Timesheets.js: When SMIL Meets HTML5 and CSS3

Fabien Cazenave
INRIA
655 avenue de l'Europe
38334 Saint Ismier, France
fabien.cazenave@inria.fr

Vincent Quint
INRIA
655 avenue de l'Europe
38334 Saint Ismier, France
vincent.quint@inria.fr

Cécile Roisin
Grenoble University & INRIA
655 avenue de l'Europe
38334 Saint Ismier, France
cecile.roisin@inria.fr

ABSTRACT

In this paper, we explore different ways to publish multimedia documents on the web. We propose a solution that takes advantage of the new multimedia features of web standards, namely HTML5 and CSS3. While JavaScript is fine for handling timing, synchronization and user interaction in specific multimedia pages, we advocate a more generic, document-oriented alternative relying primarily on declarative standards: HTML5 and CSS3 complemented by SMIL Timesheets. This approach is made possible by a Timesheets scheduler that runs in the browser. Various applications based on this solution illustrate the paper, ranging from media annotations to web documentaries.

Categories and Subject Descriptors

I.7 [**Document and Text Processing**]: Document Preparation—*Markup languages, Multi/mixed media, Standards*

General Terms

Design, Experimentation

Keywords

Declarative languages, Multimedia, Web applications, SMIL, HTML5

1. INTRODUCTION

The multimedia web is evolving very rapidly. With the advent of HTML5, web pages can now integrate graphics, sound and video seamlessly. In addition, scripting languages can animate content such as text, graphics, pictures, as well as continuous media. These multimedia contents can be played on an increasing number of terminals, ranging from the traditional desktop to the smallest pocketable mobile device, and with good performances.

Rendering true multimedia web content is becoming easier. The user experience on web pages embedding video and sound is now much smoother than back in the days when specific plug-ins were required. Modern web browsers can now render natively all sorts of contents embedded in HTML5 pages. In addition, they are supporting graphic formats such as SVG, including its animation feature, as well as the latest properties introduced in CSS to handle transitions and animations, not to mention their ability to manipulate all these contents through powerful script engines.

With these novel multimedia features, new web applications are made possible. They are very diverse, but they all have in common the addition of a time dimension to the usual web document. Time could be present through such continuous content as video or sound, through a time structure added to discrete contents, through animations performed by executing some code (script), or through a combination of these. Concretely, these timed multimedia applications may be slide shows, captioned video clips, annotated audio recordings, graphic animations, augmented recorded conferences, interactive photo albums, web documentaries, and so on.

To implement such web applications, various approaches can be considered, ranging from declarative to imperative:

- The purely declarative approach was taken by SMIL [4], the first multimedia technology specially designed for the web. In SMIL, the time dimension of a document is expressed by a hierarchy of temporal operators. Applications are run by players that interpret the declarative language and play the document (Ambulant [2], RealPlayer, X-Smiles [12]).
- The imperative approach is illustrated by the many ad hoc applications written in JavaScript and ActionScript, where scripts are used to handle user interaction and to make the document change over time.

Obviously, both approaches can be combined; many applications using a declarative language are complemented by some scripting. It is worth noting that scripting languages can also be used to implement players that interpret a declarative language. For instance, the SMIL Timesheets language [17] was implemented in JavaScript (Timesheets JavaScript Engine [16], LimSee3 [10], FakeSmile[1]), but this does not impact application developers, who still work declaratively.

The choice between both approaches should take many criteria into account. What efforts are necessary to make sure the content can be enjoyed on different devices? Is accessibility for people with disabilities granted? What kind of user interaction is supported? Can (some parts of) the application be easily reused in another application? How easy is it to maintain applications? Is it possible to reference some particular piece of content or some specific state or

DocEng'11, September 19–22, 2011, Mountain View, California, USA.

[1] http://leunen.d.free.fr/fakesmile/

date in the document? How can synchronization with continuous content be achieved? Answers to these questions may vary strongly depending on the approach.

In this paper, we try to identify the advantages of each approach. We consequently propose a trade-off that brings the best of each world regarding the questions raised above, and we propose tools that help achieving the best trade-off. In particular, we have chosen to explore a solution built upon HTML5 and SMIL Timing.

The rest of the paper is organized as follows. The next section presents various categories of multimedia web applications. It is followed by a review of the current means available to implement these applications. Sections 4 and 5 propose solutions based on SMIL Timesheets as a new way to improve the current situation. Section 6 provides a few examples to illustrate these solutions. Finally the conclusion summarizes the main contributions of the paper and envisions future work.

2. MULTIMEDIA APPLICATIONS

For the sake of clarity, we divide multimedia web applications in two main categories, depending on the role of the time dimension.

2.1 Media-Driven Applications

We call *media-driven applications* the broad category of multimedia applications where a piece of continuous content plays the role of a backbone for the whole application. In these applications, typically, an audio or video recording constitutes the main content, and various elements (often discrete media such as text or pictures) are associated to parts of this main content to annotate it. Interactive features are also available to the user for moving freely in the main content and to interact with complementary information. This category is exemplified by such applications as:

- a captioned movie: the main content is the movie itself, and the captions constitute the associated content. The display of each caption is precisely synchronized with the video. In addition to the usual VCR controls, a menu allows the user to change caption language or to hide them at any time. A very fine-grained synchronization may be required when the annotation is the transcript of the audio track, like in the MIT150 Infinite History project [11].
- a commented program in an on-line radio archive: the backbone is an audio file, the recorded radio program. Some pictures and textual annotations are associated with specific parts of the program to illustrate them or to provide additional information. Usual controls are provided to pause, play, move forward/backward, and to hide/show additional content.
- a videotaped talk with synchronized slides (see Figure 1): the main content is the video recording of the speaker, which is complemented with the slides (pictures and/or text) the speaker used when giving the talk. Slides are synchronized with the video and displayed next to it. In addition to the usual controls for a video, an interactive table of contents allows the user to freely navigate through the video and the sequence of slides.

