/tdv/suporte/ TDVUnificacao.php	*Script* PHP utilizado pelo *shell script* de unificação. Este *script* verifica a base de dados por solicitações pendentes de unificação, recupera dados de pedaços e atualiza a situação de solicitações e pedaços. Vide apêndice D.
/srv/tdv/suporte/ TDVDesalocacao.php	*Script* PHP que efetua a desalocação dos pedaços caso os mesmos estejam sem nenhuma atualização de acordo com a tolerância especificada. Vide apêndice D.
/srv/tdv/suporte/ TDVDesalocacao.sh	*Shell script* que aciona o *script* PHP que efetua a desalocação de pedaços. Vide apêndice D.
/srv/tdv/logs/divisao	Pasta que armazena os *logs* de atividade do *shell script* de divisão. Vide apêndice D para exemplo.
/srv/tdv/logs/unificação	Pasta que armazena os *logs* de atividade do *shell scripts* de unificação. Vide apêndice D para exemplo.

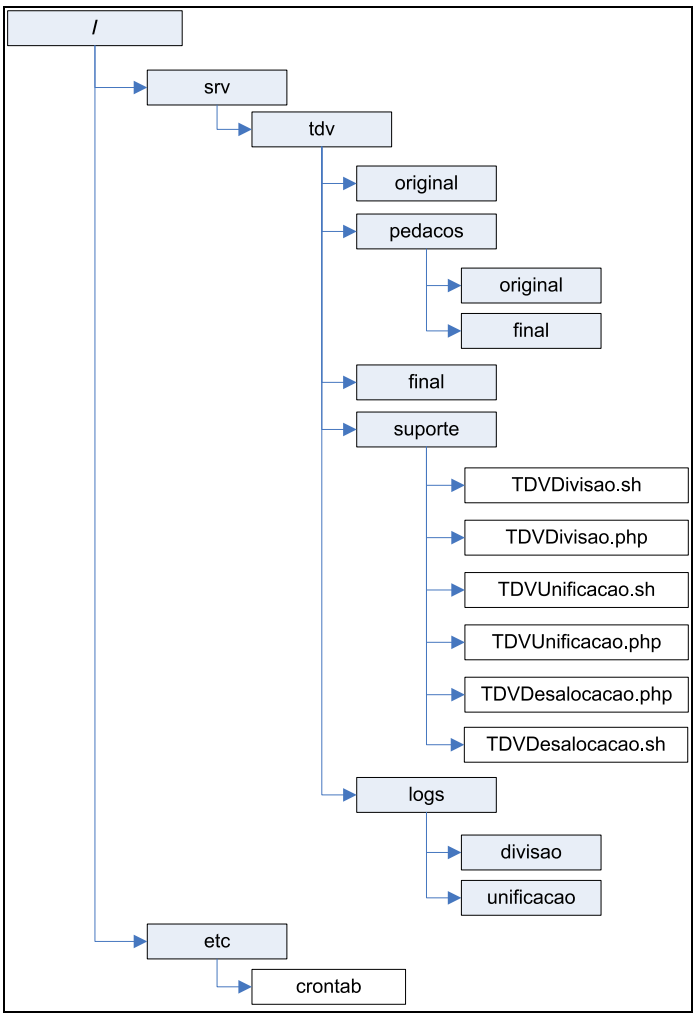

Figura 4.13 – Estrutura de diretórios para os *scripts* de apoio

4.2.5 Descrição do Nó de Processamento

Cada computador que se conecta ao servidor através da aplicação cliente desenvolvida é um nó de processamento. O nó de processamento realiza grande parte do processamento, sendo responsável pela efetiva transcodificação dos pedaços de vídeo. Nos itens seguintes são descritas as principais características da aplicação cliente desenvolvida para utilização nos mesmos.

4.2.5.1 Sistema Operacional

A aplicação cliente foi projetada para ser utilizada tanto em plataforma *Windows* como em distribuições *Linux*, garantindo a portabilidade da mesma. Para garantir esta portabilidade foi utilizada a linguagem de programação Java, versão 6, para o desenvolvimento da mesma. Esta linguagem foi escolhida pela sua grande utilização, portabilidade e familiaridade do autor com a mesma.

4.2.5.2 Descrição da Aplicação

A aplicação utilizada pelos nós de processamento, desenvolvida em Java, utiliza o conceito de orientação a objetos e foi desenvolvida através da utilização do editor *NetBeans*[40] *6.0 beta 2*, disponível gratuitamente. Esta escolha deu-se em função da facilidade de utilização do mesmo, principalmente para a criação de interfaces gráficas em linguagem Java e a familiaridade do autor com o mesmo.

Nos itens seguintes são descritas as bibliotecas de apoio utilizadas e a composição da aplicação.

4.2.5.2.1 Ferramentas e Bibliotecas de Apoio

Uma vantagem importante da orientação a objetos é a reutilização, permitindo que projetos possam usufruir de bibliotecas desenvolvidas e extensivamente testadas por comunidades de usuários. Neste projeto são utilizadas diversas bibliotecas amplamente difundidas para satisfazer parte dos requisitos funcionais da aplicação, conforme listagem abaixo:

- *Apache Commons Configuration*[41]: biblioteca criada pela *Apache Software Foundation* que permite a leitura, gravação e atualização de arquivos de configuração de forma transparente para a aplicação. Neste projeto esta biblioteca é utilizada para gerar e atualizar o arquivo de configuração utilizado pela aplicação;

- *Apache Commons Net*[42]: biblioteca criada pela *Apache Software Foundation* que possui suporte a diversos protocolos. Neste projeto esta biblioteca é utilizada para implementar um cliente FTP, permitindo a conexão da aplicação ao servidor VSFTPD para envio de arquivos transcodificados;

- *Apache Jakarta HttpComponents HttpClient*[43]: biblioteca criada pela *Apache Software Foundation* que implementa um cliente HTTP, possibilitando a execução de atividades semelhantes às executadas por um navegador *web*. Neste projeto

[40] Para obtenção da ferramenta visite <http://www.netbeans.org>
[41] A biblioteca e o código fonte da mesma estão disponíveis no *website* <http://commons.apache.org/configuration/>
[42] A biblioteca e o código fonte da mesma estão disponíveis no *website* <http://commons.apache.org/net/>
[43] A biblioteca e o código fonte da mesma estão disponíveis no *website* <http://jakarta.apache.org/httpcomponents/>

esta biblioteca é utilizada para permitir o envio e recebimento de dados através do protocolo HTTP e requisições GET e POST;

- *JackSUM*[44]: biblioteca de código fonte aberto criada por Johann Nepomuk Loefflmann, que implementa mais de 50 algoritmos de geração e verificação de *checksums*. Neste projeto esta biblioteca é utilizada para geração e verificação de *checksums* SHA-256.

A aplicação também utiliza a ferramenta *PsKill*[45], versão 1.12, desenvolvida pela *Microsoft* que permite o encerramento de processos em sistemas operacionais *Windows* NT ou superior através de linha de comando. A ferramenta foi utilizada para encerrar o processo da ferramenta *Mencoder* caso o usuário decida encerrar a aplicação durante o processo de transcodificação.

4.2.5.2.2 Comunicação

Conforme descrito anteriormente a comunicação entre os nós de processamento e o servidor é feita através da utilização do protocolo HTTP e envio de mensagens. Para satisfazer este requisito a aplicação possui as seguintes classes:

- *Connection*: classe responsável por efetuar a conexão com o servidor *web* e executar transações recebidas das outras classes. A classe disponibiliza também mecanismos para obter o retorno enviado pelo servidor;
- *Configuration*: classe responsável por armazenar as configurações do nó de processamento e dados de conexão como o endereço do servidor *web*, usuários e senhas. É utilizada pela classe *Connection*;
- *TransactionType*: classe que define os tipos de transações que podem ser utilizadas pelas demais classes: autenticar nó de processamento, solicitar pedaço para transcodificação, solicitar *download* de arquivo, informar andamento de *download*, informar andamento de transcodificação, informar andamento de *upload* e desconectar nó de processamento;
- *Transaction*: classe que permite a criação uma nova transação com base no tipo de transação informado (*TransactionType*) e lista de parâmetros a serem utilizados;

[44] A biblioteca e o código fonte da mesma estão disponíveis no *website* <http://www.jonelo.de/java/jacksum>

[45] Para mais informações e obtenção da ferramenta visite <http://www.microsoft.com/technet/sysinternals/utilities/pskill.mspx>

- *ResponseParser*: classe utilitária que recebe a resposta enviada pelo servidor e formata a mesma gerando uma coleção de dados que pode ser consultada pelas demais classes. Utilizada pela classe *Connection*;
- *SHA256*: classe utilitária que codifica o texto recebido em um *hash* utilizando o padrão SHA-256. Esta classe é utilizada pela classe *Configuration*.

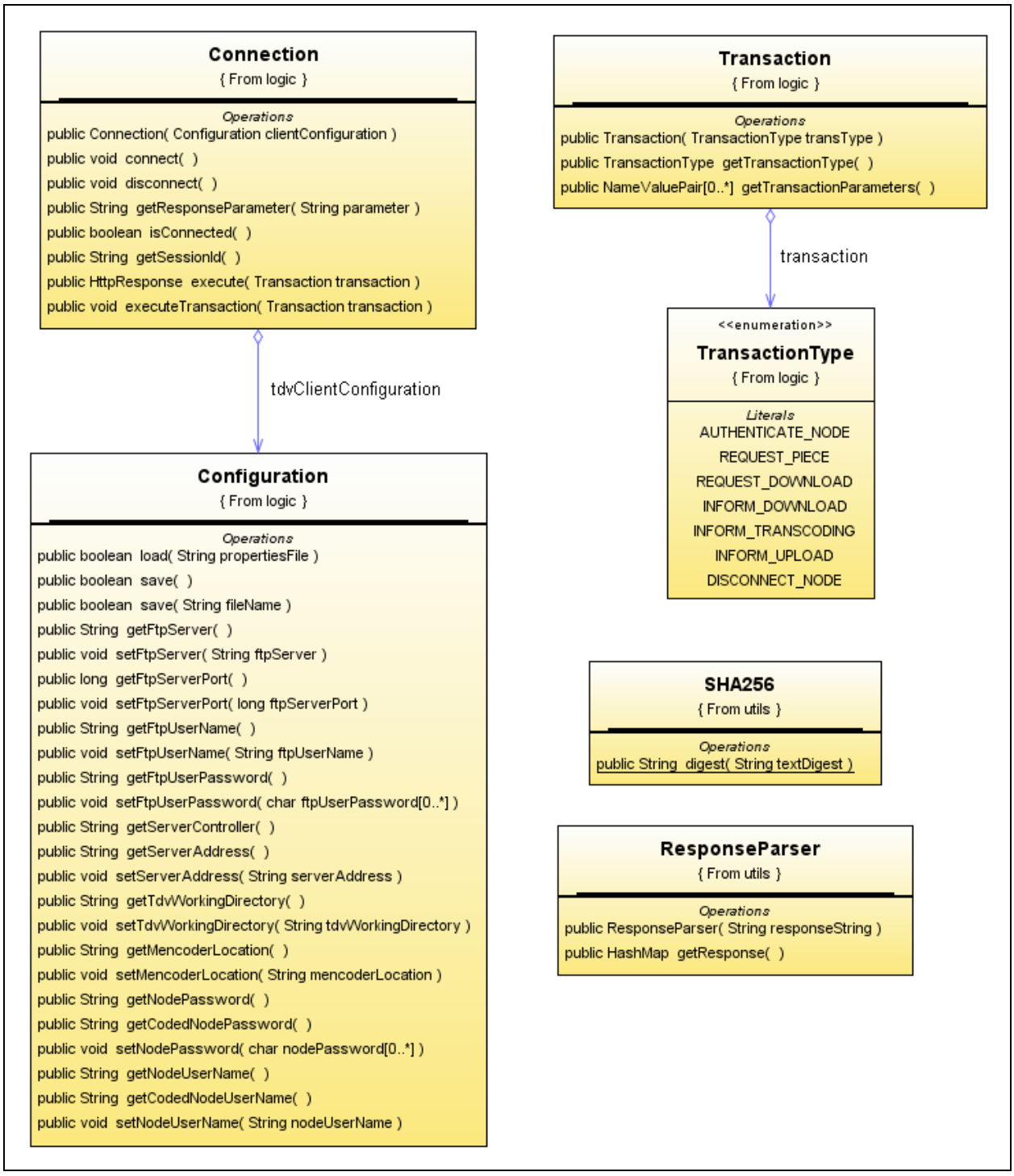

Figura 4.14 – Principais classes de conexão do servidor

4.2.5.2.3 Processos

Durante o desenvolvimento da aplicação foi necessário encapsular as etapas do processo de transcodificação, para atingir este objetivo foram definidos processos que possuem uma *interface* comum chamada *GenericProcess*, que define alguns métodos:

- *getErrorReason()*: método que retorna o motivo do erro de acordo com os definidos na classe *ErrorReason;*
- *executeProcess()*: método que inicia a execução do processo e retorna verdadeiro se o mesmo for executado com sucesso ou falso se ocorrer algum erro;
- *abortProcess()*: método que aborta o processo em execução, utilizado principalmente quando o usuário do nó de processamento deseja encerrar a aplicação.

