Enhancing an XML Publishing Workflow for Web Multimedia

Cécile Roisin, Fabien Cazenave, Vincent Quint, Ludovic Gaillard, Dominique Saint-Martin

Enhancing an XML Publishing Workflow for Web Multimedia

Cécile Roisin, Fabien Cazenave, Vincent Quint, Ludovic Gaillard, Dominique Saint-Martin

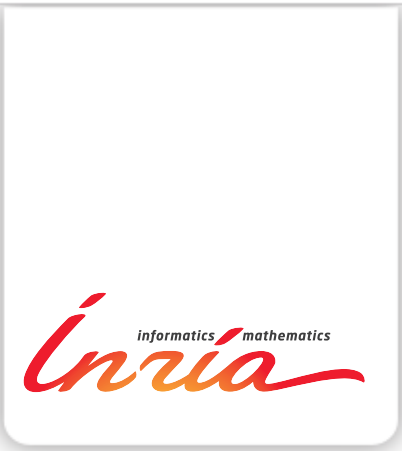
RESEARCH REPORT

N° 7836

July 2011

Project-Team WAM

ISSN 0249-6399 ISRN INRIA/RR--7836--FR+ENG



Enhancing an XML Publishing Workflow for Web Multimedia

Cécile Roisin*, Fabien Cazenave†, Vincent Quint†, Ludovic Gaillard‡, Dominique Saint-Martin§

Project-Team WAM

Research Report n° 7836 — July 2011 — 25 pages

Abstract: A challenge for the C2M project is to develop a document workflow for authoring and producing web multimedia documents with a special attention for the aesthetic quality of the final rendering. In this report, we discuss the requirements for advanced multimedia authoring services from a document engineering point of view. Our approach consists in providing a post-editing service to allow authors to adjust their multimedia presentation directly on the final form of the document. The first step of the proposal is to provide a web rendering engine based on the latest advances in web standards. For that purpose, we have developed timesheets.js, a JavaScript library for publishing multimedia web documents. It takes advantage of the new features of the HTML5 and CSS3 web standards and enables synchronization of multimedia contents. The second step consists in designing web-aware authoring tools based on this library, thus providing authors with direct editing services for producing high quality multimedia documents.

Key-words: multimedia documents, publishing workflow, web, authoring, web standards, HTML5, CSS3

* Université Pierre Mendès-France and INRIA
† INRIA
‡ Ina
§ Ina GRM

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Extension d'une chaîne de publication XML pour le multimédia sur le web

Résumé : Un des objectifs du projet C2M est de développer une chaîne éditoriale pour la création et la publication de documents multimédia, en portant une attention particulière à la qualité esthétique du résultat final. Dans ce rapport, nous identifions les prérequis pour un service de création de documents multimédia, en nous plaçant dans la perspective de l'ingénierie documentaire. Notre approche consiste à fournir un service de post-édition qui permette aux auteurs d'ajuster les détails de leurs présentations multimédia directement sur la forme finale. La première étape consiste à fournir un moteur de rendu fondé sur les récents progrès des standards web. Dans cette optique, nous avons développé timesheets.js, une bibliothèque JavaScript pour la publication de documents multimédia sur le web. Elle s'appuie sur les nouvelles fonctionalités des langages HTML5 et CSS3 pour permettre la synchronisation des contenus multimédia. La seconde étape consiste à créer des outils d'édition adaptés au web en utilisant cette bibliothèque, fournissant ainsi aux auteurs les moyens d'éditer directement les résultats de la chaîne éditoriale, pour produire des documents multimédia de grande qualité.

Mots-clés : documents multimédia, chaîne éditoriale, web, édition, standards web, HTML5, CSS3

Preface

The work presented in this paper was conducted in the C2M project,[1] funded by the French National Research Agency (ANR) under its CONTINT 2009 programme. This report, which is a deliverable of the C2M Research Task, is also available on the C2M web site.[2]

[1] `http://scenari.utc.fr/c2m/co/00accueil.html`
[2] `http://www.utc.fr/ics/c2m/DOCS/L1g/pdf/c2m_L1g_20110706.pdf`

1 Introduction

The web is nowadays the natural way for accessing information, including multimedia content. Coping with multimedia resources is a major requirement in most web applications for two reasons: first, applications have to handle more and more multimedia content (pictures, video, audio, animations), and second, the use of rich media content is a playful way to provide intuitive and efficient access to information.

A challenge is to propose industrial solutions that allow multimedia developers to achieve mass production with high quality results. Document engineering techniques (XML technologies, platforms for document production) have been used for a long time to address these needs. However when dealing with multimedia content, automatic tools are not always sufficient for generating quality documents; manual editing of documents in their publishing format is often necessary to tune a number of details. Moreover, as a pure wysiwyg paradigm is not relevant for time-based and interactive content, users require means that help them to rapidly experiment the rendering of their authoring actions.

Our approach to this issue consists in providing a post-editing service that allows authors to adjust their presentations directly on the final form of the document. The first step of the proposal is to provide a web rendering engine using the latest advances in web standards. We have developed Timesheets.js, a JavaScript library for publishing multimedia web documents that takes advantage of the new features of HTML5 and CSS3, and makes it possible to synchronize multimedia contents. The initial phases of this work have been described in the first intermediate C2M report [9] and the main results are published in [8] and [7]. The second step is the design and development of web-aware authoring tools based on this library in order to provide authors with direct editing services to produce high quality documents while preserving the advantages of using an XML production workflow.

The rest of this report is organized as follows: an overview of different solutions proposed so far is presented in the next section. Then, the principles of the multimedia production workflow are explained, illustrated by a use case developed in the context of the C2M project. The first step of our web multimedia standard-based application is presented in section 4. Section 5 is devoted to the multimedia authoring features we propose. Section 6 discusses new directions to progress in the integration of such authoring services in publishing workflows, taking into account emerging specifications from W3C, such a CSS Regions.

