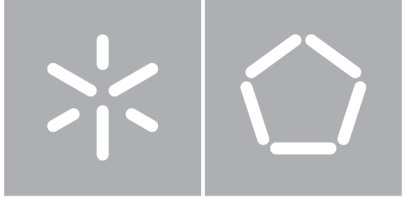
Universidade do Minho
Escola de Engenharia

Marco Pereira Carneiro

Um player web para redes de ecrãs públicos

Outubro de 2013



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marco Pereira Carneiro

Um player web para redes de ecrãs públicos

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Rui João Peixoto José

Outubro de 2013

Agradecimentos

Os meus agradecimentos são para todos que, de forma directa ou indirecta, contribuíram para a realização desta dissertação. No entanto, gostaria de realçar as seguintes pessoas e entidades.

Ao meu orientador Professor Rui José, pela orientação, compreensão, paciência e disponibilidade com que acompanhou o trabalho desenvolvido. Agradeço, também, pela leitura atenta, correcções e sugestões feitas durante a orientação e escrita da dissertação.

Aos colegas do grupo UBICOMP pelos comentários e sugestões; em especial ao Bruno Silva.

Ao meu amigo e 'afilhado' Miguel Andrade, pelo apoio e troca de sugestões, comentários e assistência nas etapas iniciais da dissertação.

À Universidade do Minho, por ter sido a minha casa nos últimos sete anos.

À minha família e amigos pelo apoio e incentivo; em especial à minha mãe pela atenção, inspiração, interesse, paciência, disponibilidade e compreensão em todas as etapas ao longo do mestrado.

À Professora Maria Inês, pela leitura atenta e respectiva correcção da dissertação.

Resumo

As redes de ecrãs públicos incluem um elemento de software, normalmente designado player, que funciona junto de cada ecrã para obter e apresentar os conteúdos. Esse player é quase sempre software proprietário e recebe as suas instruções num formato igualmente proprietário.

O aparecimento de redes abertas de ecrãs públicos e as crescentes potencialidades das tecnologias web abrem caminho para o surgimento de players web suficientes genéricos para poderem servir as necessidades de sistemas heterogéneos, capazes de funcionar numa vasta gama de plataformas e com base em formatos de dados comuns. A existência de players com essas características representaria um passo tão fundamental para as redes de ecrãs públicos como o surgimento da web representou para a partilha e consulta de informação na Internet.

Este trabalho visa desenvolver um projecto de suporte à apresentação de conteúdos em redes abertas de ecrãs públicos e insere-se nos objetivos do projeto europeu PD-NET. O trabalho a desenvolver inclui o estudo dos requisitos do sistema, a implementação com base em tecnologia HTML5 e Javascript, e a respectiva avaliação. O player deverá ter a capacidade de escalonar conteúdos web, de acordo com as instruções recebidas de um servidor num formato aberto.

Abstract

The networks of public displays include an element of software, commonly referred to as player, which operates at each screen to obtain and present the contents that will be presented there. This player software is almost always proprietary and receives instructions in a format that is also proprietary.

The emergence of open networks of public displays and the growing potential of web technologies pave the way for the emergence of web players generic enough to be able to serve the needs of heterogeneous systems and capable of operating in a wide range of platforms and based formats on common data. The existence of players with these characteristics would represent a essential step for public displays networks, with a similar impact as the emergence of the web as had for information sharing on the internet.

This work aims to support the presentation of content on public displays of open networks and fits into the objectives of the european project PD-NET. It includes the study of system requirements, an implementation based on HTML5 and Javascript technology, and the respective evaluation. The player should have the ability to schedule web content, in accordance with instructions received from a server in an open format.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

HTML5 HyperText Markup Language 5

HTML4 HyperText Markup Language 4

HTML HyperText Markup Language

XML Extensible Markup Language

WebGL Web Graphics Library

CDS Content Descriptor Set

XHTML Extensible HyperText Markup Language

API Application Programmin Interface

SVG Scalable Vector Graphics

W3C World Wide Web Consortium

PC Personal Computer

SFF Small Form Factor

COM computer-on-module

NOOBS New Out Of Box Software

HDMI High-Definition Multimedia Interface

DVI Digital Visual Interface

RSS Rich Site Summary

JSON JavaScript Object Notation

URL Uniform Resource Locator

SMIL Synchronized Multimedia Integration Language

HTTP Hypertext Transfer Protocol

DOM Document Object Model

HTTPS Hypertext Transfer Protocol Secure

CSS Cascading Style Sheets

UI User Interface

CSP Content Security PolicyUser Interface

NPAPI Netscape Plugin Application Programming Interface

SPA Single Page Application

MIME Multipurpose Internet Mail Extensions

USB Universal Serial Bus

SSL Secure Sockets Layer

SQL Structured Query Language

CPU Central Processing Unit

UUID Universally Unique IDentifier

1 Introdução

Neste capítulo inicial é feito o enquadramento da dissertação, apresentada a motivação, os desafios inerentes à mesma, identificados os principais objectivos, terminando com a apresentação da estrutura da dissertação, incluindo uma descrição sumária dos capítulos.

1.1 Enquadramento

Os ecrãs públicos sempre fizeram parte da visão da computação ubíqua, mas a sua utilização apenas atraiu um maior interesse nos últimos anos devido à crescente ubiquidade dos ecrãs em relação às suas dimensões, às suas localizações, assim como as novas tecnologias de projecção. No entanto, apesar do aumento de interesse se ter reflectido no aparecimento de aplicações e novas formas de utilização dos ecrãs públicos, revela-se que estes dispositivos não são muito valorizados pelo utilizador ao qual se destinam.

As razões principais relacionam-se com o facto de as redes de ecrãs públicos recorrerem à utilização de software proprietário [2, 3], limitando as suas capacidades a nível tecnológico, resultando em conteúdos restritos ao software no qual executam, tornando-se repetitivo e não interactivo, tipicamente utilizado como pontos de distribuição de informação previa-

mente definida; geralmente conteúdo institucional ou publicitário, o qual é muitas vezes interpretado, pelos utilizadores, como conteúdo estático, tendo em conta os locais onde os ecrãs se encontram instalados e o tipo de escalonamento ser realizado de forma cíclica sequencialmente ou aleatória através de conjuntos básicos de regras e prioridades.

O aparecimento e a crescente potencialidade das tecnologias web, como o HTML5 e o *JavaScript*, fez surgir uma oportunidade para o desenvolvimento de uma aplicação web, denominada de player web. Esta proporciona uma oportunidade para repensar e redefinir a forma como é utilizada o modelo de rede de ecrãs públicos, bem como enriquecer os espaços físicos envolventes numa rede de ecrãs públicos. Ao recorrer a informação mais relevante e abrangente, valoriza a interacção entre os ecrãs e os utilizadores através de conteúdos de terceiros que são disponibilizados através da web [4]. Possibilita ao player a exibição de conteúdos mais dinâmicos. Este novo modelo de redes de ecrãs públicos pode originar novas aplicações interactivas e sensíveis ao novo contexto do player web. Dada a abundância de conteúdos e fontes de informação que se encontram na web proporciona uma solução atractiva para o problema de conteúdos estáticos. O conteúdo fornecido por estas fontes é um importante recurso e pode representar um papel fundamental no desenvolvimento de novas aplicações e contribuir para um ambiente mais cativante e de maior utilização.

Nesta dissertação explora-se o potencial das tecnologias web como base para um player de ecrãs públicos. Enquanto o *JavaScript* manipulará a interacção do utilizador para cada página multimédia, a sincronização será responsável pela comunicação com o ficheiro de formato aberto (*schedule*), em contrapartida o HTML5 será a plataforma web em que esta aplicação irá ser desenvolvida, simultaneamente, o player estará numa plataforma de navegadores web, no qual a comunidade de programadores é enorme e aberta para possíveis aplicações web que possam vir a ser embutidas neste player, aumentando o respectivo leque de conteúdos para o utilizador, de forma a aumentar a utilização dos ecrãs públicos.

Em suma, o desenvolvimento de uma aplicação web para o player, o modelo de apresentação de conteúdos e regras poderão ser configuráveis

para cada rede de ecrãs, bem como o conteúdo a ser exibido, ser mais amplo e atractivo.

1.2 Desafios

O desenvolvimento de um player web apresenta vários desafios, que não são normalmente encontrados no desenvolvimento de uma aplicação nativa. A particularidade de desenvolver uma aplicação nas tecnologias web existentes pode criar novas oportunidades para a propagação de aplicações web dedicadas a uma rede de ecrãs públicos, no qual é possível englobar conteúdos de terceiros, e, paralelamente, existe uma grande variedade de sistemas do género, noutro tipo de plataformas, que oferecem uma oportunidade de análise e levantamento de requisitos, nomeadamente:

- adoptar a caracterização similar de um player em aplicação nativa para o ambiente tecnológico web;
- determinar as capacidades e limitações das potencialidades web na integração do player web;
- capacidades e limitações dos ambientes de execução, relativamente ao navegador web;
- aceitação de um novo modelo *open source* de redes de ecrãs públicas de forma a aumentar a dinamização do conteúdo através de aplicações web de terceiros;
- pressupostos de utilização diferentes.

Logo, várias questões devem ser consideradas durante o desenvolvimento do player para uma rede de ecrãs públicos, nomeadamente a plataforma computacional e ambiente de execução mais adequado para a o funcionamento de um player web; o formato e a estrutura *schedule* que melhor se adequa para definir um conjunto de regras e prioridades; o melhor método possível para escalonamento e exibição de conteúdos de terceiros de forma

independente, íntegra e segura para o player e o controlo do estado de operação do player para assegurar a funcionalidade do mesmo através de tolerância a faltas.

1.3 Objectivos

O objectivo desta dissertação é desenvolver um protótipo de uma aplicação web, denominada de player, para uma rede de ecrãs públicos. O objectivo deste protótipo é receber dinamicamente uma lista de conteúdos a ser exibido no ecrã, assim como um conjunto de regras do mesmo e ser um elo de comunicação com os vários conteúdos multimédia.

Mais especificamente, os objectivos desta dissertação são:

- identificar e caracterizar os principais desafios inerentes à apresentação de conteúdos web em nós de uma rede aberta de ecrãs públicos;
- caracterizar o espaço de desenho associado a players baseados em tecnologias web para redes heterogéneas de ecrãs públicos;
- desenvolver um player baseado na web que permita apresentar aplicações web e que sirva as principais necessidades identificados no espaço de desenho;
- desenvolver um formato de representação de regras de escalonamento de conteúdos web, que satisfaça os requisitos inerentes à apresentação de conteúdos dinâmicos em redes abertas de ecrãs públicos.

1.4 Organização da dissertação

A dissertação está composta em seis capítulos, que se encontram organizados da seguinte forma:

1. Introdução

   Este capítulo apresenta o contexto do trabalho a desenvolver, bem como os desafios e os objectivos inerentes.

2. Estudo e análise do trabalho relacionado

   Este capítulo apresenta a revisão bibliográfica ao estado de arte relacionado. Inclui um estudo geral sobre as plataformas computacionais e players existentes no mercado, bem como as tecnologias web.

3. Estudos técnicos

   Este capítulo aborda o estudo de várias soluções tecnológicas de forma a poder implementar as funcionalidades descritas no capítulo anterior.

4. Identificação dos requisitos e opções tecnológicas

   Este capítulo apresenta a caracterização das funcionalidades para o player, bem como as opções tecnológicas tomadas relativamente ao ambiente de execução, as suposições relativamente ao *schedule* e o modelo de aplicações web.

5. Protótipo

   Este capítulo apresenta a arquitectura e estrutura modular aplicacional que compõe o player, e a implementação do protótipo do player e respectiva definição dos módulos e estado de operação.

6. Conclusão

   Por fim uma conclusão sobre o trabalho desenvolvido e indicadas algumas áreas para trabalho futuro.

2
Estado da Arte

Este capítulo apresenta uma análise do trabalho relacionado, às tecnologias web e às plataformas computacionais e players. São analisados e comparados vários trabalhos que abordam o funcionamento e estrutura de várias redes de ecrãs públicos, bem como o funcionamento de players para este tipo de redes. Também é feita uma breve análise às tecnologias web para o qual o player é desenvolvido. Por fim, são analisadas as plataformas computacionais e os players existentes no mercado, a fim de determinar as funcionalidades e limitações destas áreas relativamente ao ambiente de execução.

2.1 Trabalho relacionado

O estudo do modelo de aplicações web para uma rede de ecrãs públicos [5] contribui, no entendimento do conceito de aplicações nos ecrãs públicos como um modelo de aplicação baseado na web de forma a permitir que qualquer pessoa interessada no desenvolvimento de aplicações web a possa lançar numa rede de ecrãs públicos. Caracteriza detalhadamente as especificações web, utilizadas nas redes de ecrãs públicos, realçando a importância dos ecrãs públicos serem, em termos tecnológicos, uma área em expansão e uma visão de afastamento de uma rede fechada e proprietária

para uma rede mais aberta, permitindo que esta área seja mais abrangente no mercado. Os desafios são semelhantes, devido ao factor de adaptação às tecnologias web neste domínio. Apesar de ser exclusivamente dedicado ao desenvolvimento de aplicações web, esta dissertação incorpora-se com a interacção deste tipo de aplicações para ecrãs públicos, pois ambos pertencem ao mesmo projecto do InstantPlaces, daí a sua importância no estudo desta matéria.

O UBI-hotspot [3, 4] é um sistema de gestão do estado real de ecrãs públicos interactivos, através de ecrãs virtuais descritos através de XML. Ajuda a compreender as experiências na separação, na decisão do layout da lógica operacional que permite uma rápida configuração do modelo de interacção, bem como a implementação do sistema nas máquinas de estado finais, demonstrando a estrutura de representação declarativa dos ecrãs virtuais, bem como o algoritmo de transição. Apresenta um design e implementação de uma *framework* baseada na web para gestão de ecrãs públicos. Através desta *framework*, permite a integração e o lançamento de aplicações web de terceiros, alojados em qualquer parte da internet, nas partições dinâmicas dos ecrãs, para cada ecrã virtual, permitindo à rede dos ecrãs ser mais aberta e escalável.

O Lively Kernel [2] é um sistema, desenvolvida pela Sun Microsystems, que testa ao limite todas as capacidades de um navegador web, através da implementação de uma aplicação web altamente interactiva. O resultado deste projecto é útil na análise de limitações, desafios e oportunidades relacionadas com o navegador web, como plataforma de aplicações. São abordadas várias tecnologias existentes na web e verifica-se que algumas destas se destacam em relação a outras, como por exemplo, as linguagens dinâmicas, os *mashups* de tecnologias e a enorme dependência nas ferramentas proprietárias. O desenvolvimento desta aplicação identifica-se muito na área web, pois consideram a web como a plataforma de software, o navegador web como o novo sistema operativo e o *Javascript* como a *'de facto'* linguagem dinâmica. Não existe qualquer tipo de instalação ou actualização, possui suporte para a manipulação directa e uniforme e com experiência consistente no desenvolvimento, maleabilidade e suporte dinâmico a altera-

ções. A chave de todo este projecto é a uniformização. É também útil, que nas observações tidas em conta no desenvolvimento de uma aplicação web num navegador web, ajudando a ter percepção relativamente a uma série de problemas de usabilidade, interacção de utilizador, rede e segurança, interoperabilidade do navegador web, compatibilidade, estilo de desenvolvimento, testes, lançamento e performance; além destes problemas sugere resoluções para alguns dos problemas mencionados.

Os avanços feitos nos navegadores web em termos de compatibilidade [6], mostram ainda a existência de várias diferenças em termos de implementação e diferenças na performance. As inúmeras diferenças entre os navegadores web ainda é elevada podendo ficar bem pior. O Lively Kernel, sendo um projecto com tamanhas características de uniformização, demonstra limitações nos navegadores web como plataforma adequada de unificação/standardização, sendo necessário melhor adaptação dos mesmos ao desenvolvimento de aplicações web.

A mudança de paradigma de software para a web [7], aborda novas tecnologias como o HTML5 e WebGL que removem as limitações e transformam a web numa plataforma real de software. A tendência para a web causará uma mudança de visão em relação ao desenvolvimento de software. São mencionados diversos passos importantes na direcção do HTML5, bem como o benefício das aplicações web, tendo como características principais o facto de não necessitar de instalação, actualização, lançamento numa escala global, bem como aplicações desenvolvidas num formato aberto e independentes de software proprietário; uma especial referência às linguagens dinâmicas, com especial destaque para a linguagem de *JavaScript* dada a sua popularidade e o aumento das capacidades de programação.

A utilização de *JavaScript* como linguagem de programação [8], no desenvolvimento do Lively Kernel, aborda várias experiências e comentários relativamente à linguagem, bibliotecas, configuração do software, estilo de desenvolvimento, entre outros. Relativamente à linguagem, são abordados os seguintes factores: linguagem muito tolerante a faltas, uma falta de modularidade e problemas sintácticos. Quanto às bibliotecas, são muito incompletas, dada a falta de uniformização, devido à proliferação de bibli-

otecas que existem no desenvolvimento de aplicações. No que concerne à configuração de software, não existem mecanismos de gestão para largas aplicações que possam conter vários módulos de desenvolvimento. O estilo de desenvolvimento nesta linguagem é muito diferente a *C*, *C++* e *Java*. É necessário um desenvolvimento por etapas, pois os testes ao código são muito importantes e devem ser sempre faseados, dada a tolerância a faltas na linguagem, bem como o estado final da aplicação ser difícil de determinar quando está completamente terminada, devido à flexibilidade da linguagem. Relativamente às máquinas virtuais *Java*, estas são bastante fiáveis, apesar de haver ainda muito espaço para a optimização da performance do *JavaScript*, pois a gestão da capacidade de memória nas máquinas virtuais actuais de *Java* são pouco adequadas a aplicações amplas.

O Yarely [9] é um player, desenvolvido pela Universidade de Lancaster, no Reino Unido. Este software foi desenvolvido para corrigir uma série de preocupações que decorriam do anterior sistema que era utilizado, o e-Campus. Apesar das excelentes infraestruturas que o sistema possuía, a dependência numa única infra-estrutura de eventos centralizada, levou a uma série de preocupações: fragilidade (o sistema era demasiado sensível a falhas de ligação); escalabilidade (a infra-estrutura de eventos tinha dificuldades em escalar quando o número de ecrãs na rede aumentava); falta de suporte para ecrãs em estado *offline* (não era possível executar os ecrãs sem ligação à infraestrutura central); *firewalls*, dificuldade no *pre-fetch*. Este conjunto de limitações representava uma barreira à escalabilidade do sistema para outro contexto. Tendo em conta estas limitações, foi definido um conjunto de objectivos para o novo sistema e estruturada uma nova arquitectura de forma a poder resolucionar os problemas da versão anterior. A análise efectuada aos objectivos vem definir os princípios-chave da nova implementação e corrigir problemas mencionados anteriormente. A linguagem escolhida para o desenvolvimento da aplicação foi *Python* e o CDS (*scheduler*) foi em XML, toda a implementação foi efectuada em módulos-chave, dos quais foram divididos da seguinte forma: gestor de subscrições (módulo responsável pela comunicação com os servidores, para efectuar pedidos e receber *scheduler's*, faz pedidos periódicos para verificar actualizações); gestor de

sensores (módulo responsável por ler dados de sensores e outros tipos de *environment sources*); *playlist* e escalonamento (módulo responsável por processar o ficheiro do *scheduler* e fazer a exibição dos conteúdos no ecrã e ir efectuando as alterações necessárias conforme as regras explícitas no *scheduler*); renderização do conteúdo e gestão do ciclo de vida (renderização do conteúdo refere-se à representação física no hardware, a gestão do ciclo de vida refere-se à cache do conteúdo media que se guarda em cache); análise (módulo responsável por devolver informação aos servidores relativamente ao seu funcionamento e estatísticas).

A análise deste software foi útil para ter uma percepção a nível da arquitectura e uma separação lógica, bem como a nível de implementação com a separação de funcionalidades-chave em módulos de forma a que cada módulo fique responsável pelas suas próprias funcionalidades e assim evitar problemas a priori.

2.2 As tecnologias web

A utilização de tecnologias web como base para o desenvolvimento de players *Digital Signage* pode permitir a criação de software aberto para que outros possam contribuir ao seu melhoramento e desenvolvimento. Pode também permitir a utilização desses players de uma forma abrangente que consiga alcançar o maior número possível de ecrãs. A utilização de HTML5 como tecnologia base para o conteúdo é fundamental. Não só torna o conteúdo dos players dinâmicos e mais atractivos como expande o horizonte a terceiros no desenvolvimento de aplicações web através de HTML5, que podem vir a ser divulgadas numa rede de ecrãs públicos. A utilização de formatos abertos uniformes também é importante para evitar incompatibilidades com o hardware existente no mercado e utilizado por algumas marcas para exibir conteúdos nos ecrãs através de outro tipo de software.

O HTML5 é uma linguagem *markup* para estruturar e exibir conteúdo para a web. É a quinta revisão da linguagem uniforme HTML que fora criada

em 1990 e uniformizada como HTML4 em 1997 [10]. O principal objectivo consiste em melhorar a linguagem com suporte multimédia, mantendo-o facilmente legível pelos *developers* e consistentemente compreendida por computadores e dispositivos (navegadores web, *parsers*, etc...).

O aparecimento desta linguagem é uma resposta à observação de que o HTML e o XHTML, em uso comum na web, são uma mistura de características introduzidas por várias especificações, juntamente com aquelas introduzidas por produtos de software proprietário tais como navegadores web, *plugins*, e os muitos erros de sintaxe em documentos web existentes.

É, também, uma tentativa de definir uma linguagem de *markup* única, que possa ser escrita com a sintaxe de HTML ou XHTML. Inclui, também, modelos de processamento detalhados para incentivar implementações interoperáveis, que se extende, melhora e racionaliza o *markup* disponível para os documentos, introduzindo *markup* e APIs para aplicações web complexas.

Em particular, o HTML5 introduz uma nova série de características sintácticas, bem como a integração de vectores gráficos escaláveis (SVG). Estas novas características estão desenhadas para facilitar e tratar do conteúdo gráfico e multimédia sem ter que recorrer a plugins e APIs. Novos elementos foram desenhados para enriquecer o conteúdo semântico dos documentos, outros elementos foram alterados, uns redefinidos e/ou uniformizados.

Todos estes passos já alcançados começaram por volta de 2004 com o objectivo principal de alcançar um novo modelo de uniformização. Esse facto deve-se principalmente ao seu antecessor, o HTML4, não ser actualizado desde o ano 2000, e o W3C exclusivamente focalizado no desenvolvimento do XHTML2.

O desenvolvimento do HTML5 provém, também, do interesse de empresas tecnológicas que pretendiam um desenvolvimento de tecnologias e que eram incompatíveis com os browsers existentes [11]. O W3C, na altura, rejeitou esse pedido, o que originou a uniformização das *Web Applications 1.0* e *Web Forms 2.0*. Estas duas especificações foram posteriormente absorvidas pelo HTML5 [12]. A especificação do HTML5 foi adoptada no ponto de início de 2007 pelo novo grupo de trabalho do W3C.

Desde então, tem havido enormes esforços com as empresas de software

proprietário em transformar estes novos elementos, uniformizá-los e eliminar a dependência de documentos web sob esse tipo de elementos, bem como um planeamento detalhado em tornar o HTML5 como W3C *Recommended*.

A definição da linguagem HTML5 encontra-se completa desde Dezembro de 2012 como W3C *Candidate Recommendation* [13], apesar de haver dúvidas em relação ao sucesso do HTML5 como uniformização da linguagem e elementos HTML que não funcionam totalmente nos navegadores web, enquanto esses elementos de software proprietário funcionam sem qualquer problema [14]. Através das várias versões que a linguagem irá sofrer, existe um planeamento para as versões HTML5.1 e HTML5.2 bem como as respectivas datas de uniformização do mesmo. Estima-se que a primeira versão do HTML5 seja W3C *Recommended* por volta de 2014 [15].

O *JavaScript* é uma linguagem de programação open source geralmente utilizada como parte do navegador web para criar interfaces para o utilizador e páginas web dinâmicas.

É uma linguagem multi-paradigma, orientada a objectos, imperativa e funcional baseada através das linguagens de programação *Self* e *Scheme*. Apesar do nome, não existe qualquer relação com a linguagem de programação *Java*, aliás são muito diferentes na sua semântica, e a sua sintaxe é influenciada pela linguagem de programação *C*.

Com o aumento da popularidade da web, as linguagens script e outras linguagens dinâmicas estão a voltar à luz da ribalta, recebendo uma atenção elevadíssima que nunca se verificou desde a sua existência, devido a esta nova geração de programadores que vive à volta deste tipo de linguagens como *JavaScript*, *Perl*, *PHP*, *Ruby*, entre outras.

Devido à simplicidade das tarefas de *scripting* que eram executadas em *JavaScript*, a reputação da mesma não era bem vista pela comunidade de programadores. A sua utilização sempre foi muito limitada, principalmente para simples scripts de conteúdos web e animações.

No entanto, com a introdução da *Web 2.0* e tecnologias como *Ajax*, juntamente com esta nova geração de programadores, a web começa a transformar-se numa plataforma alvo para aplicações de software avançadas (sociais, jogos, produtividade) e as aplicações em *JavaScript* começam a

crescer. Os sistemas de software e aplicações, que eram convencionalmente programados através de linguagens estáticas como *C*, *C++* ou *Java*, começam a ser desenvolvidos com linguagem dinâmica, que era originalmente desenhada para concepção de scripts simples em vez de algo em grande escala.

Este tipo de sistemas e aplicações estão direccionadas à web, para serem utilizados por um vasto número de utilizadores através dos seus navegadores web em vez de um sistema operativo específico. A criação de aplicações *JavaScript* e bibliotecas para o mesmo começam a surgir e continuam a aparecer cada vez mais, para o desenvolvimento de melhores aplicações *JavaScript*, no mesmo sentido a percepção negativa à volta da linguagem começa a dissipar-se e a ser considerada como a mãe das linguagens dinâmicas na web [8].

Uma aplicação web é uma aplicação que é acedida por vários utilizadores na web, normalmente desenvolvida numa linguagem de programação com suporte no browser, como o *JavaScript*. Este tipo de aplicações ficou popular devido à ubiquidade dos navegadores web e à conveniência da utilização dos mesmos como plataforma de aplicação. A capacidade de actualizar e manter aplicações web, sem necessidade de distribuição e instalação de software em potenciais milhares de computadores, é uma das razões chave para a sua popularidade, bem como a capacidade de compatibilidade entre várias plataformas.

Para chegar às aplicações web, os documentos web sofreram grandes alterações. No início, as páginas web não passavam de documentos simples com texto, imagens estáticas e hiperligações. Posteriormente, numa fase que coincide com as empresas a tentar tirar proveito da web através de publicidade e vendas de mercadoria, as páginas web tornaram-se interactivas, com animações, uma proliferação de plugins para animar os conteúdos web, comunicação avançada e pequenos scripts de *JavaScript*. Actualmente, este conjunto de tecnologias web para a criação de aplicações web fica conhecida como *Web 2.0* [2, 7].

A maioria destas tecnologias baseia-se na linguagem de programação dinâmica *Javascript* e estes sistemas são *mashups* (junção de várias tecno-

logias). Algumas empresas (Facebook, MySpace), foram responsáveis pelo aumento de popularidade nas aplicações web com as suas plataformas de desenvolvimento de aplicações.

2.3 Plataformas computacionais para media players

Neste tópico são abordados e analisados um conjunto de sistemas de hardware existentes no mercado, uns mais indicados que outros para o propósito da aplicação. Independentemente do conjunto específico de hardware escolhido, foram analisados separadamente todas as suas camadas, para determinar as suas compatibilidades e limitações, bem como os custos do mesmo em valor monetário e energético. São abordados temas como o ambiente de execução (relativamente ao hardware, dado os seus custos e facilidade de integração num local), a gestão de plataforma web e configurações (esforço necessário para o bom funcionamento da aplicação num ambiente web, *plugins*, hardware necessários, outros) e os navegadores web suportados, dado o foco na web e portabilidade.

2.3.1 Personal Computer

O PC, é um computador de normal utilização, cujo tamanho, capacidade e preço é útil para qualquer índividuo nos dias de hoje. Pode ser composto por uma grande variedade de hardware e raramente apresenta qualquer limitação ao nível dos sistemas operativos existentes, bem como ao nível tecnológico, desde os navegadores web e as linguagens de programação. Este dispositivo é construído de forma a ser robusto, estável e fiável para o utilizador dada a sua utilização diária. Este dispositivo é, certamente, um dos mais robustos e fiáveis na utilização de plataforma de media player, mas as suas dimensões não são as ideais, pois a sua arquitectura e dimensões de hardware seriam uma limitação na colocação de media players.

2.3.2 Small Form Factor

O SFF, é um computador desenhado para minimizar o seu volume. Estão disponíveis na sua maior variedade de tamanhos e formas tornando-os populares dado a sua portabilidade. Apesar da sua reduzida dimensão, suporta as mesmas características de um PC, no entanto está limitado nas suas opções de expansão dado o seu tamanho. Existem várias marcas e modelos deste tipo de dispositivos, mas também existe uma variante deste tipo de computadores COM, isto é, um computador construído numa placa de circuito único, normalmente utilizado em sistemas embutidos dado o seu tamanho físico reduzido e o baixo consumo de energia [16].

2.3.3 Smart TV

As Smart TVs surgiram em equipamentos através da integração da internet com as tecnologias *Web 2.0*, bem como a convergência da tecnologia entre computadores e televisões. Este tipo de aparelhos tem um elevado foco na media online interactiva e menos na transmissão tradicional de media. Este conceito ainda se encontra a emergir na área tecnológica apresentando um custo elevado, com opção de software proprietário ou *open source*. A maioria destes aparelhos está dependente do sistema operativo que executam, ficando dependentes da appstore da plataforma que é instalada [17].

2.3.4 Raspberry Pi

O Raspberry Pi é uma placa de circuito de um computador desenvolvida com o intuito de estimular o ensino básico de ciências de computação nas escolas. As suas especificações consistem numa simples placa mãe, um processador, uma placa gráfica, memória RAM e um cartão de memória. A limitação do hardware nestes dispositivos deve-se a decisões que foram tomadas para manter o hardware a um preço extremamente reduzido e acessível e com um nível básico de funcionalidade [18]. Esta plataforma é conhecida através dos projectos desenvolvidos na área da *Digital Signage*, que é utilizado por empresas como Xibo e Conceito. Relativamente ao

nível dos sistemas operativos, dada a sua arquitectura ARM11, fica limitado aos sistemas operativos que correm nesse tipo de arquitectura (NOOBS, Raspbian "wheezy", Debian "wheezy", Pidora, RISC OS, entre outros...). Dada esta limitação em termos de sistema operativo, existe uma série de navegadores web que funcionam nestes sistemas operativos, mas nenhum relevante em termos de utilização (Arora, Chromium, entre outros).

2.3.5 pcDuino

O pcDuino é uma plataforma de mini PC eficaz e de elevada performance que funciona com sistemas operativos como o Lubuntu (versão mais leve e simples do Ubuntu) e o Android. As suas especificações consistem numa simples placa mãe, um processador, uma placa gráfica, memória RAM, interface de rede e um cartão flash para armazenamento e possui uma saída de interface HDMI para monitores/ecrãs. É especialmente dedicado às comunidades crescentes de *open source* [19]. Relativamente aos navegadores web também existe uma limitação de opções (Google Chrome, Chromium, Firefox) dado a restrição do sistema operativo imposto pelo hardware. Dada a sua especificidade e limitação de hardware é uma solução de custos reduzidos e de baixo custo energético.

2.3.6 Rikomagic

O Rikomagic [20] é um mini PC de dimensões de uma drive. As suas especificações consistem numa simples placa mãe, um processador, uma placa gráfica, memória RAM, interface de rede wireless para permitir ligações à internet ou outros aparelhos wireless, um cartão flash para armazenamento e uma saída de interface HDMI para monitores/ecrãs. Este dispositivo é controlado através de um comando wireless e/ou um rato/teclado wireless com possibilidade de navegação na internet. Este dispositivo limita a escolha de navegadores web dada a restrição de sistema operativo em Android, funcionando apenas com Google Chrome e Firefox. Dada a sua especificidade e limitação no hardware, esta solução tem custos reduzidos e é de baixo custo

energético.

2.3.7 Chromebox

O Chromebox é um dispositivo que surge por origem do Chromebook. São classificados como SFF, este aparelho dispõe de uma porta de rede, USB, DVI, audio, entre outros. Este computador corre originalmente com o sistema operativo Google Chrome OS, tendo sido desenhado para estar ligado à internet e suportar aplicações desenvolvidas na web, ao invés, das aplicações tradicionais que estão alojadas no computador. Como tal, com a restrição do sistema operativo, apenas o navegador web Google Chrome funciona neste sistema operativo, muito pela razão de o sistema operativo ter sido desenvolvido à volta do navegador web da Google. Em termos de custos, o preço é acessível e equilibrado em relação a outros dispositivos SFF [21].

2.3.8 Samsung Cloud Display

O Samsung Cloud Display é um monitor que fornece tecnologia *PC-over-IP*. Este dispositivo é formado por um conjunto de hardware composto por uma placa mãe, memória RAM, placa gráfica, placa de rede, som, entre outros tal e qual como um PC, apenas todo este hardware se encontra embutido no monitor. Relativamente à utilização do sistema operativo, neste dispositivo não foi possível encontrar informação, consequentemente nada pode ser incluído relativamente aos navegadores web, mas os custos são elevados dado a inclusão do hardware no próprio monitor [22].

2.3.9 Análise comparativa

Após a análise possível dos dispositivos de hardware para media player, é necessário identificar e comparar as suas diferenças e limitações para determinar as dependências entre as várias camadas necessárias para o funcionamento de uma aplicação baseada na web. Dada a diversidade de

abordagens e pressupostos, esta análise será feita com base nas seguintes dimensões:

- hardware (nome do dispositivo);
- sistema operativo;
- dependência do sistema operativo com o hardware;
- navegador web.

