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.

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].

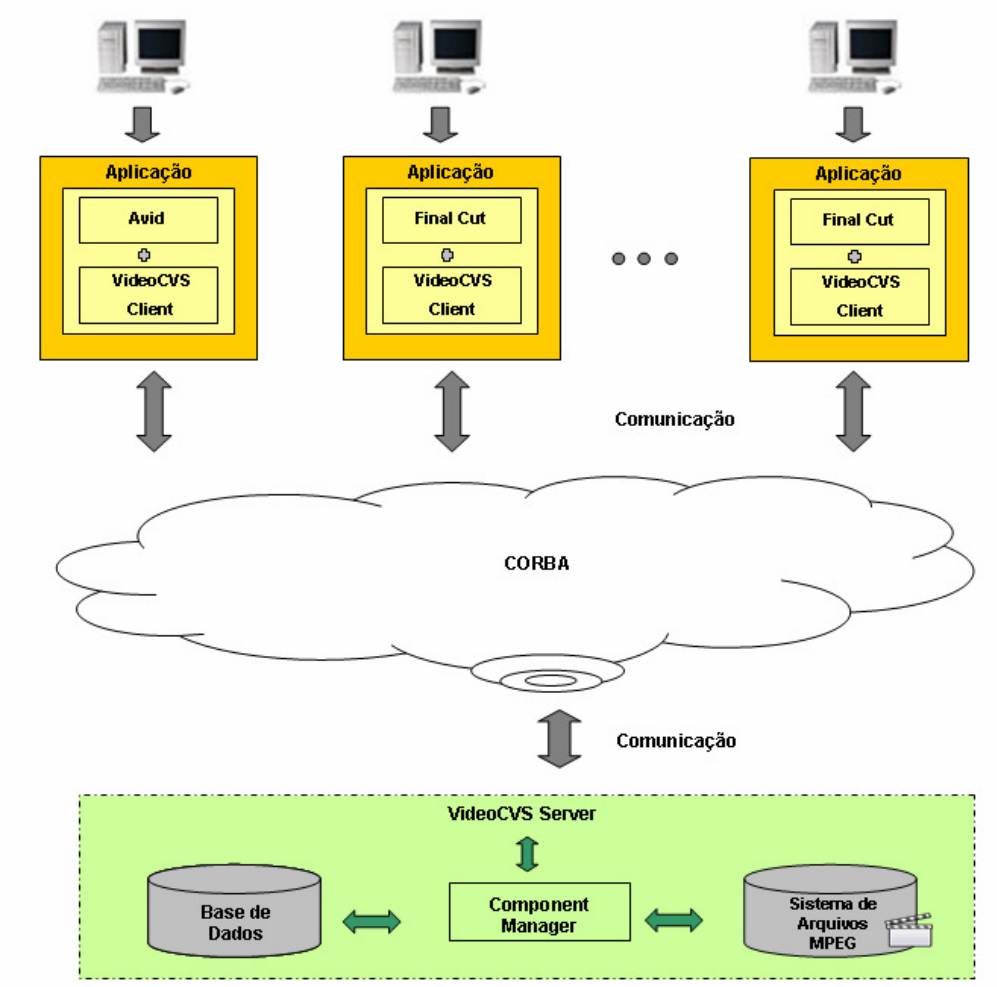

Figura 30 – Arquitetura do *VideoCVS*

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.

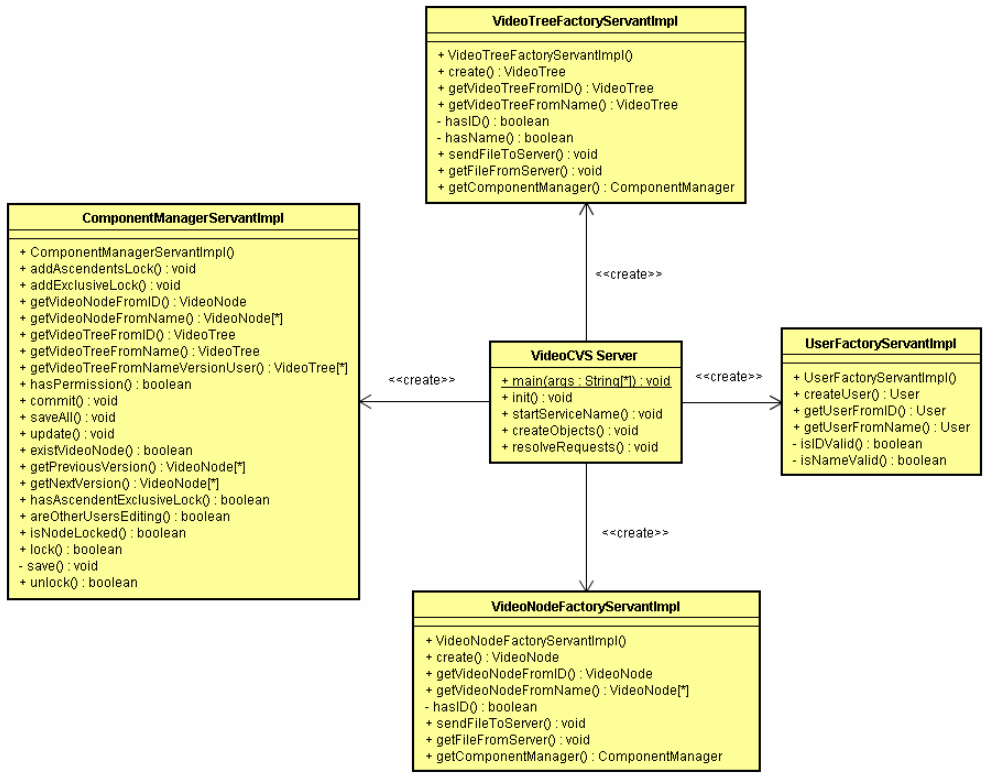

Figura 31 – Diagrama de classes do *VideoCVS Server*

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.

