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.

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

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>>>**.

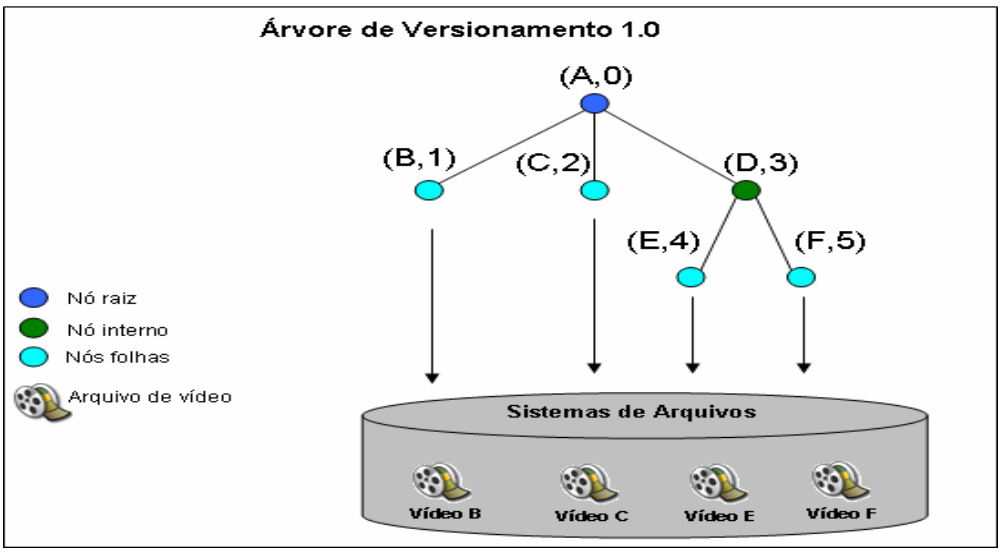

Figura 10 – Exemplo de uma árvore de versionamento de vídeo

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) $\alpha = < A <B> <C> < D <E> <F> > >$

(Eq. 12) $\beta = < A' <B'> <C> < D' <E'> <F> > >$

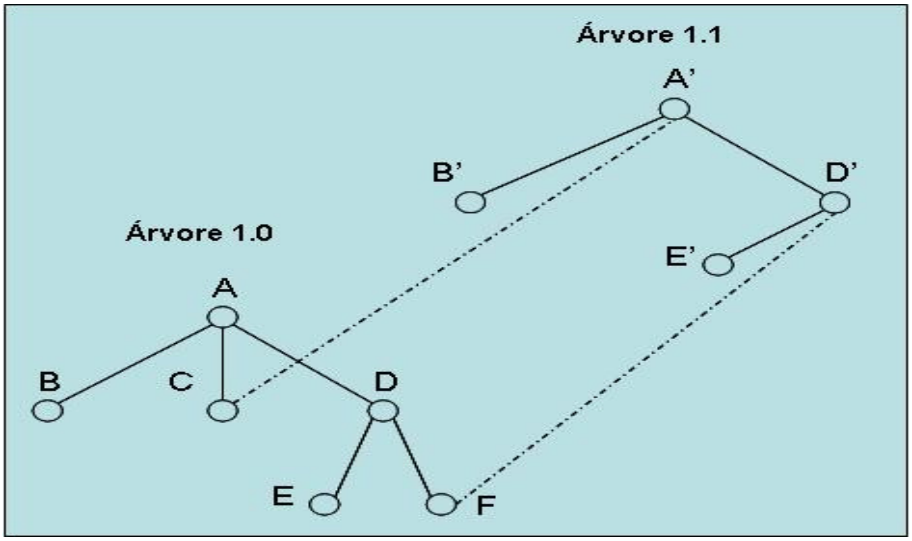

Figura 11 – Compartilhamento de nós entre árvores de versionamento

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.