In all these applications, the master media object comes with its intrinsic clock which is used to schedule the whole application. The

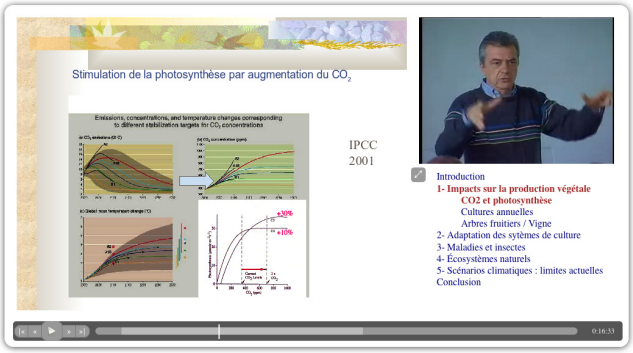

Figure 1: Videotaped talk with synchronized slides

time dependencies of all other contents are expressed relatively to this clock. In the latest example, each slide is associated with two dates: the time when it must be displayed and the time when it must disappear. Both dates are relative to the master clock. In the same way, items of the table of contents have two dates: the time when the corresponding section starts in the video and the time when it stops. These dates are used to highlight the relevant entry in the table of contents, or to skip to the right position in the video when the user clicks an entry.

Because of this time structure, it is often easy in this kind of application to superimpose one or several sectioning structures on the master media object. Typically, in the example above, both the sequence of slides and the table of contents play this role. The slide titles may be used to associate a series of labels with the timeline, while the headings from the table of contents add a hierarchical structure. This can be reflected through the user interface: when moving the pointing device along the timeline, the user can see various levels of labels, which are helpful to choose a particular part in a long video.

This is for instance the approach taken in the Advene platform where multiple levels of annotations can be associated with a movie [3].

2.2 Event-Driven Applications

As opposed to media-driven applications, *event-driven applications* are not organized around a single continuous media object that provides the main synchronization scheme. Instead, they are made of a collection of multiple media objects related by links such as temporal relations or user interactions.

To illustrate this concept, let us take the example of a slide show. Each slide may be a single picture or a piece of text, but it could also be a small multimedia document itself, with various (possibly continuous) media objects and some interactive features. Slides are linked together to define a preferred sequence for presentation. They also offer the user a way to conveniently move from one slide to the next/previous one in this sequence. A table of contents (or an index) may be available, with links to every slides, offering the user another way to access slides, in any order.

This kind of organization may be used for a photo album for instance. The table of contents contains thumbnails of all pictures, and each slide is constituted by a single photograph with a caption, comments and buttons to go to the next or previous slide, or to the

thumbnail index. The same kind of organization is used for the slides displayed on a large screen during a talk with a tool such as HTML Slidy [13].

The category of event-driven applications is broader than the slide show family. It actually includes all applications where there is no dominant time structure, but multiple, different time structures for different parts of the document. Concurrent time structures may occur simultaneously, or some parts may have no intrinsic time dimension, only temporal relations with other pieces of content. For instance, a media object may have to be presented after (or at the same time as) another one. These relations are typically those defined by Allen [1].

3. STATE OF THE ART

3.1 Multimedia Web Authoring

The issue of integrating multimedia content into web pages dates back from the early days of the web. In the initial version of WorldWideWeb, the first web browser ever (1990), even still pictures where displayed in separate windows. They were included in the text later on (1993), when the NCSA-Mosaic browser introduced the `img` tag in HTML. Integration of continuous media objects, in particular video, followed the same way, but much later. For a long time, these objects were handled by separate programs (plug-ins) and were therefore difficult to integrate in the document. Only when SMIL introduced the `video` and `audio` elements could web documents really include continuous media. SMIL was followed in this direction by SVG and more recently by HTML5. All three languages now support audio and video objects natively.

The time dimension of documents was not the priority in the first solutions developed for multimedia web content. Web formats have left aside the issue of synchronization for a long time. Plug-ins did not allow the various components of a web page to be synchronized with continuous media through a standard API. The first step in this direction was made with SMIL, with a rather radical approach: time is the main (and almost the only) dimension for structuring a document. As a consequence, the hierarchical structure offered by other web formats for representing the logical organization of documents can hardly be expressed with SMIL.

Using an XML environment is a natural way to put the focus on the logical dimension when creating multimedia content: the document is structured in XML to encode its logical organization, and time relations are expressed as part of this structure [6]. This option requires some export mechanism to produce documents in a format that can be accepted by a web browser. The main issue is that the authoring and the publishing languages are different. The original XML code has to be converted to SMIL, Flash or HTML5+JavaScript. Authors are prevented from using familiar web languages such as HTML and CSS, and if they want to make adjustments to the final form, they may have to do complex reverse transformations to update the source document [15].

The drawbacks of the conversion is balanced by the advantages of the declarative approach. Authors think in terms of a multimedia *document* instead of a multimedia application they would have to *program* with a scripting language. The document-oriented (declarative) approach thus makes multimedia web authoring available to a broader audience. The declarative approach also provides advantages from an engineering point of view. It makes it easier to maintain and reuse content. Multimedia documents can then be used in a document workflow, for instance.

