Rodrigo Borges da Silva Santos

PUC-Rio - Certificação Digital Nº 0420994/CB

Sistema de Controle de Versões para Edição Cooperativa de Vídeo MPEG-2

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA

Programa de Pós-Graduação em Informática

Rio de Janeiro

15 de Março de 2007

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Pontifícia Universidade Católica
do Rio de Janeiro

Rodrigo Borges da Silva Santos

Sistema de Controle de Versões para Edição Cooperativa de Vídeo MPEG-2

PUC-Rio - Certificação Digital Nº 0420994/CB

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Luiz Fernando Gomes Soares
Co-orientador: Marco Antônio Casanova

Rio de Janeiro,

15 de março de 2007

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Rodrigo Borges da Silva Santos

Sistema de Controle de Versões para Edição Cooperativa de Vídeo MPEG-2

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Luiz Fernando Gomes Soares
Orientador
Departamento de Informática - PUC-Rio

Marco Antonio Casanova
Co-Orientador
Departamento de Informática - PUC-Rio

Bruno Feijó
Departamento de Informática - PUC-Rio

Rogério Ferreira Rodrigues
Departamento de Informática - PUC-Rio

José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 15 de março de 2007.

PUC-Rio - Certificação Digital Nº 0420994/CB

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

Rodrigo Borges da Silva Santos

Bacharel em Ciência da Computação graduado pela Universidade Federal de Pernambuco (UFPE) em outubro de 2003.

Ficha Catalográfica

Santos, Rodrigo Borges da Silva.

Sistema de controle de versões para edição cooperativa de vídeo MPEG-2 / Rodrigo Borges da Silva Santos ; orientador: Luiz Fernando Gomes Soares ; co-orientador: Marco Antônio Casanova. – 2007.

110 f. : il. (col.) ; 30 cm

Dissertação (Mestrado em Informática)– Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2007.

Inclui bibliografia

1. Informática – Teses. 2. Controle de versão. 3. Edição cooperativa de vídeo. 4. MPEG-2. 5. Compartilhamento de versão. I. Soares, Luiz Fernando Gomes. II. Casanova, Marco Antônio. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

Este trabalho é dedicado

aos meus pais, Edmilson e Antônia,
e as minhas irmãs, Renata e Rafaella, por todo amor que dedicam a nossa família.

À Patrícia,
que mesmo distante, é minha inseparável companheira, fonte de amor e de minha alegria.

PUC-Rio - Certificação Digital Nº 0420994/CB

Agradecimentos

Aos meus orientadores, Professor Luiz Fernando e Marco Antonio Casanova, por terem me orientado com paciência e respeito, sempre solícitos em ouvir minhas dúvidas e inquietações.

À minha família, pelo apoio e incentivo demonstrados em todos os momentos de minha vida.

À Patrícia, por seu amor, carinho e compreensão.

À equipe do Laboratório TeleMídia, pela amizade e companheirismo, que muito contribuíram para a realização deste trabalho de uma forma mais prazerosa. Em especial, a Macarrão (vulgo Laiola) e Lambão (vulgo Rafael Rodrigues), pela amizade, presença constante, apoio e orientação indispensáveis à efetivação deste trabalho.

A todos os meus amigos e primos pelo carinho e confiança. Aos companheiros de longa data, amigos de Pernambuco, em especial Moacir, Zeca, Katyusco e Lulu. Também aos amigos Malcher, Renato e Vinicius.

Aos membros da banca pelos comentários pertinentes e pelas revisões precisas. Agradeço também aos professores e funcionários do Departamento de Informática da PUC-Rio que colaboraram para a conclusão deste trabalho.

À CNPQ pelo apoio financeiro que proporcionou a realização deste trabalho.

Resumo

Santos, Rodrigo Borges da Silva; Soares, Luiz Gomes Soares (orientador); Casanova, Marco Antônio (co-orientador). **Sistema de Controle de Versões para Edição Cooperativa de Vídeo MPEG-2.** Rio de Janeiro, 2007. 110p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os avanços das tecnologias de captura, armazenamento e compressão de vídeo digital estão motivando o desenvolvimento e a disponibilização de novos serviços e sistemas para manipulação e gerenciamento de acervos de vídeo. Um exemplo disso são os sistemas de gerenciamento, edição e compartilhamento de versões utilizados pelos produtores de conteúdo audiovisual. Entretanto, tais funcionalidades são requisitos não encontrados em um único sistema. Este trabalho descreve um sistema que possibilita a edição cooperativa de dados audiovisuais no formato MPEG-2 permitindo o controle de versão, a visualização e manipulação do seu conteúdo por partes (segmentos). Esse sistema colaborativo tem ainda como vantagens a divisão de tarefas, a fusão das contribuições e a extração de informações da autoria de cada versão.

PUC-Rio - Certificação Digital Nº 0420994/CB

Palavras-chave

Controle de Versão; Edição Cooperativa de Vídeo; MPEG-2; Compartilhamento de Versão.

Abstract

Santos, Rodrigo Borges da Silva; Soares, Luiz Fernando Gomes (advisor); Casanova, Marco Antônio (co-advisor). **Version Control System for Cooperative MPEG-2 Video Editing.** Rio de Janeiro, 2007. 100p. Master Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Technological advances in areas such as capture, storage and compression of digital video are stimulating the development of new services and systems for manipulation and management of huge amount of video data. An example of this, are the systems of management, editing and sharing of versions used by producers of audiovisual content. However, such functional requirements are not found in one system. This work describes a system that makes possible the cooperative edition of audiovisual data in MPEG-2 format, allowing the version control, visualization and manipulation of its content by segments. This collaborative system still has advantages as the division of tasks between editors, the fusion of different versions and the extraction of information of authorship from each version.

PUC-Rio - Certificação Digital Nº 0420994/CB

Key words

Version Control; Cooperative Video Editing; MPEG-2; Version Sharing.

Sumário

1 Introdução 13
1.1. Motivação 13
1.2. Objetivos 16
1.3. Estrutura da Dissertação 17

2 Conceitos Gerais do Padrão MPEG-2 18
2.1. O Fluxo MPEG-2 de Sistemas 19
2.2. O Fluxo MPEG-2 de Vídeo 21
2.2.1. Estruturas de um fluxo de vídeo MPEG-2 21
2.2.2. Controle de ocupação do *buffer* 25

3 Trabalhos Relacionados 30
3.1. Sistemas de Controle de Versão 30
3.1.1. Conceitos básicos 31
3.1.2. Envio e resgate de versões 33
3.1.3. Trabalho Colaborativo 34
3.2. Sistemas de Edição de Vídeo Colaborativo 36
3.2.1. Mecanismos de Segmentação e Remontagem do MPEG 36
3.2.2. Ferramentas de Edição de Vídeo Colaborativo 38

4 Modelo de Dados do Controle de Versões para Edição Cooperativa de Vídeo 42
4.1. Árvore de Versionamento de um Vídeo 42
4.2. Acessos Concorrentes aos Nós 45
4.2.1. Versão Temporária e Permanente da Árvore de Versionamento 50
4.3. Granularidade Múltipla na Árvore de Versionamento 52
4.3.1. Bloqueios Explícitos e Implícitos no modo Exclusivo ou Compartilhados 53
4.3.2. Bloqueios Intencionais no modo Exclusivo ou Compartilhado 54
4.3.3. Protocolo de Bloqueio em Duas Fases 56

PUC-Rio - Certificação Digital Nº 0420994/CB

4.3.4. Nós Compartilhados entre Árvores de Versionamento 58
4.4. Mecanismo de Segmentação 60
4.5. Mecanismo de Remontagem 63
4.6. Fusão entre Árvores de Versionamento 64

5 Implementação 68
5.1. Desenvolvimento do *VideoCVS* 68
5.2. Arquitetura do *VideoCVS* 69
5.3. Modelo Entidade e Relacionamento do *VideoCVS* 75
5.3.1. Classes dos Objetos Remotos 76
5.3.2. Classes do Repositório 78
5.4. Mecanismo de Segmentação 79
5.5. Mecanismo de Remontagem 81
5.6. Exemplo de Uso do *VideoCVS* 82
5.6.1. Criação de uma Árvore de Versionamento 82
5.6.2. Checkout de uma Árvore de Versionamento 86
5.6.3. Edição Colaborativa de uma Árvore de Versionamento 87
5.6.4. Fusão entre Árvores de Versionamento 91
5.6.5. Remontagem dos Segmentos da Árvore de Versionamento 94

6 Conclusão 96
6.1. Contribuições da Dissertação 96
6.2. Trabalhos Futuros 97

Referências 101

Anexo A 105

PUC-Rio - Certificação Digital Nº 0420994/CB

Lista de figuras

Figura 1 – Efeito do processo de decodificação e codificação em cascata........... 15
Figura 2 – Estrutura do fluxo MPEG-2.. 19
Figura 3 – Sincronização entre o codificador e o decodificador......................... 20
Figura 4 – Estrutura do fluxo MPEG Vídeo 21
Figura 5 – Ordem de apresentação e codificação dos quadros............................ 25
Figura 6 – Exemplo de múltiplas versões organizadas em revisões.................... 32
Figura 7 – Histórico de versões no sistema estilo CVS. 34
Figura 8 – Histórico de versões no sistema estilo SVN. 34
Figura 9 – Detecção das Tomadas de Cenas de um Noticiário de Telejornal....... 40
Figura 10 – Exemplo de uma árvore de versionamento de vídeo 43
Figura 11 – Compartilhamento de nós entre árvores de versionamento.............. 45
Figura 12 – Problema do acesso simultâneo de um nó. 46
Figura 13 – Método *Lock-Modify-Unlock ou Exclusive Lock*............................ 47
Figura 14 – Método C*opy-Modify-Merge* ou *Optimistic Merges*........................ 48
Figura 15 – Continuação do exemplo do método *copy-modify-merge*................ 49
Figura 16 – Versão permanente e temporária 50
Figura 17 – Versão temporária editada por um único usuário 51
Figura 18 – Versão temporária editada por vários usuários................................ 52
Figura 19 – Árvore de versionamento com bloqueio X e S 53
Figura 20 – Árvore de versionamento com bloqueio intencional 55
Figura 21 – Exemplo de bloqueio de um nó no modo exclusivo 59
Figura 22 – Estado de bloqueio dos nós 60
Figura 23 – Fases do mecanismo de segmentação 61
Figura 24 – Tratamento das bordas dos segmentos 63
Figura 25 – Detecção das tomadas de cenas de um vídeo 63
Figura 26 – Fusão de duas árvores de versionamento 66
Figura 27 – Fusão das árvores de versionamento com intervenção do usuário 66
Figura 28 – Fusão de duas árvores de versionamento com nós não equivalentes 67
Figura 29 – Fusão de duas árvores de versionamento com nós invertidos........... 67

PUC-Rio - Certificação Digital Nº 0420994/CB

Figura 30 – Arquitetura do *VideoCVS* 70
Figura 31 – Diagrama de classes do *VideoCVS Server*.................................... 71
Figura 32 – Exemplo de commit na arquitetura do *VideoCVS*........................... 73
Figura 33 – Diagrama de classes do *VideoCVS Client* 74
Figura 34 – Modelo ER do sistema *VideoCVS*.. 75
Figura 35 – Diagrama de classes dos objetos remotos do modelo ER 77
Figura 36 – Diagrama de classes dos objetos remotos do repositório................. 78
Figura 37 – Diagrama de classes do mecanismo de segmentação 79
Figura 38 – Diagrama de classes da estrutura do MPEG-2 de vídeo 80
Figura 39 – Diagrama de classes do mecanismo de remontagem 81
Figura 40 – Tela inicial do VideoCVS Client 83
Figura 41 – Primeira tela da criação da árvore de versionamento....................... 84
Figura 42 – Segunda tela da criação da árvore de versionamento....................... 84
Figura 43 – Quadros iniciais dos segmentos do vídeo *pinpong.m2v*.................. 85
Figura 44 – Commit da árvore de versionamento pelo usuário 1........................ 85
Figura 45 – Operação de *checkout* de uma árvore de versionamento 86
Figura 46 – Resultado da busca do *checkout* 87
Figura 47 – Bloqueio no modo exclusivo permitido ao usuário 1....................... 88
Figura 48 – Bloqueio no modo exclusivo negado ao usuário 2 88
Figura 49 – Bloqueio no modo compartilhado negado ao usuário 2................... 89
Figura 50 – Clicando na opção Preview para visualizar o vídeo 89
Figura 51 – Bloqueios no modo exclusivo permitido ao usuário 2..................... 90
Figura 52 – Desbloqueio de um nó da árvore de versionamento pelo usuário 2 .. 90
Figura 53 – Permissão negada para bloquear na fase de encolhimento................ 91
Figura 54 – *Merge* entre versões 92
Figura 55 – Fusão entre as árvores de versionamento *tree_v12* e *tree_v13*.......... 92
Figura 56 – Intervenção manual do usuário na fusão entre *tree_v13* e *tree_v14* .. 93
Figura 57 – Árvores de versionamento resultantes após intervenção do usuário.. 93
Figura 58 – Operação de inserção de nós na árvore de versionamento................ 94
Figura 59 – Árvore de versionamento *tree_v2* para remontagem 95
Figura 60 – *Preview* da remontagem da tree_v2 95

PUC-Rio - Certificação Digital Nº 0420994/CB

Lista de tabelas

Tabela 1 – Parâmetros repeat_first_field e top_field_first 23
Tabela 2 – Resumo da sintaxe das camadas do MPEG-2 de Vídeo 24
Tabela 3 – Matriz de compatibilidade dos modos de bloqueio 56
Tabela 4 – Exemplo usando o protocolo de bloqueio em duas fases 57

PUC-Rio - Certificação Digital Nº 0420994/CB

1
Introdução

Os sistemas cooperativos ou sistemas colaborativos são sistemas de informação que fornecem suporte computacional aos usuários que tentam resolver cooperativamente um problema, sem que todos os usuários estejam no mesmo local, ao mesmo tempo. Com base nas pesquisas realizadas na área denominada, *Computer Supported Cooperative Work* (CSCW), ou trabalho cooperativo suportado por computador, foram desenvolvidas diversas ferramentas de software para implantação de sistemas cooperativos, entre essas, os sistemas de edição cooperativa de vídeo.

O sistema de edição cooperativa de vídeo é um sistema onde os editores de vídeo trabalham em equipe e tem por finalidade dividir as tarefas de edição e mesclar as contribuições de cada editor de vídeo. Visando contribuir no processo de desenvolvimento e produção dos vídeos, é incluído nessa ferramenta o conceito de sistemas de controle de versão. O controle de versão auxilia no processo de desenvolvimento de sistemas, mantendo um histórico de alterações, permitindo que vários usuários trabalhem sobre os mesmos arquivos ao mesmo tempo e minimizando os possíveis conflitos de edições.

Este capítulo descreve as principais motivações para se desenvolver um sistema de controle de edição cooperativa de vídeo, bem como detalha os objetivos e apresenta a estrutura desta dissertação.

1.1.
Motivação

Acervos volumosos de vídeo estão se popularizando devido ao avanço das tecnologias de captura, armazenamento e compressão de vídeo digital. A melhora significativa das transmissão e recepção de dados e o aumento da capacidade de processamento dos computadores também favorecem essa popularização. Todos

PUC-Rio - Certificação Digital Nº 0420994/CB

esses fatores motivam o desenvolvimento e a disponibilização de novos serviços e produtos para manipulação e gerenciamento dos acervos audiovisuais.

Entre os sistemas diretamente relacionados a esses acervos destacam-se: televisão interativa e digital, sistemas hipermídia, bibliotecas digitais, serviços de telemedicina, ambientes virtuais de aprendizado, ambientes de montagem cinematográfica, entre outros. Um exemplo de serviço é o fornecimento e manipulação de vídeo por demanda [Tobin99], onde arquivos de vídeo digital, com conteúdos variados, como filmes ou programas de TV, podem ser remotamente acessados, manipulados e armazenados em servidores.

Na montagem cinematográfica, a edição de um filme é um processo de corte e montagem por meio analógico (edição linear) ou digital (edição não linear). Este processo é necessário pois os filmes normalmente são gravados em partes, divididos por cenas ou tomadas, que são feitas diversas vezes e por diferentes ângulos. A edição de vídeo consiste em decidir que tomadas usar, quais são as melhores e uní-las na seqüência desejada. Pode-se inclusive montar as seqüências fora da ordem cronológica de gravação ou do próprio tempo do filme. Editar um filme ou vídeo não se limita a escolher as melhores cenas, pois é nesta fase da produção que também são inseridos os efeitos especiais, trilhas sonoras e legendas.

Produtores e montadores de vídeo necessitam de um sistema que torne possível o gerenciamento, a edição e o compartilhamento das versões de um vídeo. Entretanto, tais funcionalidades são requisitos não encontrados num único sistema pois, atualmente, não existe uma integração entre sistemas de controle de versão e ferramentas de edição audiovisual. Além disso, o processo de edição de vídeo é realizado seqüencialmente pelos editores, e não concorrentemente. Por essas restrições, observou-se a necessidade de elaborar um sistema distribuído e colaborativo que atendesse a esses requisitos.

O sistema proposto neste trabalho utiliza um mecanismo de segmentação que tem a função de particionar o vídeo no domínio comprimido e estruturá-lo sob a forma de uma árvore. Essa segmentação refere-se à identificação de regiões em um quadro de vídeo que são homogêneas em algum sentido [Tekalp00]. No contexto deste trabalho, ela tem a conotação específica de se referir à detecção de transição entre tomadas de câmera, essencial para indexação, recuperação e navegação (*browsing*) em acervos de vídeo [Tobin99]. Os segmentos são

PUC-Rio - Certificação Digital Nº 0420994/CB

definidos através de intervalos dos índices dos quadros de um vídeo e podem ser manipulados de forma única e independente, dando assim a opção da divisão de trabalhos entre editores.

Devido à decomposição do vídeo em segmentos, também se faz necessário um mecanismo para posterior remontagem. Esse mecanismo deve estar atento às particularidades do formato de vídeo, no caso desta dissertação o formato MPEG-2, e ao modo como foi realizada a segmentação.

A forma trivial do mecanismo de segmentação é sua realização em fluxos não comprimidos, exigindo que o fluxo original seja inicialmente decodificado e, após a edição, seja novamente codificado. A edição de dados audiovisuais diretamente em sua forma comprimida, no entanto, apresenta várias vantagens como menor memória necessária, menor tempo de acesso aos dados armazenados e menor requisito de poder de processamento e retardo [EgAA00]. A maior vantagem, no entanto, reside sobre a qualidade do arquivo original que é mantida. Os processos de decodificação e recodificação provocam perdas na qualidade original de um arquivo audiovisual.

Como exemplo, o efeito de perda de qualidade apenas devido ao processo de decodificação e recodificação pode ser visualizado, para um fluxo de vídeo, na Figura 1, onde foram utilizadas seqüências de teste padronizadas de fluxos MPEG de Vídeo (*Mobile & Calendar*, *Basketball* e *Horseriding*) [BrDK97]. O gráfico também mostra que as perdas no processo de codificação são menores caso o tipo de cada figura, em relação ao fluxo original, seja mantido.

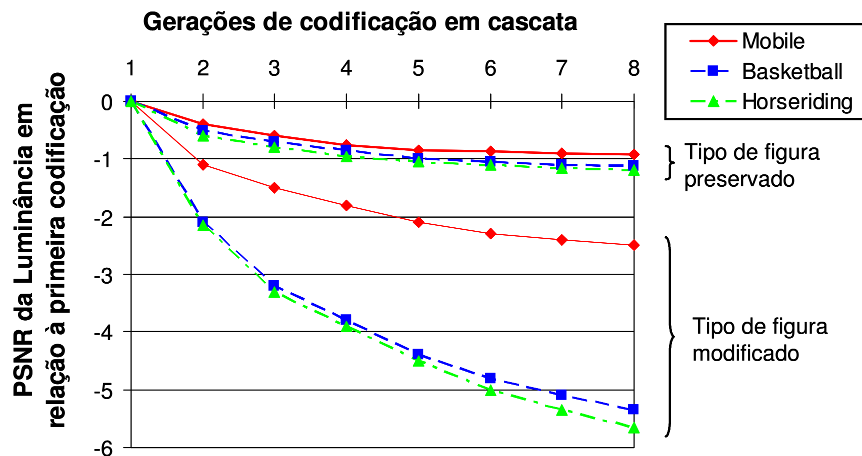

Figura 1 – Efeito do processo de decodificação e codificação em cascata

PUC-Rio - Certificação Digital Nº 0420994/CB

Em resumo, o que se propõe nesta dissertação é um sistema que possibilita a edição compartilhada de dados audiovisuais no formato MPEG-2 [ITUT00a][ITUT00b], permitindo a visualização e manipulação dos seus conteúdo por partes (segmentos). A divisão de tarefas, a fusão das contribuições e a extração de informações da autoria de cada versão serão facilidades essenciais que beneficiarão o trabalho colaborativo.

1.2. Objetivos

Esta dissertação tem como principal objetivo prover o controle de versão das edições cooperativas de vídeo no formato MPEG-2. Para isso, foi definida uma estrutura de dados que permite o gerenciamento e a recuperação das edições do vídeo versionado, no decorrer do tempo. O conteúdo audiovisual é organizado numa estrutura chamada de árvore de versionamento, conforme apresentado no Capítulo 4.

Outro objetivo, desta vez relacionado ao trabalho colaborativo, é permitir que o processo de produção de vídeo seja realizado concorrentemente entre os editores. Afim de garantir a concorrência e a divisão de tarefas no processo de edição, foi desenvolvido um sistema distribuído que controla os acessos simultâneos da árvore de versionamento. Este sistema distribuído permite que vários usuários possam acessar e editar as versões, mesmo estando dispersos geograficamente. O sistema provê a sua integração a uma ferramenta comercial de edição de vídeo (por exemplo, *Final Cut Pro* [FCut06]e *Avid Xpress* [Avid06]), uma vez que não faz parte deste trabalho a implementação de uma ferramenta de edição de vídeo.

Além disso, este trabalho apresenta um mecanismo de remontagem dos segmentos do vídeo MPEG-2 no domínio comprimido. Este mecanismo concatena os arquivos de vídeo estruturados como folhas da árvore de versionamento, na composição da versão final do arquivo audiovisual comprimido.

PUC-Rio - Certificação Digital Nº 0420994/CB

1.3.
Estrutura da Dissertação

Esta dissertação encontra-se organizada como se segue. O Capítulo 2 apresenta os conceitos do padrão MPEG-2, descrevendo o fluxo MPEG-2 de sistemas e o fluxo de vídeo. Em seguida, descreve o gerenciamento da ocupação do buffer de entrada do exibidor, uma vez que os mecanismos de segmentação do vídeo em trechos e remontagem dos mesmos devem estar atentos a esse controle do buffer.

O Capítulo 3 apresenta os trabalhos relacionados ao assunto. Aborda as principais técnicas e mecanismos de edição de dados audiovisuais e alguns sistemas de controle de versão, anunciando as estratégias utilizadas e as questões não resolvidas de cada proposta.

O Capítulo 4 descreve o modelo de dados do controle de versão das edições cooperativas de vídeo MPEG-2 proposto na dissertação. Primeiramente, apresenta a estrutura da árvore de versionamento de um vídeo e como é feito o controle de acesso concorrente aos nós da estrutura. Em seguida, descreve como foi feito a segmentação do vídeo em trechos MPEG-2 e o mecanismo de remontagem desses trechos no domínio comprimido. Por fim, descreve o mecanismo que realiza a fusão entre as árvores de versionamento, contribuindo assim para a mesclagem das edições cooperativas.

O Capítulo 5 descreve a implementação do sistema proposto, que integra o controle de versão às edições cooperativas de vídeo. Além disso, apresenta exemplos de uso do sistema desenvolvido.

Por fim, o Capítulo 6 apresenta as considerações finais sobre o trabalho, destacando as principais contribuições desta dissertação e possíveis trabalhos futuros.

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

2
Conceitos Gerais do Padrão MPEG-2

Vários padrões de codificação, compactação e compressão foram desenvolvidos com o intuito de diminuir os requisitos de recursos para a transmissão e o armazenamento de sinais digitais de vídeo e áudio. Visando estabelecer padrões internacionais para a representação e codificação de informações audiovisuais em formato digital com compressão, a ISO (*International Organization for Standardization*) e a IEC (*International Electrotecnical Commission*) estabeleceram o grupo de trabalho MPEG (*Motion Picture Coding Experts Group*), que iniciou seus trabalhos em maio de 1988. A família de padrões produzidos foi popularmente conhecida como padrões MPEG e inclui, entre outros, os conjuntos de padrões MPEG1 e MPEG-2.

O padrão MPEG-2 foi iniciado em 1990, como uma evolução do MPEG1, e publicado em 1995. O objetivo deste padrão é prover uma taxa de vídeo de 1,5 Mbps a 15 Mbps, adequado para sinais de televisão padrão (SDTV – *Standard Definition Television*) e taxas de 15 Mbps a 30 Mbps para sinas de televisão de alta definição (HDTV – *High Definition Television*). Para taxas inferiores a 3 Mbps, o padrão MPEG1 pode apresentar maior eficiência que o MPEG-2.

O padrão MPEG-2 é descrito pelo conjunto de especificações ISO/IEC 13818, cujas principais definições estabelecem a forma de compressão para o fluxo multiplexado de sistemas [ISO00a], para o vídeo e para o áudio [ISO00b]. No âmbito do ITU-T (*International Telecommunication Union – Telecommunication Standardization Sector*), os padrões MPEG-2 *Systems* e MPEG-2 *Video* estão descritos nas recomendações H.222.0 [ITUT00a] e H.262 [ITUT00b], respectivamente.

Este capítulo descreve os principais conceitos do padrão MPEG-2, salientando os parâmetros que são diretamente alterados pelas operações de segmentação do vídeo em trechos e remontagem dos mesmos.

2.1.
O Fluxo MPEG-2 de Sistemas

A estrutura de um fluxo definida pelo padrão MPEG-2 pode ser visualizada na Figura 2 e está dividida em duas camadas: a camada de compressão e a camada de sistema. A camada de sistema, definida no padrão *MPEG-2 Systems*, é responsável pela divisão e encapsulamento de cada fluxo comprimido em pacotes; pela inserção de informações de sincronização entre fluxos de mídias diferentes; pela multiplexação dos fluxos encapsulados; e pelo transporte da informação de referência do relógio utilizado no codificador. A camada de compressão refere-se à codificação de cada um dos dados audiovisuais, conforme especificado nos padrões MPEG-2 Áudio e Vídeo.

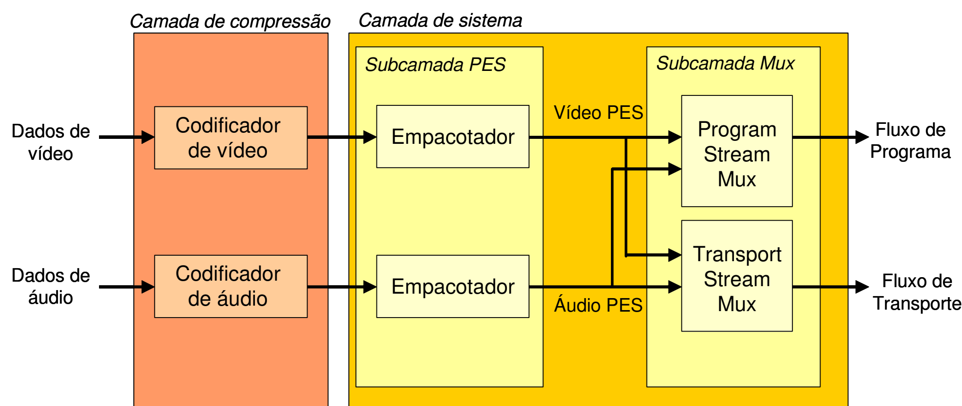

Figura 2 – Estrutura do fluxo MPEG-2

Os dados individuais de cada mídia, após sofrer o processo de compressão, são denominados de fluxos elementares e são divididos em pacotes na subcamada PES (*Packetized Elementary Stream*). As principais funções desempenhadas pela subcamada PES são a identificação exclusiva de cada fluxo, realizada através do parâmetro *stream_ID*, e a sincronização intra e intermídia, discutida a seguir. Os dados empacotados, ou seja, os fluxos de áudio, vídeo ou dados PES, são enviados à subcamada de multiplexação, onde é inserida a informação de referência de relógio do codificador.

Na camada de sistemas estão definidos dois formatos: o Fluxo de Transporte (TS), que contém um ou mais programas e é apropriado para a transmissão e o armazenamento em ambientes ruidosos, onde a ocorrência de erros é freqüente; e o Fluxo de Programa (PS), que contém apenas um programa e é adequado para uso em ambientes com baixas taxas de erros. Cada programa é definido como um

PUC-Rio - Certificação Digital Nº 0420994/CB

conjunto de fluxos elementares, vídeo, áudio e dados, por exemplo, que podem ou não ter algum relacionamento temporal entre si de um programa, utiliza uma mesma base de tempo, ou referência de relógio.

Através da inserção de marcas de tempo (*timestamp*), tanto nos fluxos PES quanto no fluxo de sistema, a sincronização intra e intermídia é obtida. A marca de tempo é uma amostra do contador da respectiva base de tempo, em um determinado instante. As marcas de tempo inseridas no fluxo de sistema, na subcamada de multiplexação, permitem, ao decodificador, a recuperação da referência do relógio utilizado pelo codificador. Elas são denominadas de *System Clock Reference* (SCR) e *Program Clock Reference* (PCR), respectivamente, para os fluxos TS e PS, e são definidas em termos de um relógio de sistema comum denominado STC (*System Time Clock*). Os valores das marcas de tempo SCR e PCR significam o instante de tempo em que o último bit desses campos entra no decodificador. O intervalo de tempo máximo permitido entre o envio de duas marcas consecutivas é de 0,7s. O processo de geração e extração das marcas de tempo relativas ao SCR e PCR é ilustrado na Figura 3. O padrão MPEG não considera os retardos introduzidos pela rede de comunicação.

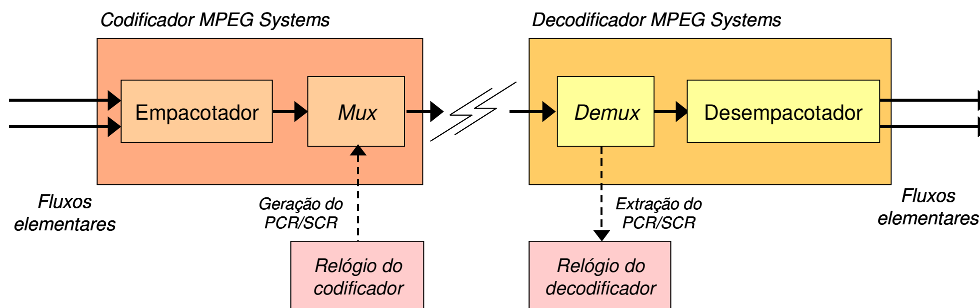

Figura 3 – Sincronização entre o codificador e o decodificador

Após o empacotamento dos respectivos dados em cada PES, alguns pacotes são escolhidos para transportar marcas de tempo. Dentre essas, dois tipos são definidos: o *Presentation Time Stamp* (PTS) e o *Decoding Time Stamp* (DTS). O PTS indica o instante de tempo em que a unidade de apresentação (figura, para o vídeo, e quadro, para o áudio) deve ser exibida. O DTS, presente apenas no fluxo de vídeo, indica o instante de tempo em que a unidade de apresentação deve ser entregue ao respectivo decodificador e é utilizado quando é necessária a reordenação de quadros, no decodificador.

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

2.2. O Fluxo MPEG-2 de Vídeo

O padrão MPEG-2 de vídeo utiliza mecanismos para eliminar, ou reduzir, a redundância temporal existente entre quadros consecutivos. A estrutura do fluxo codificado através do MPEG-2 de vídeo é hierárquica e contém seis camadas: seqüência, grupo de quadros ou figura (GOP), quadro ou figura, *slice*, macrobloco e bloco, conforme ilustrado na Figura 4.

As imagens de um vídeo são representadas por quadros de vídeo, os quais são representados por três matrizes retangulares de inteiros: uma matriz de luminância e duas matrizes de crominância. O termo quadro é utilizado tanto para imagens ainda não codificadas através do padrão MPEG-2 quanto para as imagens após a codificação. As informações de um quadro podem ser separadas em campos denominados *top field* e *bottom field*, compostos por linhas ímpares e pares de cada matriz que compõe um quadro, respectivamente.

Um quadro codificado através do padrão MPEG-2 pode representar um quadro ou um campo codificado. Um sinal de vídeo que contenha quadros que representem campos é dito ser um vídeo entrelaçado. Se o fluxo de vídeo contiver apenas quadros que representem quadros, ele é dito ser progressivo.

