Data-independent sequencing with the Timing Object

A JavaScript Sequencer for single-device and multi-device Web media. *

Ingar M. Arntzen
Norut Northern Research Institute
Tromsø, Norway
ingar@norut.no

Njål T. Borch
Norut Northern Research Institute
Tromsø, Norway
njaal@norut.no

ABSTRACT

Media players and frameworks all depend on the ability to produce correctly timed audiovisual effects. More formally, sequencing is the process of translating timed data into correctly timed presentation. Though sequencing logic is a central part of all multimedia applications, it tends to be tightly integrated with specific media formats, authoring models, timing/control primitives and/or predefined UI elements. In this paper, we present the *Sequencer*, a generic sequencing tool cleanly separated from data, timing/control and UI. Data-independent sequencing implies broad utility as well as simple integration of different data types and delivery methods in multimedia applications. UI-independent sequencing simplifies integration of new data types into visual and interactive components. Integration with an external *timing object* [7] ensures that media components based on the Sequencer may trivially be synchronized and remote controlled, both in single-page media presentations as well as global, multi-device media applications [5, 6, 7, 16]. A JavaScript implementation for the Sequencer is provided based on *setTimeout*, ensuring precise timing and reduced energy consumption. The implementation is open sourced as part of *timingsrc* [2, 3], a new programming model for precisely timed Web applications. The timing object and the Sequencer are proposed for standardization by the W3C Multi-device Timing Community Group [20].

CCS Concepts

•Information systems → Multimedia content creation; •Software and its engineering → Organizing principles for web applications; Abstraction, modeling and modularity;

*This research is funded in part by MediaScape (EU, FP7-ICT-2013-10, grant agreement 610404).

MMSys'16, May 10-13, 2016, Klagenfurt, Austria

DOI: http://dx.doi.org/10.1145/2910017.2910614

Keywords

sequencing, web, timed data, timing object, timed visualization, multimedia, media synchronization, multi-device, distributed, intra-destination media synchronization, inter-destination media synchronization

1. INTRODUCTION

Multimedia frameworks are always built around the idea of organizing and playing back media relative to some timeline. For continuous media, audio and video frames are laid out back-to-back, usually filling the entire timeline. For discrete media, such as timed subtitles or timed comments, the distribution (along the timeline) may often be more non-uniform and possibly sparse. In this paper we focus exclusively on the sequencing of discretely timed media - or more generally - timed data. *Sequencing* here refers the process of translating *timed data* (playlist, track, script, log, time-series, score, etc.) into correctly timed execution (playback, visualization, operation etc.).

Timed Web applications usually achieve timed behavior either by adopting an existing multimedia framework, by using text track support of HTML5 media elements, or by building custom solutions in JavaScript. Unfortunately, these options all have limitations. Advanced frameworks tend to dictate media formats, authoring model, timing model, control primitives, and possibly UI. In addition, they do not always integrate well with other frameworks or media systems. Text tracks provide a simple, programmatic mechanism for sequencing, but dependence on the HTML5 media element is problematic, particularly if the presentation lacks an audio/video component, or aims to switch dynamically between multiple media elements. Custom sequencing solutions imply the overhead of reinventing common functionality, and often lead to shortcut solutions, bugs and limited reusability.

Instead, we advocate a programming model where timing and sequencing functionality are made available as independent, generic programming tools, applicable across application domains. The timing object [5, 7] is the fundamental building block of this programming model, defining a unifying foundation for timing, synchronization and control. This paper presents the Sequencer, a generic sequencing tool as an additional building block for timed media applications.

The main design goal has been to provide a Sequencer that is data-independent and UI-independent. Data-independence addresses the coexistence and integration of diverse data types, and UI-independence simplifies integration of new data types into visual and interactive components. With

timing and sequencing available as independent programming tools, there are no longer any restrictions with respect to delivery methods, data types and data sources. Sequencing tools may also be used to implement correctly timed data delivery (e.g. timed pre-fetching). Support for dynamic sequencing allows dynamic data sources to be used without introducing any additional complexity for the programmer. Finally, the timing object supports distributed timing through Shared Motion [5, 6, 16]. In short, this opens up multi-device sequencing to any connected Web agent, independent of data format, delivery mechanism or UI framework. Furthermore, this approach extends to any IP-connected device or process, provided only that timing and sequencing functions described in this paper have been made available in native code.

The scope of this paper is limited to sequencing in application level JavaScript, so precision beyond millisecond is out of scope. The Sequencer is also a quite simple concept, limited to sequencing of timed data defined by points and intervals on a timeline. More sophisticated sequencing tools may be built on top. A JavaScript Sequencer implementation is open sourced on GitHub [2, 3]. It is functionally correct, complete, precise, efficient, and ready for professional usage.

2. RELATED WORK

The concept of sequencing has a long history, including popular mechanical instruments such as music boxes and barrel organs, often driven manually by a crank. Based on this heritage, a variety of software sequencers have been developed, particularly within the MIDI [15] framework. Such sequencers tend to be full featured entities, encapsulating aspects of recording, representation, storage, playback and control. In this paper, the term sequencer is interpreted more narrowly;

> *sequencer*: logic that translates timed data into correctly timed execution.

2.1 Polling versus timeouts

Poll-based sequencers base their operation on repeated comparison between timing source and timed data. For instance, text tracks [24] are evaluated repeatedly as part of the main processing loop in HTML5 media elements [28]. MIDI sequencers are driven by periodic clock pulses. Similarly, in Flash, Silverlight or Popcorn.js [1, 14, 17], execution of animations and media cue points are commonly based on the timeupdate event from the media player, or other fixed frequency timers (renderEvent, requestAnimationFrame). Unfortunately, the precision of poll-based sequencers is always limited by the polling frequency (e.g. 4Hz in HTML5 media). Excessive polling may improve this, but at the cost of increased energy consumption, which, in the Web context may ultimately lead to battery drainage in mobile devices and lagging user interfaces. Instead, the Sequencer is driven by *setTimeout*, making it both precise and energy efficient, particularly for timed data sparsely distributed along the timeline.

