Building timeline-based web experiences: Mozilla Popcorn.js

Alice Schut
Media Technology
MSc student
Leiden University
aliceschut@live.nl

Bernd Dudzik
Media Technology
MSc student
Leiden University
bernd_dudzik@hotmail.com

Andrés Pardo Rodríguez
Media Technology
MSc student
Leiden University
apardor@gmail.com

ABSTRACT
This report will give an overview of the context surrounding the development of popcorn.js, by focusing on the transformation in usage of time-based media in the web over the past decade. Furthermore, an in-depth description of the technology behind popcorn.js' working is given, together with an account of its strengths and weaknesses, and examples of intended and unintended applications.

1. PURPOSE, CONTEXT AND HISTORY

1.1. Purpose

Traditionally time based-media such as audio and video were not natively supported by web browsers, but needed third-party software to be rendered properly. These plugins thus enhanced the inherent abilities of the browser by enabling it to handle unknown content types. However, the fact that each one of them was entirely in control of the embedded media content led them to being black boxes on a web page: The media content inside was only accessible by the plugin itself. This prevents the establishment of connections with anything beyond the plugin-scope and puts the content outside the reach of other parts of the web through using technologies like JavaScript or CSS.

Popcorn.js is a JavaScript library developed by the Mozilla Foundation that combines several open source technologies to bridge this gap and allow for interactions between time-based media content and the remainder of the web. By using it, videos and audio-files can trigger changes in other web content: Reaching a certain point in a timeline can cause the layout of its embedding website to change, or content to be fetched from a related Twitter stream or a Wikipedia article.

1.2. Context

Sharing and linking information has been the World Wide Web's primary purpose since its beginning. In it's early days text was the main type of available information in the web, while rich media content was very scarce. Additionally, the flow of content production was unidirectional: Users were retrieving and reading text documents, but the technology was still not used as a collaboration tool. Web inventor Tim Berners-Lee described practice of its usage at the end of the 1990s as rather a "publication medium but less of a collaboration medium" [2].

However, with the advent of the Web 2.0, the way information was shared had been drastically changed. New technologies and design strategies for the web enhanced the collaboration possibilities within the medium and enriched the user's experience by allowing better manipulation of content and nourished an environment for user participation [19]. Technological advances in connection speed and computation performance, enabled people to actively participate in media production and led to the rise of online video-platforms – YouTube being the most popular one. Video-Sharing services have spread over the web at such a fast rate that it has lead to an explosion of online video content [6] . For this reason videos will be soon the main web content [6].

The popcorn.js library can be seen as a result of these developments. It allows users of the Web and software developers to harness different types of time-based media in a dynamic fashion and to create tools that enable further usage by others – potentially to create their own tools and products. These products will not be static, but dynamic, as the Web 2.0 itself.

Another important backdrop against which the relevance of popcorn.js has to be seen lies in existing technologies for handling time-based media on the web. The functionality that it offers is not something that has not been previously possible to developers using proprietary technologies and substantial programming knowledge: E.g. Adobe's Flash environment provided an interface to JavaScript with the introduction of its ActionScript 3.0, thus allowing for communication between content inside the plugin-scope and outside of it [5].

The crucial difference is that it makes this functionality accessible to everybody free of charge: Popcorn.js was published as open software under the MIT License and is based entirely on open standards and technologie, such as HTML5 and JavaScript.

1.3. History

Some representative video ´s milestones in the web are listed bellow:

- 1990: Tim Berners-Lee developed the first web server and launched the world wide web.
- June 24th, 1993: First live streaming concert by Severe Tire Damage [17]
- 1995: RealNetworks broadcasted a baseball game (Yankees V.S. Seattle Mariners).
- 1996: Microsoft ActiveMode media player was released.
- 1997: Realplayer streaming application was released.
- 1999: Apple quicktime 4 was released.
- 2002: Adobe Flash became the main streaming technology.
- 2005: first youtube video [9].

