Matheus Eduardo B. Morais e Bruno Rubin

Suporte a Codificação e Decodificação do WebP no Mozilla Firefox

Porto Alegre

2013/06/17

Matheus Eduardo B. Morais e Bruno Rubin

Suporte a Codificação e Decodificação do WebP no Mozilla Firefox

Orientador:
Isabel H. Manssour

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL (PUCRS)

Porto Alegre

2013/06/17

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

O desempenho de técnicas de codificação multimidia está melhorando constantemente em termos de taxa de compressão, funcionalidades de código, algoritmos, e robustez contra erros mesmo em um ritmo mais lento em relação ao que estávamos acostumados até há uma década atrás. Um dos últimos *codecs* que é esperado para melhorar este cenário é o algoritmo WebP lançado recentemente pela empresa Google, cujo objetivo principal é fornecer imagens com uma maior taxa de compressão, mas sem perda de qualidade.

Naturalmente, o novo formato se torna pouco utilizado até que todos os navegadores dêem suporte. Dado o cenário atual, este Trabalho de Conclusão propõe disseminar a utilização do *codec* WebP através da implementação do codificador e decodificador deste formato no navegador Mozilla Firefox.

Abstract

Performance of multimedia coding techniques are constantly improving in terms of compression ratio, code features, algorithms, and robustness against errors even at a slower pace compared to what we were used to a decade ago. One of the last codecs that is expected to improve this scenario is the algorithm WebP recently released by Google company, whose main goal is to provide images with higher compression ratio but without loss quality.

Naturally, the format becomes rarely used until all browsers give support. Given the current scenario, this paper provides to spread the use of codec WebP through the implementation of encoder and decoder of this format in Mozilla Firefox browser.

1 Introdução

1.1 Contextualização

Uma tecnologia importante da comunicação utilizada nos dias de hoje dentro da internet é a chamada WWW, ou *World Wide Web*, na qual usuários acessam páginas da *Web* escritas em uma linguagem específica definida por uma organização internacional (W3C), que é renderizada através de um software chamado de navegador. O usuário, então, pode ler as informações descritas ali, e acessar os demais conteúdos da página através de seus *hiperlinks* de uma maneira fácil e intuitiva.

Com a crescente evolução dos meios de comunicação, a necessidade de total interatividade do usuário com as páginas aumenta cada vez mais. Uma página da *Web* não é somente texto como era nos primórdios, ela é composta de uma série de diferentes atores, tais como folhas de estilo, imagens e vídeos. Estes atores existem para aumentar a facilidade no uso e proporcionar uma experiência mais agradável para o usuário final.

Apesar da mudança na necessidade dos usuários da *Web*, poucas alteração foram feitas no protocolo HTTP, cuja última revisão (1.1) ocorreu no ano de 1999, e na linguagem HTML que sofreu seu último ajuste em maio de 2000. Essencialmente a tecnologia implementada atualmente é a mesma criada no início dos anos 90 e apesar de sofrerem pequenas revisões pós criação, continuam não atendendo totalmente as demandas que surgem. Um fator que comprova este fato é a diversidade de tecnologias paralelas (*plugins*) não oficiais que foram criadas para preencher estas lacunas. O *Flash*, o *JavaFX* e o *Silverlight* são algumas das tecnologias que ilustram este cenário. Outro fator importante é que mesmo com o avanço das tecnologias de comunicação, nem todas as pessoas têm acesso a esta evolução. Seja por uma questão econômica ou mesmo pela carência de infraestrutura. Existe atualmente o desenvolvimento da quinta versão para a linguagem HTML, que apesar de ainda não possuir uma versão final, já tem uma grande popularidade na Internet, e conta com o suporte de todos os principais navegadores (1).

Recentemente a empresa estadunidense *Google Inc.* promoveu uma série de iniciativas em conjunto com a comunidade internacional no esforço chamado de "*Let's make the Web*

Faster"(2), que consiste em um processo de reengenharia dos protocolos, linguagens e demais itens que compõem a *Web*. Dentro deste processo novos padrões e formatos estão sendo sugeridos para melhorar a performance e a qualidade na experiência do usuário as páginas. Uma das sugestões é a utilização de um novo método para compressão de imagens chamado de *WebP*, criado para ser utilizado principalmente na *Web*. Este formato é capaz de comprimir aproximadamente 39% mais que a melhor solução utilizada hoje que é o formato JPEG, e apresentar imagens com o mesmo nível de qualidade visual (3). O suporte à renderização de imagens *WebP* está implementado atualmente apenas nos navegadores Chrome e Opera.

Os estudos mais recentes mostram que a capacidade de compressão do formato *WebP* tem cerca de 39% de ganho quando comparado com JPEG e JPEG200, como pode-se observar na figura 1.1:

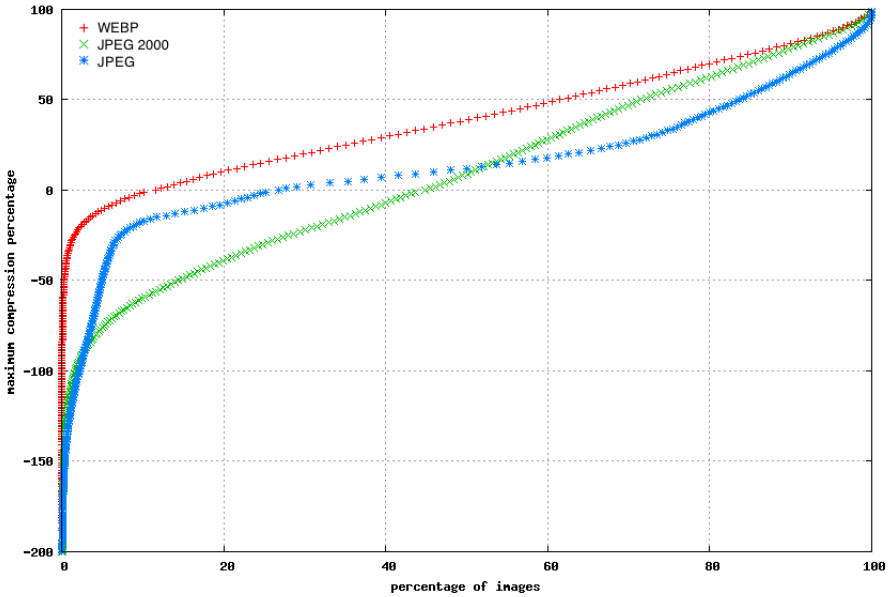

Figura 1.1: Comparação do tamanho das imagens e porcentagem de compressão (3)

Embora a implementação do suporte ao *WebP* esteja disponível somente para dois navegadores (Chrome e Opera), estes constituem um percentual de uso de cerca de 42,38% de todos os navegadores utilizados dentro da internet nos dias de hoje. Se houvesse suporte ao *WebP* dentro do Mozilla Firefox este percentual aumentaria para 62,14%, já que cerca de 19,76% dos navegadores utilizados hoje são Mozilla Firefox (4), como ilustra a Figura 1.2.

Apesar de não existir nenhuma diretiva da W3C para qual método de compressão de imagens utilizar por padrão, e não se sabe se haverá tal tipo de registro neste sentido no futuro, é importante ressaltar que quanto mais navegadores suportarem a compressão usando o formato *WebP*, mais os desenvolvedores vão optar por utilizar este formato dada suas vantagens. Uma força de aproximadamente 63% de suporte dentro dos navegadores da internet também fará com que aqueles que ainda não decidiram pela implementação do suporte ao novo formato, também o façam em virtude da pressão exercida pela maioria.

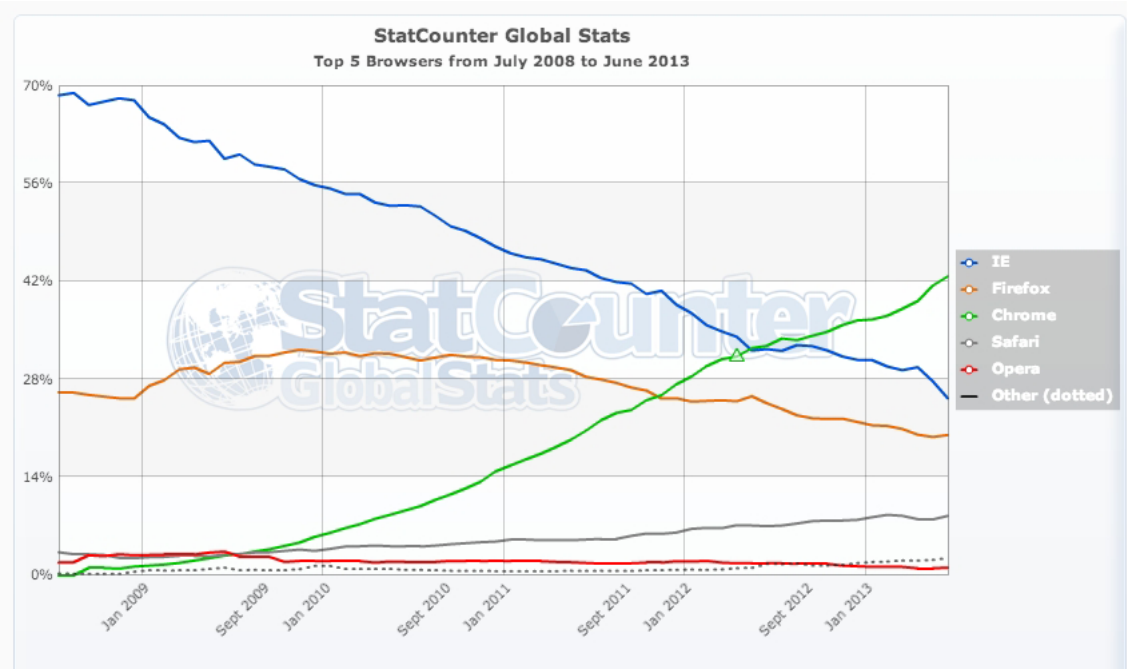

Figura 1.2: Percentual de utilização dos Navegadores na Web (4)

1.2 Motivação

A necessidade pela busca da informação cresceu exponencialmente de alguns anos pra cá. O crescente avanço tecnológico, trazendo consigo a popularização dos smartphones e a facilidade no acesso à informação, contribuiu de forma expressiva para que cada vez mais as pessoas estejam conectadas em todos os lugares. Da mesma forma que a disponibilidade dessas informações e o acesso à elas deve ser feito de forma cada vez mais rápida.

