UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB

GUSTAVO VOLPON VALENTE

TRANSMISSÃO DE VÍDEOS COM HTML5

MONOGRAFIA DE ESPECIALIZAÇÃO

LONDRINA - PR
2013

GUSTAVO VOLPON VALENTE

TRANSMISSÃO DE VÍDEOS COM HTML5

Monografia apresentada na Universidade Tecnológica Federal do Paraná – Campus Londrina como requisito parcial para obtenção do título de “Especialista em Desenvolvimento Web”.

Orientador: Prof. Msc. Thiago Prado de Campos

LONDRINA - PR
2013

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Londrina
Diretoria de Pesquisa e Pós-Graduação
Especialização em Desenvolvimento Web

TERMO DE APROVAÇÃO

Título da Monografia

TRANSMISSÃO DE VÍDEOS COM HTML5

por

GUSTAVO VOLPON VALENTE

Esta monografia foi apresentada às 14h00 do dia **23** de **fevereiro** de 2013 como requisito parcial para a obtenção do título de ESPECIALISTA EM DESENVOLVIMENTO WEB. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho _____.

(aprovado, aprovado com restrições ou reprovado)

_____
Prof. Msc. Thiago Prado de Campos
(UTFPR)

_____
Prof. Dr. César Cusin
(FAP-CE)

_____
Prof. Esp. Reinaldo Ferraz
(W3C.br/NIC.br)

Visto da coordenação:

_____
Prof. Msc. Thiago Prado de Campos
Coordenador da esp. em Desenvolvimento Web

_____
Prof. Walmir Eno Pottker
Coordenador de Pós-Graduação Lato Senso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso

RESUMO

VALENTE, GUSTAVO V. Transmissão de vídeos com HTML5**.** 2012. 50 f. Monografia (Especialização em Desenvolvimento Web), Universidade Tecnológica Federal do Paraná. Londrina, 2012.

Este estudo apresenta uma proposta para reprodução de vídeos educativos na internet, tendo o elemento *video*, definido na versão 5 da linguagem HTML, o principal conceito abordado, explorando além da aplicação do elemento, informações relacionadas à arquitetura de distribuição de vídeos e APIs de mídia. O projeto desenvolvido visa à redução dos ruídos que surgem na comunicação durante o processo de transmissão de conteúdos no formato multimídia com objetivo de garantir uma experiência contínua para o usuário.

Palavras-chave: HTML5 vídeo. Vídeo educativo na internet. Arquitetura de distribuição de vídeo.

ABSTRACT

VALENTE, GUSTAVO V. Transmissão de vídeos com HTML5**.** 2012. 50 f. Monografia (Especialização em Desenvolvimento Web), Universidade Tecnológica Federal do Paraná. Londrina, 2012.

This monograph presents a proposal to playing educational videos on the internet, with the video element, defined in HTML5, the main concept discussed by exploring beyond the application of the element, information related to the video distribution architecture and APIs media. The project is aimed at the reduction of noise in communication that arise during the process of transmission of content in multimedia format in order to ensure continuous user experience.

Keywords: HTML5 video. Educational video on the internet. Architecture of video distribution.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	*Application Programming Interface*
AVI	*Audio Video Interleave*
CISCO	*Cisco Visual Networking Index*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
EAD	Ensino a Distância
IDL	*Interface Description Language*
GIF	*Graphics Interchange Format*
GPL	*General Public License*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
HTML	*HyperText Markup Language*
IBOPE	Instituto Brasileiro de Opinião Pública e Estatística
MIDI	*Musical Instrument Digital Interface*
MIME	*Multipurpose Internet Mail Extensions*
MPM	*Multi-Processing Module*
MP4	*Music Player 4*
RTP	*Real-Time Transport Protocol*
RTSP	*Real-Time Streaming Protocol*

SUMÁRIO

1. INTRODUÇÃO

A reprodução de imagens em sequência sempre fascinou as pessoas. Segundo Delvin (2012), quando as imagens começaram a ser incorporadas nos documentos HTML (*HyperText Transfer Protocol*) logo surgiram as primeiras animações comumente usadas nos sites através de arquivos em formato GIF (*Graphics Interchange Format*), uma sequência de figuras compactadas em um único arquivo. Mas além de possuir em sua paleta apenas 256 cores, tornando-o limitado para trabalhar, por exemplo, com imagens fotográficas, o formato GIF não é um arquivo multimídia e dessa forma não fornece nenhum controle ao usuário.

O inicio da reprodução de arquivos multimídia através de um navegador se deu com o formato MIDI (*Musical Instrument Digital Interface*), cujos arquivos podiam ser embutidos no documento através do elemento *embed* para reproduzir músicas de fundo (DELVIN, 2012). Os arquivos nos formatos MIDI e GIF eram relativamente pequenos, com curta duração e qualidade baixa, pois havia limitações da própria internet que tornava inviável a transferência de arquivos pesados pela rede, tendo em vista às conexões lentas com uma das principais.

O aumento expressivo da velocidade de conexão trouxe a viabilidade de transmissão de conteúdos multimídias em conjunto da capacidade de controlar, através de players de mídia, áudios e vídeos em uma página *Web*. Provocando dessa forma um aumento fenomenal na popularidade de vídeos na internet.

Ao longo do último ano a audiência de vídeos na internet teve um crescimento expressivo. De acordo com Watson (2011), mais de 3,5 bilhões de vídeos são vistos diariamente no portal de vídeos YouTube, números que representam um crescimento de 500 milhões de exibições diárias no primeiro semestre de 2011.

No Brasil, segundo IBOPE (Instituto Brasileiro de Opinião Pública e Estatística), o total de pessoas que navegam em sites de vídeos chegou a 31,8 milhões em abril de 2011 (Tabela 1).

Tabela 1 – Alcance e número de usuários únicos, em milhões – subcategorias Vídeos/Filmes e Transmissão de Mídia – trabalho e domicílios, Brasil – abril de 2010 e abril de 2011

Subcategorias	Abril de 2010		Abril de 2011	
	Usuários	Alcance	Usuários	Alcance
Vídeos/Filmes	24,1	65,9%	30,7	71,7%

Transmissão de mídia	14,8	40,3%	16,3	38,1%
Total	25,7	70,8%	31,8	74,3%

Fonte: IBOPE Nielsen Online – NetView (2011).

Diante desse cenário o EAD (Ensino a Distância) ganhou proporções que vão desde vídeo aulas elaboradas até simples vídeos divulgados por qualquer pessoa, como por exemplo, uma aula presencial gravada através da câmera do celular posicionada no fundo da sala.

EAD é uma ação educativa onde o processo comunicacional é realizada com uma separação física entre a pessoa que aprende e a pessoa que ensina (SANTOS 2000).

Sartori e Roesler (2005), citaram a internet com uma das ferramentas tecnológicas que potencializou o EAD, permitindo a troca de informações entre alunos e instituições de ensino de modo rápido e eficiente. De acordo com Lobo (1999), por intermédio de aulas virtuais interativas, distribuídas em formatos de vídeos, proporciona aos estudantes um aprendizado efetivo.

O avanço tecnológico dos equipamentos digitais é sem dúvida outro fator que contribuiu para o crescimento do número de vídeos disponíveis na internet. Um exemplo é a produção de vídeos simples gravados por meio de um aparelho *Smartphone*, onde os esforços para produção do vídeo são pequenos e através do mesmo equipamento conectado a uma rede *Wi-Fi* ou 3G é possível em poucos passos efetuar o envio do arquivo à internet.

Nesta pesquisa vídeos educativos estão relacionados a qualquer vídeo que tenha um objetivo comum de transferir conhecimentos, seja ele prático ou teórico, sobre determinado assunto. Seja uma simples receita de bolo feito em casa até palestras transmitidas ao vivo de qualquer lugar do mundo. Desde claro, que o produtor ou responsável tenha os direitos autorais e permissão para a divulgação do material.

Obviamente um simples vídeo não substitui uma aula presencial, porque o ensino depende de diversos fatores. Apenas uma sequência de imagens gravadas não é suficiente para compreensão de determinados assuntos e nem deve ser abordada como ensino a distância. Portanto, o foco principal desta pesquisa não é explorar a metodologia do ensino a distância ou as vantagens e desvantagens de uma aula expositiva, mas sim colaborar com soluções que contribuam no

desempenho da comunicação de vídeos com conteúdo educativo distribuídos na internet. Pois o método é tão importante quanto à mídia escolhida.

> “Quando se utilizam várias mídias, conseguem-se abordagens diferentes, representações diferentes e focos diferentes.” (FIORENTINI; CARNEIRO, 2001).

2. JUSTIFICATIVA

Para Souza (2006), uma das razões que temos que nos comunicar é para entretermos e sermos entretidos. Existem meios para deixar vídeos mais atrativos através da produção audiovisual, com metas traçadas para transmitir informações específicas e com a finalidade de conquistar ou facilitar o esclarecimento do receptor. Este é o foco das vídeos aulas elaboradas para EAD. Porém uma grande parte dos vídeos publicados na internet trata-se de vídeos com pouco investimento, produzidos e divulgados na maioria das vezes pelos próprios usuários que nem sempre possuem conhecimento técnico de produção audiovisual, resultando em um cenário com milhares de informações disponíveis em vídeo, porém de difícil compreensão.

A comunicação mais poderosa é aquela que vai ao encontro das expectativas do receptor (SOUSA, 2006). Tratando-se da comunicação através da internet a interação é indispensável para obter uma comunicação bem sucedida.

Com relação a interação em vídeos de conteúdos educacionais, existem alguns fatores, definidos por ruídos, que dificultam, ou até mesmo impedem a passagem da mensagem do emissor para receptor. Um desses fatores trata-se da falta de elementos diversos integrados ao vídeo. Por exemplo, os slides durante o acompanhamento de uma palestra. Não somente tê-los disponíveis, mas sim permitir acompanhá-los sincronizados no tempo real da apresentação.

Assistir um filme de terror sem volume, faz o medo diminuir, pois são vários os elementos agregados em um vídeo que podem influenciar o processo de comunicação. Moran (1995), inclusive, relata a influência de efeitos sonoros utilizados no vídeo para provocar associações. Em uma sala de aula, há determinados momentos que exige do professor a utilização de recursos como representações gráficas no quadro, leitura de um estudo de caso, acompanhamento de slides, etc. Estes recursos devem ser atribuídos como meios facilitadores para a transmissão de informações de um vídeo no qual houve incapacidade de registrar um determinado assunto com precisão através apenas de uma sequencia de imagens gravadas. *Canvas*, hipertexto, figura, *WebGL*, *CSS*, *SVG*, geolocalização, áudio, etc., são exemplos de elementos que podem ser integrados a um vídeo no seu tempo real de transmissão.

Outro fator que pode dificultar a comunicação está relacionado à demora ou ausência de *feedback* para os emissores da mensagem. Pois ao contrário do EAD,

em uma aula presencial existe a interação próxima entre professores e alunos tornando possível, ou mais fácil para professores avaliarem a dificuldades dos alunos e, assim, poder alterar o conteúdo abrangido. Ou seja, um professor pode perceber a dificuldade dos alunos e tomar uma ação imediata ou futura, por exemplo, dando mais ou menos ênfase em determinados assuntos, alterando a sequência da aula, o tempo dedicado em cada tópico e assim por diante.

A coleta de dados das ações dos usuários durante a apresentação do vídeo pode minimizar esse tipo de ruído ou prover algumas percepções. O número de vezes em que usuários retornaram a um determinado ponto do vídeo ou quanto tempo o usuário gastou para finalizar cada tópico do vídeo são dados que poderiam sinalizar dificuldades que comprometem a qualidade do ensino.