2.2 Points and intervals

Basic timing mechanisms such as timed cue points in Flash and Silverlight essentially provide callbacks at specific moments in time, in reference to playback of a timing source. Such mechanisms typically work exclusively on points on the timeline, not intervals. For instance, a set-top-box may trigger a commercial on a secondary device, at the correct moment during playback of a TV show. With point-based sequencing the management of cue duration (if applicable) is left to the programmer. In contrast, Sequencers based on intervals distinguish between entering or leaving intervals. For example, both text tracks [24] and the Sequencer emit *enter* and *exit* events. Additionally, the Sequencer provides specific support for singular points, and maintains a set of currently *active* intervals.

2.3 Playback, time-shifting and control

Sequencing mechanisms also differ with regards to properties of their timing source. Many sequencing mechanisms are based on an internal timing source, i.e. the system clock or other monotonically increasing counters. For instance, WebAudio [29] schedules audio samples based on a monotonic clock defined by the audio subsystem. If the timing source supports temporal control, e.g. time-shifting or rate-changes, sequencing logic must be strengthened accordingly. The HTML5 media clock supports modification of currentTime, pause/play, as well as adjustments to playback rate, and the sequencing of text tracks should always be consistent. Interestingly, text track sequencing emits events only during playback, not after time-shifting while media playback is paused [24]. We regard this as a weakness in the standard as additional code must be written to handle this rather typical usage. The Sequencer has full support for the more precise and expressive timing model of the timing object. This includes unrestricted changes to position, velocity and even acceleration.

2.4 Playlist sequencing

Playlist-based sequencing is popular in linear presentations. A playlist is typically defined as an ordered list of objects with start time and duration. Objects may be organised back-to-back on the timeline, or gaps may be allowed. For instance, the BBC recently open sourced a video compositor [21] for dynamic playback of HTML5 videos (and more), as part of their object-based media initiative [11]. In principle, sequencing a track of timed video URL's is not very different from sequencing a track of timed subtitles. In practise though, HTML5 videos are not as lightweight as subtitles, and will require just-in-time prefetching and synchronization in order to produce a seamless viewing experience [8, 9, 13]. The Sequencer supports back-to-back intervals, gaps between, or overlapping intervals. It may also be used to ensure timed prefetching.

2.5 Relative timing

Sequencing in SMIL [27, 31] is based on a graph representation of timing relations between objects. For instance, the playback of sibling objects in the graph may be arranged sequentially (seq) or in parallel (par). However, at some point relative timing statements (between objects) must be compiled into absolute timing statements. This may occur during initialization, or dynamically during playback. For instance, in SMIL the concept of *indeterminate timing* allows the timing of an object to be specified by user input during playback, possibly affecting the timing of other objects in the graph.

Currently, the Sequencer has no particular support for

relative timing. Points and intervals are independent and defined with absolute timing references relative to the timing object. However, this only means that compilation from relative to absolute timing must be performed by the application programmer, as part of parsing and registering application-specific timed data. As the Sequencer supports modifications of points and intervals at any time, this compilation may occur at load or during playback in response to changes in the data model.

That said, improved support for data models with relative timing could be layered on top of the Sequencer, integrated into it, or there could be other types of Sequencers that provide specializations in this regard. At this point though we aim to keep the Sequencer concept as simple and generic as possible, emphasising its role as basic building block.

2.6 Time containers and looping

The SMIL authoring model also supports time containers and looping playback for groups of objects. These features are out of scope for the Sequencer itself, as the Sequencer and its timing object correspond to a single time container. However, similar effects may be achieved by exploiting the flexibility of the timingsrc programming model [2, 3] (of which the Sequencer is a central part). For example, different timing containers correspond to multiple Sequencer instances combined with different timing objects or timing converters (e.g. SkewConverter and LoopConverter).

2.7 Data formats

There is often a tight integration between data formats and sequencing logic. The text track mechanism of HTML5 works on JavaScript objects, but specifies predefined structure and types for cues. For instance, text tracks support the WebVTT format [30]. In MIDI, timed musical commands are stored in the *Standard MIDI File (SMF)* format. In Flash, the proprietary file format *ShockWave Flash (SWF)* maps graphical commands and media objects to frame numbers, durations and scene coordinates. MPEG-4 [18] adds support for synchronization and composition of multiple media streams, including discrete media such as graphical objects (2D and 3D). In particular, the MPEG-4 Systems part [19] defines an architecture for media clients (terminals) that integrates media formats, delivery methods, temporal and spacial composition, interaction and rendering. In SMIL, object references, timing intervals (start, end), timing relations (pre, seq), layout relations as well as interactivity are all tightly coupled in declarative XML. In particular, the tight coupling of timed data and layout is limiting. This issue is adressed by the Ambulant SMIL player [10, 22] which introduces abstract factory functions for data and rendering, thereby decoupling core player functionality from proprietary data formats and data sources. The SMIL State [12] extension deals with the same issue, simplifying the integration with custom data sources, through the concept of shared *variables*.

In contrast, the Sequencer uses unique keys to separate sequencing logic entirely from data model and delivery methods. Application logic extracts intervals and keys from the data model, and reacts to correctly timed event callbacks from the Sequencer. This way, timed data from different data sources may be rendered together, without introducing any restrictions on data formats or delivery methods.