3.2 Multimedia Web Technologies

There are three main classes of techniques (SMIL, Flash, HTML5) for developing synchronized multimedia documents for the web. Figure 2 reviews these techniques, based on the following criteria:

- Timing and synchronization: This is the most complex part of many advanced multimedia applications. Therefore the language must provide efficient ways to alleviate the task of web developers in this area.
- Scriptability: Whatever the level of declarativeness of a language, it is important to be able to go beyond the limitations of the language by using scripts that extend its capabilities in some particular situations (this requirement has been highlighted by [14]).
- Logical structure: Document formats on the web are typically declarative structured languages. By describing first and foremost the logical organization of a document, they facilitate accessibility, adaptability, reuse, device independence, and processability.
- Content/presentation separation: Keeping this principle, which is widely applied on the web with style sheets, offers flexibility in customizing and adapting content for different contexts of use.
- Temporal links [7]: It is common practice on the web to link to a specific position within a text document, thanks to fragment identifiers in URIs. The equivalent for a timed document is to point to a specific date or state in the execution of the document. This feature allows timed documents to integrate nicely in the web.
- Native rendering in web browsers: The browser is the main tool for accessing the web on all devices. To make sure web users can use multimedia applications, it is important that applications can be run in web browsers.
- Standard compliance: W3C standards are well known to web developers. A solution based on these standards is more likely to be adopted by them. These standards are also widely implemented, in particular in browsers. Applications published according to W3C standards can meet a large audience.
- Device friendliness: Even if web browsers implementing W3C standards are available on all devices, there are significant differences in the capabilities of these devices. The chosen technology should be compatible with as many device classes as possible.

We use these criteria to review the technologies used for developing multimedia web applications (see Figure 2):

- SMIL was designed specifically for multimedia web applications. Its main focus is a rich time structure and content synchronization. Unfortunately, few media players and no web browser support SMIL. Moreover, SMIL has not evolved along with popular web languages such as HTML and CSS. It does not allow to separate presentation from content, and it is impossible to script a SMIL document in a web browser. This makes it difficult to integrate the SMIL language into other web applications. In addition, the structure it represents is the time structure, not the logical structure of a document.

	SMIL	Flash	HTML5 + CSS3
Declarative timing and sync.	+	-	-
Scriptability	-	+	+
Logical structure	-	-	+
Content/presentation separation	-	-	+
Temporal links	+	-	-
Native rendering in web browsers	-	-	+
W3C standard compliance	+	-	+
Desktop friendliness	+/-	+	+
Mobile friendliness	+/-	+/-	+

Figure 2: Technology Comparison

- Flash is currently by far the main technology for multimedia applications. It works well on most desktop browsers, but it is not so widely supported on mobile devices (smartphones, tablets). Since it comes as a binary format, it raises accessibility issues and requires more work to address indexability aspects. More generally, as Flash is not rendered natively by web browsers there are strong limitations regarding the content/presentation separation, and the interactions between a web page and its multimedia (Flash) content are more complex.
- HTML5 was specified for the web with audio/video in mind. HTML5 is complemented by the CSS3 style sheets language. With HTML5 and CSS3, web developers can take advantage of the clean content/ presentation separation, as well as many other web features, but they have to rely on JavaScript to handle timing, synchronization and user interactions. Concerning multimedia applications, HTML5 with its `audio` and `video` elements is supported natively by modern web browsers and, thanks to plug-ins, fallback solutions exist for audio/video contents in legacy browsers (Internet Explorer 6 to 8).

3.3 Declarative Solutions

As seen on Figure 2, Flash suffers from too many limitations to be considered a good web citizen. Web developers are left with SMIL and HTML5+CSS3 (see Figure 3), but neither solution is completely satisfactory.

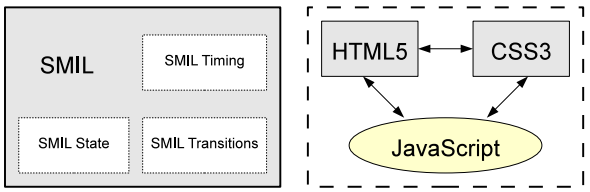

Figure 3: SMIL vs. HTML5

The latest version of SMIL, namely SMIL 3.0 [4], provides two modules (SMIL State and SMIL Transitions) in addition to the central SMIL Timing module, in order to better cope with advanced multimedia requirements. It is a declarative approach to multimedia contents:

- Timing and Synchronization features are defined by the SMIL Timing module. SMIL Timing brings a simple and very powerful way to describe, in a declarative way, media synchronization and user interaction management for multimedia applications.
- SMIL State [8] and its variant for NCL [14] aim at providing variables inside the declarative time structure (as defined by SMIL or NCL) to cope with the need for controlling document playback, but the resulting syntax is a bit verbose and requires a specific SMIL implementation. Besides, it still cannot be extended to more complex imperative scenarios (e.g. functions, prototypes, objects).
- SMIL Transition Effects can be used to enhance the user experience in pure SMIL documents. CSS3 transitions are the counterpart for HTML5.