O desenvolvimento do codificador e decodificador do formato *WebP* no *Mozilla Firefox*, trará de imediato a melhora no tempo de resposta na exibição das imagens dentro do navegador, visto que o tamanho dos arquivos no servidor será reduzido. Se considerar que mais de 60% do tamanho total de um *website* são imagens, os benefícios podem ser substanciais. É importante ressaltar que a utilização em massa deste novo formato contribui, consequentemente, na diminuição do trafego de rede na internet de maneira global, isso resulta diretamente em maior vazão para outros recursos que possuem um maior consumo de banda, como é o caso de um *streaming* de vídeo por exemplo.

Um *case* de utilização do novo formato, é o da própria *Google*, que substituiu as imagens, de formato PNGs e JPEGs para WebP, em sua *Web Store* do navegador Chrome, diminuindo a utilização de banda dirária, de acordo com o Gerente de Produto do Google, Stephen Konig (5):

> *By converting PNGs and JPEGs to WebP, the Chrome Web Store was able to reduce image sizes by about 30% on average. Given the number of requests Chrome Web Store serves, this adds up to several terabytes of savings every*

day.

1.3 Objetivo

O objetivo principal deste trabalho é desenvolver uma solução que habilite o suporte de codificação e decodificação do formato WebP dentro do Mozilla Firefox.

A implementação do decodificador irá permitir ao usuário acessar e exibir, através do navegador, qualquer imagem que estiver no formato WebP. Já o codificador irá possibilitar que *sites* desenvolvidos utilizando tecnologia HTML5, e que possuam a funcionalidade *canvas*, possam gerar gráficos no formato WebP através do próprio navegador, sem que seja necessário a utilização de qualquer software adicional (6).

Embora a implementação do suporte ao *WebP* esteja disponível somente para dois navegadores (Chrome e Opera), estes constituem um percentual de uso de cerca de 42,38% de todos os navegadores utilizados dentro da internet nos dias de hoje. Se houvesse suporte ao *WebP* dentro do Mozilla Firefox este percentual aumentaria para 62,14%, já que cerca de 19,76% dos navegadores utilizados hoje são Mozilla Firefox (4).

O trabalho desenvolvido foi entegue à comunidade do Firefox, mas sua aplicação oficial na árvore da Mozilla não está no foco do projeto. O objetivo deste trabalho não visa a aceitação por parte da equipe do Mozilla, em integrar a implementação desenvolvida ao código fonte do repositório oficial do Firefox, mas somente em contribuir com o desenvolvimento da renderização nativa do formato WebP dentro do navegador.

1.4 Organização do Texto

Será o última coisa a ser escrita, pois depende da finalização dos demais capítulos.

2 *Fundamentação Teórica*

2.1 A World Wide Web

A internet hoje em dia se fundamenta nos princípios da *World Wide Web* ou somente *Web* como é conhecida. Os usuários se conectam na rede mundial de computadores e utilizam as páginas da *Web* para realizar todo e qualquer tipo de ação dentro da rede, desde conversar com seus amigos e familiares até realizar compras em uma página de *e-commerce*.

A *World Wide Web* foi inventada no CERN (Centro Europeu de Pesquisa Nuclear), com principal objetivo de servir como suporte para a consulta e comunicação interna dos projetos de pesquisa (7). A ideia inicial era utilizar hipertextos, os quais funcionariam também como *hiperlinks* e fariam ligações para demais páginas contendo outro tipo de informação. Toda plataforma desenvolvida internamente dentro do CERN para a *Web* foi posteriormente disponibilizada ao domínio público, sem cobrança de royalties sob qualquer parte. Mais tarde Tim Berners-Lee, o fundador da *Web* dentro do CERN, criou a o *World Wide Web Consortium* com o intuito de definir padrões de utilização e interoperabilidade entre tecnologias dentro da rede. Conhecido também como W3C, é composto por funcionários de diversas empresas que trabalham em período integral para manter e evoluir os padrões especificados pela organização (8).

Além de ser utilizada por pessoas, a *Web* também é utilizada para comunicação entre sistemas. Efetivamente são computadores acessando outros computadores para realizarem a troca de informações, podendo ser em tempo real ou não. Como os padrões definidos tem suas especificações abertas, a *Web* também se torna uma ótima ferramenta para integrar sistemas diferentes em plataformas diferentes. Hoje em dia os chamados *Web-Services* são utilizados juntamente com os conceitos de Orientação a Objetos para integração entre sistemas. A comunicação utilizando os *Web-Services* se baseia nos princípios e padrões especificados pelo W3C para a *Web* (9).

Uma parte importante da *Web* são os chamados *Web Browsers* ou Navegadores que servem para que as páginas da *Web* possam ser acessadas e operadas. Os navegadores são ferramentas que interpretam os padrões descritos pela W3C e fazem a interface para que os humanos

consigam entender as informações descritas nas páginas e consigam navegar por elas de forma intuitiva, utilizando os *hiperlinks*.

Com o advento da qualidade na comunicação da internet, as páginas da *Web* se tornaram cada vez mais elaboradas e interativas. Atualmente elas não são mais compostas apenas de hipertextos com *hiperlinks*, esses componentes simples deram espaço para um conjunto de maior complexidade que inclui o processamento de imagens, vídeos e sons. Embora estas funcionalidades dinâmicas existam, não há um padrão claro a respeito delas. A HTML, que é a linguagem de programação da Web, não oferece suporte nativo para todas estas implementações, e plugins adicionais não padronizados são utilizados para implementar estas necessidades específicas (o Flash Player e o Java FX são um exemplo).

A W3C trabalha em uma evolução da linguagem HTML mais preparada para este tipo de cenário, a chamada HTML5. Dentro da HTML5 existem vários detalhes em discussão e um dos mais importantes é a questão da forma como vídeo e imagem serão interpretados. O valor 5 da HTML vem por essa ser a quinta revisão completa da linguagem, que promete ser uma das maiores revisões feitas até hoje (10).

2.2 Navegadores

Com a explosão do uso da *World Wide Web* na década de 90 (11), diversas grandes empresas da computação, como Microsoft, Google e Apple, investiram no desenvolvimento de novos softwares que possibilitassem aos usuários a busca de dados e informações que estivessem publicadas na Internet. Na computação, um navegador, também chamado de *Web Browser*, é um *software* com uma interface gráfica que possui como principal funcionalidade a apresentação de documentos virtuais da Internet, popularmente chamado de páginas da *Web*.

O navegador é, talvez, o software mais amplamente utilizado na história e teve uma evolução significativa nos últimos quinze anos. Hoje em dia, usuários dos mais variados níveis, utilizam navegadores em diversos tipos de *hardware*, desde celulares e *tablets* até computadores comuns (12).

De acordo com as estatísticas de mercado apresentadas na Tabela 2.1, os navegadores mais utilizados mundialmente são: Opera (Opera Software), Safari (Apple), Firefox (Mozilla Corporation and Mozilla Foundation), Internet Explorer (Microsoft) e Chrome (Google), em ordem crescente de popularidade (4).

A principal funcionalidade de um navegador é apresentar ao usuário o recurso da *Web* que ele escolher. Este recurso geralmente é um documento em formato HTML, podendo ser também uma imagem, um arquivo PDF ou outro formato. A localização do recurso é especificada pelo usuário através de uma URL (*Uniform Resource Locator*), a qual é traduzida pelo navegador

Tabela 2.1: Estatísticas de utilização dos Navegadores por mês (4).

2013	**Chrome**	**Internet Explorer**	**Firefox**	**Safari**	**Opera**
Maio	41.38%	27.72%	19.76%	7.96%	1%
Abril	39.15%	29.71%	20.06%	8%	1.01%
Março	38.07%	29.3%	20.87%	8.5%	1.17%
Fevereiro	37.09%	29.82%	21.34%	8.6%	1.22%
Janeiro	36.52%	30.71%	21.42%	8.29%	1.19%
2012	**Chrome**	**Internet Explorer**	**Firefox**	**Safari**	**Opera**
Dezembro	36.42%	30.78%	21.89%	7.92%	1.26%
Novembro	35.72%	31.23%	22.37%	7.83%	1.39%
Outubro	34.77%	32.08%	22.32%	7.81%	1.63%
Setembro	34.21%	32.7%	22.4%	7.7%	1.61%
Agosto	33.59%	32.85%	22.85%	7.39%	1.63%
Julho	33.81%	32.04%	23.73%	7.12%	1.72%
Junho	32.76%	32.31%	24.56%	7%	1.77%

e utilizada para realizar a solicitação das informações para um servidor web e posteriormente, exibi-las em sua janela (12). Um exemplo deste fluxo é apresentado na Figura 2.1.