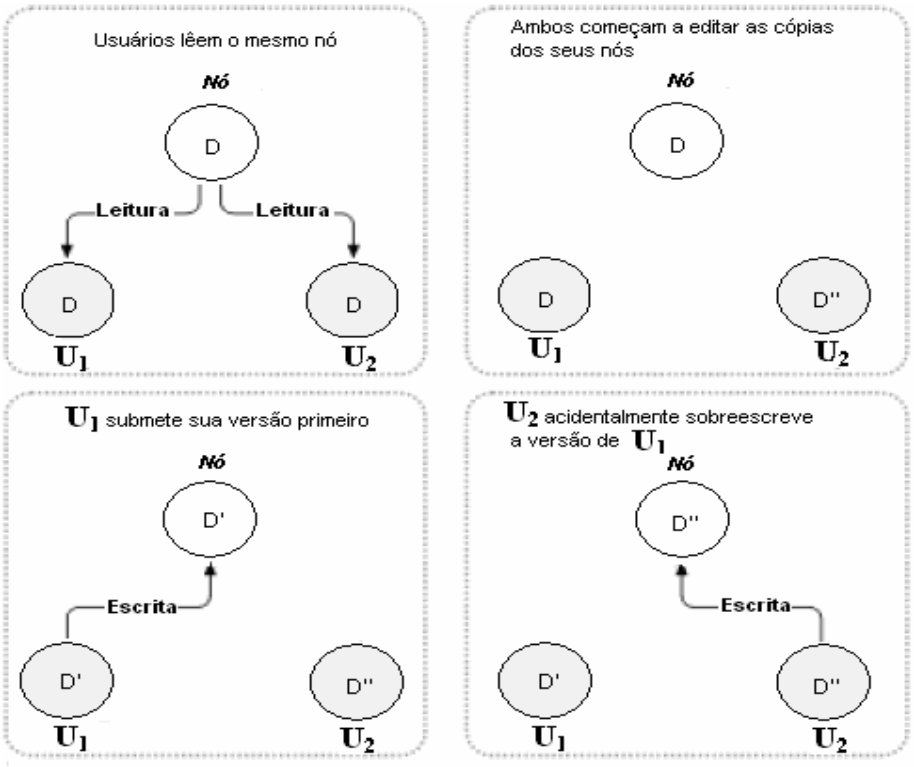

Figura 12 – Problema do acesso simultâneo de um nó.

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 (*commit*) 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.

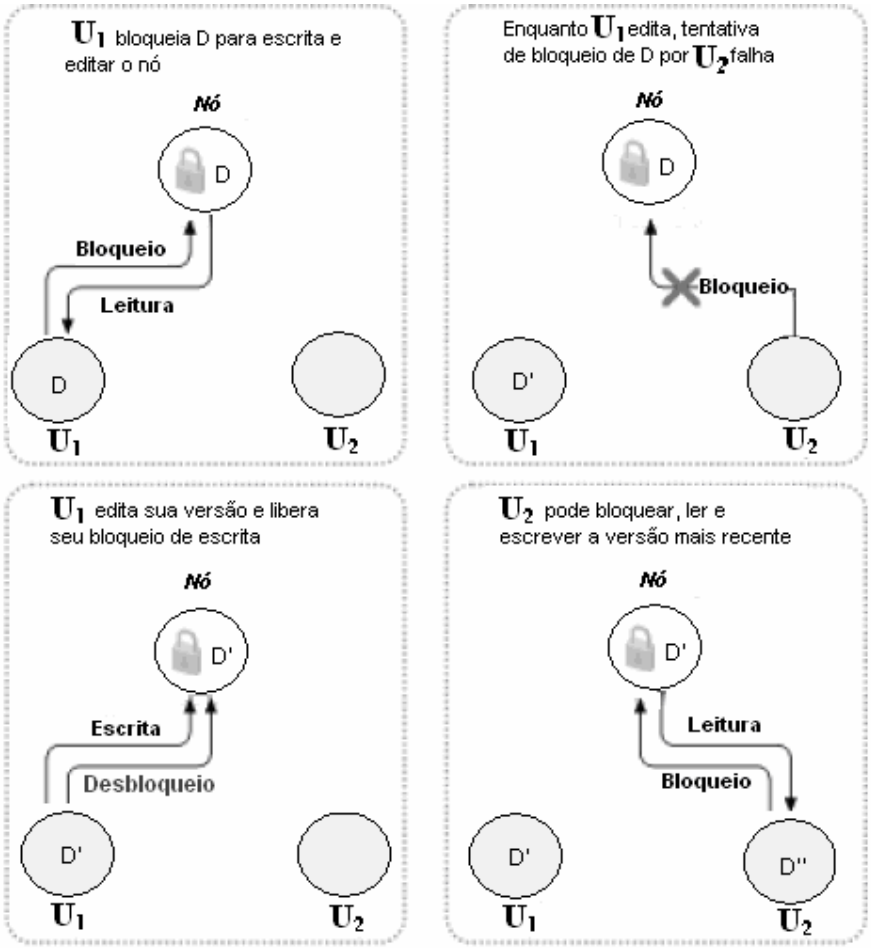

Figura 13 – Método *Lock-Modify-Unlock ou Exclusive Lock*

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*.

