Making Space & Time Uniformly Identifiable in the Web via Media Fragments

(Invited Paper)

Erik Mannens
Ghent University - IBBT
ELIS - Multimedia Lab
Ghent, Belgium
Email: erik.mannens@ugent.be

Davy Van Deursen
Ghent University - IBBT
ELIS - Multimedia Lab
Ghent, Belgium
Email: davy.vandeursen@ugent.be

Rik Van de Walle
Ghent University - IBBT
ELIS - Multimedia Lab
Ghent, Belgium
Email: rik.vandewalle@ugent.be

***Abstract*—There's a gap in consumer communications on how networks, devices, and services handle and identify space & time in media resources in the Web. W3C's Media Fragments URI specification, which we edited and contributed to over the last couple of years, not only bridges this gap by homogeneously opening up time-related media to the Internet crowd, but also makes time-related annotation feasible for hyperlinking into that media, thus finally also providing the necessary support for this already omnipresent 'third dimension' *time* into the Internet. As such, this uniform approach using the standardised (semantic) web technology stack will improve search, discovery, usage, management, security, and linking of related media resources and its fragments.**

I. Introduction

Media resources on the WWW used to be treated as 'foreign' objects as they could only be embedded using a plugin that is capable of decoding and interacting with these media resources. The HTML5 specification, however, is a game changer and most of the popular user agents have committed to support the newly introduced `<video>` and `<audio>` elements [1][1]. However, to make media, and in particular video, a 'first-class citizen' on the Web, it needs to be as easily linkable as a simple HTML page. In order to share or bookmark only the interesting parts of a video, we should be able to link into or link out of this time-linear media resource. If we want to further meet the prevailing accessibility needs of a video, we should be able to dynamically choose our preferred tracks that are encapsulated within this media resource, and we should be able to easily identify only specific Region Of Interests[2] (ROI) within this media resource. And last but not least, if we want to browse media resources based on (encapsulated) semantics, we should be able to master the full complexity of rich media by also enabling standardised media annotation [2], [3].

The mission of the W3C Media Fragments Working Group (MFWG) [4], which is part of W3C's Video in the Web activity[3], is to provide a mechanism to address media fragments on the Web using URIs [5], [6]. The objective of the proposed specification is to improve the support for the addressing and retrieval of sub-parts of media resources, as well as the automated processing of such sub-parts for reuse within the current and future Web infrastructure [7]. Example use cases are the bookmarking or sharing of media fragment URIs with friends via social network notifications by linking to specific regions of joint images, the automated creation of fragment URIs in search engine interfaces by having selective previews, or the annotation of media fragments when tagging audio and video spatially and/or temporally [8]. The examples given throughout this paper to explain the Media Fragments URI specification are based on the following scenario: Steve –a long-time basketball enthusiast– posts a message on his team's blog containing a Media Fragment URI, that highlights 10 seconds of an NBA video clip showing the same nifty move as he did in last Saturday's game. Needless to say that this Media Fragments specification will have a major impact on the complete end-to-end media production chain. From the moment footage is shot, edited and enriched with extra information down the media production workflow chain, until a specifically chosen clip is viewed by an interested end-user, one will be able to *uniquely identify, link to, display, browse, bookmark, re-composite, annotate, and/or adapt* spatial and/or temporal sub-clips of media resources, e.g., a camera might automatically annotate footage with the exact geo-coordinates the moment it is shot, a news editor might quickly browse through a months' footage by means of highlight captions in search of one particular item, whereas an end-user might create a video mash-up from his bookmarked video segments to share with his friends on a social platform.

The outline of this paper is the following. Firstly, we describe the boundaries and semantics of a Media Fragments URI in Section II and show how the syntax should look like, whereas Section III elaborates on how a media fragment specified as a URI fragment can be resolved stepwise using the HTTP protocol [9]. We then highlight our reference implementation in Section IV. Finally, future work is outlined and conclusions are drawn in Section V.