2.8 UI integration

Sequencing logic also tends to be integrated with predefined UI elements. For instance, most Web players, from simple slide show viewers to full fledged multimedia players, define a single rectangular screen area for visual presentation and built-in controls. In contrast, the Sequencer makes a point of not providing any predefined UI bindings. It provides *enter* and *exit* events, and leaves it entirely to application code to implement appropriate effects in the DOM. This ensures that the rich UI capabilities of the Web platform can be fully exploited for timed applications. In short, timing objects and Sequencers reduce the challenge of *timed* Web programming to the problem of *regular* Web programming. UI elements for control and progression bind to the timing object, not the Sequencer, and may be developed independently as reusable components.

2.9 Dynamic data

Some sequencing solutions allow timed data to change dynamically during playback, without requiring a full reload or other disturbances to the presentation. Such dynamism is key to a number of attractive features in multimedia, including interactivity, live authoring, collaborative content production, personalization, adaptation and responsiveness. In SMIL, interactivity is supported by *indeterminate timing*, allowing the end of a time interval to remain unspecified until user input is provided. However, as timing relations are defined in declarative XML, implementing dynamic changes is complicated. As mentioned above, SMIL State [12] addresses this by allowing dynamic variables to be shared between SMIL runtime and external components.

Dynamic sequencing is particularly important in multi-device media, where timed data may be hosted by online services and shared between multiple viewers across the Internet. Modifications made to online data sources should become visible for all connected viewers, preferably as soon as the data is available. Text tracks and the Sequencer support this by allowing cues to be added, modified or removed at any time during playback. This is a bit harder to achieve in the timeout based execution model of the Sequencer.

3. TIMING OBJECT

The Sequencer is directed by an external timing object [7].

Figure 1: The timing object is represented as movement (in real-time) of a point, along a timeline (axis). At any moment the timing object has well defined position, velocity and acceleration. The current position is marked with a red circle, and forward velocity is indicated by the red arrow.

The timing object is a generalization over common timing concepts such as clocks, timers and playback controls, and proposed as a unifying approach for timing and temporal control in Web applications. As illustrated in figure 1 the timing object implements media playback as deterministic motion along a timeline (axis). The timing object is based on concept of Media State Vector [6] and is essen-

tially defined by a clock and a vector. The vector describes the initial state of the current movement, timestamped relative to the clock. This way, future states of the timing object may be calculated precisely from the initial vector and elapsed time. Velocity and acceleration describe continuous movement along the timeline, whereas a discrete jump on the timeline is achieved by dictating a new position. Zero velocity and acceleration (i.e. no movement) is considered a special case of movement.

Crucially, the timing object is also designed to be shared between independent media components, including Sequencers. This way media playback may be precisely synchronized across media components, and media control (i.e. motion changes) may affect all components in unison. Media components may listen to change events emitted by the timing object, to learn about motion changes in a timely manner. This gives raise to a simple API for the timing object, with two operations and one event.

3.1 Timing Object API

Constructor

The timing object constructor optionally specifies range for the timeline.

```
var to = new TimingObject(options);
```

Query

The *query* operation calculates the current state of the timing object, i.e. current position, velocity and acceleration.

```
var v = to.query();
console.log("pos " + v.position);
console.log("vel " + v.velocity);
console.log("acc " + v.acceleration);
```

Update

The *update* operation requests a modification of the timing object by providing new values for position, velocity and/or acceleration.

```
// play from current position
to.update({velocity:1.0});
// jump to start and pause
to.update({position: 0.0, velocity: 0.0});
```

Change Event

Event callbacks may be registered and unregistered using *on()* and *off()* methods.

```
var h = function(){}; // event listener
s.on("change", h);
s.off("change", h);
```

4. SEQUENCER

The Sequencer manages a collection of $(key, interval)$ associations, where *intervals* define the temporal validity of *keys*. A $(key, interval)$ association is also known as a *cue*. The Sequencer then emits *enter* and *exit* events at the correct time, as cues dynamically become *active* or *inactive*, using the timing object as timing source. The Sequencer also supports dynamic changes to its collection of cues during playback.

4.1 Sequencing with the Timing Object

The Sequencer exploits the determinism of the timing object to calculate exactly when future events should be emitted. This way execution may be driven by timeouts . Calculations are triggered on every change event, and performed based on a fresh query result from the timing object. The Sequencer never updates the timing object.

As a mediator of media control, the timing object is quite expressive, supporting any position, velocity or acceleration on the timeline. This includes the discrete, step-wise movement of a slide-show, but also the continuous playback of a video, be it regular speed, slow-motion or fast-forward. Negative velocities, or acceleration may be more useful for data visualization and animation. As the Sequencer is intended as a generic tool, it must support all possible states and state transitions of the timing object, and ensure that the set of active cues always remains consistent. For example, if the timing object skips to a new position on the timeline, enter and exit events must be emitted accordingly. Or, during playback, enter and exit events must be emitted at precisely the correct time.

Note also that the Sequencer supports dynamic switching from one timing object to another. For example, this allows switching from public live presentation to private time-shifted presentation, or joining a friend in a co-viewing session, by switching to his timing object.

4.2 Sequencing points and intervals

The Sequencer works on timed cues, where temporal validity of keys are expressed in terms of intervals or singular points. Singular points are considered a special case of interval, with length zero.

An interval is expressed by two floating point values *low* and *high*, where $low <= high$. If $low == high$ the Interval is said to represent a singular point $[low]$. An interval may or may not include its endpoints. This is defined by optional boolean flags *lowInclude* and *highInclude*. For example, $[a, b], [a, b), (b, a], (a, b)$ are distinct intervals. If *lowInclude* and *highInclude* are omitted, $[a, b)$ is the default value. Special values $-Infinity$ or $Infinity$ may be used to create unbounded intervals, e.g. $[low, Infinity]$ or $[-Infinity, high]$.