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*.

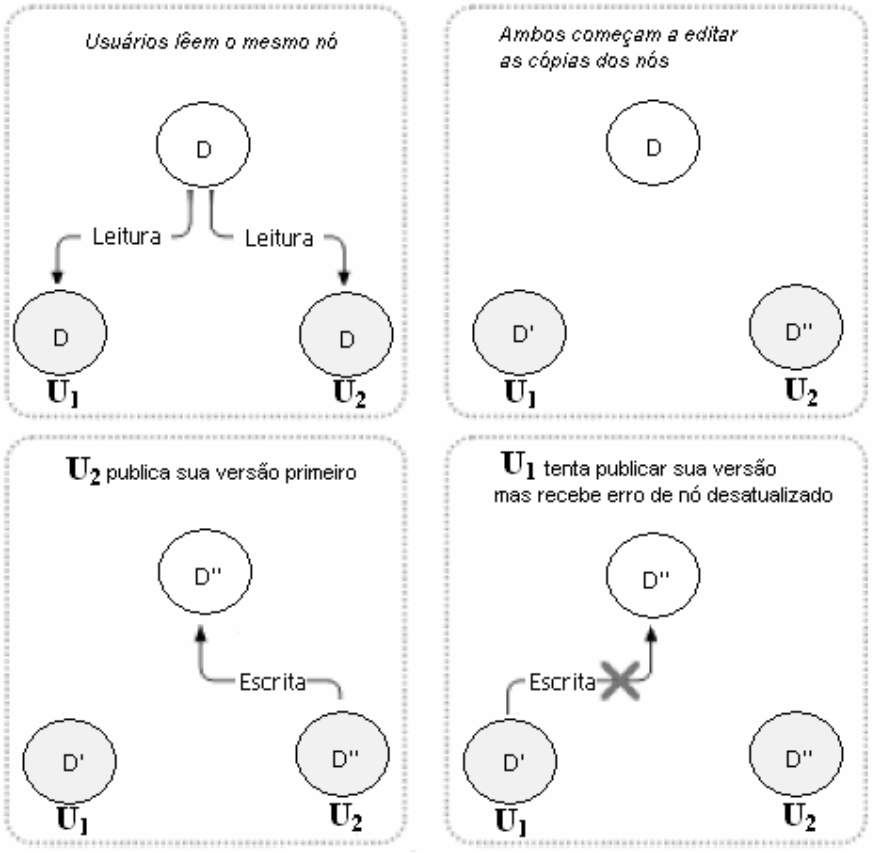

Figura 14 – Método C*opy-Modify-Merge* ou *Optimistic Merges*

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

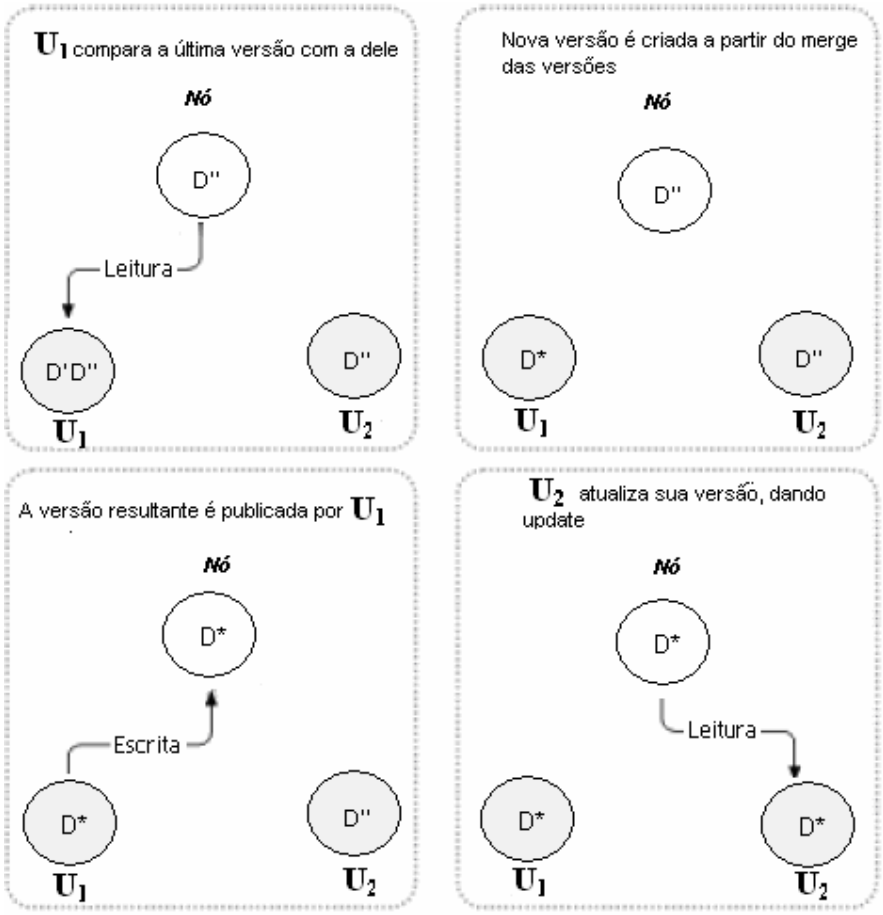

Figura 15 – Continuação do exemplo do método *copy-modify-merge*

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.

