Synchronizing Web Documents with Style

Rodrigo Laiola Guimarães[1], Dick C. A. Bulterman[2], Pablo Cesar[3] and Jack Jansen[3]

[1] IBM Research
Rua Tutóia 1157
04007900 São Paulo, Brazil
+55 11 2132 2283

[2] FX Palo Alto Laboratory
3174 Porter Drive
CA 94304 Palo Alto, USA
+1 650 842 4800

[3] CWI: Centrum Wiskunde & Informatica
Science Park 123
1098 XG Amsterdam, The Netherlands
+31 20 592 9333

rlaiola@br.ibm.com, dick.bulterman@fxpal.com, p.s.cesar@cwi.nl, jack.jansen@cwi.nl

ABSTRACT

In this paper we report on our efforts to define a set of document extensions to Cascading Style Sheets (CSS) that allow for structured timing and synchronization of elements within a Web page. Our work considers the scenario in which the temporal structure can be decoupled from the content of the Web page in a similar way that CSS does with the layout, colors and fonts. Based on the SMIL (Synchronized Multimedia Integration Language) temporal model we propose CSS document extensions and discuss the design and implementation of a proof of concept that realizes our contributions. As HTML5 seems to move away from technologies like Flash and XML (eXtensible Markup Language), we believe our approach provides a flexible declarative solution to specify rich media experiences that is more aligned with current Web practices.

Categories and Subject Descriptors

D.3.2 [**Language Classifications**]: Specialized application languages; I.7.2 [**Document and Text Processing**]: Document Preparation – *Format and notation, Hypertext/hypermedia, Languages and Systems, Multi/mixed media, Standards.*

General Terms

Design, Experimentation, Standardization, Languages.

Keywords

Structured timing and synchronization, Time Style Sheets, HTML5, CSS3, JavaScript, SMIL, W3C.

1. INTRODUCTION

This year marks the 25th anniversary (or thereabouts) of the World Wide Web (also known as WWW, W3 or simply Web). What started as a method for scientists to structure, interlink and share research has changed profoundly the way we communicate and share information. From the technological perspective, today's Web also looks very different from what it was on its birth. While in the beginning Web documents contained only static information and limited formatting, supporting technologies like CSS and JavaScript – and later AJAX (*Asynchronous JavaScript and XML*) – contributed immensely to a new era of dynamism within Web pages.

As the Web establishes itself as a crucial content delivery and consumption platform, the temporal aspect now plays an important role in the discussions within the HTML Working Group. Among many new features, HTML5 (*HyperText Markup Language* version 5) offers built-in support for audio and video content. In spite of these advances, the HTML language still provides limited support to create rich media experiences like video mashups [6][13]. On the one hand, structured timing and synchronization has for long been part of the multimedia research agenda within the WebMedia community [1][8][11][12]. On the other hand, much of those contributions are still to be seen on the Web. We believe that one of the reasons for that is that most previous efforts focused on XML-based solutions, which define a whole new set of tags that impact how Web documents should be authored. In this context, the question we ask ourselves is:

How can we integrate structured timing and synchronization on Web documents other than based on the XML language profile?

In this paper we look at structured timing and synchronization within Web documents from a new perspective. Our primary contribution is a set of document extensions that allow timing and synchronization of HTML elements to be specified with CSS – a style sheet language conceived primarily for describing the look and formatting of a document. Our CSS document extensions, named *Time Style Sheets*, are based on the SMIL [2] temporal model. As HTML5 directly aims at addressing issues of reliability, security and performance of technologies like Flash[1], we believe this paper presents a number of insights to be considered as the Web evolves.

In particular, the requirements that motivated our work include:

i. *Support readability, maintainability and reusability:* although timing and synchronization of elements within a Web document can be achieved with scripting, we aim for declarative solutions that favor accessibility and interoperability; and

ii. *Separate temporal semantics from document structure:* instead of adding new tags to the HTML language, we believe that a less intrusive manner to time and synchronize elements within a Web page is necessary. To this extent, the CSS functional module is naturally decoupled from the document structure.

The remaining of this paper is organized as follows. Section 2 overviews related work. Section 3 introduces a set of CSS document extensions to define structured timing and synchronization in Web documents. These functionalities are

[1] Some technologies mentioned in this paper, if unknown, could very easily be identified via a simple online search; therefore they will not be Web-referenced.