2 State of the Art

In this overview of previous work in multimedia document production, we consider three key topics: XML document workflows, layout engines, and multimedia document engineering.

2.1 C2M workshop on multimedia authoring

In the context of the C2M project, a research workshop was held in January 2010 in Grenoble. In this early phase of the project, the goal was to allow several teams to present their on-going activities on the three topics we wanted to investigate. Here is a short synthesis of the main presentations done at the workshop and the recent evolutions of the presented work.

2.1.1 Direct web authoring using templates

Amaya [2] is an open source stand-alone web editor developed jointly by INRIA and W3C to ease editing, publishing and sharing complex web pages containing text (in HTML, XHTML), graphics (in SVG) and mathematical expressions (in MathML), with style sheets (in CSS). A recent evolution of the editor was to introduce a new generation of document templates. When producing a specific type of web document, authors are faced with all the possibilities provided by the (X)HTML language, and they have to make a number of difficult decisions. A templating language called XTiger (eXtensible Templates for Interactive Guided Editing of Resources) was designed to tackle this problem by specifying how the document language has to be used for representing a certain type of document [13]. The idea behind XTiger is to:

- use (X)HTML as the basic document format: it is indeed possible to exploit the full potential of the semantic tags of (X)HTML (and now HTML5) and to use microformats to add semantics to web pages, while taking advantage of the existing (X)HTML infrastructure.
- constrain the way an author uses (X)HTML to produce a specific type of document.

A XTiger template is a skeleton representing a given type of document, expressed in the format of the final documents to be produced ((X)HTML). A template also includes some statements, expressed in the XTiger language, that specify how an instance based on this minimal document can evolve and grow to make a full, well structured document. The XTiger language has been used in Amaya in order to demonstrate that it is possible to combine full web authoring with constraint-based editing to produce valid and well structured documents [25].

2.1.2 XTiger XML and the AXEL editing library

The work on XTiger templates has developed in two complementary directions:

- To push it further as a tool for the web, the template-driven editor was reimplemented as a pure web application, i.e. an application that runs in the browser, instead of a dedicated, stand-alone editor that has to be installed.
- To broaden it scope, the XTiger language was also extended in such a way it could be used for producing documents not only in the (X)HTML language, but also in any XML language. This extended version is called XTiger XML.

This combination of XTiger XML and a JavaScript library called AXEL [28] (that implements an editor as a web application) allows various XML contents to be authored within any web browser (see Figure 1 extracted from [29]). Indeed The XTiger XML language and its implementation in the AXEL library make it easy to create and update XML data and documents on the web, without any specific tool. The editing engine provides a simple and intuitive user interface, which allows any web user to create and edit XML content. Compared to existing XML editors which require the use of schemas, and transformation languages for web publication, the use of document templates makes XTiger XML easier to learn and to manipulate. This system provides a template-driven editor that allows any web user to easily enter structured content.

In addition, a conversion tool was developed as an XSLT transformation, to generate an XML Schema from an XTiger XML template. This allows XML processing tools and applications to safely process this content [23].

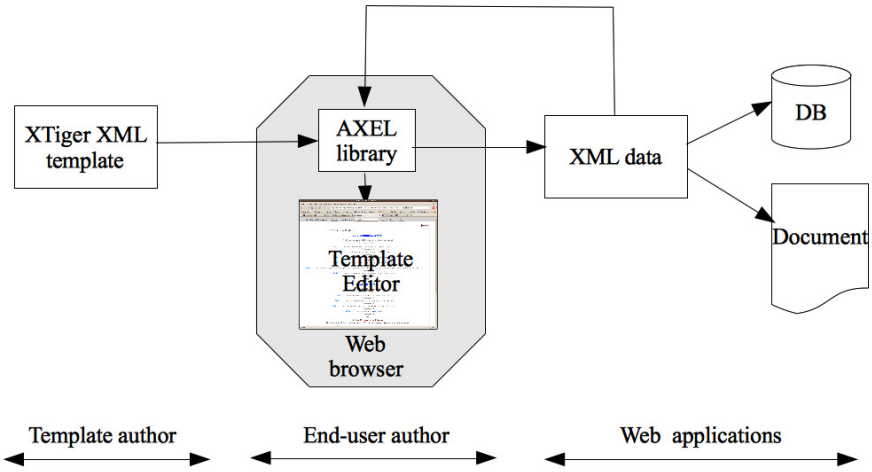


Figure 1: Architecture of XTiger+AXEL on-line document tool

2.1.3 The Kolekti document production system

Kolekti [3] is a web-based framework that provides tools for the design and production of multi-targeted and personalized documents. It proposes a "topic-map" oriented document architecture: a document results in the assembling of content modules (topics) organized as a hierarchy defined in a frame (table of content). Content reuse and collaboration is possible at the module level because a module can be referenced by several frames and is accessed through the WebDav protocol. The document model includes conditions and variables definitions that can be added to the content module in order to specify multi-target outputs. During the publishing phase, the Kolekti processor analyzes these definitions and generates the adapted final document.

The server side of the Kolekti web application is written in Python and uses the WebDav protocol. The client side takes advantages of the Ajax technology to provide fluent update of the displayed content.

This application uses exclusively standard formats and standard technologies, and therefore lets users free to choose among a wide range of applications (for content authoring, for instance) and gives them a reasonable confidence in long-term availability of the produced data.

2.1.4 SCENARI

The discussions at the workshop about the SCENARI platform [27] were devoted to the problem of generating high quality documents from the authoring workflow: how to formally specify rules that could be expressed by a person. Typical examples are image balancing in a page, or content splitting into pages/column. This problem is a long standing issue in electronic publishing (see for instance [21]) and is still open (see next section for recent research activity on that subject).