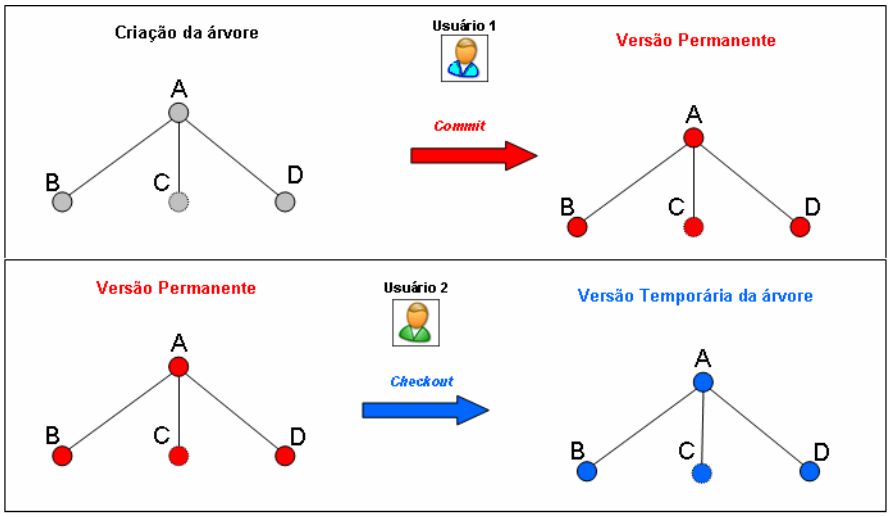

Figura 16 – Versão permanente e temporária

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

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.

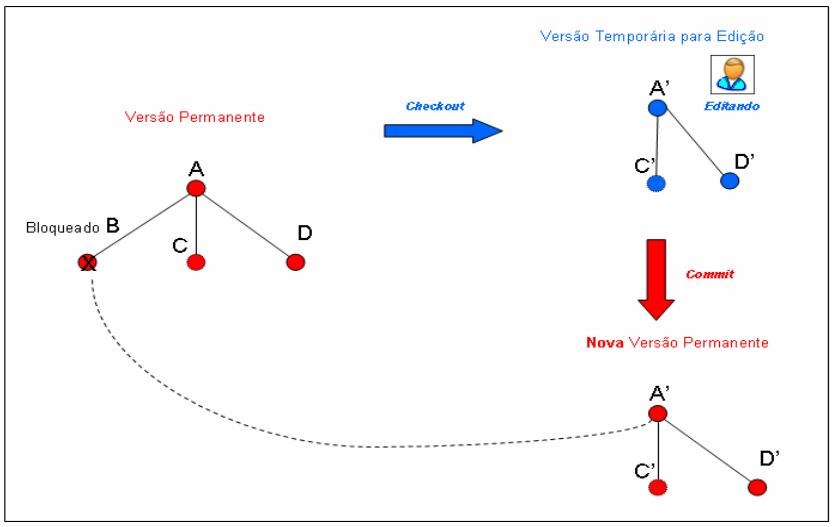

Figura 17 – Versão temporária editada por um único usuário

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.