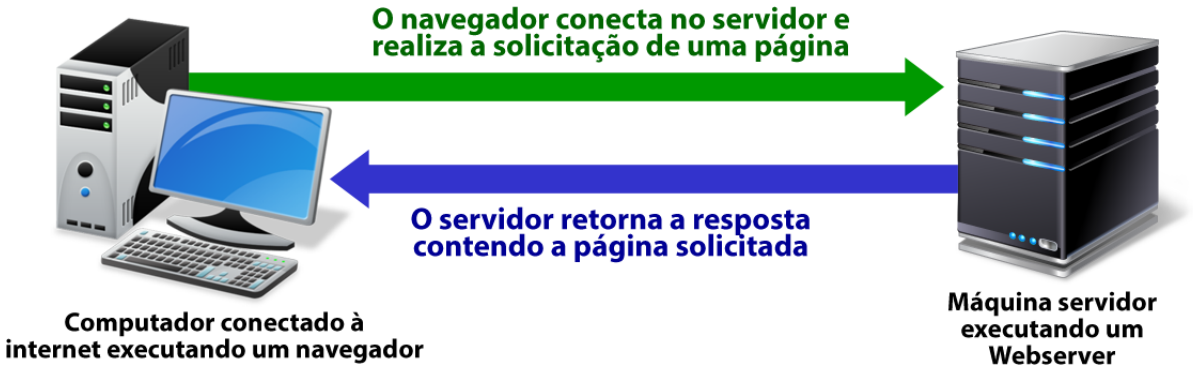

Figura 2.1: Fluxo básico de requisição de uma página da *Web* para um servidor.

A forma com que o navegador interpreta e exibe arquivos HTML (*HiperText Markup Language*) e folhas de estilo CSS (*Cascading Style Sheet*) são especificações mantidas pela W3C.

A interface gráfica de um navegador possui diversos elementos que são comuns, independente do software que esteja sendo utilizado. Entre estes elementos, pode-se citar:

- Barra de endereços para digitação de URL;
- Botão de Home para levar o usuário para a página inicial;
- Botões de Voltar e Avançar para as páginas que já foram visualizadas;
- Botão de Favoritos para guardar URLs de interesse do usuário.

- Botão de Atualizar e Parar, para atualizar e cancelar, respectivamente, o carregamento de páginas.

A concorrência existente entre as empresas e os produtos oferecidos atualmente, se dá principalmente pela velocidade de navegação e a utilização dos recursos computacionais. A cada dia, surgem novos padrões e estudos de novas tecnologias relacionadas, que possibilitam que os navegadores tenham um melhor desempenho na execução das suas funções. Um exemplo é o HTML5, que tem como objetivo tornar o conteúdo das páginas mais dinâmico, deixando de lado o conceito do navegador que apenas realiza a apresentação do documento, mas que interaja como uma aplicação única através dos recursos e ferramentas disponibilizadas (13).

2.3 HTML5

Em 1998 o W3C decidiu que eles não iriam mais continuar o desenvolvimento da linguagem da *Web* até então, a HTML *(HyperText Markup Language)*. Ao invés disto eles iriam trabalhar em um novo formato o qual acreditavam ser o futuro da *Web*, assim nasceu a linguagem XHTML *(Extensible HyperText Markup Language)* que se fundamentava nos princípios da XML *(Extensible Markup Language)*. Desta forma a revisão 4.01 da HTML foi congelada, e os esforços se direcionaram para a evolução da XHTML que em sua versão 2.0 trazia além de muitas mudanças revolucionárias, a quebra de toda compatibilidade das aplicações escritas nas versões anteriores da especificação.

Muitos não gostaram desta nova iniciativa, em particular um grupo que trabalhava no desenvolvimento do navegador Opera. Estes indivíduos não se convenceram que o XML seria o futuro, e além de não adotar a XHTML, começaram a trabalhar na especificação de uma linguagem HTML extensiva que não quebrasse a compatibilidade das versões anteriores. Este grupo é conhecido como WHATWG (*Web Hypertext Application Technology Working Group*). A especificação foi inicialmente chamada de *Web Forms* 2.0 e mais tarde foi nomeada para HTML5 por ser a quinta revisão da HTML (14). O W3C, então, adotou a versão do grupo WHATWG como base para a nova versão do HTML e começou a trabalhar na quinta revisão, e apesar de não ter sido lançada nenhuma versão final da especificação, praticamente todos navegadores já implementam o suporte à nova definição da HTML. Este processo de desenvolvimento da nova versão passou a ser desenvolvida simultaneamente, pelo W3C e pelo grupo WHATWG.

Por trás do HTML5 há uma série de princípios de design definidos. De uma forma resumida, pode-se citar três princípios principais (14):

- Interoperabilidade entre os navegadores: Possuir o mesmo comportamento em todos os navegadores, independente de plataforma e tecnologia;

- Definição de tratamento de erros: Maneira que os navegadores devem reagir à marcações inválidas;
- Evoluir a linguagem para facilitar a criação de conteúdo *Web*.

O HTML5 possui como paradigma ser livre de *plugins*, que são considerados atualmente como os vilões dos navegadores. Os *plugins*, apesar de adicionar novos elementos e componentes ao navegador, trazem consigo uma série de problemas, tais como:

- Nem sempre podem ser instalados;
- Podem ser desativados ou bloqueados;
- Comprometem a segurança e são alvos de ataques;
- São difíceis de integrar ao resto do conteúdo HTML.

Os *plugins* muitas vezes também apresentam dificuldades de integrar seus recursos com o resto do conteúdo exibido pelo navegador, o que acarreta em problemas na renderização do documento. Isto ocorre porque os plugins utilizam um modelo de processamento à parte do qual é utilizado para a página *Web*. Neste ponto que o HTML5 propõe melhorias, trazendo estes recursos de forma nativa na linguagem. Ao invés de utilizar elementos de estilo no formato CSS e *scripts* com JavaScript, o HTML5 apresenta recursos que não existiam nas versões anteriores. Não são apenas novos elementos que fornecem novas funcionalidades, mas a interação nativa com *scripts* e estilos que possibilita fazer muito mais do que antes(14).

Uma das necessidades que a HTML5 deve suprir é o suporte para processamento de vídeo e som que nos dias de hoje se fundamenta no *plugin Adobe Flash*. Para esta implementação foi escolhido o formato WebM que possibilita aos navegadores renderizar nativamente vídeo e áudio sem a necessidade de instalações adicionais. O formato WebM é composto pelo método de compressão de vídeo VP8 e o padrão de áudio Vorbis. O WebP é parte integrante do WebM, porém, com foco voltado para compressão de imagens.

3 *WebP*

3.1 Introdução

O WebP é um novo formato de imagem que implementa compressão *lossless* e *lossy* para imagens na *Web*. Na computação a compressão *lossless* é aquela que permite a exata reconstrução de uma informação original a partir da informação comprimida, já a compressão *lossy* é aquela na qual porções da informação original são descartadas (perdidas) para que haja efetivamente a compressão. Desta forma, a compressão *lossy* é utilizada para situações onde a integridade da informação não é exatamente necessária, como é o caso de imagens, vídeo e áudio. Utilizando uma técnica de compressão *lossless* o WebP consegue apresentar imagens 26% menores no tamanho quando comparado com o formato PNG. Utilizando imagens com *lossy* os resultados são ainda melhores, reduz de 25 a 34% o tamanho da imagem quando comparado com o formato JPEG. Ele também fornece suporte à transparência com apenas 22% de *bytes* adicionais. O WebP utiliza uma metodologia semelhante aquela usada na compressão VP8, que faz parte do pacote WebM para o processamento de vídeo e áudio (15).

A tecnologia por trás do WebP é fundamentada no conceito de *block prediction*. Cada bloco tem seu valor previsto baseando-se nos três blocos acima dele e no primeiro bloco a esquerda dele. Existem quatro métodos para previsão dos blocos, a horizontal, a vertical, a DC e a *TrueMotion*. As informações que foram previstas incorretamente e os blocos que não são previsíveis são comprimidos em um sub-bloco de 4x4 *pixels*. No final o resultado é comprimido utilizando *Entropy encoding*, que é um método *lossless* amplamente usado para compressão de dados.

O suporte à codificação e decodificação do WebP está disponível nos navegadores Google Chrome e Opera. Como o Mozilla Firefox já possui suporte ao WebM, alguma parte do código para a implementação do WebP já reside dentro do navegador uma vez que tanto WebP quanto VP8 utilizam a mesma técnica para a compressão de imagens pois, no caso de vídeo, o mesmo conceito é utilizado para compressão dos *frames*.

3.2 Mozilla Firefox e o WebP

O formato WebP foi disponibilizado em 2010 mas nunca conseguiu ser integrado ao Mozilla Firefox. Inclusive na época um *bug* foi aberto dentro do *bugzilla* da Mozilla solicitando a implementação do formato WebP (16). Naquele tempo a equipe principal de desenvolvimento do Mozilla Firefox decidiu por não realizar a implementação em virtude de uma série de considerações contrárias ao formato WebP (17). O time de desenvolvimento do WebP levou em conta as críticas que recebeu dos desenvolvedores do Mozilla Firefox e fez uma série de ajustes para suprir as necessidades que estavam sendo demandadas. Embora praticamente todos os apontamentos dados tenham sido ajustados, ainda não há uma clara razão do por que ele ainda não foi implementado. Em 2010 um *patch* rudimentar foi escrito e submetido por um membro da comunidade, mas não foi aceito. Existem também algumas extensões que fazem com que o Firefox consiga exibir imagens WebP, porém estas extensões não caracterizam um desenvolvimento adequado visto que elas dependem de *plugins* adicionais. A proposta é que o navegador seja capaz de realizar a renderização nativamente.