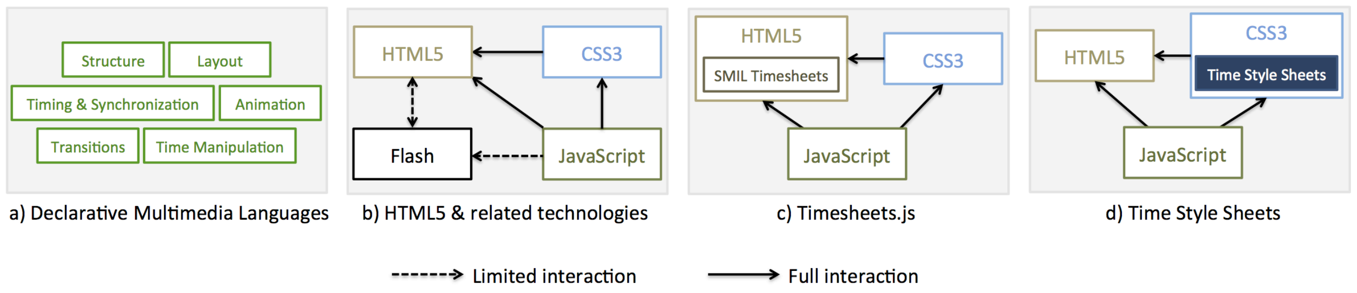

Figure 1. Technology comparison.

targeted to structural elements as well as to continuous media like HTML5 audio and video. Then, in Section 4 we discuss the implementation of a proof of concept that realizes our contributions. Finally, Section 5 is dedicated to concluding remarks and points to future directions.

2. RELATED WORK

In the context of this work, it is worth discussing mechanisms that allow timing and synchronization to be integrated into documents. In the remaining of this section we review some declarative multimedia languages and other representative alternatives targeted specifically to the Web domain (see Figure 1).

2.1 Declarative Multimedia Languages

NCL (*Nested Context Language*) is an XML-based language used to specify interactive multimedia presentations within the Brazilian Terrestrial Digital TV System (SBTVD-T) and ITU-T [14]. NCL has a strict separation between the document content and structure, and it provides non-invasive control of presentation timing, linking and layout (see Figure 1a). In NCL authors can declaratively describe the temporal behavior of a multimedia presentation using connectors and links. Although the application of such feature has been considered in the Web domain before [9], the use of links to specify the temporal synchronization of elements seems a step far ahead that current Web is yet not ready to embrace (nearly all Web links are used as user interaction points for page navigation).

SMIL is the main multimedia container format supported by W3C, the World Wide Web Consortium. Like NCL, SMIL is an XML-based integration format, and as such, it does not directly define media objects (with the exception of timed text content). Instead, SMIL acts as a container format in which spatial, temporal, linking and interactive primitives can be used to position, schedule and control a wide assortment of multimedia presentations. In spite of the efforts to make SMIL the multimedia language for the Web[2], it has not been widely deployed in this context (although it is one means of animating SVG – *Scalable Vector Graphics* [7]).

In this work we argue that a *fat-free* alternative to XML-based approaches is necessary, as demonstrated by the efforts to support the temporal aspect within CSS3 Animation and Transition working drafts.

2.2 Timing and Synchronization on the Web

HTML is the markup language to create Web pages. One of the innovative features of HTML5 is the introduction of the `<audio>` and `<video>` elements as first-class citizens of the HTML language. These elements implicitly define a temporal dimension for the referenced media object. HTML5 though provides a very restricted scope of temporal synchronization that only applies to the video and its captions (via the `<track>` element). Embedded scripting (such as JavaScript) is the primary means of controlling time and affecting the behavior of HTML documents.

HTML5 also introduces the `<canvas>` element, which allows for dynamic, scriptable rendering of 2D shapes and bitmap images. HTML5 `<canvas>` can be used with JavaScript to create high-demanding applications like games, in which pixel level manipulation and performance are essential. Our work differs not only in the programming paradigm (JavaScript is procedural), but also due to the fact that we target timing and synchronization of HTML elements instead of low-level pixels in a bitmap image.

CSS level 3 (or simply CSS3) brings the temporal aspect in the Transition and Animation modules. CSS Transitions[3] provide an easy way to do simple animation, but they give little control to the author on how the animation progresses (CSS property values are interpolated between start and end states of the animation). Similarly, CSS Animations[4] change the presentational value of CSS properties over time. The main difference is that CSS Animations allows the author to specify CSS property changes as a set of `@keyframes` rules. Many aspects of the animation can be controlled, including whether or not to delay its start time, how many times the animation iterates, and whether or not the animation should be running or paused. In this paper we take the CSS temporal aspect one step further by considering that the presentation of any group of elements (and not only animations) can be synchronized and controlled.

While JavaScript and Flash (this last to a more restricted extent) can be used to handle timing, synchronization and interaction (see Figure 1b), a number of efforts advocate for more accessibility and reusability. With this in mind, Popcorn.js[5], an HTML5 media framework written in JavaScript, enables the creation of rich time-based interactive experiences on the Web. Using a plugin factory mechanism, Popcorn.js allows the presentation of a video, audio or other media to control and be controlled by arbitrary elements within a Web page. Although plugins can be easily reused, authors need to be familiarized with JavaScript, since all configurations are done in that language.