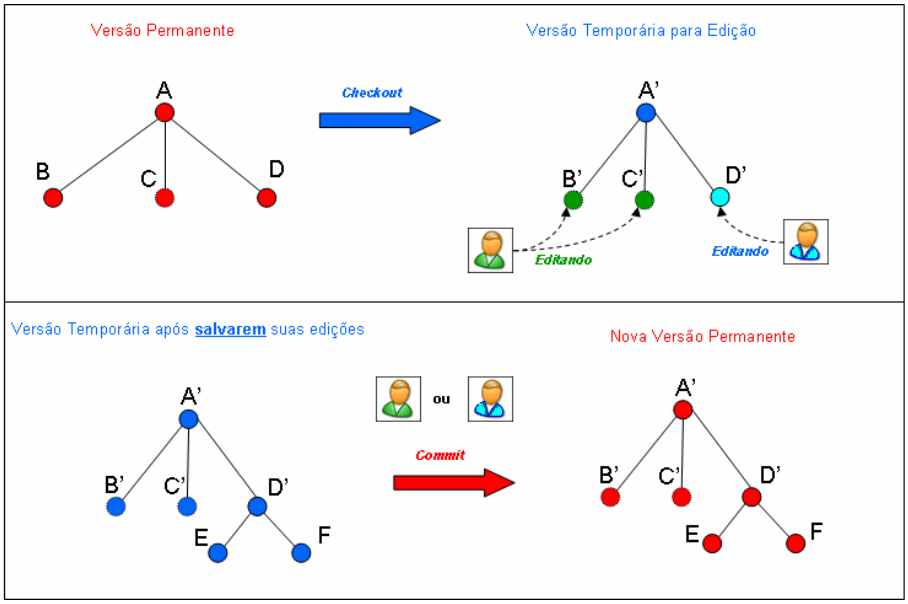

Figura 18 – Versão temporária editada por vários usuários

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.

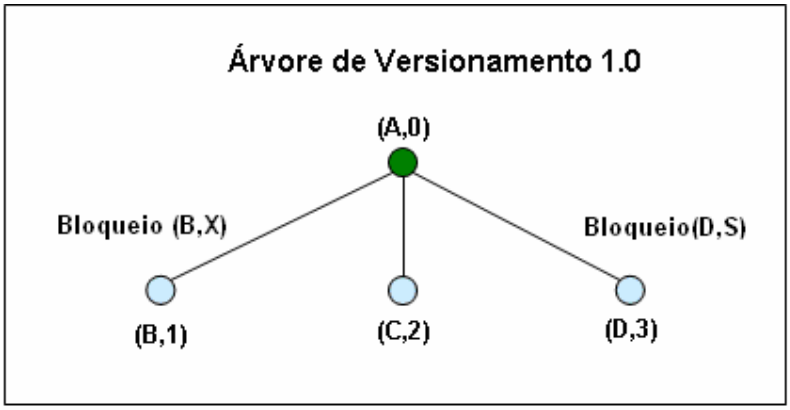

Figura 19 – Árvore de versionamento com bloqueio X e S

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.

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.

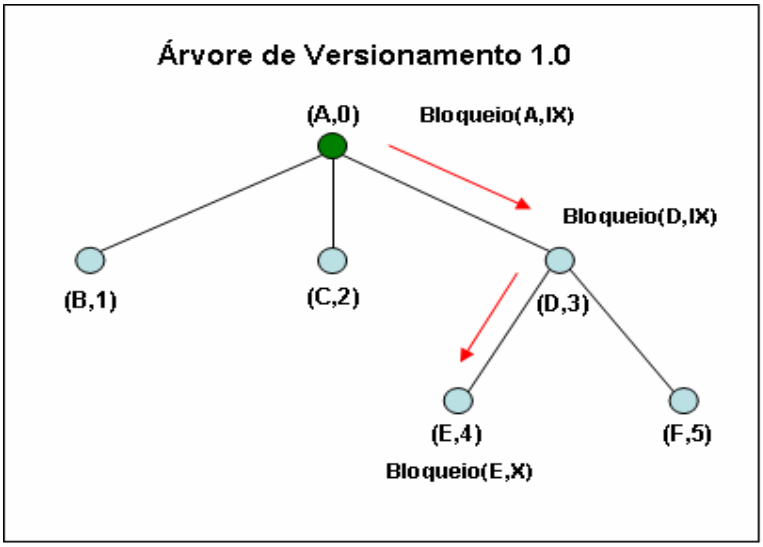

Figura 20 – Árvore de versionamento com bloqueio intencional

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

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.