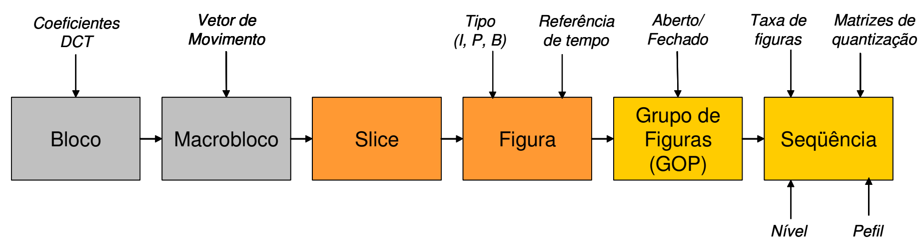

Figura 4 – Estrutura do fluxo MPEG Vídeo

2.2.1. Estruturas de um fluxo de vídeo MPEG-2

A primeira camada do padrão MPEG-2 de vídeo codificado é a seqüência. Um fluxo de vídeo é composto por um conjunto de seqüências, que são utilizadas para transportar informações sobre as dimensões dos quadros e a sua relação de aspecto, as taxas de quadro e de bit, e as matrizes de quantização utilizadas na codificação dos macroblocos, caso os valores-padrão não sejam utilizados. O parâmetro *progressive_sequence*, presente no elemento *Sequence Extension*,

especifica o tipo de sinal de vídeo utilizado, ou seja, se o fluxo de vídeo contém quadros entrelaçados ou não.

A taxa de quadro é fixa, sendo determinada através dos parâmetros *frame_rate_code*, da estrutura *Sequence Header* e que indica o valor base (*frame_rate_value*) da taxa de quadros, e os parâmetros *frame_rate_extension_n* e *frame_rate_extension_d*, presentes na estrutura *Sequence Extension*. Ela é calculada conforme a equação abaixo:

(Eq. 1) $$frame_rate = frame_rate_value * \left[\frac{frame_rate_extension_n + 1}{frame_rate_extension_d + 1} \right]$$

A camada grupo de quadros ou figuras (GOP) informa o instante de tempo referente ao primeiro quadro que o compõe, de acordo com o valor do relógio de referência utilizado pelo codificador, e é utilizado para prover acesso aleatório ao fluxo de vídeo. Um GOP também indica a dependência temporal entre quadros pertencentes a GOP diferentes. GOP fechados são aqueles formados apenas por quadros cujas referências localizam-se no próprio GOP. Em um GOP aberto, a decodificação de um de seus quadros requer a decodificação anterior de um quadro localizado em outro GOP.

A camada de quadro ou figura especifica o tipo de predição utilizado na codificação de uma imagem e fornece a informação de temporização de cada quadro. A quantidade de bits utilizada para a codificação de cada quadro é variável e é influenciada pelo valor do *quantum* adotado no processo de quantização da imagem.

O método de predição interquadro por compensação de movimento realiza a comparação de cada macrobloco de um quadro com macroblocos de quadros vizinhos. O macrobloco do quadro vizinho, escolhido para servir de referência na operação de predição, será aquele que menos se diferenciar do macrobloco a ser codificado. Um vetor de movimento é definido para indicar a diferença entre as localizações espaciais do macrobloco a ser codificado e o de referência, sendo transmitido junto ao macrobloco codificado. Cada macrobloco especifica sua posição em relação ao macrobloco anterior, a indicação do método de predição utilizado e quais os blocos de luminância e crominância estão codificados.

A especificação da estrutura de um quadro, estabelecendo se esta representa um campo ou um quadro, é determinada, em um fluxo de vídeo codificado,

PUC-Rio - Certificação Digital Nº 0420994/CB

através do parâmetro *picture_structure*, presente no elemento *Picture Coding Extension* que está contido em cada quadro (elemento *Picture*). Para os propósitos desta dissertação, não serão considerado os quadros que representam campos. Os termos figura e quadro serão utilizados como sinônimos.

As imagens são codificadas em quadros do tipo I (*Intracoded*), P (*Predictive Coded*) ou B (*Bidirectional Predictive Coded*). Quadros I são codificados utilizando-se informações contidas no próprio quadro original. Quadros P são codificados de forma preditiva em relação ao quadro I ou P anterior. Por fim, quadros B são codificados de forma preditiva em relação aos quadros I ou P, anteriores ou posteriores. Dessa forma, para a decodificação de um quadro B, é necessário que o quadro posterior, ao qual aquele se referencia, já tenha sido decodificado.

Cada quadro codificado possui o parâmetro *temporal_reference* que funciona como um contador, módulo 1.024, o qual é incrementado a cada novo quadro e é utilizado para que o decodificador possa identificar eventuais perdas de quadros.

O parâmetro *vbv_delay*, contido no elemento *Picture Header*, indica o tempo que o quadro deve permanecer no *buffer* de entrada do decodificador, exceto quando possui valor hexadecimal FFFF.

Dois outros parâmetros, contidos no elemento *Picture Coding Extension*, são importantes para o processo de decodificação: *repeat_first_field* e *top_field_first*. Em vídeos progressivos, eles indicam a quantidade de vezes que uma figura deve ser apresentada, após sua decodificação. A relação entre esses parâmetros é mostrada na Tabela 1.

Tabela 1 – Parâmetros repeat_first_field e top_field_first

Parâmetros			Significado
progressive sequence	**Repeat first field**	**Top field first**	
1	0	0	Quadro deve ser apresentado uma vez
1	1	0	Quadro deve ser apresentado duas vezes
1	1	1	Quadro deve ser apresentado três vezes

PUC-Rio - Certificação Digital Nº 0420994/CB

A camada *slice* contém um conjunto de macroblocos, pertencentes a uma mesma linha da imagem codificada. No entanto, nem todos os macroblocos precisam ser inseridos no fluxo comprimido, os quais são chamados de *skipped macroblocks*. Em seu cabeçalho, um *slice* especifica a linha de macroblocos a que se refere e o fator de escala utilizado na determinação dos coeficientes DCT. Essa camada é especificada para facilitar o correto posicionamento espacial das amostras, no processo de exibição da imagem, em casos onde haja perda de dados. A perda de algumas amostras pode causar erro no posicionamento espacial das outras amostras pertencentes ao mesmo *slice*. Esse efeito, porém, não é cumulativo e os outros *slices* podem ser apresentados corretamente.

Cada macrobloco indica o modo de predição utilizado durante sua codificação e seu respectivo vetor de movimento. Também contém um fator de escala para permitir o controle do *quantum* do processo de quantização.

Um resumo das informações mais importantes definidas na sintaxe do padrão MPEG-2 de Vídeo e contidas em cada estrutura pode ser visualizado na Tabela 2.

Tabela 2 – Resumo da sintaxe das camadas do MPEG-2 de Vídeo

Nome da Camada	Elementos da sintaxe
Seqüência	Tamanho dos quadros
	Taxa de quadros por segundo
	Taxa de bits por segundo
	Tamanho do buffer de entrada do decodificador
	Parâmetros de codificação programáveis
GOP	Unidade de acesso aleatório
Quadro	Informação de temporização (temporal reference)
Slice	Informação de endereçamento em relação ao quadro
Macrobloco	Modo de codificação
	Vetores de movimento
	Quantização
Bloco	Coeficientes DCT

Para facilitar a decodificação, a ordenação das figuras no fluxo transmitido, ou ordem de codificação, é diferente da ordem segundo a qual as figuras devem ser exibidas, também chamada de ordem de apresentação. A ordem de codificação garante que os quadros codificados de forma preditiva são recebidos, pelo

PUC-Rio - Certificação Digital Nº 0420994/CB

decodificador, sempre após a recepção dos respectivos quadros utilizadas como referência. A Figura 5 exemplifica a disposição dos quadros segundo as ordens de apresentação e de codificação.

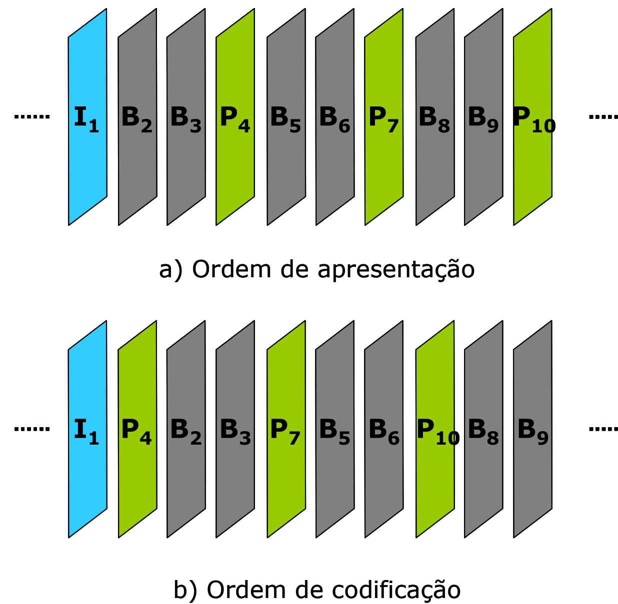

Figura 5 – Ordem de apresentação e codificação dos quadros.

2.2.2. Controle de ocupação do *buffer*

O gerenciamento da ocupação do *buffer* de entrada do receptor é realizado através da manipulação da quantidade de bits de cada figura, da taxa de bits do fluxo codificado e do tamanho do *buffer*, definido pelo codificador. O padrão MPEG-2 sugere a adoção do controle de taxa do modelo TM5 (*Test Model 5*), cujo objetivo é determinar, de forma adaptativa, o tipo de predição temporal para cada macrobloco, a matriz de quantização a ser aplicada e a taxa de transmissão, em bits por segundo, de cada quadro, de forma a evitar situações de *overflow* e *underflow* do *buffer* do receptor. As operações são realizadas nas camadas GOP, de quadro e de macrobloco, e são compostas por três atividades: a alocação de bits, o controle de taxa e a quantização adaptativa.

Na primeira etapa, um número fixo de bits é alocado para cada GOP, obtido através da taxa máxima de transmissão desejada, em bits por segundo, e o número de quadros contidos no GOP. Essa etapa é responsável pela estimativa do número de bits disponível para a codificação do próximo quadro, sendo realizada,

PUC-Rio - Certificação Digital Nº 0420994/CB

portanto, antes da codificação de cada quadro. Depois, à medida que cada quadro é codificado, a ocupação do *buffer* é monitorada, realimentando o sistema e definindo um valor de referência do *quantum* a ser utilizado por cada macrobloco. Por último, a quantização adaptativa manipula o valor de referência de acordo com a atividade espacial em cada macrobloco, de forma a determinar o valor exato do *quantum* relativo à codificação do respectivo macrobloco.

Detalhando o algoritmo especificado pelo TM5, inicialmente, é calculado o total de bits que o GOP deve conter, de acordo com a taxa de bits desejada, a taxa de quadros e o número de quadros dos tipos I, P e B que ainda devem ser inseridos no GOP corrente, conforme a equação abaixo:

(Eq. 2) $$R_{GOP} = (N_i + N_p + N_b) \times \left(\frac{bit_rate}{frame_rate} \right)$$

Em seguida, são determinadas as “medidas de complexidade globais” (*global complexity measures*), denominadas de X_i, X_p ou X_b, de acordo com o tipo de quadro, segundo as seguintes fórmulas:

$$X_i = S_i Q_i$$

(Eq. 3) $$X_p = S_p Q_p$$

$$X_b = S_b Q_b$$

Onde S_i, S_p e S_b representam o número de bits, conforme seu tipo, e as variáveis Q_i, Q_p e Q_b são os parâmetros de quantização médios, calculados de acordo com a média dos valores de quantização utilizados durante a codificação de todos os macroblocos. Todos os parâmetros anteriores referem-se aos últimos quadros codificados, de acordo com o respectivo tipo. Os valores iniciais para os parâmetros de quantização médios são:

$$X_i = \frac{(160 \times bit_rate)}{115}$$

(Eq. 4) $$X_p = \frac{(60 \times bit_rate)}{115}$$

$$X_b = \frac{(42 \times bit_rate)}{115}$$

PUC-Rio - Certificação Digital Nº 0420994/CB

A quantidade de bits (T_i, T_p e T_b) a ser utilizada na codificação do próxima quadro é determinada conforme as equações abaixo. Os parâmetros K_p e K_b são constantes cujos valores atribuídos são 1.0 e 1.4, respectivamente:

$$T_i = máx\left\{ \frac{R_{GOP}}{\left(1 + \frac{N_p X_p}{X_i X_p} + \frac{N_b X_b}{X_i K_b}\right)}, \frac{bit_rate}{8 \times picture_rate} \right\}$$

(Eq. 5) $$T_p = máx\left\{ \frac{R_{GOP}}{\left(N_p + \frac{NbK_p X_b}{K_b X_p}\right)}, \frac{bit_rate}{8 \times picture_rate} \right\}$$

$$T_b = máx\left\{ \frac{R_{GOP}}{\left(N_b + \frac{N_p K_b X_p}{K_p X_b}\right)}, \frac{bit_rate}{8 \times picture_rate} \right\}$$

A segunda etapa refere-se ao controle da taxa e destina-se à obtenção do parâmetro de quantização Q_j referente ao macrobloco *j*, conforme a expressão:

(Eq. 6) $$Q_j = \left(\frac{d_j \times 31}{2}\right)\left(\frac{picture_rate}{bit_rate}\right)$$

onde d_j representa o nível de ocupação do *buffer* virtual e é calculado de acordo com o tipo de quadro.

A terceira etapa determina o valor de cada elemento da matriz de quantização a ser utilizado, a partir da atividade espacial (act_j) calculada para cada macrobloco e o respectivo parâmetro de quantização Q_j. O modelo TM5 especifica que:

(Eq. 7) $$act_j = 1 + mín(vblk_1, vblk_2, \cdots, vblk_8)$$

(Eq. 8) $$vblk_n = \frac{1}{64} \times \sum_{k=1}^{64} \left(P_k^n - P_mean_n\right)^2$$

(Eq. 9) $$P_mean_n = \frac{1}{64} \times \sum_{k=1}^{64} P_k^n$$

PUC-Rio - Certificação Digital Nº 0420994/CB

O valor da atividade (act_j) é, então, normalizado, obtendo-se N_act_j e, finalmente, determinando-se o valor do *quantum* a ser aplicado ao respectivo macrobloco:

(Eq. 10) $mquant_j = Q_j \times N_act_j$

Além do modelo TM5 do MPEG, outros algoritmos para o controle da ocupação do *buffer* do receptor foram propostos, tais como: o algoritmo de alocação de bits proposto por Song e Chun [SoCh03]; o esquema de controle de taxa através de histogramas baseado em estimativas de taxa e distorção (*Rate-Distortion Estimation*), proposto por Hong *et al.* [HYLK03]; e o algoritmo proposto por He e Mitra [HeMi02], que adota um relacionamento linear entre a taxa de codificação de bits e o percentual de zeros nos coeficientes DCT.

PUC-Rio - Certificação Digital Nº 0420994/CB

2.2.2.1. Situações de *overflow* e *underflow*

A ocorrência de *overflow* ou *underflow* do *buffer* de entrada do receptor é gerada por problemas no processo de codificação, alterando a taxa de produção de quadros em relação à taxa nominal. A taxa de codificação de quadros pode sofrer variações devido aos métodos utilizados para o cálculo da quantidade de bits e da taxa de transmissão dos dados, como, por exemplo, o estabelecido pelo TM5. O processo de codificação é responsável por, mesmo havendo pequenos desvios da taxa instantânea de codificação de quadros em relação à taxa nominal, manter a ocupação do *buffer* dentro dos limites adequados, evitando o *overflow* ou *underflow*. Fatores externos ao padrão MPEG também podem causar perturbações no decodificador, tais como os decorrentes da rede de comunicação.

A situação de *underflow* ocorre se a taxa de quadros gerada na saída do codificador for inferior à taxa de quadros prevista para o fluxo de vídeo. Nessa situação, o decodificador consome quadros do *buffer* mais rapidamente do que novos quadros entram no mesmo. Se essa situação persistir por um tempo suficientemente grande, todos os quadros do *buffer* serão retirados e o decodificador não encontrará um novo quadro completo para ser apresentado.

O *overflow* ocorre quando a taxa de entrada de quadros no *buffer* é superior à taxa nominal de apresentação de quadros, por um tempo suficientemente grande.

Os quadros se acumulam do *buffer* até que sua ocupação alcance o limite máximo. A partir desse momento, novos dados que cheguem ao *buffer* são descartados, gerando perda de quadros. Situações de *overflow* também podem acontecer se o algoritmo de alocação de bits, no processo de codificação, for inadequado e não limitar corretamente a quantidade de bits de cada figura.

Situações de *overflow* e *underflow* também podem ser provocadas por falhas na rede de comunicação devido a perdas de pacotes, fazendo com que a seqüência de quadros que entrem no *buffer* seja diferente da gerada pelo codificador. A perda de quadros, no entanto, prejudica a correta operação do decodificador, pois, caso tenham sido perdidos quadros configurados para terem sua apresentação repetida, o decodificador não recebe essa informação, adiantando a exibição do quadro seguinte. O funcionamento é normalizado após o recebimento de um novo GOP, que provê informações temporais para o acesso aleatório do fluxo.

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

3
Trabalhos Relacionados

Vários trabalhos têm sido publicados sobre a edição de dados audiovisuais, discutindo os mais diversos problemas envolvidos, podendo ser classificados de acordo com seus objetivos, técnicas e algoritmos utilizados. Em relação a vídeos MPEG-2, na maioria dos trabalhos será discutida a maneira como foram solucionados, se existentes, os mecanismos de corte e concatenação de trechos de vídeos e o controle do *VBV buffer*. Em se tratando do controle de versão, será discutida o modelo de versionamento, levando em consideração os aspectos dos mecanismos de armazenamento, compartilhamento e recuperação do histórico das edições. Além disso, serão abordados o problema da mesclagem de contribuições e os métodos de prevenção para que as versões não sejam sobrescritas incorretamente.

Considerando todos os trabalhos analisados, é importante notar que nenhum deles realiza o controle de versão para a edição cooperativa de vídeo MPEG-2. Pode-se afirmar que nenhum deles apresentou soluções de integração entre um sistema que gerenciasse versões e edições de vídeo no domínio comprimido. Os trabalhos descritos a seguir foram selecionados analisando-se os algoritmos propostos e sua aplicabilidade em sistemas comparáveis a esta dissertação, mesmo que sejam necessárias adaptações ou modificações nos algoritmos originais.

Neste capítulo, serão apresentados alguns sistemas de controle de versão e mecanismos utilizados para a segmentação e concatenação dos trechos audiovisuais. Além disso, serão apresentados sistemas de edição de vídeo colaborativos.

3.1.
Sistemas de Controle de Versão

Os sistemas de controle de versão foram desenvolvidos para versionar qualquer tipo de arquivo (texto ou binário). Seus principais objetivos são armazenar os itens que são produzidos no desenvolvimento, permitir o acesso de

maneira controlada a quaisquer versões desses elementos, armazenar informações de histórico dos itens de forma a estabelecer a base para o controle da evolução do produto, e reter informações que permitam automatizar o acesso aos conjuntos de itens de configuração, que compõem o produto num específico estado de seu desenvolvimento.

Atualmente, existem no mercado diversos produtos disponíveis para o controle de versões, variando quanto às funcionalidades e licenças oferecidas. Um traço em comum é que algumas delas apresentam custos de licença relativamente alto, muitas vezes desestimulando ou mesmo inviabilizando a adoção dessas ferramentas por parte de empresas de pequeno porte, como é o caso dos sistemas *Microsoft Visual SourceSafe* [MVSS04] e o *Rational Clear Case* [RCC06]. Optar por uma solução comercial geralmente está relacionado à garantia, pois as soluções livres não se responsabilizam por erros no software e perdas de informações, apesar das soluções livres poderem ter melhor desempenho e segurança que as comerciais.

Por outro lado, há diversas ferramentas de controle de versão de boa qualidade, como o *Concurrent Version System* [WCVS01], *SubVersion* [SubV06] e o *Revision Control System* [RCS91], desenvolvidas sob o conceito de software livre e, portanto, de domínio público.

Com a necessidade crescente de ferramentas de controle de versão e com os restritivos custos das ferramentas comerciais de controle de versão, o *Concurrent Version System* e o *SubVersion* aparecem como alternativas muito atraentes para organizações que necessitam aperfeiçoar o processo de desenvolvimento.

3.1.1. Conceitos básicos

Cada sistema tem sua particularidade, mas a maioria deles compartilham alguns conceitos básicos. Todos os arquivos e diretórios que compõem um projeto ficam sob a responsabilidade do sistema de controle de versão num local denominado de repositório. À medida que o projeto evolui, o repositório passa a guardar múltiplas versões dos arquivos que compõem o projeto. Essas múltiplas versões, organizadas em revisões, funcionam como uma foto de todos os arquivos do projeto em um determinado momento do tempo. Revisões antigas são mantidas

PUC-Rio - Certificação Digital Nº 0420994/CB

e podem ser recuperadas e analisadas a qualquer momento. Geralmente, o acesso é feito por um cliente pela rede (via *socket*) e pode ser feito localmente quando o cliente está na mesma máquina do servidor..

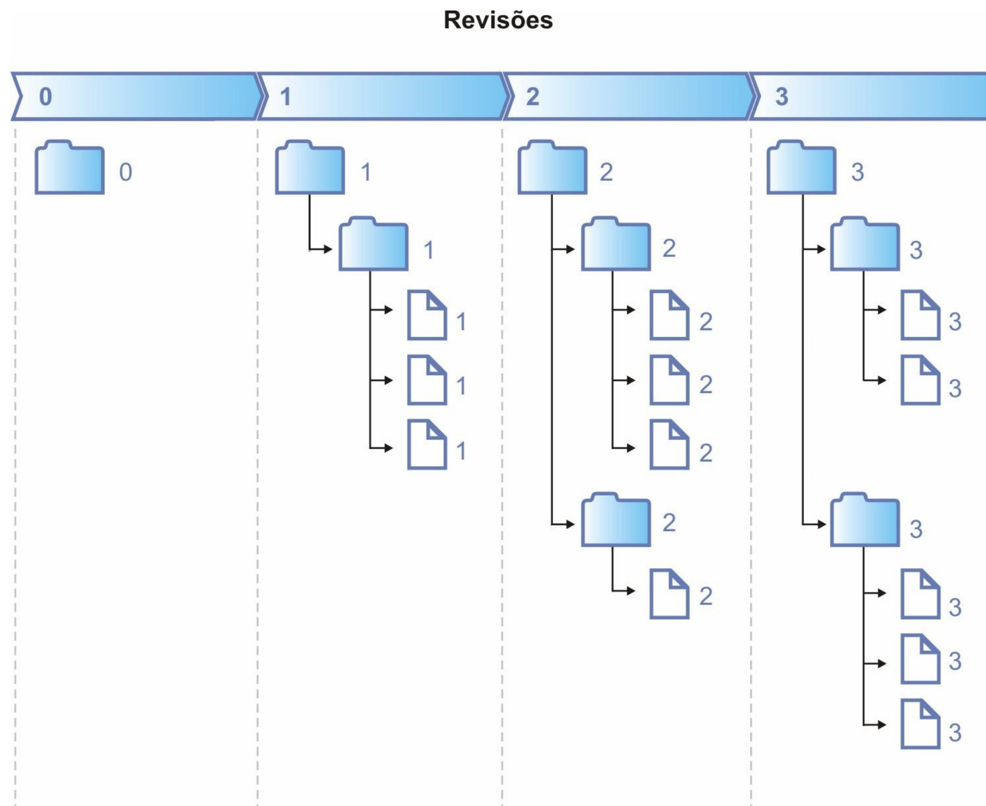

Figura 6 – Exemplo de múltiplas versões organizadas em revisões

O repositório armazena a informação - um conjunto de documentos - de modo persistente num sistema de arquivos ou num banco de dados específico. Cada servidor pode ter vários sistemas de controle de versão e cada sistema pode ter diversos repositórios, limitando-se na capacidade de gerenciamento do software e também no limite físico do hardware. Geralmente um repositório possui um endereço lógico que permite a conexão do cliente. Esse endereço pode ser um conjunto IP/porta, uma URL, um caminho do sistema de arquivos etc.

Cada usuário possui em sua máquina uma cópia local (também chamada de *working copy*) somente da última versão de cada documento. Essa cópia local geralmente é feita num sistema de arquivos simples. A cada alteração relevante do desenvolvedor/editor é necessário atualizar as informações do servidor submetendo as alterações. O servidor então guarda a nova alteração juntando com todo o histórico mais antigo. Se o usuário deseja atualizar sua cópia local é

PUC-Rio - Certificação Digital Nº 0420994/CB

necessário atualizar as informações locais, e para isso é necessário baixar novidades do servidor.

3.1.2. Envio e resgate de versões

Como dito anteriormente na Seção 3.1.1, o sistema de controle de versão armazena todo o histórico de desenvolvimento do documento, desde o primeiro envio até sua última versão. Isso permite que seja possível resgatar uma determinada versão de qualquer data mais antiga, evitando desperdício de tempo no desenvolvimento para desfazer alterações quando se toma algum rumo equivocado.

Cada envio das alterações é na maioria dos sistemas chamado de *commit* (as vezes *submit*), que seria o mesmo que efetivar ou submeter as alterações no repositório. Cada envio produz uma nova versão no repositório e é armazenado como "uma fotografia" do momento. A fim de minimizar conflitos de versões, facilitar no desfazer de alterações e também no controle do histórico, recomenda-se, em todos os sistemas [SubV06] [WCVS01] [RCC06] e [MVSS04], que uma alteração seja enviada cada vez que o software estiver minimamente estável, como por exemplo, a cada nova parte (um novo método) ou a cada alteração relevante que esteja funcionando corretamente. Não é recomendável o envio quando o documento como um todo possa causar alguma dificuldade no desenvolvimento de outro colaborador, como por exemplo um código não compilando ou com algum bug que comprometa a execução geral.

Em sistemas no estilo do [WCVS01], o controle de versão é feito por cada documento individualmente. Assim, o resgate de uma revisão num determinado momento, não ficam alinhadas em função da versão, como demonstrado na Figura 7. Nos sistemas mais modernos, como o [SubV06], o controle de versão é feito por cada envio ao servidor, e portanto, há sempre uma versão global para todos os documentos. Toda revisão (fotografia) de qualquer momento será sempre uma coluna alinhada como mostra a Figura 8.

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

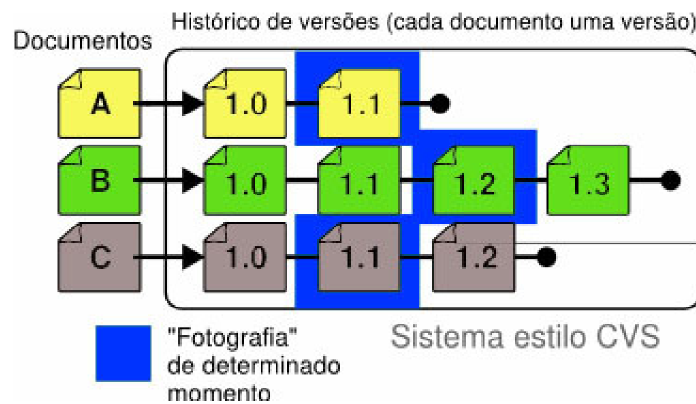

Figura 7 – Histórico de versões no sistema estilo CVS.

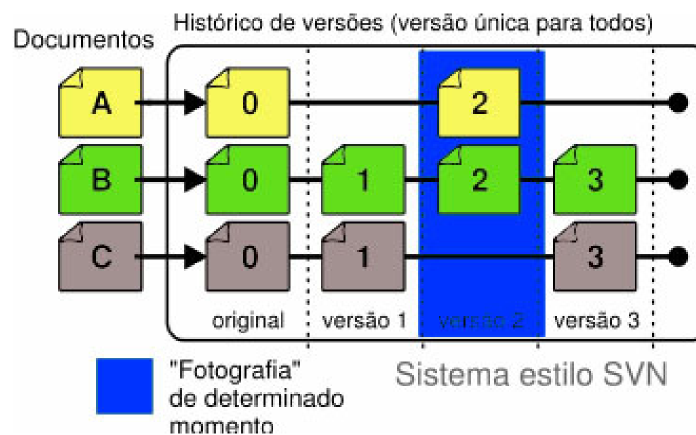

Figura 8 – Histórico de versões no sistema estilo SVN.

3.1.3. Trabalho Colaborativo

Os sistemas de controle de versão são bastante úteis quando se trabalha com vários usuários de modo simultâneo, resolvendo de modo muito satisfatório conflitos de versões [WCVS01] [SubV06]. A maioria dos sistemas possuem diversos recursos como mesclagem ou fusão de versões, e divisão de ramo do desenvolvimento para auxiliar nessas tarefas.

Com relação ao trabalho em equipe num sistema de controle de versão, há basicamente dois métodos (ou filosofias) relevantes de edição que são o método de *Lock-Modify-Unlock* (Trava-Modifica-Destrava, também conhecido por *Exclusive Lock*) e o método *Copy-Modify-Merge* (Copia-Modifica-Resolve, também conhecido por *Optimistic Merge*). Esses métodos visam minimizar os

conflitos das edições e permitir o compartilhamento de arquivos entre os usuários e serão mais detalhados e exemplificados na Seção 4.2.

Optimistic Merge é um método de conflitos otimista que é geralmente padrão na maioria dos sistemas abertos, como o [WCVS01] e [SubV06]. Esse método presume que os conflitos de edições são tão pequenos e pontuais — se houver relativa atualização freqüente do repositório — que a grande maioria das fusões podem ser feitas de forma automática, restando para fusões manuais somente casos em que a alteração for feita num mesmo ponto de um mesmo arquivo. Segundo [SubV06] a prática demonstra que essa política é, na verdade, também a mais realista. No entanto, nesse método, a operação de *merge* não funciona corretamente quando se trata de arquivos binários. Somente arquivos textos e mesmo assim, em alguns casos, podem apresentar conflitos e inconsistências de dados após a mesclagem. Em casos como esses, é preciso uma intervenção manual de algum usuário para resolver os conflitos e inconsistências.

A vantagem da *Optimistic Merge* é que os editores podem trabalhar paralelamente editando um mesmo arquivo em qualquer momento. Após a edição e envio das alterações para o repositório, o sistema realiza o *merge* caso haja uma nova versão do arquivo enviado. Os conflitos após a fusão automática (*merge*) em arquivos textos não são freqüentes, como é afirmado nos trabalhos [SubV06] [WCVS01]. Já em arquivos binários como os vídeos, a mesclagem de versões através da comparação entre esses tipos de arquivos é bastante complexa. Em nenhum dos sistemas citados, as contribuições de cada um dos editores de um vídeo dificilmente poderiam ser mescladas numa única versão, pois, uma vez feito o *merge*, o arquivo resultante possivelmente estaria corrompido e impossibilitado de ser usado.

Nos casos de haver muitas modificações simultâneas sem intervalos pequenos de atualização mútua num mesmo arquivo, é possível que a fusão manual se torne tão complicada a ponto de desperdiçar parcialmente ou totalmente o trabalho de um ou de outro usuário do sistema. Esse é um dos principais motivos que leva alguns sistemas, como as soluções comerciais [RCC06] e [MVSS04], a adotar outra política de trabalho: *Exclusive Lock*. Essa política consiste, basicamente, em deixar apenas um usuário editar um arquivo de cada vez, permanecendo o arquivo bloqueado enquanto não houver um reenvio

PUC-Rio - Certificação Digital Nº 0420994/CB

(atualização) do editor ao servidor. É possível realizar o mecanismo de fusão (*merge*) nessa política, mas apenas após a liberação (ou desbloqueio) do editor.

É importante observar que nenhum desses trabalhos relacionados ao controle de versão está integrado a uma ferramenta de edição de vídeo. Como já mencionado anteriormente, o sistema proposto nesta dissertação tem como um de seus requisitos a integração de uma ferramenta comercial de edição de vídeo a um sistema de controle de versão.

3.2.
Sistemas de Edição de Vídeo Colaborativo

Nesta seção, serão apresentados alguns trabalhos que utilizam mecanismos de segmentação e remontagem do formato de vídeo MPEG. Em seguida, serão apresentados alguns sistemas de edição de vídeo colaborativos.

3.2.1.
Mecanismos de Segmentação e Remontagem do MPEG

Um dos mecanismos de segmentação e remontagem do MPEG feito para prover o compartilhamento e acesso aos vídeos modificados entre usuários foi apresentado por *Talreja* e *Rangan* [TaRa97]. Os autores descrevem um modelo estrutural de indexação hierárquica de vídeos, de forma a acelerar o processo de acesso randômico, como meio para realizar operações de edição diretamente nos fluxos comprimidos, tais como corte, cópia e concatenação de segmentos de fluxos MPEG de Sistemas. No trabalho, também é descrito um sistema de gerenciamento de vídeo, indexando diferentes conteúdos e formatos de mídia.