É feita uma primeira abordagem ao hardware e os seus componentes para analisar a compatibilidade de sistemas operativos em relação a essa plataforma. Através do conhecimento da compatiblilidade dos sistemas operativos nessas plataformas, verifica-se quais os navegadores web que suportam os referidos sistemas operativos e através dessa comparação é possível saber quais os ambientes de execução possíveis nessas plataformas web. Concluindo, a análise destas várias plataformas é fundamental para saber qual a mais adequada para a aplicação a ser desenvolvida.

A tabela 2.1 demonstra as dependências a cada nível dos dispositivos de hardware analisados; enquanto a tabela 2.2 demonstra as dependências dos sistemas operativos relativamente aos navegadores web mais utilizados a nível mundial.

Após efectuar um estudo sobre as dependências de hardware relativamente ao sistema operativo e respectivo navegador web, aliado com as dependências dos navegadores web mais utilizados a nível mundial com os sistemas operativos, é possivel destacar alguns pontos, que poderão influenciar ou servir de base no desenvolvimento da dissertação. Os navegadores web que mais suporte oferecem, relativamente aos sistemas operativos, são o Google Chrome, Firefox e Opera. Relativamente à depêndencia de hardware, e tendo em conta os navegadores web que mais suporte oferecem, é possível destacar as plataformas de PC, SFF e SmartTV.

Hardware	Sistemas Operativos	Navegadores Web
PC	Windows Linux Mac OS X	Internet Explorer Firefox Google Chrome Opera
SFF	Windows Linux Mac OS X	Internet Explorer Firefox Google Chrome Opera
Raspberry Pi	Debian/ARM Linux distro NOOBS Raspbian "“wheezy"” Debian "“wheezy"” Arch Linux ARM Pidora RISC OS Chromium OS Android	Arora Chromium
pcDuino	Lubuntu Android	Chromium Google Chrome Firefox
Rikomagic	Android	Google Chrome Firefox
Chromebox	Chromium OS	Google Chrome
Smart TV	Windows Mac OS X Android	Internet Explorer Firefox Google Chrome Opera
Samsung Cloud Display	Desconhecido	Desconhecido

Tabela 2.1: Dependências de hardware

Navegador web	Sistemas Operativos
Google Chrome	Windows XP SP2+/Vista/7/8 OS X 10.6+ Linux iOS Android
Internet Explorer	Windows 7/8/Server 2012
Firefox	Windows XP/2003/Vista/2008/7/8/2012 OS X 10.6+ Linux
Opera	Windows XP/Vista/7/8 OS X 10.6+ Linux
Safari	OS X 10.7+ iOS

Tabela 2.2: Dependências dos navegadores web

2.4 Players Digital Signage abertos

Em *Digital Signage*, o player é o componente que se encontra directamente ligado ao ecrã e que tem como responsabilidade obter, escalonar e visualizar os conteúdos respectivos. Normalmente, os players dos ecrãs operam como parte de redes de ecrãs públicos de forma isolada, tendo, cada rede, que criar e gerir o seu próprio conteúdo. O aparecimento de aplicações web e o fácil lançamento das mesmas, cria a oportunidade para modelos mais abertos em que um mesmo player poderia ser utilizado no contexto de várias redes. Nesta secção é feita uma análise dos vários players que de alguma forma incluem algum tipo de abertura no seu funcionamento, seja na utilização de formatos de conteúdos standard, seja na disponibilização do respectivo código em modelo open source. Não serão portanto considerados quaisquer sistemas proprietários como os que são mais comuns em sistemas de *Digital Signage*. A única excepção será o player da Ubisign que será utilizado como referência na estruturação da análise comparativa.

Para cada player serão apresentados os vários critérios e elementos comuns entre os mesmos. Serão abordadas as dependências relativamente ao tipo de hardware e sistema operativo e, ou motor web necessários para o

seu funcionamento. Serão analisadas as características de funcionamento do player, tendo em conta o funcionamento da cache, o comportamento no caso de falha de conectividade e a monitorização. Por fim, uma análise ao formato utilizado para as instruções e regras no player, assim como à estrutura geral do funcionamento do player.

2.4.1 Ubisign

A Ubisign [23] disponibiliza um serviço que oferece as ferramentas de software necessárias para a gestão de uma rede de ecrãs, permitindo de forma remota a gestão de conteúdos multimédia. Através deste software escalável e com suporte para tecnologias *web 2.0*, a possibilidade de design de layout, a gestão da rede a partir de qualquer navegador web, a independência no tipo de hardware e a possibilidade de comunicação a fontes externas, via RSS/XML através de um interface amigável pode facilmente disponibilizar nos ecrãs públicos todas as vantagens de informar, entreter e interagir. Relativamente às características, este software permite que o conteúdo seja alocado na cache para manter o seu normal funcionamento em caso de falha na rede, bem como uma aplicação que permite o arranque do player em caso de uma falha no mesmo, mas nada se conhece relativamente ao formato utilizado para o *schedule*.

2.4.2 Xibo

O Xibo [24] é um pacote de software que fornece um *digital signage* de forma gratuita e *open source*. As suas características principais passam por uma gestão de manutenção centralizada através de uma interface web, simples e de fácil lançamento, com um vasto suporte de media, não requer nenhum hardware específico, corre nos sistemas operativos Windows e Linux e suporta RSS.

Está dividido em quatro módulos: o módulo do *'layout'*, responsável pelo layout a ser exibido nos ecrãs; o módulo do *'content'*, responsável pela gestão do conteúdo a ser exibido; o módulo do *'schedule'*, responsável pela

calendarização do conteúdo a ser exibido; o módulo do *'display'* que é responsável pela conjunção dos componentes anteriores e funcionamento do mesmo.

Esta aplicação é do género cliente-servidor. O servidor, é uma aplicação web *PHP/MySQL* que corre em qualquer sistema operativo, é a administração central do interface para os ecrãs. Utilizado também para o upload de conteúdo, layouts, *schedule* e outros. O cliente, é um ecrã (televisor, monitor, projector) que é utilizado para exibir o conteúdo. Cada cliente pode ter o seu próprio layout, conteúdo e *schedule*. Requer uma espécie de computador, ligado ao ecrã para que possa comunicar com o servidor Xibo, para que este possa exibir os conteúdos.

O cliente Xibo existe em duas versões: *.NET* e *Python*, a primeira foi o início do projecto e a mais estável, quanto ao *Python* apresenta grande potencial dada a uniformização com os sistemas operativos e poderá vir a ser o único cliente.

O XML Schema é responsável pela definição do layout, conteúdo e *schedule*. É através deste tipo de ficheiros que é feita a configuração entre o servidor e o cliente.

2.4.3 Risevision

A Risevision [25] é uma empresa dedicada ao desenvolvimento e suporte de ecrãs públicos, de forma gratuita, tendo desenvolvido dois players *digital signage open source* para Windows 7 e Linux, utilizando como navegador web pré-definido o Google Chrome.

Não existe software para instalação ou actualização, apenas necessita de um web browser e através de uma interface web é possível gerar o player. Se os conteúdos (imagens, vídeos, aplicações web) correm num browser, então correm neste player, dispondo de uma plataforma própria de desenvolvimento para aplicações web através da Google API Engine.

Este tipo de software está dividido em várias entidades. Existe um módulo *'Player'*, que é um sistema operativo nativo que corre no *'Viewer'* e é responsável pelas actualizações automáticas do navegador web e os seus

componentes, gestão da cache, hardware, *'Schedule'*, e outros. O *'Display'* é o ecrã, capaz de exibir o conteúdo, podendo ser um LCD, ecrã plasma ou monitor. O *Viewer* é a aplicação que corre no web browser e é exibido no *Display*, é responsável por dinamizar e alterar o conteúdo baseado no *'Schedule'* e *'Presentation'*. O *'Schedule'* é responsável por definir uma sequência de URLs a ser exibido após uma duração definida. O *'Presentation'* é uma página web que contêm um ou vários *'Placeholders'*, que definem a apresentação HTML, layout e o objecto JSON que contém a *'Presentation'* ou até possivelmente uma lista de *'Gadgets'*. O *'Placeholder'* é uma área fixa dentro do *'Presentation'* e finalmente o *'Gadget'* é uma aplicação desenvolvida na plataforma de aplicações fornecida pela Risevision.

2.4.4 Concerto

O Concerto [26] é um software gratuito, *open source* em Linux *kernel-base*, baseado na web para transmitir anúncios digitais através de imagens, vídeos e intermediar uma ligação fácil com a comunidade através da divulgação de alertas, mensagens, publicidade entre outros. Implementado através da linguagem de programação *PHP*, e utilizando *JavaScript* e outras bibliotecas para outro tipo de funcionalidades. É possível, através de uma interface web, efectuar a gestão de layouts, conteúdos media, monitorização e gerir o *schedule* para o ecrã. Necessita de um Concerto SFF, que contém um processador, uma placa gráfica, sem disco rígido e corre uma imagem de um sistema operativo com um cartão de memória. Actualmente, Concerto2 encontra-se em desenvolvimento em *Ruby on Rails*.

2.4.5 AOpen

A AOpen [27] é um software com interface web, utilizado num sistema de ecrãs para publicidade que exibe conteúdo interactivo através de duas versões: HeartTouch, com suporte para o Windows XP, para um fácil acesso e gestão de conta e melhor segurança, sistema indicado para pequenos empresários. É possível exibir informação através de um URL ou de uma

apresentação de um ficheiro PowerPoint. Manualmente, é possível escolher um layout, definir uma imagem de fundo, entre outros. Relativamente à versão AlwaysOpen, fornece um sistema de gestão de um *schedule* através de um calendário com suporte para Windows XP, oferecendo mais capacidade para outros tipos de conteúdo media, RSS *feeds*, gestão de monitorização, gestão remota entre outros. Ambas estas versões de software estão dependentes de dispositivos de hardware desenvolvidos pela AOpen.

2.4.6 IAdea

O software da IAdea [28] é compatível com os standards do SMIL e HTML5. Através de uma interface web é possível efectuar uma gestão remota dos players, o conteúdo media flexível (imagens, vídeos, aplicações web), lançamento fácil, gestão do layout e sensores interactivos. Funciona através de dispositivos SFF proprietários desta marca, com o sistema operativo Linux *kernel-base*. Oferece a possibilidade de adição de widgets HTML5 e funcionamento em modo offline, configuração do *schedule* e duração do conteúdo no ecrã.

2.4.7 Sapo Digital Signage

O código para este player foi desenvolvido internamente na Sapo para a Codebits 2012 sobre aparelhos Raspberry Pi [29]. Este tipo de cliente não permite qualquer tipo de conteúdo a ser alocado localmente, à excepção de *templates*. Este player corre ficheiros JSON como *playlists* que contêm lista de URLs para páginas HTML com vários tipos de conteúdo. O servidor tem capacidade de enviar ao player novo conteúdo em tempo real, bem como novas *playlists*. A comunicação entre o player e o servidor foi *downgraded* de websockets para mecanismos simples de *pooling*.

O player também tem capacidade de correr o seu próprio servidor HTTP, permitindo operações no estado *offline* (no caso do servidor estar desligado, tendo a capacidade de mostrar conteúdo e guardar na cache).

2.4.8 TargetR

O TargetR [30] consiste numa plataforma que apresenta uma solução escalável, fiável, leve e de baixo custo permitindo exibir conteúdo em qualquer equipamento. Através de uma interface web é possível alocar qualquer tipo de conteúdo media (imagens, vídeos), aplicações web de terceiros, fazer a gestão do *schedule* através de canais e conteúdos, entre outros. É um software utilizado para as plataformas de Android, mas há uma versão para o Raspberry Pi, apesar de haver algumas incompatibilidades no que toca a conteúdos media.

2.4.9 Novisign

A Novisign [31] é uma das primeiras empresas a oferecer digital signage baseado em Android e que é possível ligar a qualquer ecrã ou até mesmo uma tablet Android. Através desta aplicação com uma interface web, é possível transmitir para qualquer local, efectuar a gestão do schedule, conteúdos media. Os conteúdos são estáticos (imagens e vídeos), isto é, necessitam de ser enviados para um servidor de forma a poderem ser exibidos nos ecrãs.

2.4.10 OpenSplash

O OpenSplash [32] é uma multi-plataforma gratuita que pode ser utilizada através de qualquer gestão de conteúdo e sistema de escalonamento dada a sua alta extensabilidade através de uma arquitectura de *plugins*. Este player foi desenvolvido para Windows e Linux com os seguintes princípios básicos de conteúdos (vídeos, imagens e texto) em várias zonas do ecrã. Esta solução apenas oferece o player, ficando este responsável pelo pedido de uma lista de conteúdos e schedule alojados num servidor, com capacidade de suporte de vídeos e conteúdos dinâmicos.

2.4.11 Signagelive

O Signagelive [33] é uma aplicação que permite a exibição de conteúdo media através de qualquer navegador web, através dos sistemas operativos Windows e Android com um leque considerável de funcionalidades (suporte para conteúdo dinâmico, diferentes modos de execução, gestão de conteúdos, monitorização, tolerância a faltas, entre outros). Através de um interface web, é possível gerir e monitorizar o player no qual se pretende executar o conteúdo media.

2.4.12 Análise comparativa

Após a análise possível dos players, é necessário identificar e comparar para determinar as dependências e diferenças entre os mesmos, bem como a possibilidade de reutilização de algum player. Dada a diversidade de abordagens e pressupostos, esta análise será feita com base nas seguintes dimensões:

- ambiente de execução (sistema operativo e tipo de aplicação);
- licença do software do player (*open source* ou proprietário);
- formato do *schedule*;
- uniformização do *schedule* (*open source* ou proprietário);
- lista de funcionalidades do software.

É feita uma primeira abordagem ao ambiente de execução, pois existem sistemas que são dependentes de sistemas operativos e, ou até mesmo certos motores web dado as limitações de hardware em que o software é executado. É necessário um estudo do formato do *schedule* para compreender os parâmetros e regras associados nos players, pois podem utilizar uma linguagem uniformizada pelo W3C ou algo proprietário. As características de funcionamento para os players, tais como a exibição do tipo de conteúdo media (estático ou dinâmico); a monitorização remota, para permitir

Players	Ambiente de execução		Licença	Linguagens
	SO	Aplicação		
Ubisign	Windows	WPF	Proprietário	.NET
Xibo	Windows/Linux	Web browser	Open source	.NET e Python
Risevision	Windows/Linux	Google Chrome	Open source	Desconhecido
Concerto	Independente	Web browser	Open source	Ruby on Rails
AOpen	Windows	Aplicação nativa	Proprietário	Desconhecido
IAdea	Windows/Linux	Web browser	Proprietário	Desconhecido
Sapo DS	Linux	Web browser	Open source	Python
TargetR	Android/Linux	Web browser	Proprietário	Desconhecido
Novisign	Android	Web browser	Proprietário	Desconhecido
OpenSplash	Windows/Linux	Web browser	Open source	C#
Signagelive	Windows	WPF	Proprietário	Desconhecido

Tabela 2.3: Ambiente de execução dos players

controlar e monitorizar o estado dos ecrãs espalhados por uma rede; o suporte de funcionamento aquando a ligação à rede é interrompida e o arranque automático de um player, no caso de um falha externa, ao software também são importantes para análise para perceber o seu funcionamento. Concluíndo, o estado de desenvolvimento geral de um player é fulcral, para análise das limitações, características e estudo dos formatos utilizados no seu funcionamento.

A tabela 2.3, demonstra a dependência dos players relativamente ao seu ambiente de execução, tendo em conta os sistemas operativos e em que tipo de aplicação executam, bem como o tipo de software.

A tabela 2.4, mostra a dependência dos players relativamente ao formato *schedule* utilizado bem como a sua uniformização desse mesmo formato.

A tabela 2.5, compara as funcionalidades e características principais dos players.

Apesar do bom leque de players open source existentes no mercado, é de estranhar a inexistência de um player completamente desenvolvido na linguagem dinâmica *JavaScript*, apesar de existirem alguns desenvolvidos em *Python*. Acreditamos que através desta linguagem e no poder que as tecnologias web terão num futuro próximo, bem como com uma maior uniformização da linguagem HTML5, e no desenvolvimento de uma *framework*

Players	Formato	Uniformização
Ubisign	SMIL	Proprietário
Xibo	XML Schema	Open source
Risevision	JSON	Open source
Concerto	JSON	Open source
AOpen	Desconhecido	Proprietário
IAdea	SMIL	Proprietário
Sapo DS	JSON	Open source
TargetR	Desconhecido	Proprietário
Novisign	Desconhecido	Proprietário
OpenSplash	Desconhecido	Proprietário
Signagelive	SMIL	Proprietário

Tabela 2.4: Formato schedule dos players

Players	Monitorização remota	Arranque e recuperação auto	Interface local	Suporte offline
Ubisign	Sim	Sim	Sim	Sim
Xibo	Sim	Sim	Não	Não
Risevision	Sim	Sim	Sim	Sim
Concerto	Sim	Não	Não	Não
AOpen	Sim	Não	Não	Não
IAdea	Sim	Não	Sim	Sim
Sapo DS	Não	Sim	Sim	Sim
TargetR	Sim	Não	Não	Não
Novisign	Sim	Não	Não	Não
OpenSplash	Não	Não	Sim	Não
Signagelive	Sim	Sim	Sim	Sim

Tabela 2.5: Comparação das características e funcionalidades dos players

de comunicação entre o player e o conteúdo a ser exibido, parece-nos que poderia ser possível aumentar o interesse de uma comunidade de *developers* de aplicações para esta rede de ecrãs. É necessário ter em conta o desenvolvimento das características analisadas para poder concorrer paralelamente com estas versões de software, mais concretamente na monitorização remota, a especificação de um formato aberto de schedule, o funcionamento em modo offline quando existe a perda de conectividade e um sistema de arranque automático em caso de falha externa ao player.

2.5 Navegadores e motores web

Os motores web dos browsers surgiram através de uma aproximação modular para separar a camada engine da camada de UI. A separação destas camadas foi fundamental para abrir a possibilidade de implementação do *engine* noutro tipo de aplicações, tais como, clientes de correio electrónico, sistemas de ajuda online, entre outros, e ao mesmo tempo a possibilidade de melhorar e evoluir o interface de utilizador no navegador web sem ter que haver preocupação com o *engine* do mesmo.

Existem vários motores web no mercado, sendo os mais populares WebKit, Trident e o Gecko.

Os navegadores web são constituídos com uma aproximação modular e divididos entre a interface do utilizador e o motor. O motor faz a maior parte das tarefas, essencialmente recebe como argumentos o URL e o espaço rectangular do ecrã, após a execução, devolve o documento correspondente ao URL e exibe a representação gráfica desse documento no espaço rectangular. Também executa tarefas de hiperligações, *cookies*, scripting, plugins e outros. A interface do utilizador fornece uma barra de menu, de endereço, de estado, *bookmark*, histórico e outros. Serve também para embutir o *engine* e servir de interface entre o utilizador e o engine. A vantagem desta aproximação é a facilidade de mais tarde possibilitar o *web browser engine* em vários tipos de aplicações.

O navegador web está a tornar-se cada vez mais uma vasta plataforma

para aplicações web, pois permite o desenvolvimento de mais aplicações web, ao contrário das plataformas tradicionais. Além de aplicações web, são vistas como serviços para o utilizador, consistem em dados, código e outros recursos localizados em qualquer parte do mundo não requerendo qualquer tipo de instalação ou actualização, o que torna o navegador web como uma plataforma de lançamento para estas aplicações. A web adoptou livremente, a uniformização da multi-plataforma, a implementação de software livre e uma extensibilidade distribuída [6].

2.5.1 Plugins

Um *plugin* é um conjunto de componentes de software de terceiros que adiciona capacidades específicas a uma aplicação de software superior, como os navegadores web. Caso seja suportado, os *plugins* permitem a personalização de funcionalidades de uma aplicação, como: reprodução de vídeos, antí-virus e apresentação de tipos de arquivos não suportados pelos browsers. Este tipo de sistema, representou um papel importante no desenvolvimento de aplicações complexas ricas na web. No entanto, com estas capacidades, os plugins apresentam desvantagens. São aplicações que correm dentro do navegador web e consomem recursos adicionais para a sua execução, estão mais expostos a mais ataques vulneráveis e são baseados em tecnologia proprietária tornando difícil o controlo do seu suporte entre os vários navegadores web e sistemas operativos. Felizmente, o aparecimento do HTML5 oferece capacidades similares aos vários plugins existentes. Esta tecnologia tem um grande suporte em todo os navegadores web, tornando possível aos developers, o desenvolvimento de aplicações em todos os navegadores web com a mesma linguagem e sem preocupações de código adicional relativamente a *frameworks* e dependências de terceiros [34].

2.5.2 WebKit

O código que deu origem ao WebKit começou em 1998 sob o nome de KDE HTML (KHTML) engine e KDE JavaScript (KJS). Este projecto começou

na Apple em 2001 como uma variante do KHTML com a explicação que esta tecnologia permitia um desenvolvimento mais fácil comparado com as tecnologias existentes. Com uma biblioteca adaptada ao OS X, esta variante do KHTML foi renomeada WebCore e JavaScriptCore [35].

No entanto, a troca de código e colaboração entre ambas as partes foi complicada, pois o código base divergia e ambas as partes tinham diferentes aproximações em relação ao código [36].

Eventualmente, foi atingido um ponto de ruptura entre KHTML e a Apple e ambas as partes decidiram tomar caminhos diferentes [37]. Após esta separação, a Apple divulgou todo o código relativamente ao WebCore no qual resultou uma nova reaproximação entre ambas as partes e numa estreita colaboração [38]. Resultante da divulgação do código pela parte da Apple e a reaproximação e colaboração foi criada uma equipa de nome WebKit [39].

Actualmente, encontra-se em constante desenvolvimento por equipas KDE, Apple, Nokia, RIM, entre outros. Os navegadores web mais famosos que utilizam este motor web são o Google Chrome e o Apple Safari e é desde Novembro de 2012 o motor web mais utilizado no mercado.

2.5.3 Trident

Trident é o motor web proprietário da Microsoft Windows para o browser Internet Explorer. Foi apresentado em 1997 para a versão 4 do Internet Explorer e tem vindo a ser constantemente actualizado, continuando a ser utilizado nos dias de hoje. A Microsoft tem vindo a fazer alterações para melhorar a uniformização de componentes da web e suporte para novas tecnologias, pretendendo melhorar significativamente o motor web para ser mais competitivo e moderno em comparação com outros.

2.5.4 Gecko

O Gecko é um motor web *open source*, utilizado por várias aplicações da Mozilla, e tem como navegador web o Firefox.

Foi desenhado para suportar uniformização de formatos abertos da Internet por diferentes aplicações para exibir páginas web e, em alguns casos, interfaces para aplicações através duma API rica que serve para uma vasta variedade de aplicações com *internet-enabled*.

O desenvolvimento deste motor web começou em 1997 pela Netscape, originalmente concebido para o Netscape Navigator 1.0 e actualizado ao longo dos anos que sempre foi considerado inferior ao Trident da Microsoft. Era lento e não respeitava as regras de uniformização W3C com várias limitações e falhas numa série de características.

Dado o número elevado de falhas, um novo motor web estava a ser desenvolvido paralelamente ao antigo, com a intenção de o integrar no Netscape quando se encontrasse estável.

Após o lançamento do projecto Mozilla em 1998, o novo motor web foi lançado sob licença open source com o nome Raptor, mais tarde alterado para NGLayout devido a problemas de marca [40]. Posteriormente, a Netscape mudaria o nome de NGLayout para Gecko, anunciando que o navegador web utilizaria Gecko como engine em substituição, evitando grandes alterações a ser desenvolvidas.

2.5.5 Presto

Presto é o motor web proprietário do navegador web Opera. Apresentado em 2003, para o sistema operativo Windows, continuou o seu desenvolvimento efectuando alterações para melhorar a performance e uniformização de componentes web. Uma das qualidades deste motor web é o facto de ser dinâmico, no qual as páginas web podem ser re-renderizadas completa ou parcialmente através de eventos do DOM, permitindo visualizações dos elementos da página independentemente da execução dos scripts [41].

2.5.6 Visão geral e análise comparativa

O browser convencional evoluiu das páginas estáticas de HTML para plataformas que executam aplicações web, independentemente de qualquer

Navegador web	Motor web	Utilização mundial
Google Chrome	WebKit	36,5%
Internet Explorer	Trident	30,7%
Firefox	Gecko	21,4%
Safari	WebKit	8,3%
Opera	Presto	1,2%
Outros	-	1,9%

Tabela 2.6: Comparação na utilização dos navegadores web

plataforma ricas em características. Esta transição obrigou o navegador web a ter capacidade de aguentar estas tecnologias mantendo a performance e compatibilidade [2].

Apesar desta proliferação de novas tecnologias web, a compatibilidade entre navegadores web melhorou, havendo aspectos onde ainda são diferentes, dado os tipos de implementação e performance.

A web tornou-se num ambiente de lançamento para vários tipos de aplicações web, e paralelamente os *plugins* têm o seu *downside*, dado o conteúdo na aplicação ter a possibilidade de não estar disponível noutro tipo de navegador web, pois são componentes de software necessários para a operabilidade de certas funcionalidades (vídeos, anti-vírus, etc) nas aplicações, o mesmo acontece para os web widgets; pois a combinação da utilização de métodos convencionais, rígidos e práticas explícitas de instalação, com linguagens e ferramentas que resultam em aplicações que executam lentamente e muitas vezes difíceis de manter, como tal as aplicações web são desenvolvidas com o intuito de uniformização entre os browsers.

Após uma análise dos motores web existentes no mercado, é necessário identificar a relação entre o motor web e o navegador web, bem como o grau de utilização destes browsers a nível mundial. A tabela 2.6 demonstra a relação entre os motores web utilizados e os navegadores web, bem como as suas versões actuais e utilização mundial dos mesmos até à data (Janeiro 2013).

Efectuado o levantamento sobre o nível de utilização dos motores web existentes no mercado, é possível destacar que o Google Chrome e o Safari são os navegadores web com a maior taxa de utilização. No entanto, é

necessário ter em conta o nível de utilização do Internet Explorer e o Firefox, pois um dos objectivos desta dissertação é a capacidade de uniformização da aplicação nos vários navegadores web.

2.6 Resumo

Neste capítulo foi efectuado uma análise ao trabalho relacionado e respectivas tecnologias web, abordando trabalho que foi analisado segundo um conjunto de perspectivas de forma a demonstrar o relacionamento com a dissertação, seguido de uma análise às plataformas computacionais, aos players *digital signage* e aos motores web existentes no mercado, bem como as respectivas comparações e limitações dos mesmos.

3
Estudos técnicos

Este capítulo apresenta uma série de estudos técnicos que foram realizados para perceber a capacidade das tecnologias em relação às funcionalidades descritas para o player. Também é feita uma breve análise ao funcionamento dos motores web, conjuntamente com o *JavaScript* para compreender o modelo de execução do código através dos navegadores web. Por fim, é feita uma série de testes para determinar a plataforma de execução, que ofereça as capacidades necessárias à implementação das funcionalidades descritas no capítulo anterior.

3.1 Suporte para cache na web

Para permitir que o player exiba conteúdos de uma forma rápida e ofereça melhor experiência ao utilizador, foi necessário perceber o funcionamento das caches ao nível do navegador web. Como tal, foi feito um estudo que visa explorar o funcionamento das caches do navegador web, perceber a importância da sua utilização, regras e respectivo funcionamento.

As web caches situam-se entre um ou mais servidores web (também conhecidos como servidores de origem) e um ou vários clientes, nos quais os pedidos efectuados passam pela cache e guarda as cópias dessas respostas, como páginas HTML, imagens e ficheiros (colectivamente conhecidos como

representações), para si mesmo. Caso exista um outro pedido para o mesmo URL, pode utilizar a resposta previamente guardada ao invés de voltar a pedir o mesmo ao servidor de origem. As duas principais razões pelas quais as web caches são utilizadas são:

- redução da latência - como o pedido é satisfeito pela cache (que se encontra próxima do cliente) em vez do servidor de origem, leva menos tempo para obter as representações e exibi-las. Este processo dá uma experiência de tornar a web mais responsivo;
- redução do tráfego de rede - como as representações são reutilizadas, reduz a quantidade de largura de banda utilizada pelo cliente. Isto economiza dinheiro se o cliente tiver que pagar pelo tráfego, e mantém as necessidades da largura de banda a um nível inferior, bem como uma melhor gestão do mesmo.

Através das preferências de qualquer navegador web moderno, é possível encontrar as propriedades para cache. Através deste tipo de propriedade, é possível reservar uma parte do disco rígido do computador, para guardar as representações que foram vistas. A cache do navegador web actua de acordo com regras bastante simples. Geralmente, por sessão, verificará se as representações se encontram *fresh*, isto é, se as representações não foram alteradas e continuam válidas.

Uma das preocupações em relação à utilização de caches provém do receio na perda do controlo da página web, bem como servir conteúdo que está desactualizado ou obsoleto. Por outro lado, se a página web for concebida correctamente, as caches podem ajudar na velocidade em economia de tempo e espaço nas ligações aos servidores. A diferença entre uma página que dificulta o processo de cache pode demorar vários segundos, em relação a outra que aproveita as vantagens de cache, comparativamente pode parecer instantânea, levando a uma melhor experiência para o utilizador [42].

O facto é que as caches são sempre utilizadas, independentemente da forma em que as páginas web estão concebidas, como tal, se não configurar a página a efectuar a cache correctamente, será armazenada de igual forma utilizando os valores prédefinidos.

3.1.1 As regras

Como já foi enunciado, a cache do browser actua segundo regras simples que são usadas para determinar quando servir uma representação da cache, se estiver disponível. Algumas destas regras são definidas nos protocolos (HTTP 1.0 e 1.1), e outras definidas pelo administrador da cache (ou o utilizador da cache do navegador web, ou o administrador do *proxy*).

As regras mais comuns, são as seguintes:

1. se o cabeçalho da resposta disser à cache para não manter a representação, não será mantido;
2. se o pedido for autenticado ou seguro (HTTPS), não será armazenado na cache;
3. uma representação em cache é considerada *fresh* (isto é, ser capaz de ser enviado ao cliente sem verificar com o servidor de origem) se:
   (a) tem um tempo de expiração ou outro controlador de idade, e ainda está dentro do período *fresh*, ou;
   (b) se a cache viu a representação recentemente, e foi modificada relativamente há muito tempo;
4. representações *fresh* são servidas directamente a partir da cache, sem verificar com o servidor de origem;
5. se uma representação é obsoleta, uma solicitação é efectuada ao servidor de origem para validá-la, ou indicar à cache se a cópia que possui ainda é válida;
6. sob certas circustâncias, por exemplo, quando desconectado da rede, a cache pode servir representações obsoletas sem verificar com o servidor de origem.

Se nenhum *validator* (*E-Tag* ou *Last-Modified header*) estiver presente na resposta e não obtendo qualquer informação explícita de *freshness*, será normalmente, mas nem sempre, considerada *uncacheable*. Juntos, *freshness*

e *validation*, são as formas mais importantes para uma cache trabalhar com conteúdo. Uma representação *fresh* estará sempre disponível instantaneamente a partir da cache, enquanto uma representação validada vai evitar o envio de toda a representação de novo se esta não mudou [42].

3.1.2 HTML meta tags e HTTP headers

Existem duas maneiras de controlar as caches: através de HTML *meta tags* e *HTTP headers*.

As *meta tags* são utilizadas na secção *<HEAD>* do documento que descreve os atributos, normalmente utilizadas para marcar como *uncacheable* ou a expirar após um certo período de tempo. Apesar da facilidade de utilização, não são bastante efectivas, porque são poucos os browsers que obedecem a essas regras.

Por outro lado, os *HTTP headers* oferecem um maior controlo sobre a forma como as caches lidam com as representações. Os atributos não podem ser vistos no HTML, e são normalmente gerados pelo servidor web, no entanto, é possível controlá-los num certo grau, dependendo do servidor que é utilizado [42]. Seguidamente são abordados os *HTTP headers* mais relevantes:

- *Expires*;
- *Cache-Control*;
- *Last-Modified*;
- *If-None-Match*.

Os *HTTP headers* são enviados pelo servidor antes do HTML, e só são vistos pelo navegador web e caches intermediárias. Uma resposta típica de *HTTP headers* pode ter esta aparência:

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
```

```
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 30 Oct 1998 14:19:41 GMT
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```

3.1.3 Os controladores da cache HTTP headers

3.1.3.1 Expires HTTP header

O *Expires HTTP header* é o meio mais básico de controlar as caches através da indicação do período de tempo no qual a representação associada é considerada *fresh*. Após esse período de tempo, as caches irão sempre verificar, com o servidor de origem, se a representação foi alterada.

A maioria dos servidores web permite definir as respostas ao *Expires header* de várias maneiras. Normalmente, permitem definir um tempo absoluto para expirar, baseado na última vez em que o cliente obteve a representação (tempo do último acesso), ou na última vez em que o documento foi alterado no servidor (última modificação).

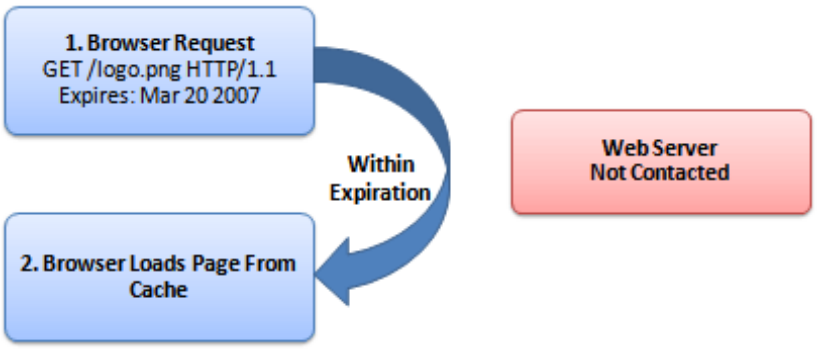


Figura 3.1: Sequência de passos de um HTTP Expires

O único valor válido no *Expires header* é uma data HTTP, conforme o seguinte exemplo:

```
Expires: Fri, 30 Oct 1998 14:19:41 GMT
```

Apesar da sua utilidade, este *header* tem algumas limitações. Primeiro, porque existe uma data envolvida, os relógios no servidor web e a cache precisam de estar sincronizados; se ambos tiverem a ideia errada da data, a cache não funcionará correctamente. Outro dos problemas tem a ver com a facilidade de actualizar a data do *Expires HTTP header*, caso isso se verifique, todos os pedidos serão sempre efectuados ao servidor, aumentando a carga e latência no mesmo [43].