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,

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

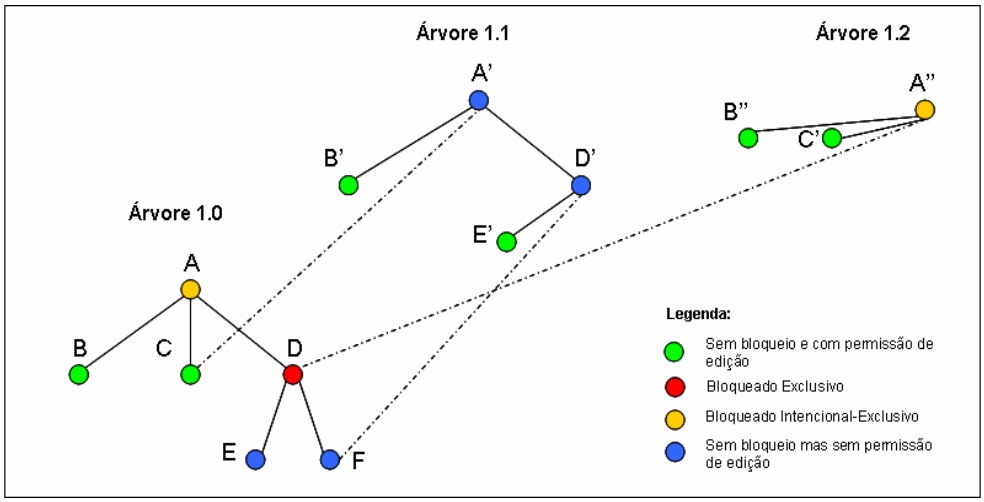

Figura 21 – Exemplo de bloqueio de um nó no modo exclusivo

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.

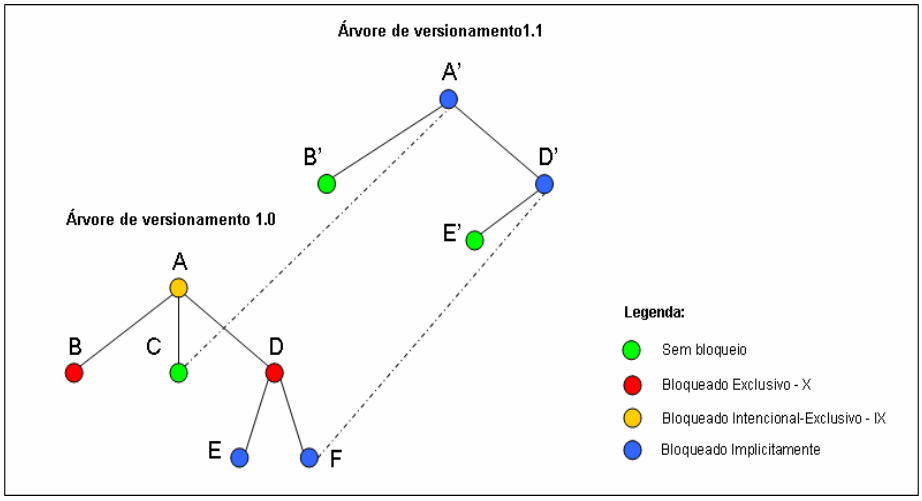

Figura 22 – Estado de bloqueio dos nós

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

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

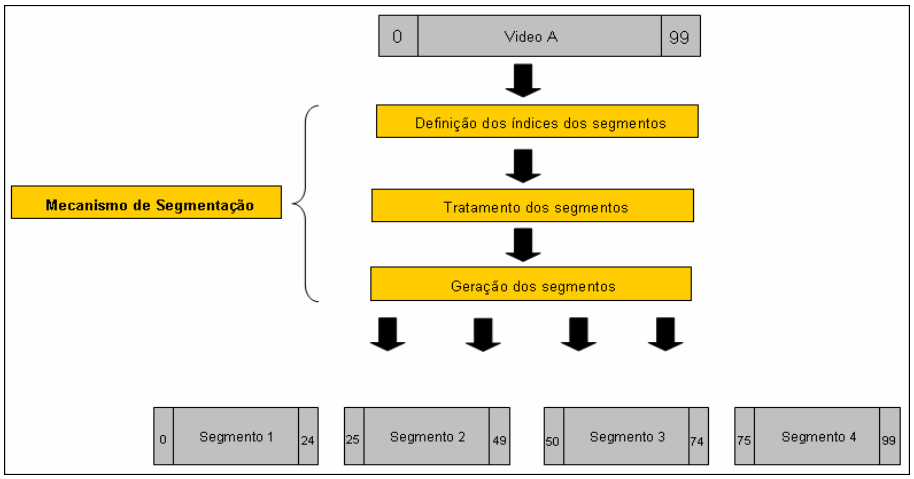

Figura 23 – Fases do mecanismo de segmentação

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.

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.