Ainda existem alguns fatores que podem comprometer a experiência do usuário, relacionados diretamente com tecnologia utilizada no sistema de reprodução e principalmente com a escolha certa de uma arquitetura de distribuição que atenda com êxito a demanda dos usuários.

3. REVISÃO BIBLIOGRÁFICA

Até o momento, a maneira mais utilizada para reproduzir vídeos na internet é através do elemento *object* com apoio de *plug-ins* (programa externo para adicionar funções específicas a um sistema). Antes da versão cinco da linguagem HTML não havia um método definido e especificado para servir conteúdos multimídia. Elevando dessa forma, a popularidade dos *plugins*. (DELVIN, 2012)

Durante anos, os fabricantes dos *plugins* para transmissão de vídeos QuickTime, RealPlayer e Flash Player disputaram o mercado pelo domínio de reprodução de vídeo na internet. Enquanto, QuickTime e RealPlayer focaram seus objetivos como plataformas de reprodução, a ferramenta Flash Player investiu em um ambiente de desenvolvimento, ganhando dessa forma a maior parte do mercado. Em 2010, o Flash estava disponível em quase 97% dos navegadores desktop (HOGAN, 2010).

Por outro lado, sempre houve uma discussão em torno da necessidade do uso de *plugins* para o modelo de transmissão de vídeos. Jobs (2010), relatou que a tecnologia Flash nunca iria aparecer em seus melhores aplicativos. Em vez disso, ele saiu fortemente a favor do padrão aberto HTML5.

Na visão de Powers (2011), existem alguns fatores que prejudicam uma aplicação *Web* com a interferência de *plugins*, podendo até gerar consequências graves aos usuários:

- Instabilidade: Caso um *plugin* necessário não esteja instalado no computador, é necessário baixá-lo e mantê-lo constantemente atualizado. Função geralmente atribuída ao usuário, que nem sempre tem disponibilidade e conhecimento para execução dessa tarefa. Além disso, *plugins* de diferentes fabricantes podem conflitar uns com os outros, causando instabilidade nos navegadores.

- Segurança: Outro fator que deve ser levado em consideração é em relação à falta de segurança, pois *plugins* ainda são um dos principais alvos de *malware*. Em março de 2011, a empresa Adobe divulgou um boletim informando uma vulnerabilidade crítica no Adobe Flash Player 10.2.152.33 e versões anteriores permitindo um invasor assumir o controle do sistema afetado.

- Peso: O elemento de *object* é bastante poderoso na medida que aceita um grande número de argumentos, mas este por sua vez, torna a aplicação o quanto pesada. Pois requer muito código na passagem de parâmetros simples.

A falta de suporte em alguns dispositivos móveis é mais um fator determinante que surge para que desenvolvedores repensem sobre o uso de *plugins* em aplicações *Web*. Para exibir áudio e vídeo em uma página *Web* acessada por meio de aparelhos recentes do fabricante Apple, por exemplo, o HTML5 seria a única opção garantida (CRUSE; JORDAN, 2011).

Segundo Reis (2012), uma pesquisa realizada pelo Comitê Gestor da internet no Brasil, revela que o uso de internet em celulares no país aumentou 340% em 2011. Sendo 17% o número de internautas brasileiros que tenham acessado a rede por dispositivos móveis.

Mesmo assim, não é possível anunciar o fim da tecnologia Flash, com exceção para dispositivos móveis, ela ainda terá seu lugar garantido nos sites por algum tempo. Isto porque, embora as desvantagens do *plugin*, nem todos os navegadores, principalmente nas suas versões antigas, oferecem suporte ao elemento *video* (POWERS, 2011).

Por outro lado, a maioria das funcionalidades da HTML5, é suportada pelos principais navegadores em suas versões atuais, além de existir mecanismos que permitem detectar suporte dessas funcionalidades e assim criar condições para oferecer um conteúdo alternativo (SAMY, 2011).

User Agent	**Version**
Internet Explorer	9+
Google Chrome	3+
Firefox	3.5+
Opera	10,5+
Opera Mini	11+
Safári	3.1+
iOS	3.0+
Android OS	2.0+

Quadro 1 - Navegadores mais populares com suporte ao elemento *video*
Fonte: Powers (2011, p. 1)

Devido o suporte ao elemento *video* em navegadores modernos, os desenvolvedores já não dependem exclusivamente do Flash Player ou qualquer outro *plugin* (POWERS, 2011). O método de entrega com o elemento *video* permite a reprodução de dados multimídia de forma fácil, pois, de acordo com Delvin (2012),

o vídeo passa a ser tratado como um objeto nativo do navegador, trazendo benefícios imediatos como:

- Não necessidade de *plugins*;
- Velocidade e desempenho, sendo que com poucas linhas de código é possível exibir um vídeo.
- Controle padrão fornecido pelos navegadores;
- Acessibilidade através das facilidades de sincronização de legendas e controle do vídeo via teclado.

Direcionado para o desenvolvedor, outra vantagem encontrado no uso do novo elemento está na simplicidade de codificação da marcação de um vídeo. No seu modo mais simples segue a mesma lógica da marcação de uma imagem, porém com *tag* de fechamento.

```
1  <!DOCTYPE html>
2  <head>
3      <title>Título do documento</title>
4      <meta charset="utf-8" />
5  </head>
6  <body>
7      <video src="video_exemplo.mp4">
8      </video>
9  </body>
10 </html>
```

Figura 1 - Marcação do elemento *video* em um documento HTML

À respeito do código exibido na Figura 1, uma curiosa constatação que pode ser feita ao se criar documentos na versão cinco da linguagem HTML é que ele simplifica a informação passada aos navegadores sobre qual o tipo de documento escrito. A primeira linha de um documento em linguagem HTML é conhecida como declaração de DOCTYPE e tem o objetivo de ajudar os validadores de sintaxe a determinarem as regras de validação que devem ser usadas (HOGAN, 2010).

Figura 2 - Player de mídia nativo do navegador Google Chrome

Um *player* de mídia é um termo usado para descrever uma parte do *software* (programa) que tem a capacidade de reproduzir arquivos de multimídia através de um controle com interface gráfica. (DELVIN, 2012). A forma de acessar suas funcionalidades e a interface gráfica são diferentes em cada navegador. Mas de acordo com Pfeiffer (2010), as principais funcionalidades de controles são praticamente as mesmas:

- Controle de volume;
- Botões *play*/*pause* alternado;
- Exibição da linha do tempo de reprodução;
- Botão para reprodução em tela cheia;
- Exibição da posição do tempo da reprodução.

O controle do *player* de mídia disponível no navegador pelo elemento *video* é separada em dois tipos, os controles visíveis e os controles ocultos, que podem ser alcançados através de um clique com botão direito do mouse em cima do elemento (PFEIFFER, 2010).

3.1 ARQUITETURA DE DISTRIBUIÇÃO

A escolha inadequada da arquitetura de distribuição de um vídeo pode comprometer seriamente a qualidade do serviço evitando que o usuário tenha uma experiência

contínua devido a fatores como, a demora no carregamento do conteúdo ou travamento do sistema. (PEREIRA, 2008).

3.1.1 Métodos de Entrega

De acordo com Ozer (2011), é importante reconhecer que existem várias opções para fornecer vídeos pela internet, e dependendo da opção utilizada pode haver um impacto significativo na forma de produção dos arquivos. Para Avila (2008), existem duas formas de transmissão, sob demanda e em tempo real. Já Forouzam (2006), cita três maneiras: sob demanda, tempo real e interativo.

Como o próprio significado do termo, o método de entrega em **tempo real** é voltado para vídeos transmitidos ao vivo. Já a transmissão **interativa** citada por Forouzam (2008), diferencia-se do modelo de tempo real por trabalhar com dois canais de comunicação ao mesmo tempo, de forma que o mesmo usuário pode receber e enviar imagens e som. É o caso, por exemplo, de uma transmissão por vídeo conferência.

Entrega **sob demanda**, é a forma mais utilizada pela própria necessidade do mercado, maneira que os portais mais populares de compartilhamento de vídeo utilizam. Parecido com *download*, porém com a diferença que ao reproduzir após a solicitação do usuário, o conteúdo fica armazenado apenas em uma memória temporária, conhecida como *buffer* (FOROUZAM, 2008).

Baseado na distribuição de vídeos com HTML5, Pfeiffer (2010), cita três métodos principais de entregas para obter a transmissão de conteúdos multimídia. O *streaming* tradicional, *download* progressivo e HTTP *streaming* adaptativo.

3.1.1.1 *Streaming* Tradicional

Utiliza servidores *Web* especializados para o fornecimento apenas dos quadros de um vídeo que está sendo assistido através de protocolos tradicionais de *stream*, como RTP (*Real-Time Transport Protocol*), ou RTSP (*Real-Time Streaming Protocol*), de forma que os dados não ficam armazenados em *cache* ou no cliente. O usuário recebe o conteúdo, decodifica, exibe e este é descartado logo após sua exibição. (LARSON; COSTANTINI, 2007).

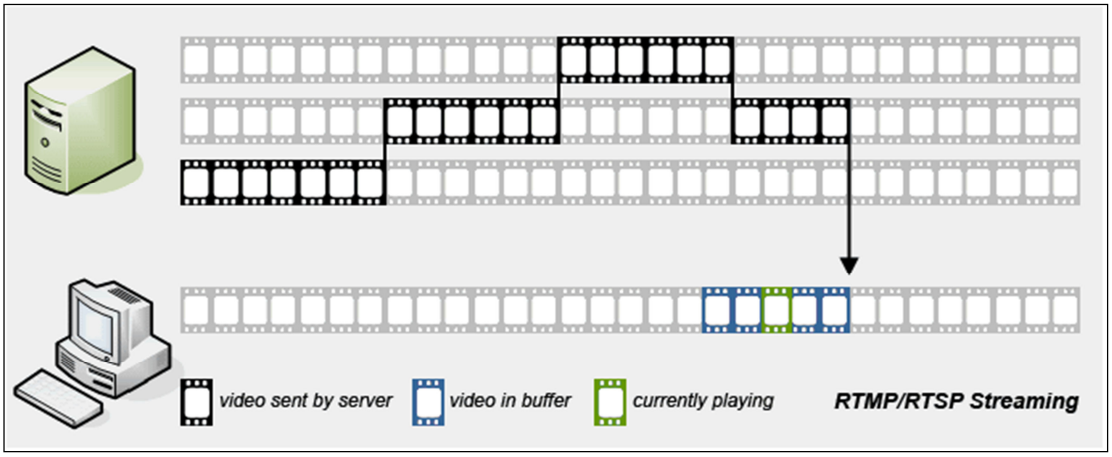

Figura 3 - Método de entrega Streaming Tradicional
Fonte: Wijering (2011).

Para Pfeiffer (2010), os serviços de *streaming* tradicional buscam atender a transmissão ao vivo e oferecer mais proteção aos arquivos através de uma transmissão criptografada. Por outro lado, no caso de uma transmissão criptografada, também é possível obter resultados satisfatórios usando o método de *Download* Progressivo de um servidor *Web* combinado com http ou https.

A maior desvantagem do *streaming* tradicional é devido os frequentes bloqueios dos protocolos dedicados RTP e RTSP feito por *firewalls* corporativos, além da falta de apoio para RTSP no lado do servidor, sendo que a maioria das empresas de hospedagem web prefere não investir em servidores dedicados para para a trasmissão de vídeos. (WIJERING, 2011).

3.1.1.2 *Download* Progressivo

É o método de entrega mais utilizado atualmente. Define-se como um processo de transferência de dados multimídia via protocolo http (*Hypertext Transfer Protocol*), entregue de um servidor *Web* para um computador do usuário, possibilitando-o de visualizar o conteúdo assim que tenha dados suficientes, antes mesmo da cópia completa do arquivo (ISLAN 2010).