Pelo que se pode averiguar, parece que há por trás disso uma razão política na não adoção do formato WebP até o presente momento. Apesar do Google ter aumentado os investimentos no desenvolvimento do Mozilla Firefox em 2011, o Google Chrome vem tomando espaço do Firefox na internet e há um temor de que a própria existência do Firefox esteja ameaçada graças ao grande sucesso do navegador concorrente. Entretanto pelo que foi pesquisado dentro da comunidade há alguma chance de que caso uma boa implementação seja feita, ela poderá ser aceita dentro do código oficial.

3.3 Especificação do Container WebP

Como um formato de imagem, o WebP utiliza um *container* RIFF (*Resource Interchange File Format*) para armazenamento das informações necessárias para utilização do formato dentro de um arquivo. O *container* do WebP também oferece como característica o suporte para a compressão *lossless* e o efeito de transparência que pode ser obtido através de um canal *alpha*. De forma geral o arquivo WebP é composto da imagem representada por uma matriz de *pixels* no formato VP8 e opcionalmente a informação em relação a transparência.

Basicamente, existem três tipos de arquivos: o formato simples com compressão *lossy*, o formato simples com a compressão *lossless* e o formato estendido. Em todos os três tipos existe o cabeçalho do arquivo, que é composto inicialmente pelos caracteres 'RIFF' em ASCII, seguido da informação em relação ao tamanho do arquivo em *bytes* e por último os caracteres 'WEBP' em ASCII.

A figura 3.1 ilustra o formato simples com compressão *lossy*, o formato simples com a compressão *lossless* e o formato estendido, respectivamente.

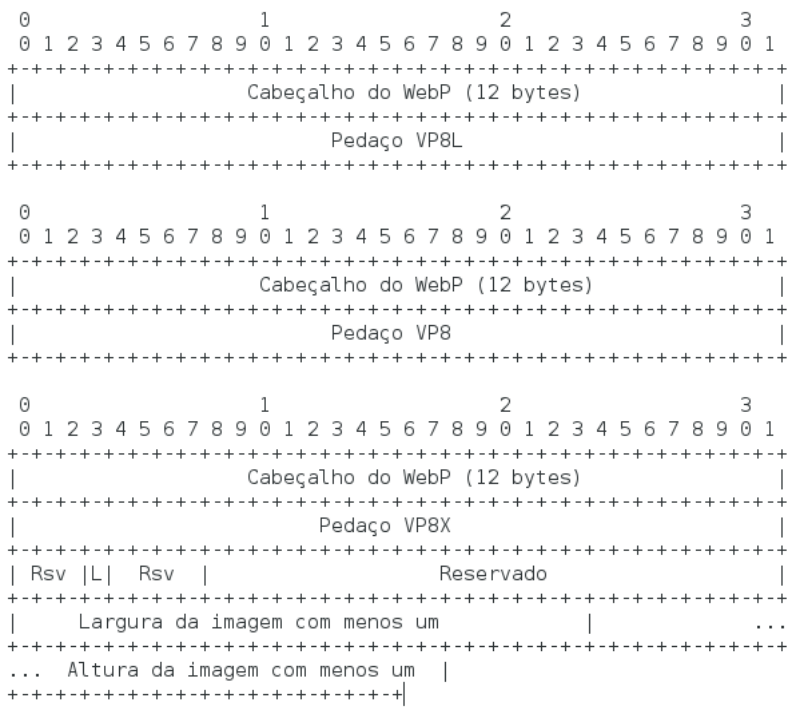

Figura 3.1: Formato simples com compressão *lossy*, formato simples com a compressão *lossless* e formato estendido

A diferença nos formatos simples com compressão *lossy* e *lossless* está no pedaço VP8 que por padrão faz a compactação com perda de dados, enquanto que o pedaço VP8L utiliza a compactação sem perda. Já o formato estendido é composto por um pedaço VP8X, mais a informação do canal *alpha* (L) e o *bitstream* VP8 ou VP8L. Caso o *bit alpha* seja setado com valor 1 no formato estendido, será acrescentado um pedaço adicional para a transparência, como mostra a figura 3.2.

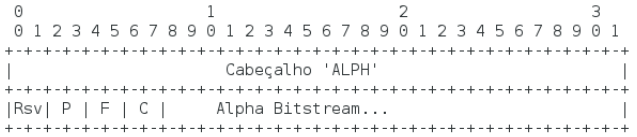

Figura 3.2: Pedaço que é acrescentado quando o *bit alpha* é definido para 1

3.4 A API libwebp

A equipe de desenvolvimento do WebP mantém uma biblioteca chamada libwebp que oferece suporte para a criação de ferramentas para a manipulação do formato. No que tange a

este trabalho, foram utilizadas os cabeçalhos para codificação e decodificação (encode.h e decode.h). Também foi necessário utilizar a biblioteca que define os tipos básicos de variáveis e estruturas definidos pelo formato WebP (type.h).

De acordo com a avaliação feita no código fonte e nas definições da biblioteca, as seguintes funções foram utilizadas para criação do decodificador:

- WebPIDecGetRGB
- WebPGetDecoderVersion
- WebPGetInfo
- WebPDecodeRGB
- WebPDecodeRGBA
- WebPDecodeARGB
- WebPDecodeBGR
- WebPDecodeBGRA

As funções utilizadas para criação do codificador foram as seguintes:

- WebPMemoryWrite
- WebPGetEncoderVersion
- WebPEncodeRGB
- WebPEncodeRGBA
- WebPEncodeBGR
- WebPEncodeBGRA

A documentação da biblioteca libwebp é bem simples e direta mas oferece diversos exemplos de utilização, o que a torna bem didática. Nos exemplos criados para este trabalho até o momento, não houve qualquer dificuldade com a utilização das funções e procedimentos definidos pela biblioteca. Os exemplos trabalhados foram a criação de decodificadores e codificadores rudimentares escritos em C++ para o formato WebP com o objetivo de entender melhor como funciona internamente a API. A utilização dela foi peça fundamental para implementação do projeto. A versão da libwebp utilizada dentro do trabalho foi a 0.3.0.

3.5 Estudo de caso do WebP

Com o objetivo de apresentar os ganhos de compressão do formato de imagem WebP em relação aos formatos JPEG e PNG, é mostrado a seguir uma série de 3 pares de imagens diferentes, uma em seu formato original e a outra já convertida para o WebP, os detalhes com relação ao tamanho final do arquivo e os recursos utilizados em cada um dos casos. As imagens foram retiradas aleatoriamente da internet e convertidas utilizando o conversor padrão fornecido juntamente com a libwebp.

O primeiro caso é de uma paisagem montanhosa cuja a largura é de 1150 pixels e a altura de 863 pixels. A imagem na figura 3.3 está no formato JPEG e seu tamanho original é de 430K. O processo de conversão desta imagem para WebP levou exatos 0,428 milissegundos em uma máquina com processador Intel Core I5 de 2.5GHz. Após a conversão para o formato WebP, como pode ser visto na figura 3.4, a imagem resultante foi reduzida para 228K, aproximadamente 46,98% de ganho na compressão.

Figura 3.3: Imagem no formato JPEG com tamanho de 430K

Figura 3.4: Imagem no formato WebP com tamanho de 228K

Como o primeiro caso retrata uma imagem com tamanho consideravelmente pequeno, foi

feita uma avaliação com uma imagem maior para verificar se o comportamento do formato WebP é alterado. A imagem mostrada na figura 3.5 também está no formato JPEG e possui largura de 4928 pixels e a altura de 3264 pixels. O tamanho original da imagem é de 6996K. O processo de conversão desta imagem para WebP levou 5098 milissegundos em uma máquina com processador Intel Core I5 de 2.5GHz. A imagem no formato WebP, como pode ser visto na figura 3.6, foi reduzida para 1008K. Neste segundo caso o ganho foi maior ainda, uma redução de 85,59% com relação a imagem original.

Figura 3.5: Imagem no formato JPEG com tamanho de 6996K

Figura 3.6: Imagem no formato WebP com tamanho de 1008K

O último caso é uma imagem no formato PNG que utiliza o recurso de transparência (*alpha*), ela possui largura de 1353 pixels por 1034 pixels de altura. A imagem no formato original tem tamanho de 840K conforme mostra a figura 3.7, e o processo de conversão para o formato WebP levou 0,328 milissegundos em uma máquina com processador Intel Core I5 de 2.5GHz. A imagem resultante tem tamanho de 56K, uma redução de 93,33% com relação ao tamanho original como pode ser visto na figura 3.8. O ganho maior neste caso pode ser explicado pelo próprio recurso de transparência que tem compressão mais eficiente no formato WebP, e

pelo pequeno número de cores na imagem o que também acaba favorecendo os algoritmos do WebP. Todos os casos estudados sofreram compressão do tipo *lossy*.

Figura 3.7: Imagem com *alpha* no formato PNG com tamanho de 840K

Figura 3.8: Imagem com *alpha* no formato WebP com tamanho de 56K

4 *Trabalhos Relacionados*

Neste capítulo são apresentadas alguns desenvolvimentos e projetos que se assemelham ao assunto deste trabalho, evidenciando soluções distintas e suas características.

- JPEG 2000 Atualização do padrão JPEG para compressão de imagens.
- JPEG XR Alternativa mais leve ao JPEG 2000 para compressão de imagens.
- Primeiro patch com suporte a decodificação do WebP no Firefox.