Em relação às operações de edição, *Talreja* e *Rangan* discutem algoritmos de corte e concatenação dos fluxos de áudio e vídeo que compõem um fluxo MPEG de forma correlacionada, ou seja, mantendo a sincronização entre os mesmos e sendo adequados tanto para fluxos de programa quanto para fluxos de transporte. Em relação à edição de fluxos de vídeo, os autores discutem operações de corte realizadas nos limites de um GOP e de uma figura, salientando a facilidade da manipulação nos limites de GOP fechados. A edição em qualquer parte do fluxo requer a eliminação das dependências temporais porventura

PUC-Rio - Certificação Digital Nº 0420994/CB

existentes entre as figuras que estarão presentes no fluxo resultante e as que serão eliminadas.

Um outro trabalho relacionado ao mecanismo de segmentação está descrito em [VFSC06]. Nesse trabalho é apresentado um método de identificação das transições entre as tomadas de cenas de um fluxo comprimido de MPEG-1 e MPEG-2 de vídeo e é proposto um algoritmo de segmentação baseado na assinatura de codificação [VFSC06]. Basicamente esse mecanismo de segmentação atua sobre as tomadas de cenas automaticamente identificáveis por um processo composto de passos onde se verifica os padrões da codigicação dos quadros de cada GOP. Como a maioria dos cortes desses segmentos aparece em quadros independentes, a segmentação não apresentou grandes problemas com relação aos quadros das bordas.

Também relacionado ao mecanismo de segmentação do vídeo está descrito em *Cheng et al.* [ChZZ04]. *Cheng* apresenta um esquema de conversão de tipo de quadro, requerendo a aplicação da transformada I-DCT, ou seja, através da decodificação parcial dos coeficientes DCT, de forma a efetuar a concatenação de segmentos de vídeo.

Tanto os algoritmos de edição correlacionada dos fluxos de vídeo e áudio quanto a conversão de tipo dos quadros poderiam ser utilizados no sistema proposto nesta dissertação, entretanto, o mecanismo de manipulação e eliminação das dependências temporais em relação aos quadros consecutivos nas bordas entre os segmentos editáveis, será realizado, por simplicidade, através do mecanismo de duplicação de quadros dependentes, como pode ser visto em maiores detalhes na Seção de Segmentação do Vídeo do Capítulo 4. Além do mais, o mecanismo que Cheng descreve, apesar de funcionalmente estar de acordo com o sistema proposto, não condiz com a manutenção da qualidade do vídeo editado. Como se sabe, decodificar para em seguida recodificar um arquivo MPEG-2 significa perda na qualidade do vídeo original.

Meng e *Chang* publicaram diversos trabalhos sobre a edição de fluxos de vídeo comprimidos, sem que seja requerida a completa decodificação dos mesmos e de forma a manter a conformidade com o padrão MPEG [MeCh96] [MeCh97]. Nos artigos são discutidos métodos de controle de ocupação do *buffer* do decodificador em aplicações de corte e concatenação de trechos de vídeos, uma vez que o processo de recodificação necessita seguir os mesmos tipos de

PUC-Rio - Certificação Digital Nº 0420994/CB

restrições que a codificação original, particularmente os referentes à taxa de bits e ao controle da ocupação do *buffer* do decodificador.

A operação de concatenação de fluxos de vídeo também impacta no controle da ocupação do *buffer* do decodificador. Situações de *overflow* ou *underflow* podem ocorrer e dependem do cálculo de ocupação do *buffer* realizado para cada fluxo de vídeo. Caso o nível de ocupação do *buffer* após o último quadro do primeiro fluxo de vídeo for maior que o nível previsto, durante o processo de codificação do segundo fluxo de vídeo poderá haver *overflow* do *buffer*. Também devido ao fato dos codificadores do primeiro e do segundo fluxo de vídeo não garantirem uma taxa de codificação estritamente fixa, podendo haver pequenos desvios que são compensados em outros quadros, podem ocorrer situações de *overflow* ou *underflow* após a operação de concatenação.

O problema de *overflow* é resolvido através da inserção de bits de enchimento sempre que a ocupação do *buffer* atingir um nível alto. Para evitar situações de *underflow*, os autores propõem dois métodos: a inserção de seqüências sintéticas de desvanecimento entre os trechos de vídeo a serem concatenados; e o descarte seletivo de coeficientes DCT antes da operação de concatenação.

Apesar de *Meng* e *Chang* terem desenvolvido diversos trabalhos sobre a edição em fluxos MPEG, suas técnicas consideram apenas a manipulação de vídeos e requerem a decodificação parcial para a obtenção dos coeficientes DCT.

3.2.2. Ferramentas de Edição de Vídeo Colaborativo

Em [ScHK04], foi apresentado o sistema *FilmEd* que tem como principais funcionalidades a indexação, busca, descrição, anotação e discussão colaborativa em tempo real de vídeo MPEG-2. Esse sistema dá suporte à colaboração entre grupos, e seus usuários são capazes de abrir arquivos de vídeo MPEG-2 e compartilhar ferramentas que colaborativamente segmentam, buscam, descrevem, anotam e discutem os vídeos de interesse. Para que fosse possível a detecção automática de tomadas de cenas e a segmentação de arquivos MPEG-2, o *FilmeEd* utilizou um kit comercial de desenvolvimento de software conhecido por *Mediaware SDK* [Mware06]. O sistema possui uma interface simples para os

PUC-Rio - Certificação Digital Nº 0420994/CB

usuários. Entretanto, a versão atual não se preocupa em armazenar o histórico das anotações, discussões nem as versões das alterações realizadas.

Outro trabalho desenvolvido pela Forbidden Technologies, o FORscene [FScene07] foi projetado para permitir a edição de vídeo colaborativa. Este trabalho tem como principal vantagem executar integrado a *browsers,* pois o sistema foi implementado como uma aplicação *web* e toda sua interface foi desenvolvida em applet java. É um sistema que não precisa de instalação, codecs ou configurações nas máquinas e possui vários conceitos web 2.0 implementadas.

O *FORscene* não segmenta nem concatena arquivos de vídeo no formato MPEG-2, somente AVI, MPEG-1 e MJPEG. Nele, é possível a edição de trechos dos vídeos, redimensionamento para variadas plataformas como celulares e iPods, bem como anotação e *chat* entre os colaboradores. Assim como o *FilmEd*, ele também não realiza o controle das versões das alterações dos vídeos.

Outros trabalhos como o IBM MPEG-7 Annotation Tool [IMAT06], Movie Tool [Rico06], ZGDV VIDETO [Videto06], COALA LogCreator [Coala06], CSIRO's CMWeb tools [CSIRO06], Microsoft's MRAS [Bargeron01], apresentam sistemas e ferramentas que também oferecem suporte à indexação e anotação de vídeo MPEG1 e MPEG-2, entretanto usados somente para ambientes não distribuídos em que essas anotações podem ser salvas e compartilhadas assicronamente.

Em Zhang *et al.* [Zhang03] foi apresentada uma ferramenta capaz de identificar e representar o conteúdo do vídeo MPEG, além de aplicar técnicas de representação de conhecimento e análise do conteúdo do vídeo para a construção indexada, possibilitando uma ágil recuperação e seleção de vídeos. Além disso, é descrito um sistema que possibilita a interação do usuário com os objetos de vídeo. Nele, é utilizado um mecanismo de detecção de tomadas de cenas denominado de *parser.* A função do *parser* é realizar o reconhecimento das tomadas ou *shots* através de parâmetros denominados de chaves, que verificam os tipo das tomadas detectadas, como pode ser exemplificado na Figura 9. Basicamente, o mecanismo de segmentação das tomadas de cenas recodifica os tipos dos quadros das bordas dos segmentos. No exemplo, são identificados os repórteres âncoras de um noticiário de telejornal, os comerciais de tv, as notícias gerais do telejornal e as notícias do tempo. O passo seguinte ao *parsing* tem como objetivo classificar e indexar todas as tomadas de cenas verificadas e inseri-las

PUC-Rio - Certificação Digital Nº 0420994/CB

numa base de dados. Esse segundo passo é denominado de *indexing*. Em seguida, oferece-se o mecanismo denominado de recuperação e seleção (*retrieval* e *browsing*, respectivamente) dos vídeos armazenados. Nesse último passo, usuários podem acessar o banco de dados através de consultas baseadas em textos e/ou exemplos de imagens, ou navegar interagindo com ícones significativos. Usuários também podem navegar pelos resultados da consulta. E tudo isso com o enfoque na visualização gráfica para facilitar a recuperação e seleção.

Nessa dissertação não serão aplicados os mecanismos de reconhecimento, análise e classificação dos segmentos detectados para a construção da árvore de versionamento composta pelos segmentos do vídeo. Esse mecanismo poderia ser uma extensão ao sistema proposto. Além disso, o mecanismo de seleção e navegação implementados neste trabalho são bem mais simples que os descritos na ferramenta de Zhang.

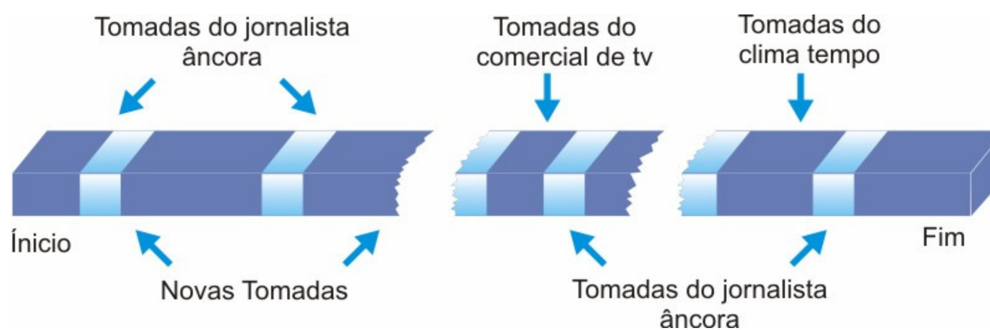

Figura 9 – Detecção das Tomadas de Cenas de um Noticiário de Telejornal.

Ichimura e *Matsushita et al.* [IcMa05] desenvolveram um sistema de edição de vídeo não comprimido para *web*, onde os vídeo clipes são coletados através da internet e podem ser compartilhado entre os usuários. O sistema permite aos usuários reutilizar facilmente os clipes dos outros usuários permitindo a filmagem por diferentes câmeras e formatos digitais. Além disso, o sistema é capaz de extrair informações como datas e hora do vídeo gravado e de automaticamente sincronizá-las. O protótipo do sistema faz uso basicamente de três mecanismos: o de coleta ou armazenamento dos vídeos; o de compartilhamento dos vídeos; e de edição dos vídeos. Na coleta, o vídeo capturado das câmeras digitais diminuem a qualidade do vídeo original, pois é feita uma conversão do vídeo digital de alta definição para um arquivo de vídeo de baixa qualidade. Essa conversão é necessária, pois os arquivos de vídeo originais são muito grandes para serem exibidos/executados na internet. Enquanto o vídeo é capturado e convertido os

PUC-Rio - Certificação Digital Nº 0420994/CB

vídeo clipes são montados e então armazenados num banco de dados. Esse sistema é denominado de *VideoBlock* e provê aos usuários uma interface via *browser* para cortar, colar e buscar vídeos armazenados no banco de dados. Além disso, provê um exibidor (tocador) que apresenta um *preview* dos vídeos nos PCs dos usuários. Todas as operações de edição do usuário são armazenadas no servidor através da linguagem declarativa S*ynchronized Multimedia Integration Language* (SMIL), responsável por representar as mídias do vídeo e sincronizar o conjunto independente dos objetos multimídia. Com o uso de SMIL as edições dos usuários não modificam o arquivo do vídeo.

Outros trabalhos como o de *Hitch* [Hitch06] e *Silver* [CLMS02] também permitiam a edição de vídeo remota pela internet através de navegadores *web* (*browsers*). Tal qual vários trabalhos anteriores, [Hitch06], [CLMS02], [IcMa05] também não visam controlar as versões das edições de vídeo.

PUC-Rio - Certificação Digital Nº 0420994/CB

4
Modelo de Dados do Controle de Versões para Edição Cooperativa de Vídeo

Neste Capítulo 4 são apresentados a estrutura de dados utilizada nesta dissertação para o controle de versões (árvore de versionamento) das edições de um vídeo e o controle de acesso concorrente aos dados (nós dessa árvore de versionamento). Em seguida, são apresentados os mecanismos de segmentação e remontagem do vídeo MPEG-2. Por fim, é descrito o mecanismo de fusão de versões.

PUC-Rio - Certificação Digital Nº 0420994/CB

4.1.
Árvore de Versionamento de um Vídeo

A estrutura de dados usada para o controle de versões é uma árvore genérica, denominada *árvore de versionamento de um vídeo,* cujas sub-árvores de cada nó estão ordenadas, e as folhas descrevem um conjunto ordenado de trechos do vídeo. O nó raiz, os outros nós internos e as folhas da árvore de versionamento possuem um rótulo associado.

O rótulo de um nó é um conjunto de informações composto por um número de identificação único, um nome e um usuário de criação do nó. As folhas da árvore de versionamento têm por finalidade descrever o conteúdo de um trecho do vídeo. Logo, cada folha aponta para um vídeo no sistema de arquivos. Por isso, o rótulo de cada uma das folhas também é composto por um elemento de descrição do vídeo apontado. Esse elemento é responsável por descrever o nome do arquivo, o número de quadros, o formato do vídeo e o endereço relativo onde o arquivo MPEG-2 se encontra.

Uma árvore de versionamento possui um número de identificação único, um nome, um elemento que descreve a sua versão e um usuário de criação. Esse elemento que descreve a versão é composto por um número de identificação único, um nome, um tipo de versão e um *timestamp*. O tipo da versão é também

PUC-Rio - Certificação Digital Nº 0420994/CB

conhecido como etiqueta da revisão e se refere a uma data específica ou a uma linha de desenvolvimento das alterações feitas sobre a árvore de versionamento.

Um exemplo de uma árvore de versionamento é ilustrado na Figura 10. A árvore de versionamento é composta pela raiz de nome **A** e número de identificação zero, representado por (A,0), e tem três sub-árvores, ou, equivalentemente, o nó **A** tem 3 filhos. Os nós **B** e **C** são folhas, e portanto, não apontam para nenhum outro nó. O nó interno **D** possui dois nós filhos, sendo o primeiro a folha **E** seguido de **F**, respectivamente. Como dito anteriormente, todos os rótulos dos nós dessa árvore possuem um número de identificação e nome, e somente os rótulos de cada uma das folhas **B**, **C**, **E** e **F** possuem o elemento de descrição do conteúdo do vídeo.

A versão 1.0 da árvore de versionamento não foi criada a partir de uma outra estrutura, e portanto, é considerada uma nova *release* do vídeo. Tanto a árvore de versionamento como os nós dela foram criados por um único usuário e todos os nós podem ser acessados por outros usuários. Dessa forma, é preciso controlar o acesso concorrente aos nós, para que mais de um usuário possa editar a mesma árvore de versionamento ao mesmo tempo. Este controle concorrente é discutido na seção seguinte.

Ainda nesse exemplo, a representação da árvore de versionamento segue o padrão <raiz sa_1 sa_2...sa_n>. Com esta notação, a árvore é representada por **<A <B> <C> <D <E> <F>>>**.

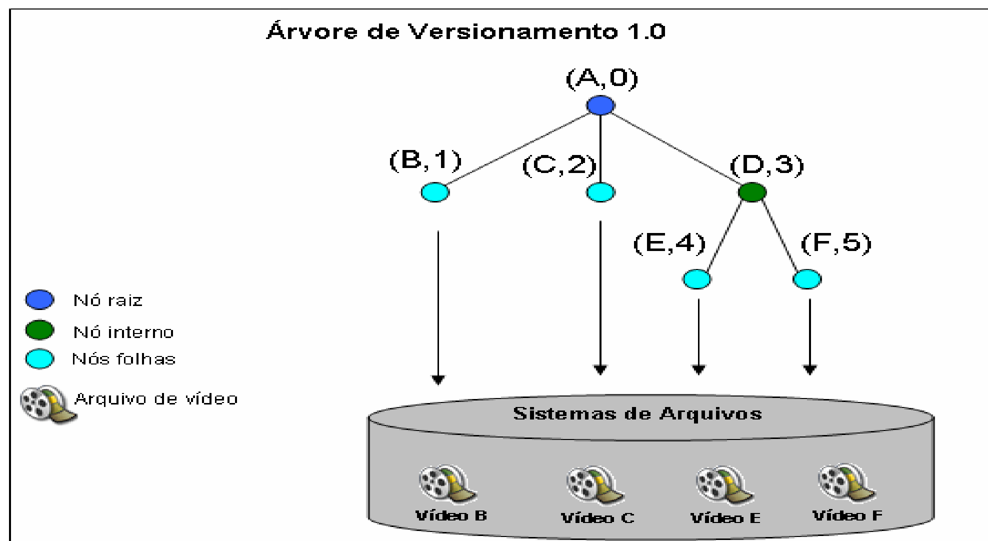

Figura 10 – Exemplo de uma árvore de versionamento de vídeo

Uma árvore de versionamento pode compartilhar nós com outras árvores. O compartilhamento de nós permite que um nó tenha mais de um pai. Entretanto, cada um desses pais deve pertencer a árvores distintas. No exemplo da Figura 11, o nó **C** da árvore de versionamento 1.0 é filho do nó **A** da mesma árvore como também é filho do nó **A’** da 1.1.

Uma árvore de versionamento pode ser *derivada* ou *de origem*. O relacionamento de derivação entre duas árvores de versionamento é representado por ponteiro de derivação, da árvore de versionamento derivada para a árvore de versionamento de origem. No exemplo da Figura 11, a primeira árvore de versionamento 1.0 é considerada de origem e a árvore de versionamento 1.1 é derivada da primeira.

Os nós da árvore de versionamento também podem ser considerados *derivados* ou *de origem*. O relacionamento de derivação entre dois nós também é representado por ponteiro de derivação, do nó derivado para o nó de origem.

No exemplo da Figura 11, uma árvore de versionamento 1.1 é derivada da árvore de versionamento 1.0. A árvore de versionamento 1.1 possui nós derivados dos nós da árvore de versionamento de origem. Além disso, a árvore de versionamento 1.1 possui nós compartilhados com a árvore de versionamento de origem. Assim, a árvore de versionamento 1.1 não possui uma cópia do nó **C** em sua estrutura, assim como não possui uma cópia do nó **F**. Os nós **C** e **F** são considerados nós compartilhados por essas duas árvores de versionamento citadas. A árvore de versionamento 1.1 possui uma raiz **A’** com três filhos ordenados, consecutivamente, por **B’**, **C** e **D’**. Note que **B’** tem um ponteiro de derivação para o nó B e **D’** também tem um ponteiro de derivação para **D**. Da mesma forma, o nó **E’** é uma derivação e representa uma nova versão de **E**. Embora **D’** seja uma derivação que representa uma nova versão de **D**, eles ainda assim compartilham o mesmo filho **F**. A árvore de versionamento 1.0 é representada pela notação da equação 1 e a árvore de versionamento 1.1 pela equação 2.

(Eq. 11) α = < A <B> <C> < D <E> <F> > >

(Eq. 12) β = < A’ <B’> <C> < D’ <E’> <F> > >

PUC-Rio - Certificação Digital Nº 0420994/CB

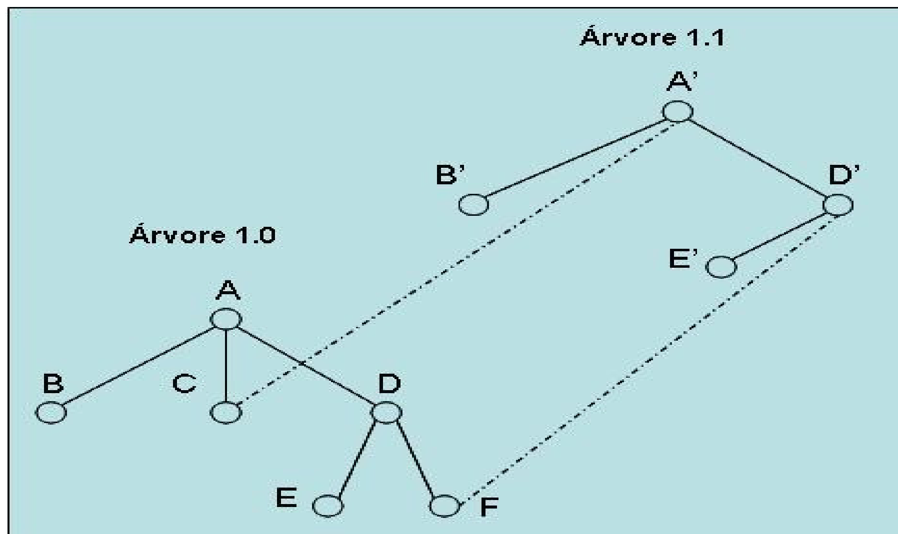

Figura 11 – Compartilhamento de nós entre árvores de versionamento

PUC-Rio - Certificação Digital Nº 0420994/CB

4.2. Acessos Concorrentes aos Nós

O acesso simultâneo a um nó de uma árvore de versionamento é permitido apenas quando se deseja fazer sua leitura. Quando é preciso alterar um nó, o sistema deve prevenir incompatibilidades e inconsistências no objeto manipulado. O controle de acesso concorrente e o compartilhamento de nós de uma árvore de versionamento são requisitos essenciais no sistema desta dissertação.

Alguns métodos podem ser aplicados com o intuito de controlar o acesso simultâneo aos nós de uma árvore de versionamento, como já comentado no capítulo anterior, são eles:

- Método *Lock-Modify-Unlock* (Bloqueia-Modifica-Desbloqueia também conhecida por *Exclusive Lock*)
- Método *Copy-Modify-Merge* (Copia-Modifica-Mescla também conhecida por *Optimistic Merge*)

Lock-Modify-Unlock

O método *Lock-Modify-Unlock* ou *Exclusive Lock* é uma solução onde somente é permitido que um usuário por vez possa modificar um mesmo objeto, no caso, o mesmo nó da árvore de versionamento. A política usada é baseada em

locks, ou seja, bloqueios de objetos pelo usuário antes de obter permissão de leitura e escrita.

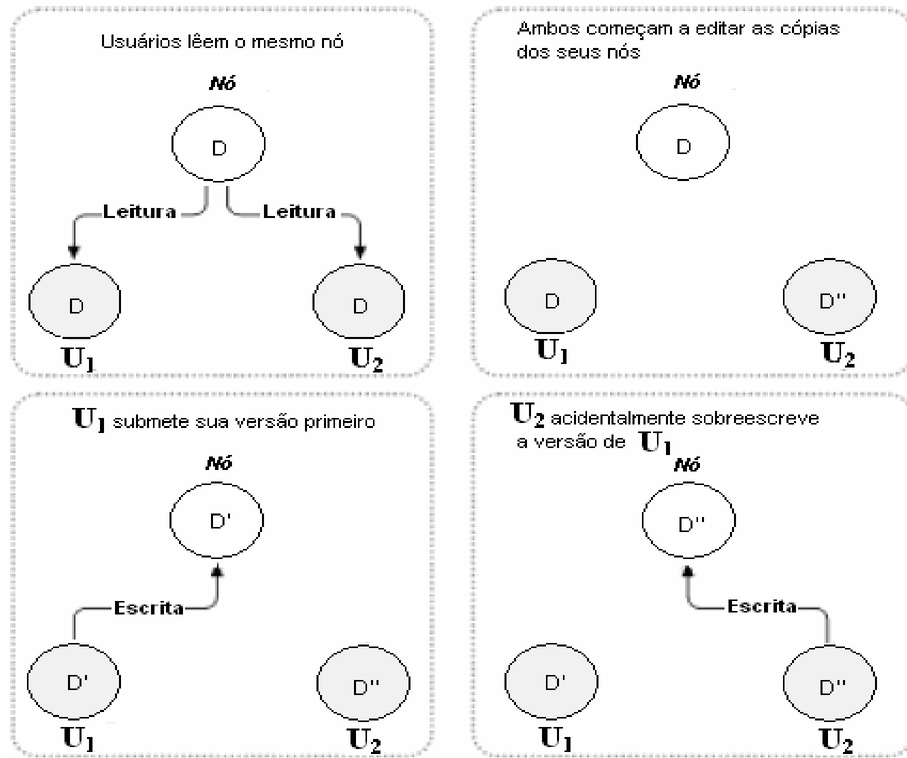

Figura 12 – Problema do acesso simultâneo de um nó.

No exemplo da Figura 12, $\mathbf{U_1}$ poderia bloquear o nó **D**, pois este não estaria sendo usado por nenhum outro usuário. Em seguida, $\mathbf{U_2}$ tentaria bloquear o nó de vídeo **D**, porém sem êxito, pois **D** já está de posse de $\mathbf{U_1}$. Somente após a liberação completa (c*ommit*) ou desbloqueio deste nó **D** por parte de $\mathbf{U_1}$, $\mathbf{U_2}$ poderia bloqueá-lo para, enfim, ter a permissão de escrita. As versões de cada um estariam armazenadas e nunca sobrescritas. A Figura 13 descreve essa solução.

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

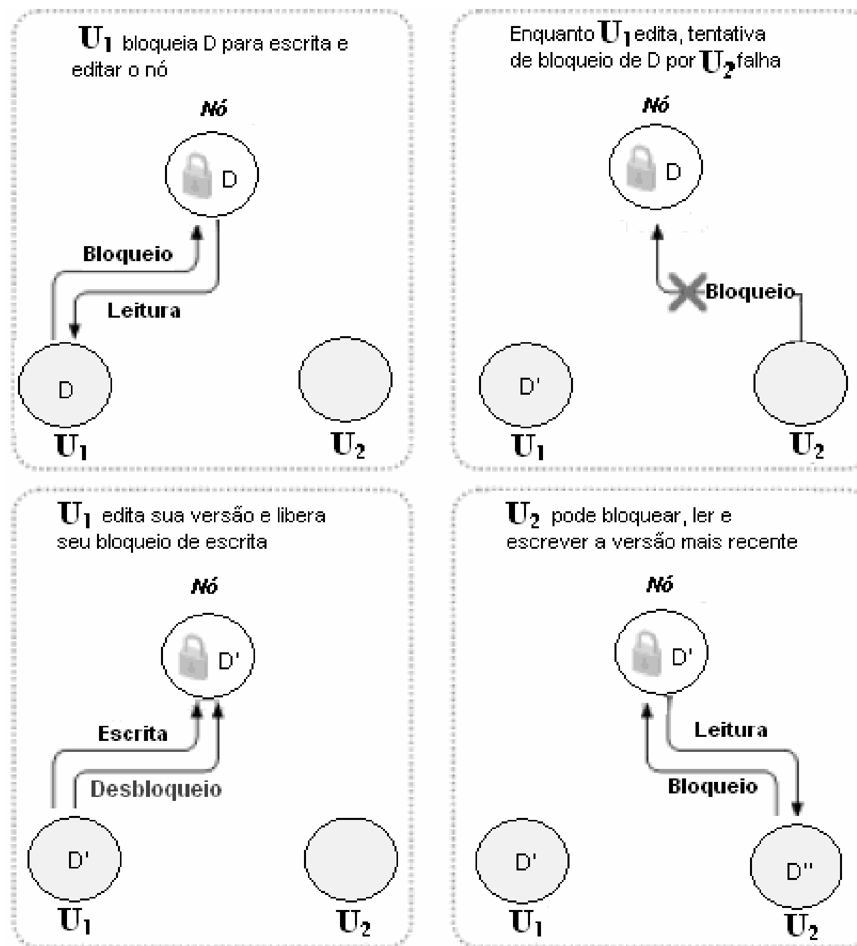

Figura 13 – Método *Lock-Modify-Unlock ou Exclusive Lock*

Apesar desse método construir resultados satisfatórios, ele pode gerar alguns problemas, como a espera pelo desbloqueio por parte daquele que travou o nó primeiramente. Caso um outro usuário queira ter a permissão para escrita, e o primeiro usuário esqueça de liberar o nó, essa espera atrasaria o trabalho.

Um segundo problema vem do fato de que bloqueios podem criar uma falsa sensação de segurança nas edições. Suponha que $\mathbf{U_1}$ bloqueia no modo exclusivo e edita um nó **A** qualquer, e $\mathbf{U_2}$ bloqueia no modo exclusivo e edita um nó **B** qualquer. Mas suponha que o nó **B** dependa de **A**, pois **B** é filho dele. Logo, as modificações feitas neles são semanticamente inconsistentes, o que resultaria num produto final incompatível.

A Seção 4.3 descreve um método que define uma hierarquia de granularidade de nós da árvore de versionamento e que controla o acesso concorrente de todos os nós baseado em *locks*.

PUC-Rio - Certificação Digital Nº 0420994/CB

Copy-Modify-Merge

O método *Copy-Modify-Merge* ou *Optimistic Merge* é um método otimista de controle de acessos concorrentes onde se presume que os conflitos de edições são pequenos e pontuais. Esse método é uma solução onde cada usuário cria uma cópia do nó localmente. Em seguida, cada usuário modifica suas respectivas cópias, para finalmente publicá-las numa base de dados central.

Tal método é ilustrado nas Figura 14 e Figura 15. Suponha, no exemplo, que $\mathbf{U_1}$ e $\mathbf{U_2}$ criaram cópias de um mesmo nó **D**. Eles trabalham concorrentemente e fazem edições em D, em suas respectivas cópias. Primeiramente, $\mathbf{U_2}$ salva suas alterações e as submete à base de dados. Depois de um certo tempo, quando $\mathbf{U_1}$ tenta salvar e enviar suas alterações, o sistema retorna uma mensagem para $\mathbf{U_1}$ informando que o nó da árvore de versionamento em questão encontra-se desatualizado, impossibilitando assim o seu *commit*.

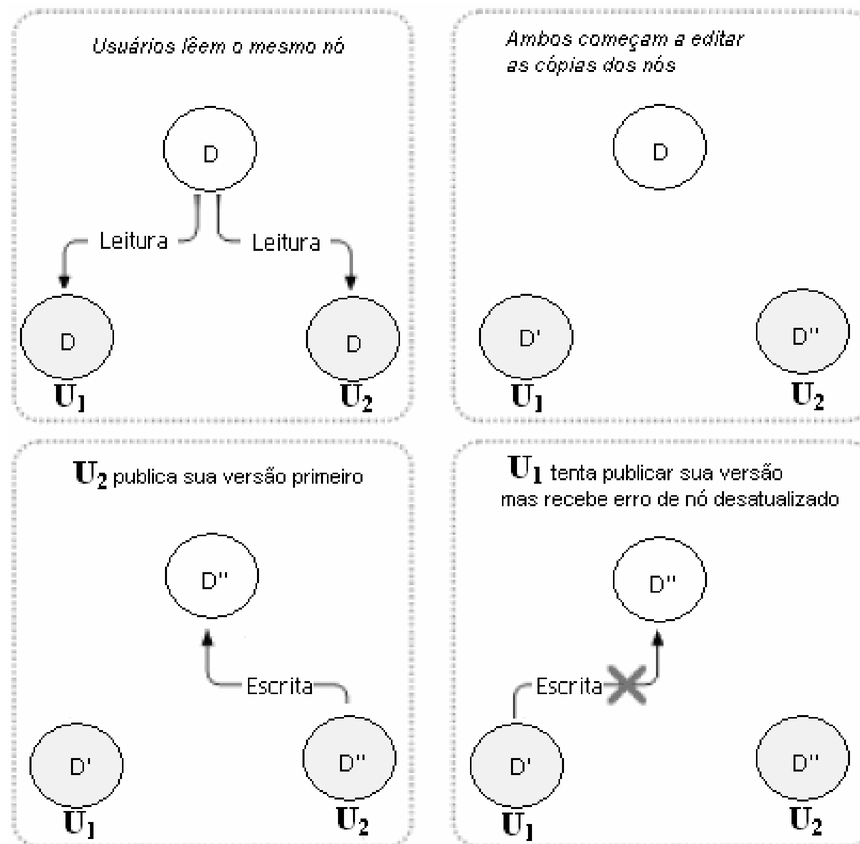

Figura 14 – Método C*opy-Modify-Merge* ou *Optimistic Merges*

PUC-Rio - Certificação Digital Nº 0420994/CB

Como **U_1** recebeu do sistema uma mensagem informando que ele não tinha permissão para manipular nem muito menos submeter o nó **D**, **U_1** solicita então o *merge* da sua cópia local com a mais atual da base do sistema. Somente após o *merge* apresentar nenhum conflito, **U_1** pode enfim enviar suas alterações do nó para a base de dados. Dessa forma, tanto as contribuições de **U_2** quanto as de **U_1** se encontram integradas e atualizadas.