De acordo com Pfeiffer (2010), uma das vantagens desse método está relacionada ao fato de que o usuário não precisa esperar até que todo o vídeo seja baixado para assisti-lo, além de ter uma condição interessante que permite ao navegador parar o *download* e continuar após a posição de reprodução estar próxima da posição ainda não gravada. Dessa forma economiza-se o uso de banda. Outra vantagem é que ao contrário de *streaming* tradicional, com *download*

progressivo é possível carregar todo o conteúdo do vídeo com a transmissão parada.

Figura 4 - Método de entrega Download Progressivo

No entanto, como desvantagem, além da falta de segurança em relação à proteção dos arquivos, entregar conteúdo com *Download* Progressivo pode ocasionar que a taxa de dados exceda a largura da banda de conexão, obrigando dessa forma o usuário a esperar mais para assistir o vídeo. Portanto, recomenda-se escolher uma taxa de dados que forneça um bom equilíbrio entre a qualidade e o tempo de espera (OZER, 2011).

3.1.1.3 HTTP *Streaming* Adaptativo

Pfeiffer (2010), descreve o HTTP *streaming* adaptativo com um processo que permite a entrega, por um servidor http, do vídeo ajustado com base na mudança das condições da rede e do software do usuário, a fim de garantir a melhor experiência possível para o telespectador.

HTTP *streaming* adaptativo é baseado na transferência progressiva de pequenos arquivos que podem ser codificados em um ou mais formatos, porém os arquivos entregue ao usuário são escolhidos com base sobre uma estimativa das condições atuais da rede. (ISLAM, 2010).

As vantagens em relação ao *download* progressivo está em situações quando, por exemplo, durante uma transmissão ocorre uma queda na velocidade de conexão. Nestas condições, o servidor automaticamente vai passar a enviar um arquivo de taxa de dados inferior assegurando uma reprodução contínua, mas de qualidade inferior.

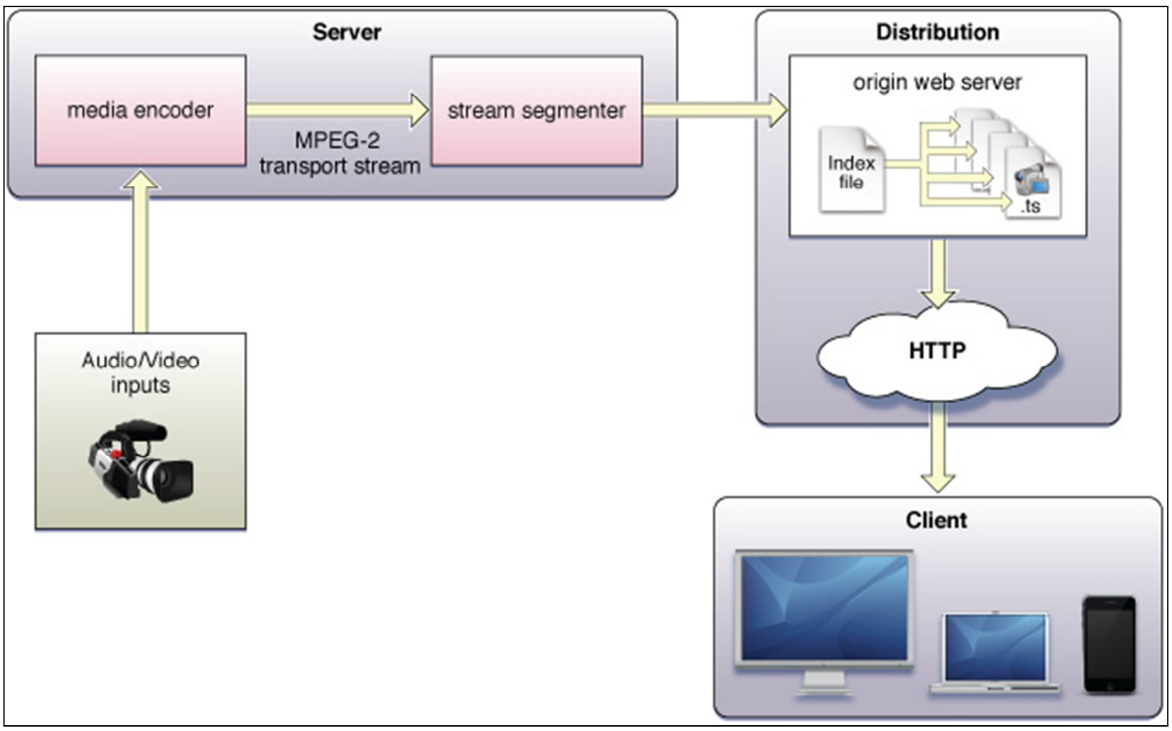

Figura 5 - Método de entrega HTTP Streaming Adaptativo
Fonte: IOS Developer Library (2011).

Para Wijering (2011) o método HTTP *streaming* adaptativo oferece o melhor de todos os cenários possíveis, do ponto de vista que tem a capacidade de obter grande qualidade de vídeo para aqueles com alta velocidade de conexão, e ao mesmo tempo um fluxo razoável de qualidade para usuários, por exemplo, com Wi-Fi, celular ou simplesmente acesso através de uma conexão lenta.

Por se falar em melhoria do desempenho de distribuição de acordo com o dispositivo usado pelo usuário, a notação do elemento *source (*empregado dentro do elemento vídeo*)* permite a utilização de uma expressão denominada *media query* no atributo *media* para indicar a quais dispositivos esta fonte é mais adequada, desta forma possibilita a entrega de diferentes arquivos conforme o tamanho da tela do dispositivo, por exemplo (LAWSON; SHARP, 2012).

```
<video controls>
      <source src=hi-res.webm ... media="(min-device-width:
      ¬ 800px)">
      <source src=lo-res.webm>
      ...
</video>
```

Figura 6 – Código HTML com *min-device-width* no elemento *source*
Fonte: Lawson; Sharp (2012, p. 123).

A desvantagem do HTTP *streaming* adaptativo em relação aos outros métodos de entrega é a falta de padronização (WIJERING, 2011). Por se tratar de uma tecnologia recente ainda há falta de suporte de alguns distribuidores de conteúdo (Quadro 2).

Device	Progressive Download	RTMP/RTSP Streaming	Adaptive HTTP Streaming
Safari e IE9	MP4	-	-
Firefox e Chrome	WebM	-	-
iOs	MP4	-	HLS
Android	MP4, WebM	RTSP	HLS (a partir de 3.0)

Quadro 2 - Suporte aos métodos de entrega através de dispositivos e servidores
Fonte: Wijering (2011).

A melhor arquitetura de distribuição vai depender da necessidade da demanda. Portanto é imprescindível fazer uma avaliação do cenário. Em situações onde será preciso efetuar uma transmissão ao vivo ou proteger o conteúdo, o *streaming* tradicional será a melhor escolha, já em outras situações o *download* progressivo pode ser mais adequado por permitir uma experiência contínua do usuário (PEREIRA, 2008).

3.1.2 Container e *Codec*

Se preocupar apenas no formato de arquivos que são usados na transmissão dos dados, como por exemplo, se disponibilizar arquivos AVI (*Audio Video Interleave*) ou arquivos MP4 (*Music Player 4*), seria uma simplificação excessiva (HOGAN, 2010).

De acordo com Pilgrim (2011), quando o usuário assiste a um vídeo pela internet o *player* de mídia está executando pelo menos três processos:

- Interpretando o formato da embalagem para descobrir qual faixa de vídeo e áudio está disponível.
- Decodificando o fluxo de vídeo para exibir as imagens na tela.
- Decodificando o fluxo de áudio para enviar o som na saída de áudio.

Portanto AVI e MP4 são apenas os formatos das embalagens comprimidas de vídeo, também conhecidas por containers, que armazenam fluxos de informações. Da mesma maneira que, por exemplo, um arquivo compactado pode conter vários e diferentes arquivos armazenados (PILGRIM, 2011). Sem a

compressão dos dados dificilmente se conseguiria atingir a necessidade mínima de banda para reproduzir os arquivos sem interrupções.

Um container pode possuir também, além do fluxo de mídia, metadados como, por exemplo, legendas ou subtítulos. E tem a capacidade de encapsular diversos *codec* diferentes, mas por outro lado, nem todos os fluxos de vídeo são compatíveis com todos os containers (POWERS, 2011). Segundo Pfeiffer (2010), apenas alguns *codec* são normalmente encontrados em determinados recipientes. WebM, por exemplo, é definido para conter apenas VP8 e Vorbis enquanto Ogg Theora normalmente contém, Vorbis, Speex, ou FLAC.

O *codec* é um algoritmo codificado em fluxo de vídeo, e dessa forma o *player* de mídia decodifica esse fluxo conforme o *codec* de vídeo. (POWERS, 2011). Para Pfeiffer (2010), *codec* são apenas os dados de áudio ou vídeo comprimidos e entregue a um container responsável por encapsular as amostras em uma estrutura que permite mais tarde a decodificação desses dados.

A maioria dos *codec* de vídeos modernos, ao invés de armazenar cada *frame* individualmente, armazenam apenas as diferenças entre os *frames*, minimizando dessa forma a quantidade de informação necessária exibidas em um frame atrás do outro (PILGRIM, 2011).

Hickson (2009), relata que um *codec* bem sucedido deve: ser implementável sem custos; ter os chips *off-the-shelf hardware decoder* disponíveis; ser usado amplamente e o suficiente para justificar o excesso de exposição de patente e ter uma qualidade por *bit* alta o suficiente para gerar grandes volumes.

Atualmente os principais containers suportado pelos navegadores mais populares compatíveis com o elemento *video* são o MPEG-4 (http://www.mpeg.org), Ogg Theora (http://www.theora.org) e WebM (http://www.webmproject.org), conforme ilustrado no quadro 3.

	WEBM (VP8 *codec*)	**MP4 (H.264 *codec*)**	**OGV (OGG THEORA *codec*)**
Opera	Sim	Não	Sim
Firefox	Sim	Não	Sim
Chrome	Sim	Sim (será descontinuado)	Sim
IE9+	Sim (mas o codec deve ser instalado manualmente)	Sim	Não
Safári	Não	Sim	Não

Quadro 3 - *Codec* de vídeo suportado nos navegadores mais populares
Fonte: Lawson; Sharp (2012, p. 118).

A seguir explicamos alguns destes containers.

3.1.2.1 MPEG-4

Envolve arquivos com o *codec* de vídeo H.264 e de áudio AAC. Padronizado pelo grupo MPEG em 2003, é um dos formatos mais populares, além de possuir alta qualidade (POWERS, 2011). Jägenstedt (2009), considera H.264 incompatível com a plataforma aberta da *Web* devido à sua licença. Para Powers (2011), o H.264 é uma escolha controversa por causa das patentes detidas no *codec*. Questões de patentes levaram reprodutores de vídeo como, Firefox e Opera, nunca o terem apoiado. Embora os arquivos de vídeos sejam codificados em H.264 sem custos ao usuário, os reprodutores de vídeos, como navegadores, estão sujeitos ao pagamento de *royalties*.

3.1.2.2 WebM

Criado por uma empresa chamada On2 e posteriormente comprado pelo Google, ao contrário de muitos outros recipientes, WebM suporta apenas um codec de áudio, Vorbis, e um codec de vídeo, VP8 (POWERS, 2011). Segundo Pfeifer (2010), a qualidade do VP8 está próxima do H.264 e além de possuir a vantagem de ser livre da cobrança *royalties* é uma plataforma de código aberto. Por isso tem grandes chances de ser adotado como codec base da HTML5.