4.1 JPEG 2000

JPEG 2000 é um padrão de compressão de imagens, criado pela Joint Photographic Experts Group com o objetivo de ser amplamente utilizado em diversas áreas. Suas principais aplicações se deram em câmeras digitais, possibilitando imagens com resoluções maiores, de maior qualidade, utilizando uma compressão dessas informações em um arquivo fisicamente menor; na *Internet*; em impressões avançadas; em imagens médicas e outros setores chaves. O objetivo do JPEG 2000 não é apenas melhorar o desempenho de compressão em relação ao JPEG, mas adicionar (ou melhorar) características como escalabilidade e capacidade de edição.

O formato JPEG 2000 trouxe alguns benefícios em relação ao JPEG, sendo eles:

- Compressão da imagem sem perda de qualidade
- Preservação da transparência nas imagens
- Uso de máscaras (canais alpha) para especificar uma área da imagem que necessite ser salva em uma taxa de compressão mais baixa (perda de informação de imagem) do que outras áreas que possui um menor interesse para o usuário
- Preservação das informações EXIF nos arquivos de imagem
- Opções do usuário quanto ao tamanho, qualidade e número de imagem de visualização miniaturas em um site.

A principal vantagem oferecida pelo formato JPEG 2000 é a flexibilidade no manuseio do*codestream* obtido após uma compressão de imagem utilizando este algoritmo. O *codestream* obtido, pode ser descodificado de diversas maneiras.

O JPEG 2000 foi publicado como um padrão ISO, ISO / IEC 15444. Atualmente o formato não é suportado em navegadores web e, portanto, não é geralmente utilizado na *Internet*.(18)

4.2 JPEG XR

O formato de imagem JPEG XR (abreviação para *JPEG extended range*) é um padrão de compressão de imagens desenvolvido e patenteado pela Microsoft sob o nome de HD Photo. Com a a popularidade do formato JPEG 2000 em baixa, devido à pouco suporte de aplicativos e falta de melhorias no formato, a Microsoft desenvolveu o JPEG XR, na qual se diz comparável com o JPEG 2000, e mais eficiente que o JPEG. Ao contrário do JPEG 2000, este novo formato proposto pela Microsoft é proprietário, ou seja, necessita de drivers e *plug-ins* específicos para codificação e decodificação das imagens em plataformas que ainda não possui suporte nativo.

O formato JPEG XR suporta a compressão de imagens com e sem perda de qualidade. Em relação ao JPEG original, oferece várias melhorias chave, como:

- Melhor compressão dos dados
- Compressão sem perda de qualidade
- Suporte de cores com maior precisão
- Suporte a mapeamento de transparência
- Suporte a metadados

Atualmente o JPEG XR é suportado por diversas aplicações Adobe Flash Player 11.0, Adobe AIR 3.0, Sumatra PDF 2.1, Windows Imaging Component, .NET Framework 3.0, Windows Vista, Windows 7, Windows 8, Internet Explorer 9, Internet Explorer 10, porém não é comumente utilizado em sites da *Internet*.(19)

4.3 O Primeiro Patch

Em 2010 o engenheiro do Google Vikas Arora criou um *patch* para dar suporte apenas ao processo de decodificação do WebP dentro do Mozilla Firefox (16). Esse *patch* foi submetido à equipe de desenvolvimento com o objetivo de ser incluído oficialmente dentro do código fonte do Firefox. Entretanto, o *patch* não estava funcionando corretamente, havia problemas no

processo de compilação e ainda utilizava a versão 0.1 da libwebp, que estava em fase inicial de desenvolvimento e possuía inúmeros erros. Esta primeira tentativa também utilizava diversas estruturas e padrões obsoletos que não deviam mais ser utilizadas dentro do Firefox, como por exemplo a utilização do tipo primitivo PRUint32 ao invés do uint32_t que é do padrão C ANSI. Estes fatores levaram na não adoção da implementação criada por Vikas Arora.

Quando foi falado publicamente do nosso interesse em conduzir uma nova implementação, Vikas Arora, entrou em contato conosco e se colocou a disposição para auxiliar na correção do seu *patch* anterior ou na elaboração de um novo, também realizando a implementação do processo de codificação do formato WebP.

5 *Descrição do Projeto*

5.1 Objetivos

Conforme apresentado na seção 1.3, o objetivo principal deste trabalho é desenvolver uma solução que habilite o suporte de codificação e decodificação do formato WebP dentro do navegador Mozilla Firefox. Pretende-se, realizar em tempo de execução, a renderização de imagens do formato WebP através da utilização do Mozilla Firefox, possibilitando que o navegador realize a leitura e exibição deste formato de imagem para o usuário.

A motivação do desenvolvimento desta solução, conforme descrito na seção 1.2, se dá pelo fato do formato WebP já se encontrar em um estado avançado de desenvolvimento e maturidade, e já estar recebendo suporte nativo dos browsers Google Chrome e Opera.

O aumento na capacidade de compressão do formato *WebP* trará de imediato a melhora no tempo de resposta para o *download* de imagens no navegador do usuário e também diminuirá o trafego na comunicação entre cliente e servidor. É importante ressaltar também que a popularização e utilização em massa deste novo formato irá beneficiar na diminuição do trafego de rede dentro da internet de uma maneira global, e isso resulta diretamente em uma vazão maior para outros recursos que consomem mais banda, como é o caso de um *streaming* de vídeo por exemplo.

No ano de 2010 um *patch* rudimentar foi escrito e submetido por um membro da comunidade, mas não foi aceito. Existem também algumas extensões que fazem com que o Firefox consiga exibir imagens WebP, porem estas extensões não caracterizam um desenvolvimento adequado visto que elas dependem de *plugins* adicionais. A proposta é que o navegador seja capaz de realizar a renderização nativamente. Nenhum trecho do *patch* enviado em 2010 através do *bug* aberto será aproveitado dentro deste trabalho, será necessário uma produção a partir do zero.

Pelo que se pode averiguar, parece que há por trás disso uma razão política na não adoção do formato WebP até o presente momento. Apesar do Google ter aumentado os investimentos no desenvolvimento do Mozilla Firefox em 2011, o Google Chrome vem tomando espaço do Firefox na internet e há um temor de que a própria existência do Firefox esteja ameaçada graças

ao grande sucesso do navegador concorrente. Entretanto pelo que foi pesquisado dentro da comunidade há alguma chance de que caso uma boa implementação seja feita, ela poderá ser aceita dentro do código oficial, embora isto não seja o principal objetivo do trabalho.

Como o tempo para implementação do projeto é reduzido não haverá espaço para a re-escrita de uma nova API dentro do Firefox. O que será feito é a incorporação da biblioteca libwebp desenvolvida pelo Google dentro do Firefox e o trabalho efetivamente será a criação do decodificador e codificador integrado com as bibliotecas do próprio Firefox.

5.2 Projeto

Visando atingir os objetivos descritos para este trabalho, foi criado o seguinte cronograma de atividades a ser seguido na disciplina de Trabalho de Conclusão I:

1. Estudo do formato WebP e da biblioteca libwebp;
2. Estudo e análise do código do Mozilla Firefox;
3. Estudo dos padrões de codificação do Mozilla Firefox;
4. Estudo da viabilidade de implementação;
5. Estudo do patch proposto em 2010;
6. Desenvolvimento da primeira parte da documentação para o trabalho final.

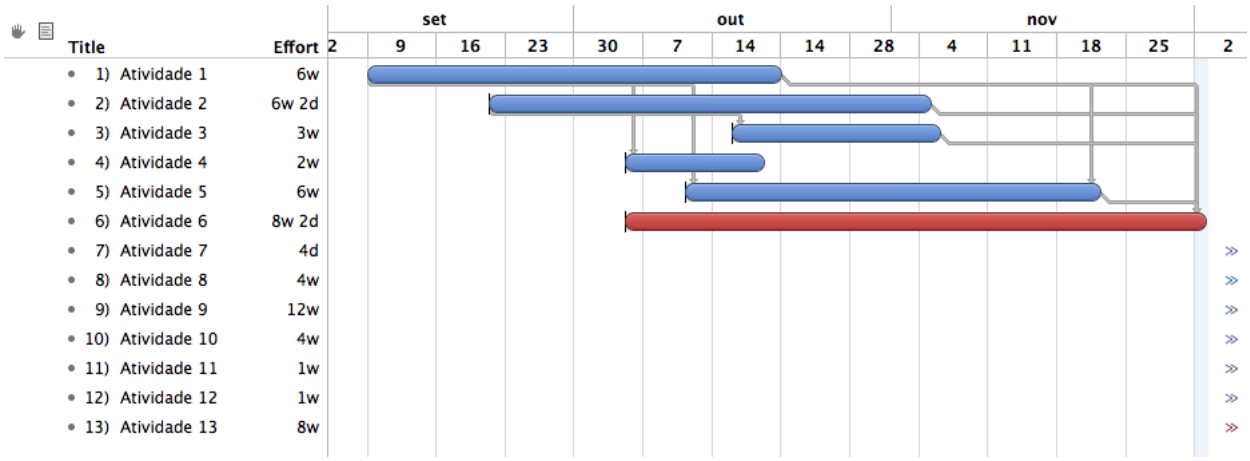

Figura 5.1: Cronograma de atividades da primeira parte.

Para a segunda parte (Trabalho de Conclusão 2) do desenvolvimento do projeto, foi feito um levantamento das tarefas com base nos estudos realizados e foram identificados as seguintes necessidades:

1. Criação de um *fork* da última versão do projeto no repositório oficial.
2. *Merge* da biblioteca libwebp no código do Mozilla Firefox.
3. Criação do codificador e decodificador para o formato WebP.
4. Ajuste no inicializador e carregador das imagens do Mozilla Firefox.
5. Criação dos arquivos para compilação.
6. Teste da implementação e avaliação dos resultados
7. Desenvolvimento da segunda e última parte da documentação para o trabalho final.