Most closely to our work, Timesheets.js[6] proposes a declarative approach to synchronize Web documents. Timesheets.js relies on

[2] http://www.w3.org/TR/NOTE-HTMLplusTIME

[3] http://www.w3.org/TR/css3-transitions/

[4] http://dev.w3.org/fxtf/web-animations/

[5] http://popcornjs.org/

[6] http://wam.inriaalpes.fr/timesheets/

a JavaScript implementation of a SMIL Timesheets scheduler that runs in the Web browser. Cazenave et al. [4] suggest that a small set of new tags and attributes from Timesheets.js would facilitate the integration of structured timing and synchronization in the HTML and SVG languages (see Figure 1c). Our work is related, but instead we focus on providing document extensions to the CSS language (see Figure 1d).

3. OUR APPROACH

In this paper we propose a means to support structured timing and synchronization of HTML elements by extending the CSS language with a set of SMIL functionalities. Instead of supporting the whole SMIL profile – which defines a vast number of modules, some of which used in very specific situations – we aim at some core features defined in the Basic SMIL timing model.

By design, we decided to keep the document structure in HTML, event handling in JavaScript and the presentation layout in CSS, while *Time Style Sheets* (TSS), a CSS language extension, defines the presentation temporal semantics. This way there is a clear functional separation between the information organization, the behavior manipulation, the presentation layout, and now the timing and synchronization behavior within a Web page (requirement i). As part of the CSS language, TSS definitions are declarative (requirement ii) and follow the same rules of cascade, specificity and inheritance. In the remaining of this section we provide a detailed overview of the facilities provided by Time Style Sheets to encode temporal presentations on the Web.

3.1 Timing Properties

As aforementioned, our approach builds on a subset of the Basic SMIL timing model. This module defines when elements in a presentation get scheduled and, once scheduled, how long they will be active. One of the challenges of the integration of the SMIL functionality in a non-XML language is to determine how SMIL's time containers and attributes can be modeled in the target language. Our design choice was to define a set of timing properties and values within the CSS language as summarized in Table 1 and Table 2. These properties can also be read and modified via JavaScript. This provides a clean mechanism to add limited functionality to the existing CSS specification without major integration overhead.

A simple example of integrating the CSS property-based timing extension functionality is given in the following HTML fragment:

```
01. <div id="slideshow">
02.   <img src="img1.png" />
03.   <img src="img2.png" />
04.   <img src="img3.png" style="timing-delay:1s" />
05. </div>
06.
07. <style>
08.   #slideshow {
09.      timing-container: seq;
10.      timing-interaction-count: infinite;
11.   }
12.
13.   #slideshow img {
14.      timing-delay: 0s;
15.      timing-duration: 2s;
16.      border: 1px solid green;
17.   }
18. </style>
```

Here, the `<div>` element named *slideshow* is defined to behave as a SMIL sequential container that will be played infinite times (lines 09-10). The children elements, in this case the `<img>` elements, also have timing properties to control their temporal behavior. As illustrated in Figure 2, the presentation of an image will start immediately one after another, and each will last for 2 seconds (lines 14-15). The exception is the last image, which will start with a 1-second delay as specified in line 04 (inline styles have precedence over embedded CSS definitions). Note that timing properties can also be easily combined with existing CSS properties (line 16).

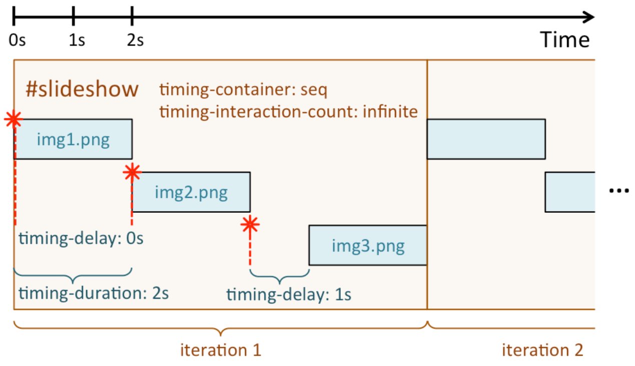

Figure 2. Slideshow example: each image waits for the previous child of the sequence to finish, and then it plays.

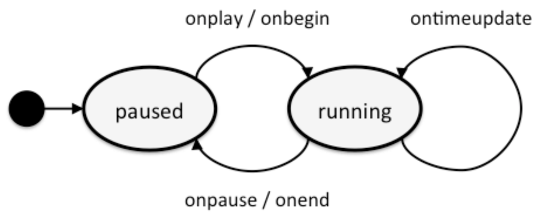

Figure 3. State machine and associated events.

Besides the specification of an element's temporal behavior, TSS also allows for playback control through timing properties. Internally, an element initiates on the *paused* state, and it changes to *running* whenever is its turn to play (see Figure 3). If the `timing-play-state` property is defined as *running*, the transition in the state machine will only occur at the appropriate moment (e.g. if we had specified it for image #3, this would still play after the presentation of image #2). Alternatively, it is possible control the state machine of an element via JavaScript. TSS is also compliant with the `autoplay` attribute of the `<audio>` and `<video>` elements. If this attribute is specified, it has a higher precedence than a TSS definition, and the `timing-play-state` property assumes the *running* value. Otherwise, the behavior is the same as for the other elements. The state transitions are associated with 5 main events: *onplay*, *onbegin*, *onpause*, *onend* and *ontimeupdate*. Timing events are discussed in the next subsection.

