Summer 7-18-2011

Managing Multiple Media Streams in HTML5: The IEEE 1599-2008 Case Study

S. Baldan

L. A. Ludovico

Davide Andrea Mauro
Marshall University, maurod@marshall.edu

Recommended Citation

Baldan, S., Ludovico, L.A. and Mauro, D.A. "Managing Multiple Media Streams in HTML5: The IEEE 1599-2008 Case Study." International Conference on Signal Processing and Multimedia Applications (SIGMAP 2011), Seville (Spain),18-21 July 2011.

MANAGING MULTIPLE MEDIA STREAMS IN HTML5
The IEEE 1599-2008 Case Study

S. Baldan, L.A. Ludovico and D.A. Mauro
LIM - Laboratorio di Informatica Musicale, Dipartimento di Informatica e comunicazione (DICo)
Universitá degli Studi di Milano, Via Comelico 39/41, I-20135 Milan, Italy
http://www.lim.dico.unimi.it
{*ludovico, mauro*}*@dico.unimi.it, stefano.baldan@studenti.unimi.it*

Keywords: HTML5; Web Player; Media Streaming; IEEE 1599.

Abstract: This paper deals with the problem of managing multiple multimedia streams in a Web environment. Multimedia types to support are pure audio, video with no sound, and audio/video. Data streams refer to the same event or performance, consequently they both have and should maintain mutual synchronization. Besides, a Web player should be able to play different multimedia streams simultaneously, as well as to switch from one to another in real time. The clarifying example of a music piece encoded in IEEE 1599 format will be presented as a case study.

1 INTRODUCTION

The problem of managing digital multimedia documents including audio and video in a Web environment is still a challenging matter. Often this problem is solved by requiring the installation of extensions and third-party plugins into Web browsers. As explained in Section 2, the draft of HTML5 tries to face the problem by adopting a multimedia-oriented architecture based on *ad hoc* tags and APIs. It is worth recalling that currently HTML5 is not a standard but a draft, however it is under development by a W3C working group and the most recent versions of the main browsers available on the marketplace already support its key features. Among others, we can cite Google Chrome, Mozilla Firefox 4, Opera 11, and Microsoft Internet Explorer 9.

HTML5 provides Web designers with some syntactical instruments to support multimedia streams natively. An interesting aspect that has not been deeply explored yet is the simultaneous transmission over the Web of a number of multimedia streams, all related to the same object and having constraints of mutual synchronization. On one hand, traditional players for audio/video streams implemented through HTML5 are already available, as the environment offered by this (future) standard makes this activity easy. On the other hand, the problem analyzed here goes one step beyond, as the multimedia player has to support multiple synchronized streams and it has to implement advanced seek functions.

2 RELATED WORKS

In this section some related works based both on HTML 5 or other platforms are presented. Considering the relative novelty of HTML5, most approaches focus on the use of competing technologies such as Adobe Flash and Microsoft Silverlight platforms.

- In (Daoust et al., 2010) the authors first analyze the new possibilities opened by <video> tag, then they focus on the different strategies for streaming media on the Web, such as HTTP Progressive Delivery, HTTP Streaming, and HTTP Adaptive Streaming;
- (Harjono et al., 2010) studies the matter in depth, pointing out how open standards can reduce the dependency on third party products such as Adobe Flash or Microsoft Silverlight;
- MMP Video Editor is a browser-based video editor that consumes all Silverlight supported codecs and produces a project file that can be transformed into an EDL (Edit Decision List) or into a Smooth Streaming Composite Stream Manifest (CSM). It relies on Microsoft components such as IIS (Internet Information Services) and Windows Media Services in order to stream media. For instance, this software framework was used in an initiative

where videos of theatrical performances are presented in a synchronized fashion with related materials (Baraté et al., 2011);

- (Faulkner et al., 2010) presents a survey on Web video editors, comparing in particular EasyClip to Youtube Video Editor;
- (Vaughan-Nichols, 2010) is a general introduction to the new possibilities offered by HTML5;
- In (Laiola Guimarães et al., 2010) the main concern is the annotation within videos. For our goals, this feature can be used for the presentation of lyrics synchronized with audio/video contents. The paper compares the possibilities of HTML5 with respect to NCL and SMIL;
- (Pfeiffer and Parker, 2009) poses the accent on the accessibility of tag <video>;

What clearly emerges from this overview is the absence of scientific works focusing on multiple media streams, e.g. the simultaneous presence of an audio and a video track to be synchronized within a media player as well as the management of a number of alternative media.