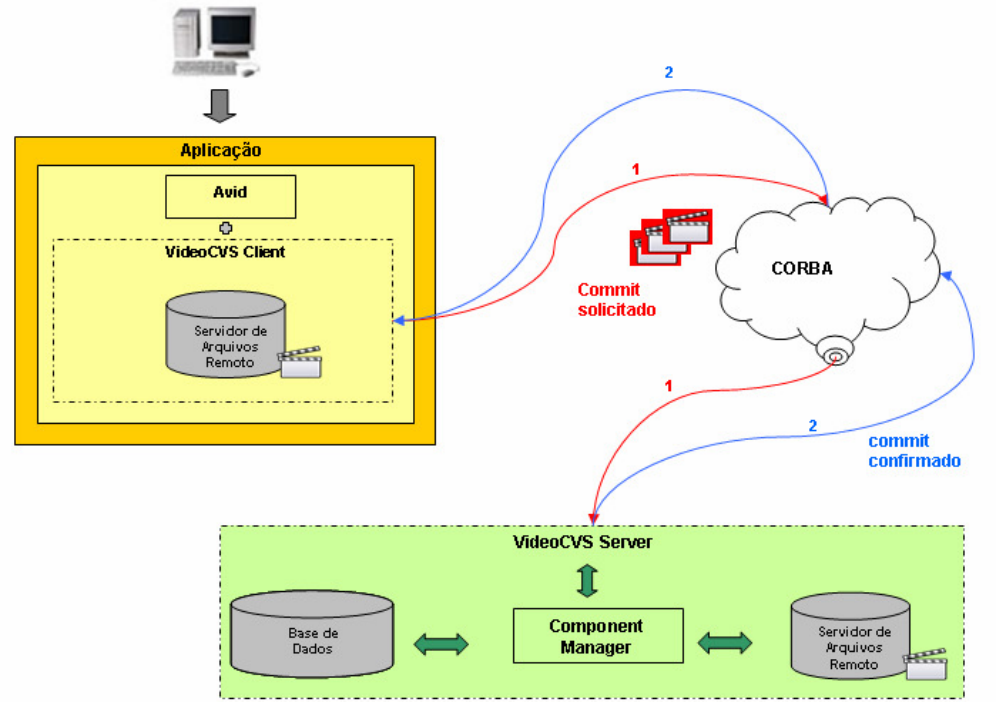

Figura 32 – Exemplo de commit na arquitetura do *VideoCVS*

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

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

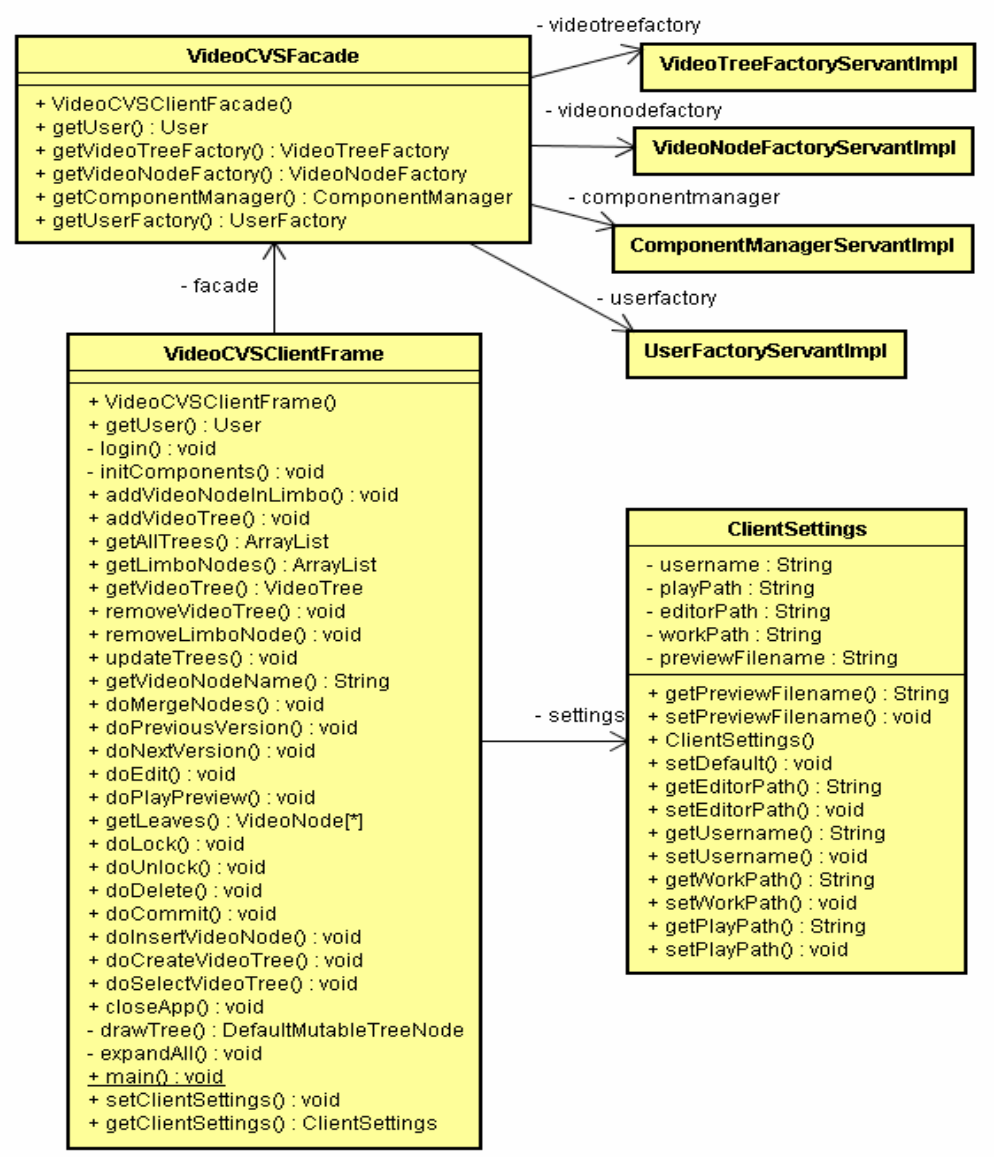

Figura 33 – Diagrama de classes do *VideoCVS Client*

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].

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*.

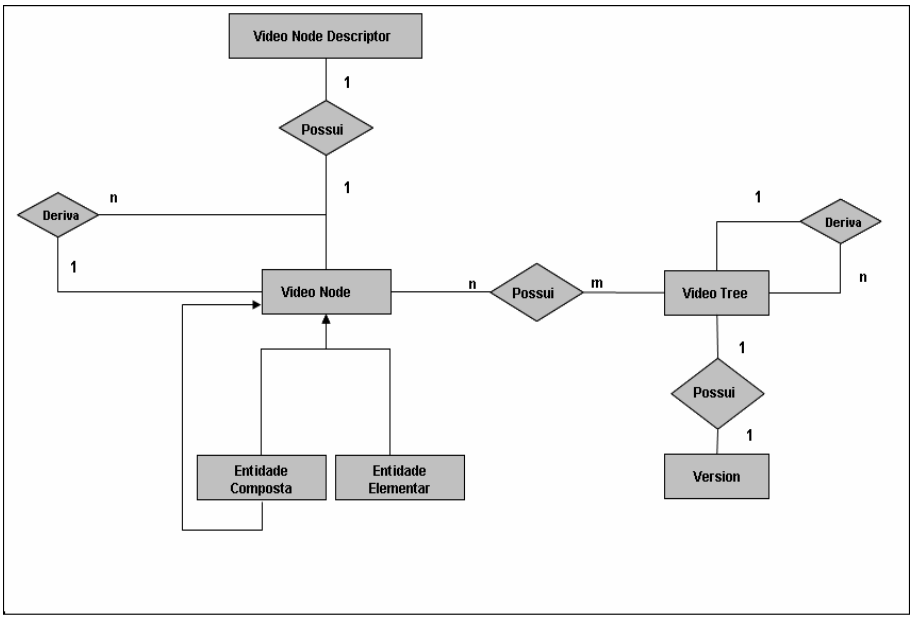

Figura 34 – Modelo ER do sistema *VideoCVS*

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

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.

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.