2.2 XML-Based Editorial and Production Workflows

An XML production workflow aims at providing a sound separation of content and logical structure from presentation information. That separation is considered as the key paradigm in document engineering systems. The most widely spread professional XML authoring tools, such as Framemaker [1] are based on this paradigm. SCENARI[27] belongs to this category and adds a set of components for the definition of document models and their authoring and publishing formats. Integrating this concept in web tools as been explored in [24]. A related problem is the production of documents with variable content such as those targeted by the Document Product Line DPL system [22]).

General-purpose XML editing tools are usually too complex to be used by non-professional users [1]. This inherent complexity has been generally got round with template-based solutions (such as provided by SCENARI). Others make use of conversion techniques applied to (clean) content edited with text editors such as MS Word or Open Office in order to generate the XML content [16]. In addition (and complementary to the template-based approach), it is also possible to propose solutions that take advantage of web technologies. Indeed, we can identify a common evolution in recent systems: providing direct access to authoring services through a web interface. This trend takes the opposite approach from standard XML software (edit in XML and publish in HTML). AXEL [29] is a typical example of this trend. Similarly, Maxwell *et al.* [19] have developed a transformation process from XHTML to ICML, the publicly available format of Adobe InDesign, in order to allow InDesign to be used for adjusting layout and pagination.

2.3 Layout Engines

Numerous approaches to heterogeneous hierarchical layout have been studied. Most of them follow principles that have been stated thanks to the long experience with structured documents:

- separating competencies and skills between designers (who could provide the initial design) and non professional users.
- a formatting process that insures aesthetic layout while making as little changes as possible
- efficient UI features to handle complex authoring requests.

Among complex formatting problems that are still open, we can identify the following:

- Aesthetic criteria specifications and their implementation in a formatting system require sound modeling to cope with incompatibilities between criteria. In particular, a simple bottom-up process is not sufficient, because layout constraints may exist at each level. For instance, Di Iorio *et al.* [11] discuss hierarchical layout defined at a high level, with layout options searched by higher-level agents, attempting to satisfy declared desiderata; a formatting engine transforms topological layouts into actual PDF files.
- Coping with variable data documents with high quality rendering requires complex formatting and transformation strategies. In [18], Document Description

Framework (DDF) is proposed to allow declared constraints and pagination to be supported thanks to an extensible XML/SVG hierarchical layout structure. Later, these authors used pre-evaluation of invariant sections to increase efficiency of final document generation [17].

As a conclusion, we can cite what has been stated by Hurst, Li and Marriott in their recent article [15] where they have done a comprehensive review of automatic document formatting techniques. While solutions to micro-typographical issues are efficient, it is not the case for macro-typographical issues. Moreover "solving the entire problem seems prohibitively expensive". Finally an open issue they sketch is the need of "better support for authoring documents that can take advantage of better automatic layout". This last issue is one of the objectives we have addressed in the C2M project.

2.4 Authoring Multimedia Documents

Multimedia document engineering is less developed due to the difficulty to cope with the additional time dimension inherent to multimedia content. Authors often have to become developers to be able to define how media elements are synchronized, in addition to the other authoring tasks (content structuring, layout and navigation definitions). To cope with such an unfriendly approach to authoring, two complementary directions have been studied: (1) providing users with rich user interfaces that help them to handle the time dimension of their documents, and (2) restricting complexity through a template-based approach.

The most popular way to visualize time is to use its spatial representation called "timeline". Authors can act on timelines by direct manipulation. The combination of this feature with structure and spatial information can be handled with multi-view GUIs. For instance, [33] and [5] hide the intrinsic complexity of SMIL [4] behind such advanced GUIs.

The last-mentioned tools offer a wide set of authoring features because they present users with the whole expressive power of the underlying multimedia language (SMIL, XMT or NCL). This is not required most of the time, and several tools propose a template-based approach to restrict this expressive power and therefore simplify the authoring process. LazyCut [14], LimSee3 [20] and XTemplates [12] (for the NCL2.0 connector-based language) are template-based authoring tools. This approach paves the way for producing a more engineering-oriented process for multimedia document production.

Therefore, using an XML environment is a natural way to put the focus on the logical dimension of documents when creating multimedia content [10]: the document is structured in XML to encode its logical organization, and time relations are expressed as part of this structure. However, this approach raises two main difficulties:

- As it requires some export mechanism to produce documents in a format that can be published on the web, i.e. accepted by a browser, it does not allow updates and adjustments to be performed directly on the resulting documents. A way to cope with this drawback is to use incremental transformations to reflect changes in the original XML structure, but this technique has some limitations [30].

- The distance between the document state in the authoring phase and its rendition state in the production phase is too large. This prevents the author from performing efficient authoring actions. This also implies numerous round trips in the authoring process.

In C2M, we have worked to reduce these drawbacks through to a two-step approach of multimedia content production. The rest of the paper presents this work.

3 A Document Workflow for Multimedia Production

3.1 A Use Case: INA Webradio

In the context of the C2M project, we have worked with INA, the French national archive of audiovisual, to publish on the web archived radio programs enhanced with associated material. Such sources have been produced by GRM (Groupe de Recherches Musicales) and numerous examples can be found in the inagrm web site.[3]

A typical INA Webradio page[4] involves (see Figure 2):

- a rich audio player, i.e. a segmented timeline displaying a specific HTML fragment for each section of the audio track (bottom of Figure 2);
- buttons that users can click to display complementary content – possibly involving other multimedia sources (top right corner of Figure 2).

Figure 2: INA Webradio Application

The idea is to propose a visually enhanced experience of a radio program while allowing users to browse the content in a non-linear way, for instance with the table of contents (Sommaire) of Figure 2.

[3] `http://www.inagrm.com/grm-webradios`
[4] `http://wam.inrialpes.fr/timesheets/public/webRadio/`

3.2 Requirements for this Use Case

Report [26] identifies all the actors and their roles in the different steps of such an application. We recall here the main authoring requirements that have been identified in this document and that have been discussed during the C2M project.