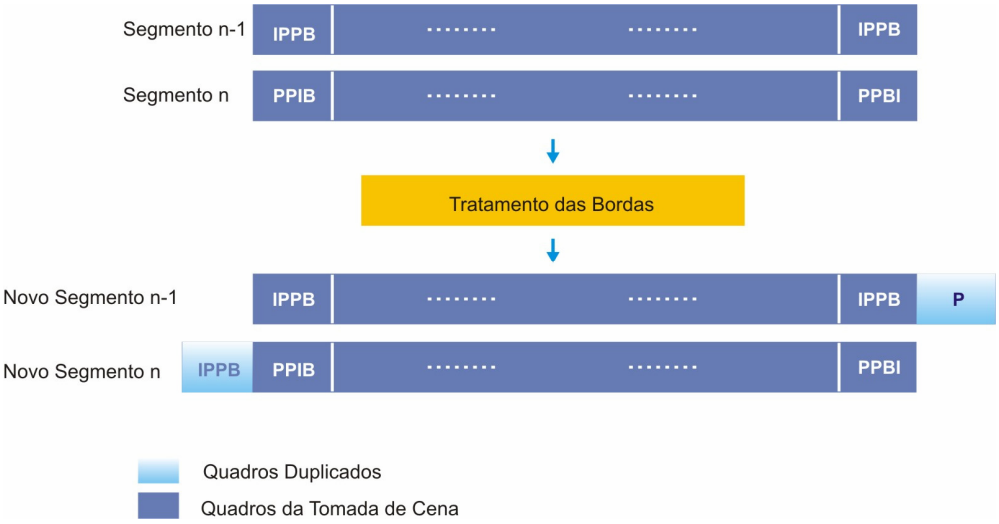

Figura 24 – Tratamento das bordas dos segmentos

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.

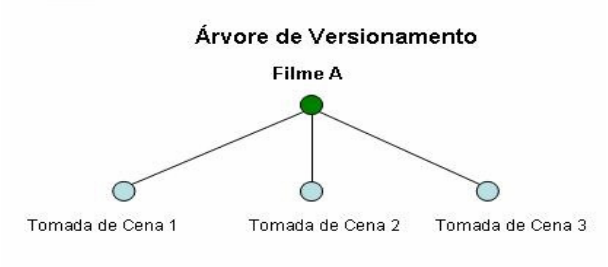

Figura 25 – Detecção das tomadas de cenas de um vídeo

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,

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.

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''**.

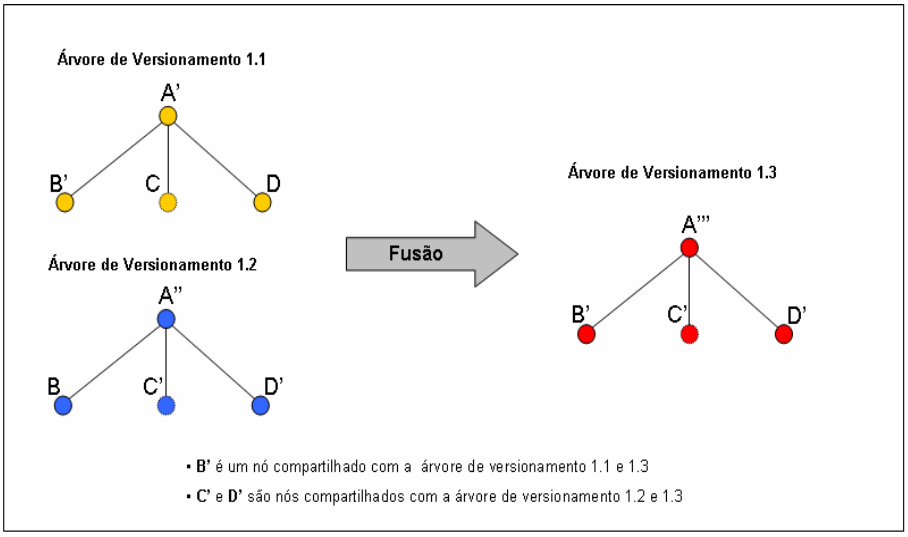

Figura 26 – Fusão de duas árvores de versionamento

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.