3.1.2.3 Ogg Theora

Desenvolvido pela Fundação Xiph.Org, Theora é um *codec* livre que permite aos fabricantes de navegadores reproduzirem conteúdos de vídeos sem custos. Porém a desvantagem dessa tecnologia fica por conta da qualidade de vídeo ser inferior referente a outras tecnologias disponíveis (PFEIFFER, 2010). De acordo com Lawson e Sharp (2012), realmente só é útil o uso de Ogg no caso de haver necessidade de incluir um vídeo com suporte para versões mais antigas de navegadores como o Firefox 3.x.

Não existe um *codec* base padronizado na linguagem HTML. Um *codec* de linha de base poderia ser definido por um método de codificação suportada e implementada por todos os fabricantes de navegadores, com o qual os desenvolvedores pudessem contar sem precisar se preocupar com o ambiente que está rodando sua aplicação. (PFEIFFER, 2010).

É possível declarar vários elementos *source* apontando cada um para um tipo de arquivo (Figura 7), e diante de vários elementos *source*, o navegador vai verificar, na ordem o primeiro e consultar se ele consegue reproduzi-lo. Neste caso, o navegador levará em consideração o atributo *type*, que dá informações explícitas sobre o recipiente tipo MIME ou ainda com base na extensão do arquivo indicado. Caso não se enquadre na exigência daquele navegador parte para verificação do segundo elemento e assim por diante (LAWSON; SHARP, 2012).

```
<video>
    <source src="video.webm" type='video/webm; codecs="vp8, vorbis"'>
    <source src="video.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />
    <source src="video.ogv" type='video/ogg; codecs="theora, vorbis"' />
    <p>Seu navegador não suporta elemento video</p>
</video>
```

Figura 7: Marcação HTML de vários elementos source

Pode ser adicionada qualquer marcação HTML dentro do elemento *video*, incluindo os elementos *object* e *embed* (Figura 8). Assim, por exemplo, pode-se fornecer uma alternativa de reprodução com suporte a navegadores antigos (PFEIFFER, 2010).

```
<video src="video.webm" controls width="720" height="480">
    <object type="application/x-java-applet" width="720" height="480">
        <param name='movie' value='player.swf'>
        <param name='allowfullscreen' value='true'>
        <param name='allowscriptaccess' value='always'>
    </object>
    <embed src="player.swf" type="application/x-shockwave-flash"
        allowscriptaccess="always" allowfullscreen="true"
        width="720" height="480"></embed>
</video>
```

Figura 8: Marcação dos elementos *object* e *embed* entre elemento *video*

3.2 API HTML5 VÍDEO

3.2.1 Propriedades, Métodos e Eventos

De acordo com Lawson e Sharp (2012), a versão cinco da linguagem HTML além do elemento *vídeo*, também inclui APIs (*Application Programming Interface*) que facilita a manipulação do conteúdo e reprodução através de JavaScript. As APIs de mídia são compostas por uma série de funções acessíveis por programação JavaScript, e incluindo funções como ajustar volume, procurar determinado ponto do vídeo, reproduzir e pausar o vídeo, etc. (DELVIN, 2012).

JavaScript é a linguagem de script usada para tarefas de programação do lado do cliente que superam as limitações da HTML e CSS. Tarefas essas que vão desde a simples manipulação de uma da interface do usuário até a execução de um programa complexo de análise de imagem, proporcionando total flexibilidade para alterar qualquer objeto no DOM (PFEIFFER, 2010).

A linguagem JavaScript organiza todos os elementos de um documento *Web* em uma hierarquia. No caso do elemento *video*, possui propriedades (Quadro 4), métodos (Quadro 5) e eventos (Quadro 6).

Propriedade	**Descrição**
videoWidth	Atributo somente de leitura que contém a largura real em pixels do arquivo de vídeo.
videoHeight	Atributo somente de leitura que contém a altura real em pixels do arquivo de vídeo.
currentTime	Contém o tempo atual de reprodução que o vídeo está sendo exibido em segundos. O evento *timeupdate* é gerado quando este valor muda. O evento *ended* é gerado quando este valor chega ao fim.
currentSrc	Inicialmente vazio, contém a URL para a fonte de mídia que está realmente selecionado pelo player de mídia.
Duration	Contém o tempo de duração do áudio ou vídeo. Este é um atributo somente de leitura e só pode ser disponível se o vídeo for pré-carregado.
Ended	É um atributo booleano que indica se o recurso de vídeo terminou a transmissão.
Buffered	Contém o intervalo de tempo do elemento vídeo que o navegador tem atualmente armazenado em buffer.
Paused	É um atributo booleano que indica recurso de vídeo está atualmente em status de pausa.
Played	Retorna um objeto *TimeRanges* indicando os intervalos de tempo, se houver.
Seeking	É um atributo booleano que indica se o navegador está atualmente à procura de uma posição diferente da reprodução do vídeo.
Volume	Contém a configuração de volume do áudio ou vídeo. 0 é o menor valor e 1,0 é o mais alto.
playbackRate	Indica a velocidade desejada à qual o vídeo deve ser transmitido. Os valores podem variar entre -1,0 e 1,0.
initialTime	Contém a posição inicial de reprodução de áudio ou recurso de vídeo.
startOffsetTime	Contém um objeto Date que representa o deslocamento da linha do tempo atual.

Quadro 4 - Principais Atributos IDL consistente entre os navegadores
Fonte: Adaptado de Delvin (2012)

É importante saber que cada navegador pode oferecer APIs de mídia únicas (WILCOX, 2010).

Método	Descrição
load()	Faz com que todas as atividades em um recurso de mídia seja suspenso imediatamente, e o elemento em questão é reposto para os valores padrões.
play()	Informa o elemento de mídia para iniciar a reprodução de um recurso de mídia.
pause()	Faz com que um recurso de mídia parar de jogar.
canPlayType(type)	Toma uma representação em cadeia de um tipo de MIME como um parâmetro e retorna um valor de seqüência indicando a capacidade de percepção do navegador para reproduzir esse tipo de mídia particular.
addTextTrack(kind, [label],[language])	Retorna e adiciona uma faixa de texto, que é acrescenta também à lista dos elementos.

Quadro 5 - Principais Métodos consistentes entre os navegadores
Fonte: Adaptado de Delvin (2012)

O JavaScript acrescenta interatividade em documentos *Web* a partir das ações executadas pelo usuário, dessa forma os eventos são os responsáveis por detectar essa ação possibilitando então aplicar uma reação sobre os acontecimentos com base no valor de mudanças e chamadas dos métodos (DELVIN, 2012).

Evento	Descrição
loadstart	É gerado quando o navegador começa apontar aos dados de mídia para carregar.
progress	É gerado quando o navegador está recuperando dados de mídia.
suspend	É gerado quando o navegador busca dados de mídia, mas faz uma pausa e até buscar o recurso de mídia inteiro.
abort	É gerado quando o navegador interrompe a busca de dados de mídia, mas não por causa de um erro.
error	É gerado quando ocorreu um erro enquanto o navegador busca os dados de mídia.
emptied	É gerado quando a conexão de rede é perdida enquanto o navegador busca dados de mídia ou o método *load()* foi chamado quando um método já estava em andamento.
stalled	É gerado quando o navegador está tentando buscar dados de mídia, mas por alguma razão os dados não estão sendo transferidos.
loadedmetadata	É gerado quando a duração e as dimensões do recurso de mídia foram adquiridos pelo navegador.
loadeddata	É gerado quando o navegador encontra a posição inicial do recurso de mídia.
canplay	É gerado quando o navegador pode iniciar a reprodução, pela primeira vez, mas não pode garantir que a após o inicio da reprodução não vai ter que fazer uma pausa para buscar mais dados de mídia.
canplaythrough	É gerado quando o navegador é capaz de reproduzir o recurso de mídia do início ao fim, sem ter que fazer pausas para buscar mais dados de mídia.
playing	É gerado quando a reprodução de um recurso de mídia está pronto para começar, depois de ter sido previamente parado ou atrasado devido à falta de dados suficientes de mídia.

waiting	É gerado quando o navegador parou a reprodução devido a dados insuficientes de mídia sendo disponível.
seeking	É gerado quando se estiver buscando dados de mídia é definido como verdadeiro.
seeked	É gerado quando se estiver buscando os dados de mídia é definida como falsa.
ended	É gerado quando o recurso de mídia interrompe a transmissão porque terminou o vídeo.
durationchange	É gerado quando o atributo do recurso de mídia de duração foi atualizado.
timeupdate	É gerado quando a posição do recurso de mídia de reprodução atual foi alterado.
play	É gerado quando a reprodução normal do vídeo é retomada após o status de pausa.
pause	É gerado após a chamada do método *pause()*.
ratechange	É gerado quando os atributos *playbackRate* são alteradas.
volumechange	É gerado quando o volume de recursos de mídia ou atributo mudo é alterado.

Quadro 6: Principais Eventos consistentes entre os navegadores
Fonte: Adaptado de Delvin (2012)

3.2.2 Atributos de conteúdo

Há também vários meios de atributos de conteúdo específicos que são compartilhados pelo elemento *video* (POWERS, 2011). A diferença de aplicação entre os atributos de conteúdo e os atributos IDL, é que apenas o primeiro pode ser usado diretamente na marcação e os atributos IDL são expostos por um elemento do DOM, Document Object Model (WILCOX, 2010).

Lista de atributos de conteúdo do elemento *video*:

- *autoplay*: Atributo para iniciar o vídeo automaticamente sem a intervenção do usuário. Com o elemento *video* sem o recurso de *autoplay* atribuído, o navegador baixa apenas dados suficientes a partir do início para conferir se é capaz de descodificar o arquivo e também para descodificar o cabeçalho do vídeo. (PFEIFFER, 2010).

Para Lawson e Sharp (2012), o recurso *autoplay* pode piorar a experiência do expectador. Usuários de dispositivos móveis, provavelmente, não vão gostar de usar banda sem efetuar uma solicitação através do player de mídia do vídeo.

- *controls*: Atribui ao player de mídia um controle padrão de interação do usuário (PFEIFFER, 2010). Tornando o vídeo mais acessível, também é possível mover-se através de controles usando apenas o teclado (LAWSON; SHARP, 2012).

- *poster*: Esse atributo aponta para uma imagem que o navegador irá usar enquanto que o vídeo não está pronto para ser reproduzido, ou enquanto o expectador não autorize a transmissão do vídeo (LAWSON; SHARP, 2012).

Sem o atributo *poster*, a maioria dos navegadores vai exibir o primeiro *frame*, que pode não ser a melhor forma de representar o conteúdo do vídeo (PFEIFFER, 2010).

- *muted*: Com esse recurso atribuído, o vídeo passa a ter um status inicial silenciado, e permanece dessa forma até que usuário altere através do controle do player de mídia (LAWSON; SHARP, 2012).
- *loop*: Esse atributo joga o vídeo em um loop infinito, fazendo com que o vídeo reinicie automaticamente depois de terminar a leitura. É um atributo booleano, portanto não é possível especificar um número de voltas, apenas se tem loop ou não. (PFEIFFER, 2010).
- *preload*: Recurso que permite controlar o comportamento do armazenamento dos dados de vídeo, sugerindo ao navegador pré-carregar o vídeo, antes mesmo da reprodução do mesmo, na expectativa de que o usuário irá ativar os controles (LAWSON; SHARP, 2012).