Conforme citado, é possível obter o erro que ocorreu durante o processo. Os erros possíveis estão mapeados na classe *ErrorType*:

- *NO_ERROR*: não ocorreu erro, estado padrão inicial;
- *CONNECTION*: erro de conexão, geralmente devido à variável de sessão inválida ou expirada, exige uma nova autenticação junto ao servidor;
- *FILE_NOT_FOUND*: o arquivo solicitado ao servidor, geralmente durante o processo de *download*, não foi encontrado pelo mesmo;
- *NO_PIECE_ALLOCATED*: não existe pedaço alocado para o nó de processamento. Este erro pode ocorrer caso o nó de processamento pare de responder durante algum tempo e durante este período o servidor efetue a desalocação do pedaço vinculado ao mesmo. O procedimento padrão para este erro é solicitar um novo pedaço para transcodificação;
- *DOWNLOADING*: erro durante o processo de *download* do arquivo;
- *CHECKSUM*: *checksum* do arquivo recebido é inválido, ou seja, o arquivo foi corrompido durante a transferência, ou erro de verificação de *checksum* no servidor, ou seja, o arquivo enviado pelo nó de processamento ao servidor foi corrompido durante a transferência;
- *TRANSCODING*: erro durante o processo de transcodificação do arquivo;
- *UPLOADING*: erro durante o processo de envio do arquivo transcodificado ao servidor (*uploading*);

- *CONFIGURATION*: erro decorrente do envio de parâmetros inválidos ou incompletos ao servidor para uma determinada transação;
- *SEVERE*: erro severo, decorrente de falha no servidor;
- *HALT*: erro decorrente de solicitação de aborto do processo em execução, ou seja, o processo não foi concluído com sucesso pois foi necessário abortar o mesmo devido à solicitação do usuário ou falha nas transações de envio de informação.

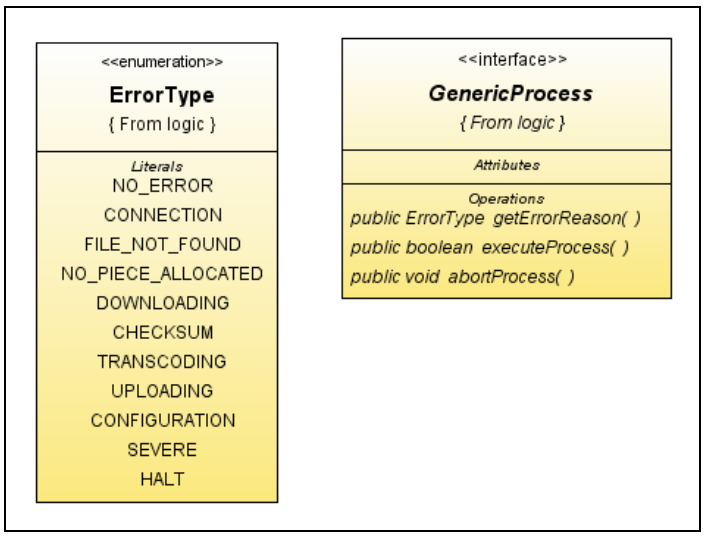

Figura 4.15 – Interface *GenericProcess* e classe *ErrorType*

4.2.5.2.3.1 Processos de Conexão e Desconexão

As funcionalidades de conexão ao servidor e desconexão do mesmo são providas pelas classes *ConnectProcess* e *DisconnectProcess*, respectivamente. Estas classes implementam a interface *GenericProcess* e utilizam a classe *ErrorType* em conjunto com diversas outras.

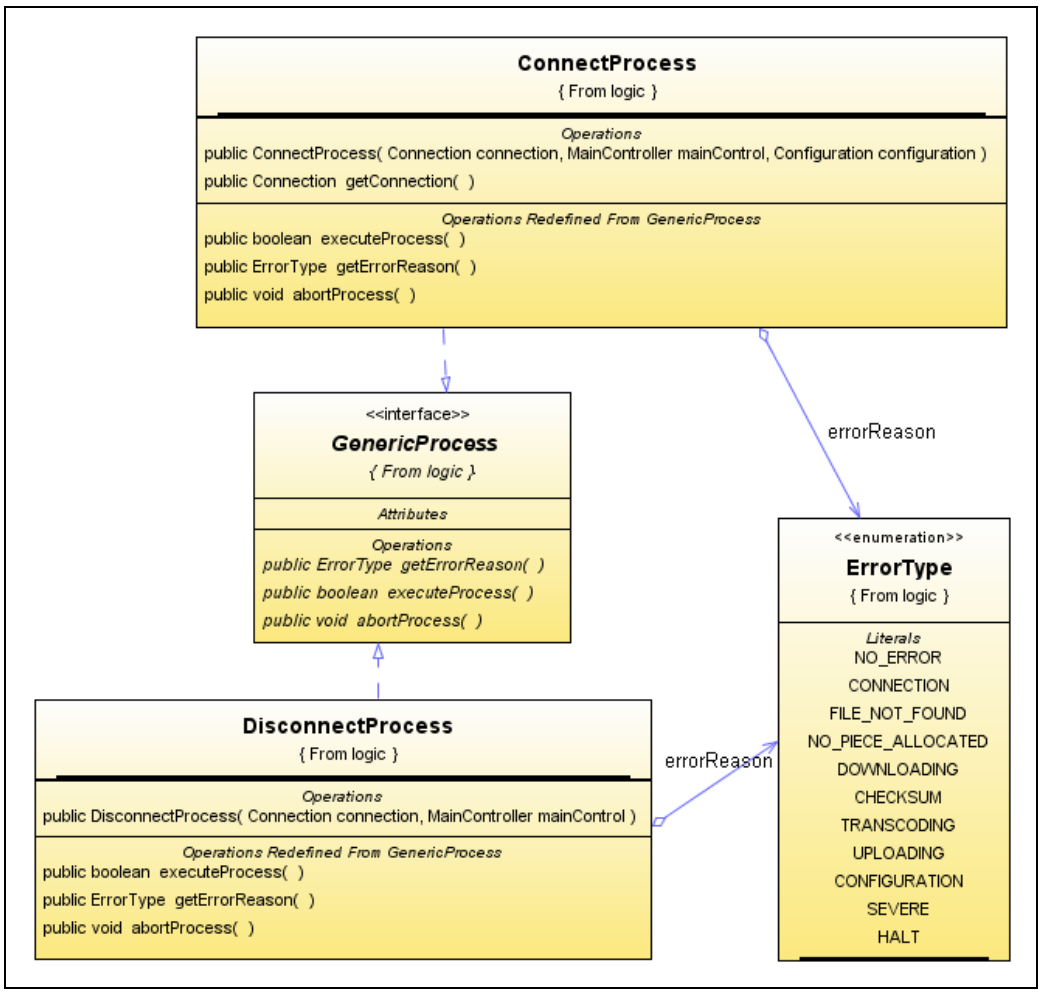

Figura 4.16 – Classes dos processos de conexão e desconexão

4.2.5.2.3.2 Processo de Solicitação de Pedaço

O processo de solicitação de um novo pedaço para transcodificação é realizado pela classe *RequestPieceProcess*. Esta classe recebe uma referência para um objeto da classe *Piece*, que é responsável pela guarda de dados referentes ao pedaço alocado para o nó de processamento. Após o término do processo, o objeto da classe *Piece* já terá alguns dados, como nome do arquivo, *checksum* do arquivo recebido e parâmetros de codificação atribuídos.

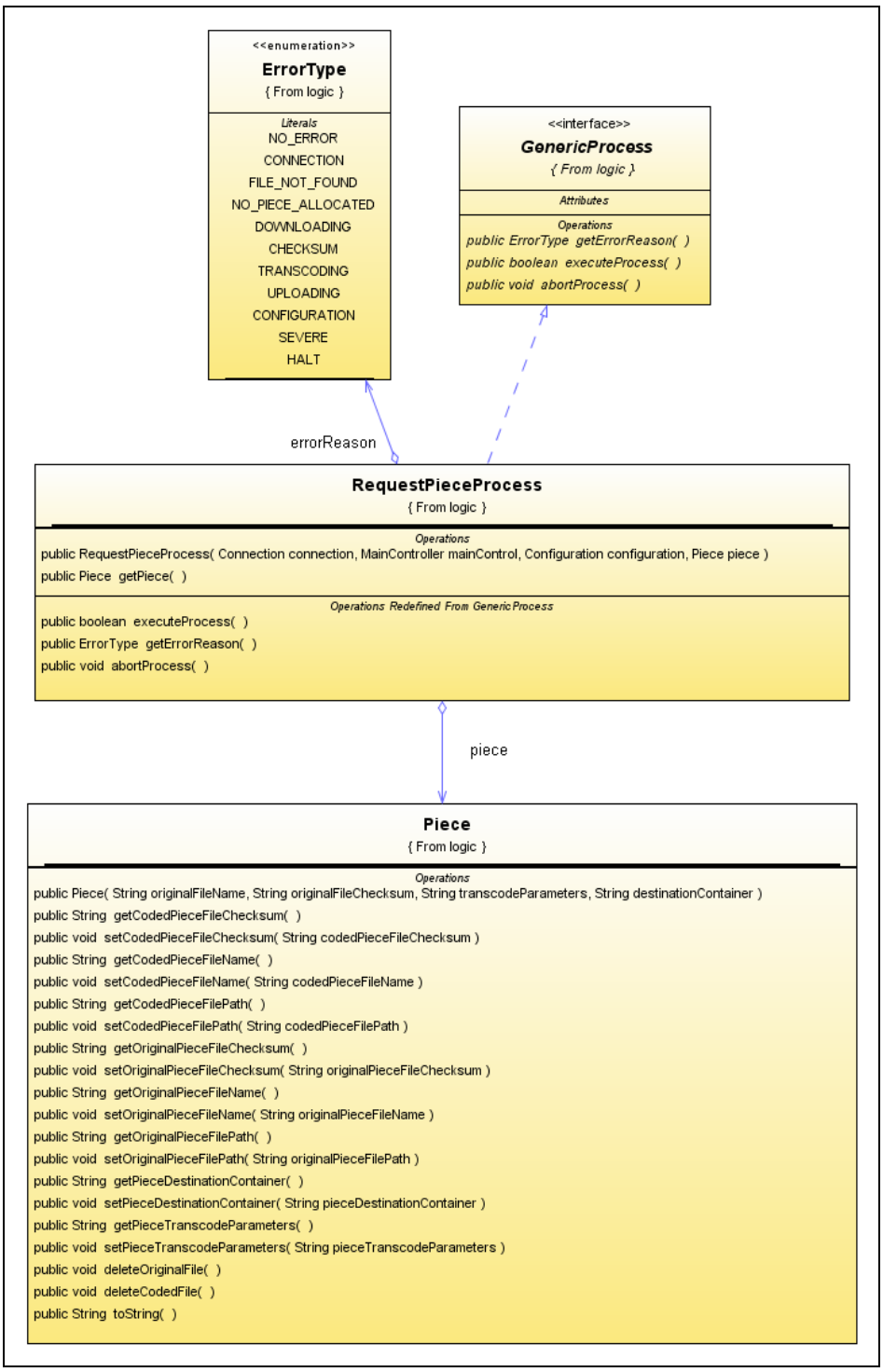

Figura 4.17 – Classes do processo de solicitação de pedaço

4.2.5.2.3.3 Processo de *Download*

O processo de *download* do arquivo vinculado ao pedaço alocado é realizado pela classe *DownloadPieceProcess*. Esta classe cria uma instância da classe *Downloader* que efetivamente efetua o *download* do arquivo para o nó de processamento. Após o término do processo de *download* o nó de processamento terá o arquivo do pedaço alocado em disco e com o *checksum* do mesmo verificado.

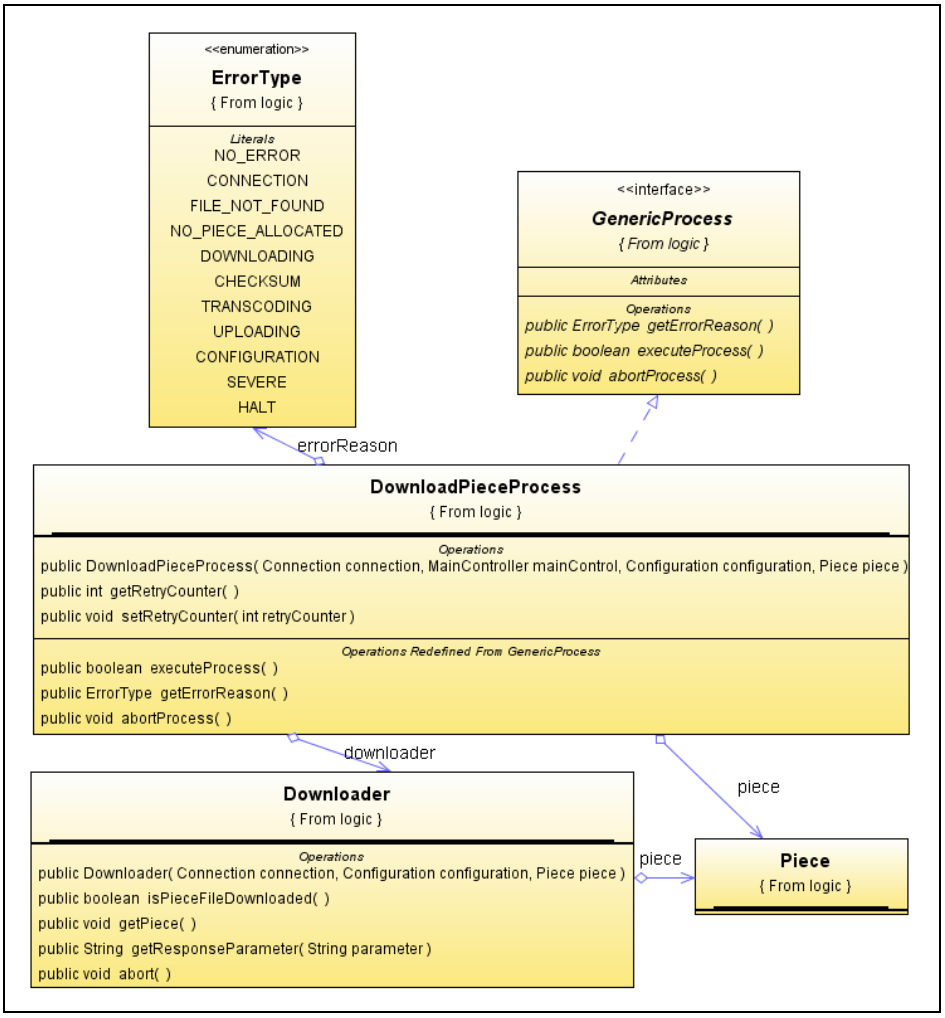

Figura 4.18 – Classes do processo de *download* de pedaço

4.2.5.2.3.3 Processo de Transcodificação

O processo de transcodificação, ou seja, a conversão do pedaço de vídeo original com base nos parâmetros especificados pelo servidor, é efetuado pela classe *TranscodeProcess*. Esta classe cria objetos das classes *Transcoder*, responsável por efetivamente iniciar a transcodificação e *TranscodeInformation* que envia informações sobre o andamento do processo de transcodificação para o servidor de forma assíncrona. A classe *Transcoder* utiliza a classe utilitária *ExecHelper* para acionar a ferramenta de codificação através da linha de comando do sistema operacional. Assim como os demais processos, este também possui a opção de abortar o procedimento em execução caso necessário.