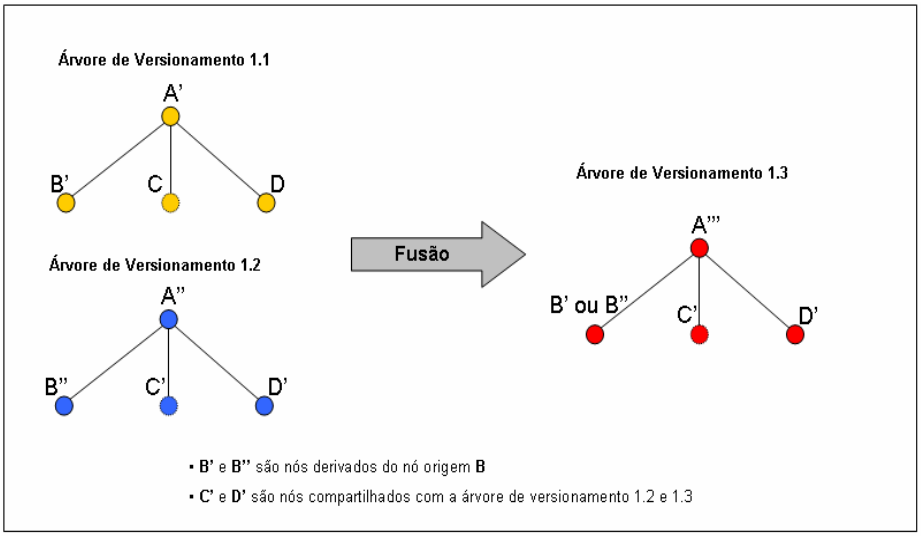

Figura 27 – Fusão das árvores de versionamento com intervenção do usuário

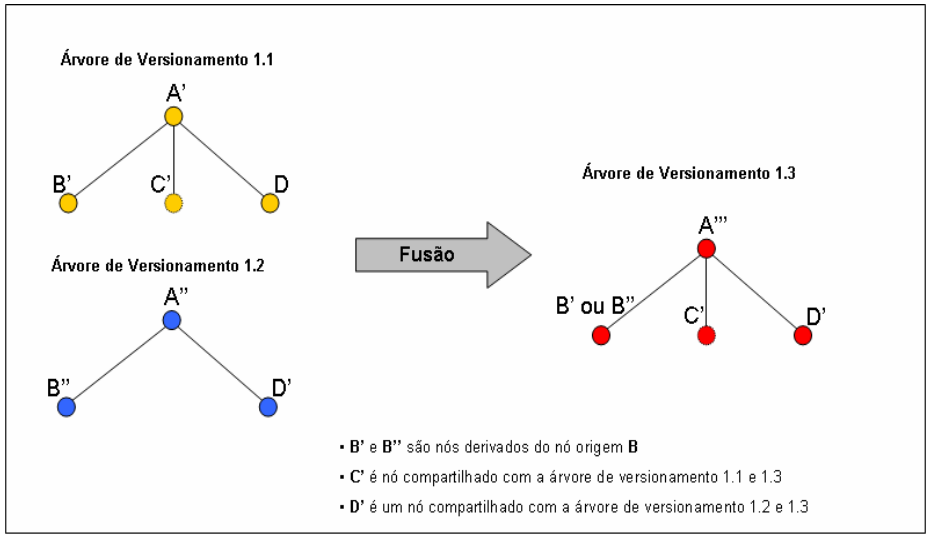

Figura 28 – Fusão de duas árvores de versionamento com nós não equivalentes

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.

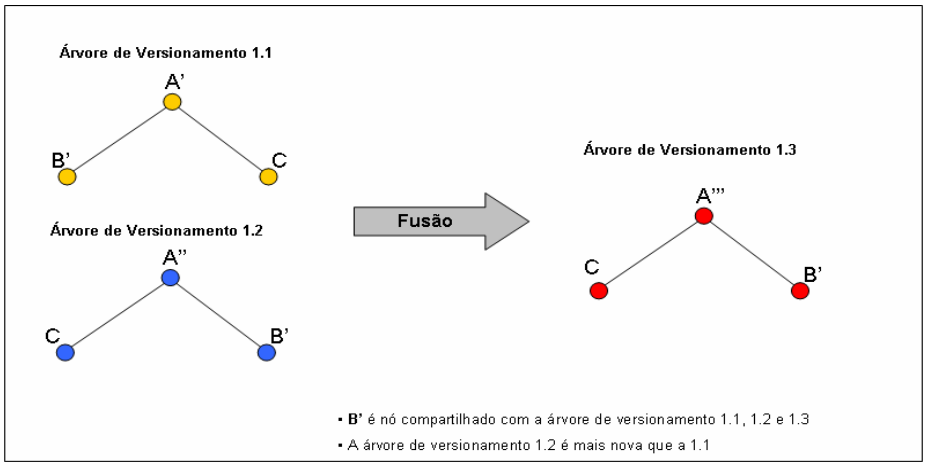

Figura 29 – Fusão de duas árvores de versionamento com nós invertidos