HAL
open science

A Lightweight Framework for Authoring XML Multimedia Content on the Web

Christine Vanoirbeek, Vincent Quint, Stéphane Sire, Cécile Roisin

To cite this version:

Christine Vanoirbeek, Vincent Quint, Stéphane Sire, Cécile Roisin. A Lightweight Framework for Authoring XML Multimedia Content on the Web. Multimedia Tools and Applications, 2012, 10.1007/s11042-012-1159-0 . hal-00712637

HAL Id: hal-00712637

https://inria.hal.science/hal-00712637v1

Submitted on 27 Jun 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Multimed Tools Appl manuscript No.
(will be inserted by the editor)

A Lightweight Framework for Authoring XML Multimedia Content on the Web

Christine Vanoirbeek · Vincent Quint · Stéphane Sire · Cécile Roisin

Received: 27 May 2011 / Accepted: 10 June 2012

Abstract This paper addresses the issue of authoring XML multimedia content on the web. It focuses on methods that apply to different kinds of contents, including structured documents, factual data, and multimedia objects. It argues in favor of a template-based approach that enhances the ability for multiple applications to use the produced content. This approach is illustrated by AXEL, an innovative multi-purpose client-side authoring framework (previously described in [23]), intended for web users with limited skills. The versatility of the tool is illustrated through a series of use cases that demonstrate the flexibility of the approach for creating various kinds of web content.

Keywords Authoring paradigms · Authoring for the web · End user authoring

1 Introduction

Since its inception, the web has turned into an universal repository of rich content – i.e. content including text, images, graphics, music and videos – that is easily accessible through a web browser. Simultaneously, the web has progressively provided incentives for building complex back-end frameworks exposing attractive web based

C. Vanoirbeek · S. Sire
EPFL
EPFL/IC/GR-VA, BC Building - Station 14, 1015 Lausanne, Switzerland
E-mail: christine.vanoirbeek@epfl.ch
E-mail: stephane.sire@epfl.ch

V. Quint
INRIA
655 avenue de l'Europe, 38334 Montbonnot, France
E-mail: vincent.quint@inria.fr

C. Roisin
University of Grenoble and INRIA
655 avenue de l'Europe, 38334 Montbonnot, France
E-mail: cecile.roisin@inria.fr

interfaces to end users. These interfaces have grown to the point where some applications are running in the browser for the most part, thus becoming web applications.

While a significant part of the web is fed from several sources of information, human users (authors) are also entering content directly through various kinds of tools. Authoring has rapidly evolved towards the production of HTML pages including complex contents such as tables or embedded components. Whereas tables have been added directly to the HTML language, multimedia objects were (and still are) poorly handled, because the dedicated standard, SMIL [2], has not been widely adopted, but also because alternate solutions require coding skills. However, the introduction in HTML5 of continuous content with the audio and video tags paves the way for a better integration of multimedia content in web documents.

In this context, the approach proposed in this paper shows how client-based authoring can fit a broad variety of needs, ranging from static HTML documents to multimedia content. To a large extent, this approach was made possible by the recent evolutions of the web, conducted mainly by two categories of actors: standard organizations and web developers.

As a standard organization, W3C played an important role in transferring research results from the hypertext and document engineering domains to web technologies. Among them, three features are important:

- The first one is a clear separation of the content from the appearance of a document. This resulted in core technologies for delivering content on the web (HTML and CSS), that constitute now the basis for the web platform, including audio and video modalities.
- The second advance is the XML language, that incidentally led to a re-writing of the hypertext language for the web: XHTML. It encouraged the use of well-formed documents and opened the way to a set of languages and technologies for dealing with contents represented as hierarchical structures (XSLT for transforming XML data, DOM and SAX for processing content, XPath and XQuery for accessing content, etc.)
- The third point is the capability to formally define generic content models for document classes. This was originally addressed by DTDs for document publishing purpose. Several other languages were then created (XML Schema, RELAX NG, Schematron, etc.) to provide additional features, such as typing, that significantly enhanced the processing potential offered to applications. As a consequence, there is a trend to develop authoring solutions that offer users the possibility to generate XML content compliant to a generic model (or schema).

Simultaneously, the community of web developers has promoted web browser-oriented technologies, thus significantly contributing to a new evolution of the web as a medium: it is becoming a means for end users to feed and, to a certain extent, control the content of web sites, through web based interfaces and applications.

At the beginning, the authoring capabilities provided by a web browser were merely limited to the use of HTML forms. The Web 2.0 perspective created a real craze to provide users with new capabilities allowing them to enter content simply by using a browser, for instance in wikis and blogs.

From a technological point of view, the capabilities offered to end users to populate repositories of information through web browsers have been proposed under multiple forms (embedding proprietary applications, browser specific plug-ins, etc.). The development of script languages running on the client (among them, the now ubiquitous JavaScript) is the most significant change that considerably extended the potential offered to end users in terms of interaction. This evolution led to the development of so-called rich Internet applications that allow web pages to adapt their content and appearance according to user interactions, but also provide mechanisms to interact with back-end systems.

An important feature, aroused by the XML specification, is the development of a standard API, the DOM, to deal with data using XML trees as the primary representation of information. Coupled with a script language such as JavaScript, the DOM opened the door to the development of very powerful client-side applications for web authoring and for producing XML content.

As a matter of fact, it appears that the web progressively made it possible to merge sources of information that have been traditionally clearly separated: documents, data and multimedia content. Despite the fact that information available on the web may be produced in many different ways, it is to be observed that information is ultimately delivered in a document/hypertext way relying on hierarchical structures. XML and related languages and technologies are currently demonstrating the potential of this approach to bridge the gap between document, data and multimedia content.

All these evolutions of the web, its technologies and its usage have led us to devise an editing process that allows non XML savvy web users to easily create multimedia web content directly with their browser. This process is based on a lightweight framework called AXEL that was introduced in a previous paper [23]. In the present paper we extend the discussion introduced in the previous paper and we demonstrate, by using this tool for a variety of applications, that (i) web technologies enable a unified process for authoring documents, data and multimedia content, and (ii) that this process can be used by people with no previous knowledge of XML.

The paper is organized as follows. The next section gives an overview of the most common methods used for authoring XML content for the web, emphasizing three typical forms of web content: documents, data and multimedia; it also highlights the limitations of these methods. This overview is followed in section 3 by a short presentation of AXEL, the tool we have developed for authoring on-line all these contents. Its main features are discussed, including the templating language, the architecture and the user interface. This is illustrated by several applications involving different types of content. Section 4 discusses the results we have obtained so far. Finally, the conclusion summarizes the main contributions and suggests different extensions.