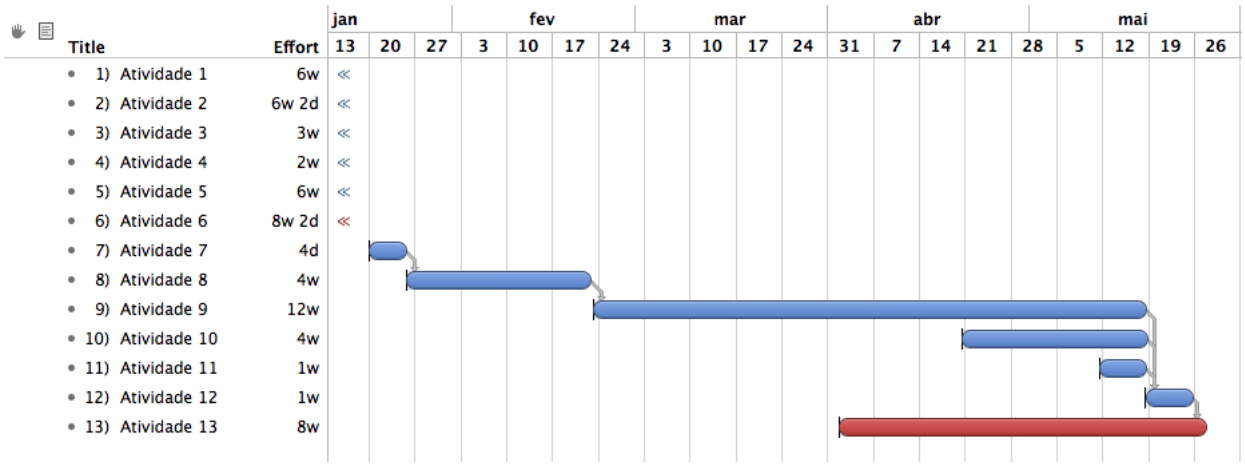

Figura 5.2: Cronograma de atividades da segunda parte.

O planejamento futuro para implementação vislumbra a criação de um *fork* a partir do projeto atual disponível na árvore oficial do Firefox. O trabalho então será feito em cima deste projeto separado para evitar que as atualizações constantes dos mantenedores atrapalhe o desenvolvimento que será realizado.

5.3 Ambiente de Desenvolvimento

As principais ferramentas utilizadas para o desenvolvimento deste trabalho estão listadas na tabela 5.1.

5.4 Modelagem e Arquitetura

A arquitetura de desenvolvimento do Mozilla Firefox permite a extensão de funcionalidades na renderização de imagens sem a necessidade de alteração de grandes porções de código já

Tabela 5.1: Ferramentas utilizadas para o desenvolvimento do trabalho.

Ferramenta	**Descrição**
LaTeX	Conjunto de macros para processamento de texto. Está sendo utilizado para criação de todo material textual.
GIT	Ferramenta utilizada para controle versão distribuída.
AbnTeX	Biblioteca LaTeX para processamento de texto utilizando as normas da ABNT.
BibTeX	Ferramenta utilizada para gerenciamento das referências descritas na documentação.
VIM	Editor de texto.
Bash	Interpretador de comandos que está sendo utilizado para compilação e geração da documentação através de *scripts*.
DIA	Ferramenta utilizada para o desenho de diagramas.
GDB	Depurador utilizado no desenvolvimento da implementação do trabalho.
GCC	Coleção de compiladores GNU. Foram utilizados os compiladores de C e C++.
Firebug	Extensão para o Firefox que é utilizada para inúmeras atividades de depuração e estatística do navegador.

existente. É uma arquitetura bem modular e com baixo acoplamento, o que permite a fácil integração de novos componentes quando necessário.

De uma maneira geral, o núcleo alterado para implementar a codificação e decodificação do formato WebP é a parte responsável pela inicialização das imagens (Image.h e Image.cpp) e pelo carregamento das imagens (ImageLoader.h e ImageLoader.cpp). Com o codificador e o decodificador implementados, foi necessário fazer alterações neste módulos para que o navegador possa nativamente manipular as imagens no formato WebP. Na figura 5.1 é possível ver um diagrama macro que mostra o nível de alteração necessária nos módulos de imagem e como eles se relacionam entre si.

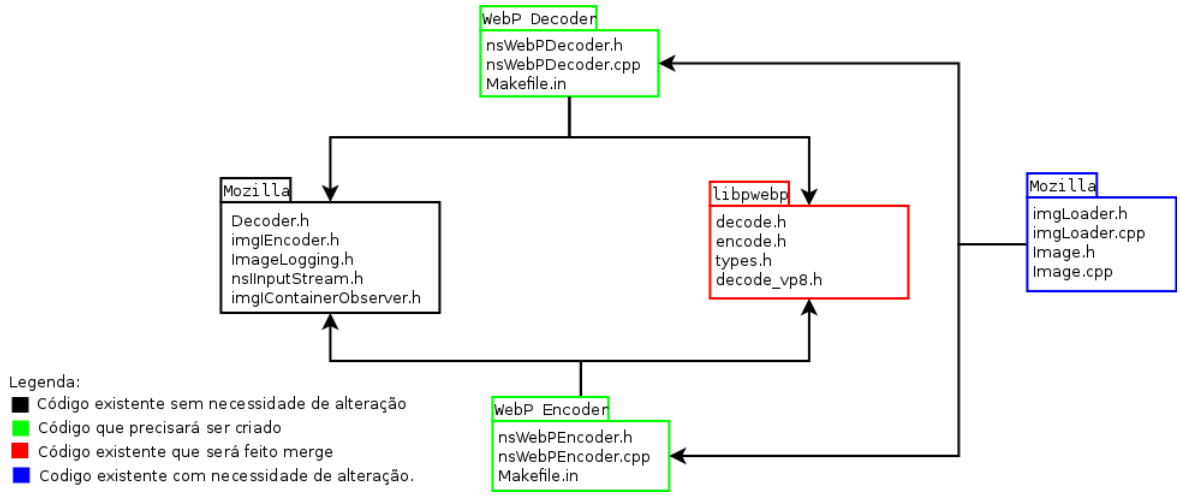

Figura 5.3: Diagrama dos componentes e suas relações

Também foram necessárias pequenas alterações em outros pontos como por exemplo a

definição do *mime-type* para o formato WebP que foi acrescentado dentro do Firefox, mas estas não caracterizaram mudanças significativas de forma que a maior parte do trabalho foi realizada em cima do núcleo de manipulação de imagens do navegador, conforme mostrado no diagrama dos componentes.

5.5 Sistema de compilação do Firefox

Para realizar o desenvolvimento do codificador e decodificador, foi necessário entender como o sistema de compilação do Firefox funcionava. A fusão da libwebp com o Firefox só poderia ser feita caso este sistema fosse detalhadamente entendido, uma vez que o processo de compilação do mesmo serviria também para compilar o código da libwebp.

O sistema de compilação ou o *build system* do Firefox é o mesmo utilizado por todos os outros softwares mantidos pela fundação Mozilla (Thunderbird, Sunbird e etc). Ele utiliza o tradicional software *autoconf* para montar os arquivos de configuração e checar dependências antes do processo de compilação, e o *make* para efetivamente realizar a compilação dos arquivos fonte. O processo é basicamente composto de 3 fases:

1. Configuração
2. Preparação do *backend* e definição do *build*
3. Invocação do *build backend*

Na primeira fase o *autoconf* é invocado para que as características do sistema e do compilador utilizados possam ser determinadas. Aqui também é realizada toda verificação por dependências necessárias para realização do *build*. A segunda fase na verdade é a aplicação da configuração encontrada na primeira fase para todo o código fonte existente que será compilado. A terceira e última fase é onde ocorre efetivamente a compilação do código, baseando-se nas configurações definidas nas fases anteriores. Desta forma o processo pode ser definido como uma árvore, caso qualquer uma das fases falhe, a próxima etapa não será iniciada.

Toda orquestração do processo de compilação é baseada no arquivo moz.build existente dentro de cada diretório que contém um fonte a ser compilado ou a ser exportado como referência (arquivos de cabeçalho por exemplo). A sintaxe de definição do arquivo moz.build obedece as regras regidas pela linguagem Python. Na figura 5.4 é possível observar como a hierarquia está montada e como cada software está envolvido dentro do processo. Como existem várias linguagens de programação existentes dentro do Firefox, a fase de preparação fica responsável por acionar o compilador correto para cada fonte.

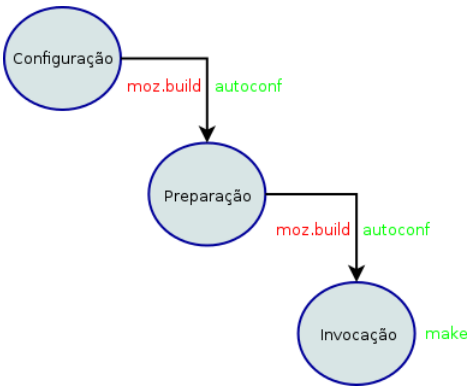

Figura 5.4: Estrutura do sistema de compilação do Firefox

5.6 Implementação do decodificador

O Firefox é desenvolvido na linguagem de programação C++ e utiliza os conceitos de orientação à objetos de forma massiva. A criação do decodificador passou primeiramente por uma avaliação dos outros decodificadores de imagem já existentes dentro do Firefox, mais especificamente o nsJPEGDecoder para decodificação de imagens JPEG e o nsPNGDecoder para decodificação de imagens PNG. Para a implementação foi necessário criar uma nova classe para a decodificação do formato WEBP que herdasse a classe Decoder original do Firefox. A classe Decoder por sua vez parte da premissa que os seguintes métodos precisam ser implementados:

- void InitInternal
- void WriteInternal
- void FinishInternal

De uma forma bastante simples, estes métodos são uma espécie de gancho que servem para inicializar o processo de decodificação, realizar efetivamente a decodificação e finalizar o processo para a classe Decoder. O processo de inicialização é utilizado para repassar o *buffer* da imagem para o decodificador responsável através do método InitInternal. De posse da imagem a decodificação é efetivamente realizada pelo método WriteInternal. O acionamento do método FinishInternal siginifica para o Firefox que a imagem já foi devidamente decodificada e não há mais nada a ser realizado.