1.4. VIDOS: A Predecessor of Video's Manipulation Through the Web.

In the late 90's and early 2000's video in the web was starting to spread. Online systems like VIDOS provided tools to edit videos in the web. VIDOS was a java-based server-client system that enabled users to cut, edit and download videos online. The main purpose of this application was to provide a free editing service, instead of forcing users to buy expensive editing software. The developers of this online tool were convinced, at that time, that this was the right path to follow because the network's bandwidth was improving exponentially, so videos could be handled and manipulated efficiently over the web. This, at the end, enabled the video format to become popular in web communication, so the developers of VIDOS pursued to create a tool that allowed users to get the most out of it: "as part of the trend from desktop-based to network-based solutions, the VIDOS system enables an individual Internet user anywhere in the world,(...) to undertake sophisticated spatial and temporal video editing" [3]. Nevertheless, this type of online editing application did not really work, as desktop-based editing software became cheaper and available within the computers' operating systems (as with iMovie and Movie Maker).

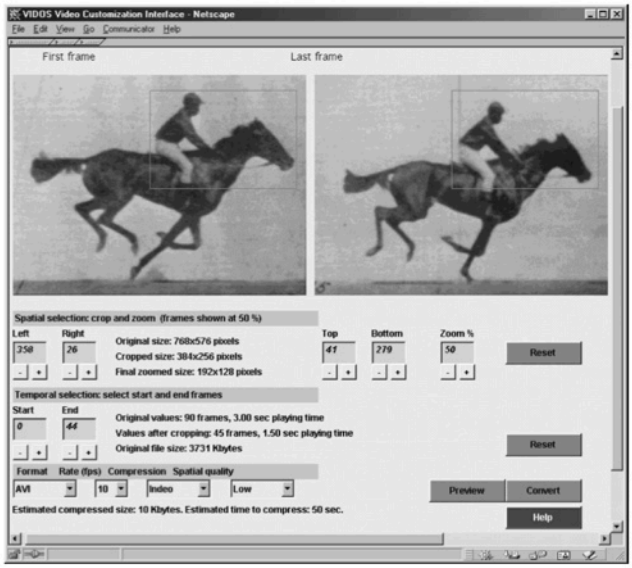

Figure 1. VIDOS view.

The difference between VIDOS and Popcorn.js is that the first is a web-based video editor, while the latter is a video-based web editor [16]. Furthermore, VIDOS was a tool only for editing, it didn't explore the features that videos had in the web, while popcorn.js focuses in the interaction between time-based media content and the remainder of the web (as it was explained before). VIDOS was one of the first approaches for developing a tool that helped to manipulate and handle videos in the web, but it was focused solely for editing purposes. With popcorn.js editing is not the goal; enriching the video's content with external information is.

2. OPERATING PRINCIPLES

The following section will provide an overview over the key technologies involved in the working of popcorn.js. Additionally, an in-depth view of the structure of the API and plugin-architecture is given, together with a detailed description of each components function.

2.1. Key technologies

As described above, popcorn.js main purpose is providing connectivity between time-based media content and and other web content. At its core, *Popcorn.js* draws heavily on introductions to HTML that have been introduced in the fifth version of the standard for the markup language.This version makes it more easy to handle and include multimedia and graphic content to the webpage. Through <video> and <audio> tags media-related features can be accessed, that are supported by the underlying APIs that are an integral part of HTML5.

As mentioned above, before these new features were introduced, developers had to call upon third party plugins that were capable of rendering audio and video content. The introduction of these features can be considered a big leap forward to a standardization for the handling of time-based media content across browsers and devices. Through the extension of the DOM (Document Object Model) in HTML5 by the HTML Media Element (and its children HTML Video Element and HTML Audio Element) specific media related properties have become accessible and controllable. Popcorn.js can be understood as wrapper around these media-related additions to the HTML standard and without them, would not have been possible: It turns

media into interactive JavaScript objects, that can trigger and listen for events.

Popcorn.js is not the only project attempting build a library around the HTML Media Element or trying to provide its functionality to older browsers by building transitional bridges. Examples include the video.js project [18] and the audio.js project [1]. However, popcorn includes a plugin-architecture to provide the same interface used to wrap the functionality of the HTML Media Element, also for other kind of players (e.g. YouTube, read more information below). This creates possibilities for seamless connectivity, independent of the precise media type actually in use.