[1]At the time of writing, the following browsers support the HTML5 media elements: IE 9, Firefox 3.5, Chrome 4, Safari 4, Opera 10
[2]http://en.wikipedia.org/wiki/Region_of_interest
[3]http://www.w3.org/2008/WebVideo/Activity.html

II. Media Fragments URI's

A. Media Resource Model

We assume that media fragments are defined on top of 'time-linear' media resources, which are characterised by a single timeline (see also Figure 1). Such media resources usually include multiple tracks of data all parallel along this uniform timeline. These tracks can contain video, audio, text, images, or any other time-aligned data. Each individual media resource also contains control information in data headers, which may be located at certain positions within the resource, either at the beginning or at the end, or spread throughout the data tracks as headers for those data packets. There is also typically a general header for the complete media resource. To comply with progressive decoding, these different data tracks may be encoded in an interleaved fashion. Normally, all of this is contained within one single container file.

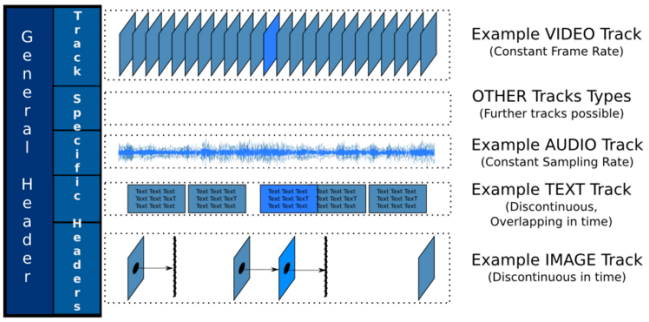

Fig. 1. Video Resource Model.

B. Requirements

We formally define a number of requirements for the identification and access of media fragments. Based on these requirements, we motivate a number of design choices for processing Media Fragments URIs.

- *Independent of media formats*. The Media Fragments URI specification needs to be independent of underlying media formats, such as MP4 [10] or Ogg [11].
- *Fragment axes*. Media fragments need to be accessed along three different axes: temporal, spatial, and track. Additionally, media fragments also could be identified through names.
- *Context awareness*. A media fragment must be a secondary resource of its parent resource. This way, the relationship between the media fragment and its parent resource is kept. Moreover, when accessing media fragments, user agents need to be aware of the context of the media fragment, i.e., where is the fragment located within the original parent resource.
- *Low complexity*. The Media Fragment URI specification should be kept as simple as possible. For instance, defining spatial regions is limited to the definition of rectangular regions which should be sufficient for most applications.
- *Minimize impact on existing infrastructure*. Necessary changes to all software components –be it user agents, proxies, or media servers– in the complete media delivery chain should be kept to a bare minimum. Furthermore, the access to media fragments should work as much as possible within the boundaries of existing protocols, such as FILE, HTTP(S), and RTSP [12].
- *Fragment by extraction*. Preferably, it should be possible to express media fragments in terms of byte ranges pointing to the parent media resource. This makes the fragments real sub-resources of the 'de facto' media resource. Therefore, we consider media segments obtained through a re-encoding process to not be treated as media fragments.

C. Fragment Dimensions

1) Temporal Axis: The most obvious *temporal dimension* denotes a specific time range in the original media, such as 'starting at second 10, continuing until second 20'. Temporal clipping is represented by the identifier '`t`', and specified as an interval with a begin and an end time (or an in-point and an out-point, in video editing terms). If either or both are omitted, the begin time defaults to 0 second and the end time defaults to the end of the entire media resource. The interval is considered half-open: the begin time is part of the interval, whereas the end time on the other hand is the first time point that is not part of the interval. The time units that can be used are Normal Play Time (npt), real-world clock time (clock), and Society of Motion Picture and Television Engineers (SMPTE) timecodes [12], [13]. The time format is specified by name, followed by a colon, with '`npt:`' being the default. Some typical temporal examples can be seen in band 1 of Table I.

2) Spatial Axis: The *spatial dimension* denotes a specific spatial rectangle of pixels from the original media resource. The rectangle can either be specified as pixel coordinates or percentages. A rectangular selection is represented by the identifier '`xywh`', and the values are specified by an optional format '`pixel:`' or '`percent:`' (defaulting to pixel) and 4 comma-separated integers. These integers denote the top left corner coordinate (x,y) of the rectangle, its width and its height. If percent is used, x and width should be interpreted as a percentage of the width of the original media, and y and height should be interpreted as a percentage of the original height. Some typical spatial examples are shown in band 2 of Table I.