There is a strong need for producing "in the large": INA wishes a larger production of Webradio programs; that implies the development of an industrial production process. As shown in the state of the art, the use of an XML production workflow seems to be the answer, but with an important risk of loosing the aesthetic quality of the multimedia documents produced. The main (non linear) operations identified for this authoring process are:

- Spatial organization of the content.
- Temporal positioning of that content, relativly to the master audio.
- Transitions definition between these illustrative elements. Transitions are set between segments of the linear audio progression in order to give the readers/auditors the correct narrative perception of the Webradio document. This authoring feature is therefore fundamental for the author.
- Navigation definition inside the whole Webradio document. Unlike a pure radio program where the listening time is completely defined by the program director, a Webradio program must propose several reading paths thanks to various navigation options (as illustrated in Figure 2). Moreover, the application should enable several progression models that could be mapped to the main Webradio structure.

Another need is related with the evolution of terminals and their usage: most users are now equipped not only with a PC but also with handheld devices such as tablets and mobile phones. Multimedia applications must then be available on all these devices too. The use of a smartphone is even more obvious for an application such as the Webradio, where audio is the main media.

Finally, users are now used to rich multimedia content with a great deal of interactivity and complex synchronized content. All these features must be handled by the author in order to capture the attention of the reader/player through the production of non-linear rhythms in the document progression (mixing types of transitions, various effects introduced in the display, etc.)

Therefore, while a template-based authoring process is recognized to be the most efficient way for producing Webradios (easiness, rapidity), a great deal of freedom together with an adapted UI must be given to the content author. The authors need:

- Direct and quick views of the resulting document: integrated previewing must be provided in the authoring tool.
- Direct manual adjustments of the temporal segmentation through an interactive timeline.
- A rich set of available transition models to create the wished progression from one shot to the other.

3.3 Document Processing Workflow

INA uses a SCENARI-based, XML publishing workflow to create multimedia documents. The SCENARI authoring chain[5] is used to split a continuous media object into several contiguous time segments, to associate an HTML fragment with each time segment, and finally to publish a dynamic multimedia document as a Flash object. Right now more than one hundred of Webradio programs have been produced. One of the main limitations of this document processing workflow, besides the usual Flash-related issues (accessibility, indexability, compatibility with mobile devices...), is that neither content authors nor web designers can efficiently control the presentation of the resulting multimedia documents.

The C2M project has been the opportunity to experiment a new workflow for better covering the previously mentioned requirements. Our approach is to keep the SCENARI content editor, but publish multimedia documents in HTML5 + CSS + SMIL Timesheets. The publishing engine is realized with our Timesheets.js library, which is presented in section 4.

The major benefit of this process is to obtain a clear separation of concerns:

- content: the whole content is expressed in HTML5. All HTML5 fragments are defined in SCENARI;
- synchronization: SMIL Timesheets [32] are used both to define the time segments of the main audio track and to describe user interactions;
- presentation: generic CSS style sheets are created by web designers for a consistent integration of these multimedia documents in the main website, and content authors can use specific style rules when necessary.

Moreover this separation provides a strong basis for adding interactive authoring features that allow authors to act on their documents in their final publishing format, thus covering the needs expressed in section 3.2. We are developing such an authoring tool that shortens the cycle between editing and rendering and enables fine-grained direct temporal and spatial adjustments. This second phase in the two-step authoring process is described in section 5.

The general idea here is to reconcile the efficiency of an XML publishing workflow with the flexibility of CSS style sheets, while allowing multimedia documents to be modified in a way that is familiar to web developers and authors.

4 The Timesheets.js Library

The SMIL language [4] has been available for a while for publishing multimedia applications on the web, but the lack of widely deployed tools has limited its impact. Among its most attractive capabilities are its timing and synchronization features, which can be combined with other document languages.

With the advance of HTML5, and notably its new graphic, audio and video contents, it is now possible to develop multimedia standard-based applications that can run natively in web browsers. The only missing pieces are the SMIL timing and synchronization features, that are not currently supported by browsers; but this problem can be solved by implementing these features in JavaScript. This is what we have done with the timesheets.js library.

[5] `http://scenari-platform.org/projects/scenari/en/pres/`

With this library, sophisticated multimedia applications can be developed in a completely declarative way, and based on languages that are widely known to web developers, such as HTML and CSS.

SMIL has been thought as a full specification language, describing all aspects of a multimedia document: content, presentation, synchronization, and interaction. However, the SMIL 3.0 Timing and Synchronization module[6] (a.k.a. SMIL Timing) is also designed to be integrated into other languages, thus bringing synchronization and user interaction features to otherwise a-temporal document languages. As specified in this module, timing can be inserted *in-line* in the markup of a static document thanks to two attributes for timing integration: `timeContainer` and `timeAction`.[7]

This is complemented by another W3C specification, SMIL Timesheets,[8] that allows the most significant SMIL timing features of a document to be gathered in external resources called *timesheets*, thus separating the timing and synchronization aspects from the host language, and allowing time behavior to be shared among several documents. To paraphrase the SMIL Timesheets specification, SMIL Timing and SMIL Timesheets can be seen as a temporal counterpart of inline style and external CSS stylesheets, respectively.

4.1 Using HTML5, CSS3 and SMIL Timing

We propose [6] to combine HTML5+CSS3 and SMIL Timing/Timesheets. We take advantage of the recent addition of new media objects such as audio and video to HTML, and new style properties such as animation and transition to CSS3. Adding SMIL Timing extends these multimedia features significantly. It allows, for instance, some discrete parts (text, images) of a HTML page to be synchronized declaratively with the continuous parts or with other discrete parts. This also allows user interactions to be specified in a purely declarative way.

Our approach can be summed up in three points:

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

With this approach, interactive multimedia web applications can be developed and published using only declarative languages. Although scripting remains an option for special cases, most applications can be created without a single line of JavaScript.