2.2. API Structure: Core and Plugin-Architecture

Popcorn.js can be structured in two broad parts: A *core* and a collection of *add-ons*. The core deals with with wrapping the functionality of the *HtmlMediaElement's* methods, properties and events into a consistent API. The add-on part of the library consists of various plugins and methods that allow the establishment of connections to other web services or to media formats. In the following a detailed description of the various parts of the API and their functionality will be given, together with some examples of their usage.

2.2.1. Core

- Media Methods: These provide control over the media element as well as access to the current status of the media object in question, such as query and control the volume of the current popcorn instance. Especially important are mechanisms for playback control and a method for binding event handling callbacks to events (see below).

- Media Properties: Grants access and manipulation possibilities to a small set of properties of an active popcorn instance. It includes the frame rate as well as as a reference to an instance of the currently loaded medium. It is important to not that properties in this context does not refer to the actual media properties (access for these is handled in via the Media Methods), but merely properties of the popcorn.js instance in question.

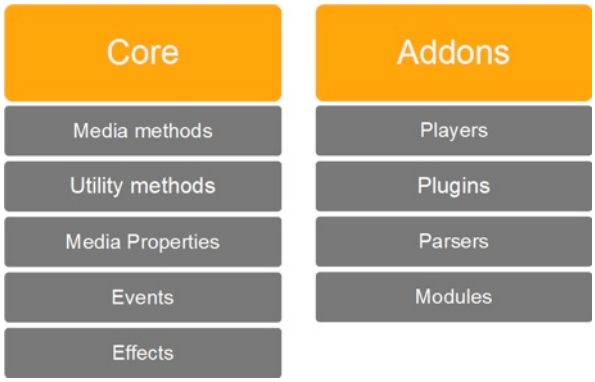

Figure 2: Popcorn.js API structure based on official reference [13]

- Events: Can be triggered from anywhere inside the code at any time and all listener functions associated with it will be executed. Encompasses the standard HTML5 media events [14], although not all of them may be implemented for all of the different player versions (e.g. YouTube). The event system makes it easy in Popcorn.js to trigger actions at specific instances during media playback, e.g. to perform an action whenever the video is paused, one would simply have to add a listener for the pause event and then provide a callback function that performs the desired action.

- Utility Methods: Provide non-essential helper functions created for easing the process of working with the library itself. Examples include a specific for-each iterator and an function to merge-objects with each other – Cases that may be neglectable for the majority of users, but might proof beneficial when dealing with specific situations.

- Effects: This part of the core functionality allows for CSS classes to be dynamically assigned to elements and brought about through functions that are executed prior to track events, thus allowing to signal changes to the end-user.

2.2.2. Add-ons

- Players: One of popcorn's most noteworthy features is its ability to deal with ample type of media players outside of the native HTML5 one, while being able to use the same syntax and function calls for each of them. This abstraction allows developer to use media hosted on platforms like YouTube or Vimeo, without having to possess extensive knowledge of each of the web media platform's API. A well documented plugin API allows for users to contribute to the existing set of players.

- Base player: Emulates the HTMLVideoElement to use popcorn without an video or audio element, thus providing a simple timeline functionality for projects. Furthermore, it acts as a base-class from which all other custom players inherit their properties.

- Web media: Custom players for media hosted on the web. At the time of writing these include Youtube, Vimeo and Sound cloud.

- *Smart player:* Allows for the usage of any existing player without having to specify which player, e.g. to

create a chain of HTML5 video together with YouTube or Vimeo clips

- Plugins: In order to allow for interaction between a popcorn program and content from supported web services, it possesses a plugin-architecture for users to develop additional functionality. The content can then easily accessed and displayed depending on events, e.g. to show a specific Google Map at a certain point in the media timeline. A list of all currently available plugins can be found in the official popcorn documentation [15].

- Parsers: A specific set of parsers allow popcorn projects to react on stored data. The parsing produces a timeline of events to potentially trigger actions in the script. Supported formats include specific subtitle formats like SBV or SSA, but also general data formats like JSON and XML.

- Modules: This category is interesting, because it highlights the dual goal of popcorn's endeavor to both provide a library that mades HTML5 media more easily accessible and to enhance the connectivity of time-based media with the web. While the former is addressed in the core functionalities of the library, the add-ons enhance the latter. Modules, on the other hand are neither of both, because they provide additional core functionalities. These encompass the possibility to create a seamless chain of different media to be played back, and even add a popcorn specific HTML attribute to mark data for popcorn's parsers.