O atributo *preload* pode assumir os valores *none*, *metadata* ou auto. O valor *none*, indica ao navegador que não deve pré-carregar nada até que o usuário ative o controle. O valor *metadata* sugere que o navegador armazene antes da ativação do controle apenas informações como dimensão, primeiro *frame*, tempo de duração, etc. O valor auto, permite que o navegador comece a armazenar o arquivo inteiro (LAWSON; SHARP, 2012).

- *src*: Atributo que recebe o valor com o local apontado que se encontra arquivo (PFEIFFER, 2010).

Como já foi dito no tópico 3.1.3, existe um elemento *source* que tem a mesma função do atributo *src*, portanto não é recomendado o uso de ambos na declaração de um mesmo vídeo.

```
1  <video controls loop poster="imagem.png" preload="metadata">
2
3  </video>
```

Figura 9 – Marcação HTML do elemento *video* com atributos

3.3 CSS (*CASCADING STYLE SHEETS*)

CSS é uma linguagem de design padrão que quando combinado com uma linguagem de marcação estrutural oferece informações ao navegador para apresentar de forma personalizada os aspectos visuais de um documento *Web*, como por exemplo, espaçamento entre parágrafos, efeitos textuais, cores de fundo, posicionamentos dos elementos e uma série de outros efeitos (YORK, 2005).

Os elementos de mídia da HTML5 são elementos de bloco. Portanto, um elemento *video* pode receber os mesmos efeitos de estilização que, por exemplo, um elemento *div* (POWERS, 2011).

```
1  <!DOCTYPE html>
2  <head>
3      <title>Vídeo com CSS</title>
4      <meta charset="utf-8" />
5      <style type="text/css">
6          .player {
7              width:500px;
8              height:282px;
9              border-radius:5px;
10             box-shadow:0 0 12px black;
11             border:10px solid #ccc;
12             padding:6px;
13             background:red;
14         }
15     </style>
16 </head>
17 <body>
18     <video class="player" src="video_exemplo.mp4" controls>
19
20     </video>
21 </body>
22 </html>
```

Figura 10 – Declaração de estilos CSS para o elemento *video*

Figura 11 - Elemento *video* estilizado com CSS

4. METODOLOGIA DE DESENVOLVIMENTO

São diversos os fatores que podem acarretar no fracasso de um projeto. O uso de uma metodologia é importante principalmente para obter a prevenção dos riscos visando à entrega de um sistema com qualidade e em prazos coerentes.

A fim de atingir as metas e requisitos em curto período de tempo, optou-se por utilizar no projeto dessa pesquisa a metodologia de desenvolvimento ágil Scrum.

Scrum não é definido como um processo ou técnica de desenvolvimento, mas sim como um *framework* capaz de controlar os processo e riscos, que emprega uma abordagem iterativa e incremental e assim otimizar a previsão e controle de riscos do projeto (Schwaber, 2009).

4.1 PAPÉIS NO SCRUM

De acordo com Schwaber (2009), os integrantes do projeto, chamados de Time Scrum é composto pelo *Scrum Master*, pelo Dono do Produto e pelo Time.

Scrum Master: é o líder da equipe de desenvolvimento. Responsável por assegurar que a metodologia seja seguida e que a equipe de desenvolvimento seja produtiva, removendo as barreiras encontradas durante o desenvolvimento do projeto.

Dono do Produto: é o representante dos interesses do cliente. Define as características do produto priorizando os requisitos e é o responsável por maximizar o valor do trabalho que o Time realiza.

Time: é a equipe de desenvolvimento responsável pela produção do produto. Deve possuir poucos integrantes e é auto organizável, ou seja, na reunião de planejamento o time se organiza para atender melhor as tarefas definidas para o projeto.

4.2 CICLO DE DESENVOLVIMENTO

O *framework* Scrum é baseado em interações chamadas de *Sprint*, definido por eventos com duração fixa. Inicialmente é feita a definição das funcionalidades que serão implantadas, representada por uma lista, denominada *Product Backlog*. A prioridade estipulada na lista é marcada através de pontuação, onde nessa ocasião também pode ter a relação da quantidade de horas que cada funcionalidade deve ocupar.

No inicio da *Sprint* ocorre a *Sprint Planning*, reunião de planejamento. Nessa ocasião surge o *Sprint Backlog*, lista com os passos necessários para implantação de uma funcionalidade do sistema definindo todas as tarefas que devem ser executadas para o desenvolvimento de cada funcionalidade. Nessa lista também é estipulada a quantidade de horas de cada passo dando margem para comparar com o total de horas prevista no *Product Backlog*.

Durante a execução da *Sprint* é realizada a *Daily Scrum*, encontros rápidos de no máximo 15 minutos, com participação da equipe, onde o os membros devem responder três perguntas rápidas. O que fez hoje? O que pretende fazer amanhã? Qual impedimento apareceu no caminho?

Essa reunião tem o objetivo de acompanhar o andamento do projeto e averiguar o progresso do desenvolvimento através do *BurnDown* Chart, gráfico simples para que através deste possa acompanhar e calcular a velocidade do desenvolvimento do projeto. Possui dois eixos X e Y, onde X representa os dias da *Sprint* e Y o trabalho restante definido por horas, dias, *user story* ou pontos dependendo da necessidade de cada caso. Com a representação do gráfico é possível facilmente controlar diariamente se as metas estão sendo vencidas. Além do gráfico, o acompanhamento das tarefas ainda pode ser controlado através de um

quadro dividido em três status: tarefas a realizar, tarefas em andamento e tarefas realizadas.

Antes de finalizar a Sprint, acontece o *Sprint Review*. Neste evento é feita a apresentação, ao dono do produto, do que foi realizado, onde o mesmo confere se as metas estabelecidas para a Sprint foram atingidas ou não. Finalizada a Sprint passada, imediatamente inicia um novo clico sem intervalos de tempo.

4.3 ADAPTAÇÃO DA METODOLOGIA

Considerando o fato de o projeto ter sido realizado por apenas um pessoa, a algumas etapas do Srum tiveram que ser adaptadas. Não houve, por exemplo, a necessidade de definir os papeis e de realizar as reuniões diárias durante o ciclo de desenvolvimento, apenas uma revisão geral no final de cada *Sprint*. E como o sistema apresenta poucas funcionalidades também não houve a necessidade de gerar o gráfico *BurnDown*.

5. DESENVOLVIMENTO DO PROJETO

5.1 DESCRIÇÃO DO PROJETO

Tendo em vista os ruídos e barreiras citadas na justificativa deste estudo (Item 2), foi desenvolvido um sistema com o objetivo de apresentar propostas capazes de melhorar o desempenho da comunicação de vídeos aulas.

O tema da vídeo aula é relacionado ao instrumento musical violão. Grande parte dos vídeos, desse tema, disponíveis na internet são produzidas e divulgadas pelos próprios usuários, portanto, a gravação do arquivo de vídeo usado no exemplo foi baseado em uma produção caseira a partir da *webcam* de um notebook, sem uso de programas específicos para edição ou técnicas de produção audiovisual. Composto apenas por imagens de uma pessoa tocando o instrumento.

Como pode ser visto no escopo do projeto (Figura 11), a página possui além do vídeo, um material de apoio com imagens dos acordes sincronizados no tempo atual de exibição e uma área para exibir ações do usuário durante o acompanhamento do vídeo e uma área com um controle customizado.

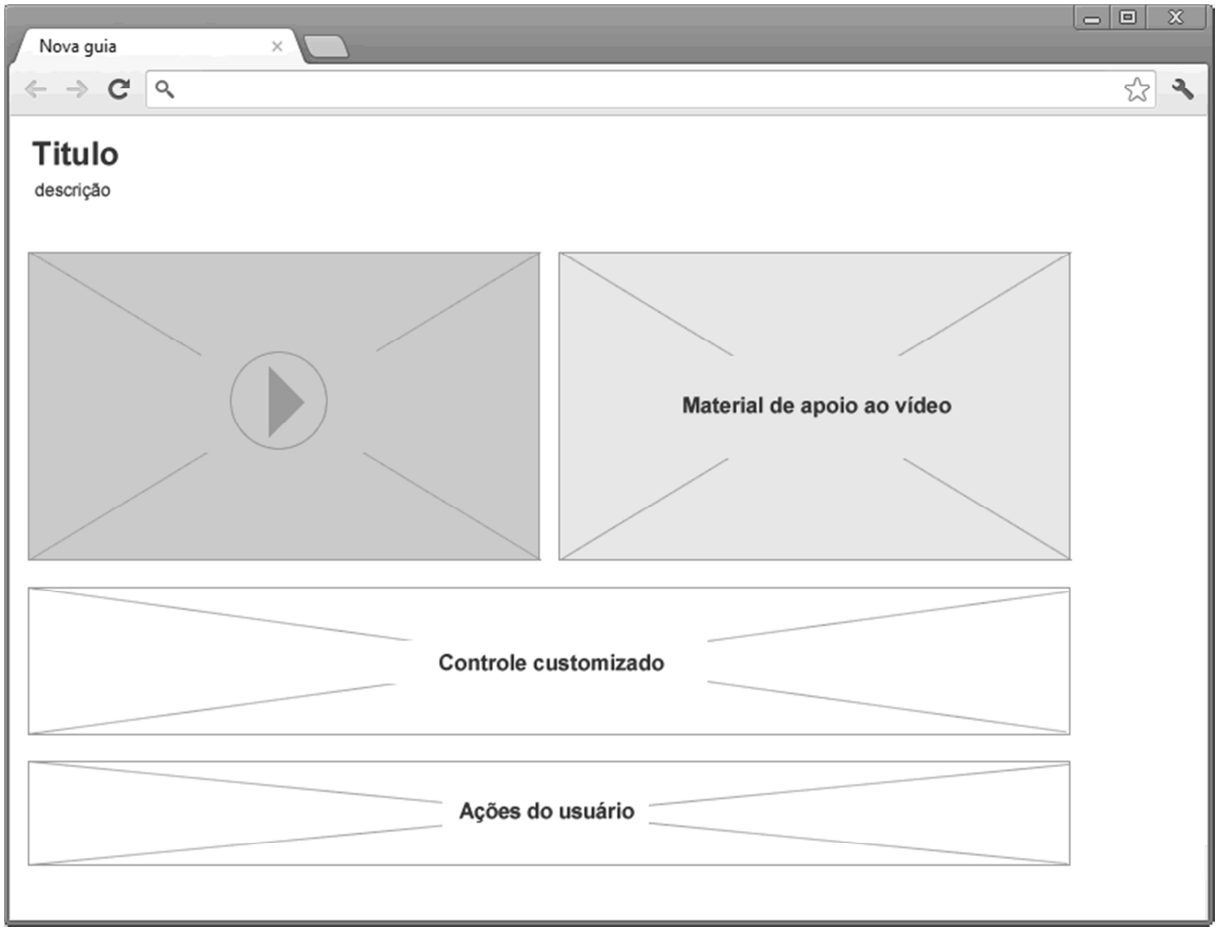

Figura 12 - Escopo do projeto

5.2 DEFINIÇÃO DAS FUNCIONALIDADES E TAREFAS

De acordo com a metodologia de desenvolvimento citado no item 4, inicialmente foi definido o *Product Backlog* contendo a lista de funcionalidades do sistema em ordem de prioridade de entrega e pontuação estipulada pela relação entre custo e tempo de desenvolvimento (Quadro 7).

Funcionalidade	Prioridade	Custo/Hora
Arquitetura de distribuição	1	10
HTML e CSS da estrutura da página	2	03
Desenvolvimento do controle customizado	3	04
Captação dos dados das ações do usuário	4	06
Sincronização dos objetos com o vídeo	5	08

Quadro 7 – Definição do Product Backlog

Após a definição *do Product Backlog*, foi gerado o *Sprint Backlog*, contendo todas as tarefas necessárias para a execução de cada uma das funcionalidades (Quadro 8).