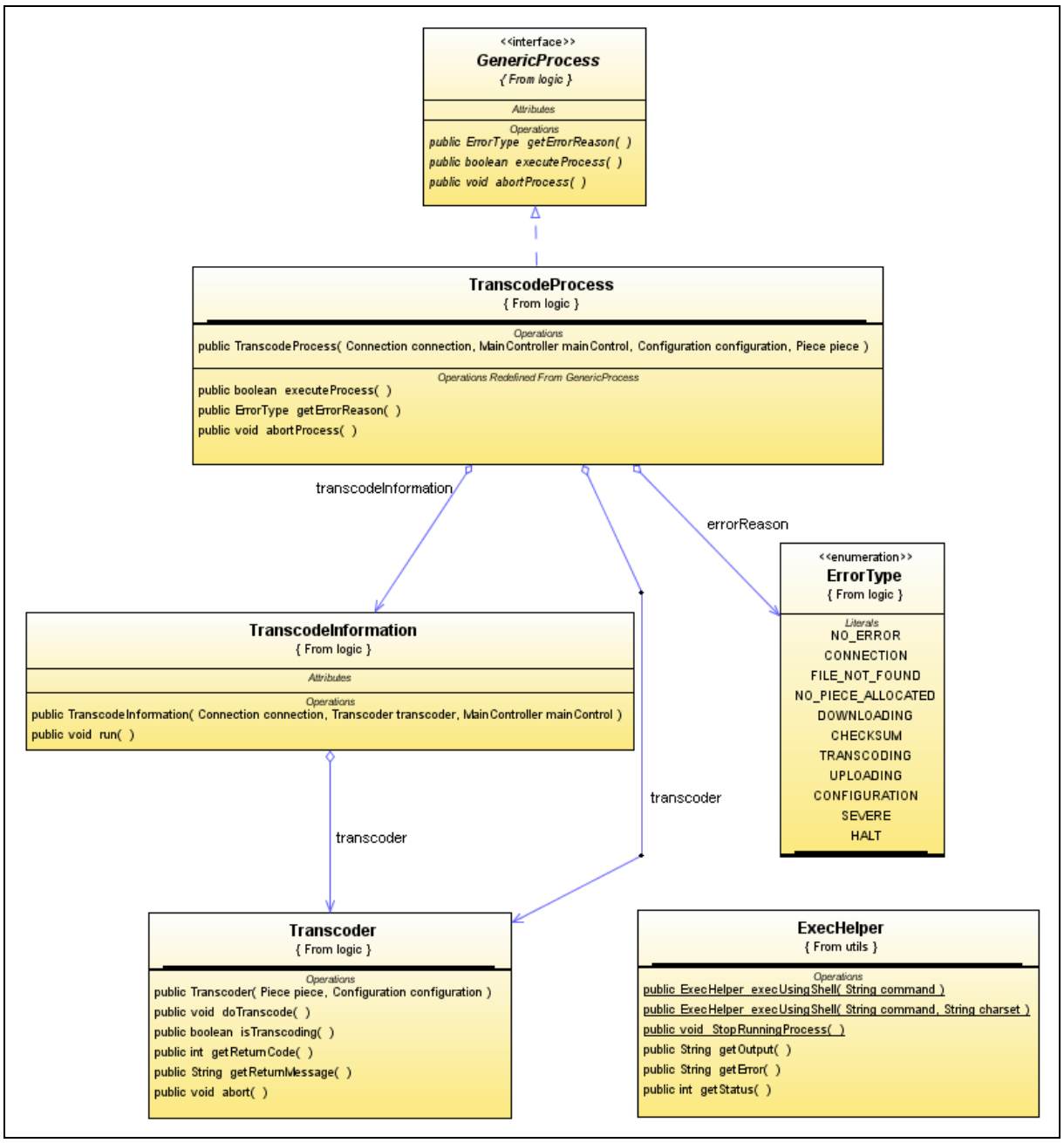

Figura 4.19 – Classes do processo de transcodificação

4.2.5.2.3.4 Processo de *Upload*

O processo de *upload* do arquivo transcodificado para o servidor é realizado pela classe *UploadProcess*. Esta classe utiliza a classe *Uploader* para gerenciar o processo de transferência. A classe *Uploader* por sua vez instancia a classe *JakartaFtpWrapper* que abre uma conexão com o servidor FTP e transfere o arquivo transcodificado. Durante todo o processo, a classe *UploadInformation* informa o andamento do mesmo ao servidor. Após o fim deste processo o arquivo transcodificado estará disponível no servidor e o nó estará pronto para receber outro pedaço.

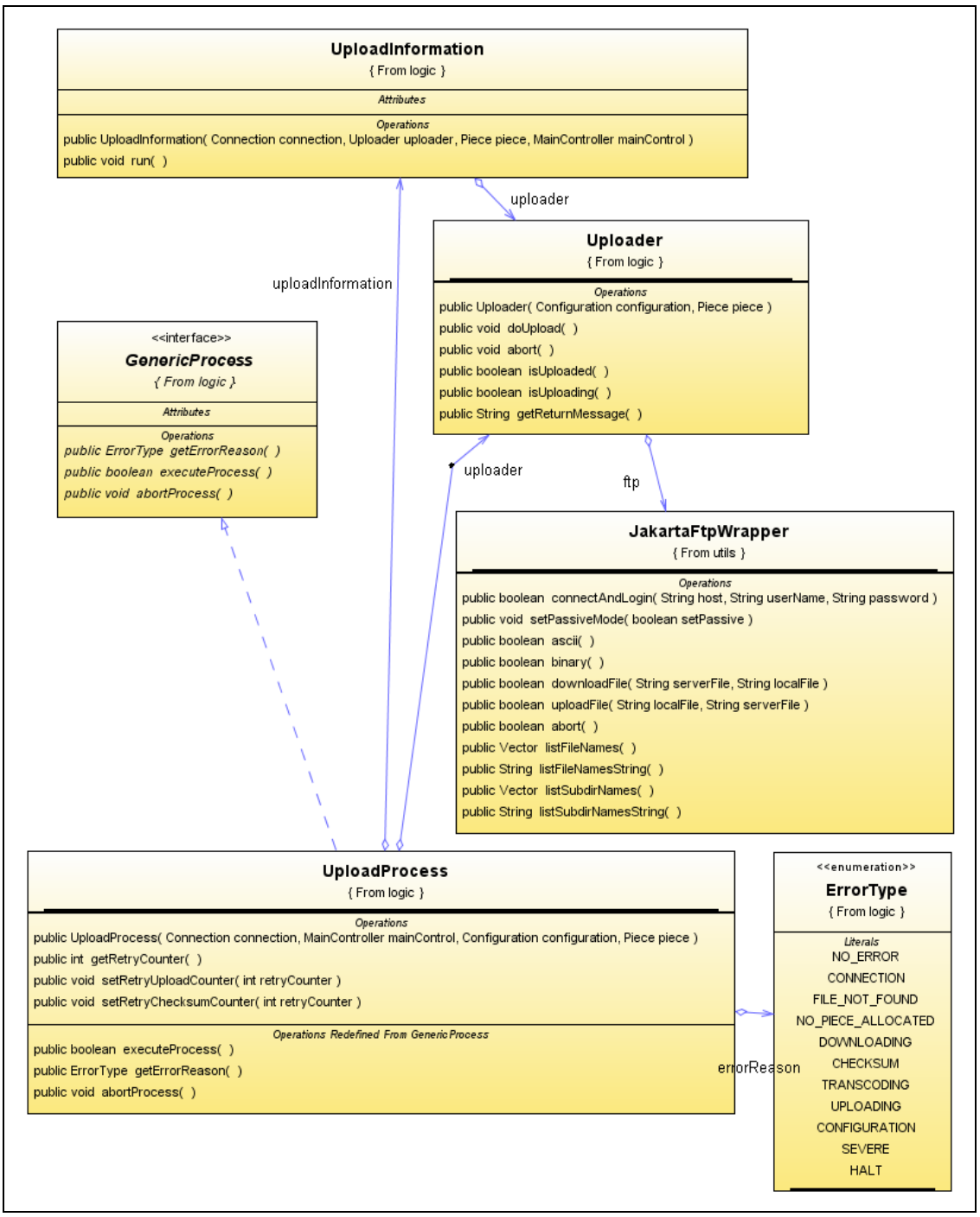

Figura 4.20 – Classes do processo de *upload*

4.2.5.2.3.5 A classe *Job*

Para efetuar o controle do fluxo da aplicação de forma integrada foi criada a classe *Job* que possui referências para todas as outras classes de processo. A classe *Job* é instanciada pelo controlador principal da aplicação, que será citado posteriormente. A utilização da classe *Job* facilita o gerenciamento por parte do controlador e permite que o mesmo solicite que a tarefa em execução seja abortada sem a necessidade de conhecer em que estágio a mesma se encontra.

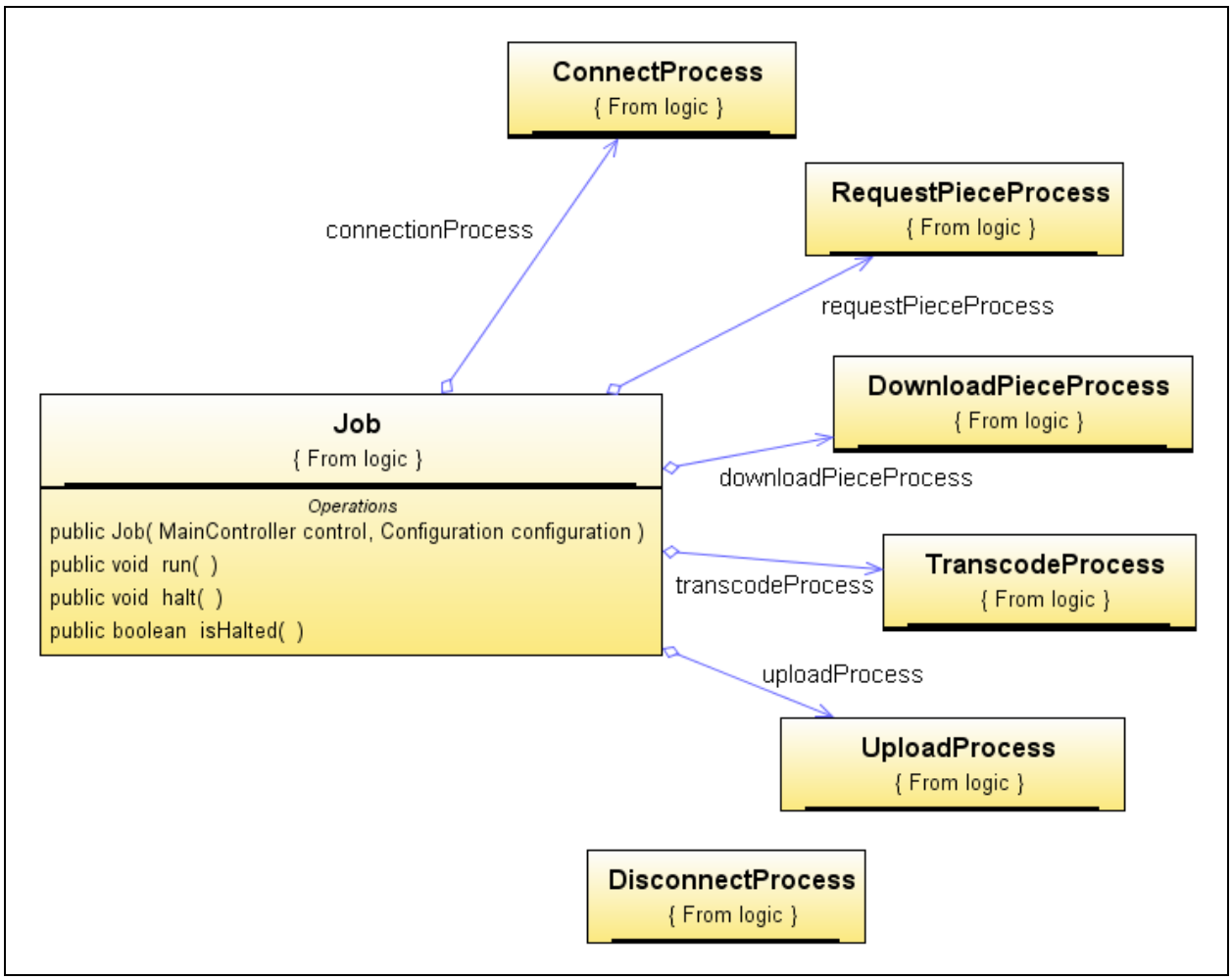

Figura 4.21 – Classes *Job* e principais relacionamentos

4.2.5.2.4 Camada de Apresentação

Para facilitar a interação com o usuário a aplicação possui uma camada de apresentação composta de três telas principais. As telas são acionadas através de classes controladoras chamadas *MainController* e *ConfigurationController*. As interfaces são renderizadas pelas classes *MainFrame, ConfigurationFrame* e *AboutFrame.*

4.2.5.2.4.1 Janela Principal

A interface principal da aplicação é composta de um *menu* de opções e uma área de texto que exibe os comandos efetuados e os retornos recebidos, conforme figura 4.22. A interface é renderizada pela classe *MainFrame* que é acionada pela classe controladora *MainController.* A figura 4.23 exibe as principais classes responsáveis pelo controle e renderização da janela principal.