3. STRENGTHS AND WEAKNESSES

Popcorns strict usage of open-standards and technology leads to it being easily accessible for a broad community of users: No expensive software has to be bought, or money to be invested in learning material. Because of its community driven development, new features are added in fast development cycles and errors can be directly reported and addressed on an open developer platform [12]. Popcorn's plugin architecture also leverages the power of the community for usage and development: It makes it easy for lay-programmers to connect their projects to important web services or even develop an own plugin that provides such a functionality.

One strong side of popcorns support of HTML media is the possibility of improved performance, due to not having to run a second application (a plugin) while playing.

However, since *popcorn.js* draws so heavily on HTML5 and its native media-handling, and it also inherits some its problems. Not all web browsers currently in use support the HTML5 <video> and <audio> elements fully – especially Microsoft's *Internet Explorer 8.0* turns out to be problematic and does not handle these tags properly [4]. Additionally, the situation is complicated due to the differences in video container format and codec support that different browsers provide for HTML media: While there is no limitation in principle on either, there is no single combination that is fully supported by all major browsers. This lead to the establishment of the good practice of having to store video-files in multiple formats to ensure cross-browser support (usually in the triplet of *MP4*, *Ogg* and *WebM*, also see the introduction in *6.* below for an example).

Another problematic aspect lies in popcorns struggle to make using different kind of media as simple as possible by creating a unified API layer. However, there are still properties of individual third-party APIs that cannot be bridged. Certain function calls and media events cannot be transferred as of the point of writing. As an example: Some properties of the YouTube player cannot be handled by function calls from popcorn.js, but have to be passed directly when accessing the media in a query string of the URL. This makes working seamlessly with different media types harder.

Furthermore, parts of the API documentation is still in development. This results in sometmes confusing terms or references to (as of now) undocumented features, thereby making it harder for novices to learn how to properly work with popcorn.

4. INTENDED APPLICATIONS

4.1. Web-based editing tools

The best known application made with Popcorn.js is Mozilla's own Popcorn Maker application: An online authoring and editing tool, allowing users to easily enhance, remix and share web videos, without the requirement to possess any sort of programming skills. It is part of the Mozilla Foundations *Webmaker* project [10] – an initiative with the aim to empower people to producing content themselves.

Figure 3: PopcornMaker

4.2. Non-linear narratives

The interactive documentary *One Millionth Tower* by Katerina Cizek [11] demonstrates the usage of *popcorn.js* to create non-linear narratives . In concert with other web technologies, such as WebGL, popcorn is responsible for the timing of camera motions and animations in this documentation about a highrise in the vicinity of Toronto. Additionally, it enhances the project with live data from social media platform or web services like Google Maps. It allows for changes in both direction to happen: Points in the video timeline are able to influence content of the surrounding website, while at the same time user interaction and data impact the content of the documentary.

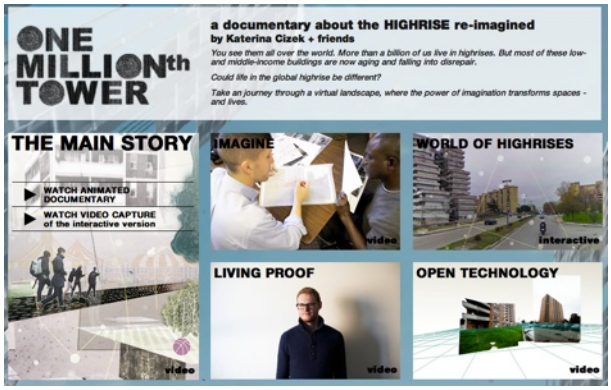

Figure 4: One Millionth Tower

4.3. Hyperaudio

Additionally to video, popcorn.js also has been used to enhance the connectivity of audio to the rest of the web. Developer Mark Boas created several projects that deal with the connection of transcribed speech to other web-based media or to trigger annotations based on timeline-events.An good example provides an interactive transcript of an US presidential debate in 2012 between then candidates Romney and Obama for the web presence of Al-Jazeera, in which usage of keywords such as “Jobs” are counted and displayed per candidate [8]. Additionally, popcorn is used to link an interactive transcript directly to the according part of the source media. Broes describes the potential impact of these new possibilities as ”*Hyperaudio is to audio as Hypertext is to text*” [21].