Funcionalidade e Tarefas	Prioridade	Custo/Hora
Arquitetura de distribuição	**1**	**10**
Método de Entrega e Servidor	1.1	01
Gravação de um arquivo de exemplo	1.2	06
Definição dos codes e containers	1.3	01
Codificação do arquivo	1.4	02
HTML e CSS da estrutura da página	**2**	**03**
Marcação HTML	2.1	01
Estilização CSS dos elementos	2.2	01
Marcação HTML do elemento vídeo	2.3	01
Desenvolvimento do controle customizado	**3**	**06**
Marcação HTML dos botões	3.1	01
Desenvolvimento JavaScript das funções do controle	3.2	03
Captação dos dados das ações do usuário	**4**	**06**
Sincronização dos objetos com o vídeo	**5**	**08**

Quadro 8 – Definição do Sprint Backlog

5.3 ARQUITETURA DE DISTRIBUIÇÃO

Apesar do avanço tecnológico e da rápida absorção de vídeos na internet como meio de educação, implantar um sistema de transmissão de vídeo via internet pode ser custoso do ponto de vista que seja necessário adquirir um servidor dedicado ao *streaming* de vídeo. Não se tratando de uma transmissão ao vivo, com objetivo de manter o projeto em baixo custo e garantir o suporte nos navegadores mais populares, o método de entrega escolhido foi o *download* progressivo. Nesse caso, foi necessário apenas de um servidor *Web* que processa e recebe requisições http. Existem vários servidores *Web* , o Apache (http://www.apache.org) é um dos mais populares e optou-se por utilizá-lo no projeto por ser totalmente em código aberto, gratuito e ainda possuir uma grande comunidade de desenvolvimento ativa.

Para desocupar o máximo de memória exigida pelo servidor o ideal seria uma arquitetura que trabalha com conceito de *multi-thread*, onde ao invés de processos as requisições sejam atendidas por *threads*. No Apache, a MPM, (*Multi-*

Processing Module), que se encaixa nessa situação é a *multi-process/multi-thread Worker*, capaz de servir um grande número de pedidos com menos recursos do sistema.

Visando dar suporte ao maior público de espectadores, o vídeo foi disponibilizado em diferentes *codec* e containers, e testado pelos navegadores mais populares. O único navegador que não deu suporte ao elemento *video* foi o Internet Explorer na versão 8, nos outros mostrados no quadro 9, todos os recursos do sistema funcionou corretamente.

Navegador	Versão
Chrome	21.0.1180.60 m
Firefox	18
Internet Explorer	9
Opera	12
Safari	5

Quadro 9 – Navegadores testados para executar o sistema desenvolvido

Para realizar a embalagem comprimida dos fluxos de informações, utilizou-se o programa Miro Video Converter (http://www.mirovideoconverter.com), que é um programa de código aberto para conversão de vídeo em múltiplos formatos com licença GPL (*General Public License*).

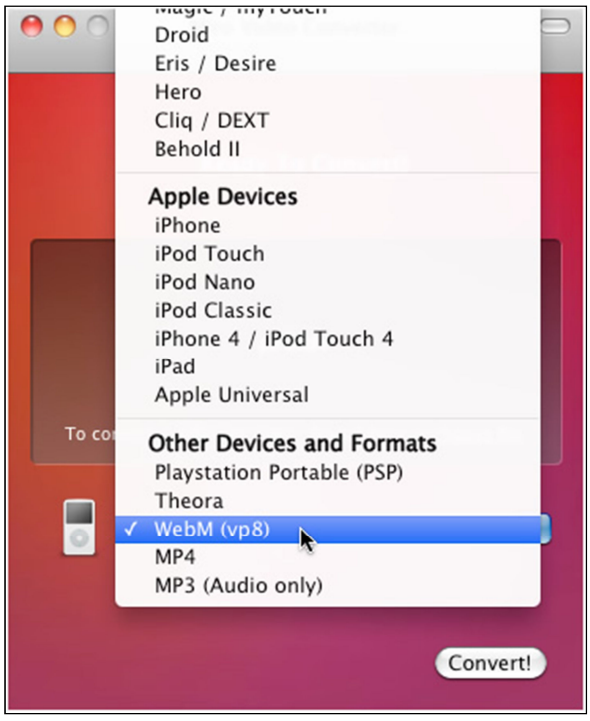

Figura 13 - Tela do programa Mira Video Converter

Mesmo que o foco principal da pesquisa não seja a análise entre containers, a partir do quadro 9 pode ser feita a comparação dos arquivos de dados embalados e do arquivo original gravado. Através de programas específicos, seria possível efetuar o mesmo processo com a seleção de quantidade de percas de qualidade e consequentemente alterar o peso do arquivo. Como citado no item 3.1.1.1, é importante escolher uma taxa de dados que forneça um bom equilíbrio entre a qualidade e o tempo de espera.

Formato	Resolução (pixel)	Peso	Tempo
AVI (*original da gravação)	640 x 480	197.000 KB	00:00:11
MP4	640 x 480	955 KB	00:00:11
WebM	640 X 480	1.076 KB	00:00:11
Theora	640 X 480	424 KB	00:00:11

Quadro 10 – Dados do arquivo original da gravação e arquivos após embalagem

5.4 HTML E CSS DA ESTRUTURA DA PÁGINA

As figuras 13 e 14 apresentadas a seguir, são referentes à marcação HTML e estilização CSS dos elementos que compõe a estrutura da página, segundo a proposta elaborada no escopo do projeto (Figura 12). A estrutura do documento foi separada por camadas, marcadas por elementos *div*, em três áreas principais, topo, conteúdo e rodapé. Na área referente ao conteúdo foram definidas outras quatro áreas para receber conteúdos referentes ao player de vídeo, material de apoio, controle customizados e amostras das ações do usuário.

Também a figura 13, pode-se observar a marcação do elemento *video*, entre as linhas 18 e 23. O código para essa marcação foi escrito conforme orientações especificadas no item 3. Porém é importante ressaltar que o tipo MIME informado para os *codec* H.264 e AAC são de um perfil de baixa complexidade, no qual exige menos CPU na decodificação.

```html
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Título do Documento</title>
<link rel='stylesheet' href='estilo.css' type='text/css' media='all' />
</head>

<body>
    <div id="topo" class="area-ext">
      <h1 id="musica">Título da Música</h1>
      <h2 id="artista">Nome do Artista</h2>
    </div>

    <div id="conteudo" class="area-ext">
        <div class="area player-video">
            <h3>Video</h3>
            <video poster="capa.gif" width="522" height="294">
                <source src="video_aula.webm" type='video/webm;
                codecs="vp8.0, vorbis"'>
                <source src="video_aula.mp4" type='video/mp4;
                codecs="avc1.42E01E, mp4a.40.2"'>
                <source src="video_aula.ogv" type='video/ogg;
                codecs="theora, vorbis"'>
                Navegador não suporta o elemento video
            </video>
        </div>
        <div class="area material-apoio">
            <h3>Material de apoio</h3>
        </div>
        <div class="area botoes-controle">
            <h3>Controle</h3>
        </div>
        <div class="area acoes-usuario">
            <h3>Ações</h3>
        </div>
    </div>

    <div id="rodape" class="area-ext">
      <p>Informações de rodapé</p>
    </div>
</body>
</html>
```

Figura 14 – Marcação HTML das áreas definidas no escopo do projeto

```css
@charset "utf-8";
/* Documento CSS */

body, h1, h2, h3 {
    font-family: Arial, Helvetica, sans-serif;
    margin: 0;
    padding: 0
}
.area-ext {
    margin: 10px auto;
    width: 980px;
    clear: both;
    float: none
}
.area {
    margin: 15px 0;
    padding: 10px;
    border: 1px solid #999;
}
.player-video {
    float: left;
    width: 522px
}
.material-apoio {
    float: right;
    width: 388px;
    height: 418px
}
.botoes-controle, .acoes-usuario {
    clear: both;
}
```

Figura 15 – Estilos CSS das áreas definidas no escopo do projeto

5.5 DESENVOLVIMENTO DO CONTROLE CUSTOMIZADO

Além de botões responsáveis pelas funcionalidades básicas de um player de mídia, como, inicio, parada da transmissão e controle do volume, também foram incluídos botões para controlar a velocidade da transmissão e outro que possibilita o avanço e recuo através de saltos no tempo do vídeo.

Como o vídeo não sofreu efeitos de edição, disponibilizar a opção para controlar a velocidade torna-se importante no caso do usuário tiver necessidade de acompanhar a transmissão de forma lenta, por exemplo, em um determinado ritmo da música que exige maior grau de complexidade.

Definido que não existe necessidade da transmissão ser mais rápida que a padrão, não existe opção no controle de aumentar a velocidade. Dessa forma, também reduz o risco da taxa de dados exceder a largura da banda de conexão.

Conforme a figura 14, a primeira etapa para criação do controle foi incluir no documento HTML os botões, marcados pelo elemento *button*, além de outros elementos necessários para facilitar a comunicação ao usuário, como, informações sobre o volume atual e ponto atual que encontra a transmissão.

```
1  <div class="area botoes-controle">
2      <h3>Controle</h3>
3      <div class="controle">
4          <button id="play" title="iniciar vídeo">Play</button>
5          <span id="exibeTempo"></span>
6      </div>
7      <div class="controle">
8          <p>Volume: <span id="exibeVolume">1.00/1.00</span></p>
9          <button id="maisVolume" title="aumentar volume">+</button>
10         <button id="menosVolume" title="diminuir volume">-</button>
11         <button id="mudo" title="ligar e desligar som">Desligar som</button>
12     </div>
13     <div class="controle">
14         <button id="lento" title="diminuir velocidade">Diminuir velocidade</button>
15         <button id="vlNormal" title="velocidade normal">Velocidade Normal</button>
16         <button id="recuar" title="recuar 10 segundos">Recuar -10s</button>
17         <button id="avancar" title="avançar 10 segundos">Avançar +10s</button>
18     </div>
19 </div>
```

Figura 16 – Código HTML referente aos botões do controle customizado

Já com os objetos inclusos no documento a segunda etapa foi desenvolver as funções de cada elemento do controle através de programação JavaScript.

Na figura 15, até a linha 10 do código, é feita a declaração das variáveis com valor igualado ao DOM de cada objeto. A linha 12 refere-se à remoção do atributo *controls,* na tentativa de garantir que o controle nativo do navegador não seja exibido ao usuário. As linhas 14 a 16 representam a função responsável por exibir na tela o momento atual da reprodução do vídeo. O restante do código refere-se às funções de eventos aplicadas aos botões do controle.