3.1.3.2 Cache-Control header

O HTTP 1.1 introduziu uma nova classe de *headers*, os *Cache-Control headers*, para oferecer mais controlo sobre o conteúdo e resolver as limitações do *Expires*.

Os *Cache-Control headers* incluem as seguintes propriedades:

- *max-age=[segundos]* - especifica a quantidade máxima de tempo que a representação será considerada *fresh*. Similar ao *Expires*, esta propriedade é relativa ao tempo do pedido, ao invés do tempo absoluto. *[segundos]* é o número de segundos a partir do momento do pedido no qual se deseja que a representação seja *fresh*;

- *s-maxage=[segundos]* - similar ao *max-age*, mas só se aplica a caches partilhadas;

- *public* - marca respostas autenticadas como *cacheable*; normalmente se a autenticação HTTP é necessária, as respostas são automaticamente privadas;

- *private* - permite caches que são específicas para um utilizador (por exemplo, num browser) a armazenar a resposta; caches partilhadas não podem;

- *no-cache* - força a cache a submeter pedidos ao servidor de origem para validação antes de libertar uma cópia. Isto é útil para assegurar que a autenticação é respeitada, ou para manter *freshness*, sem sacrificar todas as vantages da cache;

- *no-store* - instrui as caches a não manter uma cópia da representação sob quaisquer condições;
- *must-revalidate* - indica à cache que deve obedecer a qualquer informação de *freshness* que é fornecida àcerca de uma representação. O HTTP permite que a cache sirva representações obsoletas sob condições especiais;
- *proxy-revalidate* - semelhante ao must-revalidate, excepto que só se aplica às caches de *proxy*.

Quando *Cache-Control* e *Expires* estão presentes, *Cache-Control* tem precedência [42].

3.1.3.3 Last-Modified header

O *validator* mais comum é o tempo em que o documento foi alterado, tal como é indicado no *Last-Modified header*. Quando uma cache tem uma representação armazenada que inclui um *Last-Modified header*, pode ser utilizado para pedir ao servidor se a representação mudou desde a última vez em que foi visto [43].

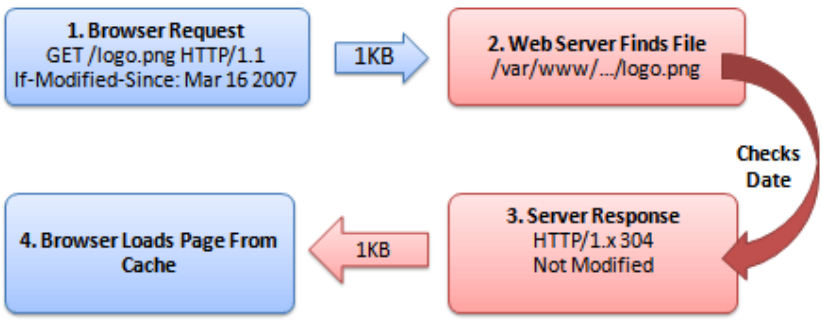


Figura 3.2: Sequência de passos de um HTTP Last-Modified

```
1. Browser: Obter logo.png, só se foi modificado após Mar 16, 2007.
2. Servidor: (Verifica data da última alteração)
3. Servidor: Não foi modificado desde essa data.
4. Browser: Exibe a versão que está na cache.
```

3.1.3.4 If-None-Match header

ETag's são identificadores únicos que são gerados pelo servidor e alterados todas as vezes que a representação muda. Como o servidor controla como a *ETag* é gerada, as caches têm a certeza que o *ETag* corresponde quando é feito um pedido *If-None-Match*, a representação é realmente idêntica [43].

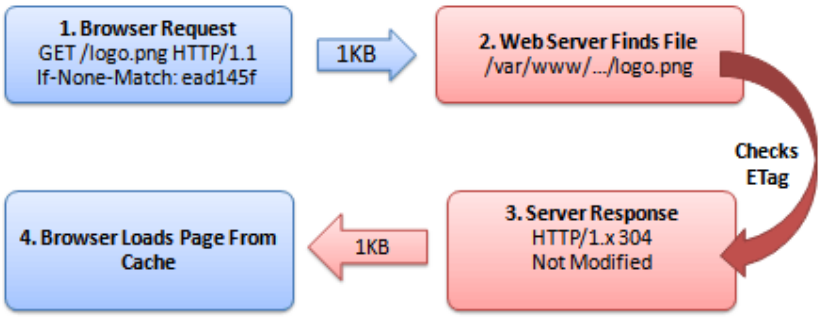


Figura 3.3: Sequência de passos de um HTTP If-None-Match

```
1. Browser: Obter logo.png, se a tag não for identical a “ead145f”?
2. Servidor: (Verifica ETag do logo.png)
3. Servidor: A versão da ETag é “ead145f”. Não foi modificada.
4. Browser: Exibe a versão que se encontra na cache.
```

3.2 Suporte para cache de vídeos em browsers

Para permitir que o player exiba vídeos sem qualquer tipo de espera ou interrupção, mantendo o estado de execução, foi necessário explorar o funcionamento de cache em algumas plataformas de vídeo, explorando o funcionamento da cache para cada plataforma específica através dos respectivos *HTTP headers* que cada vídeo contém.

Foram estudadas três plataformas de vídeo (Youtube, Metacafe e Vimeo), nas quais é feita uma análise ao *HTTP header* de cada vídeo, escolhido aleatoriamente, e registar os dados observados para compreender como é feito a cache dos vídeos.

Filename	1361730381_50e8c18c96e3ba089d506496f95fd2e0.mp4
URL	http://vl.mccont.com/ItemFiles/%5BFrom%20www.metacafe.com%5D%209901500.25879189.4.mp4?__gda__=1361730381_50e8c18c96e3ba089d506496f95fd2e0
Content type	video/mp4
File size	14.110.626
Last accessed	24-02-2013 16:26:34
Server time	24-02-2013 16:26:22
Server last modified	15-02-2013 07:46:28
Expire time	01-03-2013 18:19:18
Server name	Footprint Distributor V4.8
Server response	HTTP/1.1 200 OK
Content encoding	-
Cache name	f_000621
Cache control	max-age=604800
ETag	"d74fa2-4d5be92aa9d00"

Tabela 3.1: Recolha de dados do HTTP Header da plataforma de vídeo Metacafe

Recorrendo à análise da tabela 3.1, é possível verificar que, na plataforma de vídeo Metacafe, apesar de ter um *Expire Time* superior ao *Last Acessed*, também contêm um *Cache-Control* que atribiu um tempo máximo de *freshness* na cache.

Na plataforma Youtube, apesar do *Expire Time* ser idêntico ao *Server Time* para forçar a cache a *freshness*, o *Cache Control* toma precedência e está no modo *private*, com um tempo máximo definido no *max-age*, conforme se pode verificar na tabela 3.2.

Finalmente, podemos verificar na tabela 3.3 que a cache é efectuada pelo *If-None-Match header*, isto é, é efectuado sempre um pedido ao servidor de origem para verificar se a tag se encontra inalterada. Caso isso aconteça, é exibido a cópia que se encontra em cache.

Resumindo - esta análise permitiu verificar que em algumas plataformas, o processo de cache funciona de maneira diferente sendo algo que não podemos controlar directamente, pois encontra-se nos respectivos *HTTP header* de cada elemento. Uma das formas de controlar esta situação seria a

Filename	algorithm=throttle-factor&burst=40&cp=U0hVRldOVl9KU0NONV9PSFpK
URL	http://r3—sn-2vgu0b5auxaxjvh-v2ve.c.youtube.com/videoplayback?algorithm=throttle-factor&burst=40&cp=U0hVRldOVl9KU0NONV9PSFpKOmZoNmJPeF9qTEtE&cpn=rm_st8J6n5j5mGby&expire=1361738959&factor=1.25&fexp=930503%2C920704%2C912806%2C902000%2C922403%2C922405%2C929901%2C913605%2C925006%2C931202%2C908529%2C920201%2C930101%2C906834%2C901451&id=58bd5fa10283dce2&ip=188.83.128.15&ipbits=8&itag=34&keepalive=yes&key=yt1&ms=au&mt=1361716876&mv=m&newshard=yes&range=13-1783807&ratebypass=yes&signature=4E6B153A6F89A7D769874C41E03F2F7A8F480B71.BF42E9062DB731B55C29961F5A2E8442B26C0F6&source=youtube&sparams=algorithm%2Cburst%2Ccp%2Cfactor%2Cid%2Cip%2Cipbits%2Citag%2Csource%2Cupn%2Cexpire&sver=3&upn=MVq8qKVls24
Content type	video/x-flv
File size	654.805 (máximo observado - 1.781.760 [1,7Mb])
Last accessed	24-02-2013 14:43:20
Server time	24-02-2013 14:42:15
Server last modified	23-11-2009 03:15:27
Expire time	24-02-2013 14:42:15
Server name	gvs 1.0
Server response	HTTP/1.1 200 OK
Content encoding	-
Cache name	f_00055e
Cache control	private, max-age=21724
ETag	-

Tabela 3.2: Recolha de dados do HTTP Header da plataforma de vídeo Youtube

Filename	aktimeoffset=0&aksessionid=a515cdc112802a1bf99799e578199894&to.mp4
URL	http://av.vimeo.com/23283/149/18375352.mp4?aktimeoffset=0&aksessionid=a515cdc112802a1bf99799e578199894&token=1361729588_cae13d5625dd8785cc2cf494a7470eb5
Content type	video/mp4
File size	23.011.094
Last accessed	24-02-2013 16:13:30
Server time	24-02-2013 16:13:09
Server last modified	01-04-2010 06:26:52
Expire time	-
Server name	Apache
Server response	HTTP/1.1 200 OK
Content encoding	-
Cache name	f_0005f0
Cache control	-
ETag	"1becab4a24c6b1748e25e44f44b3526d:1270103212"

Tabela 3.3: Recolha de dados do HTTP Header da plataforma de vídeo Vimeo

implementação de um proxy, mas é necessário ter em conta os direitos de autor e possíveis violações de lei, na implementação desta solução.

Este estudo foi realizado em Mac OS X 10.5.8, utilizando os navegadores web Safari 5.0.6, Google Chrome 21.0.1180.90 e Mozilla Firefox 16.0.2 através da ferramenta Firebug e/ou File Juicer; e Windows 7, utilizando o Google Chrome 25.0.1364.97 através da ferramenta *Firebug* e também as aplicações *ChromeCacheViewer* e *VideoCacheViewer*, sendo a última ideal para visualizar o vídeo alojado na cache.

3.3 Aplicações web em modo offline

De forma a permitir que o player continue a exibir conteúdo no caso da ligação à internet ser interrompida, foi necessário estudar o funcionamento de aplicações web no estado offline. Para explorar as capacidades de cache no navegador web, a estrutura e o funcionamento deste tipo de aplicações,

como alternativa ao estado de execução de conteúdo, bem como o estado de execução do próprio player de forma a assegurar o seu funcionamento sem interrupções, independentemende do estado da ligação.

Uma aplicação web em modo offline [44] é constítuida por uma lista de URLs (HTML, CSS, *JavaScript*, imagens, entre outros), na qual o index dessa página aponta para um ficheiro de *manifest*. O browser lê essa lista de URLs declarada no ficheiro *manifest*, efectua o download dos recursos declarados, efectua a cache localmente e mantêm automaticamente uma cópia dos recursos até que estes sejam alterados. Caso a aplicação seja acedida sem uma ligação à rede, o browser irá exibir as cópias alojadas localmente.

Através deste processo, a aplicação web em modo offline oferece três vantagens:

- navegação em modo offline - os utilizadores podem navegar na aplicação quando estiver offline;
- velocidade - os recursos estão armazenados localmente em cache;
- carga do servidor reduzida - o navegador web só irá transferir recursos do servidor quando o ficheiro do *manifest* for alterado.

Mas também existem algumas desvantagens:

- restrições de espaço - alguns navegadores web colocam restrições à quantidade de espaçamento para a aplicação;
- dependência dos recursos - no caso de o ficheiro de manifest ou um recurso específico estiver indisponível, o processo inteiro da cache falha.

3.3.1 O processo de criação

Como já foi referenciado no tópico anterior, este tipo de aplicação é criado à volta da lista que se encontra no ficheiro do *manifest*. Como tal, o

processo de criação [45] está dividido em duas fases: a referenciação ao ficheiro do *manifest* e à estrutura do mesmo.

Para referenciar o ficheiro de *manifest*, utiliza-se um atributo no elemento <*html*> da seguinte maneira:

```
<html manifest="/example.appcache">
  ...
</html>
```

O atributo *manifest* deve ser incluído em todas as páginas da aplicação, caso contrário o navegador web não efectuará a cache (a menos que esteja explicitamente listado no ficheiro *manifest*). Este pode indicar para um URL absoluto ou um caminho relativo, mas se for um URL absoluto deve estar sob a mesma origem que a aplicação web e deve ser servido com o *Content-Type text/cache-manifest*, através do ficheiro *default* do servidor web ou configuração *.htaccess*.

Por exemplo, para servir este tipo de ficheiro em *Apache*, basta adicionar a seguinte linha ao ficheiro de configuração:

```
AddType text/cache-manifest .appcache
```

Concluída a primeira fase, é necessário agora perceber o que realmente contem o ficheiro *manifest* e como é constituido.

A primeira linha de cada ficheiro *manifest* é a seguinte:

```
CACHE MANIFEST
```

Após a primeira linha, todos os ficheiros de manifest são divididos da seguinte maneira:

- *CACHE* - os ficheiros listados debaixo deste cabeçalho (ou imediatamente após *CACHE MANIFEST*) serão explicitamente armazenados em cache após o download;
- *NETWORK* - os ficheiros listados nesta secção são considerados recursos que necessitam de ligação ao servidor;

- *FALLBACK* - uma secção opcional que especifica as páginas fallback no caso de um recurso estar inacessível. O primeiro endereço é o recurso, o segundo é o *fallback*. Ambos os endereços devem ser relativos e da mesma origem que o ficheiro *manifest*.

Há também uma quarta secção (*SETTINGS*) que especifica as configurações para o comportamento da aplicação, que é raramente utilizado, existindo de momento as duas definições seguintes:

- *prefer-online* - indica ao navegador web que os dados armazenados em cache não devem ser utilizados se existir uma ligação à rede activa;
- *fast* - valor pré-definido. Utiliza a cache mesmo com uma ligação à rede activa.

Estas secções podem ser listadas em qualquer ordem e cada secção pode aparecer em mais que um ficheiro *manifest*. No caso de não existir nenhuma secção no ficheiro, todos os recursos listados estão implicitamente na secção *CACHE*.

Após compreensão da estrutura do ficheiro *manifest*, fica um exemplo do mesmo:

```
CACHE MANIFEST
# 2010-06-18:v2

# Explicitly cached 'master entries'.
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Resources that require the user to be online.
NETWORK:
login.php
/myapi
http://api.twitter.com
```

```
# static.html will be served if main.py is inaccessible
# offline.jpg will be served in place of all images in images/large/
# offline.html will be served in place of all other .html files
FALLBACK:
/main.py /static.html
images/large/ images/offline.jpg
*.html /offline.html
```

De notar que as linhas no ficheiro a começar com a letra *'#'*, são linhas de comentário no ficheiro, mas também são utilizadas para outro propósito. Por exemplo, no caso de alterar uma imagem ou ficheiro que esteja contido na lista de recursos do *manifest*, essas alterações não serão armazenadas; isto porque o navegador web apenas verifica alterações ao próprio ficheiro do *manifest* e não aos recursos contidos no mesmo. Como tal, é obrigatório editar o ficheiro *manifest*, por exemplo, através de uma linha de comentário e respectiva versão conforme se verifica no exemplo anterior.

Depois de a aplicação entrar em modo offline, permanece em cache até que uma das seguintes situações se constate:

- o utilizador apaga o espaço de armazenamento dedicado ao navegador web;
- o ficheiro de manifest é modificado;
- a cache da aplicação é actualizada automaticamente através da programação de um script, recorrendo ao objecto *applicationCache*.

3.3.2 O fluxo de eventos

Quando o navegador web visita uma página que contenha um ficheiro *manifest*, são disparados vários eventos no objecto da *applicationCache*. Detalhadamente, este é o fluxo de eventos [46] que ocorrem:

1. assim que percebe que existe um atributo *manifest* no elemento *<html>*, o navegador web dispara o evento *checking*. O evento *checking* é sempre disparado, independentemente se a página em questão

tiver sido visitada anteriormente ou outra página qualquer que indique o mesmo ficheiro *manifest*;

2. se o navegador web nunca viu o ficheiro *manifest*:
   (a) o evento *downloading* dispara, e começa a fazer download dos recursos listados no *manifest*;
   (b) enquanto efectua o download, o browser irá periodicamente disparar eventos de *progress*, que contêm informações sobre quantos ficheiros já terminaram o download e quantos se encontram em espera;
   (c) depois de todos os recursos listados no *manifest* terminarem o download com sucesso, o browser dispara um ultimo evento, *cached*. Este é o sinal que a aplicação está armazenada na cache e pronto a ser utilizada em modo offline;
3. se o navegador web já visitou a página ou qualquer outra página que aponta para o mesmo ficheiro *manifest*, o mesmo já conhece a cache relativa ao *manifest*, podendo conter já alguns recursos ou até mesmo a aplicação completa. Neste ponto, questiona se o manifest foi modificado deste a última vez que foi consultado.
   (a) Se a resposta for negativa, então o navegador web dispara um evento *noupdate* e exibe a aplicação.
   (b) Se a resposta for positiva, então o navegador web dispara um evento *downloading* e inicia o download de todos os recursos listados no *manifest*.
   (c) Enquanto efectua o download, o browser irá periodicamente disparar eventos de *progress*, que contêm informações sobre quantos ficheiros já terminaram o download e quantos se encontram em espera.
   (d) Depois de todos os recursos listados no *manifest* terminarem o download com sucesso, o browser dispara um último evento, *updateready*. Este é o sinal que a nova versão da aplicação está

armazenada na cache e pronta a ser utilizada em modo offline. No entanto, para ser exibida é necessário executar o método *swapChange()* no objecto da *applicationCache*.

O navegador web dispara um evento *error* e pára, se algo de errado acontece durante o fluxo de eventos. A seguir, uma lista de eventos, que podem hipoteticamente originar erros:

- o ficheiro do *manifest* devolve um erro HTTP 404 (*page not found*) ou 410 (*permanently gone*);
- o ficheiro *manifest* foi encontrado e estava inalterado, mas a página HTML que apontava para o mesmo falhou no download;
- o ficheiro *manifest* foi alterado quando uma actualização decorria;
- o ficheiro *manifest* foi encontrado e estava modificado, mas o browser falhou a efectuar o download um dos recursos listados no *manifest*.

3.3.3 Suporte e visão global

Até à data (Março de 2013) [47], todos os navegadores web modernos suportam esta funcionalidade conforme se demonstra na figura 3.4, bem como as vantagens e desvantagens na tabela 3.4.

Offline web applications - Working Draft

Method of defining web page files to be cached using a cache manifest file, allowing them to work offline on subsequent visits to the page

*Usage stats:	Global
Support:	68.67%
Partial support:	0.1%
Total:	68.77%

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser
								2.1	
								2.2	
						3.2		2.3	
						4.0-4.1		3.0	
	8.0					4.2-4.3		4.0	
	9.0	18.0	24.0	5.1		5.0-5.1		4.1	
Current	10.0	19.0	25.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0
Near future		20.0	26.0		12.5				10.0
Farther future		21.0	27.0						

Figura 3.4: Suporte de aplicações web offline nos navegadores web

Vantagens	Desvantagens
Suporte nos browsers modernos;	Se um recurso falha no download, todo o processo de cache falha;
Simples API;	Todos os recursos têm que ser declarados no ficheiro do manifest;
Simples fluxo de eventos;	Obrigatório modificar o ficheiro do manifest para o navegador web efectuar alterações que existam na aplicação;

Tabela 3.4: Visão global das aplicações web offline

3.4 HTML5 Storage

De forma a permitir que o player tenha capacidade de guardar dados dos conteúdos e do player, localmente no navegador web, mantendo o seu estado de execução e uma capacidade rápida de resposta, foi necessário explorar as APIs que o HTML5 oferece, dado os vários tipos de armazenamento, oferecendo a possibilidade de armazenamento de conteúdo, seja do player ou do conteúdo a ser exibido.

Sendo assim, foi necessário abordar este tópico, estudar e compreender as categorias no qual estão sub-divididos os tipos de armazenamento em HTML5:

- *Web storage*;
- *IndexedDB*;
- *File storage*.

3.4.1 Web storage

Web storage é uma API do HTML5 que oferece benefícios importantes sobre os *cookies* tradicionais. Apesar da especificação se encontrar em W3C *draft status* [48], a maioria dos browsers modernos já suportam esta funcionalidade.

O web storage é um complemento e reforço à limitação que os *cookies* [49] têm. Os pontos fortes do web storage são os seguintes:

- o espaço que providencia é muito superior (a maioria dos browsers permite 30 a 50 *cookies* com um tamanho máximo de 4KB, enquanto o web storage oferece entre 5 a 10MB), dependendo do navegador web;
- fácil utilização dado a simples API de partes chave/valor;
- persistência dos dados no browser, ao contrário dos *cookies* em que os valores são enviados para servidores web remotos.

Os dados que estão sendo armazenados podem ser acedidios utilizando *JavaScript*, proporcionando a vantagem e a capacidade de executar scripting do lado do cliente; permitindo a execução de inúmeras tarefas que tradicionalmente são executadas do lado do servidor e bancos de dados relacionais.

Existem dois tipos de objectos de web storage:

- *sessionStorage* - só está disponível na tab do browser ou na sessão da janela. Foi concebido para armazenar dados numa única sessão de página web;
- *localStorage* - os dados continuam disponíveis entre várias sessões do navegador web.

No entanto, todos os dados armazenados em web storage, num determinado navegador web, não estão disponíveis em qualquer outro navegador web, dado que o processo de criação e armazenamento dos objectos são diferentes entre navegadores web, tal e qual como nos *cookies*.

É possível utilizar o método *setItem()* para declarar uma chave e um valor, bem como o método *getItem()* para obter o valor dado uma chave da seguinte maneira:

```
sessionStorage.setItem('myKey', 'myValue');
var myVar = sessionStorage.getItem('myKey');

localStorage.setItem('myKey', 'myValue');
var myVar = localStorage.getItem('myKey');
```

Apesar deste método simples, apenas é possível armazenar strings o que é uma desvantagem significativa. No entanto, é possível através do método *JSON.stringify()* converter objectos em *strings*, bem como imagens através de *Base64 encoding* [50].

3.4.1.1 Fluxo de eventos e segurança

Sempre que há armazenamento de dados em *localStorage*, o evento de *storage* é disparado em todas as tabs e janelas do navegador web; Contudo este evento pode ser utilizado para superar as *race conditions* entre as janelas do browser/tabs. Se o utilizador tiver a mesma página aberta em vários locais, o evento é sincronizado, sendo que o evento é apenas disparado quando o novo valor é diferente do que está armazenado, caso contrário nada acontece.

A utilização de web storage não é mais segura que a utilização de *cookies*. A utilização de SSL ou o protocolo HTTPS resolve a maior parte dos problemas de segurança, mas em qualquer caso, não é aconselhável guardar informação sensível através deste método.

3.4.1.2 Limitações

Como já foi referenciado anteriormente, existem limitações impostas pelos navegadores web quanto ao espaço reservado para web storage, apesar de o valor médio rondar os 5MB. No caso de se obter uma excepção *QUOTA_EXCEEDED_ERR*, significa que o espaço permitido foi excedido. Através da tabela 3.5 é possível visualizar a quota de web storage permitida entre os vários browsers [51] até à data (Março de 2013).

Nenhum navegador web suporta qualquer mecanismo para pedidos de mais espaço de armazenamento no cliente, no entanto permitem que o utilizador possa controlar o espaço de armazenamento.

Navegador web	localStorage	sessionStorage
Google Chrome 27.0	2.49 M	2.49 M
IE 10 in Compatibility Mode 9.0	4.75 M	4.75 M
Firefox 21.0	4.98 M	unlimited
Safari 6.0	2.49 M	unlimited
Opera 12.1	1.85 M	4.92 M

Tabela 3.5: Quotas de web storage entre os navegadores web

3.4.1.3 Suporte e visão geral

Até à data (Março 2013) [47], todos os navegadores web modernos suportam esta funcionalidade conforme se demonstra na figura 3.5, bem como as vantagens e desvantagens na tabela 3.6.

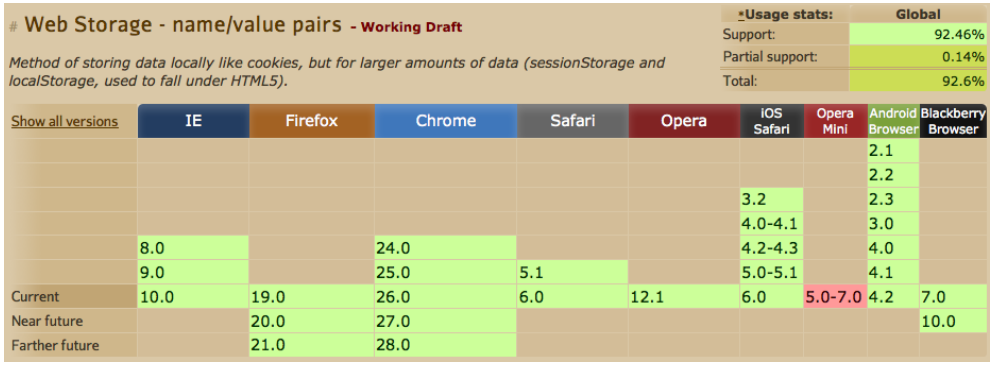


Figura 3.5: Suporte de web storage nos navegadores web

3.4.2 IndexedDB

O IndexedDB [52] é uma base de dados web HTML5 que permite que as aplicações web armazenem dados no navegador web do utilizador. Ao contrário do web storage, o IndexedDB é mais poderoso e útil para aplicações que necessitam de armazenar grandes quantidades de dados, complementando a capacidade de queries, o que permite, a este tipo de aplicações, melhor capacidade de resposta e performance em termos de interactividade.

O Web SQL Database, era uma das versões de armazenamento de base de dados para o HTML5, mas o W3C decidiu abandonar este modelo por

Vantagens	Desvantagens
Suporte nos browsers modernos; Simples API; Eventos semânticos disponíveis para manter sincronização entre outras janelas/tabs; Simples fluxo de eventos;	Desempenho fraco para dados complexos quando se utiliza a API síncrona; Desempenho fraco na pesquisa de dados complexos, devido à falta de indexação; Necessidade de assegurar a consistência e integridade dos dados, uma vez que são efectivamente não estruturados;

Tabela 3.6: Visão global da web storage

considerá-lo ultrapassado ou fora da linha [53], apostando na uniformização do IndexedDB.

A sua API é uma especificação de uma base de dados de índices que existe no navegador web. É feito de registos que contêm simples valores e objectos hierárquicos. Cada um dos registos consiste num caminho chave e num valor correspondente (que pode ser desde *strings*, datas, objectos ou *arrays JavaScript*). Pode incluir índices para uma resposta rápida de registos e armazenamento de grandes quantidades de objectos.

O IndexedDB tem dois modos na API (síncronos e assíncronos). Maioritariamente, é utilizado a API assíncrona, enquanto a síncrona é utilizada com Web Workers.

3.4.2.1 Os conceitos chave

Os conceitos chave [54] desta funcionalidade a ter em conta são os seguintes:

- IndexedDB não é o mesmo que uma base de dados relacional, é orientada a objectos. Da forma em que esta aplicação é concebida é um dos aspectos importantes;
- armazenamento de pares chave/valor - o valor é um objecto que pode conter uma ou mais propriedades. A chave pode ser baseada num gerador de chaves ou derivado do caminho que define o valor;

- a API é maioritariamente assíncrona - nunca devolve valores, necessita de uma função *callback* para obter os dados que se pretende;
- SQL não é utilizado - utiliza *queries* num índice que produz um indicador que pode ser utilizado para iterar através de um conjunto de resultados;
- impede aplicações de aceder aos dados de uma origem diferente - apenas é possível obter dados da mesma origem.

3.4.2.2 Suporte e visão geral

Até à data (Março 2013) [47], a maioria dos navegadores web modernos suportam esta funcionalidade conforme se demonstra na figura 3.6, bem como as vantagens e desvantagens na tabela 3.7.

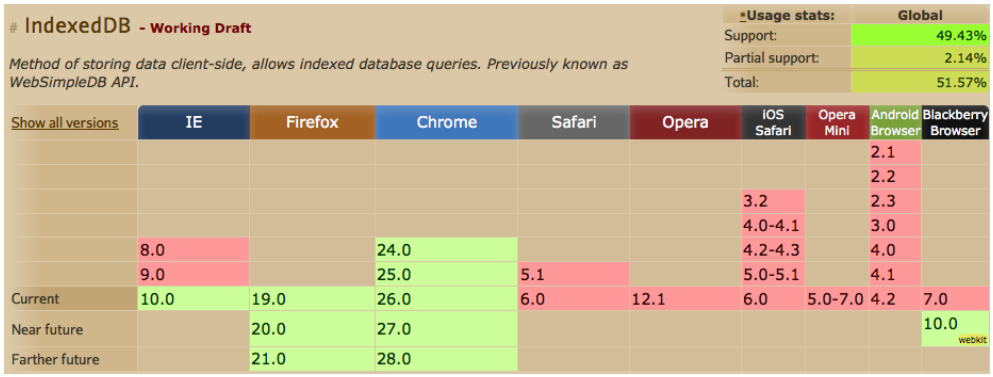


Figura 3.6: Suporte de IndexedDB nos navegadores web

3.4.3 File storage

A File storage [55] está dividida entre *Directories* e *System* e preenche uma parte que não é suportada por nenhuma das técnicas anteriormente descritas. Permite o armazenamento de conteúdo binário, criação de hierarquia de directórios, assim como armazenamento de grandes estruturas de dados.

Vantagens	Desvantagens
Queries para pesquisa através do índice; Similar ao web storage (par chave/valor); Capacidade de armazenamento de grandes quantidades de dados; Maioritariamente assíncrono; Simples API;	Suporte limitado nos web browsers; Em processo de standardização pelo W3C; Relutância em abandonar o modelo obsoleto do Web SQL Database;

Tabela 3.7: Visão global do IndexedDB

Apesar da enorme potencialidade, é uma técnica que apenas pode ser utilizada através do *prefix* do Google Chrome (webkit) [47] e encontra-se em processo de desenvolvimento e standardização pelo W3C, conforme demonstra a figura 3.7.

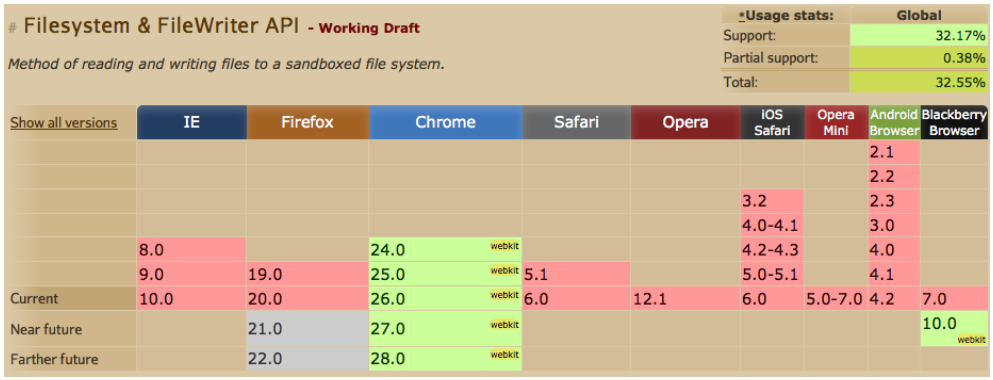


Figura 3.7: Suporte de File API nos navegadores web

3.4.4 Quota Management

A Quota Management API [56] é outra funcionalidade que se encontra em desenvolvimento pelo Google Chrome permitindo a consulta de espaço utilizado para uma aplicação web, bem como a gestão do mesmo e o pedido de mais espaço caso seja necessário [57].

Uma aplicação web pode pedir mais espaço sob três condições: temporário, persistente ou ilimitado.

3.4.4.1 Armazenamento temporário

O armazenamento temporário é o mais fácil de obter, mas normalmente nem é preciso dada a quantidade de espaço disponível por pré-definição pelos browsers. Este tipo de armazenamento serve para casos como cache.

Propriedades do armazenamento temporário:

- as aplicações têm um espaço considerável já pré-definido;
- não é garantido a existência dos dados previamente guardados. Podem ser eliminados quando o navegador web necessite de espaço livre no disco rígido.

3.4.4.2 Armazenamento persistente

O tipo de armazenamento persistente é o que fica no navegador web, a não ser que o utilizador o elimine localmente. Apenas está disponível para aplicações web que utilize o *File System API*, mas, eventualmente, estará disponível para outras APIs como IndexedDB e Web storage. Como a acção do utilizador é necessária nesta opção de armazenamento, as aplicações não têm espaçamento pré-definido.