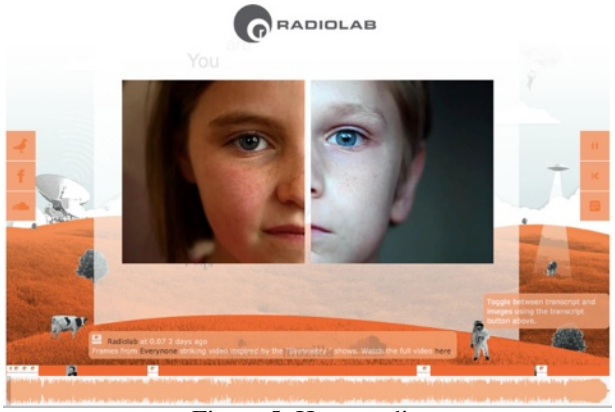

Figure 5: Hyperaudio

5. UNINTENDED APPLICATIONS

Discussing unintended applications for Popcorn.js can be tricky, as apparently there is no unintended application for it. Rather, is it possible to argue that Popcorn.js allows different possibilities for managing video content, by adding layers of information from the web’s content. Nevertheless, it is for this reason that is important to discuss if Popcorn.js really gives a breakthrough for video handling in the web (which will be discussed within the FINAL THOUGHTS of this paper).

6. GETTING STARTED

The following section is a brief guide into how to build a simple popcorn programm, dynamically displaying a footnote at a certain moment in the timeline of a YouTube video. Since popcorn draws on commonly spread webtechnologies, there are hardly any special prerequisites have to be fulfilled for this project. However, as mentioned above, the library draws heavily on HTML5 media, thus the browser used todisplay the product should support HTML5 (or have the Adobe Flash Player plugin installed as a fall back solution). Some basic knowledge of HTML and JavaScript are assumed in the description and setting up of the example.

The first step to use the popcorn.js library in a web project is to create a minimal HTML5 document and to include the library in it. This can either be done through downloading it or through referencing it direcly on the official website. The code below will do the latter (This URL always refers tothe latest stable production version of the library) :

```
<html>
  <head>
    <script src="http://popcornjs.org/code/dist/popcorn-complete.js"></script>
  </head>
  <body>
  </body>
</html>
```

The next step is to extend the markup in the body of the document by two -element with the IDs *videoContainer* and *footnoteContainer* (additions are marked orange below):

```
<html>
  <head>
    <script src="http://popcornjs.org/code/dist/popcorn-complete.js"></script>
  </head>
<body>
        <div id="videoContainer"></div>
        <div id="footnoteContainer"></div>
 </body>
</html>
```

Now it is time to use the popcorn library itself by adding some JavaScript to the document (additions are marked orange below). The code added below will wait till the DOM is ready, and then create an instance of Popcorn. In the constructor two arguments are passed: The container in which the video shold be added in the document (in our case the -element *videoContainer)* and the URL of the video hosted on YouTube. The URL below is purely fictional and you will have to replace it with a real one for the example to actually work. Finally, the function *pop.footnote()* adds a footnote to the popcorn instance, together with some informations on when in the timeline it should appear and disappear, what is to be displayed and in which element. For the example this would give us a footnote appearing at second 2, disappearing at second 15 and displaying "Hello World!" in the element labeled *footnoteContainer*.

```
<html>
<head>
  <script src="http://popcornjs.org/code/dist/popcorn-complete.js"></script>
<script>
   document.addEventListener("DOMContentLoaded", fucntion(){
      var pop = Popcorn.youtube( "#videoContainer", "http://www.youtube.com/watch?v=videoID");
     pop.footnote({
        start:2,
        end: 15,
        text:"Hello World!",
        target: "footnoteContainer"
        });
     pop.play();
   },false);
  </head>
  <body>
      <div id="videoContainer"></div>
      <div id="footnoteContainer"></div>
  </body>
</html>
```

