Audio oriented UI components for the web platform

Victor Saiz, Benjamin Matuszewski, Samuel Goldszmidt

Audio oriented UI components for the web platform

Victor Saiz, Benjamin Matuszewski, Samuel Goldszmidt
IRCAM – Centre Pompidou, STMS lab IRCAM-CNRS-UPMC
1, Place Igor Stravinsky
75004 Paris
{firstname.lastname}@ircam.fr

ABSTRACT

This paper presents a set of web-native tools for visualising and interacting with time-based objects. These visualisations are rendered as part of the document using web standard technologies, allowing for an easy integration and interaction with the elements on the same document without the help of non-native technologies such as Adobe Flash, Microsoft's Silverlight or Oracle's Java.

Categories and Subject Descriptors

[**Information systems**]: Multimedia content creation, Web interfaces, Browsers; [**Human-centred computing**]: Human computer interaction (HCI), Hypertext / hypermedia, Graphical user interfaces, Web-based interaction, User interface tool-kits, Systems and tools for interaction design, Visualisation systems and tools; [**Applied computing**]: Hypertext languages; [**Software and its engineering**]: Open source

General Terms

Documentation, Performance, Design, Experimentation, Human Factors, Standardization, Languages, User experience.

Keywords

HTML5, Open Web Standards, ECMAScript, Interaction, Visualisation, Graphical User Interface, Web Components, Web Audio API

1. INTRODUCTION

Traditionally in the web platform there was a collection of UI elements available through the standard HTML specification `<input | select | button...>`. With the arrival of the HTML5 [24] specification, the collection has been expanded considerably with new UI elements such as `<color | date | range | datalist...>` and form validation, but none of them lives up to the potential of some of their new media APIs yet, from knobs to waveform visualisers, spectrograms, breakpoint functions, timelines etc. These new media elements presents new interaction and editing challenges specially for media streams like audio buffers.

After widespread adoption (2008 to 2013 [14]), these new APIs are made available in millions of browsers. That, along the improvements in Javascript runtime engine's brings faster performance, new multimedia capabilities, wider access to different types of data formats (webVTT [27], JSON [6]) and sources (FileSystem API) to the browser. This particular context makes a new range of multimedia applications possible for the first time without the use of non-native plugins today.

> "At some point recently, the browser transformed from being an amazing interactive document viewer into being the world's most advanced, widely-distributed application runtime." [7]

In this paper we present a project where we are set to explore these new capabilities in order to develop richer means of representation and interaction for audiovisual and time based objects.

2. GOALS AND REQUIREMENTS

Our goal is to release a collection of UI elements for audio streams and time based objects.

For content editors we plan to release ready to use visualisers packaged as Web Components [26]. Whereas for Javascript developers we will be shipping the same components with full Javascript APIs. Finally, on the lowest level, visualisation developers wanting to tap into the deepest core of the components will also have access to these small pieces of functionalities and use, compose and combine them as needed into more complex visualisations.

Given the open source nature of the project [12], our APIs and internal architecture need to be easy to use and understand, aiming for modularity and emphasizing on reusability, compatibility and independence.

WAC '15, Paris, France

3. STATE OF THE ART

The research in this project started out as part of an earlier IRCAM project named "Écoutes signées" driven by the APM team [10]. The project aimed to explain the listening process, with an emphasis in active listening, and find ways for non-specialist audiences to experience music the way composers, performers, musicians or musicologists do. [8]

Most of the components we will provide have been part of the desktop platform's repertoire in various forms for a long time, audio editors, workstations etc. Among them, Max/-Mubu [22] and Variations Audio Timeliner [19] were the inspiration for some of our use cases.

There are also several GUI libraries based on the Web Audio API [25] for the standard-web platform. These are significant projects developed by members of the community such as Prong [1], Kievii [2], WAAX/mui [5], Nexusosc [18], Interface.js [21], Peak.js [20], wave surfer [15], Timeline.js [9] etc. While some of them have similar goals to ours, most of them just touch upon different UI problems, and the ones that do are just not conceived nor designed to be used outside their particular applications.

Lastly, the excellent work in the browser for non web-standard solutions such as Java applets, Flash applications (e.g. the work of Michelle and Ebert for the Flash platform [17] [16] has been especially important for the audio community in the web) dealt with the same issues we face today in terms of interaction design.