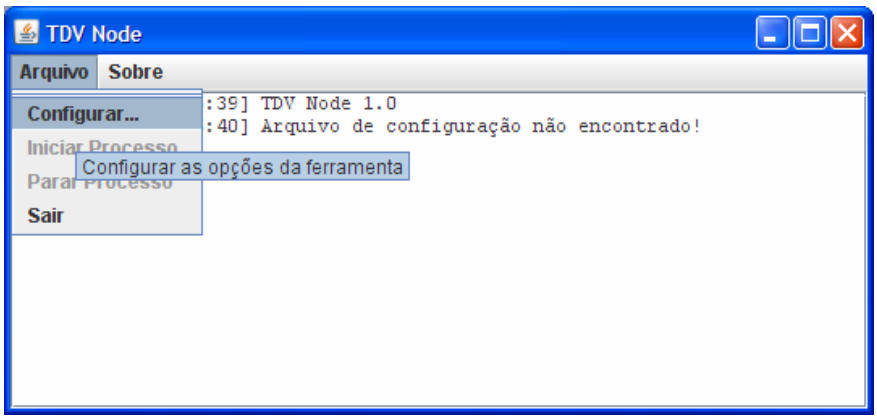

Figura 4.22 – Janela principal da aplicação cliente

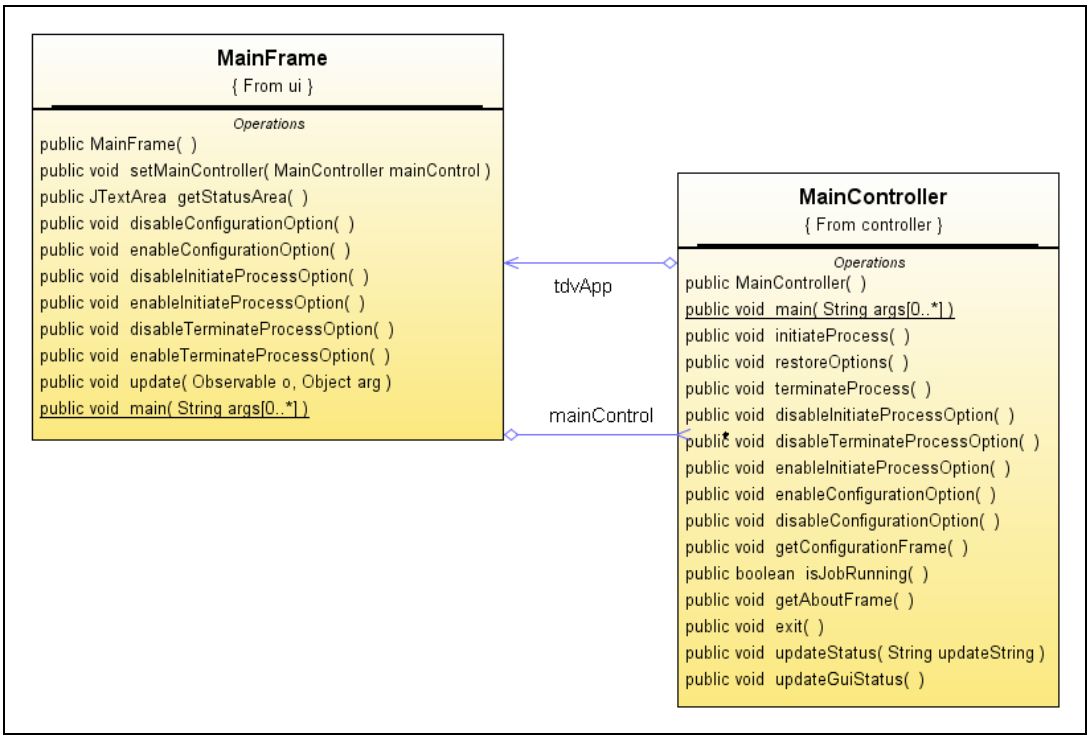

Figura 4.23 – Classes principais para renderização e controle da janela principal

4.2.5.2.4.2 Janela de Configuração

A janela de configuração da aplicação exibe os campos que devem ser preenchidos para permitir que a mesma prossiga com o processamento. Os dados necessários estão listados na figura 4.24. Por segurança a aplicação não efetua a gravação das senhas dos usuários do servidor *web* e servidor FTP em disco rígido, os demais dados são armazenados em um arquivo de configuração.

Ao parar o cursor encima de algum campo, é exibida uma descrição para auxiliar o usuário no preenchimento do mesmo. A aplicação também valida se o endereço IP

especificado para o servidor FTP pode ser alcançado, caso negativo, exibe uma mensagem de erro. Em conjunto com esta validação a aplicação também verifica se todos os demais campos foram preenchidos, caso negativo, exibe diversas mensagens de erro de validação.

Em ambiente *Linux*, não é necessário informar a localização da ferramenta *Mencoder*, tendo em vista que a mesma, caso instalada, pode ser acionada diretamente via linha de comando, sendo assim, a última opção de configuração é desabilitada automaticamente.

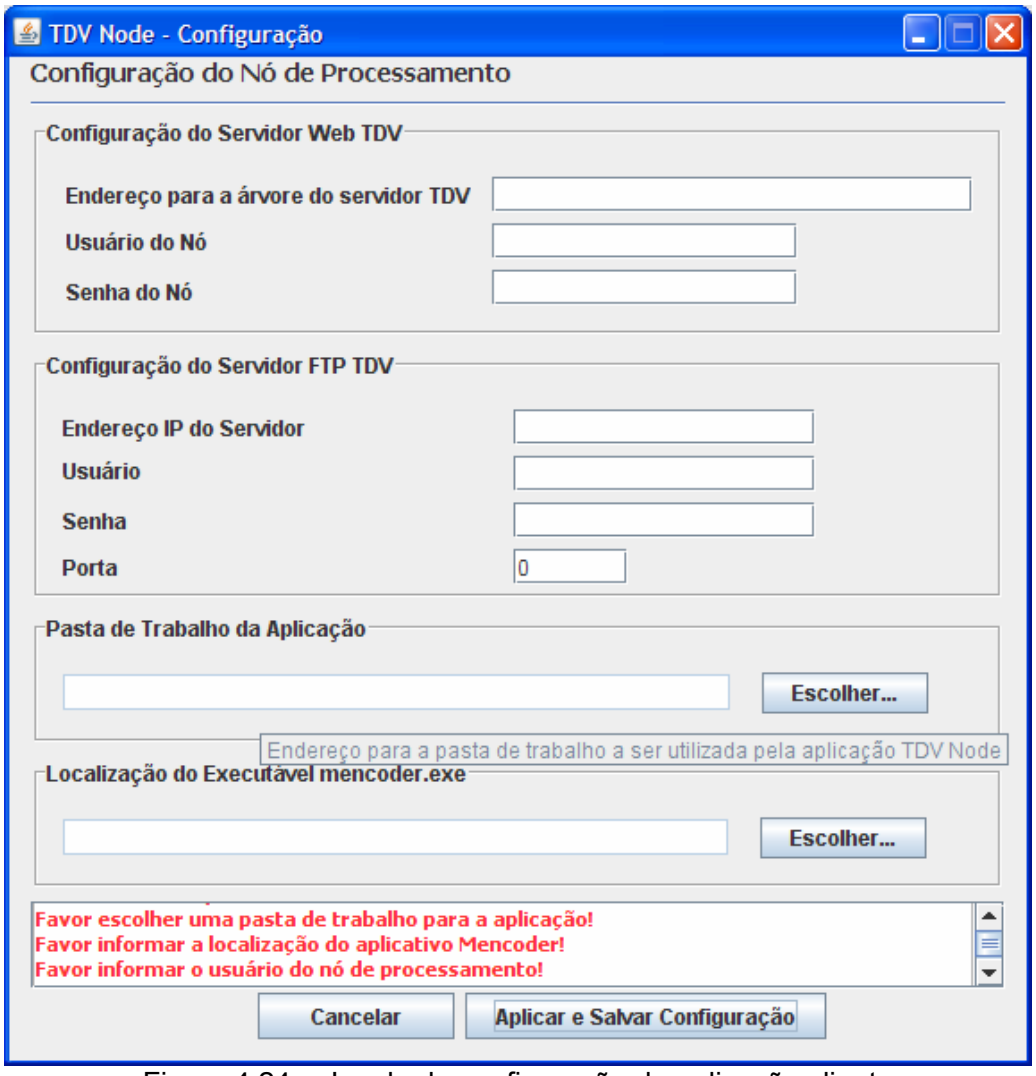

Figura 4.24 – Janela de configuração da aplicação cliente

Ao clicar no botão “Escolher” é aberta uma tela de seleção de pasta ou arquivo, exibida na figura 4.25, conforme a opção, que é renderizada pela classe utilitária *FileChooser*.

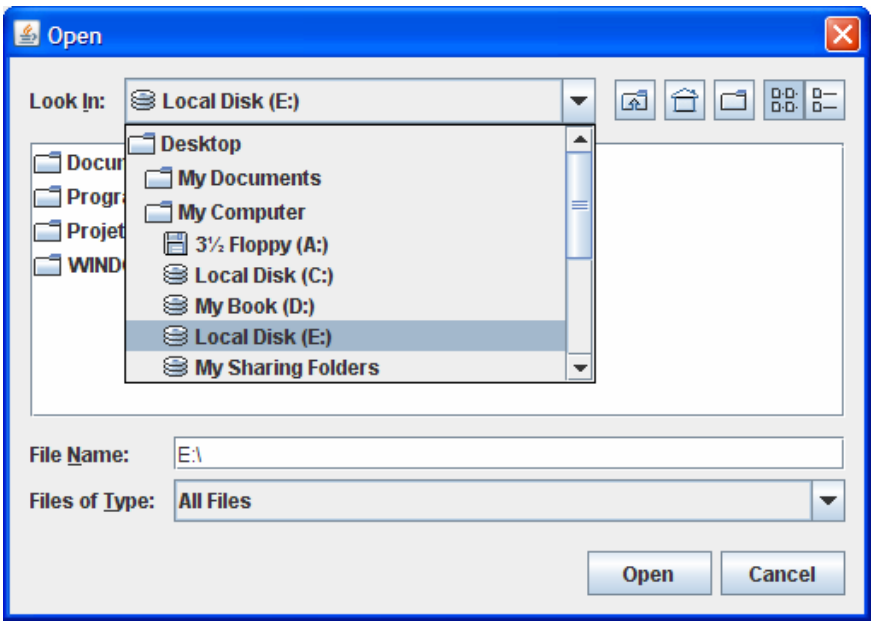

Figura 4.25 – Janela de seleção aberta

Como forma de facilitar o controle, a classe controladora *ConfigurationController* é instanciada pela classe *MainController*. Todo o controle da janela de configuração é efetuado apenas pela *ConfigurationController*. A figura 4.26 exibe as principais classes utilizadas durante a exibição e manipulação da janela de configuração.

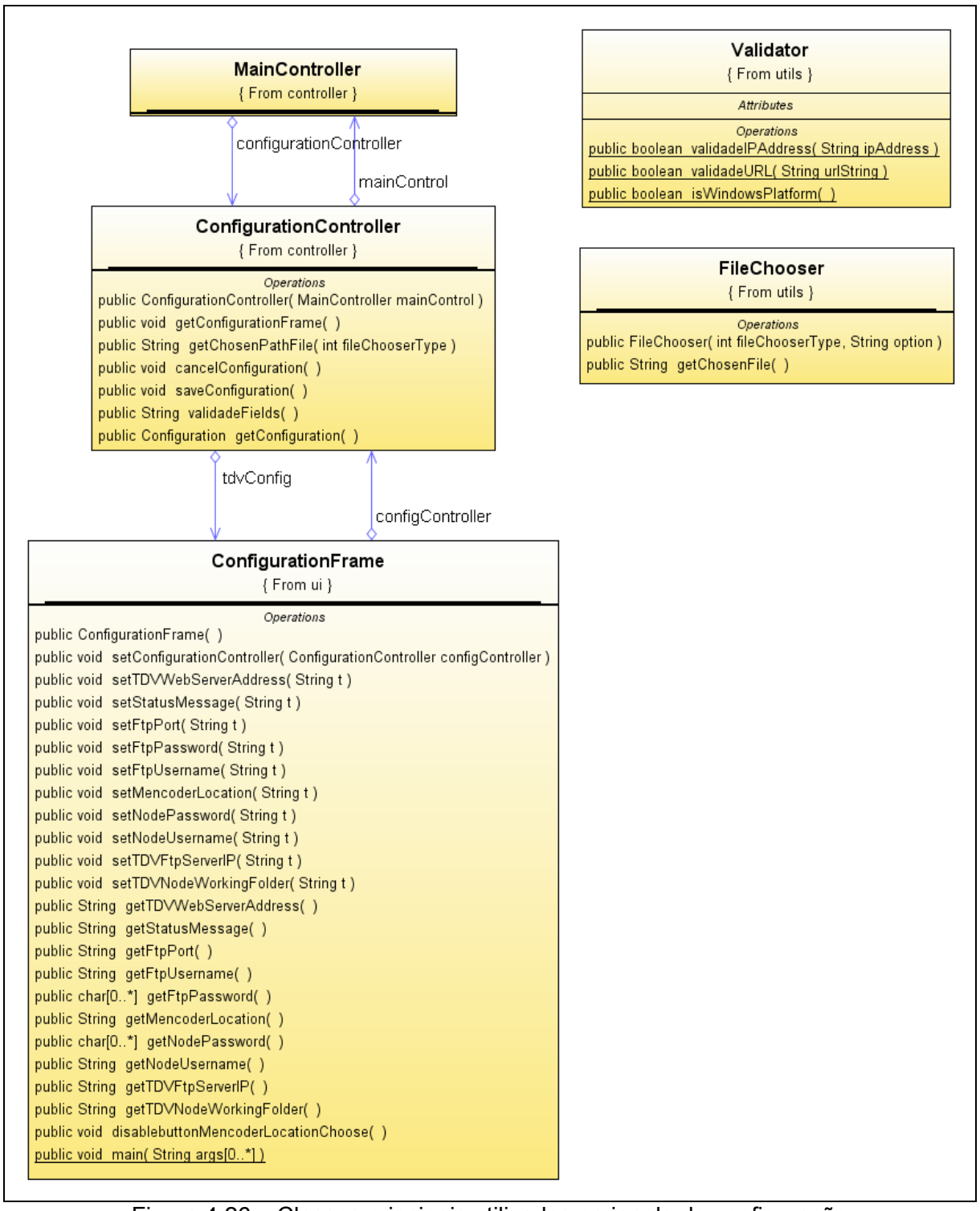

Figura 4.26 – Classes principais utilizadas na janela de configuração

4.2.5.2.4.2 Janela sobre a Aplicação

A janela de informações sobre a aplicação exibe o nome e registro acadêmico do autor deste projeto e o nome do professor orientador, indicando a destinação da aplicação. Esta janela é renderizada pela classe *AboutFrame* que é instanciada diretamente pela classe controladora *MainController*.