2 Authoring XML content for the web

Authoring paradigms for producing different kinds of XML content – data, document and multimedia – are separated. In this section, we synthesize the major trends in the domain, emphasizing important features such as the way to constrain the content

with a model, compliance to web standards, and usability issues in relation with the targeted audience.

2.1 Authoring document-centric content

The typical way to edit XML documents is to use an XML editor, i.e. a stand-alone application that runs on a personal computer and that uses a DTD or a schema to make sure the document produced is valid (in the XML sense). The first tools of this kind were developed for SGML, the ancestor of XML, and even before the birth of SGML, for structured documents using different formats [19, 11]. Most features offered by the precursors, by the SGML tools and by most XML tools date back from a pre-web period. Some more recent editors have been developed specifically for XML, but they are still based on the same principles: a DTD or a schema drives the editor to make sure the author can only enter a valid structure. In the broad category of XML editors, one can identify tools that provide an authoring environment where the focus is put on the document itself (ArborText Epic, FrameMaker), and those that provide a development environment (oXygen, XMLSpy), where developers can also prepare applications related to the edited documents. The latter category can manipulate schemas, XSLT transformations, and style sheets, for instance.

These stand-alone XML editors allow their users to take advantage of all features of the XML language, but they require a good knowledge of both XML itself and the document model in use, as expressed in a DTD or a schema. These editors are mostly intended for trained technical writers, who are more concerned with complex documentation than with the web: these tools are essentially web agnostic. An interesting exception is XOpus [27] which takes the approach of schema-driven editing, but runs in the browser. Similarly, easyDITA [10] proposes a complete web solution for collaboratively authoring and sharing content, but only for DITA documents. As opposed to the previous tools, both tools require a web environment and allows authors to work on-line.

The presence of a schema that constrains the editor is often a difficulty for untrained users. They feel more comfortable with HTML stand-alone editors, that are closer to word processors, and therefore seem familiar to most authors. In addition, all these HTML editors take the web into account, even if the most popular ones require specific operations for publishing content on the web. The counterpart of the ease of use of HTML editors is the lack of structure in the documents they produce. Authors are inclined to create a rather flat structure with a lot of style, and they are in any case limited to the kind of structure allowed by the HTML language. Some tools even generate invalid HTML code.

There are also on-line HTML editors. Usually called Rich Text Editors, they are used to enter content for wikis, blogs or on-line mail clients. These tools are very simple and generate still less structured content and less valid HTML code, but any web user is supposed to be able to use them and they are fully part of the web: they run in the browser and the content entered by users is immediately available on the web. However, the notion of a DTD or a schema, even the HTML DTD, has almost disappeared.

This is a real issue, as loosing validity and conformance to a model means that the content can not be safely processed in any other way than displaying it in a tolerant web browser. A middle ground can be found with templates. Templates are used server-side to generate HTML pages on the fly from data stored in a CMS or a data base. Different kinds of templates can also be used client-side to help authors structure the information they are entering through an editor.

Client-side templates for HTML pages are close in their purpose to those for word processors. They help authors to provide well structured information where it is needed, but do not force them to structure everything in a document. This may be especially useful for entering microformats in web pages, that can then be (re)used in different contexts. Templates can also help authors to organize HTML pages and to use the HTML markup consistently for certain types of web pages [5, 13].

2.2 Authoring data-centric content

Despite the fact that XML inherits from SGML, its ancestor designed for representing structured documents, it is currently also widely used to deal with data-oriented content within enterprise information systems. The adoption of XML by enterprises is clearly related to the information integration problem. XML – and its family of technologies – has become a de facto solution for sharing content between disparate information systems [17]. Considering XML content as a single authoritative source offers a number of advantages, such as publishing rich content in a myriad of formats or re-purposing it for exploitation by web services as well as by any standard compliant application.

From an end user perspective, the creation of XML data-oriented content, is mainly supported by the use of XForms. This technology is extensively used in commercial applications or services offered to end users, either for general purpose (such as registration to an event, booking flights or hotels, accessing bank services, etc.) or inside an organization to collect internal data (travel expenses, various announcements, etc.) [12]. As XForms is not currently supported natively by browsers, a number of frameworks have been developed which use the AJAX technology to offer it to end users.

The XForms language allows developers to specify, in a declarative way, a number of constraints regarding the pieces of information entered by end users through a form. It offers a binding mechanism that describes data types in accordance with the XML Schema standard data types. It also makes it possible to specify additional features which may be compared to functional dependencies in relational databases or to calculations in spreadsheets.

XForms, which is based on the Model View Controller approach, provides interesting features to guide end users during the data entering process. In addition to the expression of constraints on nodes (identified by XPath expressions), it offers indeed a view layer composed of intent-based user interface controls that may be adapted to users' habits by styling the user interface with CSS.

However, the compatibility with standards for generic content models (schemas) is weak for two main reasons:

- XForms is limited to expressing constraints related to data types (more precisely XML Schema data types); it does not support the complex elements that are defined by a schema. This is confusing because, for instance, the optionality of an element, which is not reflected on the XML data types, is proposed by the binding mechanism of XForms.
- As already mentioned, XForms can state computational dependencies among data elements, a feature that is not available in schema languages used to define generic content models. This is generally addressed by using XSLT.

As a consequence, even if most system providers promote the automatic generation of XForms forms according to an existing schema, it is not a simple issue and it does not prevent developers from adapting the XForms models. This is a potential source of inconsistency. XForms also allows, through its binding mechanism, to specify if a field is read-only. This is also an issue, because such a property is in fact dependent on the context of use of data.

Although forms are a good means for entering structured data, other paradigms have been proposed, inspired from other kinds of documents, such as variable-data documents [16].

2.3 Authoring web multimedia content

The area of multimedia authoring is wide, firstly because multimedia content tends to become the main information vehicle. Consequently many formats and associated tools exist, such as MPRO [1] for MPEG-4, HyperProp Editor [24] and Composer [15] for the iTV NCL language. Here, we focus on multimedia content that is published on the web and that can be authored together with other web contents. Therefore, we are only addressing the structure-based paradigm, as defined in [3]. Note that the above mentioned tools usually rely on a template-based authoring user interfaces, in order to hide the inherent complexity of the multimedia authoring task [21, 20]. Thanks to their intrinsic modularity and their declarative paradigm, structured languages are good candidates for the development of such template-based editing features.