Há apenas um botão para iniciar e parar o vídeo, que tem sua função alterada ao clicar no objeto. Dessa forma o mesmo recebe a função de iniciar ou parar a reprodução. Mas é importante ressaltar que no caso da reprodução ser interrompida, por motivo de falta de conteúdo pronto para ser exibido ao usuário, a função do botão de iniciar continua ativada e assim que tiver conteúdo disponível novamente, à transmissão continua sem necessidade de ação por parte do usuário.

```javascript
var video = document.getElementsByTagName("video")[0];
var exibeTempo = document.getElementById("tempo");
var botaoPlay = document.getElementById("play");
var botaoAvancar = document.getElementById("avancar");
var botaoRecuar = document.getElementById("recuar");
var botaoLento = document.getElementById("lento");
var botaoVlNormal = document.getElementById("vlNormal");
var botaoMudo = document.getElementById("mudo");
var exibeVolume = document.getElementById("exibeVolume");
var volume = video.volume;

video.removeAttribute("controls");

video.addEventListener('timeupdate',function(event){
    exibeTempo.innerHTML = parseInt(video.currentTime) + 's';
}, false);

botaoPlay.addEventListener('click', function() {
    if (video.paused || video.ended) {
        if (video.ended) { video.currentTime = 0; }
        this.innerHTML = "Pause";
        this.title = "pausar video";
        video.play();
    } else {
        this.innerHTML = "Play";
        this.title = "iniciar video";
        video.pause();
    }}, false);

botaoAvancar.addEventListener('click', function() {
    video.currentTime += 10;
}, false);
botaoRecuar.addEventListener('click', function() {
    video.currentTime -= 10;
}, false);

botaoLento.addEventListener("click", function () {
    video.playbackRate -= .20;
}, false);
botaoVlNormal.addEventListener("click", function () {
    video.playbackRate = 1;
}, false);

document.getElementById("menosVolume").addEventListener("click", function () {
    if (video.volume > 0.1) {
        video.volume -= 0.1;
        exibeVolume.innerHTML=video.volume.toFixed(1)+"/1.0";
    }
}, false);

document.getElementById("maisVolume").addEventListener("click", function () {
    if (video.volume < 1.0) {
        video.volume += 0.1;
        exibeVolume.innerHTML=video.volume.toFixed(1)+"/1.0";
    }
}, false);

document.getElementById("mudo").addEventListener("click", function () {
    if (video.muted) {
        video.muted = false;
        mudo.innerHTML="Desligar som";
    } else {
        video.muted = true;
        mudo.innerHTML="Liga som";
    }
},false);
```

Figura 17 – Código JavaScript referente as funções dos botões

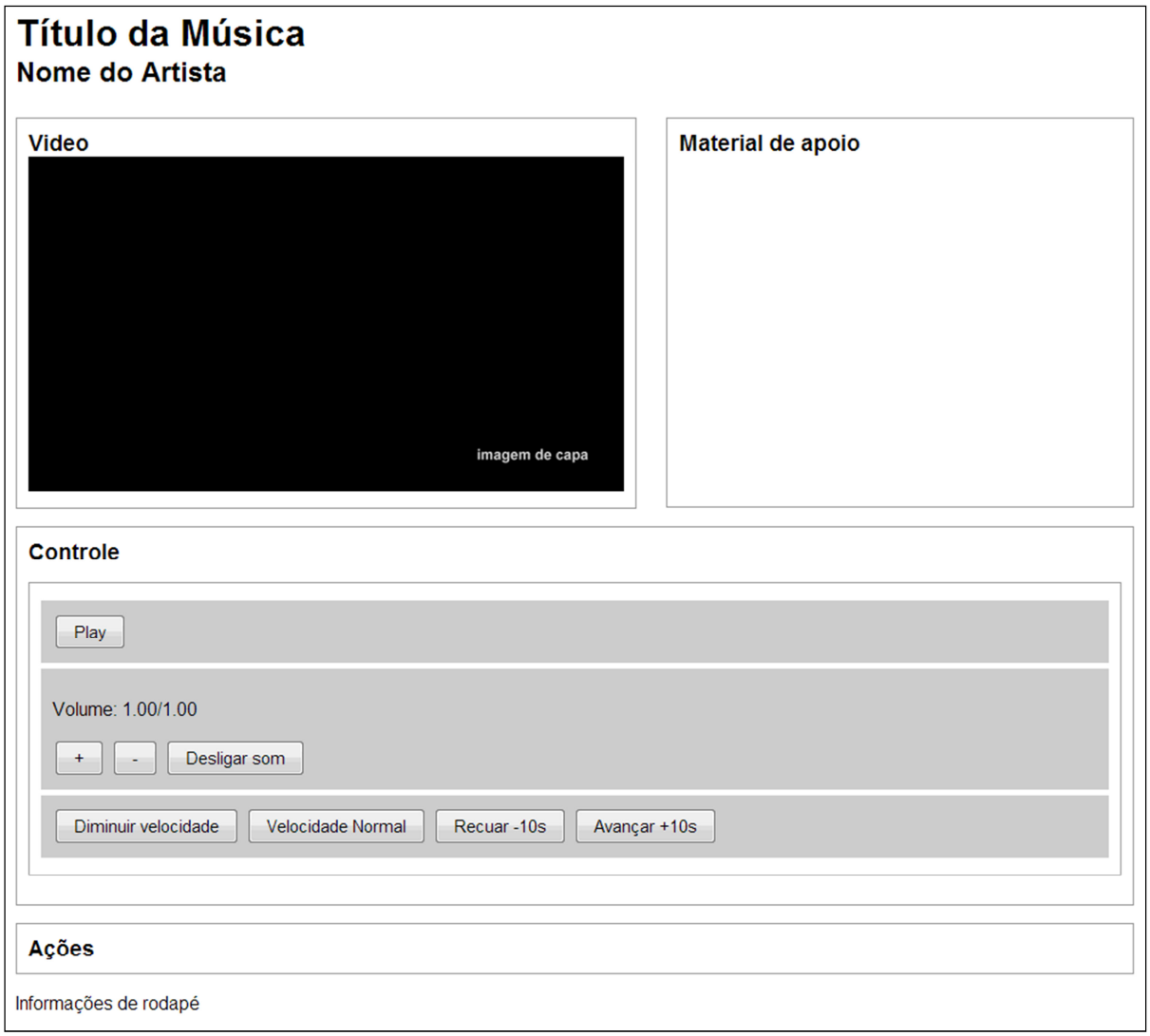

Figura 18 – Tela do navegador com mídia player e o controle customizado

5.6 CAPTAÇÃO DOS DADOS DAS AÇÕES DO USUÁRIO

Já com o player de mídia customizado pronto, foi possível registrar as ações geradas a partir do uso do controle.

Como pode ser visto na figura 18, agora a função do botão *play*, além de iniciar a reprodução do vídeo, passa também a gravar uma variável do tipo inteiro com o valor incrementando cada vez que o botão é clicado pelo usuário. Dessa forma, a mesma lógica poderia ser utilizada em todos os botões, com objetivo de fornecer esses dados de base para quem emite o vídeo. Assim o mesmo teria condições de identificar os pontos mais críticos da comunicação.

```
var qtdPlay = 0;
var nrPlay = document.getElementById("nrPlay");

botaoPlay.addEventListener('click', function() {
  if (video.paused || video.ended) {
    if (video.ended) {
      video.currentTime = 0;
    }
    this.innerHTML = "Pause";
    this.title = "pausar video";
    video.play();

    qtdPlay = qtdPlay + 1;
    nrPlay.value=qtdPlay;

  } else {
    this.innerHTML = "Play";
    this.title = "iniciar video";
    video.pause();
  }
}, false);
```

Figura 19 – Código JavaScript para gerar valor quando botão *play* é acionado

A única alteração que o documento HTML sofreu, foi o acréscimo de um formulário e um campo *input* do tipo *text* que recebe o valor atualizado cada vez que o botão *play* é acionando pelo usuário (Figura 20).

```
<div class="area acoes-usuario">
    <h3>Ações</h3>
    <form id="form1" name="form1" method="post" action="">
        <label for="nrPlay">Número de play</label>
        <input name="nrPlay" type="text" value="0" id="nrPlay" size="10" />
    </form>
</div>
```

Figura 20 – Marcação do campo *input* que recebe valores com ações do usuário

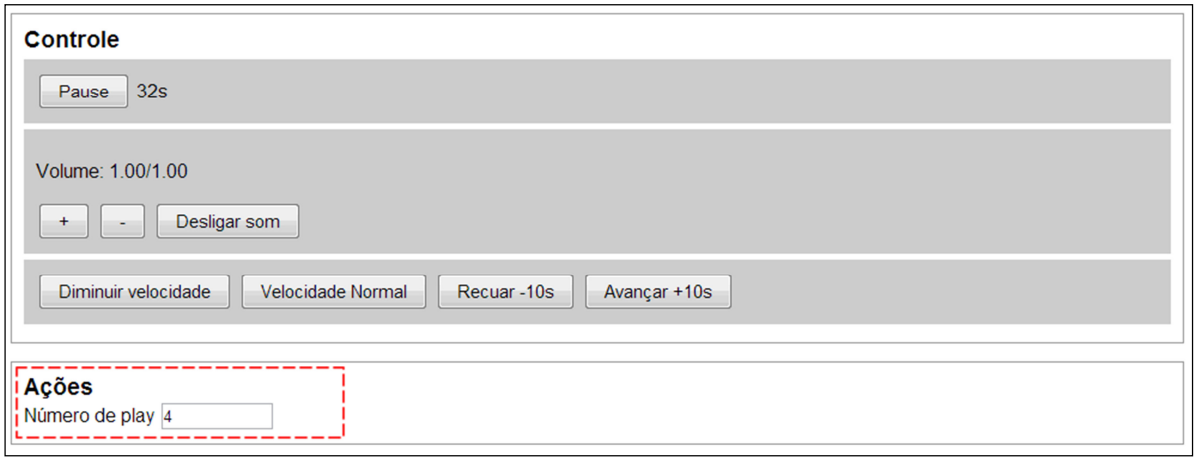

Figura 21 – Exibição da quantidade de vezes que o botão *play* foi acionado

5.7 SINCRONIZAÇÃO DOS OBJETOS COM O VÍDEO

Conforme as notas musicais são exibidas durante a reprodução do vídeo, imagens ilustrativas dos acordes tocados são mostrados ao usuário, servindo como um material de apoio para a vídeo aula. Inicialmente uma imagem, sem nenhuma informação, foi composta no documento HTML (Figura 22), pois o primeiro *frame* do vídeo não possui nenhum acorde.

```
1  <div class="area material-apoio">
2      <h3>Material de apoio</h3>
3      <img src="notas/vazio.png" id="imgNota" alt="Notas" />
4  </div>
```

Figura 22 – Marcação do objeto *img* sincronizado com o vídeo

Para ocorrer a alteração das imagens conforme o tempo do vídeo vai passando, foi aplicada uma função ao elemento *video*, capaz de verificar o momento atual exibido e assim substituir a imagem do material de apoio conforme as variações estipuladas na função (Figura 23).