This fine grained control over endpoint inclusion may sometimes be required for all the states of a media presentation to be well defined. In particular, if presentation states are defined by non-overlapping, back-to-back intervals along the timeline, e.g. $[a, b), [b, c), [c, d), ...$ endpoint inclusion/exclusion helps avoid ambiguities at the endpoints.

Endpoint inclusion/exclusion also matters when multiple intervals have the same endpoint, as it affects the ordering of emitted events. If the playback direction of the timing object is forwards, events will be emitted according to the ordering below. If the direction is backwards, the ordering is reversed. Note also that the Sequencer emits both enter and exit events for singular points.

-) exit interval with endpoint excluded
- [enter interval with endpoint included
- [enter singular point
-] exit singular point
-] exit interval with endpoint included
- (enter interval with endpoint excluded

4.3 Data-independent sequencing

The Sequencer is data-independent, working on (key, interval) associations known as cues. To use the Sequencer, application programmers parse objects from the data model and register cues with the Sequencer. Whenever the Sequencer emits an *enter* event or an *exit* event, the appropriate cue is provided to the event listener, thereby allowing application code to resolve the appropriate object in the data model. This way, the keyspace decouples the Sequencer from the data model. The Sequencer does not itself generate keys, but leaves design of the keyspace entirely to the application. In this respect the Sequencer is similar to an associative array.

4.4 Window sequencing

The Sequencer supports two modes of operation, *default sequencing* or *window sequencing*. The difference relates to how active and inactive states are defined for cues.

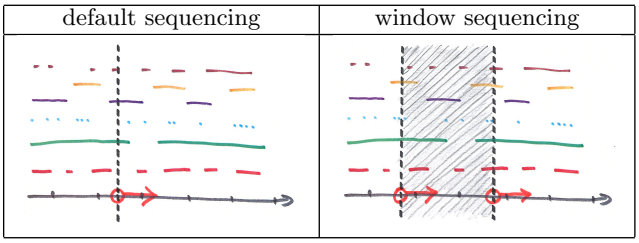

Table 1: Sequencing of multiple tracks of timed data. Different tracks have different colors. Default sequencing involves one timing object, whereas window sequencing involves two.

Table 1 (left) illustrates default sequencing with a *sequencing point* (red circle, vertical line) moving along the timeline. This moving point is defined by the current position of the timing object. Cues intersected by the vertical line are active cues. More formally, in default mode a cue is active if its cue interval covers the sequencing point. In the illustration cues from the green and purple track are currently active. A cue from the brown track has just been exited, and in a short while a blue cue will be entered and then quickly exited.

Table 1 (right) illustrates window sequencing with a *sequencing window* (gray area between vertical lines) moving along the timeline. The sequencing window is defined by two sequencing points, each defined by a timing object. Cues visible between these vertical lines are active cues. More formally, a cue is active if its cue interval is fully or partially covered by the sequencing window. In the illustration 4 brown cues, 2 yellow, 1 purple, 2 green and 3 red cues are currently active.

Window sequencing may be helpful for precisely timed visualization of a sliding window of timed data. Prefetching and buffering of timed data may also be orchestrated in a timed manner, following the same metaphor. Note that the two timing objects can be controlled independently, so the Sequencer allows great flexibility in the control of the sequencing window at any time. For instance, a buffer might grow or shrink during playback to implement certain optimisations in resource consumption. Note also that default sequencing is a special case of window sequencing, where the sequencing window has collapsed into a single sequencing point.

4.5 Sequencer events

The Sequencer supports three distinct event types; *enter*, *exit* and *change*. Enter and exit events relate to changes is the set of active cues, i.e. a cue becoming active or inactive. In contrast, change events relate to modifications of active cues which *remain* active. For instance, the interval of an active cue might have been stretched. Change events allow visualizations driven by the Sequencer to pick up all relevant events for active cues.

Note also that intervals representing singular points will emit both enter and exit events during playback. If the timing object is paused precisely at such a singular point, only the enter event is emitted. The exit event will be emitted as the position is later changed.

4.6 Dynamic sequencing

The Sequencer allows changes to the cue collection at any time, with immediate and consistent effects. Removing an active cue will cause an exit event to be emitted. Similarly, adding a new cue will cause an enter event if the new cue is active. Modification of an existing cue is supported by replacement. Modifications may cause an active cue to become inactive (exit event), or an inactive to become active (enter event). As mentioned above, if cue modification causes an active cue to remain active, a change event is emitted.

5. PROGRAMMING

5.1 Sequencer API

The following outlines the essential parts of the Sequencer API. The complete API documentation, example code and demonstrations are available at the timingsrc [2] Website.

Constructor

The Sequencer constructor takes one, or optionally two timing objects as parameters, implying default or window sequencing mode. The Sequencer is immediately operational.

```
var to = new TimingObject();
var s = new Sequencer(to);
```

Adding and removing cues

addCue associates a key (*String*) with an interval (*Interval*), replacing previous associations if necessary. This way, *addCue* also supports cue modification. *removeCue* removes a key interval.

```
var iv = new Interval(12.2, 14.4);
s.addCue("mykey", iv);
s.removeCue("mykey");
```

Active cues

The Sequencer maintains a list of active cues.

```
var isActive = s.isActive("mykey");
var keys = s.getActiveKeys();
```

Cue events

Events emitted by the Sequencer include the context of the event, in particular its key, interval and event type. Event callbacks may be registered and unregistered using *on()* and *off()* methods.

```
e.key      // cue key (String)
e.interval // cue interval (Interval)
e.type     // cue event type (String)
```

```
var h = function(e){console.log(e.key)};
s.on("enter", h);
s.off("enter", h);
```

The three basic event types of the Sequencer are *enter*, *exit*, or *change*. Additionally the Sequencer provides a special fourth event type *events* which provides all basic event types to a single event handler, delivered in a list. This allows programmers to process Sequencer events in batch mode. This may be useful when multiple events occur simultaneously, for instance if interval endpoints have same value. If so, all effects may be applied to the UI in a single operation.