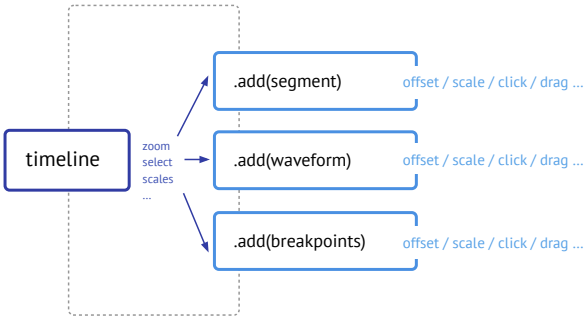
4. THE ARCHITECTURE

At its core the library sits on top of the visualization library D3. [4]

Among the reasons why to use D3 as the base for our visualisation tools is the fact that it already is one of the most widespread visualisation libraries, backed by a big and active community of developers. The library also includes an extensive set of commonly needed utilities for visualisation tasks such as scales, axes, data mining and data manipulation functions. At its core it is tightly coupled to the DOM, which not only aligns perfectly with our commitment to being W3C standard compliant, but also it allows for a seamless interaction between our component's SVG nodes and the rest of the elements on the document. Last but not least, an important factor for this choice is it's modular approach that gives us the possibility to export subsets of the aforementioned strictly based on every particular need.

Since our potential users – the Javascript developers – are most likely familiar with already existing UI libraries and because we are making extensive use of D3, one important decision regarding the general architecture and our APIs is to adhere to D3's community guidelines and frequent patterns [3] and its functional coding style. By doing so, we deliver fully compatible tools that can integrate with other tools developed by D3's community.

Figure 1: Architecture overview



4.1 The Timeline

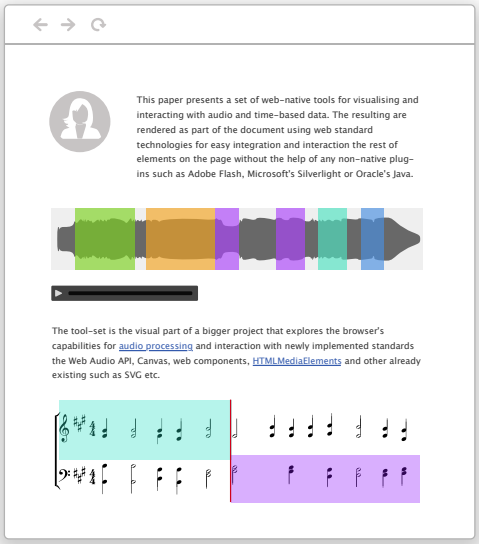
When visualising timed objects it is very important to ensure that the time axes of each and every visualisation are kept consistently, even more so when we are to zoom in and edit the visualized data. For that reason, all the architecture evolves around the `Timeline` class, which not only is responsible for time axis consistency, but also formalises the means by which every visualiser is edited, positioned and zoomed in. In order to work with components, the `Timeline` provides a plugin-like API that allows the user to add and remove component instances extending the class `Layer` (see figure 2 lines 8 and 9) at any time. Layers inside the `Timeline` are then called upon their different life cycle events (`initialisation`, `data-binding`, `update` and `draw`) by the `Timeline`. Similarly, the `Timeline` has basic layout capabilities that permits the vertical positioning of its layers (via the layer's `top` parameter as seen in figure 4 line 6), while locking their horizontal position, size and scale and hence keeping the time data in sync.

Figure 2: Instantiating a timeline and adding layers to it

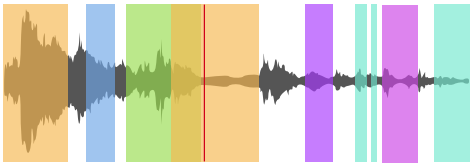
```
1   // Timeline instance
2   var graph = timeline()
3         .width(800)
4         .height(150)
5         .xDomain([0, 100]);
6   // add our layers to the timeline
7   graph
8     .add(segmentLayer)
9     .add(waveformLayer);
10  // d3 call to draw our timeline
11  d3.select('.timeline')
12    .call(graph.draw);
13
14  ...
```

With all of this in place, this architecture allows for component composability, shared functionality and time consistency.