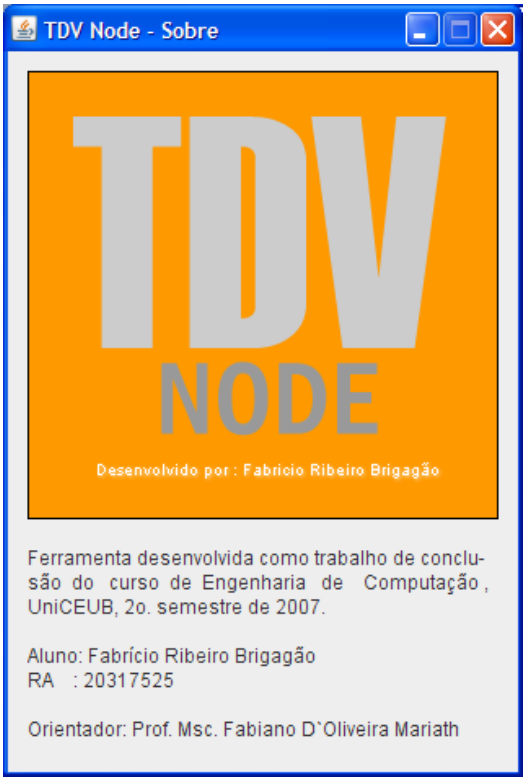

Figura 4.27 – Janela de informações sobre a aplicação

4.2.6 Segurança e Tolerância à Falhas

Alguns mecanismos de segurança foram adotados de forma a tornar a ferramenta mais segura e tolerante a falhas. Estes mecanismos são descritos sucintamente abaixo:

- Utilização de usuários independentes para o servidor *web* Apache, banco de dados *MySQL* e servidor *VSFTPD*, o que limita o acesso dos mesmos as demais pastas e arquivos do sistema operacional;

- Banco de dados *MySQL* configurado para aceitar apenas acesso local, ou seja, conexões provenientes da rede não serão aceitas pelo banco de dados, garantindo que apenas os *scripts* e servidores localizados no mesmo equipamento do mesmo possam efetuar acessos e pesquisas;

- Utilização de algoritmo SHA-256[46] (*Secure Hash Algorithm)* de 256 *bits* para a guarda de nomes de usuários e senhas em banco de dados. Este algoritmo foi

[46] Padrão desenvolvido pelo NIST (*National Institute of Standards and Technology*), dos Estados Unidos. Mais informações sobre o padrão podem ser obtidas através do *website* <http://www.itl.nist.gov/fipspubs/fip180-1.htm>

escolhido em detrimento do MD5[47], de 128 *bits*, e do SHA-1[48], de 160 *bits*, por possuir um maior número de combinações o que resulta em um maior esforço computacional para a decifração dos dados codificados. O algoritmo SHA-256 também foi utilizado para a geração de somas de verificação, também chamadas de *checksums*, para os arquivos enviados e recebidos pelo servidor. O servidor e os nós de processamento verificam estes *checksums* como forma de garantir que os arquivos de vídeo transferidos não foram corrompidos durante o processo de transferência;

- O servidor dinamicamente realoca pedaços que estão há algum tempo sem atualização. Os nós de processamento envolvidos, caso retornem ao funcionamento, são informados que não existem mais pedaços alocados para os mesmos e através desta informação solicitam novos pedaços;

- Alguns procedimentos como o recebimento de arquivos e envio de arquivos estão sujeitos a falhas. Com base nesta premissa, os nós de processamento efetuam novas tentativas casos um erro seja detectado. Após a terceira tentativa os nós de processamento informam o erro ao servidor, que desaloca os pedaços vinculados aos mesmos;

- A conexão dos nós de processamento ao servidor FTP é feita através da utilização de usuário e senha. Sendo assim, acessos não autorizados são evitados;

- Os nós de processamento efetuam autenticação junto ao servidor, utilizando usuário e senha próprios, antes de prosseguir com os demais procedimentos;

- Ao autenticar junto ao servidor, é gerado uma variável de sessão para cada nó de processamento, utilizando algoritmo SHA-1, esta variável de sessão é utilizada durante o resto do processo e sempre é requerida pelo servidor. Caso a mesma esteja ausente o servidor retorna um erro de autenticação e o nó de processamento deve efetuar uma nova autenticação. Ao desconectar do servidor, a variável de sessão vinculada ao nó de processamento é destruída;

---

[47] Padrão desenvolvido por Ronald Rivest em 1991. Mais informações sobre o mesmo podem ser obtidas através da RFC 1321 disponível em <http://tools.ietf.org/html/rfc1321>

[48] Padrão desenvolvido pelo NIST (*National Institute of Standards and Technology*), dos Estados Unidos. Mais informações sobre o padrão podem ser obtidas através do *website* <http://www.itl.nist.gov/fipspubs/fip180-1.htm>

- A aplicação utilizada pelos nós de processamento não efetua a guarda das senhas do usuário do nó de processamento e do servidor FTP. As mesmas são armazenadas apenas em memória durante o período de execução da ferramenta;

- O servidor controla a ocorrência de erros no processamento de um pedaço vinculado a um nó de processamento, desta forma, pedaços que já apresentaram problemas com alguns nós de processamento não são realocados para os mesmos;

- O acesso à interface *web* de gerenciamento é feito através de verificação de usuário e senha do administrador, que são armazenados de forma codificada em banco de dados.

4.3 Considerações Finais

Esta seção contém as considerações finais do desenvolvimento do projeto, subdivididas em dificuldades encontradas, resultados obtidos e limitações.

4.3.1 Dificuldades Encontradas

Foram encontradas diversas dificuldades neste projeto principalmente devido ao fato do mesmo utilizar diversas tecnologias que tiverem que ser ajustadas para trabalhar em conjunto.

Uma das grandes dificuldades foi a necessidade de aprender e configurar diversos servidores, como os servidores *web*, banco de dados e servidor FTP em ambiente *Linux*. O processo de configuração exigiu tempo e pesquisa para que o servidor funcionasse plenamente.

A segunda grande dificuldade foi a necessidade de aprendizado de diversas tecnologias como a programação em *shell script*, a qual não era de conhecimento do autor até este projeto, e que foi utilizada para a geração de dois *scripts* altamente críticos. A pesquisa das outras linguagens também exigiu tempo, embora o autor possuísse contato prévio com as linguagens Java e PHP. Especificamente na linguagem PHP, foi necessário estudar como um *script* PHP poderia transferir um arquivo com base em uma requisição sendo que antes desta transferência o mesmo deveria atualizar a base de dados.

A busca por ferramentas e bibliotecas também foi uma dificuldade inerente ao projeto, as bibliotecas escolhidas só foram utilizadas após pesquisa e estudo de como implementar as

funcionalidades necessárias com as mesmas, a citar, o cliente HTTP e o cliente FTP. Localizar uma versão da ferramenta *Mencoder* com as bibliotecas necessárias compilada para o ambiente *Windows* também foi uma dificuldade, pois poucos usuários utilizam a mesma neste ambiente.

A definição dos parâmetros de vídeo a serem utilizados e a modelagem da base para permitir o armazenamento destes parâmetros de forma simples e dinâmica também foi uma dificuldade, mas a solução final atendeu o projeto.

Durante o desenvolvimento e a realização de testes foram encontradas algumas limitações da ferramenta, a citar: defasagem no sincronismo entre o áudio e vídeo no arquivo unificado final e limitações do processo de divisão.

Ao unificar os pedaços de vídeo transcodificados foi verificado que ocorria uma defasagem de sincronismo entre a imagem obtida e o áudio do vídeo. Como forma de tentar solucionar o problema, foram efetuados testes com a divisão e envio apenas da imagem para transcodificação, sendo que o áudio era extraído diretamente do arquivo original pelo servidor e incluído no arquivo final em conjunto com a imagem. Esta segunda abordagem resultou em uma total falta de sincronismo entre o vídeo e o áudio obtido, sendo descartada.

A análise deste problema apontou como causa o processo de divisão do arquivo original, que pode utilizar o tempo como fator determinante para a divisão ou a posição em número de quadros (*frames*). A primeira opção gera descarte de quadros (*frames*) que são exibidos entre dois segundos tendo como vantagem a pequena alteração na imagem final mas possuindo a desvantagem de descartar parte do áudio nos pontos de junção. A segunda opção, utilizando o número de quadros (*frames*) como fator de divisão, gera defasagem ou repete parte do vídeo e áudio, tendo em vista que diversos quadros são exibidos em um mesmo segundo, além de exigir um controle mais complexo e possuir pouco suporte por parte das ferramentas utilizadas. Em conjunto com esta primeira causa, foi verificado que o suporte limitado da ferramenta *Mencoder* ao *container* MP4 estava resultando na geração de arquivos problemáticos, contribuindo para o aumento da defasagem.

Para solucionar o problema acima uma nova abordagem dos processos de divisão e unificação foi utilizada. O processo de divisão baseado em tempo foi alterado para utilizar a ferramenta *avisplit*, descrita anteriormente, que permite a divisão de vídeos que utilizam o *container* AVI em pedaços de acordo com o tamanho determinado. A utilização desta ferramenta diminuiu drasticamente os problemas com o corte de áudio e vídeo nos pontos de

divisão contribuindo para aumentar a fidelidade do vídeo transcodificado em relação ao original.

Em conjunto com a ação acima, o processo de unificação foi alterado para extrair as faixas de áudio e vídeo dos arquivos transcodificados, que passaram a utilizar o *container AVI*, para a geração do arquivo final que utiliza *container* MP4. A extração das faixas de áudio e vídeo com sua posterior unificação em arquivos MP4 diminui significativamente a defasagem existente entre o áudio e o vídeo do arquivo final transcodificado, pois contornou a limitação da ferramenta *Mencoder*, a citar, o suporte limitado à codificação direta com o armazenamento dos dados em *container* MP4. O armazenamento no novo *container* passou a ser efetuado pelas ferramentas *mp4creator* e *MP4Box*, ambas citadas anteriormente.

Enfim, o projeto, devido ao grande número de tecnologias utilizadas, foi de grande dificuldade e consumiu bastante tempo na pesquisa, configuração, desenvolvimento e integração de todas elas, o que resultou em um grande aprendizado.

4.3.2 Resultados Obtidos

A ferramenta atingiu o seu objetivo principal, a citar, possibilitar a transcodificação distribuída de arquivos de vídeo. É importante ressaltar que este objetivo foi atingido através de diversas tecnologias diferentes e todas de código aberto, o que permite a livre alteração do código, contribuindo para implementações futuras.

A utilização de sistema operacional *Linux* contribuiu significativamente para a redução do tempo de resposta do servidor, que é capaz de processar uma requisição rapidamente, podendo ser acessado por diversos nós de processamento simultaneamente. O processo de divisão de vídeo, programado em *shell script*, demonstrou ser extremamente rápido, sendo capaz de dividir um arquivo de vídeo de 700 MB em apenas um minuto, efetuando a gravação de registro em base de dados e calculando *checksum* de cada pedaço gerado. Este processo demonstrou a velocidade de programas escritos em *shell script*, influenciada principalmente pelo uso de *pipes*, que permitem o encadeamento de comandos em ambiente *Linux*. A velocidade dos processos de divisão e unificação é crítica para diminuir o *overhead* da utilização de processamento distribuído.

As atividades de configuração dos servidores, modelagem da base de dados, desenvolvimento dos *scripts* do servidor bem como a aplicação utilizada pelos nós de processamento também foram realizadas com sucesso.

Para verificar se houve redução de tempo de transcodificação através da utilização da ferramenta criada foram efetuados testes, sendo que durante os mesmos foram utilizados dois arquivos de vídeo com boa qualidade de áudio e imagem e que utilizavam o padrão de compressão de vídeo MPEG-4 Parte 2 e padrão de compressão de áudio MP3, possuindo as características citadas na tabela 4.28.

Tabela 4.28 – Características principais dos arquivos de teste

Característica	**Arquivo 1**	**Arquivo 2**
Tamanho	78.1 MB	68.4 MB
Padrão de Compressão de Vídeo	MPEG-4 Parte 2	MPEG-4 Parte 2
Codec de Vídeo Utilizado	XVID	XVID
Bitrate do Vídeo	2560.1 kbps	2078.7 kbps
Quadros por segundo (*fps*)	25	23.976
Padrão de Compressão de Áudio	MPEG-1 *Layer* III (MP3)	MPEG-1 Layer III (MP3)
Codec de Áudio Utilizado	Não disponível	Não disponível
Bitrate do Áudio	192 kbps	224 kbps
Container	AVI	AVI
Duração	3 minutos e 56 segundos	04 minutos e 07 segundos

Ambos os arquivos foram codificados através da utilização dos parâmetros listados na tabela 4.29.

Tabela 4.29 – Parâmetros de codificação utilizados durante os testes

Característica	**Arquivo 1**	**Arquivo 2**
Padrão de Compressão de Vídeo	MPEG-4 Parte 10 (H.264)	MPEG-4 Parte 10 (H.264)
Bitrate do Vídeo	700 kbps	700 kbps
Quadros por segundo (*fps*)	25	25
Padrão de Compressão de Áudio	MPEG-4 Parte 3 (AAC)	MPEG-4 Parte 3 (AAC)
Bitrate do Áudio	96 kbps	96 kbps
Container	MP4	MP4
Dimensões do Vídeo	640 *pixels* x 480 *pixels*	640 *pixels* x 480 *pixels*

Os testes foram realizados em laboratório com cinco nós de processamento, conforme citado no item 4.1, e os resultados obtidos foram similares em todas as simulações realizadas. A tabela 4.30 lista o resultado de uma das simulações.