The most frequent approach is based on HTML, CSS and JavaScript (see right part of Figure 3):

- Developers of multimedia applications rely on HTML5 to describe the content with its logical structure, and on CSS3 for the presentation. This way, they can separate content from presentation.
- The document playback is controlled in JavaScript by using element APIs and DOM events as defined by W3C recommendations. In particular, the HTMLMediaElement API provides an efficient control for `audio` and `video` elements.
- Developers still have to address most timing and user interaction issues with specific JavaScript code. This is the main difficulty in designing multimedia applications with HTML5: the scripts are designed for a specific DOM structure and use pre-defined CSS classes, which makes such JavaScript developments very difficult to maintain and reuse, unless the relationship between classes and features is carefully documented.
- In many applications, especially in event-driven applications, developing and debugging scripts represent a large part of the development effort.

King *et al.* [9] have proposed XML language extensions to allow multimedia systems to react to dynamic events, and to handle continuous real-time dependencies. In our solution, we take the same implementation approach (a scheduler engine in JavaScript) but we can now take advantage of new features of declarative web languages to partially cover the same needs.

4. PROPOSED SOLUTION

4.1 SMIL Timing and Timesheets

Based on the observations above, it appears that combining HTML5+CSS3 and SMIL Timing would bring a good solution (see Figure 4). SMIL Timing specifies two attributes, `timeContainer` and `timeAction` for integrating timing and synchronization features into HTML and XML documents.[2] Basically, SMIL Timing allows an a-temporal language such as HTML5 to be extended with timing features.

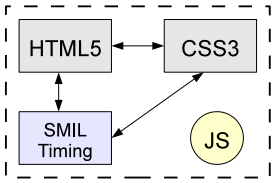

Figure 4: SMIL with HTML5+CSS3

SMIL Timesheets reuses a significant subset of SMIL Timing and allows timing and synchronization to be separated from content and presentation. This can be seen as the counterpart of CSS style sheets in the timing domain: like in CSS, these features are gathered either in an external resource linked to the document, or in a `timesheet` element in the document itself. Like CSS style sheets, timesheets can be associated not only to HTML pages but also to other types of documents such as SVG drawings, for instance, or even to compound documents made of HTML and SVG.

Our approach [5] is based on SMIL Timesheets and can be summed up in three points:

- use HTML5+CSS3 for structuring the content and for rendering it natively in the browser with a clean content/presentation separation;
- rely on SMIL Timing to handle timing, media synchronization and user interaction;
- do not ever redefine what already exists in HTML, SVG and CSS (e.g. animations and transitions), as illustrated in Figure 7.

4.2 Timesheet Engine

As SMIL Timing and Timesheets are not supported natively by web browsers, a JavaScript implementation of these specifications is required. We have developed timesheets.js,[3] which is an open-source, cross-browser, dependency-free library that supports the common subset of the SMIL Timing and SMIL Timesheets specifications.

This still relies on JavaScript, but no specific JavaScript development is required from a web developer for most multimedia applications: the whole application is created using only declarative languages. When such an application is running, some parts of it (HTML and CSS) are executed natively by the browser, some other parts are executed by the browser's JavaScript engine.

[2]http://www.w3.org/TR/SMIL3/smil-timing.html#Timing-IntegrationAttributes

[3]http://wam.inrialpes.fr/timesheets/public/timesheets.js

Timesheets.js is not the first SMIL Timesheets engine running in the browser. Vuorimaa has developed a Timesheets JavaScript Engine [16] but it has a few limitations for our use cases:

- As it has been developed in 2007, before the raise of SVG 1.2 and HTML 5, it does not support continuous media. In the LimSee3 project [10] (2008), this timesheet engine was adopted to handle the Timing module better and to play continuous media elements through a VLC plug-in, but it still cannot use any event sent by these continuous media, which leads to weak synchronization.
- Both implementations handle only internal timesheets; the W3C Timesheets 1.0 specification does not mention explicitly any other way to use timesheets with a-temporal languages, but we wanted to support both internal and external timesheets, as well as inline SMIL Timing markup, within the same parser.
- Both implementations rely on clock arrays and try to determine begin/end values as soon as possible, which is fine in a fully declarative approach, but is limitative when it comes to adjusting time containers dynamically with JavaScript.

The FakeSmile project has also experimented with SMIL Timesheets, but this implementation is focused on SVG animations and could not be easily reused in a broader scope.

For these reasons, we felt it was preferable to start a new development. After all, our implementation is only about 2000 lines of code, and the whole engine is less than 10 Kbytes in the minified/gzipped version. Technically speaking, the timesheet scheduler is very modular by design:

- Each node declared in the timesheet as a time container has its own clock, methods, properties and event handlers.
- Each time container parses its own descendants (time nodes) and pre-calculates the begin/end time values according to its temporal behavior: `seq`, `par` or `excl`.
- All time containers expose a significant part of the HTMLMediaElement API (which is exposed by the `audio` and `video` elements): web developers can control SMIL time containers with the usual `.play()` / `.pause()` methods, check the time with the `.currentTime` property and register to standard `timeupdate` DOM events.

To put it another way, we wanted the timesheets.js library to be more than just an implementation of the SMIL Timing and Synchronization module. It actually offers a declarative framework for web-based multimedia applications, which can easily be extended to fit specific needs (see Section 5).

It is worth to mention that, as SMIL Timesheets and SMIL Timing are not intended only for HTML documents, the timesheets.js library can be used with SVG content too, and therefore with compound documents. This allows synchronized multimedia applications to include vector graphics in addition to the usual HTML content, and to use time constraints within drawings and between (parts) of drawings and other parts of a HTML document.