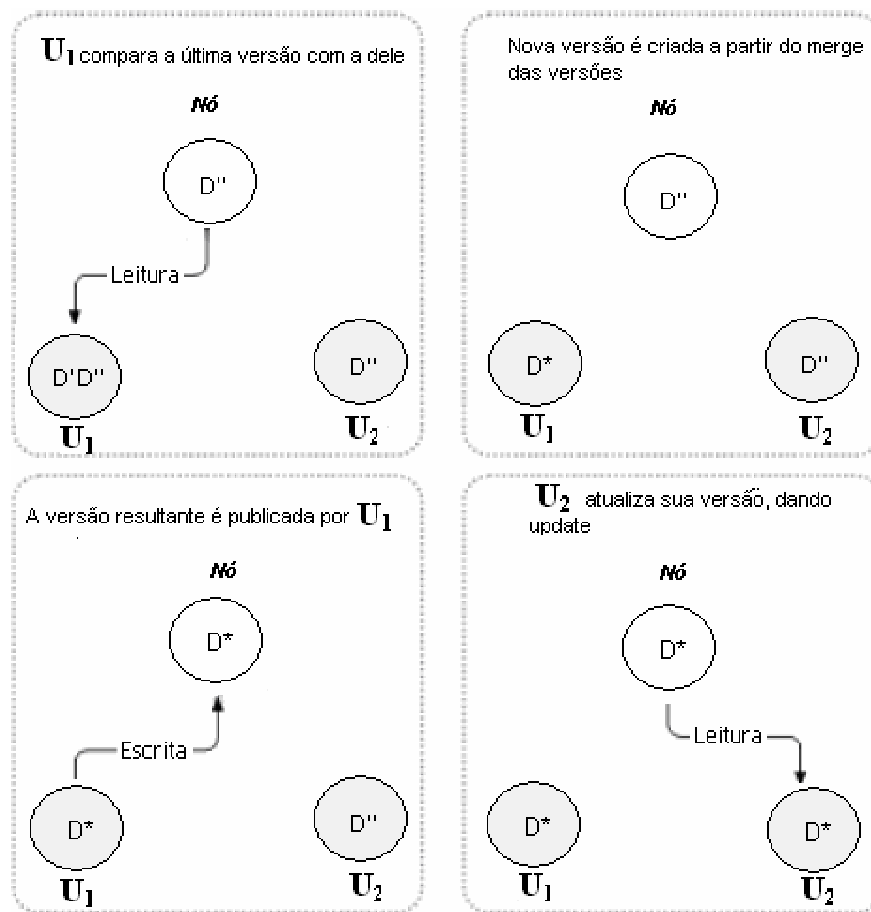

Figura 15 – Continuação do exemplo do método *copy-modify-merge*

Uma vantagem do *copy-modify-merge* sobre o *lock-modify-unlock* é que editores de vídeo podem trabalhar paralelamente alterando um mesmo nó da árvore a qualquer momento. Uma transação do método *lock-modify-unlock* bloqueia de modo exclusivo um determinado nó e não permite alterações concorrentes sobre este mesmo nó bloqueado.

O método *copy-modify-merge* é baseado na suposição de que os arquivos podem ser mesclados (*mergeable files*). Apesar dos arquivos MPEG-2 possuírem

uma estrutura definida, o *merge* não é simples já que exigiria uma análise de quais quadros foram modificados por quais usuários, além da composição do vídeo mesclado quadro-a-quadro a partir dos vídeos originais. Por isso, adotou-se o método *lock-modify-unlock* para realizar o controle concorrente dos nós de vídeo da árvore de versionamento.

4.2.1. Versão Temporária e Permanente da Árvore de Versionamento

A versão de uma árvore de versionamento pode ser temporária ou permanente. A versão permanente de uma árvore é gerada quando o usuário deseja submeter as alterações realizadas sobre ela no repositório do sistema. Já a versão temporária de uma árvore é gerada quando o usuário deseja alterar uma versão permanente de uma árvore. Quando um ou mais usuários desejam editar colaborativamente e concorrentemente uma mesma versão permanente da árvore afim de resultar numa única versão, o sistema gerencia através de bloqueios o controle de acesso dos nós, como será discutido em detalhes na Seção 4.3.

Todo commit de uma versão temporária gera uma nova versão permanente da árvore de versionamento. Um usuário pode bloquear ou não um nó de uma versão permanente. Os nós bloqueados na versão permanente permanecem bloqueados na versão temporária quando recuperados.

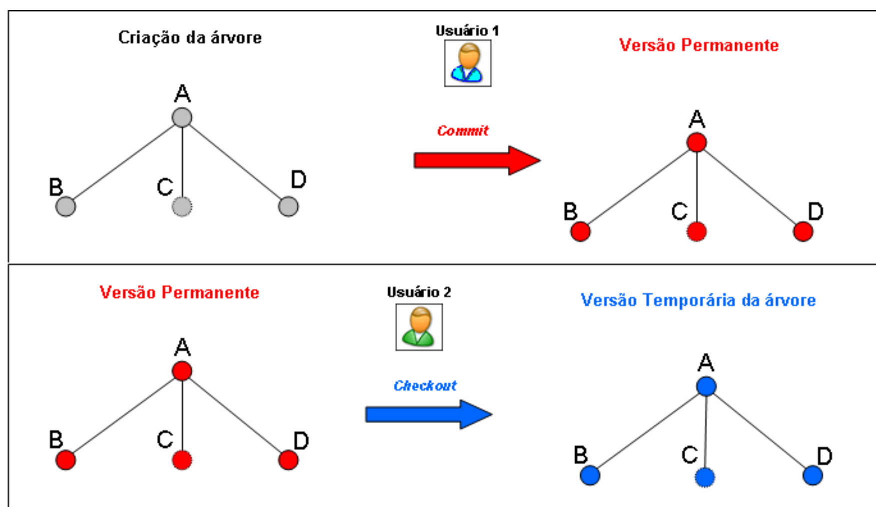

Figura 16 – Versão permanente e temporária

Um exemplo de criação de uma versão permanente e temporária de uma árvore de versionamento é mostrado na Figura 16. Nele, o usuário 1 cria uma

PUC-Rio - Certificação Digital Nº 0420994/CB

árvore de versionamento e submete através da operação de *commit* suas alterações, resultando numa versão mais recente e permanente da árvore. Em seguida, como o usuário 2 deseja alterar os nós da versão permanente que o usuário 1 submeteu, ele solicita ao sistema uma versão temporária para edição realizando um *checkout* da versão mais recente.

Como dito anteriormente, todo usuário tem a possibilidade de bloquear a edição e leitura de determinados nós de modo exclusivo de uma versão permanente da árvore. Esse bloqueio exclusivo garante que novas versões permanentes compartilhem nós entre elas, obrigando que determinadas versões de nós não seja modificado. No exemplo da Figura 17, o usuário realiza o *checkout* da versão permanente com o nó B bloqueado de forma exclusiva por outro usuário. No fim das modificações, uma nova versão permanente foi gerada compartilhando entre esta nova e a que originou o no B.

PUC-Rio - Certificação Digital Nº 0420994/CB

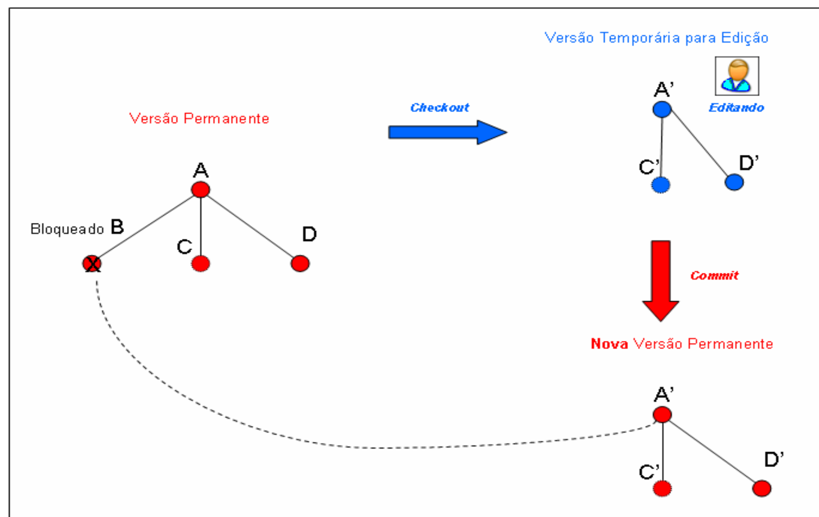

Figura 17 – Versão temporária editada por um único usuário

Um outro exemplo ilustrado na Figura 18 mostra a edição concorrente de uma mesma versão temporária por dois usuários. Nesse exemplo, os usuários realizam *checkout* da versão permanente e iniciam suas respectivas edições. No fim das modificações, uma versão permanente é gerada, mesclando assim as alterações dos dois usuários numa única e nova versão permanente.

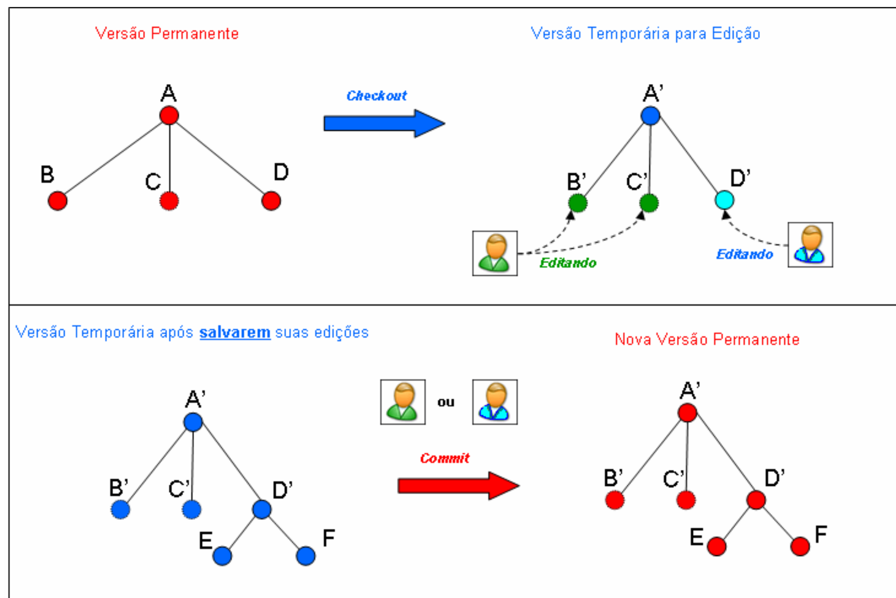

Figura 18 – Versão temporária editada por vários usuários

PUC-Rio - Certificação Digital Nº 0420994/CB

4.3. Granularidade Múltipla na Árvore de Versionamento

Com a finalidade de realizar o controle de acesso concorrente aos nós da árvore de versionamento, é utilizado o conceito da granularidade múltipla, bastante difundido nos sistemas de gerenciamento de banco de dados [SiKS99]. Esse conceito define um mecanismo que permite ao sistema estipular níveis múltiplos de granulação de dados, onde as granulações menores estão aninhadas dentro das maiores.

Por definição, a granularidade múltipla significa que, quando uma transação bloqueia um objeto **O** explicitamente num determinado modo **M**, todos os objetos ancestrais de **O** na hierarquia ficam bloqueados intencionalmente no modo **M**. Além disso, uma transação só poderá bloquear um objeto **O** no modo **M**, se e somente se, **O** não estiver bloqueado por outra transação, implícita ou explicitamente, em um modo **N** incompatível com **M**.

Como se pode notar, novas definições de modo de bloqueio são utilizados nesse conceito da granularidade múltipla. Por esse motivo, esta seção foi subdivida em quatro subseções. A Seção 4.3.1 descreve os modos simples de bloqueios exclusivos e compartilhados dos nós da árvore de versionamento. Em seguida, a Seção 4.3.2 descreve a nova classe de bloqueio da granularidade

múltipla, que são os bloqueios intencionais também aplicados aos nós da árvore de versionamento. A Seção 4.3.3 descreve o protocolo de bloqueio em duas fases que, juntamente com o conceito acima, trata do acesso concorrente dos nós. Por fim, a Seção 4.3.4 descreve os modos de bloqueios em situações onde existem nós compartilhados em mais de uma árvore de versionamento.

4.3.1. Bloqueios Explícitos e Implícitos no modo Exclusivo ou Compartilhados

Na árvore de versionamento de um vídeo, cada nó pode ser bloqueado individualmente por um usuário. O modo de bloqueio de cada nó pode ser exclusivo ou compartilhado.

No modo compartilhado, se uma transação T_i obtém o bloqueio compartilhado (representado por S) sobre um nó da árvore de versionamento, então T_i poderá ler, mas não poderá escrever nesse nó. Por outro lado, no modo exclusivo, se uma transação T_i obtém o bloqueio exclusivo (representado por X) sobre um nó da árvore de versionamento, então Ti poderá ler e escrever nesse nó.

É importante ressaltar que uma transação pode desbloquear um nó explicitamente bloqueado da árvore de versionamento a qualquer momento, mas deve manter o bloqueio enquanto se deseja operar sobre ele.

No exemplo da Figura 19, o nó B está bloqueado no modo exclusivo, e portanto, nenhum outro usuário conseguirá acessá-lo. Por sua vez, o nó D está bloqueado no modo compartilhado, tornando possível que outros usuários também tenham permissão para lê-lo.

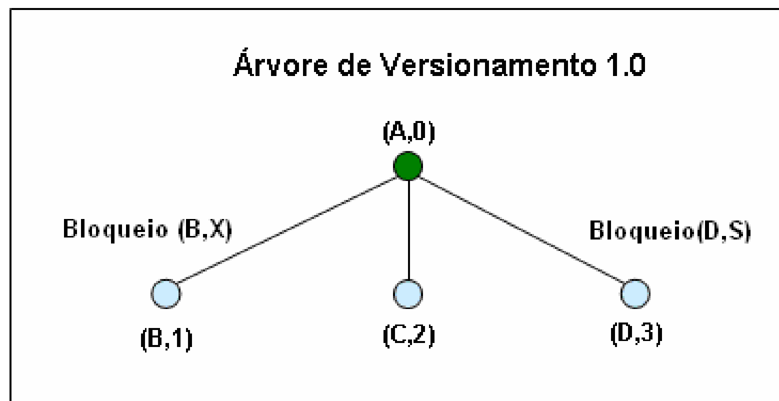

Figura 19 – Árvore de versionamento com bloqueio X e S

PUC-Rio - Certificação Digital Nº 0420994/CB

Além dos modos de bloqueios compartilhados ou exclusivos, os bloqueios também podem ser explícitos ou implícitos. O bloqueio é explícito quando se aplica diretamente ao nó da árvore de versionamento em que foi solicitado o bloqueio. Já o bloqueio é implícito quando aplicado aos nós descendentes de um nó bloqueado de forma explícita.

Numa transação, quando um nó é bloqueado de forma explícita, tanto no modo exclusivo como compartilhado, imediatamente todos os seus descendentes são também bloqueados com o mesmo modo de forma implícita. Note que não é feito nenhum bloqueio explícito dos descendentes de um nó explicitamente bloqueado.

Esse caso pode ser visto no exemplo da Figura 19, quando a raiz A for bloqueada no modo exclusivo de forma explícita. Como conseqüência, todos os filhos da raiz estarão bloqueados no modo exclusivo de forma implícita.

PUC-Rio - Certificação Digital Nº 0420994/CB

4.3.2. Bloqueios Intencionais no modo Exclusivo ou Compartilhado

Como descrito na Seção 4.3.1, os bloqueios realizados de forma explícita ou implícita sobre um determinado nó da árvore de versionamento podem estar no modo exclusivo ou compartilhado. Apesar dos modos de bloqueios já garantirem o acesso exclusivo ou compartilhado dos nós da árvore de versionamento, uma outra classe de bloqueio se faz necessária.

No exemplo da Figura 20, suponha que uma transação $\mathbf{T_i}$ bloqueia de forma explícita a folha **E** da árvore 1.0, no modo exclusivo. Suponha agora que uma outra transação $\mathbf{T_j}$ queira bloquear a folha **E**. Quando $\mathbf{T_j}$ emite uma solicitação de bloqueio para **E**, $\mathbf{T_j}$ precisaria percorrer a árvore desde a raiz **A** até o nó **E**. Somente, quando $\mathbf{T_j}$ atingisse este nó **E**, $\mathbf{T_j}$ perceberia que **E** está bloqueado num modo incompatível com o desejado por ela.

A princípio, determinar se uma transação tem ou não sucesso, é necessário percorrer a árvore de versionamento, desde a raiz até o nó requisitado para bloqueio. Por isso, para que não haja a necessidade de percorrer a árvore de versionamento inteira, foi introduzida na granularidade múltipla uma outra classe de bloqueio, chamada de bloqueio intencional.

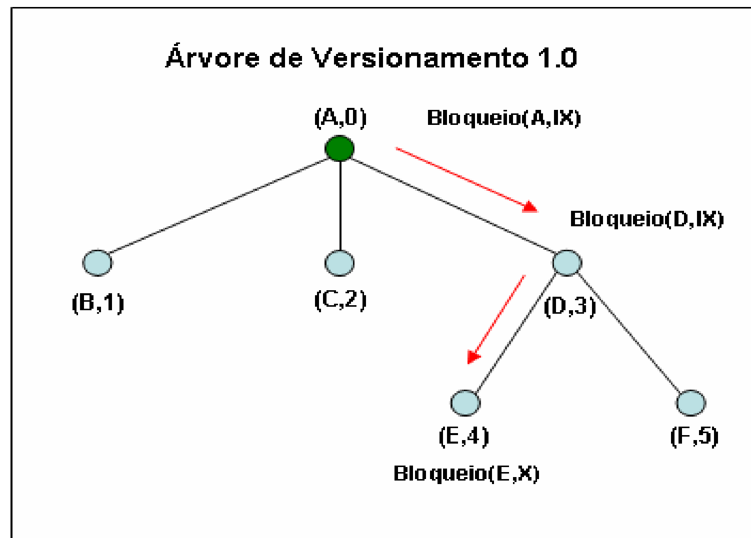

Figura 20 – Árvore de versionamento com bloqueio intencional

PUC-Rio - Certificação Digital Nº 0420994/CB

Os bloqueios intencionais são realizados sobre todos os antecessores de um determinado nó antes que este nó seja bloqueado de forma explícita. Dessa maneira, uma transação não precisa pesquisar a árvore de versionamento inteira para determinar se poderá bloquear um nó. Uma transação que queira bloquear de forma explícita, no modo M, um nó Z por exemplo, deve antes caminhar da raiz a Z, e enquanto se caminha, segue bloqueando intencionalmente os antecessores de Z no modo M. Ao chegar em Z, um bloqueio explícito no modo M é realizado. Note que todos os antecessores de Z estão devidamente bloqueados intencionalmente no modo M.

O modo intencional é dividido em três outros tipos para indicar qual o modo de bloqueio explícito foi aplicado sobre um nó.

O primeiro modo intencional de bloqueio é chamado de modo de bloqueio *compartilhado-intencional* (*intention-shared* – IS). Esse primeiro modo é definido quando um bloqueio explícito de modo compartilhado está sendo feito no nível mais baixo da árvore.

O segundo tipo é parecido com o IS, uma vez que um nó é bloqueado no modo *exclusivo-intencional* (*intention-exclusive* – IX). Nele, o bloqueio explícito de modo exclusivo ou compartilhado está sendo feito no nível mais baixo da árvore de versionamento. Por fim, o terceiro tipo é chamado de bloqueio no modo *compartilhado e exclusivo-intencional* (*shared intention-exclusive* – SIX). Nele, a sub-árvore cuja raiz é o nó em questão está bloqueada de forma explícita no modo

compartilhado e o bloqueio explícito no modo exclusivo está sendo feito em algum nó de nível mais baixo da árvore.

A Tabela 3 mostra a compatibilidade entre todos os modos de bloqueios aplicados sobre a árvore de versionamento:

Tabela 3 – Matriz de compatibilidade dos modos de bloqueio

	IS	**IX**	**S**	**SIX**	**X**
IS	Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	Falso
IX	Verdadeiro	Verdadeiro	Falso	Falso	Falso
S	Verdadeiro	Falso	Verdadeiro	Falso	Falso
SIX	Verdadeiro	Falso	Falso	Falso	Falso
X	Falso	Falso	Falso	Falso	Falso

PUC-Rio - Certificação Digital Nº 0420994/CB

4.3.3. Protocolo de Bloqueio em Duas Fases

Além de seguir a matriz de compatibilidade apresentada na Tabela 3, as transações devem seguir o protocolo de bloqueio em duas fases (*Two-Phase Locking Protocol*) para garantir serialização das atualizações [SiKS99].

O protocolo de bloqueio em duas fases exige que cada transação solicite o bloqueio e desbloqueio em duas fases: a fase de expansão e de encolhimento. Durante a fase de expansão, uma transação só pode obter bloqueios sobre os nós, mas não pode liberar nenhum outro. Durante a fase de encolhimento, uma transação só pode liberar bloqueios, mas não pode obter nenhum bloqueio novo.

Uma transação, inicialmente, está na fase de expansão, solicitando bloqueios, até que libera um bloqueio e entra na fase de encolhimento, não podendo solicitar novos bloqueios. Nesse protocolo, o ponto de bloqueio é o instante em que a transação libera o primeiro bloqueio, e caracteriza o fim da fase de expansão.

O bloqueio da árvore de versionamento é feito caminhando-se da raiz às folhas, enquanto que o desbloqueio é feito caminhando-se das folhas à raiz. Cada transação T_i pode bloquear um nó Z qualquer, usando as seguintes regras:

- Observar a matriz de compatibilidade de bloqueio (Tabela 3);

- A raiz da árvore precisa ser bloqueada primeiro e pode ser bloqueada em qualquer modo;
- Um nó Z pode ser bloqueado por T_i no modo S ou IS somente se o pai de Z for bloqueado por T_i no modo IX ou IS.
- Um nó Z pode ser bloqueado por T_i no modo X, SIX ou IX somente se o pai de Z estiver bloqueado por T_i no modo IX ou SIX.
- Ti pode bloquear um nó somente se ele não desbloqueou outro nó anteriormente (transação feita em duas fases).
- Ti pode desbloquear um nó Z somente se nenhum dos filhos de Z estiver bloqueado por T_i.

PUC-Rio - Certificação Digital Nº 0420994/CB

Um exemplo do protocolo de bloqueio em duas fases e das regras pode ser visualizado na Figura 20 e encontra-se descrito na Tabela 4. Para esse exemplo, suponha que duas transações T_1 e T_2 iniciem seus bloqueios concorrentemente. Suponha que T1 bloqueia o nó D no modo exclusivo antes que T2 solicite o bloqueio de A no modo exclusivo, no instante t_1.

No mesmo instante t_1, T_2 tenta bloquear a raiz A. Entretanto o sistema acusa que esta transação T_2 não tem permissão para bloquear A no modo exclusivo. Isto ocorre porque, como já explicado anteriormente, o nó A, que é um antecessor de D, foi bloqueado no modo intencional-exclusivo. No instante t_3, a transação T_2 consegue bloquear o nó B, pois seu pai está bloqueado no modo intencional-exclusivo. Nesse mesmo instante, T_1 inicia a sua fase de encolhimento, iniciando assim os desbloqueios dos nós bloqueados no modo exclusivo por ele.

No instante t_4, embora A esteja bloqueado no modo intencional-exclusivo pela transação T_1, esse modo é compatível com o modo intencional-compartilhado solicitado por T_2 (ver Tabela 3). Portanto a operação é realizada com sucesso. No instante t_5, a transação T_1 é finalizada e a T_2 continua bloqueando outros nós para em seguida iniciar a sua fase de encolhimento

Tabela 4 – Exemplo usando o protocolo de bloqueio em duas fases

Instante	Transação de T_1	Transação de T_2
t_0	Bloqueio (A, IX) ✔	-
t_1	Bloqueio (D, X) ✔	Bloqueio (A, X) ✖

t_2	-	-
t_3	Desbloqueio (D, X) ✓	Bloqueio (B, X) ✓
t_4	-	Bloqueio (A, IS) ✓
t_5	Desbloqueio (A, IX) ✓	Bloqueio (C, S) ✓
t_6	-	Bloqueio (C, X) ✓
t_7	-	Desbloqueio (B, X) ✓
		Desbloqueio (A, IS) ✓

As regras do protocolo de bloqueio utilizada nas árvores de versionamento ajudam a minimizar o *overhead* por bloqueio e aumentam o trabalho concorrente. Entretanto, situações de *deadlock* podem acontecer. O sistema desenvolvido não previne que essas situações deixem de surgir.

4.3.4. Nós Compartilhados entre Árvores de Versionamento

Uma outra situação que necessita de controle de concorrência ocorre quando existem um ou mais nós compartilhados entre várias árvores de versionamento. Obrigatoriamente, um nó compartilhado possui pais em diferentes árvores de versionamento. Nesses casos, como os nós pais são antecessores do nó filho, os bloqueios intencionais também são realizados.

No exemplo da Figura 21, existem três árvores de versionamento 1.0, 1.1 e 1.2. Neste exemplo, C e F são nós compartilhados pelas árvores de versionamento 1.0 e 1.1. Já D é um nó compartilhado pelas árvores de versionamento 1.1 e 1.2. A árvore 1.0 é considerada de origem, pois ela deriva a 1.1 e 1.2.

Suponha que um usuário U_1 bloqueie no modo exclusivo o nó D que, por sua vez, provoca o bloqueio dos ascendentes dele no modo intencional-exclusivo. O bloqueio intencional-exclusivo é realizado nos ascendentes A e A'', das árvores de versionamento de origem e derivada, respectivamente. Tanto A como A'' são pais de D, porém em diferentes árvores. Os nós descendentes de D também estão bloqueados de forma implícita, no modo exclusivo.

É importante ressaltar que, apesar de D' ou A' não pertencerem à árvore de versionamento 1.0, eles também estão bloqueados implicitamente. Isto porque, D' e A' são considerados ascendentes do nó F que está bloqueado de forma implícita,

PUC-Rio - Certificação Digital Nº 0420994/CB

no modo exclusivo. Vale lembrar que U_1 somente pode desbloquear D, uma vez que ele não pretenda bloquear mais nenhum outro nó.

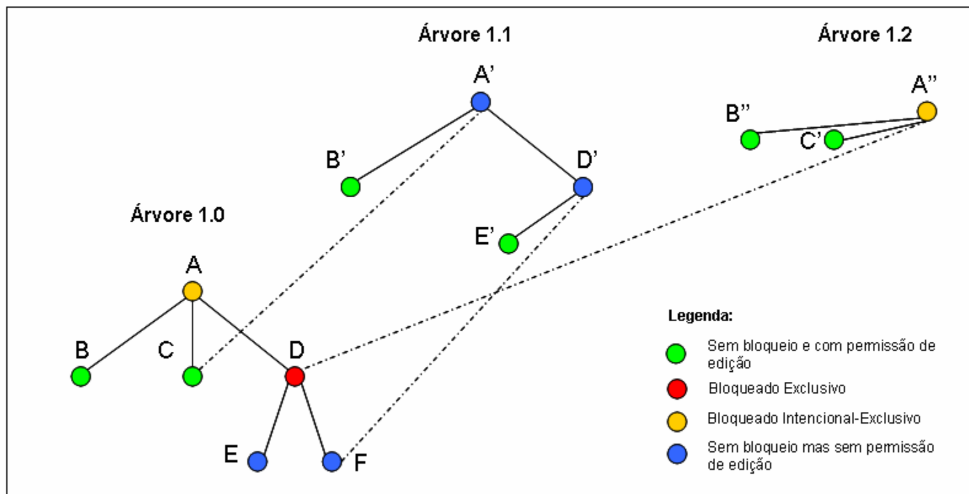

Figura 21 – Exemplo de bloqueio de um nó no modo exclusivo

Após todas essas apresentações sobre os modos de bloqueios que auxiliam no controle do acesso concorrente dos nós, um outro elemento é associado ao rótulo do nó, chamado de *estado* do nó.

O estado de um nó descreve o modo de bloqueio efetuado sobre ele, e é representado por uma estrutura com quatro elementos. O primeiro elemento do estado descreve o modo de bloqueio por ele adquirido, e o segundo, a quantidade de bloqueios. A quantidade de bloqueios é acrescida de 1 somente quando o nó é bloqueado intencionalmente. Essa quantidade de bloqueios, quando zero, define que o nó não possui bloqueios. Quando maior que zero define que um ou mais bloqueios foi efetuado no nó. A medida que desbloqueios são realizados e influenciam o nó, a sua quantidade é decrescida de 1. O terceiro elemento é responsável por informar qual usuário realizou o bloqueio no modo exclusivo e, finalmente, o último informa a data do bloqueio no modo exclusivo. Nos modos compartilhados, o terceiro e quarto elemento não são utilizados.

No exemplo da Figura 22, suponha que B e D estejam bloqueados no modo exclusivo. Então, o estado de A está bloqueado no modo intencional-exclusivo com dois bloqueios. Supondo que B seja desbloqueado, logo o estado do antecessor dele, A, continua sendo representado pelo modo de bloqueio intencional-exclusivo, porém com um único bloqueio.

PUC-Rio - Certificação Digital Nº 0420994/CB

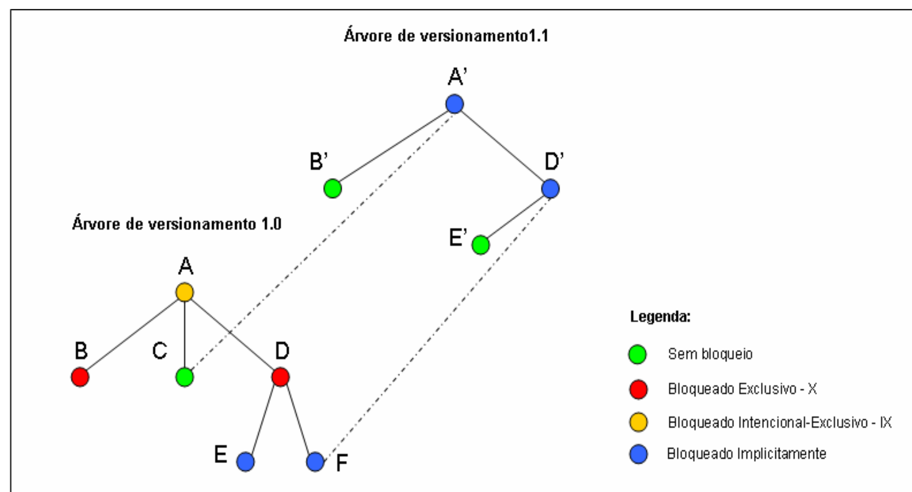

Figura 22 – Estado de bloqueio dos nós

4.4. Mecanismo de Segmentação

A árvore de versionamento de vídeo de origem não possui regra de criação, podendo ser construída com total liberdade pelo usuário. A fim de garantir o trabalho colaborativo e a divisão de tarefas no processo de edição da árvore de versionamento de um vídeo, foi desenvolvido um mecanismo capaz de segmentar o vídeo no formato MPEG-2 e estruturá-lo na árvore de versionamento.

O algoritmo de detecção de tomadas de cenas [Vasconcelos05] tem por finalidade identificar os intervalos de cada segmento de vídeo MPEG-2. Após identificados esses intervalos, o mecanismo de segmentação inicia o processo de geração de cada segmento. Cada segmento gerado, por sua vez, é uma porção reduzida do vídeo original MPEG-2, ou seja, o conjunto dessas porções, se concatenadas na ordem em que foram segmentadas, resultam no vídeo original.

Antes da geração de cada segmento, é preciso realizar uma verificação dos tipos de quadros das bordas de cada uma das tomadas de cenas identificadas. As bordas de uma tomada não podem ter quadros dependentes das tomadas anteriores e posteriores, respectivamente. Caso tenham, um tratamento das bordas é realizado, conforme será descrito posteriormente.

Em resumo, o mecanismo de segmentação ilustrado na Figura 23 é composto por três fases. A primeira fase é responsável por definir os índices dos segmentos de um vídeo original. A segunda fase realiza o tratamento das bordas

PUC-Rio - Certificação Digital Nº 0420994/CB

dos intervalos dos segmentos definidos na primeira fase. A terceira e última fase gera os segmentos em arquivos MPEG-2.