Tabela 4.30 – Resultado dos testes utilizando processamento distribuído

Descrição	Arquivo 1	Arquivo 2
Data e Hora do Cadastramento da Solicitação	22.11.2007 – 19:25:31	22.11.2007 – 20:38:14
Data e Hora do Encerramento da Solicitação	22.11.2007 – 19:32:08	22.11.2007 – 20:45:06
Tempo Total Decorrido	6 minutos e 37 segundos	6 minutos e 52 segundos

Os valores obtidos com os testes efetuados em laboratório foram comparados aos valores obtidos quando o mesmo processo de transcodificação era efetuado diretamente pelo servidor com a utilização de duas ou apenas uma *thread* de processamento. Os dados obtidos nestes testes estão listados nas tabelas 4.31 e 4.32.

Tabela 4.31 – Resultado dos testes utilizando apenas o servidor com 2 *threads*

Descrição	Arquivo 1	Arquivo 2
Data e Hora do Início	23.11.2007 – 21:23:13	23.11.2007 – 21:57:10
Data e Hora do Encerramento	23.11.2007 – 21:29:48	23.11.2007 – 22:03:27
Tempo Total Decorrido	6 minutos e 35 segundos	6 minutos e 17 segundos

Tabela 4.32 – Resultado dos testes utilizando apenas o servidor com 1 thread

Descrição	Arquivo 1	Arquivo 2
Data e Hora do Início	23.11.2007 – 21:34:44	22.11.2007 – 22:04:20
Data e Hora do Encerramento	23.11.2007 – 21:43:53	22.11.2007 – 22:13:30
Tempo Total Decorrido	9 minutos e 9 segundos	9 minutos e 10 segundos

Os dados acima demonstram que o servidor foi mais rápido do que o sistema distribuído apenas quando eram utilizadas 2 *threads* para efetuar o processo de transcodificação.

Alguns fatores contribuíram para elevar o tempo necessário para concluir a solicitação no sistema distribuído:

- O *script* de divisão é acionado de minuto a minuto, logo, o arquivo 1 aguardou, no mínimo, 29 segundos para que o processo iniciasse, o que contribuiu para o aumento do tempo total. O arquivo 2 também aguardou, no mínimo, 46 segundos para ser dividido, o que representa um aumento considerável no tempo total de processamento da solicitação;

- O *script* de unificação também é acionado de minuto a minuto, logo, é possível que uma solicitação encerre a transcodificação no minuto anterior e tenha que aguardar a próxima execução do processo de unificação para ser concluída;
- A aplicação residente nos nós de processamento verifica o servidor a cada dez segundos por pedaços disponíveis para transcodificação, logo, durante este intervalo a solicitação fica ociosa;

- A comunicação entre os nós de processamento no laboratório utilizado para os testes é efetuada através da utilização de *hub* o que pode ter impactado o desempenho do sistema tendo em vista que o tráfego de dados não é direcionado ao computador de destino, como ocorre em um *switch*, mas enviado a todos os computadores.

Com base nos três primeiros itens citados anteriormente e com o auxílio dos dados disponíveis no banco de dados da aplicação e nos *logs* dos processos de divisão e unificação, pode-se refazer a tabela 4.30 gerando a tabela 4.33 que desconsidera o tempo ocioso gasto pelo servidor ao aguardar o início dos processos de divisão e unificação.

Tabela 4.33 – Resultado dos testes utilizando processamento distribuído

Descrição	**Arquivo 1**	**Arquivo 2**
Data e Hora do Cadastramento da Solicitação	22.11.2007 – 19:25:31	22.11.2007 – 20:38:14
Data e Hora do Início do Processo de Divisão	22.11.2007 – 19:26:01	22.11.2007 – 20:39:01
Data e Hora do Término do Processo de Divisão	22.11.2007 – 19:26:11	22.11.2007 – 20:39:07
Data e Hora do Recebimento do Último Pedaço Codificado	22.11.2007 – 19:31:04	22.11.2007 – 20:44:21
Data e Hora do Início do Processo de Unificação	22.11.2007 – 19:32:01	22.11.2007 – 20:45:01
Data e Hora do Término do Processo de Unificação	22.11.2007 – 19:32:09	22.11.2007 – 20:45:07
Data e Hora do Encerramento da Solicitação	22.11.2007 – 19:32:08	22.11.2007 – 20:45:06
Tempo Total Decorrido	6 minutos e 37 segundos	6 minutos e 52 segundos
Tempo Ocioso Aguardando o	30 segundos	47 segundos

Processo de Divisão		
Tempo Ocioso Aguardando o Processo de Unificação	57 segundos	40 segundos
Tempo Total Aguardando Processos	1 minuto e 27 segundos	1 minuto e 27 segundos
Tempo Real de Processamento da Solicitação	5 minutos e 10 segundos	5 minutos e 25 segundos

Com base na tabela 4.33 pode-se concluir que o tempo real de processamento do sistema distribuído, o qual ainda inclui o tempo gasto com a transferência de arquivos e o processamento de requisições, é menor do que o tempo total gasto pelo servidor em ambos os casos. A tabela 4.34 demonstra os valores em percentuais.

Tabela 4.34 – Diferença em percentuais entre o servidor e o sistema distribuído

Descrição	**Arquivo 1**	**Arquivo 2**
Servidor com 2 *threads*	6 minutos e 35 segundos	6 minutos e 17 segundos
Sistema Distribuído *com* Tempo Ocioso	6 minutos e 37 segundos	6 minutos e 52 segundos
Sistema Vencedor : % de Tempo Economizado	Servidor : 0,50% de ganho	Servidor : 8,50% de ganho
Servidor com 2 *threads*	6 minutos e 35 segundos	6 minutos e 17 segundos
Sistema Distribuído *sem* Tempo Ocioso	5 minutos e 10 segundos	5 minutos e 25 segundos
Sistema Vencedor : % de Tempo Economizado	Sistema Distribuído : 21,52% de ganho	Sistema Distribuído : 13,79% de ganho
Servidor com 1 *thread*	9 minutos e 9 segundos	9 minutos e 10 segundos
Sistema Distribuído *com* Tempo Ocioso	6 minutos e 37 segundos	6 minutos e 52 segundos
Sistema Vencedor : % de Tempo Economizado	Sistema Distribuído : 27,69% de ganho	Sistema Distribuído : 25,09% de ganho
Servidor com 1 *thread*	9 minutos e 9 segundos	9 minutos e 10 segundos
Sistema Distribuído *sem* Tempo Ocioso	5 minutos e 10 segundos	5 minutos e 25 segundos
Sistema Vencedor : % de Tempo Economizado	Sistema Distribuído : 43,53% de ganho	Sistema Distribuído : 40,91% de ganho

Algumas medidas podem ser adotadas visando diminuir o tempo ocioso do servidor utilizado neste projeto, as vantagens e desvantagens de cada medida são listadas na tabela 4.35.

Tabela 4.35 – Possíveis medidas para a redução do tempo ocioso do servidor

Medida	Vantagens	Desvantagens
Diminuição do intervalo de tempo de execução dos *scripts* de divisão e unificação	Diminuição do tempo ocioso da solicitação, tornando os pedaços disponíveis para alocação em um menor espaço de tempo.	Pode ocasionar sobrecarga do servidor, pois resultará em um número maior de requisições ao mesmo.
Diminuição do intervalo de tempo que a aplicação residente nos nós de processamento aguarda antes de enviar uma nova solicitação de pedaço para transcodificação	Diminuição do tempo ocioso da solicitação e dos nós de processamento, contribuindo para reduzir o tempo total gasto para o processamento da solicitação.	Pode ocasionar sobrecarga do servidor e resultar em falhas nos nós de comunicação, tendo em vista o aumento do número de requisições.
Substituição do *hub* do laboratório por *switch*	O tráfego de dados será direcionado apenas para o equipamento de destino, evitando a redundância e melhorando o desempenho dos processos de transferência de arquivos e dados.	Necessidade de compra do *switch* caso não exista nenhum disponível.

5 CONCLUSÃO

A crescente demanda por serviços de transcodificação de vídeo digital, conforme citado na introdução deste trabalho, pode usufruir dos benefícios da utilização do processamento distribuído como forma de viabilizar a realização do processo de transcodificação através da utilização de computadores existentes nas empresas e nas residências de usuários domésticos.

A ferramenta desenvolvida demonstrou ser capaz de processar as solicitações de transcodificação de forma satisfatória, gerando arquivos transcodificados onde são praticamente imperceptíveis os pontos de unificação, ou seja, com um alto grau de fidelidade quando comparado ao arquivo original.

As ferramentas comerciais existentes atualmente utilizam *hardware* e *software* proprietários o que as tornam inacessíveis à grande maioria das empresas e usuários além de prejudicar significativamente a mobilidade dos recursos. A abordagem utilizada no projeto favorece a mobilidade à medida que são necessários apenas o equipamento servidor e a aplicação cliente, a qual é executável tanto em sistemas operacionais Windows como Linux.

A utilização da linguagem de programação Java aliada à utilização de ferramentas de código aberto reduz a zero os custos com a aquisição de *software* proprietário e torna a aplicação cliente portável para diversos sistemas operacionais, permitindo a sua utilização em redes heterogêneas.

As evoluções deste trabalho, citadas no item 5.1, se implementadas, tendem a tornar a ferramenta mais flexível, robusta e rápida, favorecendo a sua adoção em detrimento da aquisição de ferramentas proprietárias para necessidades de pequeno e médio porte.

5.1 Sugestões de Trabalhos Futuros

O projeto desenvolvido abre um leque interessante de linhas de pesquisa e soluções que podem ser criadas com base no trabalho aqui apresentado. As sugestões estão divididas em dois grupos: evoluções do trabalho atual e linhas de pesquisa alternativas.

- Sugestões de evolução do trabalho atual
    - Inclusão de um número maior de opções de transcodificação e *codecs*, tornando a ferramenta mais versátil;

- Solução das limitações descritas visando a redução do tempo de ociosidade da ferramenta;

- Desenvolvimento das interfaces *web* de cadastramento de administradores, cadastramento de *codecs*, cadastramento de parâmetros e cadastramento de opções, permitindo maior interatividade do usuário com a aplicação e facilitando a adaptação da mesma;

- Utilização de divisão de vídeo com base em tamanho de arquivo adaptativo, ou seja, o nó de processamento informa o tamanho do arquivo que deseja receber e o servidor dinamicamente gera e disponibiliza o arquivo com o tamanho solicitado;

- Inclusão de percentuais de *download,* transcodificação e *upload* na base de dados com o ajuste da aplicação cliente e do *scripts* do servidor, melhorando o nível de detalhamento dos relatórios de andamento de solicitação;

- Adaptação da aplicação cliente para que a mesma fique em *stand-by* e entre em funcionamento apenas quando detectado que o nó de processamento está ocioso;

- Modelagem e inclusão de tabela de histórico de processamento de uma solicitação e os seus pedaços vinculados, possibilitando o registro de cada transação efetuada bem como os códigos, parâmetros e mensagens enviadas e recebidas;

- Criação de *scripts* de apoio vinculados ao servidor que dinamicamente criem usuários independentes para o servidor FTP sempre que um novo nó de processamento for cadastrado, possibilitando melhor controle sobre as atividades dos nós de processamento;

- Avaliação da possibilidade de utilização de um canal de comunicação seguro entre o servidor e os nós de processamento, contribuindo para a melhoria da segurança dos dados trafegados;

- Desenvolvimento de biblioteca em C ou C++ encapsulando as opções disponíveis na ferramenta *Mencoder* possibilitando uma integração maior com a aplicação cliente desenvolvida em Java através da utilização da tecnologia JNI[49] (*Java Native Interface*);

[49] Para mais informações sobre a tecnologia JNI visite <http://java.sun.com/j2se/1.4.2/docs/guide/jni/>

- Avaliação e abordagem da possibilidade de utilização de espelhamento do servidor *web* utilizando um servidor de arquivos comum a todos como forma de aumentar a escalabilidade da ferramenta desenvolvida;
- Avaliação e implementação de mecanismos alternativos de comunicação entre o servidor e os nós de processamento com o objetivo de torná-la mais eficiente.

- Linhas de pesquisa alternativas:
  - Substituição da arquitetura utilizada por uma arquitetura orientada a serviços, permitindo a alocação dinâmica de diversos tipos de processos aos nós de processamento;
  - Utilização do modelo de comunicação *peer-to-peer (P2P)* possibilitando o cadastramento de solicitações de transcodificação de forma descentralizada, permitindo que cada nó de processamento se torne cliente e servidor ao mesmo tempo;
  - Abordagem da codificação de vídeo em tempo real para a digitalização de novas produções, criando uma ferramenta utilizável em emissoras de televisão;
  - Criação de um sistema de tarifação de processamento, onde o usuário informa o processo a ser realizado e paga pelo uso dos nós de processamento, contribuindo para a criação de uma nova forma de negócio;
  - Elaboração de um servidor dedicado ao processamento distribuído de vídeo, retirando funcionalidades do *kernel* que não são inerentes ao processo e otimizando configurações de forma a torná-lo o mais eficiente possível.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ADVANCED Audio Coding. *Hydrogenaudio Knowledgebase*. Disponível em: <http://wiki.hydrogenaudio.org/index.php?title=AAC>. Acesso em: 01 nov. 2007.

ADVANCED Audio Coding. *Wikipedia*. Disponível em: <http://en.wikipedia.org/wiki/Advanced_Audio_Coding>. Acesso em: 01 nov. 2007.

ALVES, M. B. M.; ARRUDA, S. M. *Como Fazer Referências*: Bibliográficas, Eletrônicas e demais formas de documentos. Disponível em: <http://www.bu.ufsc.br/framerefer.html>. Acesso em: 16 nov. 2007.