Dentro do método WriteInternal o *buffer* é passado para o tipo WebPDecBuffer que através da chamada do método WebPIDecGetRGB da classe decode.h do WEBP, captura as informações de altura e largura da imagem baseadas no *buffer* e realiza a decodificação. O buffer resultante é uma *array* no formato RGB (*Red, Green and Black*), adicionalmente com o canal alfa para transparência caso necessário. O método gfxPackedPixel é chamado por sua vez para realizar o preenchimento de cada pixel da imagem com o valor correspondente. O resultado

é a renderização da imagem utilizando os mecanismos já implementados do Firefox após a execução do método FinishInternal.

Como se pode observar em nenhum momento é necessário saber quais são os algoritmos utilizados para realizar a decodificação. Toda esta complexidade fica encapsulada dentro da libwebp que é instrumentada pelas classes criadas para realizar o processo.

5.7 Implementação do codificador

Antes de realizar a implementação do codificador dentro do Firefox, foi preciso entender a biblioteca de codificação da libwebp. O próprio cabeçalho principal da biblioteca (encode.h) é a documentação para o processo de codificação. Existem 3 tipos de estruturas fundamentais para o codificador. Abaixo o detalhamento destas estruturas e a sua respectiva função.

Tabela 5.2: Estruturas fundamentais para o codificador WebP.

Estrutura	**Descrição**
WebPPicture	É a estrutura principal do codificador contendo o *buffer* da imagem, informações da altura e largura, tipo de codificação (RGB ou YUV) e outros dados relevantes.
WebPConfig	Conjunto de dados de configuração e recursos para a imagem, como por exemplo se há ou não a opção de transparência (*alpha*), qualidade da imagem e etc.
WebPMemoryWriter	É responsável por realizar a manipulação de dados dentro de uma estrutura WebPPicture. A escrita de qualquer segmento de *buffer* dentro de uma WebPPicture depende desta estrutura.

O processo de codificação da libwebp possui vários recursos adicionais que não foram citados dentro deste trabalho por não terem sido utilizados no desenvolvimento do codificador.

Assim como no desenvolvimento do decodificador, a criação do codificador passou por uma avaliação dos outros codificadores que já estão implementados no navegador. Como a implementação do nsPNGEncoder estava mais organizada e bem documentada, ela serviu como base para criação do codificador WebP. A classe principal criada para o processo de codificação deveria herdar a classe imgIEncoder do Firefox. Como premissa da classe pai, todos os métodos a seguir precisaram ser implementados para o codificador nsWEBPEncoder:

- InitFromData
- StartImageEncode
- GetImageBufferUsed

- GetImageBuffer
- AddImageFrame
- EndImageEncode
- Close
- Available
- Read
- ReadSegments
- IsNonBlocking
- AsyncWait
- CloseWithStatus

Dos 13 métodos listados a interação com a libwebp foi criada dentro dos métodos StartImageEncode, AddImageFrame e EndImageEncode. Tudo começa no método orquestrador InitFromData, ele é o primeiro método a ser chamado e a sua função é receber o *buffer* do chamador. A partir dai o método IniFromData executa o método StartImageEncode que é responsável por inicializar as estruturas de dados WebPPicture, WebPConfig e WebPMemoryWriter. O método AddImageFrame possui uma iteração que lê linha a linha do *buffer* e realiza a construção da codificação através do método WebPMemoryWrite da libwebp. Este método faz a gravação dos dados dentro da estrutura WebPPicture que é efetivamente a imagem no formato WebP. O último método chamado é o EndImageEncode que realiza a liberação da memória utilizada por todo o processo de codificação.

Ao fim da execução da classe nsWEBPEncoder desenvolvida no trabalho, a imagem já está devidamente codificada no formato WebP.

5.8 Resultados Obtidos

Através da implementação realizada, o Firefox conseguiu codificar e decodificar o formato WebP com sucesso. Como pode ser observado na figura 5.6, está disposta uma página que possui uma imagem no formato WebP em um Firefox sem a implementação, consequentemente a imagem não é exibida. Já na figura 5.6 é possível verificar que a mesma imagem no formato WebP é exibida corretamente, dentro de um Firefox que possui a implementação do decodificador. Na figura 5.7 foi criado um HTML que utiliza o método *canvas* para desenhar uma figura no navegador. O método *toDataURL* é executado com o parâmetro *image/webp* para

fazer com a imagem seja renderizada no formato WebP. Para este caso especificamente não há como mostrar uma diferença clara através de imagens pois no Firefox sem a implementação do codificador, quando o método *toDataURL* é utilizado com o parâmetro *image/webp*, ele simplesmente realiza a renderização da imagem no formato PNG por não possuir o suporte ao WebP. Isto ocorre conforme orientação no rascunho da especificação do elemento *canvas* para o HTML5 (20):

> *Only support for image/png is required. User agents may support other types. If the user agent does not support the requested type, it must return the image using the PNG format.*

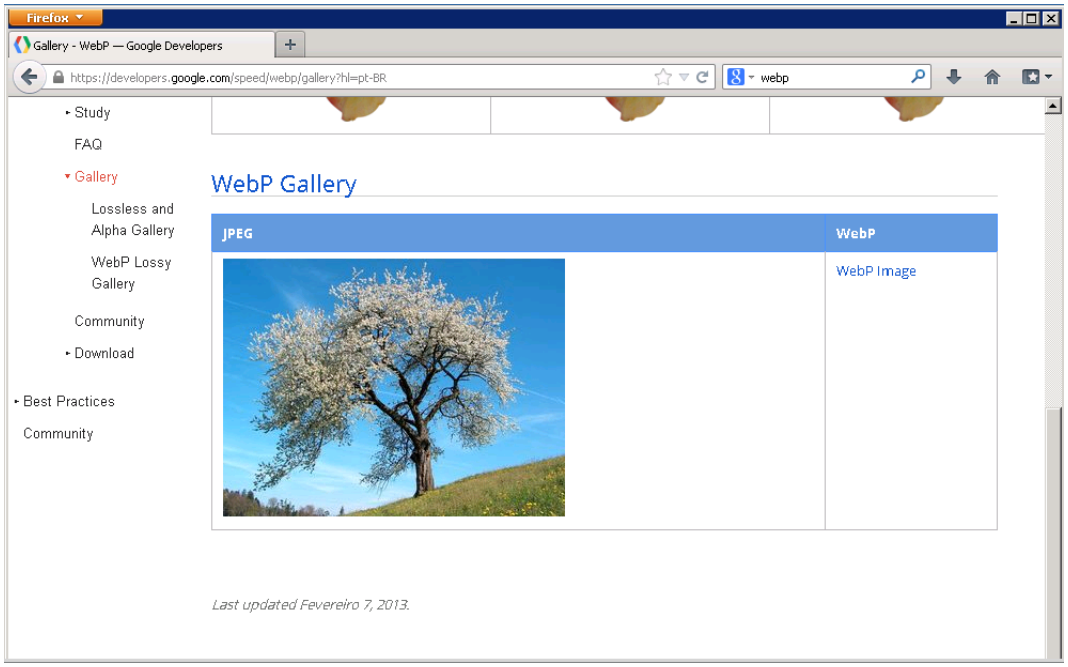

Figura 5.5: Firefox sem a implementação do trabalho

Afim de avaliar o desempenho de forma prática, foi desenvolvido uma página da *web* composta de imagens no formato PNG e JPEG, e uma segunda página exatamente igual a primeira apenas com as imagens convertidas de PNG e JPEG para WebP. A página em sua formação original possui tamanho de 5953K e após a conversão das imagens para WebP, o tamanho de todo o *site* passou a ser de 907K. A tabela 5.3 mostra uma comparação dos tempos para carregamento da página bem como o nível de utilização de *bytes* trafegados durante o processo de abertura da página.

O teste consistiu apenas na entrada da página principal, não houve nenhuma navegação adicional dentro do *site*. A coluna ”Tempo Site normal” indica o tempo de carregamento da página com as imagens no formato PNG/JPEG. A coluna ”Tempo Site WebP” indica o tempo de carregamento da página com as imagens no formato WebP. As colunas ”Bytes Site normal” e ”Bytes Site WebP” indicam a quantidade de bytes trafegados durante a abertura da página com as imagens no formato PNG/JPEG e com imagens no formato WebP respectivamente.