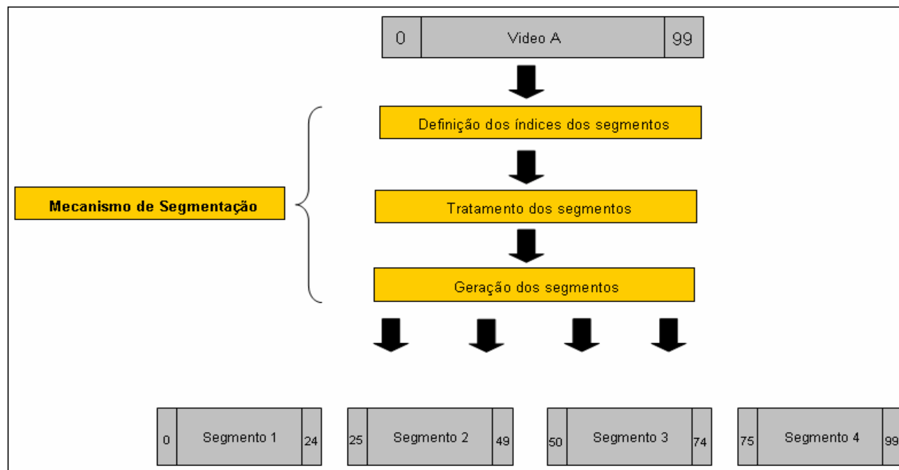

Figura 23 – Fases do mecanismo de segmentação

O tratamento das bordas é utilizado quando existem dependências de quadros nas bordas dos segmentos adjacentes entre si. A duplicação de nós dependentes das bordas entre os segmentos consecutivos é a técnica utilizada para que os segmentos possam ser visualizados corretamente nos exibidores (*players*).

O tratamento das bordas dos segmentos do vídeo está relacionado aos tipos de quadros I, P, B do padrão MPEG-2. Os quadros I não utilizam nenhum tipo de predição e por isso não possuem dependência alguma. Já os quadros P são codificados referenciando um quadro I ou P anteriormente codificado – predição *forward* – usando compensação de movimento. Tais quadros referenciados são chamados âncoras. Um quadro B é codificado usando predição com compensação de movimento em relação a dois outros quadros I ou P: um anterior e outro posterior no tempo. Durante os processos de codificação e de decodificação, os quadros âncoras devem ser processados antes do quadro dependente.

Como todo segmento deve iniciar com um quadro I, pois os *players* de exibição não conseguem reproduzir arquivos MPEG-2 iniciados com quadros P ou B, as bordas de um segmento são tratadas após o método de detecção do mecanismo de segmentação informar que o primeiro quadro se trata de um quadro P ou B. Da mesma forma, se um segmento termina com quadros B, sua borda deve ser tratada de forma a possibilitar a exibição desse quadro B.

PUC-Rio - Certificação Digital Nº 0420994/CB

A Figura 24 ilustra um exemplo de tratamento. No exemplo, suponha dois segmentos **n-1** e **n**. A borda da esquerda do segmento **n-1** inicia com um quadro I, logo não há necessidade de tratamento. Já a borda da direita, termina com um quadro B, logo percebe-se que existe a dependência deste último quadro com outros do segmento consecutivo. Nesse caso, o novo segmento **n-1** recebe uma cópia do primeiro quadro P do segmento **n**, que então é incluído ao seu final. O início do segmento **n** também precisa de tratamento, uma vez que o primeiro quadro da borda da esquerda é do tipo P. O algoritmo então concatena quadros do segmento anterior ao atual, até que seja encontrado um quadro I. Logo, o novo segmento **n** terá mais quatro quadros, originados do segmento **n-1**.

É importante ressaltar que a duplicação de quadros é em relação aos segmentos originais e não aos segmentos já tratados. O quadro P, por exemplo, que foi adicionado ao novo segmento **n-1** não aparece na borda da esquerda do novo segmento **n**, porque ele já faz parte e não precisaria ser incluído novamente.

Ao se dividir um vídeo em segmentos, um cuidado adicional deve ser tomado. Cada segmento deve ser tratado de forma a representar um fluxo independente MPEG-2. Isto significa que todos os cuidados de codificação relativos ao controle do buffer de decodificação detalhados no Capítulo 2 devem ser observados. Para não haver perda da qualidade do vídeo, o ideal é que tal tratamento seja realizado sem a necessidade da decodificação e recodificação do vídeo.

Alem da técnica de duplicação de quadros utilizada, poderia ser considerado o mecanismo descrito em [ChZZ04], que apresenta um esquema de conversão de tipo de quadro, requerendo a aplicação da transformada I-DCT, ou seja, através da decodificação parcial dos coeficientes DCT, de forma a efetuar a concatenação de segmentos de vídeo. Esse mecanismo, entretanto, altera a qualidade do vídeo no momento que é feita a recodificação dos segmentos.

PUC-Rio - Certificação Digital Nº 0420994/CB

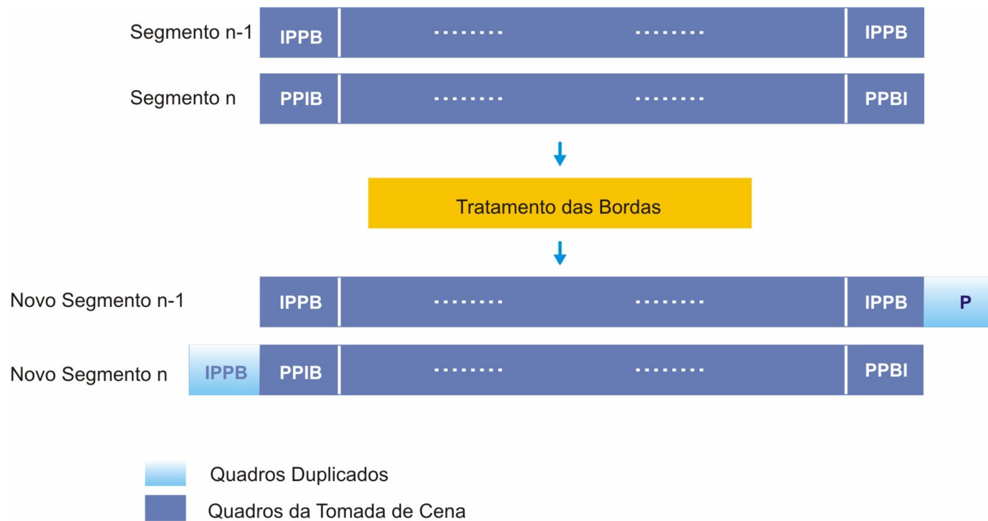

Figura 24 – Tratamento das bordas dos segmentos

Por fim, após a detecção das tomadas e do método de tratamento das bordas, são gerados os segmentos MPEG-2 de vídeo que se tornam folhas da árvore de versionamento. Um exemplo disso, seria a detecção de três tomadas de cenas de um Filme A e o tratamento e segmentação das tomadas, como mostra a Figura 25.

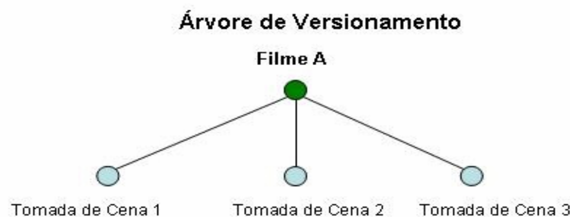

Figura 25 – Detecção das tomadas de cenas de um vídeo

4.5. Mecanismo de Remontagem

O mecanismo de remontagem é realizado quando um editor pretende visualizar o vídeo da árvore de versionamento. Quando solicitado pelo usuário, o mecanismo aplica a concatenação ordenada dos conteúdos das folhas da árvore de versionamento.

É importante salientar que, durante esse mecanismo, nenhuma decodificação ou recodificação dos arquivos MPEG é feita. A concatenação,

PUC-Rio - Certificação Digital Nº 0420994/CB

assim como a segmentação de um vídeo, é realizada no domínio comprimido, pois somente assim a qualidade do arquivo original permanece inalterada.

No exemplo da Figura 10, a visualização da árvore de versionamento do vídeo se dá pela concatenação ordenada das folhas B, C, E e F.

A operação de concatenação de arquivos no formato MPEG-2 também pode impactar o controle da ocupação do *buffer* do decodificador do vídeo MPEG-2 na hora da apresentação. Situações de *overflow* ou *underflow* podem ocorrer e dependem do cálculo de ocupação do *buffer* realizado para cada fluxo de vídeo. Assim, o processo de remontagem do vídeo deve ser implementado com cautela, uma vez que os decodificadores de vídeo MPEG-2 não devem apresentar comportamentos inesperados na hora da leitura e apresentação do vídeo.

Como as folhas da árvore de versionamento são considerados arquivos MPEG-2 independentes entre si, a sua edição pode modificar de tal maneira que o conteúdo da folha passa a não ser mais uma continuação da folha anterior ou uma antecipação da folha posterior. Vale ressaltar que é na fase da edição de um arquivo de vídeo onde são incluídos os efeitos especiais, trilhas sonoras, legendas, mudanças da ordem dos quadros, entre outras edições. Quando um usuário bloqueia explicitamente no modo exclusivo uma folha da árvore de versionamento, ele pode modificar o conteúdo da folha livremente. Não há restrições ao usuário no momento da edição do conteúdo da folha da árvore de versionamento. Por esse motivo, sempre quando houver edições nas bordas de um segmento, um alerta deve ser ativado quando da remontagem do vídeo.

PUC-Rio - Certificação Digital Nº 0420994/CB

4.6.
Fusão entre Árvores de Versionamento

O mecanismo de fusão é utilizado para mesclar as contribuições entre árvores de versionamento. A fusão é realizada entre duas árvores de versionamento derivadas e produz uma árvore resultante. Basicamente, o mecanismo compara os *timestamps* dos nós das duas árvores derivadas e seleciona um desses nós. A escolha do nó está relacionada com a idade dos *timestamps* rotulado nos nós.

Para que o mecanismo seja melhor entendido, é importante definir um novo relacionamento de derivação entre nós, chamado de *nós derivados equivalentes*.

Dois nós são considerados *derivados equivalentes* quando eles pertencem a árvores de versionamento diferentes e essas duas árvores possuem algum parentesco entre si. No exemplo da Figura 27, **B'** é um nó derivado do nó **B** de origem e pertence a árvore de versionamento 1.1. Como **B''** também é um nó derivado de **B** e pertence à árvore de versionamento 1.2 que, por sua vez, pode ser considerada "irmã" da 1.1, então **B'** e **B''** são nós derivados equivalentes.

Esse mecanismo recebe como entrada duas árvores de versionamento derivadas de uma árvore de origem. Uma das duas árvores é definida como *base*, pois o posicionamento dos filhos da árvore base será seguido pela árvore resultante. A raiz da árvore que tiver a maior quantidade de filhos é escolhida como árvore base. Caso o número seja igual, então a árvore base é aquela com *timestamp* de criação mais recente.

Depois de escolhida a árvore base, o mecanismo compara o *timestamp* de cada um dos filhos da árvore base ao filho equivalente derivado na outra árvore. Se dois nós forem *equivalentes derivados*, o protocolo de decisão entre eles se comporta da seguinte forma:

- Quando o *timestamp* dos dois nós são idênticos, então qualquer um dos nós é incluído na árvore de versionamento resultante.
- Quando o *timestamp* de um nó for maior que o do outro, e o *timestamp* do menor for igual ao nó de origem, então o nó de maior *timestamp* é preferido e incluído na árvore resultante.
- Quando o *timestamp* de um nó for maior que o do outro e o *timestamp* do menor for diferente do nó de origem, então o sistema solicita uma intervenção do usuário, para optar qual nó deve ser incluído na árvore resultante. Note que o mecanismo não realiza qualquer merge de conteúdo. Se isso for desejado, deve ser feito externamente ao mecanismo.

O mecanismo é ilustrado no exemplo da Figura 26. Nota-se que, neste exemplo, não foi preciso uma intervenção do usuário no protocolo de decisão do nó. Entretanto na Figura 27, essa intervenção manual é necessária para escolher entre o nó **B'**e **B''**.

PUC-Rio - Certificação Digital Nº 0420994/CB

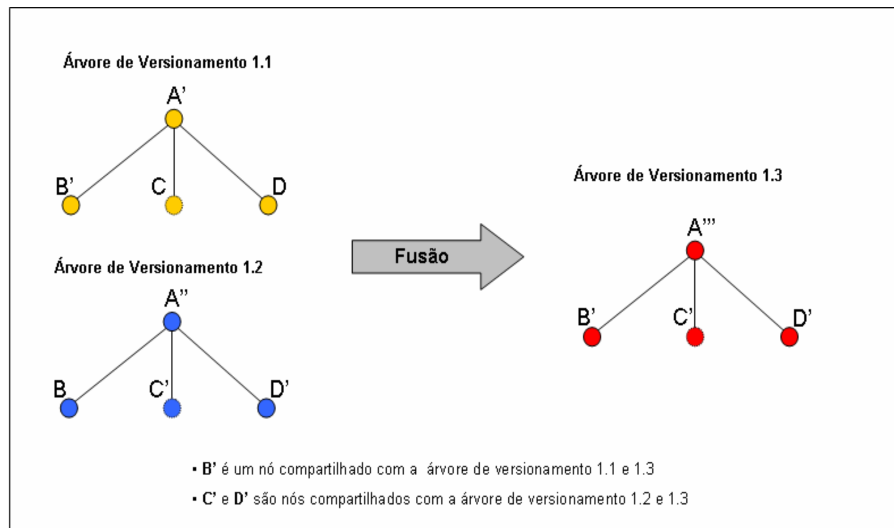

Figura 26 – Fusão de duas árvores de versionamento

Durante a fusão, se não existir um nó equivalente ao recuperado, o próprio nó recuperado é incluído na árvore resultante. O exemplo da Figura 28 ilustra esse caso. O nó C' não possui nó equivalente na árvore de versionamento 1.2, logo ele é incluído na resultante. Note que neste mesmo exemplo, também é preciso uma intervenção manual do usuário.

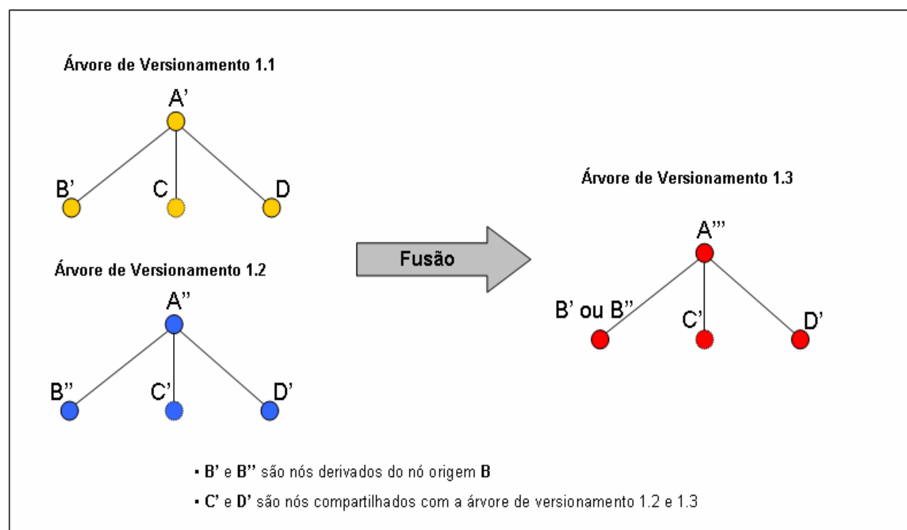

Figura 27 – Fusão das árvores de versionamento com intervenção do usuário

PUC-Rio - Certificação Digital Nº 0420994/CB

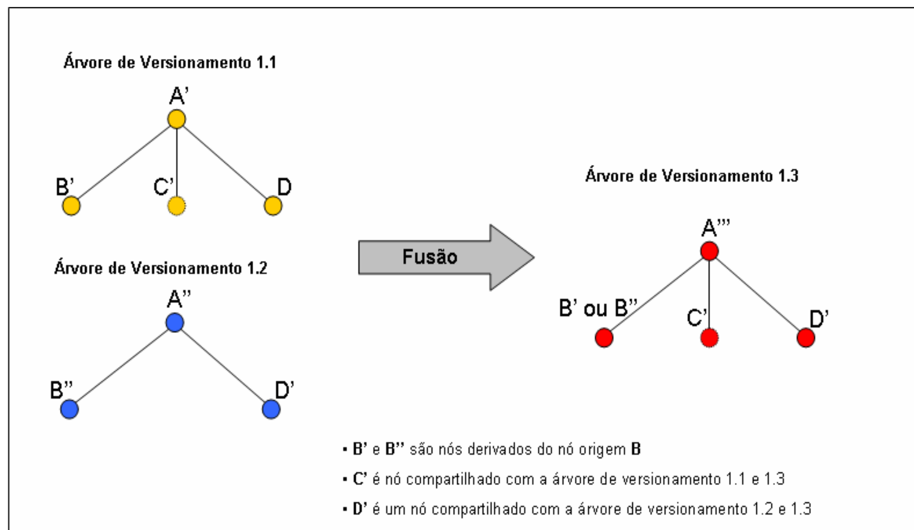

Figura 28 – Fusão de duas árvores de versionamento com nós não equivalentes

Outro caso é ilustrado na Figura 29. Nesse exemplo, as raízes das árvores de versionamento 1.1 e 1.2 têm a mesma quantidade de filhos, porém com ordem inversa. A decisão é por manter a ordem da árvore mais nova determinada pelo *timestamp* de criação da mesma.

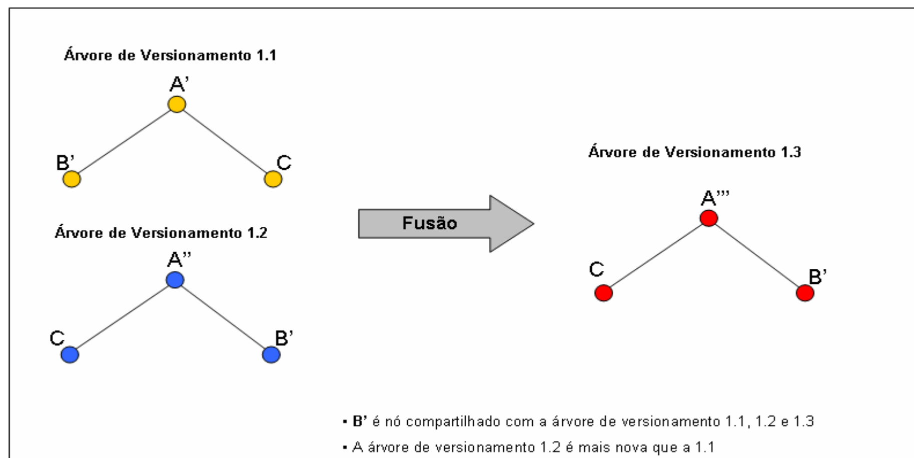

Figura 29 – Fusão de duas árvores de versionamento com nós invertidos

PUC-Rio - Certificação Digital Nº 0420994/CB

5
Implementação

Este capítulo descreve a implementação do primeiro protótipo do sistema de controle de versão das edições cooperativas de vídeo MPEG-2, denominado *VideoCVS*.

Primeiramente, este capítulo apresenta as tecnologias utilizadas no desenvolvimento e a arquitetura do *VideoCVS*. Em seguida, apresenta o modelo entidade e relacionamento do *VideoCVS* e as classes que implementaram esse modelo. A arquitetura das classes do sistema foi modelada seguindo o paradigma de orientação a objetos e de sistemas distribuídos. Para as principais classes do sistema implementado, são apresentados os atributos e métodos definidos. O capítulo também apresenta como foram implementados os mecanismos de segmentação e remontagem do vídeo MPEG-2. Finalmente, o capítulo apresenta um exemplo de uso do *VideoCVS*.

5.1.
Desenvolvimento do *VideoCVS*

O *VideoCVS* é um sistema baseado na arquitetura cliente e servidor que realiza o controle de versão das edições cooperativas de vídeo MPEG-2. Um dos requisitos do sistema *VideoCVS* foi ser implementado independente de plataforma, buscando uma total interoperabilidade. Nesse sentido, o sistema foi implementado buscando obedecer a padrões.

Sua implementação foi desenvolvida sobre a plataforma Windows e Linux, utilizando a linguagem de programação Java e bibliotecas padronizadas. A comunicação cliente e servidor do sistema foi implementada utilizando CORBA (*Common Object Request Broker Architecture)*. Para isso, o *Object Request Broker*[1] (ORB) utilizado foi o JacORB [JacORB06].

---

[1] ORB é considerado o núcleo do CORBA, pois manipula as requisições dos objetos, intermediando a comunicação entre o cliente e o servidor.

PUC-Rio - Certificação Digital Nº 0420994/CB

A ferramenta *Ant* [Holzner05] foi extensivamente utilizada para compilação. Na implementação do *VideoCVS*, o *Ant* facilitou a compilação em diversos ambientes e plataformas, e ajudou na execução do servidor e dos testes automatizados. O alvo padrão do arquivo de configuração realiza a compilação da IDL CORBA (*Interface Definition Language*), compila o código fonte automaticamente gerado, bem como os demais códigos fonte do sistema, e por fim, gera os arquivos *jars* de distribuição.

Para a persistência dos dados, foi utilizado o *Hypersonic SQL Database* - HSQLDB [Hsql06]. O HSQLDB é um servidor de banco de dados de código aberto, que permite a manipulação dos dados em uma arquitetura cliente-servidor, ou *standalone*. Uma grande vantagem de utilização do HSQLDB é por ele ser multiplataforma e ocupar um pequeno espaço em disco. Outra característica do sistema é a possibilidade do bancos de dados ser manipulado em disco, memória ou em formato texto. Trata-se de uma tecnologia flexível e muito útil na construção de aplicações que manipulam banco de dados.

5.2. Arquitetura do *VideoCVS*

O *VideoCVS* é baseado na arquitetura cliente e servidor, e está organizado em dois principais componentes: o *VideoCVS Server* e o *VideoCVS Client*. A Figura 30 ilustra a arquitetura do sistema VideoCVS.

O *VideoCVS Server* é um componente responsável por criar objetos e por registrá-los no serviço de nomes (*CORBA COS Naming*). Esse componente usa os *skeletons*[2] gerados pelo compilador IDL, para facilitar a comunicação com os clientes remotos. Os objetos criados no *VideoCVS Server* seguem o padrão de projeto *Factory*[3]. A Figura 31 mostra o diagrama de classes dos objetos criados e referenciados no serviço de nomes do *VideoCVS Server*.

---

[2] Skeletons são interfaces estáticas para os serviços remotos. É responsável por receber as requisições do cliente e repassá-las ao servidor.

[3] *Factory* é um padrão de projeto que permite a criação de objetos ou famílias de objetos relacionados ou dependentes, através de uma única interface e sem que a classe concreta seja especificada [GHJV95].

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

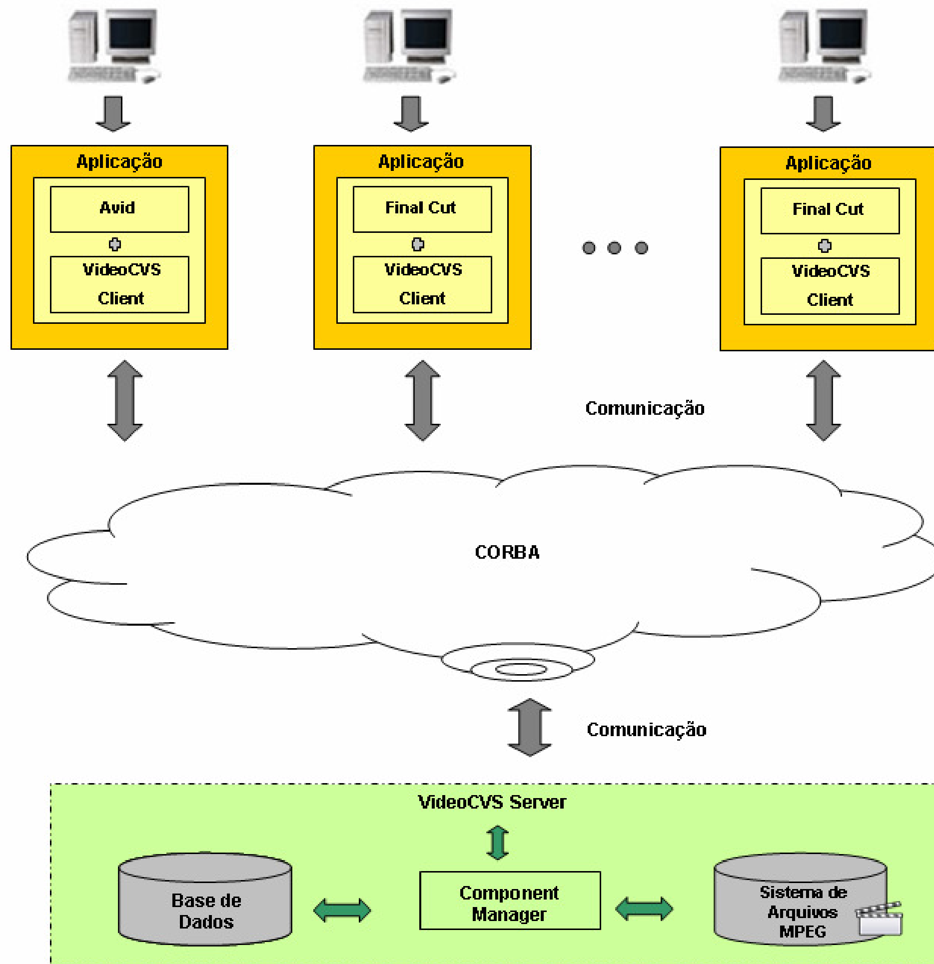

Figura 30 – Arquitetura do *VideoCVS*

A classe *VideoTreeFactoryServantImpl* é a fábrica do objeto remoto da classe *VideoTree*, que por sua vez é a árvore de versionamento de um vídeo. Assim como a classe *VideoNodeFactoryServantImpl* é a fábrica do objeto remoto da classe *VideoNode*, que por sua vez, é o nó da árvore.

PUC-Rio - Certificação Digital Nº 0420994/CB

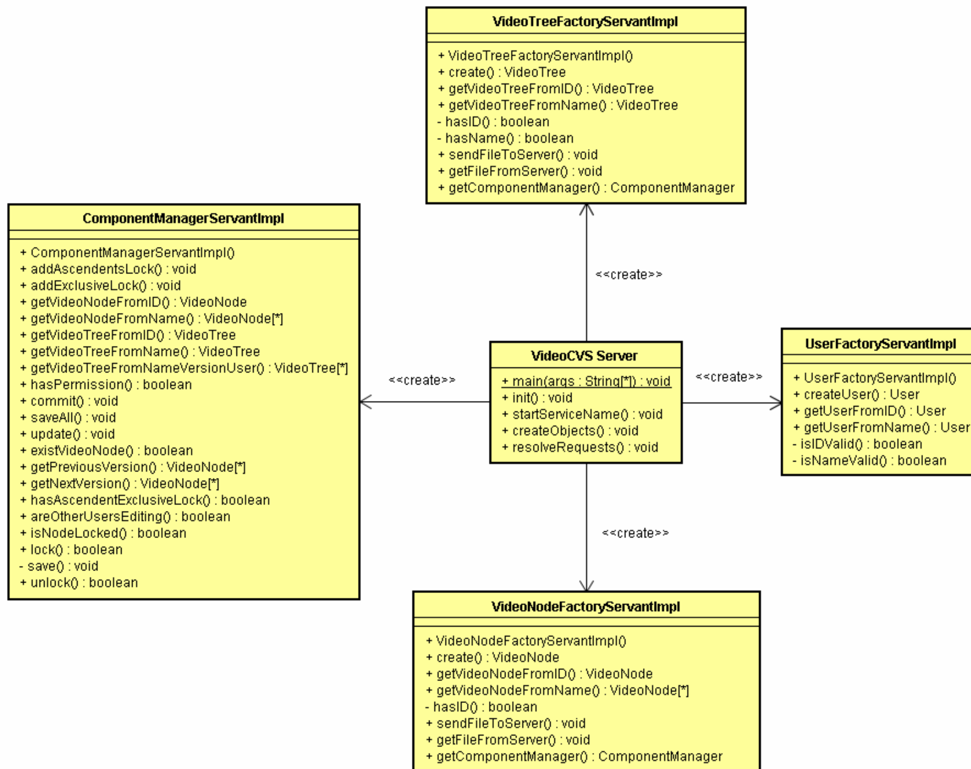

Figura 31 – Diagrama de classes do *VideoCVS Server*

Da mesma maneira, a classe *UserFactoryServantImpl* é a fábrica do objeto remoto *User*, que representa o usuário do sistema. Finalmente, o *ComponentManager* é um objeto remoto responsável por tratar e gerenciar as transações realizadas sobre a árvore de versionamento. Esse componente tem acesso às tabelas da base de dados do sistema e é onde estão implementados o protocolo de bloqueio em duas fases e a granularidade múltipla. Por esses motivos, o *ComponentManager* também é publicado no serviço de nomes do *VideoCVS Server*.

Além da criação e referenciação dos objetos no serviço de nomes, o *VideoCVS Server* também tem por finalidade aguardar as invocações das solicitações dos *VideoCVS Clients*, que são os clientes remotos. O método *resolveRequest* do *VideoCVS Server* executa o ORB do servidor, que aguarda pelas solicitações dos clientes. Já o cliente remoto é responsável por obter a referência para os objetos do servidor e fazer a chamada aos serviços oferecidos.

Similarmente ao servidor, o cliente usa os s*tubs*[4] gerados pelo compilador IDL como base da aplicação cliente.

Nas classes *VideoTreeFactoryServantImpl* e *VideoNodeFactoryServantImpl* estão implementados os métodos de criação (método *create*) e seleção do objeto (métodos *get*). Os métodos *sendFileToServer* e *getFileFromServer,* dessas classes*,* são os métodos responsáveis por enviar e recuperar o conteúdo de um determinado vídeo para os seus respectivos servidores de arquivos MPEG-2 remotos. É importante ficar claro que, tanto o *VideoCVS Server* como o *Client* instanciam, cada um, um próprio servidor de arquivos MPEG-2.

No exemplo da Figura 32, suponha que um usuário solicita o *commit* de uma nova árvore de versionamento. Essa árvore possui folhas que, por sua vez, apontam para arquivos MPEG-2 no sistema de arquivos da máquina local. Quando o *VideoCVS Client* emite a solicitação, o *VideoCVS Server* atende e repassa para o *ComponentManager*, que é o gerenciador das transações. Ao receber essa operação, o gerenciador faz a verificação e aprova o *commit*, armazenando as informações da nova árvore de versionamento na base. Como o commit foi aprovado, o *ComponentManager* solicita que o mecanismo de transferência de arquivos entre os servidores de arquivos remotos do cliente e do servidor seja realizado. Por fim, o servidor de arquivos remoto do *VideoCVS Client* envia os trechos dos vídeos da nova árvore para o servidor de arquivos MPEG-2 do *VideoCVS Server.*

O mecanismo de transferência de arquivos utilizado e instanciado, por cada *VideoCVS Client* e pelo *VideoCVS Server,* foi o *GridFS* [Santos06]. O *GridFS* é um sistema que permite o gerenciamento de arquivos distribuídos em diversas plataformas e ambientes, sendo capaz de atender um grande número de usuários. Além disso, o desempenho das operações de cópias de arquivos no *GridFS* se mostrou equivalente ao desempenho obtido por servidores e clientes FTP tradicionais. Basicamente, os métodos *sendFileToServer* e *getFileFromServer* realizam o envio do arquivo de um servidor para o outro, utilizando chamadas remotas para transferir os diversos blocos que compõem o arquivo.

---

[4] *Stubs* são interfaces estáticas que atendem as solicitações do cliente.