5.2 Example

Making a timed presentation using the Sequencer only requires a few simple steps.

Create Webpage

Create Webpage with timing object, Sequencer and elements for data viewer and timing object controls.

```
<html>
  <head>
    <script text="javascript">
      var to = new TimingObject();
      var s = new Sequencer(to);
      // app logic here
    </script>
  </head>
  <body>
    <div id="viewer"></div>
  </body>
</html>
```

Parse and register timed data

Timed data is defined in an array. Cues are registered with the Sequencer using array indexes as unique keys.

```
var array = [
  { data: 'A', start: 0, end: 1 },
  { data: 'B', start: 2, end: 3 },
  { data: 'C', start: 4, end: 5 },
  ...
];
for (var i=0; i<array.length; i++) {
  var o = array[i];
  var iv = new Interval(o.start,o.end);
  s.addCue(i.toString(),iv);
}
```

Implement UI

Set up event handlers for Sequencer events. Data associated with active cues is displayed.

```
var v = document.getElementById("viewer");
s.on("enter", function (e) {
  var i = parseInt(e.key);
  v.innerHTML = array[i].data;
});
s.on("exit", function (e) {
  v.innerHTML = "";
});
```

Start presentation

Start playback by interacting with the timing object.

```
to.update({position:0.0, velocity:1.0});
```

5.3 Programming model

As illustrated by the above example, the data-independence and UI-independence of the Sequencer enable an attractive programming model for timed Web applications. Timing is solved and encapsulated by timing objects and Sequencers, leaving the programmer with two fairly simple tasks; 1) parsing timed data and 2) implementing a viewer.

1. Given a data model, a parser function extracts temporal information from the data model and registers cues with the Sequencer. If the data model is dynamic, new changes must be reflected in the Sequencer too.

2. Implementing a viewer for timed data typically involves visualizing the correct data at the correct time in the DOM. This is achieved by implementing handlers for Sequencer events; enter, exit and change. Handler logic is application-specific and works directly with the given (application-specific) data model.

In short, the programmer defines input data and output visualisation, with the unrestricted power of the Web platform at hand. This is also perfectly aligned with reactive, event-driven and data-driven models for Web programming.

Note also that application programmers have much flexibility in how timed data is mapped onto the timeline. For instance, consider the linear presentation of a chess game. The linear state may be defined either as a sequence of board positions, or as a sequence of piece moves. Which to choose is up to the programmer. If linear state is a sequence of board positions, each board position will be mapped to an interval on the timeline. Enter and exit events from the Sequencer will then trigger state transitions between board positions in the presentation. If linear state is rather a sequence of piece moves, each piece move will be mapped to a singular point on the timeline. Events from the Sequencer may then be used to calculate board positions, perhaps by applying a piece move to the current board position.

6. IMPLEMENTATION

We have implemented the Sequencer in plain JavaScript. Source code for the timing object and the Sequencer is open sourced on GitHub as part of the timingsrc [2, 3] repository maintained by W3C Multi-device Timing Community Group [20].

The execution of the Sequencer is ultimately driven by changes in the timing object or changes in the cue collection. For example, whenever a change event is emitted by the timing object, the set of active cues must be re-evaluated to remain consistent with the new state of the timing object. Similarly, changes in the cue collection must be reflected correctly in the set of active cues, and appropriate events must be emitted.

In addition, when the timing object specifies non-zero velocity or acceleration, the set of active cues must be re-evaluated at precisely the correct time, as cues become active or inactive. The Sequencer does this by calculating exactly *when* future tasks are due, and then schedules their execution using a timeout.

Mathematical equations for linear motion under constant acceleration are used to calculate future time-intersections between the timing object and intervals on the timeline. If acceleration is non-zero, calculations involve solving quadratic equations. These are still cheap calculations. Scheduled tasks are put on a sorted task queue, from which the main loop of the Sequencer processes all due tasks. When all due tasks have been processed, a new timeout is scheduled based on the calculated time of the next task. This way, the Sequencer maintains at most one timeout at any time (two timeouts in window sequencing mode). If either the timing object or any of the cues are changed, scheduled tasks need to be recalculated.

Device	Browser	Avg	Min	Max
Desktop	Chrome	-0.373	-0.827	0.588
Desktop	Firefox	-0.049	-1.037	7.413
Laptop	Chrome	4.324	-0.431	5.494
Laptop	Firefox	3.783	-0.051	5.787
Mobile	Chrome	1.215	-0.348	4.232
Mobile	Firefox	1.610	-0.379	10.114

Table 2: Measuring *lateness* of setTimeout callbacks. A negative value implies that the timeout was not late, but early. Values in milliseconds.

The precision of the Sequencer is limited by the precision of JavaScript *setTimeout*. Table 2 gives a rough indication of how setTimeout behaves in Chrome and Firefox browsers running on a few common device types. The desktop is a Hewlett Packard office computer running Ubuntu, the Laptop is a MacBook Air running OSX, and the mobile is a Samsung S4 running Android.

In the experiment 100 timeouts are processed over a period of 100 seconds. For each timeout, the *lateness* of the timeout callback is computed, using *performance.now()* as reference. Values are given in milliseconds. Negative values imply that the timeout was not late, but early.

The results show that average precision may be expected within a few milliseconds, which is quite acceptable for a wide variety of applications. In particular, this is well below the screen refresh rate, which is often 60Hz or lower (17 milliseconds or higher).