4.2 A Basic Example

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

[6] `http://www.w3.org/TR/smil/smil-timing.html`
[7] `http://www.w3.org/TR/SMIL3/smil-timing.html#q48`
[8] `http://www.w3.org/TR/timesheets/`

```
<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>
```

In this piece of HTML code, several in-line SMIL Timesheets attributes add a temporal behavior:

- The `smil:timeContainer` attribute turns the `div` element into a SMIL time container. Value `seq` defines a sequence in which child 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 `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. Here is an equivalent markup using an external timesheet:

```
<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 file `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>
```

Like in CSS, selectors are used to associate elements from the HTML document with time behaviors defined in the external timesheet. For instance, the `select` attribute of element `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.

4.3 Timesheets Engine

As SMIL Timing and SMIL Timesheets are not supported natively by web browsers, a JavaScript implementation of these specifications is required to make them available widely. We have developed timesheets.js[9] which is an open-source, cross-browser, dependency-free library that supports the common subset of the SMIL Timing and SMIL Timesheets specifications.

This implementation obviously relies on JavaScript, but as stated above, no specific JavaScript development is required from a web developer. When an application is running, some parts of it (HTML and CSS code) are executed natively by the browser, some other parts are executed by the browser's JavaScript engine.

Timesheets.js is not the first SMIL Timesheets engine running in the browser. Vuorimaa [31] has developed a Timesheets JavaScript Engine, but it was before the time of HTML5. Therefore, it can synchronize only discrete contents.

Our implementation is available in open source under the MIT license. It is rather compact (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 time container node 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: sequential, parallel or exclusive.
- All time containers expose a significant part of the HTMLMediaElement API (which is exposed by the `audio` and `video` elements of HTML5): 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.

4.4 Application: INA Webradio

The timesheet of the Webradio application presented in section 3 is given below, with some simplification:

```
<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"/>
  </excl>
  <!-- extra material: multimedia pages -->
  <excl>
    <item select="#extra2"
          begin="open2.click; toc-extra2.click"
          end="close2.click; section2.end"/>
```