3 THE IEEE 1599 FORMAT

IEEE 1599-2008 is a format to describe single music pieces. For example, an IEEE 1599 document can be related to a pop song, to an operatic aria, or to a movement of a symphony.

Based on XML (eXtensible Markup Language), it follows the guidelines of IEEE P1599, "Recommended Practice Dealing With Applications and Representations of Symbolic Music Information Using the XML Language". This IEEE standard has been sponsored by the Computer Society Standards Activity Board and it was launched by the Technical Committee on Computer Generated Music (IEEE CS TC on CGM) (Baggi, 1995).

The innovative contribution of the format is providing a comprehensive description of music and music-related materials within a unique framework. In fact, the symbolic score - intended here as a sequence of music symbols - is only one of the many descriptions that can be provided for a piece. For instance, all the graphical and audio instances (scores and performances) available for a given piece are further descriptions; but also text elements (e.g. catalogue metadata, lyrics, etc.), still images (e.g. photos, playbills, etc.), and moving images (e.g. video clips, movies with a soundtrack, etc.) can be related to the piece itself. Please refer to (Haus and Longari, 2005) for a complete treatment of the subject. Among other applications, such a rich description allows the design and implementation of advanced browsers.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM
 "http://standards.ieee.org/downloads/1599/1599
                          -2008/ieee1599.dtd ">
<ieee1599>
  <general>...</general>
  <logic>...</logic>
  <structural>...</structural>
  <notational>...</notational>
  <performance>...</performance>
  <audio>...</audio>
</ieee1599>
```

Figure 1: The XML stub corresponding to the IEEE 1599 multi-layer structure.

3.1 Definition of Music Event

The mentioned comprehensiveness in music description is realized in IEEE 1599 through a multi-layer environment. The XML format provides a set of rules to create strongly structured documents. IEEE 1599 implements this characteristic by arranging music and music-related contents within six layers (Ludovico, 2008):

- *General* - music-related metadata, i.e. catalogue information about the piece;
- *Logic* - the logical description of score in terms of symbols;
- *Structural* - identification of music objects and their mutual relationships;
- *Notational* - graphical representations of the score;
- *Performance* - computer-based descriptions and executions of music according to performance languages;
- *Audio* - digital or digitized recordings of the piece.

In IEEE 1599 code, this 6-layers layout corresponds to the one shown in Figure 1, where the root element `IEEE 1599` presents 6 sub-elements.

Since contents are displaced over various levels, what is the device that keeps heterogeneous descriptions together and allows to jump from one description to another? The *Logic* layer contains an *ad hoc* data structure that answers the question. When a user encodes a piece in IEEE 1599 format, he/she must specify a list of music events to be organized in a linear structure called "spine". Please refer to Figure 2 for a simplified example of spine. Inside this structure, music events are uniquely identified by the `id` attribute, and located in space and time dimensions through `hpos` and `timing` attributes respectively.

Each event is “spaced” from the previous one in a relative way. In other words, a 0 value means simultaneity in time and vertical overlapping in space, whereas a double value means a double distance from the previous music event with respect to a virtual unit. The measurement units are intentionally unspecified, as the logical values expressed in spine for time and space can correspond to many different absolute values in the digital objects available for the piece.

Let us consider the example shown in Figure 2, interpreting it as a music composition. Event `e3` forms a chord together with `e2`, belonging either to the same or to another part/voice, as the attributes’ values of the former are 0s. Similarly, we can affirm that event `e3` happens after `e0` (and `e1`), as `e4` occurs after 2 time units whereas `e1` (and `e2`) occurs after only 1 time unit. For further details please refer to the official document about IEEE 1599 standard (IEEE TCCGM, 2008).

In conclusion, the role of the structure known as spine is central for an IEEE 1599 encoding: it provides a complete and sorted list of events which will be described in their heterogeneous meanings and forms inside other layers. Please note that only a correct identification inside spine structure allows an event to be described elsewhere in the document, and this is realized through references from other layers to its unique `id` (see Section 3.2). Inside the spine structure only the entities of some interest for the encoding have to be identified and sorted, ranging from a very high to a low degree of abstraction.

In the context of music encoding in IEEE 1599, how can be a *music event* defined from a semantic point of view? One of the most relevant aspects of the format, which confers both descriptive power and flexibility, consists in the loose but versatile definition of event. In the music field, which is the typical context where the format is used, a *music event* is a clearly recognizable music entity, characterized by well-defined features, which presents aspects of interest for the author of the encoding. This definition is intentionally vague in order to embrace a wide range of situations. A common case is represented by a score where each note and rest are considered music events. The corresponding spine will list such events by as many XML sub-elements (also referred as *spine events*).

However, the interpretation of the concept of music event can be relaxed. A music event could be the occurrence of a new chord or tonal area, in order to describe only the harmonic path of a piece instead of its complete score, made of notes and rests.