3) Track Dimension: The *track dimension* denotes one or multiple tracks, such as 'the English audio track' from a media container that supports multiple tracks (audio, video, subtitles, etc). Track selection is represented by the identifier '`track`', which has a string as a value. Multiple tracks are identified by multiple name/value pairs. Note that the

interpretation of such track names depends on the container format of the original media resource as some formats only allow numbers, whereas others allow full names. Some informative track examples are highlighted in band 3 of Table I.

4) Named Dimension: The *named dimension* denotes a named section of the original media, such as 'chapter 2'. It is in fact a semantic replacement for addressing any range along the aforementioned three axes (temporal, spatial, and track). Name-based selection is represented by the identifier '`id`', with again the value being a string. Percent-encoding can be used in the string to include unsafe characters (such as a single quote), as again can be seen in band 4 of Table I. Interpretation of such strings depends on the container format of the original media resource as some formats support named chapters or numbered chapters (leading to temporal clipping), whereas others may support naming of groups of tracks or other objects. As with track selection, determining which names are valid requires knowledge of the original media resource and its media container format.

As the temporal, spatial, and track dimensions are logically independent, they can be combined where the outcome is also independent of the order of the dimensions. As such, the fragments of band 5 of Table I should be byte-identical.

TABLE I
EXAMPLES OF VALID MEDIA FRAGMENT SYNTAXES

t=npt:10,20	results in the time interval [10,20[
t=,20	results in the time interval [0,20[
t=smpte:0:02:00,	results in the time interval [120,end[
xywh=160,120,320,240	results in a 320x240 box at x=160 and y=120
xywh=pixel:160,120,320,240	results in a 320x240 box at x=160 and y=120
xywh=percent:25,25,50,50	results in a 50%x50% box at x=25% and y=25%
track=1&track=2	results in only extracting track '1' and '2'
track=video	results in only extracting track 'video'
track=Kids%20Video	results in only extracting track 'Kids Video'
id=1	results in only extracting the section called '1'
id=chapter-1	results in only extracting the section called 'chapter-1'
id=My%20Kids	results in only extracting the section called 'My Kids'
#t=10,20&track=vid&xywh=pixel:0,0,320,240	
#track=vid&xywh=0,0,320,240&t=npt:10,20	
#xywh=0,0,320,240&t=smpte:0:00:10,0:00:20&track=vid	

III. RESOLVING MEDIA FRAGMENTS WITH HTTP

In the previous section we described the Media Fragments URI syntax, whereas in this section we present how a Media Fragments URI should be retrieved using the HTTP protocol. We foresee that the logic of the processing of a Media Fragments URI will be implemented within smart user agents, smart servers and sometimes in a proxy cacheable way. We observe that spatial media fragments are typically interpreted on the user agent side only (i.e., no spatial fragment extraction is performed on server-side) for the following reasons:

- Spatial fragments are typically not expressible in terms of byte ranges. Spatial fragment extraction would thus require transcoding operations resulting in new resources rather than fragments of the original media resource according to the semantics of the URI fragments defined in the corresponding RFC [6].
- The contextual information of extracted spatial fragments is not really usable for visualization on client-side.

In the remainder of this section, we describe how to resolve Media Fragments URIs using the HTTP protocol focusing on the temporal dimension for the sake of simplicity.

A. User Agent mapped Byte Ranges

As stated in our scenario in Section I, Steve can now show off his awesome play using his smart phone displaying the specific scene posted on his blog by using the following Media Fragments URI:

http://example.com/video.ogv#t=10,20

Since Steve does not want to use his entire monthly operator fee with the unnecessary bandwidth cost of downloading the full movie, he uses a smart user agent that is able to interpret the URI, determine that it only relates to a sub-part of a complete media resource, and thus requests only the appropriate data for download and playback. In this scenario, media fragments can be served by traditional HTTP Web servers. However, a smart user agent is necessary to parse the Media Fragments URI syntax. Furthermore, it requires knowledge about the syntax and semantics of various media codec formats. We assume that Steve's smart phone runs a smart user agent (e.g., an HTML5 compliant browser) that can resolve the temporal fragment identifier of the Media Fragments URI through a HTTP byte range request. A click on this URI triggers the following chain of events:

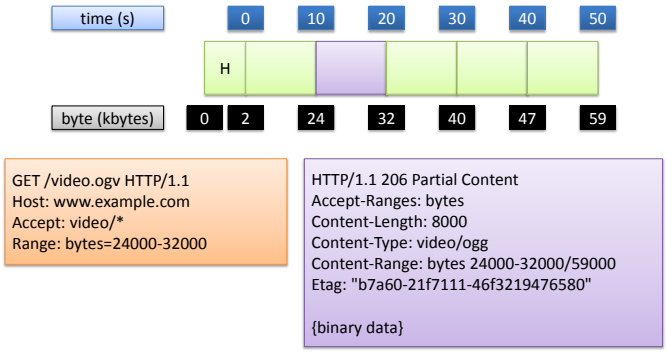

Fig. 2. User Agent requests Fragment for the first Time.

1) The user agent checks if a local copy of the requested fragment is available in its buffer, which is not the case.
2) We consider that the user agent knows how time is mapped to byte offsets for this particular Ogg media

format. It just parses the media fragment identifier and maps the fragment to the corresponding byte range(s).

3) The user agent sets up the decoding pipeline for the media resource at http://example.com/video.ogv by just downloading the first couple of bytes of that file that corresponds to the resource's header information.
4) The MIME-type of the resource requested is confirmed. The user agent can use the header information to resolve the fragment byte ranges. Based on the calculated time-to-byte mapping and the extracted information from the resource's header of where to find which byte, the user agent sends one or more HTTP requests using the *HTTP Range request* header (see Figure 2) for the relevant bytes of the fragment to the server. In this particular case, an HTTP 1.1 compliant Web server will be able to serve the media fragment.
5) The server extracts the bytes corresponding to the requested range and responds with a *HTTP 206 Partial Content response* containing the requested bytes.
6) Finally, the user agent receives these byte ranges and is able to decode and start playing back the initially requested media fragment.

More profound client side examples (fragment requests which are already buffered and fragment requests of a changed resource) can be found in the Media Fragments specification [7].

B. Server mapped Byte Ranges

We now assume that the user agent is able to parse and understand the Media Fragments URI syntax, but is unable to perform by itself the byte range mapping for a given request. In this case, the user agent needs some help from the media server to perform this mapping and deliver the appropriate bytes. In this case, the server has to help the client to setup the initial decoding pipeline (i.e., there is no header info from the resource on client side yet). When Steve starts to play this by now 'infamous' 10 seconds of video, the following events occur:

1) The user agent parses the media fragment identifier and creates an *HTTP Range request* expressed in a different unit than bytes, e.g., a time unit expressed in seconds. Furthermore, it extends this header with the keyword '`include-setup`' (see Figure 3) in order to let the server know that it also requires the initial header of the resource to initiate the decoding pipeline.
2) The server extracts the header information of the media resource as requested with '`include-setup`'.
3) The server, which also understands this time unit, resolves the *HTTP Range request* and performs the mapping between the media fragment time unit and byte ranges, and extracts the corresponding bytes.
4) Once the mapping is done, the server wraps then both the header information and the bytes requested in a multi-part *HTTP 206 Partial Content response*. As depicted in Figure 3, the first part contains the header data needed for setting up the decoding pipeline, whereas subsequent parts contain the requested bytes of the needed fragment. Note that a new header, named '`Content-Range-Mapping`', is introduced to provide the exact mapping of the retrieved byte range corresponding to the original '`Content-Range`' request expressed with a time-unit. The decoder might need extra data, before the beginning and/or after the end of the requested sequence, since this initial period might not correlate to a random access unit of the clip to start/end with. In analogy with the '`Content-Range`' header, the '`Content-Range-Mapping`' header also adds the instance-length after the slash-character '`/`'.
5) Finally, the user agent receives the multi-part byte ranges and is able to setup the decoding pipeline (using the header information), decodes, and starts playing back the media fragment requested.

