A SMIL player for any web browser

Ombretta Gaggi and Luca Danese
Dept. of Pure and Applied Mathematics, University of Padua
via Trieste, 63, 35121 Padua, Italy
gaggi@math.unipd.it, ldanese@studenti.math.unipd.it

***Abstract*—SMIL, *Synchonized Multimedia Integration Language*, is a W3C markup language for the definition of complex multimedia presentations. SMIL documents need a specific player for its playback and cannot be rendered by modern browsers. In this paper we present *SmilingWeb*, a first tentative to implement a cross-platform player for SMIL presentations contained in web pages. We implement a JavaScript library, based on the web standards, which allows to solve the synchronization of media items contained in multimedia presentations through the use of any available browser. The proposed solution is cross-platform and cross-browser, therefore it can be potentially used by any user. The player has been tested with the SMIL Testsuite, provided by W3C and with a set of very complex multimedia presentations in order to check its support to the standard and its scalability. All the tests reported positive results.**

I. INTRODUCTION

SMIL [1], *Synchonized Multimedia Integration Language*, is a W3C standard that allows to create hypermedia presentations. SMIL describes both the temporal behavior of media items contained in a multimedia presentation and their spatial layout. Moreover, the playback of the media objects may be completely changed by the user, who can follow a link, click on an image, or even simply move the mouse over or out of an item.

Even if SMIL was not intended as a substitute for Adobe Flash®, it can be a valid alternative for simple animations or other synchronization of multimedia objects contained into web pages. However, more than ten years have passed after the first definition of this language in 1998, SMIL is used, as examples, to describe interaction in brazilian digital TV, as language to describe the MMS, *Multimedia Message System*, and in the *Daisy digital talking books*, i.e., book accessible to visually impaired users [2], but its diffusion in the web pages is still lower than expected. One of the key problems of this low diffusion is due to the complexity of the authoring activity [3], but also to the lack of a fully-featured SMIL player is a strong obstacle to its adoption. In fact, a SMIL document cannot be rendered by web browsers but needs a suitable player. At the moment, the only available player for the third version of the standard is Ambulant Player [4]. Unfortunately, Ambulat player is a stand alone-application, therefore it cannot be used to render SMIL presentation inside a web page. There exists a plug-in, but it is available only for a limited subset of browsers. Therefore at the moment, the multimedia presentations created with the standard SMIL cannot be rendered on web pages, but only as stand-alone applications. Moreover, the available players are often platform-dependent, therefore it does not exist a solution which works on all operating systems. Since web users are very heterogeneous in terms of used device, operating system and browser, this is a big problem.

In this paper we present *SmilingWeb*, a first tentative to implement a cross-platform player for SMIL presentations contained in web pages. We implement a JavaScript engine, based on the web standards, to allow the playback of multimedia presentations through the use of any available browser. The proposed solution is cross-platform and cross-browser, therefore it can be potentially used by any user.

The possibility to use SMIL in web pages is very important since, differently from Adobe Flash®, this standard can be used to improve accessibility of web pages. As an example, SMIL is suggested to create accessible subtitles for audio or video files. Moreover, SMIL allows to consider different alternative inside a presentation in order to better match users profile.

Our player does not fully support the third version of the standard, since a limited subset of features are not currently implemented, but the state of the implementation allows to render complex, interactive, multimedia presentations. The player has been tested using the SMIL 3.0 Testsuite with positive results.

The paper is organized as follows: Section II discusses background and related work. The implementation of the player is described in Section III and Section IV describes the tests made on the player. Finally, we concluded in Section V.

II. BACKGROUND AND RELATED WORK

A. The SMIL language

A SMIL file is divided in two sections: the `layout` section defines the regions, i.e., rectangular areas on the user screen in which media items are visualized, and the `body` section contains the definitions of media items involved in the presentation, and the temporal relationships among them. The language SMIL also allows to define *transitions*, i.e., visual effects between two objects, and *animations*, i.e., modifications to the value of some attributes (e.g., the color, the size, the position, etc.) of a media item.