```
<ieee1599>
  ...
  <logic>
    <spine>
      <event id="e0" timing="0" hpos="0" />
      <event id="e1" timing="1" hpos="1" />
      <event id="e2" timing="1" hpos="1" />
      <event id="e3" timing="0" hpos="0" />
      <event id="e4" timing="2" hpos="2" />
      <event id="e5" timing="2" hpos="2" />
      ...
    </spine>
    ...
  </logic>
  ...
</ieee1599>
```

Figure 2: An example of simplified spine.

3.2 Events in a Multi-layer Environment

After introducing the concept of spine event, and after the creation of the spine structure, events are ready to be described in the multi-layer environment provided by IEEE 1599.

This section will show that the concept of heterogeneous description is implemented in IEEE 1599 by heterogeneous descriptions of each event contained in spine. As stated in Section 3, the format includes six layers. While heterogeneity is supported by the variety of layers, inside each layer we find homogeneous contents, namely contents of the same type. The *Audio* layer, for example, can link *n* different performances of the same piece. In order to obtain a valid IEEE 1599 document, not all the layers must be filled; however their presence provides richness to the description.

In musical terms, the layer-based mechanism allows heterogeneous descriptions of the same piece. For a composition, not only its logical score, but also the corresponding music sheets, performances, etc. can be described. In this context instead, heterogeneity is employed in order to provide a wide range of audio descriptions of the same environment. This concept will become clear in the following.

Now let us focus on the presence and meaning of events inside each layer. The *General* layer contains mainly catalogue metadata that are not referable to single music events (e.g. title, authors, genre, and so on).

The *Logic* layer, which is the core of the format, provides music description from a symbolic point of view: it contains both the spine, i.e. the main time-space construct aimed at the localization and synchronization of events, and the symbolic score in terms of pitches, durations, etc.

Originally, the *Structural* layer was designed to contain the description of music objects and their causal relationships, from both the compositional and the musicological point of view. This layer is aimed at the identification of music objects as aggregations of music events and it defines how music objects can be described as a transformation of previously described objects. As usual, event localization in time and space is realized through spine references.

For the remaining layers, the meaning of events is more straightforward. The *Notational* layer describes and links the graphical implementations of the logic score, where music events - identified by their spine id - are located on digital objects by absolute space units (e.g. points, pixels, millimeters, etc.). In the case of environmental sounds, the places where they are recorded can be identified over a map. These maps can be the counterpart of the graphical scores as regards our work.

The *Performance* layer is devoted to computer-based performances of a piece, typically in sub-symbolic formats such as Csound, MIDI, and SASL/SAOL. This layer is not used for our goals.

In the *Audio* layer events are described and linked to audio digital objects. Multiple audio tracks and video clips, in a number of different formats, are supported. The device used to map audio events is based on absolute timing values expressed in milliseconds, frames, or other time units.

Finally, let us concentrate layer by layer on the cardinalities supported for events. In the *Logic/Spine* sub-layer, the cardinality is 1 - namely the presence is strictly required - as all the events must be listed within the spine structure. In the *Audio* layer, on the contrary, the cardinality is $[0..n]$ as the layer itself can be empty (0 occurrences), it can encode one or more partial tracks where the event is not present (0 occurrences), it can link a complete track without repetitions (1 occurrence), a complete track with repetitions (n occurrences), and finally a number of different tracks with or without repetitions (n occurrences). Similarly, the *Notational* layer supports $[0..n]$ occurrences.

In the case study described in this paper, we will concentrate on an IEEE 1599 document having multiple videos, namely multiple instances inside the *Audio* layer.

4 THE IEEE 1599 WEB PLAYER

The IEEE 1599 format has been already used with success to synchronize multiple and heterogeneous media streams within domain-specific applications, in an offline scenario. In this chapter we will see how Javascript and HTML5 together (especially the new audio and video APIs) can be used to build a streaming Web player, which is able to support advanced multimedia synchronization features in a Web environment, running inside a common browser.

4.1 Preparing IEEE 1599 for the Web