ANDERSON, D. *Apache 2 and PHP 5 (mod_php) on Linux*. Disponível em: <http://dan.drydog.com/apache2php.html>. Acesso em: 02 out. 2007.

APACHE SOFTWARE FOUNDATION. *Commons Configuration*: Configuration Overview. Disponível em: <http://commons.apache.org/configuration/userguide-1.2/overview.html>. Acesso em: 27 out. 2007.

_____. *HttpClient Preference Architecture and Configuration Guide*. Disponível em: <http://jakarta.apache.org/httpcomponents/httpclient-3.x/preference-api.html>. Acesso em: 20 out. 2007.

BHOGAL, K. S. *Minding the Queue*: Java 1.5 Adds a New Data Structure Interface. Disponível em: <http://www.devx.com/Java/Article/21983>. Acesso em: 21 out. 2007.

BRANDENBURG, K. MP3 and AAC Explained. *AES INTERNATIONAL CONFERENCE ON HIGH QUALITY AUDIO CODING*, 17., 1999. Disponível em: <http://www.telos-systems.com/techtalk/hosted/Brandenburg_mp3_aac.pdf>. Acesso em: 01 nov. 2007.

COMER, D. E. *Interligação de Redes com TCP/IP*. 5. ed. Rio de Janeiro : Elsevier, 2006. v.1.

CONVERSE, T.; PARK, J. *PHP 4 A Bíblia*. Rio de Janeiro : Campus, 2001.

CRANE, D.; PASCARELLO, E.; JAMES, D. *Ajax in Action*. Greenwich : Manning Publications Co., 2006.

FERREIRA R. E. *Linux*: Guia do Administrador do Sistema. São Paulo l: Novatec Editora Ltda., 2003.

FOR Loop. *Linux Shell Scripting Tutorial (LSST) v1.05r3*. Disponível em: <http://www.freeos.com/guides/lsst/ch03sec06.html/>. Acesso em: 18 out. 2007.

GHANBARI, M. *Standard Codecs*: Image Compression to Advanced Video Coding. London : The Institution of Electrical Engineers, 2003.

GIRELLO, M. *Manual de Orientação na Elaboração de Referências*. Disponível em: <http://biblioteca.fop.unicamp.br/ManualSimplificado1.pdf>. Acesso em: 16 nov. 2007.

IEEE. *IEEE Standard Computer Dictionary*: A Compilation of IEEE Standard Computer Glossaries. New York : Institute of Electrical and Electronics Engineers, 1991.

INTRO to Cron. *UnixGeeks.org*, 30 dez. 1999. Disponível em <http://www.unixgeeks.org/security/newbie/unix/cron-1.html>. Acesso em: 19 out. 2007.

JIA, W.; ZHOU, W. *Distributed Network Systems*: From Concepts to Implementations. Boston : Springer Science + Business Media Inc., 2005.

LAFFERS, R. *How to Install Apache, PHP And MySQL On Linux*. Disponível em: <http://laffers.net/howtos/howto-install-mysql?scrib_howto_install_mysql_start=1>. Acesso em: 02 out. 2007.

MICROSOFT CORPORATION. Microsoft Developer Network. *AVI RIFF File Reference*. Disponível em: <http://msdn2.microsoft.com/en-us/library/ms779636.aspx>. Acesso em: 09 set. 2007.

MOTA, S. *Usando o pattern Observable*. Disponível em: <http://www.guj.com.br/java.tutorial.artigo.47.1.guj>. Acesso em: 25 out. 2007.

MOVING PICTURES EXPERTS GROUP. *MPEG-4 Overview*. Disponível em: <http://www.m4if.org/resources/Overview.pdf>. Acesso em: 12 set. 2007.

MP3. *Hydrogenaudio Knowledgebase*. Disponível em: <http://wiki.hydrogenaudio.org/index.php?title=MP3>. Acesso em 01 nov. 2007.

MP3. *Wikipedia*. Disponível em: <http://en.wikipedia.org/wiki/MP3>. Acesso em: 01 nov. 2007.

NEVES, J. C. *Programação Shell Linux*. 5. ed. Rio de Janeiro : Brasport, 2005.

NORGUET, J. *Java FTP Client Libraries Reviewed*. Disponível em: <http://www.javaworld.com/javaworld/jw-04-2003/jw-0404-ftp.html>. Acesso em: 17 out. 2007.

_____. *Update: Java FTP Libraries Benchmarked*. Disponível em: <http://www.javaworld.com/javaworld/jw-03-2006/jw-0306-ftp.html>. Acesso em: 17 out. 2007.

PETERS, M. *Securing Apache.* Disponível em: <http://www.linux.com/feature/113744>. Acesso em: 02 nov. 2007.

QUICK Tour Round Java 5. *Clan Productions*. Disponível em: <http://clanproductions.com/java5.html/>. Acesso em: 22 out. 2007.

REFRESHING Page Without Reload Using Ajax and PHP. *CodingForums.com*, 03 mar. 2007. Disponível em: <http://www.codingforums.com/showthread.php?t=108877>. Acesso em: 02 nov. 2007.

RICHARDSON, I. *H.264 and MPEG-4 Video Compression*: Video Coding for Next-Generation Multimedia. Chichester : John Wiley & Sons Ltd., 2003.

ROBICHAUX, J. *Java FTP Tips*. Disponível em: <http://www.nsftools.com/tips/JavaFtp.htm>. Acesso em: 17 out. 2007.

RUSSINOVICH, M. *PsKill v.1.12*. Disponível em: <http://www.microsoft.com/technet/sysinternals/utilities/pskill.mspx>. Acesso em: 30 out. 2007.

SETTING up a PHP 5 with Apache 2 and MySQL 4.1.3. *Builder AU*. Disponível em: <http://www.builderau.com.au/program/linux/soa/Setting-up-a-PHP-5-with-Apache-2-and-MySQL-4-1-3/0,339028299,339130604,00.htm>. Acesso em: 02 out. 2007.

SHELL Scripting: Creating report/log file names with date in filename. *NixCraft*, 05 fev. 2006. Disponível em: <http://www.cyberciti.biz/tips/shell-scripting-creating-reportlog-file-names-with-date-in-filename.html>. Acesso em: 17 out. 2007.

SIERRA, K.; BATES, B. *Sun Certified Programmer for Java 5 Study Guide*. EmeryVille : McGraw-Hill/Osborne, 2006.

SUN MICROSYSTEMS. *Set Implementations*. Disponível em: <http://java.sun.com/docs/books/tutorial/collections/implementations/set.html>. Acesso em: 21 out. 2007.

SYLVESTER, I. Transcoding: The Future of the Video Market Depends on It. *IDC Executive Brief,* Nov. 2006. Disponível em: <https://focus.ti.com/seclit/ml/sprl094/sprl094.pdf>. Acesso em: 02 set. 2007.

TANENBAUM, A. S. *Redes de Computadores*. 4. ed. Rio de Janeiro : Elsevier, 2003.

TANENBAUM, A. S.; STEEN, M. *Distributed Systems*: Principles and Paradigms. New Jersey : Prentice-Hall Inc., 2002.

TAYLOR, D. *How can my Shell script test to see if it`s already running?*. Disponível em: <http://www.askdavetaylor.com/shell_script_test_to_see_if_its_already_running.html/>. Acesso em: 17 out. 2007.

TEXAS INSTRUMENTS. *Evolution in Video Entertainment Will Cease Without Multi-Format Transcoding*. Disponível em: <http://www.ti.com/corp/docs/landing/transcoding/index.htm>. Acesso em: 02 set. 2007.

WIEGAND, T. *et al.* Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 13, n. 7, p. 560-561, Jul. 2003. Disponível em: <http://ip.hhi.de/imagecom_G1/assets/pdfs/csvt_overview_0305.pdf>. Acesso em: 06 set. 2007.

WONG, Y.; BURG, J.; MCCOY, L. *Integrated Digital Media Curriculum Development*. Department of Computer Science and Department of Art – Wake Forest University. Disponível em: <http://digitalmedia.wfu.edu/project/digital-media-curriculum-development>. Acesso em: 06 set. 2007.