SMIL does not define a reference model for the data structure, but only tags for describing media objects behavior. Synchronization is achieved essentially through two tags: `seq` to render two or more objects sequentially, one after the other, and `par` to play them in parallel.

Using attributes `begin`, `end`, and `dur` it is possible to fix the start and the end time of a media item. Consider a media item inside a `par` block. If its attributes `begin`, `end` or `dur` are undefined, the media item starts at the beginning of the `par` block and ends with its natural termination. Otherwise it begins (ends) a certain amount of time after the beginning of the `par` block, given by the corresponding attribute value. In script 1, the attribute `begin = ''intro.activateEvent''` makes audio *artwork* start when the user clicks on video item *intro*. Therefore, the beginning and the end of a media item can be defined with reference to the tag in which it is contained, or according to a particular event, even completely changing the semantics of tags `par` and `seq`, as in the case of script 1. The attribute `dur` defines the duration of an object.

```
⟨par id="par1" dur="60s"⟩
    ⟨video id="intro" end="20s"/⟩                                    (1)
    ⟨audio id="artwork" begin="intro.activateEvent" dur="30s"/⟩
    ⟨img id="picture" begin="artwork.begin+5"/⟩
⟨/par⟩
```

The tag `excl` is used to model some user interactions. It provides a list of children elements, and only one of them may play at any given time. We refer to [1], [2] for more details about this standard.

B. Related work

Since SMIL's first appearance, many authoring tools have been implemented [2] offering their users different facilities like visual editors or preview windows and some players. The first available player was RealNetworks *RealPlayer*® [5]. RealPlayer, and its successor RealOne, implements the SMIL standard up to version 2.1. It was a commercial product, available both *freeware* and as a paid version (RealPlayer Plus). It is a stand alone application available both for Microsoft and Apple operating systems, even if in the second case, the users reported a longer list of bugs. Unfortunately, RealPlayer does not support the third version of the standard.

The only player that support SMIL 3.0 is *Ambulant* [4]. This player is available *freeware* for all operating systems as a stand alone application, or as a plug-in for the Mozilla FireFox and Apple Safari[1]. Tests made has shown that, despite the large support of the numerous features of the language, the application often does not work as soon as the complexity of the synchronization of media items involved in the presentations increased, therefore, the player is often unusable.

Valente et al [6] consider the problem of finding out temporal conflicts into SMIL documents and implement a player, based on a formal description of SMIL elements through automata, which enables the generation of a valid scheduling for rendering, considering QoS problem. The authors compare different players' behaviors in case of temporal inconsistencies and denote that they are implementation-dependent. This approach is not extensible: the automaton which describes the obtained behavior needs to be re-built after any changes.

Microsoft Internet Explorer® supported a selection of SMIL Boston[2] components till version 6.

Unfourtunately, the implementation of SMIL was abandoned in Internet Explorer 7 in favor of SAMI, *Synchronized Accessible Media Interchange* [7], a technology to provide closed captioning to a wide range of multimedia products, which works with Microsoft software only.

Some tentative approaches to create a cross-browsers SMIL player are documented in [8], but the proposed solutions, Soja and S2M2, are limited to the first version of the standard, therefore of no practical use.

Therefore at the moment, SMIL multimedia presentation can be played as a stand alone application with Ambulant player in any operating system, but it cannot be incorporated into web page, even if, both XHTML and SMIL are XML languages and therefore can be used inside the same document. This limitation is due to the state of software, i.e., browsers, implementation and not to the SMIL specification, therefore we think that future implementations of the standard will overcome this limitation.

We must note here that supporting SMIL is a very complex task, since this language provides a good expressivity, i.e. it allows to define rich and interactive multimedia documents, thus mastering its timing relations is not easy. To the best of our knowledge, no browsers support SMIL natively (only versions 5.5 and 6 of Microsoft Internet Explorer partially did in the past), but some of them declare to plan to support this standard in the future. Moreover, there exist JavaScript implementations of few SMIL modules (e. g. JavaScript implementation of smilText is available in the web page of the Ambulant Player) which make documents, using that modules, accessible with any browser.