7. FINAL THOUGHTS

7.1. Hypermedia Aesthetics

After taking a broad look into Popcorn.js a question raises: Does it really takes out videos from being black boxes in web pages? Several critics and remarks against how Popcorn.js displays and retrieves the added layers of information explain that it drives the attention away from the content of the video, as additional content pops in its sides [20]. The way video is managed is not really a breakthrough. Yes, it allows to add content and connect different sources of information with the moving image, but video is still just played in a window. Gabriel Shalom explains in his Hypercubist Manifesto that video is digital, but that there is still an analog approach towards it based on frame, flat and linear assumptions [7]. The real potential of the video in the web is still unexplored according to these ideas.

The first approaches towards video with the Popcorn.js reminds the way video and text information are connected within a news broadcast. Usually the anchorman tells the story while a small picture that refers to the news is displayed; then, while the video of the news is running, additional information is added via text notes. Both type of sources are presented at the same time, but they still act independent of each other (the additional text-based information is outside the video's frame). Marshall McLuhan once said that "each new medium looks to its predecessors for its content" [7], so it is possible to suggest that the linking of video and other layers of information in the web is still seen as if they belong to television. This is why Gabriel Shalom argues that video games take moving images one step further than film, as they can go back and return through the frames, and as they don't follow a linear time line [7].

One approach that tries to exploit the digital qualities of videos in the web is "The Flight of the Navigator", developed by the Mozilla Audio Team. This seeks to bring the web's information into the video's frame, rather than present it within a box as it has been done

Figure 6: Flight of the Navigator view

before [20]. Its goal is to integrate the web and the video by exploring new approaches for linking content, breaking the linear sequence of videos and thinking about video through novel ideas of space and time. Popcorn.js is a very important tool that started breaking the black box that locked video inside the web. It gave the chance to start combining different layers of information, but the new challenge is to merge these layers.

REFERENCES

1. Audio.js . http://kolber.github.io/audiojs/

2. Berners-Lee, T. Weaving the Web. Orion Business Books, London, Great Britain, 1999. p.62

3. Boudier, T., Shotton, D. M. VIDOS, A System for Video Editing and Format Conversion Over the Internet. Computer Networks 34 (2000). p. 931-944.

4. Building HTML5 Applications. Practical Cross-Browser HTML5 Audio and Video. http://msdn.microsoft.com/en-us/magazine/hh781023.aspx

5. External Interface - AS3. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/external/ExternalInterface.html

6. Grassi, M., Morbidoni, C., Nucci, M. A Collaborative Video Annotation Systema Based on Semantic Web Technologies. Springer Science+Business Media, LLC (2012).

7. Hypercubism Manifesto. http://vimeo.com/14604303

8. Interactive Video Transcript of Denver Debate. http://www.aljazeera.com/indepth/interactive/2012/10/20121049528478583.html?s=15705)

9. Me at the Zoo. First Youtube Video Ever Posted. http://www.youtube.com/watch?v=jNQXAC9IVRw\

10. Mozilla PopcornMaker. https://popcorn.webmaker.org

11. One Millionth Tower. http://highrise.nfb.ca/onemillionthtower/

12.Popcorn.js Developer Platform. https://webmademovies.lighthouseapp.com/projects/63272-popcornjs/overview

13. Popcorn.js Documentation. http://popcornjs.org/popcorn-docs/index.html)

14. Popcorn.js Events Documentation. http://popcornjs.org/popcorn-docs/events

15. Popcorn.js Plugins. http://popcornjs.org/popcorn-docs/plugins/

16. Richter, B. Moskowitz, B. Mozilla Popcorn: Web Video Interaction Using Client-Side Javascript. ACM SIGMM Records. Vol. 4, No. 1 (March 2012).

17. Severe Tire Damage House Band of the Internet. http://www.std.org/text/filler.html

18. Video.js HTML 5 Video Player. http://www.videojs.com

19. Web 2.0 Compact Definition: Trying Again. Tim O'Reilly. http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html

20. Web Made Movies. An Aesthetic for Web Made Movies. http://videos.mozilla.org/serv/webmademovies/justintime.ogv

21. What is Hyperaudio? http://blog.appsfuel.com/2012/08/30/what-is-hyperaudio