Finally, instead of using the SMIL name convention, in TSS we chose to name timing properties after similar concepts that are already in use within the CSS3 Animation and Transition specifications (e.g. `timing-delay` corresponds to the `begin` attribute in SMIL, `timing-duration` to `dur` and `timing-iteration-count` to the `repeatCount` attribute).

3.2 Timing Events

HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document. These events are normally used in combination with functions, which will only be

Table 1. Timing properties.

Property	Description	Syntax
`timing-container`	Specifies how the presentation of the children elements will be scheduled. Available containers are `par`-allel and `seq`-uential	**CSS Syntax:** `timing-container: (par)|seq|initial;` **JavaScript Syntax:** `object.style.timingContainer = "seq";`
`timing-delay`	Defines when an element will start. Its value is defined in seconds (s) or milliseconds (ms)	**CSS Syntax:** `timing-delay: time (0s)|initial;` **JavaScript Syntax:** `object.style.timingDelay = "2s";`
`timing-duration`	Specifies how many seconds or milliseconds an element takes to complete one cycle	**CSS Syntax:** `timing-duration: time (implicit|infinite)|initial;` **JavaScript Syntax:** `object.style.timingDuration = "1s";`
`timing-iteration-count`	Defines how many times an element should be played	**CSS Syntax:** `timing-iteration-count: number (1)|infinite|initial;` **JavaScript Syntax:** `object.style.timingIterationCount = "infinite";`
`timing-play-state`	Specifies whether an element is running or paused. This property can be used in JavaScript to pause or resume an element's playback in the middle of a cycle	**CSS Syntax:** `timing-play-state: (running)|paused|initial;` **JavaScript Syntax:** `object.style.timingPlayState = "paused";`

Note: the default value is specified within parenthesis.

Table 2. Property values.

Value	Description
initial	Sets a given property to its default value
infinite	Only applicable for the `timing-iteration-count` and `timing-duration` properties. Specifies that the element should be played infinite times or one cycle will never end, respectively
number	A number that defines how many times a given property should be considered. Default value is 1
time	Defines the number of seconds or milliseconds. The default value is 0 for `timing-delay`. For the `timing-duration` property, it is infinite for static media (e.g. image) and implicit to continuous media
paused	Specifies that the element is paused
running	Default value. Specifies that the element is running

executed when the event occurs. The Timing Events proposed in the Time Style Sheets specification (see Table 3) extend current DOM events and follow the W3C DOM Level 2 standard model. The Timing Events associated to the Time Style Sheets framework can be normally used within HTML elements as shown below.

HTML Syntax:

```
<element onbegin="SomeJavaScriptCode">
```

JavaScript Syntax:

```
object.onbegin=function(){SomeJavaScriptCode};
```

Alternatively, it is also possible to register/unregister event listeners on event target objects using the JavaScript methods `addEventListener` and `removeEventLister` (IE8: `attachEvent` and `detachEvent`), respectively.

Table 3. Timing events.

Event	Description
onbegin	The event occurs when the playback cycle of an element starts
onend	The event occurs when the playback cycle of an element ends
onplay	The event occurs when the playing state of an element changes to *running*
onpause	The event occurs when the playing state of an element changes to *paused*
ontimeupdate	The event occurs when the playback time changes

3.3 Timing Pseudo-Classes

In Time Style Sheets, we can use CSS pseudo-classes to define specific presentation styles for different playback phases. In general, CSS pseudo-classes are keywords added to selectors to specify special states or relations to an element. They take the form of `selector:pseudo_class {property: value;}`, simply with a colon in between the selector and the name of the pseudo-class. For example, `:hover` can be used to apply a style when the user hovers over the element specified by the selector.

In our framework, 2 pseudo-classes have been defined to style an element when its playback cycle effectively *starts* (after timing-delay is computed) or *ends*, as follows.

```
selector:active { /* after computing delay */
   property: value;
}

selector:not-active { /* applied onend   */
   property: value;
}
```

As a matter of completeness, the example presented in Subsection 3.1 will only bring on the expected behavior (one image being

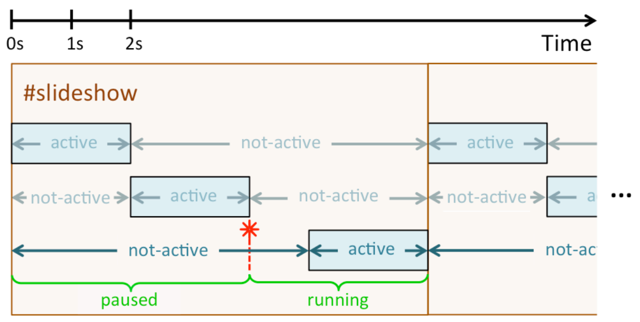

Figure 4. An example of the different phases and states used to describe the presentation of elements.