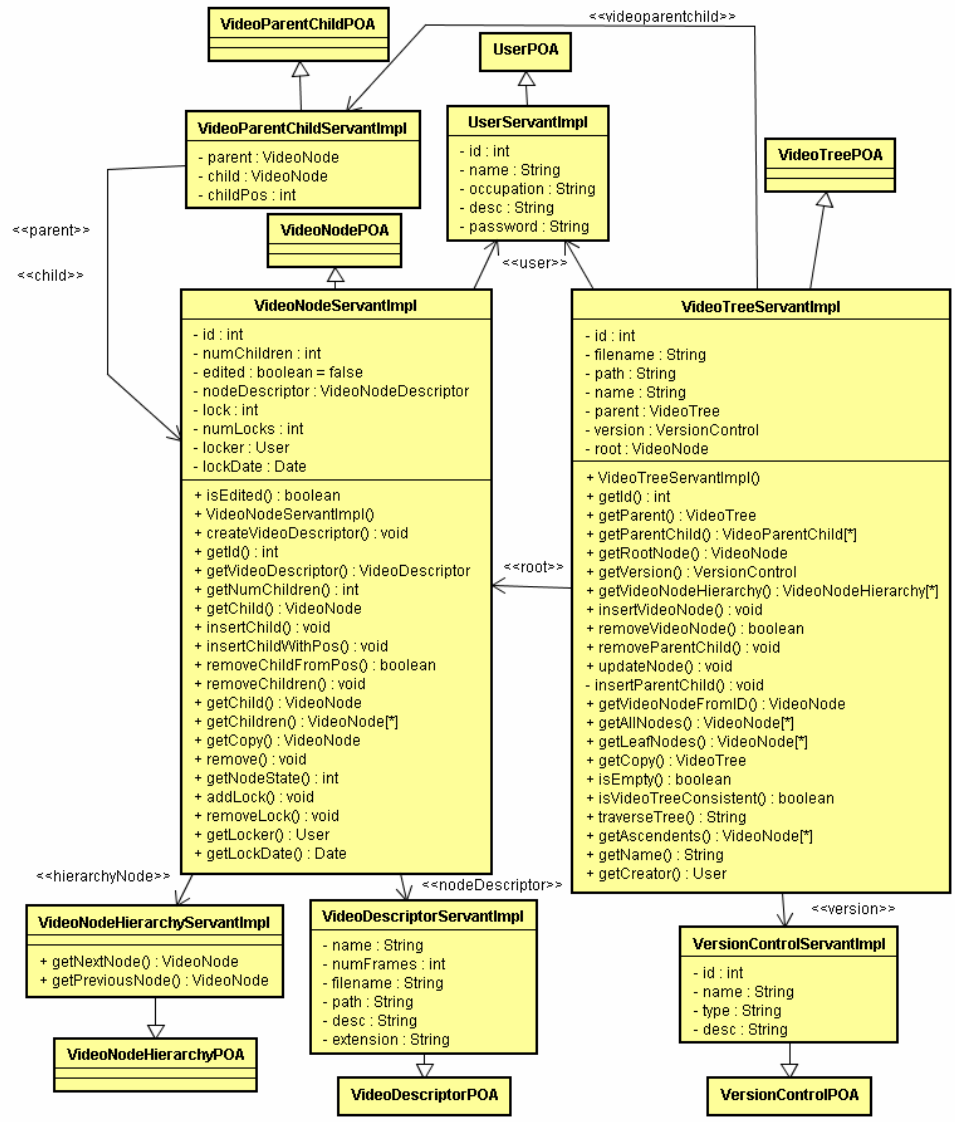

Figura 35 – Diagrama de classes dos objetos remotos do modelo ER

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.

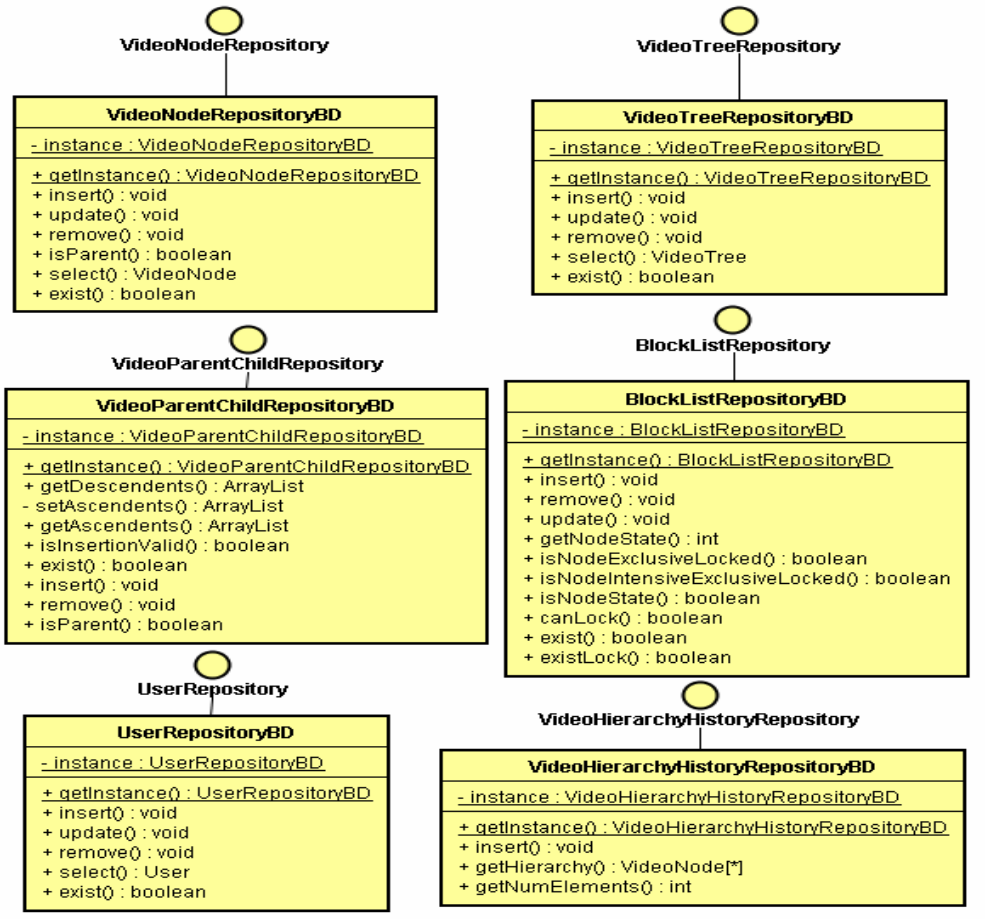

Figura 36 – Diagrama de classes dos objetos remotos do repositório

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.