On the other hand, setTimeout is handled by the browser main event queue, implying that unrelated processing in the browser may cause timeouts to be significantly delayed. This is particularly true on low-end devices. Modest improvements might be possible in JavaScript, for instance by scheduling timeouts early and finding ways to spend the remaining time, until the timeout is due. However, busy looping is not an elegant solution, causing side effects such as lagging UI and increased power consumption. Instead, this highlights the need for native timing support in Web browsers to improve, especially on low-end devices.

Finally, the Sequencer is designed to work effectively with a large cue collection. Internally, the Sequencer uses binary search for efficient lookup.

7. EVALUATION

The Sequencer has been in repeated use for a couple of years already in a variety of multimedia applications. In this period a number of bugs have been identified and resolved. Corner cases have been tested and verified by carefully scripted tests. The API has also been adjusted for simple use and is now considered stable and production quality. The Sequencer is currently functionally complete, correct, reliable and easy to use. To indicate the broad utility of the Sequencer, we detail a selection of scenarios in Web-based multimedia where we have found the Sequencer to be a valuable tool.

7.1 Editing subtitles during playback

The HTML5 media element provides built-in support for subtitles. Still, we have found it quite attractive to ignore this feature and instead use the Sequencer for subtitle presentation, for instance in a transparent layer on top of the media element. The MediaSync library [2, 3, 8, 9] ensures that the Sequencer and the media element are precisely synchronized via a shared timing object. This setup provides flexibility in rendering of subtitles. More importantly, since there is no obligation to use a standard format (e.g. WebVTT) we could easily integrate with application-specific JSON data from an online service. As the online service supports notifications and the Sequencer supports dynamic cue changes, we can immediately demonstrate live editing and authoring of subtitles during playback. Subtitles may for instance be stretched, time-shifted or edited. With support for multi-device playback this qualifies as a rudimentary system for collaborative viewing and commenting on linear content.

7.2 Multi-device slide show

A simple slide show may be created by mapping objects to integers, and using the timing object to implement stepwise slide navigation. The Sequencer then provides the correct object to the slide viewer at any time. We have used this to craft a multi-device slide show where each slide is a Web page. This means we can remote control a multi-device Web-based slideshow for a global audience. Typically we also provide a secondary track of slides for a secondary view, on mobile or similar. One Sequencer runs the slide show, but each individual slide may also include Sequencers for other purposes.

7.3 Time-shifting live data

Live production of timed cues presents a particular challenge, as both endpoints of a cue interval may not be known at production time. This is easily solved with the Sequencer. For instance, one may simply define a preliminary cue interval that stretches into infinity: $[start, Infinity)$, and then later when the endtime is available the cue interval is simply replaced with $[start, end)$. We have used this technique as a basis for live production of HTML5 chess visualization. An interactive chess board widget allows two players to play a game of chess. Every time a piece is moved, a timestamp from the timing object (production time) is used to end the previous board position and start the next. In effect, we capture a time series of board positions, based on the natural interactions of the players. This time series is then readily available for time-shifted playback. In fact, the board visualization widget uses the Sequencer to present the correct

board position, given the current state of the timing object (visualization time). If the same timing object is used for production-time and visualization-time, this corresponds to live (real-time|direct) presentation. Time-shifted playback of live data only requires a different (time-shifted) timing object to be used for visualization-time. Finally, this entire experience was made multi-device by representing board positions and timing objects as online resources. This way, we could demonstrate natural authoring of multimedia through distributed timed capture of collaborative interaction, with distributed live and time- shifted playback. The timing complexity of this application is considerable, yet fully encapsulated by timing objects and Sequencers.

7.4 Secondary device, companion view

Companion apps and second screen applications are often motivated as a way of providing supplementary information and interactive capabilities to linear experiences. Typically these applications are developed as native applications for smartphones and tablets, with temporal alignment based on audio fingerprinting, local network communication or similar near-range techniques. Using sequencing and online timing objects, we made a simple, Web-based companion view for videos. We use the subtitles track from the video to produce a list of timed keywords. For each keyword, we provide a relevant information card, an external Web page or similar. Using the Sequencer these keywords are then replayed and visualized on a phone or tablet, allowing the user to look at the device for contextual information. Note that the companion view does not have neither audio nor video, but is based only on the subtitles combined with an online timing object for synchronization. Implementing such timed companion views as live Web pages is a very attractive approach. Compared to native applications, they are simpler in development, require no installation and work across platforms. Even more interesting, live Web pages produced as timed Web pages trivially support time-shifting. This means that live Web productions become reusable for on-demand media consumption.

A complication in this kind of use case is that smartphones and tablets are power-saving or sleeping while idle. So, as the user unlocks the screen, network connections might need to be reconnected and correct data must be rendered as soon as possible. However, as long as effects of reconnects materialise as change events to timed data or timing object, this is indistinguishable from normal operation, from the perspective of the Sequencer. As a result no additional efforts are required from the programmer and correctness is not in conflict with established mechanisms for reduced power consumption.

7.5 Limitations

The Sequencer is intended as a generic and simple programming tool. For this reason the design of the Sequencer has been guided by the minimalist principle. For example, as discussed in Section 2.5, the Sequencer does not provide special support for relative timing.

Periodic cues is another useful, yet unsupported feature. Periodicity may be currently be achieved by copying cues, though this is not a particularly elegant solution. A second approach to repetition would be to loop the timing object. A third approach is to build repeat support into the cue collection of the Sequencer. This has been done and verified to work, but is currently not included in the Sequencer.

More flexibility with respect to the key-space could also be useful. For example, the Sequencer cues could associate multiple intervals to one key, or multiple keys to one interval. Also, a hierarchical name space for keys could simplify sequencing of multiple unrelated data sources. Following this, support for event filtering could be added based on key prefixes.

In the end, these extra features would complicate the API and possibly obfuscate the basic concept. Instead of aiming for one Sequencer that does it all, we imagine a selection of Sequencer variants. Application programmers would then pick the ones that match their problem. More specialised Sequencers may even be crafted by wrapping and extending the basic Sequencer. This would only strengthen the concept.