PUC-Rio - Certificação Digital Nº 0420994/CB

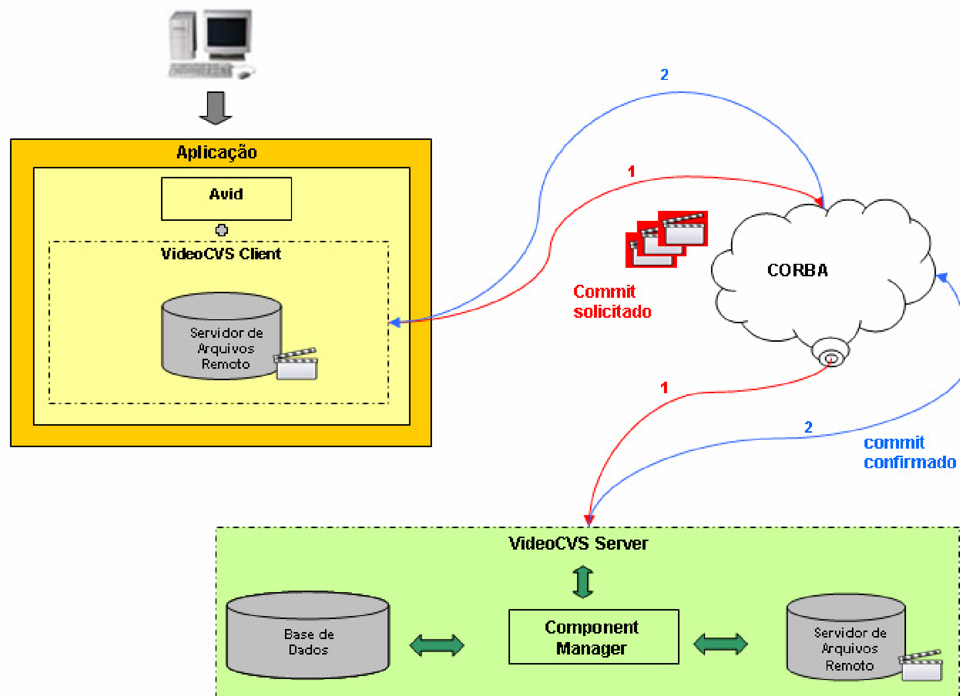

Figura 32 – Exemplo de commit na arquitetura do *VideoCVS*

Os serviços oferecidos pelas fábricas (*factories*) dos objetos criados podem ser visualizados no arquivo IDL definido, no anexo A. Dentre os principais serviços implementados, é importante ser citado:

- o checkout ou recuperação de uma determinada versão da árvore de versionamento;
- o commit de uma determinada árvore de versionamento;
- a criação e edição de uma determinada árvore de versionamento;
- a edição de um nó;
- a fusão entre duas árvores de versionamento;
- a segmentação de um arquivo MPEG-2 de vídeo;
- e a remontagem de uma árvore de versionamento.

Do lado do cliente, o *VideoCVS Client* pode ser integrado a uma ferramenta comercial de edição de vídeo. Essa integração acontece no momento que o cliente

PUC-Rio - Certificação Digital Nº 0420994/CB

deseja editar o conteúdo de uma folha da *VideoTree*. As ferramentas de código aberto utilizadas foram a Mpeg2Cut2[5] e VLC[6].

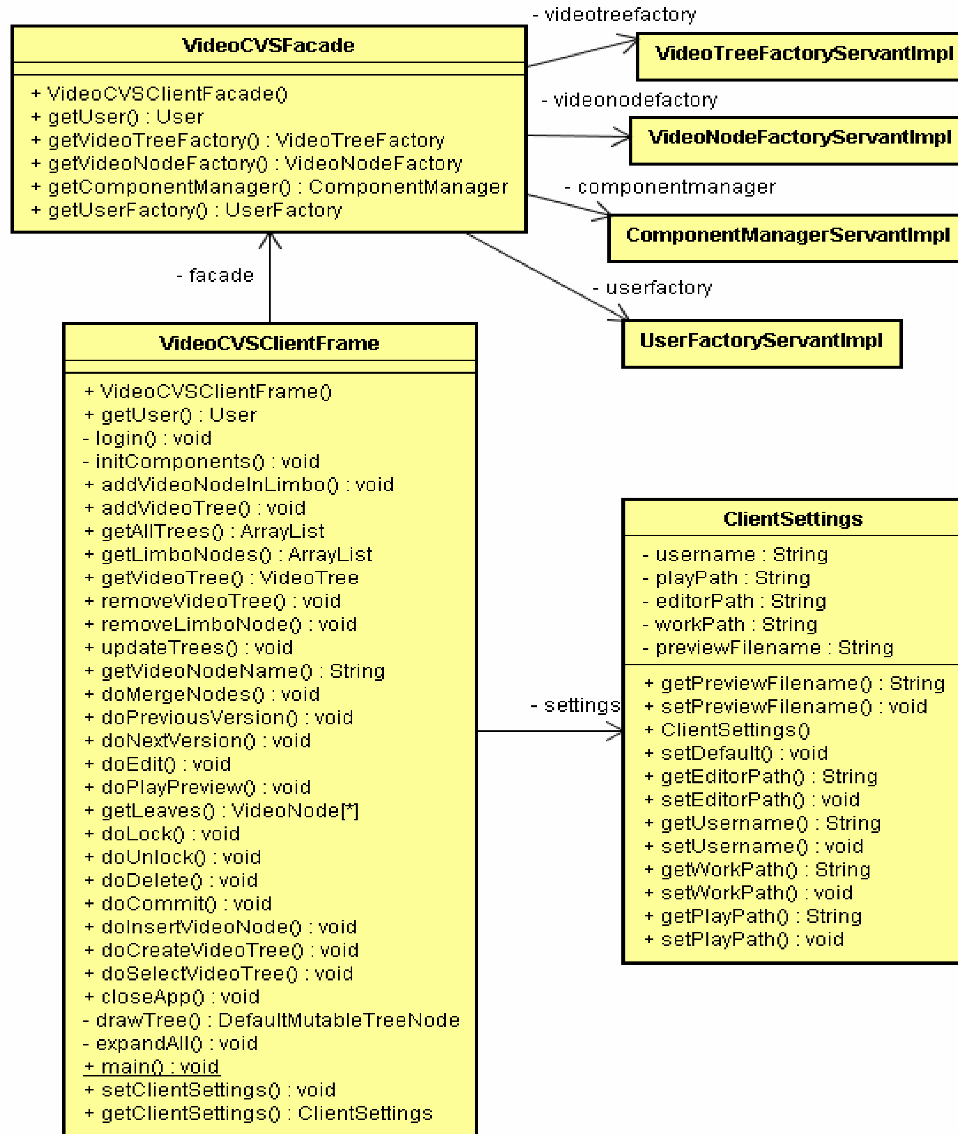

Figura 33 – Diagrama de classes do *VideoCVS Client*

A Figura 33 ilustra o diagrama de classes do *VideoCVS Client*. Uma classe fachada denominada *VideoCVSFacade* foi implementada, baseada no padrão de projeto *Facade*[7]. Para que a fachada implemente e disponibilize os serviços à

[5] http://www.geocities.com/rocketjet4/

[6] http:// www.videolan.org/

[7] Facade é um padrão de projeto onde se define uma classe composta pelas funcionalidades do sistema, abstraindo do cliente aspectos irrelevantes de implementação da API [GHJV95].

PUC-Rio - Certificação Digital Nº 0420994/CB

aplicação cliente, ela recupera as referências das classes *factories,* através do serviço de nomes iniciado no *VideoCVS Server*. Os atributos da classe *ClientSettings* definem a ferramenta de edição utilizada pela aplicação *VideoCVS Client*.

5.3. Modelo Entidade e Relacionamento do *VideoCVS*

A Figura 34 apresenta o modelo entidade e relacionamento (modelo ER) do sistema *VideoCVS*.

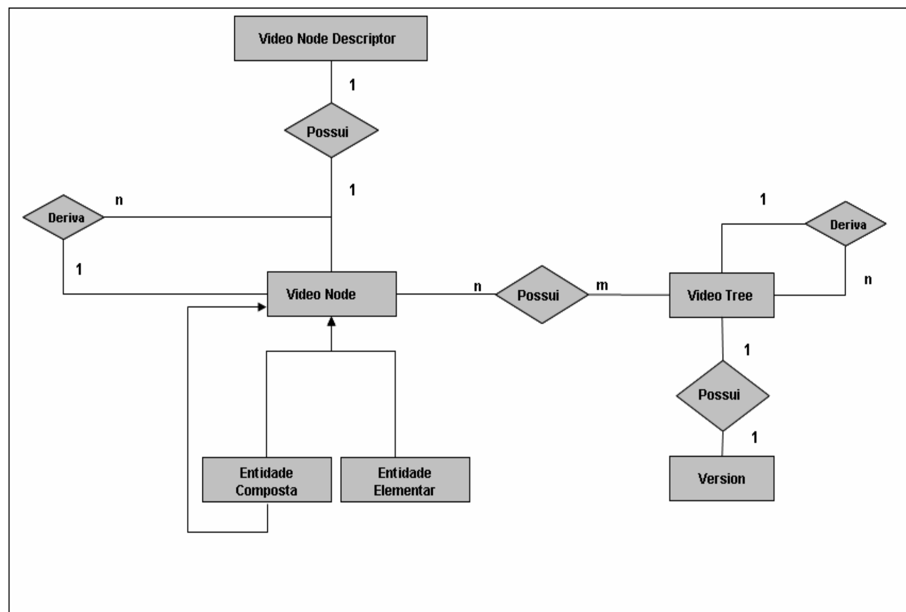

Figura 34 – Modelo ER do sistema *VideoCVS*

No modelo, uma árvore de versionamento de um vídeo corresponde a uma entidade *VideoTree*. Um nó da árvore corresponde a uma entidade *VideoNode*. Uma árvore de versionamento é composta por mais de um nó e um nó pode ser compartilhado por várias árvores. Logo, o relacionamento entre os conjuntos de entidade *VideoTree* e *VideoNode* é n:m.

Cada árvore de versionamento está associada a uma única entidade *Version*, que descreve a versão da árvore. Assim, como cada entidade *Version* só pertence a uma única árvore, o relacionamento entre as entidades é 1:1.

O relacionamento de derivação das árvores e dos nós também estão representados no modelo. Uma árvore de versionamento de origem deriva mais de

PUC-Rio - Certificação Digital Nº 0420994/CB

uma árvore, bem como um nó de origem deriva mais de um nó. Ambos os relacionamentos de derivação são então 1:n.

Uma entidade *VideoNode* pode ser *elementar* ou *composta*. Uma entidade elementar corresponde a uma folha e uma entidade composta corresponde aos nós internos da árvore de versionamento. Cada folha descreve o conteúdo de um trecho do vídeo, portanto uma entidade *VideoNode* pode estar associada a uma entidade *VideoNodeDescriptor.* Uma entidade *VídeoNodeDescriptor* descreve o nome do vídeo apontado, o endereço relativo dele no sistema de arquivos remoto do servidor, a quantidade de quadros e o formato de arquivo do vídeo.

As classes dos objetos remotos que descrevem o modelo ER são apresentadas na Seção 5.3.1. Em seguida, as classes do repositório que representam a camada de persistência do sistema são apresentadas na Seção 5.3.2.

PUC-Rio - Certificação Digital Nº 0420994/CB

5.3.1.
Classes dos Objetos Remotos

As classes que implementam as entidades do modelo ER são objetos remotos CORBA. Por isso, cada classe implementada estende a sua respectiva classe *Portable Object Adapter*[8] (POA). A Figura 35 mostra o diagrama de classes que representa o modelo ER descrito na seção anterior.

No diagrama, a classe *VideoTreeServantImpl* descreve os objetos *VideoTree,* que correspondem a árvores de versionamento de vídeo. Cada objeto desta classe é composto por uma raiz do tipo *VideoNode*, um número inteiro de identificação, um nome do tipo *String*, uma versão do tipo *Version* e um criador da árvore do tipo *User*. Além disso, como uma árvore pode ser derivada de uma árvore de origem, o objeto a ela correspondente possui um atributo *parent* do tipo *VideoTree.* A classe *VersionControlServantImpl* contém os objetos *Version*.

Da mesma forma, a classe *VideoNodeServantImpl* contém os objetos *VideoNode* que correspondem ao nós das árvores. Em cada objeto desta classe, o rótulo do nó é representado por um número inteiro de identificação, um nome do tipo *String* e um criador do nó do tipo *User*. Quando o nó é folha, então o objeto correspondente da classe *VideoNodeServantImpl* também possui o objeto

---

[8] POA é a especificação atual para o adaptador de objetos CORBA e considerada a entidade identificável no contexto de um servidor. http://www.omg.org/

VideoNodeDescriptor, que descreve o nome do arquivo, o número de quadros, o formato do vídeo e endereço relativo onde o arquivo MPEG-2 se encontra. Este último objeto está representado na classe *VideoNodeDescriptorServantImpl*.

Como já comentado na Seção 4.3.4, o estado de um nó descreve o modo de bloqueio efetuado sobre ele, e é uma estrutura representada por quatro elementos. O primeiro elemento do estado de um objeto *VideoNode,* que define o modo de bloqueio nele efetuado, está representado pelo atributo *lock*. O segundo elemento, que define o número de bloqueios, está representado pelo atributo *numLocks*. O atributo *locker* representa o *User* que realizou o bloqueio no modo exclusivo e, finalmente, o atributo *lockDate* representa o *timestamp* de bloqueio exclusivo.

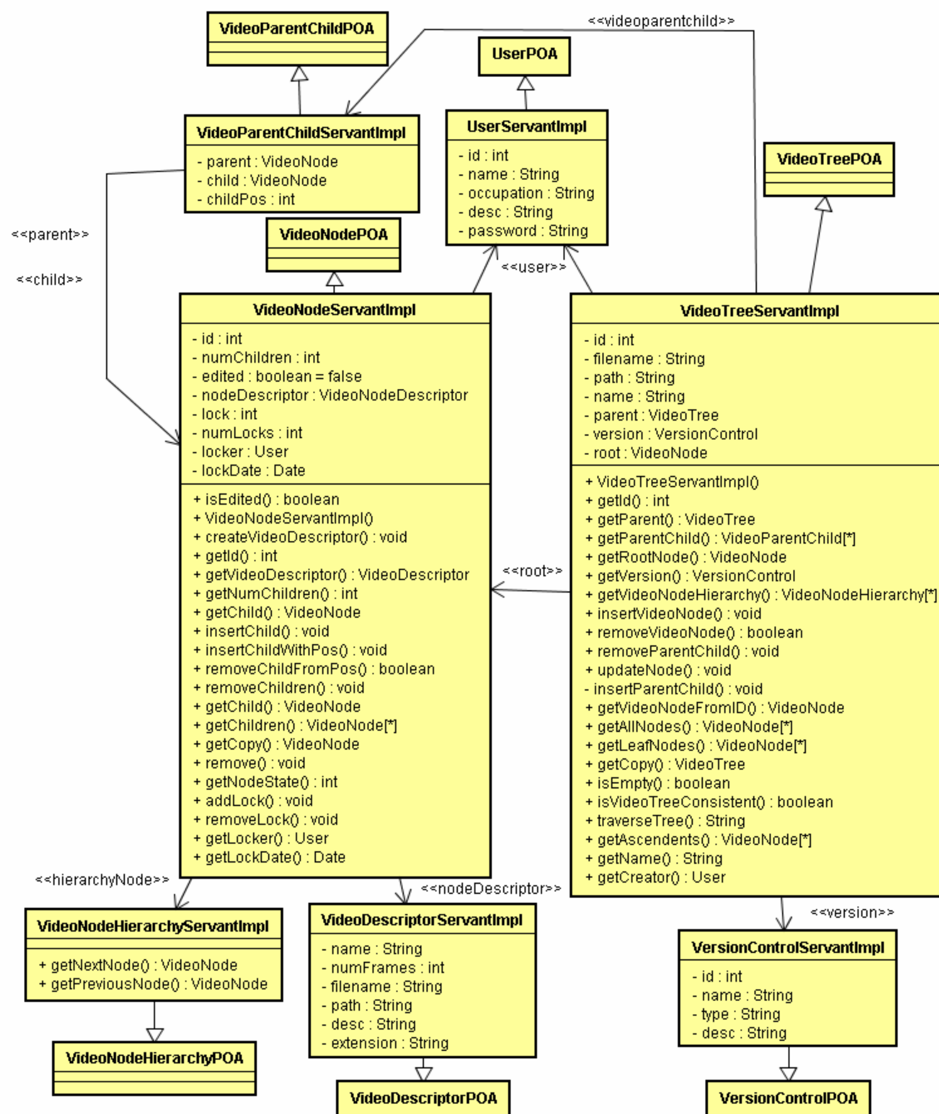

Figura 35 – Diagrama de classes dos objetos remotos do modelo ER

PUC-Rio - Certificação Digital Nº 0420994/CB

Os objetos da classe *VideoParentChildServantImpl* descrevem os relacionamentos entre os nós pais e filhos. Logo, um objeto *VideoTree* representando uma árvore de versionamento possui um conjunto de objetos do tipo *VideoParentChild*. Os objetos da classe *VideoNodeHierarchyServantImpl* descrevem o relacionamento de derivação entre os nós.

5.3.2. Classes do Repositório

A camada de dados do *VideoCVS* cuida da persistência dos objetos remotos durante o funcionamento da aplicação, e do armazenamento e recuperação dos dados. Nela estão definidas as classes do repositório que lêem, gravam, procuram e atualizam os objetos remotos na base de dados. Essas classes estão ilustradas na Figura 36.

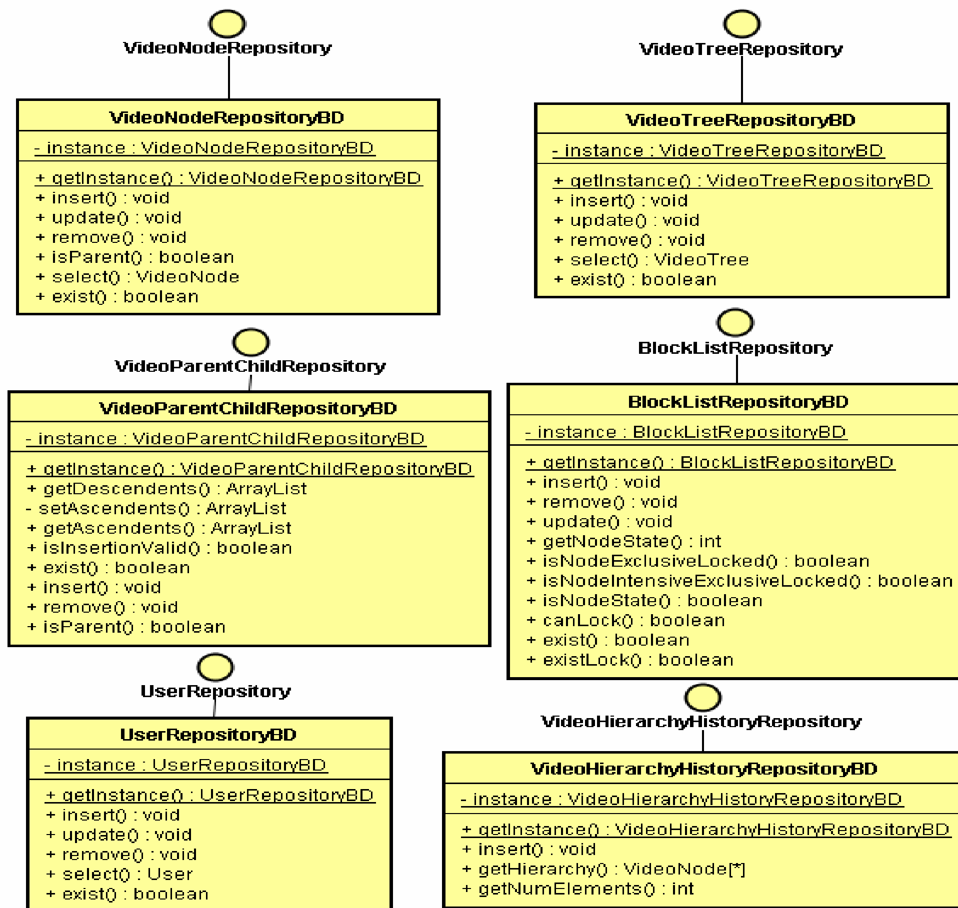

Figura 36 – Diagrama de classes dos objetos remotos do repositório

PUC-Rio - Certificação Digital Nº 0420994/CB

Todas as classes do repositório estão implementadas seguindo o padrão de projeto *Singleton*[9].

5.4. Mecanismo de Segmentação

O mecanismo de segmentação é responsável por particionar um arquivo MPEG-2 de vídeo, em porções reduzidas do mesmo formato. A classe *VideoSegmentation* representa esse mecanismo, ilustrada na Figura 37. A primeira etapa do mecanismo, representada pelo método *setShotIndexes*(), define os índices dos segmentos do vídeo. A segunda etapa, representada pelo método *bordersHandler*(), realiza o tratamento das bordas dos segmentos definidos na etapa inicial. Por último, o método *generateSegments*() gera os segmentos em arquivos no formato MPEG-2 de vídeo.

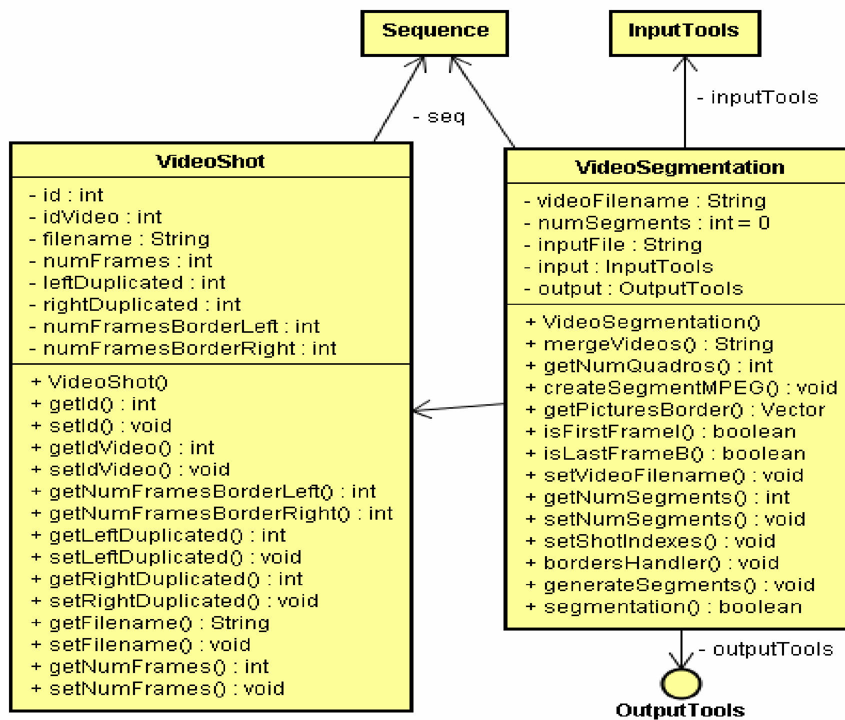

Figura 37 – Diagrama de classes do mecanismo de segmentação

[9] *Singleton* é um padrão de projeto onde se garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto [GHJV95].

PUC-Rio - Certificação Digital Nº 0420994/CB

A classe *VideoSegmentation* tem como atributos o nome do arquivo de entrada a ser segmentado, o objeto *InputTools* e o *OutputTools*. O *InputTools* e *OutputTools* são as classes responsáveis por manipular os arquivos de entrada e saída, respectivamente. Os objetos segmentados são representados pela classe *VideoShot.* Tanto a classe *VideoSegmentation* como a *VideoShot* usam um objeto *Sequence*, que representa a primeira camada da estrutura hierárquica do padrão MPEG-2 vídeo. Toda a estrutura do padrão MPEG-2 vídeo está ilustrada no diagrama de classes da Figura 38.

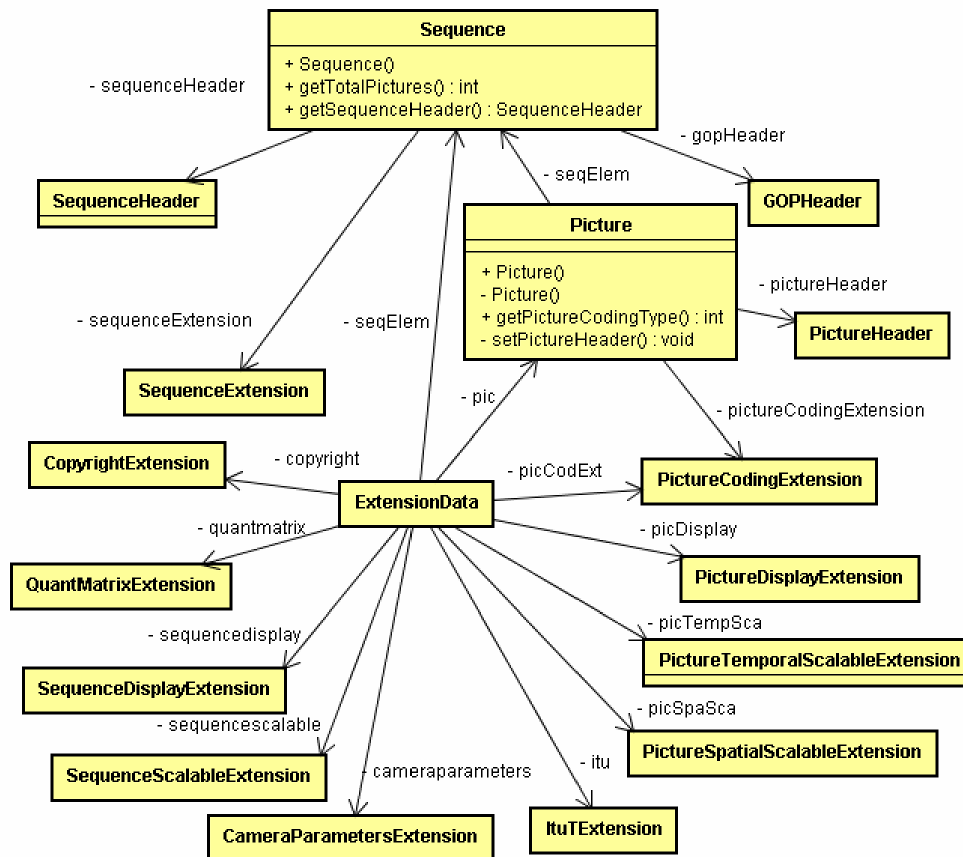

Figura 38 – Diagrama de classes da estrutura do MPEG-2 de vídeo

PUC-Rio - Certificação Digital Nº 0420994/CB

5.5. Mecanismo de Remontagem

O mecanismo de remontagem é responsável por concatenar os conteúdos das folhas da árvore de versionamento e gerar um arquivo MPEG-2 vídeo como resultado. A classe *VideoConcatenation* representa esse mecanismo, ilustrado na Figura 39. O método *bordersHandler*() dessa classe verifica se as bordas dos segmentos definidos no mecanismo de segmentação possuíam quadros duplicados. Caso existam e o objeto *VideoShot* correspondente não tenha sido alterado, o método retira os quadros duplicados das bordas. Por outro lado, caso existam mas o objeto *VideoShot* correspondente tenha sido alterado, o método não retira os quadros duplicados.

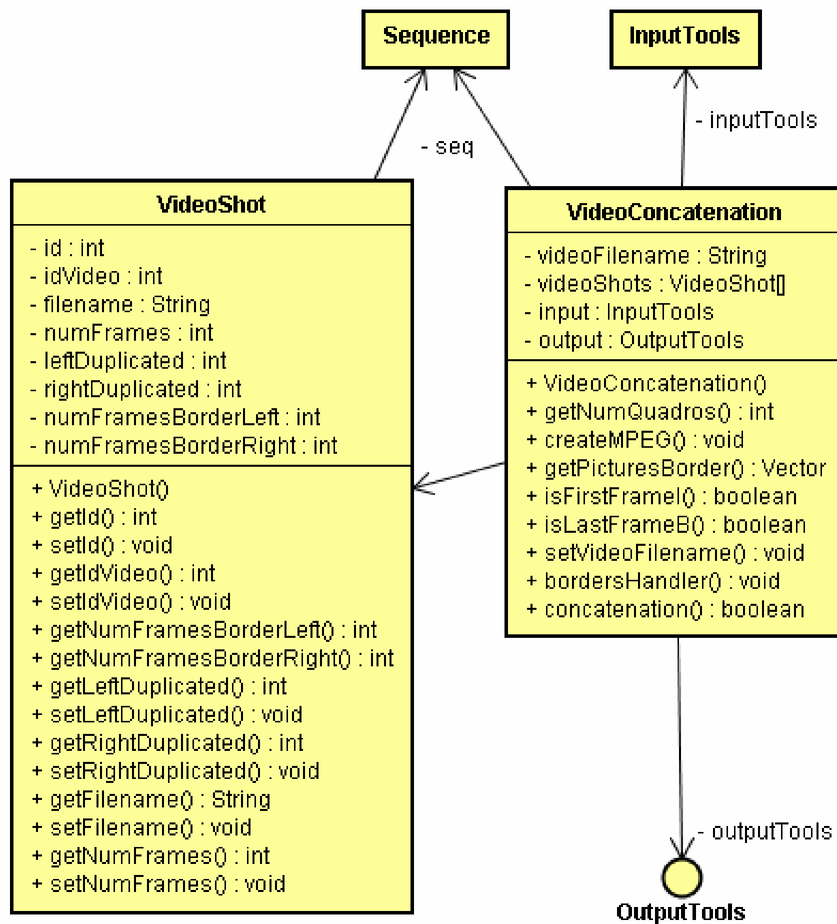

Figura 39 – Diagrama de classes do mecanismo de remontagem

PUC-Rio - Certificação Digital Nº 0420994/CB

Assim como a classe *VideoSegmentation* (descrito na Seção 5.4), a *VideoConcatenation* possui o nome do arquivo do vídeo, o objeto *InputTools*, *OutputTools* como atributos, além do *array* de segmentos do tipo *VideoShot*. Por fim, a classe também usa o objeto *Sequence* para estruturar os arquivos MPEG-2.

5.6. Exemplo de Uso do *VideoCVS*

O componente *VideoCVS Client* é uma aplicação cliente implementada com uma interface gráfica baseada nos pacotes Swing e Awt da linguagem Java. Esta seção apresenta um exemplo de uso do VideoCVS num ambiente colaborativo de edição de vídeo. Primeiramente, esta seção apresenta como uma árvore de versionamento de origem é criada e como pode ser feito *checkout*. Além disso, é mostrado como o usuário pode utilizar o mecanismo de segmentação do vídeo. Em seguida, apresenta como são editados colaborativamente os nós da árvore de versionamento por dois usuários distintos, dispersos geograficamente. Mensagens exemplificando o protocolo de bloqueio em duas fases e o conceito de granularidade múltipla, na edição cooperativa da árvore criada, são também mostrados. A seção também apresenta a fusão entre duas árvores de versionamento criadas por dois usuários. Por fim, a seção apresenta um exemplo de remontagem dos vídeos de uma árvore de versionamento.

PUC-Rio - Certificação Digital Nº 0420994/CB

5.6.1. Criação de uma Árvore de Versionamento

No exemplo de uso do VideoCVS, suponha dois usuários, 1 e 2, dispersos geograficamente, como sendo editores de vídeo. Os usuários utilizam o VideoCVS como ferramenta para controlar as versões de suas respectivas edições. O arquivo utilizado no exemplo é o de uma partida de *ping-pong* realizada entre dois competidores e vista por alguns telespectadores, composto por 450 quadros e codificado no formato MPEG-2 vídeo.

A Figura 40 ilustra a interface gráfica do VideoCVS Client e a opção *Remote->New vídeo tree*, da barra de ferramentas. Basicamente, o *VideoCVS Client* contém dois painéis, onde o primeiro, denominado *principal*, exibe as

árvores de versionamento e seus respectivos nós, e o segundo, denominado *console*, exibe os comandos realizados e algumas informações extras.

Uma vez selecionada a opção *Remote->New vídeo tree*, o sistema exibe duas telas, solicitando que o usuário informe os dados iniciais da árvore de versionamento de origem que será criada. Essas duas telas estão ilustradas na Figura 41 e Figura 42.

Na primeira tela de criação da árvore, o usuário informa o nome, o arquivo de vídeo, o tipo da versão e a árvore pai. Como esta árvore é de origem, nenhuma *Vídeo Tree Parent* é informada. Na segunda tela, o usuário informa o nome da raiz da árvore, além de decidir se haverá a segmentação do arquivo de vídeo informado. O mecanismo de segmentação é realizado, quando o usuário informa os índices dos quadros onde haverá os cortes, além de selecionar a opção *Shots generated automatically*. A Figura 42, mostra que os índices dos segmentos são informados pelo usuário 1, manualmente. O usuário 1 informa que o vídeo deve ser segmentado nos índices 190 e 302 do arquivo original. Como o arquivo possui mais de 302 quadros, o sistema particiona o vídeo em 3 segmentos.

