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

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..

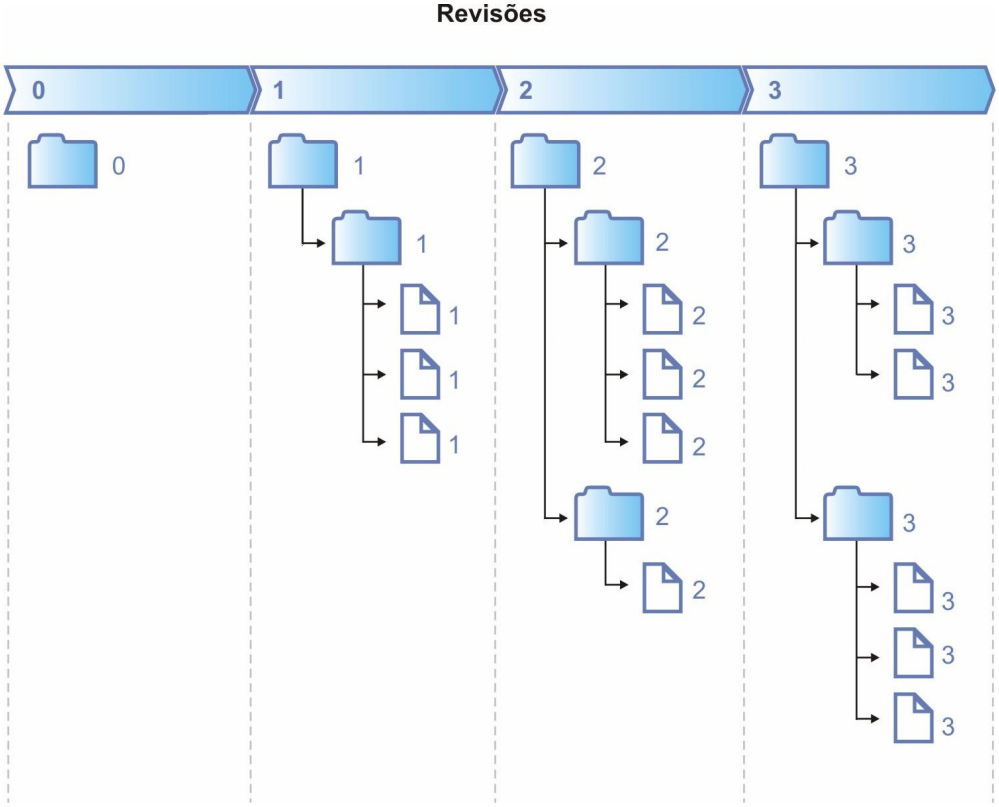

Figura 6 – Exemplo de múltiplas versões organizadas em revisões

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

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.

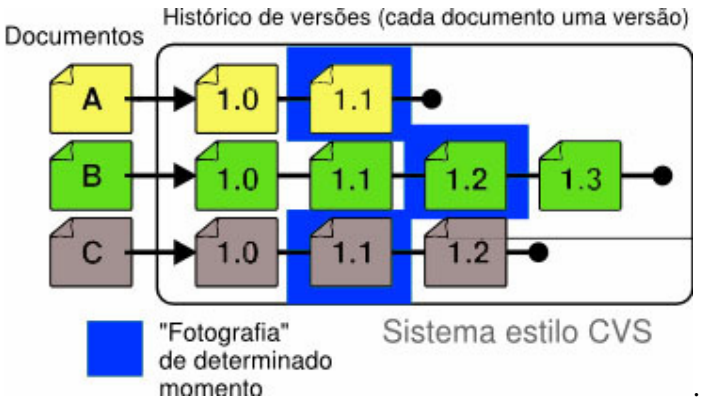

Figura 7 – Histórico de versões no sistema estilo CVS.

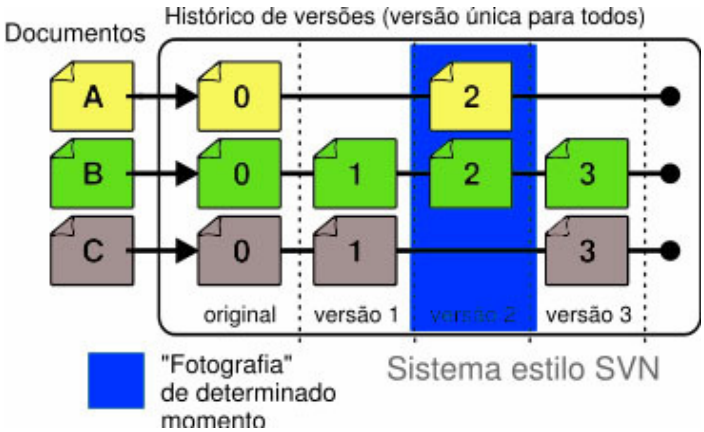

Figura 8 – Histórico de versões no sistema estilo SVN.

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

(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

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

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

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

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.

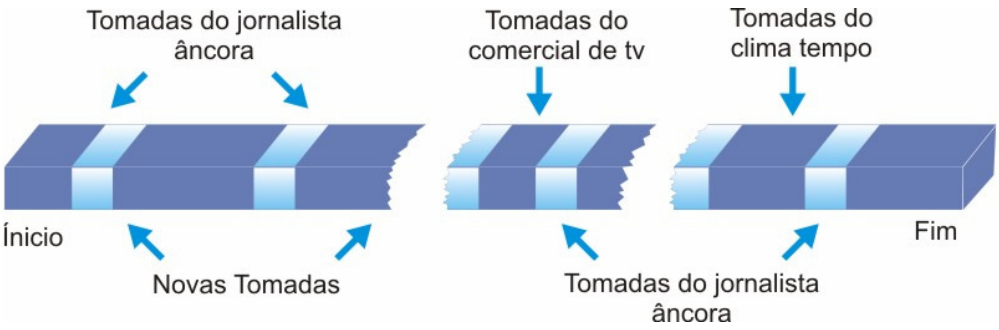

Figura 9 – Detecção das Tomadas de Cenas de um Noticiário de Telejornal.

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

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.