displayed and disappearing after the other, sequentially) if we define the timing pseudo-classes as presented below. Figure 4 illustrates these phases and states overlap. One could argue the `display` property equals `none` should be implicit for the `not-active` pseudo-class, but we opted to leave the specification of this behavior to the author. In SMIL a similar functionality can be achieved using the `timeAction` attribute. Although this attribute can produce interesting presentations, we believe it may also lead to confusion (e.g. its value can refer to a CSS class, style or a specific property). Pseudo-classes provide a more elegant and normalized solution that is aligned with current Web practices.

```
#slideshow img:active {
   /* not necessary in this example */
}

#slideshow img:not-active {
   display: none;
}
```

3.4 Support to Rich Media Content

We have seen that Time Style Sheets defines a set of CSS properties, events and pseudo-classes that allow for structured timing and synchronization of HTML elements within a Web page. In this section we discuss a complementary set of properties targeted to media elements, in particular continuous media. These definitions can be useful when creating rich media experiences like video mashups [6][13]. In Table 4 we summarize these additional timing properties. In order to exemplify the use of such properties we present 2 examples.

Table 4. Timing properties related to media.

Property	Description
`timing-clip-begin`	Specifies the time at which a continuous media stream begins playing, relative to the start of the media file. The value of this property must be specified in seconds (s) or milliseconds (ms)
`timing-clip-end`	Specifies the time at which a continuous media stream stops playing, relative to the start of the media file. The value of this property must be specified in seconds (s) or milliseconds (ms)
`timing-volume`	Defines the relative output of an audio object. It takes a value from 0.0 to 1.0. The default value is 1.0, which corresponds to 100%. A lower value makes the audio play more silently
`timing-sync-master`	Identifies which element (by its unique id) should be used as the master synchronization clock. By default an element follows its internal clock

First, we introduce a *playlist* example composed of an audio track (lines 02-06) and a collection of video elements identified as *video_seq* (lines 07-26). These 2 components will be played in parallel and the playlist will last for 2min (lines 31-32). On the other hand, the video_seq's inner videos will play sequentially one after another (as specified in lines 36 and 45-47). Note that instead of playing the entire video file, the `timing-clip-begin` and `timing-clip-end` properties are used to define video clips of interest (lines 08, 13 and 21). Alternatively, the use of Media Fragments[7] could be considered to specify the temporal dimension of a media clip using URIs (*Uniform Resource Identifier*). It is also worth mentioning that only the audio element will reproduce sound (line 02), while the audio of the video clips will be played silently – but will still be played (line 40).

```
01. <div id="playlist">
02.    <audio style="timing-volume:1.0">
03.      <source src="track.ogg" type="audio/ogg"/>
04.      <source src="track.mp3" type="audio/mpeg"/>
05.      Your browser does not support HTML5 audio.
06.    </audio>
07.    <div id="video_seq">
08.      <video style="timing-clip-begin:0s;
                      timing-clip-end:5s;">
09.        <source src="v01.ogg" type="video/ogg" />
10.        <source src="v01.mp4" type="video/mp4" />
11.        Your browser does not support HTML5 video.
12.      </video>
13.      <video style="timing-clip-begin:30s;
                      timing-clip-end:34s;">
14.        <source src="v02.ogg" type="video/ogg" />
15.        <source src="v02.mp4" type="video/mp4" />
16.        Your browser does not support HTML5 video.
17.      </video>
18.
19.      ...
20.
21.      <video style="timing-clip-begin:13s;
                      timing-clip-end:19s;">
22.        <source src="vN.ogg" type="video/ogg" />
23.        <source src="vN.mp4" type="video/mp4" />
24.        Your browser does not support HTML5 video.
25.      </video>
26.    </div>
27. </div>
28.
29. <style>
30.    #playlist {
31.       timing-container: par;
32.       timing-duration: 120s; /* it lasts 2 min */
33.    }
34.
35.    #video_seq {
36.       timing-container: seq;
37.    }
38.
39.    #video_seq > video {
40.       timing-volume: 0; /* videos kept on mute */
41.       width: 480px;
42.       height: 320px;
43.    }
44.
45.    #video_seq > video:not-active {
46.       display: none;
47.    }
48. </style>
```

[7] http://www.w3.org/TR/media-frags/

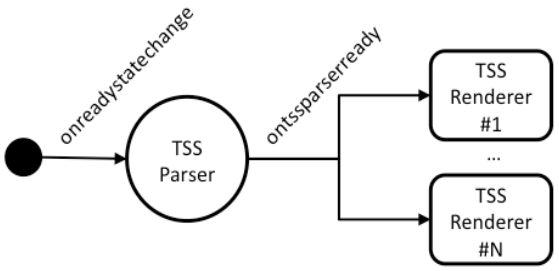

Figure 5. Implementation diagram.