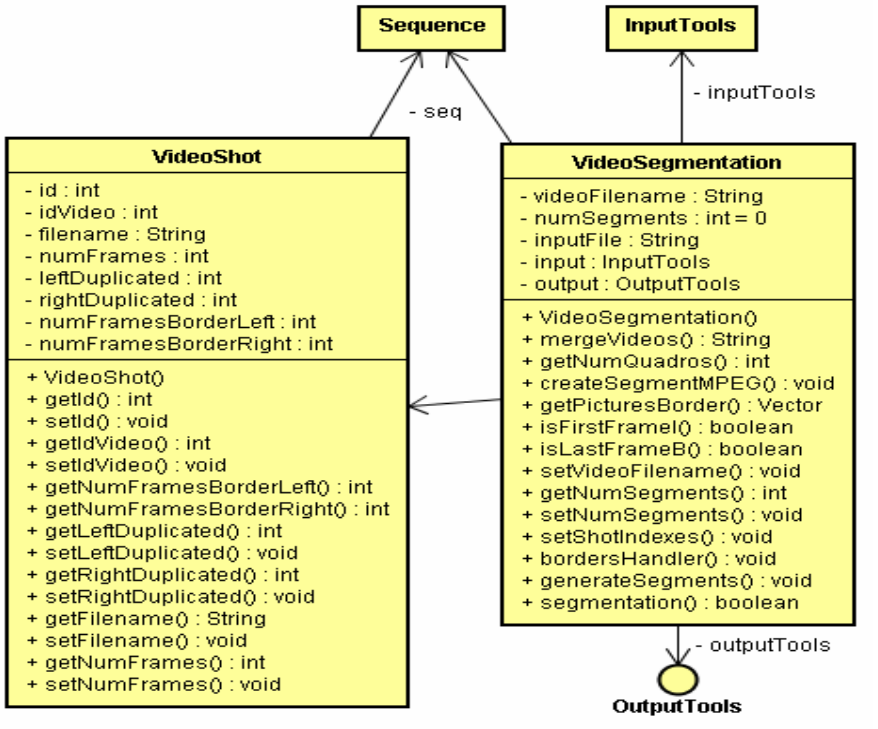

Figura 37 – Diagrama de classes do mecanismo de segmentação

[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].

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.

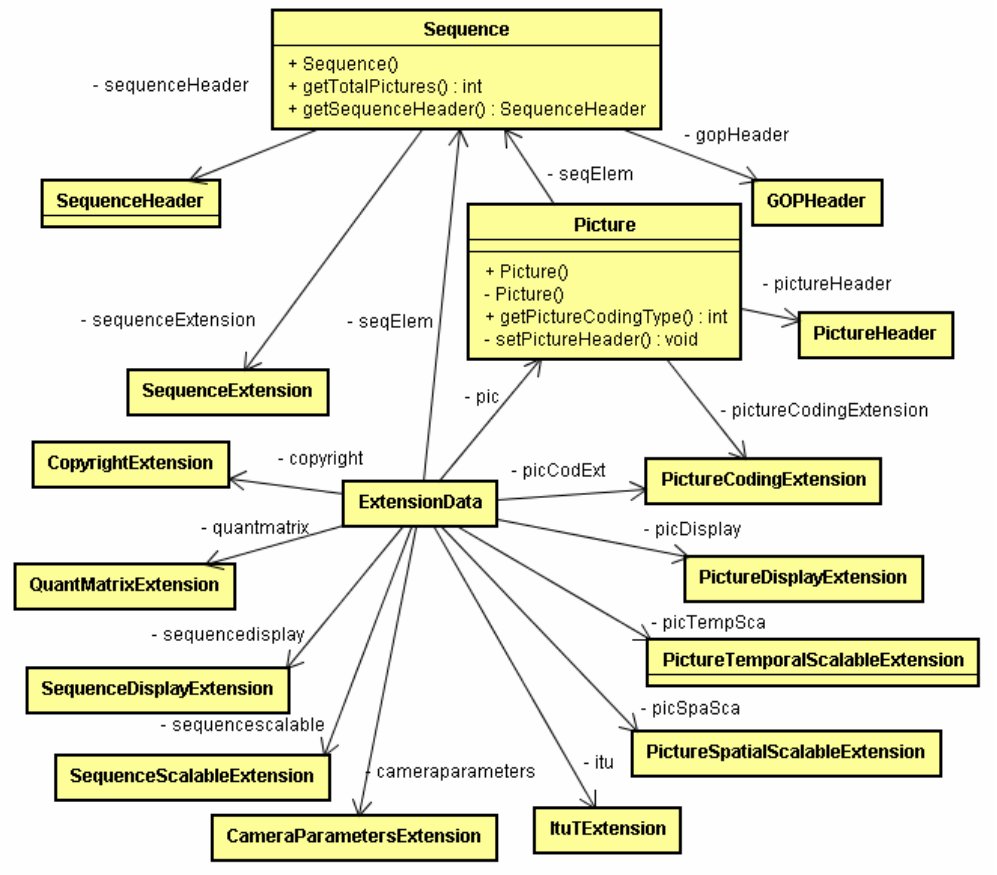

Figura 38 – Diagrama de classes da estrutura do MPEG-2 de vídeo

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.

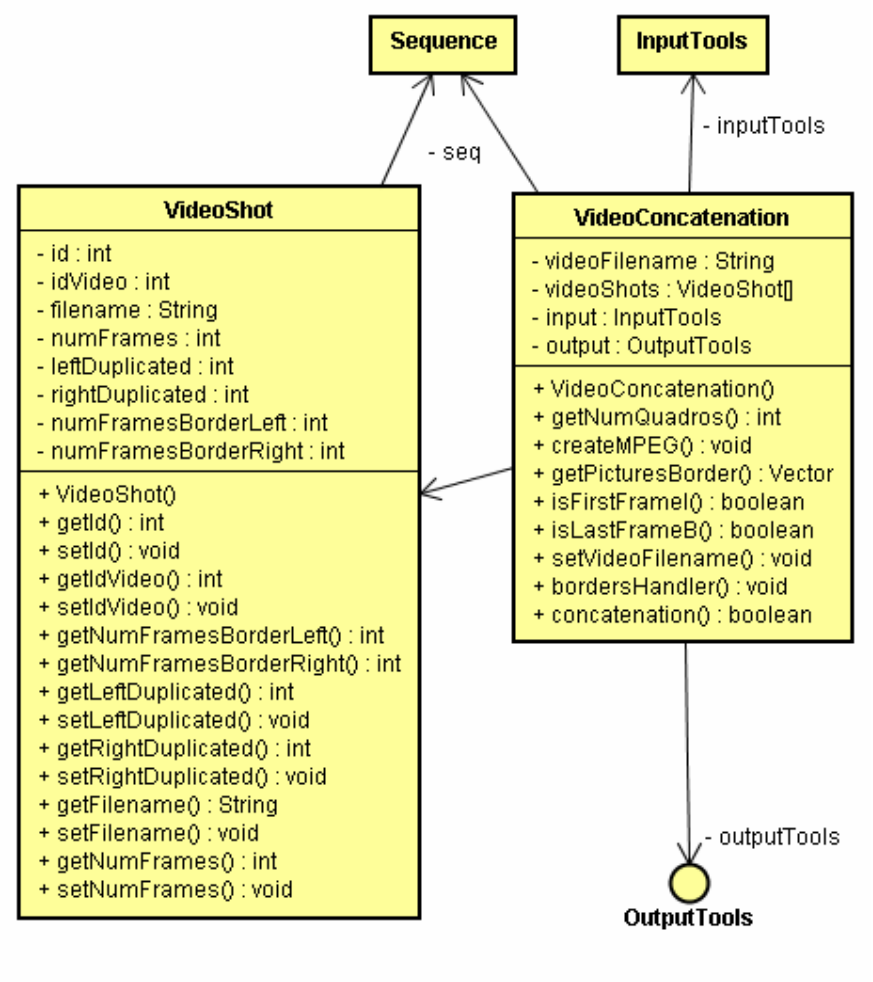

Figura 39 – Diagrama de classes do mecanismo de remontagem

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.

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 c*onsole*, 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.