In several multimedia authoring systems [9, 20], templating is a means to bring reusability and expressiveness with usability. However authoring in this context is still an open issue because of the intrinsic complexity of the temporal structures and relationships that have to be handled. For instance, XTemplate 3.0 [20] proposes a wizard to guide the user in finding the appropriate hypermedia template among a template base. But this wizard tool misses the modular feature provided by the XTemplate 3.0 language, because it is restricted to one template per document. Therefore, while the language provides powerful expressiveness and reusable component features, it is still too complex for many users. As stated in [20], a graphical authoring tool would be useful but difficult to develop because of the complexity of the concepts involved.

A natural way to publish synchronous multimedia in XML on the web is to use the SMIL language [2], which was designed exactly for that purpose. Several authoring environments were developed for producing this kind of content (e.g. GR*i*NS [4],

LimSee2 [8]), but the SMIL technology is not widely deployed on the web. Only a few players are available and no web browser provide support for the SMIL language. It is then difficult to reach a wide audience on the web with the SMIL format. The alternative is to represent content in a widely supported, a-temporal document format such as HTML, and to add synchronization and user interactions by scripting. The issue then is the programming skills that are expected from authors, and the difficulty for web users to author content.

Another option is to use HTML pages that embed synchronous multimedia applications coded in some proprietary format (Flash or Silverlight for instance). There are convenient authoring tools for these formats, but contents encoded in such non-standard formats raise several issues: they are poorly integrated in the web (how to link to a specific part of them?), they are difficult to reuse, they require special plug-ins which are often not available for mobile devices, and they can not be synchronized with the rest of the page where they appear.

The situation has changed recently with the raise of HTML5, which supports natively continuous content. New features have also been introduced in the CSS3 language. With these novel technologies, it is now easier to include sound and video in HTML pages, without requiring any plug-in. It is also possible to introduce certain temporal behaviors within HTML5 pages thanks to the new CSS3 animation and transition modules.web

However, HTML5 and CSS3 are not sufficient for all multimedia applications, but with the help of SMIL Timesheets [26], user interaction and complex synchronization schemes can be added to HTML5 pages. Some JavaScript implementations of SMIL Timesheets are already available [25, 6, 7], that allow any web browser to play synchronized multimedia content represented in standard, declarative web languages. Programming skills are no longer necessary, and the same authoring techniques used for HTML+CSS applications can be used to create this content. However, additional features are required for easily manipulating time structures and time dependencies.

3 AXEL: a lightweight web authoring tool based on a template language

In this section we present a web-based authoring environment that bridges the gap between the separate authoring paradigms discussed in section 2. It is aimed at providing web users with the capability to produce documents, data and multimedia contents in an uniform way. It relies on XTiger XML, a template definition language that constrains, but in a user friendly way, authors to create reusable content. It is to be observed that templates are reflecting the know-how of communities of users, companies or individuals whose objective is to share and exploit content provided on the web by end users; the definition of the templates themselves is addressed by this category of persons in order to produce reusable content in a specific context. Typical use cases are illustrated in section 3.2.

Our authoring framework is implemented with AXEL (Adaptable XML Editing Library), a client-side JavaScript library[1] for authoring XML template-based content on the web. A detailed technical description is available in [23].

[1] https://github.com/ssire/axel

3.1 AXEL features

AXEL has been designed and implemented to fulfill two main requirements:

- Proposing a unified approach for authoring valid XML documents mixing document, data and multimedia contents;
- Providing non XML-savvy developers and integrators with a lightweight framework for web based on-line editing of valid XML content as well as a user interface that accommodates end users' practices.

3.1.1 An unified approach

Following the observations of section 2.1, the idea behind XTiger XML is to let authors edit a HTML document and to get rid of the complexity of XML (both its syntax and its schemas) while editing. However, as the objective is to produce a well structured XML document, some constraints have to be set somewhere, and an XML structure must be created. That is the role of the XTiger XML language. It constrains the HTML document being edited by associating with it the structure of the final XML document. A XTiger XML template defines a document type, in the same sense as a XML schema, but in addition it also defines the visual aspect of the document and the user interface that are used when editing it.

A XTiger XML template is the skeleton, expressed in HTML, of a document that has a structure close to the target XML document. In addition, a template contains structural constraints expressed by XTiger XML elements. As opposed to the traditional XML approach, where a document and its schema (its structural constraints) are separate resources, the HTML document being edited with AXEL and its XTiger XML constraints are mixed in a single structure: each part of the HTML document that must follow certain structure rules is encapsulated in a XTiger XML element that expresses these rules.

The example below shows how the part that contains the abstract and the keywords of this article is represented in the template (in the example, elements prefixed by `xt:` are XTiger XML elements, the others are HTML elements):

```
<div class="abstr">
  <p class="heading">Abstract</p>
  <xt:repeat minOccurs="1" label="abstract">
    <p>
      <xt:use types="text" label="Parag">Enter abstract</xt:use>
    </p>
  </xt:repeat>
</div>
<p class="keywords">
  Keywords:
  <xt:repeat minOccurs="2" label="keywords">
    <xt:use types="text" label="keyword">Enter keyword</xt:use>,
  </xt:repeat>
</p>
```

In a template, all HTML elements that are not within a XTiger XML element are mandatory and can not be changed by authors using this template. In the example above, there is always a single `division` of class `abstr` that contains a first paragraph with the only word "Abstract". This `division` is always followed by a paragraph of class `keywords` starting with the string "Keywords:".

HTML elements can be changed by an author using the template only when they are in a XTiger XML element, which indicates what changes are allowed. Here, the first `xt:repeat` element means that several paragraphs (at least one), and only paragraphs (HTML `p` elements), can appear after the mandatory first paragraph in the `abstr division`. The second `xt:repeat` element means that there must be at least two `keyword` elements after the mandatory "Keywords:" string.

Finally, the `xt:use` elements indicate what kind of content can be created at a given position by an author using the template. In the example above, only text (`types="text"`) can be created in the variable parts. In this example, the content of the `xt:use` elements is just a prompt that the author is supposed to replace. The `xt:use` element also allows other media objects (audio, video) or more complex structures to be inserted at a given position, and it can propose different options for these structures (this is explained in more details in [23]).