4.3 A Basic Example

As an example, here is the very simple case of a rotating banner where three images are displayed one after another:

```
<script type="text/javascript" src="timesheets.js"/>
<div smil:timeContainer = "seq"
     smil:timeAction    = "display"
     smil:repeatCount   = "indefinite">
  <img smil:dur="3s" src="image1.png"/>
  <img smil:dur="3s" src="image2.png"/>
  <img smil:dur="3s" src="image3.png"/>
</div>
```

- The `smil:timeContainer` attribute turns the `div` element into a SMIL time container. Value `seq` defines a sequence in which elements play one after the other.
- The `smil:timeAction` attribute defines *how* the element is to be activated. In this case, the `display` CSS property is set to `block` when the element is active, `none` otherwise. The same mechanism can be used to trigger CSS transitions and animations.
- The `smil:repeatCount` attribute indicates the number of iterations.
- The `smil:dur` attribute specifies the duration of the element.

As a result, the three images are displayed one after the other, each one during 3 seconds, and this is repeated indefinitely, thus creating a rotating banner. The same result may be achieved with an external timesheet, clearly separating timing from content. Here is an equivalent markup:

```
<script type="text/javascript" src="timesheets.js"/>
<link href="banner.smil" rel="timesheet"
      type="application/smil+xml"/>
<div id="banner">
  <img src="image1.png"/>
  <img src="image2.png"/>
  <img src="image3.png"/>
</div>
```

where the external timesheet `banner.smil` contains:

```
<?xml version="1.0" encoding="UTF-8"?>
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  <seq repeatCount="indefinite">
    <item select="#banner img" dur="3s"/>
  </seq>
</timesheet>
```

Attribute `select` of `item` performs a `querySelectorAll()` action: for each DOM node that is matched by the `#banner img` selector, a SMIL item is created. This allows the same timesheet to be reused for several HTML pages: the SMIL markup above always works whatever the number of images in the banner.

4.4 Supported SMIL Features

Our timesheet scheduler supports a significant subset of both SMIL Timing and SMIL Timesheets (see Figure 5). As the same parser is used for both inline timing and timesheets (internal or external), a few SMIL Timing features are also supported in timesheets, and vice-versa.

The `timeContainer` and `timeAction` attributes are the two "integration attributes", as mentioned in the SMIL Timing specification. However, the SMIL Timesheets draft does not mention any `timeAction` attribute. This attribute is essential because it specifies *how* an element is activated in terms of CSS properties.

The `begin` and `end` attributes are supported with two restrictions: only positive values are taken into account; all time formats and event-values (e.g. "button.click") are supported, but there is no support for mixed time *and* event-value yet (e.g. "button.click+5s").

The `item` element and its `select` attribute are very specific to SMIL Timesheets and are thus ignored by our implementation in inline timing markup. On the other hand, the `first`, `prev`, `next`, and `last` attributes, which have been proposed by the SMIL Timesheets specification to control `excl` containers easily (for "lazy user interaction"), do make sense in an inline timing context, and are therefore supported.

4.5 Proposal: Transition Triggers

Timesheets.js fully supports the SMIL `timeAction` attribute, as defined in the SMIL Timing recommendation.[4] To get a sharper control on the way elements are activated, our timesheet scheduler sets a 3-state `smil` custom attribute to targeted elements, containing values `idle` | `active` | `done`, before | during | after element activation respectively.

This attribute can be used in CSS selectors to specify asymmetric transitions like in the following example which defines a carousel effect:

```
div[smil=idle] { /* state before transition */
  opacity: 0;
  transform: scale(0.3) translate(+200%);
}
div[smil=done] { /* state after transition */
  opacity: 0;
  transform: scale(0.3) translate(-200%);
}
div[smil=active] { /* state when active */
  opacity: 1;
  /* "transform: none;" is implicit */
}
```

Setting a custom attribute is a working solution, but a more satisfying solution would be to define SMIL-specific pseudo-classes in CSS.

4.6 Proposal: mediaSync Attribute

The SMIL Timesheets draft does not mention any way to synchronize explicitly a time container with a continuous media. The SMIL

[4] http://www.w3.org/TR/SMIL3/smil-timing.html#Timing-timeActionAttribute

Node	SMIL Timing	SMIL Timesheets	timesheets.js
timeContainer	+	+	+
timeAction	+	-	+
begin, end	+	+	+/-
dur	+	+	+
fill, endSync	+	+	+/-
repeatDur, repeatCount	+	+	+/-
item, select	N/A	+	+
first, prev, next, last	-	+	+

Figure 5: SMIL Timing and Timesheets Support in timesheets.js

Timing recommendation defines a boolean `syncMaster` attribute on media elements and time containers, that forces other elements in the time container to synchronize their playback to this element.[5] However, as the `audio` and `video` elements do not exist in SMIL timesheets, this `syncMaster` attribute is usable only with inline markup, and not with timesheets.

In order to define the same synchronization feature with timesheets we have introduced the `mediaSync` attribute, that can be used either with inline markup or within timesheets, which refers to a continuous media element through a CSS selector (in order to be consistent with the timesheet-specific `select` attribute).