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

```
    <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, not provided here). 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 in the HTML5 file). The right part of Figure 2 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 (see Figure 2). 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, like in section 4.2, 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.

4.5 Browser Support

These HTML5 + SMIL Timesheets multimedia documents are supported by all desktop browsers: either natively by all modern browsers implementing the `audio` and `video` tags of HTML5, or through a Flash plugin for Internet Explorer 6 to 8. In other words, the HTML5 version of these INA Webradio pages is supported by the same desktop browsers as the full-Flash version, though modern browsers bring a better user experience with the native multimedia player: better responsiveness and better CPU resource usage.

Unlike the Flash version, these HTML5 Webradio pages can be served to mobile devices: most smartphones support the `audio` and `video` tags natively and can play

multimedia resources without draining the battery. In addition, as the layout is defined by style sheets, a specific touchscreen layout can be served to mobile devices with a declarative CSS media query.

4.6 Extensibility: DOM events and JavaScript

Most of the synchronization logic and user interaction description can be defined with SMIL Timesheets. However, there are cases where a more specific dynamic interaction should be implemented. In these cases, some JavaScript code can be written by web developers, taking advantage of a few DOM events.

Each node fires `begin` and `end` DOM events when it is activated and deactivated by the timesheet scheduler, respectively. This allows some specific JavaScript code to be triggered easily – either with an `onbegin` / `onend` attribute in the HTML document, or with event listeners in the JavaScript code.

Timesheets.js also exposes an API that allows time containers to be created dynamically. This mechanism is used in the `timesheets-controls.js` companion library to display a segmented media timeline and to keep it synchronized with the main audio track.

5 Authoring the HTML5 Document

The availability of both a structured-based authoring platform (as provided by SCENARI) and a JavaScript scheduler for a browser-based execution of HTML5 multimedia content (as described in the previous section), leads to a new editing architecture that addresses the authoring needs presented in section 3.2. This section is devoted (1) to the presentation of direct authoring services for multimedia web content, and (2) to their integration into the document workflow.

5.1 Direct Editing

The main goal of this modular approach is to enable “wysiwyg”, fine-grain editing of the resulting multimedia documents.

With the initial document processing workflow, content editors could specify the semantic content for each audio section, but had no easy way to refine the resulting document: that was a one-way, XML-to-Flash conversion. Every modification had to be done within the XML editor, then published in Flash to see how it looks like – and that only applied to semantic content. Now that the multimedia document is published in HTML5, it is possible to fine-tune all timing and presentation details, after conversion, directly in a Firefox extension developed for that purpose.

Another benefit with the new workflow is that the workload can be shared efficiently between content authors and web designers: as presentation is entirely defined by CSS style sheets, web designers can work directly on the published multimedia documents to adapt the look to the website and to propose visual transitions between successive time segments. Content authors can see how all HTML fragments are displayed with these style sheets.

As content, presentation, and synchronization are defined in three separate resources (HTML5 document, CSS style sheet, and SMIL Timesheets, respectively), all local modifications can be easily backported to the XML publishing workflow.

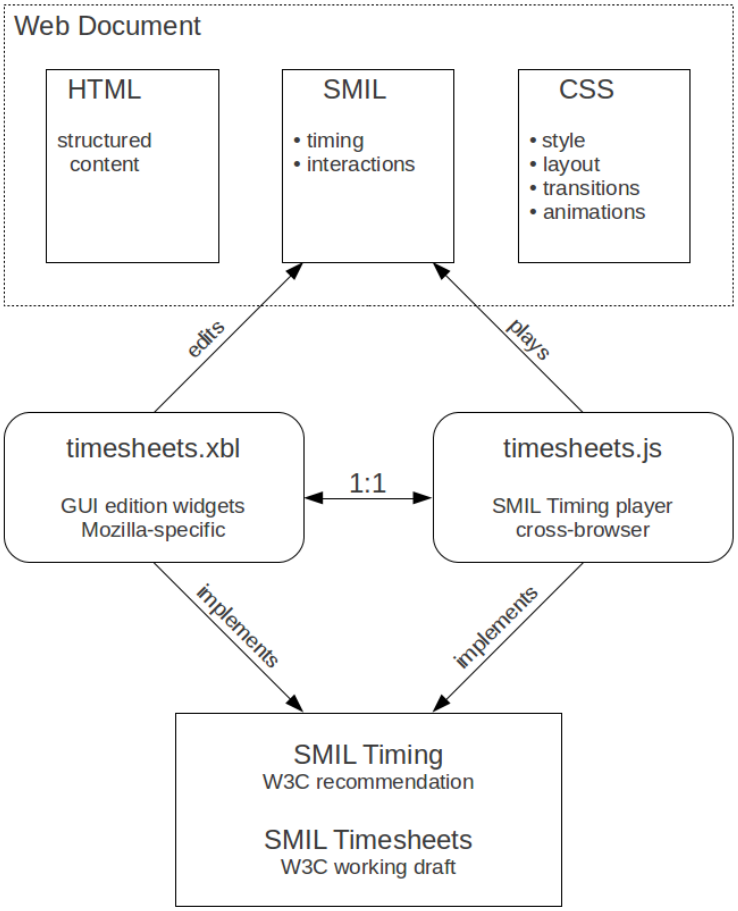


Figure 3: Direct Edition of SMIL Timesheets

5.2 Time Templates

We rely on Mozilla technologies to provide a "wysiwyg" editing environment. In particular, all editing templates are described with XUL (XML User interface Language), which is very similar to HTML and used for all Mozilla addons (e.g. Firefox extensions).

To ease the creation of such XUL templates, we provide two sets of XBL elements (XML Binding Language), as shown on Figure 4.

#timebox and #waveform are not really specific to SMIL Timesheets but they are useful for most time- and media-related templates:

- the #timebox binding defines specific `<textbox>` inputs for time values in HH:MM:SS format;
- the #waveform binding defines `<waveform>` elements to display the audio waveform of audio/video tracks and allows the user to select sharply a time fragment in such tracks.

#timeContainer and #timeNode are very specific to SMIL Timesheets: they fol-

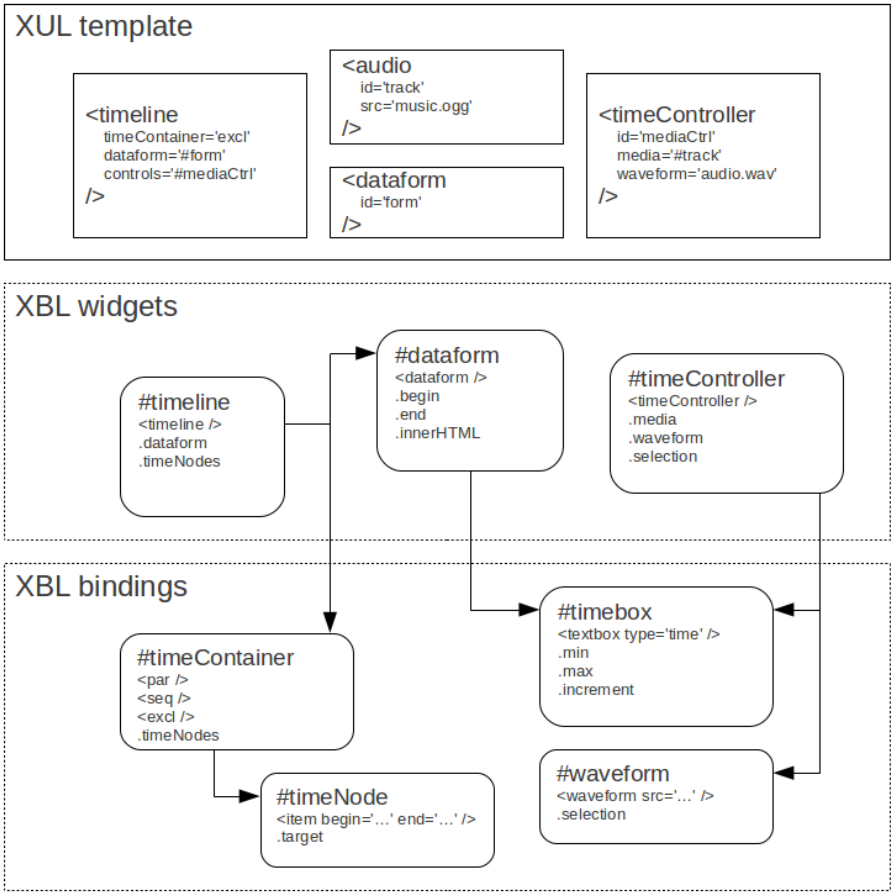


Figure 4: XUL Templates for SMIL Timesheets

low the same SMIL syntax. These bindings are a visual representation of SMIL time containers and time nodes:

- the #timeContainer binding is a generic, block-level element that renders the three possible types of SMIL time containers: `<par>`, `<seq>`, `<excl>`;