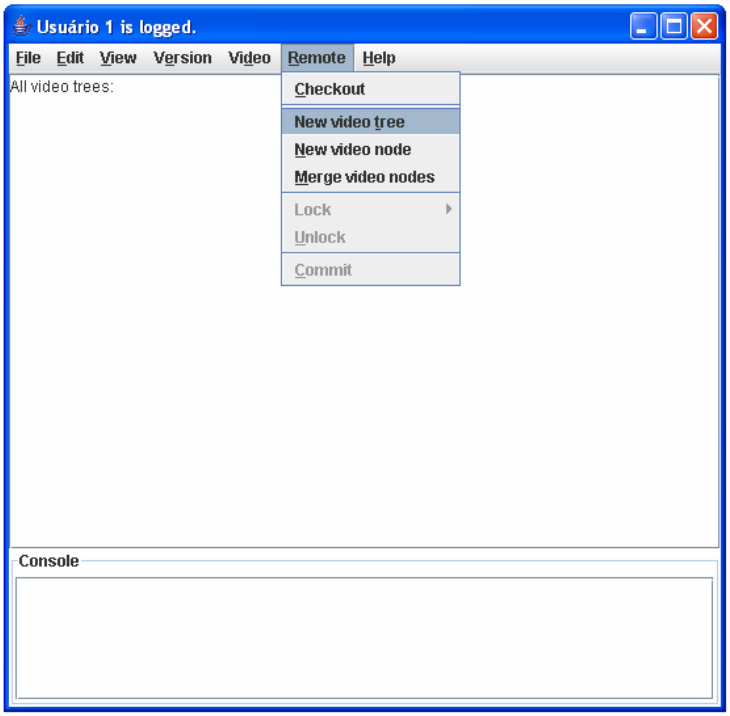

Figura 40 – Tela inicial do VideoCVS Client

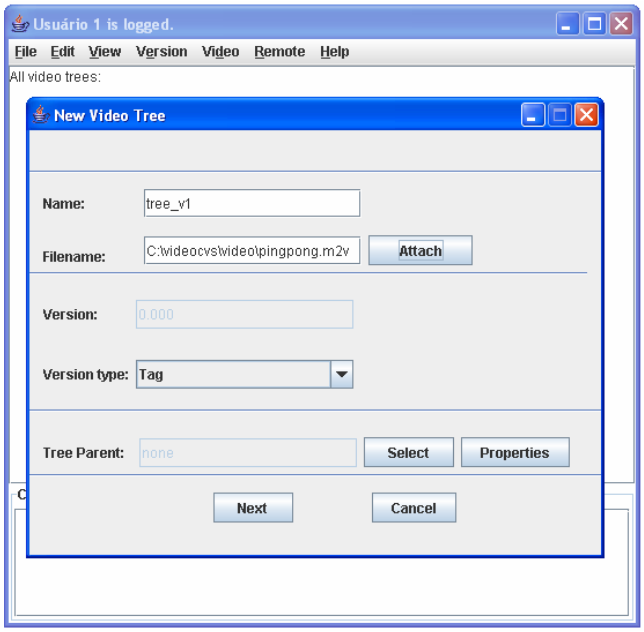

Figura 41 – Primeira tela da criação da árvore de versionamento

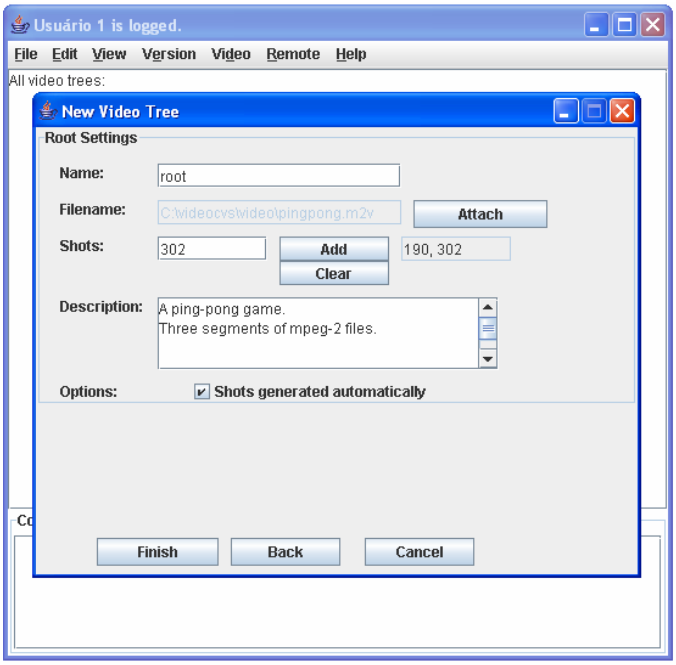

Figura 42 – Segunda tela da criação da árvore de versionamento

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.

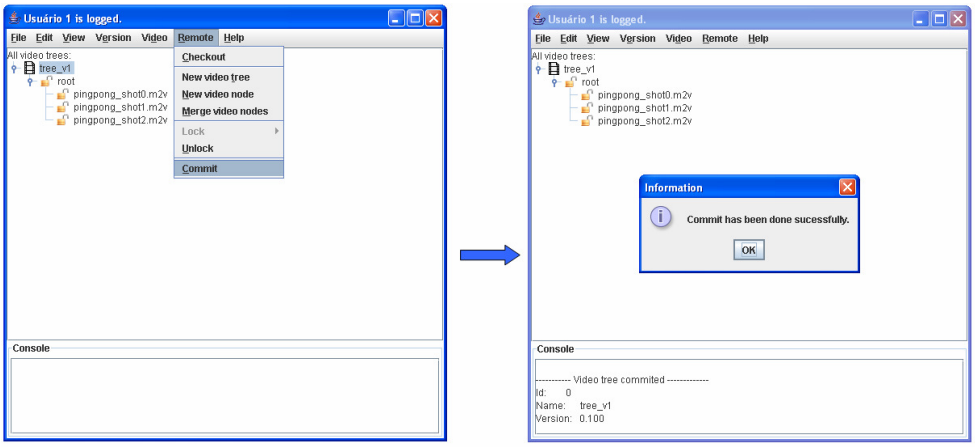

Figura 44 – Commit da árvore de versionamento pelo usuário 1

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.

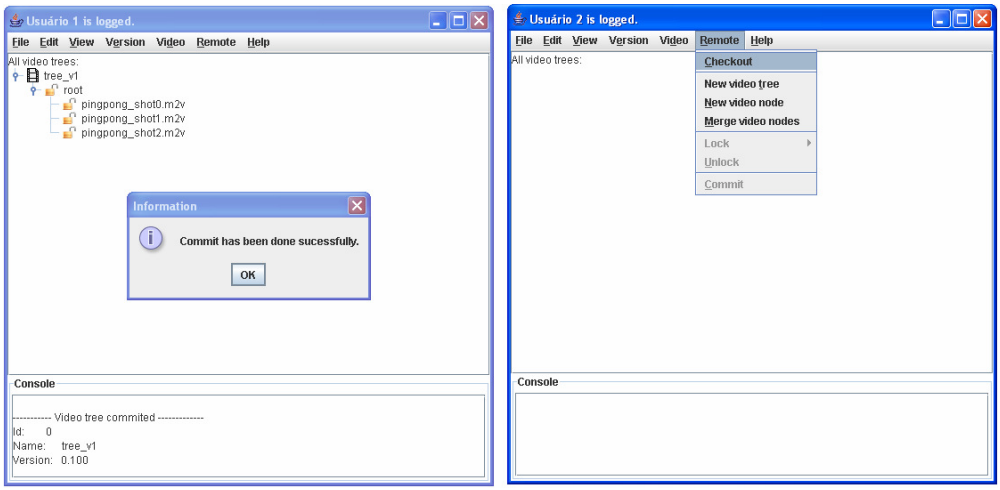

Figura 45 – Operação de *checkout* de uma árvore de versionamento

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.