Propriedades do armazenamento persistente:

- o navegador web informa o utilizador se mais espaço é pedido;
- as aplicações não têm espaço persistente por pré-definição;
- os dados estão garantidos, em acessos posteriores.

3.4.4.3 Armazenamento ilimitado

O armazenamento ilimitado, por enquanto, é uma opção única para o Chrome Extensions e Chrome Apps. Utilizando a propriedade de armazenamento no ficheiro *manifest*, ultrapassa todas as restrições impostas pelo armazenamento temporário e persistente.

Propriedades do armazenamento ilimitado:

- exclusivo para Chrome Extensions e Chrome Apps;
- espaço ilimitado oferecido sem qualquer indicação ao utilizador.

3.4.4.4 Suporte e visão geral

Até à data (Março 2013) [58], apenas o Google Chrome suporta esta técnica e conta com o apoio do Firefox, conforme demonstra a figura 3.8. Encontra-se em processo de uniformização para o W3C de forma que esta técnica poderá vir a ser abrangida pelos restantes navegadores web modernos da actualidade.

Figura 3.8: Suporte de Quota Management API nos navegadores web

O plano é alargar esta API a todas as técnicas de armazenamento de HTML5, de forma a ser possível obter um maior e melhor controlo e gestão do espaço necessário para cada aplicação, de forma a respeitar o utilizador com o seu consentimento.

3.5 Caracterização do espaço de execução das aplicações web

Tendo em conta a variedade dos contextos de execução das aplicações web para os navegadores web, é necessário um estudo e análise ao funcionamento dos navegadores web para determinar as suas capacidades e limitações na exibição de conteúdo dinâmico; paralelamente é estudado o funcionamento do *JavaScript* e o *threading* (web workers) para explorar as

capacidades e limitações de performance do player na execução de scripts das aplicações web nos navegadores web. O elemento *sandbox iframe* é estudado e explorarado ao nível de performance dos scripts nos conteúdos embutidos e à segurança e estabilidade do player, dada a execução sequencial de scripts nos elementos *iframes*. Por fim, são analisadas alternativas para resolução de problemas encontrados nas análises efectuadas, de forma a poder identificar e mapear o melhor contexto de execução possível.

3.5.1 Navegadores web

A principal função de um navegador web é exibir conteúdos que normalmente são pedidos por um utilizador através do interface que estes dispõem. A maneira de como o browser interpreta e exibe os ficheiros HTML e CSS está especificado e é mantido segundo a organização W3C. Apesar de na maiora dos browsers modernos a interface de utilizador ser comum, não existe nenhuma especificação formal para o mesmo, no entanto a especificação HTML5 elenca alguns elementos comuns que existam entre eles, tal como a barra de endereços, a barra de estado e uma barra de ferramentas.

Os componentes principais [59] que constituem um navegador web, conforme a figura 3.9, são os seguintes:

1. a interface de utilizador - inclui a barra de endereços, os botões Anterior/Seguinte, menus de *bookmarking*, etc. Apenas a secção da janela que exibe o conteúdo não pertence à interface de utilizador;

2. o motor web - executa as acções entre a interface do utilizador e o mecanismo de renderização;

3. o motor de renderização - responsável para exibir o conteúdo pedido.

4. *networking* - utilizado para pedidos HTTP. Possui uma interface de plataforma independente e está sob implementações para cada plataforma;

5. *backend* da interface do utilizador - plataforma utilizada para desenhar widgets básicos como combo boxes e janelas. Debaixo do mesmo utiliza os métodos da interface do utilizador do sistema operativo;

6. interpretador de *JavaScript* - utilizado para efectuar parse e executar código *JavaScript*;

7. persistência dos dados - o navegador web necessita gravar todo o tipo de dados no disco rígido, tal como cookies e outro tipo de armazenamento.

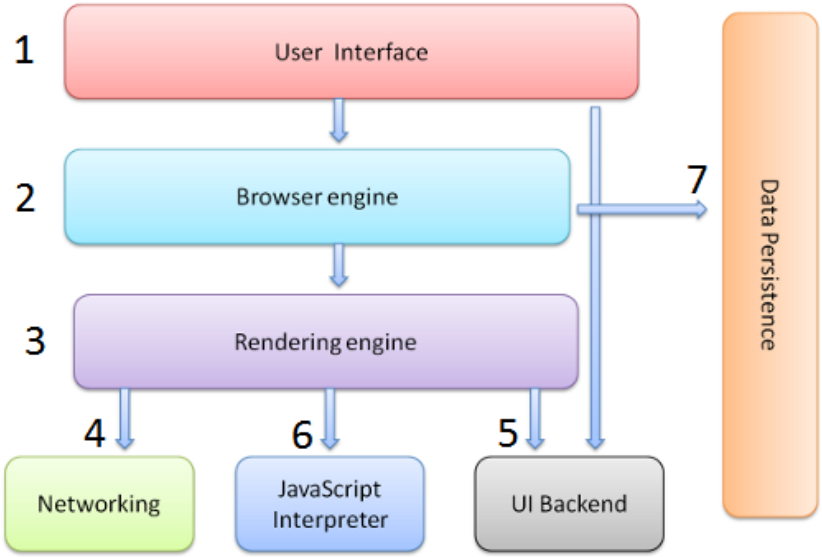


Figura 3.9: Componentes dos navegadores web

3.5.1.1 O processo de renderização

A responsabilidade do motor de renderização é exibir o conteúdo que foi pedido no ecrã do navegador web. Este processo é um dos mais importantes, dada a variedade que existe no mercado entre os vários motores web.

O motor de renderização [59], obtem conteúdo do documento pedido através da camada *Networking*.

O motor de renderização através deste fluxo, conforme demonstra a figura 3.10, efectua o seguinte:

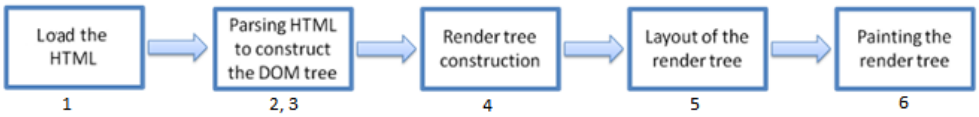


Figura 3.10: O fluxo básico do motor de renderização

1. carrega a página HTML do servidor no qual está hospedado;

2. após carregamento do ficheiro HTML, o *parser* transforma as *tags* HTML em nós do DOM e constrói uma árvore DOM (ou árvore de conteúdos). Enquanto o *parse* é efectuado, o motor web recebe outros recursos (*JavaScript*, CSS, outros), quando isto acontece, estes recursos entram em lista de espera;

3. cada navegador web contêm um CSS pré-definido. Os estilos (*styles*) precisam de ser aplicados à árvore do DOM, pelo que, uma árvore de renderização é necessária para indentificar os elementos que carecem de renderização após a árvore de estilos esteja concluída;

4. enquanto a árvore do DOM é construída, o navegador web compõe a árvore de renderização. Esta árvore é composta por elementos visuais (como cor e dimensões), que indicam a ordem pela qual devem ser exibidos;

5. quando a renderização é criada e adicionada à árvore, esta ainda não tem uma posição e dimensão definida no ecrã, sendo que através deste processo, os valores são sempre recalculados;

6. a última fase é a pintura. A árvore de renderização volta a ser percorrida e cada nó será pintado através da camada de *backend* do interface do utilizador.

De notar que todo este processo é gradual, isto é, o motor de renderização irá exibir os conteúdos no ecrã logo que disponíveis, não necessitando aguardar pela conclusão de outros elementos na árvore que estejam em processamento.

O fluxo de eventos [59, 60] é bastante similar nestes dois motores web (WebKit e Gecko), mas de uma forma mais detalhada, sendo possível encontrar os fluxos simples já definidos no processo de renderização, conforme as figuras 3.11 e 3.12.

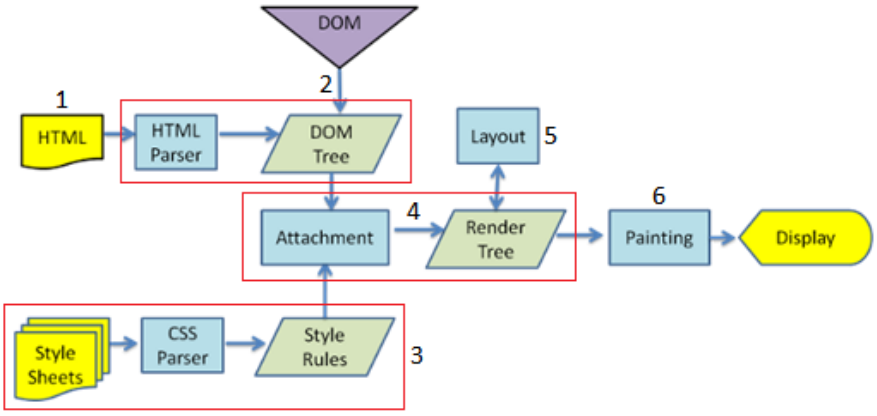


Figura 3.11: O fluxo principal do WebKit

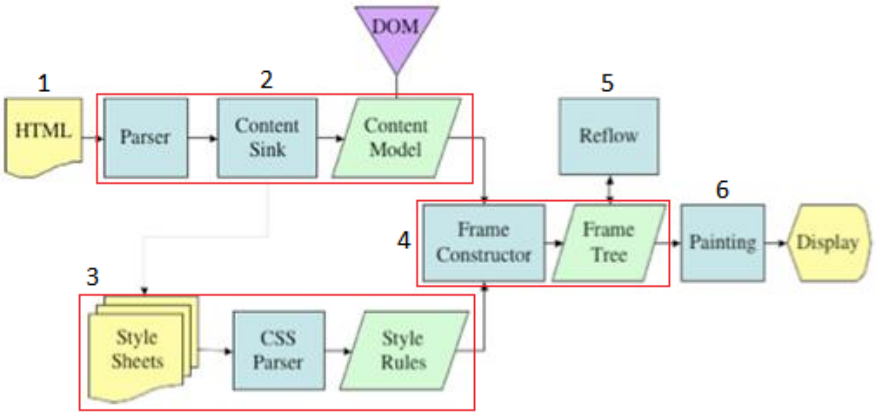


Figura 3.12: O fluxo principal do Gecko

No entanto, este fluxo de eventos não é o mesmo, quando as páginas sofrem alterações devido aos scripts de *JavaScript* ou através da interacção do utilizador, como tal o fluxo do processo de renderização é diferente [60] e toma a forma de um ciclo, conforme demonstra a figura 3.13.

- se elementos do DOM são adicionados ou removidos, o navegador

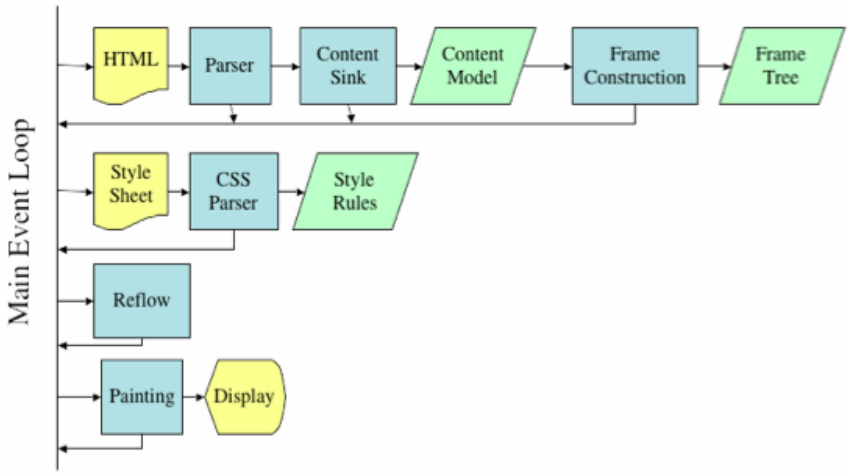


Figura 3.13: O fluxo de renderização para páginas dinâmicas

web irá efectuar o processo normal de renderização que foi descrito anteriormente;

- se um atributo de estilo ou um elemento é alterado, o estilo para o elemento precisa de ser recalculado, a página necessita de ser re-desenhada e re-pintada: o para uma melhor performance, as alterações ao estilo devem ser feitas ao lote e depois lidas como um conjunto para que a lista de espera fique com maior disponibilidade.

  – para uma melhor performance, as alterações ao estilo devem ser feitas ao lote e depois lidas como um conjunto para que a lista de espera fique com maior disponibilidade.

- algumas alterações de estilo são fáceis:

  – alterar a dimensão/posição não requer recomputação do estilo, apenas re-desenho e re-pintura;

  – alterações na cor não requer re-desenho, apenas re-pintura;

  – efectuar scrolling na página não requer re-computação, apenas re-pintura.

- re-desenho, porque a posição ou tamanho mudou é tipicamente recursivo:
  - algumas alterações nos atributos do nó filho, pode disparar alterações na árvore toda até ao *root*;
  - algumas alterações nos atributos do nó pai, pode disparar alterações em todas as folhas;
  - os navegadores web conseguem detectar que apenas uma secção da árvore pode sofrer alterações efectuando um re-desenho nessa sub-árvore.

3.5.1.2 A arquitectura

Todos os navegadores web começaram como um único processo, modelo único de *thread*. Este era um modelo aceitável, visto que as páginas web eram apenas documentos simples que necessitavam ser exibidos. No entanto, a web evoluíu e focou-se no formato de aplicações web, isto é, apesar de serem páginas web, o seu conteúdo é extremamente rico, com muito código, e com alto nível de processamento. Começaram então a surgir problemas de estabilidade, performance e segurança, e para a resolução da maioria destes problemas, foi adoptada uma arquitectura de multi-processamento. As tendências que forçaram os navegadores web a mudar de arquitectura foram as seguintes:

- performance - múltiplos processos exploram múltiplos núcleos;
- segurança - o navegador web pode lançar um novo processo num modo de privilégio inferior, para reduzir/eliminar o impacto de código malicioso;
- estabilidade uma página/script com má performance/malicioso não tem qualquer impacto por estar isolada num processo.

Internet Explorer

O primeiro navegador web com suporte a uma arquitectura de múltiplos processos foi o Internet Explorer 7, com cada janela do browser a correr o seu próprio processo.

Na versão 8 do Internet Explorer, melhoraram o modelo [61], introduzindo um processo por cada tab, contudo o processo de renderização ficava num processo único de forma a melhorar o processo de arranque.

Firefox

O Firefox utiliza um modelo de processo único, a razão para o qual ainda não adoptaram uma arquitectura de multi-processo deve-se à separação de páginas diferentes com a mesma origem.

No entanto, a existência de rumores [62] sobre uma possível alteração do motor web existente para o Servo, que está a ser desenvolvido paralelamente ao Gecko a fim de suportar uma arquitectura multi-processo.

Google Chrome

O Google Chrome segue uma arquitectura de multi-processamento [63], similar à do Internet Explorer. O processo de renderização também tem o seu próprio processo, separado de todos os processos por cada tab.

Opera

O Opera encontra-se em vias de mudar de motor web, conforme foi anunciado [64], como tal é expectável que tenha arquitectura similar ao Google Chrome.

3.5.2 JavaScript threading

Apenas existe uma thread *JavaScript* por janela. Outras actividades como renderização, download, etc, são geridas por threads diferentes, com prioridades diferentes.

O processo partilhado por *JavaScript* e as actualizações ao interface do utilizador é frequentemente referido como *browser UI thread*. O *browser UI*

thread funciona num sistema simples de filas de espera onde as tarefas são mantidas até que o processo fique inactivo. Uma vez inactivo, a próxima tarefa na fila é recuperada e executada. Estas tarefas são código *JavaScript* para executar ou actualizações à interface do utilizador, que incluem redesenho e repinturas. A parte mais interessante deste processo é que cada entrada pode resultar numa ou mais tarefas a serem adicionadas à fila.

Considerando o seguinte exemplo onde o clique de um botão resulta numa mensagem a ser exibida no ecrã:

```
1  <html>
2    <head>
3      <title>Browser UI Thread Example</title>
4    </head>
5    <body>
6      <button onclick="handleClick()">Click Me</button>
7      <script type="text/javascript">
8        function handleClick(){
9          var div = document.createElement("div");
10         div.innerHTML = "Clicked!";
11         document.body.appendChild(div);
12       }
13     </script>
14   </body>
15   </html>
```

Quando o botão é clicado, este indica ao *browser UI thread* para adicionar duas tarefas à fila de espera. A primeira tarefa é uma actualização à interface do utilizador para o botão, que necessita de alterar a sua aparência indicando que foi clicado, e a segunda tarefa é uma execução *JavaScript* que contêm o código para *handleClick()*, de modo que este é o único código a ser executado neste método. Assumindo que o *browser UI thread* se encontra inactiva, a primeira tarefa é recuperada e executada para actualizar a aparência do botão, e depois o método *JavaScript* é recuperado e executado. Durante a execução, o método *handleClick()* cria um novo elemento * * e acrescenta-o ao elemento *<body>*, efectuando uma nova mudança na

interface do utilizador. Isto significa que durante a execução do *JavaScript*, uma nova tarefa de actualização é adicionada à fila de espera, de tal forma que a interface do utilizador é actualizada logo que a execução do *JavaScript* esteja completa.

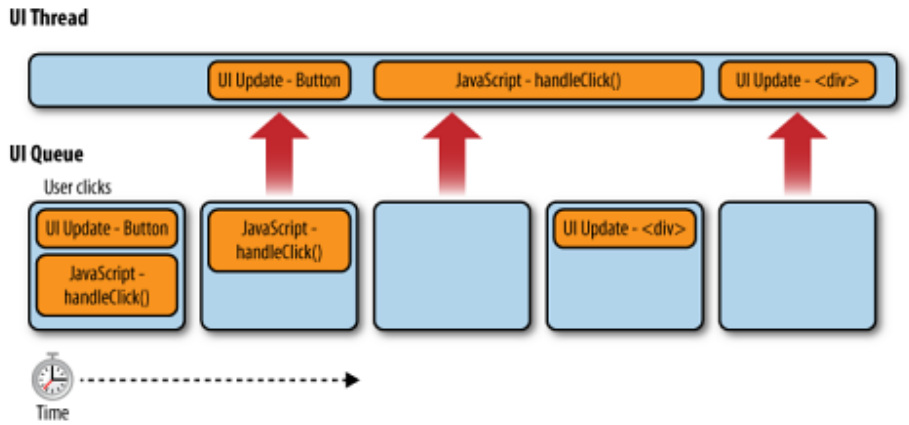


Figura 3.14: Tarefas adicionadas na thread UI quando o utilizador interage na página

Quando todas as tarefas da lista de espera tenham sido executadas, o processo torna-se inactivo e aguarda por mais tarefas a serem adicionadas à fila de espera. O estado inactivo é o ideal, pois todas as acções do utilizador resultam numa actualização imediata do interface do utilizador. Se o utilizador tentar interagir com a página enquanto uma tarefa está em execução, não só não haverá uma actualização imediata do interface do utilizador, mas uma nova tarefa para actualização da interface do utilizador pode nem ser criada e listada na fila de espera. Na verdade, a maioria dos navegadores web param de listar tarefas para a thread de interface de utilizador enquanto o *JavaScript* está a ser executado, o que significa que é imperativo concluir as tarefas de *JavaScript* o mais rápido possível de modo a não afectar a experiência de utilização ao utilizador.

Os navegadores web colocam limites de tempo sobre a quantidade do mesmo que necessitam a executar *JavaScript*. Esta é uma limitação necessária para assegurar que código malicioso não bloqueie o navegador web do utilizador ou o computador através da realização de operações intensivas

que nunca terminam. Existem dois limites:

- *call stack size* - relacionado à memória do sistema disponível;
- *long-running script* - o navegador web regista o tempo em que o script está a ser executado e quando atinge um certo limite, imposto pelo navegador web, o utilizador é questionado sobre a continuação do script.

Concluíndo, na maioria dos navegadores web, a renderização e o *JavaScript* utilizam um sistema único de eventos, isto é, quando o *JavaScript* se encontra em execução, não ocorre qualquer tipo de renderização. A única excepção a todo este sistema é o Opera, muito pelo motivo do funcionamento do motor web, pois é possível visualizar partes da renderização enquanto o código *JavaScript* é executado. No entanto, como o Opera se encontra em processo de mudança de motor web, esta excepção poderá deixar de existir [65].

3.5.3 Web workers

Um web worker, segundo o W3C [66], é um script *JavaScript* executado a partir de uma página HTML que corre em *background*, independentemente de outros scripts de interface do utilizador que também podem ser executados na mesma página HTML. Web workers têm a capacidade de utilizar multi-cores do CPU com mais eficácia.

A visão do W3C passa por ter os web workers a executar scripts de longa duração que não são interrompidos por scripts de interface do utilizador. Manter estes web workers sem interrupção por acções do utilizador, permite às páginas web para resposta contínua ao mesmo tempo em que estes executam longas tarefas em *background*.

Os web workers são executados em *threads* isoladas. Como resultado, o código do mesmo que é executado precisa de estar contido num ficheiro separado. Mas antes disso, o primeiro passo é criar um objecto *worker* na página principal. O construtor leva o nome do script do *worker*:

```
var worker = new Worker('task.js');
```

Se o ficheiro especificado existir, o browser irá gerar uma nova *thread* para o *worker*, que é descarregada de forma assíncrona. Depois de criado o *worker*, é preciso iniciá-lo invocando o método *postMessage()*:

```
worker.postMessage(); // Arranca o worker.
```

A comunicação entre o *worker* e a página principal é feita utilizando o modelo de eventos e o método *postMessage()*. Dependendo da versão do navegador web, o *postMessage()* pode aceitar *strings* ou objecto JSON como passagem de argumento.

Através do seguinte exemplo, é possível verificar a passagem de uma *string* "*Hello World*" para um *worker task.js*. O *worker* simplesmente devolve a mensagem que lhe foi passada.

Script da página principal:

```
var worker = new Worker('task.js');

worker.addEventListener('message', function(e) {
  console.log('Worker said: ', e.data);
}, false);

worker.postMessage('Hello World'); // Envia os dados para o worker
```

Script do *worker*:

```
self.addEventListener('message', function(e) {
  self.postMessage(e.data);
}, false);
```

Quando o *postMessage()* é invocado na página principal, o *worker* lida com essa mensagem através da definição do *onmessage* para o evento da mensagem. A carga da mensagem ("*Hello World*") é acessível através do *event.data*. As mensagens passadas entre a página principal e os *workers* são copiadas e não partilhadas. Existem duas maneiras de parar um *worker*:

invocando *worker.terminate()* através da página principal ou *self.close()* dentro do próprio *worker* [67].

A maioria dos navegadores web têm implementado um algoritmo de clonagem estruturado, que permite a passagem de tipos de dados mais complexos para o *worker*, tais como ficheiros, *arrays*, *blobs*. No entanto, a passagem deste tipo de dados invocando *postMessage()*, uma cópia é sempre feita, existindo uma sobrecarga notável no sentido de obter os dados entre o *worker* e a página principal. Apesar de o algoritmo de clonagem funcionar, é possível recorrer à técnica da transferência de objectos.

Com esta técnica, os dados são transferidos de um contexto para outro, tratando-se de uma cópia-zero, o que melhora bastante o desempenho de envio de dados para um *worker*. Uma vez que o conteúdo foi transferido para o *worker*, a versão que está na página principal é apagada e fica inutilizável.

Para utilizar a técnica da transferência de objectos, é necessário umas alterações ao *postMessage()*:

```
worker.postMessage(arrayBuffer, [arrayBuffer]);
window.postMessage(arrayBuffer, targetOrigin, [arrayBuffer]);
```

No caso do *worker*, o primeiro argumento são os dados e no segundo é a lista de itens que devem ser transferidos [68].

Devido ao comportamento de *multi-thread*, os web workers só têm acesso a um subconjunto de recursos do *JavaScript*:

- o objecto *navigator*;
- o objecto *location* (só de leitura);
- *XMLHTTPRequest*;
- *setTimeout()/clearTimeout()* e *setInterval()/clearInterval()*;
- cache da aplicação (web storage);
- importação de scripts externos através do método *importScripts()*;
- gerar outros web workers.

E não têm acesso ao:

- DOM;
- objecto *window*;
- objecto *document*;
- objecto *parent*.

3.5.3.1 Suporte

Até à data (Abril 2013) [47], a maioria dos navegadores web modernos suportam esta funcionalidade conforme se demonstra na figura 3.15.

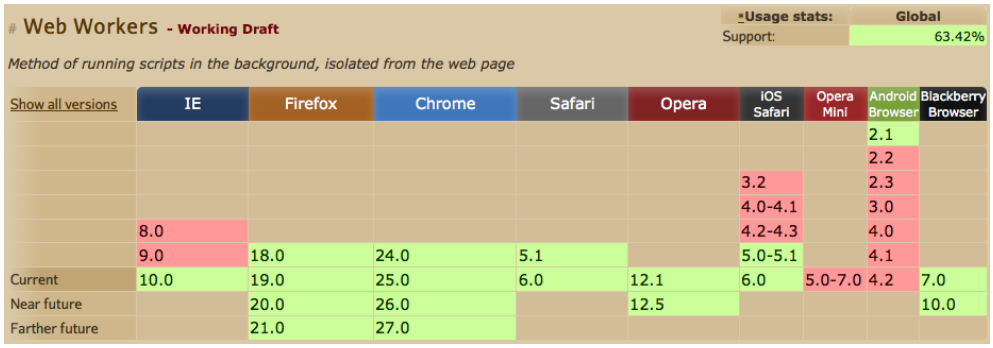


Figura 3.15: Suporte dos web workers nos navegadores web

3.5.4 iFrames

Um *iframe* é um elemento HTML que permite embutir uma página externa dentro de outra página. O *iframe* é configurado como uma estrutura da janela de um tamanho onde pode ser visualizado ao longo de toda a página.

Ao contrário das outras estruturas do HTML, que são utilizadas para dividir o ecrã em múltiplas janelas, o *iframe* é tipicamente utilizado para conseguir construir uma experiência rica na web, que inevitavelmente envolve a incorporação de componentes e conteúdo sobre o qual não existe

controlo. Aplicações de terceiros, widgets, entre outros, desempenham um papel fundamental na experiência geral do utilizador e o conteúdo gerado pelo utilizador é por vezes mais importante que o conteúdo nativo da página. Apesar do conteúdo e experiência que representam, aumentam o risco de algo malicioso possa acontecer à página principal, cada anúncio, widget que é incoporado aumenta a possibilidade de ataques com intenção maliciosa.

O CSP [69], pode mitigar os riscos associados a esses dois tipos de conteúdo, proporcionando a capacidade de confiar especificamente em certos conteúdo, apesar da limitação binária que o CSP oferece, permitindo ou não certo recurso.

O carregamento de conteúdos de terceiro num *iframe* fornece uma medida de separação entre a aplicação e o conteúdo carregado. O documento embutido não terá acesso à página principal ou aos dados armazenados localmente, ficando limitado à sua própria estrutura.

A resolução encontrada para este mecanismo foi a introdução do atributo *sandbox* [70], no elemento *iframe* de forma a controlar as permissões das funcionalidades atribuídas aos *iframes*. É possível instruir o browser a carregar o conteúdo num ambiente de baixo privilégio, permitindo apenas um subconjunto de recursos necessários para exibir a sua aplicação.

O atributo *sandbox* funciona através de uma lista de permissões [71]. É inicializado removendo todas as permissões possíveis:

```
<iframe sandbox="" src="..."> </iframe>
```

Depois, através de certos atributos é possível permitir outras funcionalidades aos conteúdos embutidos. A aplicação embutida conforme o exemplo acima mencionado encontra-se sujeito às seguintes restrições:

- scripts estão desactivados - *JavaScript* não será executado no documento embutido;
- tratamento *unique origin* - o documento embutido é carregado numa *unique origin*, o que significa que todos os *same-origin* irão falhar; *unique origins* coincidem com nenhumas outras *origins*, nem com elas próprias;

- formulários desactivados - o conteúdo embutido não tem permissões para fazer, e enviar formulários para qualquer destino;
- *plugins* desactivados - qualquer tipo de ActiveX, Flash ou Silverlight não será executado;
- o documento embutido não pode criar novas janelas ou caixas de diálogo;
- o documento embutido apenas pode navegar entre de si, e não no seu pai de nível superior;
- características que disparam automaticamente (reprodução automática de vídeos, popups, etc) estão bloqueadas;
- o atributo *seamless* é ignorado nos *iframes* que se encontrem no documento embutido.

Com a excepção de *plugins*, cada uma destas restrições pode ser levantada, adicionando um atributo ao elemento do *sandbox*:

- *allow-forms* - permite a submissão de formulários;
- *allow-popups* - permite popups; *unique origins* coincidem com nenhumas outras *origins*, nem com elas próprias;
- *allow-pointer-locks* - permite *pointer lock*;
- *allow-same-origin* - por definição, uma página *iframe* do mesmo domínio tem a possibilidade de aceder ao DOM do pai. Com o atributo *sandbox*, a página será tratada como não sendo da mesma origem; este atributo tem que ser especificado a fim de ser tratado como *same-origin*. Se pretender aceder à web storage do mesmo domínio, precisa do elemento *allow-scripts*;
- *allow-scripts* - permite execução *JavaScript* e permite disparo automático de eventos;
- *allow-top-navigation* - permite ao documento embutido a navegação à janela superior.

3.5.4.1 Suporte

Até à data (Março 2013) [47], a maioria dos navegadores web modernos suportam esta funcionalidade conforme se demonstra na figura 3.16.

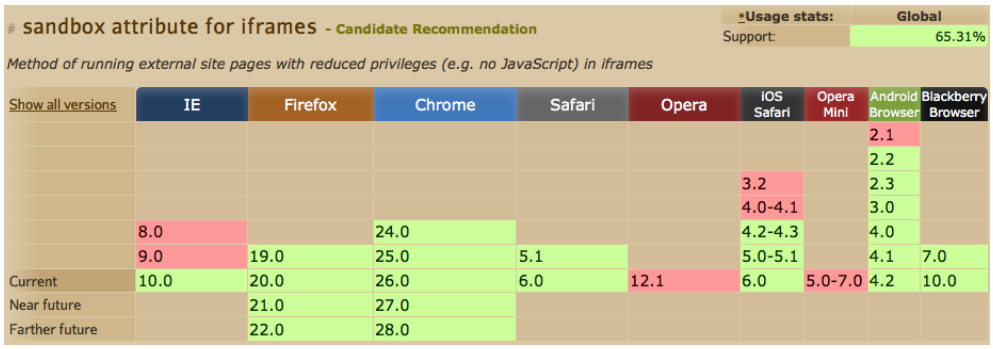


Figura 3.16: Suporte de sandbox iframes nos navegadores web

3.5.5 Análise das alternativas tecnológicas

3.5.5.1 Os problemas

Após o estudo do elemento *iframe* e das possíveis restrições que se impunham através do atributo *sandbox*, foi possível perceber, até que ponto era permitido a execução de scripts, a camada de separação entre o conteúdo da página principal e o conteúdo embutido era eficaz. Através do desenvolvimento de uma versão inicial do player, utilizando uma página principal que embutia várias aplicações web, tendo uma delas um script de longo tempo de execução, constataram-se vários problemas de resposta ao interface do utilizador. O facto de permitir scripts nas aplicações embutidas, serve para minimizar as limitações aos conteúdos de terceiros, assegurando apenas a segurança da página principal.

Este problema surge do facto das *iframes* serem executadas dentro da thread da página principal, quando um *iframe* é aberto contendo um ficheiro de *JavaScript* a ser executado, as restantes tarefas devem aguardar pela conclusão do script; até mesmo as outras *iframes* e a página principal. Isto significa que as execuções dos *iframes* são sequenciais, mas também a página

principal não pode abortar/encerrar a execução do script, uma vez que fica bloqueado pela execução do mesmo.

Uma vez que os scripts que estão a ser executados dentro do *iframe* estão fora de controlo, não se pode garantir que não irá bloquear o player.

Problemas	Descrição
<iframe>	Se o conteúdo de um iframe tiver um script que tenha uma execução longa, o interface de utilizador do navegador web poderá ficar sem capacidade de resposta até que este termine. Todos os outros conteúdos da página principal ficam também sem capacidade de resposta.
Processo Threading	Como *JavaScript* é *single thread*, é necessário uma alternativa para tentar encontrar uma resolução para este problema de forma a não limitar os conteúdos embutidos.

Tabela 3.8: Descrição dos problemas

3.5.5.2 As alternativas

Pesquisa efectuada sobre o assunto em questão, foram escolhidas três plataformas para encontrar possíveis soluções:

1. Google Chrome Extension com recurso a NPAPI plugin;
2. Google Chrome Apps;
3. Awesomium.

Google Chrome Extension

O Google Chrome Extensions [72] é uma maneira de desenvolver pequenas aplicações web que podem modificar e melhorar as funcionalidades do navegador web Google Chrome, através das tecnologias web como HTML, *JavaScript* e CSS.

As extensões agrupam um conjunto de ficheiros num só, no qual o utilizador descarrega e instala. Este conjunto significa que, ao contrário das aplicações web normais, as extensões não precisam de depender dos conteúdos web.

Para construir uma extensão Chrome basta seguir estes quatro passos:

- criar um ficheiro *manifest*;
- declarar os recursos no *manifest*;
- carregar a extensão no Google Chrome;
- desenvolver a aplicação desejada.