- the #timeNode binding defines inline-level `<item>` elements; the `begin` and `end` properties/attributes of these `<item>` elements define the width and position within the parent time container according to its time semantics.

These four bindings are used by higher level XBL widgets that provide an "out of the box" GUI, including toolbars and event synchronization mechanisms, that can be easily included in XUL templates.

- the #timeController widget synchronizes a `<waveform>` element with a media element (`<audio>` or `<video>`) and embeds a toolbar to control the zoom level and time position;
- the #timeline widget adds a toolbar to a #timeContainer element (`<par | seq`

`| excl>`) to easily add, remove, modify and sort time nodes (`<item>` elements), and binds every time node to its own data form;

- the #dataform widget is a very generic input form defining the timing and content of each time node target; the "begin" and "end" properties are required (and must be implemented with a #timebox binding), all other properties can be defined freely to match the template's needs.

With these two sets of XBL elements, designing a XUL template becomes accessible to XML content authors:

- high-level "XBL widgets" can be used directly in a XUL document, or extended with additional GUI elements (e.g. by adding a specific toolbar button to a `<timeController>` element). With this approach, most generic needs can be addressed with a rather simple XUL markup and very little JavaScript code.
- low-level "XBL bindings" elements can be used to provide GUI controls for other media- or timesheets-related applications relying on the Mozilla platform.

5.3 Integration in the SCENARI Document Workflow

Figure 5 describes the workflow (resulting from discussions we had in June 2010 in the C2M project and drawn by S. Spinelli):

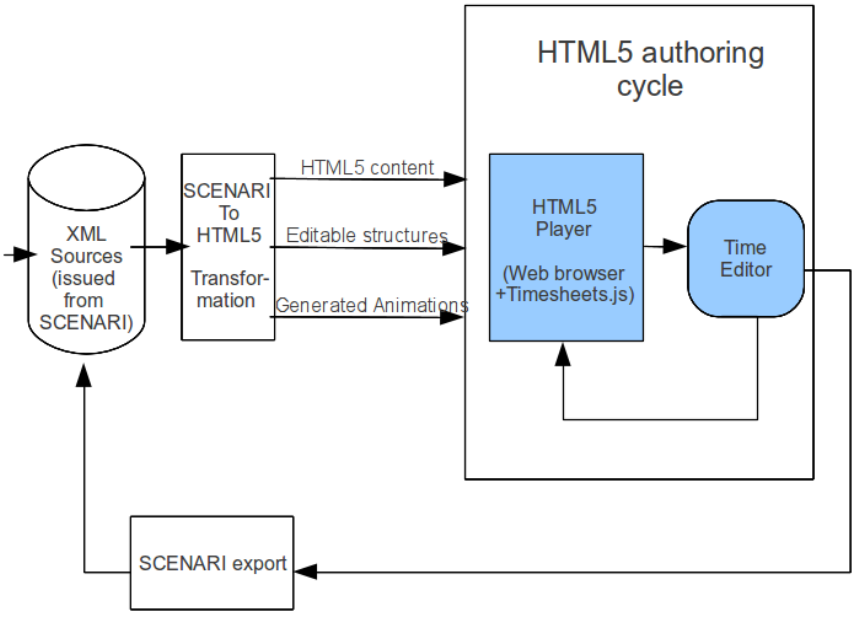


Figure 5: Authoring Workflow from SCENARI to HTML5

- The XML sources resulting from the SCENARI authoring phase are transformed into HTML5 (plus the corresponding CSS style sheets and SMIL Timesheets).

- The author can play the document in her/his prefered browser (as described in section 4).
- The author can edit the document for adjusting the timing of the content and for modifying transitions and animations (as described in section 5.1). She/he can then replay the document thanks to the previewing feature. Note that, in order to allow a complete SCENARI cycle, the editing actions are predefined and rely on the template specification.
- A SCENARI export service is used to re-inject the new values in the SCENARI XML structures.

6 Conclusion

This research report has two main objectives:

- Designing an authoring architecture for web multimedia documents consistent with structured document systems, standard compliant, and at the state of the art.
- Presenting the developments that have done for experimenting with the proposed approach.

The work done in this project aims at combining two worlds: a semantic-oriented authoring approach, as provided by an XML document workflow, and a direct web-based editing system. The first one guaranties homogeneous rendering while the second enables direct adjustements on final form of the document. Bridging these two worlds is made possible by using declarative standard web languages (namely HTML5, SMIL and CSS) and by implementating their timing part in browsers (the Timesheets.js library). The authoring components are indeed directly mapped to the document structures.

As described in this document, the main use case on which this work has been built and validated is the Webradio application. It has the advantage of requiring non trivial document structures together with rich synchronization and navigation features (timeline, table of contents, access to complements). This has provided the first test-bed for the initial versions of the Timesheets.js library through rich interactions between developers and users from INA.

In addition, the template-based approach of web multimedia documents built on HTML5, SMIL and CSS have been experimented and used for several categories of documents, as can be seen on the web site devoted to this technology.[10]

While the main options have been validated through two scientific publications, more implementation and experimentation have to be done on the authoring tool to evaluate (1) the benefits for the author of the direct authoring step, and (2) the level of quality that can be obtained with the post-editing phase. This will require to complete the integration components in SCENARI.

[10] `http://wam.inrialpes.fr/timesheets/`

References

[1] Adobe. Adobe FrameMaker 10, http://www.adobe.com/framemaker.

[2] Amaya. Amaya, a web editor, http://wam.inrialpes.fr/software/amaya/.

[3] S. Bonhomme. Kolekti, a web application for document production, http://www.kolekti.org/.

[4] D. Bulterman et al. *Synchronized Multimedia Integration Language (SMIL 3.0)*. W3C Recommendation, http://www.w3.org/TR/smil/, Dec. 2008.

[5] D. C. A. Bulterman and L. Hardman. Structured multimedia authoring. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1:89–109, Feb. 2005.

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

[7] F. Cazenave, V. Quint, and C. Roisin. Timesheets.js: Tools for web multimedia. In *ACM Multimedia 2011 Open-Source Software Competition*. ACM, Nov. 2011.