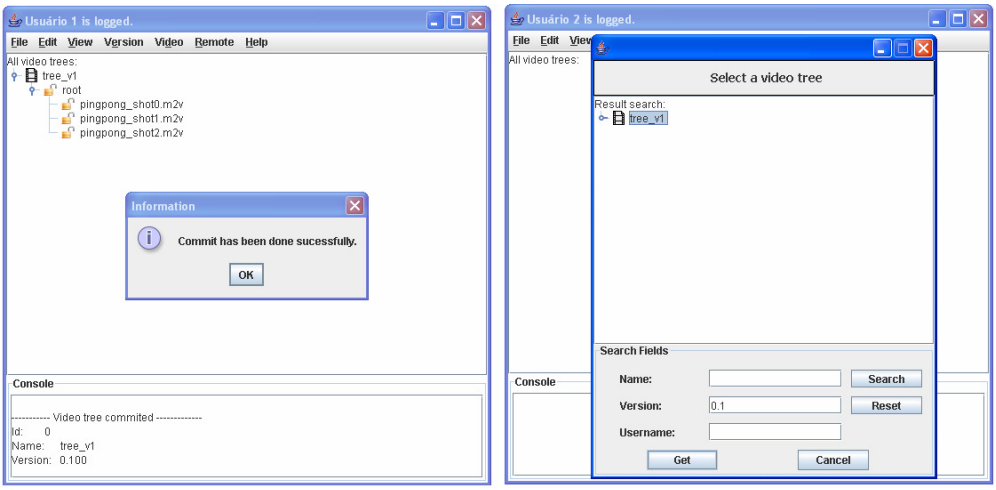

Figura 46 – Resultado da busca do *checkout*

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.

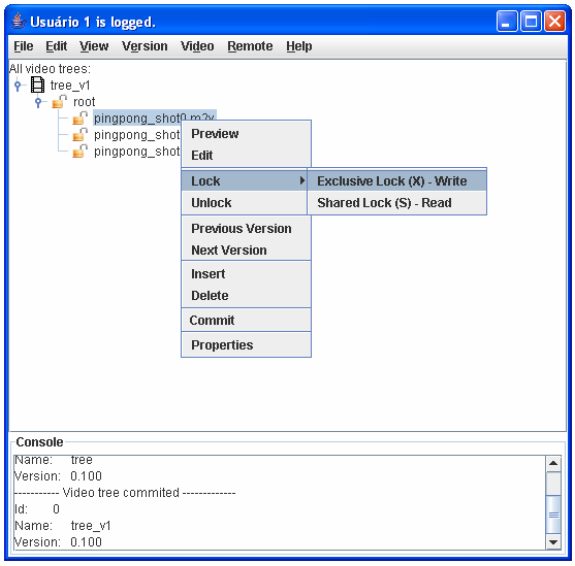

Figura 47 – Bloqueio no modo exclusivo permitido ao usuário 1

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.

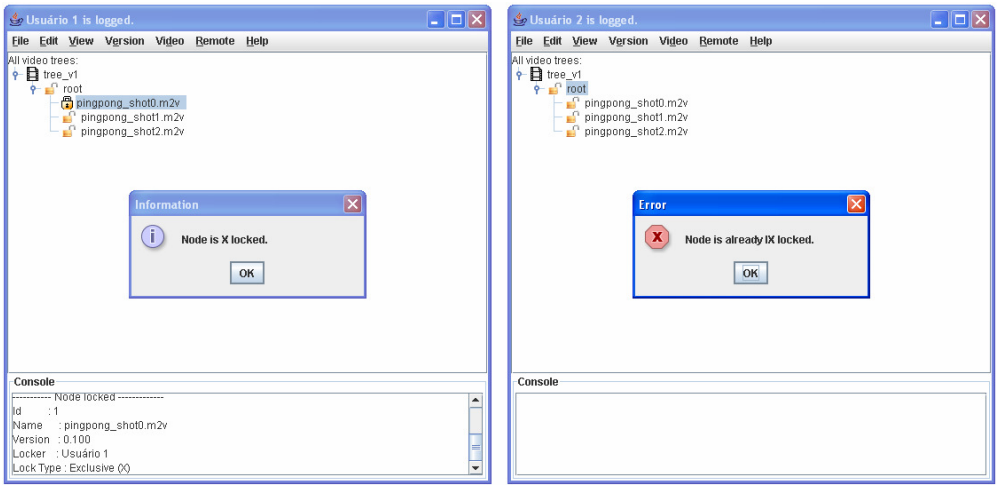

Figura 48 – Bloqueio no modo exclusivo negado ao usuário 2

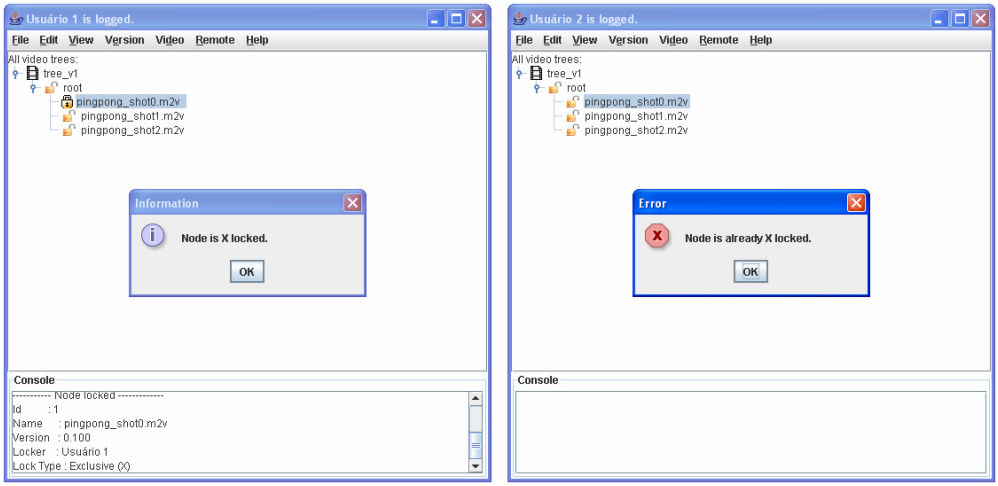

Figura 49 – Bloqueio no modo compartilhado negado ao usuário 2

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*.