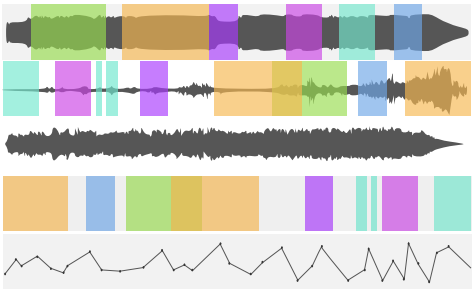
Figure 3: Different layer and timeline possibilities



a) Multi-media document.



b) Waveform with segments and cursor.



c) Multi-track environment.

As shown in figure 3.a, separate `Timeline` objects can be instantiated across a document containing one or many layers, empowering the user to build richer layouts where the visualisations can coexist around other multimedia content. Similarly in figure 3.b we can see how with layers we can create complex timelines, by combining and overlapping them with other layers while keeping the time axis's aligned. Finally as seen in figure 3.c, when not used to make complex elements, layers can be easily laid down vertically, creating multi-track-like environments etc.

4.2 The components

Components are in charge of their own rendering and editing capabilities. Every component extends the `Layer` class. This class abstracts the common methods (`params, data, etc.`) and life-cycle hooks (`load, setScales, delegateEvents, etc.`) for every component, leaving to the component the implementation or extension of it's specific ones (`update, draw, xZoom, handleDrag, etc.`). As mentioned earlier, by extending the `Layer` class it becomes rather simple to implement our own visualisations and incorporate them into a `Timeline` which will render them in place and keeps us away from inconsistencies. Each `Layer` is configured via it's `params` method, or it's singular equivalent `param` (see figure 4 lines 4 to 13). The parameters passed via this method are used to configure layer properties, where as specific data accessors have their own separate methods on every component (see figure 4 line 16). By default, the components are not editable, nor available for selection, but interactions can be defined through the `params` method using the `interactions` key (see figure 4 lines 7 to 10).

The components also provide an operation interface that allows to modify their elements state programmatically. With such commands one can edit the position and size of elements without any mouse interaction, reducing the library intelligence and leaving those decisions to the application developer. In turn the `Timeline` event system consumes such APIs to manipulate its components.

Figure 4: Segment initialization and configuration

```
1  // Segment component
2  var segmentLayer = segment()
3    // several params object
4    .params({
5      height: 200,
6      top: 10,
7      interactions: [
8        'editable',
9        'selectable'
10     ]
11   })
12   // one param
13   .param( 'color ', 'red ')
14   .data(model) // external data
15   // pass in the data accessor
16   .y((d,v=null) => {
17     if(v!==null) d.volume = v;
18     return d.volume;
19   });
```

4.2.1 *Delivered components*

The following components are already available in the library:

- Waveform component
- Segments component
- Breakpoint component (automation curves etc.)
- Label component
- Marker component

The majority of the components are rendered as `SVG groups`. Their interactions are registered with the `Timeline` (see 4.4). The requirements for a component are the following: They must allow for easy addition, modification and deletion of their items inside the `Timeline`. They should also be zoomable on the x-axis.

The waveform component on the other hand, which visualizes data sampled at constant rates such as audio files, is a special case in the collection, since the handling of the audio data comes with an added complexity given the size of such data. To deal with this, in the current implementation, the chosen strategy is to generate a sub-sampled snapshot of the audio data. Behind the scene, the layer defines the number of samples per pixels for the rendering, and according to that value it uses the sub-sampled snapshot or the raw data to generate the final data to display. However, rendering this amount of data via an `<svg:path>` can sometimes become very heavy for the `DOM`, in those cases, a more performant canvas rendering (via `<svg:foreignobject>`) alternative is also available. Figure 5 illustrates a basic instantiation and configuration of a waveform component.

Figure 5: Waveform instantiation and configuration

```
1  var arrayBuffer = audioBuffer
2        .getChannelData(0)
3        .buffer;
4
5  var waveformLayer = waveform()
6    // mandatory configuration
7    .data(arrayBuffer)
8    .duration(audioBuffer.duration)
9    .sampleRate(audioBuffer.sampleRate)
10   // optionnal configuration
11   .params({renderingStrategy: 'canvas'})
12   .color('#C0CFCF')
```

4.3 The data

Data uniformity is a wider problem than the scope of this library. While some of the visualisation libraries focus on data parsing and filtering (e.g. Miso.dataset [11]) our use cases go beyond fetching, formatting and filtering since we edit and share our data with external elements (in our particular case the audio components of the Waves library [13]). For that reason we want to be working directly with the original data structure. This is obviously limiting since the data needs to come in with a given structure, in our case the data is expected to be provided as an array of objects. To mitigate the limitations, every component has specific data accessors that provide read and write access to the data points as shown in (see figure 4 lines 16 to 19).