III. SmilingWeb

SmilingWeb is a web player for multimedia presentations designed with the SMIL standard implemented as a JavaScript library. We aim at developing a solution suitable for all available browsers, therefore we paid particular attention to their differences in terms of support to HTML5 and other solutions like CSS3 and AJAX. The first problem that we have to consider was the possibility to play a video or an audio file without the need for a plug-in. HTML5 [9] offers the ability to easily embed media into HTML documents, using `<audio>` and `<video>` elements, but it is still a W3C working draft, and its support is still limited. Moreover, modern browsers provide support for these tags, but for a very limited subset of audio and video codec. For this reason, our player uses Modernizr [10], a small and simple JavaScript library that helps to take advantage of the emerging web

[1] A plug-in for Internet Explorer is currently available in the Ambulant web pages, but at the time of writing it cannot be installed on all available versions of Internet Explorer.

[2] SMIL Boston was a preliminary version of SMIL 2.0, supported by Microsoft, which aims at introducing transition effects and animations that are not included in the first version of this standard.

technologies (CSS3, HTML5) while still maintaining a fine level of control over older browsers that may not yet support these new technologies. Modernizr allows *SmilingWeb* to play a video or an audio file natively, thanks to HTML5 features, or to require a plug-in only if the browser does not support it. Modernizr also allows to avoid code forking and to design an application compatible with future development of browsers, since it does not test the browser name and version, but it tests its capability. This means that, when all browsers will support HTML5 features, *SmilingWeb* will not need any redesign.

A similar consideration can be done with the support for CSS3, which should be very useful to implement some transition effects. Differently from HTML5, new browsers begin to support CSS3 properties, but not using the standard notation, but adding a particular prefix related to the specific rendering engine[3]. Moreover, the set of transition effected supported deeply varies from browser to browser. This means that, the use of CSS3 implies that the transition effects applied to media objects can be present or not according to the browser. For this reason we decide to use another solution, the well-known JavaScript library jQuery [11], since the browsers support to CSS3 is still inadequate. Future versions of *SmilingWeb* will probably adopt the CSS3 standard as soon as browsers support will be sufficient. Moreover, jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions, therefore it also used to implement SMIL animations.

The *SmilingWeb* player has been designed to support SMIL 3.0. It does not implement all the tags of the language, but the subset of unsupported tags is limited. In particular, it supports:

- all tags and attributes of the Layout Module, with the only exception of the tag `regPoint` and the attribute `soundLevel`[4];
- all tags and attributes of the Linking Module, with the only exception of attributes used to control the volume;
- tags `img`, `video`, `audio`, `brush`, `text` and `ref` for definition of media items;
- all tags of the Timing Module, with the only exception of `priorityClass`;
- all tags of the Animation Module and
- the tag `transition` for the transition effects.

The complete set of supported tags and attributes is detailed in Table I.

Moreover, the player supports events handling and the temporal synchronization of text through smilText. We must note here that the player does not support all video and audio codecs, but this limitation is due to the set of codecs supported by the browsers and the plug-in. In fact, if the browser does not support HTML5, the flowplayer plug-in [12] is used, an Open Source (GPL 3) video player for the web.

[3]E.g. the property `border-radius` becomes `-moz-border-radius` for Mozilla FireFox while Apple Safari and Google Chrome have recently replaced their `-webkit-border-radius` with the standard property.

[4]We must note here that this attribute does not work even in commercial products like RealPlayer.