[8] F. Cazenave, V. Quint, and C. Roisin. Timesheets.js: When SMIL meets HTML5 and CSS3. In *DocEng 2011: Proceedings of the Eleventh ACM Symposium on Document Engineering*. ACM, Sept. 2011.

[9] F. Cazenave and C. Roisin. Moteur d'édition et de publication html de contenus webradio orienté "esthétique", http://scenari.utc.fr/c2m/docs/L1d/pdf/. Research Report, Projet C2M, Feb. 2011.

[10] 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.

[11] A. Di Iorio, L. Furini, F. Vitali, J. Lumley, and T. Wiley. Higher-level layout through topological abstraction. In *Proceeding of the eighth ACM Symposium on Document Engineering*, DocEng '08, pages 90–99, New York, NY, USA, 2008. ACM.

[12] J. dos Santos and D. Muchaluat-Saade. Xtemplate 3.0: spatio-temporal semantics and structure reuse for hypermedia compositions. *Multimedia Tools and Applications*, pages 1–29, 2011. 10.1007/s11042-011-0732-2.

[13] F. C. Flores, V. Quint, and I. Vatton. Templates, microformats and structured editing. In D. Brailsford, editor, *Proceedings of the 2006 ACM Symposium on Document Engineering, DocEng 2006*, pages 188–197. ACM Press, Oct. 2006.

[14] X.-S. Hua, Z. Wang, and S. Li. Lazycut: content-aware template-based video authoring. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pages 792–793, New York, NY, USA, 2005. ACM.

[15] N. Hurst, W. Li, and K. Marriott. Review of automatic document formatting. In *Proceedings of the 9th ACM Symposium on Document Engineering*, DocEng '09, pages 99–108, New York, NY, USA, 2009. ACM.

[16] Lodel. Lodel, logiciel d'édition électronique, http://www.lodel.org/index376.html.

[17] J. Lumley. Pre-evaluation of invariant layout in functional variable-data documents. In *Proceedings of the 10th ACM Symposium on Document Engineering*, DocEng '10, pages 251–254, New York, NY, USA, 2010. ACM.

[18] J. Lumley, R. Gimson, and O. Rees. Extensible layout in functional documents. In *Proceedings of IS&T/SPIE Symposium Electronic Imaging*, pages 177–188, 2006.

[19] J. W. Maxwell, M. MacDonald, T. Nicholson, J. Halpape, S. Taggart, and H. Binder. Xml production workflows? start with the web. *The Journal of Electronic Publishing*, 13, 2010.

[20] J. Mikac. Limsee3, a multimedia authoring tool, http://limsee3.gforge.inria.fr/, 2009.

[21] F. Mittelbach and C. A. Rowley. The pursuit of quality. In C. Vanoirbeek and G. Corey, editors, *Proceedings of the 1992 Electronic Publishing, Text processing and document manipulation*, EP'92, pages 77–94, Cambridge, 1992. Cambridge University Press.

[22] M. C. Penadés, J. H. Canós, M. R. Borges, and M. Llavador. Document product lines: variability-driven document generation. In *Proceedings of the 10th ACM Symposium on Document Engineering*, DocEng '10, pages 203–206, New York, NY, USA, 2010. ACM.

[23] V. Quint, C. Roisin, S. Sire, and C. Vanoirbeek. From templates to schemas: Bridging the gap between free editing and safe data processing. In *DocEng 2010: Proceedings of the Tenth ACM Symposium on Document Engineering*, pages 61–64. ACM, Sept. 2010.

[24] V. Quint and I. Vatton. Techniques for authoring complex XML documents. In J.-Y. Vion-Dury, editor, *Proceedings of the 2004 ACM Symposium on Document Engineering, DocEng 2004*, pages 115–123. ACM Press, Oct. 2004.

[25] V. Quint and I. Vatton. Structured templates for authoring semantically rich documents. In *Proceedings of the 2007 international workshop on Semantically aware document processing and indexing, ACM International Conference Proceeding Series; Vol. 259*, pages 41–48. ACM, May 2007.

[26] D. Saint-Martin, L. Gaillard, G. Chauvé, V. Carpentier, and S. Poinsart. Expression de besoins, http://scenari.utc.fr/c2m/DOCS/L4a/pdf/. Research Report, Projet C2M, Feb. 2011.

[27] Scenari. Scenari platform, http://scenari-platform.org.

[28] S. Sire. Axel + xtiger xml editing library, http://media.epfl.ch/Templates/.

[29] S. Sire, C. Vanoirbeek, V. Quint, and C. Roisin. Authoring xml all the time, everywhere and by everyone. In *Proceedings of XML Prague 2010*, pages 125–149. Institute for Theoretical Computer Science, Mar. 2010.

[30] 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.

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

[32] P. Vuorimaa, D. Bulterman, and P. Cesar. *SMIL Timesheets 1.0*. W3C Working Draft, http://www.w3.org/TR/timesheets/, Jan. 2008.

[33] D. Weck. Limsee2, http://limsee2.gforge.inria.fr/., 2008.

Contents

1 Introduction **5**

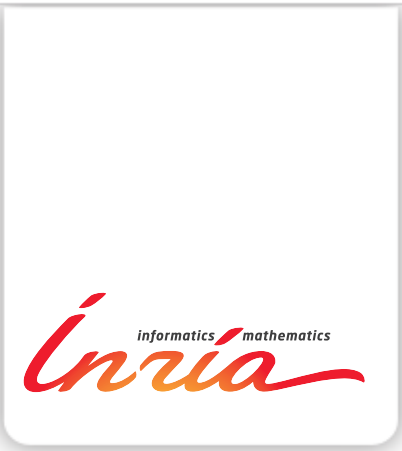
2 State of the Art **5**
2.1 C2M workshop on multimedia authoring 5
2.1.1 Direct web authoring using templates 6
2.1.2 XTiger XML and the AXEL editing library 6
2.1.3 The Kolekti document production system 7
2.1.4 SCENARI 7
2.2 XML-Based Editorial and Production Workflows 8
2.3 Layout Engines 8
2.4 Authoring Multimedia Documents 9

3 A Document Workflow for Multimedia Production **10**
3.1 A Use Case: INA Webradio 10
3.2 Requirements for this Use Case 11
3.3 Document Processing Workflow 12

4 The Timesheets.js Library **12**
4.1 Using HTML5, CSS3 and SMIL Timing 13
4.2 A Basic Example 13
4.3 Timesheets Engine 15
4.4 Application: INA Webradio 15
4.5 Browser Support 16
4.6 Extensibility: DOM events and JavaScript 17

5 Authoring the HTML5 Document **17**
5.1 Direct Editing 17
5.2 Time Templates 18
5.3 Integration in the SCENARI Document Workflow 20

6 Conclusion **21**



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399