The IEEE 1599 standard supports a huge variety of container formats and codecs for its media streams, greater than what can be handled by any HTML5 browser implementation. An almost complete list of what is currently supported by the most popular browsers can be found in (Pilgrim, 2010). For this reason, audio and video streams referred inside an IEEE 1599 file must be first converted into an appropriate format to be consumed by browsers.

For this particular case study a Python script has been written to load the IEEE 1599 file inside a DOM (Document Object Model). It looks for any reference to audio or video files and automatically converts them into Theora video and Vorbis audio format inside an Ogg container. This particular combination has been chosen because its components are all open standards and also because it is well supported inside browsers such as Mozilla Firefox, Google Chrome and Opera. Other combinations may be supported in the next future to extend compatibility to a wider range of browsers.

Other related media are not converted, because they mainly consist of images (e.g. booklets, music sheets, etc.) which have been supported by almost every browser for quite a long time. All media files are then organized inside a uniform folder structure and their references inside IEEE 1599 document are updated. Broken references are deleted, as well as the parts of the IEEE 1599 specification which are not useful for this application.

IEEE 1599 files can be huge: most of them are several megabytes in size and contain thousands lines of XML tags. This is a problem for a streaming player both if we parse the whole document at once and store it inside a DOM and if we use event-driven parsing techniques such as SAX: in the former case we have to download the whole IEEE 1599 file, causing an annoying delay before actually playing something; in the latter we can stream the XML together with media, but this adds a big amount of overhead and reduces considerably the bandwidth available to actual contents.

Compression can help us solve the problem. XML is an extremely verbose and redundant text-based format, so dictionary-based compression works great on this kind of information (usually more than 95%

compression rate). Every modern browser supporting HTML5 also supports HTTP compression, so exchanging compressed data needs little effort. During the preparation phase explained above, IEEE 1599 files are gzip-compressed and stored with a custom file extension (.xgz). The web server is configured to associate the right MIME-type (text/xml) and content encoding (x-gzip) to that file extension. When a client requests an IEEE 1599 file, the server HTTP response instructs the browser to enable its HTTP compression features in order to decompress the file before using it. In this way all the advantages of DOM parsing can be exploited and no overhead is added to the stream, at the cost of a little initial delay.

4.2 Infrastructure and Application Design

After preparing the material to make it usable by a HTML5 browser, let us choose how to make it available over the net. There are two main alternatives: either using a common Web server and adopting the progressive download approach, like almost every "streaming" player for the Web, or setting up a full fledged streaming server. While the latter option permits the use of protocols specifically designed for streaming (like RTP and RTSP) and may therefore be more flexible, our choice has fallen on the former one because it is effective for our purposes, easier to implement and widely used in similar application domains. Moreover, HTTP/TCP traffic is usually better accepted by the most common firewall configurations, and less subject to NAT traversal problems. Finally, Web users seem to be less annoyed by some little pauses during the playback rather than by quality degradation or loss of information.

On the client side, the fundamental choice is what media streams to request, when to request them and how to manage them without clogging the wire or the buffer. Three possible cases have been studied:

- *One stream at a time:* Among all the available contents, just the stream currently chosen by the user for watching or listening is requested and buffered. When another stream is selected, the audio (or video) buffer is emptied and the new stream is loaded. The main advantage of this solution is that only useful data are sent on the wire: at every time, the user receives just the stream he/she requested. The principal drawback, on the other side, is that every time the user decides to watch or listen to other media streams, he/she has to wait a considerable amount of time for the new contents.
- *All the streams at the same time:* All the available contents are requested by the client and sent over the net. This approach drastically reduces delays when jumping from one media stream to another, at the cost of a huge waste of bandwidth caused by the dispatch of unwanted streams. In order to reduce network traffic, contents which are not currently selected by the user may be sent in a low-quality version, and upgraded to full quality only when selected. With this approach, a smooth transition occurs: when the user selects a new media stream, the client instantly plays the degraded version, and switches to full quality as soon as possible, namely when the buffer is sufficiently full.