The markup below is an example of a captioned movie that uses the `mediaSync` attribute and inline timing markup: each caption is an HTML paragraph synchronized with the `video` element.

```
<video src="myvideo.webm" />
<div smil:timeContainer = "excl"
     smil:timeAction    = "display"
     smil:mediaSync     = "video">
  <p smil:begin="0:00.00" id="intro">
    Title <br />
    by <a href="http://homepage/">Director</a>
  </p>
  <p smil:begin="0:04.93" smil:end="0:10">...</p>
  <p smil:begin="0:11.14">                ...</p>
  ...
  <p smil:begin="1:05.00" id="conclusion">...</p>
</div>
```

The timesheet scheduler parses the value of the `mediaSync` attribute and performs a `querySelector()` on its value. As there is only one `video` element in this page, value `video` is enough. This could also be done with `syncMaster`, but the main point is that the proposed markup is suitable for timesheets as well. Another benefit is that the `video` element does not have to be nested in the time container, which helps separate the content from the timing logic.

The SMIL Timing module includes a note about hyperlink implication on the `seq` and `excl` time containers.[6] It allows links to activate a particular time node in these time containers. URI `http://website.tld/page.html#conclusion`, for instance, refers to the element with a `conclusion` id in the above example and sets the video playback to the corresponding time value (1:05.00 in this case). This is an easy way to create temporal pointers.

5. EXTENSIBILITY

A declarative language describing the time structure and user interactions may be enough for most cases, but in more complex scenarios, an imperative language like JavaScript is necessary. Instead of having to choose between JavaScript and SMIL Timing, our implementation allows developers to define the main time structure declaratively and to extend it with JavaScript code when necessary.

5.1 DOM Event Listeners

As mentioned in the SMIL Timing module, SMIL target nodes fire `begin` and `end` DOM events when activated and deactivated, respectively. Web authors are used to set DOM event listeners to trigger specific actions. These `begin` and `end` events are already well known to SVG authors who use declarative SVG animations. Note that these `begin` and `end` events can also be used as event values in the `begin` and `end` SMIL attributes – again, like in declarative SVG animations.

5.2 JavaScript API

In our implementation, SMIL time containers can be controlled dynamically through two kinds of JavaScript APIs:

- `seq` and `excl` containers expose the same API as the HTML `select` element, i.e. mainly the `selectedIndex` property and the `onchange` DOM event.
- all time containers expose a significant subset of the HTMLMediaElement API (same as `audio` and `video` elements in HTML5), e.g.: `.currentTime`, `.duration` properties, `.play()`, `.pause()` methods, `timeupdate`, `playing`, `paused` DOM events.

We rely on existing web APIs wherever it makes sense. The `seq` and `excl` containers can be seen as HTML block level `select` elements, and all SMIL containers can be seen as general media elements – especially when synchronized with an `audio` or `video` element.

Besides, the SMIL Timing implementation comes with a JavaScript API that can be used to retrieve or create SMIL time containers dynamically.

[5] http://www.w3.org/TR/SMIL3/smil-timing.html#Timing-ControllingRuntimeSync

[6] http://www.w3.org/TR/SMIL3/smil-timing.html#Timing-HyperlinkImplicationsOnSeqExcl

5.3 Custom Timing Attributes

To keep the benefit of a declarative approach the timesheet scheduler can also be dynamically extended to support custom timing attributes that are too specific to be addressed by the SMIL specification. Timesheets.js provides a way to parse such custom attributes when time containers are initialized. Every custom attribute can be defined by a JavaScript file; we introduce two new attributes, `navigation` and `controls`, which are defined by two libraries.

5.3.1 Example: "navigation"

When using SMIL Timing for interactive presentations, a simple and common case is to ease navigation within the main time container. We are proposing a non-standard `navigation` attribute for this, with the following values:

arrows lets arrow keys control the execution of the time container: left/right to select the previous/next time node, up to reset the current time node, and down to emulate a mouse click on the current time node target;

click detects mouse clicks on the time container: the left/ middle buttons select the next/previous time node;

scroll selects the previous/next time node on mouse scroll;

hash updates the fragment identifier (`#id`) in the URI when a time node target has an `id` attribute.

A single `navigation` attribute may have multiple values (which must be separated by semi-colons). For instance, `navigation="arrows; hash;"` activates the arrow-key navigation mode and updates the fragment identifier every time a time node target has an `id` attribute. When this JavaScript extension is loaded, all time containers are checked and mouse/keyboard event listeners are dynamically attached to the target time containers.

Though the navigation extension has proven to be useful for slide shows,[7] it is mainly proposed as a simple code base (less than 150 significant lines of code) for developers intending to write their own custom timing features.

5.3.2 Example: "controls"

Another very frequent need is a user interface for handling time containers: as modern web browsers provide native controls for continuous media, we offer similar but richer controls that take advantage of time container features. Such time controllers typically include (see bottom of Figure 1): a play/pause toggle button; first/prev/next/last buttons (sequential access); a table of contents, i.e. a nested list of links pointing to time nodes (direct access); a graphical timeline, either continuous (like for usual audio/video players) or segmented, where each segment is a link to a time node.

Instead of providing a single UI component, and in order to keep the flexibility of SMIL time containers, these controller elements are defined by a microformat: each UI component has a class name that is used both to define its presentation (in CSS) and its behavior (in JavaScript).

To highlight the current active time node (e.g. a timeline segment or table of contents item) when the time container is running, a time container is created dynamically with a `mediaSync` attribute pointing to the main time container.

A typical use case of such a time controller is a recorded talk[8] where slides are synchronized with the audio track. The timeline may be segmented to display the corresponding slide heading when the mouse hovers over it.

6. USING TIMESHEETS

In this section, we present real applications that were developed using the technology discussed above.

6.1 Media-Driven Applications

A captioned video[9] is a simple example of the category of applications presented in section 2.1. The whole application is implemented in a single HTML5 file, whose content is basically the same as in section 4.6.