With respect to implementation, the JavaScript Sequencer is limited by timing capabilities of current Web browsers, in particular the precision of the setTimeout mechanism. Currently precision is limited to about a millisecond and processing in the browser may cause timeouts to be significantly delayed.

In addition, packing the timeline too densely with cues (or equivalently applying too much velocity to the timing object), as well as defining time consuming tasks in event handlers would be problematic. This though is not specific to the Sequencer, but applies to JavaScript applications in general.

If precise sequencing was supported natively by Web browsers, it is likely that timing guarantees could be stronger and precision improved.

8. DISCUSSION

8.1 Any kind of timed data, any purpose

The Sequencer API is inspired by text track API supported by HTML media elements. However, by isolating sequencer logic from media elements, specific data formats and UI solutions, the value of the Sequencer as a generic programming concept becomes more evident.

Sequencing logic is recognised as part of any media framework. Using the Sequencer, programmers may easily build new frameworks and timed components. The Sequencer may be used to produce Web-based visualisations from any kind of timed data, or to organize the correct execution of any timed operation. Examples of timed resources might include timed images, text, CSS, JSON, HTML, scripts, geolocations, timed sensor-data, SVG, audio samples, canvas operations etc. In other words, anything Web can be timed.

The Sequencer may also be used for timed actions that do not produce audiovisual effects. For instance, consider timed pre-fetching of data, or testing a system with timed network requests from distributed clients, while simulating temporal patterns in network load. The ability to present data at the correct time may also be used to reduce complexity in content transfer. In particular, real-time (multicast) streams are often used as a mechanism to preserve time-ordering and timing relations across a network. With timestamped messages and a Sequencer on the receiving end, temporal relations may be correctly re-created, independent of the mechanism used for data transfer. This gives more flexibility in the choice of transport mechanism, and potentially reduces complexity at the sender-side.

8.2 Sequenced media

The Sequencer is designed exclusively for timed data. Still, importantly, the Sequencer plays a fundamental role in rich media productions, where presentations are being synthesised in real time from a mix of continuous and discrete media sources. In such presentations a variety of video and audio tracks may be anchored to the same timeline, possibly overlapping, or with gaps in-between. The Sequencer then orchestrates timely loading and unloading of a variety of media objects, including audio and video tracks.[1] In broadcasting, sequencing has typically been organized relative to a master video track. Instead, by putting sequencing logic at the center, sequenced media offers a more general model where the standard broadcasting approach is merely a special case. In short, for sequenced media the timing object defines motion (playback, progress) along the timeline and the Sequencer implements the temporal structure of the presentation.

For broadcasters, the transition to sequenced media is set to become a major shift in the years to come. The BBC has labelled this object-based broadcast OBB (or object-based media) [11], and promotes this concept as a re-invention of broadcasting for the IP world. The core idea is to represent and disseminate media as individual objects (as opposed to a byte stream), and then synthesise them into timed presentation on the client-side. This is key to a number of desirable features in media products, such as personalization, accessibility, interactivity, dynamism, extensibility, live editing/authoring, responsiveness and adaptation. Composite Media [4] was a contemporary definition of the same idea. Like OBB, Composite Media emphasises client-side, real-time synthesis of media objects, from independent online data sources. Significantly though, Composite Media takes the idea one step further, as it is based on a model for distributed timing and temporal control that works in the Web environment. This effectively extends the scope of sequenced media from single-device playback to globally synchronized experiences.

8.3 The timed Web

The availability of timing and sequencing tools in JavaScript has profound implications for the Web as multimedia platform. The Web always was a multimedia platform, but early support for timed multimedia was mainly provided by dedicated plugins isolated from the Web runtime. SMIL [31] and SVG [26] provided timing and vector animations. Later, Flash gained wide adoption by providing support for audio, video and animations, as well as cross-platform consistency and effective authoring tools. The plugin-based multimedia model favoured sophisticated, feature-rich frameworks and complex (proprietary) media formats. Still, the disconnect from the Web platform itself was problematic. This was partly compensated by replicating popular features of the Web runtime inside the media framework, essentially making it a browser within a browser.

HTML5 brought proper support for AV media to the Web, opening the possibility for native multimedia support in Web browsers. However, with no explicit timing model, Web-based multimedia got centered around media elements. The idea was to synchronize other media to the playback of the media element and text track support was introduced for this purpose. Media elements were also strengthened with Media Source Extensions (MSE) [25], and further integration with SVG and Canvas [23] animations has also been suggested. Still, this AV centric model has some clear limitations. The timing model is coarse, and difficulties arise when media presentations include multiple media elements, or none.

Now, with the introduction of the timing object and the Sequencer, a proper timing model is available directly in the Web runtime. This eliminates the limitations of the AV centric model, and online timing objects enable precisely synchronized multimedia playback, globally, on multiple devices. Furthermore, multimedia developers may harness the full power of the Web browser as a multimedia playback renderer. This includes unrestricted use of well established tools and technologies for Web development, as well as integration with application specific data models and online services. Hopefully, with native Web support for timing and sequencing, the Web may realise its true potential as the ultimate platform for distributed, timed multimedia. This is the timed Web.

[1] media elements additionally require synchronization for AV playback to be correctly aligned with the timeline. The MediaSync library [2, 3, 8, 9] is available for this purpose.