Supported SMIL elements		
LEGEND: LM = Layout Module, MM = Media Module, LkM = Linking Module, TM = Timing Module, AM = Animation Module, TrM = Transition Module, ST = SmilText Module, *= partial support		
Tags	LM	root-layout, region
	MM	text, img, video, audio, animation, brush, ref
	LkM	a, area
	TM	par, seq, excl
	AM	animate, animateMotion, animateColor, set
	TrM	transition*
	ST	smilText, tev, br, clear, div, p, span
Attributes	LM	height, width, backgroundColor, top, left, bottom, right, z-index, backgroundOpacity*, backgroundImage, backgroundRepeat, fit, showBackground
	MM	src, id, region, fill*, color, transition
	LkM	shape, coords*, sourcePlaystate, href, alt, tabindex, accesskey, show*
	TM	begin, end, dur, repeatCount*
	AM	targetElement, attributeName, from ,to, values
	TrM	transIn*, direction, dur, subtype, type
	ST	next, textAlign, textBackgroundColor, textColor, textFontFamily, textFontSize, textFontStyle, textFontWeight
Admitted Values	MM	fill: transition
	LkM	coords: a list of positive integer
	TM	begin: a positive time value t, an event ev, ev+t end: a positive time value t, an event ev, ev+t, indefinite dur: a positive time value t, indefinite

TABLE I
SMIL TAGS, ATTRIBUTES AND VALUES, SUPPORTED BY *SmilingWeb*

A. The scheduler

The most important module of *SmilingWeb* is the scheduler, i.e., the engine that solves the synchronization constraints of the presentation to find out the correct begin and end time of each element to be scheduled. By *element* we mean all SMIL objects which can be synchronized, i. e., media items, smilText tags, animations or transition effects. A *task* is therefore the set of operations needed to start an element.

The scheduler must be *correct* and *efficient* to avoid the user lies in wait for long time. Moreover, efficiency is particularly important in case of synchronization due to user interactions: it may happen that the scheduler had to re-calculate the entire scheduling after an event like the user clicking on an image. In this case, the playback could be paused or be subject to delays if the computation does not end in time. While the user is used to wait the beginning of playback of multimedia presentations, long response times after interactions are poorly tolerated.

The scheduler engine loads the SMIL file only once and saves all the information about media, transitions and animations into a tree structure. For each element, the collected information are the id, a pointer to the tag in which it is contained, if available, and the start and end time. If the element `begin` and `end` attributes contain a time value, they are simply saved in the tree structure, otherwise, they will be calculated according to their definitions.

SMIL language allows to synchronize elements according

to two kinds of events, *internal* events, i.e., the begin or end time of another element, and *external* events, i.e., user's interactions, e.g., the user clicking on an image or keying a key on the keyboard. The first kind requires the scheduler to search the tree for the id of the referred object and simply retrieve the time value of its start (or end). The second kind is very different since, if the definition of a SMIL tag is bound to a user event for its start or end time, they must be solved *on-the-fly* during playback, therefore the scheduler does not return a point in time but the event contained in the definition.

Before the calculation of the start and end time of each element, the player pre-loads the images and other media files contained in the presentation, so to avoid pause during presentation rendering to wait for distributed media items[5].

Synchronization constraints are solved by the function $findRendering()$ which recursively considers all the tags of the presentation. Initially, it receives as input the entire SMIL file, and creates the tree structure. Then it calculates the start/end times of elements, starting by the first synchronization tag contained in `body` and current time instant equal to 0. More precisely, if the element el does not contain any references to external events, $findRendering(el, currentTime)$:

1) calculates el start time,
2) calculates el end time,
3) recursively applies $findRendering(c_i, time_i)$ to all the element's children c_i, with current time instant
   - equal to the one received as input in two cases, if el is a `par` or an `excl` tag, i.e., $\forall i\ time_i = currentTime$, or the current child is the first child of a `seq`, i.e., $c_1 = currentTime$;
   - equal to the time instant in which the previous child ends otherwise,
4) adjusts, if necessary, the end time of el[6].

Calculating the start and the end time of a SMIL tag is a complex task, since both values may depend on offset, internal and external events. A detailed description of different possible combinations of attributes' values are reported in [13]. We report here only a brief, and not complete, description of the algorithm chosen to solve the synchronization constraints introduced by SMIL tags.