APÊNDICE A – TELAS DA INTERFACE WEB

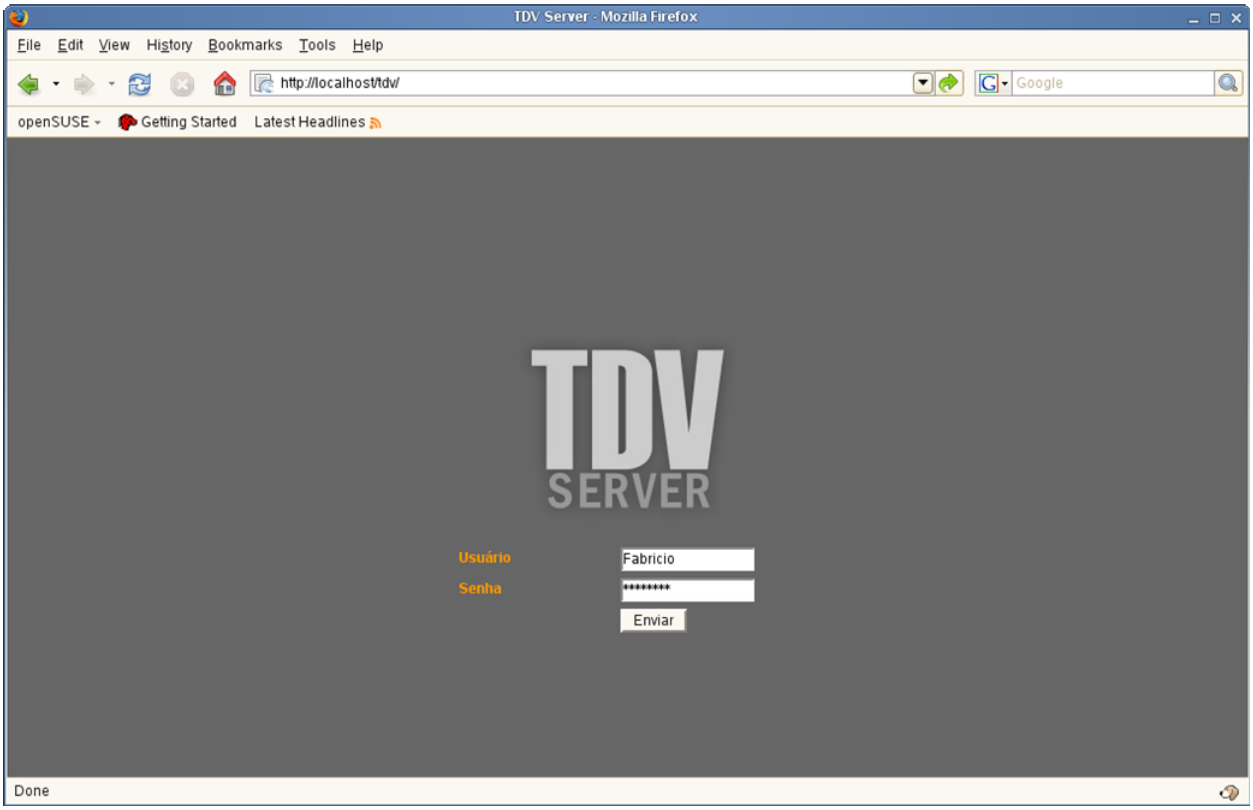

Figura A.1 – Tela inicial de autenticação da interface *web*

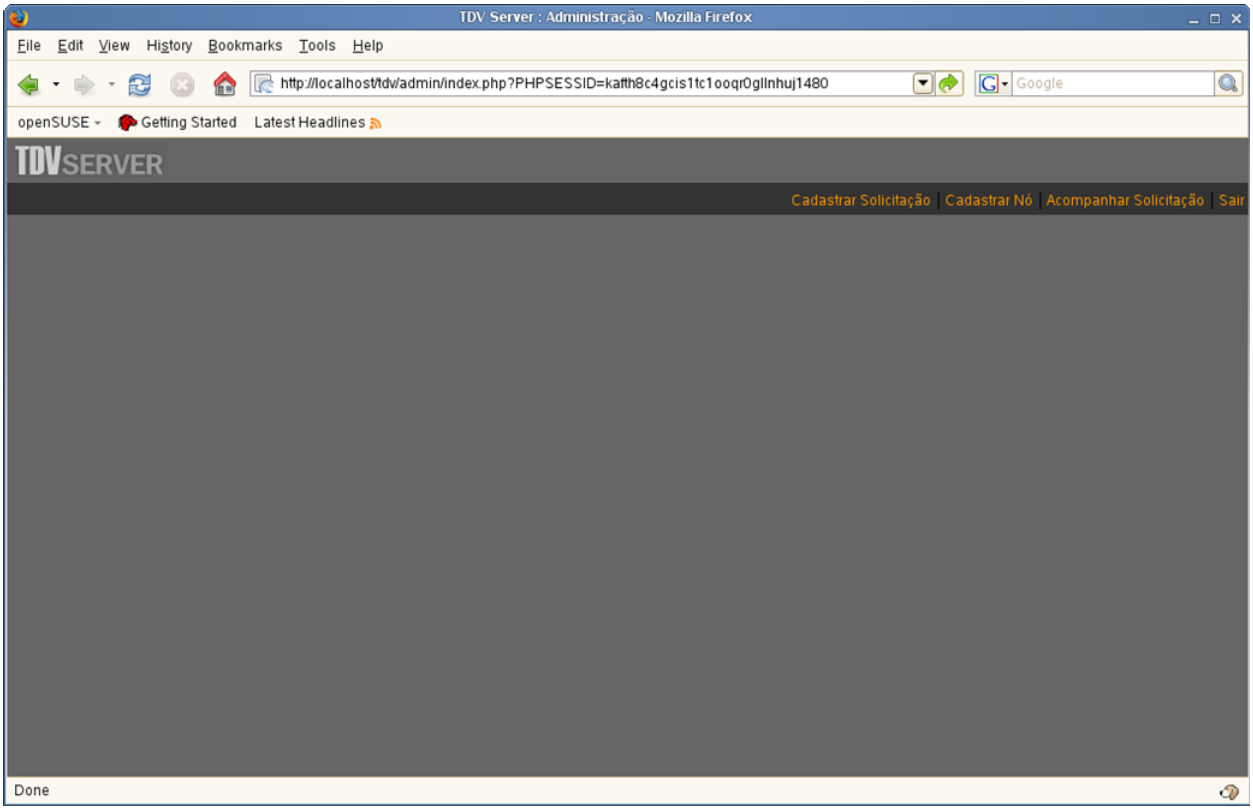

Figura A.2 – Tela inicial da interface do administrador após a autenticação

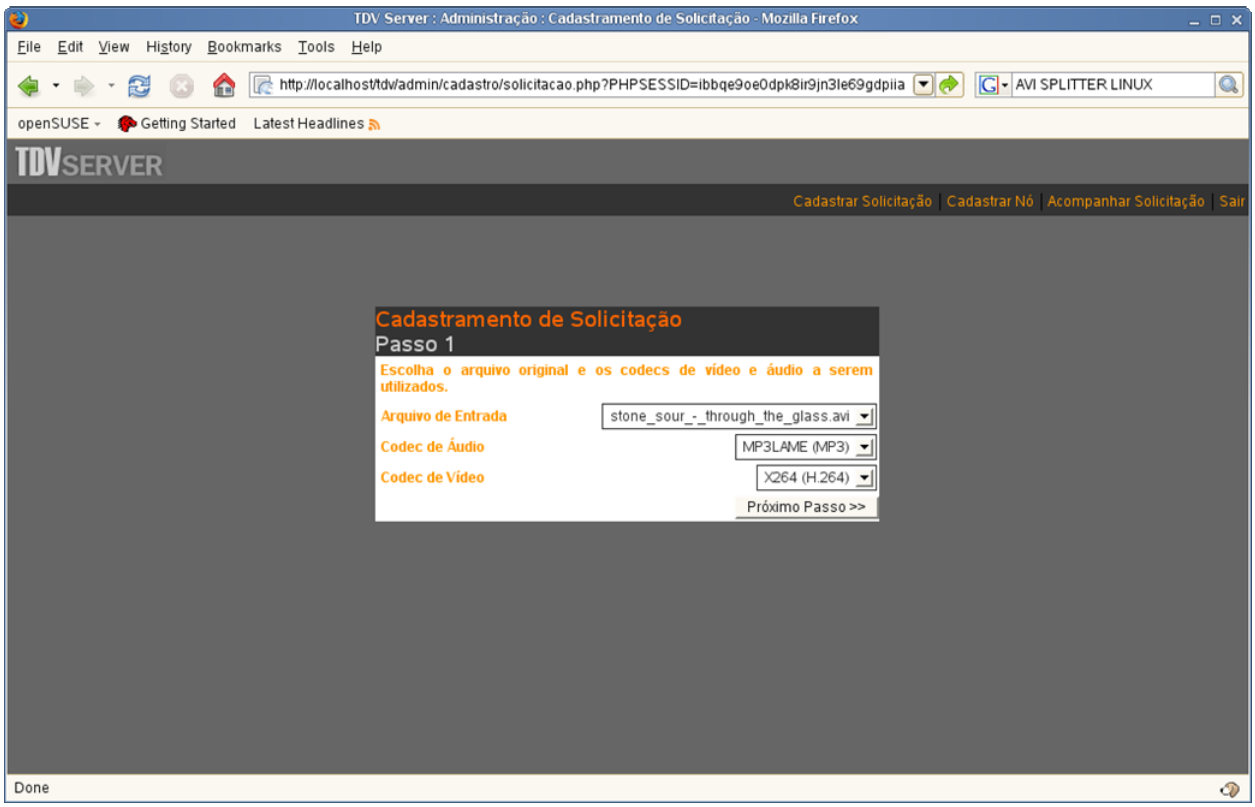

Figura A.3 – Tela inicial do cadastramento de uma nova solicitação

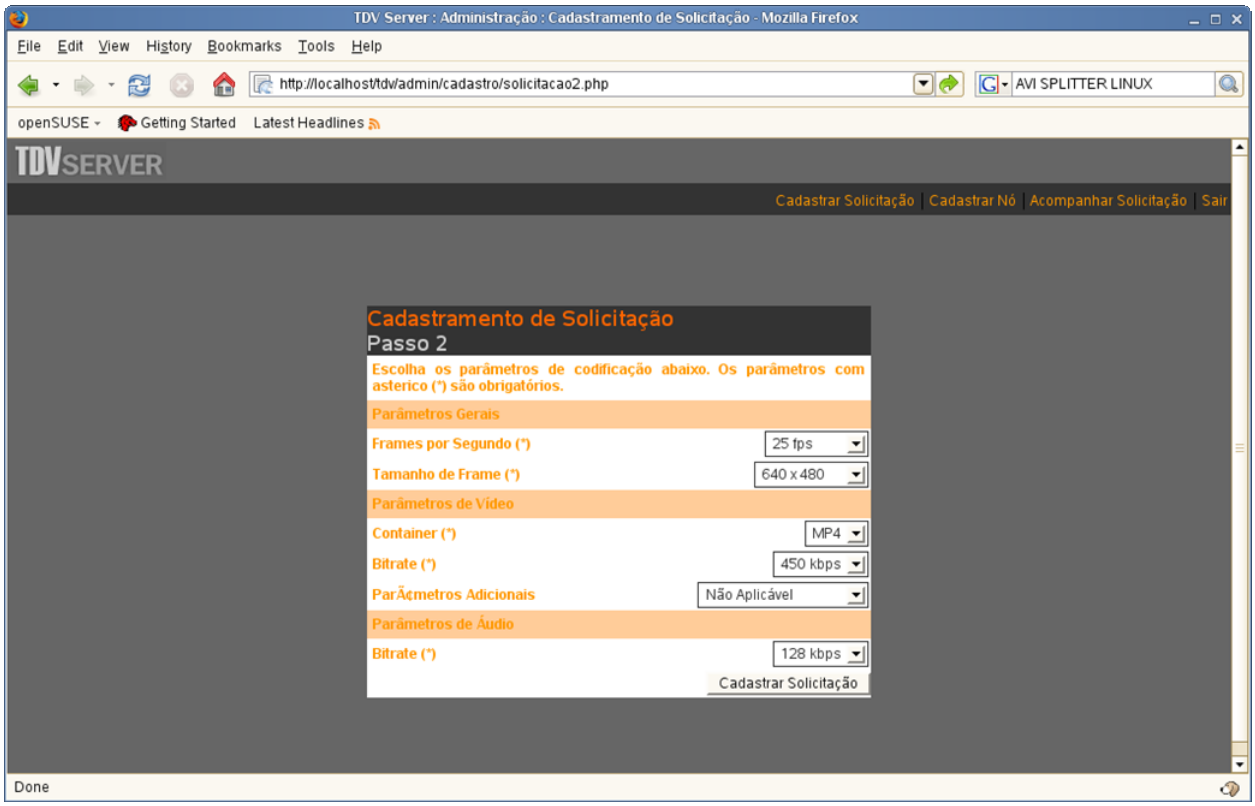

Figura A.4 – Tela de escolha dos parâmetros da nova solicitação

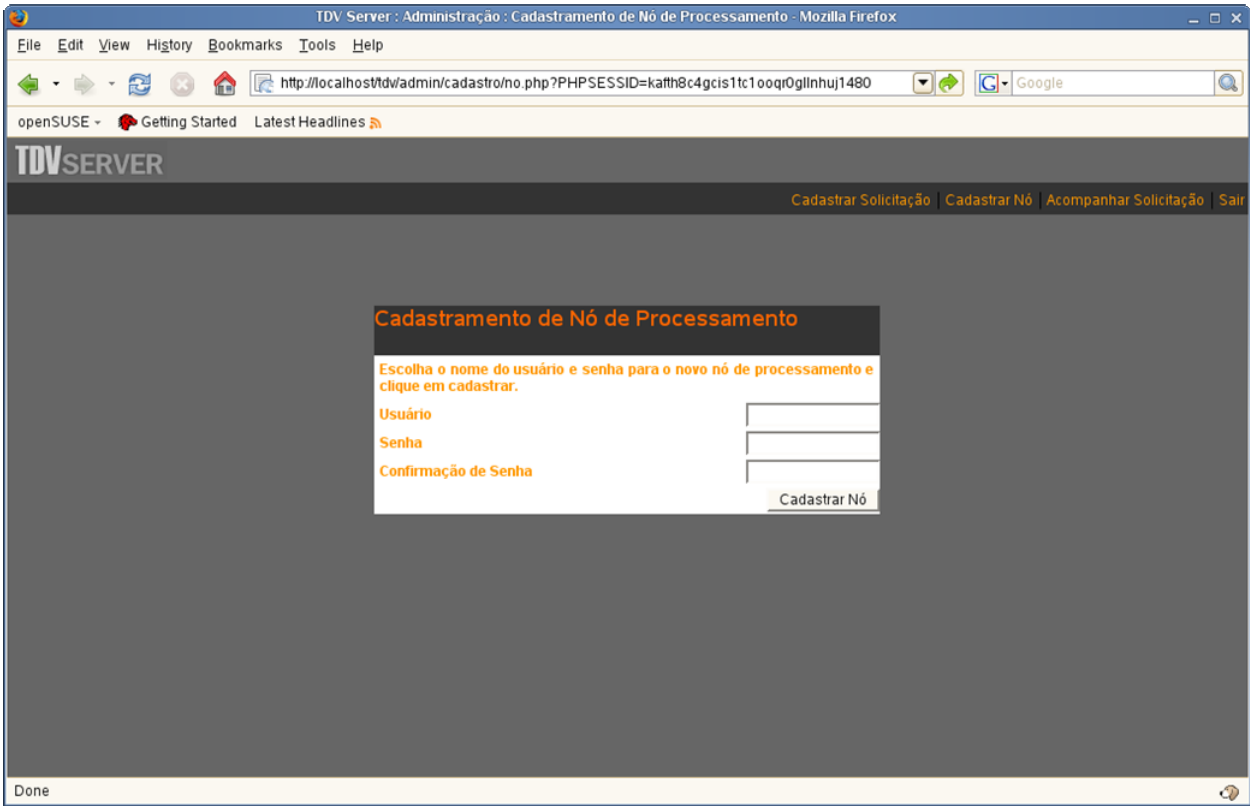

Figura A.5 – Tela de cadastramento de nós de processamento

Figura A.6 – Tela de acompanhamento das solicitações

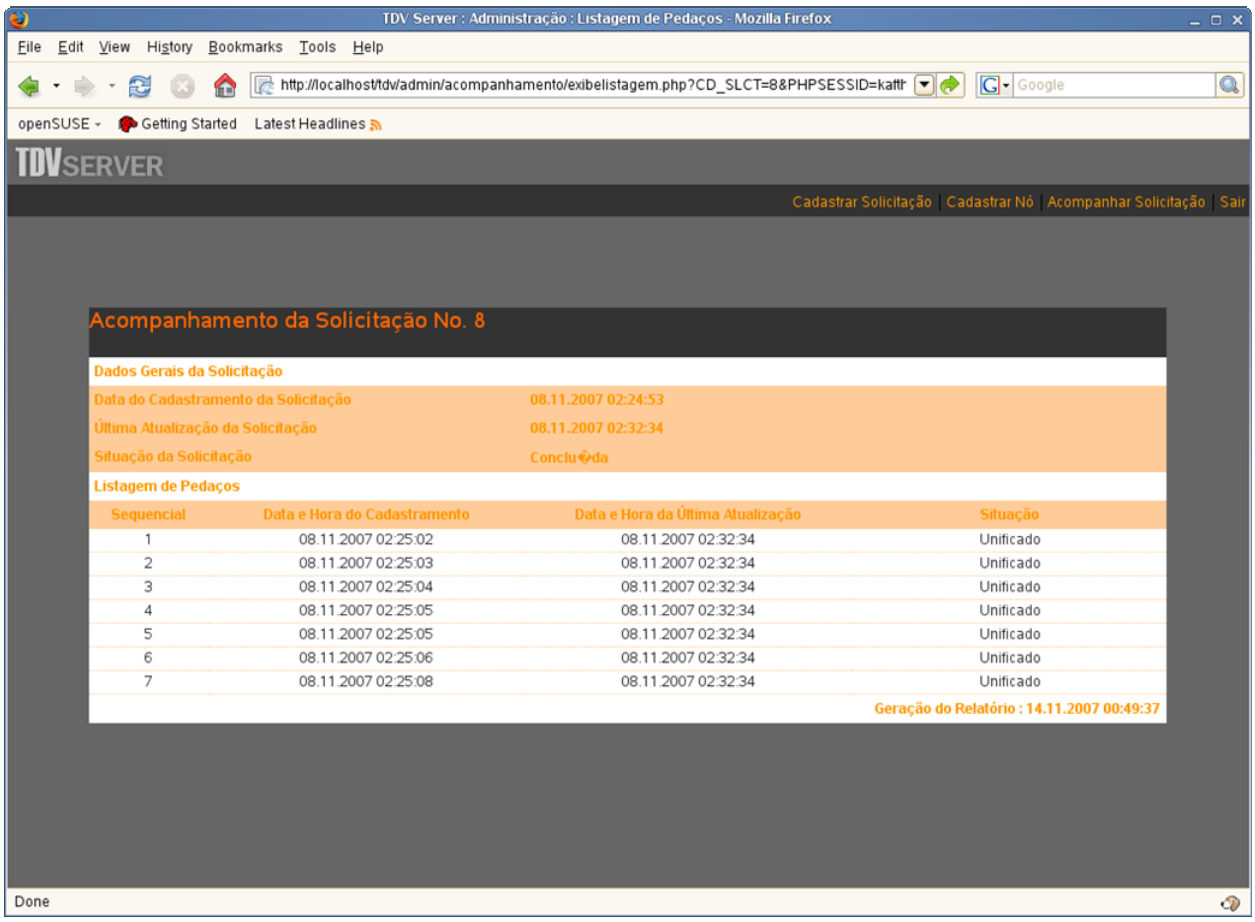

Figura A.7 – Tela de detalhamento dos pedaços de uma solicitação