Através dos passos mencionados, desenvolveu-se uma aplicação para explorar o funcionamento das extensões começando com o ficheiro *manifest*:

```
{
  "manifest_version": 2,
  "name": "Hello World!",
  "description": "My first extension.",
  "version": "1.0",
  // Browser UI: browser_action, page_action
  "browser_action": {
    "default_icon": "icon.png"
  },
  "background": {
    "page": "bg.html"
  },
  "content_scripts": [ {
    "js": [ "js/jquery.min.js", "js/appv2.js", "bg.js" ],
    "matches": [ "http://*/*", "https://*/*" ]
  }],
  "permissions": [
    "tabs", "notifications"
  ],
  "web_accessible_resources": [
    "icon.png"
  ]
}
```

Relativamente à estrutura do ficheiro *manifest*, podemos declarar o seguinte:

- *manifest_version* - indica a versão do *manifest*;

- *name* - nome da extensão;
- *version* - versão da extensão;
- *description* - descrição simples da extensão;
- *browser UI* - pode ser *browser_action* ou *page_action*, serve para indicar o modo como deve ser exibido no interface de utilizador do navegador web;
- *background* - indica o nome do ficheiro que contém o script inicial da extensão;
- *content_scripts* - para declarar os scripts de *JavaScript* que estão contidos na extensão devido ao CSP;
- *permissions* - utilizado para dar permissões à extensão para aceder a determinados recursos disponíveis na API;
- *web_acessible_resources* - para declarar os recursos web disponíveis na extensão para o exterior;

Declarado o ficheiro de *manifest*, é necessário construir o script de arranque da extensão que foi declarado:

- bg.html

```
1 <!DOCTYPE html>
2   <html lang="en">
3    <head>
4    <script type="text/javascript" src="bg.js"></script>
5    </head>
6    <body>
7    </body>
8   </html>
```

- bg.js

```
1 chrome.browserAction.onClicked.addListener(function (tab) {
2     chrome.tabs.create({ url: "index.html" });
3     /*
4     chrome.windows.create({ url: "index.html", focused: true, type: "normal" }, function(tab) {
5         chrome.windows.update(tab.id, { focused: true, state: "fullscreen" })
6     });
7   */
8 });
```

O script tem um *listener* que irá abrir uma nova tab no navegador web com o *index.html*. A secção comentada do *script bg.js* contêm um exemplo de como abrir a página principal no estado *fullscreen*.

Resultados

Com a página principal em execução, dentro do script principal, em vez de carregar aplicações embutidas em *iframes*, foi testado carregar cada aplicação através de uma nova janela:

```
1 chrome.windows.create({ url: app.src, left: wleft, top: wtop, width: wwidth, height: wheight, focused: false, type: "popup" }, function(tab) {
2     self.windowid = tab.id;
3     console.log('Window ID is: ' + self.windowid);
4     // chrome.windows.update(tab.id, { focused: true, state: "maximized" })
5 });
```

Este script cria uma nova janela com o url, dimensões e tipo (normal, popup ou panel) que estejam nos argumentos e que tenham sido recebidos pelo *scheduler*. Posteriormente, após a criação da janela, é possível actualizar o estado da mesma (*maximized*, *minimized*, *fullscreen*) para o que se pretende.

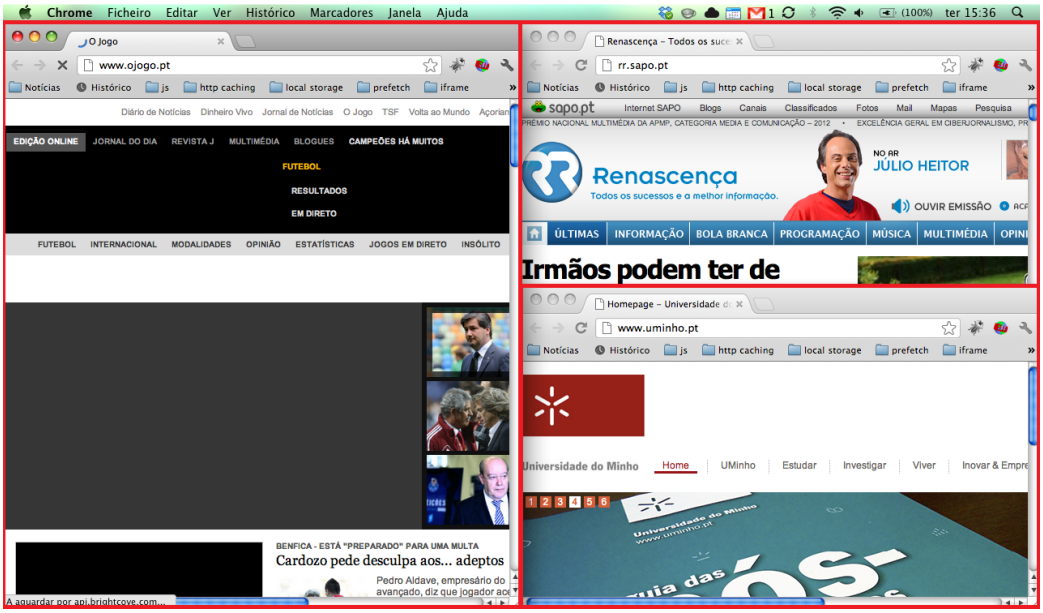


Figura 3.17: Aparência da aplicação com o valor tipo normal

Figura 3.18: Aparência da aplicação com o valor tipo popup

Figura 3.19: Aparência da aplicação com o valor tipo panel

Com o script elaborado é preciso analisar as diferenças na forma em como as janelas são criadas, alterando o valor do tipo (*type*):

- *type:* "*normal*"

  Com o tipo no valor "*normal*", é possível verificar na figura 3.17 o seguinte:

  - três janelas do navegador web estão abertas;
  - ambas foram abertas nas posições definidas no *schedule*;
  - o estado *fullscreen* apenas acontece à ultima janela activa.

- *type:* "*popup*"

  Com o valor "*popup*" no tipo, é possível verificar na figura 3.18 o seguinte:

  - três janelas de popup estão abertas;
  - ambas estão abertas nas posições definidas pelo *schedule*;

  - o estado *fullscreen* apenas acontece ao último popup activo.

  Existindo uma solução possível através do desenvolvimento de um *plugin* NPAPI de forma a remover o interface do utilizador que envolve o conteúdo exibido.

- *type: "panel"*

  Com o valor "*panel*" no tipo, é possível verificar na figura 3.19 o seguinte:

  - três janelas de panel estão abertas;
  - ambas não estão abertas nas posições definidas pelo *schedule*, devido a uma restrição imposta pela Chrome Extensions;
  - o estado *fullscreen* apenas acontece ao último panel activo.

Eventualmente poderá existir uma solução através do desenvolvimento de um *plugin* NPAPI de forma a remover o interface do utilizador que envolve o conteúdo exibido.

Observações

Registados os resultados obtidos sobre as possíveis alternativas de exibição de conteúdo através de uma extensão, é necessário compará-los.

Apesar de haver soluções mais ideais que outras, conforme se verifica na tabela 3.9, observou-se que uma possível escolha neste tipo de solução, acrescentava mais uma série de problemas:

- browser UI - seria necessário encontrar uma maneira de modificar as janelas para *chromeless*, isto é, sem qualquer tipo de interface para o utilizador;
- *fullscreen* - apesar de funcionar, limita-se a uma janela, se encontrado uma solução para a resolução do problema mencionado anteriormente, ficaria resolvido dado os conteúdos serem exibidos nas dimensões definidas;

- dimensões - este problema está relacionado com o problema do *fullscreen*, resolvido este também deixaria de ser um problema;

- processo/*threading* - não foi possível encontrar qualquer tipo de solução para este problema.

Concluíndo, esta alternativa aparenta trazer mais problemas do que soluções aos problemas que já tínhamos anteriormente.

Plugins NPAPI

Aproveitando o HTML e *JavaScript* e embutindo um plugin NPAPI [73], é possível reutilizar código proprietário de outras extensões, o que permite colocar código binário nativo do *JavaScript*.

O código que é executado num *plugin* NPAPI tem as permissões do utilizador e não é *sandboxed* ou protegido de qualquer maneira dos navegadores web. Através de uma *framework* de NPAPI, o FireBreath [74], foi pesquisado se haveria alguma maneira, de através do *plugin*, remexer em código nativo do navegador web, mas infelizmente apenas se descobriu o seguinte:

- os plugins NPAPI deconhecem, de todo, informação sobre a interface de utilizador do navegador web;

Problemas	type: "normal"	type: "popup"	type: "panel"
<iframe>	Este problem desaparece, porque é utilizado o chrome.windows para abrir janelas do navegador web	Idem	Idem
Processo Threading	O problema mantêm-se	Idem	Idem
Browser UI	Completo	Apenas na parte superior	Similar ao do tipo popup
Fullscreen	Apenas funciona para a última janela activa, sobrepondo-se às restantes	Idem	Idem
Dimensões	Funciona	Funciona	Não funciona

Tabela 3.9: Descrição dos problemas nos tipos de janelas

- existe uma possibilidade, muito inferior, através de uma API do *window* para efectuar certas modificações, mas caso o navegador web sofra actualizações e alterasse algo interno, deixaria de funcionar;

- vislumbra-se um trabalho moroso e possivelmente sem resultados.

Google Chrome Apps

O Google Chrome Apps [75] é uma alternativa ao desenvolvimento de aplicações com a mesma capacidade de aplicações web nativas e seguras como uma página web. Tal como aplicações web, uma Chrome App é desenvolvida através de HTML5, *JavaScript* e CSS, mas uma aplicação deste género além de ter o mesmo comportamento que uma aplicação nativa, possui capacidades mais poderosas que uma aplicação web normal.

Quando um utilizador executa uma Chrome App, o foco é dedicado a essa aplicação e nas tarefas que a mesma oferece; como tal, este tipo de aplicações não oferece o interface de utuilizador normal que os navegadores web modernos oferecem, tais como a barra de endereços, barra de estados, entre outros. Tal como as aplicações nativas, estas não são exibidas dentro do navegador web, mas sim com uma nova janela independente do navegador web.

Este tipo de aplicações carrega sempre localmente, permitindo que as aplicações sejam menos dependentes da ligação à rede. Uma vez instalada, o utilizador têm o controlo sobre o ciclo de vida da aplicação, podendo abrir e fechar a aplicação e até desinstalar a mesma.

Resumindo, uma Chrome App são aplicações web modificadas. É utilizado o mesmo código, as mesmas *frameworks*, e as mesmas ferramentas para o desenvolvimento da aplicação, apenas algumas funcionalidades do navegador web são removidas e outras APIs da web são desactivadas ou substituídas, para melhorar a segurança e as práticas de programação.

Novas funcionalidades são disponibilizadas de forma a ajudar na construção de aplicações similiares às aplicações nativas. Este tipo de modelo têm como objectivo proporcionar aos utilizadores uma experência mais nativa e o modelo de segurança é reforçado de forma a assegurar que as APIs não

são abusadas.

Para construir uma Chrome App basta seguir os seguintes passos:

- criar um ficheiro *manifest*;
- criar um script de *background*;
- criar uma página de janela;
- criar os ícones.

Através dos passos mencionados, desenvolve-se uma aplicação para explorar o funcionamento do Chrome Apps, começando com o ficheiro *manifest*:

```
{
 // Required
 "name": "Hello World!",
 "version": "0.1",
 "manifest_version": 2,
 // Recommended
 "description": "My first packaged app.",
 "icons": { "16": "calculator-16.png", "128": "calculator-128.png" },
 // "default_locale": "en",
 // Pick one (or none) OF browser_action, page_action, theme, app
 "app": {
    "background": {
     "scripts": [ "background.js" ]
    }
 },
 "minimum_chrome_version": "23",
 "permissions": [ "fullscreen", "webview", "experimental" ]
}
```

Relativamente à estrutura do ficheiro *manifest*, podemos declarar o seguinte:

- *manifest_version* - indica a versão do *manifest*;
- *name* - nome da extensão;

- *version* - versão da extensão;
- *description* - descrição simples da extensão;
- *icons* - descreve o tipo e localização dos ícones para a aplicação;
- tipo de aplicação - pode ser *browser_action*, *page_action*, *theme* ou *app*. Serve para indicar a modo de como deve ser exibido no interface de utilizador do browser;
- *background* - lista os recursos necessários para o arranque da aplicação;
- *permissions* - utilizado para dar permissões à extensão para aceder a determinados recursos disponíveis na API;
- *minimum_chrome_version* - para declarar a versão mínima do Google chrome para o funcionamento da aplicação.

Declarado o ficheiro de *manifest*, é necessário construir o script de arranque da extensão que foi declarado:

```
1 chrome.app.runtime.onLaunched.addListener(function() {
2    // Tell your app what to launch and how.
3    chrome.app.window.create('window.html', {
4        width: 1920,
5        height: 1080
6     });
7 });
```

Este script cria uma nova janela com o URL *window.html* e com as dimensões definidas no argumento.

Após o script, é necessário definir a página de arranque da aplicação (*window.html*):

```
1 <!DOCTYPE html>
2 <html>
3  <head>
4  </head>
5  <body>
```

```
6      <div>Hello, world!</div>
7     <webview id="wv0" style="width: 450px; height: 300px; border↩
          : 2px solid red" src="http://db.tt/FCCA7nuz"></webview>
8     <webview id="wv1" style="width: 450px; height: 300px; border↩
          : 2px solid red" src="http://www.google.com"></webview>
9     <webview id="wv2" style="width: 450px; height: 300px; border↩
          : 2px solid red" src="http://www.jn.pt"></webview>
10    <webview id="wv3" style="width: 450px; height: 300px; border↩
          : 2px solid red" src="http://www.stackoverflow.com"></↩
          webview>
11   </body>
12  </html>
```

Nada de relevante pode ser encontrado na criação da página da aplicação à excepção do elemento *webview* [76]. Esta tag é utilizada para embutir conteúdo de terceiros na aplicação. Dado o modelo de segurança apertado, o conteúdo de terceiros é contido no conteúdo do *webview*; a página principal controla como a página embutida é colocada e renderizada.

Esta *tag* apresenta diferenças relativamente ao elemento *iframe*, pois o *webview* é executado num processo separado ao da página principal; como tal não tem as mesmas permissões que a aplicação e todas as interacções entre a aplicação e o conteúdo embutido serão assíncronas. Desta maneira a aplicação fica segura do conteúdo embutido.

Resultados

Com a aplicação desenvolvida em execução, foram resolvidos alguns problemas e novos surgiram:

- o elemento *webview* comprovou um funcionamento diferente do elemento *iframe*, as páginas embutidas carregam aleatoriamente, mas caso uma delas tenha um script de longa execução, o problema aparenta persistir;
- a funcionalidade de *fullscreen* não está disponível, mas adicionando a propriedade *frame: 'none'* ao script de *background*, muda a aparência

da janela retirando o interface de utilizador.

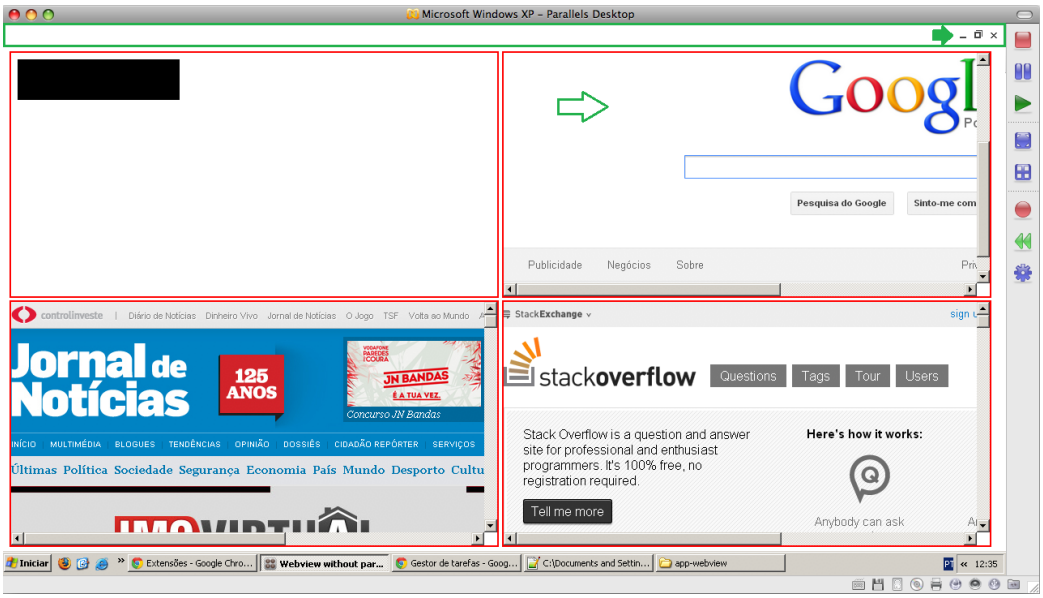


Figura 3.20: Aparência da aplicação com o valor frame chrome

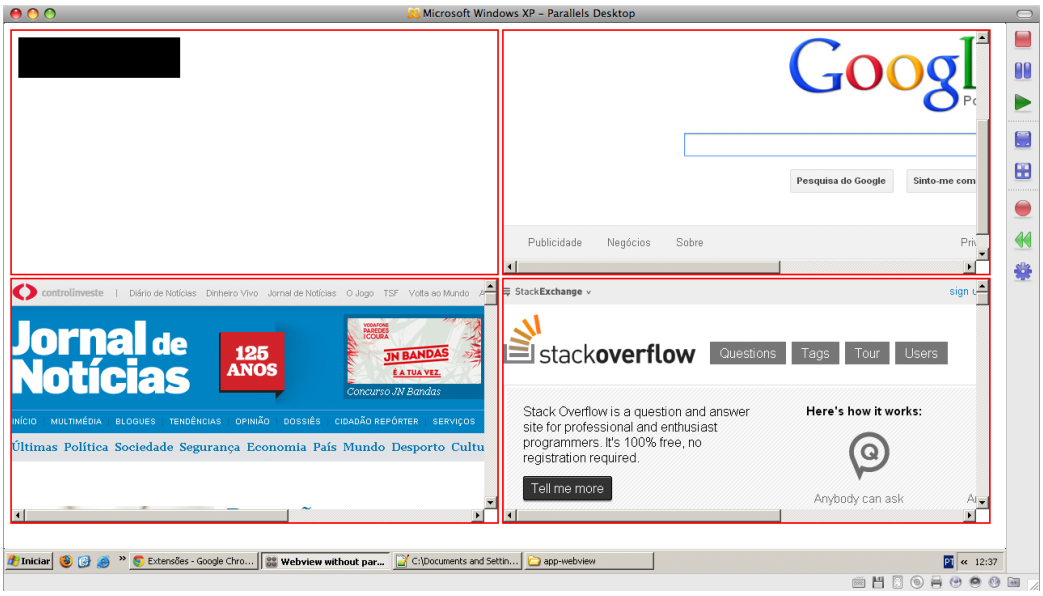


Figura 3.21: Aparência da aplicação com o valor frame none

A figura 3.20, demonstra a aparência da aplicação com a propriedade

frame: 'chrome'. Aparenta ser um *popup* com apenas interface de utilização no topo com botões no canto superior direito.

A figura 3.21, demonstra a aparência da aplicação com a propriedade *frame: 'none'*. Aparenta ser a melhor solução para simular o ambiente de *fullscreen*, apenas sendo necessário esconder a barra do menu iniciar do sistema operativo.

No entanto, apesar do funcionamento diferente do elemento *webview*, o comportamento que apresentava era aleatório, isto é, não havia um carregamento sequencial independentemente da colocação da aplicação com um script de longa execução e quando o mesmo acontecia, os outros elementos *webview* aguardavam pela conclusão do mesmo.

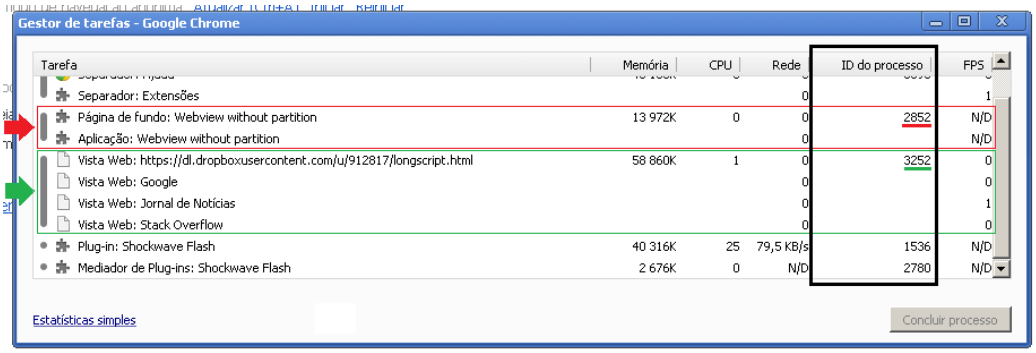


Figura 3.22: Descrição dos processos no gestor de tarefas sem o elemento partition no webview

Analisando a figura 3.22, é possível verificar no gestor de tarefas do Google Chrome (através de *Shift+ESC*) o seguinte:

- a aplicação principal é executada no processo ID 2852;
- os elementos embutidos na aplicação são executados e partilhados no mesmo processo ID 3252.

Do facto, fica provado que existe uma separação entre a aplicação e o conteúdo embutido, mas o facto de o conteúdo embutido partilhar o mesmo processo não resolve o problema de separar os conteúdos para cada processo.

Após um estudo mais cuidado da API do *webview*, existe um elemento *partition* que força a separação de processos entre os conteúdos embutidos numa aplicação, face a isto, altera-se a aplicação adicionando o elemento partition nos *webviews*:

```
1  <!DOCTYPE html>
2  <html>
3   <head>
4   </head>
5   <body>
6      <div>Hello, world!</div>
7     <webview id="wv0" partition="p0" style="width: 450px; height↩
          : 300px; border: 2px solid red" src="http://db.tt/↩
          FCCA7nuz"></webview>
8     <webview id="wv1" partition="p1" style="width: 450px; height↩
          : 300px; border: 2px solid red" src="http://www.google.↩
          com"></webview>
9     <webview id="wv2" partition="p2" style="width: 450px; height↩
          : 300px; border: 2px solid red" src="http://www.jn.pt"></↩
          webview>
10    <webview id="wv3" partition="p3" style="width: 450px; height↩
          : 300px; border: 2px solid red" src="http://www.↩
          stackoverflow.com"></webview>
11  </body>
12 </html>
```

Após execução da aplicação, observa-se o resultado esperado sendo possível visualizar os vários conteúdos embutidos a serem carregados, conforme demonstra a figura 3.23.

Após execução da aplicação, observa-se no gestor de tarefas o resultado esperado, conforme demonstra a figura 3.23.

- a aplicação principal é executada no processo ID 2388;
- os elementos embutidos na aplicação são executados nos processos ID 996, 192, 3160 e 1280.

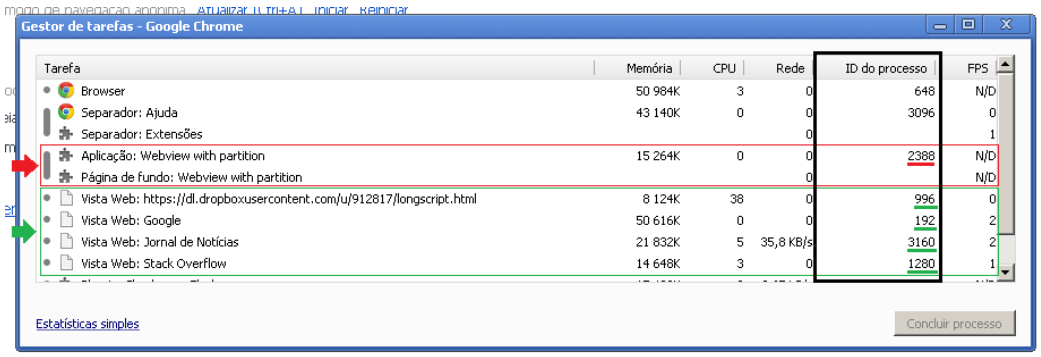


Figura 3.23: Descrição dos processos no gestor de tarefas com o elemento partition no webview

É possível verificar que todas as aplicações estão a correr separadamente, pois a aplicação que simula uma longa execução de um script de *JavaScript*, está a utilizar mais carga do CPU (38%) que as outras aplicações as quais utilizam uma carga bastante inferior enquanto realizam tráfego de rede.

Portanto, o que se obtinha sem o elemento *partition*, era uma separação da aplicação no seu processo e todos os outros *webviews* num processo diferente e partilhado; enquanto adicionando *partition*, todas as *webviews* têm o seu próprio processo.

Relativamente ao problema do *fullscreen*, adicionando a propriedade *state* (*normal*, *maximized*, *minimized* ou *fullscreen*) encontra-se em desenvolvimento pela equipa da Google Chrome. No entanto é possível utilizar esta versão, efectuando o download da versão beta do Google Chrome 29.0.

A versão do script de *background* de forma a suportar *fullscreen* é a seguinte:

```
1 chrome.app.runtime.onLaunched.addListener(function() {
2   // Tell your app what to launch and how.
3   chrome.app.window.create('window.html', {
4       width: 1280,
5       height: 800,
6       state: 'fullscreen' // Needs Google Chrome dev channel, ↩
            because Chrome Apps 'state' API is on dev stage
7    });
```

```
8 });
```

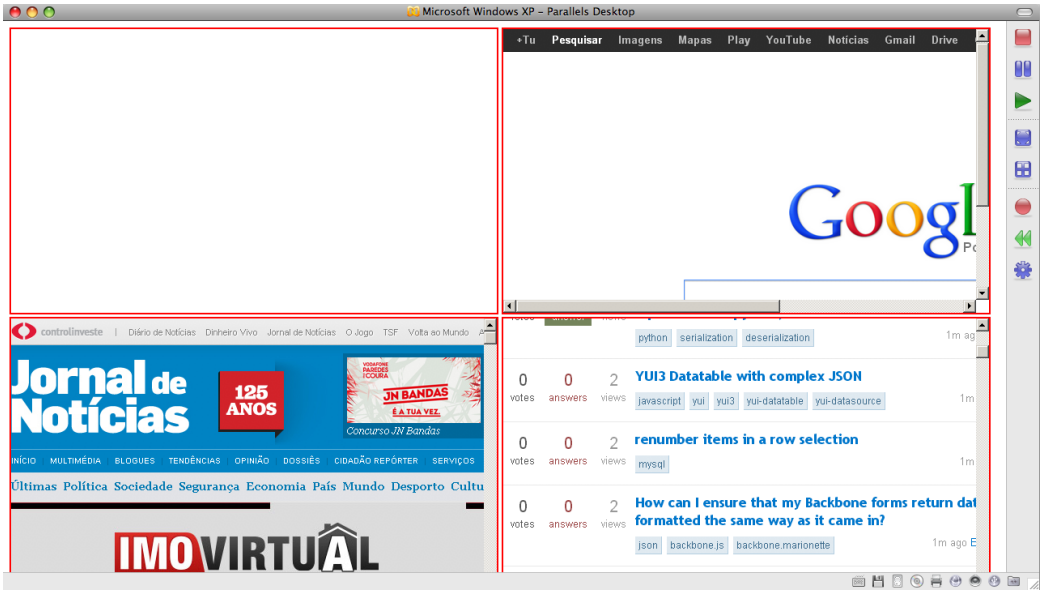


Figura 3.24: Aparência da aplicação no estado fullscreen

Concluíndo, a figura 3.24 demonstra a aplicação no estado de *fullscreen*.

Observações

Recolhidos e analisados todos os resultados obtidos, é necessário compará-los.

Problemas	frame: "chrome"	frame: "none"
<iframe>	Substituido pelo elemento webview, o problema referente ao iframe desaparece	Idem
Processo Threading	Para a aplicação ter uma separação de processo dos conteúdos embutidos, o elemento webview tem que ser utilizado, bem como partition para a separação entre conteúdos	Idem
Browser UI	Contem uma interface no topo da janela, similar a um popup	Sem interface
Fullscreen	Funciona numa versão posterior e beta do Google Chrome	Idem
Dimensões	Funciona	Funciona

Tabela 3.10: Descrição dos problemas no Google Chrome Apps

Ao comparar o resultados registados na tabela 3.10, verifica-se que os problemas iniciais, ficaram solucionados recorrendo a esta alternativa:

- *<iframe>* - este elemento passa a ser substituído pelo elemento *webview*, que funciona apenas em Chrome Apps, contendo o conteúdo numa espécie de caixa, tal como o *sandbox iframe*, mas também a capacidade de separação de processos entre conteúdos é fulcral para esta decisão;

- processo/*threading* - para ter todos os conteúdos a ser executados separadamente, o elemento *partition* do *webview* deve estar declarado, senão os conteúdos embutidos partilhariam o mesmo processo;

- *browser UI thread* - existem alternativas para escolher o tipo de interface, mas não é muito relevante dado o funcionamento do *fullscreen*;

- *fullscreen* - actualmente em fase de desenvolvimento, mas disponível caso utilize a versão 29.0 do Google Chrome;

- dimensões - funciona correctamente e até oferece mais opções de escolha entre mínimos e máximos para os conteúdos embutidos.

Com esta análise e os resultados comparados, a alternativa escolhida para o desenvolvimento do player é através da plataforma do Chrome Apps.

Awesomium

O Awesomium [77] é uma plataforma que recorre ao desenvolvimento de aplicações através do interface do utilizador da web para aplicações nativas, tais como Steam, Spotify, entre outras. Apesar de ter sido uma das alternativas em consideração e de ter potencial para tal, não foi feito qualquer teste após a escolha ter recaído na plataforma Chrome Apps.

3.5.5.3 Notas finais

Depois de uma análise aos contextos de execução e exploradas as alternativas para mapear um contexto de execução para o player, conclui-se que

a plataforma que melhores resultados apresenta é a utilização de Chrome Apps através do navegador web da Google Chrome. Além de esta plataforma executar em qualquer um dos sistemas operativos mais utilizados a nível mundial; proporciona uma vasta solução de recursos, vantajosos para o desenvolvimento das funcionalidades do player, entre eles, a capacidade de execução paralela de scripts nos conteúdos do player, a capacidade de controlar o carregamento de conteúdos no player, entre outros.

3.6 Resumo

Neste capítulo foram efectuados vários estudos técnicos, que serviram para explorar as tecnologias web; nomeadamente o suporte cache na web e vídeos; o funcionamento das aplicações web em modo offline e a capacidade de armazenamento do HTML5 para garantir o estado de execução e a estabilidade na exibição de conteúdos; e uma caracterização do espaço de execução das aplicações web analisando o funcionamento dos navegadores web e o threading do *JavaScript*; o elemento do *iframe* e uma análise às alternativas tecnológicas que solucionassem os problemas encontrados na caracterização do espaço, mapeando e identificando contextos adequados à execução das funcionalidades necessárias no player.

4

Identificação de requisitos e opções tecnológicas

Este capítulo caracteriza, detalhadamente os requisitos concretos que deverão ser suportados por um player para redes abertas de ecrãs públicos. É feita uma análise ao ambiente de execução, para o player e as respectivas implicações no que concerne às possíveis tecnologias a ser utilizadas, dado os princípios web. É feita uma especificação, de uma forma sucinta, de todas as camadas lógicas que compõem a aplicação, bem como, uma análise de alto nível da camada mais alta do sistema e descrita nos domínios funcionais; e por fim uma sugestão às possíveis hipóteses de escalonamento à implementação do modelo do *scheduler*.

4.1 Ambiente de execução

O ambiente de execução ideal para o player, é um dos motores web mais utilizados a nível mundial (WebKit, Gecko ou Trident). Não é obrigatório a utilização de um navegador web pois este é dividido entre um motor web e a interface de utilizador. Caso seja desenvolvido outro tipo de interface de utilizador e embutir o motor web, o player funcionará como se fosse uma espécie de navegador web.

Os motores web mencionados anteriormente são compatíveis com a linguagem *JavaScript*, e todos eles ainda apresentam limitações em certos elementos do HTML5. Contudo, continuam a ser feitos esforços para a uniformização do mesmo. Deve evitar-se a utilização de plugins por motivos de uniformização, podendo representar um problema para outros motores web. É preferível então, desenvolver uma aplicação estável e uniformizada, o mais abrangente possível ao invés da utilização de plugins no desenvolvimento da aplicação.

Relativamente ao sistema operativo, o motor web Trident da Microsoft é limitado dada a sua incompatibilidade com a maioria dos sistemas operativos (Unix, Mac OS X). Dadas estas suposições e de forma a abranger o maior leque de utilizadores, o desenvolvimento da aplicação deve focar-se nos ambientes web engine WebKit e Gecko.

Deve ter-se em consideração, o desenvolvimento de um script de arranque para o player, predefinindo o URL do player, bem como o arranque automático do navegador web, quando o sistema operativo está carregado.

Relativamente às plataformas computacionais é preciso ter em consideração a capacidade e limitações das mesmas para a execução do sistema operativo e respectivo navegador web, bem como as dimensões e custos monetários e energéticos.

4.2 Princípios da web

O player deve ser desenvolvido em *Javascript*, dada a sua ubiquidade e uniformidade com a maioria dos navegadores web, servindo de plataforma web para a exibição de todo o tipo de conteúdo multimédia; desde imagens, vídeos e aplicações web e controlado através de um *schedule* que é descarregado no arranque do player, que faz iteracção entre as várias aplicações introduzidas, bem como as suas dimensões e posicionamento no ecrã.

Dado o seu desenvolvimento em *JavaScript*, elimina as restrições em todos os sistemas operativos, bem como nos navegadores web.

4.3 Funcionalidades dos players

Com base na análise dos players descritos na secção de estado da arte, foi possível identificar uma lista de requisitos, que no seu conjunto definem o espaço de design para este tipo de produto conforme demonstra a tabela **??**.

4.4 O modelo de aplicações web

O modelo de aplicações web para um player traduz-se num mecanismo que permita a terceiros colocar o seu conteúdo numa rede de ecrãs públicos através de múltiplos domínios. Por meio deste tipo de aplicações, é possível criar conteúdo rico e adaptável para ecrãs públicos, utilizando ferramentas web convencionais e tecnologias como HTML, CSS e *JavaScript*.