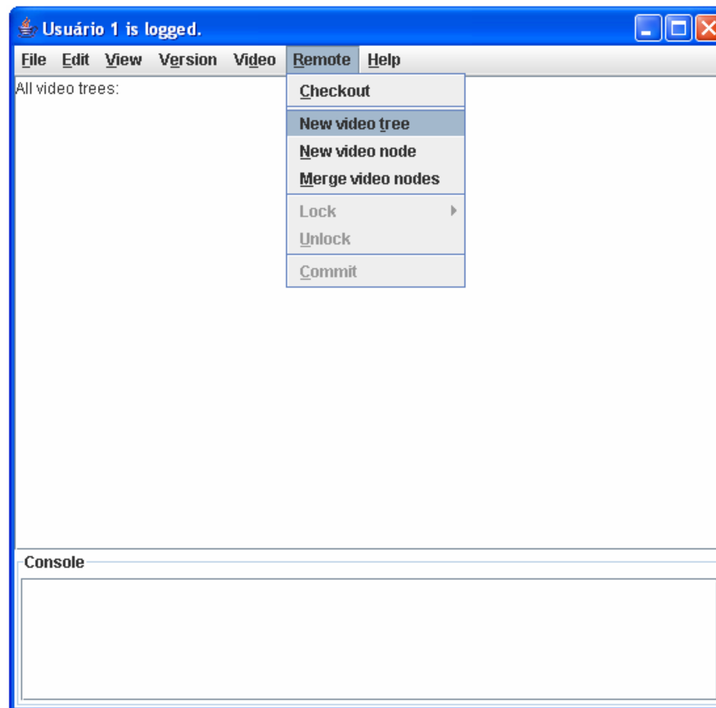

Figura 40 – Tela inicial do VideoCVS Client

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

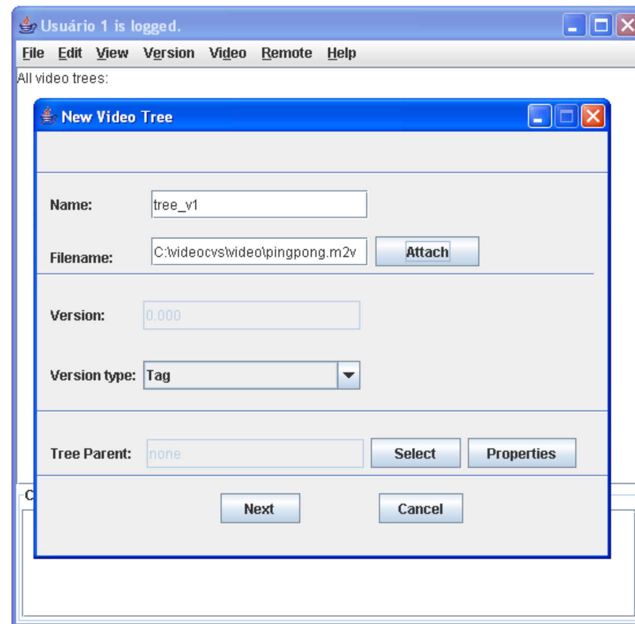

Figura 41 – Primeira tela da criação da árvore de versionamento

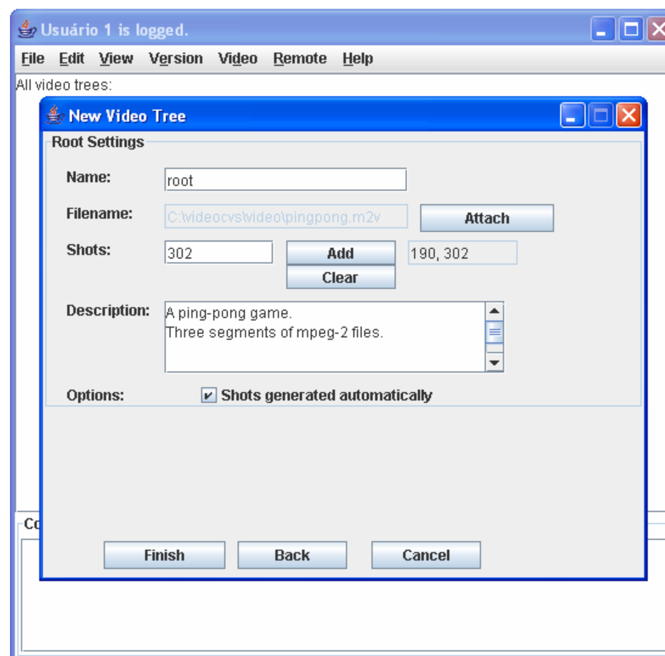

Figura 42 – Segunda tela da criação da árvore de versionamento

No intervalo do segmento 1, o vídeo original inicia com um quadro I e termina com um quadro B, logo foi preciso ser feito o tratamento da borda da direita deste segmento. Depois do tratamento, o segmento 1 passou a ter 1 quadro a mais. No intervalo do segmento 2, o vídeo original iniciava com quadro I e terminava com um P, logo não foi preciso ser feito o tratamento da bordas. Por

fim, como o segmento 3 iniciava com um quadro B, foi preciso ser feito o tratamento da borda da esquerda. Depois do tratamento, o segmento passou a possuir quatro quadros a mais.

Após os tratamentos dos segmentos, o mecanismo de segmentação gera os três nós da árvore descritos por: *pinpong_shot0.m2v*, *pinpong_shot1.m2v* e *pinpong_shot2.m2v*. Os quadros iniciais dos três segmentos estão ilustrados na Figura 43.

Figura 43 – Quadros iniciais dos segmentos do vídeo *pinpong.m2v*

Depois que a árvore é gerada, o usuário 1 realiza o *commit*, gerando assim a primeira versão da árvore de versionamento na base de dados. O console da Figura 44 ilustra o commit realizado.

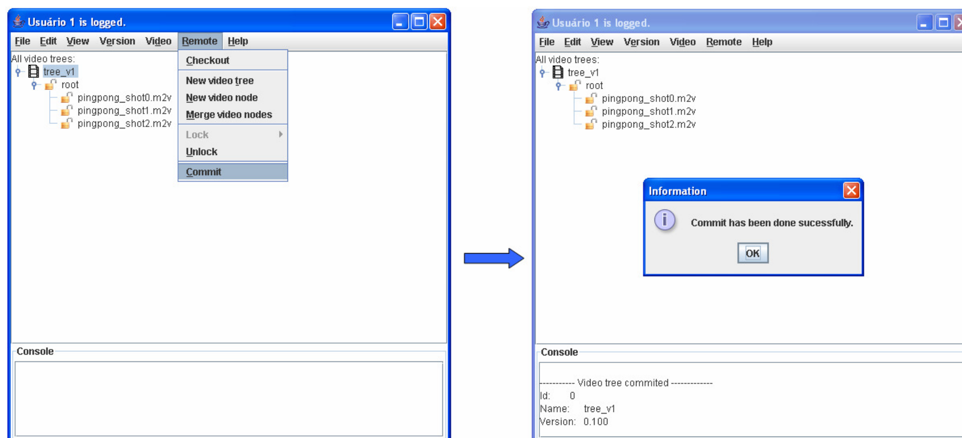

Figura 44 – Commit da árvore de versionamento pelo usuário 1

PUC-Rio - Certificação Digital Nº 0420994/CB

5.6.2.
Checkout de uma Árvore de Versionamento

Após a criação da árvore de versionamento tree_v1 pelo usuário 1, suponha que o usuário 2 tinha conhecimento que havia uma versão 0.1 da árvore de versionamento do vídeo *pingpong.m2v*. Logo, ele solicita o checkout da versão, como mostra a Figura 45. A solicitação de checkout de uma versão é realizada através da opção *Remote->Checkout* selecionada na barra de ferramentas.

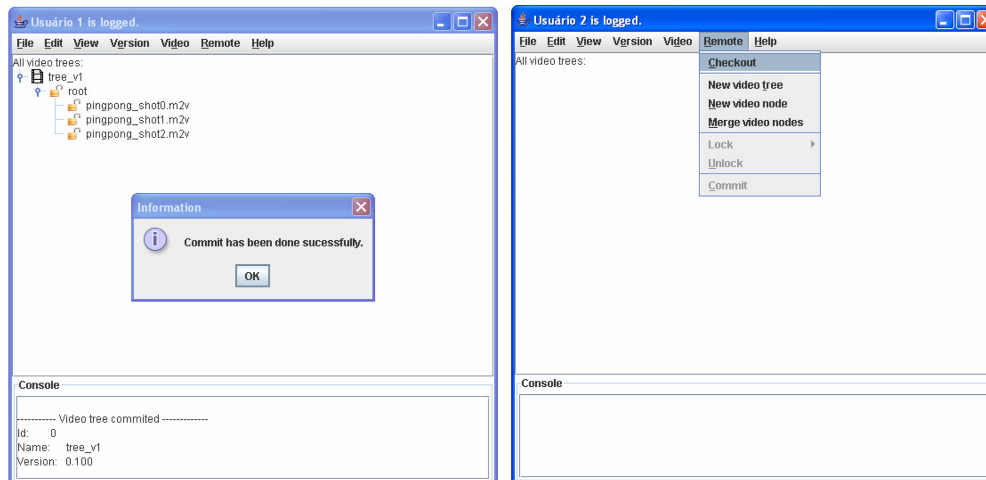

Figura 45 – Operação de *checkout* de uma árvore de versionamento

Quando o usuário 2 solicita o *checkout*, o sistema exibe uma outra tela como mostra a Figura 46. A tela exibida solicita que o usuário informe os campos de procura da árvore de versionamento. Essa busca pode ser realizada pelo nome da árvore de versionamento, número de versão ou nome do usuário que criou a árvore. Após informar alguns desses dados, o usuário deve clicar no botão *Search*, para que a busca seja realizada. Quando a busca é concluída, o sistema exibe, na mesma tela, as árvores de versionamento que satisfazem os campos informados pelo usuário.

PUC-Rio - Certificação Digital Nº 0420994/CB

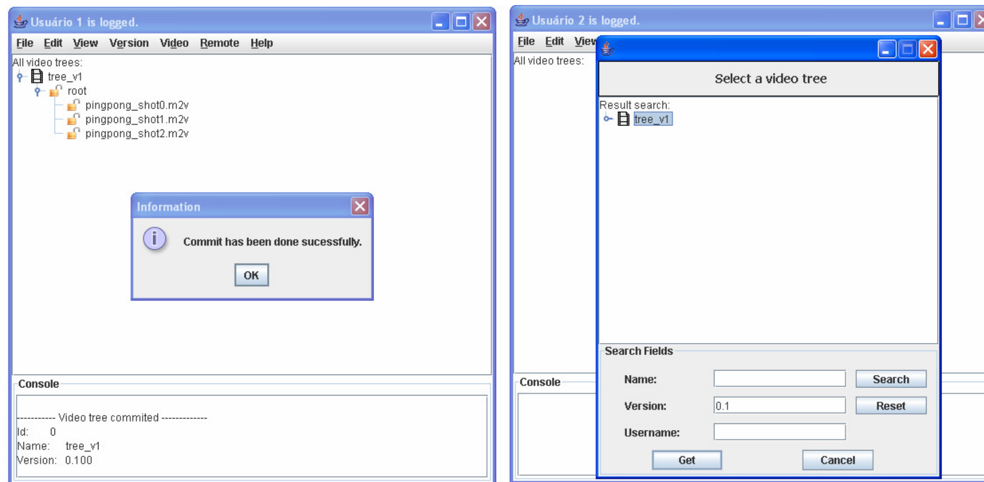

Figura 46 – Resultado da busca do *checkout*

No exemplo da Figura 46, o usuário 2 busca a árvore de versionamento *tree_v1*, criada pelo usuário 1, informando o número de versão 0.1. O sistema então encontra a versão da árvore solicitada e exibe o resultado da busca. Para que a *tree_v1* seja carregada no painel principal do sistema, o usuário seleciona a árvore e clica no botão *Get*.

5.6.3. Edição Colaborativa de uma Árvore de Versionamento

No exemplo da Figura 47, o usuário 1 deseja alterar o nó *pinpong_shot0.m2v*, e por isso, ele o bloqueia, de forma explícita, no modo exclusivo. A Figura 48 mostra que, quase no mesmo instante, o usuário 2 tenta bloquear, de forma explícita no modo exclusivo, o nó pai de *pinpong_shot0.m2v*. Entretanto, o sistema verifica que o nó já está bloqueado, no modo *intencional-exclusivo* (IX), e como não é compatível com o bloqueio solicitado pelo usuário 2, o bloqueio é negado. O mesmo acontece, quando o usuário 2 tenta bloquear o nó *pingpong_shot0.m2v*, só que dessa vez, no modo compartilhado, como mostra a Figura 49.

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

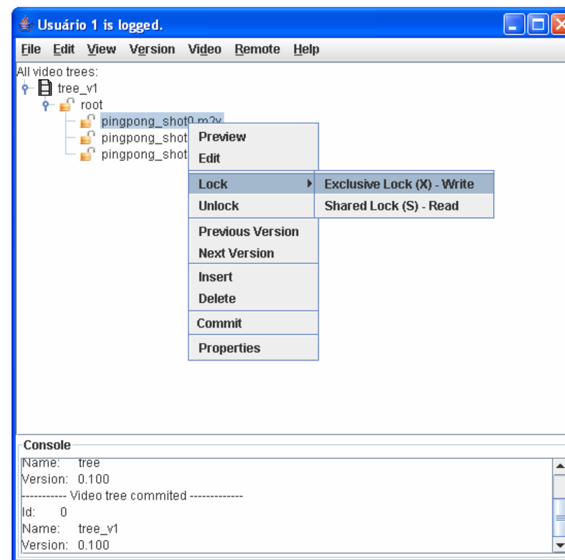

Figura 47 – Bloqueio no modo exclusivo permitido ao usuário 1

Opções de *preview* e edição de um nó são possíveis, como mostra o *popup* exibido na Figura 47. Quando o usuário clica na opção *Preview*, o sistema toca o vídeo num determinado *player* de exibição que é determinado pelo usuário, como mostra a Figura 50. Da mesma forma, quando o usuário clica na opção *Edit*, o sistema abre o conteúdo da folha da árvore de versionamento na ferramenta de edição de vídeo do usuário. Tanto o *player* de exibição como a ferramenta de edição de vídeo são determinados através da opção *Vídeo->Preferences*, da barra de ferramentas do *VideoCVS*. Além disso, outras operações como inserção, remoção de um nó, commit e propriedades do elemento selecionado (nó ou árvore de versionamento) são disponibilizadas no sistema.

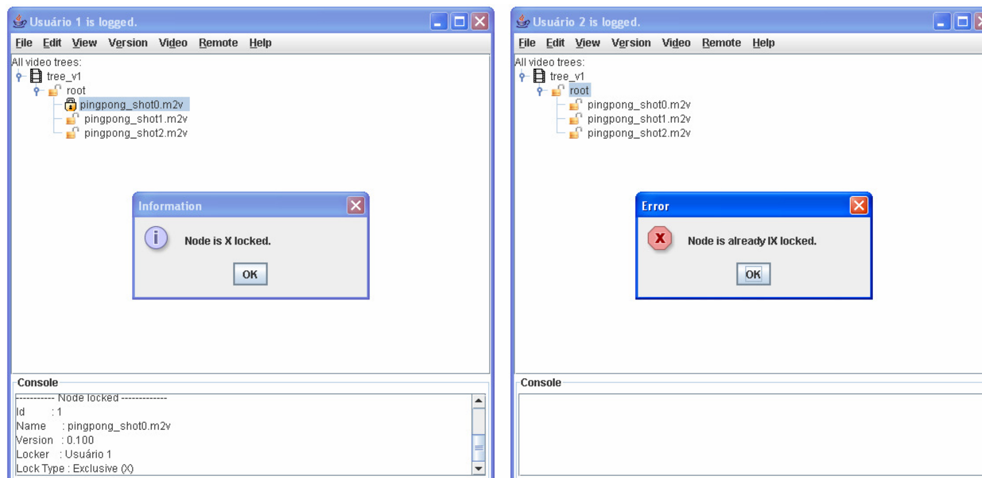

Figura 48 – Bloqueio no modo exclusivo negado ao usuário 2

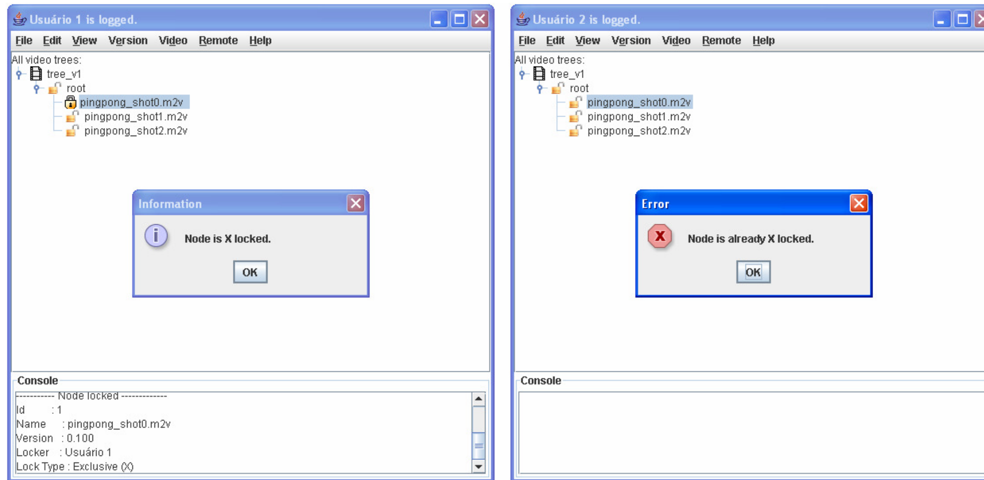

Figura 49 – Bloqueio no modo compartilhado negado ao usuário 2

Como existiam outros nós sem bloqueios, o usuário 2 consegue realizar os bloqueios de *pinpong_shot1.m2v* e *pinpong_shot2.m2v*, de forma explícita, no modo exclusivo, como mostra a Figura 51.

Assim que o usuário 2 decide desbloquear um determinado nó, a fase de encolhimento do protocolo de bloqueio em duas fases é iniciado. A Figura 52 mostra que o usuário inicia essa fase desbloqueando o nó *pinpong_shot1.m2v*.

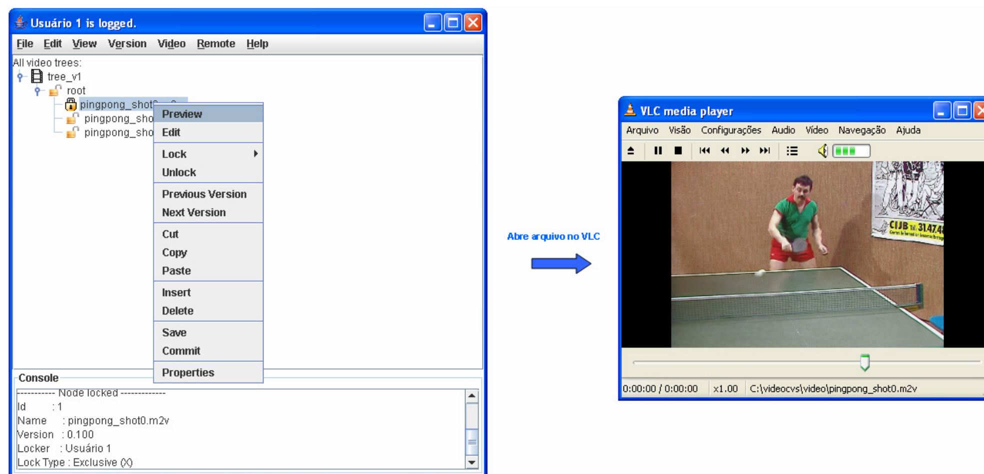

Figura 50 – Clicando na opção Preview para visualizar o vídeo

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

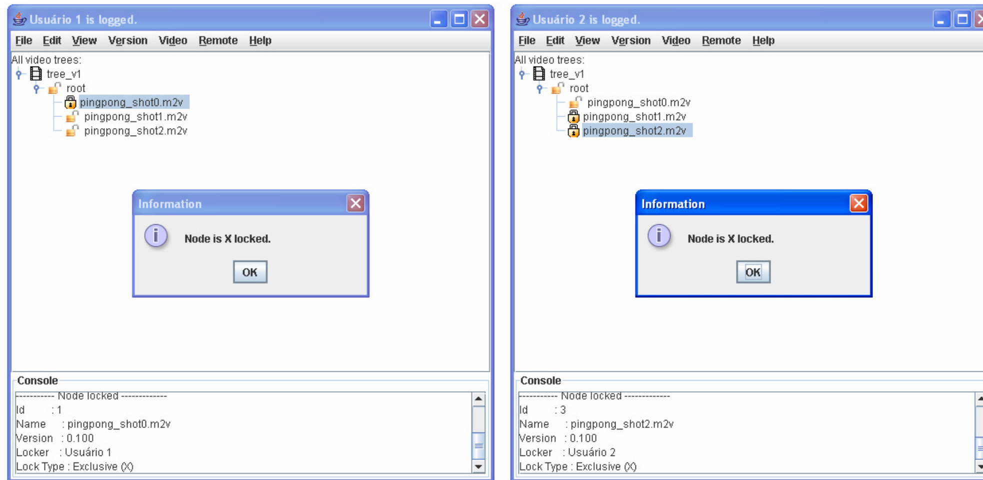

Figura 51 – Bloqueios no modo exclusivo permitido ao usuário 2

Quando a fase de encolhimento é iniciada, só é permitido ao usuário 2 realizar desbloqueios e edições dos nós bloqueados por ele. Como o usuário 2 tenta novamente bloquear o nó *pinpong_shot1.m2v,* o sistema informa que é preciso desbloquear todos os seus bloqueios, para que possa iniciar uma outra fase de expansão (bloqueios), como mostra a Figura 53.

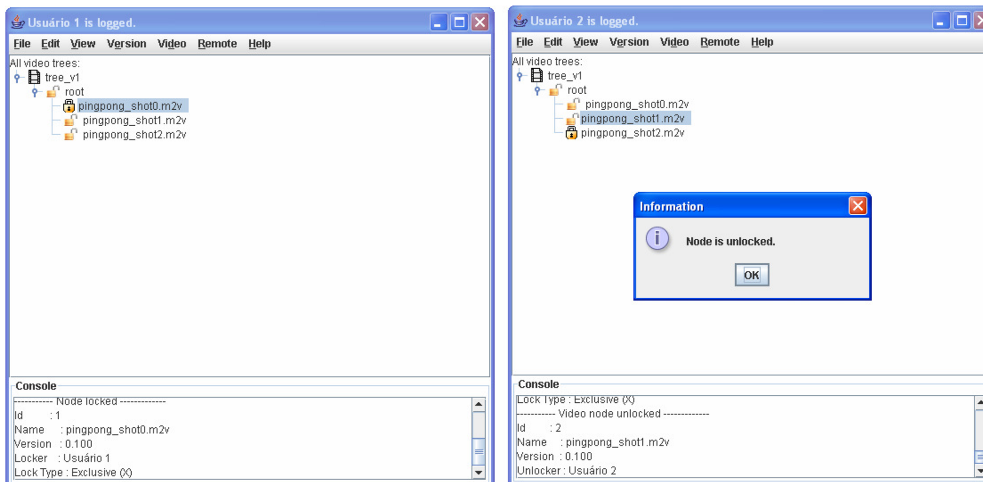

Figura 52 – Desbloqueio de um nó da árvore de versionamento pelo usuário 2

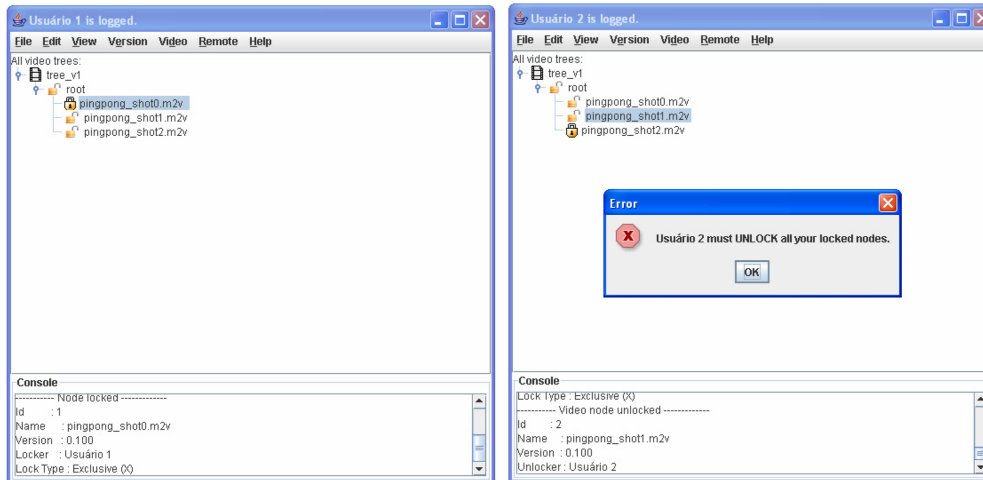

Figura 53 – Permissão negada para bloquear na fase de encolhimento

5.6.4. Fusão entre Árvores de Versionamento

A operação de *merge* de uma árvore de versionamento é selecionada na opção *Version->Merge Versions*, da barra de ferramentas do *VideoCVS*, como está ilustrado na Figura 54. Duas árvores de versionamento, *tree_v12* e *tree_v13* foram criadas, pelos usuários 1 e 2, respectivamente. Como *tree_v12* e *tree_v13* são derivadas da árvore *tree_v1* (do exemplo 1), elas são árvores consideradas irmãs e os nós entre elas são equivalentes derivados.

A Figura 55 ilustra a árvore resultante do mecanismo de fusão entre a *tree_v12* e *tree_v13*. É importante notar que não foi precisa uma intervenção manual do usuário que solicitou a fusão.

PUC-Rio - Certificação Digital Nº 0420994/CB

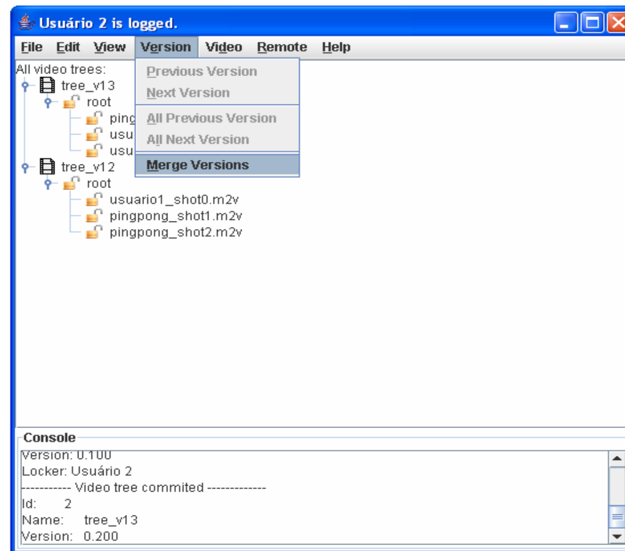

Figura 54 – *Merge* entre versões

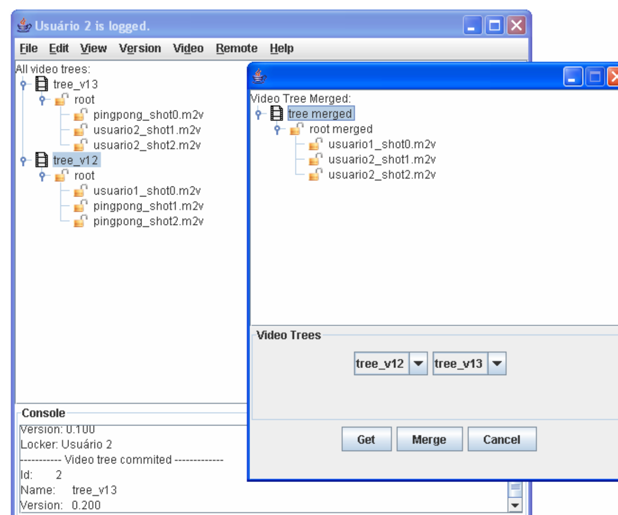

Figura 55 – Fusão entre as árvores de versionamento *tree_v12* e *tree_v13*

Na Figura 56, é apresentado um exemplo de fusão entre duas árvores de versionamento, *tree_v13* com a *tree_v14,* em que é precisa uma intervenção manual do usuário na decisão do segundo filho da raiz da árvore resultante. O sistema ao verificar que os nós *usuario1_shot1.m2v* e *usuario2_shot1.m2v* são novas versões do nó de origem (*pingpong_shot1.m2v* da *tree_v1*), solicita que o usuário escolha qual dos dois nós é preferido. A Figura 57 ilustra a árvore

PUC-Rio - Certificação Digital Nº 0420994/CB

resultante do *merge* entre *tree_v13* e *tree_v14*, com a seleção da primeira opção e, em seguida, com a seleção da segunda opção.

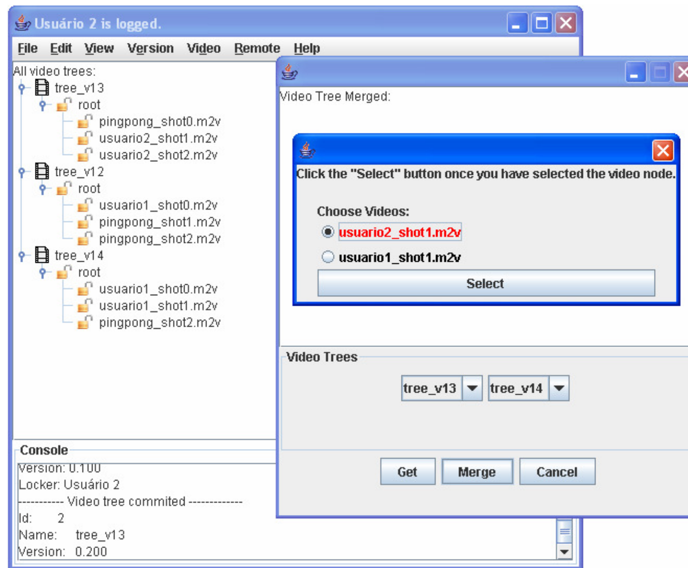

Figura 56 – Intervenção manual do usuário na fusão entre *tree_v13* e *tree_v14*

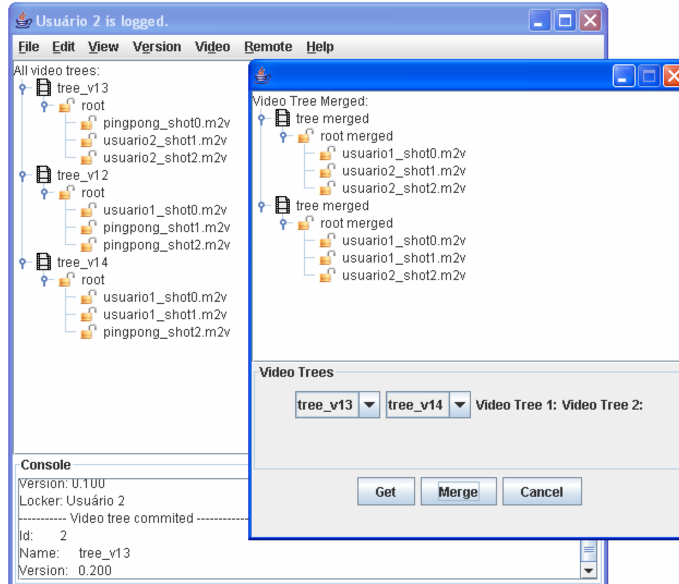

Figura 57 – Árvores de versionamento resultantes após intervenção do usuário

PUC-Rio - Certificação Digital Nº 0420994/CB

5.6.5. Remontagem dos Segmentos da Árvore de Versionamento

A remontagem dos segmentos de uma árvore de versionamento é realizada através da opção *Vídeo->Preview* da barra de ferramentas. Para mostrar essa operação, será apresentado um segundo exemplo.

No segundo exemplo, suponha que o usuário 1 cria uma árvore de versionamento denominada de *tree_v2*, cuja raiz tem o nome de *root*, e a raiz possui um único filho de nome *Gols*. A inclusão de um nó na árvore de versionamento é realizada através da opção *Insert* do *popup*, como mostra a Figura 58. Ao nó *Gols,* são incluídas três folhas que possuem vídeos de gols feitos durante algumas copas do mundo. A primeira folha exibe um gol da copa do mundo de futebol de 1986, a segunda folha exibe um outro gol na copa de 2002, e por último, a terceira exibe um gol da copa de 58. A árvore resultante para remontagem é ilustrada na Figura 59.