In that example, the HTML `div` element is a time container for the video and all captions. Each caption is a HTML `p` element which contains the time when it must be displayed (`smil:begin` attribute) relatively to the beginning of the video. If a caption must disappear exactly when the next one is displayed, that is enough, as the container is `excl`usive, but if it must disappear earlier, its end date has to be explicitly stated.

The full HTML5 language may be used in each caption. The first caption takes advantage of this feature to add a link to the home page of the director and to split the text into two lines. CSS may also be used to select a particular font, its size and color, or to set the position of captions within the `div` element, i.e. over the movie.

6.2 Event-Driven Applications

We have worked with INA, the French national archive of audiovisual, to publish on the web archived radio programs enhanced with associated material.[10] At first glance, the time structure could look similar to the captioned video example discussed above, but the goal here is not only to synchronize pictures or text with the audio content. The objective is really to create an application where the user receives help for moving across the audio recording and is free to choose the associated information s/he wants, which could be multimedia too, with other audio recordings, for example. This is an example of an event-driven application as defined in section 2.2.

In this application (see screenshot on Figure 6), all the content is specified by a HTML5 document, while timing and user interactions are defined in a separate timesheet that refers to elements in the HTML5 file through attributes `select`, `mediaSync` and `controls`, as can be seen in this simplified version:

```
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  <!-- slide show / main section -->
  <excl timeAction="display" mediaSync="#main"
        controls="#timeController" dur="20:47">
    <item select="#section1" begin="00:00.000"/>
    <item select="#section2" begin="01:12.120"/>
    <item select="#section3" begin="04:41.742"/>
```

[7] see http://wam.inrialpes.fr/timesheets/slideshows/slidy

[8] see http://wam.inrialpes.fr/timesheets/slideshows/audio

[9] see http://wam.inrialpes.fr/timesheets/annotations/video

[10] see http://wam.inrialpes.fr/timesheets/public/webRadio/

Figure 6: Enhanced radio program

```
  </excl>
  <!-- extra material: multimedia pages -->
  <excl>
    <item select="#extra2"
          begin="open2.click; toc-extra2.click"
          end="close2.click; section2.end"/>
    <item select="#extra3"
          begin="open3.click; toc-extra3.click"
          end="close3.click; section3.end"/>
  </excl>
  <!-- extra material: audio -->
  <par mediaSync="#track2a" controls="#timeline2a"
       dur="2:24.039"/>
  <par mediaSync="#track2b" controls="#timeline2b"
       dur="3:59.928"/>
  <!-- extra material: rotating pictures -->
  <seq timeAction="display"
       repeatCount="indefinite">
    <item select="#extra4 img" dur="3s"/>
  </seq>
</timesheet>
```

In this timesheet, the first `excl` element specifies a slide show synchronized with the audio recording (the `audio` element identified by the `main` id in the HTML5 file). The time structure of this part is similar to the captioned video example. Elements identified as `section`*n* are `div`isions in the HTML5 file that contain text and pictures. They define the main slides of the slide show.

The second `excl` element allows the user to display additional slides (identified as `extra`*n* in the HTML5 file) on request. This is achieved through buttons included in the main slides (ids `open`*n* refer to `button` elements which are part of the main slides). The right part of Figure 6 shows such a button. Similarly, additional slides contain buttons (ids `close`*n*) the user can click for closing them.

The element identified as `timeController` in the HTML5 file and referred by the first `excl` element specifies, in addition to the usual controls for an audio stream, a table of contents that the user can display with a button. The items of this table of contents have ids `toc-extra`*n*. Clicking them not only skips to the corresponding section of the main audio track, but also displays the corresponding additional slide, as specified in the second `excl` element.

The role of the `par` elements is to associate controls with the additional audio tracks, which are part of an additional slide in the HTML5 file. These audio tracks are activated as soon as the user opens the additional slide that contains them. S/he is then free to use the controls for listening to one of these oral comments.

Finally, the `seq` element at the end of the sample code specifies that all images contained in the `extra4` additional slide must be presented one after the other, each during 3 seconds, repeatedly. This automatic picture show starts as soon as the user clicks the button displaying the fourth additional slide.

7. CONCLUSION

Because most web developers are used to the HTML-CSS-JS triple, we have extended its capabilities with timing features borrowed from the SMIL language for enabling multimedia web applications. This approach fully preserves the declarative nature of web formats and their structural model for most applications, while scripts are still available to cover the most complex cases.

Developers do not have to choose between a logical and a temporal structure. Both kinds of structure can co-exist in the same document. In addition, temporal references can be used easily. DOM events, including SMIL time events, are available, which allows complex temporal and interaction behaviors to be defined. Content can be dynamically generated when necessary (e.g. `timeController` structures). Rich media navigation, as well as table of contents navigation, can be provided thanks to additional libraries. Content reusability and multi-device rendering are possible (the videotaped talk of Figure 1, for instance, runs on the iPhone and the iPad too). Components defining common behavior can be used in a declarative way.