This defines what HTML structure can be created by an author using the template, but the template also contains information about the final XML structure that will be generated from the HTML structure. This is achieved by the `label` attribute. This attribute indicates that an XML element must appear in the output and it specifies its name. A typical XML structure generated from the template above is:

```
<abstract>
  <Parag>First paragraph of the abstract.</Parag>
  <Parag>Second and last paragraph.</Parag>
</abstract>
<keywords>
  <keyword>kw1</keyword>
  <keyword>kw2</keyword>
  <keyword>kw3</keyword>
</keywords>
```

XTiger XML is well suited for specifying how multimedia content can be included in structured documents, in particular with the `xt:use` element. Combined with the typing feature of XTiger XML, the AXEL engine can then provide document authors with a simple user interface, hiding the complexity of multimedia manipulation, while producing rich web content with media synchronization and interaction. This is achieved by using in the template the required CSS and SMIL Timesheets features for making the final documents synchronized and interactive. Only the template developer has to handle the complexity of synchronization and interaction. The document author has just to decide which of the predifined multimedia features offered by the template s/he wants to include in a document. Some templates may also allow the author to adjust a few selected parameters for fine tuning synchronization and interaction.

This very short introduction to XTiger XML templates obviously does not present all the possibilities of the language. For more details, refer to [23] or [22]. Section 3.2 provides examples that show how these templates have been exploited to edit various types of documents in different application frameworks.

3.1.2 A lightweight editing framework

To edit a document, AXEL, the client-side template engine, loads a template in a browser window, and transforms it into an interactive editing application. It follows the constraints expressed by XTiger XML elements and allows the author to develop the document by creating new elements and entering content under these constraints. When the author creates a new item, in a `xt:repeat` element for instance, it clones that part of the template. That way, the constraints that apply to each part of the document are always available locally, thus simplifying the task of the editor. Basically, AXEL only manipulates the DOM tree; the rendering is performed by the browser rendering engine.

To enter some content in a `xt:use` element for text, the author has just to click and type. The editing engine creates dynamically a HTML `textarea` element; text editing in this element is then handled by the browser itself. The AXEL engine copies the entered text into the `xt:use` element in the DOM tree. For continuous content, the engine allows the author to enter the URI of some video or audio content available online.

When the author saves the document, AXEL creates the XML structure. To do so, it traverses the DOM tree, and for each XTiger XML element with a `label` attribute, it creates a XML element in the output structure. It also stores the content of `xt:use` elements of type `text` in leaves of the generated XML structure.

The principle behind the editing engine is to take advantage of all features offered by the browser (access to DOM tree, rendering, content editing, spell checking). Combined with a simple constraint language (XTiger XML) that provides the required information exactly where it is needed, this approach makes the editing engine both light and fast.

The user interface is directly derived from the template, in a way similar to such editors as Citrus [14]. User interface controls are inserted in the HTML document everywhere the templates allow authors to create, change or delete elements. Except for these controls, what the author sees is a HTML document formatted by the browser as usual (see Figures 2, 4, 5). The template defines the HTML elements that represent the intended XML structure. In addition, CSS style sheets may be used to customize the look of the document on which the author interacts.

With this approach, the user interface can be anything between full WYSIWYG and form-based, depending on the template and its associated style sheets. In any case, it does not show any XML syntax, which makes AXEL easily usable by everyone, while still generating well structured XML content. The whole process does not require complex schemas and transformations, but still provides useful, automatically processable XML information.

Figure 1 illustrates the overall process. An (X)HTML document embedding the template is exploited by the AXEL library for two purposes as represented by the

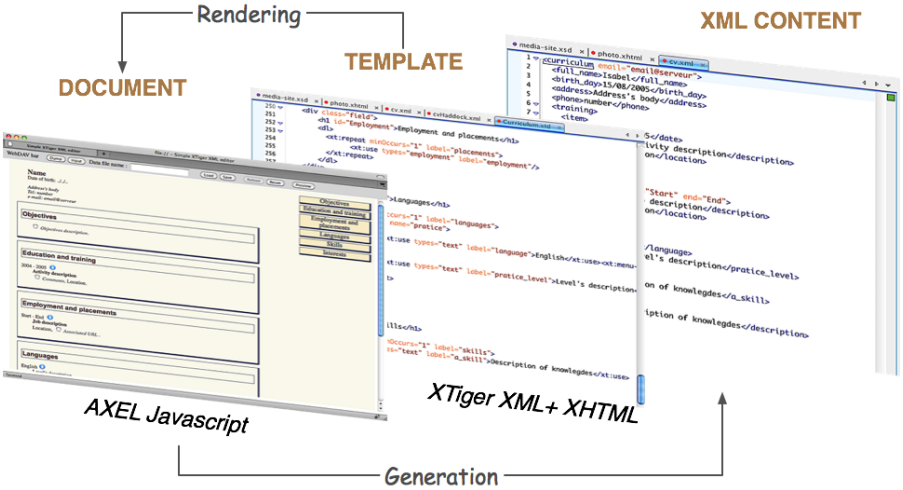

Fig. 1 Use of template by the AXEL library

two arrows: (i) providing authors with an intuitive interface (rendered through the use of CSS) to guide them while editing valid XML content and (ii) generating XML content that may be subsequently exploited by various back-end systems.

3.2 AXEL applications

The versatility of AXEL is demonstrated through examples of different authoring applications. The examples shown in this section cover the three types of web content introduced in section 2. They have been created and used during the last two years. These applications emphasize AXEL's main strength: while sharing a minimal common look and feel, all editing applications are customized to better fit the authoring task for a given XML content model.

With AXEL, the process for developing an application is split into three main activities, each one performed by people with different skills, like in any web application:

1. Software developers, with skills in JavaScript, may be required to adapt or extend the AXEL library if the application has some special requirements (for instance new widgets or plug-ins for authoring specific content). Not all applications require this step.
2. Web developers, with skills in HTML and XML, create XTiger XML templates that follow the XML structure to be generated and the HTML structure to be presented to the author.
3. Web designers develop the CSS style sheets that will define the look of the HTML pages supporting interaction with authors.

When this work is done, all web users are able to create and edit the specific content for the application. Web users become online authors, but no specific skill is

expected from them. The chosen examples illustrate that through various combinations of document-, data- and multimedia-oriented contents:

1. The first example consists in writing up an article like the present one: it is obviously mainly document-oriented, but several elements (such as authors, affiliations or bibliographic references) may be considered as data-oriented.
2. The second example consists in preparing a research project proposal: even if it is mainly document-oriented, it contains more data-oriented content (a typical case is the description of tasks from which a Gantt chart is automatically generated).
3. The third example concerns the publication of company showcases on the web: it contains the three categories of content: factual data about the company such as its address, document-oriented content such as the description of products and services, and multimedia content such as a video clip.
4. The fourth example concerns the creation of a slide show: the content is well balanced between document and multimedia elements; it also contains some data-oriented information (such as the date and the details about the related event, or the coordinates of the author).