Next, we illustrate a rich media example that could be used in an education environment [3][10][15]. In this second use case, a video element identified as *video_quiz* (lines 01-05) is used as the *conductor* of an interactive experience. We then prompt the user with a *quiz* (lines 06-12) at certain point of the video playback. For that, we first need to specify the video as the master synchronization timing of the quiz (line 16), and then set the moment at which the quiz must be presented (20s after the start of the video, as shown on line 17). The `timing-sync-master` property indicates that instead of being controlled internally, the clock of the video element will send time updates to control the presentation of the quiz.

```
01. <video id="video_quiz">
02.    <source src="movie.ogg" type="video/ogg" />
03.    <source src="movie.mp4" type="video/mp4" />
04.    Your browser does not support HTML5 video.
05. </video>
06. <form id="quiz">
07.    <h3>Which nation has won the most World
           Cups?</h3>
08.    <input type="radio" value="germany">...
09.    <input type="radio" value="italy">...
10.    <input type="radio" value="brazil">...
11.    <input type="submit" value="Submit"
           onclick="checkQuiz();">
12. </form>
13.
14. <style>
15.    #quiz {
16.       timing-sync-master: #video_quiz;
17.       timing-delay: 20s;
18.    }
19.
20.    #quiz:not-active {
21.       display: none;
22.    }
23. </style>
24.
25. <script>
26.    var quiz = document.getElementById("quiz");
27.    quiz.addEventListener("onplay", function(ev){
28.       ev.stopPropagation();
29.
30.       document.getElementById("video_quiz").
             style.timingPlayState = "paused";
31.    });
32.
33.    function checkQuiz() {
34.       ...
35.       /* if answer is correct, resume video */
36.       document.getElementById("video_quiz").
             style.timingPlayState = "running";
37.       ...
38.    }
39. </script>
```

We still make use of extra scripting to customize the presentation behavior. Once the quiz appears to the user, the video playback is paused (lines 26-31). When the user answers the quiz and press the *submit* button (defined on line 11), the answer is checked and, if correct, the video playback is then resumed (lines 33-38).

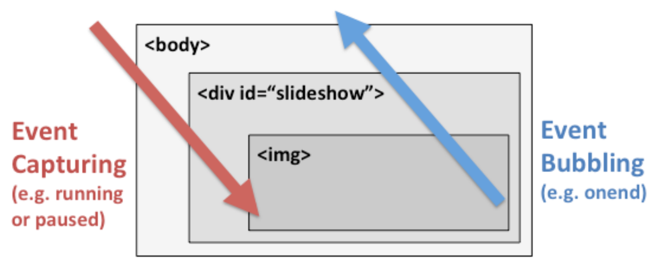
Event Capturing (e.g. running or paused)
<body>
<div id="slideshow">
<img>
Event Bubbling (e.g. onend)

Figure 6. Event propagation methods.

3.5 Extensibility

As shown in this section, Time Style Sheets allows authors to time and synchronize elements within a Web page. However, the framework can still be extended to support new functionalities. As we saw in Subsection 3.4, a complementary set of properties is necessary to media elements, in particular continuous media.

Another consideration to make is that we decided to support only explicit time values, instead of more complex *syncbase* or *eventbase* timing. As a reminder to an interested reader, among other things SMIL allows specifying the activation, duration and termination of an element based on other elements' timing (e.g. start image #2 2s after image #1 has begun) or interactive events (e.g. start image #2 after clicking on image #1), respectively. Although one might think we lose in terms of expressiveness, we believe our approach favors readability. By using predictable timing we also can easily support the change of time style on the fly. It is important to mention that with unpredictable playback times (e.g. a live video stream) a different approach would be necessary. In the current design we opted to yield up some of these concepts to keep things simple. We believe this is a topic that can be explored in future work.

4. IMPLEMENTATION

A TSS compliant agent is composed of a parser that interprets a TSS definition and a renderer that implements the semantics specified in such document (see Figure 5). As current Web browsers do not support the specification of custom CSS properties and pseudo-classes, we developed a TSS parser and renderer engine in JavaScript. To make use of our TSS proof of concept implementation, an author just has to import the JavaScript files in the head of the HTML document. Once the page is completely loaded by the browser, the TSS parser examines the associated styles and triggers a custom `ontssparserready` event. At this moment, a TSS renderer that listens to such event takes over and schedules the document presentation accordingly. Note that different renderers can implement different semantics. For instance, one could provide a visual representation of the scheduling for authoring purposes instead of focusing on the document playback.