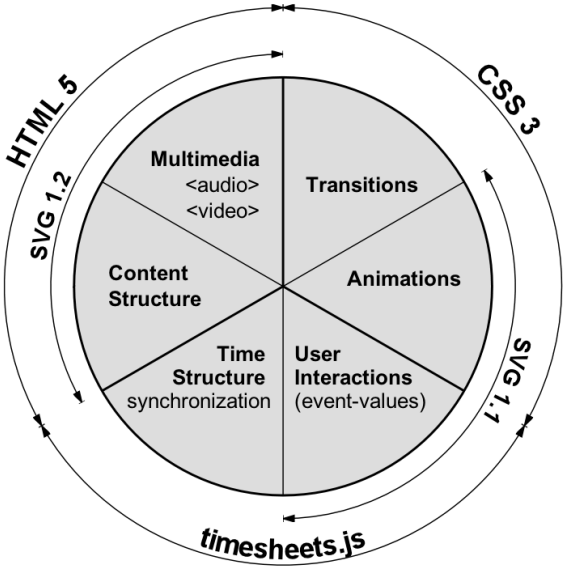

Figure 7: HTML5, CSS3, SVG and timesheets.js

The approach developed here and the timesheets.js library are not restricted to HTML5 documents. They can be used in graphic applications based on SVG, or in compound documents mixing HTML5 and SVG. This is illustrated by Figure 7 (a timed version is also available on-line[11]). The gray disc represents the main features

[11] http://wam.inrialpes.fr/timesheets/slideshows/svg.html

expected from web multimedia languages, which are covered by SMIL to a large extent; the curved arrows show how they have been adopted by (or transposed in) other web languages. Timesheets.js closes the loop, thus bringing the full power of SMIL to usual web compound documents that can be rendered natively in the browser.

Several multimedia applications based on timesheets.js are presented in this paper. These are not just examples; most of them are deployed on varions web sites. As an example, multiple videotaped talks such as the one of Figure 1 are published by ENS-Lyon.[12]

A further step would be to introduce the features implemented by timesheets.js directly in the document languages of Figure 7. Most timing attributes are already part of the future SVG 2.0 language. Adding three more attributes, `mediaSync`, `timeAction` and `timeContainer`, would enable SVG applications such as slide shows or media annotation. This would also help to put temporal structures on SVG animations. There would be more work for HTML5, but reserving a prefix such as `smil-` could facilitate the integration of timing attributes later. CSS3 already includes transitions and animations, but it would be easier if the three states (idle, active, done) where available as pseudo-classes in selectors.

Because they are so close to usual web documents, multimedia documents based on the timesheets.js library may be developed in the same way as any web page, and can even be hand-coded. But, they could also benefit from specialized tools that would help developers to handle time information in documents. Now that a robust scheduler engine is available, the next step in our work is to develop specialized authoring tools.

8. ACKNOWLEDGEMENTS

The research presented in this paper was conducted in the C2M project, funded by ANR, the French National Research Agency, under its CONTINT 2009 programme.

The authors are grateful to Dominique Saint-Martin from INA-GRM for providing the web radio application and to Gérard Vidal from ENS Lyon for the videotaped talk application. Both have provided real use cases and valuable feedback that were key in testing and validating timesheets.js.

9. REFERENCES

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26:832–843, Nov. 1983.

[2] Ambulant. Ambulant open SMIL player, http://www.ambulantplayer.org/.

[3] O. Aubert, P.-A. Champin, Y. Prié, and B. Richard. Canonical processes in active reading and hypervideo production. *Multimedia Systems*, 14:427–433, 2008.

[4] D. Bulterman et al. *Synchronized Multimedia Integration Language (SMIL 3.0)*. W3C Recommendation, Dec. 2008.

[5] F. Cazenave. A declarative approach for HTML Timing using SMIL Timesheets, http://wam.inrialpes.fr/timesheets/, 2011.

[6] R. Deltour and C. Roisin. The LimSee3 multimedia authoring model. In D. Brailsford, editor, *Proceedings of the 2006 ACM Symposium on Document Engineering, DocEng 2006*, pages 173–175. ACM Press, Oct. 2006.

[7] ISO. Iso/iec 10744:1992 - information technology – hypermedia/time-based structuring language (hytime).

[8] J. Jansen and D. Bulterman. SMIL State: an architecture and implementation for adaptive time-based web applications. *Multimedia Tools and Applications*, 43:203–224, 2009.

[9] P. King, P. Schmitz, and S. Thompson. 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, pages 57–66, New York, NY, USA, 2004. ACM.

[10] J. Mikác, C. Roisin, and B. Le Duc. An export architecture for a multimedia authoring environment. In *Proceeding of the eighth ACM Symposium on Document Engineering*, DocEng '08, pages 28–31, New York, NY, USA, 2008. ACM.

[11] MIT. Infinite history, http://mit150.mit.edu/infinite-history, 2011.

[12] K. Pihkala and P. Vuorimaa. Nine methods to extend SMIL for multimedia applications. *Multimedia Tools and Applications*, 28:51–67, 2006.

[13] D. Raggett. HTML Slidy: Slide shows in HTML and XHTML, http://www.w3.org/talks/tools/slidy2/, 2005.

[14] L. F. Soares, R. F. Rodrigues, R. Cerqueira, and S. D. Barbosa. Variable and state handling in NCL. *Multimedia Tools and Applications*, 50:465–489, Dec. 2010.

[15] L. Villard. Authoring transformations by direct manipulation for adaptable multimedia presentations. In *Proceedings of the 2001 ACM Symposium on Document Engineering*, DocEng '01, pages 125–134, New York, NY, USA, 2001. ACM.

[16] P. Vuorimaa. Timesheets JavaScript Engine, http://www.tml.tkk.fi/~pv/timesheets/, 2007.

[17] P. Vuorimaa, D. Bulterman, and P. Cesar. *SMIL Timesheets 1.0*. W3C Working Draft, Jan. 2008.

[12] see http://html5.ens-lyon.fr/