9. CONCLUSIONS

This paper emphasizes sequencing as a vital part of multimedia, and advocates the decoupling of sequencing logic from data model, UI and timing/control. The Sequencer is presented; a generic tool for sequencing of cues defined by intervals and points on the timeline. Our JavaScript implementation of the Sequencer is based on setTimeout, ensuring precise timing of event callbacks while reducing energy consumption. It is also designed to work effectively with a large cue collection. The Sequencer uses the timing object as external timing source. This implies that the Sequencer supports temporal controls appropriate for a wide range of media applications, and is readily available for traditional page-local playback scenarios. More impressively, through the use of online timing objects, the Sequencer supports distributed multimedia Web applications depending on globally synchronized playback. The mathematical model used for timing objects and the Sequencer allows great flexibility while keeping the concepts well defined and easy to understand. Support for window sequencing further broadens the utility of the Sequencer.

We argue that this generic Sequencer concept, realised as an easy-to-use, standalone programming tool, has considerable value for programmers of timed Web applications. In particular, the Sequencer reduces development costs and boosts innovation. This has consistently been confirmed through repeated use in a wide variety of multimedia applications over the last few years. The Multi-device Timing Community Group advocates standardization of the timing object and sequencing tools, as central concepts in a new programming model for timed, multi-device, Web-based multimedia applications.

10. ACKNOWLEDGEMENTS

The authors would like to thank François Daoust and Dominique Hazael-Massieux of the W3C for reviewing this work and for sharing their insights into the evolution and current state of related Web standards. We also want to thank the reviewers for helpful comments.

11. REFERENCES

[1] Adobe. Adobe Flash. https://www.adobe.com/products/flashruntimes.html.

[2] I. M. Arntzen and N. T. Borch. Timingsrc: A programming model for timed web applications, based on the Timing Object. Precise timing, synchronization and control enabled for single-device and multi-device Web applications. http://webtiming.github.io/timingsrc/.

[3] I. M. Arntzen and N. T. Borch. Timingsrc: Open source implementation. https://github.com/webtiming/timingsrc.

[4] I. M. Arntzen and N. T. Borch. Composite Media, a new paradigm for online media. In *2013 NEM Summit (Networked Electronic Media)*, NEM Summit '13, pages 105–110. Eurescom, 2013.

[5] I. M. Arntzen, N. T. Borch, F. Daoust, and D. Hazael-Massieux. Multi-device linear composition on the web; Enabling multi-device linear media with HTMLTimingobject and Shared Motion. In *Media Synchronization Workshop (MediaSync) in conjunction with ACM TVX 2015*. ACM, 2015.

[6] I. M. Arntzen, N. T. Borch, and C. P. Needham. The media state vector: A unifying concept for multi-device media navigation. In *Proceedings of the 5th Workshop on Mobile Video*, MoVid '13, pages 61–66, New York, NY, USA, 2013. ACM.

[7] I. M. Arntzen, F. Daoust, and N. T. Borch. Timing Object; Draft community group report. http://webtiming.github.io/timingobject/.

[8] N. T. Borch and I. M. Arntzen. Distributed synchronization of html5 media. Technical Report 15, Norut Northern Research Institute, 2014.

[9] N. T. Borch and I. M. Arntzen. Mediasync report 2015: Evaluating timed playback of html5 media. Technical Report 28, Norut Northern Research Institute, 2015.

[10] D. C. Bulterman, J. Jansen, K. Kleanthous, K. Blom, and D. Benden. Ambulant: A fast, multi-platform open source SMIL player. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, MULTIMEDIA '04, pages 492–495, New York, NY, USA, 2004. ACM.

[11] T. Churnside. Object-Based Broadcasting. http://www.bbc.co.uk/rd/blog/2013-05-object-based-approach-to-broadcasting, 2013.

[12] J. Jansen and D. C. Bulterman. Enabling adaptive time-based Web applications with SMIL State. In *Proceedings of the Eighth ACM Symposium on Document Engineering*, DocEng '08, pages 18–27, New York, NY, USA, 2008. ACM.

[13] J. Jansen, P. Cesar, R. L. Guimaraes, and D. C. Bulterman. Just-in-time personalized video presentations. In *Proceedings of the 2012 ACM Symposium on Document Engineering*, DocEng '12, pages 59–68, New York, NY, USA, 2012. ACM.

[14] Microsoft. Microsoft Silverlight. http://www.microsoft.com/silverlight/.

[15] MMA. The MIDI Manufacturers Association (MMA). https://www.midi.org/specifications.

[16] Shared Motion by Motion Corporation. http://motioncorporation.com.

[17] Mozilla. Popcorn.js the HTML5 media framework. http://popcornjs.org/.

[18] MPEG-4. http://mpeg.chiariglione.org/standards/mpeg-4.

[19] MPEG-4 Systems. http://mpeg.chiariglione.org/standards/mpeg-4/systems.

[20] Multi-device Timing Community Group. https://www.w3.org/community/webtiming/.

[21] M. Shotton. HTML5 Video Compositor. https://github.com/bbc/html5-video-compositor.

[22] Ambulant open SMIL player. http://ambulantplayer.org/.

[23] W3C. HTML Canvas 2D Context. https://www.w3.org/TR/2dcontext/.

[24] W3C. HTML5 Text Track. http://dev.w3.org/html5/spec-preview/media-elements.html#text-track.

[25] W3C. Media Source Extensions. https://www.w3.org/TR/media-source/.

[26] W3C. Scalable Vector Graphics (SVG) 1.1. https://www.w3.org/TR/SVG/.

[27] W3C. SMIL 3.0 synchronized multimedia integration language. http://www.w3.org/TR/REC-smil/.

[28] W3C. Time-marches-on. http://www.w3.org/html/wg/drafts/html/master/semantics.html#time-marches-on.

[29] W3C. Web Audio API. http://webaudio.github.io/web-audio-api/.

[30] W3C. WebVTT: The Web video text tracks format. http://dev.w3.org/html5/webvtt/.

[31] W3C. Synchronized multimedia integration language (SMIL) 1.0 specification. https://www.w3.org/TR/1998/REC-smil-19980615/, Jun 1998.