In our implementation, the playback of elements is controlled using Event Capturing (top-down), while the notification of timing events uses Event Bubbling (see Figure 6). Both event propagation approaches are part of the W3C standard. For the first, this means that one can play or pause any element, without impacting the presentation of other elements within a Web page. This is true *unless* the other elements are descendants of the element into question (e.g. if a slideshow is paused, the presentation of the inner image running at that moment will also

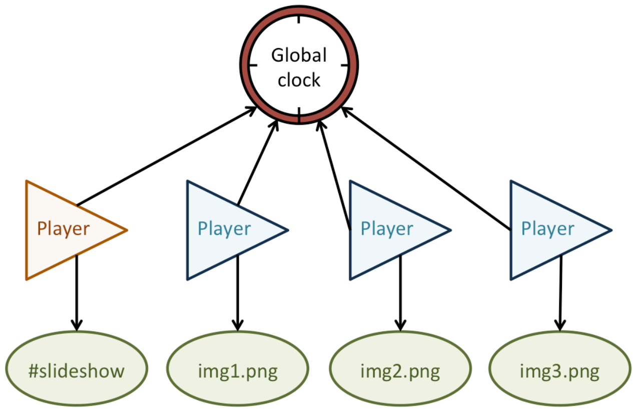

Figure 7. The renderer's implementation at a glance.

be paused). For timing events (e.g. onbegin, onend etc.) we use a complementary approach. In the Event Bubbling (bottom-up) model, an event is first captured and handled by the inner most elements, and then propagated to outer ascendants. This way, the slideshow's temporal container (in our case, sequential) gets notified of their children end event and can start the next image.

The other characteristic feature of our renderer's implementation is that time is not inherited. This means parent nodes do not provide timing information to their children – unless the `timing-sync-master` property has been defined explicitly for doing so. Each timed element has an independent player that is updated by a global clock. The global clock is a source of monotonically increasing time values unaffected by adjustments to the system clock. The time values produced by the global clock represent wall-clock milliseconds from an unspecified historical moment. Because the zero time of the global clock is not specified, the absolute values of the time values produced by the global clock are not significant, only their rate of change. The global clock is not exposed in the programming interface and nor is it expected to be exposed by markup. In the future we may consider the use of hierarchical arrangements of time relationships between timed elements.

Typically, a player is tied to the global clock such that its absolute time is calculated as a fixed offset from the time of the global clock. This offset is established by designating some moment as the player's zero time and recording the time value of the global clock at that moment. At subsequent moments, the time value of the player's timeline is calculated as the difference between the current time value of the global clock and the value recorded at the zero time. A player realizes some timed behavior (e.g. sequential container, parallel container or media) and binds itself to the source node (see Figure 7). A player also allows run-time control and it is in charge of exposing the JavaScript API (*Application Programming Interface*) discussed in Section 3.

5. FINAL REMARKS

Much has changed in the World Wide Web in these 2 decades of WebMedia conference, and it seems that at last the temporal aspect started receiving the attention it deserves in the HTML language (and related technologies). In this work we presented a set of document extensions to support structured timing and synchronization within Web pages. Time Style Sheets was designed to be fully declarative rather than procedural. This way, we also favor reusability, maintainability and readability, making it more accessible to a wider audience (requirement i). As Time Style Sheets is built on top of CSS, a style sheet language, we also keep a clear functional separation between the HTML structure and the temporal semantics (requirement ii).

One could argue that some of the facilities proposed in this work could be achieved within HTML5 by adding complex additional scripting, or perhaps by assimilating technologies like SMIL or NCL in totality. Although this may be true, we believe that the great challenge is to combine the best lessons learned in the past with current Web design principles, balancing flexibility and expressiveness to not reinvent the *wheel*. In these lines, this work shows that the SMIL temporal model transcends a specific host language (in this case XML-based) and it is a powerful abstraction to be considered when handling the temporal aspect within a Web page. These results directly answer our research question.

In the near future, we plan to make public our TSS implementation. As for rich media experiences like video mashups, there are implications on the design of the TSS agent (or player), mainly in regard to intelligent prefetching mechanisms [5] within the Web browser. We plan to explore this topic in future work. Finally, we expect that some of our contributions can bring new insights and help in the evolution of CSS within W3C.

6. ACKNOWLEDGMENTS

This work has been partially funded by the Brazilian Ministry of Science and Technology under contract no. FINEP 03.11.0371.00.

7. REFERENCES

[1] Azevedo, R.G.A., Lima, B.S., Soares Neto, C.S. and Teixeira, M.M. 2009. An approach for textual authoring of hypermedia documents based on the use of programmatic visualization and hypertextual navigation. In *Proceedings of the XV Brazilian Symposium on Multimedia and the Web* (WebMedia '09). ACM, New York, NY, USA, Article 18, 8 pages. DOI=10.1145/1858477.1858495 http://doi.acm.org/10.1145/1858477.1858495

[2] Bulterman, D.C.A and Rutledge, L.W. *SMIL3.0 - Interactive Multimedia for Web, Mobile Devices and DAISY Talking Books*. Springer-Verlag, 2009. ISBN: 978-3-540-78546-0