The start time of the element is obtained by adding the offset contained in the attribute `begin` to the current time instant received as input. If the attribute `begin` refers to an internal event, the schedule retrieves the time instant in which that event occurs and calculate the start time of the element. The end time of an element is calculated as follows:

1) it is equal to the current time instant plus the value of the attribute `end`,
2) it is equal to the start time plus the value of the attribute `dur`,
3) it is obtained resolving the time instant in which the referred event occurs and adding the offset (if present).
4) if both the attributes `end` and `dur` are undefined, the element ends together with its father, or if not possible, when all its children have terminated their rendering, if the element is a time container, i.e., a tag `par`, `seq` or `excl`.

The scheduler engine creates a hash table, named the *scheduledTasks*, to insert all the start and end times that are calculated. The scheduledTasks is chronologically sorted and contains, for each time instant returned from the scheduler, the list of tasks, i.e., the elements to start in that point in time. Once the computation of the scheduledTasks is finished, for each time instant, the player:

1) performs the tasks which must be executed at that particular time instant;
2) if an external event has occurred, checks if it affects the synchronization of other elements and, if so, it runs the scheduler engine again to calculate the new scheduledTasks;
3) searches for the next point in time in which a synchronization takes place and sets up a timer to notify the scheduler once that point has been reached;
4) suspends waiting.

Every time the scheduler starts an element, if its end time is available, it also creates a timer to stop it. Therefore many timer can apply to the same media once started, e.g., a timer to stop the media itself and a timer to start the transition effect at the end of the media playback.

The timer used to end elements does not affect the main scheduler, but are managed by the element itself, which acts like a sub-scheduler, in order to improve efficiency. This is particularly useful in case of the tag `smilText`, which can handle the timed text defined inside it, through its inner schedule, thus simplifying the solution of their synchronization constraints even in case of events. We follow the rule *"divide et impera"*: the tag `smilText` is considered like any other element by the main scheduler, and the inner schedule applies a set of simplified rules to its children to find out their start and end times.

Even events bound to user interaction are managed through the use of a secondary scheduler. Each time an event occurs, the player calculates the start and end times of the set of elements affected by that interaction, and creates a secondary scheduler which managed the tasks related to those elements. This choice has been made since a second interaction of the same kind (e.g., a user who clicks twice on the same image) causes a second start of the same set of elements. The use of a separated scheduler allows to easily stop the first instance of the scheduler, and analogously the n^{th} instance, and to create a new one with the new value of the time instant in which the event has occurred. All these operations are performed without affecting the main scheduler which can continue working, without delays.

For efficiency purpose, when the scheduledTask must be re-

[5]We must note here that while this operation is trivial in case of images, the situation is more complicated for continuous media file, especially for videos which can be streamed over the network. We plan to better consider this issue in future works.

[6]Consider the case in which a `seq`, or a `par` tag, ends together with its last child.

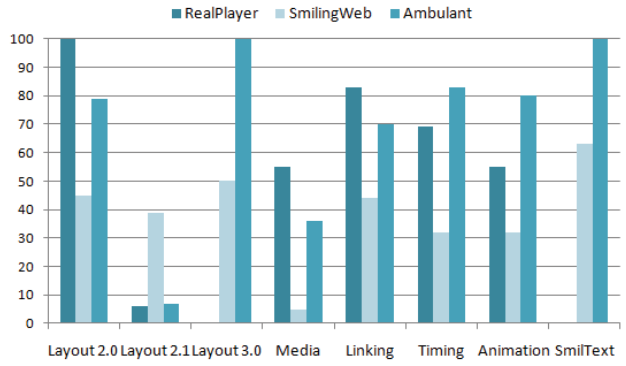

Fig. 1. Comparative analysis of the players with the SMIL 3.0 Testsuite

calculate due to a user interaction, the scheduler does not re-calculate the entire hash table, but only from the current time instant, i.e., the time instant in which the user interacts with the presentation. In this way, elements which are rendered in the past, no longer useful, are not considered again, preserving CPU resources.