3.2.1 Writing up an article

One of the first applications of AXEL was to create an authoring application for articles such as the present one. The *article* template contains typical document data such as sections, paragraphs, lists, figures and a set of meta-data such as authors, keywords and addresses. This template is embedded inside an `iframe` element of a custom-made application that adds a menu bar with the load and save buttons. It operates on a WebDAV server to support a collaborative authoring workflow.

In this primitive application, images are manually copied to the WebDAV server and an AXEL plug-in allows authors to insert an image by typing its path (e.g. `../images/slide-editor.png`). Once the path of an image is entered, the plug-in displays the image. Clicking the image replaces it with a text entry field that allows the author to change path. This simple image plug-in is of course not a production level example, however it illustrates one of the strengths of a lightweight client-side JavaScript approach for creating editors: its ability to support quick prototyping methods for extending the application with new features. As we will show in another example, we have also developed a more user friendly image upload plug-in.

This XML authoring toolchain, assembled from a document template, a single HTML host page, a few lines of JavaScript code and a WebDAV server, produces a document in a custom XML format. We have written XSLT transformations to transform that format into DocBook and LaTeX. This allows authors to benefit from other standard tools to then generate a PDF version of the article. Moreover, some features that were not available in the editing application, such as cross linking references to figures, have been implemented inside the transformation: a regexp matcher turns every *Figure N* words appearing in a paragraph or a list into a link to the *Nth* figure in the article. We use a similar technique for links to bibliographic references.

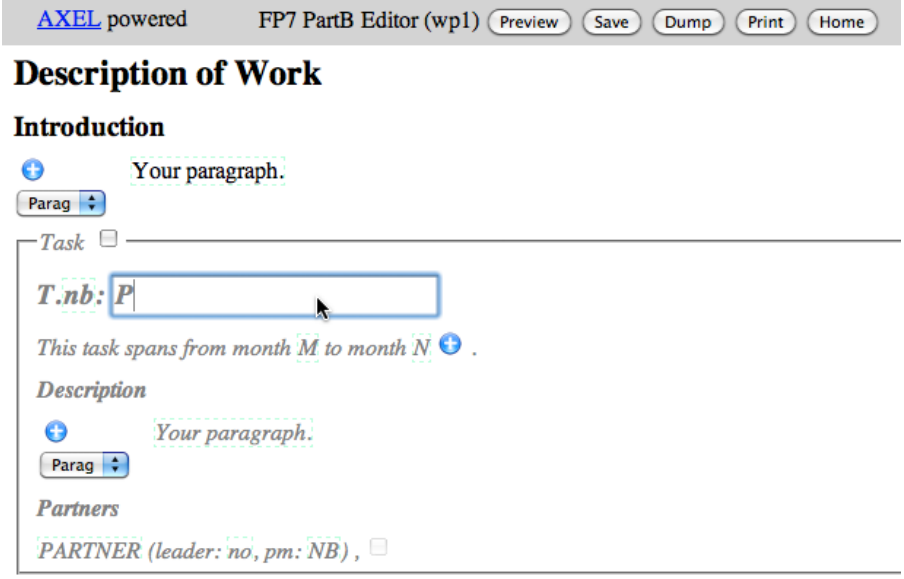

Fig. 2 Project proposal editor application and its menu bar at the top

3.2.2 Preparing a project proposal

A second example is a template designed to prepare and to submit European project proposals for the seventh research framework program. More precisely, this template is used to edit the submission document. This is indeed a hybrid example, since a specific feature of the submission document is that it must contain the breakdown of the project into work packages. In this document, each work package description is a mixture of structured document-oriented material, such as the objectives or the content of each task, intertwined with the details of the scheduling and the resource allocation in person-months for every task in every work package as shown in Figure 2.

The project proposal editor has also been integrated in a WebDAV application to enable collaborative work. It was used to prepare three different project submissions, and all work package leaders (an average of 8 of them for each project) have entered the description of their work packages through the tool. The document template was designed to generate a simple user interface that is halfway between a document and a form, as can be seen on Figure 2. We have written some XSLT transformations to generate an HTML version of the document that is very close in appearance to the MS-Word document distributed by the European commission as a model of what to submit.

The advantage of using a custom XML model (right part of Figure 3) is that it made it possible to create transformations for automatically generating the other parts of the documents, such as the Gantt chart for the whole project (as can be seen on the left part of Figure 3), or some tables summarizing deliverables and their due date, without requiring duplicate input.

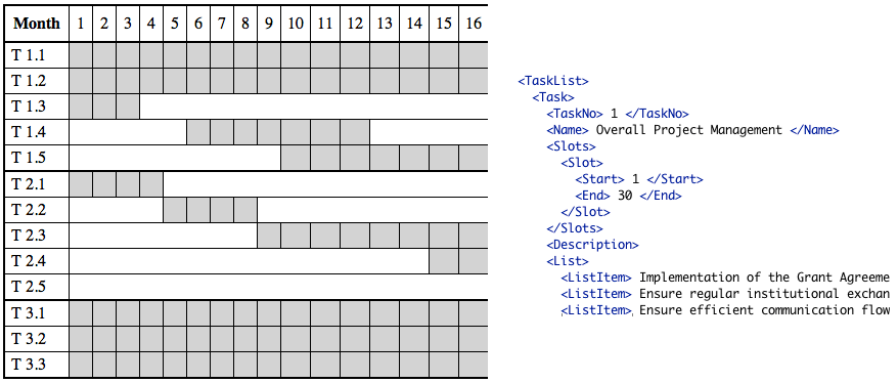

Fig. 3 A Gantt chart generated from the structured data on the right that was edited in a workpackage description

These first two examples (scientific article and project proposal) are document-centric, in that their aim is to produce standalone documents which have to be submitted or moved somewhere. For that reason they could cope with a simple WebDAV file storage. Many more applications become possible when using a native XML database for storing the document. The next section presents some more data-oriented applications realized with an XML data store.

3.2.3 Authoring showcases for the web

AXEL is also a convenient tool for creating data entry user interfaces for applications built with an XML database back-end. During the last year we have created entire web sites stored in XML databases, using the XRX (XML REST XQuery) paradigm. AXEL was used for data entry by the end users themselves.