[3] Cambruzzi, W.L., Rigo, S.J. and Barbosa, J.L.V. 2012. A proposal for managing multiple trails in educational environments. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web* (WebMedia '12). ACM, New York, NY, USA, 25-28. DOI=10.1145/2382636.2382645 http://doi.acm.org/10.1145/2382636.2382645

[4] Cazenave, F., Quint, V. and Roisin, C. 2011. Timesheets.js: when SMIL meets HTML5 and CSS3. In *Proceedings of the 11th ACM symposium on Document engineering* (DocEng '11). ACM, New York, NY, USA, 43-52. DOI=10.1145/2034691.2034700 http://doi.acm.org/10.1145/2034691.2034700

[5] Gao, B., Jansen, J., Cesar, P. and Bulterman, D.C.A. 2010. Beyond the playlist: seamless playback of structured video clips. *IEEE Trans. on Consum. Electron*. 56, 3 (August 2010), 1495-1501. DOI=10.1109/TCE.2010.5606288 http://dx.doi.org/10.1109/TCE.2010.5606288

[6] Jansen, J., Cesar, P. Guimarães, R.L. and Bulterman, D.C.A. 2012. Just-in-time personalized video presentations. In *Proceedings of the 2012 ACM symposium on Document engineering* (DocEng '12). ACM, New York, NY, USA, 59-

68. DOI=10.1145/2361354.2361368 http://doi.acm.org/10.1145/2361354.2361368

[7] King, P., Schmitz, P. and Thompson, S. 2004. Behavioral reactivity and real time programming in XML: functional programming meets SMIL animation. In *Proceedings of the 2004 ACM symposium on Document engineering* (DocEng '04). ACM, New York, NY, USA, 57-66. DOI=10.1145/1030397.1030411 http://doi.acm.org/10.1145/1030397.1030411

[8] Marques Neto, M.C. and Santos, C.A.S. 2009. StoryToCode: a model based on components for specifying interactive digital TV convergent applications. In *Proceedings of the XV Brazilian Symposium on Multimedia and the Web* (WebMedia '09). ACM, New York, NY, USA, Article 8, 8 pages. DOI=10.1145/1858477.1858485 http://doi.acm.org/10.1145/1858477.1858485

[9] Melo, E.L., Viel, C.C., Teixeira, C.A.C., Rondon, A.C., Silva, D.P., Rodrigues, D.G. and Silva, E.C. 2012. WebNCL: a web-based presentation machine for multimedia documents. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web* (WebMedia '12). ACM, New York, NY, USA, 403-410. DOI=10.1145/2382636.2382719 http://doi.acm.org/10.1145/2382636.2382719

[10] Piton-Gonçalves, J. and Aluísio, S.M. 2012. An architecture for multidimensional computer adaptive test with educational purposes. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web* (WebMedia '12). ACM, New York, NY, USA, 17-24. DOI=10.1145/2382636.2382644 http://doi.acm.org/10.1145/2382636.2382644

[11] Santanchè, A., Mota, M., Costa, D., Oliveira, N. and Dalforno, C.O. 2009. Componere: component-based in web authoring. In *Proceedings of the XV Brazilian Symposium on Multimedia and the Web* (WebMedia '09). ACM, New York, NY, USA, Article 12, 8 pages. DOI=10.1145/1858477.1858489 http://doi.acm.org/10.1145/1858477.1858489

[12] Santos, J.A.F., Braga, C. and Muchaluat-Saade, D.C. 2013. Automating the analysis of NCL documents with a model-driven approach. In *Proceedings of the 19th Brazilian symposium on Multimedia and the web* (WebMedia '13). ACM, New York, NY, USA, 193-200. DOI=10.1145/2526188.2526214 http://doi.acm.org/10.1145/2526188.2526214

[13] Shrestha, P., de With, P.H.N., Weda, H., Barbieri, M. and Aarts, E.H.L. 2010. Automatic mashup generation from multiple-camera concert recordings. In *Proceedings of the international conference on Multimedia* (MM '10). ACM, New York, NY, USA, 541-550. DOI=10.1145/1873951.1874023 http://doi.acm.org/10.1145/1873951.1874023

[14] Soares, L.F.G., Moreno, M.F., Soares Neto, C.S. and Moreno, M.F. 2010. Ginga-NCL: declarative middleware for multimedia IPTV services. *IEEE Comm. Mag.* 48, 6 (June 2010), 74-81. DOI=10.1109/MCOM.2010.5473867 http://dx.doi.org/10.1109/MCOM.2010.5473867

[15] Viel, C.C., Melo, E.L., Pimentel, M.G. and Teixeira, C.A.C. 2013. Multimedia multi-device educational presentations preserved as interactive multi-video objects. In *Proceedings of the 19th Brazilian symposium on Multimedia and the web* (WebMedia '13). ACM, New York, NY, USA, 51-58. DOI=10.1145/2526188.2526211 http://doi.acm.org/10.1145/2526188.2526211