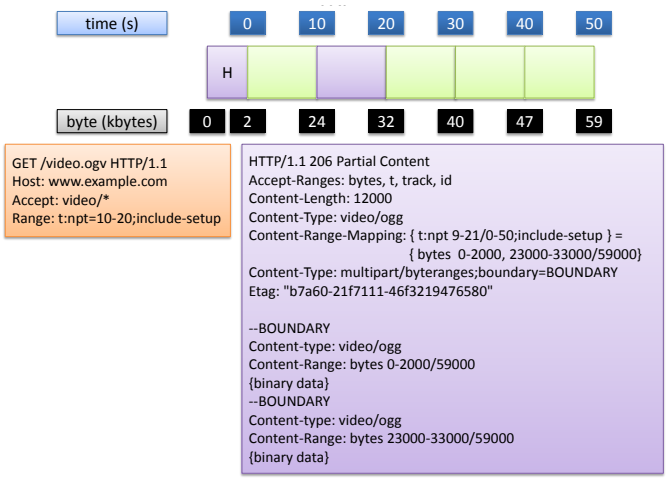

Fig. 3. Server mapped Byte Ranges including Header Information.

More profound server side examples (fragment requests with an initialised client, proxy cacheable server mapped byte ranges, and server triggered redirects) can be found in the Media Fragments specification [7].

IV. A Reference Implementation

Our NinSuna platform[4] is a fully integrated platform for multimedia content selection and adaptation, whose design and implementation are based on Semantic Web technologies. Furthermore, a tight coupling exists between NinSuna's design and a model for describing structural, semantic, and scalability information of media resources [14]. Media resources are ingested into the platform and mapped to this model. The adaptation and selection operations are based on this model and are thus independent of the underlying media formats, making NinSuna a format-independent media delivery platform [15]. The model (i.e., the NinSuna ontology[5]) is implemented using Semantic Web technologies:

[4]http://ninsuna.elis.ugent.be/
[5]http://ninsuna.elis.ugent.be/S3MediaOntology

the model is written in OWL, while the metadata instances are represented in RDF. Multimedia adaptation is then performed by selecting and adapting portions of the structural metadata using SPARQL. Additionally, a format-independent packaging technique is used, based on MPEG-B Bitstream Syntax Description Language (BSDL) [16], to deliver the media content in a fully format-independent manner to the end-user.

Suppose a client wants to retrieve the following Media Fragment URI: http://foo.com/media.mp4#t=0,10. The latter identifies the first ten seconds of the media resource http://foo.com/media.mp4. Of course, the client often has the desire not to retrieve the full resource, but only the identified subpart. In the example, only the first ten seconds of the requested media resource should be delivered to the user agent. As described above, the Media Fragment URI 1.0 specification currently defines three different axis: temporal, spatial, and track. When user agents are not able to perform the fragment extraction, they need help from Media Fragment-aware servers. Such servers typically handle only the temporal and the track axis, since extracting spatial fragments at server-side comes with a number of problems such as re-encoding and irrelevant fragment context information for user agents. In Figure 4, the structural metadata of a multimedia resource are depicted. These structural metadata are compliant to the model for media bitstreams, as discussed above. The multimedia resource is described by an *AnnotatedMultimedia* instance, which contains two *MediaBitstream* instances (i.e., a video and an audio track). Suppose we only want to select the video track from the media resource described in Figure 4, and more specifically only a subpart of the video track starting from second 2 and ending at second 10, resulting in the following Media Fragment URI: http://foo.com/media.mp4#track=video1&t=2,10. First, the track having as name 'video1' is selected. Next, data blocks are selected based on the temporal parameters (i.e., from second 2 to second 10). As shown in Figure 4, data block 50 corresponds to second 2 in the video stream. However, to guarantee that the adapted video bitstream can be decoded in a correct way, cuts in the bitstream should only be performed at random access points. Hence, data blocks are selected starting from data block 48.

As such, our model-driven media delivery platform aims at being a server-side implementation of the Media Fragments URI 1.0 specification. More specifically, the platform is able to deliver temporal and track fragments by using efficient media segment extraction methods. Moreover, since the platform is independent of underlying media formats, it is also a generic solution to resolve Media Fragments URIs. Currently, two demos (available online at http://ninsuna.elis.ugent.be/mediafragments) exist demonstrating the W3C Media Fragments Specification:

- Server-side Media Fragment extraction: both query- and fragment-based Media Fragments URIs are demonstrated.