A typical application is a web site presenting an association of professionals, with records or showcases associated with each member. The data model is highly dependent on the activity of the professionals, but in all cases, like the project proposal editor presented above, it combines some descriptive parts, including text, images and/or videos, with factual data such as addresses, email, web sites, and so on.

Figure 4 illustrates a showcase presenting start-up companies in a science park. This web site[2] was developed with AXEL, based on a specific template. Each company (currently about one hundred) is using it to enter and update their profile directly from the browser: on the right part of the figure we can note the image upload plug-in in action to enter the company's logo. A similar web site has been recently developed for a craftsmen association in Belgium. Each professional can edit her/his own showcase directly on the web site[3]. The template for this web site was quickly derived from the science park site.

One advantage of using AXEL over a classical rich text editor for such web sites is that any page content is stored as XML data and is then fully reusable. To illustrate

[2] http://societes.parc-scientifique.ch/

[3] http://www.uniondesartisansdupatrimoine.be

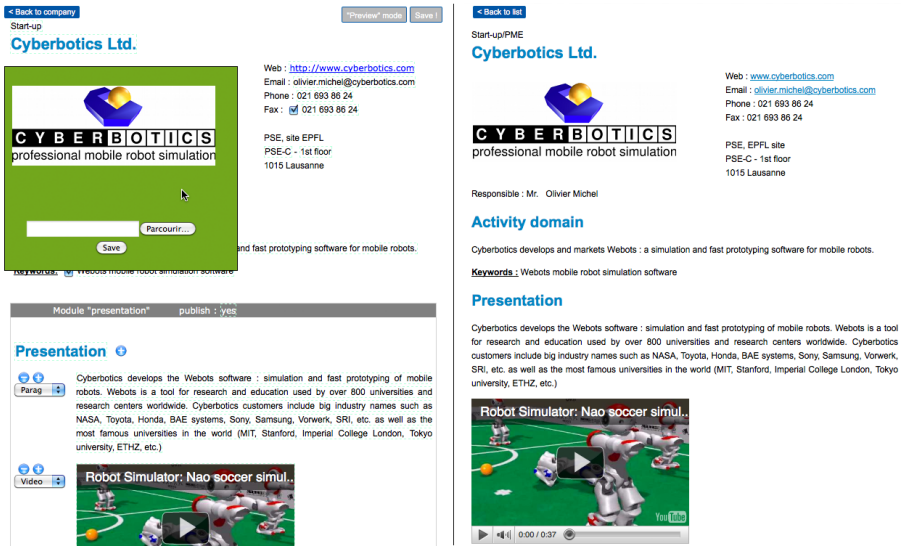

Fig. 4 Showcase editing, editor view on the left and static view on the right.

this we have made the XML content associated with any page available by adding a `.xml` prefix to its URL.

This example also uses the ability of templates to encapsulate the most complex aspects of multimedia authoring. If the template developer wishes to develop some more advanced video synchronization artifacts in the showcase, like displaying some paragraphs in sync with some video playback, s/he can create the SMIL time sheet that provides interactivity and synchronization in the resulting web pages. The availability of client-side multimedia libraries supporting SMIL time sheets, such as timesheets.js,[4] makes this simple.

The showcase web site gets huge benefits from using structured content. One benefit is that it allows to layer the site search engine so that it uses the semantics of the data model to adjust its results. In the company showcase example, the search engine considers first the terms that appear in the company name, then in the company profile, and finally in the company description. Moreover, in some contexts, such as the company name, it can use a NGRAM index instead of a full text index. Another benefit that we are starting to exploit by using multimedia libraries, is that the whole content is reusable, for instance to create a rotating banner. In that case the banner gets its content directly from the latest content entered within the database and does not require duplicated input.

We have realized some other focused data-oriented templates for various applications, such as a bibliographic data template, a restaurant menu card template, or a timetable template. These templates and the relevant applications have been described in a previous publication [23].

[4] http://wam.inrialpes.fr/timesheets/

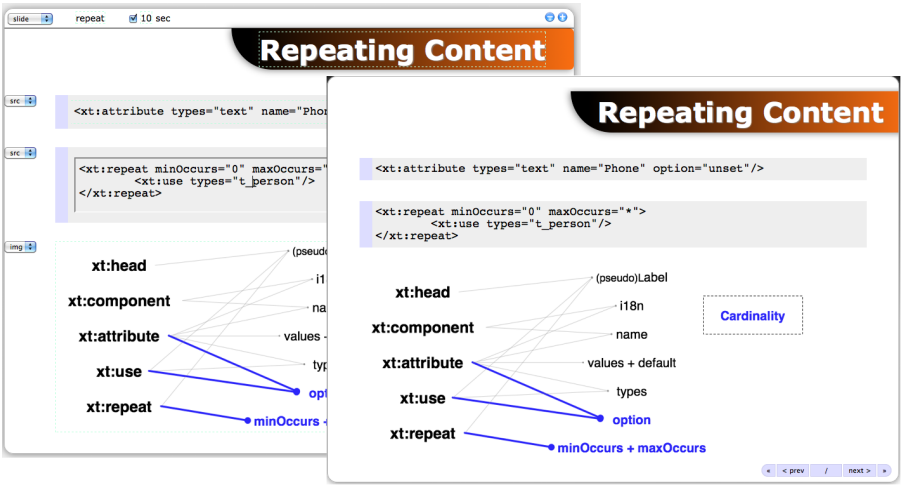

Fig. 5 Editing a slide

3.2.4 Authoring and publishing multimedia presentations on the web

One of our latest developments is a document template representing a slide show that takes advantage of SMIL Timesheets to synchronize the show and to handle user interaction, including navigation among slides. In its current version, the slide show editor displays all the slides, in presentation order, as a scrollable document in the browser window. A menu bar at the top of the editor window contains buttons to load and save the document, and also to preview the slide show, displaying one slide at a time. Figure 5 shows a slide while editing it (with associated widgets), and while playing the slide show (on the right part).

While editing, the preview is available to allow the author to check at any time what the end user will see. Playing the slideshow as a preview during the authoring phase is done simply by generating the full multimedia document from the current state of the document. Thanks to the use of web standards, this document is an HTML document which can be loaded inside a `div` element in the editor window itself. Then the multimedia library is used to play the slideshow. During playback the `div` element containing the editor is hidden and therefore every user interface controls added for editing (widgets on the left part of Figure 5) are removed. On the playback view, the document includes navigation buttons as specified by the template for the dynamic behavior of the slide (shown on the bottom right part of the figure). We are now developing a synchronization feature between the multimedia library and the editing library to easily go back and forth between the slide show player and the editor, while keeping the focus on the current slide.