No entanto, este tipo de aplicações estão optimizados para as especificidades dos ecrãs públicos, isto é, semelhante ao modelo de aplicações web móveis, onde, por exemplo, é possível exibir conteúdos optimizados para esse tipo de dispositivos. A utilização de aplicações web não exclui a existência de aplicações nativas para determinados ecrãs públicos. Mesmo assim, as aplicações web podem servir vastos cenários de aplicações, com a principal vantagem de ser mais fácil o suporte em vários tipos de ecrãs. Uma aplicação web deve ser capaz de apresentar conteúdo nos ecrãs, mas também ter a capacidade de descrever o seu conteúdo e características de forma que permita, aos ecrãs, optimizar a integração do respectivo conteúdo num âmbito mais amplo através de um *schedule*. É esperado que as aplicações sejam serviços disponibilizados através da rede, possam gerar conteúdo para múltiplas configurações de ecrãs. Por suposição, a execução de aplicações web terá a capacidade de executar o seu conteúdo através de um browser.

O modelo de aplicações desenhado para esta conjuntura, encontra-se na distribuição de responsabilidades entre a parte da aplicação que é exibida num ecrã e a outra parte alojada no servidor. Enquanto abordagens muito distintas podem ser seguidas. Dependendo da natureza da aplicação, assume-se um modelo de aplicações SPA, para o componente da aplicação que é executado no ecrã público.

Designação	Categoria	Descrição
Carregamento prévio de conteúdos	Gestão de conteúdos	Efectua o download prévio de todos os conteúdos referidos antes de ser posto em funcionamento. Este processo é normalmente suportado fora do scheduler. Verifica a lista de conteúdos, que ainda não estão na máquina e carrega-os antes de informar o scheduler da existência do um novo *schedule* para execução. Links para conteúdos remotos podem passar a ser locais antes de serem entregues ao scheduler
Obtenção prévia de conteúdos	Gestão de conteúdos	Este processo de activação de conteúdos permite ao scheduler preparar os conteúdos que se antevê vir a apresentar, de modo a garantir uma visualização flúida e sem tempos de espera por carregamento.
Armanezar conteúdo na cache	Gestão de conteúdos	Cache de conteúdo que não poder ser carregado previamente, como por exemplo um feed RSS...
Capacidade de ligar/desligar	Ambiente	É feito por uma via alternativa que é, durante a instalação configurar eventos do scheduler do sistema operativo de modo a desligar e ligar às horas pretendidas.
Ajustes de resolução	Ambiente	Como o player recebe programações com tamanho fixo, isto permite as resoluções a usar nessa programação às do ecrãs em funcionamento.
Informação do sistema	Ambiente	Comunicar ao serviço informação técnica sobre o SO, hardware, etc...
Comandos remotos	Ambiente	Desligar, reiniciar, etc.
Sensores	Ambiente	Suporte para ligação de sensores associados ao ecrã como Kinect...
Controlo do ecrã	Ambiente	Para desligar o monitor, pode ser um comando; para poupar; para esconder situações de recarregamento de conteúdos, etc.
Recuperação de crash	Tolerância a faltas	Consiste em detectar situações de bloqueio do software do player e proceder à sua reinicialização, evitando assim que esses bloqueios se mantenham; normalmente suportada na forma de um serviço de *watchdog* externo à aplicação principal do player
Prevenção de crash	Tolerância a faltas	Consiste em prevenir crashs vigiando o consumo de recursos e outras informações relevantes sobre o funcionamento da máquina; normalmente suportada na forma de um serviço de *watchdog* externo à aplicação principal do player, de modo a não ser afetada pelas falhas do próprio player. É especialmente importante em máquinas que possam gerar problemas como fugas de memória ou problemas de drivers.
Execução de registos	Registo	Provas de execução, manutenção e diagnósticos.
Screenshots	Registo	Gerar e enviar screenshots do que está a ser apresentado.
Controlo de execução	Execução	Permite parar o scheduler ou re-iniciar quando, por exemplo, é necessário trocar a programação
Scheduler	Execução	Suporta a execução de schedules
Integridade	Segurança	Garantir integridade do player. Encriptação e checksums.
Actualizações automáticas	Actualizações	Actualizações automáticas e não assistidas de software. Este processo pode ser suportado directamente pelos mecanismos próprios do sistema operativo, desde que o software esteja preparado para o efeito.

Tabela 4.1: Funcionalidades dos players

4.5 Camadas do sistema

As camadas do sistema estão baseadas em quatro camadas lógicas: hardware, sistemas operativos, navegadores web e aplicação. A camada de hardware contém as plataformas computacionais. A camada de sistemas operativos contém os sistemas operativos mais utilizados a nível mundial e os restantes ficam englobados no elemento. A camada de navegadores web contém os navegadores web mais utilizados a nível mundial, englobando os restantes no elemento *'Outros'*. Por fim, a camada da aplicação, demostra a tecnologia web necessária para o funcionamento da aplicação.

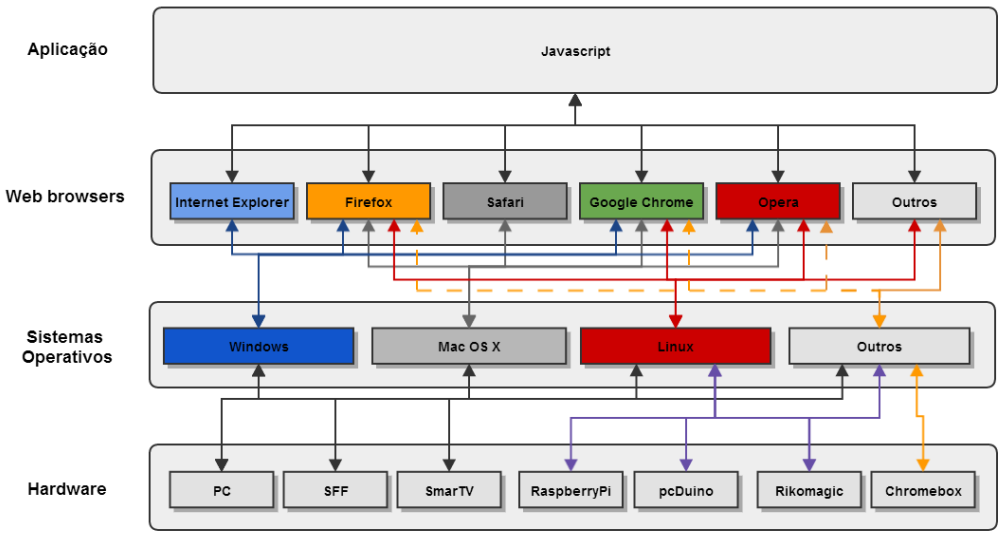


Figura 4.1: As camadas do sistema

De forma a verificar as dependências entre os vários níveis lógicos do sistema da figura 4.1, constata-se o seguinte:

- existe uma dependência entre as várias plataformas computacionais, isto é, a escolha de plataforma computacional limita e confina a escolha do sistema operativo;
- independentemente do sistema operativo, os navegadores web que oferecem mais compatibilidade são Firefox, Google Chrome e Opera;

- qualquer que seja o navegador web, todos eles executam a tecnologia web *JavaScript*.

4.6 Stacks de referência

Apesar de as tecnologias web garantirem teoricamente a portabilidade do desenvolvimento do player, existem diferenças consideráveis nas várias camadas lógicas que compõem o ambiente de execução. Mesmo não sendo perceptíveis, essas diferenças, foi necessário explorar as camadas lógicas que separam o ambiente de execução e simplificar a identificação de stacks de referência que resultam das várias combinações possíveis que a compõem.

Assim sendo, contempla-se a separação de várias plataformas computacionais estudadas e definem-se vários conjuntos específicos de execução, compatíveis para o player através do sistema operativo e navegador web conforme se demonstra nas figuras 4.2 e 4.3.

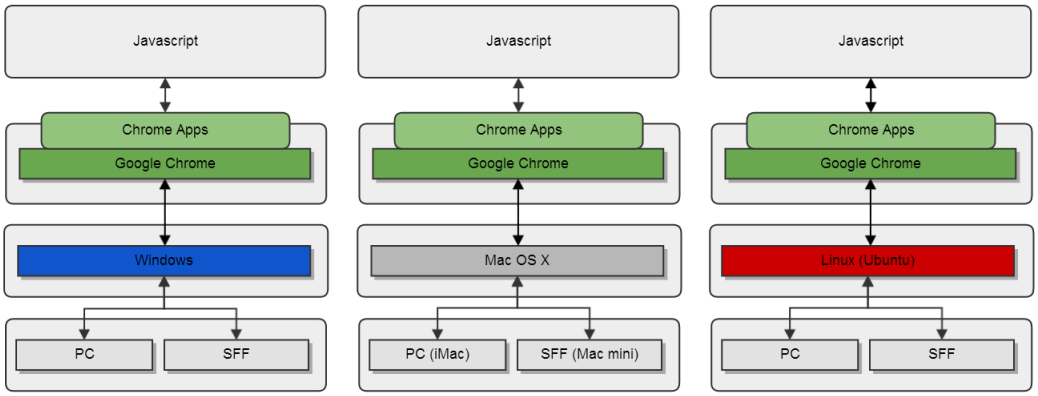


Figura 4.2: Ambientes de execução para as plataformas PC e SFF

A definição das stacks de referência serve para reduzir o elevado número de stacks de todas as combinações possíveis resultantes da estrutura das camadas de sistema (hardware, sistema operativo, navegador web e extras), contemplando desta forma uma diversidade no número de plataformas computacionais, assim como definir um conjunto específico de objectivos no desenvolvimento limitando o número de potenciais diferenças relativamente ao sistema operativo e navegador web.

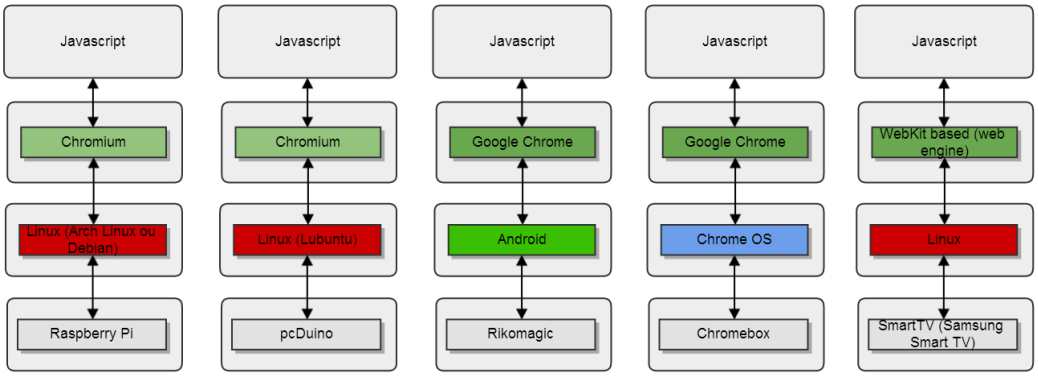


Figura 4.3: Ambientes de execução para as plataformas Raspberry Pi, pcDuino, Rikomagic, Chromebox e Samsung Smart TV

4.7 Domínios funcionais

O sistema deve permitir a interacção de vários domínios com diferentes tipos de tarefas. Os domínios mencionados são importantes no sistema, cada um à sua maneira. Não existindo coesão entre os mesmos, o sistema, eventualmente, não funcionará correctamente.

Foram definidos três domínios-chave para o funcionamento da aplicação, que apresentam uma série de problemas para os quais foi necessário definir uma solução. Assim sendo, a implementação do player vai estar maioritariamente dependente dos seguintes domínios:

- *'Scheduler'* - domínio no qual é definido o processo de estruturação e o conjunto de conteúdos e regras;

- *'Player'* - domínio responsável pela interpretação de um conjunto de conteúdos e regras; e pelo funcionamento do player através de um conjunto de funcionalidades necessárias à sua execução;

- *'Comunicação'* - domínio responsável pela comunicação entre entidades externas e o player.

4.7.1 Scheduler

O *'Scheduler'* é extremamente importante na forma em que é construído, pois a sua estrutura servirá de base para o domínio do *'Player'*. É necessário ter em conta uma especificação simples da estrutura, de formato aberto, com a possibilidade de extensão para mais regras; havendo necessidade de novas regras, determinar-se-á uma granularidade para uma estrutura recursiva para os conteúdos em vários níveis de profundidade. Concluíndo, os principais problemas deste domínio são os seguintes:

- estrutura/formato do *scheduler*;
- regras para os conteúdos e *scheduler*;
- decisão de uma linguagem aberta e de fácil compreensão;
- nível de profundidade.

4.7.2 Player

O controlador é o domínio principal do *'Player'* e apresenta vários problemas. Primeiro, é necessário processar o *scheduler* de forma a obter um conjunto de conteúdos e regras; posteriormente, criação e gestão de uma espécie de *playlist* dos conteúdos e regras correspondentes. Através desse conjunto de regras, cada conteúdo terá o seu próprio ciclo de vida, de forma a poder iterar o conteúdo contido no scheduler. Além dos principais problemas do controlador do player, abordou-se simultaneamente problemas de monitorização, autenticação e estado de operação, como possíveis funcionalidades para o player. Concluíndo, os problemas neste domínio destacam-se pelo funcionamento principal do player nos quais se destacam:

- processamento do *schedule*;
- criação de uma *playlist*;
- exibição de conteúdo;

- conteúdo independente;
- gestão do ciclo de vida de o conteúdo.

4.7.3 Comunicação

A *'Comunicação'* é um factor em consideração, muito pela interacção entre as várias entidades do sistema, como por exemplo, o *scheduler*. É necessário determinar esse tipo de ligação, pois poderá estar ligado a certas funcionalidades do domínio do *'Player'*, bem como a possibilidade de interacção com os conteúdos que serão exibidos. Concluíndo, os problemas neste domínio resumem-se da seguinte maneira:

- entidades internas do *'Player'*;
- *scheduler*;
- conteúdos.

4.8 Hipóteses de escalonamento

O *schedule* deve ser concebido num formato aberto como JSON ou XML, através das especificações já conhecidas da maior parte dos players para a representação de conteúdo multimedia, o SMIL.

Deve ser composto através de uma raíz (*schedule*), que deve conter todo o tipo de propriedades de alto nível relativamente aos conteúdos que vão ser apresentados; desde a sua identificação, nome, última actualização, e o posicionamento geral em relação ao ecrã.

Deve conter um filho e/ou lista de filhos sequencial que deve indicar o número de iteracções a ser apresentado no ecrã, bem como o posicionamento relativamente à raiz.

Por fim, uma lista de filhos sequencial onde se registará o tipo de aplicação URL e a duração, que a aplicação estará a exibida no ecrã.

5
Protótipo

Neste capítulo especifica-se, de uma forma sucinta, a arquitectura geral do sistema e a estrutura e análise aos módulos da aplicação, bem como a modelação da aplicação e a definição e caracterização da estrutura do escalonamento. É apresentado o protótipo que implementa os vários módulos do modelo proposto e respectivas interacções, bem como a análise e descrição detalhada de cada módulo, demonstrando o funcionamento e as dependências existentes entre si. Por fim, são abordadas as diferenças de implementação do protótipo do player entre navegadores web e a plataforma Chrome Apps e o estado de execução e operação do protótipo.

5.1 Arquitectura geral

Propõe-se uma arquitectura desenvolvida em camadas de modo a conseguir extensabilidade do sistema, conforme demonstra a figura 5.1, que permita:

- adição de novas funcionalidades sem impacto nas existentes;
- modularidade de forma a dividir o sistema em módulos;
- redução no esforço na manutenção da aplicação.

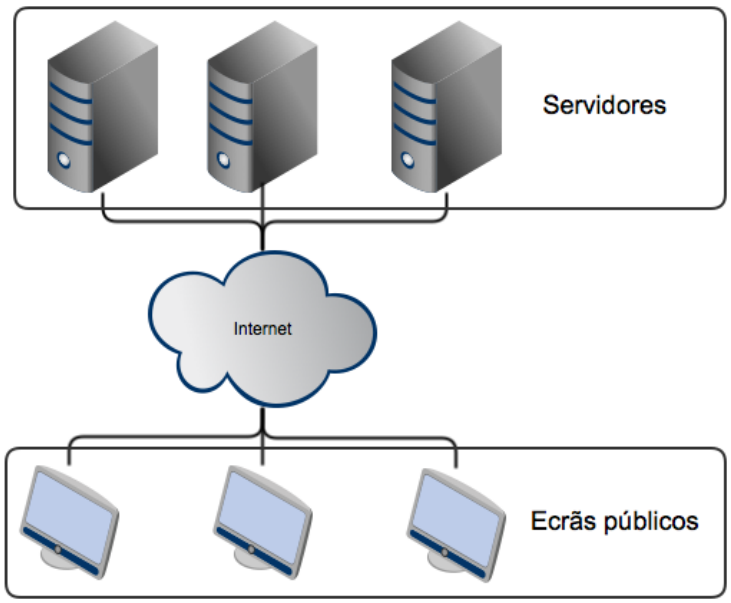


Figura 5.1: A arquitectura geral do sistema

O sistema é baseado numa arquitectura de duas camadas: a camada de ecrãs públicos e a camada de servidores. A camada de ecrãs públicos contém os players a serem executados nos respectivos ecrãs com as devidas funcionalidades e tarefas. Na camada de servidores, é implementada a lógica de controlo dos ecrãs públicos, isto é, a gestão dos *schedules*, players e também possíveis web services e bases de dados.

5.2 Estrutura e módulos aplicacionais do player web

Os módulos do sistema aplicacional permitem a apresentação de uma estrutura que envolva os mesmos de forma a separar as funcionalidades entre eles, de modo a conseguir extensabilidade a nível da aplicação de forma a permitir:

- separação lógica e independente de funcionalidades entre os vários módulos;

- adicionar novas funcionalidades aos módulos pretendidos, não afectando os módulos envolvidos;
- reutilização de módulos para outro tipo de aplicações.

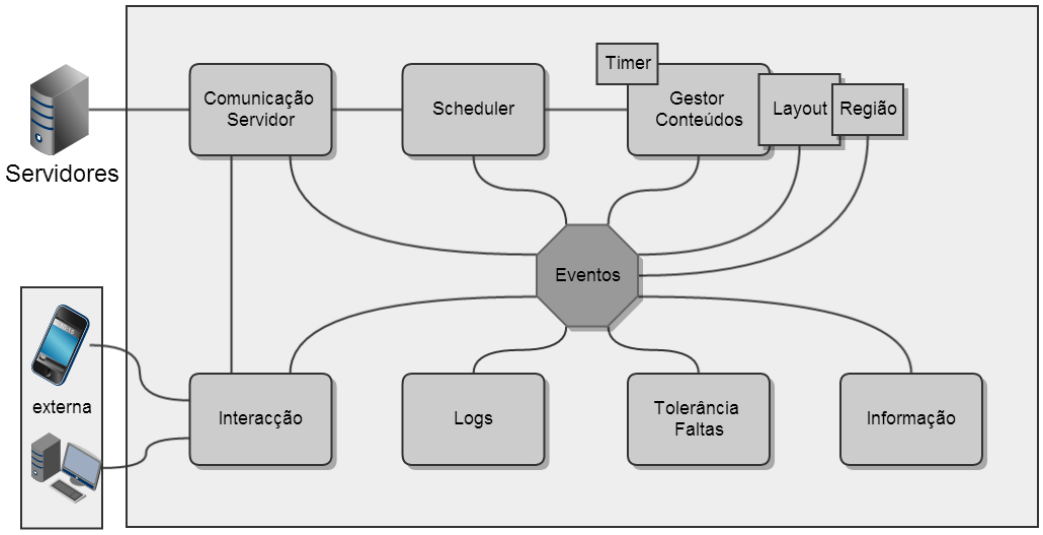


Figura 5.2: A estrutura e módulos da aplicação

A estrutura é baseada em dez módulos, conforme demonstra a figura 5.2, nomeadamente:

- *'Informação'* - este módulo é responsável pela gestão da informação no ambiente onde o player é executado, desde a sua identificação, sistema operativo, navegador web, entre outros. Os dados mantidos neste módulo são sempre verificados pelo módulo de comunicação ao servidor;
- *'Comunicação Servidor'* (Server Communication) - este módulo é responsável pela autenticação e pedido do *schedule* ao servidor. Independentemente do resultado obtido através da autenticação e respectivo pedido, é responsável pelo envio do resultado ao módulo do *'Scheduler'*;
- *'Scheduler'* - este módulo é responsável pelo tratamento e validação do *schedule*. Há vários processos a considerar neste módulo: a verificação

do *schedule* alojado localmente e a validade do *schedule* recebido no módulo de comunicação ao servidor. Os resultados possíveis a ser obtidos conforme as tarefas mencionadas anteriormente são: o *schedule* final é válido ou inválido/nulo. Independentemente do resultado, este é transmitido ao módulo de *'Gestor Conteúdos'*;

- *'Gestor Conteúdos'* (Content Management) - este módulo é responsável pela gestão de conteúdos a ser exibidos no ecrã. O primeiro passo é verificar a validade do *schedule*, caso seja inválido exibe uma mensagem de erro no ecrã; caso contrário verifica o tipo de *schedule* e inicia o processo de construção do player através de uma lista de conteúdos e lista de layouts conforme especificado no mesmo. Após o processo de criação do player, fica responsável pela iteracção dos layouts e verificação dos ciclos dos conteúdos no ecrã;

- *'Layout'* - este módulo é responsável pela criação de layouts a pedido do gestor de conteúdos. É responsável pela criação de regiões e respectivas listas dos mesmos de forma a controlar as que lhe pertencem;

- *'Região'* (Region) - este módulo é reponsável pela gestão dos conteúdos que iteram nas respectivas regiões de acordo com as regras especificadas no *scheduler*. Apenas fica responsável em adicionar/remover conteúdos das regiões e controlar os seus ciclos de iteracções;

- *'Logs'* - este módulo é responsável pelo registo de todas as acções que são executadas na aplicação;

- *'Timer'* - este módulo é responsável pela gestão do tempo de ciclos entre conteúdos que são controlados no módulo de gestão de conteúdos;

- *'Interacção'* (Interaction) - este módulo é responsável por todos os métodos de interacção externa ao player.

5.2.1 Interacção entre os módulos

Nesta secção é apresentada a modelação da aplicação de forma a perceber as funcionalidades de cada módulo e respectivas tarefas, bem como a comunicação interna e externa. Os diagramas de sequência apresentados, ilustram a forma como algumas tarefas são realizadas entre os vários módulos da aplicação. Estes módulos estão representados da seguinte maneira:

- CS: este módulo refere-se à *'Comunicação Servidor'*, sendo por isso responsável por efectuar pedidos aos servidores;
- I: este módulo refere-se à *'Informação'* do player, responsável por gerar e manter a informação local do player;
- S: este módulo refere-se ao *'Schedule'*, responsável por verificar a validade do *schedule* e conteúdo local;
- GC: este módulo refere-se ao *'Gestor Conteúdos'*, responsável pela gestão de layouts, regiões e conteúdos que estejam a ser exibidos no ecrã;
- L: este módulo refere-se ao *'Layout'*, responsável pela gestão das regiões a que lhe pertencem;
- R: este módulo refere-se à *'Região'*, responsável pela gestão de conteúdos que são exibidos no ecrã aos quais lhe pertencem.

A figura 5.3, representa o diagrama de sequência que demonstra o arranque do player no ecrã público, começando no módulo de *CS*, efectuando uma verificação de informação do player para obtenção de dados do mesmo. De seguida, caso exista uma ligação activa, tenta autenticar-se ao servidor remoto, caso a autenticação seja validada, efectua o pedido do *schedule*, caso contrário indicará ao módulo do *S* o conteúdo que tem para o mesmo.

A figura 5.4, representa o diagrama de sequência que demonstra o funcionamento geral do player no ecrã público, iniciando uma verificação do *schedule* recebido do módulo anterior e uma comparação com o possível

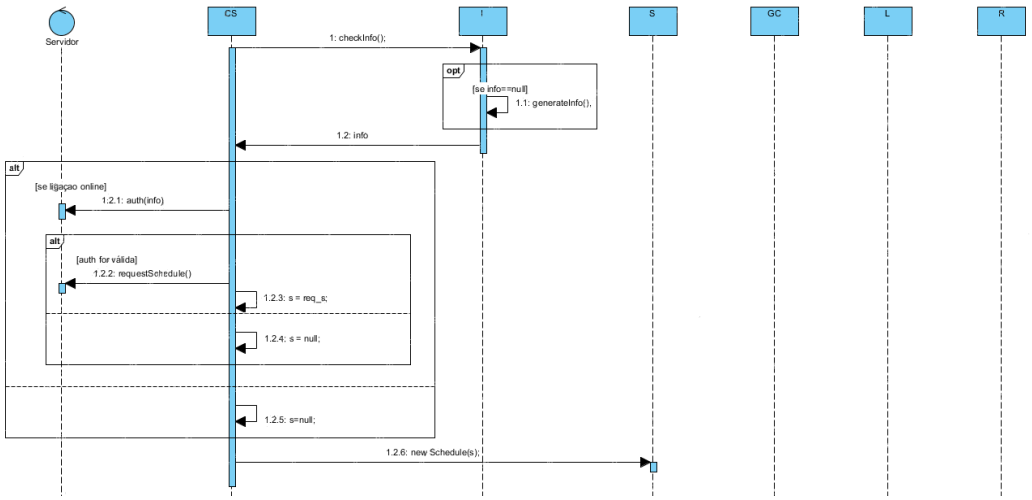


Figura 5.3: Diagrama de sequência de comunicação da aplicação com servidor

schedule local para determinar qual deles deve ser utilizado. Após o processo de comparação e validação, e no caso de o *schedule* ser inválido, irá exibir uma mensagem de erro no ecrã, caso contrário inicia o processo de *GC* do *schedule* alocando as aplicações, layouts e regiões. Finalmente, começa a exibir os conteúdos obedecendo às regras declaradas no *schedule*.

5.3 Escalonamento

O formato *scheduler* é maioritariamente baseado nas especificações do W3C SMIL 3.0 [78] e A-SMIL. É uma abordagem muito simplificada com vista a ajudar os requisitos de uma rede de ecrãs abertos através da descrição de um conjunto de conteúdos e as circunstâncias em que devem ser exibidos. Este formato, evolui gradualmente a partir da sua forma mais simples até um cenário mais complexo recaindo a escolha de linguagem para o JSON.

5.3.1 Versão 1

O formato *schedule* da versão 1 do anexo A tem como objectivo o suporte de uma única região de escalonamento de uma sequência com uma ou

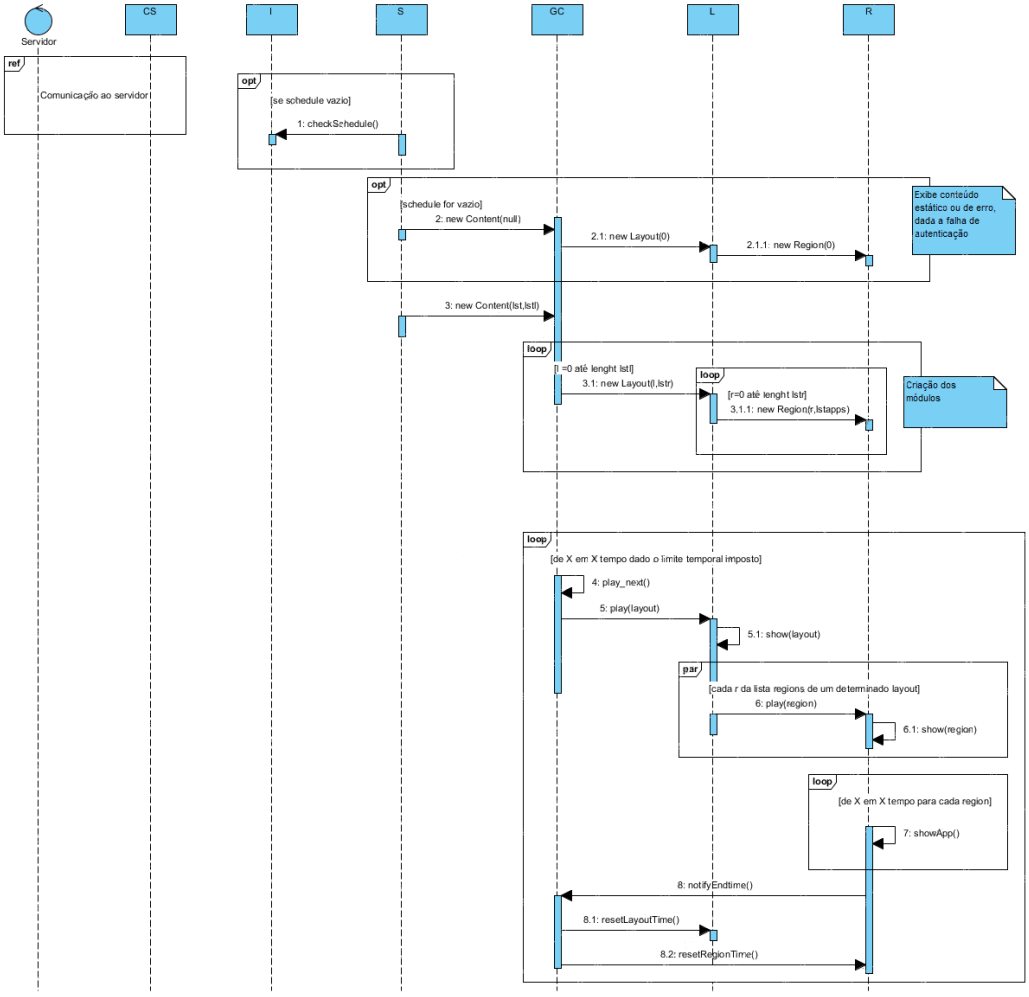


Figura 5.4: Diagrama de sequência do funcionamento da aplicação e escalonamento de conteúdos

mais aplicações. Na sua forma mais simples, assume-se que o *schedule* é constituído por uma sequência de aplicações apresentadas num *loop* numa única região. Apesar da simplicidade, esta versão define a hierarquia dos elementos de escalonamento:

- *'schedule'* - o elemento mais alto que especifica as propriedades de alto nível do *schedule*;
- *'seq'* - o elemento sequência que especifica uma sequência de aplicações;
- *'ref'* - a referência actual da aplicação a ser apresentada.

5.3.1.1 O elemento schedule

O elemento *'schedule'* define as propriedades de alto nível, como o *schedule* ID, nome, mas também especifica as posições e dimensões alvo para o ecrã:

- *'id'* - a identificação do *schedule* (*string*);
- *'name'* - o nome do *schedule* (*string*);
- *'updatedOn'* - data/hora da última actualização (*datetime RFC 3339*);
- *'left'* - a margem esquerda relativamente à janela do browser (*percentage as double*);
- *'top'* - a margem do topo relativamente à janela do browser (*percentage as double*);
- *'width'* - a largura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*);
- *'height'* - a altura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*).

O elemento *'schedule'* contém um conjunto de elementos filhos que especificam o comportamento do *schedule*. Nesta versão, o *schedule* contem um elemento *'seq'* único, especificando uma sequência.

5.3.1.2 O elemento seq

O elemento *'seq'* especifica um conjunto de aplicações a serem apresentadas sequencialmente de acordo com a sua respectiva ordem. É composto com as seguintes propriedades:

- *'repeatCount'* - especifica o número de vezes que esta sequência deve ser iterada. Pode conter um valor inteiro ou uma keyword "*indefenite*", que significa que a sequência deve iterar infinitamente;
- *'left'* - a margem esquerda relativamente ao *container* (*percentage as double*);
- *'top'* - a margem do topo relativamente ao *container* (*percentage as double*);
- *'width'* - a largura da área destino da sequência relativamente ao *container* (*percentage as double*);
- *'height'* - a altura da área destino da sequência relativamente ao *container* (*percentage as double*).

Todas estas propriedades são opcionais. O elemento *'seq'* contém um conjunto de elementos filhos *ref*, que referem às aplicações a ser apresentadas.

5.3.1.3 O elemento ref

O elemento *'ref'* especifica em particular como cada aplicação deve ser apresentada dentro do *scheduling container*. Além da referência para a aplicação, o elemento *'ref'* especifica o espaço para a apresentação, duração e tipo:

- *'src'* - o URL da aplicação;
- *'type'* - o tipo MIME da aplicação (apenas *text/html*);
- *'dur'* - a duração da apresentação em segundos (*double*).

As especificações do ecrã são sempre relativas ao container. O único elemento do *schedule* onde as especificações do ecrã são obrigatórias é no elemento da raíz (*schedule*). Em qualquer outro elemento da hierarquia, se nenhum *left/top/width/height* é definido então, esse elemento utiliza as dimensões definidas pelo *container*.

5.3.2 Versão 2

O formato *schedule* da versão 2 do anexo A tem como objectivo o suporte de várias regiões de *scheduling* de uma sequência de uma ou mais aplicações. Na sua forma mais simples, assume-se que o *schedule* é constituído por uma lista de regiões que são apresentadas em ciclo; cada região contêm uma sequência de aplicações. Esta versão define a hierarquia dos elementos de escalonamento:

- *'schedule'* - o elemento mais alto que especifica as propriedades de alto nível do *schedule*;
- *'regions'* - o elemento que especifica as propriedades de cada região;
- *'seq'* - o elemento sequência que especifica uma sequência de aplicações;
- *'ref'* - a referência actual da aplicação a ser apresentada.

5.3.2.1 O elemento schedule

O elemento *'schedule'* define as propriedades de alto nível, como o *schedule* ID, nome, mas também especifica as posições e dimensões alvo para o ecrã:

- *'id'* - a identificação do *schedule* (*string*);
- *'name'* - o nome do *schedule* (*string*);
- *'updatedOn'* - data/hora da última actualização (*datetime RFC 3339*);

- *'left'* - a margem esquerda relativamente à janela do browser (*percentage as double*);
- *'top'* - a margem do topo relativamente à janela do browser (*percentage as double*);
- *'width'* - a largura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*);
- *'height'* - a altura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*).

O elemento *'schedule'* contém um conjunto de elementos filhos que especificam o comportamento do *schedule*. Nesta versão, o *schedule* contém um elemento *'regions'* único, especificando uma lista de regiões.