IV. System Tests

SmilingWeb has been tested in two different ways. First of all, it was tested using the SMIL 3.0 Testsuite [14] with positive results. We test all the scripts of the testsuite with RealPlayer, Ambulant (both in the stand-alone application and in the plug-in version) and *SmilingWeb*. For each test, we rate the result with 0, 1 or 2 points depending on whether each player does not support tags or attributes contained in the script, experiments some problems, i.e., it provides only a partial support, or passes the test without errors. The result is shown in Figure 1, where the vertical axis represents the percentage of passed tests, and the horizontal axis divides tests according to the SMIL modules.

Figure 1 shows that, although RealPlayer's support to SMIL 2.0 features is very high, e.g. it passes 100% of tests on Layout 2.0, and 83% of tests on Linking module, its support to new features of the third version of the standard is very low or absent (e.g., it does not support SmilText). On the hand, *SmilingWeb* provides a good support to SmilText and Layout 3.0 Module (respectively 63% and 50% of passed tests). Also the support to the Timing, Linking and Animation Modules is rather good. We must note here, that, the score obtained by *SmilingWeb* is deeply influenced from how the scripts are designed. In particular, many tests regarding the Media Module used the attributes `clipBegin` and `clipEnd` which are not supported by our player. This means that *SmilingWeb* does not passed the tests, even if the particular element which the script wants to test is supported. Therefore the testsuite does not completely shows the level of support to SMIL of *SmilingWeb*.

Another consideration must be done: RealPlayer and Ambulant player are stand-alone applications, while *SmilingWeb* can be used to insert multimedia presentation into web pages. Although the authors of Ambulant player claim that a plug-in version of the player is provided, at the time of writing, it works only for Mozilla FireFox, therefore it is of no practical use on the web, where it is not possible to foresee the user browser.

Summing up, *SmilingWeb* passed about 40% of tests made. Despite this percentage is lower than other player, our player has been tested with Microsoft Internet Explorer (version 8 and 9), Mozilla Firefox (version 3.6 and 4), Google Chrome (version 10), and Opera (version 11), and works well with all browsers. All the performed tests are reported in [15].

A second series of tests consists in the execution of more complex multimedia presentations. In fact, the SMIL Testsuite allows to test what features of the standard are implemented, but each test is very simple and usually consists on few rows of code, without any nesting of tags. Our experience, made in more than 5 years of teaching and use of the standard SMIL, shows that, even if Ambulant Player obtained the higher score evaluating it with the testsuite, it is often unusable for complex multimedia presentations.

We test the player with five multimedia presentations, with different degrees of complexity. The presentations used as tests can be viewed at [15]. On average, the SMIL documents used contain about 200 lines of code (with a maximum of 443 lines) and the maximum level of nesting is 4. As an example, *"Focus on Elisa"* is a multimedia presentation about an italian singer. It begins with 36 colored blocks which move on the screen to compose the writing *"Focus on Elisa"* (see Figure 2), while the audio of a song of the singer plays in background. This intro was created with about 150 animations, 50 in parallel at the same time. Moreover, the documents also contain some menus or buttons with which the user can modify the normal behavior of the presentation. Other documents contain a karaoke with text animations synchronized with audio and images with transition effects. Another example describes the trip of the Fellowship of the Ring in the *'Lord of the Rings'* motion picture, through the use of text, images and animation of the path in a map.

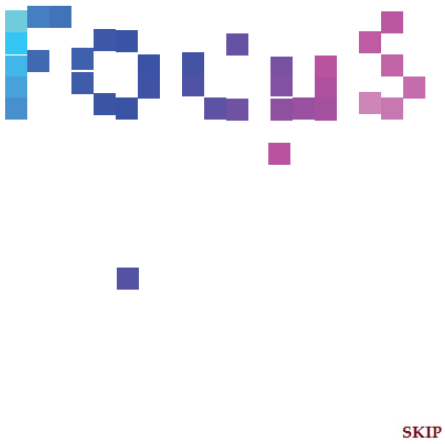

Fig. 2. Screenshot from the multimedia presentation *"Focus on Elisa"*

Even this second group of tests gives good results since the player is able to schedule and synchronize all the elements without any delay during rendering, even in case of user interactions. The user experiences a little delay before the presentation starts, but it is comparable to the loading time of common web pages. We test the presentations also with other available player, and we note that, when the presentation complexity increases, the Ambulant player begins to introduce delays or even errors in the synchronization of elements. As an example, the playback of *"Focus on Elisa"* described above, often experiences pauses and delays.