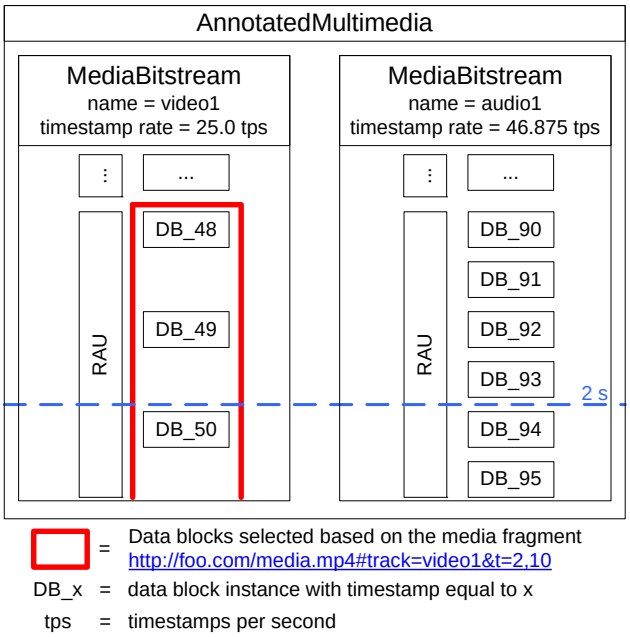

Fig. 4. Extracting Temporal and Track Fragments within NinSuna.

- Client-side Media Fragment rendering: a Media Fragments media player (both in Flash and HTML5).

V. Conclusions and Future Work

Currently, it is application dependant for all specified media fragment axes, how their defined media fragments should be rendered to the end-user in a meaningful way. Temporal fragments could be highlighted on a timing bar whereas spatial fragments could be emphasised by means of bounding boxes or they could be played back in colour while the background is played back in grey-scale. Finally, track selection could be done via dropdown boxes or buttons. Whether the Media Fragments URIs should be hidden from the end-user or not is an application implementation issue.

We already have an HTTP implementation for the W3C Media Fragments 1.0 specification [17] in order to verify all the test cases defined by our working group. Furthermore, several browser vendors are on their way of implementing a HTTP implementation themselves. In the near future, we also foresee reference implementations for the RTSP, Real Time Messaging Protocol[6] (RTMP), and File protocols.

There is currently no standardised way for user agents to discover the available named and track fragments. One could use ROE, which makes it possible to express the track composition of a media resource, in addition to naming these tracks and extra metadata info on language, role and content-type which could further help selecting the right

[6]Adobe's Real Time Messaging Protocol available at http://www.adobe.com/devnet/rtmp/

tracks. Another candidate to use is the *Media Multitrack API*[7] from the HTML5 Working Group[8]. This is a JavaScript API for HTML5 media elements that allows content authors to determine which data is available from a media resource. Not only does it expose the tracks that a media resource contains, but it also specifies the type of data that is encapsulated within the resource –e.g., audio/vorbis, text/srt, video/theora, etc.–, the role this data plays –e.g., audio description, caption, sign language, etc.–, and the actual language –e.g., RFC3066[9] language code.

With such media fragments addressing schemes available, there is still a need to hook up the addressing with the actual bytes of the resource. For the temporal and the spatial dimension, resolving the addressing into actual byte ranges is relatively obvious across any media type. However, track addressing and named addressing need to be resolved too. Track addressing will become easier when we solve the above stated requirement of exposing the track structure of a media resource. The name definition, on the other hand, will require association of an *id* or *name* with temporal offsets, spatial areas, or tracks.

Finally, hyperlinking out of media resources is something that is not generally supported at this stage. Certainly, some types of media resources -QuickTime, Flash[10], MPEG-4, and Ogg- support the definition of tracks that can contain HTML marked-up text and thus can also contain hyperlinks. It seems to be clear that hyperlinks out of media files will come from some type of textual track. But a standard format for such time-aligned text tracks does not exist yet. Re-using our specification, this challenge is still to be addressed in the near future.

In the meantime this Media Fragments URI specification, which we edited and contributed to over the last couple of years, now really opens up time-related media to the Internet crowd and makes time-related annotation [7], [8] feasible for hyperlinking into the media, thus providing the necessary support for this already omnipresent 'third dimension' *time* into the Internet.

ACKNOWLEDGMENT

The authors would like to thank the other W3C Media Fragments' participants. The research activities that have been described in this paper were partially funded by W3C/ERCIM, Ghent University, Interdisciplinary Institute for Broadband Technology (IBBT), and the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT).