This authoring process allows the author of a slide show to focus on the content, but it can also include true multimedia features in the resulting document. For example, based on the template, the AXEL engine provides automatically the table of contents and the controls that will help the final user to navigate throughout the slides. In the time sheet associated by the template with the authored content, the web devel-

oper can also define various display modes for the end user, such as automatic slide timing and incremental or progressive display. The code below[5] provides a typical SMIL time sheet for this purpose.

```
<!-- Slideshow Navigation -->
<seq timeAction="none">
 <item select="div.slide">
  <par>
   <!-- Outline/Accordion Lists -->
   <par timeAction="class:active">
    <item select=".outline> li" begin="click" end="click"/>
   </par>
   <excl timeAction="class:active">
    <item select=".accordion> li" begin="click" end="click"/>
   </excl>
   <!-- Incremental/Progressive Display -->
   <seq timeAction="visibility" begin="click" next="click">
    <item select=".incremental:not(ul):not(ol),
                  .incremental li" fill="hold"/>
   </seq>
   <par timeAction="visibility">
     item select=".progressive:not(ul):not(ol),
                  .progressive li" beginInc="2s"/>
   </par>
  </par>
 </item>
</seq>
```

4 Using AXEL

4.1 Results

We have developed several applications with the AXEL library, based on different templates, and in different collaborative settings. Each application is composed of (i) at least one template, (ii) some JavaScript code handling a menu bar for loading and saving the edited documents, and (iii) in some cases a backend application running on a server.

A few typical applications are presented in section 3. The article authoring application was used by the present authors to collaboratively write articles. This paper is currently the third one written using this system. The project proposal application has been used up to now to prepare 3 different research projects. In this application, each person in charge of a workpackage edits her/his own document. The science park directory has more than 200 authors. However, most of the initial version of the showcase was written by two coordinators. After this initial, centralized effort, the

[5] Thanks to F. Cazenave for this time sheet.

web site is now updated on a daily basis by each of the member companies. Thanks to the ease of use of the on-line editor, the site remains up-to-date, which was not the case with the previous, more conventional system.

In addition to the applications presented in section 3.2, we have developed (i) an application for editing a newsletter, (ii) a complete web site for a craftsmen association where most of the content is editable, and (iii) a standalone test specification authoring tool.

- The newsletter application is used by up to 5 authors who have already produced 9 newsletters to date. In this application, the backend server displays the name of the author who is currently editing a newsletter to prevent conflicts, since each newsletter is a single document.
- The secretary of the craftsman association has created 42 web pages presenting the craftsmen, 17 news and 10 meeting announcements (69 documents involving 4 different templates). We have started to develop a second web site for a different group with similar needs.
- The test specification authoring tool has been used to edit a lengthy specification (a 200-page document defined by 1 template) which has been created in 30 different languages using the editor. Each document was edited by one author.

In summary, as of December 2011, the concepts and technologies presented in this article have lead to the creation of at least 300 documents based on 9 templates in 6 different applications and environments. Roughly 200 authors have used AXEL to produce content for real applications. Feedback received from them highlights the ease of use provided in particular by the intuitive and non-intrusive user interface. We believe these qualities derive directly from the main options we have taken: template-driven editing of formatted documents, and online web authoring. AXEL users have reported that they feel very comfortable to edit web content in the context in which it will be used.

4.2 Discussion

The extensibility of our approach has been demonstrated through this wide range of applications. Extensibility is primarily obtained thanks the architecture based on web technologies. Plug-ins may be added to cover new needs. The AXEL library itself may be extended. By creating new templates, new types of documents and new applications can be addressed. As the editor generates XML documents, the whole set of XML processors and languages may be used to process these documents further, thus extending applications.

This last feature is based on the ability of automatically generating an XML Schema from a XTiger XML template [18]. Although no schema is required by AXEL (templates play that role), it is often useful to have an XML schema that defines the structure of the XML documents created with the tool. That way all the usual XML tools may be used to process documents safely.

The architecture also provides scalability, as the largest part of processing is distributed. All the editing code is executed on the client side. This ensures high re-

sponsiveness for authors without overloading servers, which are not involved in the interactive part of applications.

As compared with traditional XML web applications, AXEL can be considered as a simplified approach. An XTiger XML template mixes several aspects of documents that are usually separated in different resources (schema, document instance, transformations, presentation) expressed in different languages (XML Schema, XML, XSLT, HTML respectively). This separation of concerns still exists with AXEL, but there are much less resources involved. During editing, a single XTiger XML template plays simultaneously the role of a schema, a document instance, and its visual presentation; and when the document is saved as XML, the transformation is again guided by the template. Nevertheless, as explained in section 3.2, different professionals, each focused on a different aspect of the document, may be involved in the process of creating an application. Structure and presentation, for instance are clearly identified in a template: XTiger XML elements define the structure, HTML elements express a presentation. Mixing them in a single resource makes editing easier, by allowing every structure change to be immediately reflected in the presentation.

Merging structure information with document instances has some advantages for maintenance. If the document structure for an application has to change, a new version of the template is created. New documents are then authored with this new template. Note there is no ambiguity about which document uses what version of the document structure definition, as this definition is part of the document itself. If old documents need to be converted to become consistent with the new structure definition, we are in the same case as usual XML applications: we have the XML form of old documents, their XML Schema (automatically generated from the old template) and the new XML Schema (created from the new template). An XSLT transformation may then be developed for converting documents.

Requirements for authoring multimedia contents have been addressed with the objectives of simplifying the authoring task and reusing as far as possible synchronization and interaction patterns. These objectives have been achieved by typed templates (a feature of the XTiger XML language) and by using the well-established paradigm of content / style / behaviour separation. This works well, as soon as the classes of web contents to be created are well identified and are properly defined in templates.

5 Conclusion and future work

The XML mark-up approach demonstrates a number of advantages when publishing information on the web, as compared with traditional data models. XML provides a way to go beyond conventional data representation, especially with its ability to mix different types of contents in the same hierarchical structures, down to a very fine granularity level if needed. XML also brings interesting opportunities for processing and repurposing contents.

The challenge is to provide end users with web-based authoring tools to populate the web with valid XML content mixing document, data and multimedia components. The diversity of the applications we have realized during the last two years has made

us confident that the template approach together with the JavaScript library for client-side authoring is powerful enough to generate a whole family of authoring tools on the web, each one being well adapted to the information to be manipulated.