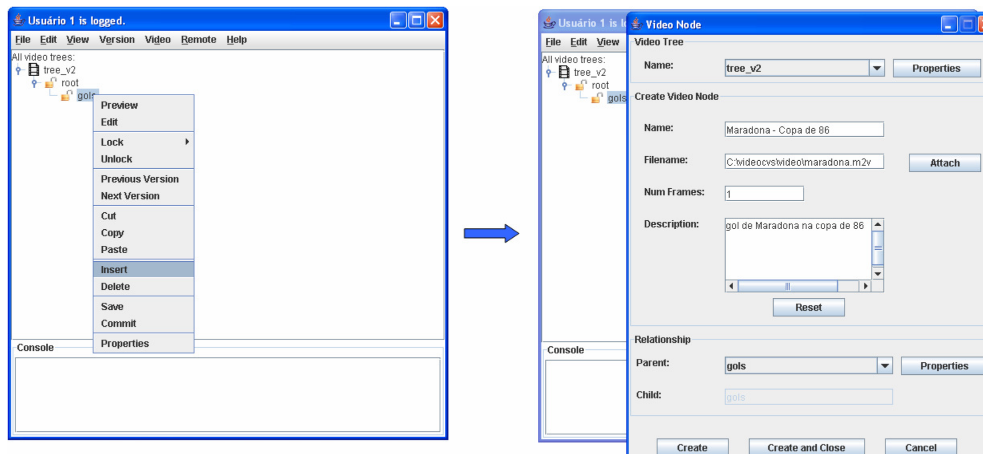

Figura 58 – Operação de inserção de nós na árvore de versionamento

Depois que o usuário seleciona a opção *Vídeo->Preview* da barra de ferramentas, o sistema abre o vídeo concatenado no *player* de exibição, como mostra a Figura 60.

PUC-Rio - Certificação Digital Nº 0420994/CB

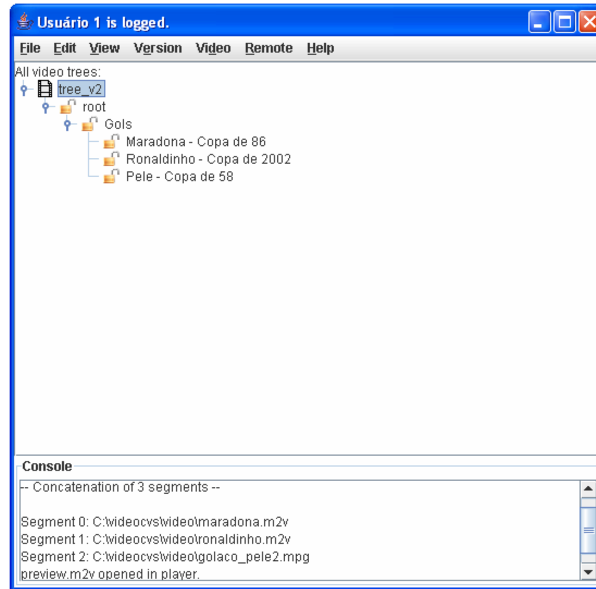

Figura 59 – Árvore de versionamento *tree_v2* para remontagem

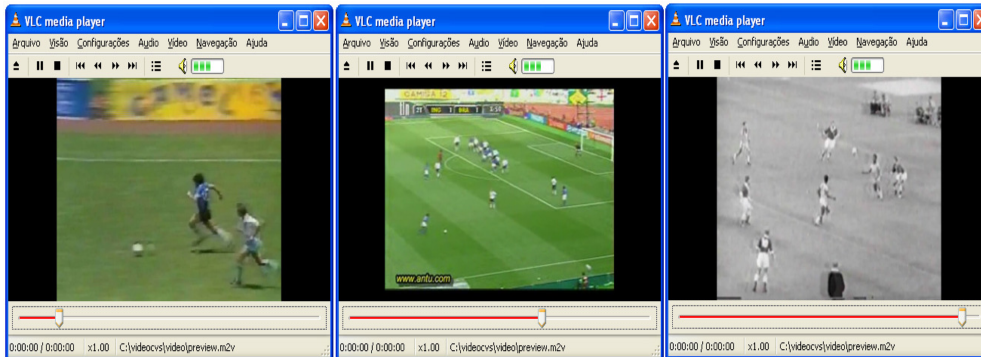

Figura 60 – *Preview* da remontagem da tree_v2

PUC-Rio - Certificação Digital Nº 0420994/CB

PUC-Rio - Certificação Digital Nº 0420994/CB

6
Conclusão

Neste trabalho, foi apresentado um sistema colaborativo capaz de controlar as versões das edições de um vídeo no formato MPEG-2, sem que os editores estejam no mesmo local, ao mesmo tempo. Este capítulo faz o fechamento da dissertação, apresentando as principais contribuições e propondo trabalhos futuros.

6.1.
Contribuições da Dissertação

As principais contribuições desta dissertação foram: definir uma estrutura de dados que permita o controle de versão das edições cooperativas de vídeo no formato MPEG-2; especificar um sistema colaborativo de edição de vídeo entre diferentes editores; definir um mecanismo de remontagem do vídeo no domínio comprimido; e implementar o primeiro protótipo do sistema de controle de versão para edição cooperativa de vídeo.

A definição de uma estrutura de dados para prover o controle de versão das edições cooperativas de vídeo permitiu o gerenciamento e a recuperação das edições do vídeo versionado, no decorrer do tempo. O conteúdo audiovisual foi organizado numa estrutura de dados chamada de *árvore de versionamento*, cujas sub-árvores de cada nó estão ordenadas, e as folhas descrevem um conjunto ordenado de trechos do vídeo. Essa definição também permite o compartilhamento de versões entre as árvores de versionamento e a extração de informações de autoria de cada versão.

A especificação de um sistema de edição colaborativa de vídeo permite a divisão de tarefas e o trabalho concorrente no processo de produção entre os editores. A partir dessa especificação, foi desenvolvido um sistema distribuído capaz de controlar os acessos simultâneos da árvore de versionamento e realizar a fusão das contribuições das versões. O sistema permite que usuários acessem versões de um vídeo ou trecho dele, mesmo estando dispersos geograficamente.

Apesar de não ter sido um objetivo deste trabalho, um mecanismo de segmentação do vídeo MPEG-2 foi implementado no domínio comprimido para permitir aos editores a manipulação dos segmentos, de forma independente. O tratamento das bordas foi utilizado quando ocorrem dependências de quadros nas bordas dos segmentos adjacentes entre si. A duplicação de nós dependentes das bordas entre os segmentos consecutivos é a técnica utilizada para que os segmentos possam ser visualizados corretamente nos *players*. Um algoritmo de detecção automática de tomadas foi apresentado, embora a implementação atual do sistema não contemple tal funcionalidade.

Outro mecanismo apresentado foi o da remontagem dos segmentos do vídeo MPEG-2, também no domínio comprimido. Este mecanismo concatena os vídeos das folhas da árvore de versionamento, na composição da versão final do vídeo comprimido.

A implementação do sistema foi desenvolvida sobre a plataforma Windows e Linux, utilizando a linguagem de programação Java, bibliotecas padronizadas e CORBA. O protótipo implementado, denominado *VideoCVS*, é um sistema distribuído baseado na arquitetura de comunicação cliente e servidor. Foram desenvolvidos os componentes do cliente e servidor para armazenar as versões e os arquivos audiovisuais, e o gerenciador das requisições dos clientes. Foi feito a integração entre o *VideoCVS* e ferramentas de edição de vídeo.

6.2.
Trabalhos Futuros

Dentre as principais características a serem incorporadas ao sistema *VideoCVS*, em trabalhos futuros, estão: a inclusão do mecanismo de detecção de tomadas de cenas automático; o suporte a outros tipos de formatos de mídias, como MPEG-4; outros meios de tratamento das bordas dos segmentos e controle de buffer quando incorporados os mecanismos de segmentação e remontagem do arquivo audiovisual; o estudo de mecanismos de fusão entre arquivos audiovisuais no formato MPEG; a inclusão de anotações em vídeo; o controle de acesso baseado em papéis e a notificação de modificações das árvores de versionamento entre os editores, em tempo real; o armazenamento das operações realizadas sobre

PUC-Rio - Certificação Digital Nº 0420994/CB

os vídeos da árvore; e o aperfeiçoamento do protótipo quanto às funcionalidades e interface gráfica.

A utilização do mecanismo de detecção automática de tomadas de cenas no *VideoCVS* poderia trazer contribuições e refinamentos na estruturação da árvore de versionamento. O mecanismo seria essencial para indexação, recuperação e navegação (*browsing*) dos acervos de vídeo.

O padrão de vídeo MPEG-4 é hoje uma das apostas para as próximas gerações de aplicações multimídia (TV Digital, Ambientes Hipermídias) da indústria. Diferente do padrão MPEG-2, que foca principalmente nos aspectos relacionados à codificação e decodificação linear de vídeo e áudio, o MPEG-4 define uma codificação baseada em objetos e possui algumas vantagens como: o reuso do material audiovisual, dada a abordagem orientada à objeto do padrão; a possibilidade de codificação dos objetos usando diferentes resoluções espaciais e temporais; a possibilidade de uso de técnicas de compactação e compressão que podem ser aplicadas individualmente, de acordo com as características de cada mídia; e a utilização de objetos sintéticos, que variam desde objetos simples de duas dimensões, como linhas e pontos, até objetos complexos de três dimensões, como animações faciais e corporais, em conjunto com os demais objetos de mídia naturais, como câmeras, microfones etc.

Embora o VideoCVS realize a segmentação e a remontagem de arquivos MPEG-2 vídeo no domínio comprimido, o que garante a qualidade do vídeo original, em termos de confiabilidade o mecanismo pode não ser satisfatório. A duplicação de quadros nas bordas dos segmentos dependentes entre si, sem o controle de bits de cada quadro pode gerar overflow para segmentos de tamanhos grandes e underflow para segmentos de tamanhos pequenos. Para otimizar esses mecanismos, é preciso manipular corretamente a quantidade de bits de cada quadro, a taxa de bits do fluxo codificado e o tamanho do buffer, depois das edições de cada segmento e na remontagem deles. Essa reformulação de bits, taxas e tamanho do buffer são operações que necessitam de estudos mais aprofundados, visto que devem ser realizadas em tempo real, no fluxo MPEG de sistemas e no domínio comprimido.

O mecanismo de fusão entre dois arquivos de vídeo poderia produzir um único arquivo de vídeo resultante. No *VideoCVS*, esse mecanismo é feito entre duas versões da árvore de versionamento, que produz uma árvore resultante, mas

PUC-Rio - Certificação Digital Nº 0420994/CB

nunca dois arquivos MPEG-2 de vídeo são juntados em um único arquivo de vídeo. Quando há nós considerados equivalentes derivados mas com timestamp maior que o nó de origem, uma intervenção manual é solicitada pelo sistema. Caso houvesse um mecanismo de fusão entre os conteúdos dos nós, esse mecanismo poderia substituir a intervenção manual. Como o formato MPEG-2 tem uma estrutura, seria interessante estudar a possibilidade de realizar esse mecanismo, comparando-se as diferenças entre os arquivos. A fusão entre arquivos MPEG-2 poderia permitir que usuários modificassem uma mesma folha de uma árvore de versionamento, em paralelo.

Uma busca refinada dos conteúdos dos nós poderia estar presente no VideoCVS. Como o VideoCVS não implementou o conceito de anotação de vídeo, o sistema é incapaz de permitir que usuários possam facilmente encontrar e também definir determinados tipos, categorias, tamanhos, resoluções de vídeos, por exemplo. A inclusão de anotações em vídeo poderia ser realizada, visto que a visualização dos segmentos anotados e também a consulta e navegação em seu conteúdo indexadas pela lista de anotações poderiam trazer benefícios aos editores. As anotações poderiam ser definidas como comentários inseridos em regiões que são destacadas em um ou mais quadros do vídeo. E essa inclusão poderia ser feita a partir da extração automática da semântica de um vídeo (ou de parte dele) ou mesmo manualmente.

O primeiro protótipo do VideoCVS não gerencia quais usuários estão modificando uma determinada árvore de versionamento nem possui um controle de acesso baseado em papéis. Uma notificação de modificações entre usuários poderia permitir que houvesse uma maior colaboração na edição de uma árvore de versionamento. Em tempo real, o sistema não possui uma rotina de verificação de quais usuários estão modificando um determinado nó da árvore. Como é possível que alguns nós possam ser bloqueados, e que os bloqueadores possam esquecer de desbloqueá-los, o sistema poderia estipular tempos de expiração de bloqueios para determinados grupos de usuários (papéis), e alguns grupos poderiam ter prioridade, de bloqueios e desbloqueios, sobre outros.

O VideoCVS não tem domínio sobre quais e onde foram as operações e modificações feitas em uma árvore de versionamento. A integração entre uma ferramenta de edição e o VideoCVS existe, mas ainda longe de ser considerada uma integração fortemente acoplada. Possivelmente, para prover essa integração

PUC-Rio - Certificação Digital Nº 0420994/CB

satisfatória, seja necessário implementar dentro do VideoCVS uma ferramenta de edição ou então incorporar às ferramentas, o VideoCVS.

Novas funcionalidades poderiam ser introduzidas no protótipo, afim de melhorar a interface gráfica para facilitar o uso do sistema. Funcionalidades como: a possibilidade de cortar uma sub-árvore e colar, esta mesma sub-árvore, em outra posição de uma árvore de versionamento; chat entre os usuários do sistema; a possibilidade de visualizar a hierarquia de derivação das árvores de versionamentos e dos nós, através de grafos; entre outros.

PUC-Rio - Certificação Digital Nº 0420994/CB

Referências

[AlYo02] Aly, S.; Youssef, A. “Synchronization-Sensitive Frame Estimation: Video Quality Enhancement”. Multimedia Tools and Applications, 17, 233-255, 2002.

[Avid06] Avid Xpress DV. http://www.avid.com/products/xpressdv/. Acesso em 2006.

[Bargeron01] Bargeron, D., Gupta, A., Grudin, J., Sanocki, E., Li, F. “Asynchronous Collaboration Around Multimedia and its Application to On-Demand Training”. 34th Hawaii International Conference on System Sciences (HICSS-34), January 3-6, 2001, Maui, Hawaii, Copyright 2001 pelo Institute of Electrical and Electronics Engineers, Inc. (IEEE), 10 pages.

[BGGS99] Bargeron, D., Gupta, A., Grudin, J., e Sanocki, E. "Annotations for Streaming Video on the Web: System Design and Usage Studies". Microsoft Research, Redmond. http://www.research.microsoft.com/research/coet/MRAS/WWW8/paper.htm. 1999

[BrDK97] Brightwell, P.; Dancer, S.; Knee, M. “Flexible Switching and Editing of MPEG-2 Video Bitstreams”. International Broadcast Convention, Amsterdam, Setembro de 1997. Disponível em http://www.bbc.co.uk/rd/pubs/papers/paper_13/paper_13.html.

[ChZZ04] Cheng, G.; Zhou, Y.; Zhao, Y. “Frame Accuracy seamless Splicing of MPEG-2 Video Streams in Compressed Domain”. IEEE Transactions on Consumer Electronics, Vol. 50, No 1, Fevereiro, 2004.

[Coala06] Swiss Federal Institute of Technology (EPFL). COALA (Content-Oriented Audiovisual Library Access) – LogCreator. http://coala.epfl.ch/demos/demosFrameset.html. Acesso em 2006.

[CLMS02] Casares, J., Long, A. C, Myers, B. A., Stevens, S. M., Corbett, A.: Simplifying Video Editing with Silver., Proc. of ACM CHI'2002 (Interactive Poster Abstract) pp. 672-673, 2002.

[CSIRO06] CSIRO. The Continuous Media Web (CMWeb). http://www.cmis.csiro.au/cmweb/. Acesso em 2006.

[CVS06] Concurrent Versions System. Ximbiot http://ximbiot.com/cvs/. Acesso em 2006.

[Ebert06] Ebert, C. Open Source – Version Control. IEEE Software. IEEE Computer Society, 2006.

[EgAA00] Egawa, R.; Alatan, A.; Akansu, A. “Compressed Domain MPEG-2 Video Editing with VBV Requirement”. IEEE ICIP '2000, Vancouver, Canadá, Setembro de 2000. Disponível em http://www.eee.metu.edu.tr/~alatan/PAPER/icip.00.pdf.

PUC-Rio - Certificação Digital Nº 0420994/CB

[FCut06] Final Cut Pro 5. http://www.apple.com/br/finalcutstudio/finalcutpro/. Acesso em 2006.

[FScene07] Forbidden Technologies. FORScene - Web-based video logging, editing, reviewing and publishing tool http://www.forbidden.co.uk/products/scene/. Acesso em 2007.

[GHJV95] Gamma, E., Helm, R., Johnson, R., and Vlissides, J..Design Patterns: Elements of Reusable Object-Oriented software. Addison-Wesley, 1995.

[HeMi02] He, Z.; Mitra, S. “A linear source model and a unified rate control algorithm for DCT video coding”. IEEE Transactions Circuits and System Video Technology, vol . 12, número 11, páginas 970-982, 2002.

[Hitch06] http://www.xecoo.co.jp/muve2000/muveNL.htm. Acesso em 2006

[Holzner05] HOLZNER, S. Ant: The Definitive Guide. O’Reilly Media, Inc., 2nd edição, 2005.

[HYLK03] Hong, S. H.; Yoo, S.; Lee, S.; Kang, H.; Hong, S.Y. “Rate Control of MPEG Video for Consistent Picture Quality“. IEEE Transactions on Broadcasting, Vol. 49, No 1, Março, 2003.

[Hsql06] HSQL Development Group, Hsql Database Engine, http://www.hsqldb.org. Acesso em 2006

[IcMa05] Ichimura, S., Matsushita, Y. “Web-based Video Editing System for Sharing Clips Collected from Multi-users”. Proceedings of the Seventh IEEE International Symposium on Multimedia (ISM’05). 2005

[IMAT06] IBM MPEG-7 Annotation Tool. http://www.alphaworks.ibm.com/tech/videoannex. Acesso em 2006.

[ISO00a] ISO/IEC 13818-1. Information technology -- Generic coding of moving pictures and associated audio information: Systems, 2000.

[ISO00b] ISO/IEC 13818-2. Information technology -- Generic coding of moving pictures and associated audio information: Video, 2000.

[ITUT00a] H.222.0: “Generic coding of moving pictures and associated audio information: systems”. Recomendação ITU-T, Fevereiro de 2000.

[ITUT00b] H.262: “Generic coding of moving pictures and associated audio information: Video”. Recomendação ITU-T, Fevereiro de 2000.

[JUnit07] Ferramenta de Testes Automatizados, JUnit. http://www.junit.org. Acesso em 2007.

[JacORB06] JacORB - The free Java implementation of the OMG's CORBA standard. http://www.jacorb.org. Acesso em 2006.

[Lesg00] Lew, M. S.; Sebe, N.; Gardner, P. C. Video Indexing and Understanding. In: Principles of Visual Information Retrieval, Springer-Verlag, London, 2000.

[MeCh96] Meng, J.; Chang, S. “Buffer Control Techniques for Compressed-Domain Video Editing”. IEEE International Symposium on Circuits and Systems (ISCAS'96), Atlanta, GA, Maio, 1996.

PUC-Rio - Certificação Digital Nº 0420994/CB

[MeCh97] Meng, J.; Chang, S. "CVEPS – A Compressed Video Editing and Parsing System". Proceedings of The Fourth ACM International Conference on Multimedia, Boston, EUA, 1997.

[Minor93] Minor, S.; Magnusson, B. "A Model For Semi-(a) Synchronous Collaborative Editing;. Proceedings of the Third European Conference on Computer Supported Cooperative Work, Kluwer Academic Publishers, 1993."

[MVSS04] Mike Pietraszak, Beny Rubinstein. "Microsoft Visual SourceSafe Roadmap". Maio de 2004.

[Mware06] Mediaware International Pty ltd. Austrália. http://www.mediaware.com.au. Acesso em 2006.

[RCC06] IBM Rational ClearCase. 2006 http:// www-306.ibm.com/software/. Acesso em 2006.

[RCS91] RCS. "RCS: A System for Version Control". Universidade de Purdue, 1991.

[Rico06] Ricoh MovieTool. http://www.ricoh.co.jp/src/multimedia/MovieTool/. Acesso em 2006.

[Santos06] Santos, M N.. Cerqueira, R. "GridFS: Um Servidor de Arquivos para Computação em Grade". Dissertação de Mestrado da PUC-Rio, Março de 2006.

[ScHK04] Schroeter, R., Hunter, J., Kosovic, D. "FilmEd – Collaborative Video Indexing, Annotation and Discussion Tools Over Broadband Networks" DSTC, The University of Queensland. 2004

[SiKS99] Siberschatz, A., Korth, H. F., e Sudarsham, S. Livro de Sistema de Banco de Dados. Capítulo 14 – Controle de Concorrência, páginas 487-490, 1999.

[Strachan96] Strachan, D. Video Compression. In: SMPTE Tutorial 105:68. February 1996.

[SubV06] "Version Control With SubVersion". Livro de Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.http://svnbook.red-bean.com/en/1.0/index.html Acesso em 2006.

[SoCh03] Song, B.; Chun, K. "A Virtual Frame Rate Control Algorithm for Efficient MPEG-2 Video Encoding". IEEE Transaction on Consumer Electronics, Vol. 49, No 2, Maio, 2003.

[TaRa97] Talreja, K.; Rangan, P. "Editing Techniques for MPEG Multiplexed Streams". IEEE Multimedia Computing and Systems, Canada, 1997.

[Tekalp00] Tekalp, A. M. "Video Segmentation". In: Bovik, A. (ed.), Handbook of Image and Video Processing, Academic Press, San Diego, p. 383-399, 2000.

[Tobin99] Tobin, B.; Examination of MPEG Macroblock Types to Find Shot Cuts. Technical Report, Dublin City University. Dublin, April, 1999.

[Vasconcelos05] Vasconcelos, C. N. Segmentação de vídeo no domínio comprimido baseado na história da compactação, Dissertação de Mestrado da PUC-Rio, Março de 2005.

PUC-Rio - Certificação Digital Nº 0420994/CB

[VFSC06] Vasconcelos, C. N.; Feijó, B.; Szwarcman, D. M.; Costa, M. "Shot Segmentation based on the encoder signature", In: MMM 2006, Beijing, Proc. 12th Int. Multimedia Modeling Conf., IEEE Press, p. 177-184, 2006.

[Videto06] Zentrum fuer Graphische Datenverarbeitung e.V. (ZGDV).VIDETO - Video Description Tool. http://www.rostock.zgdv.de/ZGDV/Abteilungen/zr2/Produkte/

videto/index_htmlen. Acesso em 2006.

[WCVS01] Don Harper. "WinCVS 1.3 User's Guida". November 1, 2001.

[WiLe97] Will, U. K; Legged, J. J. "Hyperform: A Hypermedia System Development Environment". CAN Transactions on Information Systems, Janeiro de 1997.

[Zhang93] Zhang, H. J.; Kakanhalli, A.; Smoliar, W. Automatic Partition of Full-Motion Video, Multimedia Systems 1. Vol. 1,no.1, pp. 10-28. 1993.

[Zhang94] Zhang, H. J.; Kakanhalli, A.; Smoliar, W. Content –based Video Indexing and Retrival, 1994.

[Zhang99] Zhang, H. J. Content-based Video Browsing and Retrieval. Handbook of Multimedia Computing. Boca Raton: CRC Press, 1999.

[Zhang03] Zhang, H. Content-based video analysis, retrieval and browsing. Multimedia Information Retrieval and Management - Technological Fundamentals and Applications. Springer (Ed.), 2003. Series: Signals and Communication Technology. 2003. Chapter 2, XVII, p. 476

PUC-Rio - Certificação Digital Nº 0420994/CB

Anexo A

IDL CORBA

```
#ifndef VIDEOCVS_IDL
#define VIDEOCVS_IDL


module src {

  module generate {

    interface VideoDescriptor;
    interface VersionControl;
    interface VideoParentChild;
    interface VideoNode;
    interface VideoTree;
    interface ComponentManager;
    interface User;
    interface VideoTreeFactory;
    interface VideoNodeFactory;
    interface UserFactory;
    interface VideoNodeHierarchy;


    exception FileTransferException {};

    exception PermissionDeniedException {
         string reason;
    };

    exception VideoNodeAlreadyExistException {
         string msg;
     };
     exception VideoNodeDoesNotExistException {
          string msg;
     };

    exception VideoTreeAlreadyExistException {
     string msg;
    };
    exception VideoTreeDoesNotExistException {
     string msg;
    };

    exception UserAlreadyExistException {
     string msg;
    };
    exception UserDoesNotExistException {
     string msg;
    };
```

PUC-Rio - Certificação Digital Nº 0420994/CB

```
    typedef sequence<octet> OctetSeq;
    typedef sequence<VideoNode> VideoNodeSeq;
    typedef sequence<VideoParentChild> VideoParentChildSeq;
    typedef sequence<VideoTree> VideoTreeSeq;
     typedef sequence<VideoNodeHierarchy> VideoNodeHierarchySeq;

    interface User {
       long getId();
       void setId(in long id);
       string getName();
       void   setName(in string value);
       string getOccupation();
       void   setOccupation(in string value);
       string getDesc();
       void   setDesc(in string value);
       string getPassword();
       void   setPassword(in string value);

    };

     interface ComponentManager {

       void  addAscendentsLock(in VideoTree tree, in VideoNode
node, in User user)
                        raises (PermissionDeniedException);

       void        addExclusiveLock(in VideoNode node, in User
user)
                        raises (PermissionDeniedException);

        boolean    areOtherUsersEditing(in VideoTree tree, in User
user);

        void       commit(in VideoTree tree, in User user)
                        raises (PermissionDeniedException);

        boolean         existVideoNode(in VideoNode node)
                        raises (VideoNodeDoesNotExistException);

        VideoNode  getVideoNodeFromID(in long id)
                        raises (VideoNodeDoesNotExistException);

        VideoNodeSeq    getVideoNodeFromName(in string name)
                        raises (VideoNodeDoesNotExistException);


        VideoTree  getVideoTreeFromID(in long id)
                        raises (VideoTreeDoesNotExistException);

        VideoTree  getVideoTreeFromName(in string name)
                        raises (VideoTreeDoesNotExistException);


        boolean         hasPermission(in VideoTree tree, in
VideoNode node, in User user)
                        raises (PermissionDeniedException);

        VideoNodeSeq getPreviousVersion(in VideoNode node);

        VideoNodeSeq getNextVersion(in VideoNode node);
```

PUC-Rio - Certificação Digital Nº 0420994/CB

```
        VideoTreeSeq getVideoTreeFromNameVersionUser(in string
name, in string version, in User user)
                        raises (VideoTreeDoesNotExistException);

       boolean     hasAscendentExclusiveLock(in VideoTree tree, in
VideoNode node, in User user);

        boolean    lock(in VideoTree tree, in VideoNode node, in
User user)
                        raises(PermissionDeniedException);

       boolean     unlock(in VideoTree tree, in VideoNode node, in
User user)
                        raises (PermissionDeniedException);
    };

    interface VideoDescriptor {
       string getName();
       void        setName (in string name);

       void        setNumFrames(in long numframes);
       long       getNumFrames();

       string   getDesc();
       void     setDesc (in string value);

       string getFilename();
       void   setFilename(in string name, in long idVersion);

       string   getPath();
       void     setPath (in string value);

       string   getExtension();
       void     setExtension (in string value);

    };


    interface VersionControl {
        long getId();
        void setId(in long id);

        string getName();
        void setName(in string name);

        string getType();
        void setType(in string type);

        string getDesc();
        void setDesc(in string desc);

        VersionControl getCopy();

    };

    interface VideoNode {

       long getId();
       void setId(in long id);
       void updateId();
```

PUC-Rio - Certificação Digital Nº 0420994/CB

```
   VideoDescriptor getVideoDescriptor();
   void setVideoDescriptor(in VideoDescriptor vd);

   void        setChildren(in VideoNodeSeq nodes);
   VideoNodeSeq getChildren();
   long    getNumChildren();

   void insertChildWithPos(in VideoNode node, in long pos);
   void insertChild(in VideoNode node);

   boolean removeChildFromPos(in long pos);
   boolean removeChild(in VideoNode node);

   void removeChildren();
   void remove();

   boolean isLoaded();
   void setLoaded(in boolean flag);
   VideoNode getCopy();


   boolean isEdited();
   void setEdited(in boolean edited);

};

interface VideoNodeHierarchy {
 VideoNode getNextNode();
 void setNextNode(in VideoNode node);

 VideoNode getPreviousNode();
 void setPreviousNode(in VideoNode node);

 User getUser();
 void setUser(in User user);

};

interface VideoParentChild {
   VideoNode getChild();
   void setChild(in VideoNode node);

   long getChildPos();
   void setChildPos(in long pos);

   VideoNode getParent();
   void setParent(in VideoNode node);
};


interface VideoTree {

   long  getId();
   void setId(in long id);
   void updateId();

   string getName();
   void setName(in string name);

   string getFilename();
   //void setFilename(in string filename);
```

PUC-Rio - Certificação Digital Nº 0420994/CB

```
        //void createDstFilename(in string value);

        string getPath();
        void setPath(in string path);

        VideoTree getParent();
        void setParent(in VideoTree tree);

        VersionControl getVersion();
        void setVersion(in VersionControl version);
        long getVersionID();

        VideoNode getRootNode();
        void setRootNode(in VideoNode root);

        void removeTree();

        void remove(in VideoNode node);

        void insertNode(in VideoNode parent, in VideoNode child, in
long childPos);
        boolean removeVideoNode (in VideoNode node);
        void updateNode(in VideoNode nodeOld, in VideoNode
nodeNew);

        VideoParentChildSeq getParentChild();
        void loadParentChild(in VideoNode node);
        void removeParentChild(in VideoNode node);

        VideoNode getVideoNodeFromID(in long id);
        VideoNode getVideoNodeFromName(in string name);
        boolean hasNode(in VideoNode node);
        VideoNodeSeq getAllNodes();
        VideoNodeSeq getLeafNodes();
        VideoTree getCopy();
        boolean isEmpty();
        boolean isVideoTreeConsistent();
        string traverseTree();
        VideoNodeSeq getAscendents(in VideoNode node);

        void insertHierarchy(in VideoNode node, in VideoNode
nodeDesc, in User user);
        VideoNodeHierarchySeq getVideoNodeHierarchy(in User user);
        void removeHierarchies(in User user);
    };


    interface VideoTreeFactory {
        VideoTree create(in long id, in string name, in string
filename,
             in string path, in string versionName, in string
versionType,
             in VideoNode root, in VideoTree parent, in User user)
               raises (VideoTreeAlreadyExistException,
PermissionDeniedException);
        VideoTree getVideoTreeFromID(in long id)
               raises (VideoTreeDoesNotExistException);
        VideoTree getVideoTreeFromName(in string name)
               raises (VideoTreeDoesNotExistException);

        ComponentManager getComponentManager();
```

PUC-Rio - Certificação Digital Nº 0420994/CB

```
        void sendFileToServer(in string filenameSrc, in string
filenameDst)
              raises (VideoTreeAlreadyExistException,
FileTransferException);
        void getFileFromServer(in string filename)
              raises (VideoTreeAlreadyExistException,
FileTransferException);
    };

    interface VideoNodeFactory {
       VideoNode create(in long id, in string name, in string
path, in long numFrames, in string desc, in User user)
              raises (VideoNodeAlreadyExistException);
       VideoNode getVideoNodeFromID(in long id)
              raises (VideoNodeDoesNotExistException);
       VideoNodeSeq getVideoNodeFromName(in string name)
              raises (VideoNodeDoesNotExistException);

       ComponentManager getComponentManager();

        void sendFileToServer(in string filename)
              raises (VideoNodeAlreadyExistException,
FileTransferException);
        void getFileFromServer(in string filename)
              raises (VideoNodeAlreadyExistException,
FileTransferException);
    };

    interface UserFactory {
       User createUser(in long id, in string name,
             in string pwd, in string occupation, in string desc)
              raises (UserAlreadyExistException);
       User getUserFromID(in long id)
              raises (UserDoesNotExistException);
       User getUserFromName(in string name)
              raises (UserDoesNotExistException);
    };


  };//generate

}; //src

#endif
```

PUC-Rio - Certificação Digital Nº 0420994/CB

// Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

[Baixar livros de Literatura]
[Baixar livros de Literatura de Cordel]
[Baixar livros de Literatura Infantil]
[Baixar livros de Matemática]
[Baixar livros de Medicina]
[Baixar livros de Medicina Veterinária]
[Baixar livros de Meio Ambiente]
[Baixar livros de Meteorologia]
[Baixar Monografias e TCC]
[Baixar livros Multidisciplinar]
[Baixar livros de Música]
[Baixar livros de Psicologia]
[Baixar livros de Química]
[Baixar livros de Saúde Coletiva]
[Baixar livros de Serviço Social]
[Baixar livros de Sociologia]
[Baixar livros de Teologia]
[Baixar livros de Trabalho]
[Baixar livros de Turismo]