- *Custom packetized streams:* Borrowing some principles from the piggyback forward error correction technique (Perkins et al., 1998), streams can be served all together inside a single packet, containing the active streams in full quality and the inactive ones in low quality. This implies the existence of a "smart" server, which does all the synchronization and packing work, and a "dumb" client which does not even need to know anything about IEEE 1599 and its structure.

Even if the last option presents a certain interest, it requires a custom server-side application and burdens the server with lots of computation for each client. In this paper we will focus on the first two scenarios, using the first (which is also the simplest) to draw our attention on the synchronization aspects, evolving then to the second to support multiple media streams simultaneously.

4.3 Audio and Video Synchronization

One of the key features of the IEEE 1599 format is the description of information which can be used to synchronize otherwise asynchronous and heterogeneous media. As presented in Section 3.1, every musical event of a certain interest (notes, time/clef/key signature changes etc.) should have its own unique id inside the *spine*. Those identifiers can be used to reference the occurrence of a particular event inside the various resources available for the piece: the area corresponding to a note inside an image of the music sheet, a word in a text file representing the lyrics, a particular frame of a video capturing the performance, a given instant in an audio file, and so on.

For the IEEE 1599 streaming Web player, the *Audio* layer is the most interesting. Each related audio or video stream is represented by the tag <track>, whose attributes give information about its URI and encoding format. Inside each <track> there are many <track_event> tags, which are the actual references to the events in the *spine*. Every <track_event> has two main attributes: the unique id of the related event in the spine, and the time (usually expressed in seconds) at which that event occurs inside the audio or video stream. A known limitation of the application is the following: each event must be referenced inside each track, even if it does not actually take place in that track, otherwise the synchronization mechanism will not work properly (as explained later). For example, the piano reduction track of a symphonic piece should contain the references to all the events related to the orchestra, even if they do not actually happen in that track. A check can be enforced server-side during the preparation step in order to guarantee this condition.

The HTML5 audio and video API already offers all the features needed to implement a synchronized system: media files can be play/paused and sought. On certain implementations (later versions of Google Chrome, for example) there is even the possibility to modify the playback rate, thus slowing down or speeding up the track performance. It is sufficient use the information stored inside the IEEE 1599 file to drive via JavaScript the audio and video objects inside the Web page.

The currently active media stream is used as the master timing source: while it is playing, the client constantly keeps trace of the last event occurred inside the track. When another stream is selected the client loads it, looks for the event of the new track with the same id as the last one occurred, then uses the time encoded for this event to seek the new stream. Consequently, the execution resumes exactly from the same logical point where we left it, and media synchronization is achieved. Since event identifiers are used as a reference to jump to the exact instant inside the new stream, it should now be clear why all the events related to the whole piece should be present inside each track. In other case, we could fall back to the previous (or next) event in common to both tracks, potentially very distant from the exact synchronization point. Please note that such a common event could not exist.

As stated before the event list could be large, so we have to use efficient algorithms to navigate it and find the events we need in a small time amount. Depending on the situation, certain techniques may be more convenient than others:

- During continuous playback of a single stream, a linear search over the event list is maybe the most appropriate for keeping trace of occurring events. Of course, the list has to be sorted by the occurring instant of each event;
- When the user seeks, movements inside the stream become non-linear, thus making linear search inefficient. In this case, a binary search is preferable. It is rare to find an event exactly at the seeking position, so event search must be approximated to the event immediately before (or immediately after) the current instant;
- When changing media streams, events are searched by their unique identifier. Therefore, a dictionary using it as its key becomes really useful.