5.3.2.2 O elemento regions

O elemento *'regions'* especifica um conjunto de regioões a serem apresentadas sequencialmente de acordo com a sua respectiva ordem. É composto com as seguintes propriedades:

- *'layout'* - a identificação da região (*string*);
- *'left'* - a margem esquerda relativamente à janela do browser (*percentage as double*);
- *'top'* - a margem do topo relativamente à janela do browser (*percentage as double*);
- *'width'* - a largura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*);
- *'height'* - a altura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*).

Cada elemento *'region'* contêm um elemento seq único que especifica uma sequência.

5.3.2.3 O elemento seq

O elemento *'seq'* especifica um conjunto de aplicações a serem apresentadas sequencialmente de acordo com a sua respectiva ordem. É composto com as seguintes propriedades:

- *'repeatCount'* - especifica o número de vezes que esta sequência deve ser iterada. Pode conter um valor inteiro ou uma keyword "*indefinite*", que significa que a sequência deve iterar infinitamente;
- *'left'* - a margem esquerda relativamente ao *container* (*percentage as double*);
- *'top'* - a margem do topo relativamente ao *container* (*percentage as double*);
- *'width'* - a largura da área destino da sequência relativamente ao *container* (*percentage as double*);
- *'height'* - a altura da área destino da sequencia relativamente ao *container* (*percentage as double*).

Todas estas propriedades são opcionais. O elemento *'seq'* contêm um conjunto de elementos filhos *ref* que referem às aplicações a serem apresentadas.

5.3.2.4 O elemento ref

O elemento *'ref'* especifica em particular como cada aplicação deve ser apresentada dentro do *scheduling container*. Além da referência para a aplicação, o elemento *'ref'* especifica o espaço para a apresentação, duração e o tipo:

- *'src'* - o URL da aplicação;
- *'type'* - o tipo MIME da aplicação (apenas *text/html*);
- *'dur'* - a duração da apresentação em segundos (*double*).

As especificações do ecrã são sempre relativas ao *container*. O único elemento do *schedule* onde as especificações do ecrã são obrigatórias é no elemento da raíz (*schedule*). Em qualquer outro elemento da hierarquia, se nenhum *left/top/width/height* é definido então, esse elemento utiliza as dimensões definidas pelo *container*.

5.3.3 Versão 3

O formato *schedule* da versão 3 do anexo A tem como objectivo o suporte de vários layouts num player e uma reestruturação geral dos seus níveis. Assume-se que o *schedule* é constituído por uma lista de aplicações e layouts. Os layouts contêm uma espécie de níves de prioridade, onde apenas um deles é apresentado em ciclo, cada layout contem uma lista de regiões e por fim, cada região contem uma sequência de aplicações. Esta versão define a hierarquia dos elementos de escalonamento da seguinte maneira:

- *'schedule'* - o elemento mais alto que especifica as propriedades de alto nível do *schedule*;
- *'apps'* - o elemento que especifica a lista das aplicações web;
- *'normalContent'* - o elemento que especifica as propriedades de conteúdo de prioridade normal;
- *'regions'* - o elemento que especifica as propriedades de cada região;
- *'containerList'* - o elemento que especifica a referência a uma lista de aplicações.

5.3.3.1 O elemento schedule

O elemento *'schedule'* define as propriedades de alto nível, como a identificação e a lista de conteúdos necessários para o ecrã:

- *'id'* - a identificação do *schedule* (*string*);
- *'name'* - o nome do *schedule* (*string*);

- *'updatedOn'* - data/hora da última actualização (*datetime RFC 3339*);
- *'etag'* - identificação electrónica do ficheiro para determinar as suas alterações (*string*).

O elemento *'schedule'* contém um conjunto de elementos filhos que especificam a lista de aplicações a ser exibidas no ecrã, bem como o comportamento do *schedule*. Nesta versão, o *schedule* contém um elemento *'apps'* único que especifica uma lista de aplicações e uma lista de conteúdos que especifica o comportamento do *scheduler*.

5.3.3.2 O elemento apps

O elemento *'apps'* especifica em particular a informação necessária de cada aplicação que será necessária para o escalonamento:

- *'id'* - a identificação da aplicação (*string*);
- *'type'* - o tipo MIME da aplicação (*string*);
- *'src'* - o URL da aplicação (*string*).

5.3.3.3 O elemento normalContent

O elemento *'normalContent'* especifica um conjunto de layouts a serem apresentadas sequencialmente de acordo com a sua respectiva ordem. É composto com as seguintes propriedades:

- *'id'* - a identificação do layout (*string*);
- *'name'* - o nome do layout (*string*).

Cada elemento *'normalContent'* contém um elemento *'regions'* único que especifica uma sequência de regiões.

5.3.3.4 O elemento regions

O elemento *'regions'* especifica um conjunto de regioões a serem apresentadas sequencialmente de acordo com a sua respectiva ordem. Contem as as seguintes propriedades:

- *'id'* - a identificação da região (*string*);
- *'name'* - o nome da região (*string*);
- *'left'* - a margem esquerda relativamente à janela do browser (*percentage as double*);
- *'top'* - a margem do topo relativamente à janela do browser (*percentage as double*);
- *'width'* - a largura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*);
- *'height'* - a altura da área destino do *schedule*, relativamente à janela do browser (*percentage as double*);
- *'scheduleItem'* - indica o tipo de conteúdo que se encontra no elemento containerList (*string*);
- *'limitCycle'* - indica o ciclo limite para a lista de elementos em containerList (*integer*);
- *'selector'* - indica o tipo de iteração a ser efectuado no elemento containerList (*string*).

Cada elemento *'region'* contém um elemento *'containerList'* único, que especifica uma sequência de conteúdos. As especificações do ecrã são sempre relativas ao *container* layout. Em qualquer outro elemento da hierarquia, se nenhum *left/top/width/height* é definido então, esse elemento utiliza as dimensões definidas pelo *container* layout.

5.3.3.5 O elemento containerList

O elemento *'containerList'* especifica em particular como cada aplicação deve ser apresentada dento do *scheduling container*. Além da referência para a aplicação, o elemento dur especifica o tempo válido para a apresentação:

- *'cid'* - identificação da aplicação que esteja listada no elemento *'apps'* (*string*);
- *'dur'* - a duração da apresentação em segundos (*double*).

5.4 Informação

Os sistemas de ecrãs públicos apresentam várias particularidades relativamente à sua identificação numa rede de ecrãs públicos. Estas particularidades exigem que o player tenha uma espécie de identificação única no ambiente de execução no qual executam, de forma a obter um controlo e monitorização correcta do player. Como tal, foi implementado dentro dos possíveis, uma forma simples e de fácil compreensão para a gestão de dados relativos ao player no ambiente de execução.

No arranque do player, este irá verificar se já existe informação guardada em *localStorage*, caso contrário irá registar os seguintes dados: UUID [79, 80], largura e altura do ecrã, o *user-agent*, o nome e versão do navegador web e o nome e versão do sistema operativo. O UUID é criado através de uma biblioteca que gera uma identificação aleatória do tipo *timestamp* [81], a largura e altura do ecrã são obtidos através do objecto *screen*, o *user-agent* é obtido através do objecto *navigator*, os restantes dados são obtidos através do tratamento e parse do valor registado no *user-agent*, no qual é possível determinar, com uma certa limitação, os navegadores web e os sistemas operativos mais utilizados a nível mundial.

```
UUID: c8d6afc0-1ae6-11e3-9ba3-ed5701427713
screen :: (width x height): 1280 x 800
navigator :: user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_5_8)
                         AppleWebKit/537.1 (KHTML, like Gecko)
```

```
                          Chrome/21.0.1180.90 Safari/537.1
user-agent (parsed): browser :: name: Chrome | major version: 21
                                fullversion: 21.0.1180.90
user-agent (parsed): OS :: name: Mac OS X | version: 10_5_8
```

Os campos registados no player localmente, serão utilizados em a finalidade de registo no sistema central de uma rede de ecrãs, de forma a saber quais as especificações e condições em que o player está a ser executado. A ter em conta que o valor do UUID é gerado aleatoriamente, sempre que pedido. Como tal, caso o player seja executado noutro navegador web, os valores registados em *localStorage* noutro navegador web não são transmissíveis, originando um novo UUID e respectivos novos campos.

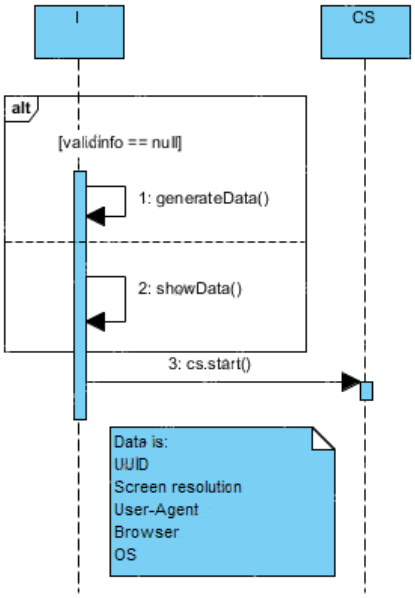


Figura 5.5: Funcionamento do módulo de Informação

Concluíndo, o módulo representado na figura 5.5, tem como funcionalidade principal a gestão e manutenção dos dados do ambiente de execução.

5.5 Comunicação Servidor

Os sistemas de ecrãs públicos exigem que um player, através dos dados de identificação que possui, uma forma de autenticação no sistema para assegurar a integridade da rede, bem como controlar os pedidos e encaminhamento de schedules de acordo com os pedidos efectuados pelos players.

Após a verificação dos dados informativos relativos ao player, o passo seguinte consiste na autenticação do player ao sistema através do UUID, sistema este que verificaria se o UUID se encontraria na base de dados, bem como os restantes dados informativos relativos ao player. Numa situação em que a ligação à rede se encontra activa, caso a autenticação seja inválida, o player exibe uma mensagem de erro no ecrã, senão efectua o pedido de *schedule* ao servidor e aguarda a recepção do mesmo. Caso a ligação à rede se encontre inactiva, este passo é ignorado e define o valor do *schedule* como nulo e inicia o seu funcionamento em modo offline.

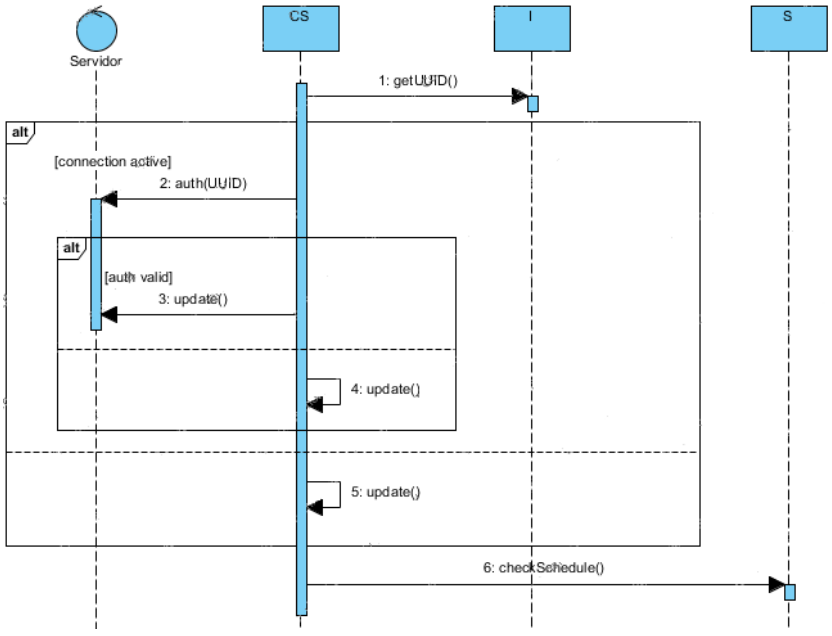


Figura 5.6: Funcionamento do módulo de Comunicação Servidor

A ter em conta que o processo de autenticação não está implementado,

dada a inexistência de um sistema que gere os registos de players numa rede de ecrãs públicos, verificando apenas localmente se o UUID está gerado localmente.

Concluíndo, o módulo representado na figura 5.6 tem como funcionalidade principal a autenticação do player e respectivo pedido do *schedule* ao sistema que gere a rede de ecrãs públicos, independentemente do estado da ligação à rede.

5.6 Schedule

Os vários tipos de *schedule* exigem que o player tenha a capacidade de análise e interpretação dos mesmos, como tal, esta secção descreve como o módulo efectua a análise, a interpretação e a escolha do *schedule*.

O primeiro passo deste módulo consiste no carregamento do *schedule* local, que se encontra armazenado em *localStorage*, caso não esteja, esse valor será nulo. Posteriormente, é feita uma verificação ao *schedule* temporário, recebido do módulo anterior, caso este seja diferente de nulo é comparado com o *schedule* local. Da comparação dos schedules resulta um dos seguintes:

- se ambos forem idênticos, regista os dados básicos do *schedule* local e o processo neste módulo termina;
- se forem diferentes, é feita uma verificação ao *schedule* temporário.

Caso seja validado pelo player, o mesmo é registado em *localStorage* para utilização futura, assim como o seu tipo.

Outro dos processos consiste na possibilidade do *schedule* temporário ser nulo, caso isso aconteça e o *schedule* local ser diferente de nulo, toma precedência registando os dados básicos do mesmo, caso contrário, o *schedule* local será declarado inválido e sem efeito para utilização no player, exibindo uma mensagem de erro.

Concluíndo, o módulo representado na figura 5.7 tem como funcionalidade principal a análise e interpretação do valor *schedule*.

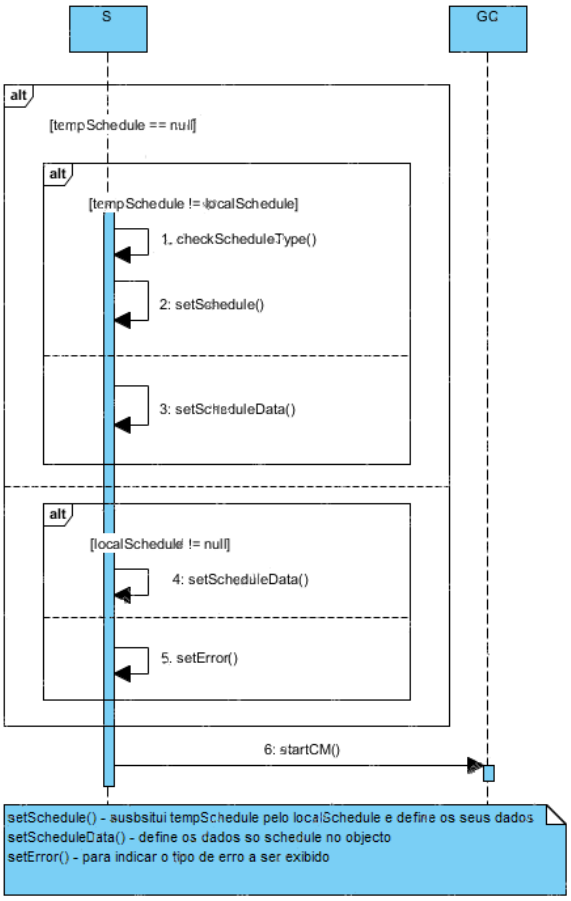


Figura 5.7: Funcionamento do módulo de Schedule

5.7 Gestor Conteúdos

O escalonamento de conteúdo num player, opera com base numa lista de itens obtidos do módulo *'Schedule'* que contém um conjunto de especificações e regras de relevância para o conteúdo que configuram, o comportamento definido para o gestor de conteúdos. Com esta informação, o escalonador deve ter a capacidade de seleccionar e controlar os conteúdos que são exibidos no ecrã.

De forma a obter um melhor controlo sobre os conteúdos a serem exibidos no ecrã, este módulo subdivide-se nos módulos de *'Layout'*, responsável pela gestão e criação das regiões dependentes a esse layout, e *'Regiões'*, responsável pela gestão dos conteúdos que são exibidos nessa região.

O primeiro passo desde módulo consiste na verificação do *schedule* local e respectivo tipo. No caso do *schedule* local ser nulo, será exibida uma mensagem de erro no ecrã indicando a falta de um *schedule*, caso contrário, através do tipo do mesmo, inicia a construção do gestor de conteúdos através do formato dado para esse tipo. Este módulo é responsável pelo registo de todas as aplicações web numa lista de aplicações, que serão exibidas no ecrã, e criação dos Layouts para o gestor de conteúdos que serão registados numa lista de layouts, conforme a figura 5.8.

Findo a criação dos Layouts e Regiões, este efectua o *'play'* do player. Este método irá definir um layout actual da lista de layouts (normalmente é o primeiro da lista), e efectua *'play'* no layout escolhido. No *'Layout'*, o método *'play'*, pega em cada elemento da lista de regiões e executa o método *'play'* presente em Regiões. Por fim, no *'Regiões'*, para cada região acontece o seguinte: é escalonada uma aplicação da lista de conteúdos para cada região, operando com base na lista de regras especificadas no *schedule*, ilustrado na figura 5.9.

Findo o tempo cíclico para uma região, esta notifica o gestor de conteúdos da conclusão da mesma. Quando todas as regiões terminarem, o gestor de conteúdos executa o método *'reset'* a todos os seus sub-módulos, de forma a limpar os valores que contam os ciclos e escalona para o próximo layout, efectuando o mesmo processo antes referido e referenciado na figura 5.10.

5.8 Logs

A informação relativa ao comportamento do player, representa um importante contributo para a monitorização do mesmo. Como tal, foi implementado uma maneira simples e de fácil compreensão o registo e consulta dessas acções.

Este módulo consiste no carregamento de uma lista de acções, representada no anexo B, guardada previamente em *localStorage*. Findo o carregamento, aguarda pela invocação de registo de acções provenientes de qualquer módulo. Um registo de uma acção consiste nos seguintes dados:

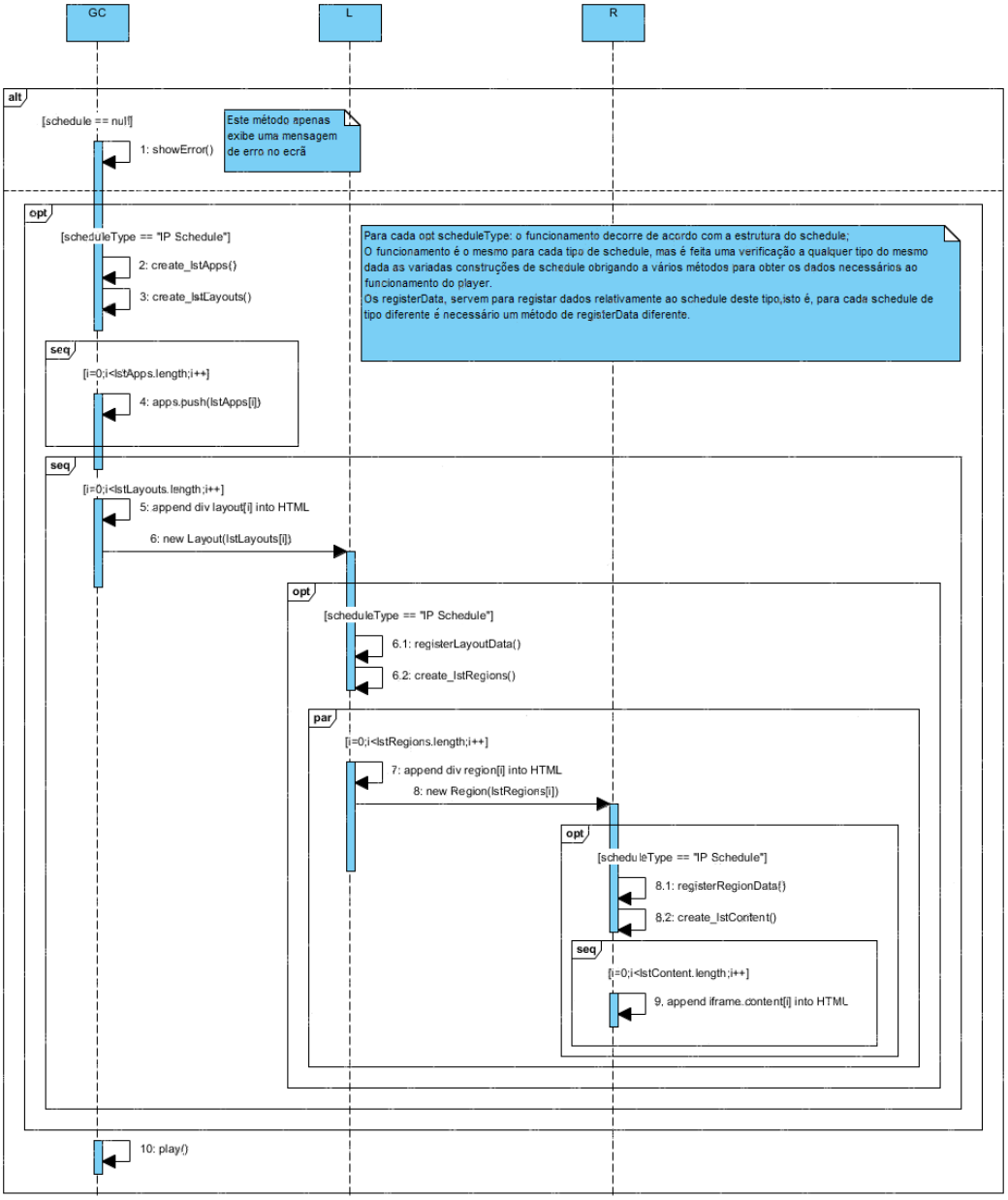


Figura 5.8: Funcionamento do primeiro passo do módulo de Gestor Conteúdos

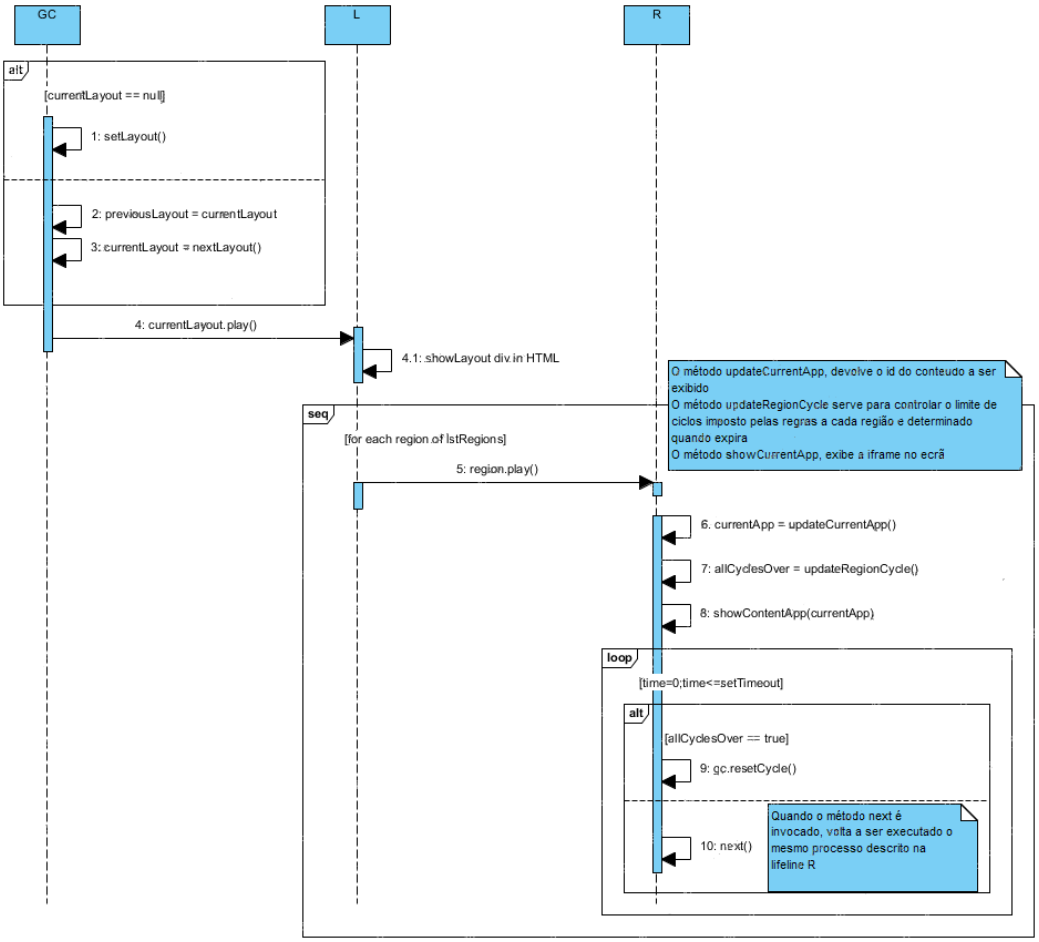


Figura 5.9: Funcionamento do segundo passo do módulo de Gestor Conteúdos

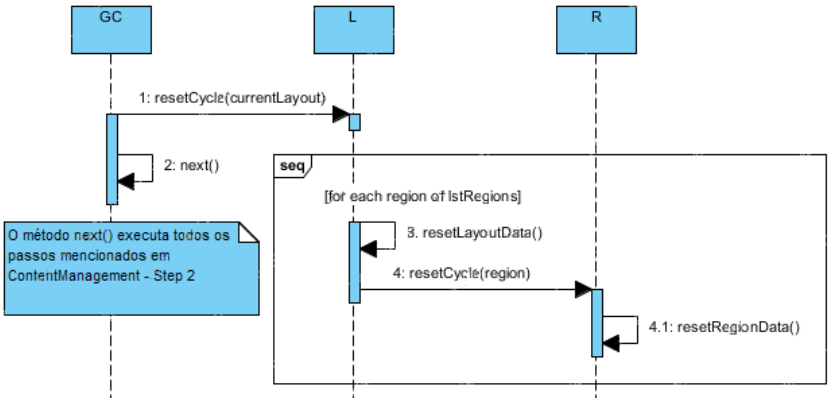


Figura 5.10: Funcionamento do terceiro passo do módulo de Gestor Conteúdos

um número de entrada, obtido da extensão do número de entradas presentes na lista de logs incrementado por o valor um; a data actual do registo e a acção, em texto, recebida na invocação. Registado uma nova entrada, concatena-se a mesma à lista de acções e actualiza-se a lista no *localStorage*.

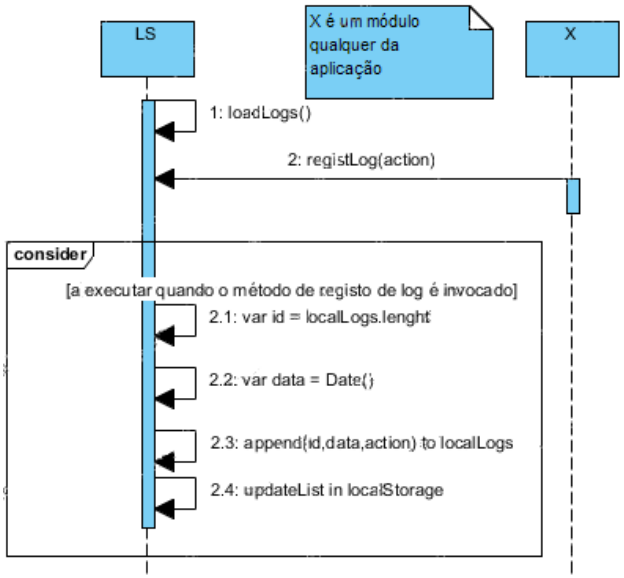


Figura 5.11: Funcionamento do módulo de Logs

Concluíndo, o módulo representado na figura 5.11, tem como funcionalidade principal o carregamento e o registo de acções a pedido dos módulos, tendo como objectivo a consulta da lista de acções através do sistema de uma rede de ecrãs possibilitando a monitorização das acções dos players nos respectivos ecrãs.

5.9 Chrome Apps

Além dos aspectos necessários ao funcionamento de uma aplicação nesta plataforma, representados no anexo C, existem duas diferenças entre o protótipo para um navegador web convencional e a plataforma Chrome Apps. Relativamente ao módulo *'Gestão Conteúdos'*, na situação onde se anexa o elemento *iframe* ao HTML:

```
var el = \$('<iframe sandbox="allow-same-origin allow-scripts" ↩
    id="'+id+'" src="' + src + '" scrolling="no" />');
```

Deve-se substituir pelo elemento *webview* da seguinte maneira:

```
var el = \$('<webview id="app" partition="' + app.cid + '" ↩
    style="width: '+wvWidth+'px; height: '+wvHeight+'px; ↩
    overflow:hidden;" src="' + src + '"> </webview>');
```

Nos restantes módulos onde é efectuado a leitura e escrita de dados através de web storage:

```
uuid = localStorage.getItem('playerUUID');
localStorage.setItem('playerUUID',uuid);
```

Deve ser substituído pelos seguintes métodos:

```
chrome.storage.local.get('playerUUID', function () {
        ...
});
chrome.storage.local.set({'playerUUID':uuid}, function() {
        ...
});
```

Apesar da maioria do desenvolvimento para um navegador web convencional ser complatível nesta plataforma, deve-se ter em consideração as APIs que a plataforma oferece e respectivas capacidades e limitações que uma plataforma convencional não oferece.

5.10 Estado de execução e operação

O funcionamento do protótipo pressupõe, da realização de um conjunto de tarefas por parte de cada módulo descrito anteriormente e dos quais é possível verificar num ecrã através de um formato especificado no *schedule* referenciado no anexo A.

Dado o seguinte esquema de *schedule*, representado na figura 5.12, é previsível que o protótipo exiba conteúdos conforme o esquema descrito

anteriormente. Caso o formato *schedule* seja correcto, é escalonado conteúdo para cada região conforme se verifica na figura 5.13:

As ilustrações do estado de execução para o protótipo são válidas para as plataformas desenvolvidas na dissertação.

5.11 Resumo

Neste capítulo foi apresentada a arquitectura geral do sistema e a estrutura modular da aplicação que combina uma caracterização responsável do protótipo. Foi, também, apresentado a estrutura e definição do formato do escalonamento para o protótipo que combina uma série de regras e prioridades, que foram implementadas no mesmo. O protótipo descreve-se pela constituição e caracterização do funcionamento e interacção dos vários módulos que o compõem. Por fim, é feita uma comparação às diferenças do desenvolvimento do protótipo para a plataforma de Chrome Apps e um exemplo do estado de execução e operação do mesmo.