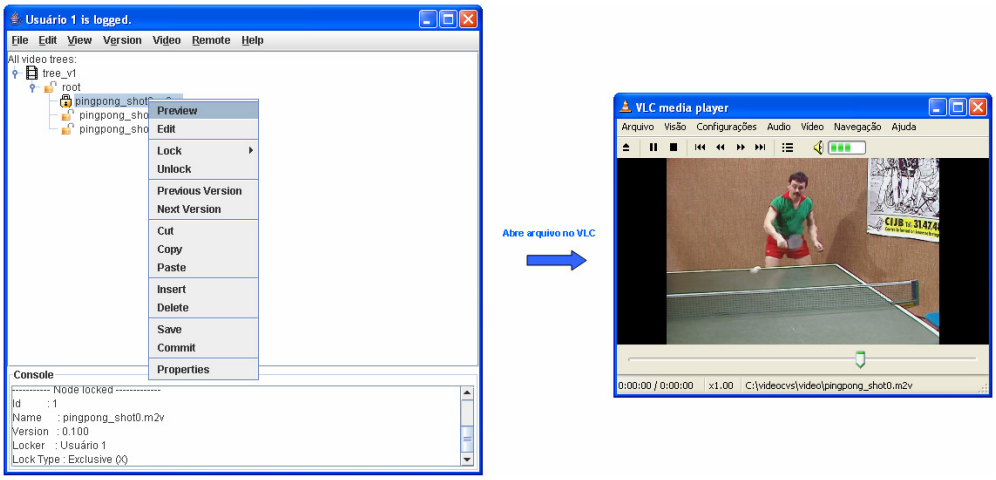

Figura 50 – Clicando na opção Preview para visualizar o vídeo

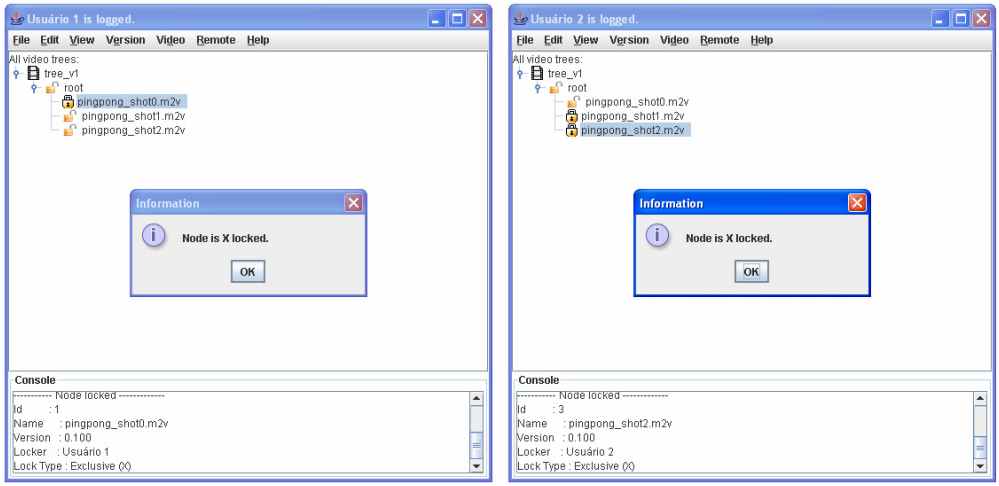

Figura 51 – Bloqueios no modo exclusivo permitido ao usuário 2

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.

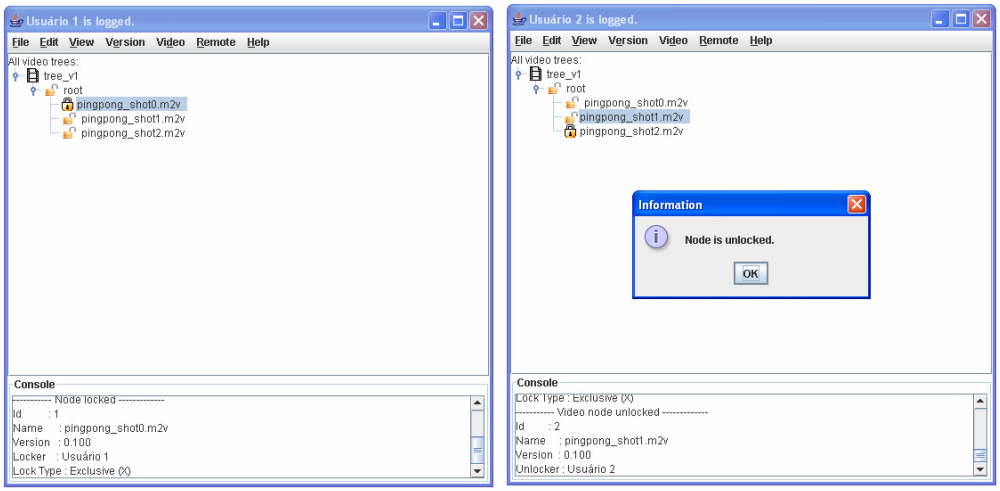

Figura 52 – Desbloqueio de um nó da árvore de versionamento pelo usuário 2

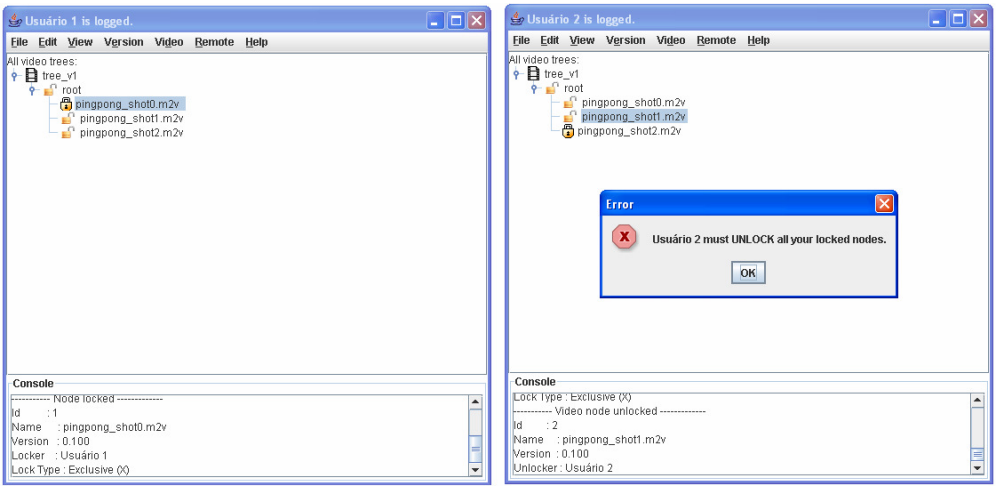

Figura 53 – Permissão negada para bloquear na fase de encolhimento

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.