4.4 The Event system

We make use of the browser's event system so changes on the layer level are notified to components via the `Timeline` using `Event delegation` [23]. Attaching event listeners on every element in each layer is not efficient, instead the `Timeline` discriminates the origin of the interaction via browser's event-listeners to then dispatch series of unified custom events available to any element of the application via the widely used observer pattern.

5. CONCLUSION AND FUTURE WORK

From where we stand today, a lot of the requirements that seemed rather complex to achieve in a web browser have proven to be increasingly possible thanks to efforts of browser vendors and the Javascript community. As a result we have managed to deliver a library [13] that includes a fair amount of usable components and increase the range of possibilities available to web developers.

However, because in some cases we encountered the need for a more granular way of composing our different layers together, we already started outlining some improvements in the architecture. For instance, in a future architecture we could decouple the rendering responsibilities from our components, and separate our `timeline` in smaller objects. These refactorings could allow us to nest several `timelines` recursively and create "multi-component" layers. Regarding the performance of the library, many things can still be improved. One of them could be the implementation of a `throttling` mechanism [12] to ensure a more consistent and fine-grained event rate across browsers.

We are in the position to state nonetheless that the web platform today delivers a solid foundation to build complex and rich user interfaces on top of the the W3C standards specification. With new technologies such as Web Components [26] integration and distribution becomes easy and seamless.

6. REFERENCES

[1] L. Barlow. Prong. `https://github.com/lukebarlow/prong`.

[2] C. Belloni. Kievii. `https://github.com/janesconference/KievII/`, 2010.

[3] M. Bostock. Towards reusable charts. `http://bost.ocks.org/mike/chart/`.

[4] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17:2301–2309, 2011.

[5] H. Choi. Waax/mui. `https://github.com/hoch/waax`.

[6] D. Crockford. JSON: The fat - free alternative to XML. In *Intelligent Search on XML Data*, 2006.

[7] T. Dale. Progressive enhancement is dead. `http://tomdale.net/2013/09/`

progressive-enhancement-is-dead/.

[8] N. Donin. Towards organised listening: some aspects of the 'Signed Listening' project, Ircam. *Organised Sound*, 9, 2004.

[9] S. Goldszmidt. JavaScript library for audio/video timeline representation. In *WWW2012*, Lyon, France, Mar. 2012. notes : / copyright : None 2012 / fiche : None.

[10] S. Goldszmidt, N. Donin, and J. Theureau. Navigation génétique dans une œuvre musicale. In *Interaction Homme-Machine*, pages 159–166, 2007.

[11] T. Guardian. Miso project. http://misoproject.com/dataset/, 2012.

[12] Ircam. Ircam on github. https://github.com/ircam-rnd.

[13] Ircam. Waves library on github. https://github.com/Ircam-RnD/waves.

[14] P. IRISH and D. MANIAN. Html5 adoption. http://html5readiness.com/.

[15] Katspaugh. wavesurfer.js. https://github.com/katspaugh/wavesurfer.js.

[16] A. Michelle. André michelle.

[17] J. Michelle, André amd Ebert. Pop forge. https://code.google.com/p/popforge/, 2007.

[18] E. music & digital media of Louisiana state university. nexusosc. https://github.com/lsu-emdm/nexusUI.

[19] T. of Indiana University and I. U. R. . T. Corporation. Variations audio timeliner audio annotation and analysis tool. http://variations.sourceforge.net/vat/, 2002.

[20] B. R&D. Peak.js. https://github.com/bbcrd/peaks.js.

[21] C. Roberts. Interface.js. http://www.charlie-roberts.com/interface/index.html.

[22] N. Schnell, A. Röbel, D. Schwarz, G. Peeters, and R. Borghesi. Mubu & friends - assembling tools for content based real-time interactive audio processing in max/msp. In *Proceedings of the International Computer Music Conference*, Montreal, Canada, August 2009.

[23] W3C. Event delegation. http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-flow-bubbling.

[24] W3C. http://www.w3.org/tr/html5/. http://www.w3.org/TR/html5/.

[25] W3C. Web audio api. https://webaudio.github.io/web-audio-api/.

[26] W3C. Web components. https://w3c.github.io/webcomponents/.

[27] W3C. Webvtt. http://dev.w3.org/html5/webvtt/.