After the IEEE 1599 file has been downloaded and stored as a DOM by the player, an event list is populated for each track and quick-sorted by occurrence time. Then, for every list, an associative array is used to bind the index of each event to its unique identifier. This data structure enables the use of all the three search techniques mentioned above, thus making the retrieval of events efficient in every possible case.

4.4 Handling Simultaneous Media

The principles explained in the previous subsection can also be applied when handling and playing multiple media streams at a time. In particular, we are referring to a number of video or audio tracks to be performed at the same time. In this case, the track which acts as master will be played smoothly and without interruption, while every other slave track will be periodically sought and/or stretched to keep it synchronized with the master. The choice of the master track can deeply influence the quality of the playback. In particular, it is well known that human perception is more disturbed by discontinuities in acoustic rather than in visual information. If both of them are present at the same time, it is better to choose one of the audio tracks as master.

Resynchronization of slave tracks can happen at fixed intervals or when needed. In the former case a timer is set and all the slave tracks are re-aligned when it expires, following the same procedure explained above for changing media streams. In the latter case, whenever an event occurs in the master track, the corresponding event is searched in all the slave tracks. Repositioning of a slave track occurs when the absolute difference between its playing position and the occurring time of the related event is greater than a certain given threshold.

5 Conclusions and Future Works

In this paper we have illustrated an application to manage multiple media streams through HTML5 in a synchronized way. The IEEE 1599 format represented for us a test case with demanding requirements, such as the presence of heterogeneous materials to keep mutually synchronized. For this kind of applications, our approach based on HTML5 and JavaScript has proved to be effective.

As regards future works, multiple media reproduction in the IEEE 1599 Web player is not supported yet. An experiment of fixed interval repositioning has been made, choosing a period of 2 seconds. While the results of resynchronization of video over a continuous and smooth audio playback are quite acceptable, the audio over audio case produced glitches which are annoying to listen. This problem could be reduced by using the volume control in the HTML5 audio and video API in order to rapidly fade out just before a repositioning and fade in again immediately after. Another improvement may be obtained by dynamically changing the playback rate of the slaves (through the implementations that support this feature). Playback rate can be estimated by comparing the time difference between the next and last event in the master with the time difference between the two corresponding events in the slave.

Other future works may include the implementation of further features like synchronizing and highlighting graphic elements (musical sheets, lyrics etc.), or using information of the *General* layer for music information retrieval and linking purposes.

REFERENCES

Baggi, D. L. (1995). Technical committee on computer-generated music. *TC*, 714:821–4641.

Baraté, A., Haus, G., Ludovico, L. A., and Mauro, D. A. (2011). A web-oriented multi-layer model to interact with theatrical performances. In *International Workshop on Multimedia for Cultural Heritage (MM4CH 2011)*, Modena, Italy.

Daoust, F., Hoschka, P., Patrikakis, C., Cruz, R., Nunes, M., and Osborne, D. (2010). Towards video on the web with HTML5. *NEM Summit 2010.*

Faulkner, A., Lewis, A., Meyer, M., and Merchant, N. (2010). Creating an HTML5 web based video editor for the general user. *CGT 411, Group 13.*

Harjono, J., Ng, G., Kong, D., and Lo, J. (2010). Building smarter web applications with HTML5. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, pages 402–403. ACM.

Haus, G. and Longari, M. (2005). A multi-layered, time-based music description approach based on XML. *Computer Music Journal*, 29(1):70–85.

Laiola Guimarães, R., Cesar, P., and Bulterman, D. (2010). Creating and sharing personalized time-based annotations of videos on the web. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 27–36. ACM.

Ludovico, L. A. (2008). Key concepts of the IEEE 1599 standard. In *Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects, IEEE CS, Lugano, Switzerland.*

Perkins, C., Hodson, O., and Hardman, V. (1998). A survey of packet loss recovery techniques for streaming audio. *Network, IEEE*, 12(5):40–48.

Pfeiffer, S. and Parker, C. (2009). Accessibility for the HTML5 <video> element. In *Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*, pages 98–100. ACM.

Pilgrim, M. (2010). *HTML5: Up and running*. Oreilly & Associates Inc.

IEEE TCCGM (2008). *IEEE Std 1599-2008 - IEEE Recommended Practice for Defining a Commonly Acceptable Musical Application Using XML.* The Institute of Electrical and Electronics Engineers, Inc.

Vaughan-Nichols, S. (2010). Will HTML 5 restandardize the web? *Computer*, 43(4):13 –15.