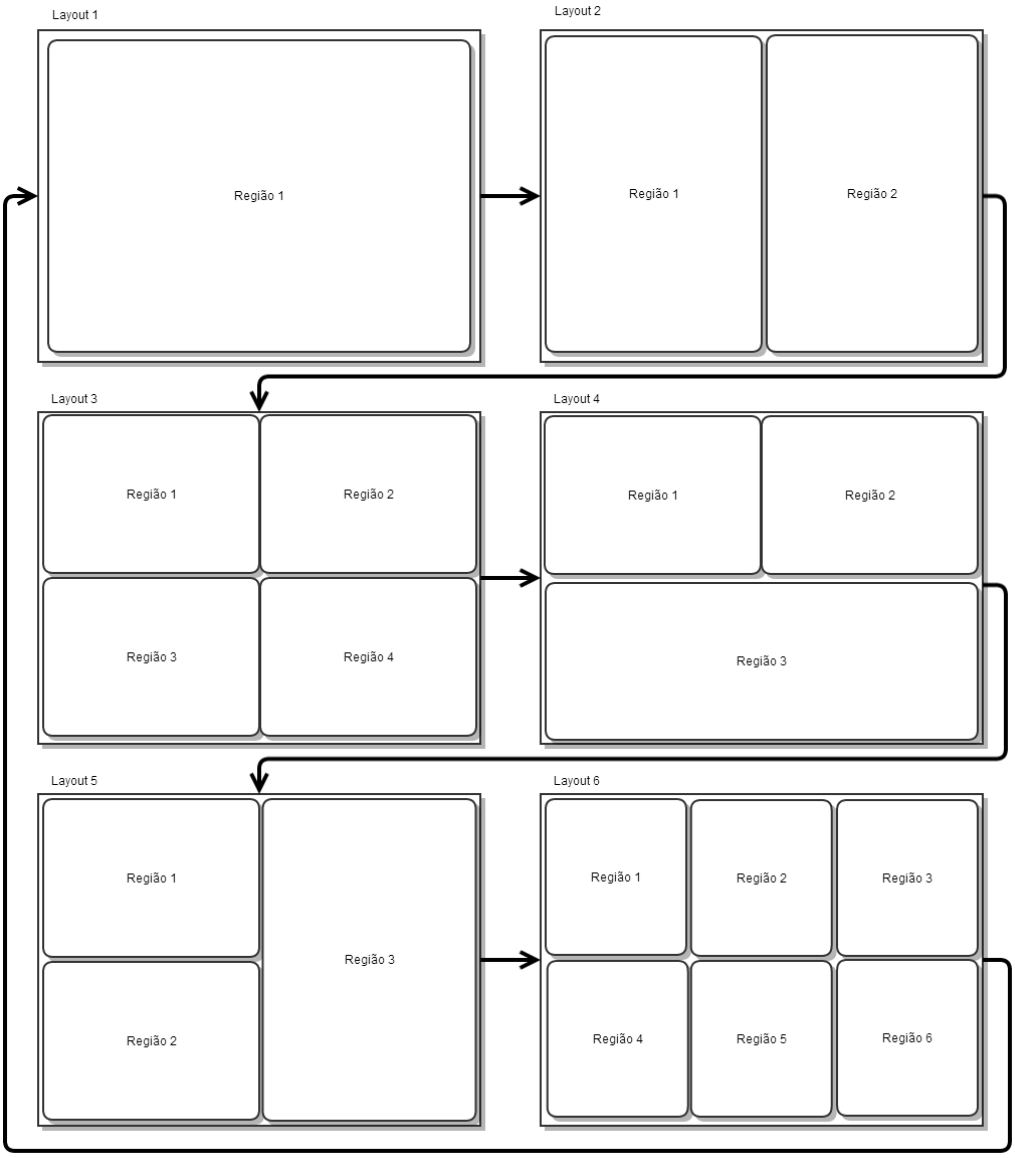


Figura 5.12: Exemplo de um esquema de escalonamento

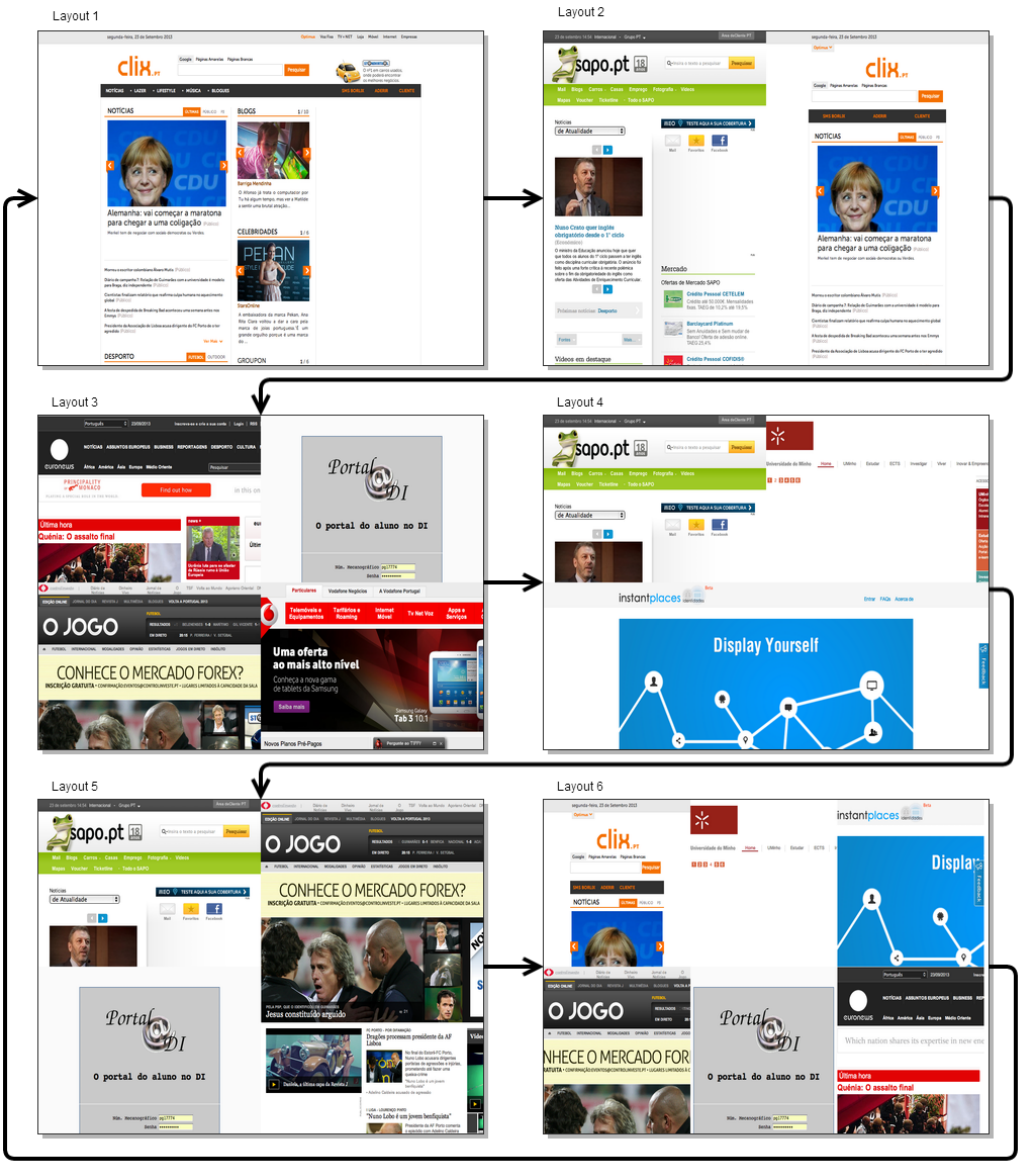


Figura 5.13: Demonstração de escalonamento de conteúdos web

6
Conclusão

Neste capítulo são apresentadas as considerações finais em relação ao trabalho desenvolvido. É feito uma síntese do trabalho realizado, que inclui o resumo do processo de investigação, designadamente questões, hipóteses colocadas e os resultados obtidos. Na parte final são apontadas linhas de investigação futura, de forma a dar continuidade ao trabalho realizado.

6.1 Síntese do trabalho realizado

O objectivo principal da dissertação era propor o desenvolvimento de um projecto *open source* de um player de apresentação de conteúdos em redes de ecrãs públicos, recorrendo às crescentes potencialidades das tecnologias web. Dada esta abordagem, a identificação e caracterização dos principais desafios inerentes à apresentação de conteúdos, permite identificar as capacidades e limitações das tecnologias web no desenvolvimento da aplicação tendo em conta as funcionalidades necessárias para o funcionamento do mesmo.

A caracterização do espaço de desenho de um player num ambiente web, é obtida através da identificação das plataformas computacionais para dispositivos media, bem como uma análise aos players existentes e respectivos ambientes de execução, de forma a obter uma análise comparativa e definir a caracterização e funcionalidades para o player.

O formato de representação das regras de escalonamento de conteúdos web foi desenvolvido com experiência em formatos similares e uniformizados SMIL.

No sentido de atingir o objectivo principal da dissertação foram realizadas as seguintes tarefas:

- foi realizado um estudo do estado de arte através de um conjunto de artigos científicos na área de players *digital signage* e das tecnologias web, de forma a contribuir para: o entendimento do conceito das redes de ecrãs públicas; o conceito na utilização de aplicações nas redes de ecrãs públicas, as capacidades e limitações na utilização das tecnologias web para o desenvolvimento de aplicações em ambiente web e a importância para o desenvolvimento da dissertação;

- foi realizada uma análise e levantamento de requisitos às plataformas computacionais para investigar as suas capacidades e limitações, de forma a poder analisar um conjunto de sistemas de hardware existentes no mercado; caracterizar o ambiente de execução dessas plataformas (hardware, sistema operativo e navegador web) e comparar os resultados obtidos com o objectivo de definir uma ou várias plataformas para a utilização de players;

- foi realizada uma análise e levantamento de requisitos aos players *digital signage* com o intuito: descrever as características e funcionalidades; verificar as dependências dos mesmos nos ambientes de execução; compará-los para ter uma visão geral: das suas funcionalidades, ambiente de execução, formatos de linguagem e formatos *schedule*, de forma a obter uma lista de funcionalidades mínimas necessárias para o funcionamento do player;

- foi realizados estudos técnicos às funcionalidades de tecnologias web, tais como, HTML5 e *JavaScript*, de forma a contribuir para o desenvolvimento de aplicações web e determinar as capacidades e limitações das funcionalidades que estas oferecem para o total funcionamento das operações necessárias na execução do player. Também foi feita

uma análise ao funcionamento dos navegadores web e respectivo desempenho do *JavaScript* com o elemento *iframe* do HTML5. Este estudo permitiu identificar algumas limitações em certos elementos do HTML5, bem como na execução de scripts por *JavaScript*, resultando em mau desempenho por parte dos navegadores web. No entanto, foram estudadas várias alternativas para a resolução destes problemas através do estudo e análise de três plataformas (Chrome Extensions, Chrome Apps e Awesomium), concluíndo Chrome Apps ser a opção mais viável para a resolução dos problemas e deixando ainda como hipótese remota a problemas futuros a plataforma Awesomium;

- foram propostas várias abordagens à implementação do *schedule* através da linguagem já conhecida da maior parte dos players para a representação de conteúdo, o SMIL. Estas foram baseadas através da linguagem de formato aberto JSON, numa tentativa de manter o mesmo aspecto na definição de items para o mesmo;
- foi proposto um modelo e uma arquitectura para o desenvolvimento do protótipo, baseado em módulos aplicacionais de forma a assegurar uma separação lógica e uma distribuição de tarefas entre os mesmos.

6.2 Trabalho futuro

Foram identificadas algumas questões que, por limitações temporais ou desenquadramento dos objectivos principais da dissertação, ficaram por desenvolver, abrindo novas oportunidades para trabalho futuro. Nesta sequência, indicam-se alguns pontos de partida para trabalho futuro:

- uma actualização à investigação das tecnologias web será fundamental e obrigatória. Estas tecnologias estão em constante actualização, por vezes, no decorrer da dissertação deparava com alterações aos estudos que já tinham sido efectuados;
- apesar da potencialidade das tecnologias web ser suficiente para a portabilidade do player, para outro tipo de plataformas referidas no

capítulo 4.6, poderão existir diferenças consideráveis relativamente ao ambiente de execução e que necessitarão de testes das plataformas de hardware, para garantir uma maior uniformização dos vários ambientes de execução, ou potencialmente, reduzir esse número a uma camada específica por forma a reduzir o número de diferenças entre as mesmas;

- uma actualização e implementação de módulos na arquitectura do player para suportar entidades externas ao player, como por exemplo, a implementação de um *back office* para pedidos e respostas de métodos (autenticação, logs, etc) ou sensores. Independentemente da funcionalidade, é necessário ter em conta os módulos já existentes e as plataformas onde estes devem ser desenvolvidos;

- a implementação do player para as stacks mencionadas no capítulo 4.6, apresentam vários desafios técnicos dada as diferenças no ambiente de execução. Apesar da aplicação se encontrar desenvolvida para várias stacks, é aconselhável manter o desenvolvimento geral em HTML5 e daí fazer *branches* da geral para outro tipo de ambientes de execução;

- o modelo de *schedule* apenas obedece a duas regras (sequencial e aleatório). A possibilidade de implementação de um modelo e algoritmo para escalonamento de conteúdo deve ser considerada vital de forma a potenciar a utilização da rede de ecrãs públicos. Como tal, sugerem-se mudanças no schedule que é utilizado actualmente, de forma a introduzir alterações ao nível de contexto, recorrendo a *tags* através dos interesses dos utilizadores, bem como ao nível temporal através da introdução de ciclo horário nos *schedules*.

A

Formatos do ficheiro schedule

Listing A.1: Schedule, versão 1

```
{
 "schedule": {
   "id": "416a18d6-7e42-4f8a-ac24-e902abe76f39",
   "name": "DSIschedule",
   "updatedOn": "2012-11-06T16:34:41.000Z",
   "left": "0.0",
   "top": "0.0",
   "width": "1.0",
   "height": "1.0",
   "seq": {
     "repeatCount": "indefinite",
     "ref": [
       {
         "src": "http://www.google.com",
         "type": "text/html",
         "dur": "10"
       },
       {
         "src": "http://www.apple.com",
         "type": "text/html",
         "dur": "9.8"
       }
     ]
   }
 }
}
```

Listing A.2: Schedule, versão 2

```
{
 "schedule": {
   "id": "416a18d6-7e42-4f8a-ac24-e902abe76f39",
   "name": "DSIschedule",
   "updatedOn": "2012-11-06T16:34:41.000Z",
   "left": "0.0",
   "top": "0.0",
   "width": "1.0",
   "height": "1.0",
   "regions": [
     {
       "layout": "content1",
       "left": "5.0",
       "top": "5.0",
       "width": "1.0",
       "height": "0.5",
       "seq": {
         "repeatCount": "indefinite",
         "ref": [
           {
             "src": "http://www.sapo.pt",
             "type": "text/html",
             "dur": "15"
           },
           {
             "src": "http://www.uminho.pt",
             "type": "text/html",
             "dur": "20"
           }
         ]
       }
     },
     {
       "layout": "content2",
       "left": "5.0",
       "top": "5.0",
       "width": "1.0",
       "height": "0.5",
       "seq": {
         "repeatCount": "indefinite",
         "ref": [
           {
             "src": "http://www.uminho.pt",
             "type": "text/html",
             "dur": "15"
           },
```

```
            {
              "src": "http://www.sapo.pt",
              "type": "text/html",
              "dur": "20"
            }
          ]
        }
      }
    ]
  }
}
```

Listing A.3: Schedule, versão 3

```
{
  "schedule": {
    "id": "416a18d6-7e42-4f8a-ac24-e902abe76f39",
    "name": "DSIschedule",
    "updatedOn": "2012-11-06T16:34:41.000Z",
    "etag": "esxrdctfvygbuhnij5464df7g8hu",
    "applications": [
      {
        "id": "app001",
        "type": "text/html",
        "src": "http://www.sapo.pt"
      },
      {
        "id": "app002",
        "type": "text/html",
        "src": "http://www.euronews.net"
      },
      {
        "id": "app003",
        "type": "text/html",
        "src": "http://www.jn.pt"
      },
      {
        "id": "app004",
        "type": "text/html",
        "src": "http://www.instantplaces.org"
      }
    ],
    "normalContent": [
      {
        "layout_id": "layout001",
        "layout_name": "layout1",
        "layout_dur": "indefinite",
```

```
34          "regions": [
35            {
36              "region_id": "region001",
37              "region_name": "regiao1",
38              "left": "0",
39              "top": "0",
40              "width": "0.5",
41              "height": "1",
42              "minWidth": "0",
43              "minHeight": "0",
44              "scheduleItem": "container",
45              "limitCycle": "1",
46              "selector": "seq",
47              "containerList": [
48                {
49                  "cid": "app001",
50                  "dur": "15"
51                },
52                {
53                  "cid": "app002",
54                  "dur": "15"
55                }
56              ]
57            },
58            {
59              "region_id": "region002",
60              "region_name": "regiao2",
61              "left": "0.5",
62              "top": "0",
63              "width": "0.5",
64              "height": "1",
65              "minWidth": "0",
66              "minHeight": "0",
67              "scheduleItem": "container",
68              "limitCycle": "1",
69              "selector": "seq",
70              "containerList": [
71                {
72                  "cid": "app003",
73                  "dur": "15"
74                },
75                {
76                  "cid": "app004",
77                  "dur": "15"
78                }
79              ]
80            }
```

```
        ]
      },
      {
        "layout_id": "layout002",
        "layout_name": "layout2",
        "layout_dur": "indefinite",
        "regions": [
          {
            "region_id": "region003",
            "region_name": "regiao3",
            "left": "0",
            "top": "0",
            "width": "1",
            "height": "1",
            "minWidth": "0",
            "minHeight": "0",
            "scheduleItem": "container",
            "limitCycle": "1",
            "selector": "seq",
            "containerList": [
              {
                "cid": "app002",
                "dur": "25"
              }
            ]
          }
        ]
      },
      {
        "layout_id": "layout003",
        "layout_name": "layout3",
        "layout_dur": "indefinite",
        "regions": [
          {
            "region_id": "region004",
            "region_name": "regiao4",
            "left": "0",
            "top": "0",
            "width": "1",
            "height": "1",
            "minWidth": "0",
            "minHeight": "0",
            "scheduleItem": "container",
            "limitCycle": "1",
            "selector": "seq",
            "containerList": [
              {
```

```
128                 "cid": "app003",
129                 "dur": "25"
130               }
131             ]
132           }
133         ]
134       }
135     ]
136   }
137 }
```

B

Registo de logs no player

Listing B.1: Registo de acções no player

```
{"actions":[
{"id":1,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):555","log":"Starting Player InstantPlaces..."},
{"id":2,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):556","log":"Checking Player information..."},
{"id":3,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):556","log":"Generating information..."},
{"id":4,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):559","log":"ServerCommunication :: Attempting to obtain a schedule ..."},
{"id":5,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):561","log":"ServerCommunication :: Schedule received"},
{"id":6,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):567","log":"Schedule :: Checking schedule..."},
{"id":7,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):568","log":"Schedule :: The type of schedule is: IP Schedule"},
{"id":8,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):568","log":"ContentManagement :: Starting Content Management..."},
{"id":9,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):569","log":"ContentManagement :: Adding applications..."},
{"id":10,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):570","log":"ContentManagement :: Application id: app001 | type: text/html | src: http://www.sapo.pt"},
{"id":11,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):570","log":"ContentManagement :: Application id: app002 | type: text/html | src: http://www.euronews.net"},
```

```
13 {"id":12,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):570","
     log":"ContentManagement :: Application id: app003 | type: text
     /html | src: http://www.jn.pt"},
14 {"id":13,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):571","
     log":"ContentManagement :: Application id: app004 | type: text
     /html | src: http://www.instantplaces.org"},
15 {"id":14,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):571","
     log":"ContentManagement :: Creating layouts..."},
16 {"id":15,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):572","
     log":"ContentManagement :: Layout id: layout001 | name:
     layout1 | dur: indefinite"},
17 {"id":16,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):573","
     log":"Layout :: Creating Regions..."},
18 {"id":17,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):573","
     log":"Layout :: Region id: region001 | name: regiao1"},
19 {"id":18,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):573","
     log":"Layout :: Region CSS | left: 0 | top: 0 | width: 0.5 |
     height: 1"},
20 {"id":19,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):574","
     log":"Region :: Adding content..."},
21 {"id":20,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):575","
     log":"Region :: Content id:: app001 | src: http://www.sapo.pt
     | dur: 25"},
22 {"id":21,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):575","
     log":"Region :: Content id:: app002 | src: http://www.euronews
     .net | dur: 15"},
23 {"id":22,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):575","
     log":"Layout :: Region id: region002 | name: regiao2"},
24 {"id":23,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):575","
     log":"Layout :: Region CSS | left: 0.5 | top: 0 | width: 0.5 |
      height: 1"},
25 {"id":24,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):577","
     log":"Region :: Adding content..."},
26 {"id":25,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):577","
     log":"Region :: Content id:: app003 | src: http://www.jn.pt |
     dur: 15"},
27 {"id":26,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):577","
     log":"Region :: Content id:: app004 | src: http://www.
     instantplaces.org | dur: 15"},
28 {"id":27,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):578","
     log":"ContentManagement :: Layout id: layout002 | name:
     layout2 | dur: indefinite"},
29 {"id":28,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):578","
     log":"Layout :: Creating Regions..."},
30 {"id":29,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):579","
     log":"Layout :: Region id: region003 | name: regiao3"},
```

```
{"id":30,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):579","log":"Layout :: Region CSS | left: 0 | top: 0 | width: 1 | height: 1"},
{"id":31,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):579","log":"Region :: Adding content..."},
{"id":32,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):579","log":"Region :: Content id:: app002 | src: http://www.euronews.net | dur: 25"},
{"id":33,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):580","log":"ContentManagement :: Layout id: layout003 | name: layout3 | dur: indefinite"},
{"id":34,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):580","log":"Layout :: Creating Regions..."},
{"id":35,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):581","log":"Layout :: Region id: region004 | name: regiao4"},
{"id":36,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):581","log":"Layout :: Region CSS | left: 0 | top: 0 | width: 1 | height: 1"},
{"id":37,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):581","log":"Region :: Adding content..."},
{"id":38,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):582","log":"Region :: Content id:: app003 | src: http://www.jn.pt | dur: 25"},
{"id":39,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):582","log":"ContentManagement :: Playing..."},
{"id":40,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):584","log":"ContentManagement :: Playing Layout ID: 0"},
{"id":41,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):585","log":"Layout :: Showing Layout: #layout001"},
{"id":42,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):586","log":"Region :: Showing Region: #region001"},
{"id":43,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):587","log":"Region :: Updating application..."},
{"id":44,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):587","log":"Region :: Updating cycle..."},
{"id":45,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):588","log":"Region :: Showing application: app001 for 25 seconds..."},
{"id":46,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):590","log":"Region :: Showing Region: #region002"},
{"id":47,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):590","log":"Region :: Updating application..."},
{"id":48,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):591","log":"Region :: Updating cycle..."},
{"id":49,"date":"Wed Sep 11 2013 14:33:46 GMT+0100 (WEST):594","log":"Region :: Showing application: app003 for 15 seconds..."},
```

```
51 (...)
52 {"id":147,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):843",
     "log":"ContentManagement :: Playing..."},
53 {"id":148,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):861",
     "log":"ContentManagement :: Playing Layout ID: 0"},
54 {"id":149,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):864",
     "log":"Layout :: Showing Layout: #layout001"},
55 {"id":150,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):865",
     "log":"Region :: Showing Region: #region001"},
56 {"id":151,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):866",
     "log":"Region :: Updating application..."},
57 {"id":152,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):867",
     "log":"Region :: Updating cycle..."},
58 {"id":153,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):868",
     "log":"Region :: Showing application: app001 for 25 seconds...
     "},
59 {"id":154,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):870",
     "log":"Region :: Showing Region: #region002"},
60 {"id":155,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):876",
     "log":"Region :: Updating application..."},
61 {"id":156,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):876",
     "log":"Region :: Updating cycle..."},
62 {"id":157,"date":"Wed Sep 11 2013 14:38:28 GMT+0100 (WEST):878",
     "log":"Region :: Showing application: app003 for 15 seconds...
     "},
63 {"id":158,"date":"Wed Sep 11 2013 14:38:43 GMT+0100 (WEST):881",
     "log":"Region :: Time is up for application app003"},
64 {"id":159,"date":"Wed Sep 11 2013 14:38:43 GMT+0100 (WEST):896",
     "log":"Region :: Updating application..."},
65 {"id":160,"date":"Wed Sep 11 2013 14:38:43 GMT+0100 (WEST):899",
     "log":"Region :: Updating cycle..."},
66 {"id":161,"date":"Wed Sep 11 2013 14:38:43 GMT+0100 (WEST):901",
     "log":"Region :: Showing application: app004 for 15 seconds...
     "},
67 {"id":162,"date":"Wed Sep 11 2013 14:38:53 GMT+0100 (WEST):872",
     "log":"Region :: Time is up for application app001"},
68 {"id":163,"date":"Wed Sep 11 2013 14:38:53 GMT+0100 (WEST):894",
     "log":"Region :: Updating application..."},
69 {"id":164,"date":"Wed Sep 11 2013 14:38:53 GMT+0100 (WEST):895",
     "log":"Region :: Updating cycle..."},
70 {"id":165,"date":"Wed Sep 11 2013 14:38:53 GMT+0100 (WEST):898",
     "log":"Region :: Showing application: app002 for 15 seconds...
     "}
71 ]}
```

C

Chrome Apps

Listing C.1: Ficheiro manifest

```
{
  // Required
  "name": "InstantPlaces Player",
  "version": "1.0",
  "manifest_version": 2,

  // Recommended
  "description": "InstantPlaces player",
  "icons": { "16": "iplogo16.png", "128": "iplogo128.png" },
  // "default_locale": "en",

  // Pick one (or none) OF browser_action, page_action, theme, app
  "app": {
    "background": {
      "scripts": [ "background.js", "js/jquery.min.js", "js/uuid.js", "js/appv5.js" ]
    }
  },

  "minimum_chrome_version": "23",

  "permissions": [ "fullscreen", "webview", "storage", "unlimitedStorage", "experimental" ]

}
```

Listing C.2: Script background

```
chrome.app.runtime.onLaunched.addListener(function() {
  // Tell your app what to launch and how.
  chrome.app.window.create('window.html', {
    width: 1920,
    height: 1080,
    state: 'fullscreen'
  });
});
```

Listing C.3: Ficheiro window da aplicação

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <!-- Always force latest IE rendering engine (even in intranet) & Chrome Frame Remove this if you use the .htaccess -->
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <title>instantplaces player</title>

    <link rel="stylesheet" type="text/css" href="css/style.css" />
    <script type="text/javascript" src="js/jquery.min.js"></script>
    <script type="text/javascript" src="js/uuid.js"></script>
    <script type="text/javascript" src="js/appv5.js"></script>
  </head>
  <body>
    <div id="content">
    </div>
  </body>
</html>
```

Bibliografia

[1] Nigel Davies, Marc Langheinrich, Rui José, and Albrecht Schmidt. Open Display Networks: A Communications Medium for the 21st Century. *IEEE Computer [Accepted for publication]*, 2012.

[2] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. Web browser as an application platform: The lively kernel experience. 2008.

[3] Tommi Heikkinen and Tomas Lindén. Declarative XML-based layout state encoding for managing screen real estate of interactive public displays. *Pervasive Computing . . .* , pages 82–87, 2011.

[4] T Linden, Tommi Heikkinen, and Timo Ojala. Web-based framework for spatiotemporal screen real estate management of interactive public displays. *Proceedings of the 19th . . .* , pages 1277–1280, 2010.

[5] Rui José, Constantin Taivan, José Miguel Andrade, and Hélder Pinto. Engineering Web Applications for Open Display Networks.

[6] F Nyrhinen and T Mikkonen. Web Browser as a Uniform Application Platform: How Far Are We? In *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, pages 578–584, 2009.

[7] A Taivalsaari, T Mikkonen, M Anttonen, and A Salminen. The Death of Binary Software: End User Software Moves to the Web. In *Creating,*

Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on, pages 17–23, 2011.

[8] Tommi Mikkonen and Antero Taivalsaari. Using JavaScript as a Real Programming Language Using JavaScript as a Real Programming Language. 2007.

[9] S Clinch, Nigel Davies, Adrian Friday, and Graham Clinch. Yarely: a software player for open pervasive display networks. *... on Pervasive Displays*, 2013.

[10] W3C. HTML5 differences from HTML4. http://www.w3.org/TR/2011/WD-html5-diff-20110405/, January 2013.

[11] W3C. Position Paper for the W3C Workshop on Web Applications and Compound Documents. http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html, January 2013.

[12] WhatWG. This Week in HTML 5 – Episode 5. http://blog.whatwg.org/this-week-in-html-5-episode-5, January 2013.

[13] W3C. HTML5 — Smile, it's a Snapshot! http://www.w3.org/QA/2012/12/html5_smile_its_a_snapshot.html, January 2013.

[14] SJ Vaughan-Nichols. Will HTML 5 restandardize the web? *Computer*, (April 2010):13–15, 2010.

[15] W3C HTML5. Plan 2014 HTML5. http://dev.w3.org/html5/decision-policy/html5-2014-plan.html, January 2013.

[16] SFF. Small Form Factor PCs. http://compreviews.about.com/od/cases/a/SFFPCs.htm, January 2013.

[17] Rtings. What is Smart TV. http://www.rtings.com/info/what-is-smart-tv, January 2013.

[18] Raspberry. Raspberry Pi - FAQ. http://scratch.mit.edu/, January 2013.

[19] pcDuino. pcDuino: miniPC + Arduino. http://www.pcduino.com/, March 2013.

[20] Rikomagic. Rikomagic - MK 802 Mini PCs. http://www.rikomagic.co.uk/, March 2013.

[21] Google. Samsung Chromebox. https://www.google.com/intl/en/chrome/devices/chromebox.html, May 2013.

[22] Samsung. Cloud Display - Samsung. http://www.samsung.com/levant/business/business-products/cloud-display, May 2013.

[23] Ubisign. Ubisign Digital Signage. http://www.ubisign.com/public/UserFiles/Downloads/ProductSheet_webVersion.pdf, February 2013.

[24] Xibo. Xibo - Digital Signage. http://xibo.org.uk/, January 2013.

[25] Risevision. Risevision - Digital Signage. http://www.risevision.com/, January 2013.

[26] Concerto. Concerto - Digital Signage. http://www.concerto-signage.org/, January 2013.

[27] AOpen. AOpen - Software. http://global.aopen.com/Platforms.aspx?id=97, January 2013.

[28] IAdea. IAdea Digital Plataforms. http://www.iadea.com/, January 2013.

[29] Sapo. Sapo Digital Signage Client. https://github.com/sapo/digital-signage-client, January 2013.

[30] TargetR. TargetR - Digital Signage. http://www.targetr.net/, January 2013.

[31] Novisign. Novisign - Android Digital Signage. http://www.novisign.com/android/android-digital-signage-app/, January 2013.

[32] OpenSplash. OpenSplash. http://www.opensplash.net/, March 2013.

[33] Signagelive. Samsung Smart Signage Platform. http://www.signagelive.com/smartlfd/, May 2013.

[34] Microsoft. Get ready for plug-in free browsing. http://msdn.microsoft.com/en-us/library/ie/hh968248(v=vs.85).aspx, February 2013.

[35] KDE. Greetings from the Safari team at Apple Computer. http://lists.kde.org/?m=104197092318639, January 2013.

[36] KHTML. So, when will KHTML merge all the WebCore changes? http://blogs.kde.org/node/1001, January 2013.

[37] KHTML. The bitter failure named "safari and khtml". http://blogs.kde.org/node/1002, January 2013.

[38] KHTML. WebCore - KHTML - Firefox: Know your facts! http://blogs.kde.org/node/1049, January 2013.

[39] KDE. Apple Opens WebKit CVS and Bug Database. http://dot.kde.org/2005/06/07/apple-opens-webkit-cvs-and-bug-database, January 2013.

[40] Mozilla. nglayout project - identity crisis. http://www-archive.mozilla.org/newlayout/gecko.html, January 2013.

[41] Opera. Opera 7.0 Beta 1 for Windows. http://www.opera.com/docs/changelogs/windows/700b1/, March 2013.

[42] Caching tutorial for web authors and web masters. http://www.mnot.net/cache_docs/, March 2013.

[43] BetterExplained. Optimize your site with HTTP caching. http://betterexplained.com/articles/how-to-optimize-your-site-with-http-caching/, March 2013.

[44] W3C. W3C HTML5 Offline web applications. http://www.w3.org/html/wg/drafts/html/master/browsers.html#offline, March 2013.

[45] HTML5Rocks. A Beginner's guide using Application Cache. http://www.html5rocks.com/en/tutorials/appcache/beginner/, March 2013.

[46] Offline Web Applications - Dive Into HTML5. http://diveintohtml5.info/offline.html, March 2013.

[47] Can i use... Can i use... Support for HTML5, CSS3, etc. http://caniuse.com/, March 2013.

[48] W3C. Web Storage. http://dev.w3.org/html5/webstorage/, March 2013.

[49] Sixrevision. Introduction to HTML5 Web Storage. http://sixrevisions.com/html/introduction-web-storage/, March 2013.

[50] IBM. Store images intelligently. https://www.ibm.com/developerworks/mydeveloperworks/blogs/bobleah/entry/html5_code_example_store_images_using_localstorage57?lang=en, March 2013.

[51] Web Storage Support Test. http://dev-test.nemikor.com/web-storage/support-test/, March 2013.

[52] W3C. Indexed Database API. http://www.w3.org/TR/IndexedDB/, March 2013.

[53] W3C. Web SQL Database. http://www.w3.org/TR/webdatabase/, March 2013.

[54] HTML5 Storage: IndexedDB. http://www.onlywebpro.com/2012/12/23/html5-storage-indexeddb/, March 2013.

[55] W3C. File API: Directories and System. http://www.w3.org/TR/file-system-api/, March 2013.

[56] W3C. Quota Management API. http://www.w3.org/TR/quota-api/, March 2013.

[57] Google Developers. Managing HTML5 Offline Storage. https://developers.google.com/chrome/whitepapers/storage, March 2013.

[58] WebPlatform compatibility. http://docs.webplatform.org/wiki/apis/quota_management/StorageQuota, March 2013.

[59] HTML5Rocks. How browsers work: Behind the scenes of a modern browser. http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/, April 2013.

[60] How browsers work and architecture. http://www.vineetgupta.com/2010/11/how-browsers-work-part-1-architecture/, April 2013.

[61] IE8 and Loosely-Coupled IE (LCIE). http://blogs.msdn.com/b/ie/archive/2008/03/11/ie8-and-loosely-coupled-ie-lcie.aspx, April 2013.

[62] TechCrunch. Why Mozilla matters and won't switch to WebKit. http://techcrunch.com/2013/02/15/why-mozilla-matters-and-wont-switch-to-webkit/, April 2013.

[63] The Chromium projects. Multi-process architecture. http://dev.chromium.org/developers/design-documents/multi-process-architecture, April 2013.

[64] Opera Developer News. 300 million users and move to WebKit. http://my.opera.com/ODIN/blog/300-million-users-and-move-to-webkit, April 2013.

[65] Nicholas C. Zakas. *High Performance JavaScript*. O'Reilly Media, 1 edition, 2010.

[66] W3C. Web Workers. `http://dev.w3.org/html5/workers/`, April 2013.

[67] HTML5 Rocks. The basics of Web Workers. `http://www.html5rocks.com/en/tutorials/workers/basics/`, April 2013.

[68] HTML5 Rocks. Transferable Objects: Lightning fast! `http://updates.html5rocks.com/2011/12/Transferable-Objects-Lightning-Fast`, April 2013.

[69] HTML5 Rocks. An introduction to Content Security Policy. `http://www.html5rocks.com/en/tutorials/security/content-security-policy/`, March 2013.

[70] W3C. The iframe element. `http://www.whatwg.org/specs/web-apps/current-work/multipage/the-iframe-element.html#attr-iframe-sandbox`, March 2013.

[71] HTML5 Rocks. Play safely in sandboxed IFrames. `http://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/`, March 2013.

[72] Google Chrome. Google Chrome Extensions. `http://developer.chrome.com/extensions/`, April 2013.

[73] Google Chrome. NPAPI Plugins. `http://developer.chrome.com/extensions/npapi.html`, April 2013.

[74] FireBreath. `http://www.firebreath.org/`, April 2013.

[75] Google Chrome. Google Chrome Apps. `http://developer.chrome.com/apps/`, May 2013.

[76] Google Chrome. Webview tag API. `http://developer.chrome.com/apps/webview_tag.html`, May 2013.

[77] Awesomium. http://awesomium.com/, May 2013.

[78] W3C. SMIL. http://www.w3.org/TR/SMIL/, November 2012.

[79] IETF.org. A Universally Unique IDentifier (UUID) URN Namespace. http://www.ietf.org/rfc/rfc4122.txt, June 2013.

[80] Wikipedia. Universally unique identifier. http://en.wikipedia.org/wiki/Universally_unique_identifier, June 2013.

[81] Generate RFC-compliant UUIDs in JavaScript. https://github.com/broofa/node-uuid, June 2013.