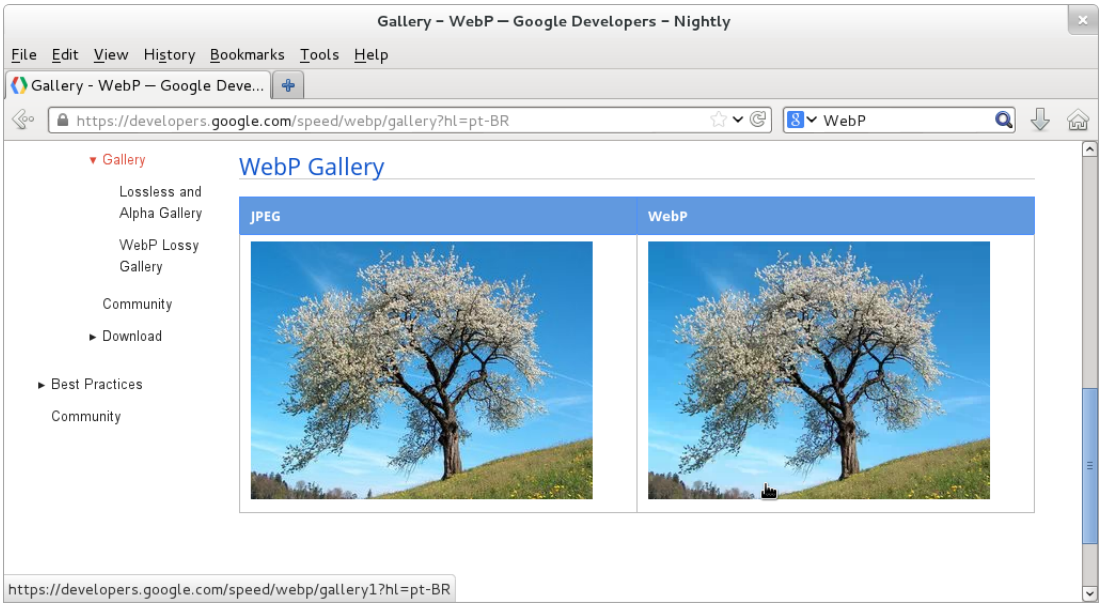

Figura 5.6: Firefox com a implementação do trabalho

Tabela 5.3: Avaliação prática do WebP

Navegador	**Tempo Site normal**	**Tempo Site WebP**	**Bytes Site normal**	**Bytes Site WebP**
Chrome	5,52 segundos	4,30 segundos	2,4 Megabytes	757 Kbytes
Opera	6,34 segundos	4,57 segundos	2,4 Megabytes	754 Kbytes
Firefox sem WebP	5,89 segundos	N/A	2,4 Megabytes	N/A
Firefox com WebP	5,64 segundos	4,39 segundos	2,4 Megabytes	695 Kbytes

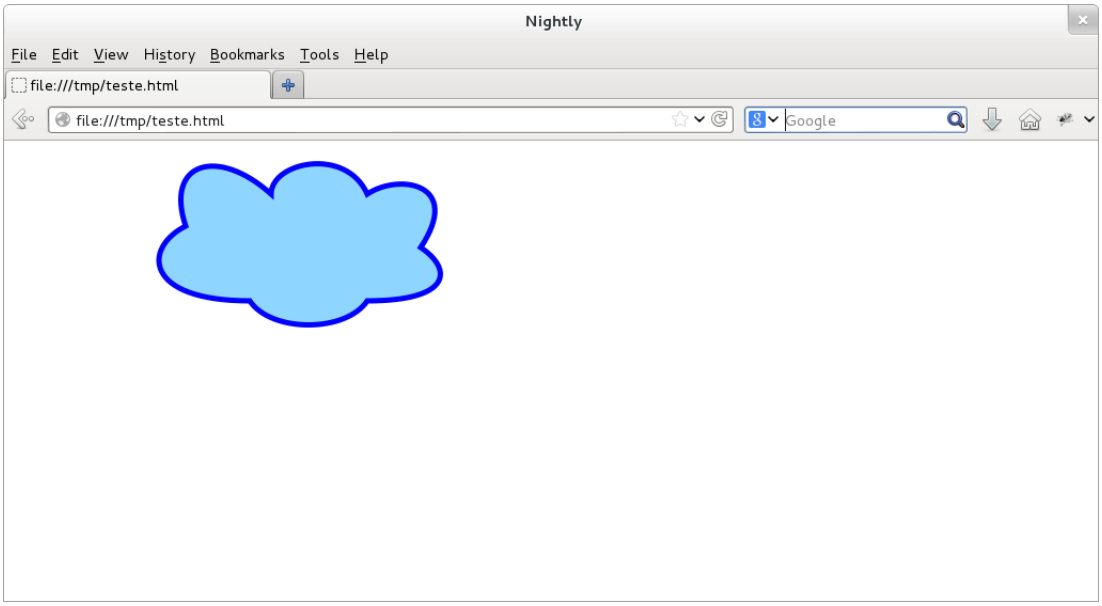

Figura 5.7: Firefox com a implementação do trabalho usando o elemento *canvas*

6 Conclusões e Trabalhos Futuros

A implementação da codificação e decodificação do formato WebP dentro do navegador Mozilla Firefox foi possível conforme demonstrado pelo estudo feito na fundamentação teórica, na motivação e no detalhamento das informações de implementação. Apesar de ter uma documentação bastante ampla, a falta de artigos científicos publicados prejudicou a composição das referências deste trabalho. Como a especificação do formato só foi feita e divulgada ao público em meados de 2010 após a compra da empresa On2 pelo Google, é possível que tenha havido pouco tempo para a criação de trabalhos acadêmicos relacionados ao WebP.

Apesar da evolução constante realizada pele equipe de desenvolvimento, existem ainda diversos processos burocráticos pendentes que atrapalham no processo de adoção da tecnologia, como por exemplo o cadastramento do formato dentro de órgãos de registro da internet como a IANA (*Internet Assigned Numbers Authority*). Sabe-se que a adoção de um formato depende não somente da qualidade da tecnologia, mas sim da amplitude de sua utilização. Por este motivo o seu sucesso está diretamente associado à disseminação da tecnologia tanto por parte dos *softwares* de manipulação de imagens, quanto pelos navegadores de internet. O formato não pode se mostrar apenas eficiente do ponto de vista de infraestrutura, os desenvolvedores precisam achar atrativos e recursos interessantes para optarem por utilizar o WebP, e esse é um dos maiores desafios que os mantenedores oficiais do formato precisam resolver.

A versão 0.3.0 da libwebp, utilizada no desenvolvimento do trabalho, oferece suporte à animação de imagens quadro a quadro, semelhante ao processo realizado pelo formato GIF. Entretanto este tipo de implementação não foi realizada no desenvolvimento do codificador e do decodificador, ficando como referência para trabalhos futuros que devem ser realizados.

Referências Bibliográficas

1 W3SCHOOLS. *HTML5 Introduction*. 2013. Acessado em: 2013-06-08. Disponível em: <http://www.w3schools.com/html/html5_intro.asp>.

2 GOOGLE. *Make the Web Faster*. 2012. Disponível em: <https://developers.google.com-/speed/>.

3 GOOGLE. Comparative study of webp, jpeg and jpeg 2000. 2010. Disponível em: <https://developers.google.com/speed/webp/docs/c_study>.

4 STATCOUNTER. Website, *StatCounter Browsers Global Stats*. 2013. Disponível em: <http://gs.statcounter.com/>.

5 KONIG, S. Blog, *Using WebP to Improve Speed*. 2013. Disponível em: <http://blog-.chromium.org/2013/02/using-webp-to-improve-speed.html>.

6 PFEIFFER, S. *The Definitive Guide to HTML5 Video*. Apress, 2010. (Definitive Guide Series). ISBN 9781430230908. Disponível em: <http://books.google.com.br-/books?id=CypiQKqHMesC>.

7 COMMUNICATIONS, C. W. How the web began. 2008. Disponível em: <http://public-.web.cern.ch/public/en/about/WebStory-en.html>.

8 W3C. Facts about w3c. 2012. Disponível em: <http://www.w3.org/Consortium/facts>.

9 CURBERA, W. A. N. F.; WEERAWARANA, S. Web services: Why and how. IBM T.J. Watson Research Center, 2001. Disponível em: <https://researcher.ibm.com/files/us-bth/nagy-.pdf>.

10 W3C. Html5: A vocabulary and associated apis for html and xhtml. Editor's Draft 1 September 2012, 2012. Disponível em: <http://dev.w3.org/html5/spec/single-page.html>.

11 BLOOMBERG. Bloomberg game changers: Marc andreesen. 2012. Acessado em: 2012-09-03. Disponível em: <http://www.bloomberg.com/video/67758394>.

12 GROSSKURTH, A.; GODFREY, M. W. A reference architecture for web browsers. IEEE Computer Society, 2005. Disponível em: <http://ieeexplore.ieee.org/ielx5/10097/32336-/01510168.pdf>.

13 TAIVALSAARI, A.; MIKKONEN, T. The web as an application platform: The saga continues. IEEE Computer Society, 2011. Disponível em: <http://ieeexplore.ieee.org/ielx5-/6068192/6068309/06068340.pdf>.

14 SHARP, B. L. . R. *Introducing HTML 5*. Second edition. [S.l.]: New Riders, 2011. 11-15 p.

15 GOOGLE. Webp compression study. 2012. Disponível em: <https://developers.google-.com/speed/webp/docs/webp_study>.

16 IMPLEMENT WebP image support. 2010. Disponível em: <https://bugzilla.mozilla.org-/show_bug.cgi?id=600919>.

17 MUIZELAAR, J. *Críticas ao WebP*. 2011. Disponível em: <http://muizelaar.blogspot-.com.br/2011/04/webp.html>.

18 GROUP, J. P. E. *Joint Photographic Experts Group, JPEG2000*. Disponível em: <http://www.jpeg.org/jpeg2000/index.html>.

19 DUFAUX, F.; SULLIVAN, G.; EBRAHIMI, T. The jpeg xr image coding standard [standards in a nutshell]. IEEE Computer Society, 2009.

20 W3C. The canvas element. 2011. Disponível em: <http://www.w3.org/TR/2011/WD-html5-20110405/the-canvas-element.html>.