We have identified several areas of development and improvement for the future. One of them is to extend the template definition language to cover the full spectrum of form design. This can be achieved by developing ad hoc primitive editors for managing usual form controls. We are planing for instance to support the XForms user interface control vocabulary directly in XTiger XML. Another direction is to extend the template language to include some constraints on the content itself, and not just on its structure, such as restricting some input to specific data types. XForms bindings is a source of inspiration in this area. Authoring multimedia content with AXEL still needs further investigation: first to experiment new templates for popular applications such as media annotation, and second to add to our library new user interface controls for editing time dependent features, for instance through a timeline.

Another area of improvement is to adapt the AXEL library to support network driven partial updates of the edited content, which is not the case currently. This would enable collaborative authoring applications with tight coupling allowing multiple authors to edit the same content at the same time. Moreover, some extensions to the XTiger XML language would allow templates to define fine grain locks and access control rules at a structure level in a descriptive way.

References

1. Boughoufalah, S., Dufourd, J.C., Bouilhaguet, F.: MPEG-Pro, an authoring system for MPEG-4 with temporal constraints and template guided editing. In: Proceedings of ICME 2000, pp. 175–178. IEEE (2000)
2. Bulterman, D., et al.: Synchronized Multimedia Integration Language (SMIL 3.0). W3C Recommendation (2008). URL http://www.w3.org/TR/smil/
3. Bulterman, D.C.A., Hardman, L.: Structured multimedia authoring. ACM Trans. Multimedia Comput. Commun. Appl. **1**, 89–109 (2005)
4. Bulterman, D.C.A., Rutledge, L., Hardman, L., Jansen, J., Mullender, K.S.: Grins: An authoring environment for web multimedia (1999). URL http://homepages.cwi.nl/ dcab/PDF/edmedia99.pdf
5. Campoy-Flores, F., Quint, V., Vatton, I.: Templates, microformats and structured editing. In: D. Brailsford (ed.) Proceedings of the 2006 ACM Symposium on Document Engineering, DocEng 2006, pp. 188–197. ACM Press (2006)
6. Cazenave, F.: A declarative approach for HTML Timing using SMIL/Timesheets (2011). URL http://wam.inrialpes.fr/timesheets/
7. Cazenave, F., Quint, V., Roisin, C.: Timesheets.js: When SMIL meets HTML5 and CSS3. In: Proceedings of the 11th ACM Symposium on Document Engineering, DocEng '11, pp. 43–52. ACM, New York, NY, USA (2011). URL http://doi.acm.org/10.1145/2034691.2034700
8. Deltour, R., Layaïda, N., Weck, D.: A cross-platform SMIL2.0 authoring tool. ERCIM News **62** (2005)
9. Deltour, R., Roisin, C.: The LimSee3 multimedia authoring model. In: D. Brailsford (ed.) Proceedings of the 2006 ACM Symposium on Document Engineering, DocEng 2006, pp. 173–175. ACM Press (2006). URL http://doi.acm.org/10.1145/1166160.1166203
10. easyDITA: Content management, authoring and production all in one. URL http://easydita.com/2011/
11. Furuta, R., Quint, V., André, J.: Interactively editing structured documents. Electronic Publishing **1**, 19–44 (1988)
12. Hill, C., Yates, R., Jones, C., Kogan, S.L.: Beyond predictable workflows: enhancing productivity in artful business processes. IBM Syst. J. **45**, 663–682 (2006)

13. Huynh, D.F., Karger, D.R., Miller, R.C.: Exhibit: lightweight structured data publishing. In: Proceedings of the 16th international conference on World Wide Web, pp. 737–746. ACM, New York, NY, USA (2007)
14. Ko, A.J., Myers, B.A.: Citrus: a language and toolkit for simplifying the creation of structured editors for code and data. In: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, pp. 3–12. ACM, New York, NY, USA (2005)
15. Laiola Guimaraes, R., Monteiro de Resende Costa, R., Gomes Soares, L.: Composer: Authoring tool for iTV programs. In: M. Tscheligi, M. Obrist, A. Lugmayr (eds.) Changing Television Environments, *Lecture Notes in Computer Science*, vol. 5066, pp. 61–71. Springer Berlin / Heidelberg (2008)
16. Lumley, J., Gimson, R., Rees, O.: Configurable editing of XML-based variable-data documents. In: Proceeding of the eighth ACM Symposium on Document Engineering, DocEng '08, pp. 76–85. ACM, New York, NY, USA (2008)
17. Pokorný, J.: XML in enterprise systems. Informatica **20**, 417–438 (2009)
18. Quint, V., Roisin, C., Sire, S., Vanoirbeek, C.: From templates to schemas: Bridging the gap between free editing and safe data processing. In: DocEng 2010: Proceedings of the 10th ACM Symposium on Document Engineering, pp. 61–64. ACM (2010). URL http://doi.acm.org/10.1145/1860559.1860572
19. Quint, V., Vatton, I.: Grif: An Interactive System for Structured Document Manipulation, pp. 200–213. Cambridge University Press (1986)
20. dos Santos, J., Muchaluat-Saade, D.: Xtemplate 3.0: spatio-temporal semantics and structure reuse for hypermedia compositions. Multimedia Tools and Applications pp. 1–29 (2011). URL http://dx.doi.org/10.1007/s11042-011-0732-2
21. dos Santos, J.A., Muchaluat-Saade, D.C.: XTemplate 3.0: adding semantics to hypermedia compositions and providing document structure reuse. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, pp. 1892–1897. ACM, New York, NY, USA (2010)
22. Sire, S.: XTiger XML language specification (2010). URL http://media.epfl.ch/Templates/XTiger-XML-spec.html
23. Sire, S., Vanoirbeek, C., Quint, V., Roisin, C.: Authoring XML all the time, everywhere and by everyone. In: Proceedings of XML Prague 2010, pp. 125–149. Institute for Theoretical Computer Science (2010)
24. Soares, L.F.G., Rodrigues, R.F., Muchaluat Saade, D.C.: Modeling, authoring and formatting hypermedia documents in the hyperprop system. Multimedia Systems **8**, 118–134 (2000)
25. Vuorimaa, P.: Timesheets JavaScript Engine (2007). URL http://www.tml.tkk.fi/˜pv/timesheets/
26. Vuorimaa, P., Bulterman, D., Cesar, P.: SMIL Timesheets 1.0. W3C Recommendation (2008). URL http://www.w3.org/TR/timesheets/
27. XOpus: The web based wysiwyg XML editor. URL http://xopus.com/