V. Conclusion

In this paper we have presented *SmilingWeb*, a JavaScript player which allows to reproduce SMIL scripts contained in web pages. The player has been tested with the SMIL Testsuite, provided by W3C and with a set of very complex multimedia presentations in order to check its support to the standard and its scalability.

All the tests reported positive results. However, we must note here that, even if *SmilingWeb* meets all the requirements in terms of scheduling of elements, in case of low network bandwidth, it is not always able to effectively calculate the correct interval of time needed to pre-load the elements. This is particulary true for continuous media like audio and video files. Therefore, in this case, it may happen that the user experiences some pauses during playback. This situation never happens for local document, or with an adequate network bandwidth. Therefore, the scheduling algorithm is correct, but we need to improve the analysis of the state of the network in order to calculate the correct time to start the presentation to have a good chance that all media items involved will be received on time. We plan to better analyze this issue in future works.

Another consideration must be done: although the other available players may have a more complete support to SMIL features, *SmilingWeb* is the first tentative to create a player for SMIL documents which can be used with all available browsers. Moreover, the technology used to implement the player have been chosen to ensure its compatibility also with future versions of the player, since we used web standards promoted by W3C.

The possibility to insert SMIL tags into web pages is particularly important, since it allows to apply synchronization to HTML tags, e.g., it allows to create a text moving around the screen, or to apply animations and transition effects without knowing JavaScript or Flash. Moreover, *SmilingWeb* allows to create accessible animations, or text description for audio and video files.

Concluding, we think that the possibility to use together XHTML and SMIL language allows the author to overcome some limitations typical of the two languages: in fact, SMIL allows the introduction of synchronization between media, but impose a fixed layout. XHTML and CSS on the other hand, define richer mechanism for layout definition, e.g., the definition of fluid layouts, i. e. web pages which adapt themselves to the size and resolution of the window on the user's screen.

References

[1] Bulterman et al, "Synchronized Multimedia Integration Language (SMIL) 3.0 Recommendation," December 2008. [Online]. Available: http://www.w3.org/TR/SMIL3/

[2] D. Bulterman and L. Rutledge, *SMIL 3.0, Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*, 2nd ed. Springer, 2009.

[3] D. Bulterman and L. Hardman, "Structured Multimedia Authoring," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 1, pp. 89–109, 2005.

[4] Ambulant Open SMIL Player, "http://www.ambulantplayer.org/," 2009.

[5] RealNetworks, "RealPlayer 10.5," http://www.real.com/.

[6] P. Valente and P. Sampaio, "TLSA Player: A tool for presenting consistent SMIL 2.0 documents." in *Proc. of ICEIS2007*, Madeira, Portugal, June 2007.

[7] Microsoft Corporation, "Understanding SAMI 1.0," http://msdn.microsoft.com/en-us/library/ms971327.aspx, February 2003.

[8] SYMM Working Group, "Synchronized Multimedia - Players," http://www.w3.org/AudioVideo/♯SMIL.

[9] Ian Hickson, "HTML5, W3C Working Draft," http://www.w3.org/TR/html5/, April 2011.

[10] Faruk Ates, Paul Irish and Alex Sexton, "Modernizr," http://www.modernizr.com/.

[11] The jQuery Project, "jQuery," http://jquery.com/.

[12] Flowpalyer Ltd., "flowplayer," http://flowplayer.org/.

[13] A. Bossi and O. Gaggi, "Enriching SMIL with assertions for temporal validation," in *Proc. of ACM MM*, September 2007, pp. 107–116.

[14] W. Chang and T. Michel, "SMIL 3.0 Testsuite." [Online]. Available: http://www.w3.org/2007/SMIL30/testsuite/

[15] Ombretta Gaggi and Luca Danese, "SmilingWeb," http://docenti.math.unipd.it/gaggi/smilingweb/.