[7]http://www.w3.org/WAI/PF/HTML/wiki/Media_MultitrackAPI

[8]On 6 May 2010 , the HTML5 draft specification introduced the WebSRT format (Web Subtitle Resource Tracks), a format intended for marking up external timed track resources.

[9]http://www.ietf.org/rfc/rfc3066.txt

[10]http://www.flash.com

REFERENCES

[1] I. Hickson and D. Hyatt, Eds., *HTML5 – A Vocabulary and associated APIs for HTML and XHTML*, ser. W3C Working Draft. World Wide Web Consortium, 4 March 2010, available at http://www.w3.org/TR/html5/.

[2] S. Pfeiffer, "Architecture of a Video Web - Experience with Annodex," in *Proceedings of the W3C Video on the Web Workshop*, Brussels, Belgium, December 2007, pp. 1–5.

[3] R. Troncy, L. Hardman, J. Van Ossenbruggen, and M. Hausenblas, "Identifying Spatial and Temporal Media Fragments on the Web," in *Proceedings of the W3C Video on the Web Workshop*, Brussels, Belgium, December 2007, pp. 1–6.

[4] W3C Media Fragments Working Group, "Media Fragments Working Group," 2010, available at http://www.w3.org/2008/WebVideo/Fragments.

[5] M. Hausenblas, R. Troncy, T. Burger, and Y. Raimond, "Interlinking Multimedia," in *Proceedings of the 2^{nd} Workshop on Linked Data on the Web*, Madrid, Spain, April 2009, pp. 1–9.

[6] Internet Engineering Task Force, "RFC 3986: Uniform Resource Identifier (URI) – Generic Syntax," 2005, available at http://tools.ietf.org/html/rfc3986.

[7] E. Mannens and R. Troncy, Eds., *Media Fragments URI 1.0*, ser. W3C Working Draft. World Wide Web Consortium, June 2010, available at http://www.w3.org/TR/media-frags.

[8] R. Troncy and E. Mannens, Eds., *Use Cases and Requirements for Media Fragments*, ser. W3C Working Draft. World Wide Web Consortium, December 2009, available at http://www.w3.org/TR/media-frags-reqs.

[9] Internet Engineering Task Force, "RFC 2616: HyperText Transfer Protocol – HTTP/1.1," 1999, available at http://www.ietf.org/rfc/rfc2616.txt.

[10] ISO/IEC, "Information technology – Coding of Audio, Picture, Multimedia and Hypermedia Information – Part 14: MP4 file format," 2003, iSO/IEC 14496-14:2003.

[11] S. Pfeiffer, "RFC 3533: The Ogg Encapsulation Format Version 0," 2003, available at http://www.ietf.org/rfc/rfc3533.txt.

[12] H. Schulzrinne, A. Rao, and R. Lanphier, "RFC 2326: Real Time Streaming Protocol," 1998, available at http://www.ietf.org/rfc/rfc2326.txt.

[13] Society of Motion Picture and Television Engineers, "SMPTE RP 136: Time and Control Codes for 24, 25 or 30 Frame-Per-Second Motion-Picture Systems," 2004, available at http://www.smpte.org/standards.

[14] D. Van Deursen, W. Van Lancker, S. De Bruyne, W. De Neve, E. Mannens, and R. Van de Walle, "Format-independent and Metadata-driven Media Resource Adaptation using Semantic Web Technologies," *Multimedia Systems Journal*, vol. 16, no. 2, pp. 85–104, January 2010.

[15] D. Van Deursen, W. Van Lancker, W. De Neve, T. Paridaens, E. Mannens, and R. Van de Walle, "NinSuna: a Fully Integrated Platform for Format-independent Multimedia Content Adaptation and Delivery based on Semantic Web Technologies," *Multimedia Tools and Applications – Special Issue on Data Semantics for Multimedia Systems*, vol. 46, no. 2, pp. 371–398, January 2010.

[16] ISO/IEC, "Information technology – MPEG Systems Technologies – Part 5: Bitstream Syntax Description Language," 2008, ISO/IEC 23001-5:2008.

[17] D. Van Deursen, R. Troncy, E. Mannens, S. Pfeiffer, Y. Lafon, and R. Van de Walle, "Implementing the Media Fragments URI Specification," in *Proceedings of the 19^{th} International World Wide Web Conference (WWW)*, Raleigh, USA, April 2010, pp. 1361–1363.