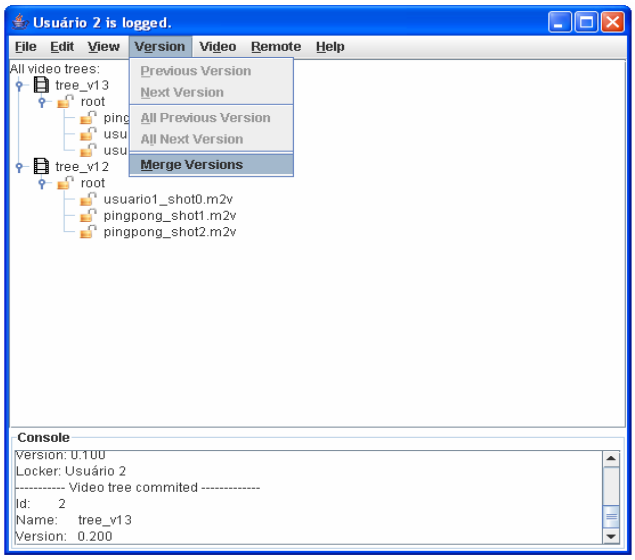

Figura 54 – *Merge* entre versões

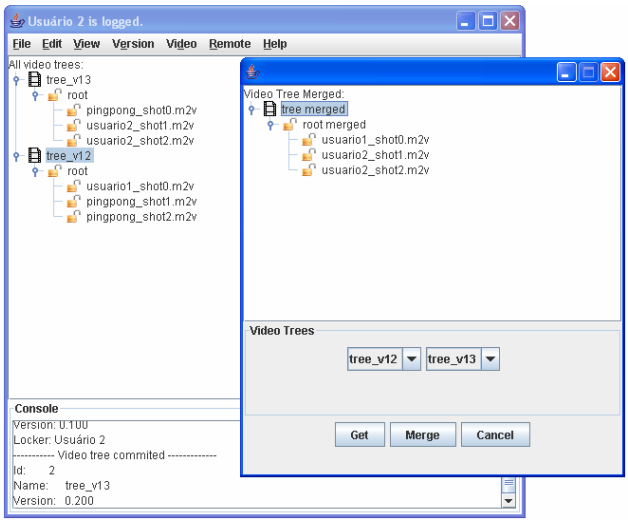

Figura 55 – Fusão entre as árvores de versionamento *tree_v12* e *tree_v13*

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

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

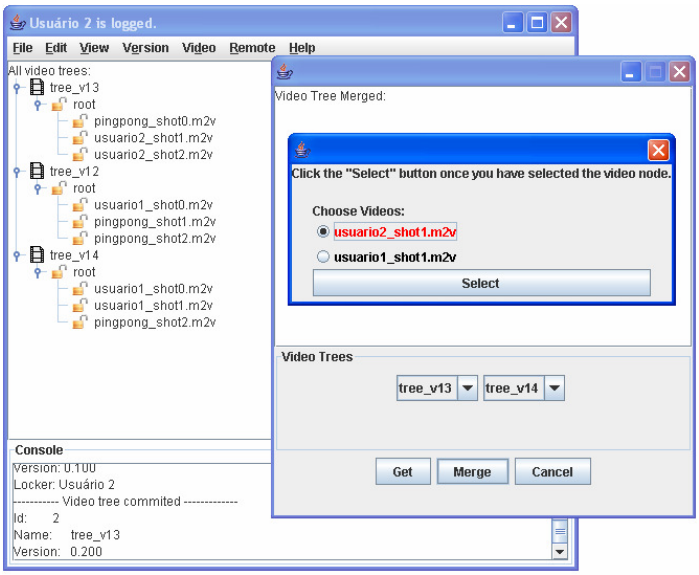

Figura 56 – Intervenção manual do usuário na fusão entre *tree_v13* e *tree_v14*

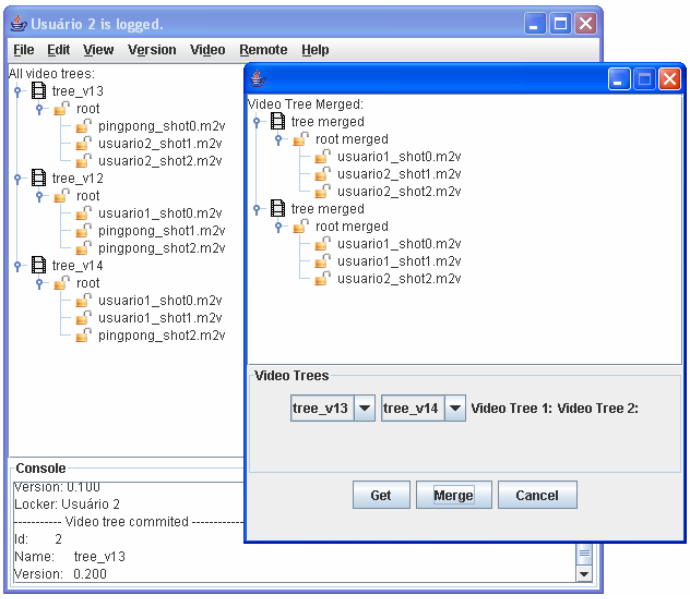

Figura 57 – Árvores de versionamento resultantes após intervenção do usuário

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.

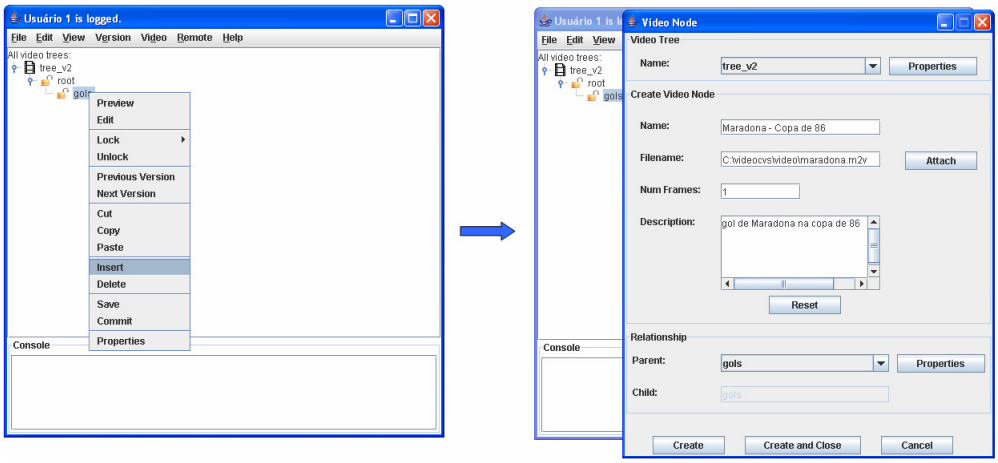

Figura 58 – Operação de inserção de nós na árvore de versionamento

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.

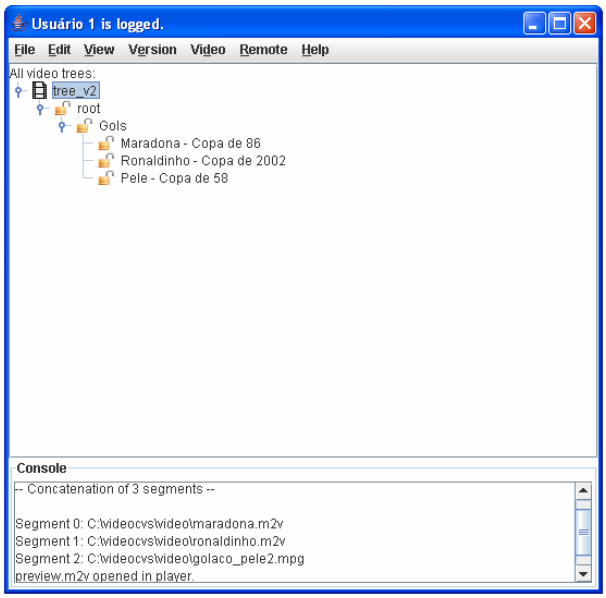

Figura 59 – Árvore de versionamento *tree_v2* para remontagem

Figura 60 – *Preview* da remontagem da tree_v2