Nas linhas de 3 a 9, da figura 23, possui a declaração das variáveis necessárias usadas na função. Uma variável chamada de “cifra”, com valor igualado ao elemento DOM da imagem que será substituída. Uma variável com nome “tempoAtual”, na qual possui o valor do tempo atual do vídeo exibido. E na sequencia do código mais quatro variáveis tendo como valor armazenado o endereço das imagens de cada nota musical. É importante observar que só foi possível armazenar na variável “tempoAtual” o último *frame* assistido pelo usuário através propriedade *currentTime*, compostas nas APIs de mídia. O restante da função, trata-se de uma estrutura de decisão simples onde a imagem do acorde é alterada conforme a variações estipuladas no tempo de reprodução do vídeo.

```
video.addEventListener('timeupdate',function(event) {

    var cifra = document.getElementById('imgNota');
    var tempoAtual = video.currentTime;

    var nota1 = "notas/c.png";
    var nota2 = "notas/g.png";
    var nota3 = "notas/em.png";
    var nota4 = "notas/d.png";

    if (tempoAtual >= 1 && tempoAtual < 2) {
        cifra.src = nota2;
    }
    else if (tempoAtual >= 2 && tempoAtual < 3) {
        cifra.src = nota4;
    }
    else if (tempoAtual >= 3 && tempoAtual < 4) {
        cifra.src = nota3;
    }
    else if (tempoAtual >= 4 && tempoAtual < 5) {
        cifra.src = nota1;
    }
    else if (tempoAtual >= 5 && tempoAtual < 7) {
        cifra.src = nota2;
    }
    else if (tempoAtual >= 7 && tempoAtual < 8) {
        cifra.src = nota4;
    }
    else if (tempoAtual >= 8 && tempoAtual < 9) {
        cifra.src = nota3;
    }
    else if (tempoAtual >= 9 && tempoAtual < 10) {
        cifra.src = nota1;
    }
    else if (tempoAtual >= 10 && tempoAtual < 11) {
        cifra.src = nota2;
    }
    else {
      cifra.src = "notas/vazio.png";
    }
}, false);
```

Figura 23 – Código JavaScript para alterar a imagem no tempo atual do vídeo

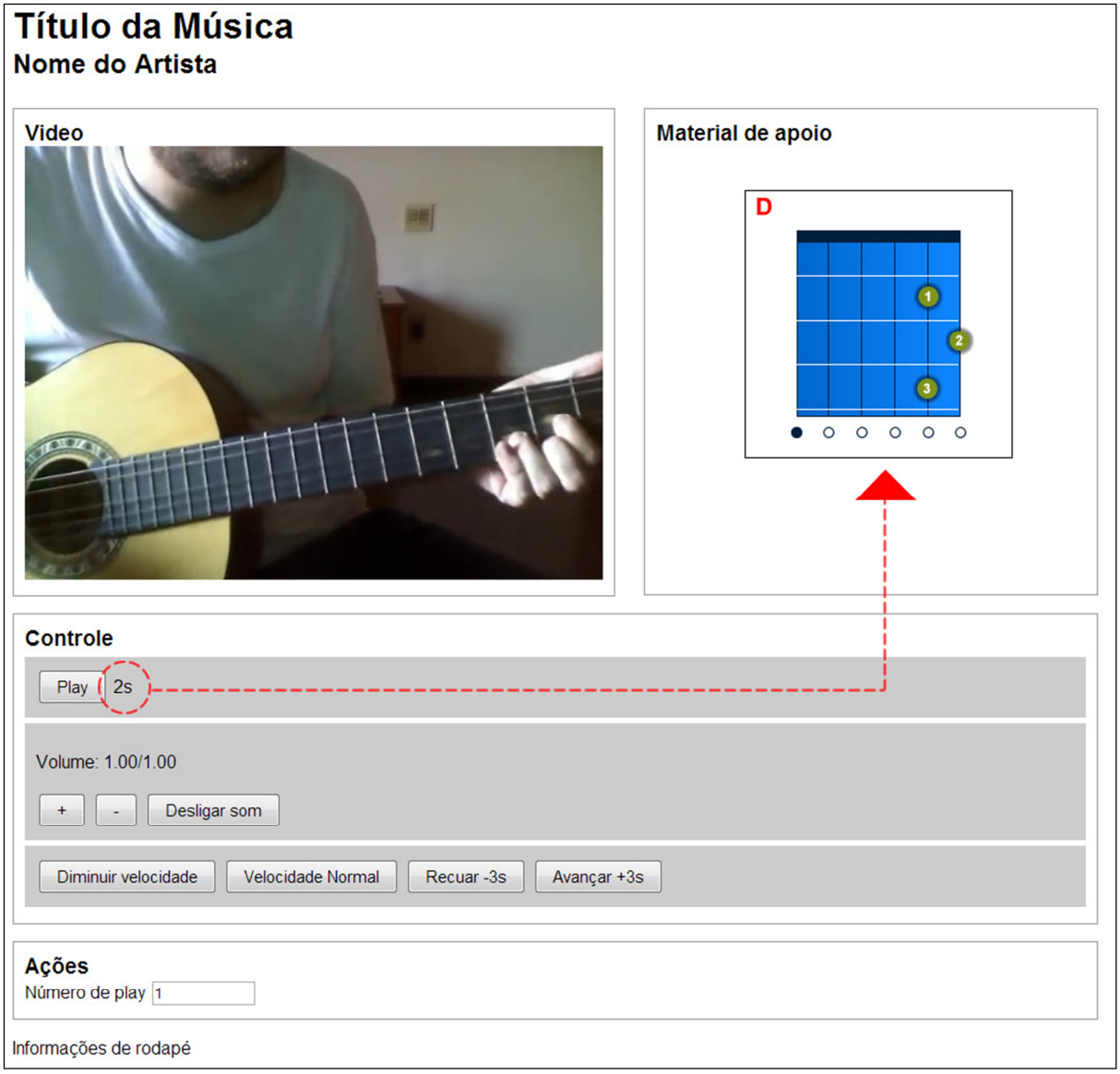

Figura 24 – Tela do navegador com elemento sincronizado ao vídeo

6. CONCLUSÃO

Este estudo abordou, através de pesquisas bibliográficas, aspectos relevantes sobre o elemento *video*, definido na versão cinco da linguagem HTML. No qual pode-se observar claramente a potencialidade dessa tecnologia, chegando a conclusão que ter especificado uma maneira padrão de servir conteúdos audiovisuais na internet, trás diversos benefícios imediatos para o usuário com relação às outras metodologias encontradas atualmente, como, velocidade, desempenho, acessibilidade e fácil manipulação do fluxo de informações.

Por outro lado por se tratar de uma tecnologia nova e pouco aplicada até o momento, existem alguns pontos que podem ser melhorados, principalmente em relação ao fato de não existir um *codec* e container específico e padronizado, onde os desenvolvedores possam contar, independente do ambiente de acesso do usuário.

Apesar de não ter sido aplicado nenhum teste de usabilidade no sistema proposto, o objetivo foi atingido. O resultado alcançado mostrou a praticidade de manipular o elemento *video* junto com a APIs de mídia. Além do fato de ser mais fácil gerenciar o conteúdo da aula usada de exemplo no sistema, como por exemplo, no caso de haver a necessidade de alterar uma determinada nota musical, não seria necessário filmar e editar novamente, apenas bastaria alterar a imagem que representa o acorde, pois as figuras não estão embaladas junto com o fluxo de informações do vídeo.

Mesmo que considerando prática, a aplicação e manipulação do elemento *video,* pode-se observar a diversidade de detalhes que cerca um sistema de reprodução de vídeos na internet. Sendo que alguns fatores, como arquitetura de distribuição, são determinantes para evitar possíveis falhas que impedem os usuários terem uma experiência contínua.

Para trabalho futuro, dado que a HTML5 é um padrão baseado na Web aberta e a proteção contra cópia é uma área não tratada por ela, pretende-se pesquisar sobre mecanismos de gerenciamento e proteção para vídeos com direitos autorais.

7. REFERENCIA BIBLIOGRÁFICA

Apple Inc. **HTTP Live Streaming**, 2011. Disponível em: <http://developer.apple.com/library/ios/#documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Introduction.html>. Acesso em 04 de junho de 2012.

AVILA, R. N. P. **Aprenda a Criar e Instalar sua Radio**. Ciência Moderna, 2008.

CISCO. **Tráfego global via Internet deve quadruplicar em 2015**, 2011. Disponível em: <http://globalnewsroom.cisco.com/easyir/BR/pt/local/press-release/Trafego-global-via-Internet-deve-quadruplicar-em-2015--763444.html> Acesso em: 16 de novembro de 2011.

DEVLIN, I. HTML5 **Multimedia: Develop and Design**. Peachpit Press, 2012.

FIORENTINI, L. M. R.; CARNEIRO, V. L. Q. (org.). **TV na escola e os desafios de hoje**: Curso de extensção para Professores do Ensino Fundamental e médio da Rede Pública. Unirede e Seed/Mec. BrasÌlia: Editora Universidade de Brasília, 2001. v.1, 2 e 3.

FOROUZAN, B. A. Comunicação **de Dados e Redes de Computadores**. 3 ed. Bookman, 2006.

HOGAN, B. P. **HTMl5 and CSS3 Develop with Tomorrow's Standards Today**. Pragmatic Bookshelf, 2010.

IBOPE. **31,8 milhões de pessoas navegaram em sites de vídeos em abril**, 2011. Disponível em: <http://www.ibope.com.br/calandraWeb/servlet/CalandraRedirect?temp=5&proj=PortalIBOPE&pub=T&db=caldb&comp=Noticias&docid=3F601F9B6DECCC398325789A004AE274>. Acesso em: 22 nov. 2011.

ISLAM, M. S. **A HTTP Streaming Video Server with Dynamic Advertisement Splicing**. 2010. Master of Science Thesis, Royal Institute of Technology (KTH) School of Information and Communication Technology, Stockholm - Sweden.

LAWSON, Bruce. Sharp, Remy. **Introducing HTML5**, 2. ed. New Riders, 2012.

LOBO, Adailton. **A utilização da videoconferência no ensino à distância, aplicada no Projeto Magister da UDESC**, 1999. Disponível em: <http://pages.udesc.br/~r4al/ti760.htm>. Acesso em: 22 novembro de 2013.

MORAN, José Manuel. **O vídeo na sala de aula. Comunicação & Educação**, São Paulo, 1995. Disponível em: <http://www.eca.usp.br/prof/moran/vidsal.htm>. Acesso em: 10 novembro de 2011.

OZER, Jan L. Video Compression for Flash, Apple Devices and HTML5. Doceo Publishing, 2011.

PFEIFFER, S. **The Definitive Guide to HTML5 Video**. Apress, 2010. (Definitive Guide Series).

PEREIRA, R. **Otimizando um Servidor Web para Download Progressivo**, 2008 Disponível em: <http://rafaelspereira.wordpress.com/2008/09/18/otimizando-um-servidor-web-para-download-progressivo/>. Acesso em: 10 de agosto de 2012.

PILGRIM, Mark. **Dive Into HTML5,** 2010. Disponível em: <http://diveintohtml5.info/index.html>. Acesso em: 30 de janeiro de 2011.

POWERS, S. **HTML5 Media**. O'Reilly, 2011.

REIS, Amanda. **Triplica o uso de internet móvel no Brasil**, 2012. Disponível em: < http://www.nic.br/imprensa/clipping/2012/midia806.htm>. Acesso em: 17 novembro de 2012.

SANTOS, A. **Ensino à Distância & Tecnologias de Informação – e-learning**. Lidel, 2000.

SARTORI, Ademilde; ROESLER, Jucimara. **Educação superior a distância: gestão da aprendizagem e da produção de materiais didáticos impressos e on-line.** Unisul, 2005.

SCHWABER, Ken. **Guia do Scrum**, 2009. Disponível em: <http://www.trainning.com.br/download/GUIA_DO_SCRUM.pdf>. Acesso em: 20 de setembro de 2012.

SOUSA, Jorge Pedro. **Elementos de teoria e pesquisa da comunicação e dos media**. 2. ed. Porto: Edições Universidade Fernando Pessoa, 2006.

W3C **Recommendation. 13 Objects, Images, and Applets**. Disponível em: <http://www.w3.org/TR/html401/struct/objects.html>. Acesso em: 23 novembro de 2011.

W3C Working Draft. **HTML5**. Disponível em: < http://www.w3.org/TR/html5/the-iframe-element.html#the-video-element>. Acesso em: 23 nov. 2011.

WATSON Lucas. **Channels of the Future**. In: MONACO MEDIA FÓRUM, 2011, Monaco. Disponível em: <http://www.youtube.com/watch?v=163ARNrJpko#!>. Acesso em: 21 nov. 2011.

WILCOX, M. **Introducing HTML5 video**, 2010. Disponível em: <http://www.ibm.com/developerworks/web/library/wa-html5video/>. Acesso em: 01 de julho de 2012.

WIJERING, Jeroen**. What is Video Streaming?**, 2011. Disponível em <http://www.longtailvideo.com/blog/19578/what-is-video-streaming>. Acesso em 03 de junho de 2012.